ADVENTIST UNIVERSITY
OF CENTRAL AFRICA

Big Data Analytics

Course Code: MSDA 9215

Group Names
NTAWUYIRUSHA Emmanuel 101287
Javan NGIRIMANA 101415
KABERUKA NSABIMANA Augustin 101061

Lecturer:
Temitope Oguntade

Assignment:
Hands-on Neo4J: US Road Network
Group work

January 2026

Contents

US Road Network Analysis Using Neo4j and Python

1 Introduction

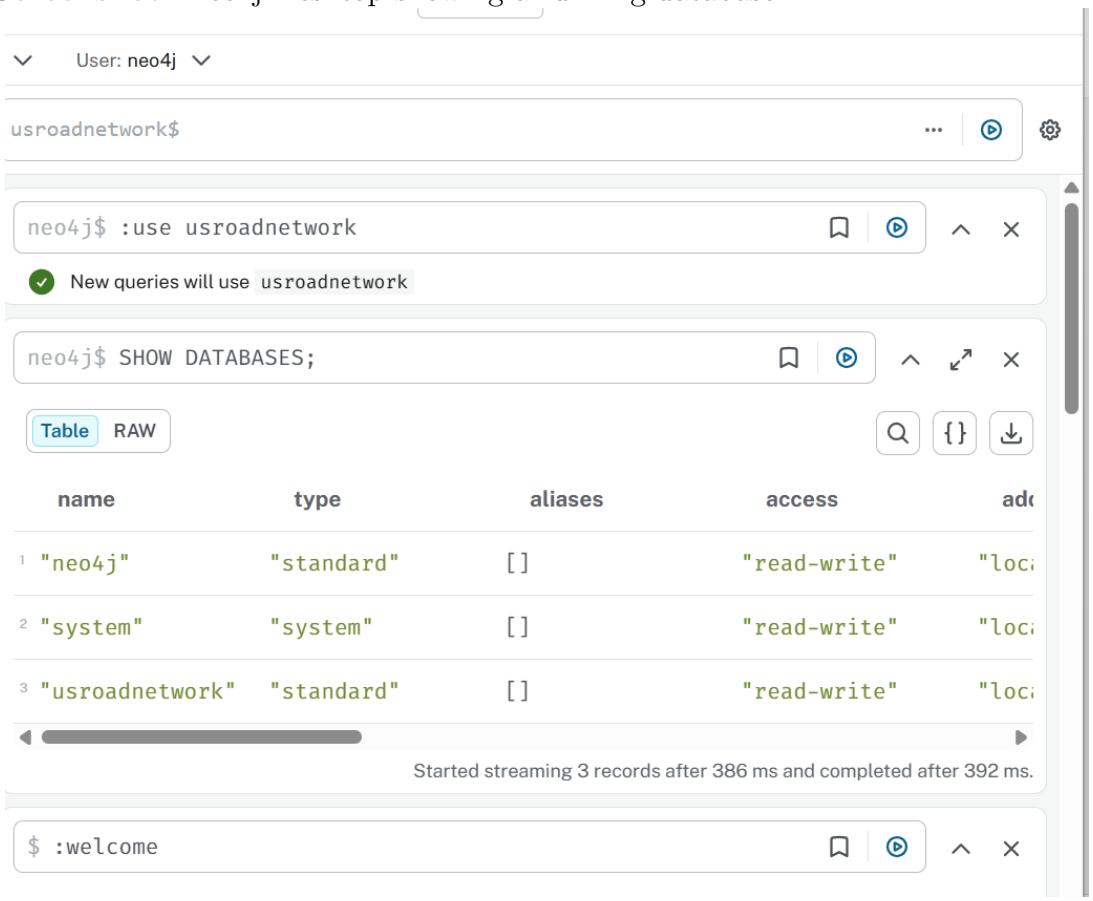
This report presents an analysis of the US Road Network using Neo4j for graph storage and querying, and Python (NetworkX and Plotly) for advanced analytics and visualization. The goal is to understand network connectivity, identify critical intersections, and analyze structural properties of a large-scale road network.

2 Neo4j Desktop Login and Setup

Neo4j Desktop was used as the graph database environment.

- Open Neo4j Desktop and log in.
- Create and start the database named `usroadnetwork`.
- Open Neo4j Browser to execute Cypher queries.

Screenshot: Neo4j Desktop showing a running database.

