Graph-based metadata management of Bnbinsight

Yutao Chen[*], Jintao Ma[*]

{yutao.chen,jintao.ma}@estudiantat.upc.edu

Contents

[*]Universitat Politècnica de Catalunya (UPC), Erasmus Mundus Big Data Management and Analytics (BDMA)

1 Purpose and Benefits of Graph-Based Solutions

In our project, we aim to provide price prediction service for the small to medium Bed and Breakfast owners located in the Catalonia area. So we will utilize graph-based solutions to ensure our data-driven decision-making service has better performance. By constructing a property graph, we could build a complex relationship between entities like listings, hosts, amenities, and locations. It will give our start-up company BnB Insight better understanding of our customers and the market.

The benefits of this proposal include:

- **Enhanced Data Relationships:** Utilizing the property graph database could bring a better understanding to our start-up business. And help us understand the complex relations between entities by creating Visualizations. For example, the association between the host and their properties, and the relation between amenities and listings.
- **Improved Query Performance:** In this project, we utilize Neo4j as one of our tools, due to the connected feature of Neo4j, we could get a better query performance in this project, the response time of query especially for complex query will be improved.
- **Advanced Analytics Capabilities:** The property graph supports advanced data analysis and Machine Learning algorithms. It could be a big advantage for us to implement the data-driven dynamic price prediction, effectively improve the speed, and easily conduct the visualization.
- **Scalability and Flexibility:** By utilizing the property graph, we could make more scalability of our project, due to the flexibility feature of the property graph. In our project, the everyday Catalonia area hotel prices and subway passengers will be recorded every day, so scalability is important in this case. And thanks to the flexi-

bility feature of the property graph, new attributes in the future could be added easily if applicable.

- **Real-time Data Insights:** The nature of facilitating real-time data analysis of property graphs makes it suitable for our dream tool. The tourism industry is always sensitive to the real-time data.

The code is available on GitHub at the following links GitHub: SDM-Joint-Project:

2 Selection of Graph Family and Justification

2.1 Property Graphs

The benefit of Property Graphs could be named a lot, firstly they own rich structures with properties on both nodes and edges. In our project, nodes like ReviewScore could have properties of listing_id, number_of_reviews_l30d, last_review, and so on.

Apart from this, the Versatility in Relationship Representation is another important factor that helps us make the decision. For example, the properties of the listing could help the listing connect with the host describing the first-serve data of a host.

Continuously, the requirement for the dynamic and reactive nature of the hospitality industry helps us decide to utilize the property graph in our project.

Last but not least, the property graph has lots of technologies and tools available, like the Neo4j, which offers robust functions to implement the solutions.

The reason we chose property graphs over knowledge graphs is thanks to all the advantages we mentioned above, and knowledge graphs are more suitable for scenarios that integrate various data sources into a coherent schema, focusing primarily on knowledge discovery and aggregation rather than operational analysis. They are less flexible when dealing with high-transaction, attribute-rich operational data typical of hotel management.

2.2 Cypher

The main reason we chose Cypher as the query language in this project is because Cypher could be implemented and maintained simply due to its easy syntax, and it performs well when querying the complex relations and properties within the graph. Also, as a declarative graph query language, the expressive and efficient query is favored by our team.

3 Graph Schema Design and Meta-Model

In our property graph design, we defined six main node types (Listing, Host, Calendar, ReviewScore, Location, Amenity) and their key attributes, constructing a comprehensive model to manage the complex relationships between listings, hosts, scores, locations, calendar records, and amenities. Node attributes such as the id, name, type, and price of listings; the id, name, start date, and superhost status of hosts; and geographical information and amenity details provide detailed information support. Relationship types include hosts owning listings (OWN), calendar records of listings (HAS_CALENDAR), review scores of listings (HAS_SCORE), geographical locations of listings and hosts (LOCATED_IN and PROPERTIES_LOCATED), and amenities of listings (HAS_AMENITY). These relationships describe the complex interactions of ownership, time, ratings, geographical distribution, and amenities between nodes. This design provides the system with flexible and efficient data management and query capabilities, meeting user needs in searching, evaluating, and selecting listings, and supporting detailed data analysis and optimization.

