NYC Restaurants' Public Health Database Project
Group VII

A. Problem Statement

Considering the recovering state of New York City in light of the COVID-19 health crisis, NYC residents are now, more than ever, concerned with their personal health and safety. This is especially true regarding the cleanliness of all public spaces, and with health and safety standards of dining areas. As restaurants, bars, and spaces begin to re-open across the city, the New York City Department of Health and Mental Hygiene (DOHMH) believes residents should have access to the safety information of the dining areas they choose to visit. For example, data on health inspection results and food poisoning cases for each restaurant could be valuable as individuals make dining decisions. Considering this, the DOHMH has several public databases that provide such information, however, they are not connected, easy to query, or user-friendly. Additionally, the DOHMH would like to act as a go-to source for residents choosing a restaurant, providing information such as the cuisine type and locations, but also providing safety information so residents can make smart decisions. Lastly, the DOHMH would like to use this data to keep themselves and the public informed on general safety information, like which community board each restaurant belongs to, which zip codes are considered "red zones" for food and dining safety, which restaurants need further inspections and to be shut down, etc. While the DOHMH has access to this data, they need to hire database experts to make their plans a reality.

B. Proposal

The New York City Department of Health and Mental Hygiene (DOHMH) has hired our team of experts to craft a database that makes DOHMH New York City Restaurant Inspection Results, Food Poisoning Cases, Community Boards, and Liquor License data easy to form queries and draw insights. The datasets we will use contain the following information: restaurant inspection data of active restaurants in the city who have been inspected and, in some cases, have only applied for a permit but have not yet been inspected; instances of food poisoning across restaurants, restaurant dining type and cuisine information, community board addresses and assignment of restaurants to each community board, and, lastly, liquor license information. The dataset is updated daily and contains information from every inspection conducted up to three years prior. Our plan of action will involve structuring the data and developing a relational schema to make sure it is normalized and ready for querying. While the goal of our efforts is to create a database that is understandable, organized, and easy to use, the analysts at the DOHMH offices can use queries that are informative for the public, providing easily accessible

information for each restaurant's inspection history, cleanest restaurants and more. The database our team designs can be used to improve the following scenarios:

- Assist the DOHMH with inspection regimen planning, as it will help determine which restaurants will need further inspections
- It can benefit the restaurants. For instance, we can find the most common violations and violation trends. This information can inform restaurant owners of the common lapses which will be useful for the health & safety reviews that they may conduct.
- Analyzing the distribution of restaurants by cuisine in each zip code region.
- Ranking restaurants by the number of violations for public awareness
- Group restaurants with a lot of violations together, for ease of checking and managing, also for customers to be aware.
- Group restaurants with a lot of incidents together, for ease of checking and managing, also for customers to be aware.
- Building an API system for consumers who are looking for safe and clean restaurants for different cuisine.

C.Team Structure and Timeline

1. Outline

Team Members	*Tasks*
Shaun	*Developing schema, developing ETL process through python, creating business cases and queries, contributing to write up and presentation*
Olivia	*Developing Schema, Designing Schema on LucidChart, help with ETL,create business cases and queries, contributing to write up and presentation*
Trang	*Developing schema, help with ETL, creating business cases and queries, contributing to write up and presentation, data visualization*
Sunny	*Developing schema, help with ETL, creating business cases and queries,contributing to write up and presentation, data visualization*

Rohan	*Developing schema, help with ETL, creating business cases and queries,contributing to write up and presentation*

2. Timeline

Task	Week 8	Week 9	Week 10	Week 11	Week 12
Source Datasets	✓	✓			
Design Database		✓	✓		
Plot ER Diagram			✓		
ETL Process		✓	✓	✓	
Develop Queries and Analysis			✓	✓	
Metabase Dashboards				✓	✓
Build Report				✓	✓
Presentation					✓

D. Sample of Data

1. Inspection Data (https://data.cityofnewyork.us/Health/DOHMH-New-York-City-Restaurant-Inspection-Results/43nn-pn8j)

	CAMIS	DBA	BORO	BUILDING	STREET	ZIPCODE	PHONE	CUISINE DESCRIPTION	INSPECTION DATE	ACTION	...	RECORD DATE	INSPECTION TYPE
0	41452504	NIGHT°	Brooklyn	6011	7 AVENUE	11220.0	7185678878	Bottled beverages, including water, sodas, jui...	05/18/2019	Violations were cited in the following area(s).	...	07/12/2020	Cycle Inspection / Initial Inspection

2. Liquor license data (https://data.ny.gov/Economic-Development/Liquor-Authority-Current-List-of-Active-Licenses/hrvs-fxs2)

	Serial Number	County	License Type Code	License Class Code	Certificate Number	Premise Name	DBA	Premise Address	Premise Address 2	Premise City	Premise State	Premise Zip	License Issued Date	License Expiration Date
0	1326100	SUFFOLK	FP	307	570754	FARRM VINEYARD LLC	NaN	3165 MAIN RD	NaN	LAUREL	NY	11948	02/24/2020	08/24/2020

3. Food poisoning data

(https://data.cityofnewyork.us/Social-Services/food-poisoning/gjkf-etq5)

