Overture Maps Data Download

Applying DuckDB in R and Python

Pukar Bhandari

2024-05-22

Table of contents

# 1. Downloading Overture Maps Data with R and Python

Overture Maps Foundation provides a collaborative, open-source initiative to create the world’s most comprehensive and interoperable geospatial dataset. As transportation planners and data analysts, we often need access to high-quality geospatial data for buildings, transportation networks, places, and administrative boundaries. This post demonstrates how to efficiently download Overture Maps data using both R and Python with DuckDB’s powerful spatial capabilities.

## 1.1 What is Overture Maps?

Overture Maps is an open-source mapping initiative that provides global-scale geospatial data across four main themes:

* **Buildings**: Footprints and building parts
* **Transportation**: Road segments and connectors
* **Places**: Points of interest and place data
* **Admins**: Administrative boundaries and localities
* **Base**: Infrastructure, land use, land cover, and water features

The data is stored in cloud-optimized Parquet format on AWS S3, making it ideal for efficient querying and analysis.

## 1.2 Prerequisites

Before diving into the code, ensure you have the following dependencies installed:

####  R

# Install required packages
install.packages(c("tidyverse", "sf", "mapview", "DBI", "duckdb", "arrow"))

####  Python

# Install required packages
pip install duckdb matplotlib geopandas pandas shapely folium pathlib fastparquet

## 1.3 Setting Up the Environment

First, we need to load our libraries and configure the environment for spatial data processing.

####  R

# Load required libraries
library(tidyverse)
library(sf)
library(mapview)
library(DBI)
library(duckdb)
library(arrow)

####  Python

import duckdb
import geopandas as gpd
import pandas as pd
import shapely.wkb
import matplotlib.pyplot as plt
import folium
import os
from pathlib import Path

## 1.4 Data Type Mapping

Overture data is organized by themes, and we need to map specific data types to their corresponding themes for proper S3 path construction.

####  R

# Define the theme map
map\_themes <- list(
 "locality" = "admins",
 "locality\_area" = "admins",
 "administrative\_boundary" = "admins",
 "building" = "buildings",
 "building\_part" = "buildings",
 "place" = "places",
 "segment" = "transportation",
 "connector" = "transportation",
 "infrastructure" = "base",
 "land" = "base",
 "land\_use" = "base",
 "water" = "base"
)

####  Python

# Define theme mapping
map\_themes = {
 "locality": "admins",
 "locality\_area": "admins",
 "administrative\_boundary": "admins",
 "building": "buildings",
 "building\_part": "buildings",
 "place": "places",
 "segment": "transportation",
 "connector": "transportation",
 "infrastructure": "base",
 "land": "base",
 "land\_use": "base",
 "water": "base",
}

## 1.5 Core Download Function

This function handles the DuckDB connection, S3 configuration, and spatial filtering to download only the data within your specified bounding box.

####  R

overture\_data <- function(bbox, overture\_type, dst\_parquet) {

 # Validate overture\_type
 if (!overture\_type %in% names(map\_themes)) {
 stop(paste("Valid Overture types are:", paste(names(map\_themes), collapse = ", ")))
 }

 # Configure S3 path
 s3\_region <- "us-west-2"
 base\_url <- sprintf("s3://overturemaps-%s/release", s3\_region)
 version <- "2024-04-16-beta.0"
 theme <- map\_themes[[overture\_type]]
 remote\_path <- sprintf("%s/%s/theme=%s/type=%s/\*", base\_url, version, theme, overture\_type)

 # Connect to DuckDB and install extensions
 conn <- DBI::dbConnect(duckdb::duckdb())
 DBI::dbExecute(conn, "INSTALL httpfs;")
 DBI::dbExecute(conn, "INSTALL spatial;")
 DBI::dbExecute(conn, "LOAD httpfs;")
 DBI::dbExecute(conn, "LOAD spatial;")
 DBI::dbExecute(conn, sprintf("SET s3\_region='%s';", s3\_region))

