Database Management
Fall Semester 2021

Project Report - LocateNY

Find your preferable accommodation within short distances to your favourite sightseeing spots in New York City

Concept Development & Prototype Design

LUCERNE UNIVERSITY OF APPLIED SCIENCES AND ARTS

SCHOOL OF BUSINESS

APPLIED INFORMATION AND DATA SCIENCE

31.12.2021

HS 2021

SUPERVISORS: MICHAEL KAUFMAN & LUIS TERAN

N-Why-Team:

Mario Corradini

Vladimira Gabor

Thomas Oliver

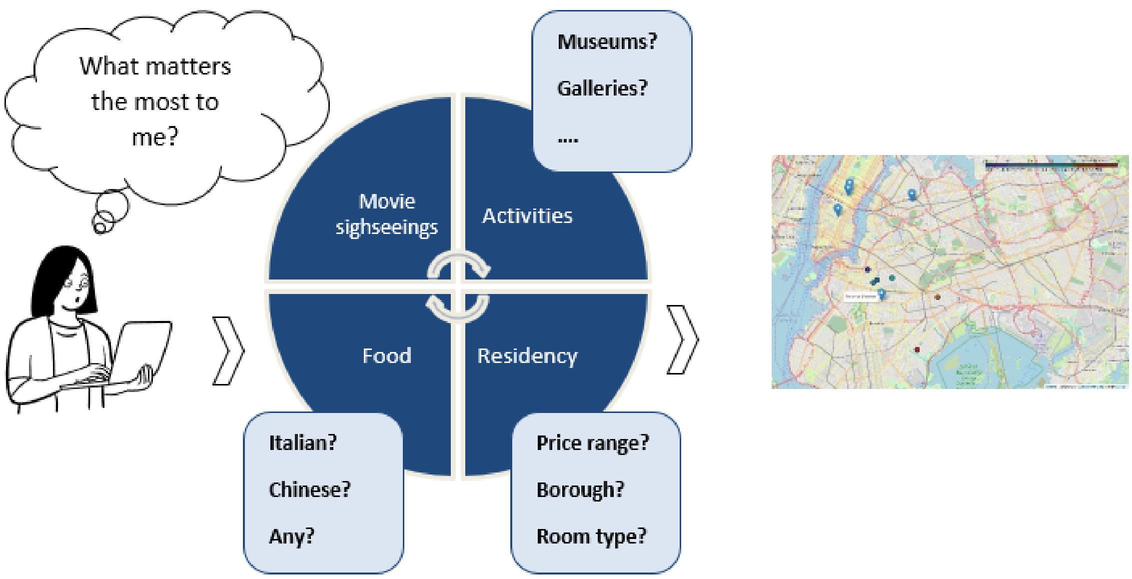
Eitam Shafran

Table of Contents

1. Introduction

There are so many nice places in New York City to visit, so you either need to plan for a long visit or maximise your time spent. LocateNY does the latter, give you a proposal of the best location to stay in a while visiting New York whilst facilitating activities planning based on the shortest commuting time to all of traveller's favourited attractions and the best rated restaurants of his favoured cuisine, in the close distance to your Airbnb residence. This tool helps, in the end, to make the visit to NY much more valuable in terms of preferred places visited in the shortest amount of time. That means it will save the traveller a lot of time in advance because he/she doesn't need to plan where to travel that much but can just rely on our service. The proof of concept focuses on New York City and its places to visit. In the future however, it could also easily be applied to other cities, when the data for it are available.

In our data set, we included information for accommodation and points of interest within New York City: Airbnb, favourite places to visit like restaurants, museums and film locations and the location of the subway stations to plan effective routes throughout the city. All these data points are used to give a final travel proposal based on individual preferences such as placement or sort of accommodation, restaurant style, type of museum or film locations to visit. Currently, this is a bespoke service but will allow for customers to input their own data in the future.



2. Use Case

The use case the authors see for LocateNY is to make the travel planning for a person's trip to NYC less complicated and time-consuming and the journey itself more valuable in terms of their interests.

The vision for the final service offering is that a user can select a range of preferences and interest that sound attractive when visiting New York City (e.g., favourite style restaurant, favourite movies shot in New York, price limitation for accommodation, and more). The app then returns the person using an interactive map with the top-recommended accommodations that fit desired preferences ranked by the shortest overall distance to all his/her favourite activities. For example, selected activities such as restaurants would be returned by ranking and limited to the top 3. For simplification purposes and clear representation of our work, we invented 3 different personas representing 3 different people with unique preferences:

- Persona A – Is looking for accommodation in Brooklyn. The cost constraint was set to $50/night. The look-up for this persona will advertise restaurants serving seafood meals. This user wishes to visit the Brooklyn Museum along with all places New York has to offer in relation to the movie "Spiderman".

- Persona B – Loves Italian food, is a huge fan of "Breakfast at Tiffany's" movie and expressed an interest to visit American Museum of Natural History. The user set accommodation preferences for Staten Island or Manhattan, with the price per night being less than $100.

- Persona C – Prefers Chinese cuisine, wishes to visit sightseeing places related to the "Ghostbusters" movie and may consider visiting The New York's Botanical Garden. The accommodation preferences were set to Queen's borough, with a price range of more than $100/ night.

Importantly to note, there isn't any limitation or logical flow for selecting preferences. Users can decide to complete all selections, most of them, or an absolute minimum. The algorithm will always yield results based on distance conveniences.

3. Underlying Data

Our database's foundation consists of numerous datasets transferred from New York's Open Data website. This "award-winning" website was created as a hub for one-stop access to information that helps the public to understand what is happening in the government at the state & local levels. The directory stores catalogued data, shared transparently with the broad public. While this may come across as an ordinary manifestation (especially when compared to the Swiss governmental approach), the extent and availability of NY's data were an impressive discovery, among the very few.

While our system architecture presents an intricate network of database objects, the provenance of those would lead to a list of three main data sources.

The primary, as mentioned earlier, has been https://opendata.cityofnewyork.us/. This website provides dossiers of insights related to the list of:

- Museum & Galleries
- Restaurants within New York
- Famous Filming Locations one can relate to (segment of the dataset is showcased below)

Film	Year	Director/Filmmaker Name	Location Display Text	LATITUDE	LONGITUDE	Borough	Neighborhood
*batteries not included	1987	Matthew Robbins	E. 5th St. East Village Manhattan	40.7224453	-73.97865057	Manhattan	East Village
12 Angry Men	1957	Sidney Lumet	New York County Courthouse 40 Foley Square Lower Manhattan	40.7137	-74.0079	Manhattan	Lower Manhattan
13 Going on 30	2004	Gary Winick	W. 47th St. and Seventh Ave. Times Square Manhattan	40.75922049	-73.98462117	Manhattan	Times Square
15 Minutes	2001	John Herzfeld	E. 60-66th St. and Madison Ave. Upper East Side Manhattan	40.7661	-73.9696	Manhattan	Upper East Side
25th Hour	2002	Spike Lee	World Trade Center Lower Manhattan	40.71179263	-74.01232839	Manhattan	Lower Manhattan

*Please note that the example above lists already processed and cleaned data entries

The secondary data source is http://insideairbnb.com/get-the-data.html. An open-source dataset from publicly available information related to the latest Airbnb listings within New York City.

host_id	host_name	neighbourhood_group (Borrought)	neighbourhood	latitude	longitude	room_type	price
2571	Teedo	Brooklyn	Bedford-Stuyvesant	40.68686	-73.9371	Entire home/apt	139
2787	John	Brooklyn	Bensonhurst	40.60966	-73.9765	Private room	149
2845	Jennifer	Manhattan	Midtown	40.75356	-73.9856	Entire home/apt	150
2868	Letha M.	Brooklyn	Bedford-Stuyvesant	40.68275	-73.9581	Entire home/apt	60

*Please note that the example above lists already processed and cleaned data entries

The third data source https://new.mta.info/ has fed our database with information around NY's subway stations.