	Unique Key	Created Date	Closed Date	Agency	Agency Name	Complaint Type	Descriptor	Location Type	Incident Zip	Incident Address
0	47057081	08/01/2020 08:06:05 PM	NaN	DOHMH	Department of Health and Mental Hygiene	Food Poisoning	1 or 2	Restaurant/Bar/Deli/Bakery	10301.0	STATEN ISLAND FERRY TERMINAL

4. Community board data

(https://data.cityofnewyork.us/City-Government/Community-Board-Leadership/3gkd-ddzn and various community board websites)

	Board Number	Chair	District Manager	Postcode	Community Board	Board Address	Board Borough
0	01	Anthony Notaro, Jr.	Noah Pfefferblit	10007.0	01 MANHATTAN	1 Centre Street, New York, NY 10007	MANHATTAN
1	06	Richard Eggers	Jesus Perez	NaN	06 MANHATTAN	P.O. Box 1672, New York, NY 10159-1672	MANHATTAN

E. Database Schema & ETL process

1. Database Schema

Our database system is derived from four NYC OpenData databases: DOHMH New York City Restaurant Inspection Results, Food Poisoning, Liquor Authority Current List of Active Licenses, and Community Board Leadership. The main purpose of this database design is to inform the DOHMH and general public on food and safety information for each restaurant, in regards to both eating and drinking, and also to provide contacts (community board) for residents to report or ask questions about certain restaurants, report cases of food poisoning, and voice concerns. Based on this, it is clear the restaurants table is one of the main tables in this schema. All the relationship sets present in the schema include the restaurant camis. The database schema is as follows:

restaurants (camis, restuarant_name, address, zip, phone_number, location_type_id, borough, board_id)
violations (violation_code, violation_description, critical_flag)
license_type (license_id, license_class_code, license_type_code, description)
incidents (food_poisening_complaint_id, descriptor)
cuisines (cuisines_id, cuisine_name)
location_types (location_type_id, location_type)
community_board (board_id, chair, district_manager, board_address)
restaurants_violations (camis, violation_code, inspection_date, grade, day_of_the_week, score)
restaurants_licenses (camis, license_id, license_issued_date, license_expiration_date)
restaurants_incidents (camis, food_poisening_compaint_id, created_date, status, due_date, close_date)
restaurants_cuisine (camis, cuisines_id)

The database is made up of eleven tables, where four are relationship sets. The restaurant table contains the camis, restaurant_name, address, zip, phone, location_type_id, borough and board_id. The location_type_id and board_id are both foreign key references to other tables explained below. The primary key for this table is the camis, representing a unique indicator for each restaurant. All of the attributes in this table are fully dependent on the primary key and the table is in 2NF, making the entire table in 3NF. The violations table contains the violation_code, violation_description, and critical_flag. The primary key is the violation_code as each code represents a different violation and the critical flag attribute indicates whether or not ("Y" or "N") the violation is considered a maintenance and sanitation issue or practice that could lead to food-bourne illnesses. Again, these attributes are all relative to the primary key alone, with non-transient dependence on other attributes. The incidents table contains the food_poisening_compaint_id and descriptor. The food_poisening_compaint_id is the primary key of this table and the descriptor provides further detail on the incident or condition. As each incident is a food poisoning incident, the descriptor is either "1 or 2" or "3 or more" indicating the number of people reported to have had food poisoning in a single complaint. Each restaurant can have had more than one complaint at different times, which can be described by the restaurants_incidents table. The license_type table contains the license_id, license_class_code, licese_type code and the description, where description explains the license type being issued. Relationship sets like restaurants_violation, restaurants_licenses, and restaurants_incidents exist because each restaurant can have more than one violation, license, or incident at a time, respectively. In reference to the restaurants_violations table, restaurants have been inspected more than once, and may have different violations, grades, day_of_the_week, and scores at each inspection. The primary key of this table is the camis, violation code,and inspection date because a restaurant could have had the same violation code twice, and the inspection_date makes the combination unique. The other attributes of this table, score and code, are relevant to the entire primary key, as each inspection could have resulted in a different score and grade, even if the violation is the same. For the restaurants_licenses table, each restaurant could obtain different licenses (ex. Liquor license, beer and wine license) at a time, and will need to renew them once expired. This explains the many to many relationship between the restaurant table and license_type table. Lastly, the restaurants_incidents shows the food poisoning incidents each restaurant has had reported. This set also contains the timestamp it was reported, along with the status, due_date, and close_date. Each of these attributes are non transitively dependent on the entire primary key, camis, food_poisening_complaint_id, and incident_created_date, ensuring 3NF. Information on these three relationship sets is important for the DOHMH and residents to see how restaurants have improved/worsened in terms of health and safety, and so each party can view the exact violations for each place. Similarly to these sets, the relationship set restaurants_cuisine is a many to many relationship and exists because some restaurants are

defined using variations of more than one cuisine type, making that attribute not atomic. For example, one restaurant's cuisine type could be "Chinese/Japanese" or "Italian/Pizza." The cuisine_type table allows us to separate each cuisine and assign it an id, and the relationship set represents the relationship that allows a restaurant to be associated with more than one cuisine. This was done by separating the cuisine types into their atomic components, for instance, "Chinese/Japanese" into "Chinese" and "Japanese". This allows the database to achieve 1NF compliance by not having multiple values within an entry.