3.1 Nodes

There were six node types: Listing, Host, Calendar, ReviewScore, Location, and Amenity. These nodes and attributes provide detailed descriptions of the properties and related information, supporting users in searching, evaluating, and selecting listings by offering all necessary details.

Node Name	Description	Key Attributes
Listing	individual property listings.	id, name, property_type, room_type
Host	the hosts who own the listings.	host_id, host_name:, host_since, host_is_superhost
Calendar	the availability and price information for listings on specific dates.	listing_id, date, price, available
ReviewScore	the review scores of listings.	listing_id, review_scores_rating
Location	geographical information about listings.	latitude, longitude, neighbourhood
Amenity	various amenities available in the listings.	amenity_detail

3.2 Node Instances

Node Type	Instance Details
Listing Node	{id: 101, name: ”Beachfront Apartment”, description: ”Stunning sea view”, listing_url: ”http://example.com/101”, latitude: 41.3851, longitude: 2.1734}
Host Node	{host_id: 201, host_name: ”John Doe”, host_since: ”2015-04-22”}
Amenity Node	{amenity_detail: ”WiFi”}
Location Node	{neighbourhood: ”Beachside”, latitude: 41.3851, longitude: 2.1734}
Calendar Node	{listing_id: 101, date: ”2024-03-20”, available: ”f”, price: 47.0}
ReviewScore Node	{listing_id: 101, review_scores_rating: 95.0}

3.3 Relationships

The relationship design defines the associations between nodes, including: OWN (hosts own listings), HAS_CALENDAR (listings have calendar records), HAS_SCORE (listings have review scores), LOCATED_IN (listings are located in a geographical location), PROPERTIES_LOCATED (a host's properties are distributed across different locations), and HAS_AMENITY (listings have various amenities). These relationships not only describe the actual connections between nodes but also reflect the complex interactions between properties, hosts, reviews, calendar records, locations, and amenities in the hotel management system, providing the system with flexible and efficient data management and query capabilities.

Relationship Type	Description
OWN (Host → Listing)	A host owns or manages one or more listings.
HAS_CALENDAR (Listing → Calendar)	A listing has one or more calendar entries detailing availability and pricing.
HAS_SCORE (Listing → ReviewScore)	A listing has one or more review scores, reflecting customer feedback.
LOCATED_IN (Listing → Location	A listing is associated with geographical locations.
HAS_AMENITY (Listing → Amenity)	A listing offers various amenities, like WiFi, parking, etc.
PROPERTIES_LOCATED (Host → Location)	A host's listings are distributed across different locations.

3.4 Relation Instances

Relationship	Instance Details
HAS_HOST	(Host {host_id: 201}) -[HAS_HOST]->(Listing {id: 101})
HAS_AMENITY	(Listing {id: 101}) -[HAS_AMENITY]->(Amenity {amenity_detail: ”WiFi”})
LOCATED_IN	(Listing {id: 101}) -[LOCATED_IN]->(Location {neighbourhood: ”Beachside”})
HAS_CALENDAR	(Listing {id: 101}) -[HAS_CALENDAR]->(Calendar {date: ”2024-03-20”})
HAS_SCORE	(Listing {id: 101}) -[HAS_SCORE]->(ReviewScore {review_scores_rating: 95.0})

3.5 Property Graph Design

The visualization graph is from Figure 1. The source graph is available at Property Graph.

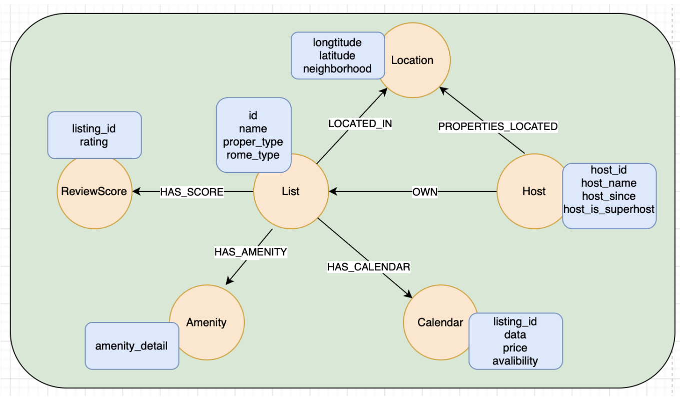


Figure 1: property graph design

4 Data Sources and Graph Population Strategy

4.1 Identifying Data Sources

The data source of this project is from the online platform Websit: Inside Airbnb.We downloaded listings and calendar and neighbourhoods data of Barcelona.

