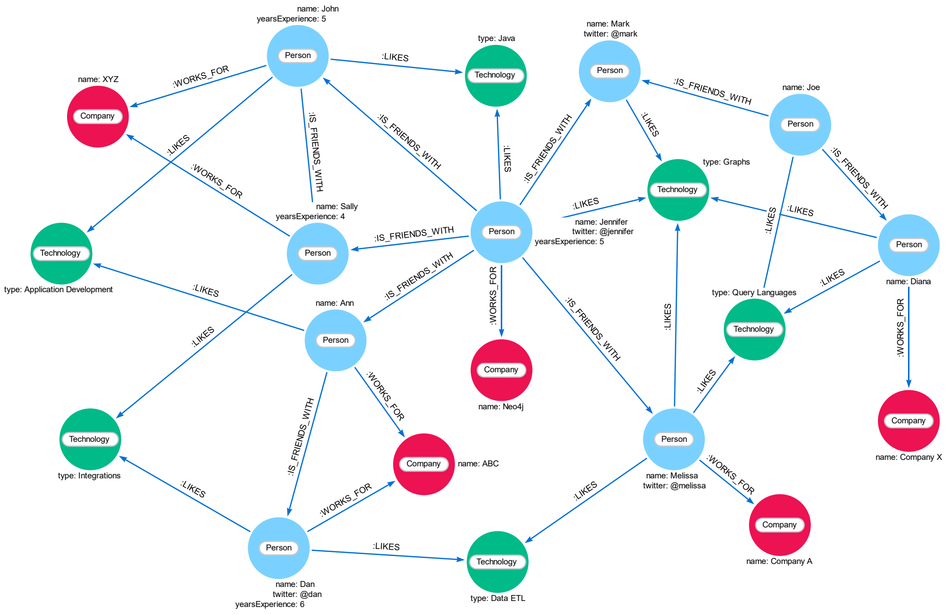
SkyRoute: Flights Graph Network



neo4j

Name: Nuwan Keshara Galappaththi

Email: nuwankeshara12@gmail.com

LinkedIn: https://www.linkedin.com/in/nuwan-keshara/

Contents

1) Project Title

- **SkyRoute: Flights Graph Network**

2) Objectives / Problem Statement

- A graph-based model of global flight routes built with the Neo4j graph database. It maps airports as nodes and routes as relationships, enabling domestic airlines network view, international airlines network view an shortest path finding with interactive spatial visualization using NeoDash.

- Dataset - https://openflights.org/data.php

❖ *The problem:*
  - The raw CSV data is unstructured and inconsistent.
  - It lacks direct analytical value for route analysis, network visualization or shortest path insights.
  - Traditional relational models make it complex to traverse relationships between flights data efficiently.

❖ *Goal*:

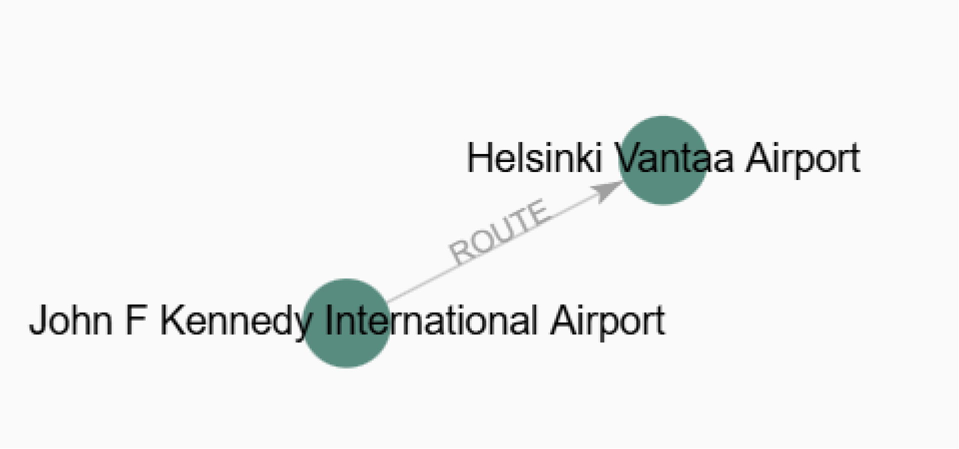
- Design and implement a graph data model and analytics system that,
  - Cleans and imports the OpenFlights dataset into Neo4j.
  - Models airports and routes as graph entities.
  - Enables domestic and international route exploration.
  - Visualizes real-world flight connectivity on a world map.
  - Supports shortest path queries and airline network analytics using NeoDash.

3) Tech Stack

Layer	**Tool**	**Purpose**
Data Source	OpenFlights Dataset (CSV) • Airports.dat • Airlines.dat • Routes.dat	Source of airport and route data
Data Ingestion & Cleaning	Python (Pandas)	Load and clean CSVs before graph import
Graph Database	Neo4j (Community Edition)/ Neo4j AuraDB Managed Service	Stores airports as nodes and routes as relationships
Visualization Layer	NeoDash	Interactive visualization of airports, airlines and routes
Query Language	Cypher	Graph traversal and path-finding logic
Graph Utilities	APOC Library / Neo4j Browser	Advanced graph operations and conditional query execution
Version Control	Git / GitHub	Code and Cypher script management

4) Architecture / Design

- Node and Relationship (Airports)-[Routes]->(Airports)



Graph Data Model:

(:Airport {iata, name, city, country, location}) -[:ROUTE {airline, updated}]-> (:Airport {iata, name, city, country, location})

(:Airline {iata, name, country})

Note: Airline nodes exist independently, while routes carry airline metadata as properties.

Workflow Overview:

1. Data Ingestion (Python + Neo4j Driver):
    - Load airport, airline and route CSVs.
    - Create constraints to ensure unique IATA codes.
    - Batch load data to Neo4j using parameterized Cypher queries.
2. Graph Construction:
    - Airports created as nodes.
    - Airlines created as nodes.
    - Routes created as relationships between Airport nodes, with airline info stored as a property.
3. Spatial Modeling:

  - Each airport has a location property (point type) for map-based visualization in NeoDash.

4. Visualization (NeoDash):
   - Interactive map views for global routes.
   - Table view to list shortest flight connections.
   - Graph view showing airport-to-airport connectivity.

5) Implementation Steps

1.Data extraction (Python)

- Read raw CSVs with Pandas.
- Standardize column names and handle nulls.

2.Data Validation and Quality Checks

- Validate data, transform null values, remove unwated columns
  - Checks for nulls
  - Validates non-negative numeric columns
  - Detects duplicates

3. Data Load into Neo4j

- Create Constraints
  - *CREATE CONSTRAINT IF NOT EXISTS FOR (a:Airport) REQUIRE a.iata IS UNIQUE;*
  - *CREATE CONSTRAINT IF NOT EXISTS FOR (al:Airline) REQUIRE al.iata IS UNIQUE;*
- Load into Neo4j
  - *MERGE (a:Airport {iata: row.iata}) SET a.name = row.name, a.city = row.city, a.country = row.country, a.latitude = toFloat(row.latitude), a.longitude = toFloat(row.longitude), a.location = point({longitude: toFloat(row.longitude), latitude: toFloat(row.latitude)});*
  - *MERGE (al:Airline {iata: row.iata}) SET al.name = row.name, al.country = row.country;*
  - *MATCH (src:Airport {iata: row.source_airport}) MATCH (dst:Airport {iata: row.destination_airport}) MATCH (al:Airline {iata: row.airline}) MERGE (src)-[r:ROUTE {airline: row.airline}]->(dst) SET r.updated = datetime();*

4.Visualization

- NeoDash dashboards were designed to show:

