Overture Maps Data Download

Applying DuckDB in R and Python

Pukar Bhandari

pukar.bhandari@outlook.com

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:

1.2.-.a R

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

1.2.-.b Python

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

1.3 Setting Up the Environment

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

1.3.-.a R

```r
# Load required libraries
library(tidyverse)
library(sf)
library(tmap)
library(DBI)
library(duckdb)
library(arrow)

# Set global options
options(scipen = 999) # avoiding scientific notation
tmap_mode(mode = "view")
```

1.3.-.b Python

```python
import duckdb
import geopandas as gpd
import pandas as pd
import shapely.wkb
import matplotlib.pyplot as plt
import folium
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.

1.4.-.a 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"
)
```

1.4.-.b Python

```
# Define theme mapping
THEME_MAP = {
    "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.

1.5.-.a 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
= ", ")))
```

```r
  }

  # 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 <- dbConnect(duckdb::duckdb())
  dbExecute(conn, "INSTALL httpfs;")
  dbExecute(conn, "INSTALL spatial;")
  dbExecute(conn, "LOAD httpfs;")
  dbExecute(conn, "LOAD spatial;")
  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)
  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)
  dbExecute(conn, sprintf("COPY (%s) TO '%s' WITH (FORMAT 'parquet');", query,
file))
  dbDisconnect(conn, shutdown = TRUE)
}
```

1.5.-.b Python

```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 THEME_MAP:
        raise ValueError(f"Valid Overture types are: {list(THEME_MAP)}")

    # Configure S3 connection
    s3_region = "us-west-2"
    base_url = f"s3://overturemaps-{s3_region}/release"
    version = "2024-04-16-beta.0"
    theme = THEME_MAP[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.

1.6.-.a R

```r
# Read existing boundary data (example: Albany, NY)
albany_boundary <- read_sf(file.path(wd, project), layer = "City_of_Albany") |
>
  st_transform(4326)

# Extract bounding box coordinates (xmin, ymin, xmax, ymax)
albany_bbox <- albany_boundary |>
  st_bbox() |>
  as.vector()

print(albany_bbox)
```

1.6.-.b Python

```python
# Define study area bounding box manually
# Manhattan example (xmin, ymin, xmax, ymax) in EPSG:4326
bbox_example = (-74.02169, 40.696423, -73.891338, 40.831263)

# Alternative: extract from existing boundary data
# boundary_gdf = gpd.read_file("your_boundary.shp")
# bbox_example = boundary_gdf.total_bounds
```

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.

1.7.-.a R

```r
# Download places data for Albany
overture_data(albany_bbox, "place", "albany_places_subset.parquet")

# Download buildings data
overture_data(albany_bbox, "building", "albany_buildings_subset.parquet")

# Download transportation segments
overture_data(albany_bbox, "segment", "albany_roads_subset.parquet")
```

1.7.-.b Python

```python
# Download buildings data for Manhattan
overture_data(bbox_example, "building", "nyc_buildings_subset.parquet")

# Download places data
overture_data(bbox_example, "place", "nyc_places_subset.parquet")

# Download transportation network
overture_data(bbox_example, "segment", "nyc_roads_subset.parquet")
```

1.8 Processing Downloaded Data

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

1.8.-.a R

```r
# Read the downloaded Parquet file
albany_places <- read_parquet("albany_places_subset.parquet")

# Convert to sf object for spatial operations
albany_places_sf <- st_as_sf(
  albany_places |> select(-sources),
  geometry = albany_places$geometry,
  crs = 4326
)

# Basic data exploration
print(paste("Downloaded", nrow(albany_places_sf), "places"))
print(colnames(albany_places_sf))
```

1.8.-.b Python

```python
# Read the downloaded data
manhattan = pd.read_parquet("nyc_buildings_subset.parquet")

# Convert to GeoDataFrame
manhattan_gdf = gpd.GeoDataFrame(
    manhattan.drop(columns="geometry"),
    geometry=shapely.wkb.loads(manhattan["geometry"]),
    crs=4326,
)

# Basic exploration
print(f"Downloaded {len(manhattan_gdf)} buildings")
print(manhattan_gdf.columns.tolist())
```

1.9 Data Visualization

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

1.9.-.a R

```r
# Interactive map using tmap (equivalent to folium)
albany_places_sf |>
  select(names$primary, categories$main, confidence) |>
  tm_shape() +
  tm_dots(col = "categories$main", size = 0.5, alpha = 0.8) +
  tm_view(view.legend.position = c("left", "bottom"))

# Simple quick visualization
```

```
albany_places_sf |>
  select(names$primary, categories$main, confidence) |>
  qtm(dots.col = "categories$main")
```

1.9.-.b Python

```
# Static plot using GeoPandas
manhattan_gdf.plot(figsize=(10, 10), alpha=0.7, column='categories',
legend=True)
plt.title("Manhattan Buildings from Overture Maps")
plt.show()

# Interactive map with folium (equivalent to tmap interactive)
m = folium.Map(location=[40.7589, -73.9851], zoom_start=12)
folium.GeoJson(
    manhattan_gdf.head(100),
    popup=folium.GeoJsonPopup(fields=['names', 'categories'])
).add_to(m)
m
```

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 on GitHub.

For more information about Overture Maps:

- Official Documentation
- Data Schema Reference
- Community Forum