Experiment report: Knowledge Graph

Done by student: **Bylkova Kristina (伯汀娜), 1820249049**
Course: **Big Data Analysis Technology**
Date: **October 2024**

MindMap

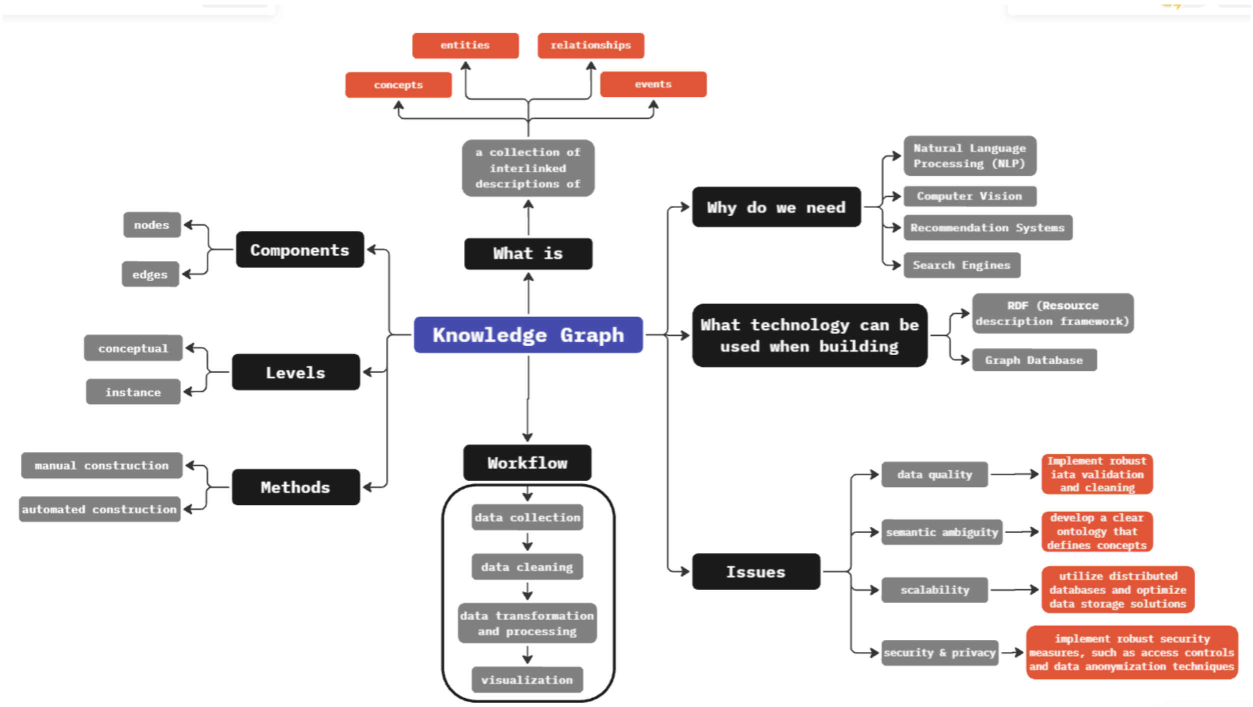

entities
relationships
concepts
events
a collection of interlinked descriptions of
What is
Knowledge Graph
Components
nodes
edges
Levels
conceptual
instance
Methods
manual construction
automated construction
Workflow
data collection
data cleaning
data transformation and processing
visualization
Why do we need
Natural Language Processing (NLP)
Computer Vision
Recommendation Systems
Search Engines
What technology can be used when building
RDF (Resource description framework)
Graph Database
Issues
data quality
Implement robust iata validation and cleaning
semantic ambiguity
develop a clear ontology that defines concepts
scalability
utilize distributed databases and optimize data storage solutions
security & privacy
implement robust security measures, such as access controls and data anonymization techniques

Figure 1: MindMap for KnowledgeGraph

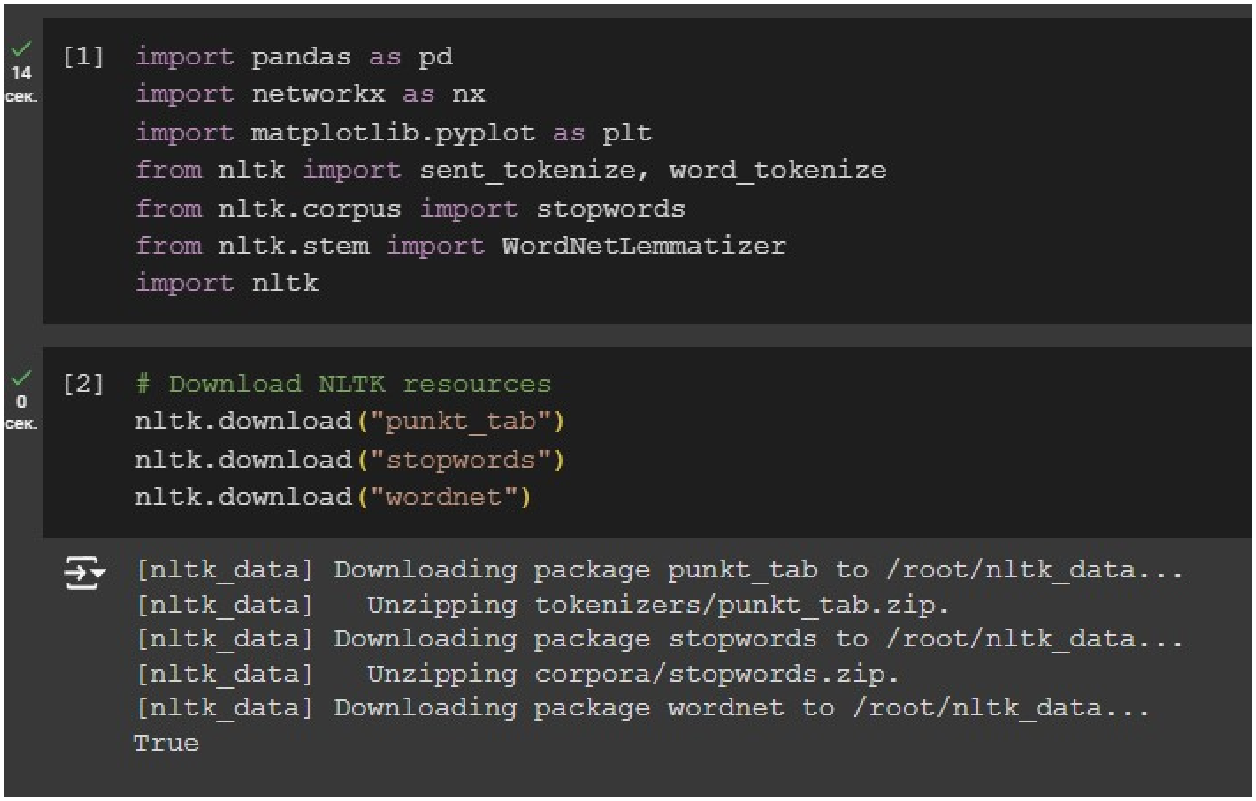
Experiment 1: Building a Knowledge Graph in NLP

