CAB FARE ESTIMATOR

A comprehensive ride-hailing fare estimation system built with Python, featuring dynamic pricing algorithms, persistent data storage and an interactive web dashboard.

📌 OVERVIEW

This project simulates a professional cab fare estimator system inspired by industry-leading ride-hailing platforms like Uber, Lyft and Ola. Built using object-oriented programming principles, it provides a realistic implementation of dynamic pricing algorithms commonly used in the transportation industry.

Key Technologies -

- Backend - Python with OOP design patterns.
- Database - SQLite for persistent trip storage.
- Frontend - Streamlit for interactive web dashboard.
- Deployment - Ngrok integration for easy sharing.

✨ CORE FEATURES

🔹 Intelligent Fare Calculation

Our pricing model incorporates multiple factors to provide realistic fare estimates -

COMPONENT	RATE	DESCRIPTION
Base Fare	₹50	Fixed starting cost
Distance Rate	₹10/km	Per kilometer charge
Time Rate	₹2/min	Per minute charge
Booking Fee	₹20	Platform service fee

🔹 Dynamic Pricing Engine

Traffic-Based Multipliers -

- 🟢 Light Traffic :- Standard rate (1.0x).
- 🟡 Medium Traffic :- +10% surcharge (1.1x).
- 🔴 Heavy Traffic :- +25% surcharge (1.25x).

Time-Based Pricing -

- Peak Hours (6-9 AM, 6-9 PM) :- +20% surcharge.
- Weekend Premium (Saturday & Sunday) :- +15% surcharge.

🔹 Promotional System

Built-in discount codes for customer acquisition and retention -

PROMO CODE	DISCOUNT	EFFECT
DISCOUNT10	10% off	Standard discount
FREERIDE	100% off	Completely free ride
HALF50	50% off	Half-price promotion

🔹 Comprehensive Data Management

SQLite Database Schema -

```
CREATE TABLE trips (
   id INTEGER PRIMARY KEY AUTOINCREMENT,
   driver TEXT NOT NULL,
   distance REAL NOT NULL,
   time INTEGER NOT NULL,
   traffic TEXT NOT NULL,
   day TEXT NOT NULL,
   start_hour INTEGER NOT NULL,
   fare REAL NOT NULL,
   promo_code TEXT,
   created_at TEXT NOT NULL
);
```

Persistent Storage Features -

- Automatic database creation and management.
- Transaction-safe operations.
- Efficient querying and indexing.
- Data integrity validation.

🔹 Advanced Analytics & Reporting

Generate comprehensive business insights -

- Financial Metrics - Total earnings, average fare analysis.
- Traffic Analytics - Distribution across traffic conditions.
- Performance Tracking - Highest/lowest fare identification.
- Driver Management - Individual earnings summaries.
- Visual Dashboards - Interactive charts and graphs.