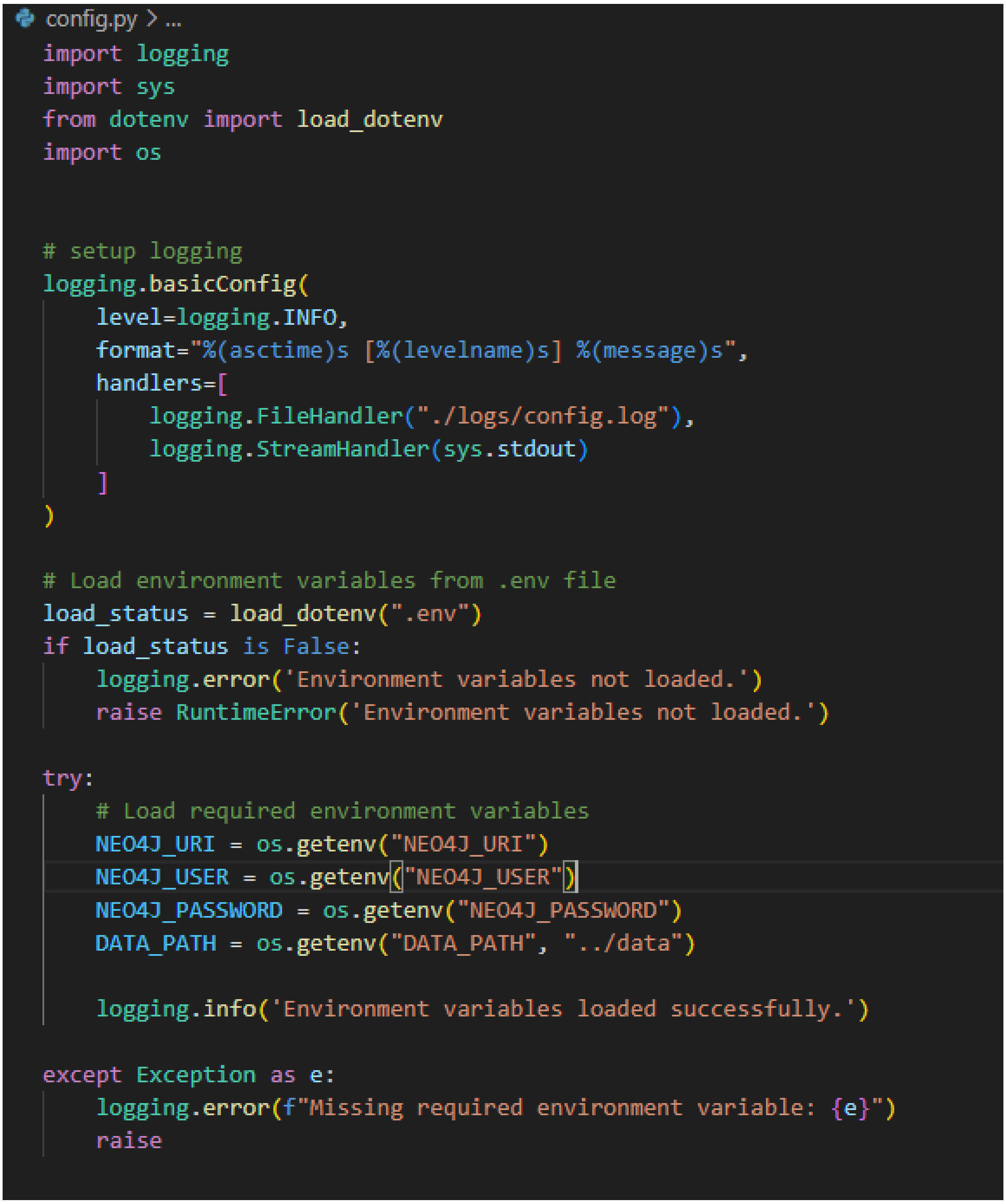
  - Map View:
  - Displays airports and connecting routes using spatial coordinates.

  - Table View:
  - Lists up to 5 shortest or multi-hop routes between countries with readable path strings.

  - Graph View:
  - Renders airports as nodes and routes as connecting relationships for visual exploration.

