
OSMnx

Release 2.1.1

Geoff Boeing

Aug 23, 2026

CONTENTS

1	Citation	3
2	Getting Started	5
3	Installation	7
4	Support	9
5	License	11
6	User Guides	13
7	Indices	21

OSMnx is a Python package to easily download, model, analyze, and visualize street networks and other geospatial features from OpenStreetMap. You can download and model walking, driving, or biking networks with a single line of code then analyze and visualize them. You can just as easily work with urban amenities/points of interest, building footprints, transit stops, elevation data, street orientations, speed/travel time, and routing.

CITATION

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

Boeing, G. (2025). [Modeling and Analyzing Urban Networks and Amenities with OSMnx](#). *Geographical Analysis* 57 (4), 567-577. doi:10.1111/gean.70009

GETTING STARTED

First read the *Getting Started* guide for an introduction to the package and FAQ.
Then work through the *Examples Gallery* for step-by-step tutorials and sample code.

INSTALLATION

Follow the *Installation* guide to install OSMnx.

SUPPORT

If you have any trouble, consult the *User Reference*. The OSMnx repository is hosted on [GitHub](#). If you have a “how-to” or usage question, please ask it on [StackOverflow](#), as we reserve the repository’s issue tracker for bug tracking and feature development.

LICENSE

OSMnx is open source and licensed under the MIT license. OpenStreetMap's open data [license](#) requires that derivative works provide proper attribution. Refer to the [Getting Started](#) guide for usage limitations.

USER GUIDES

6.1 Installation

6.1.1 Conda

The foolproof way to install OSMnx is with [conda](#) or [mamba](#):

```
conda create --strict-channel-priority -c conda-forge -n ox osmnx
```

This creates a new conda environment and installs OSMnx into it, via the conda-forge channel. If you want other packages, such as [jupyterlab](#), installed in this environment as well, just add their names after `osmnx` above. To upgrade OSMnx to a newer release, remove the conda environment you created and then create a new one again, as above. See the [conda-forge](#) documentation for details.

6.1.2 Docker

You can run OSMnx + JupyterLab directly from the official OSMnx [Docker](#) image.

6.1.3 Pip

You can also install OSMnx with [uv](#) or [pip](#) (into a virtual environment):

```
pip install osmnx
```

OSMnx is written in pure Python and distributed on [PyPI](#). Its installation alone is thus trivially simple if you have its dependencies installed and tested on your system. However, OSMnx depends on other packages that in turn depend on compiled C/C++ libraries, which may present some challenges depending on your specific system's configuration. If precompiled binaries are not available for your system, you may need to compile and configure those dependencies by following their installation instructions. So, if you're not sure what you're doing, just follow the conda instructions above to avoid installation problems.

6.2 Getting Started

6.2.1 Get Started in 4 Steps

1. Install OSMnx by following the [Installation](#) guide.
2. Read the [Introducing OSMnx](#) section on this page.
3. Work through the OSMnx [Examples Gallery](#) for step-by-step tutorials and sample code.
4. Consult the [User Reference](#) for complete details on using the package.

Finally, if you're not already familiar with [NetworkX](#) and [GeoPandas](#), make sure you read their user guides as OSMnx uses their data structures.

6.2.2 Introducing OSMnx

This quick introduction explains key concepts and the basic functionality of OSMnx.

Overview

OSMnx is pronounced as the initialism: “oh-ess-em-en-ex”. It is built on top of [NetworkX](#) and [GeoPandas](#), and interacts with [OpenStreetMap](#) APIs to:

- Download and model street networks or other infrastructure anywhere in the world with a single line of code
- Download geospatial features (e.g., political boundaries, building footprints, grocery stores, transit stops) as a `GeoDataFrame`
- Query by city name, polygon, bounding box, or point/address + distance
- Model driving, walking, biking, and other travel modes
- Attach node elevations from a local raster file or web service and calculate edge grades
- Impute missing speeds and calculate graph edge travel times
- Simplify and correct the network's topology to clean-up nodes and consolidate complex intersections
- Fast map-matching of points, routes, or trajectories to nearest graph edges or nodes
- Save/load network to/from disk as GraphML, GeoPackage, or 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
- Explore street networks and geospatial features as a static map or interactive web map
- Visualize travel distance and travel time with isoline and isochrone maps
- Plot figure-ground diagrams of street networks and building footprints

The OSMnx [Examples Gallery](#) contains tutorials and demonstrations of all these features, and package usage is detailed in the [User Reference](#).

