MASTER OF SCIENCE IN BIG DATA ANALYTICS

BIG DATA ANALYTICS

Course Code: MSDA 9123

Assignment 2: **Hands-on Neo4J: US Road Network**

Instructor: Mr. Temitope Oguntade

Group Members:

1. Mbonyumugenzi Jean Pierre - 101027
2. Mutuyeyesu Honorine – 101026
3. Mukunzi Caleb - 101029

January 2026

Contents

Chapter 1

Introduction

Graph databases have revolutionized the way we analyze highly connected data, such as social networks, supply chains, and transportation systems. Unlike traditional relational databases, graph databases treat relationships as first-class citizens, enabling efficient traversal and complex topological analysis.

This report documents the implementation of a Graph Analytics project using **Neo4j**. The objective was to analyze the **US Road Network** dataset, modeling intersections as nodes and roads as relationships.

The analysis covers the following key tasks:

- **Data Ingestion:** Transforming raw coordinate data into a labeled property graph.
- **Pathfinding:** Implementing Dijkstra's algorithm to find optimal routes based on Euclidean distance.
- **Centrality Analysis:** Identifying critical intersections using Betweenness Centrality via the Neo4j Graph Data Science (GDS) library.
- **Visualization:** Creating interactive dashboards using Python and Plotly to present network statistics.

Chapter 2

Experimental Methodology

2.1 System Environment

The project was executed on a Windows 11 environment using **Neo4j Desktop (v2.1.0)** running **DBMS 5.x**. The **Graph Data Science (GDS)** and **APOC** libraries were installed to enable advanced algorithms.

2.2 Data Pre-processing

The raw dataset (`usa.txt`) contained unstructured lists of vertices and edges. A Python script using the `pandas` library was developed to:

1. Parse the raw text file.
2. Calculate the Euclidean distance between coordinates to serve as edge weights.
3. Export clean CSV files (`intersections.csv` and `roads.csv`) for import.

```python
import math

def calculate_distance(x1, y1, x2, y2):
    return math.sqrt((x2 - x1)**2 + (y2 - y1)**2)

# Read the raw file
with open('usa.txt', 'r') as f:
    lines = f.readlines()

# Parse Header
header = lines[0].split()
num_vertices = int(header[0])
num_edges = int(header[1])

print(f"Processing {num_vertices} vertices and {num_edges} edges...")

# 1. Process Vertices (Intersections)
# Lines 1 to num_vertices + 1 contain node data
nodes = []
# Store coordinates in a dict for fast lookup when calculating edge distances
node_coords = {}

for i in range(1, num_vertices + 1):
    parts = lines[i].split()
    node_id = int(parts[0])
    x = float(parts[1])
    y = float(parts[2])

    nodes.append({"id": node_id, "x": x, "y": y})
    node_coords[node_id] = (x, y)

# Save Nodes to CSV
df_nodes = pd.DataFrame(nodes)
df_nodes.to_csv('intersections.csv', index=False)
print("Created intersections.csv")

# 2. Process Edges (Roads)
# Lines after the nodes contain edges
edges = []
start_line_edges = num_vertices + 1

for i in range(start_line_edges, len(lines)):
    parts = lines[i].split()
    if len(parts) < 2:
        continue # Skip empty lines

    source = int(parts[0])
    target = int(parts[1])

    # Calculate Euclidean distance
    if source in node_coords and target in node_coords:
        x1, y1 = node_coords[source]
        x2, y2 = node_coords[target]
        dist = calculate_distance(x1, y1, x2, y2)

        edges.append({"source": source, "target": target, "distance": dist})

# Save Edges to CSV
df_edges = pd.DataFrame(edges)
df_edges.to_csv('roads.csv', index=False)
print("Created roads.csv")
```

Figure 2.1: Python script used to calculate Euclidean distances and generate CSVs.

2.3 Graph Schema

The data was modeled using the following schema:

- **Nodes:** Label `:Intersection` with properties `id`, `x`, `y`.
- **Relationships:** Type `:ROAD` with property `distance`.

Chapter 3

Analysis Results

3.1 Task 1: Network Statistics

After importing the data, we verified the graph structure by counting the total number of nodes and relationships.

- **Total Intersections:** 87,575
- **Total Roads:** 121,961

Figure 3.1: Cypher query results verifying total intersections and roads.