Next, the location_type table describes the different locations types that each restaurant can be categorized as, such as restaurant/bar/deli/bakery or food cart vendor. We put location_type as a separate table referenced in the restaurants table rather than an attribute on the restaurant table to ensure accuracy and to allow for future table alterations, as many restaurants are defined as "Other (explained below)." This means, in the future, the DOHMH may want to alter the location_type options to allow for a more clear understanding of location type. This way, when a restaurant is declaring its location_type, it must be a type that exists in the location_type table, this is enforced using a foreign key constraint. With this said, the location_type table consists of the location_type_id as the primary key and the location_type that it is associated with. Lastly, the community_board table contains the board_id, chair, district_manager, and board_address. The primary key of this table is the board_id, and each of the attributes in this table fully depends on the key. The primary key is referenced in the restaurants table because each restaurant belongs to a community board, where residents can go to get information on or place complaints about each restaurant. Considering this, there is a one to many relationship between these two tables, as many restaurants can belong to one community board. The full E-R Diagram of this database schema can be found in **Figure 1.**

2. ETL process

The various datasets are downloaded from their respective websites and loaded onto Python for the transformation process.

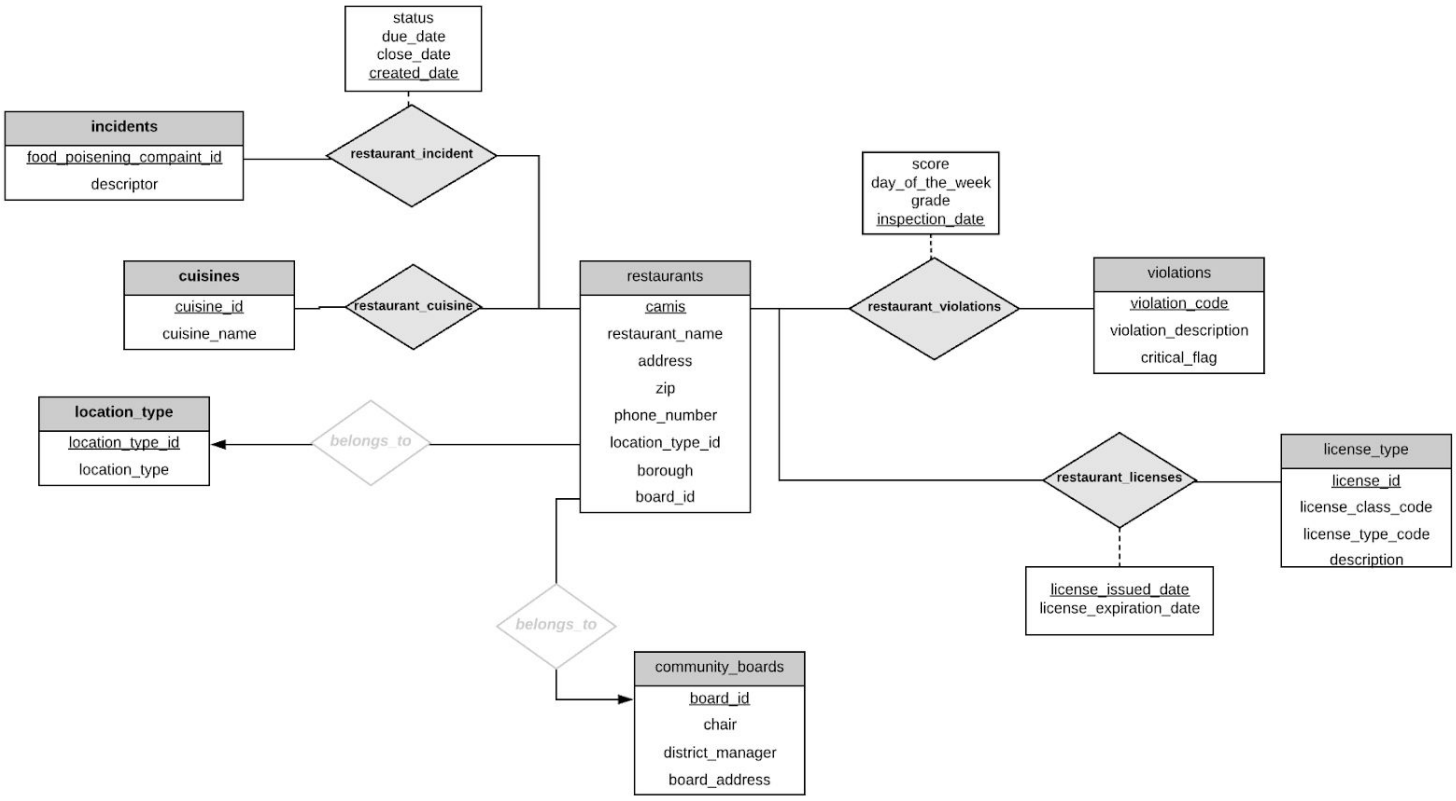
The entity and relationship sets outlined in the schema were created by joining several datasets using Python until a master dataset was created. Joins between the inspection, liquor license and food poisoning data were made on the address. Addresses on each dataset were edited to ensure the format of the address is the same across all three datasets. Addresses fields such as street and building names had to be joined together into a larger address field as the violations dataset stored addresses in such a manner.