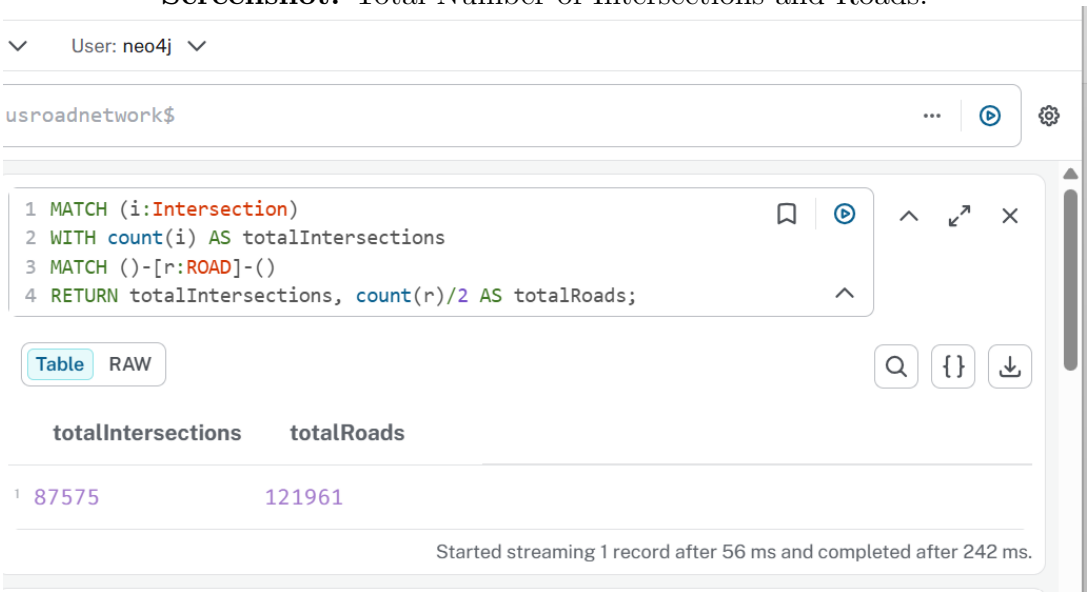


3 Task 1: Total Number of Intersections and Roads

Cypher Query:

```
MATCH (i:Intersection)
WITH count(i) AS totalIntersections
MATCH ()-[r:ROAD]->()
RETURN totalIntersections, count(r) AS totalRoads;
```

Screenshot: Total Number of Intersections and Roads.



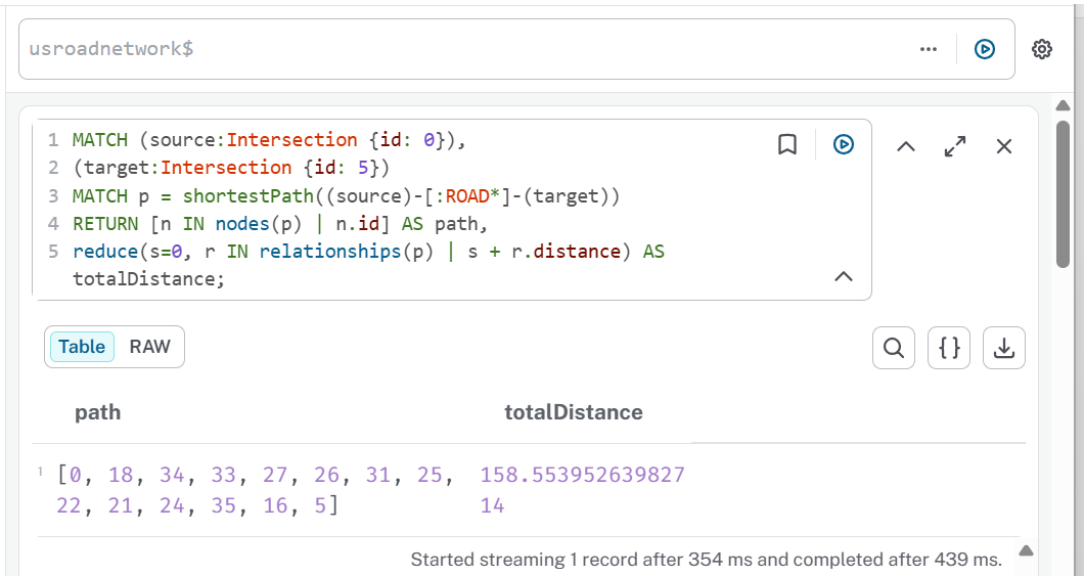
Interpretation: The US road network contains 87,575 intersections and 121,961 roads, forming a large but sparse graph. The average degree is approximately 2.8, indicating that most intersections connect to only a few roads. This structure is typical of real-world road networks and supports efficient analysis using graph algorithms such as shortest path and centrality.

