Neo4j Movie Recommendations: Using graphDatabase

By:
Rita Marina Saker (210100)
Joelle Jouni (243215)
Ali Dia (141926)
Hassan Karnib (243353)

Introduction

In this report, we discuss the integration of a Neo4j graph database with MovieLens data to perform data analysis and provide movie recommendations. The integration is achieved using the graphdatascience library in Python. The Neo4j database is connected, and various operations are performed to analyze and query the MovieLens dataset.

Connecting to Neo4j Database

The Neo4j database connection details include the URL, username, and password. After establishing the connection, basic information about the database is retrieved, such as the Neo4j version and the count of nodes in the graph.
And this is the part you should modify once you launch the program and get your user,password and boltURL.

```
DB_ULR = 'bolt://3.238.155.148:7687' #change boltURL
DB_USER = 'neo4j'                    #change user
DB_PASS = 'growth-laboratory-party'  #change password
```

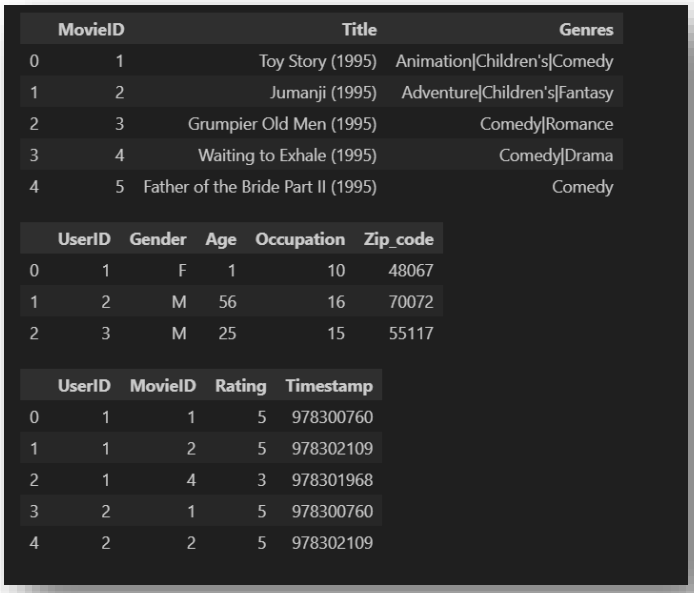
We retrieved the count of all nodes in our Neo4j graph database. It uses the Graph Data Science library's **run_cypher** method to execute a Cypher query. The query, **MATCH (n) RETURN COUNT(n)**, finds all nodes represented by the variable n and returns the count of those nodes. The result is a DataFrame containing a single column with the count of nodes. *This information is valuable for understanding the size and structure of your graph, providing a foundational insight into the overall composition of your data*.

Loading MovieLens Data

MovieLens data, obtained from GroupLens.org, specifically the "1M Dataset," is loaded into Pandas dataframes. The dataset includes information about **movies**, **ratings**, and **users**.

```
display(movies)
display(users)
display(ratings)
```

The code snippet loads movie, user, and rating data into Pandas DataFrames (**movies**, **users**, and **ratings**) from CSV files. The respective columns for each DataFrame include movie details ('MovieID', 'Title', 'Genres'), user information ('**UserID**', 'Gender', 'Age', 'Occupation', 'Zip_code'), and user ratings ('UserID', 'MovieID', 'Rating', 'Timestamp'). This data preparation step is crucial for subsequent analysis and exploration of the dataset.



	MovieID	Title	Genres
0	1	Toy Story (1995)	Animation|Children's|Comedy
1	2	Jumanji (1995)	Adventure|Children's|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama
4	5	Father of the Bride Part II (1995)	Comedy

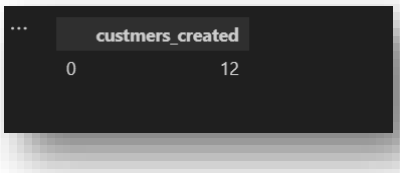
	UserID	Gender	Age	Occupation	Zip_code
0	1	F	1	10	48067
1	2	M	56	16	70072
2	3	M	25	15	55117

	UserID	MovieID	Rating	Timestamp
0	1	1	5	978300760
1	1	2	5	978302109
2	1	4	3	978301968
3	2	1	5	978300760
4	2	2	5	978302109

Inserting Data into the Graph Database

User and movie nodes are created in the Neo4j graph database, and relationships between users and movies are established based on the movie ratings.