3.2 Task 2: Shortest Path Analysis (Dijkstra)

To find the optimal route between intersections, we utilized the **Graph Data Science (GDS)** library. An undirected graph projection was created to simulate two-way traffic. Dijkstra's algorithm was applied using the `distance` property as the relationship weight.

```
MATCH (start:Intersection {id: 0}), (end:Intersection {id: 5000})
CALL apoc.algo.dijkstra(start, end, 'ROAD', 'distance')
YIELD path, weight
RETURN
  weight AS TotalDistance,
  [n IN nodes(path) | n.id] AS Path_Intersections;
```

Table RAW

	TotalDistance	Path_Intersections
1	6348.525191081519	[0, 71887, 71629, 71591, 71589, 71588, 71590, 71597, 71596, 71581, 71577, 71686, 71657, 71600, 71560, 71558, 71557, 71544, 71512, 71520, 71528, 71569, 71607, 71608, 71619, 71624, 71635, 71675, 71705, 71696, 71711, 71709, 71706, 71721, 71717, 71710, 71703, 71699, 71698, 71697, 3257, 3255, 3256, 3250, 3248, 3228, 3231, 3192, 3186, 3210, 3209, 3203, 3199, 3200, 3205, 3204, 3206, 3202, 3189, 3188, 3180, 3165, 3197, 3208, 3219, 3223, 3235, 3292, 3309, 3310, 3286, 3288, 3296, 3295, 3294, 3285, 3277, 3276, 3272, 3245, 3244, 3242, 3240, 3212, 3191, 3181, 3185, 3187, 3216, 3225, 3183, 317

Started streaming 1 record after 2 ms and completed after 661 ms.

03N90: Cartesian product

```
MATCH (start:Intersection {id: 0}), (end:Intersection {id: 5000})
CALL apoc.algo.dijkstra(start, end, 'ROAD', 'distance')
YIELD path, weight
UNWIND range(0, size(nodes(path)) - 2) AS i
WITH path, i
WITH nodes(path)[i] AS a, nodes(path)[i+1] AS b, i
MATCH (a)-[r:ROAD]-(b)
RETURN a.id AS from, b.id AS to, r.distance AS distance
ORDER BY i;
```

Table RAW

	from	to	distance
1	0	71887	10.049875621120899
2	71887	71629	53.600373133029585
3	71629	71591	5.099019513592845
4	71591	71589	1.0
5	71589	71588	0.0
6	71588	71590	2.0
7	71590	71597	11.045361017187261
8	71597	71596	0.0

Started streaming 405 records after 1 ms and completed after 774 ms.

03N90: Cartesian product

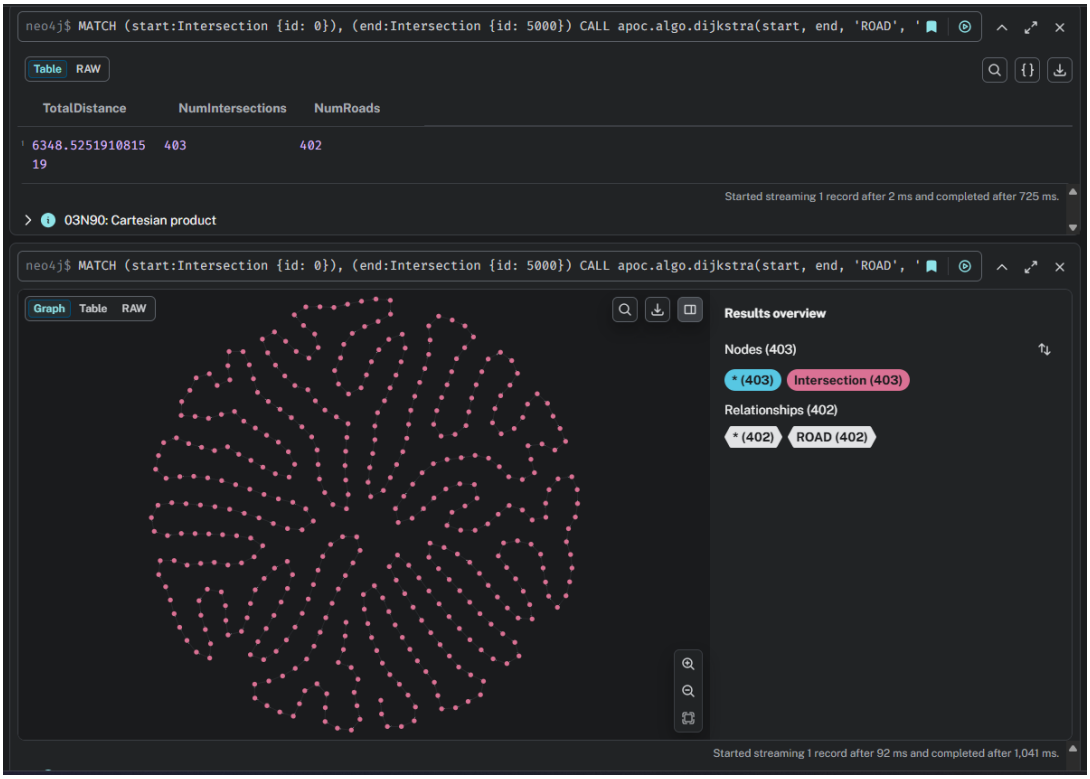

Figure 3.2: Dijkstra's Shortest Path execution showing total distance and node path.