NAME	LONGITUDE	LATITUDE
Astor Pl	-73.99107	40.730054
Canal St	-74.000193	40.718803
50th St	-73.983849	40.761728
Bergen St	-73.97499915	40.68086214
Pennsylvania Ave	-73.89488591	40.66471445
238th St	-73.90087	40.884667

*Please note that the example above lists already processed and cleaned data entries

Location coordinates being the centrepiece of our concept, the immediate need to determine missing latitude and longitude entries arose. We were able to scrape location coordinates from https://www.openstreetmap.org/ using Python's Beautiful Soup library, employing site's address information, which was almost certainly present in our datasets.

4. System Architecture

SQL works logically, just like any other language and the key is in understanding its syntax. The fundamental difference to other coding languages is that instead of taking user-provided inputs, MySQL deals with the contents of the database. As a result, it has all the tools to both access and alter data from the back-end servers.

Figure 1 depicts respective system elements and the interaction in between. We observe software applications loaded to support MySQL remote server on the right side. The left side of the figure depicts software requirements for a local machine. While remote server applications won't require any additional configurations, database engineers and product developers and other team members who wish to access LocateNY's MySQL server must satisfy, from the very start, all system installation pre-requisites: VPN, MySQL Workbench, and Python. Naturally, the access is granted through the established principal, that will bridge the connection between two divided environments. Details that define access criteria are discussed in further detail in section 13.

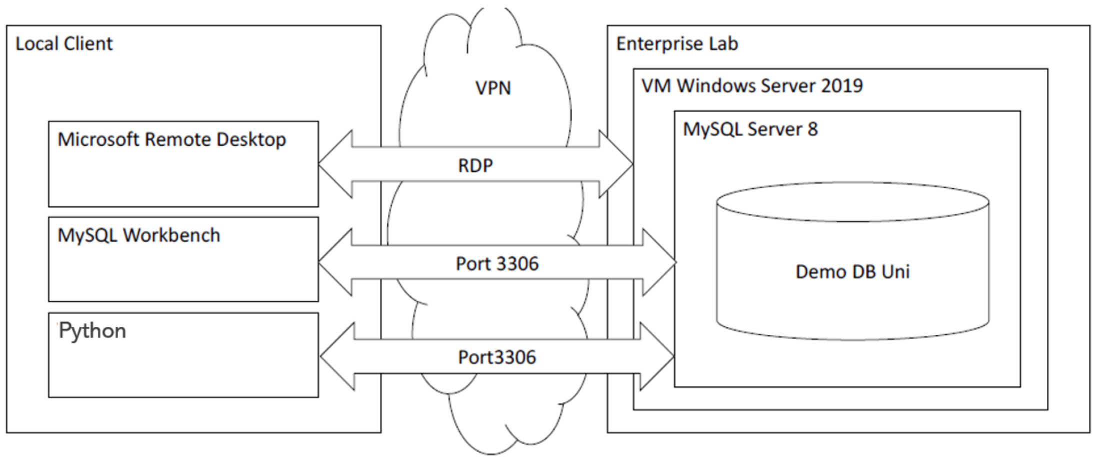


Figure 1: System Architecture

To benefit from MySQL's integration abilities, we decided to use programming language Python to connect to our database, using MySQL Connector and sqlalchemy (see Fig 2), and query it in order to perform the more elaborate distance calculations and visualisation of the results for our end user.

Connect to mysql

```
In [38]:
## Connect to mysql
import mysql.connector
create_cruzer = mysql.connector.connect(user='HSLU', password= '[redacted]', host='[redacted]')
cru = create_cruzer.cursor(dictionary=True) #dictionary=True
```

```
In [39]:
from sqlalchemy import create_engine
import urllib.parse
## create an engine using sqlalchemy
engine = create_engine('mysql+mysqlconnector://HSLU:%s@[redacted]/n_why' % urllib.parse.quote('[redacted]
```

Figure 2: Connecting python to database using mysql.connector and sqlalchemy

5. Data import and pre-processing

Before loading into the Database, the raw data had to be pre-processed as there were several issues. Firstly, and most importantly, there were several characters in the Airbnb data that could not be encoded in UTF-8 or UTF-16 that would cause import errors. Furthermore, there were a lot of superfluous columns that were outside the scope of the project that had to be dropped. Although it was possible to do the pre-processing in MYSQL, the initial pre-processing was done manually in Excel, as it was easier to undo any mistakes that were made.

To solve the issue around the characters that could not be encoded into UTF-8, Notepad++ was used to identify any non-standard characters and replaced with whitespaces. This was achieved by using Regex, similar to what could be achieved in a python script. Notepad++ was used because the time saved using a python script was negligible, but for any future data that could be imported, a pipeline for pre-processing and loading data will be implemented to accommodate any future issues around non-standard characters.

After the initial pre-processing steps, the data was uploaded into the MYSQL schema using the data import wizard as part of the MYSQL workbench. The reason for this was two-fold. Firstly, it was easier to understand the importing process using the GUI of the wizard, including setting primary keys. Secondly, due to the nature of connecting to the server via the GlobalProtect VPN we would often encounter import authentication errors, despite using the root access account. Using the wizard was the only workaround that worked.

Another workaround that did give results was using sqlalchemy in Python to upload directly into an existing table. However, if the table was at all altered it would cause an error, so using the Wizard was the most consistent means of uploading the data into the database.

6. Understanding LocateNY MySQL Server

From object-oriented database to columnar or hierarchical type of database technology, our team decided to implement relational database as it has many advantages over any other database type. It helps maintain the data integrity, accuracy, scalability, reduces data redundancy to minimum, and makes it easy to implement security methods, which are described in further details in chapters below. To pinpoint a few advantages, we could elaborate on:

» Data Accuracy: in a relational database, the design allows the existence of multiple tables related to one another with the use of primary and foreign key concepts. This approach makes the data to be non-repetitive.

» Flexibility: the prospect of levelling up and expanding the relational database conforms to the constantly shifting operational requirements. Furthermore, the relational model allows data analysts to make changes and adjustments without jeopardizing the integrity of the database.

» Normalization: by breaking down the data into smaller tables and establishing links which can be used to query and store data in a logical manner.

» Feasibility to future modifications: the records in a relational database are stored in separate tables based on their classification. This means that any number of new tables or columns of data can be injected or altered depending on predefined conditions by preserving elementary qualities of the relational database system.

6.1 Database Optimization

To optimize the performance of our database, we decided to apply an optimization technique known as normalization. In order to perform the technique, we first needed to de-normalize all of our tables into one single table (See Fig.3).

t_id	host_id	host_name	neighbour	neighbour	latitude	longitude	room_typ	price	...	Museum name
A	2845	Jennifer	Manhattar	Midtown	40.75356	-73.9856	Entire hom	150	...	N/a
A	4869	LisaRoxanne	Brooklyn	Bedford-St	40.68494	-73.9577	Entire hom	76	...	N/a
A	7356	Garon	Brooklyn	Bedford-St	40.68535	-73.9551	Private roo	60	...	N/a
A	7378	Rebecca	Brooklyn	Sunset Par	40.66265	-73.9945	Entire hom	275	...	N/a
...	...	...	...	...	...	...	...	...	...	...
M	N/a	N/a	Manhattar	45 W. 53rd	-74.0191	40.6478	N/a	N/a	...	American Folk Art Museum
M	N/a	N/a	Manhattar	Central Pa	-73.9656	40.79036	N/a	N/a	...	American Museum of Natural History
M	N/a	N/a	Manhattar	725 Park A	-73.9644	40.76989	N/a	N/a	...	Asia Society and Museum

Figure 3: Dataset after de-normalization

Once we finished this step, we were able to start with the normalization process. First, we created a new automated running number primary column in the new merged table called GUID. Then, we were able to normalize the table again, by splitting it into new tables by the type of activity or accommodation, leaving only the common data column exist in all tables in the main table. The remaining columns had information about the object's location (I.e. borough, neighbourhood, longitude, latitude) hence we named it "locations" table. The new tables (Airbnb, subway, film, restaurant, and museum), where now equipped with a new foreign key named “loc_id” referencing each object (activity/Airbnb/subway) to its location in the locations table, enabling us to finish our transformation from an object-oriented database into a normalized relational database (see Fig. 4).