Task: Complete the experiment by adding at least 1 new edge and node to the graph beyond the example provided.

Code:

Step 1: Import Libraries

Step 2: Download NLTK Resources



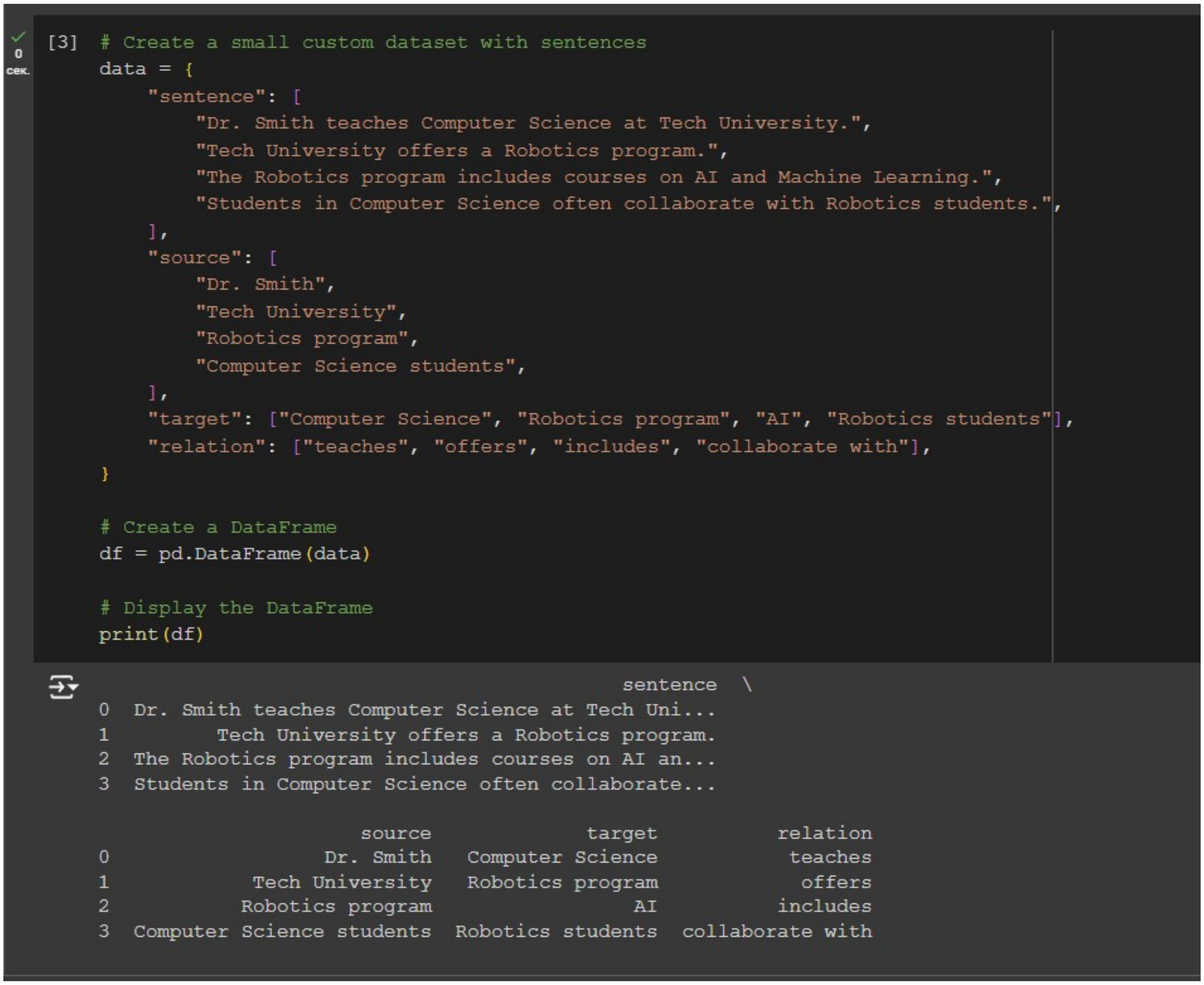
```
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
from nltk import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import nltk
```

```
# Download NLTK resources
nltk.download("punkt_tab")
nltk.download("stopwords")
nltk.download("wordnet")
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
True
```

