Assignment for Vector Data Analysis

Max Davis

November 5, 2024

Part 1: Data Preparation using Google Colab

1. Reproject Vector Data

This step states reprojecting vector data of Moscow, sourced from OpenStreetMap, for compatibility with distance calculations in meters. The original data, in EPSG:4326 (WGS 84), is reprojected to EPSG:32637 (UTM Zone 37N). This projection is accurate metric-based distance measurements. The provided Python code automates this process by reading each shapefile, checking for any integer fields that may require conversion to strings (such as `OSM_ID`), and then reprojecting to EPSG:32637 if necessary. The reprojected files are saved in a dedicated folder: `/content/reprojected`.

2. Convert to GeoJSON

After reprojection, the shapefiles are converted to GeoJSON format in EPSG:4326. This is done to facilitate with web-based mapping tools in python. The code checks each file's coordinate system and reprojects it to EPSG:4326 if needed, before saving the GeoJSON files in `/content/Gjson`.

3. Code

The code verifies coordinate systems, performs reprojection as needed, and organizes the output files systematically for further processing in spatial analysis tools such as PostgreSQL/PostGIS and QGIS , CartoDB.

4. Visualization and Verification

The data was visualized in QGIS and CartoDB, Supabase, showing layers for boundaries, buildings, highways, points of interest, and railway stations, each represented with distinct symbols(Points,Lines) and colors.

Geospatial Data Visualization

- **boundary-polygon_4326**:
- **building-polygon_4326**:
- **highway-line_4326**:
- **poi-point_4326**:
- **railway-station-point_4326**:
- **Map Layer Overview**:

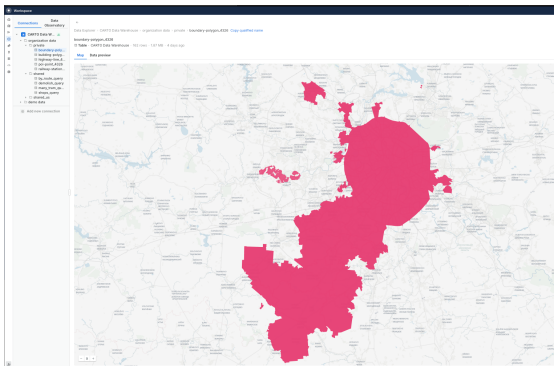

Figure 1: Boundary Polygon showing the administrative boundary of Moscow. This layer represents the outer limits and is used for spatial containment analysis.

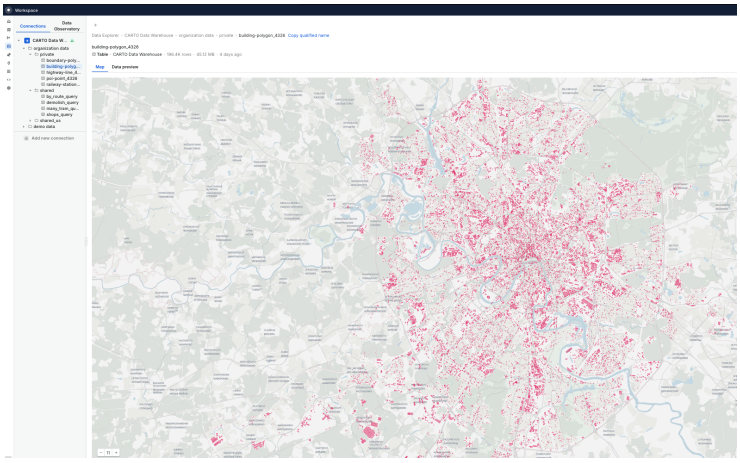

Figure 2: Building Polygon depicting individual buildings within the city of Moscow. This layer is essential for urban planning and infrastructure analysis.

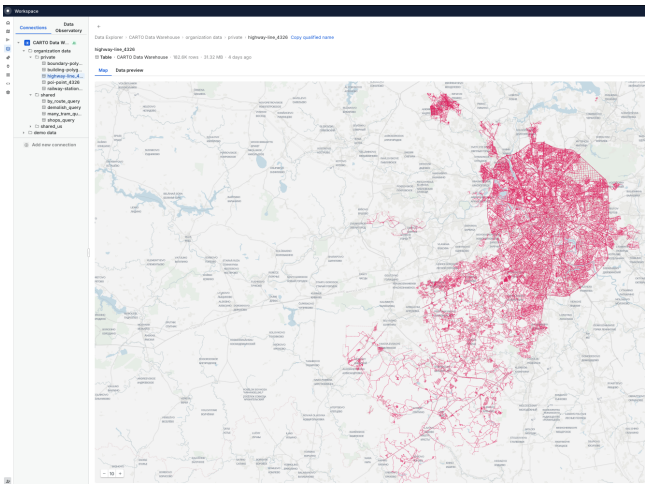

Figure 3: Highway Line layer showing the network of major roads and highways. Mainly for transportation analysis and route planning within Moscow.