Our motivation was to minimize duplicate data over the different tables, minimize or avoid data modification issues, and to simplify queries. A potential increase in script complexity is possible, but manageable.

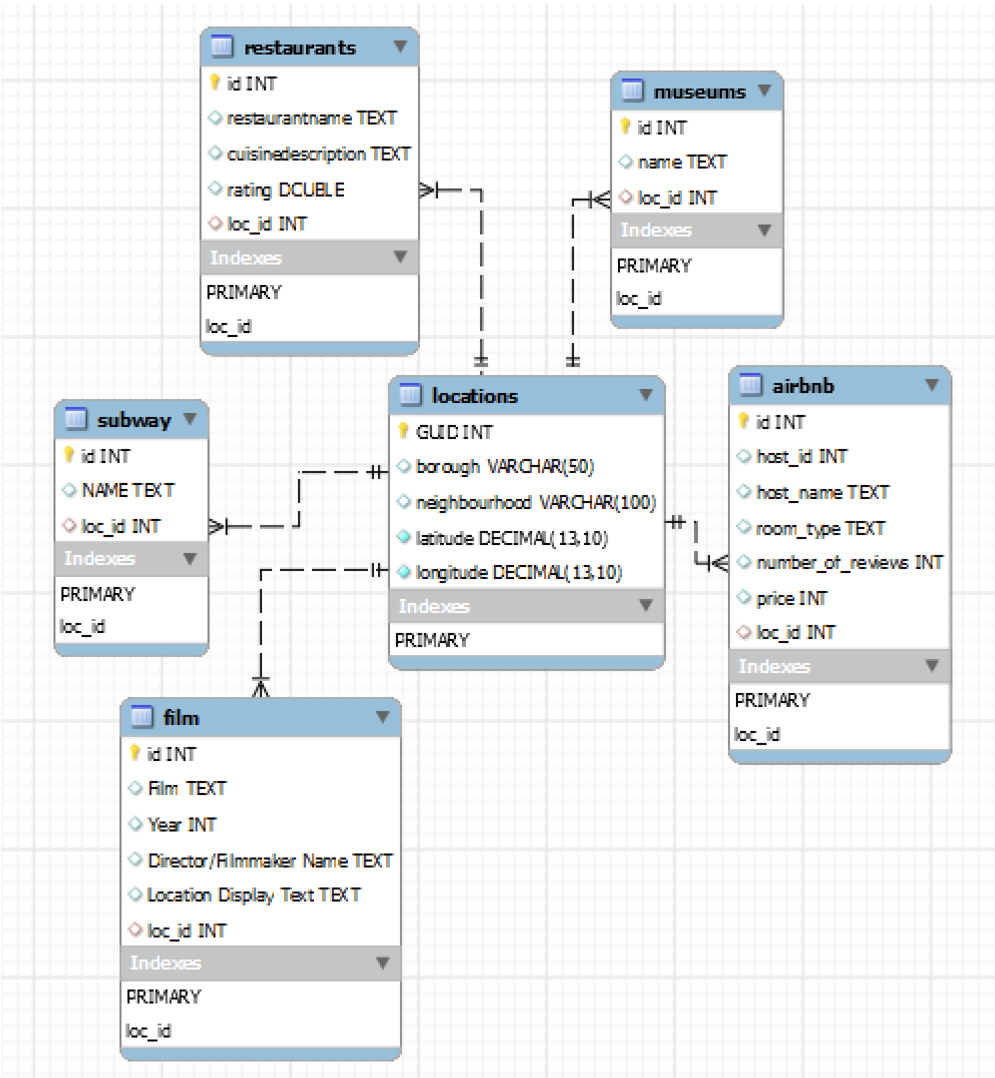


Figure 4: Data schema

The heart of our data schema is the Location table, which includes all the latitude and longitude information of the places to stay or to visit in New York, as well as the neighbourhood and borough the coordinates relate to. To connect the Locations table and the other tables, a unique value, here called the GUID, was created as a primary key (PK) in the Locations table. For the rest of the tables the same code was implemented as a foreign key (FK). The reason for this reference to the Location table is to ultimately calculate the shortest distance between the places that match with the customer's wishes.

There was no need for any triggers, functions, or stored procedures to be implemented as the data will not be updated regularly enough to justify the added complexity in the schema.

6.2 Design of Query and Analysis

In order to calculate the required distances from the top-recommended accommodations which fit each person's preferences to all of his favourite activities, two tables needed to be queried from the server to the python script:

Person_x_accommodations- Containing the longitude and latitude coordinates of the top 6 accommodations by a number of reviews which fits each person's preferences (e.g. price range and preferred borough to stay).

```
CREATE VIEW person_c_accommodations AS
   (SELECT
       joined.GUID, joined.latitude, joined.longitude, joined.number_of_reviews, joined.price
   FROM
       ((SELECT
           GUID, latitude, longitude, number_of_reviews, price
       FROM
           acc_q
       INNER JOIN (SELECT
           *
       FROM
           airbnb
       WHERE
           price >= 100) AS acc_price ON acc_q.GUID = acc_price.loc_id)  ORDER BY number_of_reviews DESC LIMIT 6) AS joined);
```

Person_x_activities- Containing the longitude and latitude coordinates of the person's favourite activities and their names. When selecting restaurants as a favourite activity the table will present the top 3 restaurants of the person favourite cuisine.

```
CREATE VIEW person_b_activities AS(
SELECT
    GUID, latitude, longitude, name
FROM locations

-- filter the locations that the person demend from loactions table
INNER JOIN

-- Create a column with all the loactions GUID which the person demand
(
(SELECT loc_id, Film AS name
FROM
   (SELECT * FROM film
WHERE
    Film = 'Breakfast at Tiffany\'s' ) AS tiffany)

UNION

(SELECT loc_id, restaurantname AS name
FROM
     (SELECT * FROM  restaurants
    WHERE
        cuisinedescription = 'Italian'
    ORDER BY rating DESC
    LIMIT 3) AS italian)

UNION

 (SELECT
    loc_id, name
FROM (SELECT * FROM museums
WHERE name = 'American Museum of Natural History') AS N_history)
) AS merged_loc

ON locations.GUID = merged_loc.loc_id
);
```

To simplify the query process, several views of repeating processes were created to make person_x_accommodations view:

"Accommodations" view- filtered view of "locations" table containing the location attributes of the Airbnb locations only. The view was created using the INNER JOIN commend utilizing the location table primary key (GUID) and the airbnb table foreign key (loc_id)

```
CREATE VIEW accommodations AS(
SELECT GUID, borough, neighbourhood, latitude, longitude FROM locations
INNER JOIN airbnb ON locations.GUID = airbnb.loc_id
)
```

"Acc_X" view- Additional views based on potential personas favourite locations to stay, in this example a specific borough, where created. In this example we filter the accommodations view even further based on persona A desire to stay in Brooklyn during his visit to New York. This view can be utilized for other personas with similar requirements.

```
CREATE VIEW acc_B AS
(SELECT GUID, latitude, longitude FROM accommodations
WHERE borough = 'Brooklyn')
```