Configuration

You can configure OSMnx using the `settings` module. Here you can adjust logging behavior, caching, server endpoints, and more. You can also configure OSMnx to retrieve historical snapshots of [OpenStreetMap](#) data as of a certain date.

Read more about the [settings](#) module in the User Reference.

Geocoding and Querying

OSMnx geocodes place names and addresses with the [OpenStreetMap Nominatim](#) API. You can use the geocoder module to geocode place names or addresses to lat-lon coordinates. Or, you can retrieve place boundaries or any other [OpenStreetMap](#) elements by name or ID. Read more about the [geocoder](#) module in the User Reference.

Using the `features` and `graph` modules, as described below, you can download data by lat-lon point, address, bounding box, bounding polygon, or place name (e.g., neighborhood, city, county, etc).

Urban Amenities

Using OSMnx's `features` module, you can search for and download any geospatial [features](#) (such as building footprints, grocery stores, schools, public parks, transit stops, etc) from the OpenStreetMap [Overpass](#) API as a GeoPandas GeoDataFrame. This uses OpenStreetMap [tags](#) to search for matching [elements](#).

Read more about the [features](#) module in the User Reference.

Modeling a Network

Using OSMnx's `graph` module, you can retrieve any spatial network data (such as streets, paths, rail, canals, etc) from the Overpass API and model them as NetworkX [MultiDiGraphs](#).

In short, MultiDiGraphs are nonplanar directed graphs with possible self-loops and parallel edges. Thus, a one-way street will be represented with a single directed edge from node u to node v , but a bidirectional street will be represented with two reciprocal directed edges (with identical geometries): one from node u to node v and another from v to u , to represent both possible directions of flow. Because these graphs are nonplanar, they correctly model the topology of interchanges, bridges, and tunnels. That is, edge crossings in a two-dimensional plane are not intersections in an OSMnx model unless they represent true junctions in the three-dimensional real world.

The `graph` module uses filters to query the Overpass API: you can either specify a built-in network type or provide your own custom filter with [Overpass QL](#). Under the hood, OSMnx does several things to generate the best possible model. It initially creates a 500m-buffered graph before truncating it to your desired query area, to ensure accurate streets-per-node stats and to attenuate graph perimeter effects. By default, it returns the largest weakly connected component. It also simplifies the graph topology as discussed below.

Read more about the [graph](#) module in the User Reference and refer to the official reference paper at the [Further Reading](#) page for complete modeling details.

Topology Clean-Up

The `simplification` module automatically processes the network's topology from the original raw OpenStreetMap data, such that nodes represent intersections/dead-ends and edges represent the street segments that link them. This takes two primary forms: graph simplification and intersection consolidation.

Graph simplification cleans up the graph's topology so that nodes represent intersections or dead-ends and edges represent street segments. This is important because in OpenStreetMap raw data, ways comprise sets of straight-line segments between nodes: that is, nodes are vertices for streets' curving line geometries, not just intersections and dead-ends. By default, OSMnx simplifies this topology by discarding non-intersection/dead-end nodes while retaining the complete true edge geometry as an edge attribute. When multiple OpenStreetMap ways are merged into a single graph edge, the ways' attribute values can be aggregated into a single value.

Intersection consolidation is important because many real-world street networks feature complex intersections and traffic circles, resulting in a cluster of graph nodes where there is really just one true intersection as we would think of it in transportation or urban design. Similarly, 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, but these 4 nodes represent a single intersection in the real world. OSMnx can consolidate such complex intersections into a single node and optionally rebuild the graph's edge topology accordingly. When multiple OpenStreetMap nodes are merged into a single graph node, the nodes' attribute values can be aggregated into a single value.

Read more about the [simplification](#) module in the User Reference.

Model Attributes

An OSMnx model has some standard required attributes, plus some optional attributes. The latter are sometimes present based on the source OSM data's tagging, the `settings` module configuration, and any processing you may have done to add additional attributes (as noted in various functions' documentation).

As a NetworkX [MultiDiGraph](#) object, it has top-level `graph`, `nodes`, and `edges` attributes. The `graph` attribute dictionary must contain a “`crs`” key defining its coordinate reference system. The `nodes` are identified by OSM ID and each must contain a `data` attribute dictionary that must have “`x`” and “`y`” keys defining its coordinates and a “`street_count`” key defining how many physical streets are incident to it. The `edges` are identified by a 3-tuple of “`u`” (source node ID), “`v`” (target node ID), and “`key`” (to differentiate parallel edges), and each must contain a `data` attribute dictionary that must have an “`osmid`” key defining its OSM ID and a “`length`” key defining its length in meters.

