Xi Lin

Network Science (6520) – Summer 2021

Abstract

To propose a method for using graph to analytics to identify and target the best reviewed listing of Cambridge city in term of price group and neighborhood. Providing visitors with the most effective information to help select the best quality short time rental properties at the psychological price point and tourist location

**USE GRAgh ANALYTICS TO INDENTIFY THE BEST REVIEWED LISTINGS OF cAMBRIDGE, mA USA IN TERM OF PRICE GROUP AND NEIGBORHOOD.**

Network Science 6520

# Primary Objective:

Airbnb, Inc. is an American company founded in 2008 that operates an online marketplace for lodging (primarily homestays for vacation rentals) and travel activities. The platform is based in San Francisco, California, and is accessible through its website and mobile app. Airbnb does not own any of the properties listed; instead, it profits by collecting a commission from each booking. The emergence of Airbnb has contributed to a boom in tourism and improved the quality of travel.

This proposal is to apply graph query to identify and target below:

# The best reviewed listings of Cambridge city in term of price group and neighborhood. Usually, the visitors or travelers are looking for the best available rental properties either by price range or location/neighborhood. This can be done via query by filters and orders, however, exploring graph could end up with more intuitive views and single hops.

# 1.Given a location (Neighborhood), show the listings in the best review score group, so I can consider them on higher priority.

# 2. Given a price (Price Group), show the listings within worst review score group, so I can avoid to book them.

# Dataset & Business Use Case:

Data was acquired directly from [Airbnb’s database](http://insideairbnb.com/get-the-data.html) for educational research purposes. The dataset we use in here is the listings and reviews data that is published by Airbnb per period per region. In US, the region usually is a city. Based upon the data volume requirement, we deliberately choose a town that has less data but good enough for demo purpose – we pick the listings and reviews data from Cambridge, MA USA as of March, 2021.

A general overview of the data set as follows (a dictionary is included in the Appendix for deeper review):

* **Listings / Listing Detail:** Listing data for Cambridge, MA in terms of link, rental type, number of bedroom and restroom, accommodation, neighborhood, review counts and average ratings, default price, so on so force.
* **Calendar:** Despite of its misleading name, this data provides the daily price (could vary) info upon each listing (a short window starting from March 2021).
* **Reviews / Review Detail**: Provides the available historically review info upon each listing, it has listing id, date, reviewer, and comments. Surprisingly no rate score – this is for privacy purpose.
* **Neighborhood**: this is a reference data set for neighborhood.

Overall, this is public data that doesn’t have a lot of financial or market data – hence we don’t see the actual rent-out records along with the past price information. Instead, we can see the listing information, and the accumulated review information.

For our project, we don’t need all of them – the core dataset is the Listings / Listing Detail and Review/Review Detail. We don’t even need all the attributes - The columns we need in Listings are listing id, host id, price, neighborhood, number of reviews, and review scores rating. In Reviews we need review id, reviewer.

### Graph Projections

# 

# Final Graph Data Model

# 

# Data preparation and cleaning：

# Download data from Airbnb data set: <http://insideairbnb.com/get-the-data.html>

# There are some data issues about listings, if they are under deco or for a period not available on market, the price is missing. But we do not want to filter them out since they still have valid neighborhood information. I created a Group 0 in Price Group node to accommodate such situation, and link those missing price listings to this group.

# There are some data issues about some listings with no review info – this is understood that if the listing never received any review which is allowed. I do not want to filter them out since they carry valid price information, so I create Group 0 in Review Group node to accommodate such situation and link those missing review listings to this group.

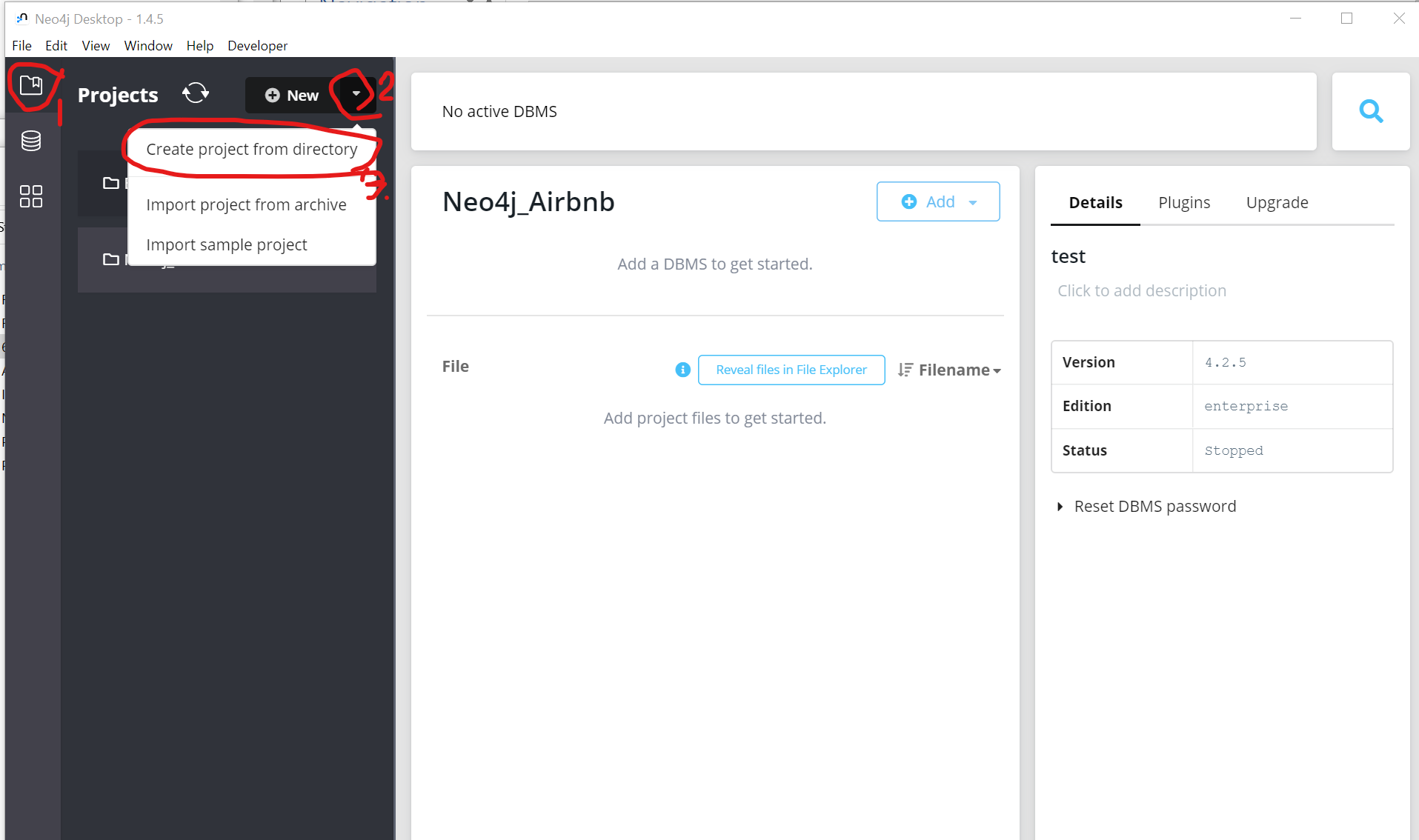
# For the region in Cambridge MA, there is no listing that has missing neighborhood, but this could happen in other region if we apply this model to analyze other cities. Hence to be more flexible and accommodating, we created a node in Neighborhood called Not Set.

# Why use group 0 for missing price or missing review? Because the group number is not only an ID, but also in its property “Seq” which is used to determine the max price or min review score etc., and Group 0 would throw them into the right place in sequence.