Community board datasets were only available for Manhattan, Queens and Brooklyn. Information for Staten Island and the Bronx was taken from their respective websites and placed into a dataset that we created. All the community board datasets were then formatted to follow the same conventions, they were then joined together into a larger NYC community board dataset. This larger dataset was then joined with the other data to form our final comminbed

dataset. Pandas was then used to create the individual tables set out in our ER diagram, which were then loaded onto our SQL database. The code for this process can be found in Annex A.

Some interesting aspects of the ETL process included the need to make the cuisine type 1NF compliant. This was done by 'exploding' the cuisine data, this expanded the cuisines data by creating new rows for every '/' that was in the cuisines name. Additionally, many of the column names had to be renamed to ensure uniformity across all datasets, this ensured easy joins and avoided confusion by not having different names for the same columns.

Finally, the tables outlined in our ER diagram were created using the sqlalchemy package in python. This package allowed us to create all the SQL tables and load the data within the Pandas table into SQL.



The full ETL code and the required datasets can be found here: https://github.com/hz2657/NYC-Restaurant-public-health-project

Figure 1: E-R Diagram

Further, the code listing to create the database is in **The Appendix.**

G. Analytics Applications

1. API application for public

Through the database that we build, we can create an useful application system for the public to gain access to inspection information quickly and easily. For example, a customer can search for the restaurant name and the system will show all the violation and food poison records of the restaurant. Public will be more aware of the restaurant's reputation toward health and hygiene. Some people are allergic to certain things. By looking at food poisoning records, he/she can make decisions and feel safer eating out. Customers can also filter by cuisine and location type to pick for themselves the restaurants that have good records. The AI application can be designed like Yelp or Whitepages so that the public can navigate to information easily.
Same ideals with

2. For Internal Analysts

Analyst can find answers of the following questions by querying the database in SQL

1. Find the most common violations and violation trends.

```
SELECT violation_description, count
FROM
(SELECT violation_code, COUNT(*) FROM
restaurants_violations NATURAL JOIN violations NATURAL JOIN restaurants
GROUP BY violation_code
ORDER BY count DESC) AS FOO
NATURAL LEFT OUTER JOIN
(SELECT DISTINCT violation_code, violation_description FROM
restaurants_violations NATURAL JOIN violations
) AS LOO
ORDER BY count DESC;
```

2. Identify restaurants by the number of violations.

```
SELECT restaurant_name_violations, camis, incidents_count, rank FROM
restaurants NATURAL RIGHT JOIN (
SELECT *
FROM
(SELECT camis, count(*) as incidents_count, DENSE_RANK () OVER (ORDER BY count(*) DESC) as
rank
FROM
(restaurants JOIN restaurants_incidents USING(camis)) AS FOO
GROUP BY camis) AS LOO
WHERE rank <=20
) AS LUU
ORDER BY rank;
```

3. Identify if there are more violations in weekdays or weekends.

```
SELECT day_of_the_week, COUNT(*) AS num
FROM restaurants_violations
```

```
GROUP BY day_of_the_week;
```

4. Which cuisine tends to have more violations?

```
SELECT c.cuisine_name, count(rv.violation_id) as violations
                    ,rank() over (order by count(rv.violation_id) desc) as s_rank
                    from restaurants_violations as rv
                    join restaurants_cuisine as rc on rv.camis=rc.camis
                    join cuisines as c on c.cuisines_id = rc.cuisines_id
GROUP BY c.cuisine_name;
```

5. Group restaurants with a lot of violations together.

```
Select restaurant_name_violations, violations,
CASE
   WHEN violations > 10  THEN 'The restaurant has many violations'
   WHEN violations < 10 AND violations >5 and violations =5 THEN 'The restaurant has some violations'
         WHEN violations = 0 THEN 'The restaurant does not have any violation'
         ELSE 'The restaurants has a few violations'
END AS Violations_group
FROM
(select r.restaurant_name_violations, count(rv.violation_code) as violations
         from restaurants_violations as rv
         join restaurants as r on r.camis=rv.camis
group by r.restaurant_name_violations) as foo
order by violations_group;
```

6. What are the most common liquor license types for each of the boroughs?

```
SELECT borough_name, license_type_code, COUNT(license_type_code)
FROM (
SELECT DISTINCT(camis), license_type_code, borough_name
FROM license_type
NATURAL JOIN restaurants_licenses
NATURAL JOIN restaurants) as sub
GROUP BY borough_name, license_type_code
ORDER BY borough_name, COUNT DESC;
```

7. Ranking restaurant cuisines by the number of incidents.

```
SELECT cuisine_name, comp_count
FROM
(SELECT cuisine_name, COUNT(food_poisoning_complaint_id) as comp_count, rank() over (order by
COUNT(food_poisoning_complaint_id) desc)
FROM restaurants_incidents as ri
NATURAL JOIN restaurants_cuisine AS rc
GROUP BY cuisine_name
ORDER BY comp_count DESC) as sub
WHERE rank <= 5;
```

8. Identify the number of restaurant location types within each borough.

```
SELECT borough_name, location_type, count(CAMIS)
FROM restaurants
NATURAL JOIN location_types
GROUP BY borough_name, location_type
ORDER BY borough_name, count DESC;
```

9. Identify zip codes with most violations.

```
SELECT zip, count(violation_code) as violations, rank() over (order by count(violation_code) desc) as rank
FROM restaurants
NATURAL JOIN restaurants_violations
GROUP BY zip;
```