4.2 Designing Data Flow Processes

Merging datasets: Firstly, we merged the data from the listings and calendar files, as mentioned above.They have common attribute columns called listing_id. We had to process three kinds of columns so that the CSV file could be uploaded successfully.They were amenities column,location_related columns and Price Column.The full code is available at Transformation by Colab.

```
df_listing = pd.read_csv('/content/drive/My Drive/Colab Notebooks/bdm/listings.csv')
df_calendar = pd.read_csv('/content/drive/My Drive/Colab Notebooks/bdm/calendar1.csv')
df_merged = pd.merge(df_calendar, df_listing, left_on='listing_id', right_on='id', how='left')
```

Processing the amenities Column: We defined the function clean_amenities to process the strings in the amenities column by parsing them into a list and converting them into a comma-separated string.Then we Applied this function to clean the data in the amenities column.

```
def clean_amenities(amenities):
    try:
        amenities_list = ast.literal_eval(amenities)
        if isinstance(amenities_list, list):
            return ','.join([amenity.strip() for amenity in amenities_list])
        else:
            return amenities
    except:
        return amenities

df['amenities'] = df['amenities'].apply(clean_amenities)
```

Processing Location-Related Columns: We defined a list of location-related column names location_columns, including neighbourhood, city, state, country, latitude, and longitude. Process these columns by cleaning missing values (filling with ”Unknown”) or converting latitude and longitude to numeric types. If a column does not exist, print a corresponding message.

```
location_columns = ['neighbourhood', 'city', 'state', 'country', 'latitude', 'longitude']
for column in location_columns:
    if column in df.columns:
        if column in ['latitude', 'longitude']:
            df[column] = pd.to_numeric(df[column], errors='coerce')
        else:
            df[column] = df[column].fillna('Unknown')
    else:
        print(f"column '{column}' not exist")
```

Processing the Price Columns: We defined the function clean_price to remove currency symbols from the price fields and convert them to floating-point numbers.We Applied this function to clean the price_x and adjusted_price columns.

```
def clean_price(price):
    try:
        return float(price.replace('$', '').replace(',', '').strip())
    except:
        return None
```

Loading to Neo4j:

Lastly, the data would be loaded to Neo4j by Python library neo4j, we will create a database connection first, then create the database driver then load the CSV Query, after executing the query, we close the server.Loading. Although we just listed the key attributes of our nodes,we still created a lot of properties of our nodes.The example of creating listing node is in the following code:

```
// Create Listing node and establish relationships
MERGE (l:Listing {id: toInteger(row.id)})
ON CREATE SET
    l.listing_url = row.listing_url,
    l.name = row.name,
    l.description = row.description,
    l.neighborhood_overview = row.neighborhood_overview,
    l.picture_url = row.picture_url,
    l.instant_bookable = row.instant_bookable,
    l.license = row.license,
    l.property_type = row.property_type,
    l.room_type = row.room_type,
    l.accommodates = toInteger(row.accommodates),
    l.bathrooms_text = row.bathrooms_text,
    l.bedrooms = toInteger(row.bedrooms),
    l.beds = toInteger(row.beds),
    l.latitude = CASE WHEN row.latitude IS NOT NULL
    THEN toFloat(row.latitude) ELSE null END,
    l.longitude = CASE WHEN row.longitude IS NOT
    NULL THEN toFloat(row.longitude) ELSE null END
MERGE (l)-[:LOCATED_IN]->(loc)
```

5 Utilizing Graph Data: Processes and Applications

5.1 Data pipeline

The data pipeline of the BnBInsight project is available in graph 2.Our BnBInsight is a provider of data analytics predictions solutions empowering hotels in Catalonia to maximize revenue and competitiveness in the dynamic tourism market.For our SDM part of the whole project, the Graph Data, Neo4J, is primarily utilized in both the descriptive and predictive analysis phases. We used Neo4j analysed datasets and the results of it helped to make predictive analysis of BDM part.There are several compelling reasons and benefits for integrating Neo4J into this data pipeline:

Firstly, the data managed by BnB Insight is highly diverse, encompassing various types of information such as pricing data, calendar data, location information. Neo4J's flexible schemaless nature makes it particularly adept at handling such heterogeneous data. It can seamlessly map and query different entities, facilitating complex relationships and interactions that are integral to the project's success.

