Technical Report: Final Project DS 5110: Introduction to Data Management and Processing

Team Members:
Cameron Jester
Matt Ferri
Khoury College of Computer Sciences
Data Science Program
jester.c@northeastern.edu, ferri.m@northeastern.edu

December 10, 2024

Contents

1 Introduction

Provide a brief introduction to the project, including the background, objectives, and scope.

This project aims to create a smart solution for tracking, monitoring, and visualizing bar lines in major cities, including Boston, which are known for being vibrant hubs of bars and restaurants. Given the vast amount of choices available, it can be difficult to pick the establishment with the shortest wait time to gain entry without being physically present. The problem at hand is how to provide users with reliable line estimates so they can make informed decisions when selecting where to go.

While programs such as Google provide a rough idea of wait times for certain establishments, our project aims to utilizes user-input to estimate wait times for bars and restaurants in the Boston area. Utilizing user-provided data will ensure a high accuracy and also provide users with the feeling like they are contributing to the positive experience of fellow peers.

This technology would not only be beneficial for users, but bar and restaurant owners as well as it allows for accurate forecasting of attendance to their establishments. Forecasting will help owners determine which dates and times are best for promotions, optimize staffing, and project profits with confidence. Our smart solution aims to enhance the efficiency of bars and increase consumer satisfaction when going out.

2 Literature Review

As mentioned, Google has a wait-time feature that is present on many places of business. These wait times are made by aggregating data from users who have opted to share their phone locations with Google. Additionally, data is not made available in a central location, meaning users would need to conduct several Google searches to compile a list of bars and their associated wait times. This methodology differs from the project at hand as the final project would aggregated data directly reported from users as opposed to taking geolocation tracking from their phones. Furthermore, this project would be exclusively applied to bars in Boston so users would have a central location for all of the information needed to make a decision.

A similar app called TablesReady is available online for queue management, but is primarily used for restaurant owners as opposed to being focused on those who are actually attending bars.

3 Methodology

As a database project, rather than a machine learning analysis type project, our methodology section talks about the dataset, minor data preprocessing, time complexity analysis, and analysis of the data into 10 minute intervals (explained in detail below). In Section 4: Results we dive deeper into database creation, SQL methods to interact with the database, and python connectors that access the data.

3.1 Data Collection

Given the lack of public data available for this kind of smart solution, it was decided to build a dataset from scratch. Careful consideration was used to determine how data would be simulated and which features would be critical to the scope of the project. Using Python coded to simulate data, the process began by creating a list of 500 fake names and 10 Boston-area bars, along with randomly generated demographic data such as age (21-60 years old), gender (male, female, N/A) , and zip code (6 zip-codes in and around the South Boston area) for each individual simulated user. User information was then stored in a dictionary, which is then written to a CSV file, capturing user ID, age, gender, and zip code. A similar process is applied to bars, where each bar is assigned a unique ID, flow rate (a constant which reflects how many patrons exit a bar in a 10 minute period), and a specific zip code, saved into a dictionary and then saved to a csv. Following this, rows of data were created to simulate user visits to bars with timestamps, the flow rate of each bar, and user estimates for wait times. The data is stored and sorted in reverse chronological order, before being saved in another CSV file which acts as the master file. This structure supports the methodologies used for storing data in SQL databases as well as accessing data for visualizations.

3.2 Data Preprocessing

Given the nature of simulating data, little preprocessing was needed to ensure the data was ready to be used for analysis. For clarity, the gender of users was changed from 'male', 'female', and 'NA' to 0,1,2 for computational efficiency. While there was minimal data preprocessing to do after simulating the data, it was up to us to design the simulated data structure in such a way as to be inherently preprocessed. In that regard we accomplished the preprocessing by being intentional with our data design decisions.

3.3 Analysis Techniques

The main focus of our data analysis was our efforts to group the "Line Estimates" submitted by users into time buckets. We decided it would be useful to know aggregated statistics of estimates over the course of a small time bucket, rather than arbitrary reviews across all night. Code was then written which aggregates bar guest estimates in 10-minute intervals for each bar, starting with a minimum timestamp for each group of bar data. Using a function that adds 10-minute increments to time values, it iterates through grouped bar data by bar id, calculates the mean guest estimate within each interval, and adjusts for a constant flow rate of bar traffic. If the difference between the mean guest estimate and the flow rate is negative, it sets the value to zero to avoid negative line lengths. The aggregated data for each bar is stored in a dictionary, capturing the start of each interval, the mean guest estimate, flow rate, and calculated line length to be used for data visualizations.