"Person_x_accomodations" view- Eventually, for the purpose of this project, we created person_x_accomodations table view. In the following example person_a_accomodations view was created utilizing the acc_b view. An additional filter was applied on the view using the INNER JOIN statement with a filtered airbnb table, where only Airbnb's which fit the person's requests (in this case price lower than 50$) remained. The final joined results was then limit to the top six Airbnb by the number of reviews.

```
CREATE VIEW person_a_accommodations AS
   (SELECT
       joined.GUID, joined.latitude, joined.longitude, joined.number_of_reviews, joined.price
   FROM
       ((SELECT
           GUID, latitude, longitude, number_of_reviews, price
       FROM
           acc_b
       INNER JOIN (SELECT
           *
       FROM
           airbnb
       WHERE
           price <= 50) AS acc_price ON acc_b.GUID = acc_price.loc_id)
           ORDER BY number_of_reviews DESC LIMIT 6) AS joined);
```

person_x_activities - For the creation of person_x_activities view a united table of all the foreign keys and name of those activities which the persona selected was first created. This table was then used to filter the locations table using an INNER JOIN statement. In the following example, for person b, the foreign keys of all Breakfast at Tiffany's movie locations were united with the top three Italian restaurants and the American museum of natural science. The table then used to extract only the coordinates and the names of those activities from the newly joined table with the locations table.

```
CREATE VIEW person_b_activities AS(
SELECT
    GUID, latitude, longitude, name
FROM locations

-- filter the locations that the person demend from loactions table
INNER JOIN

-- Create a column with all the loactions GUID which the person demand
(
(SELECT loc_id, Film AS name
FROM
   (SELECT * FROM film
WHERE
    Film = 'Breakfast at Tiffany\'s' ) AS tiffany)
```

```
UNION

(SELECT loc_id, restaurantname AS name
FROM
     (SELECT * FROM  restaurants
    WHERE
        cuisinedescription = 'Italian'
    ORDER BY rating DESC
    LIMIT 3) AS italian)

UNION

 (SELECT
    loc_id, name
FROM (SELECT * FROM museums
WHERE name = 'American Museum of Natural History') AS N_history)
) AS merged_loc

ON locations.GUID = merged_loc.loc_id
);
```

The final two views created for each persona was then could easily be queried directly into the python script using a simple SELECT * FROM table query.

Person A

In [56]:
```
Pa_acc_statment = "SELECT * FROM person_a_accommodations"
Pa_act_statment = "SELECT * FROM person_a_activities"
```

In [60]:
```
## Load tables from server
Pa_acc = pd.read_sql(Pa_acc_statment, engine)
Pa_act = pd.read_sql(Pa_act_statment, engine)
```

In [61]:
```
Pa_acc.head()
```

Out[61]:

	GUID	latitude	longitude	number_of_reviews	price
0	5	40.68668	-73.95016	372	39
1	10055	40.68452	-73.95378	279	50
2	5922	40.69503	-73.95971	275	49
3	8893	40.63155	-73.90812	252	50
4	9306	40.67306	-73.88700	227	32

7. Data visualisation

The output of predefined persona queries is presented to the user as specific location points on the New York city map (see persona_x.html attachments). Not only will a user have the ability to get an overview of all queries related "LocateNY" recommendations, but they will also have the ability to zoom in for further detail, as Persona A & B decided to do. The future prospect of the initial proof of concept will need to take into account the significance of the distinction of map icons based on a particular suggestion category. To improve user experience, colour, shape, size, text, and additional details will be defined and encoded into the service application.

For the time being, the most prominent distance indicator designed to help our user with decision-making is the map legend located in the upper right corner of every output example. The traveller will have an opportunity to analyse the convenience of potential accommodation in relation to desired sightseeing and dining recommendations. Colour gradient from light green to dark red, shows the sum of distances from Airbnb location to all displayed recommendation spots. Dark red symbolises least convenient location, while light green Airbnb mark has the most convenient address.

One of the most prominent aspects of our concept is the distance calculation formula that feeds data into a map plot. The complete code can be found in appendix G. It includes all stages needed to be complete before the generation of the semi-final product – interactive map.

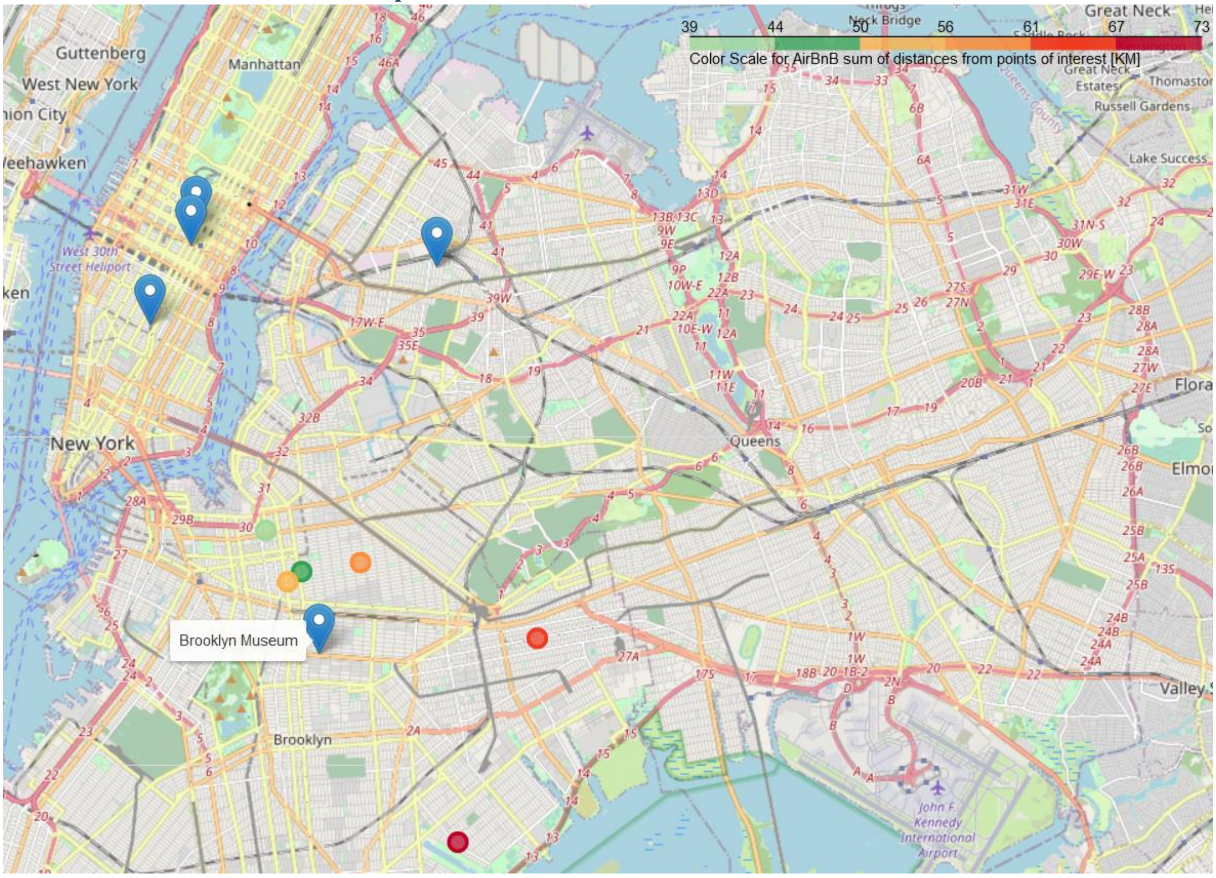
2 Sum of distance calculation function

```
## Function to calculate sum of distance from each place to stay to all␣
↪activities
def dist(place, activity):
    sum_d = []
    for i in range(len(place)):
        d = []
        for j in range(len(activity)):
```