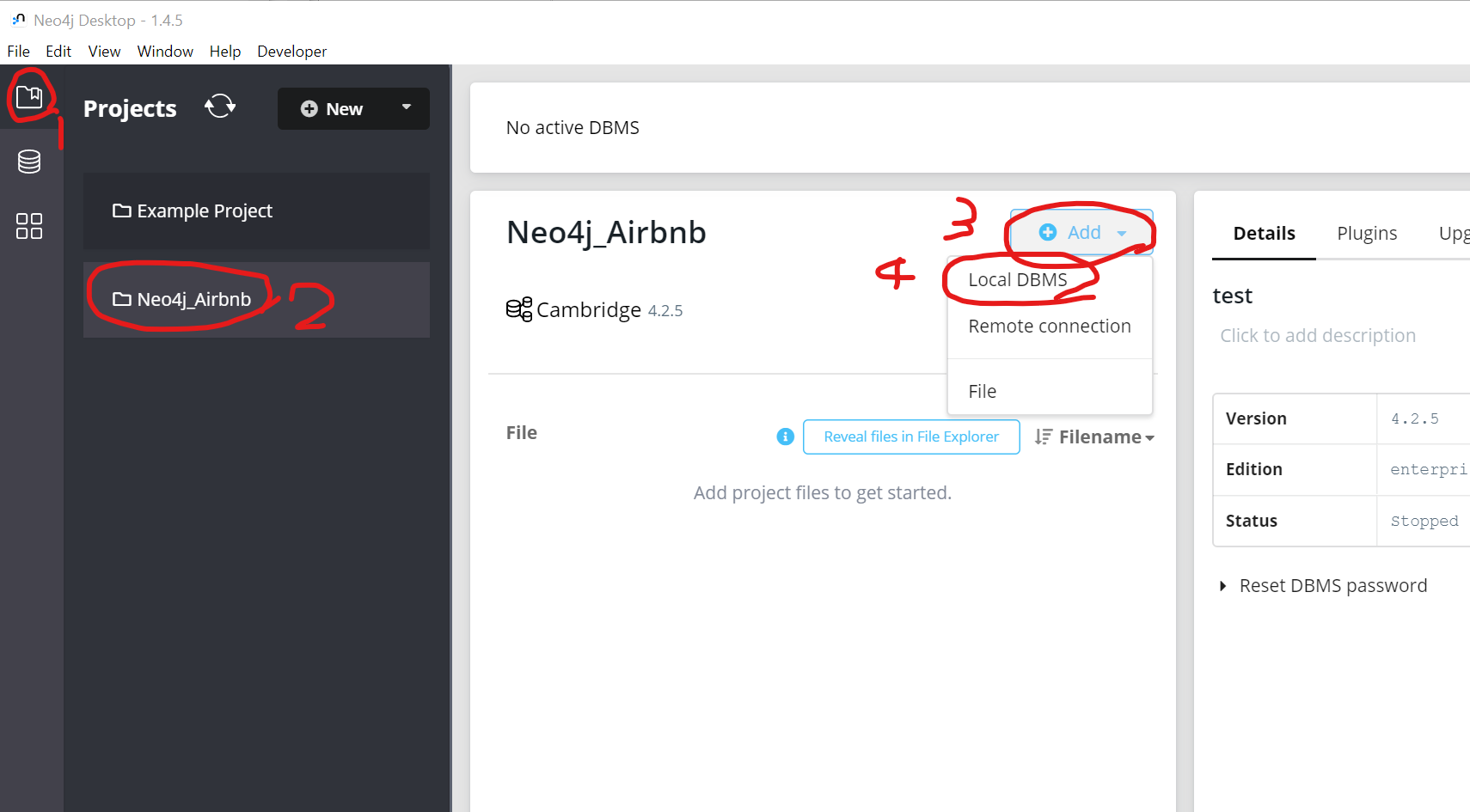
# Installation, configure, and connect

(1) Install and start Neo4j desktop

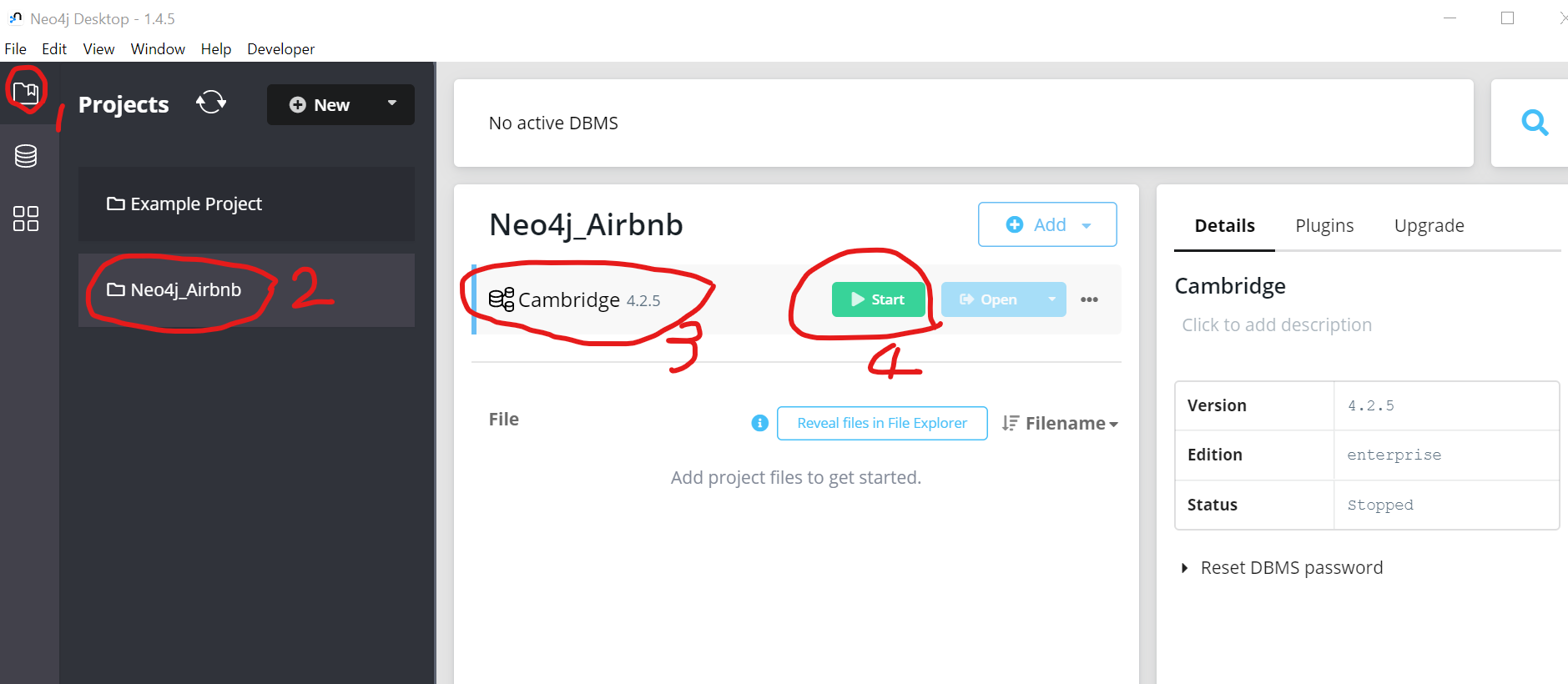
(2) Create a folder “Neo4j\_Airbnb” as a new project

****

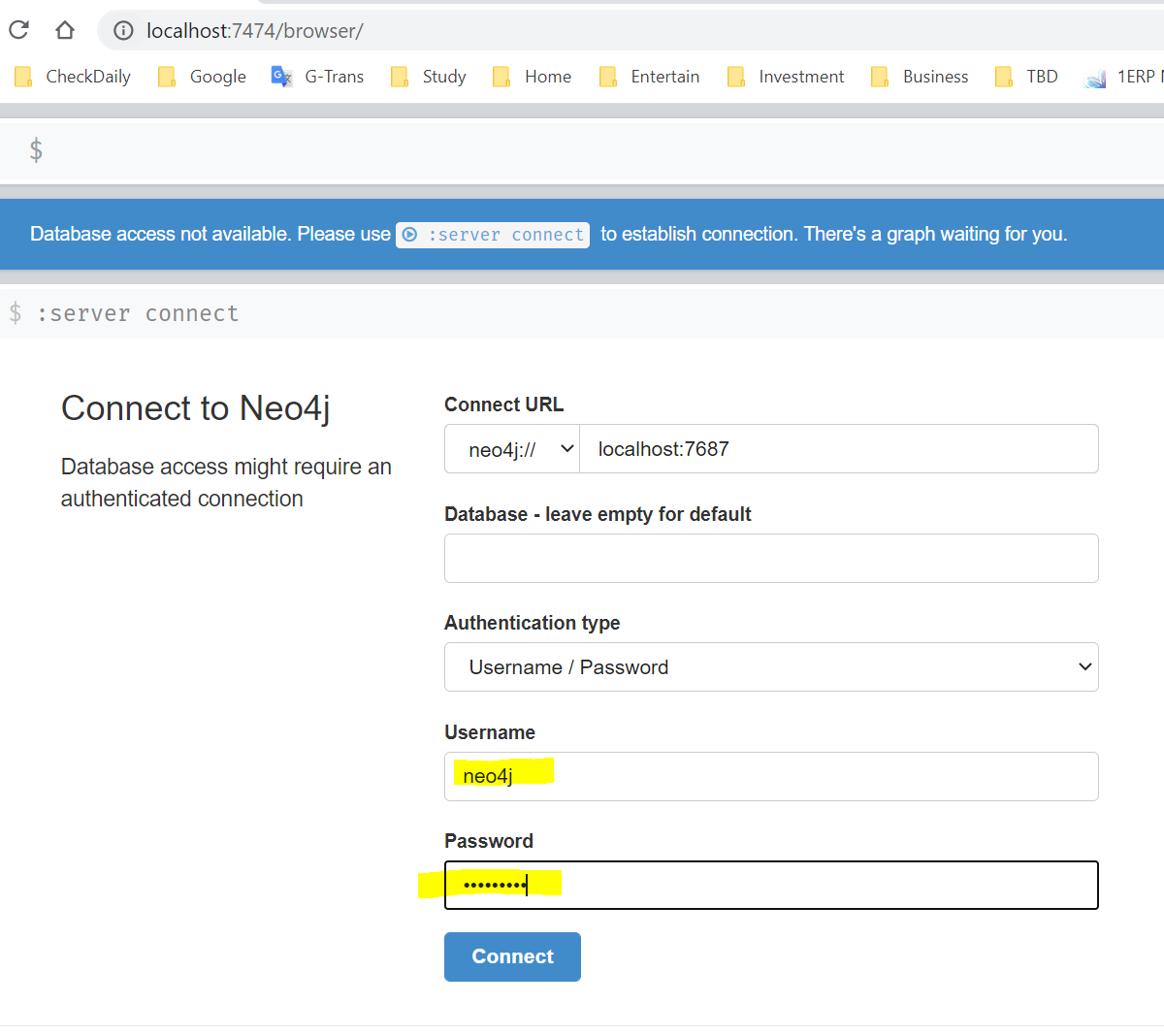
(3) Create a database “Cambridge” under project Neo4j\_Airbnb (password “Cambridge”)



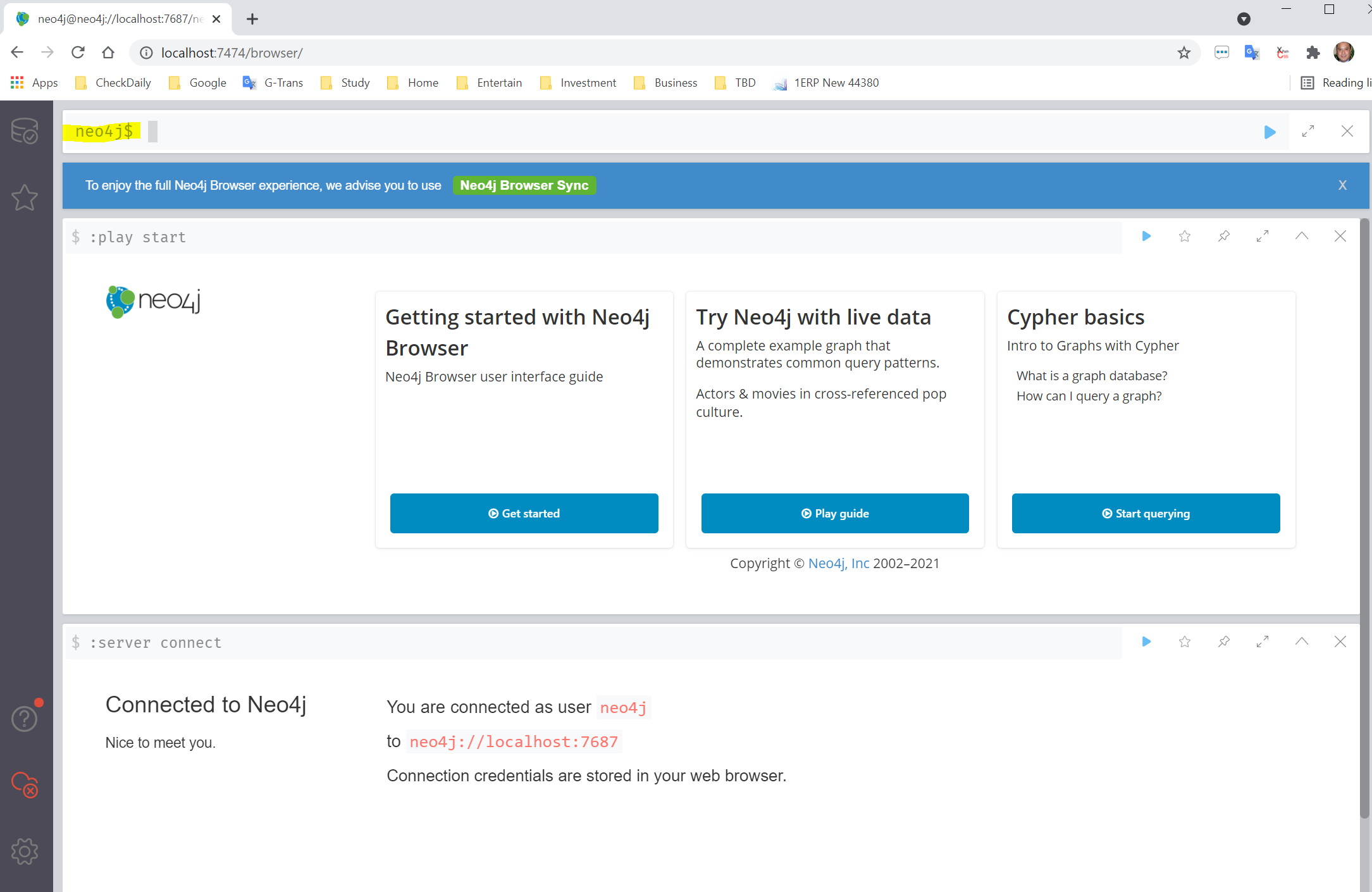
(4) Start the database “Cambridge”