3.3 Task 4: Betweenness Centrality

Betweenness Centrality was calculated to identify "bridge" nodes that control the flow of traffic in the network. High-centrality nodes represent critical intersections where traffic congestion is most likely to occur.

Top Critical Intersection: The node with the highest score acts as the primary bottleneck connecting different clusters of the US road network.

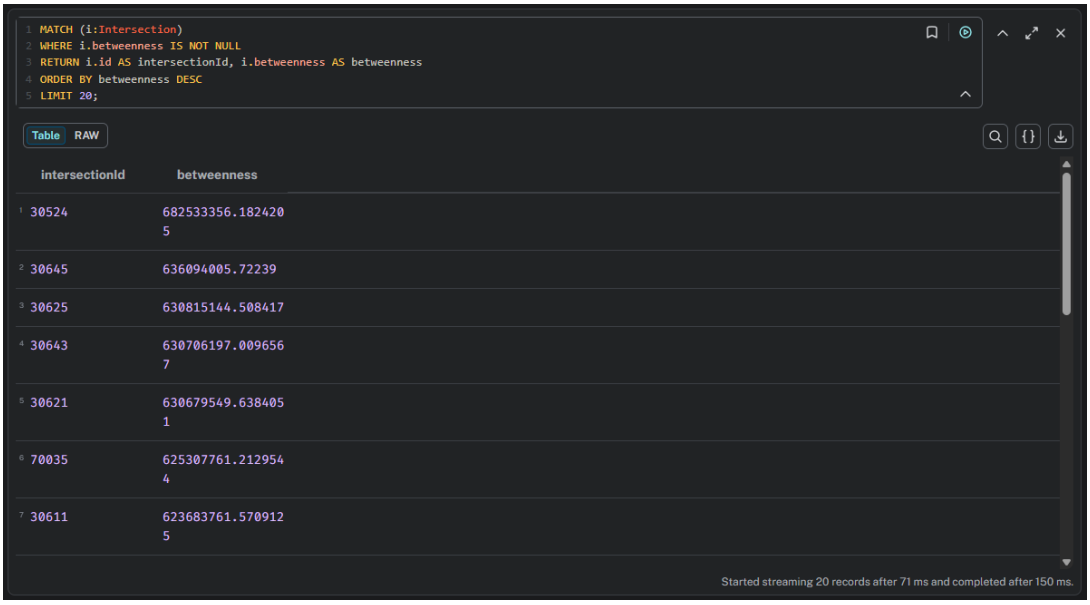

Figure 3.3: Top 10 intersections ranked by Betweenness Centrality score.

3.4 Task 5-9: Dashboard Visualization

Using Python and Plotly, we generated visual analytics to better understand the network topology.