3.4 Time Complexity Analysis

Another important focus of data analysis was our time complexity analysis. There were three key operations that we researched the time complexity of:

1. Data Loading into and Data Querying from MySQL – full tables

   (a) This had a time complexity of: O(n*m)
   (b) (n = rows, m = columns) This operation corresponds to virtually linear time. We are appending into an existing database and there is only an insertion operation into a tabular format.

2. Subquerying Data from MySQL – querying based on criteria
   (a) O(log(n) + r*m)
   (b) (n = rows, r = matching rows, m = columns) This operation is log OR linear time, depending on the number of matching rows. MySQL indexes data using binary trees and therefore reduces the search to log(n) compared to a non-indexed database that would take O(n) to search. It then costs r*m to retrieve the data, where r is the matching rows and m is the columns.

3. Aggregations – analysis of data
   (a) O(log(n) + r)
   (b) (n = rows, r = matching rows) This is greater than log time, but less than linear time. Time complexity approaches linear as matching rows increases. It takes log(n) to search for matching values, then 1 operation per matching row, or r.

4 Results

In Results, we will talk about the database we created and the configuration steps that went into creating and loading it. We utilized a site named ”freemysqlhosting.net” to create a sql database that can be remotely connected to. We used a package named mysqlconnector in python to connect to this database, and the configuration steps looked like this in python (Figure 1).

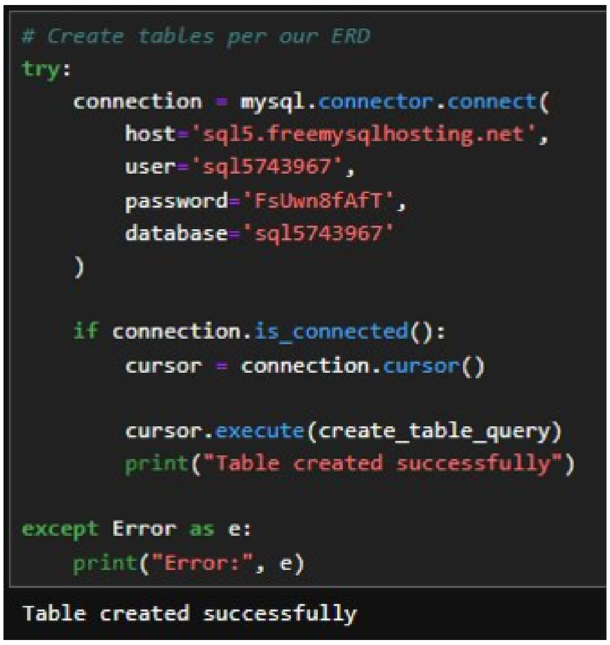
Next, we used SQL queries across the mysqlconnector package in python to create the tables. The SQL queries for creating the tables, as well as setting the fields and their appropriate constraints are in (Figure 2). The ERD for our relational database design is in (Figure 3).

And lastly, we used our python connector to load the simulated data from section 3 into this database. A free tool the MySQL server offers is called PHPMyAdmin which allowed us to view the tables and fields in a clean and easy to use format (Figure 4).

5 Discussion

Enabling patrons to have a real-time queuing app allows for a more streamlined decision process when selecting a bar to go to. A user can both submit estimates of the line length to contribute to the app, while also using the app to see line lengths near them. For users who desire a shorter line this app could greatly benefit them; either because they are in a rush, the weather is bad, they have a disability and cannot wait in a line, etc.