4 Task 2: Shortest Path Between Two Intersections

Cypher Query:

```
MATCH (source:Intersection {id: 0}), (target:Intersection {id:
   5})
MATCH p = shortestPath((source)-[:ROAD*]-(target))
RETURN [n IN nodes(p) | n.id] AS path,
reduce(s=0, r IN relationships(p) | s + r.distance) AS
   totalDistance;
```

Screenshot: Shortest Path Between Two Intersections.



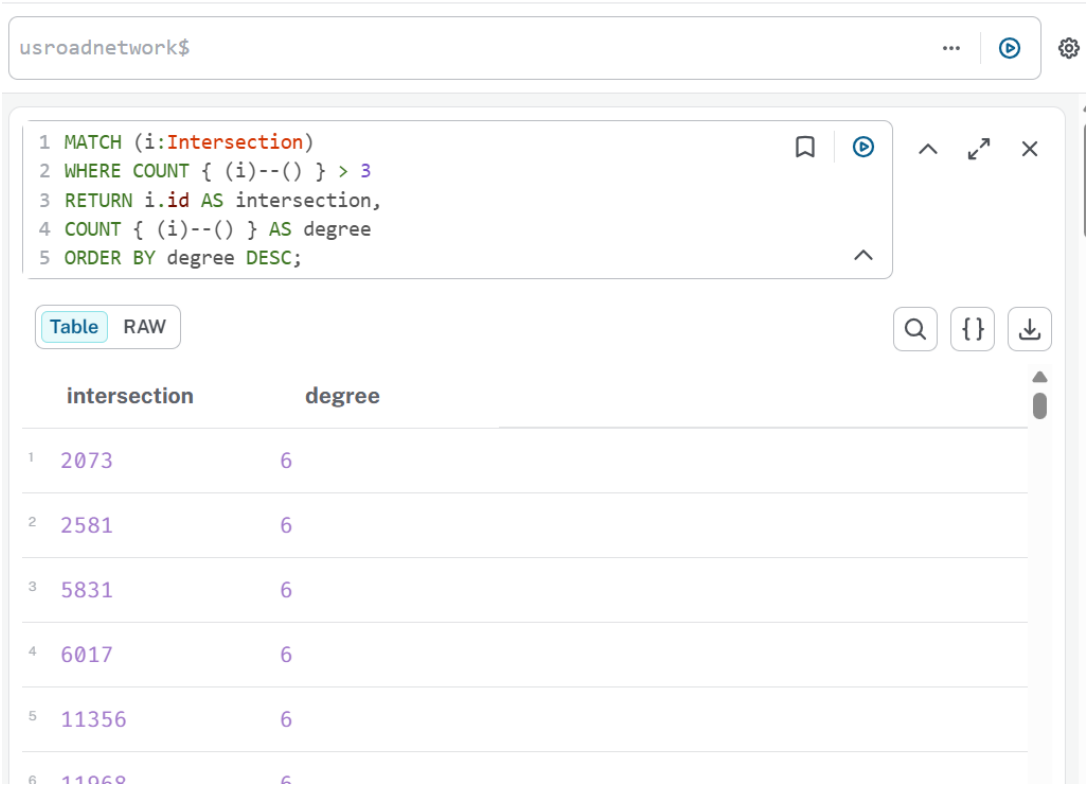
Interpretation: The shortest path between the selected source and destination intersections consists of 14 intersections, following the route $[0 \rightarrow 18 \rightarrow 34 \rightarrow 33 \rightarrow 27 \rightarrow 26 \rightarrow 31 \rightarrow 25 \rightarrow 22 \rightarrow 21 \rightarrow 24 \rightarrow 35 \rightarrow 16 \rightarrow 5]$, with a total travel distance of approximately 158.55 units. This path represents the most efficient route in terms of Euclidean distance, demonstrating how graph-based shortest-path algorithms can identify optimal navigation routes in a road network.

5 Task 3: Intersections with Degree Greater Than Threshold

Cypher Query:

```
MATCH (i:Intersection)
WITH i, COUNT { (i)--() } AS degree
WHERE degree > 3
RETURN i.id AS intersection, degree
ORDER BY degree DESC;
```

Screenshot: Intersections with Degree Greater Than Threshold.



Interpretation: The query identified more than 5,000 intersections with a degree greater than 3, indicating that a significant number of intersections are connected to multiple roads. The fetch limit of 5,000 records was reached, meaning the actual number of such intersections is higher. This highlights the presence of moderately connected junctions that play an important role in maintaining network connectivity.

6 Task 4: Betweenness Centrality Analysis

Due to the absence of the Graph Data Science library in Neo4j Community Edition, betweenness centrality was computed using Python's NetworkX library.