Moreover, Neo4J excels in graph-based data management, which is crucial for uncovering hidden patterns and relationships within the data. This capability is leveraged extensively in both the descriptive analysis, where we seek to understand current trends and relationships, and the predictive analysis, where we aim to forecast future trends and behaviors based on historical data.

The integration of Neo4J allows for advanced query capabilities, enabling the exploration of intricate relationships between different data points. For instance, we can easily query the relationship between property prices and their proximity to some specific area or analyze the impact of seasonal variations on overnight stays. Such detailed queries and relationship explorations are essential for gaining deeper insights and making informed decisions.

Additionally, Neo4J supports powerful visualization tools that help in representing data relationships graphically. This is invaluable for both data scientists and stakeholders to visualize complex networks and dependencies in an intuitive manner, enhancing their understanding and facilitating better decision-making.

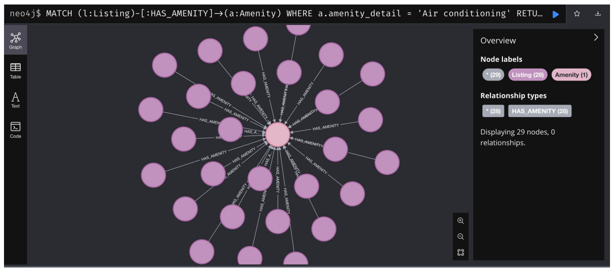
Below, we will demonstrate some of the queries and relationship explorations made possible through Neo4J, showcasing its capabilities in handling the diverse and interconnected data of the BnB Insight project. This includes examples of how we map various entities, perform sophisticated queries, and derive actionable insights from the interconnected data points.

5.2 Querying

We used some interesting and specific queries to validate our property graph.These queries would be also useful in our daily life.

- **Query properties with air conditioning available**

  In Barcelona, the summer can be very hot. We need to know which properties have air conditioning facilities. Knowing which properties are equipped with air conditioning can help hotel owners make more informed pricing decisions. Typically, properties with air conditioning can command higher prices.



```
neo4j$ MATCH (l:Listing)-[:HAS_AMENITY]→(a:Amenity) WHERE a.amenity_detail = 'Air conditioning' RETU…
```

Displaying 29 nodes, 0 relationships.

Figure 3: query1

- **Query properties with ratings higher than 4.5**

  Properties with ratings higher than 4.5 typically indicate high tenant satisfaction. By analyzing the trends of these high-rated properties, we can identify the factors that significantly impact tenant satisfaction and apply these insights to the development of new properties and the management of existing ones.