10. Group restaurants with similar number of incidents together.

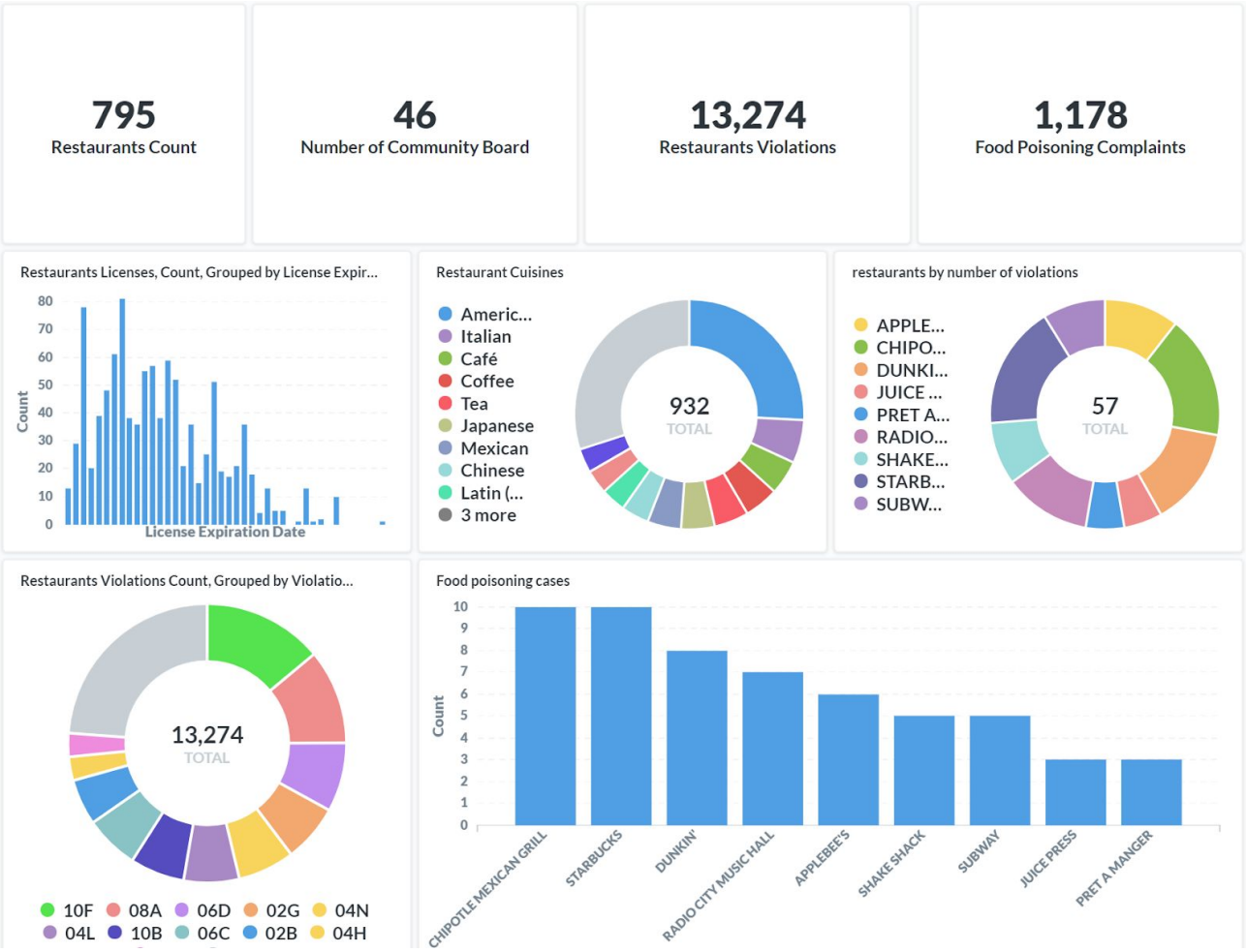
```
Select restaurant_name_violations, incidents,
CASE
   WHEN incidents > 10  THEN 'The restaurant has many food poisoning incidents'
   WHEN incidents < 10 AND incidents >5 and incidents =5 THEN 'The restaurant has some food
poisoning incidents'
        WHEN incidents = 0 THEN 'The restaurant does not have any food poisoning incident'
        ELSE 'The restaurants has a few food poisoning incidents'
END AS incidents_group
FROM
(select r.restaurant_name_violations, count(ri.food_poisoning_complaint_id) as incidents
        from restaurants_incidents as ri
        join restaurants as r on r.camis=ri.camis
group by r.restaurant_name_violations) as foo
order by incidents_group;
```

3. For reports for "C" level management

C level officers can find information from the reports made by analysts to make important business decisions. Analysts can provide reports to managers via sharing or exporting dashboards from business intelligence tools such as Metabase,Tableau or Power BI. Through the reports, managers can make business decisions such as:

1. Identify which restaurants will need further inspections and take actions
2. Decide actions plans to promote public healths and to reduce number of food poisoning

H. Metabase Dashboard



This dashboard is shown in demo, it contains the following information:

1. The number of restaurants, community board, restaurants violations, food poisoning complaints
2. Restaurants whose licenses will expire over time
3. The distribution of restaurants' cuisines in NYC
4. Restaurants with the most number of inspection violations, as shown in the pie chart, AppleBee has the most violations.
5. The most frequent restaurants violations, as shown in the pie chart, they are 10F, 08A, 06D, referring to the referenced table, these violations are:
    - Non food contact surface improperly constructed
    - Facility not vermin proof. Conditions to attract vermin and allow them to exist.
    - Food contact surface not properly washed