Python Code:

```
import networkx as nx
import pandas as pd

edges = pd.read_csv(
    r"C:\Users\ntawe\Documents\Data Science\15 Big Data Analytics
       \Assignment2\edges.csv"
)

G = nx.from_pandas_edgelist(edges, 'source', 'target')

# Approximate using sampling
bc = nx.betweenness_centrality(
    G,
    k=1000,           # number of sampled nodes
    normalized=True,
    seed=42
)
```

```
top10 = sorted(bc.items(), key=lambda x: x[1], reverse=True)[:10]

for node, score in top10:
    print(node, score)
```

Screenshot: Betweenness Centrality Analysis.

[9]:
```
import networkx as nx
import pandas as pd

edges = pd.read_csv(
    r"C:\Users\ntawe\Documents\Data Science\15 Big Data Analytics\Assignment2\edges.csv"
)

G = nx.from_pandas_edgelist(edges, 'source', 'target')

# Approximate using sampling
bc = nx.betweenness_centrality(
    G,
    k=1000,          # number of sampled nodes
    normalized=True,
    seed=42
)

top10 = sorted(bc.items(), key=lambda x: x[1], reverse=True)[:10]

for node, score in top10:
    print(node, score)
```

```
30524 0.17863774862248055
30645 0.16913217356805177
30625 0.16731069029126813
30643 0.16729514672828044
30621 0.16729131482121
30611 0.1651391576689666
30577 0.16513725076186708
30620 0.1647894595882502
30618 0.16478178823452905
30640 0.16477940426068616
```

Interpretation: The betweenness centrality analysis identified a small set of intersections with significantly higher centrality values compared to the rest of the network. The top intersections include nodes 30524, 30645, 30625, 30643, and 30621, with betweenness centrality scores ranging from approximately 0.16 to 0.18. Because the computation used an approximate betweenness centrality method (sampling with k=1000), the results provide a reliable estimation while remaining computationally feasible for a large-scale network. Despite being approximate, the analysis successfully highlights the most influential intersections in terms of network flow and resilience. Betweenness centrality reveals a small number of intersections that act as critical bridges in the road network, with high scores indicating key bottlenecks for shortest-path connectivity.

7 Task 5: Interactive Plotly Dashboard

Neo4j was connected to Python using the official Neo4j driver, and Plotly was used to create an interactive dashboard.

Python Code:

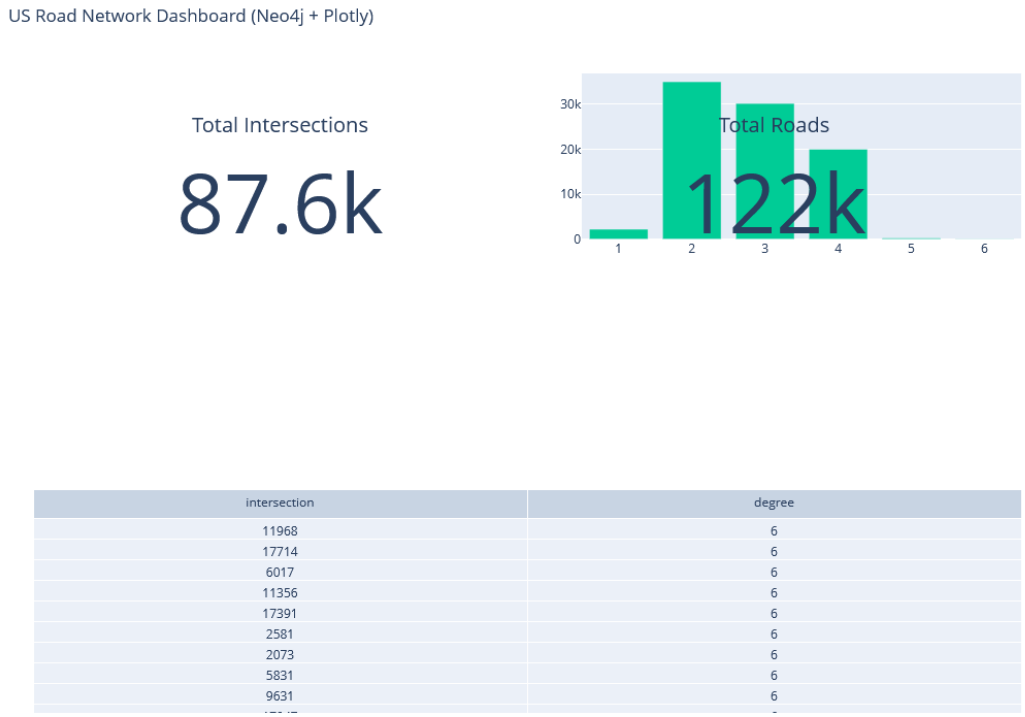
```
```

```
# loading necessary package
from neo4j import GraphDatabase
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from getpass import getpass
import networkx as nx


# Logging to Neo4j
URI = "bolt://localhost:7687"
USER = "neo4j"
PASSWORD = "my password "#i used my password to be connected
DB = "usroadnetwork"

driver = GraphDatabase.driver(URI, auth=(USER, PASSWORD))


#
# Query Neo4j Metrics Automatically
## Total Intersections
def get_total_intersections(tx):
    result = tx.run("MATCH (i:Intersection) RETURN count(i) AS
       total")
    return result.single()["total"]

## Total Roads
def get_total_roads(tx):
    result = tx.run("MATCH ()-[r:ROAD]->() RETURN count(r) AS
       total")
    return result.single()["total"]

## Degree Distribution
def get_degree_distribution(tx):
    query = """
    MATCH (i:Intersection)
    RETURN COUNT { (i)--() } AS degree, count(*) AS frequency
    ORDER BY degree
    """
    return pd.DataFrame(tx.run(query).data())

## Top 10 Most Connected Intersections
def get_top10(tx):
    query = """
    MATCH (i:Intersection)
    RETURN i.id AS intersection, COUNT { (i)--() } AS degree
    ORDER BY degree DESC
    LIMIT 10
    """
    return pd.DataFrame(tx.run(query).data())
```

```
## Execute Queries
with driver.session(database=DB) as session:
    total_intersections = session.execute_read(
       get_total_intersections)
    total_roads = session.execute_read(get_total_roads)
    degree_df = session.execute_read(get_degree_distribution)
    top10_df = session.execute_read(get_top10)

# Generate dashboard
# Build Plotly Dashboard
## KPI Cards
fig = go.Figure()

fig.add_trace(go.Indicator(
    mode="number",
    value=total_intersections,
    title={"text": "Total Intersections"},
    domain={'x': [0, 0.5], 'y': [0.6, 1]}
))

fig.add_trace(go.Indicator(
    mode="number",
    value=total_roads,
    title={"text": "Total Roads"},
    domain={'x': [0.5, 1], 'y': [0.6, 1]}
))

## Degree Distribution Bar Chart
fig.add_trace(go.Bar(
    x=degree_df["degree"],
    y=degree_df["frequency"],
    name="Degree Distribution",
    xaxis="x2",
    yaxis="y2"
))

## Top 10 Intersections Table
fig.add_trace(go.Table(
    header=dict(values=list(top10_df.columns)),
    cells=dict(values=[top10_df[col] for col in top10_df.columns
       ]),
    domain={'x': [0, 1], 'y': [0, 0.35]}
))

## Layout
fig.update_layout(
    title="US Road Network Dashboard (Neo4j + Plotly)",
    grid={'rows': 3, 'columns': 2, 'pattern': "independent"},
    height=800
)
```

```
fig.show()
```

Screenshot: Interactive Plotly Dashboard



intersection	degree
11968	6
17714	6
6017	6
11356	6
17391	6
2581	6
2073	6
5831	6
9631	6

This dashboard was developed using Python to visualize key metrics from a Neo4j graph database representing a road network.
First, a secure connection was established between Python and Neo4j using the official Neo4j Python Driver (Bolt protocol). Cypher queries were executed directly from Python to extract graph metrics such as the total number of intersections (nodes), total roads (relationships), node degree distribution, and highly connected intersections.
The extracted data was then processed using Pandas and visualized using Plotly, enabling the creation of an interactive dashboard. The dashboard includes KPI indicators, bar charts, and tabular summaries, providing an intuitive and real-time view of the road network structure.
This integration demonstrates how graph databases and Python analytics can be combined to support exploratory network analysis and decision-making.
Interpretation: The dashboard provides real-time insights into network size, degree distribution, and key intersections.

8 Task 6: Degree Distribution

Cypher Query:

```
MATCH (i:Intersection)
RETURN COUNT { (i)--() } AS degree, count(*) AS frequency
ORDER BY degree;
```

