DS 5110: Introduction to Data Management and Processing (Fall 2023)

TAXI/CAB BOOKING SYSTEM

Final Report Submission

Bharath Gajula

Introduction

We have completed a project on Taxi/Cab Booking System. In today's fast-paced world, efficient transportation services are a fundamental part of our daily lives. Using a taxi booking system is a revolutionary approach for consumers to hail and use services. It's a system that links travelers with available cabs, providing easy travel options.This type of project aims to provide a comprehensive solution for various aspects of running a taxi or cab business. The final project has fourteen database tables that together include all of the subtleties of the reservation procedures objectives such as User Management, Trip Management, Driver Management, Vehicle Management, Payment and Billing, Rating and Feedback, Promotions and Offers and Surge Pricing. Our objective was to integrate all components in a smooth manner, taking care of any issues with data discrepancies and normalization along the way. We tried to meet user expectations and industry requirements for a high-quality automobile booking system by encouraging open communication and teamwork within the team .

Important Elements:

User Interfaces

The booking site lets customers review drivers, book rides, follow the status of their journeys, see driver details, and make payments. It offers navigation, ride requests, passenger details, and earnings data to drivers.

Infrastructure at the back end:

Database: Holds user profiles, payment information, trip specifics, and driver details.

Flask: Responds to queries from user interfaces, arranges reservations, distributes rides,and controls driver-passenger communication.

Database Design

The finalized Entity-Relationship (ER) diagram is represented in Figure 1. There are no partial dependencies or transitive dependencies in the ER diagram making it 3NF normalized. We divided the ER diagram into three parts to handle customer details, driver details, and trip details.

Driver Relation:

- The Driver table contains details about each driver, and this table is mapped with the Cab_details table, which associates specific cabs with their drivers.
- The pickup_time_estimation table links drivers to estimated pickup times, allowing for efficient assignment of drivers to incoming trip requests.
- The foreign key relationship between Cab_details and Driver ensures that each cab is associated with a valid driver.

Customer Relation:

- The Customer_details table stores information about each customer, including their email, password, and contact details.
- The Rider_details table connects customers to the number of riders associated with each booking.
- The foreign key relationship between Rider_details and Customer_details ensures that each rider detail corresponds to a valid customer.

Booking Process:

- A customer initiates a trip request by providing pickup and drop-off locations.
- The system estimates the fare, assigns a driver based on pickup time estimation, and records the request in the Trip_requests table.
- If the trip is completed, details are stored in the Completed_trips table, and the customer can provide a rating and feedback in the Rating table.
- In the case of an incomplete trip, details are recorded in the Incomplete_trips table.

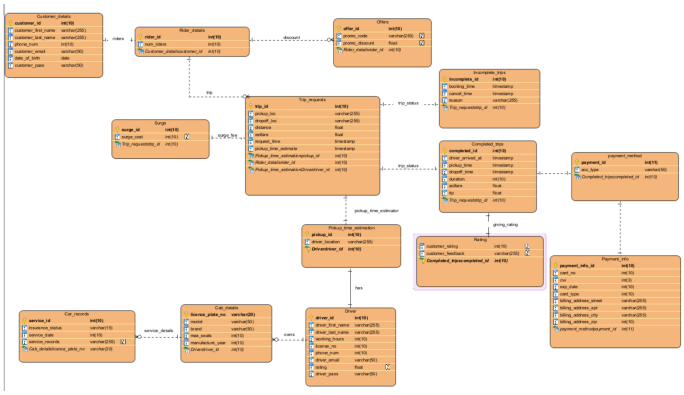

Figure 1: Final Entity Relation Diagram

Views:

Detailed_Trip_Info View: This view displays detailed information about each trip.Structure: It joins `Trip_requests`, `Driver`, `Rider_details`, and `Customer_details` tables.