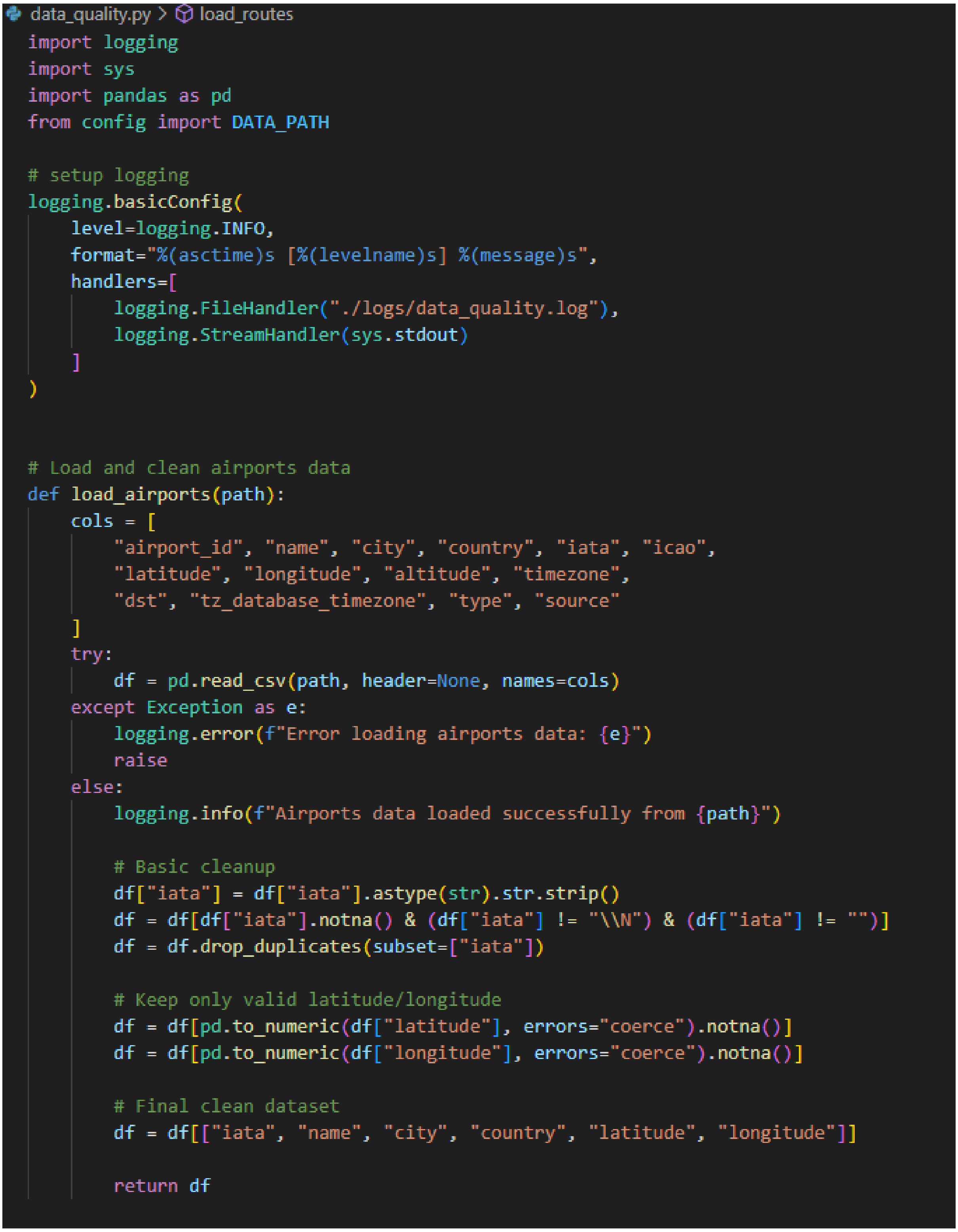
6) Scripts

- *config.py* – Load environment variable



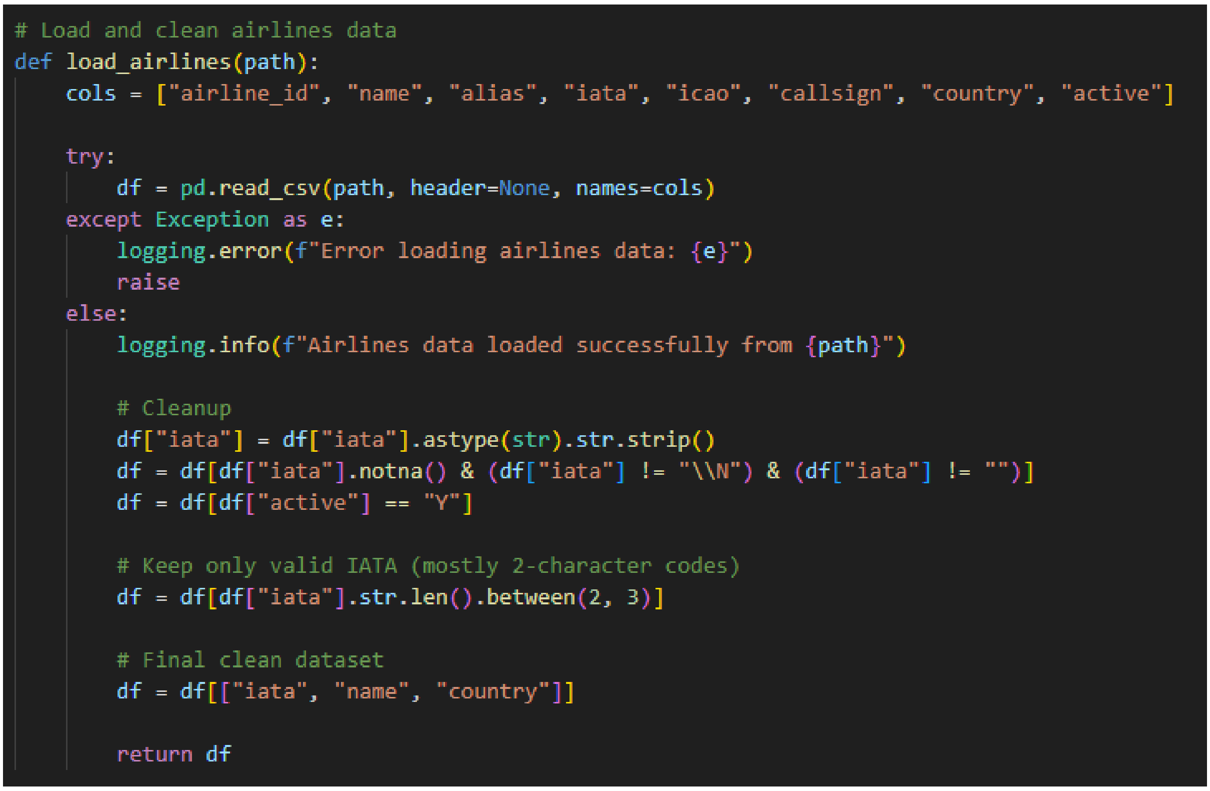
```python
import logging
import sys
from dotenv import load_dotenv
import os


# setup logging
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s [%(levelname)s] %(message)s",
    handlers=[
        logging.FileHandler("./logs/config.log"),
        logging.StreamHandler(sys.stdout)
    ]
)

# Load environment variables from .env file
load_status = load_dotenv(".env")
if load_status is False:
    logging.error('Environment variables not loaded.')
    raise RuntimeError('Environment variables not loaded.')

try:
    # Load required environment variables
    NEO4J_URI = os.getenv("NEO4J_URI")
    NEO4J_USER = os.getenv("NEO4J_USER")
    NEO4J_PASSWORD = os.getenv("NEO4J_PASSWORD")
    DATA_PATH = os.getenv("DATA_PATH", "../data")

    logging.info('Environment variables loaded successfully.')

except Exception as e:
    logging.error(f"Missing required environment variable: {e}")
    raise
```

- *data_quality.py* - Data Quality Checks

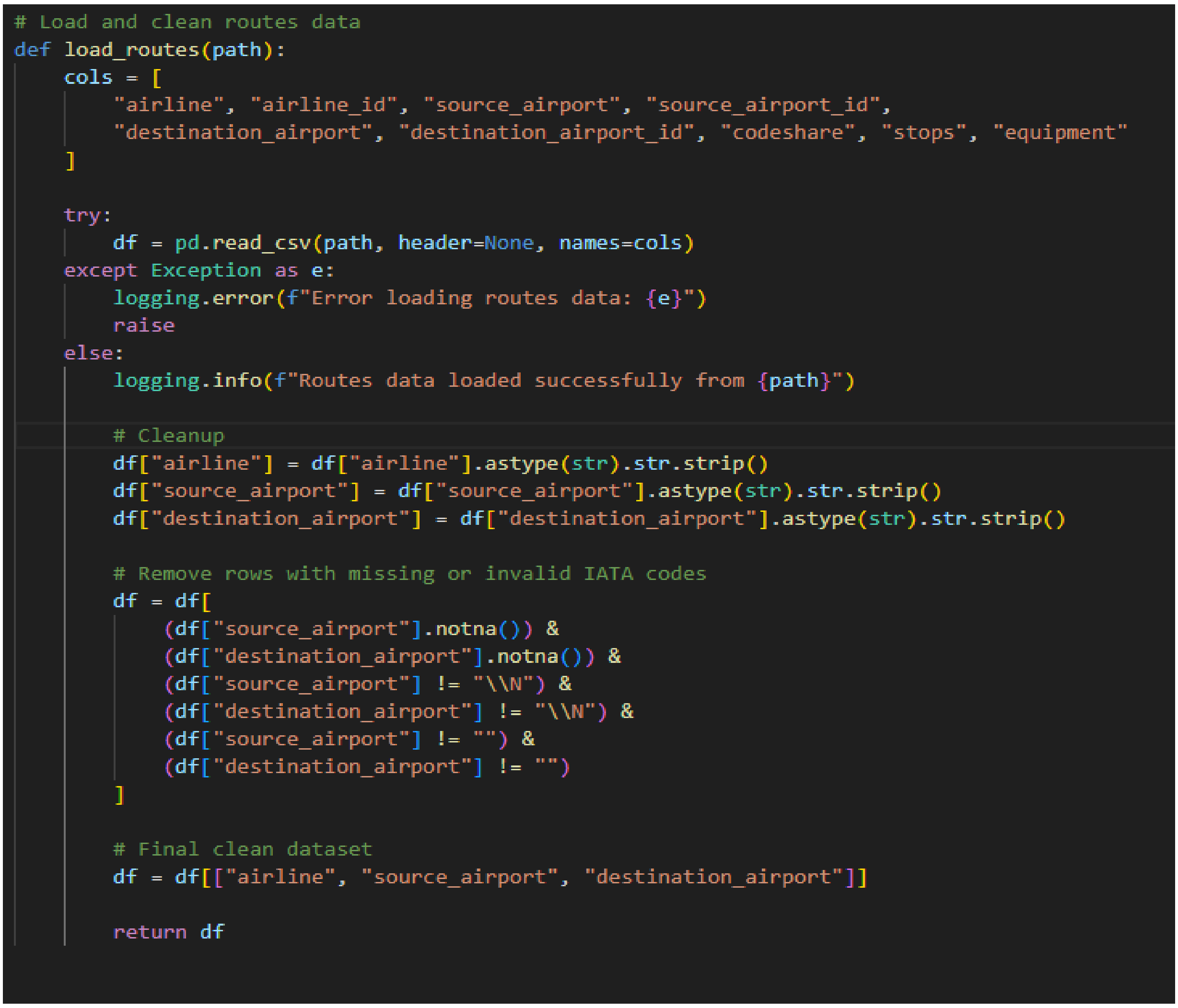


data_quality.py > load_routes

```python
import logging
import sys
import pandas as pd
from config import DATA_PATH

# setup logging
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s [%(levelname)s] %(message)s",
    handlers=[
        logging.FileHandler("./logs/data_quality.log"),
        logging.StreamHandler(sys.stdout)
    ]
)


# Load and clean airports data
def load_airports(path):
    cols = [
        "airport_id", "name", "city", "country", "iata", "icao",
        "latitude", "longitude", "altitude", "timezone",
        "dst", "tz_database_timezone", "type", "source"
    ]
    try:
        df = pd.read_csv(path, header=None, names=cols)
    except Exception as e:
        logging.error(f"Error loading airports data: {e}")
        raise
    else:
        logging.info(f"Airports data loaded successfully from {path}")

        # Basic cleanup
        df["iata"] = df["iata"].astype(str).str.strip()
        df = df[df["iata"].notna() & (df["iata"] != "\\N") & (df["iata"] != "")]
        df = df.drop_duplicates(subset=["iata"])

        # Keep only valid latitude/longitude
        df = df[pd.to_numeric(df["latitude"], errors="coerce").notna()]
        df = df[pd.to_numeric(df["longitude"], errors="coerce").notna()]

        # Final clean dataset
        df = df[["iata", "name", "city", "country", "latitude", "longitude"]]

        return df
```