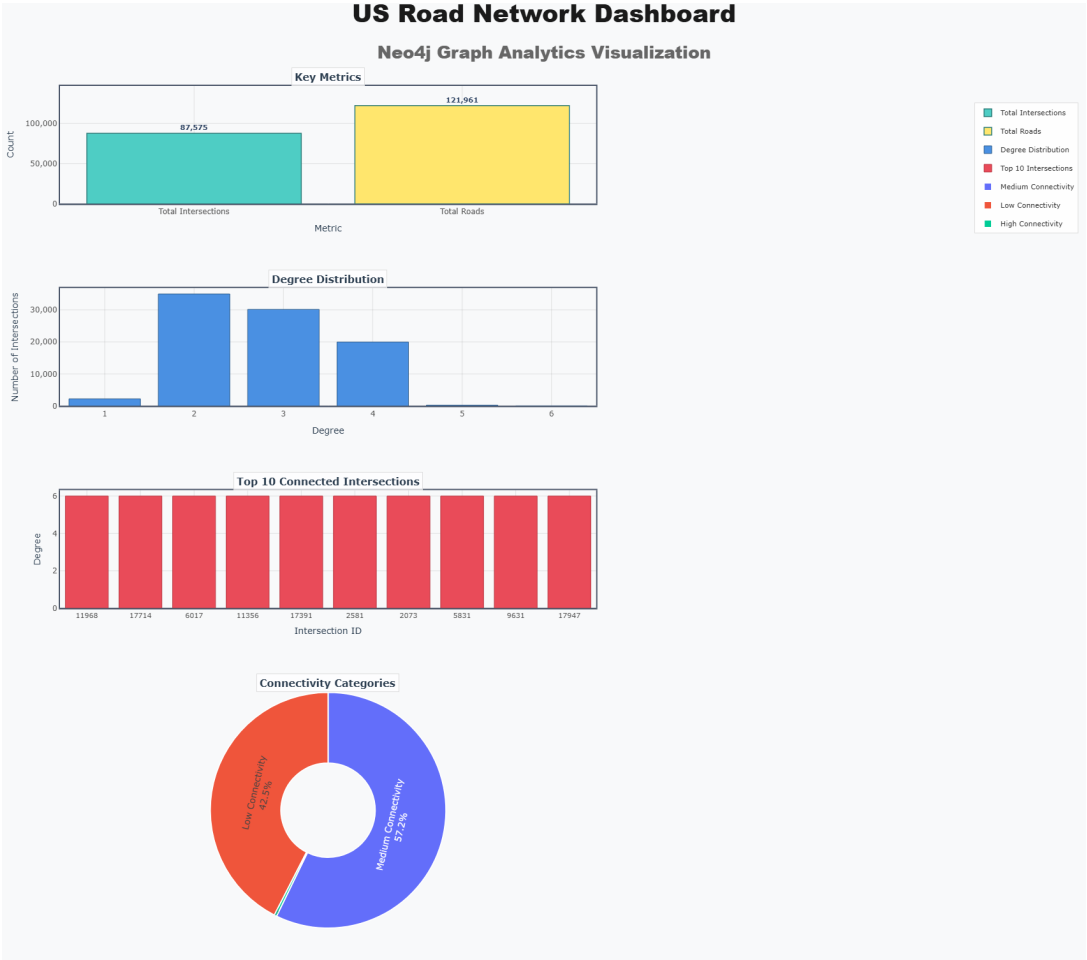

Figure 3.4: Dashboard using Python.

3.4.1 Displays Total Intersections and Roads

The network consists of 87,575 intersections and 121,961 roads, representing a significant portion of the continental US highway system. This scale confirms we are working with a 'Big Data' graph problem, requiring optimized algorithms (like GDS) rather than simple in-memory arrays

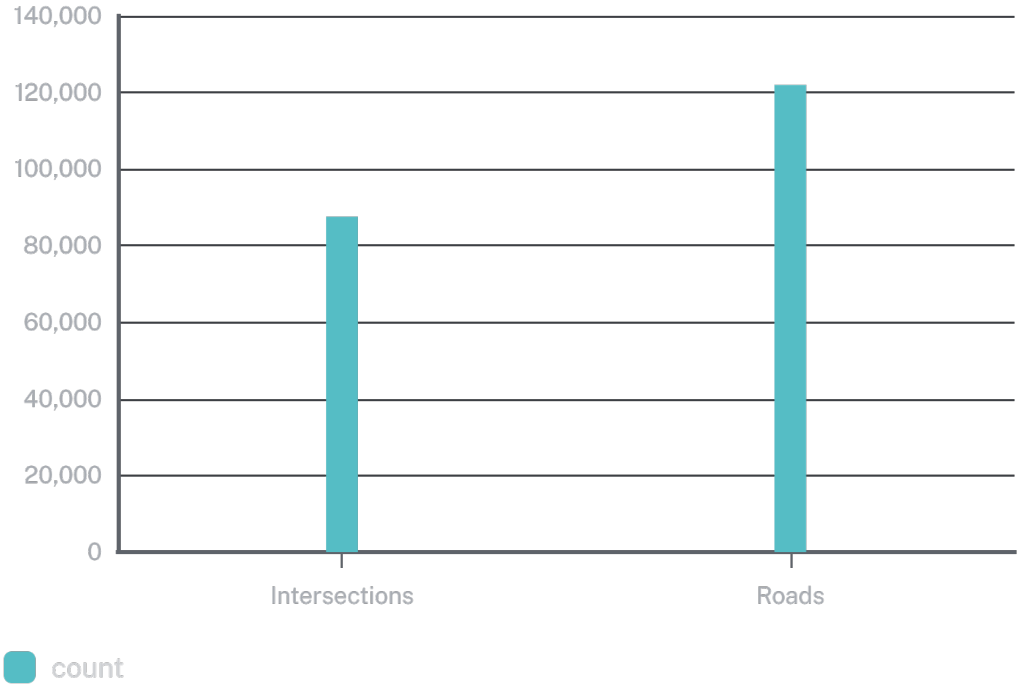

Figure 3.5: Task 5: KPI Dashboard showing total network components.