- Usage: It can be used to quickly access comprehensive trip details, including driver and customer information, which is useful for customer support and trip management.
- Application: This view is ideal for displaying trip information on a customer's booking page. It provides comprehensive details about the trip, including pickup and dropoff locations, estimated fare, driver name, and customer name.
- Benefit: Customers can see all relevant details about their booked trips in one place, enhancing transparency and trust in the service

```
-- View to display detailed information about each trip
CREATE VIEW Detailed_Trip_Info AS
SELECT
    t.trip_id,
    t.pickup_loc,
    t.dropoff_loc,
    t.distance,
    t.estfare,
    t.request_time,
    d.driver_first_name,
    d.driver_last_name,
    c.customer_first_name,
    c.customer_last_name
FROM
    Trip_requests t
JOIN
    Driver d ON t.Pickup_time_estimationDriverdriver_id = d.driver_id
JOIN
    Rider_details r ON t.Rider_detailsrider_id = r.rider_id
JOIN
    Customer_details c ON r.Customer_detailscustomer_id = c.customer_id;
```

Figure 2: Detailed Trip Info

Driver_Summary View: This view displays detailed information about each trip.Structure: It joins `Trip_requests`, `Driver`, `Rider_details`, and `Customer_details` tablesPurpose: Summarizes completed trips and total earnings by each driver.

- Structure: Joins `Driver` and `Completed_trips` tables, and aggregates data to count trips and sum earnings per driver.
- Application: Used on a driver's summary page, this view offers a quick overview of a driver's completed trips and total earnings.
- Benefit: Drivers can easily track their performance, number of trips completed, and earnings, which is crucial for their financial planning and understanding their work progress.

```
-- View to summarize completed trips and earnings by driver
CREATE VIEW Driver_Summary AS
SELECT
    d.driver_id,
    d.driver_first_name,
    d.driver_last_name,
    COUNT(ct.completed_id) AS total_trips,
    SUM(ct.actfare) AS total_earnings
FROM
    Driver d
JOIN
    Completed_trips ct ON d.driver_id = ct.Trip_requeststrip_id
GROUP BY
    d.driver_id;
```

Figure 3: Driver Summary

Incomplete_Trip_Details View:

- Purpose: Lists all incomplete trips along with cancellation reasons.
- Structure: Combines `Incomplete_trips` and `Trip_requests` tables.
- Usage: Helps in identifying patterns in trip cancellations, which can inform strategies to reduce cancellations and improve service quality.
- Application: This view is suitable for pages showing incomplete trip details. It includes information like booking time, cancellation time, and reasons for cancellation.
- Benefit: Helps both drivers and administrators understand why trips are not completed, enabling them to identify areas for improvement and take corrective actions.

```
-- View to list all incomplete trips with reasons
CREATE VIEW Incomplete_Trip_Details AS
SELECT
    it.incomplete_id,
    it.booking_time,
    it.cancel_time,
    it.reason,
    t.pickup_loc,
    t.dropoff_loc
FROM
    Incomplete_trips it
JOIN
    Trip_requests t ON it.Trip_requeststrip_id = t.trip_id;
```

Figure 4: Incomplete Trip Details.

Customer_Feedback View:

- Purpose: Provides customer feedback and ratings for each completed trip.
- Structure: Merges `Completed_trips`, `Rating`, and `Trip_requests` tables.
- Usage: Essential for monitoring customer satisfaction, addressing service issues, and improving overall customer experience.
- Application: This view can be used on pages dedicated to analyzing customer feedback and ratings for each completed trip.

- Benefit: Provides valuable insights into customer satisfaction and service quality. This information is crucial for business improvements, driver training, and enhancing customer experience.

```sql
-- View for customer feedback and ratings for each completed trip
CREATE VIEW Customer_Feedback AS
SELECT
    ct.completed_id,
    r.customer_rating,
    r.customer_feedback,
    t.pickup_loc,
    t.dropoff_loc
FROM
    Completed_trips ct
JOIN
    Rating r ON ct.completed_id = r.Completed_tripscompleted_id
JOIN
    Trip_requests t ON ct.Trip_requeststrip_id = t.trip_id;
```

Figure 5: Customer Feedback.