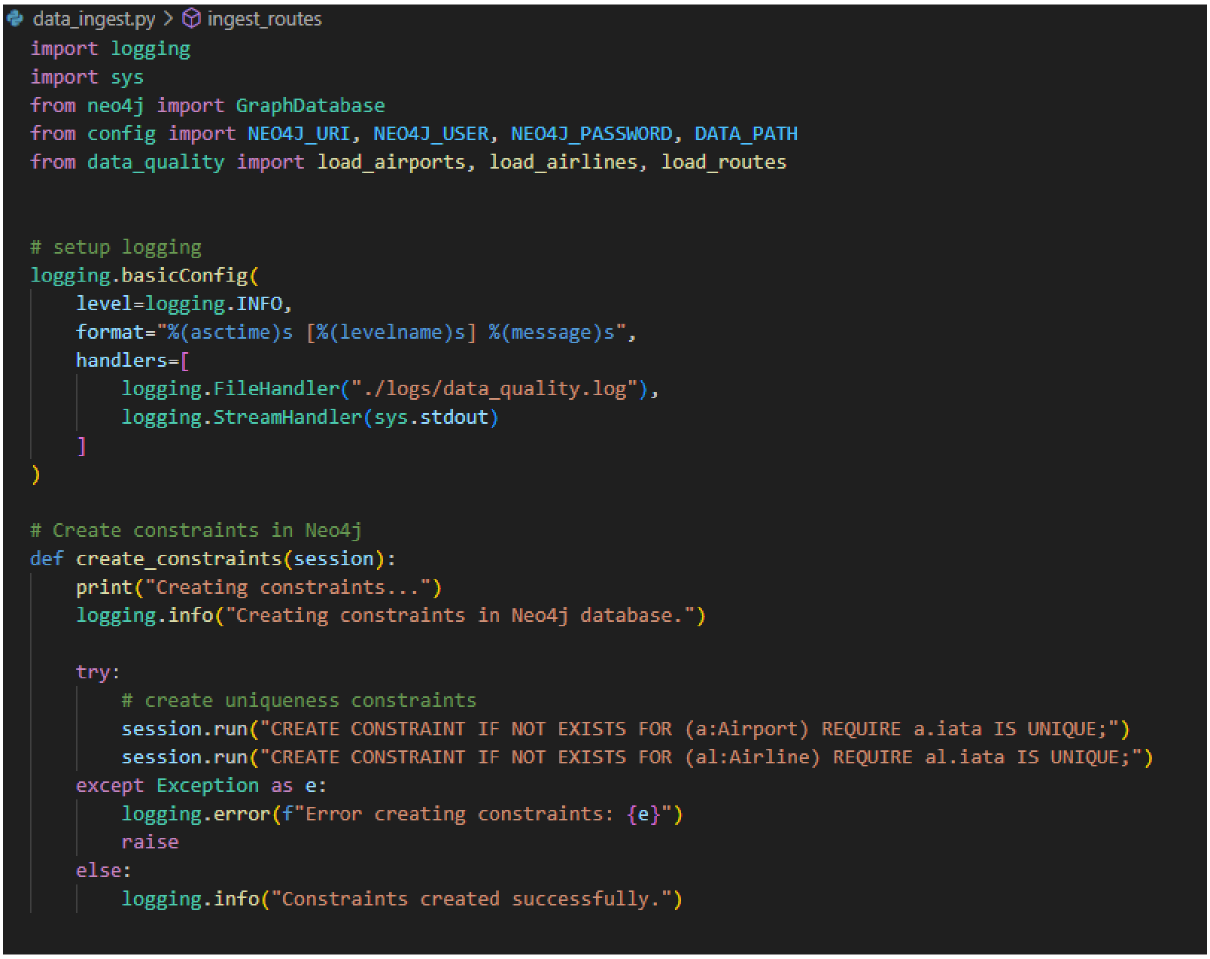


```python
# Load and clean airlines data
def load_airlines(path):
    cols = ["airline_id", "name", "alias", "iata", "icao", "callsign", "country", "active"]

    try:
        df = pd.read_csv(path, header=None, names=cols)
    except Exception as e:
        logging.error(f"Error loading airlines data: {e}")
        raise
    else:
        logging.info(f"Airlines data loaded successfully from {path}")

        # Cleanup
        df["iata"] = df["iata"].astype(str).str.strip()
        df = df[df["iata"].notna() & (df["iata"] != "\\N") & (df["iata"] != "")]
        df = df[df["active"] == "Y"]

        # Keep only valid IATA (mostly 2-character codes)
        df = df[df["iata"].str.len().between(2, 3)]

        # Final clean dataset
        df = df[["iata", "name", "country"]]

        return df
```

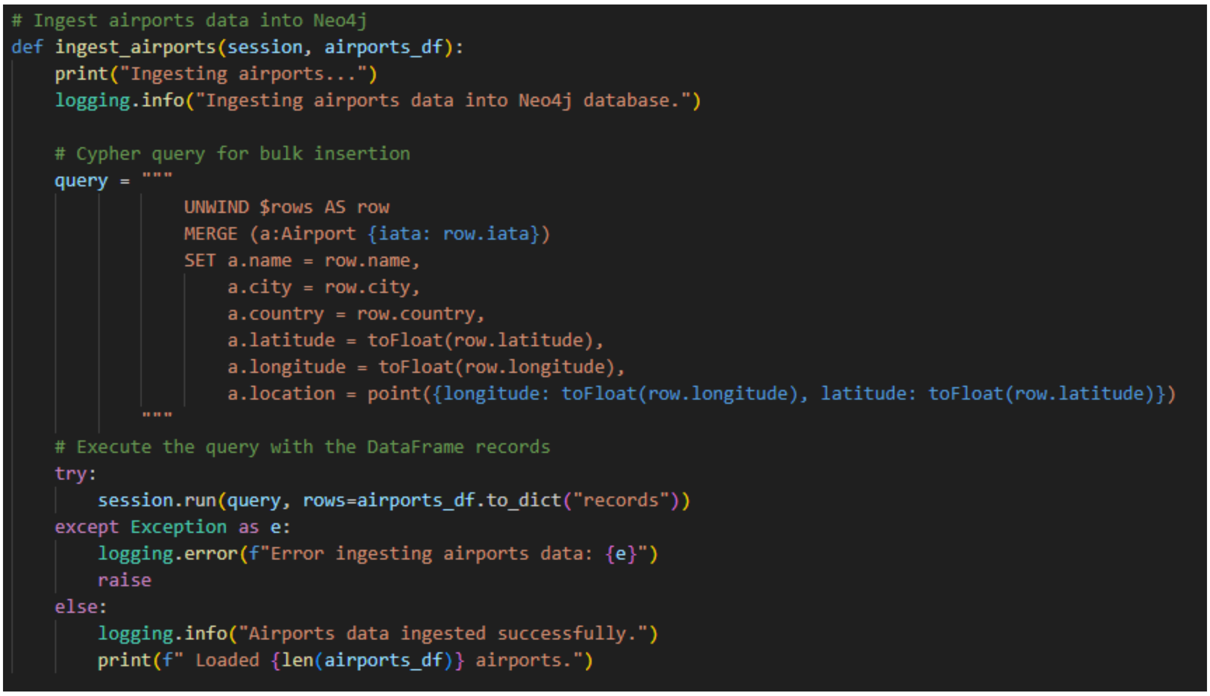


```python
# Load and clean routes data
def load_routes(path):
    cols = [
        "airline", "airline_id", "source_airport", "source_airport_id",
        "destination_airport", "destination_airport_id", "codeshare", "stops", "equipment"
    ]

    try:
        df = pd.read_csv(path, header=None, names=cols)
    except Exception as e:
        logging.error(f"Error loading routes data: {e}")
        raise
    else:
        logging.info(f"Routes data loaded successfully from {path}")

        # Cleanup
        df["airline"] = df["airline"].astype(str).str.strip()
        df["source_airport"] = df["source_airport"].astype(str).str.strip()
        df["destination_airport"] = df["destination_airport"].astype(str).str.strip()

        # Remove rows with missing or invalid IATA codes
        df = df[
            (df["source_airport"].notna()) &
            (df["destination_airport"].notna()) &
            (df["source_airport"] != "\\N") &
            (df["destination_airport"] != "\\N") &
            (df["source_airport"] != "") &
            (df["destination_airport"] != "")
        ]

        # Final clean dataset
        df = df[["airline", "source_airport", "destination_airport"]]

        return df
```