6. Restaurants with the most number of reported food poisoning cases, as shown in the bar chart, Chipotle and Starbucks have the most cases

ANNEX A

```
#import packages

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sqlalchemy import create_engine

#download all datasets (inspection results, liquor license, food poisoning, all community boards)

df = pd.read_csv("/Users/user/Downloads/DOHMH_New_York_City_Restaurant_Inspection_Results.csv")
liquor=pd.read_csv('/Users/user/Downloads/Liquor_Authority_Current_List_of_Active_Licenses.csv')
food_poisoning=pd.read_csv('/Users/user/Downloads/food_poisoning.csv')
man_board = pd.read_csv("/Users/user/Downloads/Community_Board_Leadership.csv")
bkn_board = pd.read_csv("/Users/user/Downloads/Community_Board_Contact_List.csv")
qns_board = pd.read_csv("/Users/user/Downloads/Queens_Community_Board_District_Managers_and_Chairs.csv")
brx_si_board = pd.read_csv("/Users/user/Downloads/bronx_statenisland_boards.csv")
df_0 = pd.read_csv("/Users/user/Downloads/NYSLA_License_and_Permit_Code_Classification_Categories.csv")

# create an 'address' column that joins 'building' and 'street'
cols = ['BUILDING', 'STREET']
df['Address'] = df[cols].apply(lambda row: ' '.join(row.values.astype(str)), axis=1)

inspection = df[['CAMIS','DBA','BORO', 'Address', 'ZIPCODE', 'PHONE', 'CUISINE DESCRIPTION', 'INSPECTION DATE', 'VIOLATION
CODE', 'VIOLATION DESCRIPTION', 'CRITICAL FLAG', 'SCORE', 'GRADE', 'Latitude', 'Longitude', 'Community Board']]
inspection = inspection[~inspection['DBA'].isnull()]

#merge the liquor, violation and food_poisoning data
liquor=liquor.drop('Georeference',axis=1)
df1=pd.merge(inspection,liquor,left_on='Address',right_on='Premise Address',how="inner")
df2=pd.merge(df1,food_poisoning,left_on='Address',right_on='Incident Address',how="inner")
df2['Community Board'] =df2['Community Board_y']

#select relavent columns
df2=df2[['CAMIS',
 'DBA_x',
 'BORO',
 'Address',
 'ZIPCODE',
 'PHONE',
 'CUISINE DESCRIPTION',
 'INSPECTION DATE',
 'VIOLATION CODE',
 'VIOLATION DESCRIPTION',
 'CRITICAL FLAG',
 'SCORE',
 'GRADE',
 'Latitude_x',
 'Longitude_x',
 'Community Board',
 'Serial Number',
 'County',
```

```
'License Type Code',
'License Class Code',
'Certificate Number',
'Premise Name',
'DBA_y',
'License Issued Date',
'License Expiration Date',
'Method of Operation',
'Unique Key',
'Created Date',
'Closed Date',
'Agency',
'Agency Name',
'Complaint Type',
'Descriptor',
'Status',
'Due Date',
'Resolution Action Updated Date',
'Borough',
'Park Facility Name',
'Park Borough',
'Latitude_y',
'Longitude_y',
'Location',
'Location Type']]

#ensure all the community board datasets have the same formating and are joined together
brx_si_board = brx_si_board[:15]
man_board['Borough'] = man_board['Borough'].fillna('MANHATTAN')

qns_board['Chair'] = qns_board['CB Chair - First Name'].str.cat(qns_board['CB Chair - Last Name'],sep=" ")
qns_board['District Manager'] = qns_board['District Manager - First Name'].str.cat(qns_board['District Manager - Last Name'],sep=" ")
bkn_board['Board Number'] = bkn_board['Board Number'].str.extract("([-+]?\d*\.\d+|[-+]?\d+)").astype(int)

man_board['Board Number'] = man_board['Community Board']
man_board['Address'] = man_board['Address 1']

qns_board['Board Number'] = qns_board['Board']

brx_si_board['Board Number'] = brx_si_board['Community Board']
brx_si_board['District Manager'] = brx_si_board['District Manager ']
brx_si_board['Address'] = brx_si_board['Address 1']

man_board_trunc = man_board[['Board Number','Chair','District Manager','Address','Postcode','Borough']]
bkn_board_trunc = bkn_board[['Board Number','Chair','District Manager','Address','Postcode','Borough']]
qns_board_trunc = qns_board[['Board Number','Chair','District Manager','Address','Postcode','Borough']]
brx_si_board_trunc = brx_si_board[['Board Number','Chair','District Manager','Address','Postcode','Borough']]

board_list = man_board_trunc.append([bkn_board_trunc], ignore_index = True
                    ).append([qns_board_trunc], ignore_index = True
                         ).append([brx_si_board_trunc], ignore_index = True)

board_list['Borough'] = board_list['Borough'].str.upper()
board_list['Board Number'] = board_list['Board Number'].astype(float).astype(int)
board_list['Board Number'] = board_list['Board Number'].apply('{:0>2}'.format).astype(str)
board_list['Board Number'] = board_list['Board Number'].astype(str)
board_list['Community Board'] = board_list['Board Number'] + ' ' +  board_list['Borough']
```

```
board_list['Board Address'] = board_list['Address']
board_list['Board Borough'] = board_list['Borough']

board_list = board_list.drop(columns = ['Address' ,'Borough'])


#merge community boards with the other dataset
df = pd.merge(df2, board_list, on = 'Community Board', how = 'left')

#connect to DATABASE