Triggers:

update_driver_rating Trigger:

- Purpose: Updates the driver's average rating after new customer feedback is inserted.
- Working: After a new record is inserted into the `Rating` table, this trigger calculates the average customer rating for the driver associated with the completed trip and updates the driver's rating in the `Driver` table.
- Application Use: This trigger ensures that the driver's rating is always current and reflects the latest customer feedback, which is essential for maintaining service quality and customer trust.

```sql
-- Trigger to Update Driver Rating After Customer Feedback
CREATE TRIGGER update_driver_rating
AFTER INSERT ON Rating
FOR EACH ROW
BEGIN
  UPDATE Driver
  SET rating = (SELECT AVG(customer_rating)
                FROM Rating
                JOIN Completed_trips ON Rating.Completed_tripscompleted_id = Completed_trips.completed_id
                WHERE Completed_trips.Trip_requeststrip_id = NEW.Completed_tripscompleted_id)
  WHERE driver_id = (SELECT Trip_requeststrip_id FROM Completed_trips WHERE completed_id = NEW.Completed_tripscompleted_id
END
$$
DELIMITER ;
```

Figure 6: Update driver rating trigger.

record_incomplete_trip Trigger:

- Purpose: Records a trip as incomplete when its status is updated to 'Canceled'.
- Working: When the `request_status` of a trip in `Trip_requests` is changed from 'Booked' to 'Canceled', this trigger creates a record in the `Incomplete_trips` table, noting the time of booking, cancellation, and the reason for cancellation.
- Application Use: This trigger automates the process of tracking incomplete trips, which can be used to analyze cancellation patterns and improve service reliability.

```sql
DELIMITER $$
-- Trigger to Record Cancellation in Incomplete_trips
CREATE TRIGGER record_incomplete_trip
AFTER UPDATE ON Trip_requests
FOR EACH ROW
BEGIN
  IF OLD.request_status = 'Booked' AND NEW.request_status = 'Cancelled' THEN
    INSERT INTO Incomplete_trips (booking_time, cancel_time, reason, Trip_requeststrip_id)
    VALUES (NEW.request_time, CURRENT_TIMESTAMP, 'Cancelled by user', NEW.trip_id);
  END IF;
END;
$$
DELIMITER ;
```

Figure 7: Record incomplete trip trigger.

apply_surge_pricing Trigger:

- Purpose: Applies surge pricing to trips requested during peak hours.
- Working: Before a new trip request is inserted into `Trip_requests`, if the request time is during peak hours (e.g., 5 PM to 8 PM), this trigger increases the estimated fare by a certain percentage (e.g., 20%).
- Application Use: Automatically adjusts trip fares during peak hours, ensuring dynamic pricing based on demand, which can optimize revenue and manage customer expectations.

```sql
DELIMITER $$
-- Trigger to Apply Surge Pricing
CREATE TRIGGER apply_surge_pricing
BEFORE INSERT ON Trip_requests
FOR EACH ROW
BEGIN
  IF HOUR(NEW.request_time) BETWEEN 17 AND 20 THEN
    SET NEW.estfare = NEW.estfare * 1.2; -- Assuming a 20% surge
  END IF;
END;

$$
DELIMITER ;
```

Figure 8: Apply surge pricing trigger.

archive_completed_trips Trigger:

- Purpose: Archives completed trip records older than a certain period.
- Working: Before a new record is inserted into `Completed_trips`, this trigger moves completed trip records older than a specified time (e.g., one year) into an archive table, `Completed_trips_archive`.
- Application Use: Helps in maintaining database performance by archiving old data, keeping the active tables lean and efficient for queries. This is particularly useful for reporting and historical analysis.

These triggers enhance the functionality of the taxi booking system by automating essential database operations. They help maintain data integrity, improve operational efficiency, and provide a better user experience by ensuring up-to-date and relevant data is used throughout the application.