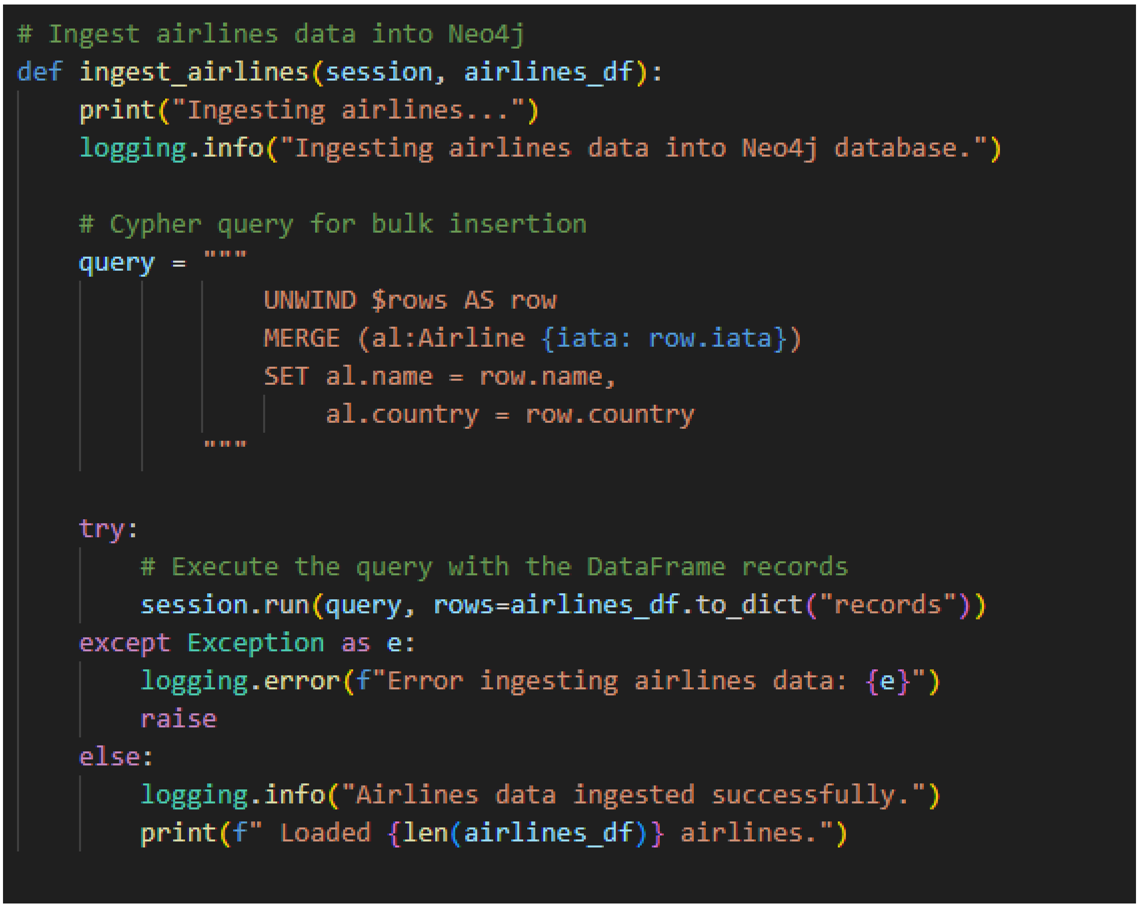
- *data_ingest.py* – ingest data into neo4j database



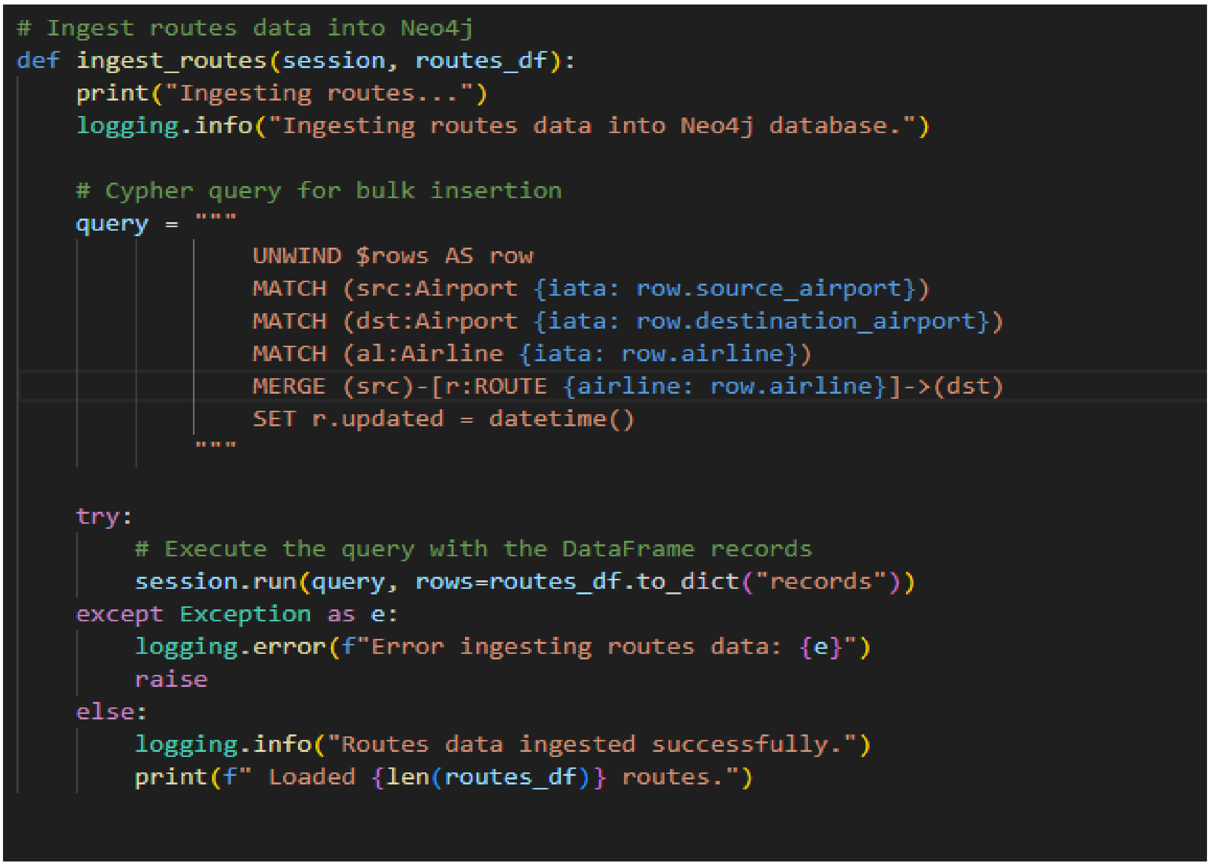
```python
import logging
import sys
from neo4j import GraphDatabase
from config import NEO4J_URI, NEO4J_USER, NEO4J_PASSWORD, DATA_PATH
from data_quality import load_airports, load_airlines, load_routes


# setup logging
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s [%(levelname)s] %(message)s",
    handlers=[
        logging.FileHandler("./logs/data_quality.log"),
        logging.StreamHandler(sys.stdout)
    ]
)

# Create constraints in Neo4j
def create_constraints(session):
    print("Creating constraints...")
    logging.info("Creating constraints in Neo4j database.")

    try:
        # create uniqueness constraints
        session.run("CREATE CONSTRAINT IF NOT EXISTS FOR (a:Airport) REQUIRE a.iata IS UNIQUE;")
        session.run("CREATE CONSTRAINT IF NOT EXISTS FOR (al:Airline) REQUIRE al.iata IS UNIQUE;")
    except Exception as e:
        logging.error(f"Error creating constraints: {e}")
        raise
    else:
        logging.info("Constraints created successfully.")
```