Figure 2: Step 1-2

Step 3: Loading the Dataset

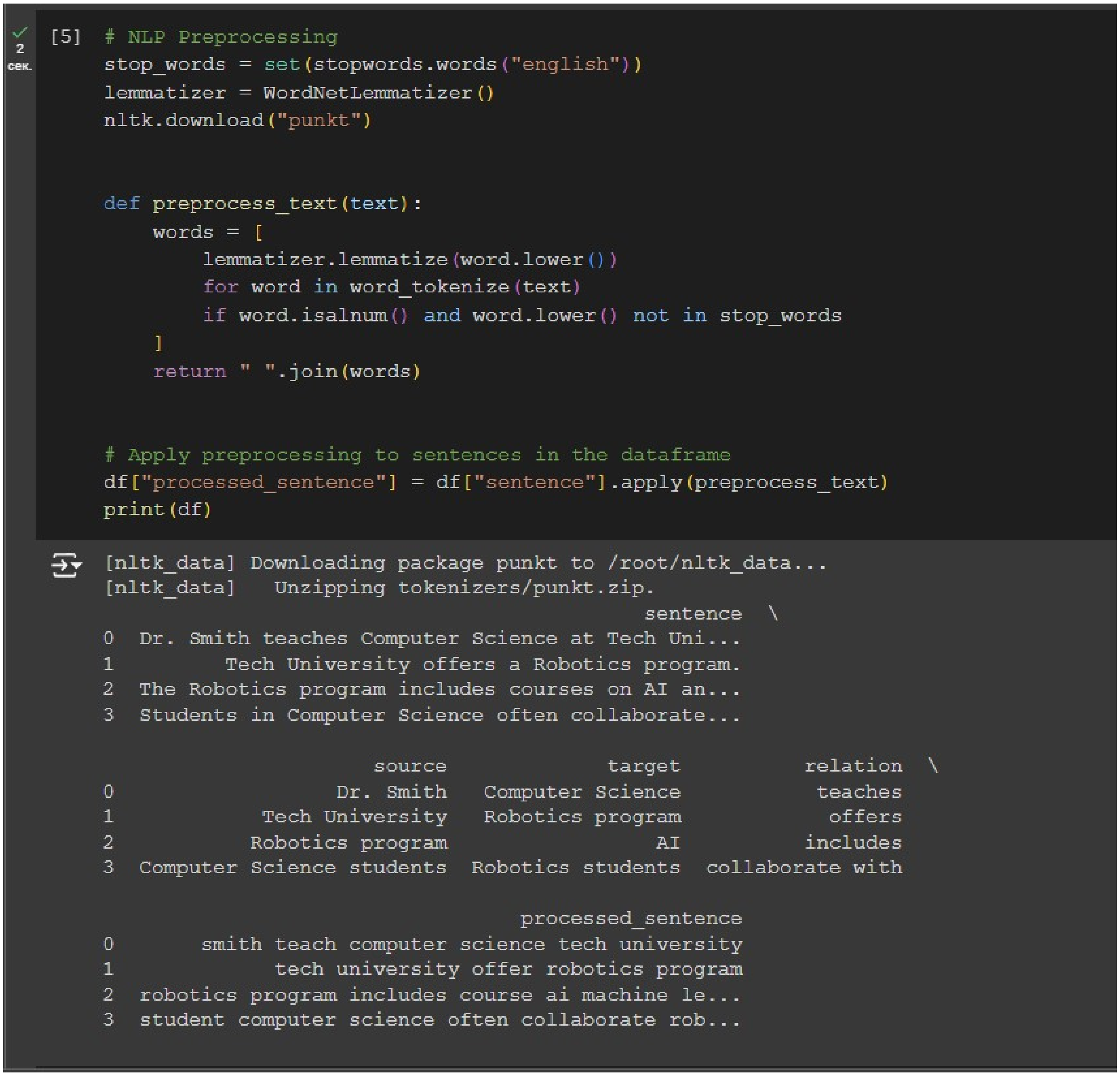


```
# Create a small custom dataset with sentences
data = {
    "sentence": [
        "Dr. Smith teaches Computer Science at Tech University.",
        "Tech University offers a Robotics program.",
        "The Robotics program includes courses on AI and Machine Learning.",
        "Students in Computer Science often collaborate with Robotics students.",
    ],
    "source": [
        "Dr. Smith",
        "Tech University",
        "Robotics program",
        "Computer Science students",
    ],
    "target": ["Computer Science", "Robotics program", "AI", "Robotics students"],
    "relation": ["teaches", "offers", "includes", "collaborate with"],
}

# Create a DataFrame
df = pd.DataFrame(data)

# Display the DataFrame
print(df)
```

```
                                            sentence  \
0  Dr. Smith teaches Computer Science at Tech Uni...
1         Tech University offers a Robotics program.
2  The Robotics program includes courses on AI an...
3  Students in Computer Science often collaborate...

                      source             target          relation
0                  Dr. Smith   Computer Science           teaches
1            Tech University   Robotics program            offers
2           Robotics program                 AI          includes
3  Computer Science students  Robotics students  collaborate with
```