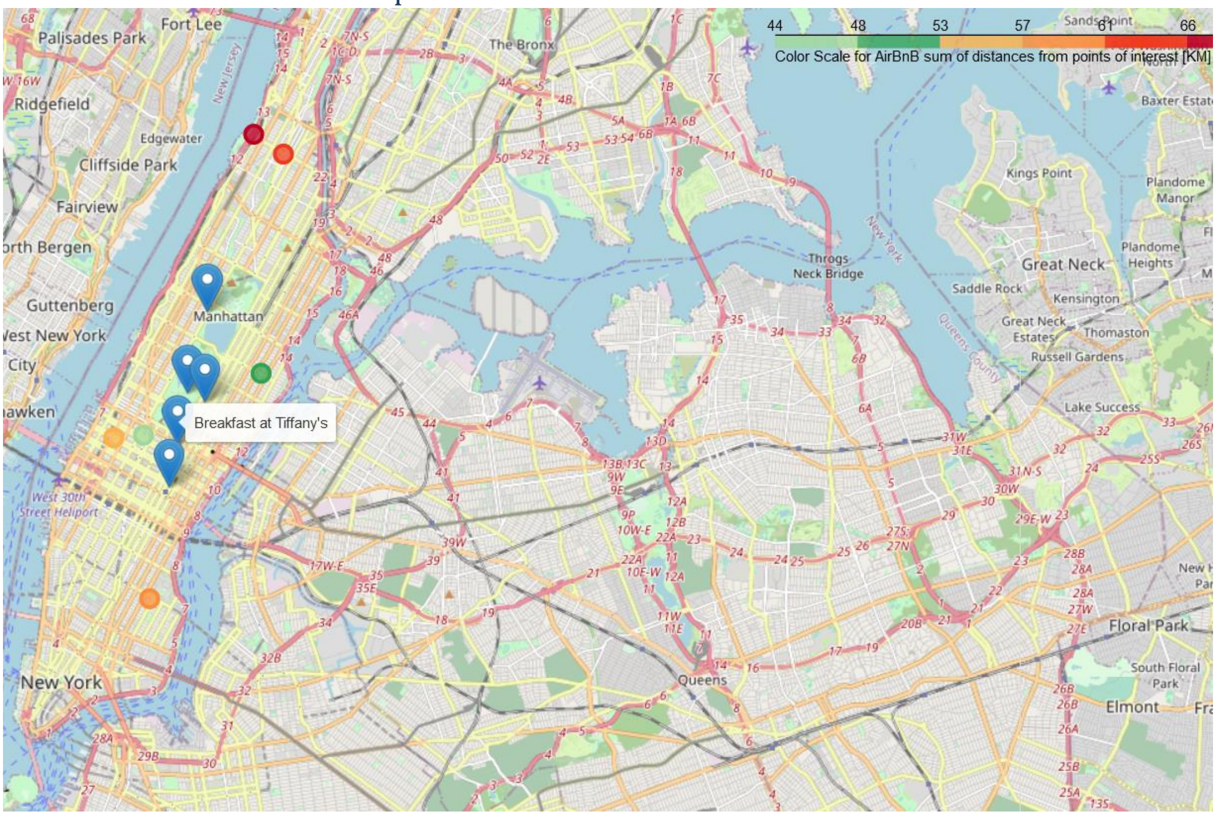
```
            orig_node = ox.get_nearest_node(G, (place.loc[i, "latitude"], place.
↪loc[i,"longitude"]))
            dest_node = ox.get_nearest_node(G, (activity.loc[j,"latitude"],␣
↪activity.loc[j, "longitude"]))
            # how long is our route in meters?
            tmp = nx.shortest_path_length(G, orig_node, dest_node,␣
↪weight='length')
            d.append(tmp)
        sum_d.append(sum(d)) # sum distance of all locations from airbnb and␣
↪append to list

    return (sum_d)
```

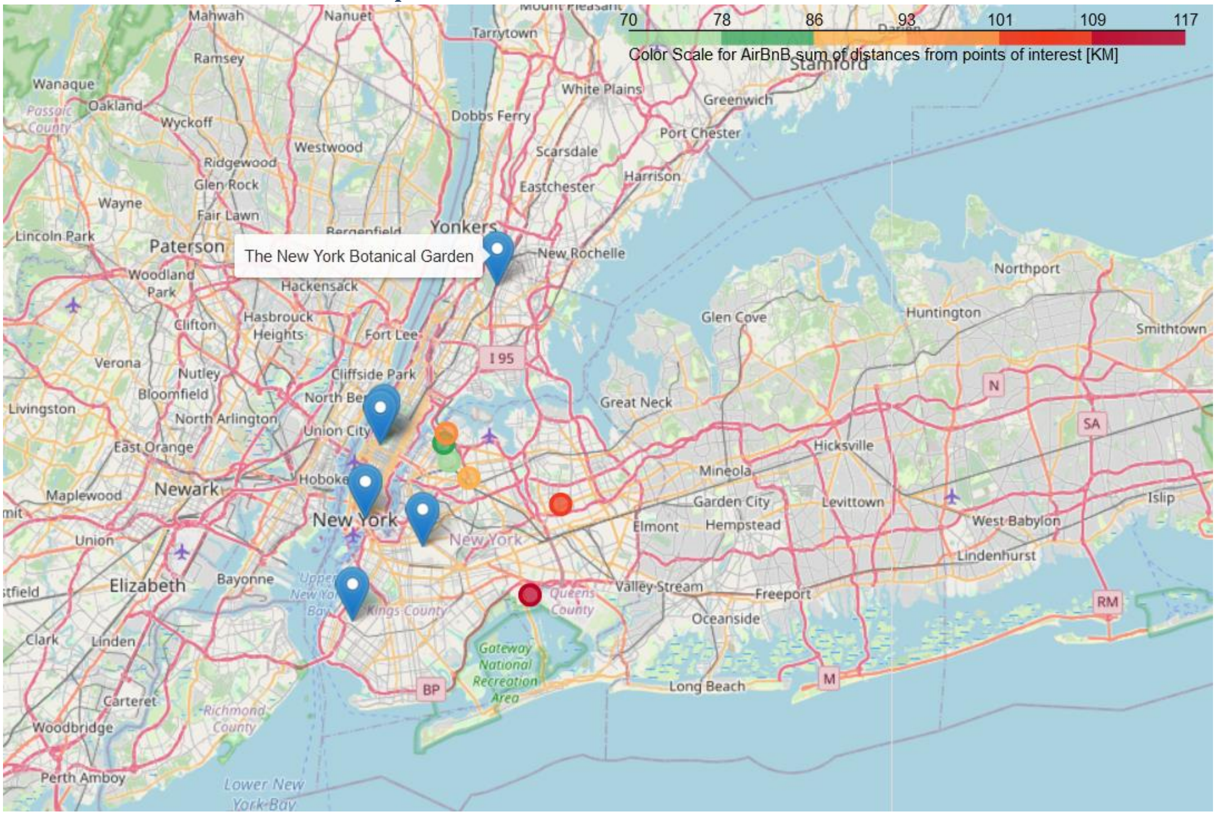
Screenshot of Interactive Map - Persona A



Screenshot of Interactive Map - Persona B



Screenshot of Interactive Map - Persona C



8. Decision support and derived recommendations

The output of this project aims to help users with planning their stay in NY City and empower them to make a better-informed decision by, if you will, simulating the location of their stay and outlining distances to/from possible NY activities. As mentioned earlier, there is no limitation to the preferences selection flow. The freedom to assess the priority of residing in the specific neighbourhood while recognizing sightseeing and dining opportunities in one map, all put in proportion to the relative distance, may objectively aid in answering How, Where, and What questions.

Conveying the space, which merges across-the-board available data to the application users, will allow them to flexibly adjust preferences and decide on optimal use of their time while visiting New York. One of the advantages is the practicality of the application. Not only can it be used before travellers arrive in New York as a planning tool, but it can also be used to query information during the stay to get inspired, whether it’s a sightseeing location, museum, or dining experience.