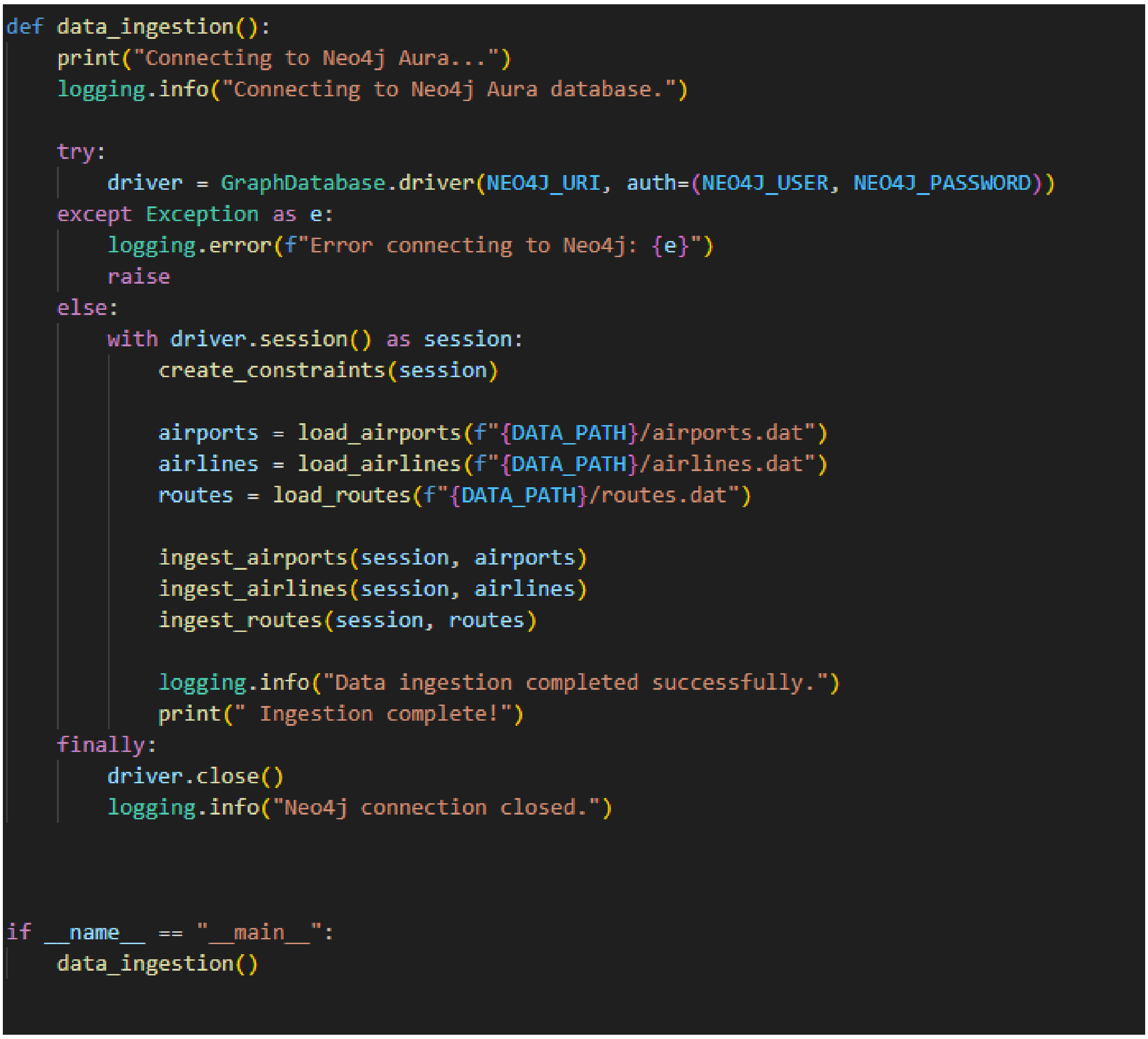
```python
# Ingest airports data into Neo4j
def ingest_airports(session, airports_df):
    print("Ingesting airports...")
    logging.info("Ingesting airports data into Neo4j database.")

    # Cypher query for bulk insertion
    query = """
            UNWIND $rows AS row
            MERGE (a:Airport {iata: row.iata})
            SET a.name = row.name,
                a.city = row.city,
                a.country = row.country,
                a.latitude = toFloat(row.latitude),
                a.longitude = toFloat(row.longitude),
                a.location = point({longitude: toFloat(row.longitude), latitude: toFloat(row.latitude)})
        """
    # Execute the query with the DataFrame records
    try:
        session.run(query, rows=airports_df.to_dict("records"))
    except Exception as e:
        logging.error(f"Error ingesting airports data: {e}")
        raise
    else:
        logging.info("Airports data ingested successfully.")
        print(f" Loaded {len(airports_df)} airports.")
```



```python
# Ingest airlines data into Neo4j
def ingest_airlines(session, airlines_df):
    print("Ingesting airlines...")
    logging.info("Ingesting airlines data into Neo4j database.")

    # Cypher query for bulk insertion
    query = """
            UNWIND $rows AS row
            MERGE (al:Airline {iata: row.iata})
            SET al.name = row.name,
                al.country = row.country
            """

    try:
        # Execute the query with the DataFrame records
        session.run(query, rows=airlines_df.to_dict("records"))
    except Exception as e:
        logging.error(f"Error ingesting airlines data: {e}")
        raise
    else:
        logging.info("Airlines data ingested successfully.")
        print(f" Loaded {len(airlines_df)} airlines.")
```



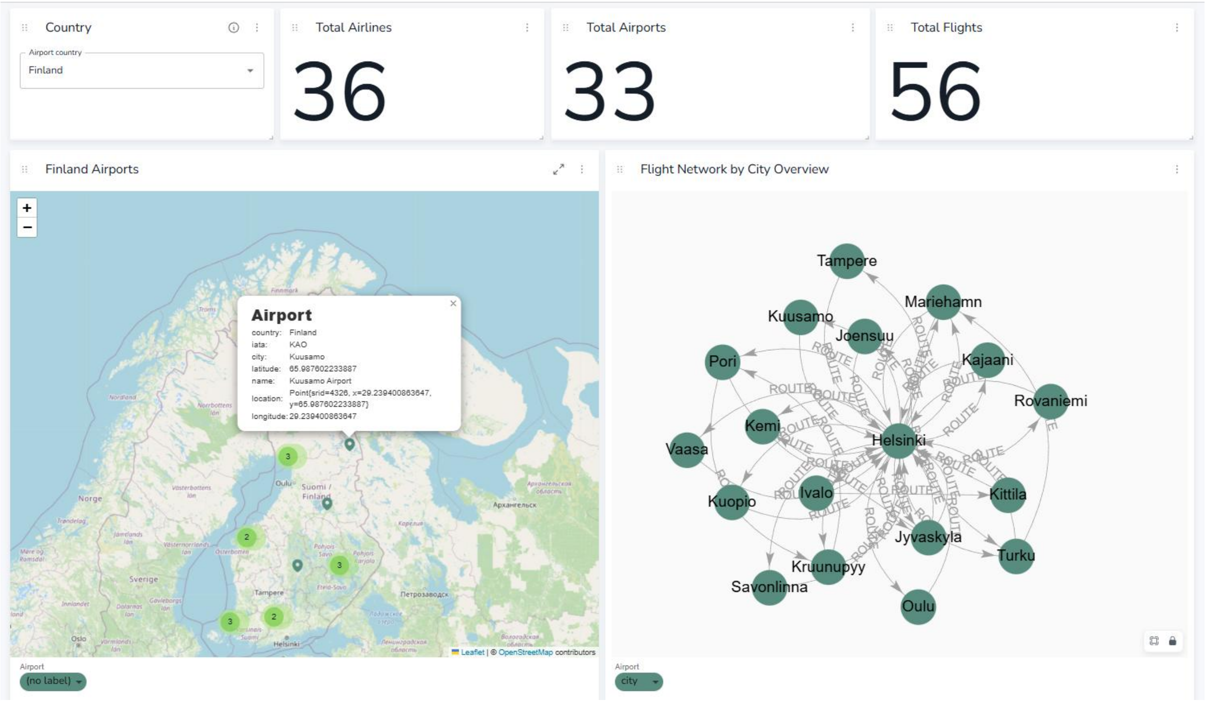
```python
# Ingest routes data into Neo4j
def ingest_routes(session, routes_df):
    print("Ingesting routes...")
    logging.info("Ingesting routes data into Neo4j database.")

    # Cypher query for bulk insertion
    query = """
            UNWIND $rows AS row
            MATCH (src:Airport {iata: row.source_airport})
            MATCH (dst:Airport {iata: row.destination_airport})
            MATCH (al:Airline {iata: row.airline})
            MERGE (src)-[r:ROUTE {airline: row.airline}]->(dst)
            SET r.updated = datetime()
            """

    try:
        # Execute the query with the DataFrame records
        session.run(query, rows=routes_df.to_dict("records"))
    except Exception as e:
        logging.error(f"Error ingesting routes data: {e}")
        raise
    else:
        logging.info("Routes data ingested successfully.")
        print(f" Loaded {len(routes_df)} routes.")
```



```python
def data_ingestion():
    print("Connecting to Neo4j Aura...")
    logging.info("Connecting to Neo4j Aura database.")

    try:
        driver = GraphDatabase.driver(NEO4J_URI, auth=(NEO4J_USER, NEO4J_PASSWORD))
    except Exception as e:
        logging.error(f"Error connecting to Neo4j: {e}")
        raise
    else:
        with driver.session() as session:
            create_constraints(session)

            airports = load_airports(f"{DATA_PATH}/airports.dat")
            airlines = load_airlines(f"{DATA_PATH}/airlines.dat")
            routes = load_routes(f"{DATA_PATH}/routes.dat")

            ingest_airports(session, airports)
            ingest_airlines(session, airlines)
            ingest_routes(session, routes)

            logging.info("Data ingestion completed successfully.")
            print(" Ingestion complete!")
    finally:
        driver.close()
        logging.info("Neo4j connection closed.")



if __name__ == "__main__":
    data_ingestion()
```