Stored Procedures:

Calculate_Estimated_Fare Procedure:

- Purpose: Calculates the estimated fare for a trip based on distance.
- Functionality: This procedure takes the trip distance as input, calculates the fare using a base fare and a per-mile rate, and adjusts for surge pricing during peak hours.
- Application Use: Can be used in the application to provide real-time fare estimates to customers when they input their trip details. This helps customers know the expected cost before booking the trip

```
-- Stored Procedure to Calculate Estimated Fare
DELIMITER //
CREATE PROCEDURE Calculate_Estimated_Fare(IN distance FLOAT, OUT est_fare FLOAT)
BEGIN
  DECLARE base_fare FLOAT;
  DECLARE per_mile_rate FLOAT;
  SET base_fare = 5.0; -- Base fare for the trip
  SET per_mile_rate = 2.0; -- Rate per mile
  SET est_fare = base_fare + (distance * per_mile_rate);
  -- Check for surge pricing and apply if necessary
  IF HOUR(CURRENT_TIME()) BETWEEN 17 AND 20 THEN
    SET est_fare = est_fare * 1.2; -- Assuming a 20% surge
  END IF;
END //
DELIMITER ;
```

Figure 10: Calculated estimated fare.

Record_New_Trip Procedure:

- Purpose: Records a new trip in the database
- Functionality: It accepts pickup and dropoff locations, and rider ID, calculates the distance and estimated fare (using Calculate_Estimated_Fare), selects a driver, and then records the trip in Trip_requests.
- Application Use: This procedure simplifies trip recording by encapsulating all necessary steps into one call. It can be used when a customer books a trip, ensuring all relevant details are stored efficiently and correctly.

```
-- Stored Procedure to Record a New Trip
DELIMITER //
CREATE PROCEDURE Record_New_Trip(IN pickup_loc VARCHAR(255), IN dropoff_loc VARCHAR(255), IN rider_id INT)
BEGIN
  DECLARE distance FLOAT;
  DECLARE est_fare FLOAT;
  DECLARE driver_id INT;
  -- Dummy distance calculation
  SET distance = 10; -- Assume a fixed distance for this example
  CALL Calculate_Estimated_Fare(distance, est_fare);
  -- Select an available driver (dummy selection for this example)
  SET driver_id = (SELECT driver_id FROM Driver LIMIT 1);
  -- Insert the new trip
  INSERT INTO Trip_requests(pickup_loc, dropoff_loc, distance, estfare, request_time, Rider_detailsrider_id, Pickup_time_estimationDriverdriver_id)
  VALUES(pickup_loc, dropoff_loc, distance, est_fare, CURRENT_TIMESTAMP, rider_id, driver_id);
END //
DELIMITER ;
```

Figure 11: Record new trip.

Update_Trip_Status Procedure:

- Purpose: Updates the status of a trip in the database.
- Functionality: Takes a trip ID and a new status (e.g., 'Completed'). If the status is 'Completed', it updates the trip record and inserts completion details into Completed_trips.
- Application Use: This procedure is essential for updating trip status, such as marking a trip as completed. It can be triggered at the end of a trip, ensuring the system accurately reflects the current state of each booking.

```
-- Stored Procedure to Update Trip Status
DELIMITER //
CREATE PROCEDURE Update_Trip_Status(IN trip_id INT, IN new_status VARCHAR(50))
BEGIN
  IF new_status = 'Completed' THEN
    -- Update the trip to completed and record the completion details
    UPDATE Trip_requests
    SET status = new_status
    WHERE trip_id = trip_id;
    -- Assume dummy values for completion details
    INSERT INTO Completed_trips(driver_arrived_at, pickup_time, dropoff_time, duration, actfare, tip, Trip_requeststrip_id)
    VALUES(CURRENT_TIMESTAMP, DATE_ADD(CURRENT_TIMESTAMP, INTERVAL 5 MINUTE), DATE_ADD(CURRENT_TIMESTAMP, INTERVAL 30 MINUTE), 25, 30.0, 5.0, trip_id)
  ELSE
    -- Update the trip status for other statuses
    UPDATE Trip_requests
    SET status = new_status
    WHERE trip_id = trip_id;
  END IF;
END //
DELIMITER ;
```

Figure 12: Update trip status.