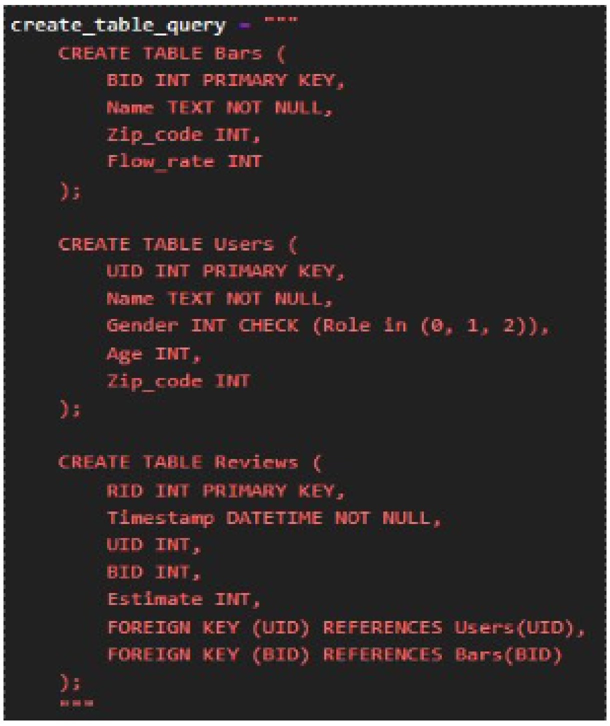
For bar owners, such metrics will provide crucial insights into the trends of their business including wait times, and demographics of customers such as age, gender, and where they are coming from through zip code. From a business perspective, this app will increase profits by analyzing trends and forecasting profitable sales tactics.



```
# Create tables per our ERD
try:
    connection = mysql.connector.connect(
        host='sql5.freemysqlhosting.net',
        user='sql5743967',
        password='FsUwn8fAfT',
        database='sql5743967'
    )

    if connection.is_connected():
        cursor = connection.cursor()

        cursor.execute(create_table_query)
        print("Table created successfully")

except Error as e:
    print("Error:", e)
```

```
Table created successfully
```

Figure 1: Python mysqlconnector Package



```
create_table_query = """
    CREATE TABLE Bars (
        BID INT PRIMARY KEY,
        Name TEXT NOT NULL,
        Zip_code INT,
        Flow_rate INT
    );

    CREATE TABLE Users (
        UID INT PRIMARY KEY,
        Name TEXT NOT NULL,
        Gender INT CHECK (Role in (0, 1, 2)),
        Age INT,
        Zip_code INT
    );

    CREATE TABLE Reviews (
        RID INT PRIMARY KEY,
        Timestamp DATETIME NOT NULL,
        UID INT,
        BID INT,
        Estimate INT,
        FOREIGN KEY (UID) REFERENCES Users(UID),
        FOREIGN KEY (BID) REFERENCES Bars(BID)
    );
    """
```

Figure 2: SQL Table Creation Queries

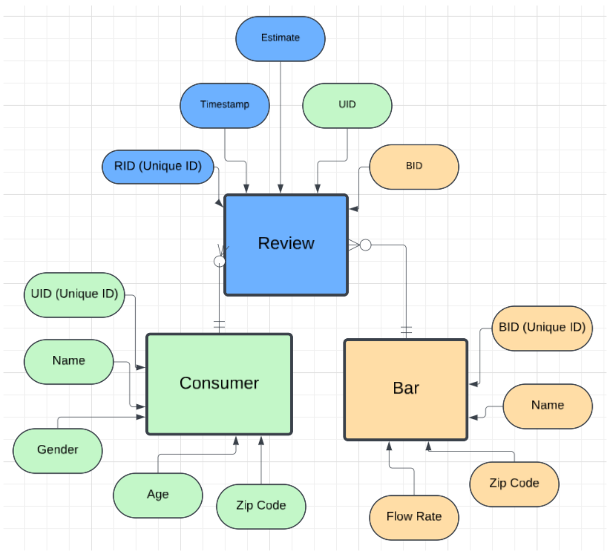
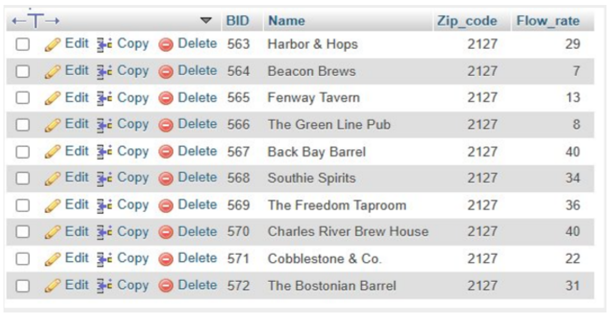


Figure 3: Database ERD



	BID	Name	Zip_code	Flow_rate
Edit Copy Delete	563	Harbor & Hops	2127	29
Edit Copy Delete	564	Beacon Brews	2127	7
Edit Copy Delete	565	Fenway Tavern	2127	13
Edit Copy Delete	566	The Green Line Pub	2127	8
Edit Copy Delete	567	Back Bay Barrel	2127	40
Edit Copy Delete	568	Southie Spirits	2127	34
Edit Copy Delete	569	The Freedom Taproom	2127	36
Edit Copy Delete	570	Charles River Brew House	2127	40
Edit Copy Delete	571	Cobblestone & Co.	2127	22
Edit Copy Delete	572	The Bostonian Barrel	2127	31

Figure 4: PHP MyAdmin - Allowing us to view our newly created tables.