7) Data Visualizations

1. Domestic Flights Overview



Country

Airport country: Finland

Total Airlines: 36

Total Airports: 33

Total Flights: 56

Finland Airports

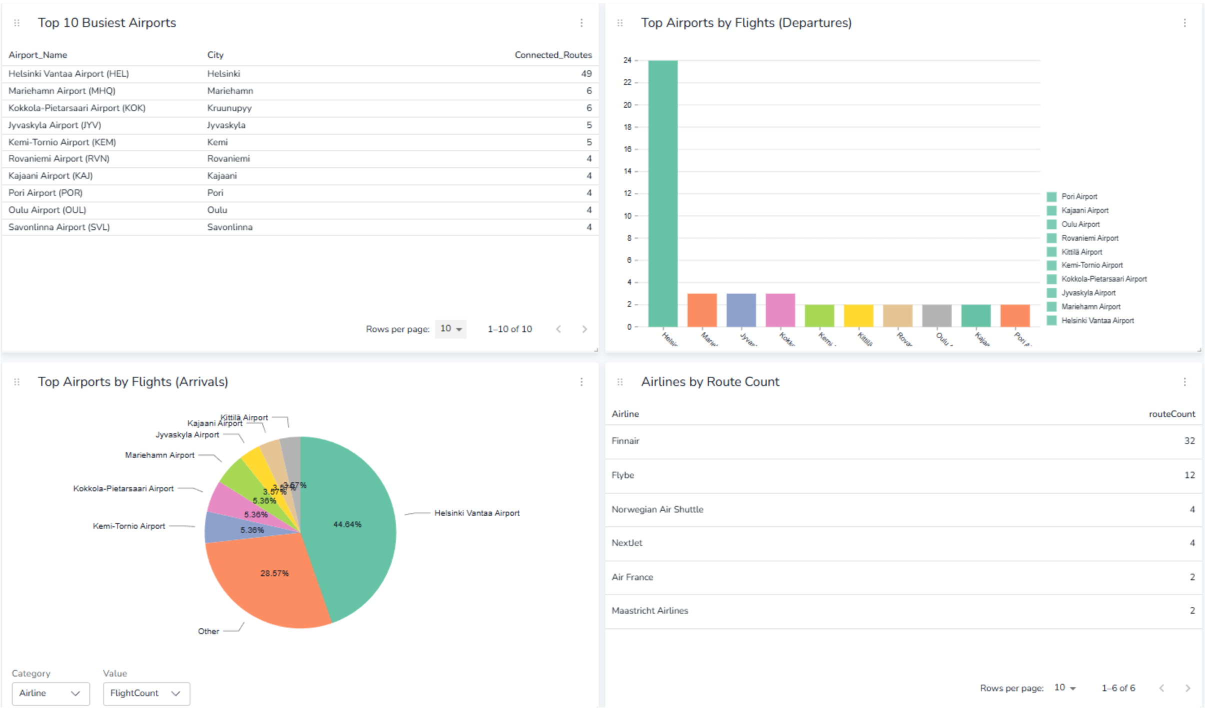
Airport

country: Finland
iata: KAO
city: Kuusamo
latitude: 65.987602233887
name: Kuusamo Airport
location: Point{srid=4326, x=29.239400863647, y=65.987602233887}
longitude: 29.239400863647

Airport: (no label)

Flight Network by City Overview

Airport: city



Top 10 Busiest Airports

Airport_Name	City	Connected_Routes
Helsinki Vantaa Airport (HEL)	Helsinki	49
Mariehamn Airport (MHQ)	Mariehamn	6
Kokkola-Pietarsaari Airport (KOK)	Kruunupyy	6
Jyvaskyla Airport (JYV)	Jyvaskyla	5
Kemi-Tornio Airport (KEM)	Kemi	5
Rovaniemi Airport (RVN)	Rovaniemi	4
Kajaani Airport (KAJ)	Kajaani	4
Pori Airport (POR)	Pori	4
Oulu Airport (OUL)	Oulu	4
Savonlinna Airport (SVL)	Savonlinna	4

Rows per page: 10 1–10 of 10

Top Airports by Flights (Departures)

Top Airports by Flights (Arrivals)

Category: Airline

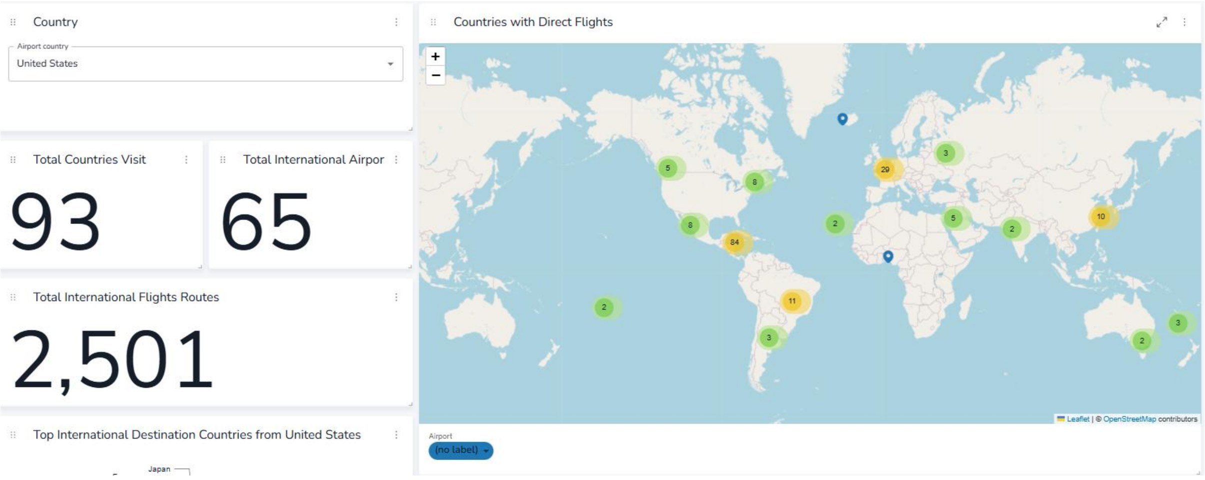
Value: FlightCount

Airlines by Route Count

Airline	routeCount
Finnair	32
Flybe	12
Norwegian Air Shuttle	4
NextJet	4
Air France	2
Maastricht Airlines	2

Rows per page: 10 1–6 of 6

2. International Flights Overview



Country

Airport country: United States

Total Countries Visit

93

Total International Airpor

65

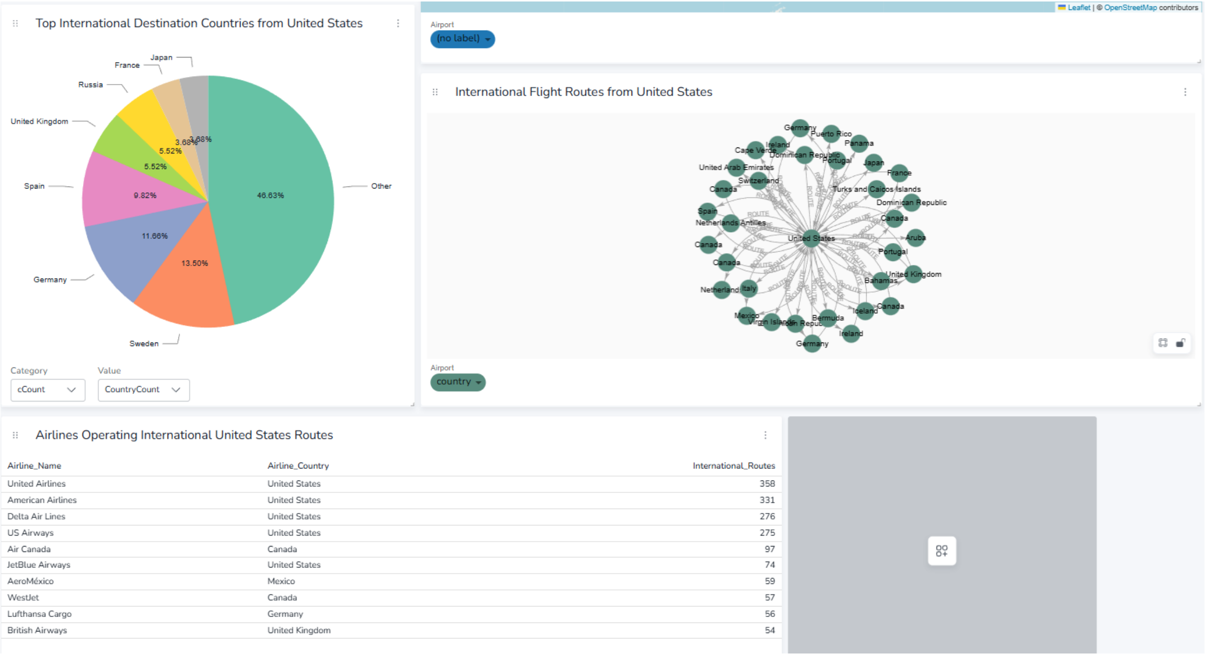
Total International Flights Routes

2,501

Top International Destination Countries from United States

Countries with Direct Flights

Airport: (no label)