These stored procedures encapsulate complex database operations, making them simpler to execute and maintain. They are particularly useful for ensuring data consistency and reducing the complexity of database interactions in the application. By using these procedures, the application can efficiently manage trip bookings, fare calculations, and trip status updates, enhancing the overall functionality and user experience of the taxi booking system.

Functions:

CalculateTotalFare Function:

- Purpose: Calculates the total fare for a trip, considering surge pricing.
- Functionality: This function takes the trip distance and a boolean indicating whether surge pricing is applicable. It calculates the fare using a base fare and a per-mile rate and applies a surge factor if necessary.
- Database Relation: It interacts with the trip data to determine the final fare, potentially factoring in surge pricing.
- Application Use: This function can be used in the application to provide customers with an accurate fare estimate at the time of booking, especially during peak hours when surge pricing might be in effect.

```
-- Function to calculate total fare with surge pricing
DELIMITER //
CREATE FUNCTION CalculateTotalFare(distance FLOAT, isSurge BOOLEAN) RETURNS FLOAT
DETERMINISTIC
BEGIN
  DECLARE baseFare FLOAT;
  DECLARE perMileRate FLOAT;
  DECLARE totalFare FLOAT;
  SET baseFare = 5.0; -- Base fare for the trip
  SET perMileRate = 2.0; -- Rate per mile
  SET totalFare = baseFare + (distance * perMileRate);
  IF isSurge THEN
    SET totalFare = totalFare * 1.2; -- Applying a 20% surge
  END IF;
  RETURN totalFare;
END //
DELIMITER ;
```

Figure 13: Calculate total fare function.

TotalTripsByDriver Function:

- Purpose: Counts the total number of completed trips by a specific driver.
- Functionality: Accepts a driver ID and returns the count of trips completed by that driver.
- Database Relation: It queries the `Completed_trips` table to count the trips associated with a given driver.
- Application Use: Useful for generating driver performance reports and statistics within the application. It can help in evaluating driver activity and workload, which is valuable for administrative and operational decision-making.

```
-- Function to count total trips by a driver
DELIMITER //
CREATE FUNCTION TotalTripsByDriver(driverId INT) RETURNS INT
DETERMINISTIC
BEGIN
  DECLARE totalTrips INT;
  SELECT COUNT(*) INTO totalTrips FROM Completed_trips WHERE Trip_requeststrip_id = driverId;
  RETURN totalTrips;
END //
DELIMITER ;
```

Figure 14: Total trips by driver function.

AverageDriverRating Function:

- Purpose: Calculates the average customer rating for a driver.
- Functionality: Takes a driver ID and computes the average rating based on customer feedback.

- Database Relation: It combines data from the `Rating` and `Completed_trips` tables to calculate the average rating for each driver.
- Application Use: This function is essential for monitoring driver performance from the customer's perspective. It can be used in the application for driver evaluation, incentivization programs, and to provide feedback to drivers for service improvement.

```
-- Function to calculate average driver rating
DELIMITER //
CREATE FUNCTION AverageDriverRating(driverId INT) RETURNS FLOAT
DETERMINISTIC
BEGIN
  DECLARE avgRating FLOAT;
  SELECT AVG(customer_rating) INTO avgRating
  FROM Rating
  JOIN Completed_trips ON Rating.Completed_tripscompleted_id = Completed_trips.completed_id
  WHERE Completed_trips.Trip_requeststrip_id = driverId;
  RETURN avgRating;
END //
DELIMITER ;
```

Figure 15: Average Driver Rating function.

These functions encapsulate complex queries and calculations, simplifying database interactions for the application. They play a crucial role in providing real-time, relevant data for fare calculation, driver performance assessment, and operational analytics. By leveraging these functions, the taxi booking system can efficiently manage fare estimation, track driver performance, and enhance overall service quality.