Figure 3: Step 3

Step 4: Pre-processing Data



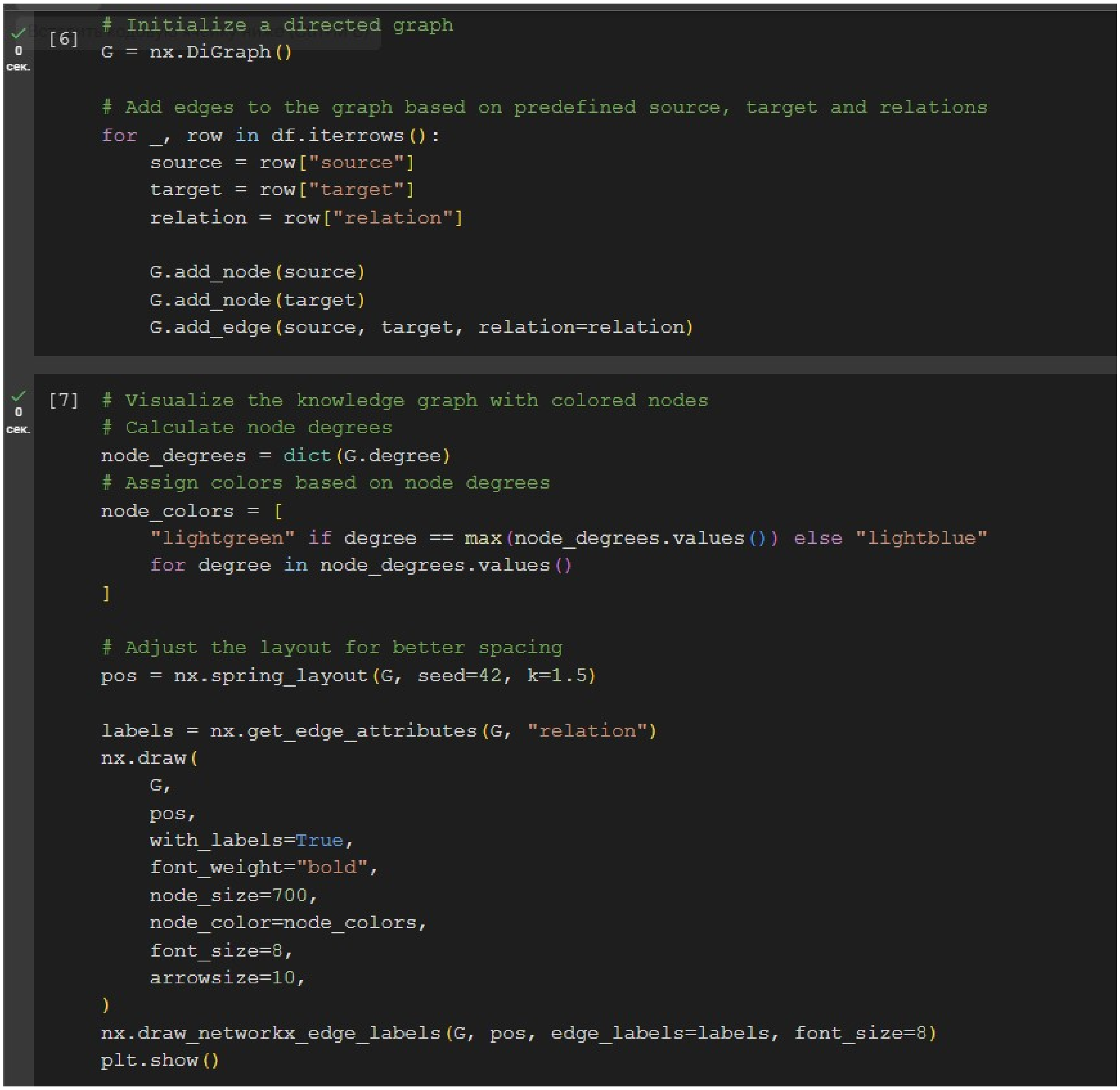
```
# NLP Preprocessing
stop_words = set(stopwords.words("english"))
lemmatizer = WordNetLemmatizer()
nltk.download("punkt")


def preprocess_text(text):
    words = [
        lemmatizer.lemmatize(word.lower())
        for word in word_tokenize(text)
        if word.isalnum() and word.lower() not in stop_words
    ]
    return " ".join(words)


# Apply preprocessing to sentences in the dataframe
df["processed_sentence"] = df["sentence"].apply(preprocess_text)
print(df)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
                                            sentence  \
0  Dr. Smith teaches Computer Science at Tech Uni...
1         Tech University offers a Robotics program.
2  The Robotics program includes courses on AI an...
3  Students in Computer Science often collaborate...

                      source             target          relation  \
0                  Dr. Smith   Computer Science           teaches
1            Tech University   Robotics program            offers
2           Robotics program                 AI          includes
3  Computer Science students  Robotics students  collaborate with

                                  processed_sentence
0       smith teach computer science tech university
1             tech university offer robotics program
2  robotics program includes course ai machine le...
3  student computer science often collaborate rob...
```

Figure 4: Step 4

Step 5: Adding Edges to the Knowledge Graph

Step 6: Visualizing the Knowledge Graph



```
# Initialize a directed graph
G = nx.DiGraph()

# Add edges to the graph based on predefined source, target and relations
for _, row in df.iterrows():
    source = row["source"]
    target = row["target"]
    relation = row["relation"]

    G.add_node(source)
    G.add_node(target)
    G.add_edge(source, target, relation=relation)
```

```
# Visualize the knowledge graph with colored nodes
# Calculate node degrees
node_degrees = dict(G.degree)
# Assign colors based on node degrees
node_colors = [
    "lightgreen" if degree == max(node_degrees.values()) else "lightblue"
    for degree in node_degrees.values()
]

# Adjust the layout for better spacing
pos = nx.spring_layout(G, seed=42, k=1.5)

labels = nx.get_edge_attributes(G, "relation")
nx.draw(
    G,
    pos,
    with_labels=True,
    font_weight="bold",
    node_size=700,
    node_color=node_colors,
    font_size=8,
    arrowsize=10,
)
nx.draw_networkx_edge_labels(G, pos, edge_labels=labels, font_size=8)
plt.show()
```