Top International Destination Countries from United States

Category: cCount

Value: CountryCount

Airport: (no label)

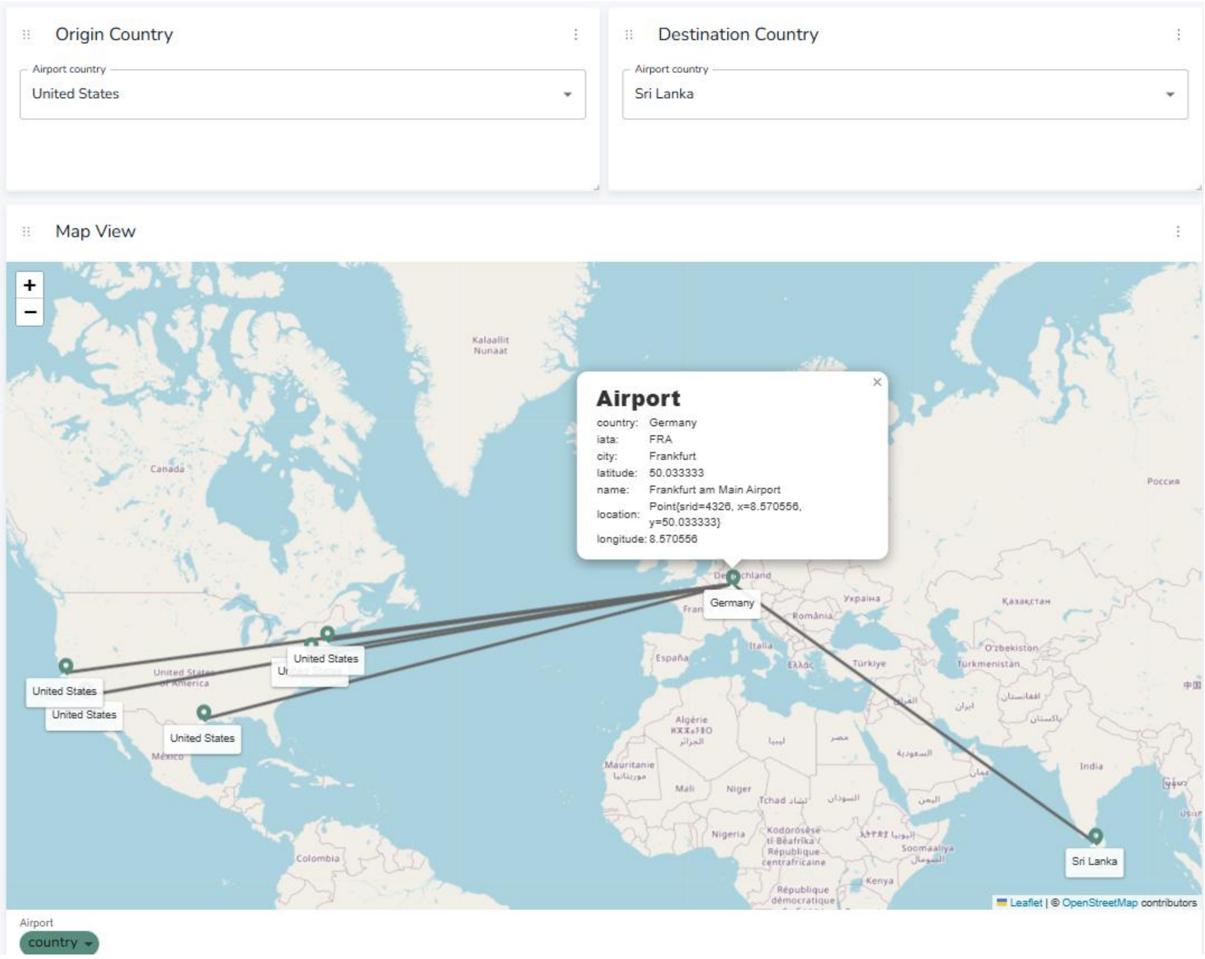
International Flight Routes from United States

Airport: country

Airlines Operating International United States Routes

Airline_Name	Airline_Country	International_Routes
United Airlines	United States	358
American Airlines	United States	331
Delta Air Lines	United States	276
US Airways	United States	275
Air Canada	Canada	97
JetBlue Airways	United States	74
AeroMéxico	Mexico	59
WestJet	Canada	57
Lufthansa Cargo	Germany	56
British Airways	United Kingdom	54

3. Find Shortest Flight Path



Origin Country

Airport country: United States

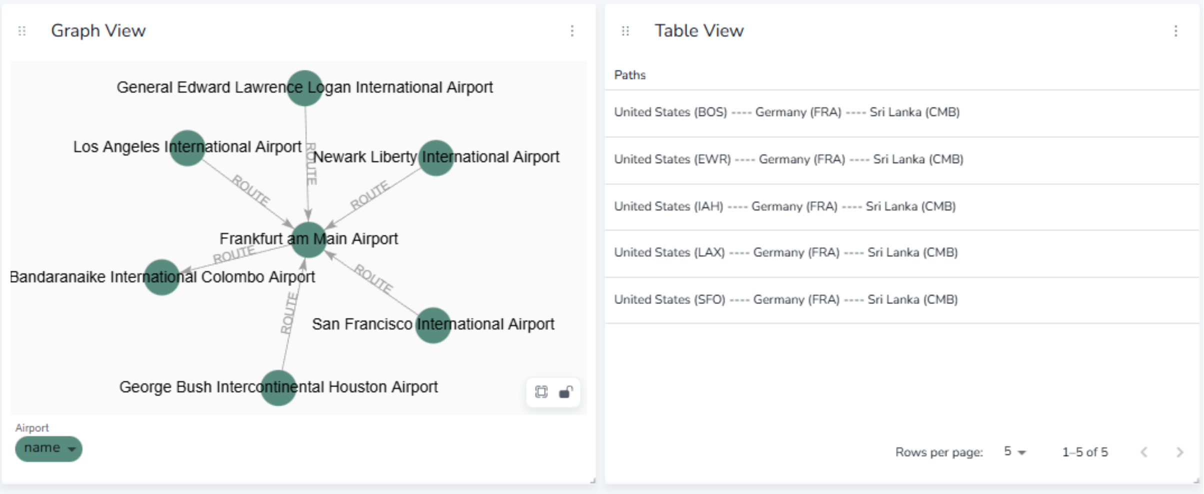
Destination Country

Airport country: Sri Lanka

Map View

Airport

country:	Germany
iata:	FRA
city:	Frankfurt
latitude:	50.033333
name:	Frankfurt am Main Airport
location:	Point{srid=4326, x=8.570556, y=50.033333}
longitude:	8.570556



Graph View

General Edward Lawrence Logan International Airport
Los Angeles International Airport
Newark Liberty International Airport
Frankfurt am Main Airport
Bandaranaike International Colombo Airport
San Francisco International Airport
George Bush Intercontinental Houston Airport
ROUTE
ROUTE
ROUTE
ROUTE
ROUTE
ROUTE
Airport
name

Table View

Paths
United States (BOS) ---- Germany (FRA) ---- Sri Lanka (CMB)
United States (EWR) ---- Germany (FRA) ---- Sri Lanka (CMB)
United States (IAH) ---- Germany (FRA) ---- Sri Lanka (CMB)
United States (LAX) ---- Germany (FRA) ---- Sri Lanka (CMB)
United States (SFO) ---- Germany (FRA) ---- Sri Lanka (CMB)

Rows per page: 5 1–5 of 5

8) Summary

- SkyRoute demonstrates a production-grade graph-based solution for modeling global flight networks.

The project showcases:

- ○ Neo4j for efficient graph modeling of real-world data.
- ○ NeoDash for spatial and relational insights.
- ○ Optimized Cypher queries with APOC for dynamic route exploration.
- ○ Clean, scalable data ingestion with schema constraints and point-based mapping.

While airlines are modeled as separate nodes, they currently act as metadata through airline properties on relationships, future work can extend this by connecting Airline nodes directly to ROUTE relationships for more semantic querying.

- Outcome:

- A fully interactive and query-optimized global flight route network built with Neo4j, ideal for exploring airline connectivity, route density and country level flight accessibility.

❖ Full Code for the project can be found on following link- https://github.com/NuwanKeshara/SkyRoute