(5) Start a browser, and enter <http://localhost:7474/browser/>, use “neo4j” as user, “Cambridge” as password to login.



(6) Now you can get this interface in a browser, where you can enter command to manipulate the database.



# Load Data (Method 1,slow)

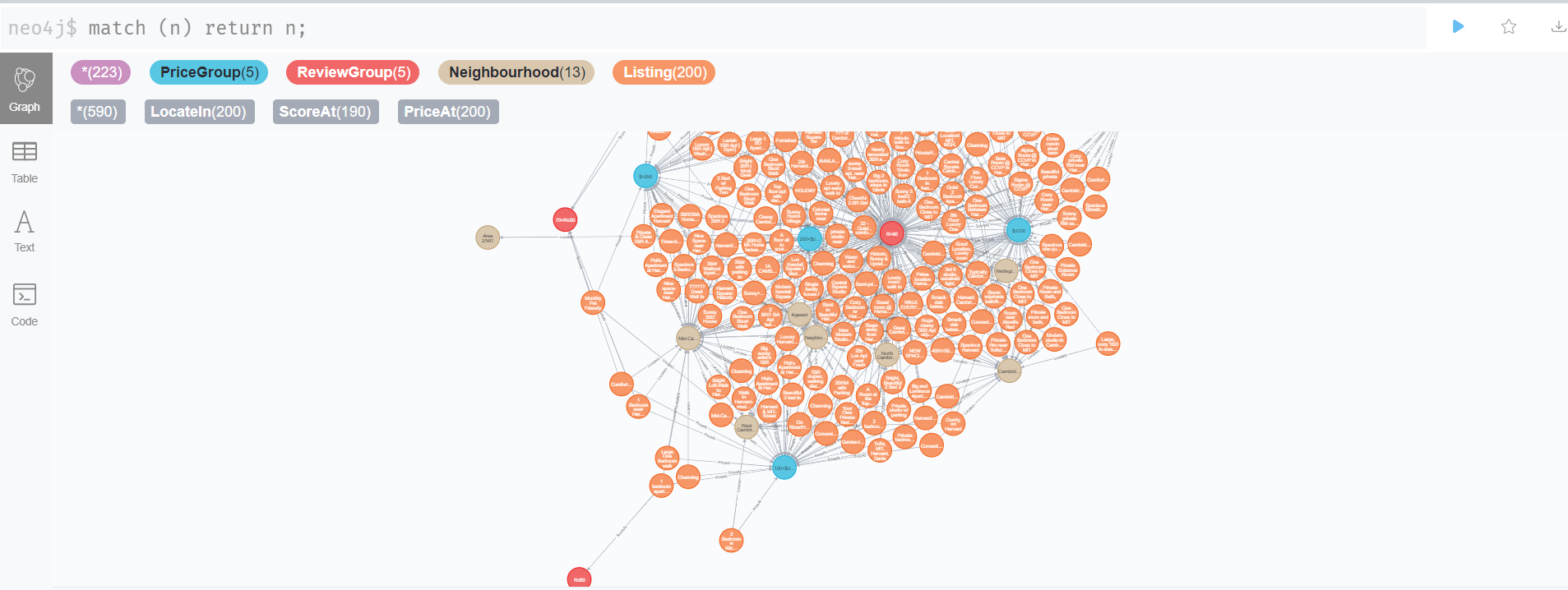
## (1) Full data set: Neo4j.Create.Fullset.txt

Total there are 817 listings, 5 price group, 5 review score group, and 13 neighborhood, and 2500 edges.

During loading process, I have spent like 4 hours to build above nodes and edges, but still cannot finish, so I decide to use a subset of the data.

## (2) Subset: Neo4j.Create.Subset.txt

I choose 200 out of the 817 listings, which end up 600 edges, and I can load them within 1.5 hours.



# Load Data (Method 2, recommended)

**Data Load (Nodes)**

(1) Review Group

CREATE CONSTRAINT pk\_ReviewGroup ON (r:ReviewGroup) ASSERT r.seq IS UNIQUE;

LOAD CSV WITH HEADERS FROM 'file:///ReviewGroup.csv' AS row

  WITH toInteger(row.seq) as v\_seq, row.description as v\_description

  MERGE(S:ReviewGroup {seq: v\_seq}) SET S.description=v\_description RETURN COUNT(S);

# Query: MATCH(r:ReviewGroup) RETURN r;

# Delete: MATCH(r:ReviewGroup) DETACH DELETE r;

(2) Price Group

CREATE CONSTRAINT pk\_PriceGroup ON (p:PriceGroup) ASSERT p.seq IS UNIQUE;

LOAD CSV WITH HEADERS FROM 'file:///PriceGroup.csv' AS row

  WITH toInteger(row.seq) as v\_seq, row.description as v\_description

  MERGE(P:PriceGroup {seq: v\_seq}) SET P.description=v\_description RETURN COUNT(P);

# Query: MATCH(p:PriceGroup) RETURN p;

# Delete: MATCH (n:PriceGroup) DETACH DELETE n;

(3) Neighbourhood

CREATE CONSTRAINT pk\_Neighbourhood ON (n:Neighbourhood) ASSERT n.seq IS UNIQUE;

LOAD CSV WITH HEADERS FROM 'file:///Neighbourhood.csv' AS row

  WITH toInteger(row.seq) as v\_seq, row.name as v\_name

  MERGE(N: Neighbourhood {seq: v\_seq}) SET N.name=v\_name RETURN count(N);

# Query: MATCH(n:Neighbourhood) RETURN n;

# Delete: MATCH(n:Neighbourhood) DETACH DELETE n;

(4) Listing

LOAD CSV WITH HEADERS FROM 'file:///Listing.csv' AS row

WITH toInteger(row.id) as ListingID, toInteger(row.host\_id) as HostID, row.name as ListingName

MERGE(L:Listing {id: ListingID}) SET L.name=ListingName, L.host\_id=HostID

RETURN COUNT(L);

**Data Load (Edges)**

**(5) Edges of PRICE\_AT between Listings and Price Groups**

(5) Edges of PRICE\_AT between Listings and Price Groups

LOAD CSV WITH HEADERS FROM 'file:///Listing.csv' AS row

WITH toInteger(row.id) as ListingID, toInteger(row.PriceSeq) as nPriceSeq, toInteger(row.Price) as nPrice

MATCH (l:Listing),(p:PriceGroup) WHERE l.id=ListingID AND p.seq=nPriceSeq

CREATE (l)-[r:PRICE\_AT {weight: nPrice}]->(p) RETURN COUNT(r);

# match (l:Listing)-[r:PRICE\_AT]->(p:PriceGroup) delete r;

## (6) Edges of SCORE\_AT between Listings and Review Groups

LOAD CSV WITH HEADERS FROM 'file:///Listing.csv' AS row

WITH toInteger(row.id) as ListingID, toInteger(row.ReviewSeq) as nReviewSeq, toInteger(row.ReviewScore) as Score

MATCH (l:Listing),(s:ReviewGroup) WHERE l.id=ListingID AND s.seq=nReviewSeq

CREATE (l)-[r:SCORE\_AT {weight: Score}]->(s) RETURN COUNT(r);

# match (l:Listing)-[r:SCORE\_AT]->(s:ReviewGroup) delete r;

## (7) Edges of LOCATED\_IN between Listings and Neighbourhood

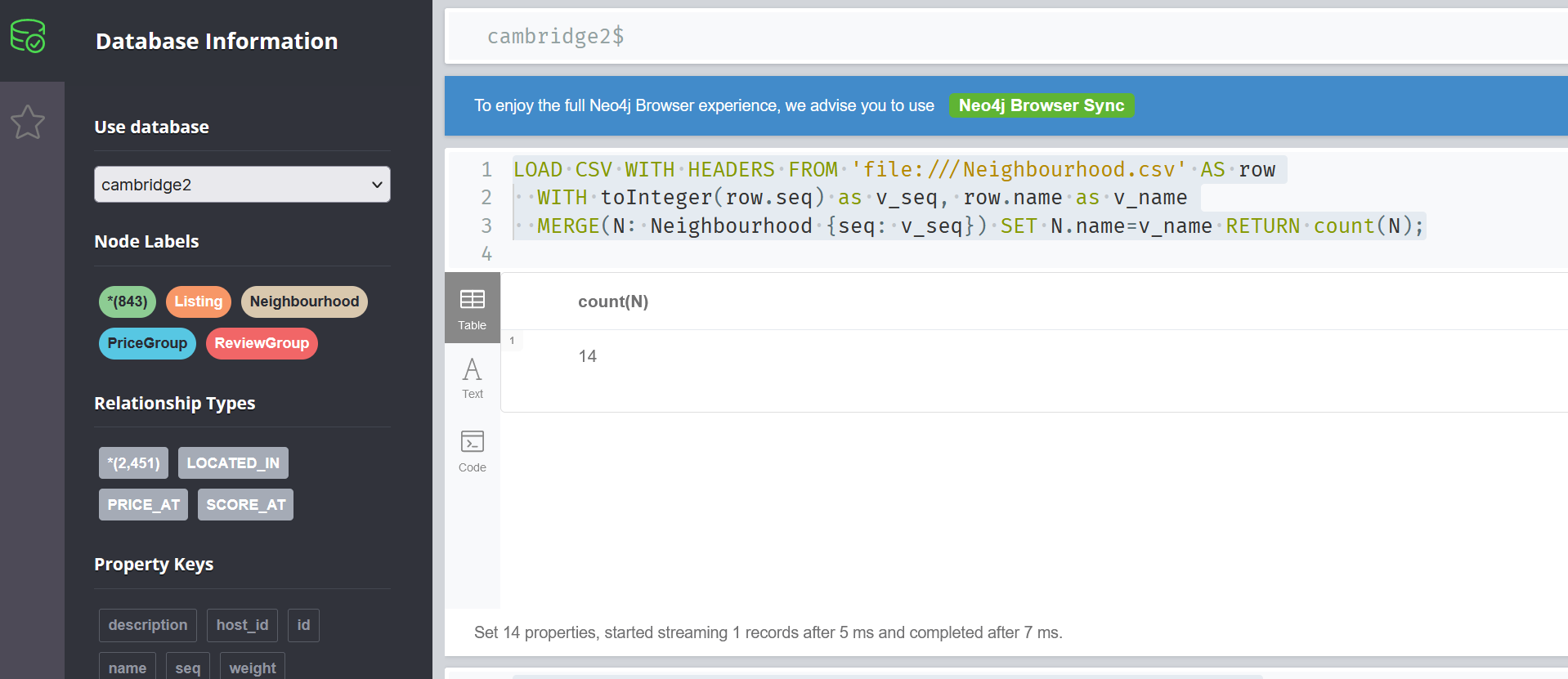
LOAD CSV WITH HEADERS FROM 'file:///Listing.csv' AS row