Figure 5: Step 5-6

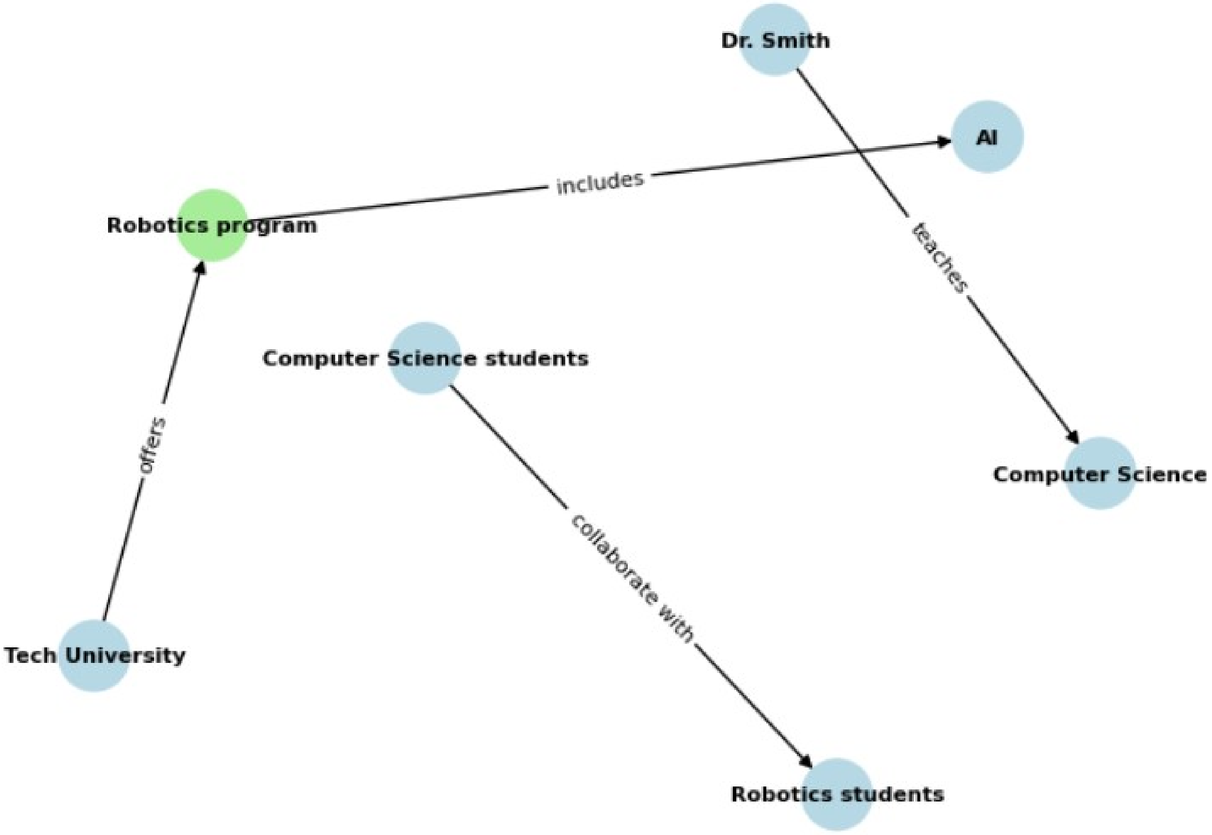
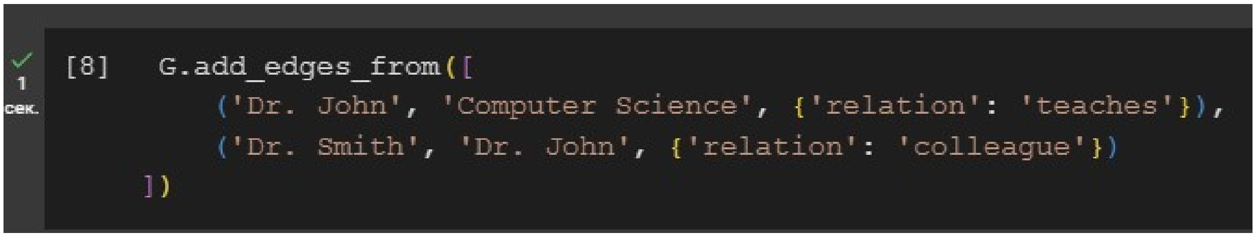


Figure 6: Graph 1

Step 7:



```
[8]   G.add_edges_from([
          ('Dr. John', 'Computer Science', {'relation': 'teaches'}),
          ('Dr. Smith', 'Dr. John', {'relation': 'colleague'})
      ])
```

Figure 7: Step 7: Adding more edges (example)

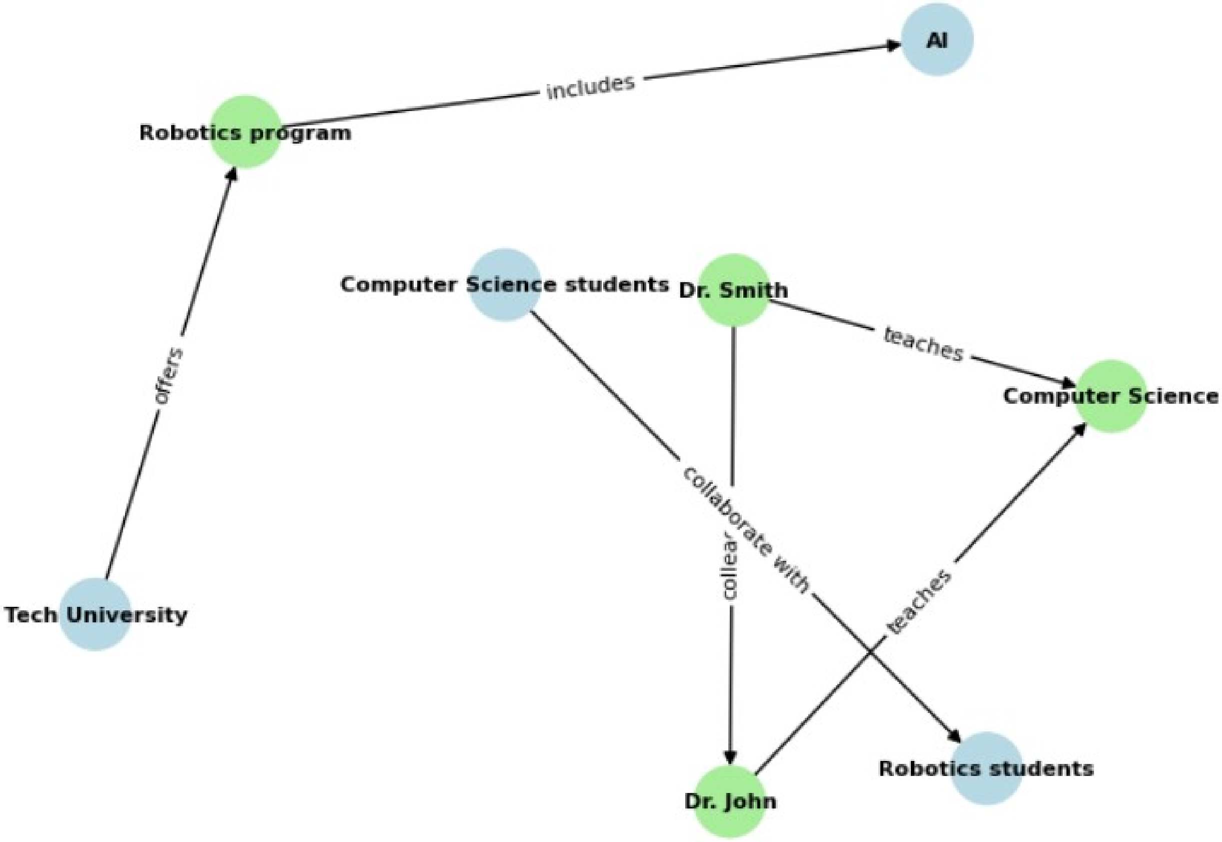
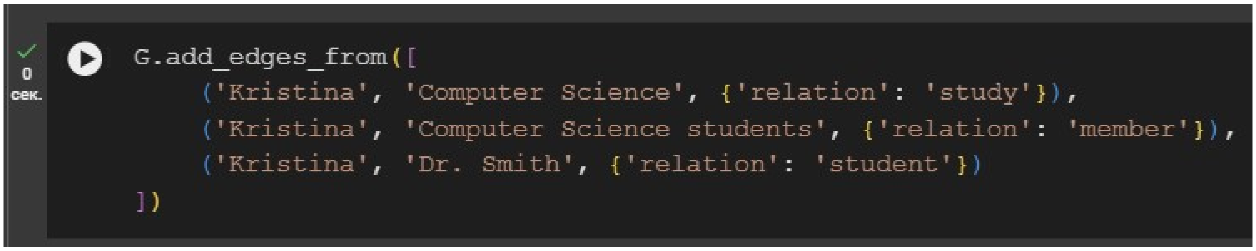