The OSMnx `graph` module automatically creates `MultiDiGraphs` with these required attributes, plus additional optional attributes based on the `settings` module configuration. If you instead manually create your own graph model, make sure it has these required attributes at a minimum.

Convert, Project, Save

OSMnx’s `convert` module can convert a `MultiDiGraph` to a [DiGraph](#) if you prefer a directed representation of the network without any parallel edges, or to a [MultiGraph](#) if you need an undirected representation for use with functions or algorithms that only accept a `MultiGraph` object. If you just want a fully bidirectional graph (such as for a walking network), just configure the `settings` module’s `bidirectional_network_types` before creating your graph.

The `convert` module can also convert a `MultiDiGraph` to/from GeoPandas node and edge [GeoDataFrames](#). The nodes `GeoDataFrame` is indexed by OSM ID and the edges `GeoDataFrame` is multi-indexed by `u`, `v`, `key` just like a NetworkX edge. This allows you to load arbitrary node/edge ShapeFiles or GeoPackage layers as `GeoDataFrames` then model them as a `MultiDiGraph` for graph analysis. The `convert` module exposes validation functions to verify that your `MultiDiGraph` or `GeoDataFrames` satisfy OSMnx requirements. Read more about the [convert](#) module in the User Reference.

You can easily project your graph to different coordinate reference systems using the `projection` module. If you’re unsure which [CRS](#) you want to project to, OSMnx can automatically determine an appropriate UTM CRS for you. Read more about the [projection](#) module in the User Reference.

Using the `io` module, you can save your graph to disk as a GraphML file (to load into other network analysis software), a GeoPackage (to load into other GIS software), or an OSM XML file. Use the GraphML format whenever saving a graph for later work with OSMnx. Read more about the [io](#) module in the User Reference.

Network Statistics

You can use the `stats` module to calculate a variety of geometric and topological measures as well as street network bearing and orientation statistics. These measures define streets as the edges in an undirected representation of the graph to prevent double-counting bidirectional edges of a two-way street. You can easily generate common stats in transportation studies, urban design, and network science, including intersection density, circuitry, average node degree (connectedness), betweenness centrality, and much more. Read more about the [stats](#) module in the User Reference.

You can also use NetworkX directly to calculate additional topological network measures.

Working with Elevation

The `elevation` module lets you automatically attach elevations to the graph’s nodes from a local raster file or the Google Maps [Elevation API](#) (or equivalent web API with a compatible interface). You can also calculate edge grades (i.e., rise-over-run) and analyze the steepness of certain streets or routes.

Read more about the [elevation](#) module in the User Reference.

Routing

The `distance` module can find the nearest node(s) or edge(s) to coordinates using a fast spatial index. The `routing` module can solve shortest paths for network routing, parallelized with multiprocessing, using different weights (e.g., distance, travel time, elevation change, etc). It can also impute missing speeds to the graph edges. This imputation can obviously be imprecise, so the user can override it by passing in arguments that define local speed limits. It can also calculate free-flow travel times for each edge.

Read more about the [distance](#) and [routing](#) modules in the User Reference.

Visualization

You can plot graphs, routes, network figure-ground diagrams, building footprints, and street network orientation rose diagrams (aka, polar histograms) with the `plot` module. You can also explore street networks, routes, or geospatial features as interactive [Folium](#) web maps.

Read more about the [plot](#) module in the User Reference.

Usage Limits

Be a good neighbor! OSMnx works with Overpass's rate limiting to avoid overwhelming their resources. Don't run multiple/parallel OSMnx instances simultaneously to circumvent their limits. These APIs are free to use, but they're not free to operate. If you will make many queries (e.g., more than 1k/day), you need to host your own local Overpass instance. Refer to the [Nominatim Usage Policy](#) and [Overpass Commons](#) documentation for API usage limits and restrictions to which you must adhere. If you configure OSMnx to use an alternative API instance, ensure you understand and follow their policies. If you feel you need to exceed these limits, just install your own hosted instance and set OSMnx to use it.

6.2.3 More Info

All of this functionality is demonstrated step-by-step in the OSMnx [Examples Gallery](#), and usage is detailed in the [User Reference](#). Feature development details are in the [Changelog](#). Consult the [Further Reading](#) resources for additional technical details and research.