WITH toInteger(row.id) as ListingID, toInteger(row.NBSeq) as nNBSeq

MATCH (l:Listing),(n:Neighbourhood) WHERE l.id=ListingID AND n.seq=nNBSeq

CREATE (l)-[r:LOCATED\_IN]->(n) RETURN COUNT(r);

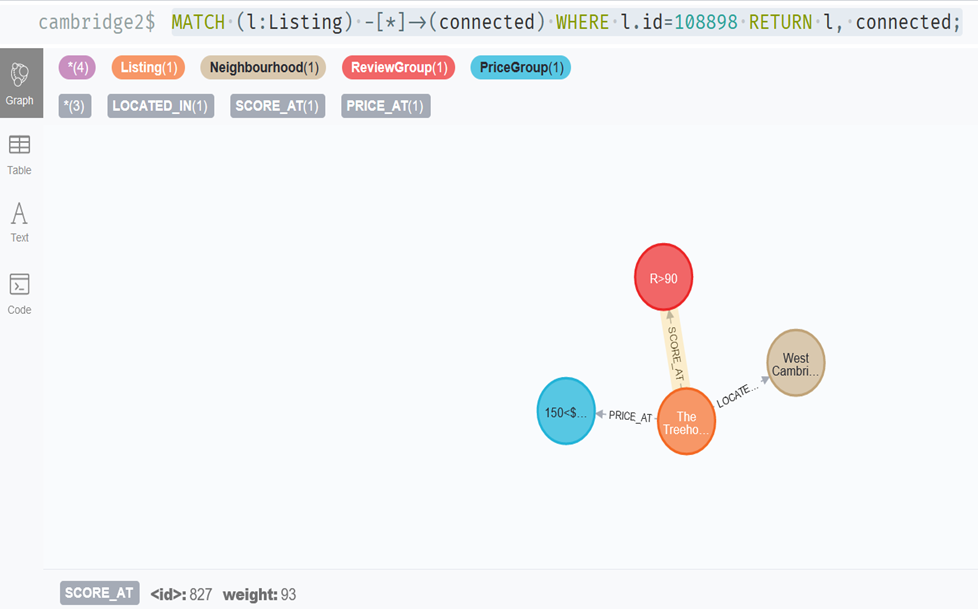
# match (l:Listing)-[r:LOCATED\_IN]->(n:Neighbourhood) delete r;



**Data Verification**

## Verification: given one listing, show its raw data relationships

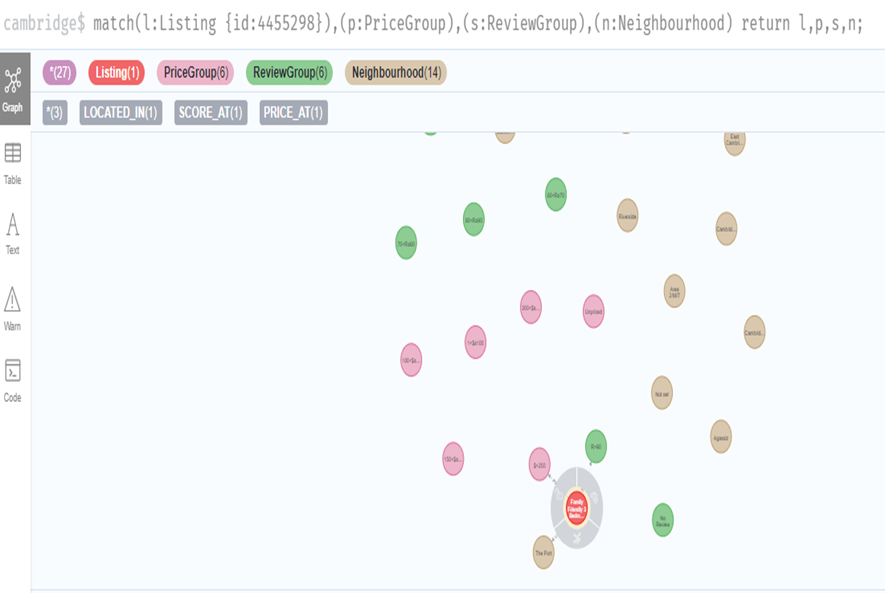
MATCH (l:Listing) -[\*]->(connected) WHERE l.id=108898 RETURN l, connected;

****

# Cypher Query

## (1) Query a listing with its edges to price group, review group, and neighborhood

match(l:Listing {id:4455298}),(p:PriceGroup),(s:ReviewGroup),(n:Neighbourhood) return l,p,s,n;



## (2) Query how many review groups are corresponding to a price group

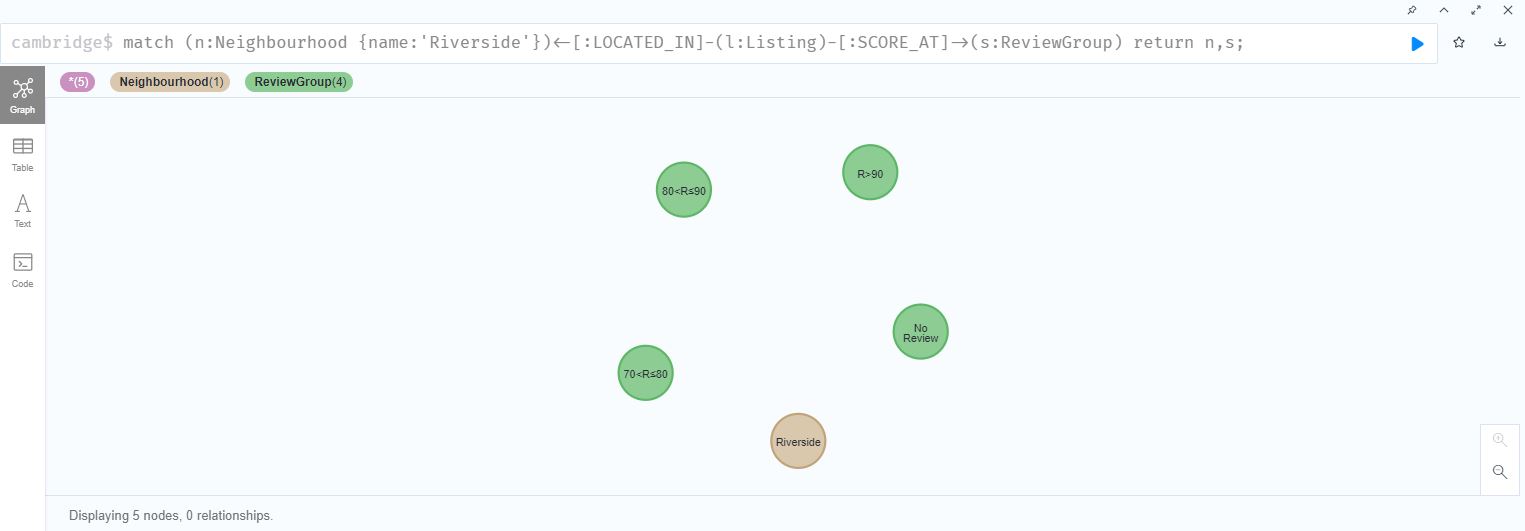
match (p:PriceGroup {seq:5})<-[:PRICE\_AT]-(l:Listing)-[:SCORE\_AT]->(s:ReviewGroup) return p,s; 