9. Analysis of the project in terms of security aspects

MySQL is scalable, reliable, and secure. The system itself provides tools and mechanisms which ensure data integrity, protect database from unauthorized access, avoid destruction and data loss. Our team has decided to instal MySQL on a remote server. Therefore, the security of a physical environment wasn't in the scope of this project. However, each team member understands the seriousness of protecting the host machine from unauthorized physical access, as well as natural or man-made disasters. From a security network perspective, to increase security of the operating system, we have enabled firewall as a restrictor of network traffic. Each team member had identical access rights, server- level, and database-level permissions. In other words, we all used a singular security principal, which inflicted a major challenge on data consistency and transactional security. Data integrity being the ultimate requirement for successful database applications, we applied ACID concept to our transactions. The principles of ACID highlight the need of sequencing transactional operations in a way which ensures (A) Atomicity, (C) Consistency, (I) Isolation, (D) Durability. (Meier et al, 2019)

MySQL principals, securables, and permissions, the root foundation of the authentication and authorizations mechanisms were universally discussed throughout the development of our concept. Should we pursue the idea and continue progressing, a thorough security structure would need to be incorporated into day-to-day business operations. To reinforce the overall solution security and reassure data integrity and consistency, our security structure must consider following components:

Two high level categories of MySQL server security principals:

- MySQL Server level (MySQL logins & server roles)
- Database level (database users & database roles)

Three hierarchical levels of securables:

- Server-level securables (objects as databases & availability groups)
- Database-level securabes (objects as schemas & full-text catalogues)
- Schema-level securables (tables, views, functions & stored procedures)

As of now, we have created only two database - level roles, one for developers and the other for our customers (see Fig. 5).

```
CREATE ROLE
        n_why_customer,
    n_why_dev;
GRANT SELECT
        ON n_why.*
    TO n_why_customer;
GRANT ALL
        ON n_why.*
    TO n_why_dev;

CREATE USER nwhycustomer@localhost IDENTIFIED BY 'defaultpassword';

GRANT n_why_customer
TO nwhycustomer@localhost;

SHOW GRANTS FOR nwhycustomer@localhost;
```

Figure 5: Current roles in LocateNY's database

10. Reflexion & Lessons Learned

After organising ourselves at the beginning of the project, some of us gathered data and one person installed the VM to upload the gathered data to it. The colleague, who installed the VM, explained the whole team how he did it, as it was for all of us new to install an own VM, so this was the first main learning we took from this project. Then each of us had to install MySQL on our own computer and connect to the VM, which was interesting to see how this works for all of us. After gathering the data needed it was uploaded to the VM, whereby it was stunning to see how long this takes for such big chunks of data. Then the SQL skills of our group got challenged considering doing the connections between the tables, creating new reference tables like the location table, rearranging tables and in the end diminish the data with views, to do queries from. The hardest part thereby was to understand how SQL works and arrange our project plan to the language. As for most of the project members it was new to work with the programming language SQL, it came along, after an intensive communication in the team with several adjustments of our plan, to find in the end the best way to approach our project task. During this part of getting the correct structure of the data schema, the team reorganised itself, because it was not worth it to have all working on it, so half of the team already went ahead with further workings to do. Like this we were much more efficient and could focus on the team members strengths.

After learning to query in MySQL it became obvious that it's much more convenient than doing it in python. However, in the data visualization part we then changed again to python. So, this project helped the team members to understand how to work together in MySQL on the same data and how to use the strengths of SQL in the storing query part and combine it in the end with python for the visualization to get the best outcome. For further improvements a front-end application would be very useful and the possibility to do more personalized queries to finally get better travel recommendations.

Security measures were another lesson learned for our team members. We haven't planned for unique access credentials, nor did we restrain any engineering account on any securable levels. In the future, having pre-defined roles assigned to user accounts, which have narrowed down permissions based on actual needs of users, will indeed be attended in more depth.

11. Appendix

A: Data Upload Script

```
SHOW TABLES FROM `n_why` like 'businesses';
CREATE TABLE `n_why`.`businesses`
(`MyUnknownColumn` int,
`INDUSTRY` text,
`BUSINESS_NAME` text,
`BUILDING` text,
`STREET` text,
`CITY` text, `ZIP`
 int, `BOROUGH` text);
PREPARE stmt FROM 'INSERT INTO `n_why`.`businesses`
(`MyUnknownColumn`,
`INDUSTRY`,
`BUSINESS_NAME`,
`BUILDING`,
`STREET`,
`CITY`,
`ZIP`,
`BOROUGH`) VALUES(?,?,?,?,?,?,?,?)';

DEALLOCATE PREPARE stmt;
```

B: Creating Locations Table Script

```
USE n_why;
CREATE TABLE location_new
(
GUID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
borough VARCHAR(50),
neighbourhood  VARCHAR(100),
latitude DECIMAL(13, 10) NOT NULL,
longitude DECIMAL(13, 10) NOT NULL,
ref_id INT NOT NULL,
t_id VARCHAR(1) NOT NULL)
;

INSERT INTO location_new ( borough, neighbourhood, latitude, longitude, ref_id, t_id)
SELECT boro, neighbourhood,  latitude, longitude, id, t_id from airbnb_new;

INSERT INTO location ( latitude, longitude, ref_id, t_id)
SELECT latitude, longitude, id, t_id from hotels_clean;

INSERT INTO location_new(borough, neighbourhood, latitude, longitude, ref_id, t_id)
SELECT Borough, Neighborhood, latitude, longitude, id, t_id from film_clean;

INSERT INTO location_new(borough, neighbourhood, latitude, longitude, ref_id, t_id)
SELECT borough, neighbourhood, latitude, longitude, id, t_id from hotels_clean;

INSERT INTO location_new(borough, latitude, longitude, ref_id, t_id)
SELECT BOROUGH, latitude, longitude, id, t_id from museums_with_geo;

INSERT INTO location_new(borough, latitude, longitude, ref_id, t_id)
SELECT BORO, latitude, longitude, id, t_id from restaurants100_with_geo;

INSERT INTO location_new(latitude, longitude, ref_id, t_id)
SELECT latitude, longitude, id, t_id from subway_clean;

select * from location_new limit 10000000;
```

C: Foreign Key Assignment

```
### AIRBNB###
ALTER TABLE airbnb ADD FOREIGN KEY (loc_id) REFERENCES location(GUID);

UPDATE airbnb a
INNER JOIN location l ON a.id = l.ref_id AND a.t_id=l.t_id
SET a.loc_id = l.GUID;

### Film_clean###
ALTER TABLE film_clean ADD COLUMN loc_id INT;

ALTER TABLE film_clean ADD FOREIGN KEY (loc_id) REFERENCES location(GUID);

UPDATE film_clean a
INNER JOIN location l ON a.id = l.ref_id AND a.t_id=l.t_id
SET a.loc_id = l.GUID;

### hotels_clean###
ALTER TABLE hotels_clean ADD COLUMN loc_id INT;

ALTER TABLE hotels_clean ADD FOREIGN KEY (loc_id) REFERENCES location(GUID);

UPDATE hotels_clean a
INNER JOIN location l ON a.id = l.ref_id AND a.t_id=l.t_id
SET a.loc_id = l.GUID;

### museums###
ALTER TABLE museums_with_geo ADD COLUMN loc_id INT;

ALTER TABLE museums_with_geo ADD FOREIGN KEY (loc_id) REFERENCES location(GUID);

UPDATE museums_with_geo a
INNER JOIN location l ON a.id = l.ref_id AND a.t_id=l.t_id
SET a.loc_id = l.GUID;

### restaurants###
ALTER TABLE restaurants100_with_geo ADD COLUMN loc_id INT;

ALTER TABLE restaurants100_with_geo ADD FOREIGN KEY (loc_id) REFERENCES location(GUID);

UPDATE restaurants100_with_geo a
INNER JOIN location l ON a.id = l.ref_id AND a.t_id=l.t_id
SET a.loc_id = l.GUID;

### subways###
ALTER TABLE subway_clean ADD COLUMN loc_id INT;

ALTER TABLE subway_clean ADD FOREIGN KEY (loc_id) REFERENCES location(GUID);

UPDATE subway_clean a
INNER JOIN location l ON a.id = l.ref_id AND a.t_id=l.t_id
SET a.loc_id = l.GUID;
```