🧑‍💻 OBJECT-ORIENTED DESIGN PRINCIPLES

Core Classes -

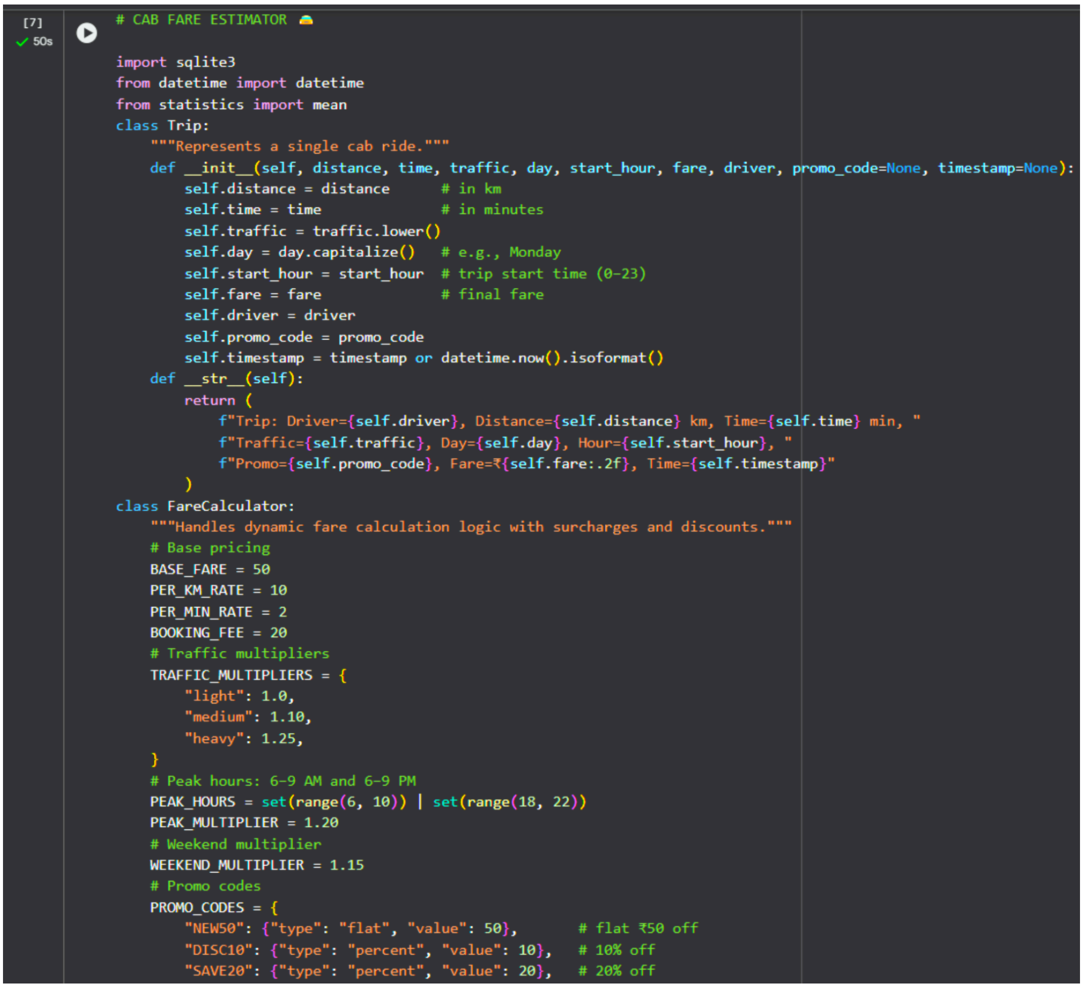
- Trip - Immutable data model representing a single ride.
    - Encapsulates trip parameters (distance, time, traffic, etc.).
    - Provides clean data validation and formatting.

- FareCalculator - Stateless pricing engine.
    - Implements all dynamic pricing rules.
    - Modular design for easy rule modifications.
    - Handles promotional code logic.

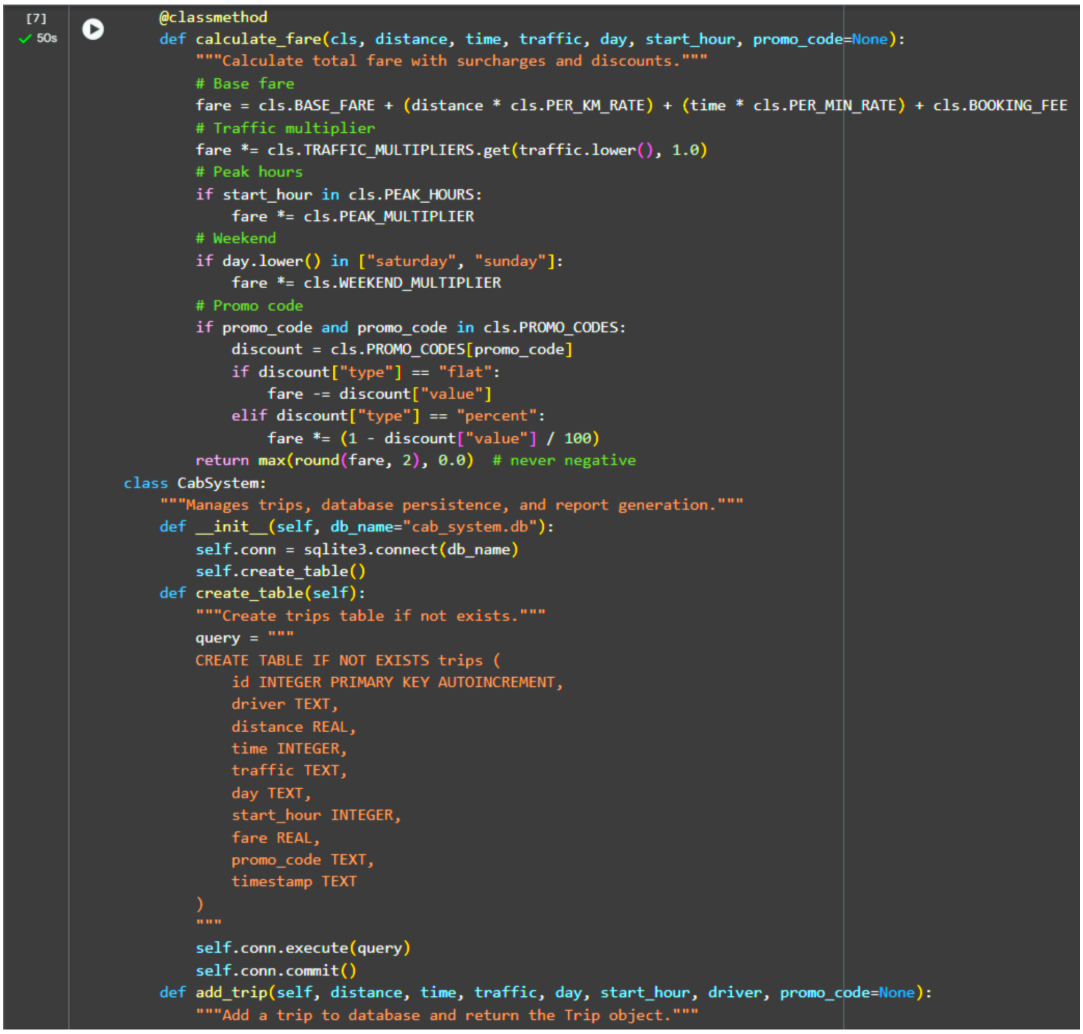
- DatabaseManager - Data persistence layer.
    - Abstracts SQLite operations.
    - Provides type-safe database interactions.
    - Handles connection management and error recovery.