Overall, this app may be used and adapted to fit a wider-audience of businesses outside of the food and beverage space. Real-time queuing technology generated from users may be integrated into daily activities from grocery stores, car dealerships, postal offices, and beyond to efficiently educate users on how and where to spend their time.

6 Conclusion

The biggest limitation faced during this project was the short schedule given to complete all aspects of this project. Having to create a mock-up of the back-end and front-end of this project required a thorough planning period that took a bit longer to accomplish than originally accounted for. An additional month or so would be needed to bring this application to the next stage of development. While we did not develop a front end for the application, the visualizations we placed in the appendix show what some common charts would look like.

Given the complexity and importance of this kind of project, there are areas of improvement for future work. Future work would include adding a bouncer estimate, which would further improve the accuracy of line estimations from users. This estimate from the bouncer of the bar would inherently be more accurate than patrons who don't care as much about accurately reporting the line and could be weighted more heavily. Similarly, this project would be created into a phone app to make it as easy as possible for bar-goers to see estimated wait times.

7 References

Google. (n.d.). *Edit your Business Profile on Google.* Retrieved from https://support.google.com /business/answer/ 6263531?hl=en Stack Overflow. (n.d.). *Questions tagged ipywidgets.* Retrieved from https://stackoverflow.com/questions/tagged/ ipywidgets Stack Overflow. (n.d.). *How do I connect to a MySQL database in Python?* Retrieved from https://stackover flow.com/ questions/ 372885/how-do-i-connect-to-a-mysql- database-in-python

References

A Appendix A: Code

Include any relevant code used in the project.

```
people = {}

zip_codes = ['02115', '03126', '02129', '02103', '02228', '02127']
genders = ['0', '1', '2'] #male, #female, #na


i = 0
for name in fake_names:
    i = i + 1
    gender = random.choices(genders, weights=[48, 48, 4])[0]
    zip = random.choice(zip_codes)
    people[name] = { "user_id": i ,"age": random.randint(21, 60), "zip_code": zip, "gender": gender}
print(people)

fieldnames = ['Name', 'User ID', 'Age', 'Gender', 'Zip Code']
with open('consumer_data.csv', 'w', newline='') as csvfile:
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
    writer.writeheader()
    for row,data in people.items():
        name, age, gender, zip_code = row, data['age'], data['gender'], data['zip_code']
        writer.writerow({
            'Name':name,
            "User ID": data['user_id'],
            'Age': data['age'],
            'Gender': data['gender'],
            'Zip Code': data['zip_code'],
        })



bars = {}

for name in fake_boston_bars:
    i = i + 1
    bars[name] = { "bar_id": i, "flow_rate": random.randint(5,40) , "bar_zipcode" :'02127'}
print(bars)

bar_fieldnames = ['Name', 'Bar ID', 'Flow Rate', 'Zip Code']
with open('bar_data.csv', 'w', newline='') as csvfile:
    writer = csv.DictWriter(csvfile, fieldnames=bar_fieldnames)
    writer.writeheader()
    for row,data in bars.items():
        name, bar_id, flow_rate, zip_code = row, data['bar_id'], data['flow_rate'], data['bar_zipcode']
        writer.writerow({
            'Name':name,
            'Bar ID': data['bar_id'],
            'Flow Rate': data['flow_rate'],
            'Zip Code': data['bar_zipcode'],
        })
```

Listing 1: Code to create csv files for users and bars from fake name lists

```


def tenMinAgg(time):
  new_time = time + pd.Timedelta(minutes = 10)
  new_time_pd = pd.to_datetime(new_time)
  return new_time_pd



bar_groups = df.groupby('bar_id')

individ_bar_agg = {}