conn_url = 'postgresql://postgres:123@localhost/SQL Project'
engine = create_engine(conn_url)
connection = engine.connect()
df=pd.read_csv("/Users/user/Downloads/insepection_7.csv")

#rename df, this ensures column names follow the naming convention set out in our ER diagram

df['camis'] = df['CAMIS']
df['restaurant_name_violations'] = df['DBA_x']
df['premise_name_liqour_license'] = df['Premise Name']
df['zip'] = df['ZIPCODE']
df['board_address'] = df['Board Address']
df['phone_number'] = df['PHONE']
df['cuisine_name'] = df['CUISINE.DESCRIPTION']
df['board_id'] = df['Community Board']
df['borough_name'] = df['Borough']
df['district_manager'] = df['District Manager']
df['violation_code'] = df['VIOLATION.CODE']
df['violation_description'] = df['VIOLATION.DESCRIPTION']
df['critical_flag'] = df['CRITICAL.FLAG']
df['license_class_code'] = df['License Class Code']
df['license_type_code'] = df['License Type Code']
df['address'] = df['Address']
df['score'] = df['SCORE']
df['grade'] = df['GRADE']
df['inspection_date'] = df['INSPECTION.DATE']
df['license_expiration_date'] = df['License Expiration Date']
df['complaint_type'] = df['Complaint Type']
df['descriptor'] = df['Descriptor']
df['food_poisoning_complaint_id'] = df['Unique Key']
df['close_date'] = df['Closed Date']
df['status'] = df['Status']
df['due_date'] = df['Due Date']
df['location_type'] = df['Location Type']
df['chair'] = df['Chair']
df['license_issued_date'] = df['License Issued Date']
df['incident_created_date'] = df['Created Date']
df['license_description'] = df['Method of Operation']
#find day of the week from inspection date
df['day_of_the_week'] = pd.to_datetime(df['inspection_date']).dt.dayofweek

#create tables
stmt = """

CREATE TABLE location_types (
    location_type_id  integer NOT NULL,
```

```
    location_type       varchar (100) UNIQUE,
    PRIMARY KEY(location_type_id)
);

CREATE TABLE community_board (
    board_id        varchar(30) NOT NULL,
    chair           varchar(70),
    district_manager  varchar(70),
    board_address   varchar(100),
    PRIMARY KEY(board_id)
);

CREATE TABLE restaurants (
    camis               integer,
    restaurant_name_violations    varchar(100) NOT NULL,
    location_type       varchar (100),
    address             varchar(200) NOT NULL,
    borough_name    varchar(15) NOT NULL,
    zip                 integer,
    phone_number        varchar (12),
    board_id        varchar(30) NOT NULL,
    PRIMARY KEY (camis),
    FOREIGN KEY(location_type) REFERENCES location_types(location_type),
    FOREIGN KEY(board_id) REFERENCES community_board(board_id)
);

CREATE TABLE violations (
    violation_code        varchar(4) NOT NULL,
    violation_description text,
    critical_flag         varchar(1) CHECK (critical_flag IN ('Y', 'N')),
    PRIMARY KEY (violation_code)
);

CREATE TABLE license_type (
    license_id      integer,
    license_class_code     char(3) NOT NULL,
    license_type_code varchar(2) NOT NULL,
    license_description text,
    PRIMARY KEY (license_id)
);

CREATE TABLE cuisines (
    cuisine_id    integer,
    cuisine_name   varchar(80) NOT NULL,
    PRIMARY KEY (cuisine_id)
);


CREATE TABLE incidents (
    food_poisoning_complaint_id    integer,
    descriptor     varchar(100) CHECK (descriptor IN ('1 or 2', '3 or More')),
    PRIMARY KEY (food_poisoning_complaint_id)
);

CREATE TABLE restaurants_violations (
    camis             integer,
    violation_code    varchar(4),
```

```
    grade            varchar(1),
    score            integer,
    inspection_date   date,
    day_of_the_week integer,
    PRIMARY KEY (camis, violation_code, inspection_date),
    FOREIGN KEY (camis) REFERENCES restaurants(camis),
    FOREIGN KEY (violation_code) REFERENCES violations(violation_code)
);

CREATE TABLE restaurants_licenses(
    camis          integer,
    license_id      integer,
    license_issued_date      date,
    license_expiration_date date NOT NULL,
    PRIMARY KEY (camis, license_id,license_issued_date),
    FOREIGN KEY (camis) REFERENCES restaurants(camis),
    FOREIGN KEY (license_id) REFERENCES license_type(license_id)
);

CREATE TABLE restaurants_incidents(
    camis       integer,
    food_poisoning_complaint_id integer,
    incident_created_date varchar(50),
    status      varchar(20) NOT NULL,
    due_date    date,
    close_date  date,
    PRIMARY KEY (camis, food_poisoning_complaint_id, incident_created_date),
    FOREIGN KEY (camis) REFERENCES restaurants(camis),
    FOREIGN KEY (food_poisoning_complaint_id) REFERENCES incidents(food_poisoning_complaint_id)
);

CREATE TABLE restaurants_cuisine(
    camis        integer,
    cuisine_id    integer,
    PRIMARY KEY (camis, cuisine_id),
    FOREIGN KEY (camis) REFERENCES restaurants(camis),
    FOREIGN KEY (cuisine_id) REFERENCES cuisines(cuisine_id)
);

"""
#execute table creation