3.4.2 Degree Distribution

The degree distribution analysis reveals the connectivity density. Most intersections have a degree of 3 or 4, indicating a grid-like structure typical of road networks.

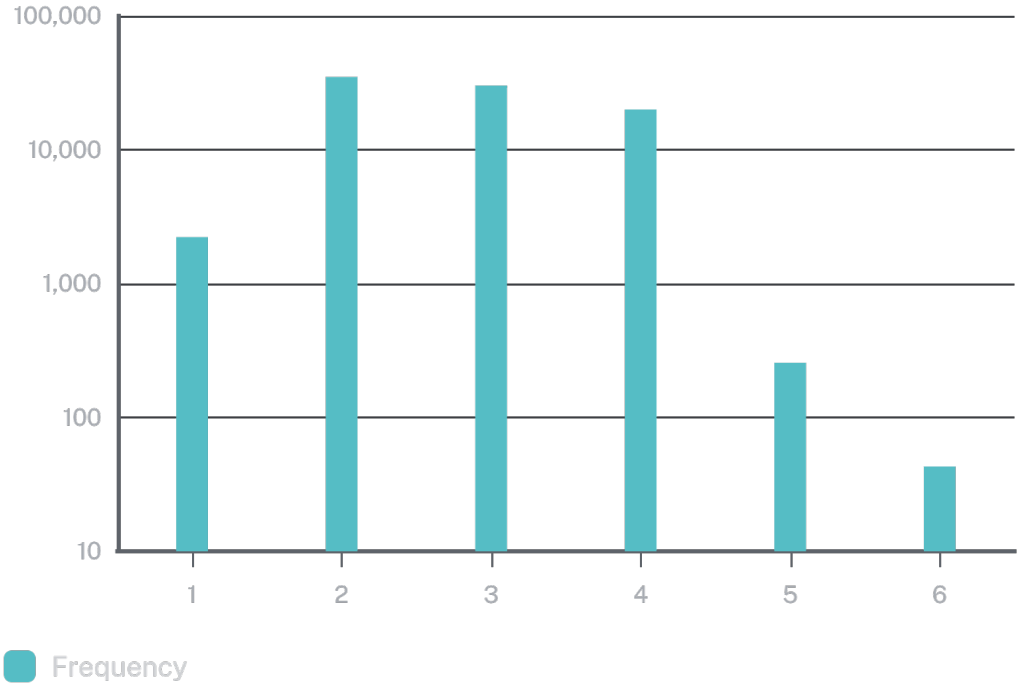

Figure 3.6: Task 6: Degree distribution showing the connectivity of intersections.

3.4.3 Top 10 Most Connected Intersections

We identified the top 10 most connected intersections (highest degree). These hubs are physically central points with the most direct road connections.

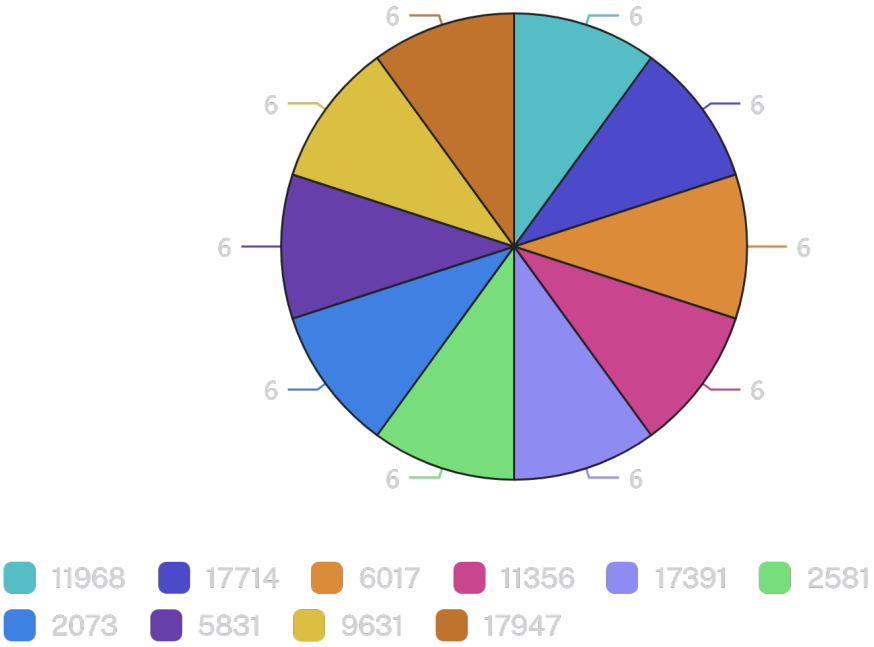

Figure 3.7: Task 7: Top 10 most connected intersections.