6.2.4 Frequently Asked Questions

How do I install OSMnx? Follow the [Installation](#) guide.

How do I use OSMnx? Check out the step-by-step tutorials in the OSMnx [Examples Gallery](#).

How does this or that function work? Consult the [User Reference](#).

What can I do with OSMnx? Check out recent [projects](#) that use OSMnx.

I have a usage question. Please ask it on [StackOverflow](#).

6.3 User Reference

This is the User Reference for the OSMnx package. If you are looking for an introduction to OSMnx, read the [Getting Started](#) guide. This guide describes the usage of OSMnx's public API.

- 6.3.1 osmnx.bearing module
- 6.3.2 osmnx.convert module
- 6.3.3 osmnx.distance module
- 6.3.4 osmnx.elevation module
- 6.3.5 osmnx.features module
- 6.3.6 osmnx.geocoder module
- 6.3.7 osmnx.graph module
- 6.3.8 osmnx.io module
- 6.3.9 osmnx.plot module
- 6.3.10 osmnx.projection module
- 6.3.11 osmnx.routing module
- 6.3.12 osmnx.settings module
- 6.3.13 osmnx.simplification module
- 6.3.14 osmnx.stats module
- 6.3.15 osmnx.truncate module
- 6.3.16 osmnx.utils module
- 6.3.17 osmnx.utils_geo module

6.4 Internals Reference

This is the complete OSMnx internals reference for developers, including private internal modules and functions. If you are instead looking for a user guide to OSMnx's public API, see the *User Reference*.

- 6.4.1 `osmnx._api_v1` module
- 6.4.2 `osmnx.bearing` module
- 6.4.3 `osmnx.convert` module
- 6.4.4 `osmnx.distance` module
- 6.4.5 `osmnx.elevation` module
- 6.4.6 `osmnx._errors` module
- 6.4.7 `osmnx.features` module
- 6.4.8 `osmnx.geocoder` module
- 6.4.9 `osmnx.graph` module
- 6.4.10 `osmnx._http` module
- 6.4.11 `osmnx.io` module
- 6.4.12 `osmnx._nominatim` module
- 6.4.13 `osmnx._osm_xml` module
- 6.4.14 `osmnx._overpass` module
- 6.4.15 `osmnx.plot` module
- 6.4.16 `osmnx.projection` module
- 6.4.17 `osmnx.routing` module
- 6.4.18 `osmnx.settings` module
- 6.4.19 `osmnx.simplification` module
- 6.4.20 `osmnx.stats` module
- 6.4.21 `osmnx.truncate` module
- 6.4.22 `osmnx.utils` module
- 6.4.23 `osmnx.utils_geo` module
- 6.4.24 `osmnx._validate` module

6.5 Further Reading

Boeing, G. (2025). *Modeling and Analyzing Urban Networks and Amenities with OSMnx*. *Geographical Analysis* 57 (4), 567-577. doi:10.1111/gean.70009

This is the official reference paper and citation for the OSMnx package.

Boeing, G. (2026). *Urban Science Beyond Samples: Up-to-Date Street Network Models and Indicators for Every Urban Area in the World*. *Environment and Planning B*, published online ahead of print. doi:10.1177/23998083261446991

This study uses OSMnx to model and analyze the street networks of every urban area in the world: over 180 million OpenStreetMap street network nodes and over 360 million edges across 10,351 urban areas in 189 countries.

Boeing, G. (2025). [Topological Graph Simplification Solutions to the Street Intersection Miscount Problem](#). *Transactions in GIS* 29 (3), e70037. doi:10.1111/tgis.70037

This paper describes and validates the algorithms implemented in OSMnx's `simplification` module and explains why graph simplification is necessary to accurately measure intersection density, street segment length, node degree, etc.

Boeing, G. (2020). [Planarity and Street Network Representation in Urban Form Analysis](#). *Environment and Planning B* 47 (5), 855-869. doi:10.1177/2399808318802941

This paper demonstrates the need for nonplanar graphs when modeling urban street networks, which was one of the original motivations for developing OSMnx.

Boeing, G. (2020). [The Right Tools for the Job: The Case for Spatial Science Tool-Building](#). *Transactions in GIS* 24 (5), 1299-1314. doi:10.1111/tgis.12678

This paper was presented as the 8th annual Transactions in GIS plenary address at the American Association of Geographers annual meeting in Washington, DC. It describes the early development of OSMnx and reviews its use in scientific research over the previous few years.

INDICES

- `genindex`
- `modindex`
- `search`