Data Collection

Data collection is done mainly by employing three methods: 1. free-to-use CSV files from online websites; 2. web scraping techniques; and 3. Python libraries like Faker or generating random data.

CSV files

Sample data from the dlptest website is used for getting names, emails, phone numbers, and addresses of people. There are some unwanted columns in the dataset; these unwanted columns are removed using the Pandas library from Python, and a proper standard is not followed for the phone number column in the dataset. We removed all spaces and '-' from the phone number column and standardized all the values in that column. Then this data is used to fill the driver details and customer details tables. Fig. 17 below shows the dataset before data cleaning and removing unwanted columns, and Fig. 18 shows the cleaned dataset after removing unwanted columns.

	SSN	gender	birthdate	maiden name	last name	first name
0	172-32-1176	m	4/21/1958	Smith	White	Johnson
1	514-14-8905	f	12/22/1944	Amaker	Borden	Ashley
2	213-46-8915	f	4/21/1958	Pinson	Green	Marjorie
3	524-02-7657	m	3/25/1962	Hall	Munsch	Jerome
4	489-36-8350	m	1964/09/06	Porter	Aragon	Robert

	address	city	state	zip	phone
0	10932 Bigge Rd	Menlo Park	CA	94025	408 496-7223
1	4469 Sherman Street	Goff	KS	66428	785-939-6046
2	309 63rd St. #411	Oakland	CA	94618	415 986-7020
3	2183 Roy Alley	Centennial	CO	80112	303-901-6123
4	3181 White Oak Drive	Kansas City	MO	66215	816-645-6936

	email	cc_type	CCN	cc_cvc	cc_expiredate
0	jwhite@domain.com	m	5270-4267-6450-5516	123	2010/06/25
1	aborden@domain.com	m	5370-4638-8881-3020	713	2011/02/01
2	mgreen@domain.com	v	4916-9766-5240-6147	258	2009/02/25
3	jmunsch@domain.com	m	5180-3807-3679-8221	612	2010/03/01
4	raragon@domain.com	v	4929-3813-3266-4295	911	2011/12/01

Figure 16: CSV file before data cleaning

	driver_last_name	driver_first_name	phone_num	driver_email
0	White	Johnson	4084967223	jwhite@domain.com
1	Borden	Ashley	7859396046	aborden@domain.com
2	Green	Marjorie	4159867020	mgreen@domain.com
3	Munsch	Jerome	3039016123	jmunsch@domain.com
4	Aragon	Robert	8166456936	raragon@domain.com

Figure 17: CSV file after data cleaning

Web Scraping

Implementation of web scraping is done using the "BeautifulSoup" library from Python. Using web scraping, we extracted car details. The model and brand names of 22 cars are extracted from an article on the hyrecar.com website about the 22 best cars for Uber drivers to use. After getting car models and brand names, the maximum number of seats in the car is extracted by making a query and scraping from the cars.com website.

Figure 18: List of cars from hyrecar.com

	brand	model	manufacture_year	max_seats
0	Honda	Odyssey	2006	6
1	Lincoln	MKT	2015	6
2	Volkswagen	e-Golf	2013	4
3	Subaru	XV CrossTrek	2011	4
4	Kia	Optima	2013	4

Figure 19: Car details dataset after web scraping

Python Libraries or Generating Random Data

The Faker library from Python is used to get the license plate numbers and license numbers of drivers. The manufacturing year of cars is generated using the Numpy library. Data in surge, trip request, completed trips, incompleted trips, offers, and rating is also generated using random data using generatedata.com.

Application Description

Main Features:
The application is composed of performing many intricate features in a smooth manner with every process connected to each other in a smooth manner.