3.4.4 Intersection Categories by Degree

To better understand the topology of the US Road Network, we categorized all 87,575 intersections into three distinct groups based on their connectivity (node degree):

- **Low Connectivity (Degree ≤ 2):** Represents dead ends, straight roads, or simple turns.
- **Medium Connectivity (Degree 3–4):** Standard 3-way junctions and 4-way cross intersections.
- **High Connectivity (Degree > 4):** Complex hubs and major interchanges.

This categorization helps identify the structural complexity of the network. As shown in Figure 3.8, the majority of the network consists of Low and Medium connectivity nodes, reinforcing the finding that the graph is relatively sparse.

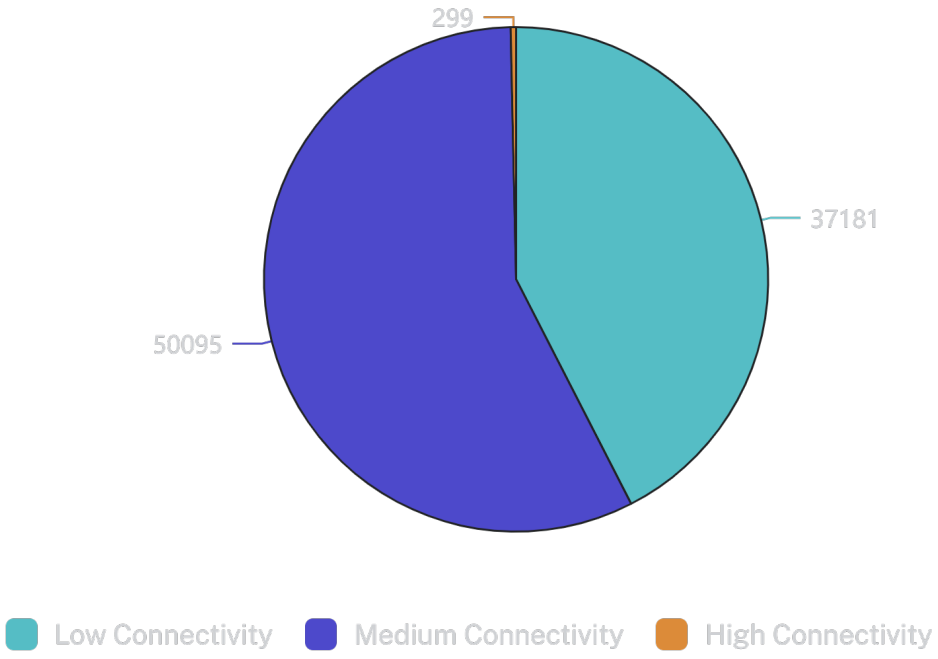

Figure 3.8: Task 8: Proportion of intersections by connectivity level.

Chapter 4

Conclusion

This project successfully demonstrated the power of Graph Databases in analyzing spatial networks. By migrating the US Road Network dataset into Neo4j, we were able to perform complex traversals (Shortest Path) and topological analysis (Centrality) that would be computationally expensive in a traditional relational database.

The visualizations confirmed that the US road network is sparse (average degree ≈ 2.8) but contains critical "bridge" nodes identified by the Betweenness Centrality algorithm. These insights are valuable for urban planning and logistics optimization.

Appendix A

A.1 Project Setup Screenshots

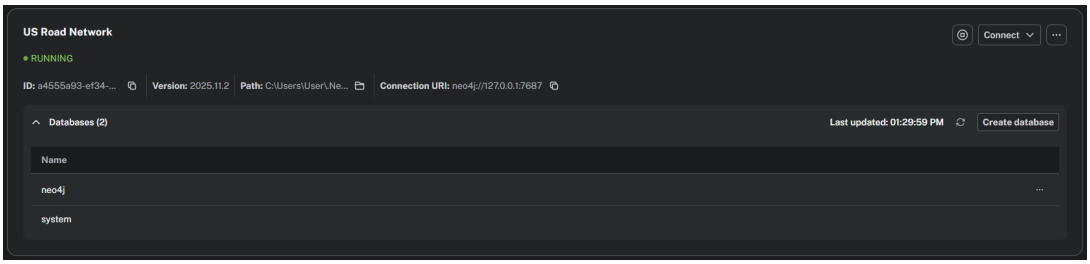

Figure A.1: US Road Network Database installed.