 # Create view and execute spatial query
 read\_parquet <- sprintf("read\_parquet('%s', filename=TRUE, hive\_partitioning=1);", remote\_path)
 DBI::dbExecute(conn, sprintf("CREATE OR REPLACE VIEW data\_view AS SELECT \* FROM %s", read\_parquet))

 query <- sprintf("
 SELECT data.\*
 FROM data\_view AS data
 WHERE data.bbox.xmin <= %f AND data.bbox.xmax >= %f
 AND data.bbox.ymin <= %f AND data.bbox.ymax >= %f
 ", bbox[3], bbox[1], bbox[4], bbox[2])

 # Save results to Parquet file
 file <- normalizePath(dst\_parquet, mustWork = FALSE)
 DBI::dbExecute(conn, sprintf("COPY (%s) TO '%s' WITH (FORMAT 'parquet');", query, file))
 DBI::dbDisconnect(conn, shutdown = TRUE)
}

####  Python

def overture\_data(bbox, overture\_type, dst\_parquet):
 """Query a subset of Overture's data and save it as a GeoParquet file.

 Parameters
 ----------
 bbox : tuple
 A tuple of floats representing the bounding box (xmin, ymin, xmax, ymax)
 in EPSG:4326 coordinate reference system.
 overture\_type : str
 The type of Overture data to query
 dst\_parquet : str or Path
 The path to the output GeoParquet file.
 """
 if overture\_type not in map\_themes:
 raise ValueError(f"Valid Overture types are: {list(map\_themes)}")

 # Configure S3 connection
 s3\_region = "us-west-2"
 base\_url = f"s3://overturemaps-{s3\_region}/release"
 version = "2024-04-16-beta.0"
 theme = map\_themes[overture\_type]
 remote\_path = f"{base\_url}/{version}/theme={theme}/type={overture\_type}/\*"

 # Setup DuckDB with spatial extensions
 conn = duckdb.connect()
 conn.execute("INSTALL httpfs;")
 conn.execute("INSTALL spatial;")
 conn.execute("LOAD httpfs;")
 conn.execute("LOAD spatial;")
 conn.execute(f"SET s3\_region='{s3\_region}';")

 # Execute spatial query
 read\_parquet = f"read\_parquet('{remote\_path}', filename=true, hive\_partitioning=1);"
 conn.execute(f"CREATE OR REPLACE VIEW data\_view AS SELECT \* FROM {read\_parquet}")

 query = f"""
 SELECT data.\*
 FROM data\_view AS data
 WHERE data.bbox.xmin <= {bbox[2]} AND data.bbox.xmax >= {bbox[0]}
 AND data.bbox.ymin <= {bbox[3]} AND data.bbox.ymax >= {bbox[1]}
 """

 file = str(Path(dst\_parquet).resolve())
 conn.execute(f"COPY ({query}) TO '{file}' WITH (FORMAT PARQUET);")
 conn.close()

## 1.6 Defining Your Study Area

For spatial analysis, you need to define a bounding box for your area of interest. This can come from existing boundary data or manual coordinates.

####  R

# Read existing boundary data (example: Salt Lake County, UT)
slco\_boundary <- sf::read\_sf(
 "data/SaltLakeCounty\_Boundary.geojson"
) |>
 sf::st\_transform("EPSG:4326")

# Extract bounding box coordinates (xmin, ymin, xmax, ymax)
slco\_bbox <- slco\_boundary |>
 sf::st\_bbox() |>
 as.vector()

print(slco\_bbox)

[1] -112.26017 40.41417 -111.55320 40.92187

####  Python

# Read existing boundary data (example: Salt Lake County, UT)
slco\_boundary = gpd.read\_file("data/SaltLakeCounty\_Boundary.geojson") \
 .to\_crs("EPSG:4326")

# Extract bounding box coordinates (xmin, ymin, xmax, ymax)
slco\_bbox = slco\_boundary.total\_bounds

print(slco\_bbox)

[-112.26016909 40.41417467 -111.55319996 40.92186613]

## 1.7 Downloading the Data

Now we can download specific data types for our study area. The function handles all the cloud connectivity and spatial filtering automatically.

####  R

# Download places data for Salt Lake County
if (!file.exists("output/slco\_places\_r.parquet")) {
 overture\_data(slco\_bbox, "place", "output/slco\_places\_r.parquet")
}

####  Python

# Download places data for Salt Lake County
if not os.path.exists("output/slco\_places\_py.parquet"):
 overture\_data(slco\_bbox, "place", "output/slco\_places\_py.parquet")

## 1.8 Processing Downloaded Data

After downloading, convert the Parquet files to spatial data formats for analysis and visualization.