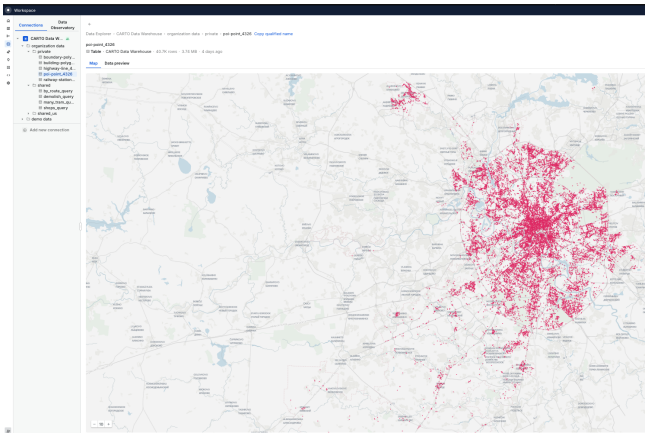

Figure 4: Points of Interest (POI) showing key locations such as shops, parks, and other significant places in Moscow. Important for accessibility and location-based analysis.

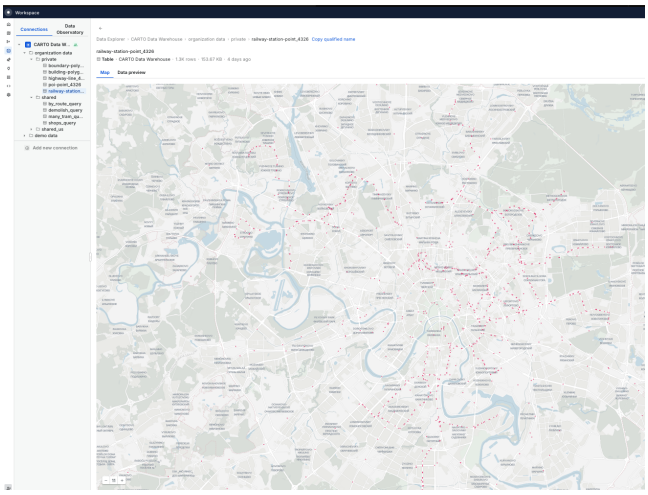

Figure 5: Railway Station Points representing the locations of railway stations in Moscow. Mainly for transit and proximity analysis within the transportation network.

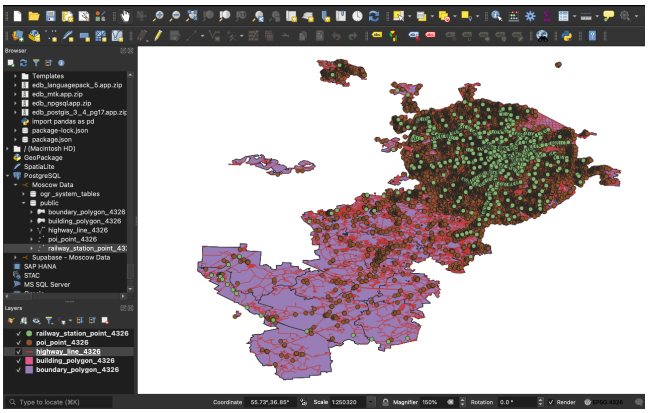

Figure 6: Overview of all layers displayed in QGIS, showcasing boundary polygons, building polygons, highway lines, points of interest, and railway station points.

Part 2: Data Analysis using Spatial SQL

In this section, I performed spatial analysis on Moscow's geospatial data using SQL queries in a PostgreSQL/PostGIS environment (IntelliJ IDEA mainly) .

Task 1: Urban Planning (Demolish Query)

Objective: Identify buildings and points of interest (POIs) that intersect with a newly planned road. This analysis helps in fixes which structures may need to be demolished for the new road construction.