Figure 8: Graph 2

Step 8:



```
G.add_edges_from([
    ('Kristina', 'Computer Science', {'relation': 'study'}),
    ('Kristina', 'Computer Science students', {'relation': 'member'}),
    ('Kristina', 'Dr. Smith', {'relation': 'student'})
])
```

Figure 9: Step 8: Adding more edges (own version)

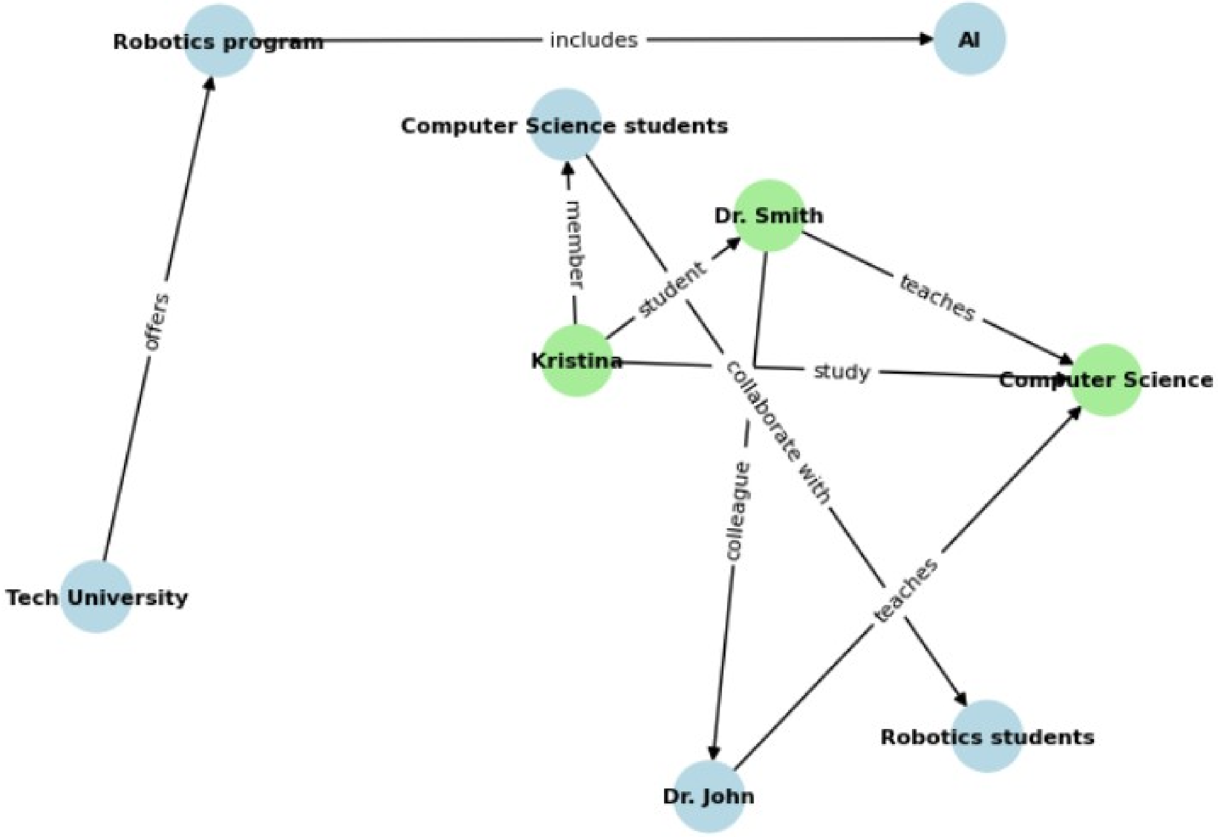


Figure 10: Graph 3

Experiment 2: Creating a Movie Graph Database in Neo4j

Tasks:

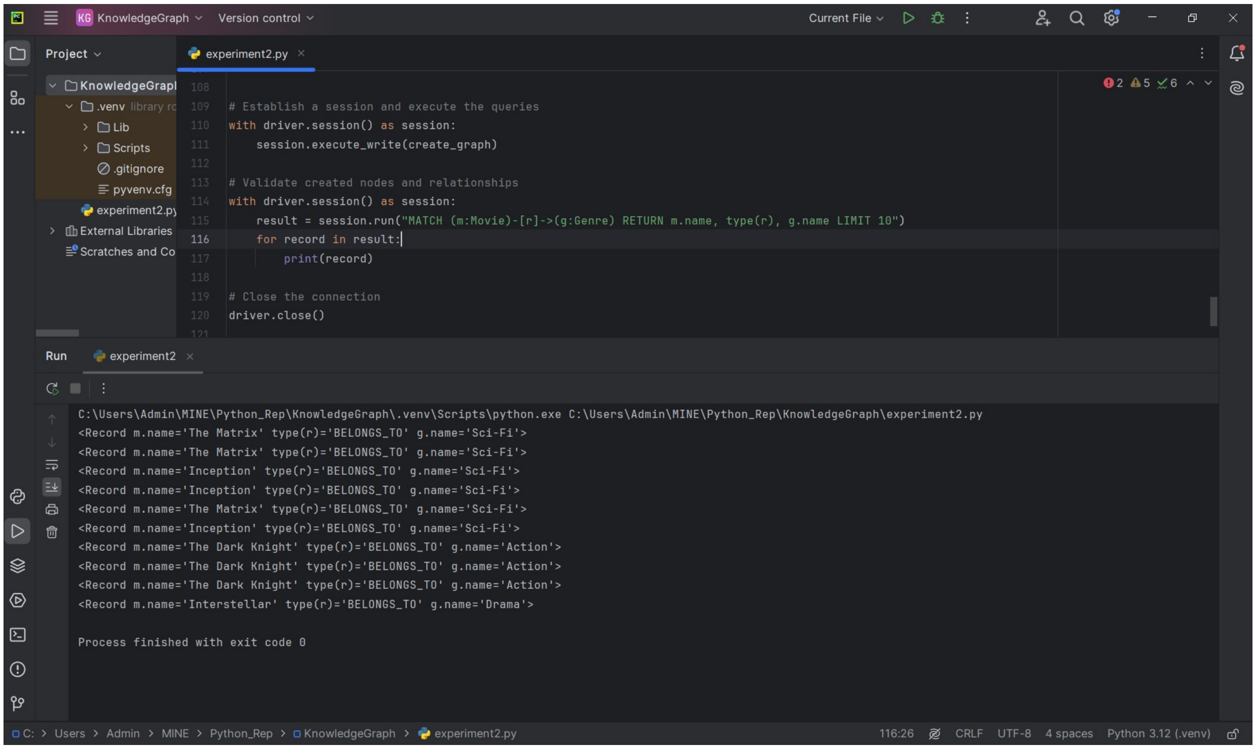
1. Define the structure of the graph database (nodes and relationships).
2. Create nodes for Movies, Genres, Actors, Directors, and Production Companies.
3. Establish relationships between the nodes using Cypher queries.
4. Execute the script in Neo4j to populate the database.
5. Validate the graph structure by querying the database.

Code:

```python
from neo4j import GraphDatabase

# Connection details
uri = "bolt://localhost:7687"  # Default URI for Neo4j
username = "neo4j"  # Default username
password = "0gy030424"  # Replace with your Neo4j password

# Create a driver instance
driver = GraphDatabase.driver(uri, auth=(username, password))


def create_graph(tx):
    # Create Movie nodes
    tx.run("CREATE (:Movie {name: 'Inception'})")
    tx.run("CREATE (:Movie {name: 'The Dark Knight'})")
    tx.run("CREATE (:Movie {name: 'Interstellar'})")
    tx.run("CREATE (:Movie {name: 'Pulp Fiction'})")
    tx.run("CREATE (:Movie {name: 'The Matrix'})")

    # Create Genre nodes
    tx.run("CREATE (:Genre {name: 'Sci-Fi'})")
    tx.run("CREATE (:Genre {name: 'Action'})")
    tx.run("CREATE (:Genre {name: 'Drama'})")
    tx.run("CREATE (:Genre {name: 'Thriller'})")

    # Create Actor nodes
    tx.run("CREATE (:Actor {name: 'Leonardo DiCaprio'})")
    tx.run("CREATE (:Actor {name: 'Keanu Reeves'})")
    tx.run("CREATE (:Actor {name: 'Uma Thurman'})")
    tx.run("CREATE (:Actor {name: 'Matthew McConaughey'})")
    tx.run("CREATE (:Actor {name: 'Christian Bale'})")

    # Create Director nodes
    tx.run("CREATE (:Director {name: 'Christopher Nolan'})")
    tx.run("CREATE (:Director {name: 'Quentin Tarantino'})")

    # Create Production Company nodes
    tx.run("CREATE (:ProductionCompany {name: 'Warner Bros'})")
    tx.run("CREATE (:ProductionCompany {name: 'Miramax'})")
    tx.run("CREATE (:ProductionCompany {name: 'Syncopy'})")
    tx.run("CREATE (:ProductionCompany {name: 'Legendary Pictures'})")

    # Create relationships between Movies and Genres
    tx.run(
        "MATCH (m:Movie {name: 'Inception'}), (g:Genre {name: 'Sci-Fi'}) CREATE (m)-[:BELONGS_TO]->(g)"
    )
    tx.run(
        "MATCH (m:Movie {name: 'The Dark Knight'}), (g:Genre {name: 'Action'}) CREATE (m)-[:BELONGS_TO]->(g)"
    )
    tx.run(
        "MATCH (m:Movie {name: 'Interstellar'}), (g:Genre {name: 'Drama'}) CREATE (m)-[:BELONGS_TO]->(g)"
    )
    tx.run(
        "MATCH (m:Movie {name: 'Pulp Fiction'}), (g:Genre {name: 'Thriller'}) CREATE (m)-[:BELONGS_TO]->(g)"
    )
    tx.run(
        "MATCH (m:Movie {name: 'The Matrix'}), (g:Genre {name: 'Sci-Fi'}) CREATE (m)-[:BELONGS_TO]->(g)"
    )

    # Create relationships between Movies and Actors
    tx.run(
        "MATCH (m:Movie {name: 'Inception'}), (a:Actor {name: 'Leonardo DiCaprio'}) CREATE (m)-[:STARS]->(a)"
    )
    tx.run(
        "MATCH (m:Movie {name: 'The Matrix'}), (a:Actor {name: 'Keanu Reeves'}) CREATE (m)-[:STARS]->(a)"
    )
    tx.run(
        "MATCH (m:Movie {name: 'Pulp Fiction'}), (a:Actor {name: 'Uma Thurman'}) CREATE (m)-[:STARS]->(a)"
    )
    tx.run(
        "MATCH (m:Movie {name: 'Interstellar'}), (a:Actor {name: 'Matthew McConaughey'}) CREATE (m)-[:STARS]->(a)"
    )
    tx.run(
        "MATCH (m:Movie {name: 'The Dark Knight'}), (a:Actor {name: 'Christian Bale'}) CREATE (m)-[:STARS]->(a)"
    )

    # Create relationships between Movies and Directors
    tx.run(
        "MATCH (m:Movie {name: 'Inception'}), (d:Director {name: 'Christopher Nolan'}) CREATE (m)-[:DIRECTED_BY]->(d)"
    )
    tx.run(
        "MATCH (m:Movie {name: 'The Dark Knight'}), (d:Director {name: 'Christopher Nolan'}) CREATE (m)-[:DIRECTED_BY]->(d)"
    )
    tx.run(
        "MATCH (m:Movie {name: 'Interstellar'}), (d:Director {name: 'Christopher Nolan'}) CREATE (m)-[:DIRECTED_BY]->(d)"
    )
    tx.run(
        "MATCH (m:Movie {name: 'Pulp Fiction'}), (d:Director {name: 'Quentin Tarantino'}) CREATE (m)-[:DIRECTED_BY]->(d)"
    )

    # Create relationships between Movies and Production Companies
    tx.run(
        "MATCH (m:Movie {name: 'Inception'}), (p:ProductionCompany {name: 'Syncopy'}) CREATE (m)-[:PRODUCED_BY]->(p)"
    )
    tx.run(
        "MATCH (m:Movie {name: 'The Dark Knight'}), (p:ProductionCompany {name: 'Legendary Pictures'}) CREATE (m)-[:PRODUCED_BY]->(p)"
    )
    tx.run(
        "MATCH (m:Movie {name: 'Pulp Fiction'}), (p:ProductionCompany {name: 'Miramax'}) CREATE (m)-[:PRODUCED_BY]->(p)"
    )
    tx.run(
        "MATCH (m:Movie {name: 'The Matrix'}), (p:ProductionCompany {name: 'Warner Bros'}) CREATE (m)-[:PRODUCED_BY]->(p)"
    )
    tx.run(
        "MATCH (m:Movie {name: 'Interstellar'}), (p:ProductionCompany {name: 'Syncopy'}) CREATE (m)-[:PRODUCED_BY]->(p)"
    )


# Establish a session and execute the queries
with driver.session() as session:
    session.execute_write(create_graph)

# Close the connection
driver.close()
```