####  R

# Read the downloaded Parquet file
slco\_places <- arrow::read\_parquet("output/slco\_places\_r.parquet")

# Convert to sf object for spatial operations
slco\_places\_sf <- sf::st\_as\_sf(
 slco\_places |> dplyr::select(-sources),
 geometry = slco\_places$geometry,
 crs = "EPSG:4326"
)

# Basic data exploration
print(paste("Downloaded", nrow(slco\_places\_sf), "places"))

[1] "Downloaded 44355 places"

print(colnames(slco\_places\_sf))

 [1] "id" "geometry" "bbox" "version" "update\_time"
 [6] "names" "categories" "confidence" "websites" "socials"
[11] "emails" "phones" "brand" "addresses" "filename"
[16] "theme" "type"

####  Python

# Read the downloaded data
slco\_places = pd.read\_parquet("output/slco\_places\_py.parquet")

# Convert to GeoDataFrame
slco\_places\_gdf = gpd.GeoDataFrame(
 slco\_places.drop(columns="geometry"),
 geometry=shapely.wkb.loads(slco\_places["geometry"]),
 crs=4326,
)

# Basic exploration
print(f"Downloaded {len(slco\_places\_gdf)} places")

Downloaded 44355 places

print(slco\_places\_gdf.columns.tolist())

['id', 'version', 'update\_time', 'sources', 'confidence', 'websites', 'socials', 'emails', 'phones', 'addresses', 'filename', 'theme', 'type', 'bbox.xmin', 'bbox.xmax', 'bbox.ymin', 'bbox.ymax', 'names.primary', 'names.common', 'names.rules', 'categories.main', 'categories.alternate', 'brand.wikidata', 'brand.names.primary', 'brand.names.common', 'brand.names.rules', 'geometry']

## 1.9 Data Visualization

Create quick visualizations to explore your downloaded data and verify the results.

####  R

# Simple quick visualization using mapview
set.seed(123)

slco\_places\_sf |>
 dplyr::sample\_n(1000) |>
 tidyr::unnest(cols = c("names", "categories")) |>
 dplyr::select(
 names\_primary = "primary",
 categories\_main = "main",
 confidence
 ) |>
 mapview::mapview(map.types = "CartoDB.Voyager", legend = FALSE)

####  Python

# Simple quick visualization using GeoPandas
slco\_places\_gdf \
 .sample(n=1000, random\_state=123) \
 .rename(columns={
 "names.primary": "names\_primary",
 "categories.main": "categories\_main"
 }) \
 [['names\_primary', 'categories\_main', 'confidence', 'geometry']] \
 .explore(tiles="CartoDB.Voyager", zoom\_start=9)

## 1.10 Available Data Types

Overture Maps provides the following data types organized by theme:

| Theme | Data Types | Description |
| --- | --- | --- |
| **Admins** | locality, locality\_area, administrative\_boundary | Administrative boundaries and place hierarchies |
| **Buildings** | building, building\_part | Building footprints and structural components |
| **Places** | place | Points of interest, businesses, and landmarks |
| **Transportation** | segment, connector | Road networks and transportation infrastructure |
| **Base** | infrastructure, land, land\_use, water | Base map features and land cover |

## 1.11 Transportation Planning Applications

This approach is particularly valuable for transportation planning workflows where you need to integrate multiple data sources for comprehensive analysis. The standardized schema and efficient spatial querying make it ideal for network analysis, land use integration, and multi-modal planning across different jurisdictions and scales.

## 1.12 Repository and Additional Resources

The complete code and examples are available in the [Overture Data Download repository](https://github.com/ar-puuk/overture-data-download/) on GitHub.

For more information about Overture Maps:

* [Official Documentation](https://docs.overturemaps.org/)
* [Data Schema Reference](https://docs.overturemaps.org/schema/)
* [Community Forum](https://github.com/OvertureMaps/data/discussions)

*Want to contribute or suggest improvements? Visit the project repository at:* <https://github.com/ar-puuk/overture-data-download>