## (3) How many review groups are corresponding to a neighbourhood

## match (n:Neighbourhood {name:'Riverside'})<-[:LOCATED\_IN]-(l:Listing)-[:SCORE\_AT]->(s:ReviewGroup) return n,s;

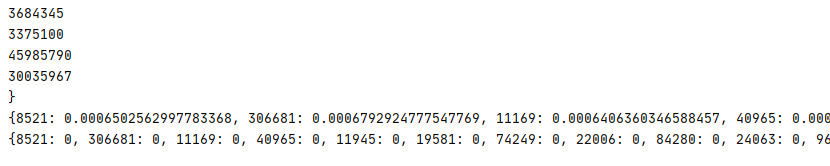
## 

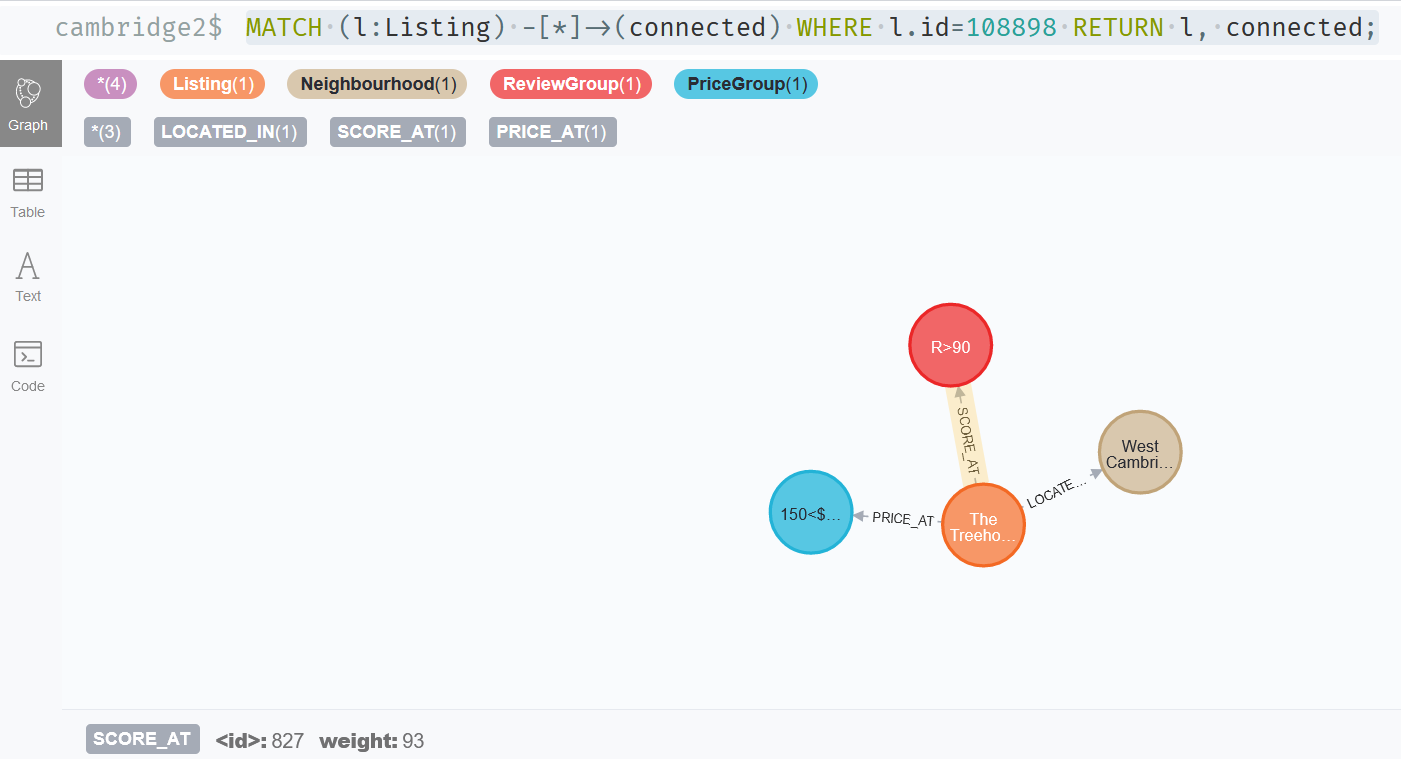


# Algorithms

# we apply three graph algorithms to the graph model we construct; we use network to implement the graph algorithms, follow is the code and result







# Cypher Actions

## (1) Given a location (Neighbourhood), show me the listings in the best review score group, so I can consider them on higher priority.

### Method one: Pure Cypher with multiple hops

with 'Riverside' as sNB

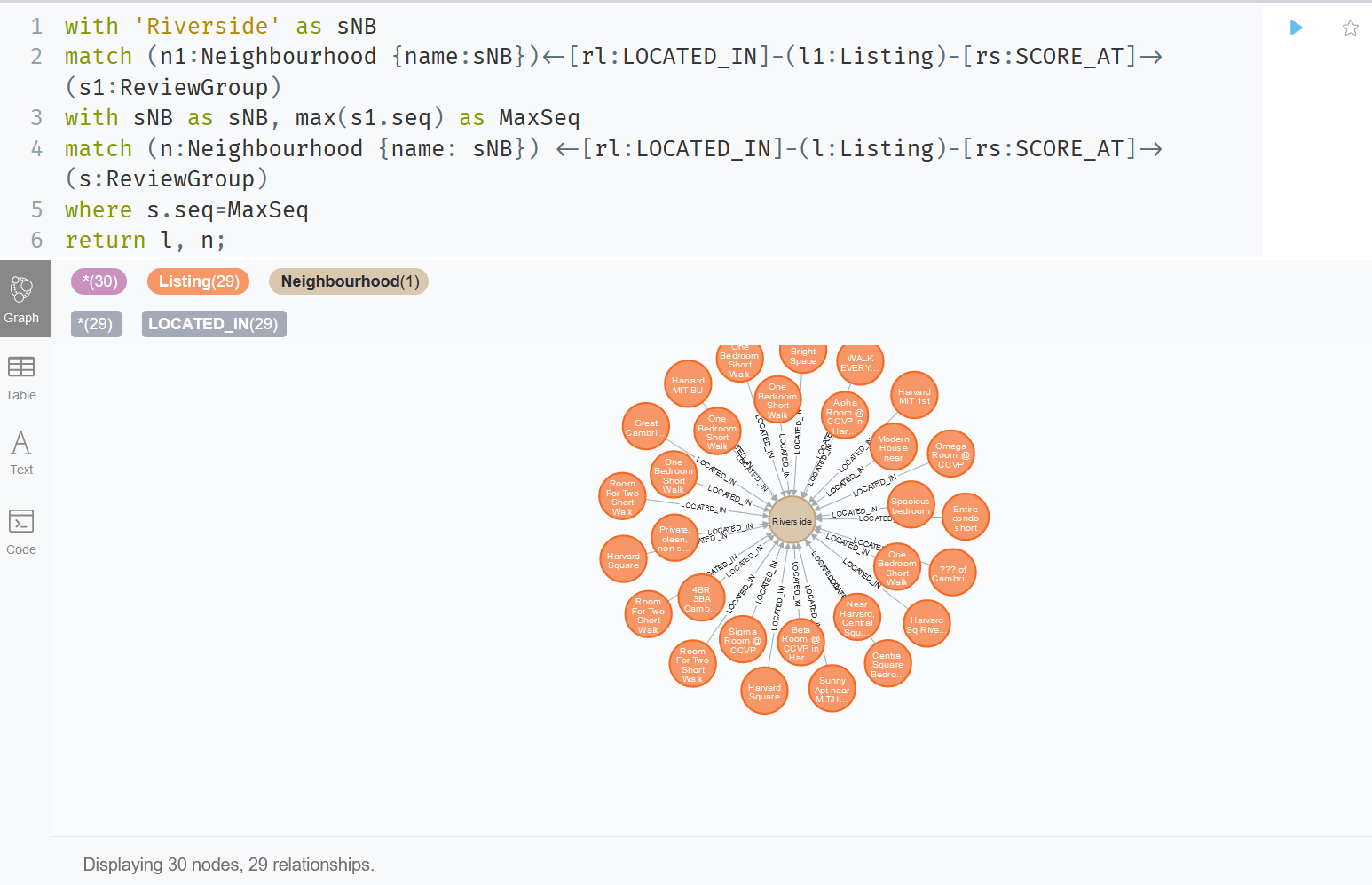
match (n1:Neighbourhood {name:sNB})<-[rl:LOCATED\_IN]-(l1:Listing)-[rs:SCORE\_AT]->(s1:ReviewGroup)

with sNB as sNB, max(s1.seq) as MaxSeq

match (n:Neighbourhood {name: sNB}) <-[rl:LOCATED\_IN]-(l:Listing)-[rs:SCORE\_AT]->(s:ReviewGroup)

where s.seq=MaxSeq

return  l, n;



Second match to find out all the listings linked with the neighbourhood and the MAX Review Group

First match to find out MAX Review Score Group which is the “Best Score”

Neighbourhood name is used in multiple places, use a variable for input once

### Method two: Use Neighbourhood.csv as a loop control to project a new set of relationship “BEST\_SCORED” from Neighbourhood back to Listings, then query with one hop.

LOAD CSV WITH HEADERS FROM 'file:///Neighbourhood.csv' AS row

  WITH toInteger(row.seq) as nb\_seq

  match (n1:Neighbourhood {seq:nb\_seq})<-[rl:LOCATED\_IN]-(l1:Listing)-[rs:SCORE\_AT]->(s1:ReviewGroup)

  with n1.seq as nb\_seq, max(s1.seq) as MaxScoreSeq

  match (n:Neighbourhood {seq: nb\_seq}) <-[rl:LOCATED\_IN]-(l:Listing)-[rs:SCORE\_AT]->(s:ReviewGroup)

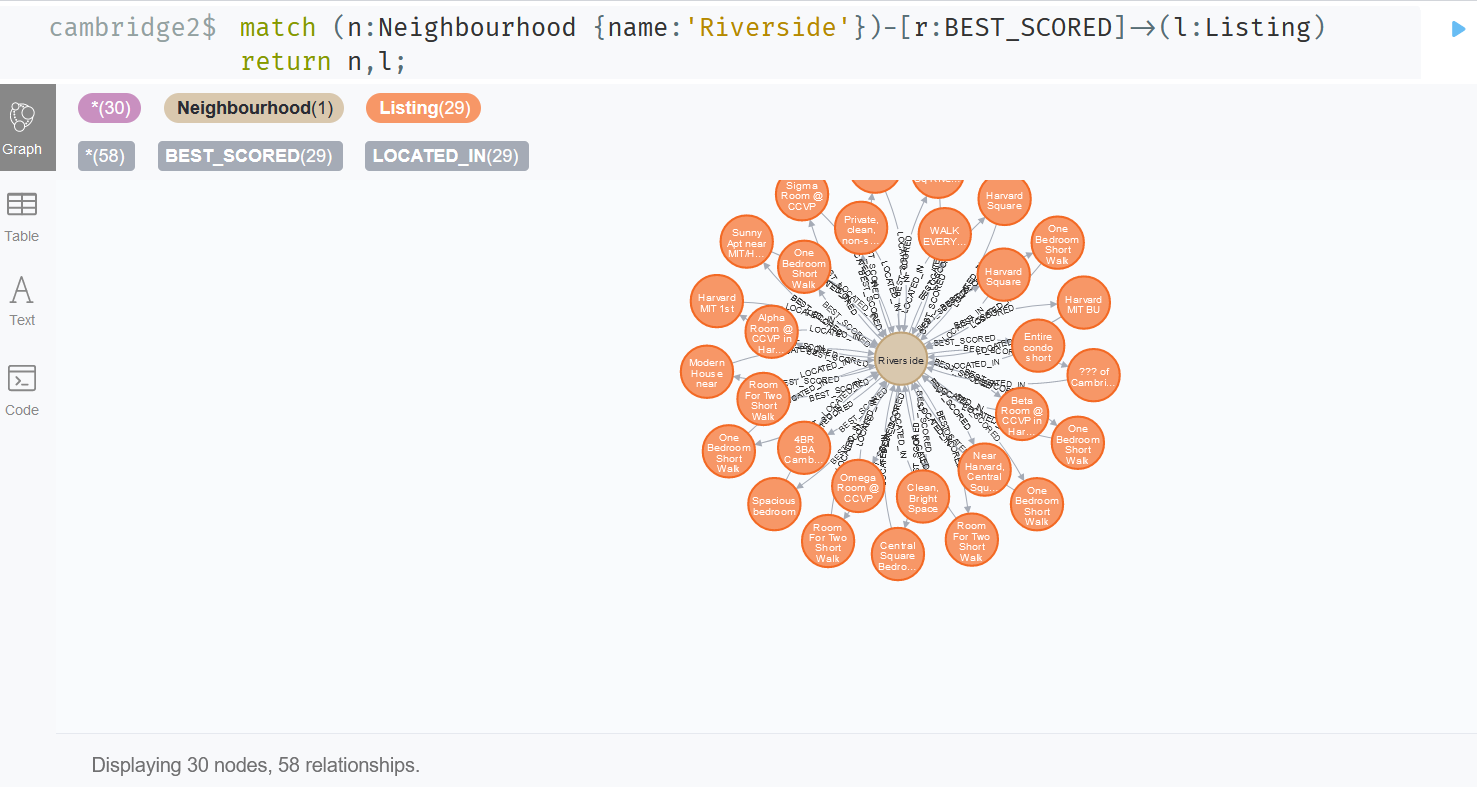
  where s.seq=MaxScoreSeq

  CREATE (n)-[r:BEST\_SCORED {weight: rs.weight}]->(l)