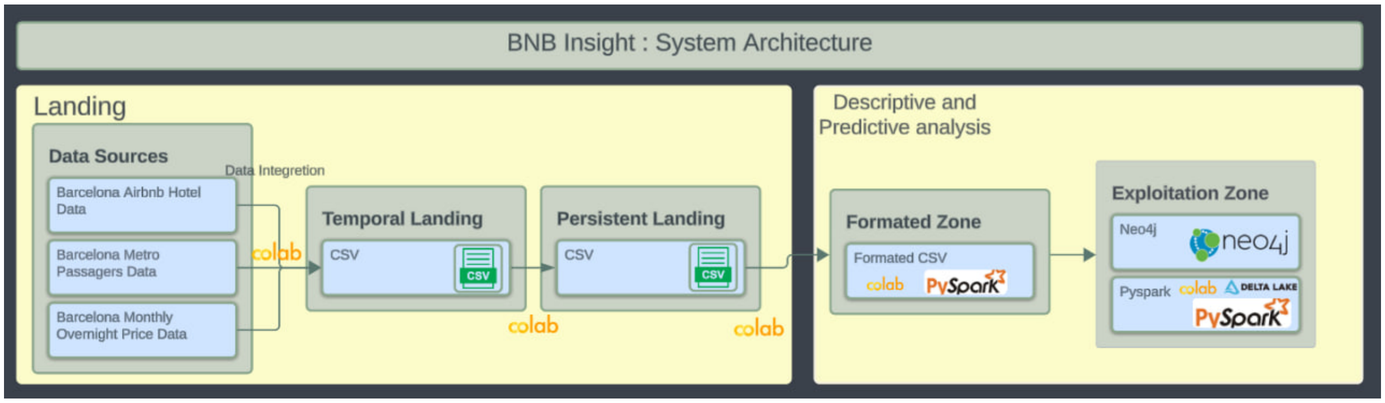
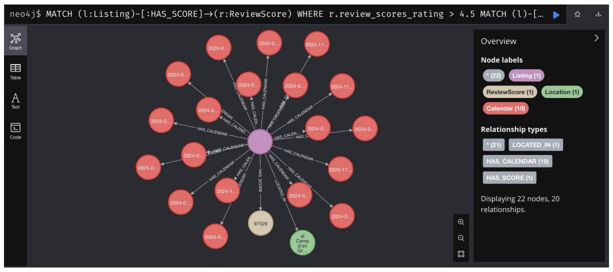


Figure 2: Data Pipeline

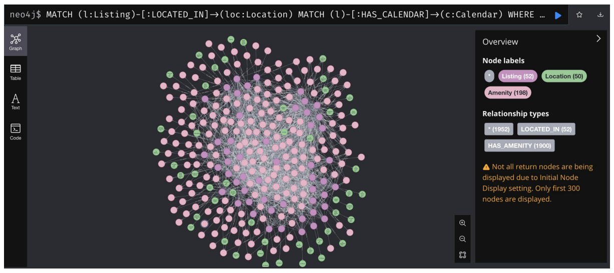


```
neo4j$ MATCH (l:Listing)-[:HAS_SCORE]→(r:ReviewScore) WHERE r.review_scores_rating > 4.5 MATCH (l)-[…
```

Figure 4: query2

- **Query properties and prices near Sagrada Família:**

  Sagrada Família is one of the most famous landmarks in Barcelona, attracting a large number of tourists every year. Based on the characteristics of properties and tenant feedback in this area, hotel owners can optimize their properties to increase appeal and improve occupancy rates.



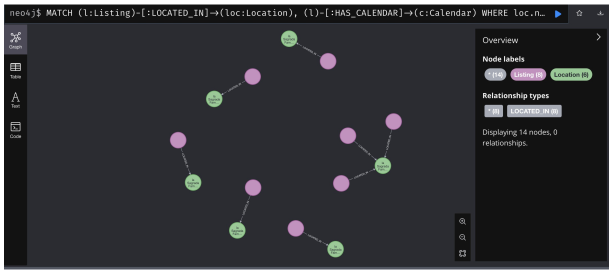
```
neo4j$ MATCH (l:Listing)-[:LOCATED_IN]→(loc:Location) MATCH (l)-[:HAS_CALENDAR]→(c:Calendar) WHERE …
```

⚠ Not all return nodes are being displayed due to Initial Node Display setting. Only first 300 nodes are displayed.

Figure 5: query3

- **Query properties, their locations, prices, and amenities on St. George's Day of Catalonia**

  St. George's Day is an important holiday in Catalonia, attracting a large number of tourists and local residents. By querying the prices of properties on St. George's Day, hotel owners can identify price fluctuation trends and dynamically adjust their property prices to maximize revenue.