SQL Query (demolish_query.sql):

```
SELECT p."osm_id" AS poi_id, p."name" AS poi_name, b."osm_id" AS building_id, p.geom
FROM building_polygon_4326 AS b
JOIN poi_point_4326 AS p ON ST_Intersects(b.geom, p.geom)
WHERE ST_Intersects(b.geom, p.geom);
```

Functionality: This query finds all POIs that intersect with buildings. By using these intersections, I can identify potential demolition sites for urban planning. The query joins the building and POI datasets using the `ST_Intersects` function, ensuring that only overlapping geometries are returned.

Visualization:

Task 2: Shop Accessibility (Shops Query)

Objective: Identify shops within 200 meters of tram stops, indicating shops accessible by public transport.

SQL Query (shops_query.sql):

```
SELECT DISTINCT p."osm_id" AS poi_id, p."shop" AS shop_type, p.geom AS poi_geom
FROM poi_point_4326 AS p
JOIN railway_station_point_4326 AS t ON ST_DWithin(p.geom, t.geom, 200)
WHERE p."shop" IS NOT NULL;
```

Functionality: The query finds shops that are located within 200 meters of tram stops. It uses the `ST_DWithin` function to filter POIs by distance, selecting only those that are within a walkable range to tram stations. This helps in assessing the accessibility of shops by public transportation.

Visualization:

Task 3: On the Road (By Route Query)

Objective: Identify highways within 100 meters of points of interest (POIs) along a specified route through Moscow.

SQL Query (byroute_query.sql):

```
SELECT h."osm_id" AS highway_id, h."name" AS highway_name, ST_Simplify(h.geom, 10) AS geom
FROM highway_line_4326 AS h
```

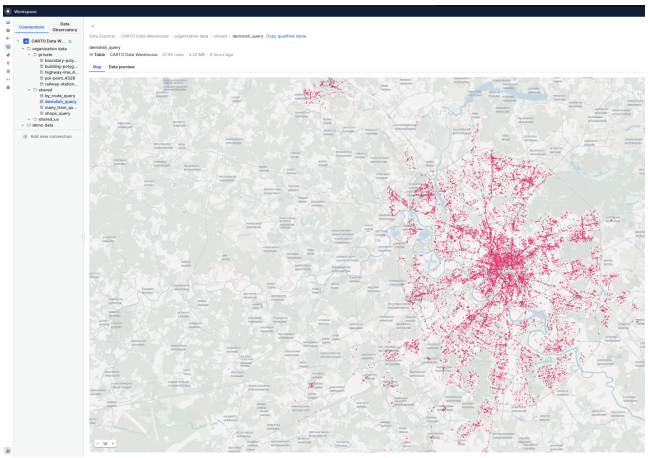

Figure 7: The map shows intersecting buildings and POIs that would need demolition for the new road. This information is needed for urban planning, as it highlights existing structures that might be affected by new infrastructure projects.

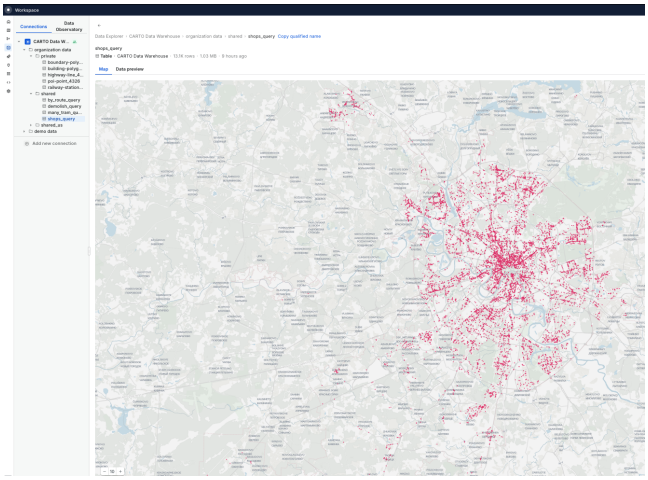

Figure 8: This map highlights the accessible shops within close proximity to tram stops, represented as points. The analysis is valuable for urban planning and public transport optimization, So this will show which areas have convenient access to commercial services.

```
JOIN poi_point_4326 AS p
ON h.geom && p.geom AND ST_DWithin(h.geom, p.geom, 100)
WHERE h."name" IS NOT NULL;
```

Functionality: The query finds highways close to POIs along a specific route. It uses spatial indexing with `&&` and the `ST_DWithin` function to identify highways within a 100-meter distance from POIs. The `ST_Simplify` function reduces geometry complexity, making it easier to render in QGIS.