  RETURN COUNT(r);

match (n:Neighbourhood {name:'Riverside'})-[r:BEST\_SCORED]->(l:Listing) return n,l;





## (2) Given a price (Price Group), show me the listings within worst review score group, so I can avoid to book them.

with '1<$≤100' as sPG

MATCH (p:PriceGroup {description: sPG}) <-[rl:PRICE\_AT]-(l1:Listing)-[rs:SCORE\_AT]->(s1:ReviewGroup)

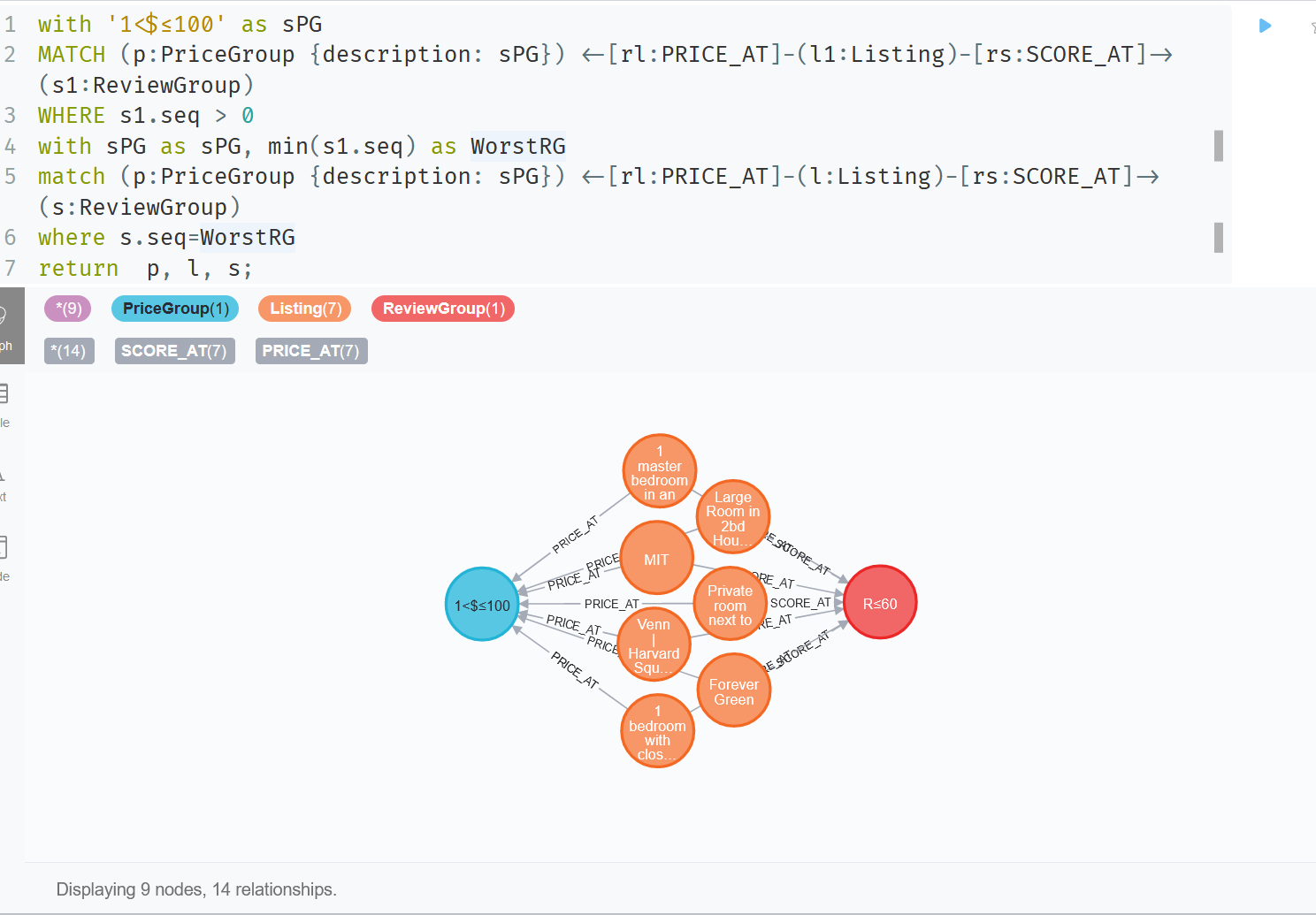
WHERE s1.seq > 0

with sPG as sPG, min(s1.seq) as WorstRG

match (p:PriceGroup {description: sPG}) <-[rl:PRICE\_AT]-(l:Listing)-[rs:SCORE\_AT]->(s:ReviewGroup)

where s.seq=WorstRG

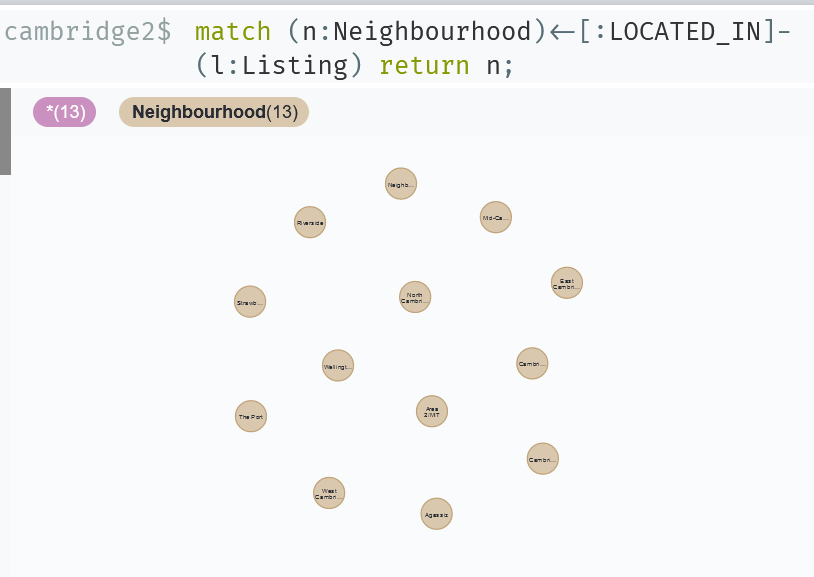
return  p, l, s;



We can adopt the same as previous question second solution to project a set of relationships to save hops, ignore it in here.

# Visualization

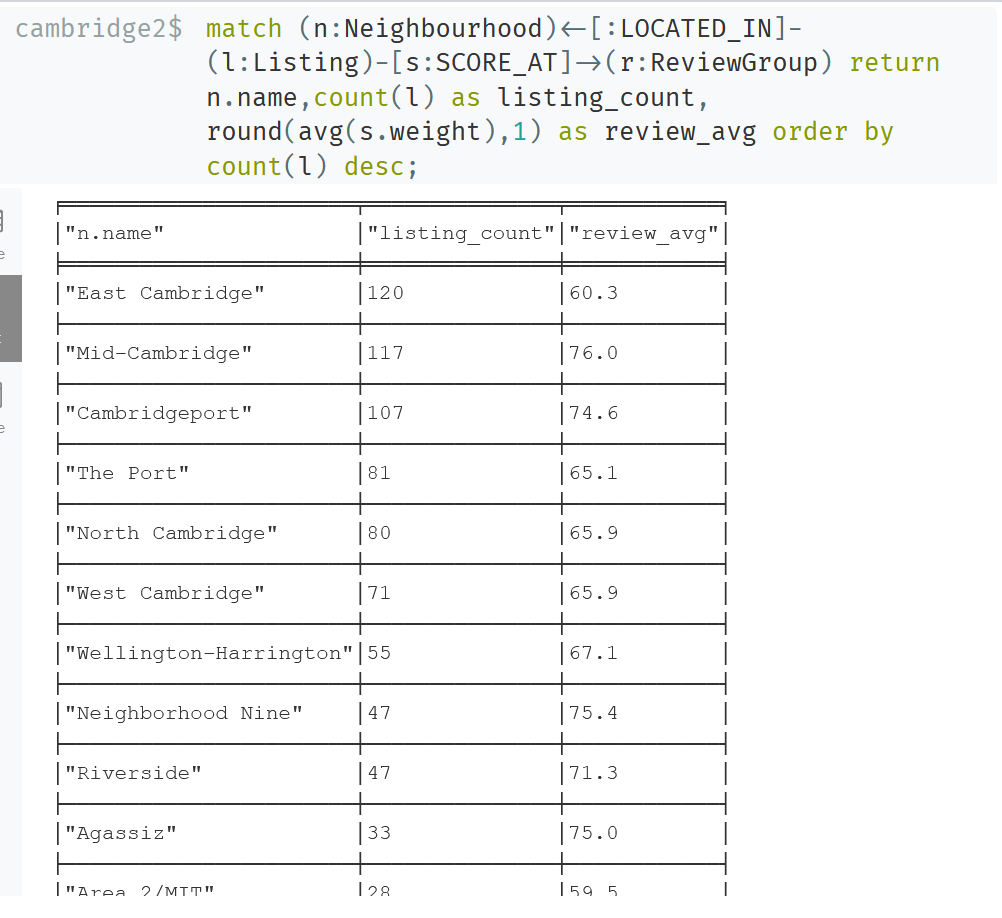
Totally there are 14 geographically spread neighborhood in Cambridge, MA for Airbnb to detail divides the city level regional market. Below shows the neighborhood along with their listing count.