- **User Registration:** Allows customers and drivers to create accounts and login through their credentials.
- **Taxi Booking:** Enable users to book taxis by specifying pick-up, drop-ff, and number of passengers.
- **Payment Integration:** Provide payment options for users.
- **Offers:** Let users choose from an amplitude of offers available for them to improve the price rates.
- **Rating and review:** Allows users to rate drivers and provide feedback after the ride
- **Car records and details:** The details of cab including its type,, number of seats etc is stored along with its insurance details

Application Flow and Data Interaction:

1. Registration and Login:
- Users and drivers register on their respective signup pages.
- Data entered in the forms is sent to the server (`app.py`), processed, and stored in the database.
- For login, the application verifies credentials against the database.

2. Booking Trips:
- Users book trips through a booking interface (not provided in the uploaded files).
- Booking details are sent to the server, which calculates fares and stores trip information in the database.

3. Payment Processing:
- After a trip, the user is directed to the payment page.
- Payment details are entered, processed by the server, and recorded in the database.

4. Viewing Trip History:
- The customer history page retrieves and displays a list of past trips for the user.
- The server queries the database for the user's trip history and sends this data back for display.

5. Driver Operations:
- Drivers log in on the driver login page to access their dashboard.
- The driver home page displays upcoming trips and earnings, retrieved from the database.
- The driver history page lists past trips, details, and earnings, fetched from historical trip data in the database.
- View Assignments: Check upcoming trips on the home page.
- Complete Trips: After finishing trips, details are updated in the system.

6. Customer Operations
- Customer Home Page (`customer_home.html`):
- Purpose: Main interface for customers to initiate trip bookings.
- Functionality: Offers options to input trip details like pickup and dropoff locations and the number of riders.
- Data Interaction: Data entered by the customer is sent to the backend. A query inserts rider details into `Rider_details` and trip requests into `Trip_requests`. The backend interacts with an

external API to calculate the trip distance and estimate the fare.

7.Booking Process:

- Flow: Upon submitting trip details on the customer home page, `app.py` processes the request.
- Data Interaction: Involves calculating the estimated fare, potentially using external APIs or internal logic, selecting a driver, and creating a new entry in the `Trip_requests` table with all relevant details. The system retrieves and stores data in various tables like `Rider_details`, `Trip_requests`, andpotentially `Pickup_time_estimation`. It ensures the trip details are accurately recorded in the database, ready for the driver to accept and proceed with the trip.
- This flow indicates a well-structured system where customer inputs are efficiently processed and integrated into the taxi booking database, ensuring a seamless booking experience.

Overall System Overview:

- The application uses Flask (`app.py`) as the backend to handle requests from various HTML pages.
- Processes user input, interacts with the database for data retrieval and storage.
- Serves the processed data back to the frontend for display, resulting in a seamless flow for both users and drivers.
- Users and drivers can register, log in, book trips, make payments, and view their respective histories.
- Data is consistently exchanged between the frontend and backend, ensuring real-time updates and accurate information for both users and drivers.

Analysis:

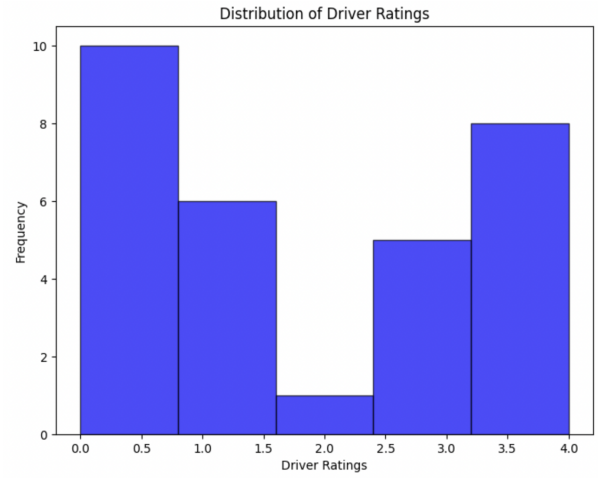

Figure 20: Average distribution of driver Ratings

We have 30 driver records in our table with the ratings information in the range of 0 being lowest to 4 being highest