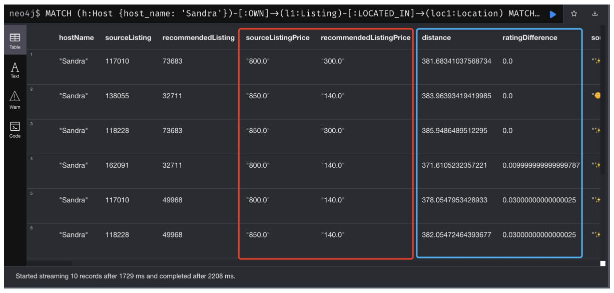
```
neo4j$ MATCH (l:Listing)-[:LOCATED_IN]→(loc:Location), (l)-[:HAS_CALENDAR]→(c:Calendar) WHERE loc.n…
```

Figure 6: query4

5.3 Reccommender

We also created two recommenders based on our property graph for hotel holders.Assuming the recommenders here is to a hotel holder named Sandra.

- **Recommender based on ratings and distance:** Recommend properties with similar amenities to the hotel owner named Sandra, and compare these properties' prices and amenities on a specific date.

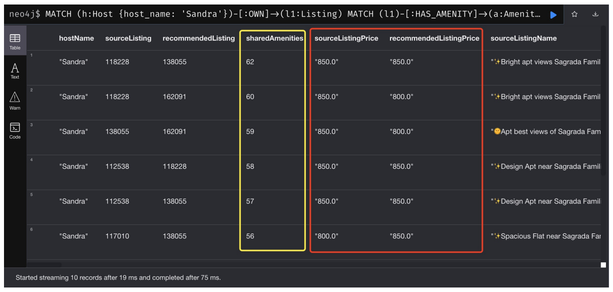


```
neo4j$ MATCH (h:Host {host_name: 'Sandra'})-[:OWN]→(l1:Listing)-[:LOCATED_IN]→(loc1:Location) MATCH…
```

	hostName	sourceListing	recommendedListing	sourceListingPrice	recommendedListingPrice	distance	ratingDifference	so
1	"Sandra"	117010	73683	"800.0"	"300.0"	381.68341037568734	0.0	"✨
2	"Sandra"	138055	32711	"850.0"	"140.0"	383.96393419419985	0.0	"😊
3	"Sandra"	118228	73683	"850.0"	"300.0"	385.9486489512295	0.0	"✨
4	"Sandra"	162091	32711	"800.0"	"140.0"	371.6105232357221	0.009999999999999787	"✨
5	"Sandra"	117010	49968	"800.0"	"140.0"	378.0547953428933	0.030000000000000025	"✨
6	"Sandra"	118228	49968	"850.0"	"140.0"	382.05472464393677	0.030000000000000025	"✨

Started streaming 10 records after 1729 ms and completed after 2208 ms.

- **Recommender based on amenities:** Recommend properties with similar amenities to the hotel owner named Sandra, and compare these properties' prices and amenities on a specific date.



```
neo4j$ MATCH (h:Host {host_name: 'Sandra'})-[:OWN]→(l1:Listing) MATCH (l1)-[:HAS_AMENITY]→(a:Amenit…
```

	hostName	sourceListing	recommendedListing	sharedAmenities	sourceListingPrice	recommendedListingPrice	sourceListingName
1	"Sandra"	118228	138055	62	"850.0"	"850.0"	"✨Bright apt views Sagrada Famil
2	"Sandra"	118228	162091	60	"850.0"	"800.0"	"✨Bright apt views Sagrada Famil
3	"Sandra"	138055	162091	59	"850.0"	"800.0"	"🌕Apt best views of Sagrada Fam
4	"Sandra"	112538	118228	58	"850.0"	"850.0"	"✨Design Apt near Sagrada Famil
5	"Sandra"	112538	138055	57	"850.0"	"850.0"	"✨Design Apt near Sagrada Famil
6	"Sandra"	117010	138055	56	"800.0"	"850.0"	"✨Spacious Flat near Sagrada Fa

```
Started streaming 10 records after 19 ms and completed after 75 ms.
```

5.4 Added Value for Recommender

These recommendations not only help hotel owners adjust their pricing strategies but also assist in machine learning predictions of hotel prices and analysis of hotel data. Below is a detailed explanation of the significance and expanded details of the recommendation system:

Optimize Pricing Strategy:

By identifying similar and nearby listings, hotel owners can obtain valuable information about market pricing. Specifically:

- **Competitive Pricing**: Understanding the pricing of nearby listings, especially those with similar ratings and amenities, can help hotel owners set competitive prices, thus attracting more tenants.
- **Price Adjustment**: Based on the recommended listing prices, hotel owners can flexibly adjust their listing prices to maximize revenue. For example, if the recommended listing prices are higher, the hotel owner can increase their prices to gain higher income; if the recommended listing prices are lower, the hotel owner might consider lowering their prices to increase rental opportunities.