Visualization:

Task 4: Popular Streets (Many Tram Query)

Objective: Identify the top 10 streets with the most tram stops within 100 meters, helping to determine popular transit corridors.

SQL Query (many_tram_query.sql):

```
SELECT h."name" AS street_name, COUNT(r."osm_id") AS tram_stop_count, ST_Centroid(h.geom) AS
FROM highway_line_4326 AS h
JOIN railway_station_point_4326 AS r ON ST_DWithin(h.geom, r.geom, 100)
WHERE h."name" IS NOT NULL
GROUP BY h."name", h.geom
ORDER BY tram_stop_count DESC
LIMIT 10;
```

Functionality: The query calculates the number of tram stops within 100 meters of each street and selects the top 10 streets with the highest counts. It groups by street name and uses `ST_Centroid` to represent the central location of each street on the map. This query highlights key streets based on tram accessibility.

Visualization:

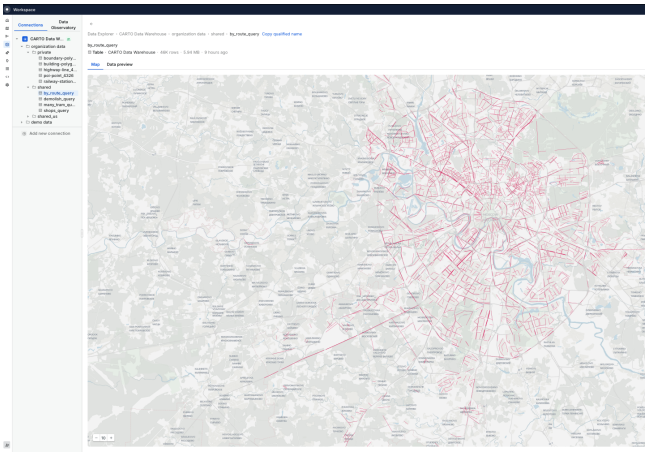

Figure 9: The map displays highways near the route, emphasizing road infrastructure close to notable locations. This analysis is beneficial for route planning, which lets an understanding of available road networks around key POIs.

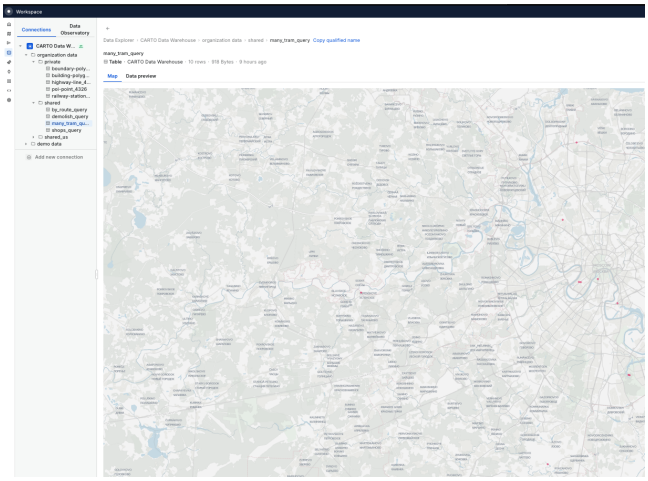

Figure 10: The map shows the top 10 streets with the highest number of nearby tram stops, useful for identifying popular areas with heavy transit activity.

Part 3: Kotlin Software Development

In this part, I implement a Kotlin project to recreate the "Shop Accessibility" task from Part 2 using 'h2gis', an embedded spatial database in a Kotlin environment.

Code Overview and Function Descriptions

- **Database Connection Functions:** The code includes two functions, `getPostgresConnection()` and `getH2GISConnection()`, for connecting to PostgreSQL and H2GIS databases, respectively. These functions streamline the connection process and simplify switching between the two databases.

```
fun getPostgresConnection(): Connection {
    val url = "jdbc:postgresql://localhost:5432/moscow_data"
    val user = "postgres"
    val password = "password"
    return DriverManager.getConnection(url, user, password)
}

fun getH2GISConnection(): Connection {
    val url = "jdbc:h2:mem:testdb;DB_CLOSE_DELAY=-1"  // In-memory H2 database
    return DriverManager.getConnection(url, "sa", "")
}
```