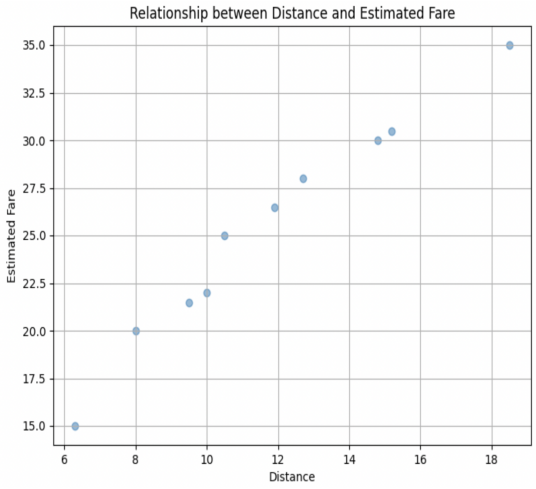

Figure 21: Point graph representing the fare increase with the increased distance

It's a point chart showing the increased fare details with the distance increasing

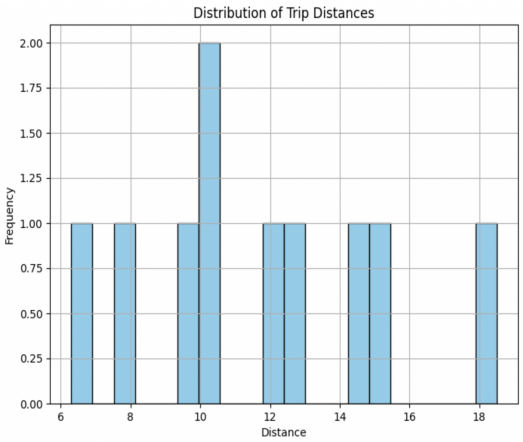

Figure 22: Graph showing the average distance traveled by customers

The distances parameter is analyzed which is showing the distance traveled by passengers 10 being the most frequent one.

Structure of Database:

- User profiles and login information are contained in the Users Table.

- Drivers Table: Holds vehicle details, availability status, and driver profiles.
- Trips Table: Contains information about each journey, such as start and end locations, timestamps, user and driver IDs, fares, ratings, and reviews.
- Operational parameters and system settings are stored in the system configuration table.

Conclusions

Lessons learned:

- **Database Design:** Recognising how crucial a well-organized database schema is to manage learned many facets of a taxi booking system.
- **Data integrity:** It is the process of guaranteeing data accuracy by establishing suitable limitations and connections between various tables like writing updates on certain actions.
- **Features of the application:** Learned about the key components of a taxi booking software, such as driver management, trip monitoring, payment integration, and user management.
- **Creating tables flow:** To simulate the request to completion process of a trip, including both finished and partial trip data.

Prospective Courses:

- Improved User Experience: Provide a more intuitive user interface with improved UX/UI design for easier navigation.
- Advanced Payment Options: Provide a variety of payment options, such as digital wallets, UPI, and extra security features, for the convenience of users..
- Predictive analytics: Make use of past trip data to forecast demand, maximize driver assignment, and generate dynamic fare estimates.
- The development of tools to track driver performance based on evaluations, comments, and trip completion rates is necessary.
- Ride Recommendation System: Set up a system that, in accordance with user choices, suggests optimum routes, favored car classes, or customized offers.

Future DS 5110 Students' Advice:

Design an effective database schema requiring a detailed understanding of the business requirements.

Understand normalization strategies to preserve data integrity and optimize database structure.Consider how database design affects application functions in the actual world. Also recognise the linkages between various tables and how they impact system performance.

To improve the functionality and scalability of the application, stay current on market trends and technological advancements.

References

Data sources

https://dlptest.com/sample-data/
https://www.hyrecar.com/cars-uber-lyft/
https://www.cars.com/
https://faker.readthedocs.io/en/master/
https://generatedata.com/generator