Market Positioning and Demand Analysis:

By analyzing the amenities and ratings of nearby listings, hotel owners can better understand market demand and positioning:

- **Facility Optimization**: By learning about the amenities of recommended listings, hotel owners can consider adding or optimizing their own listing amenities to meet tenant needs and increase rental appeal. For example, if recommended listings have particularly popular amenities (e.g., free Wi-Fi, breakfast), the hotel owner can consider adding similar amenities.
- **Rating Improvement Strategies**: By analyzing the characteristics of highly-rated listings, hotel owners can take measures to improve their own listing ratings, such as enhancing service quality or improving the living experience. If high-rated listings generally provide excellent customer service, the hotel owner can improve service quality by training staff.

Increase Rental Efficiency:

By recommending similar listings, hotel owners can better predict market trends and rental patterns:

- **Demand Forecasting**: Understanding the rental situation and reviews of nearby listings can help hotel owners predict future rental demand and market trends, allowing them to make adjustments in advance. By analyzing the occupancy rates and tenant feedback of recommended listings, they can judge changes in market demand trends.
- **Lease Term Optimization**: Based on the rental situations and pricing strategies of recommended listings, hotel owners can optimize the lease terms of their own listings, such as offering short-term or long-term rentals to maximize occupancy rates. If nearby listings show high demand for short-term rentals, the hotel owner can increase short-term rental options to improve utilization rates.

Enhance Market Competitiveness:

By understanding and comparing various data from recommended listings, hotel owners can enhance their competitiveness in the market:

- **Data-Driven Decision Making**: By comparing the prices, ratings, and amenities of recommended listings, hotel owners can make more informed decisions and improve their market competitiveness.
- **Personalized Services**: Based on the amenities and tenant reviews of recommended listings, hotel owners can provide more personalized services to meet different tenant needs, enhancing customer satisfaction and loyalty.

Assist Machine Learning Predictions:

The data collected and analyzed through these recommendations can significantly contribute to machine learning models:

- **Training Data for Price Prediction**: The price data from similar and nearby listings can be used as training data for machine learning models to predict optimal pricing strategies. This helps in developing more accurate and dynamic pricing algorithms that consider various factors such as location, amenities, and ratings.
- **Feature Engineering**: The information on amenities, ratings, and pricing from recommended listings provides

valuable features for machine learning models. These features help in identifying patterns and correlations crucial for predictive analytics.

- **Model Validation**: The recommended listings and their data can be used to validate machine learning models. By comparing predicted prices with actual prices, hotel owners can assess the accuracy of their models and make necessary adjustments.

By leveraging Neo4j for data analysis and recommendation systems, hotel owners can gain valuable insights into market pricing, amenity optimization, and service quality improvement. This not only helps hotel owners optimize pricing strategies and improve rental efficiency but also provides a better rental experience, attracting more tenants and maximizing revenue. The practical application of the recommendation system not only deepens and broadens data analysis but also makes hotel management more scientific and efficient. Additionally, the data and insights derived from these recommendations can significantly enhance machine learning models for price prediction and advanced hotel data analysis, leading to smarter decision-making and improved market competitiveness.

6 Proof of Concept Implementation and Tool Selection

6.1 Tool Selection

Graph Database: Neo4j
Reasons for Selection:

- **Performance**: Neo4j excels in handling large-scale graph data with fast query speeds.
- **Ease of Use**: Neo4j offers the simple and intuitive Cypher query language, making it easy to learn and use.
- **Ecosystem**: Neo4j has a rich ecosystem, including various tools and libraries that support multiple programming languages.
- **Compatibility**: Neo4j is highly compatible with existing systems and data sources, supporting the import of various data formats.

Data Extraction and Preprocessing Tool: Python (pandas library)
Reasons for Selection:

- **Flexibility**: Python provides powerful data processing and analysis libraries that can flexibly handle various data formats.
- **Community Support**: Python has a wealth of community resources and support, making it easy to find solutions when encountering problems.