This experiment helps to understand how to construct both a Knowledge Graph and a Movie Graph Database. I learned to visualize relationships in a knowledge graph using NetworkX and how to implement a graph database schema for movie-related data in Neo4j.

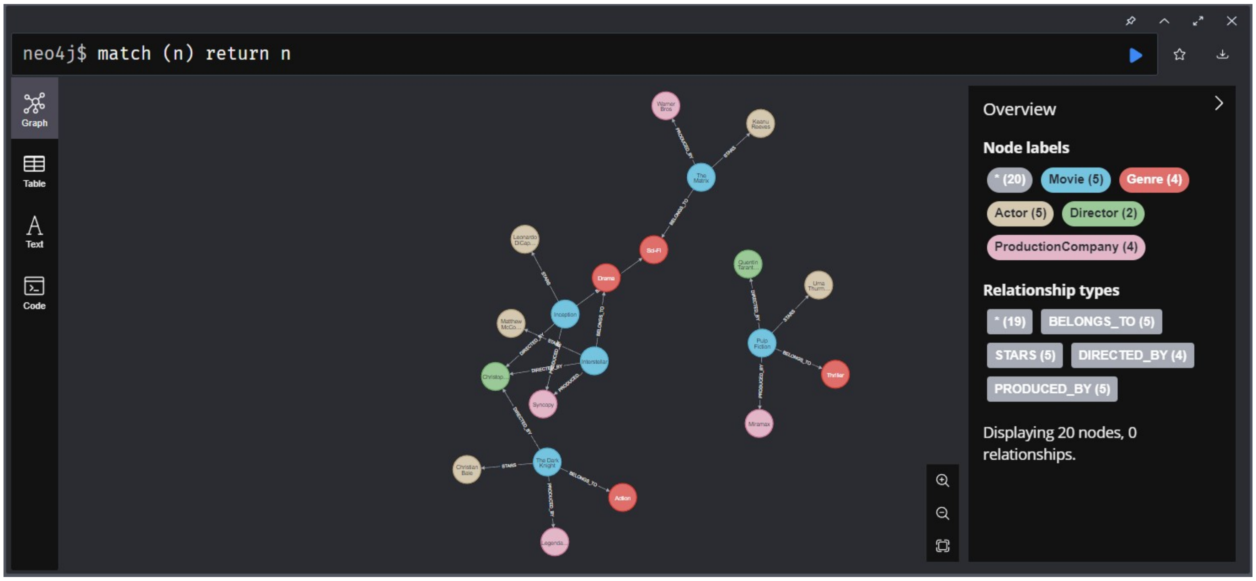
Result:



```python

# Establish a session and execute the queries
with driver.session() as session:
    session.execute_write(create_graph)

# Validate created nodes and relationships
with driver.session() as session:
    result = session.run("MATCH (m:Movie)-[r]->(g:Genre) RETURN m.name, type(r), g.name LIMIT 10")
    for record in result:
        print(record)

# Close the connection
driver.close()
```

```
C:\Users\Admin\MINE\Python_Rep\KnowledgeGraph\.venv\Scripts\python.exe C:\Users\Admin\MINE\Python_Rep\KnowledgeGraph\experiment2.py
<Record m.name='The Matrix' type(r)='BELONGS_TO' g.name='Sci-Fi'>
<Record m.name='The Matrix' type(r)='BELONGS_TO' g.name='Sci-Fi'>
<Record m.name='Inception' type(r)='BELONGS_TO' g.name='Sci-Fi'>
<Record m.name='Inception' type(r)='BELONGS_TO' g.name='Sci-Fi'>
<Record m.name='The Matrix' type(r)='BELONGS_TO' g.name='Sci-Fi'>
<Record m.name='Inception' type(r)='BELONGS_TO' g.name='Sci-Fi'>
<Record m.name='The Dark Knight' type(r)='BELONGS_TO' g.name='Action'>
<Record m.name='The Dark Knight' type(r)='BELONGS_TO' g.name='Action'>
<Record m.name='The Dark Knight' type(r)='BELONGS_TO' g.name='Action'>
<Record m.name='Interstellar' type(r)='BELONGS_TO' g.name='Drama'>

Process finished with exit code 0
```

Figure 11: Code execution



```
neo4j$ match (n) return n
```

Overview

Node labels

* (20) Movie (5) Genre (4) Actor (5) Director (2) ProductionCompany (4)

Relationship types

* (19) BELONGS_TO (5) STARS (5) DIRECTED_BY (4) PRODUCED_BY (5)

Displaying 20 nodes, 0 relationships.

Figure 12: Graph in Neo4j browser