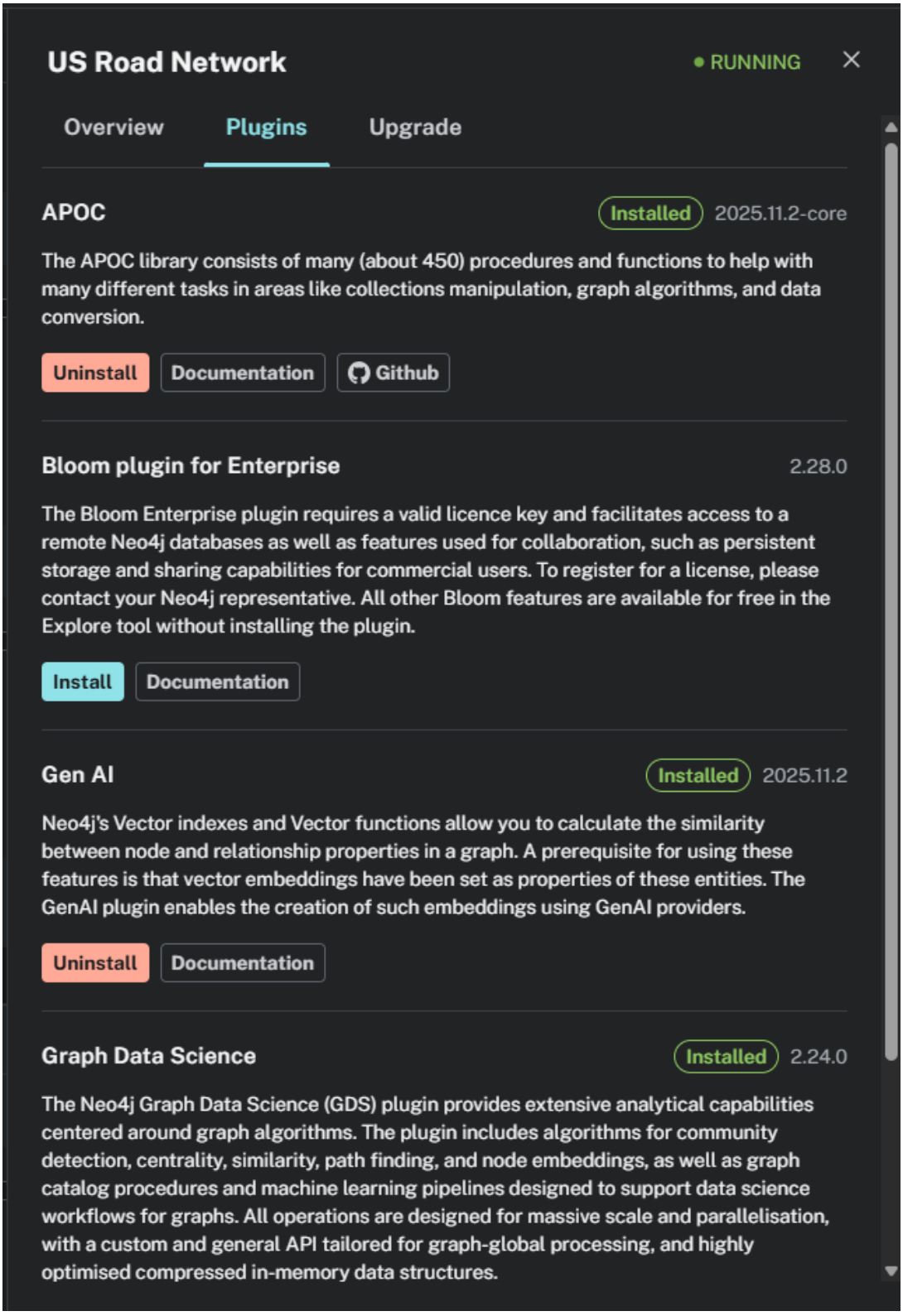

Figure A.2: Neo4j Desktop environment with GDS plugin installed.