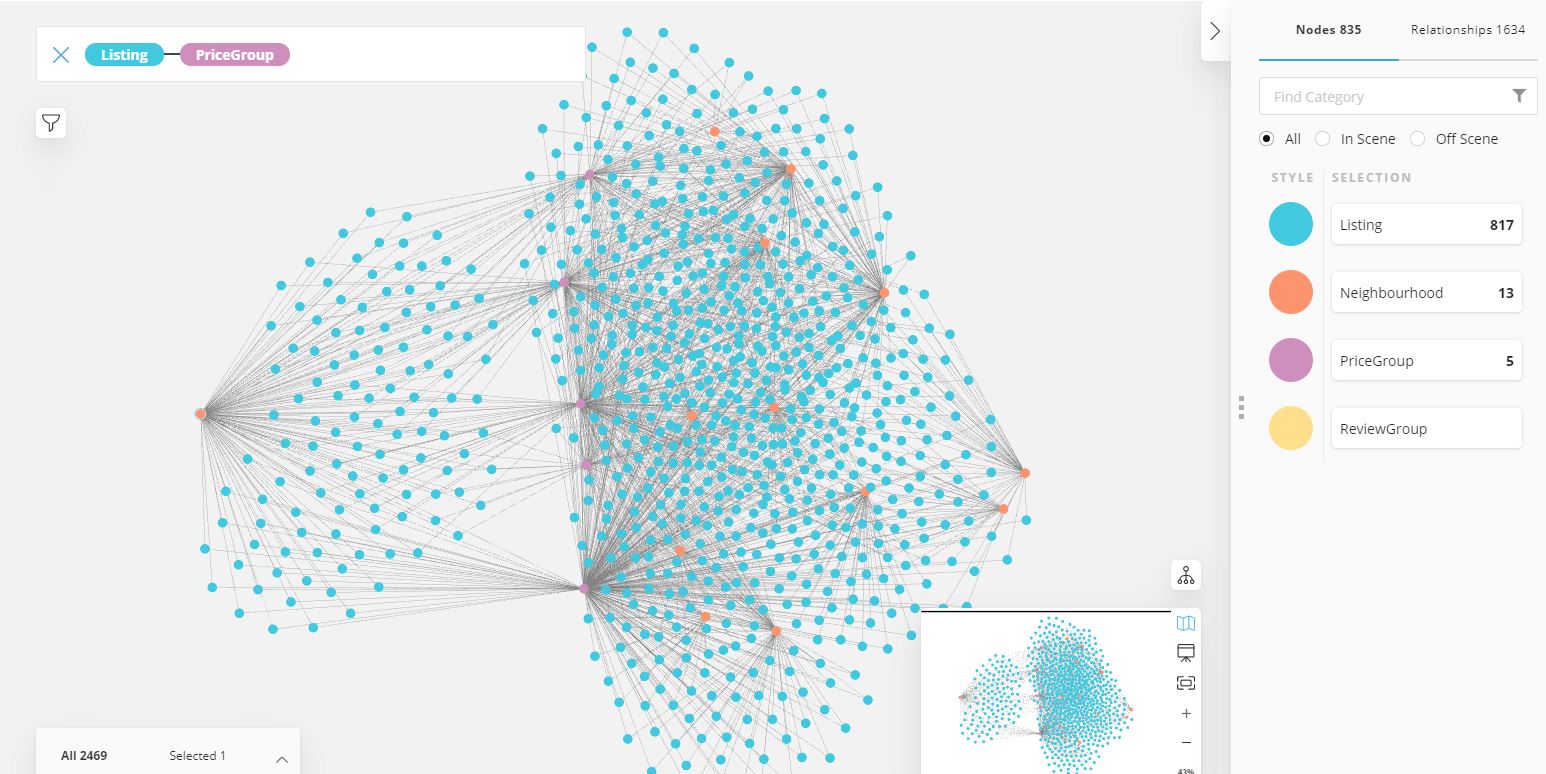
Associated with the neighborhood are the listing count and review score average

# Visualization1

match(n:Neighbourhood)<-[:LOCATED\_IN]-(l:Listing)-[s:SCORE\_AT]->(r:ReviewGroup)return n.name,count(l)as listing\_count,round(avg(s.weight),1)as review\_avg order by count(l) desc;

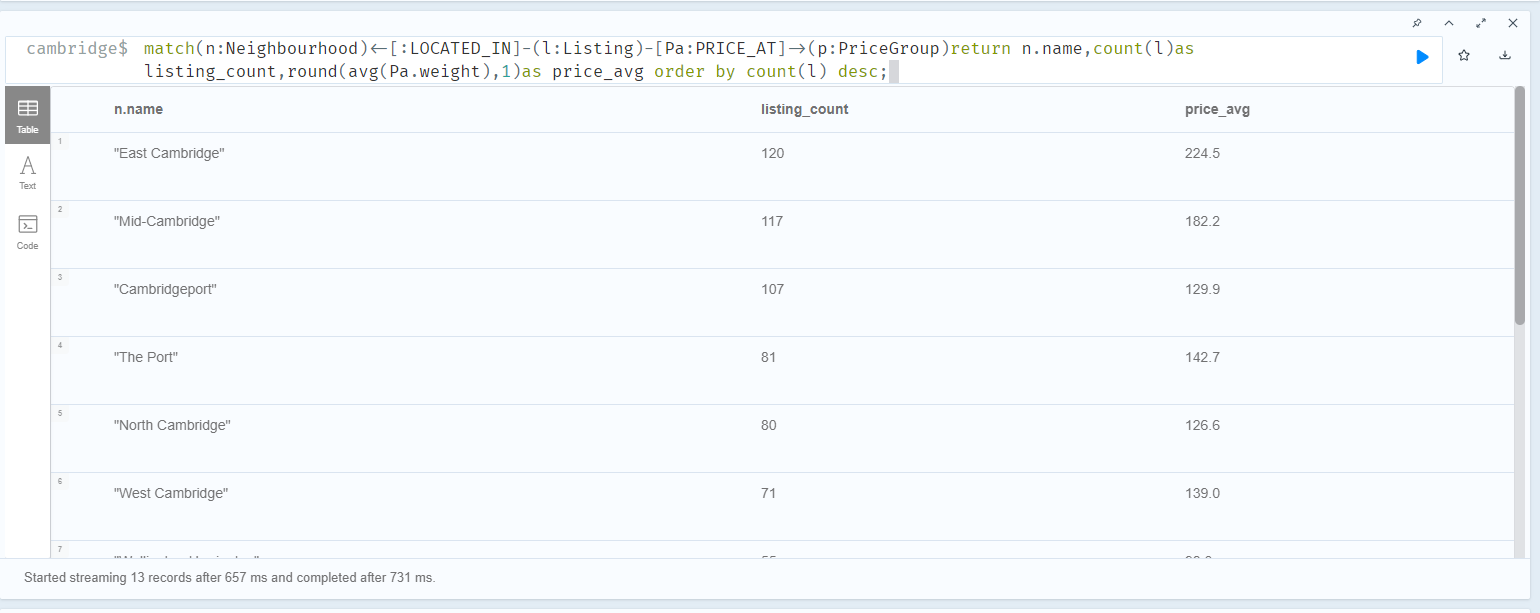
<

The orange node on the left is East Cambridge neighborhood.

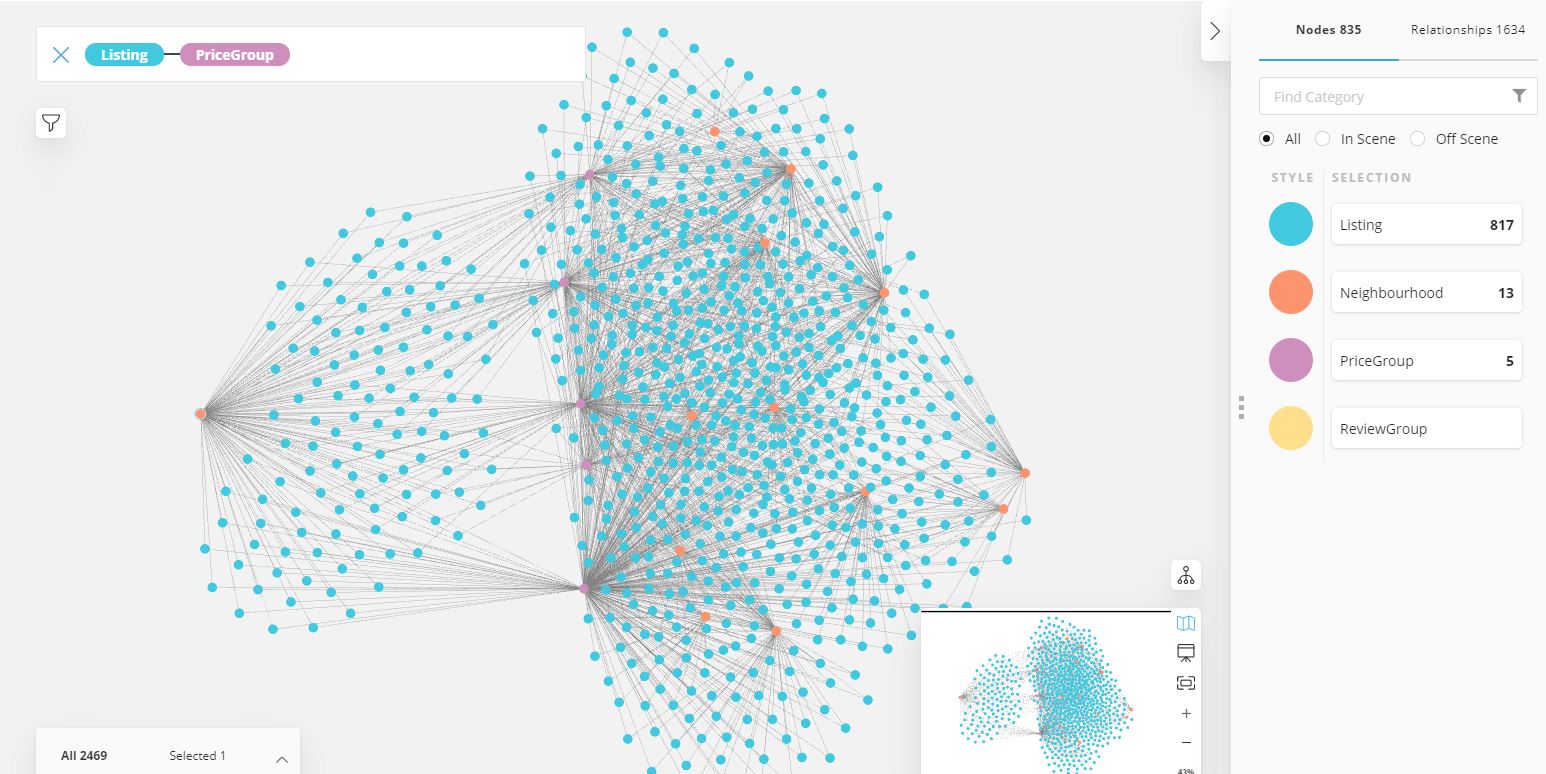


# Visualization 2

match(n:Neighbourhood)<-[:LOCATED\_IN]-(l:Listing)-[Pa:PRICE\_AT]->(p:PriceGroup)return n.name,count(l)as listing\_count,round(avg(Pa.weight),1)as review\_avg order by count(l) desc;



The orange node on the left is East Cambridge neighborhood.