Screenshot: Degree Distribution

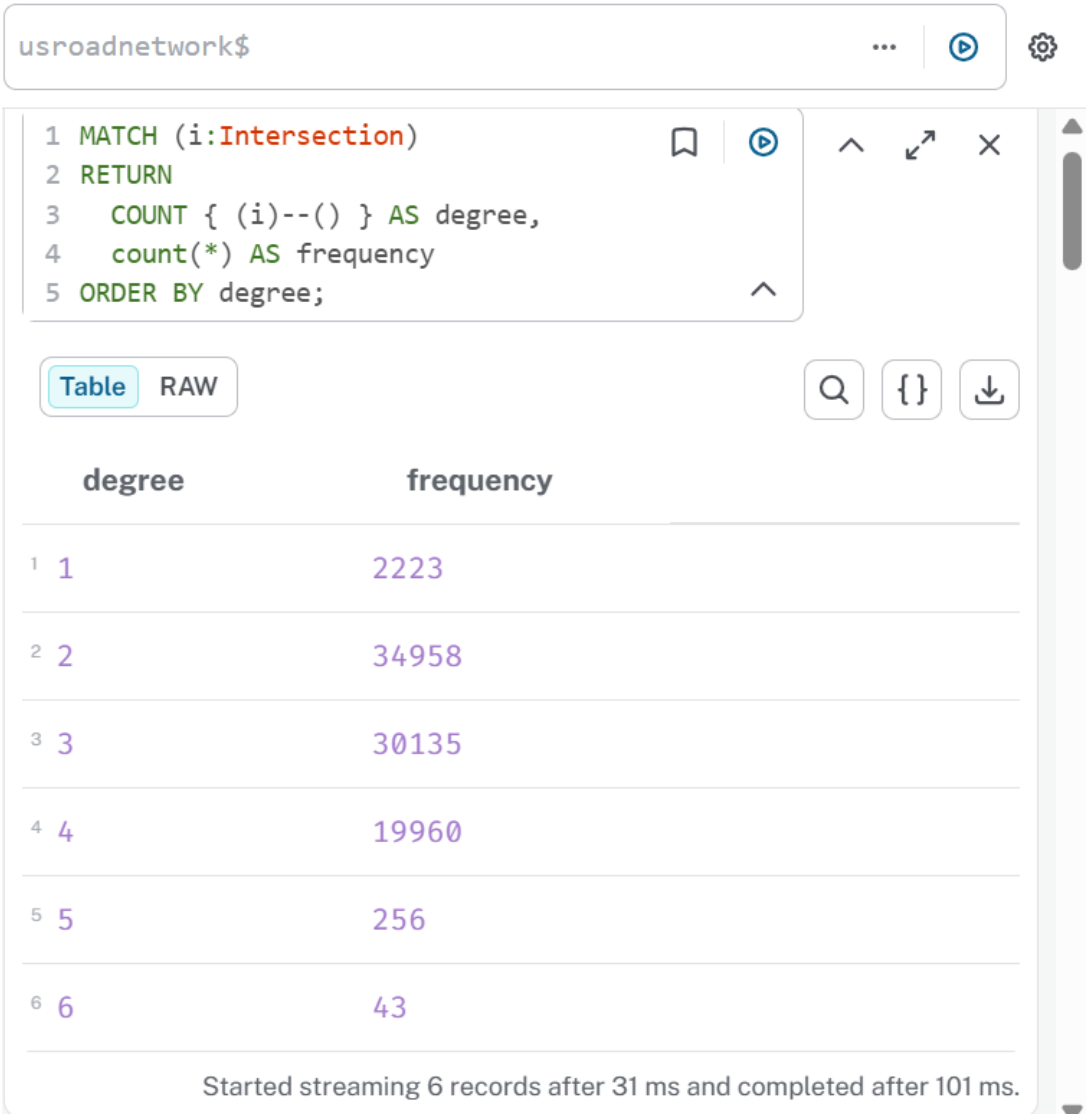


Table 1: Degree Distribution of Intersections

Degree (Connections)	Number of Intersections
1	2,223
2	34,958
3	30,135
4	19,960
5	256
6	43

Interpretation: The degree distribution analysis shows that most intersections in the road network have between two and four connecting roads, which is characteristic of real-world transportation networks. Only a small number of intersections exhibit high degrees, indicating the absence of dominant hubs. This reflects the spatial and structural constraints of road networks and suggests a decentralized and resilient network topology.

9 Task 7: Top 10 Most Connected Intersections

Cypher Query:

```
MATCH (i:Intersection)
RETURN i.id AS intersection, COUNT { (i)--() } AS degree
ORDER BY degree DESC
LIMIT 10;
```

Table 2: Top 10 Most Connected Intersections

Rank	Intersection ID	Degree
1	11968	6
2	17714	6
3	6017	6
4	11356	6
5	17391	6
6	2581	6
7	2073	6
8	5831	6
9	9631	6
10	17947	6

Screenshot: Top 10 Most Connected Intersections

User: neo4j

usroadnetwork$

```
1 MATCH (i:Intersection)
2 RETURN
3   i.id AS intersection,
4   COUNT { (i)--() } AS degree
5 ORDER BY degree DESC
6 LIMIT 10;
```