results = connection.execute(stmt)

# format data in tables set out into our ER diagram so that it can be loaded into the database
restaurants=df[['camis', 'restaurant_name_violations','location_type', 'address' ,'borough_name','zip','phone_number', 'board_id']]
restaurants= restaurants.drop_duplicates(subset='camis').reset_index()
restaurants= restaurants[['camis', 'restaurant_name_violations' ,'location_type' ,'address','borough_name','zip','phone_number','board_id']]

violations = df[['violation_code','violation_description','critical_flag']]
violations = violations.drop_duplicates(subset='violation_code').dropna(subset=['violation_code']).reset_index()
violations = violations[['violation_code','violation_description','critical_flag']]

license_type_wip = df[['license_class_code','license_type_code']]
license_type_wip = license_type_wip.drop_duplicates().reset_index()
license_type_wip['license_id'] = license_type_wip.index
license_type_wip = license_type_wip[['license_id', 'license_class_code','license_type_code']]
```

```
license_type = pd.merge(license_type_wip, df_0,  how='left', left_on=['license_class_code','license_type_code'], right_on = ['License Class Code','License Type Code'])
license_type['license_description'] = license_type['License Description']
license_type = license_type[['license_id','license_class_code','license_type_code','license_description']]

incidents = df[['food_poisoning_complaint_id','descriptor']].drop_duplicates().reset_index()
incidents = incidents[['food_poisoning_complaint_id', 'descriptor']]

location_types = pd.DataFrame(df['location_type'].drop_duplicates()).reset_index()
location_types['location_type_id'] = location_types.index
location_types=location_types[['location_type_id','location_type']]

new_df = pd.DataFrame(df.cuisine_name.str.split('/').tolist(), index=df.camis).stack()
new_df = new_df.reset_index([0, 'camis']).drop_duplicates().reset_index(drop = True)
new_df.columns = ['camis', 'cuisine_name']

cuisines = new_df[['cuisine_name']].drop_duplicates().reset_index(drop = True)
cuisines['cuisine_id'] = cuisines.index
cuisines = cuisines[['cuisine_id','cuisine_name']]

new_df1 = pd.merge(new_df , cuisines, on = 'cuisine_name', how = 'inner')

restaurants_cuisine = pd.merge(restaurants , new_df1, on = 'camis', how = 'right')
restaurants_cuisine = restaurants_cuisine[['camis','cuisine_id']]

restaurants_violations_wip =
df[['camis','violation_code','score','grade','inspection_date','day_of_the_week']].drop_duplicates().dropna(subset=['violation_code']).reset_index()
restaurants_violations = restaurants_violations_wip[['camis','violation_code','score','grade','inspection_date','day_of_the_week']]

restaurants_licenses_wip=df[['camis', 'license_class_code','license_type_code' ,'license_issued_date',
'license_expiration_date']].drop_duplicates().reset_index()
restaurants_licenses_wip=pd.merge(restaurants_licenses_wip, license_type, on = ['license_class_code','license_type_code']).drop(columns =
['license_class_code','license_type_code'])
restaurants_licenses = restaurants_licenses_wip[['camis','license_id','license_issued_date','license_expiration_date']]

restaurants_incidents=df[['camis','incident_created_date' ,'food_poisoning_complaint_id', 'status', 'due_date', 'close_date']]
restaurants_incidents=restaurants_incidents.drop_duplicates().reset_index()
restaurants_incidents = restaurants_incidents[['camis','incident_created_date', 'food_poisoning_complaint_id', 'status', 'due_date', 'close_date']]

community_board = df[['board_id', 'chair','district_manager', 'board_address']].drop_duplicates().reset_index()
community_board['district_manager'] = community_board['district_manager'].str.split('  ').str[0]
community_board = community_board[['board_id', 'chair','district_manager', 'board_address']]

#load data into database
license_type.to_sql(name='license_type', con=engine, if_exists='append', index=False)
location_types.to_sql(name='location_types', con=engine, if_exists='append', index=False)
community_board.to_sql(name='community_board', con=engine, if_exists='append', index=False)
restaurants.to_sql(name='restaurants', con=engine, if_exists='append', index=False)
violations.to_sql(name='violations', con=engine, if_exists='append', index=False)
cuisines.to_sql(name='cuisines', con=engine, if_exists='append', index=False)
incidents.to_sql(name='incidents', con=engine, if_exists='append', index=False)
restaurants_cuisine.to_sql(name='restaurants_cuisine', con=engine, if_exists='append', index=False)
restaurants_violations.to_sql(name='restaurants_violations', con=engine, if_exists='append', index=False)
restaurants_licenses.to_sql(name='restaurants_licenses', con=engine, if_exists='append', index=False)
restaurants_incidents.to_sql(name='restaurants_incidents', con=engine, if_exists='append', index=False)
```