- **Data Fetching from PostgreSQL:** The `fetchShopsData()` function connects to PostgreSQL, runs the "Shop Accessibility" query, and fetches the result as a `ResultSet`. This data is later transferred into H2GIS.

```
fun fetchShopsData(pgConnection: Connection): ResultSet {
    val statement = pgConnection.createStatement()
    val query = """
        SELECT DISTINCT p."osm_id" AS poi_id, p."shop" AS shop_type, ST_AsText(p.ge
        FROM poi_point_4326 AS p
        JOIN railway_station_point_4326 AS t ON ST_DWithin(p.geom, t.geom, 200)
        WHERE p."shop" IS NOT NULL
    """
    return statement.executeQuery(query)
}
```

- **Data Import to H2GIS:** The `importDataToH2GIS()` function receives data from PostgreSQL, creates a table in H2GIS with a spatial geometry column, and inserts each row into this table.

```
fun importDataToH2GIS(h2Connection: Connection, pgResultSet: ResultSet) {
    val createTableQuery = """
        CREATE TABLE shops (
            poi_id VARCHAR,
            shop_type VARCHAR,
            geom GEOMETRY
        )
    """
    h2Connection.createStatement().execute(createTableQuery)

    while (pgResultSet.next()) {
        val poiId = pgResultSet.getString("poi_id")
        val shopType = pgResultSet.getString("shop_type")
        val geom = pgResultSet.getString("geom")

        val insertQuery = """
            INSERT INTO shops (poi_id, shop_type, geom)
            VALUES ('$poiId', '$shopType', ST_GeomFromText('$geom'))
        """
        h2Connection.createStatement().execute(insertQuery)
    }
}
```

- **Accessibility Analysis in H2GIS:** After importing the data into H2GIS, the `analyzeShopsAccessibility()` function runs a query to count accessible shops by type, ordering them by popularity. This output is printed to the console.

```
fun analyzeShopsAccessibility(h2Connection: Connection) {
    val query = """
        SELECT shop_type, COUNT(*) AS accessible_shops
        FROM shops
        GROUP BY shop_type
        ORDER BY accessible_shops DESC
    """
    val resultSet = h2Connection.createStatement().executeQuery(query)

    println("Accessible shops by type:")
    while (resultSet.next()) {
        val shopType = resultSet.getString("shop_type")
        val count = resultSet.getInt("accessible_shops")
        println("$shopType: $count")
    }
}
```

- **Main Function:** The main function orchestrates the entire workflow: it connects to PostgreSQL, fetches data, imports it into H2GIS, performs the accessibility analysis, and then prints the results.

```
fun main() {
    println("Connecting to PostgreSQL...")
    val pgConnection = getPostgresConnection()
    val pgResultSet = fetchShopsData(pgConnection)
    println("Fetched data from PostgreSQL.")

    println("Connecting to H2GIS...")
    val h2Connection = getH2GISConnection()
    importDataToH2GIS(h2Connection, pgResultSet)
    println("Data imported to H2GIS.")

    analyzeShopsAccessibility(h2Connection)

    pgConnection.close()
    h2Connection.close()
}
```

Output

After running the program, we obtain a list of accessible shops categorized by type, sorted in descending order based on count. Below is a sample of the output:

Shop Type	Count	Shop Type	Count	Shop Type	Count
convenience	2181	antiques	33	hifi	28
supermarket	1470	interior_decoration	31	mall	25
hairdresser	808	watches	30	bookmaker	23
clothes	804	photo	28	musical_instrument	23
florist	541	general	27	garden_centre	23
car_repair	505	tyres	27	houseware	23
beauty	468	ice_cream	22	second_hand	21
mobile_phone	358	tableware	21	laundry	20
kiosk	319	vacant	19	motorcycle	18
shoes	294	deli	16	fishing	16
alcohol	243	curtain	15	tea	15
jewelry	224	variety_store	15	shoe_repair	15
car	208	dairy	14	medical_supply	14
bakery	202	bag	13	hunting	13
books	197	religion	13	bed	12
ticket	190	seafood	12	radiotechnics	12
doityourself	183	wine	12	orthopedics	11
furniture	182	coffee	10	tattoo	9
pet	156	clock	8	frame	8
butcher	149	video	8	fishmonger	7
car_parts	145	music	7	hobby	6
toys	145	hearing_aids	8	anime	6
hardware	141	sewing	8	glaziery	7
electronics	136	organic	8	massage	6
optician	121	pawnbroker	58	translation	6
newsagent	106	confectionery	51	curtain	15
yes	102	copyshop	49	fabric	43
cosmetics	99	photo_studio	36	butcher	149
travel_agency	93	bakery	202	florist	541
gift	91	car	208	shop_type	145

Table 1: Accessible shops by type and count