Table RAW

	intersection	degree
1	11968	6
2	17714	6
3	6017	6
4	11356	6
5	17391	6
6	2581	6
7	2073	6
8	5831	6
9	9631	6
10	17947	6

10 Task 8: Intersection Categories by Degree

Cypher Query:

```
MATCH (i:Intersection)
WITH COUNT { (i)--() } AS degree
RETURN
CASE
  WHEN degree <= 2 THEN 'Low'
  WHEN degree <= 4 THEN 'Medium'
  ELSE 'High'
END AS category,
count(*) AS total;
```

Screenshot: Intersection Categories by Degree

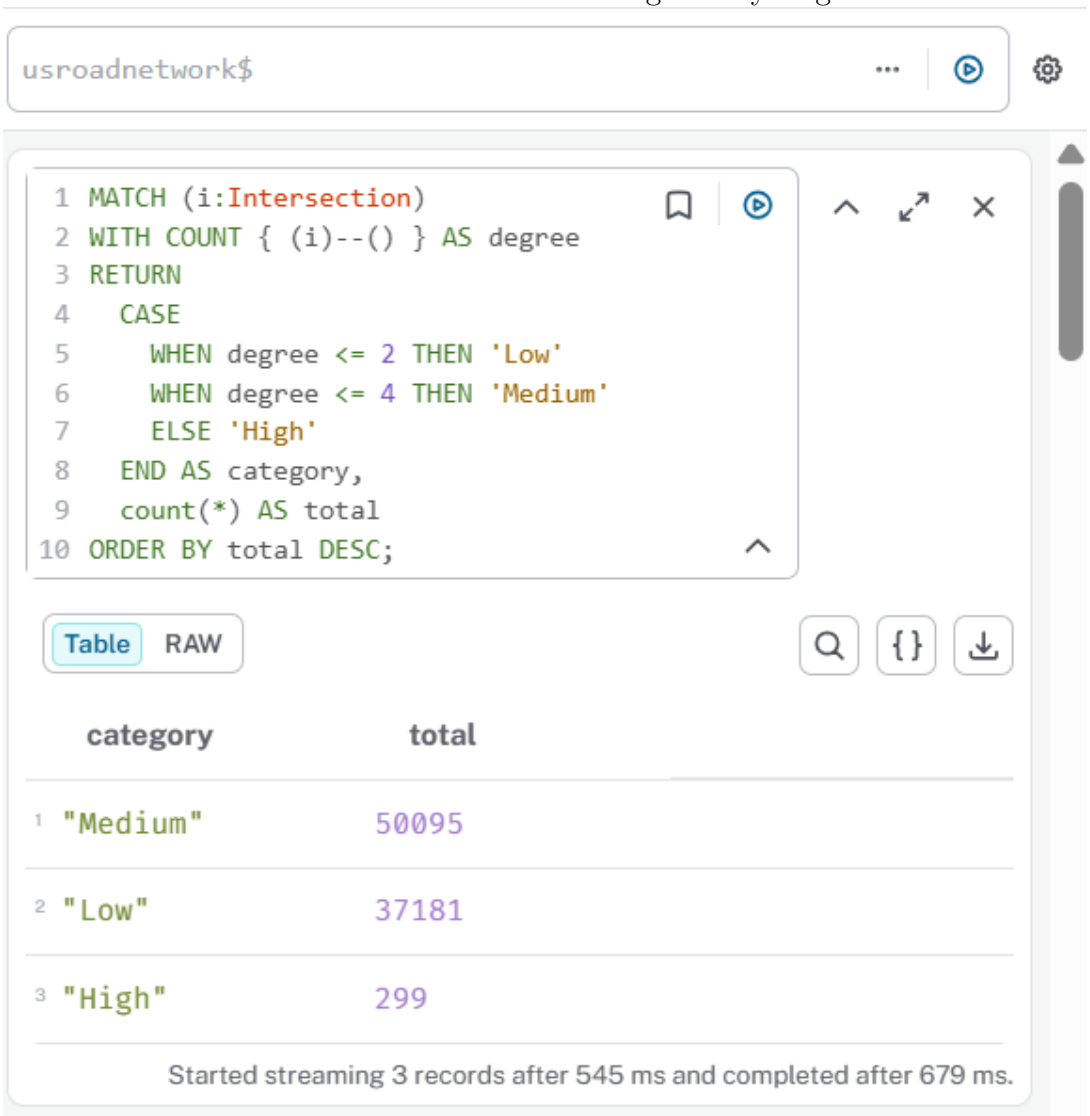


Table 3: Intersection Categories by Degree

Category	Number of Intersections
Medium	50,095
Low	37,181
High	299

Interpretation: The categorization of intersections by degree shows that the road network is dominated by medium-connectivity intersections, which account for 50,095 intersections (57.2%), indicating that most junctions connect a moderate number of roads typical of standard urban and suburban layouts. Low-connectivity intersections make up 37,181 intersections (42.5%), reflecting the presence of dead-ends, residential streets, and peripheral road segments with localized traffic patterns. In contrast, high-connectivity intersections are very rare, representing only 299 intersections (0.34%), and correspond to critical infrastructure such as major junctions, roundabouts, or highway interchanges. Overall, this distribution highlights a decentralized and structurally balanced road network, with limited reliance on highly connected hubs and a topology that supports efficient traffic distribution and network resilience.

11 Task 9: Enhanced Degree Distribution Visualization

A logarithmic-scale bar chart was generated using Plotly to emphasize the presence of highly connected intersections.

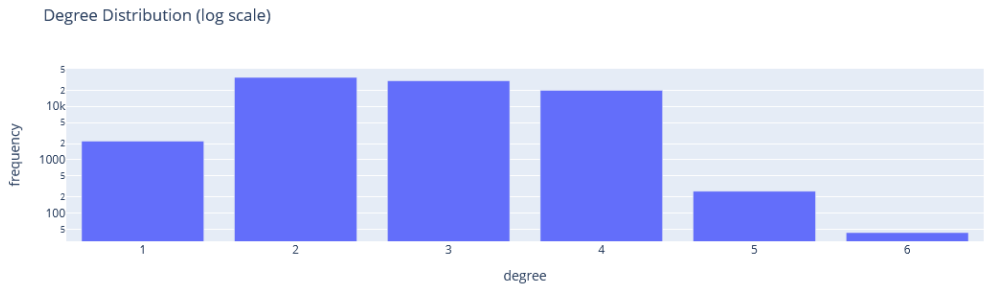
Python Code:

```
# Degree Distribution Bar Chart (log scale option)

# degree_df = query results from Task 6
fig = px.bar(degree_df, x='degree', y='frequency', title='Degree
   Distribution (log scale)')
fig.update_yaxes(type='log')
fig.show()
```

Screenshot: Enhanced Degree Distribution Visualization

9. Degree Distribution Bar Chart (log scale option)

```
# Degree Distribution Bar Chart (log scale option)

import plotly.express as px

# degree_df = query results from Task 6
fig = px.bar(degree_df, x='degree', y='frequency', title='Degree Distribution (log scale)')
fig.update_yaxes(type='log')
fig.show()
```



