Neo4j Cypher query:

1. **Import host.csv**:

LOAD CSV WITH HEADERS FROM "file:///host\_v2.csv"

AS row

WITH row WHERE row.host\_id IS NOT NULL

MERGE (h:Host {host\_id: row.host\_id})

ON CREATE SET h.host\_url = row.host\_url,

h.host\_name = row.host\_name,

h.host\_verifications = row.host\_verifications,

h.host\_since = row.host\_since,

h.host\_location = row.host\_location,

h.host\_response\_time = row.host\_response\_time,

h.host\_is\_superhost = row.host\_is\_superhost;

1. **Import listing.csv**:

LOAD CSV WITH HEADERS FROM "file:///listing\_v2.csv"

AS row

WITH row WHERE row.id IS NOT NULL

MERGE (l:Listing {listing\_id: row.id})

ON CREATE SET l.name = row.name,

l.summary = row.summary,

l.listing\_url = row.listing\_url,

l.picture\_url = row.picture\_url,

l.host\_id = row.host\_id,

l.neighbourhood = row.neighbourhood,

l.street = row.street,

l.zipcode = row.zipcode,

l.latitude = row.latitude,

l.longitude = row.longitude,

l.room\_type = row.room\_type,

l.amenities = row.amenities,

l.price = row.price,

l.extra\_people = row.extra\_people,

l.minimum\_nights = row.minimum\_nights,

l.calculated\_host\_listings\_count = row.calculated\_host\_listings\_count,

l.availability\_365 = row.availability\_365;

1. **Import review.csv**:

LOAD CSV WITH HEADERS FROM "file:///review\_v2.csv"

AS row

WITH row WHERE row.id IS NOT NULL

MERGE (r:Review {review\_id: row.id})

ON CREATE SET r.listing\_id = row.listing\_id,

r.date = row.date,

r.reviewer\_id = row.reviewer\_id,

r.reviewer\_name = row.reviewer\_name,

r.review\_scores\_rating = row.review\_scores\_rating,

r.comments = row.comments;

1. **Create relationships**:

LOAD CSV WITH HEADERS FROM "file:///listing\_v2.csv" AS csvLine MATCH (l:Listing {listing\_id: csvLine.id}) MATCH (h:Host {host\_id: csvLine.host\_id}) CREATE (l)-[:list]->(h);

LOAD CSV WITH HEADERS FROM "file:///review\_v2.csv" AS csvLine MATCH (r:Review {review\_id: csvLine.id}) MATCH (l:Listing {listing\_id: csvLine.listing\_id}) CREATE (r)-[:review]->(l);

match (r:Review) SET r.review\_scores\_rating = toInt(r.review\_scores\_rating)

1. **List all accommodation names and locations that do not provide Wi-Fi**

match (a:Listing) where not(lower(a.amenities) contains 'wifi') return a.name,a.street

1. **How many times a reviewer left reviews**

match (a:Review) with count(a) as review\_sum match (b:Review) return review\_sum/count(distinct b.reviewer\_id)

1. **List the pairs of accommodations that have more than three amentities in common**

match (a:Listing) where size(split(a.amenities, ',')) > 3 return a

1. **The listings do not have any review**

match (a:Review) where not exists(a.listing\_id) return count(a)

1. **Hosts have multiple listings, include host and listing details, such as name and price**

match (a:Listing) with a.host\_id as a\_host\_id, count(a.host\_id) as host\_id\_counts where host\_id\_counts > 1 with collect(a\_host\_id) as a\_host\_id\_collect match (b:Host) where b.host\_id in a\_host\_id\_collect with b as Host\_node, a\_host\_id\_collect match (c:Listing) where c.host\_id in a\_host\_id\_collect return c.name, c.price, c.host\_id, c.listing\_id, Host\_node.host\_url, Host\_node.host\_id, Host\_node.host\_name, Host\_node.host\_verifications, Host\_node.host\_since, Host\_node.host\_location, Host\_node.host\_response\_time, Host\_node.host\_is\_superhost

1. **The top 5 most expensive accommodations, include locations, host information, and names of the accommodation**

match (a:Listing) with a as a\_node, collect(a.host\_id) as a\_host\_id\_collect order by a.price desc limit 5 match (b:Host) where b.host\_id in a\_host\_id\_collect return a\_node.name, a\_node.street, b.host\_id, b.host\_url, b.host\_name, b.host\_since, b.host\_location, b.host\_is\_superhost, b.host\_response\_time, b.host\_verifications

1. **The top 10 most popular neighborhoods based on the total average review score ratings**

match (a:Listing) with collect(distinct(a.neighbourhood)) as a\_neighbourhood\_collect match (b:Listing) where b.neighbourhood in a\_neighbourhood\_collect return b.neighbourhood, avg(tointeger(b.price)) order by avg(tointeger(b.price)) desc limit 10

1. **Hosts whose location are different from their listings**

match (a:Listing) with collect(distinct(a.street)) as a\_street\_collect match (b:Listing) where b.street in a\_street\_collect with b.street as b\_street\_collect, collect(b.name) as b\_name\_collect , collect(b.host\_id) as b\_host\_id\_collect match (c:Host) where c.host\_id in b\_host\_id\_collect and not (c.host\_location = b\_street\_collect) return c.host\_name, c.host\_location, b\_name\_collect, b\_street\_collect

1. **How many reviews does the host “ Sunny 1950s Apartments, St Kilda East” have?**

match (a:Listing {name:"Sunny 1950s Apartment, St Kilda East

Longer stays"}) -[:review]-(b:Review) return count(b.review\_id)

1. **All reviews of the neighborhood of “Port Phillip”**

match (a:Listing {neighbourhood:"Port Phillip"}) -[:review]-(b:Review) return b.comments

1. **Accommodations were reviewed in 2017**

match (a:Review) where a.date =~ '2017.\*' return count(\*)