Query Engine: Neo4j's built-in Cypher query engine
Reasons for Selection:

- **Deep Integration**: Cypher is Neo4j's native query language, capable of efficiently executing graph queries.
- **Expressive Power**: Cypher can express complex graph query logic, making it very suitable for graph data analysis and mining.

6.2 PoC Setup

Requirements:

- **Neo4j Desktop(Preferably the latest version)**
- **Python 3.11.8**
- **Jupyter Notebook or an environment to run .ipynb files (like Google Colab)**

Data Source: Using a the dataset from this website:Airbub: Barcelona Hotel Dataset
Data Extraction and Preprocessing: Using a Python script run in Google Colab to perform data cleaning and preprocessing. The main steps include handling missing values, data type conversion, etc.The code is available at the following links Golab: SDM_Preprocessing:
Data Import into Neo4j: Using a Python script: initialize_neo4j_database to import the processed data from the CSV file into the Neo4j graph database.
Using Cypher queries: Queries and Recommenderto verify the correctness of the data and the structure of the graph on your Neo4j Browser.

6.3 Reasons for Tool Selection

- **Neo4j**: Neo4j is chosen as the graph database due to its excellent performance in handling graph data and high query efficiency. Its Cypher query language is concise and easy to use, capable of efficiently executing complex graph queries. Additionally, Neo4j has strong community support and a rich ecosystem, making it an ideal choice for handling graph data.

- **Python**: Python's pandas library provides powerful data processing capabilities, allowing flexible handling and preprocessing of various data formats. Python scripts can seamlessly integrate with the Neo4j API, facilitating data import and manipulation.
- **Cypher**: As Neo4j's native query language, Cypher has strong expressive power and can efficiently query and analyze graph data.

Through the above setup and tool selection, we can implement an end-to-end proof of concept, demonstrating how to automatically populate graphs, perform data queries, and verification. This not only verifies the construction and correctness of the graph but also showcases the practical application value of graph analysis.

References

[1] Neo4j, Inc. *Neo4j Documentation*. Available online at: https://neo4j.com/docs/, accessed June 2023.

[2] DZone. *Graph DB Updates on their growing popularity*. Available online at: https://www.dataversity.net/graph-databases-updates-on-their-growing-popularity/, accessed June 2023.

7 Appendix 1: tool selection matrices

Table 1: Tool Selection Matrix

Tool	Selected For	Justification
Neo4j	Graph Database Management	Neo4j performs well in graph handling, contains intuitive querying language (Cypher), querying efficiently, is good at handling complex relations
Python (pandas)	Data Extraction and Preprocessing	Easily implement, rich library supports
Cypher	Query Engine	Cooperate smoothly with Neo4j, is good at querying graph database
Google Colab	Development and Execution Environment	Cloud-based, avoid figuring the environment, is good at cooperation

Table 2: Comparison of different database

Aspect	Document-Oriented Databases	Key-Value Databases	Neo4J Graph Database
Relationship Management	Difficult to manage complex relationships such as listings, prices, calendar data, and locations.	Minimal to no relationship management capabilities.	Highly efficient at managing complex, dynamic relationships.
Complex Queries	Requires complex aggregation pipelines or multiple lookup operations, increasing development and maintenance complexity.	Limited query capabilities, often requiring multiple operations for complex queries.	Advanced query capabilities with simple graph queries, facilitating the exploration of intricate relationships.
Performance	Performs poorly with complex relationships and graph data, potentially affecting the timeliness and accuracy of data analysis.	High performance with simple data but poor with complex relationships.	Excels in managing graph data and complex relationships, ensuring timely and accurate data analysis.
Data Model Flexibility	Offers some flexibility mainly in document structure, but not in relationship management.	Very simple data model with almost no flexibility for relationships.	Schema-less nature ideal for handling complex, dynamic relationships.
Visualization Support	Lacks built-in visualization tools, making it difficult to intuitively understand and analyze complex data.	Lacks visualization support, requiring additional tools for data visualization.	Provides powerful graphical tools to intuitively represent data relationships, aiding in understanding and analysis.