# Analysis

Among the top three large neighborhood in terms of listing numbers, the East Cambridge is carrying the almost lowest review score. From business point of view, this is the one needs more attention, actions should be taken:

* Investigating the reason for low satisfaction;
* Giving more training to the owners in this area;
* Tightening the acceptance criteria of new listings;

Follow-up analysis could be performed with future 6-month data compare with this result to verify if the actions are effective or not.

# Conclusion

The customer could use the graphs to find best review score group in a neighborhood (ie. Riverside) as higher priority. In our demo Cypher Action1, there are 29 listings ,the customer should consider them first.

The customer could use the graphs to find the worst score group (ie price less than 100) as avoiding book listing. In our demo Cypher Action 2, there are 7 listing, the customer should avoid them.

# References

Mark Needham & Amy Hodler (2019)

Graph Algorithms

Michele Castr（2021）

The Atlas for the Aspiring Network Scientist

# Appendix

## Data Dictionary

|  |  |  |
| --- | --- | --- |
| **Listing\_Detail Table** | | |
| **Field** | **Type** | **Description** |
| Id | integer | Airbnb's unique identifier for the listing |
| listing\_url | text |  |
| scrape\_id | bigint | Inside Airbnb "Scrape" this was part of |
| last\_scraped | datetime | UTC. The date and time this listing was "scraped". |
| name | text | Name of the listing |
| description | text | Detailed description of the listing |
| neighborhood\_overview | text | Host's description of the neighbourhood |
| picture\_url | text | URL to the Airbnb hosted regular sized image for the listing |
| host\_id | integer | Airbnb's unique identifier for the host/user |
| host\_url | text | The Airbnb page for the host |
| host\_name | text | Name of the host. Usually just the first name(s). |
| host\_since | date | The date the host/user was created. For hosts that are Airbnb guests this could be the date they registered as a guest. |
| host\_location | text | The host's self reported location |
| host\_about | text | Description about the host |
| host\_response\_time |  |  |
| host\_response\_rate |  |  |
| host\_acceptance\_rate |  | That rate at which a host accepts booking requests. |
| host\_is\_superhost | boolean [t=true; f=false] |  |
| host\_thumbnail\_url | text |  |
| host\_picture\_url | text |  |
| host\_neighbourhood | text |  |
| host\_listings\_count | text | The number of listings the host has (per Airbnb calculations) |
| host\_total\_listings\_count | text | The number of listings the host has (per Airbnb calculations) |
| host\_verifications |  |  |
| host\_has\_profile\_pic | boolean [t=true; f=false] |  |
| host\_identity\_verified | boolean [t=true; f=false] |  |
| neighbourhood | text |  |
| neighbourhood\_cleansed | text | The neighbourhood as geocoded using the latitude and longitude against neighborhoods as defined by open or public digital shapefiles. |
| neighbourhood\_group\_cleansed | text | The neighbourhood group as geocoded using the latitude and longitude against neighborhoods as defined by open or public digital shapefiles. |
| latitude | numeric | Uses the World Geodetic System (WGS84) projection for latitude and longitude. |
| longitude | numeric | Uses the World Geodetic System (WGS84) projection for latitude and longitude. |
| property\_type | text | Self selected property type. Hotels and Bed and Breakfasts are described as such by their hosts in this field |
| room\_type | text | [Entire home/apt|Private room|Shared room|Hotel]  All homes are grouped into the following three room types:  Entire place Private room Shared room Entire place Entire places are best if you're seeking a home away from home. With an entire place, you'll have the whole space to yourself. This usually includes a bedroom, a bathroom, a kitchen, and a separate, dedicated entrance. Hosts should note in the description if they'll be on the property or not (ex: "Host occupies first floor of the home"), and provide further details on the listing.  Private rooms Private rooms are great for when you prefer a little privacy, and still value a local connection. When you book a private room, you'll have your own private room for sleeping and may share some spaces with others. You might need to walk through indoor spaces that another host or guest may occupy to get to your room.  Shared rooms Shared rooms are for when you don't mind sharing a space with others. When you book a shared room, you'll be sleeping in a space that is shared with others and share the entire space with other people. Shared rooms are popular among flexible travelers looking for new friends and budget-friendly stays. |
| accommodates | integer | The maximum capacity of the listing |
| bathrooms | numeric | The number of bathrooms in the listing |
| bathrooms\_text | string | The number of bathrooms in the listing.  On the Airbnb web-site, the bathrooms field has evolved from a number to a textual description. For older scrapes, bathrooms is used. |
| bedrooms | integer | The number of bedrooms |
| beds | integer | The number of bed(s) |
| amenities | json |  |
| price | currency | daily price in local currency |
| minimum\_nights | integer | minimum number of night stay for the listing (calendar rules may be different) |
| maximum\_nights | integer | maximum number of night stay for the listing (calendar rules may be different) |
| minimum\_minimum\_nights | integer | the smallest minimum\_night value from the calender (looking 365 nights in the future) |
| maximum\_minimum\_nights | integer | the largest minimum\_night value from the calender (looking 365 nights in the future) |
| minimum\_maximum\_nights | integer | the smallest maximum\_night value from the calender (looking 365 nights in the future) |
| maximum\_maximum\_nights | integer | the largest maximum\_night value from the calender (looking 365 nights in the future) |
| minimum\_nights\_avg\_ntm | numeric | the average minimum\_night value from the calender (looking 365 nights in the future) |
| maximum\_nights\_avg\_ntm | numeric | the average maximum\_night value from the calender (looking 365 nights in the future) |
| calendar\_updated | date |  |
| has\_availability | boolean | [t=true; f=false] |
| availability\_30 | integer | avaliability\_x. The availability of the listing x days in the future as determined by the calendar. Note a listing may not be available because it has been booked by a guest or blocked by the host. |
| availability\_60 | integer | avaliability\_x. The availability of the listing x days in the future as determined by the calendar. Note a listing may not be available because it has been booked by a guest or blocked by the host. |
| availability\_90 | integer | avaliability\_x. The availability of the listing x days in the future as determined by the calendar. Note a listing may not be available because it has been booked by a guest or blocked by the host. |
| availability\_365 | integer | avaliability\_x. The availability of the listing x days in the future as determined by the calendar. Note a listing may not be available because it has been booked by a guest or blocked by the host. |
| calendar\_last\_scraped | date |  |
| number\_of\_reviews | integer | The number of reviews the listing has |
| number\_of\_reviews\_ltm | integer | The number of reviews the listing has (in the last 12 months) |
| number\_of\_reviews\_l30d | integer | The number of reviews the listing has (in the last 30 days) |
| first\_review | date | The date of the first/oldest review |
| last\_review | date | The date of the last/newest review |
| review\_scores\_rating | integer |  |
| review\_scores\_accuracy | integer |  |
| review\_scores\_cleanliness | integer |  |
| review\_scores\_checkin | integer |  |
| review\_scores\_communication | integer |  |
| review\_scores\_location | integer |  |
| review\_scores\_value | integer |  |
| license | text | The licence/permit/registration number |
| instant\_bookable | boolean | [t=true; f=false]. Whether the guest can automatically book the listing without the host requiring to accept their booking request. An indicator of a commercial listing. |
| calculated\_host\_listings\_count | integer | The number of listings the host has in the current scrape, in the city/region geography. |
| calculated\_host\_listings\_count\_entire\_homes | integer | The number of Entire home/apt listings the host has in the current scrape, in the city/region geography |
| calculated\_host\_listings\_count\_private\_rooms | integer | The number of Private room listings the host has in the current scrape, in the city/region geography |
| calculated\_host\_listings\_count\_shared\_rooms | integer | The number of Shared room listings the host has in the current scrape, in the city/region geography |
| reviews\_per\_month | numeric | The number of reviews the listing has over the lifetime of the listing |

|  |  |  |
| --- | --- | --- |
| **Listings Table** | | |
| **Field** | **Type** | **Description** |
| Id | integer | Airbnb's unique identifier for the listing |
| name | text | Name of the listing |
| host\_id | integer | Airbnb's unique identifier for the host/user |
| host\_name | text | Name of the host. Usually just the first name(s). |
| Neighbourhood\_group |  |  |
| neighbourhood | text |  |
| latitude | numeric | Uses the World Geodetic System (WGS84) projection for latitude and longitude. |
| longitude | numeric | Uses the World Geodetic System (WGS84) projection for latitude and longitude. |
| room\_type | text | [Entire home/apt|Private room|Shared room|Hotel]  All homes are grouped into the following three room types:  Entire place Private room Shared room Entire place Entire places are best if you're seeking a home away from home. With an entire place, you'll have the whole space to yourself. This usually includes a bedroom, a bathroom, a kitchen, and a separate, dedicated entrance. Hosts should note in the description if they'll be on the property or not (ex: "Host occupies first floor of the home"), and provide further details on the listing.  Private rooms Private rooms are great for when you prefer a little privacy, and still value a local connection. When you book a private room, you'll have your own private room for sleeping and may share some spaces with others. You might need to walk through indoor spaces that another host or guest may occupy to get to your room.  Shared rooms Shared rooms are for when you don't mind sharing a space with others. When you book a shared room, you'll be sleeping in a space that is shared with others and share the entire space with other people. Shared rooms are popular among flexible travelers looking for new friends and budget-friendly stays. |
| price | currency | daily price in local currency |
| minimum\_nights | integer | minimum number of night stay for the listing (calendar rules may be different) |
| number\_of\_reviews | integer | The number of reviews the listing has |
| last\_review | date | The date of the last.newest review |
| reviews\_per\_month | numeric | the number of reviews the listing has over the lift time of the listing |
| calculated\_host\_listings\_count | integer | The total numbers of host listings count |
| availability\_365 | integer | avaliability\_x. The availability of the listing x days in the future as determined by the calendar. Note a listing may not be available because it has been booked by a guest or blocked by the host. |

# Extra Credit

[linseyxi/6520-final-project (github.com)](https://github.com/linseyxi/6520-final-project)