D: Generate t_id in Airbnb table

```
SELECT * FROM n_why.airbnb_new;

ALTER TABLE n_why.airbnb_new
ADD COLUMN t_id VARCHAR(1) NOT NULL DEFAULT 'A';

ALTER TABLE `n_why`.`airbnb_new`
CHANGE COLUMN `t_id` `t_id` VARCHAR(1) NOT NULL DEFAULT 'A' AFTER `id`;
```

E: Generate t_id in Airbnb & Subway table

```
SELECT * FROM n_why.airbnb;

ALTER TABLE n_why.airbnb
ADD COLUMN t_id VARCHAR(1) NOT NULL DEFAULT 'A';

ALTER TABLE `n_why`.`airbnb`
CHANGE COLUMN `t_id` `t_id` VARCHAR(1) NOT NULL DEFAULT 'A' AFTER `id`,
DROP PRIMARY KEY,
ADD PRIMARY KEY (`id`, `t_id`);

ALTER TABLE n_why.subway_clean
ADD COLUMN t_id VARCHAR(1) NOT NULL DEFAULT 'S';

ALTER TABLE `n_why`.`subway_clean`
CHANGE COLUMN `t_id` `t_id` VARCHAR(1) NOT NULL DEFAULT 'S' AFTER `id`,
DROP PRIMARY KEY,
ADD PRIMARY KEY (`id`, `t_id`);
```

F: Generate t_id in Film table

```
SELECT * FROM n_why.film_clean;

CREATE TRIGGER init_uuid BEFORE INSERT on n_why.film_clean
        FOR EACH ROW SET NEW.ID = UUID();


ALTER TABLE n_why.film_clean
ADD COLUMN id INT NOT NULL AUTO_INCREMENT PRIMARY KEY;

ALTER TABLE n_why.film_clean
ADD COLUMN t_id VARCHAR(1) DEFAULT 'F';

ALTER TABLE `n_why`.`film_clean`
CHANGE COLUMN `id` `id` INT NOT NULL AUTO_INCREMENT FIRST,
ADD UNIQUE INDEX `id_UNIQUE` (`id` ASC) VISIBLE;
;
```

G: Accommodation distance calculator

```
import networkx as nx
import osmnx as ox
import warnings
warnings.filterwarnings('ignore')
```

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

```
# download/model a street network for some city then visualize it
G = ox.graph_from_place("NYC, USA", network_type="drive")
#fig, ax = ox.plot_graph(G)
```

1 Connect to mysql

```
## Connect to mysql
import mysql.connector
create_cruzer = mysql.connector.connect(user='HSLU', password= 'HerTeam@2021!',␣
↪host='db-vm-27.el.eee.intern')
cru = create_cruzer.cursor(dictionary=True) #dictionary=True
```

```
from sqlalchemy import create_engine
import urllib.parse
## create an engine using sqlalchemy
engine = create_engine('mysql+mysqlconnector://HSLU:%s@db-vm-27.el.eee.intern/
↪n_why' % urllib.parse.quote('HerTeam@2021!'))
```

2 Sum of distance calculation function

```
## Function to calculate sum of distance from each place to stay to all␣
↪activities
def dist(place, activity):
    sum_d = []
    for i in range(len(place)):
        d = []
        for j in range(len(activity)):
```

```
Pa_acc["Distance"] = [round(num*0.001,2) for num in distance_A]
Pa_acc
```

	GUID	latitude	longitude	number_of_reviews	price	Distance
0	5	40.68668	-73.95016	372	39	41.33
1	10055	40.68452	-73.95378	279	50	41.48
2	5922	40.69503	-73.95971	275	49	38.50
3	8893	40.63155	-73.90812	252	50	72.91
4	9306	40.67306	-73.88700	227	32	58.81
5	426	40.68837	-73.93429	219	35	41.97

3.1 Plot map

```
import folium
```

```
#custom color close distance blue, far distance red
#colors = ['#000066','#003366', '#004d66', '#006666', '#663300','#660000']
#colors = ['#ffffb2','#fed976', '#feb24c', '#fd8d3c', '#f03b20','#bd0026']
colors = ['#a1d99b','#31a354', '#feb24c', '#fd8d3c', '#f03b20','#bd0026']
#colors = ['#edf8fb','#ccece6', '#99d8c9', '#66c2a4', '#2ca25f','#006d2c']

Pa_acc.sort_values("Distance", inplace=True)
```

```
Pa_acc["colors"] = colors
```

```
## Creating color legand
import branca.colormap as cmp
step = cmp.StepColormap(
 colors,
   vmin= min(Pa_acc["Distance"]),  vmax= max(Pa_acc["Distance"]),

 caption='Color Scale for AirBnB sum of distances from points of interest [KM]'␣
↪    #Caption for Color scale or Legend
)
```

```
m_A = folium.Map(location=[40.7088, -74.0108], zoom_start=11)
for index, row in Pa_acc.iterrows():
    popup_txt = "<strong>Airbnb details</strong><br>Price: " +␣
↪str(row["price"]) + "$<br>Number of reviews: " +␣
↪str(row["number_of_reviews"])
    iframe = folium.IFrame(popup_txt)
    popup = folium.Popup(iframe,
                         min_width=200,
                         max_width=200)
```

```
            orig_node = ox.get_nearest_node(G, (place.loc[i, "latitude"], place.
↪loc[i,"longitude"]))
            dest_node = ox.get_nearest_node(G, (activity.loc[j,"latitude"],␣
↪activity.loc[j, "longitude"]))
            # how long is our route in meters?
            tmp = nx.shortest_path_length(G, orig_node, dest_node,␣
↪weight='length')
            d.append(tmp)
        sum_d.append(sum(d)) # sum distance of all locations from airbnb and␣
↪append to list

    return (sum_d)
```

3 Person A

```
Pa_acc_statment = "SELECT * FROM person_a_accommodations"
Pa_act_statment = "SELECT * FROM person_a_activities"
```

```
## Load tables from server
Pa_acc = pd.read_sql(Pa_acc_statment, engine)
Pa_act = pd.read_sql(Pa_act_statment, engine)
```

```
Pa_acc.head()
```

	GUID	latitude	longitude	number_of_reviews	price
0	5	40.68668	-73.95016	372	39
1	10055	40.68452	-73.95378	279	50
2	5922	40.69503	-73.95971	275	49
3	8893	40.63155	-73.90812	252	50
4	9306	40.67306	-73.88700	227	32

```
Pa_act.head()
```

	GUID	latitude	longitude	name
0	16532	40.752589	-73.979756	Spider-Man
1	16533	40.756685	-73.978554	Spider-Man
2	16534	40.748232	-73.913999	Spider-Man
3	24928	40.736679	-73.990762	BLUE WATER GRILL
4	24836	40.669606	-73.945580	Brooklyn Museum

```
## Calculate sum of distance from each place to stay to all locations
distance_A = dist(Pa_acc.loc[:,["latitude", "longitude"]], Pa_act.loc[:
↪,["latitude", "longitude"]])
```

```
## Adding sum of distances column to accomodation table after converting from␣
↪meter to KM
```

```
    folium.CircleMarker([row['latitude'], row['longitude']], radius=7,␣
↪fill_color=row['colors'], color=row['colors'], fill_opacity=0.7,␣
↪tooltip="<strong>Airbnb</strong>", popup= popup).add_to(m_A)

## Adding a marker for each activity
for idx, eq in Pa_act.iterrows():
    folium.Marker(location=(eq['latitude'], eq['longitude']),
                  tooltip= eq["name"]).add_to(m_A)
m_A.add_child(step)
m_A
```

```
<folium.folium.Map at 0x2025bfb7be0>
```

```
m_A.save("person_A_new.html")
```