```

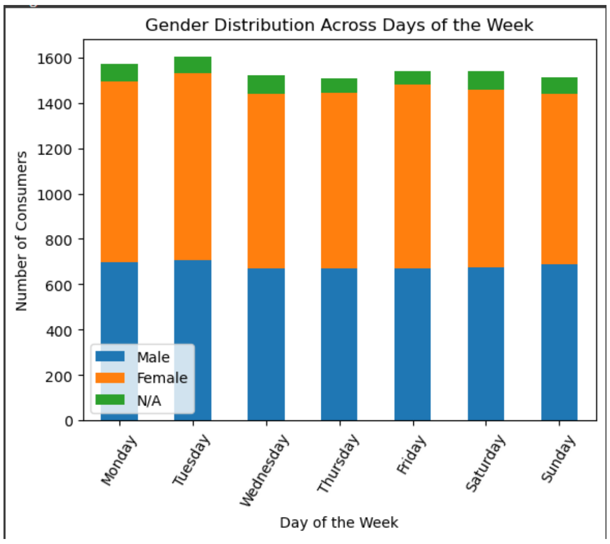


Figure 5: For Bar Use: Gender Distribution Across Days of the Week

```

for bar_id, bar_group in bar_groups:
  first_time = bar_group['timestamp'].min()
  guest_estimate_agg = []
  constant_flow_rate = bar_group['flow_rate'].iloc[0]

  for i in range(len(bar_group['timestamp'])):
    first_time_dt = pd.to_datetime(first_time)
    last_time_dt = tenMinAgg(first_time_dt)

    agg_df = bar_group[(bar_group['timestamp']>=first_time_dt) & (
   bar_group['timestamp']< last_time_dt)]
    guest_estimate_agg_calc  = agg_df['guest_estimate'].mean()
    estimate_minus_fr = guest_estimate_agg_calc - constant_flow_rate
    if estimate_minus_fr < 0: #doing this so we don't have negative
   people in line.
        estimate_minus_fr = 0
    else:
          estimate_minus_fr = estimate_minus_fr
    guest_estimate_agg.append({'interval_start': first_time_dt, '
   mean_guest_estimate_10min': guest_estimate_agg_calc, 'flow_rate':
   constant_flow_rate, 'line_length': estimate_minus_fr})
    first_time = last_time_dt

  individ_bar_agg[bar_id] = guest_estimate_agg
```

Listing 2: Code to aggregate reviews on 10 minute intervals

B Appendix B: Additional Figures

Include any additional figures or tables that support the analysis.

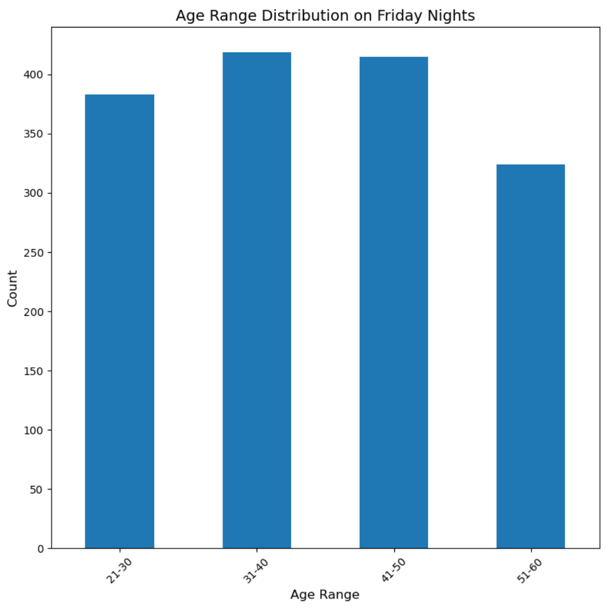


Figure 6: For Bar Use: Age Distribution

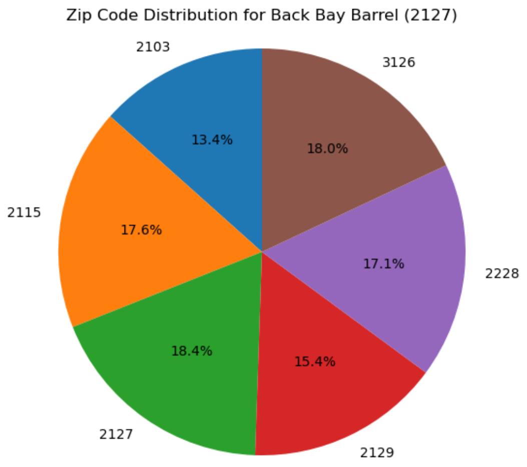


Figure 7: For Bar Use: Where Bar Patrons are Coming From

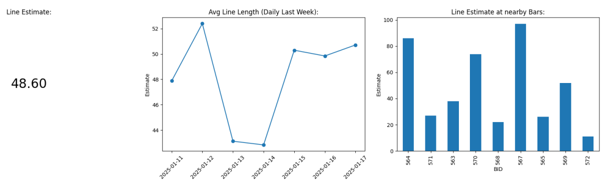


Figure 8: Example Application UI for Users