- We made a uniqueness constraint for the 'User' label based on the 'id' property. It then creates user nodes in the Neo4j graph database using the provided user data. The 'merge' operation ensures that nodes are created only if they do not already exist, based on the specified 'UserID'. The properties 'Gender' and 'Age' are set for each user node. The result is a count of the created user nodes, forming a foundational step for subsequent graph-based analyses.

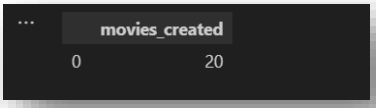


	custmers_created
0	12

THE OUTPUT CUSTOMERS_CREATED: 12 INDICATES THE SUCCESSFUL CREATION OF 12 UNIQUE USER NODES IN THE NEO4J GRAPH DATABASE. THIS OPERATION IS ESSENTIAL FOR ESTABLISHING THE

INITIAL USER NODE STRUCTURE AND ASSOCIATED PROPERTIES SUCH AS 'GENDER' AND 'AGE' IN THE GRAPH.

- The code establishes a uniqueness constraint for the 'Movie' label based on the 'id' property and creates movie nodes in the Neo4j graph database. The 'merge' operation ensures that nodes are created only if they do not already exist, based on the specified 'MovieID'. Properties such as 'Title' and 'Genres' are set for each movie node. The result, **movies_created: x**, indicates the count (**x**) of unique movies successfully created in the graph. This step is fundamental for initializing the graph with movie nodes and associated properties.



	movies_created
0	20

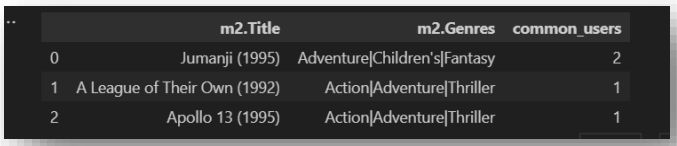
THE OUTPUT MOVIES_CREATED: 20 INDICATES THE SUCCESSFUL CREATION OF 20 UNIQUE MOVIE NODES IN THE NEO4J GRAPH DATABASE. THIS OPERATION IS ESSENTIAL FOR ESTABLISHING THE INITIAL MOVIE NODE STRUCTURE AND ASSOCIATED PROPERTIES, SUCH AS 'TITLE' AND 'GENRES', IN THE GRAPH.

The code efficiently creates 'RATED' relationships between user and movie nodes in the Neo4j graph database, iterating through chunks of the ratings data. For each chunk, it matches user and movie nodes based on 'UserID' and 'MovieID' and establishes a 'RATED' relationship with the corresponding 'Rating' property. The result provides a count of the created relationships, contributing to the construction of a comprehensive graph structure based on user ratings.

The code establishes a connection to a Neo4j database using the **neo4j** Python driver. It then queries the database to retrieve and print the schema information using the APOC procedure **apoc.meta.schema()**. This code snippet is essential for gaining insights into the graph structure and relationships in the Neo4j database.

Recommendations
Similar Movie Recommendations

A method for finding similar movies is proposed by counting the paths connecting two movies through a common user, where the movies have received a high rating from that user. An example is provided for finding similar movies to 'Toy Story (1995)'.
Based on user ratings, It looks for movies with a 5-star rating by users who also gave "Toy Story (1995)" a 5-star rating. The result includes titles, genres, and the count of common users who rated both movies. This query provides insights into movies with high user overlap and similar high ratings, serving as potential recommendations for fans of "Toy Story (1995)." The **similar_movies.head()** command previews the top results of the query.

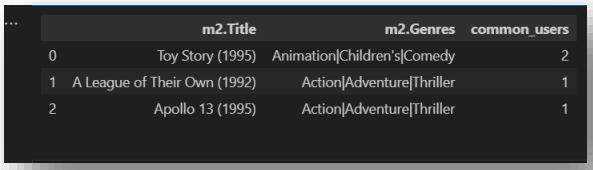


	m2.Title	m2.Genres	common_users
0	Jumanji (1995)	Adventure|Children's|Fantasy	2
1	A League of Their Own (1992)	Action|Adventure|Thriller	1
2	Apollo 13 (1995)	Action|Adventure|Thriller	1

The code queries for movies similar to "Jumanji (1995)" based on user ratings. It identifies movies with a 5-star rating by users who also rated "Jumanji (1995)" with 5 stars. The result includes titles, genres, and the count of distinct users who rated both movies. The **similar_movies.head()** command provides a preview of the top results from the query, showcasing movies that share a high rating and user

overlap with "Jumanji (1995)." This type of query is valuable for recommending movies to users who enjoyed a specific film, leveraging the preferences of a common user base.