4 Person B

– Love Italian food, breakfast at tiffany's movie and really wants to visit the American museum of natural history.
– can only afford to pay less then 100 a day, want to stay at Staten Island or Manhattan

```
Pb_acc_statment = "SELECT * FROM person_b_accommodations"
Pb_act_statment = "SELECT * FROM person_b_activities"
```

```
## Load tables from mariadb as 1 merged table
Pb_acc = pd.read_sql(Pb_acc_statment, engine)
Pb_act = pd.read_sql(Pb_act_statment, engine)
```

```
Pb_acc.head()
```

	GUID	latitude	longitude	number_of_reviews	price
0	369	40.82380	-73.94444	560	42
1	2956	40.73024	-73.98147	516	98
2	19	40.76457	-73.98317	490	68
3	6221	40.76424	-73.99152	445	52
4	615	40.82772	-73.95284	422	75

```
Pb_act.head()
```

	GUID	latitude	longitude	name
0	16410	40.762510	-73.974142	Breakfast at Tiffany's
1	16411	40.771361	-73.966430	Breakfast at Tiffany's
2	16412	40.773213	-73.971280	Breakfast at Tiffany's
3	24925	43.149367	-77.600423	GENTE
4	24934	40.753613	-73.976580	NAPLES 45 RESTAURANT

```
## Calculate sum of distance from each place to stay to all locations
distance_B = dist(Pb_acc.loc[:,["latitude", "longitude"]], Pb_act.loc[:
↪,["latitude", "longitude"]])
```

```
## Adding sum of distances column to accomodation table after converting from␣
↪meter to KM
Pb_acc["Distance"] = [round(num*0.001,2) for num in distance_B]
Pb_acc
```

	GUID	latitude	longitude	number_of_reviews	price	Distance
0	369	40.82380	-73.94444	560	42	67.61
1	2956	40.73024	-73.98147	516	98	61.77
2	19	40.76457	-73.98317	490	68	43.87
3	6221	40.76424	-73.99152	445	52	47.87
4	615	40.82772	-73.95284	422	75	70.20
5	2199	40.77780	-73.95084	403	81	46.74

4.1 Plot map

```
import folium
```

```
#custom color close distance blue, far distance red
#colors = ['#000066','#003366', '#004d66', '#006666', '#663300','#660000']
colors = ['#a1d99b','#31a354', '#feb24c', '#fd8d3c', '#f03b20','#bd0026']

Pb_acc.sort_values("Distance", inplace=True)
```

```
Pb_acc["colors"] = colors
```

```
## Creating color legand
import branca.colormap as cmp
step = cmp.StepColormap(
 colors,
   vmin= min(Pb_acc["Distance"]),  vmax= max(Pb_acc["Distance"]),

 caption='Color Scale for AirBnB sum of distances from points of interest [KM]'␣
↪    #Caption for Color scale or Legend
)
```

```
m_B = folium.Map(location=[40.7088, -74.0108], zoom_start=11)
for index, row in Pb_acc.iterrows():
    popup_txt = "<strong>Airbnb details</strong><br>Price: " +␣
↪str(row["price"]) + "$<br>Number of reviews: " +␣
↪str(row["number_of_reviews"])
    iframe = folium.IFrame(popup_txt)
    popup = folium.Popup(iframe,
                      min_width=200,
                      max_width=200)

    folium.CircleMarker([row['latitude'], row['longitude']], radius=7,␣
↪fill_color=row['colors'], color=row['colors'], fill_opacity=0.7,␣
↪tooltip="<strong>Airbnb</strong>", popup= popup).add_to(m_B)

## Adding a marker for each activity
for idx, eq in Pb_act.iterrows():
    folium.Marker(location=(eq['latitude'], eq['longitude']),
                  tooltip= eq["name"]).add_to(m_B)
m_B.add_child(step)
m_B
```

```
<folium.folium.Map at 0x2025f45c2e0>
```

```
m_B.save("person_b_new.html")
```

5 Person C

```
Pc_acc_statment = "SELECT * FROM person_C_accommodations"
Pc_act_statment = "SELECT * FROM person_C_activities"
```

```
## Load tables from mariadb as 1 merged table
Pc_acc = pd.read_sql(Pc_acc_statment, engine)
Pc_act = pd.read_sql(Pc_act_statment, engine)
```

```
Pc_acc.head()
```

	GUID	latitude	longitude	number_of_reviews	price
0	744	40.75684	-73.91286	467	149
1	3775	40.77757	-73.91580	360	308
2	8508	40.76975	-73.91937	326	123
3	7951	40.65697	-73.83344	317	135
4	1993	40.74395	-73.89418	308	125

```
Pc_act.head()
```

	GUID	latitude	longitude	name
0	16441	40.768127	-73.981955	Ghostbusters
1	16442	40.772400	-73.978700	Ghostbusters
2	24915	40.712566	-73.996961	MEI YU SPRING RESTAURANT
3	24952	40.692503	-73.940597	LINDA ASIAN KITCHEN
4	24965	40.635360	-74.009832	NEW STAR SEAFOOD RESTAURANT

```
## Calculate sum of distance from each place to stay to all locations
distance_C = dist(Pc_acc.loc[:,["latitude", "longitude"]], Pc_act.loc[:
↪,["latitude", "longitude"]])
```

```
## Adding sum of distances column to accomodation table after converting from␣
↪meter to KM
Pc_acc["Distance"] = [round(num*0.001,2) for num in distance_C]
Pc_acc
```

	GUID	latitude	longitude	number_of_reviews	price	Distance
0	744	40.75684	-73.91286	467	149	69.98
1	3775	40.77757	-73.91580	360	308	75.42
2	8508	40.76975	-73.91937	326	123	71.18
3	7951	40.65697	-73.83344	317	135	116.99
4	1993	40.74395	-73.89418	308	125	74.42
5	8772	40.72488	-73.80389	305	123	107.58

5.1 Plot map

```
import folium
```

```
#custom color close distance blue, far distance red
#colors = ['#000066','#003366', '#004d66', '#006666', '#663300','#660000']
colors = ['#a1d99b','#31a354', '#feb24c', '#fd8d3c', '#f03b20','#bd0026']

Pc_acc.sort_values("Distance", inplace=True)
```

```
Pc_acc["colors"] = colors
```

```
## Creating color legand
import branca.colormap as cmp
step = cmp.StepColormap(
 colors,
   vmin= min(Pc_acc["Distance"]),  vmax= max(Pc_acc["Distance"]),

 caption='Color Scale for AirBnB sum of distances from points of interest [KM]'␣
↪    #Caption for Color scale or Legend
)
```

```
m_C = folium.Map(location=[40.7088, -74.0108], zoom_start=11)
for index, row in Pc_acc.iterrows():
    popup_txt = "<strong>Airbnb details</strong><br>Price: " +␣
↪str(row["price"]) + "$<br>Number of reviews: " +␣
↪str(row["number_of_reviews"])
    iframe = folium.IFrame(popup_txt)
    popup = folium.Popup(iframe,
                      min_width=200,
                      max_width=200)

    folium.CircleMarker([row['latitude'], row['longitude']], radius=7,␣
↪fill_color=row['colors'], color=row['colors'], fill_opacity=0.7,␣
↪tooltip="<strong>Airbnb</strong>", popup= popup).add_to(m_C)

## Adding a marker for each activity
for idx, eq in Pc_act.iterrows():
    folium.Marker(location=(eq['latitude'], eq['longitude']),
                  tooltip= eq["name"]).add_to(m_C)
m_C.add_child(step)
m_C
```

```
<folium.folium.Map at 0x2025c047b20>
```

```
m_C.save("person_c_new.html")
```

12.Declaration of sole authorship

We hereby declare that we are the sole authors and composers of our thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, we declare that we have acknowledged the work of others by providing detailed references of said work. We also hereby declare that our thesis has not been prepared for another examination or assignment, either in its entirety or excerpts thereof.

Vladimira Gabor	M. Gabor
Mario Corradini	[illegible]
Thomas Oliver	T.Ol
Eitam Shafran	E.Shafran