Interpretation: The log-scale degree distribution highlights the highly skewed nature of connectivity in the road network. Intersections with low to moderate degrees (2–4 connections) dominate the network, appearing prominently even on a logarithmic scale, which confirms that these junction types occur in very large numbers. Degree-2 intersections are the most frequent, followed closely by degree-3 and degree-4 nodes, reflecting typical road structures such as straight segments, T-junctions, and four-way intersections. In contrast, intersections with higher degrees (5 and 6) appear far less frequently and drop sharply on the log scale, emphasizing their rarity. The use of a logarithmic y-axis makes it clear that high-degree intersections are orders of magnitude less common than low-degree ones. Overall, the plot demonstrates that the road network is not hub-dominated but instead characterized by many moderately connected intersections and very few highly connected junctions, which is consistent with the physical and spatial constraints of real-world transportation networks.

12 Conclusion

Across Tasks 1 to 9, the analysis successfully demonstrated how a graph-based approach using Neo4j and Python can be used to explore, quantify, and visualize the structural properties of a large-scale road network. By establishing a connection between Neo4j and Python, key network metrics such as total intersections, total roads, degree distribution, top connected intersections, and connectivity categories were systematically extracted using Cypher queries and analyzed with Pandas. The resulting Plotly dashboard provided an intuitive and interactive way to summarize these metrics through KPI indicators, bar charts, tables, and log-scale visualizations. The findings consistently showed that the road network is dominated by low- to medium-connectivity intersections, with very few highly connected nodes, reflecting the physical and spatial constraints of real-world transportation systems. The absence of dominant hubs and the limited number of high-degree intersections indicate a decentralized, well-balanced, and resilient network structure. Overall, these tasks collectively illustrate the effectiveness of combining graph databases, Python analytics, and interactive visualization to gain meaningful insights into transportation networks and support data-driven decision-making in urban and infrastructure planning.
This analysis demonstrates the effectiveness of combining Neo4j and Python for

large-scale road network analysis. The insights obtained are valuable for transportation planning, traffic management, and infrastructure optimization.