	m2.Title	m2.Genres	common_users
0	Toy Story (1995)	Animation|Children's|Comedy	2
1	A League of Their Own (1992)	Action|Adventure|Thriller	1
2	Apollo 13 (1995)	Action|Adventure|Thriller	1



	m2.Title	m2.Genres	common_users
0	Toy Story (1995)	Animation|Children's|Comedy	2
1	A League of Their Own (1992)	Action|Adventure|Thriller	1
2	Apollo 13 (1995)	Action|Adventure|Thriller	1

User-Based Recommendations

User-based recommendations are generated by calculating the Jaccard similarity between users. The process involves creating a graph projection and calculating the similarity scores between users. A new relationship, 'SIMILAR,' is created between similar users.

To recommend movies to a specific user ('User 1'), similar users are identified, and movies they have rated are considered. The recommendation score for each movie is calculated by considering the weighted average of ratings from similar users, with a logarithmic term to boost movies connected through multiple paths.

Making a connection to a Neo4j database, drops an existing graph named 'ProjectttNeo4jGraph', and creates a new graph projection. This projection defines nodes labeled 'User' and 'Movie' with 'RATED' relationships, considering the 'Rating' property. The output does not display a visual result but signifies the successful configuration of the graph projection for subsequent graph-based analyses in Neo4j.

The code calculates user similarity in a Neo4j graph projection named 'ProjectttNeo4j' using the Graph Data Science library. It yields pairs of users along with their similarity scores, ordered by descending similarity. This information is useful for identifying users with similar preferences or behavior in the graph. The output is a DataFrame containing user pairs (UserID1 and UserID2) and their corresponding similarity scores.

	UserID1	UserID2	similarity
0	1	9	0.500000
1	2	5	0.500000
2	4	5	0.500000
3	5	2	0.500000
4	5	4	0.500000
5	9	1	0.500000
6	1	2	0.333333
7	2	1	0.333333
8	3	4	0.333333
9	4	3	0.333333
10	1	4	0.250000
11	1	8	0.250000
12	2	4	0.250000
13	3	5	0.250000
14	4	1	0.250000
15	4	2	0.250000
16	5	3	0.250000
17	8	1	0.250000
18	1	5	0.200000
19	2	9	0.200000
20	5	1	0.200000
21	9	2	0.200000
22	1	3	0.166667
23	2	3	0.166667

The code divides user similarity data into chunks and iteratively creates 'SIMILAR' relationships between users in a Neo4j graph. It matches user nodes, merges 'SIMILAR' relationships, and sets the 'Similarity' property for each chunk. The results are stored in a DataFrame named create_similar_df, displaying the count of created relationships for each iteration. This process efficiently establishes connections between similar users in the graph based on their similarity scores.

```
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10
```

	create_rated
0	2
0	2
0	2
0	1
0	1
0	1
0	1
0	1
0	1
0	1

The code then checks for movies that are similar to those rated by a specific user (user with ID 1) based on user similarity scores and movie ratings. It matches similar users, their rated movies, and calculates a score for each movie considering both similarity and rating. The result, stored in the DataFrame similar_movies_for_user, includes movie titles, genres, and scores, ordered by descending score.
The second part of the code retrieves actual movies rated by the same user (user with ID 2). It matches the user's rated movies and returns their titles, genres, and ratings. The result is stored in the DataFrame movies_for_user, displaying the top-rated movies for the specified user.
These queries provide valuable insights into both recommended movies based on user similarity and the user's actual movie preferences.

	m.Title	m.Genres	rating
0	GoldenEye (1995)	Action|Adventure|Thriller	5
1	Johnny Mnemonic (1995)	Action	4
2	Toy Story (1995)	Animation|Children's|Comedy	4

	m.Title	m.Genres	score
0	GoldenEye (1995)	Action|Adventure|Thriller	5.000000
1	Apollo 13 (1995)	Action|Adventure|Thriller	4.568147
2	Johnny Mnemonic (1995)	Action	4.000000
3	Waiting to Exhale (1995)	Comedy|Drama	3.000000

Conclusion

The integration of Neo4j with MovieLens data provides a powerful platform for graph-based analysis and recommendations. The approach outlined in this report combines graph algorithms and data science techniques to generate meaningful recommendations based on user preferences and movie similarities. The flexibility of Neo4j allows for further exploration and enhancement of recommendation strategies.