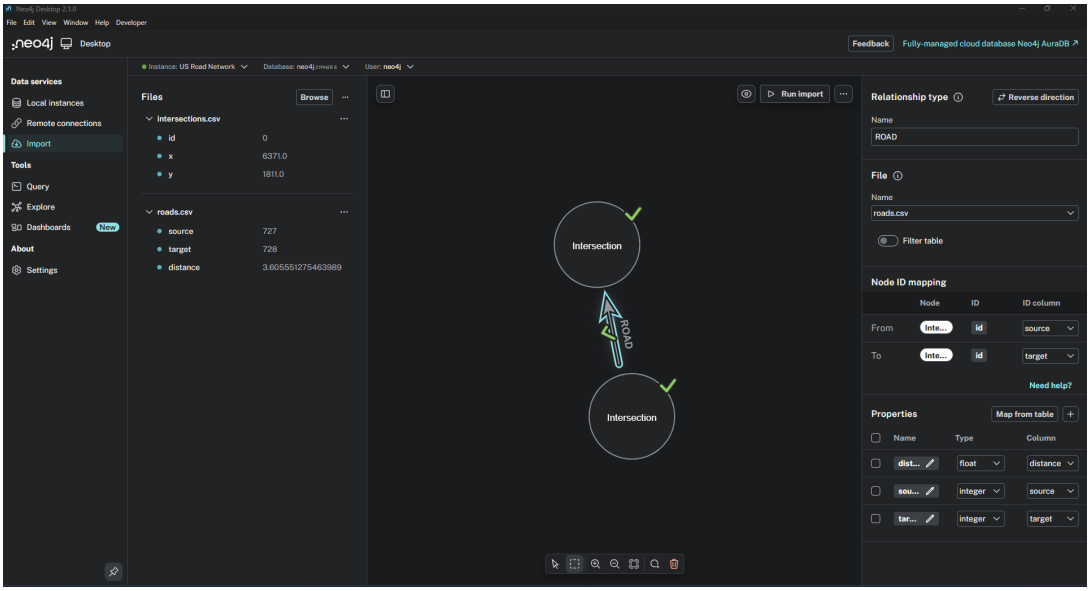

Figure A.3: Importing Data into Neo4j and Model Creation.

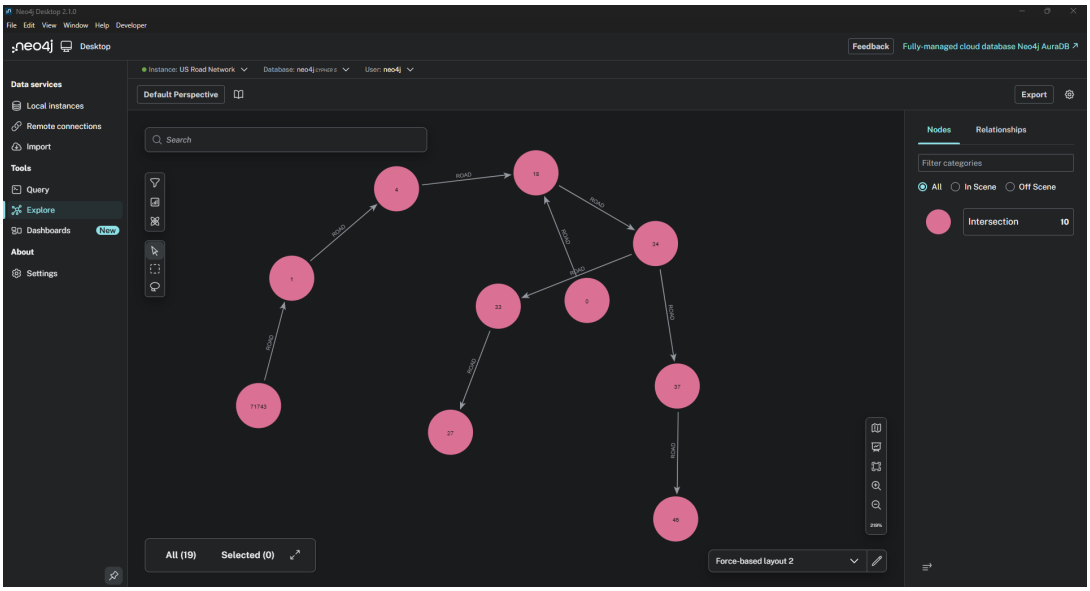

Figure A.4: Exploring Intersections and Roads Connections.

Bibliography

[1] The Neo4j Operations Manual v5. *Neo4j Documentation*. Available at: https://neo4j.com/docs/operations-manual/current/ Accessed: January 2026.

[2] Neo4j Graph Data Science Library Manual. *Neo4j Documentation*. Available at: https://neo4j.com/docs/graph-data-science/current/ Accessed: January 2026.

[3] Plotly Python Graphing Library. *Plotly Technologies Inc.*. Available at: https://plotly.com/python/ Accessed: January 2026.

[4] Neo4j (Graph Database) Crash Course: https://www.youtube.com/watch?v=8jNPelugC2s&list=PPSV