1. Overview

1.1 Summary

Our project is to create a user interface which would allow various users to see and analyze New York Yellow Cab trends from data consisting of records from early 2005 to 2022. Our project is meant to help the drivers as well as passengers decide if operating or taking a taxi is the best possible option from location A to location B at specific or random time intervals or particular periods across a calendar year. The user has the flexibility to choose parameters like time, location, payment method, trip fare, etc and view their changes over time. This information helps many groups, whether it be a taxi driver, a tourist or tech giants like Google Pay looking for their user demographic, to view meaningful trends in the various parameters.

1.2 Motivation

While the data about Yellow Cabs in NYC is easily available on the internet, there is no computerized application which acts as a one-stop for everything related to trip data based on Yellow Cabs. Building an effective system that could manage such data requires significant time and investment. For an average user, looking at the Excel files is counterproductive as they can see what the data consists of, but deriving useful conclusions for over a million of records requires a useful tool that can comprehend, visualize data and give users a reliable architecture that could be accessed from anywhere at anytime. Our goal is to fill this gap and develop an application based on a database management system which is not just storing and retrieving data, but also represent the data in order to generate more effective trends.

The dataset consists of over 20 attributes and over 2+ millions records every month, which gives us ample data to derive trends from and show a graphical representation of that particular trend, enabling users to better visualize the vast amount of data. Hence, a Database System that can handle this data requires performing millions of computations to generate results.

1.3 Trend Queries

Our application focuses on 5 main trend queries. Beyond that, the user can use various sliders and dropdowns to mix-and-match and create their own queries. The 5 queries are:

Query #1: Find the trend of passengers over time for the given time bin at the

current and k - nearest clusters to maximize the pickups.

Query #2: Find the effect of Uber and Lyft on Yellow Cabs in NYC over time.

Query #3. Find the overall trend of cab trips during major holidays across a

calendar year.

Query #4: Track the number of passengers over time before and after COVID to

see the effects that COVID may have had on people taking a taxi.

Query #5: Different payment methods used over time by passengers

ER Diagram Description

Data Source: https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

The raw data is available but there are over 20 different parameters with millions of records, each with varying levels of relevance with each other. It is necessary to create an Entity-Relation (ER) diagram to create a conceptual database design to understand how we are going to manage our database. First, we categorize the entity sets for our model. We create six entity sets:

1. Trip
2. Coordinate
3. Cab Driver
4. Passengers
5. Payment
6. Additional Charges

The most important concept is that “Trip” is the most significant of these sets, possessing the most metadata. Every form of data is on a per trip basis. Fundamentally, we are going to use the time related parameters of each trip, like data and time, for our queries. For example, when examining the trend of parameters throughout the day we would use time, but when we examine the trend of parameters over the course of a year (or years) we would use the date. Our motivation for the design is to make it easy to extract trend based queries by the time and date data per trip.

All data is associated per trip and theoretically every parameter could be an attribute of trip itself. The reason we separate into several entity sets is twofold. One, is that visually we can separate data into more meaningful segments and avoid clutter. Second, and perhaps more importantly, is that by separating into different entity sets we can avoid the overhead of keeping data around that we don’t need. For example, “Payment” is its own entity set because most queries around payment type are all related to each other. We do not need the overhead of every other attribute when all we are doing is measuring the trend of payment related data.

The entity sets should be self explanatory: each trip has associated data like the date and number of passengers, each coordinate has a latitude and longitude, each cab driver has their own ID, each passenger has their own payment method, each payment has the different ways payments can be made like through tips or tolls, and finally additional charges has all the different types of additional charges, like surge pricing. Our dataset does not provide a tripID, but we are going to create it ourselves so that each entry has a straightforward primary key. The time aspects of the trip are found with “ tpepDropoffDateTime”

and “ tpepPickupDateTime”. Notably, for privacy reasons, there is no data available for identification of passengers. To create a primary key for passengers, we associate them with the trip ID. Also, the fare total is a derived attribute and is the sum of all payments made for that trip. We use a derived attribute here because the total fare amount is a very useful statistic to have and the utility of having access to it outweighs the demerits of using a derived attribute.

The relationship sets provides the “connection” between various empty set.

* Every trip starts at a particular coordinate and ends at a particular coordinate.
* A cab driver makes a trip, drives a passenger, and accepts payment.
* Every passenger pays for a trip and makes a payment.
* Every passenger may have additional charges.

We make several key assumptions related to cardinalities: a passenger can take more than one trip, the same location can be used for many trips, and each trip need not have additional charges. That is also the reason we felt it apt to separate additional charges into its own entity, instead of using the various additional charges as attributes to “Payment” itself.

- Trip

- doLocationID

- passengerCount

- puLocationId

- puMonth

- puYear

- rateCodeId

- storeAndFwdFlag

- tpepDropoffDateTime

- tpepPickupDateTime

- vendorID

- Coordinates

- endLat

- endLon

- startLat

- startLon

- Cab Driver

- vendorID

- Passenger

- tripID

- paymentMethod

- Payment

- tripID

- paymentmethod

- tipAmount

- tollsAmount

- totalAmount

- improvementSurcharge

- mtaTax

- (derived) fareAmount

- Additional Charges

- improvementSurcharge

- mtaTax

- (derived) fareAmount

- extra

Entities for ER Diagram