- CabSystem - Main business logic coordinator.
    - Orchestrates trip booking workflow.
    - Generates comprehensive reports.
    - Manages system-wide operations.

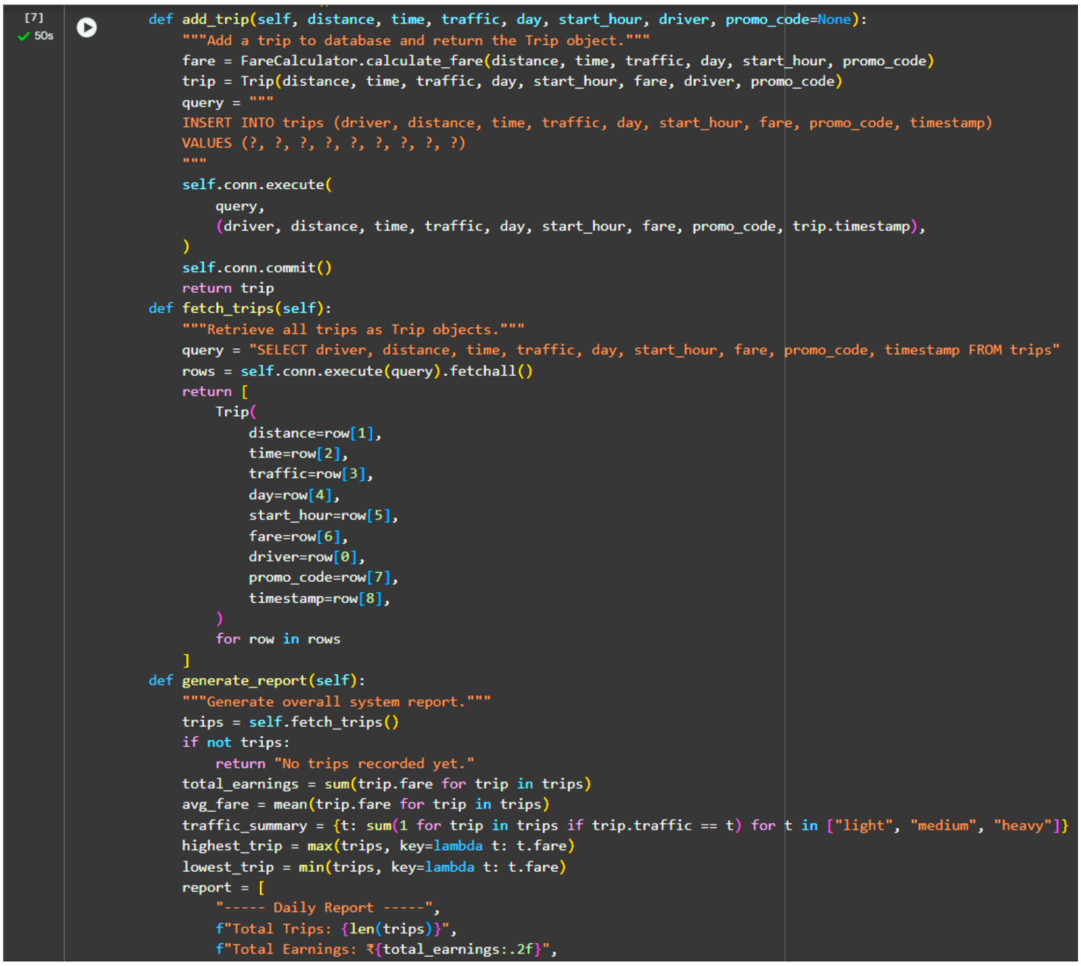
CODE SCREENSHOTS



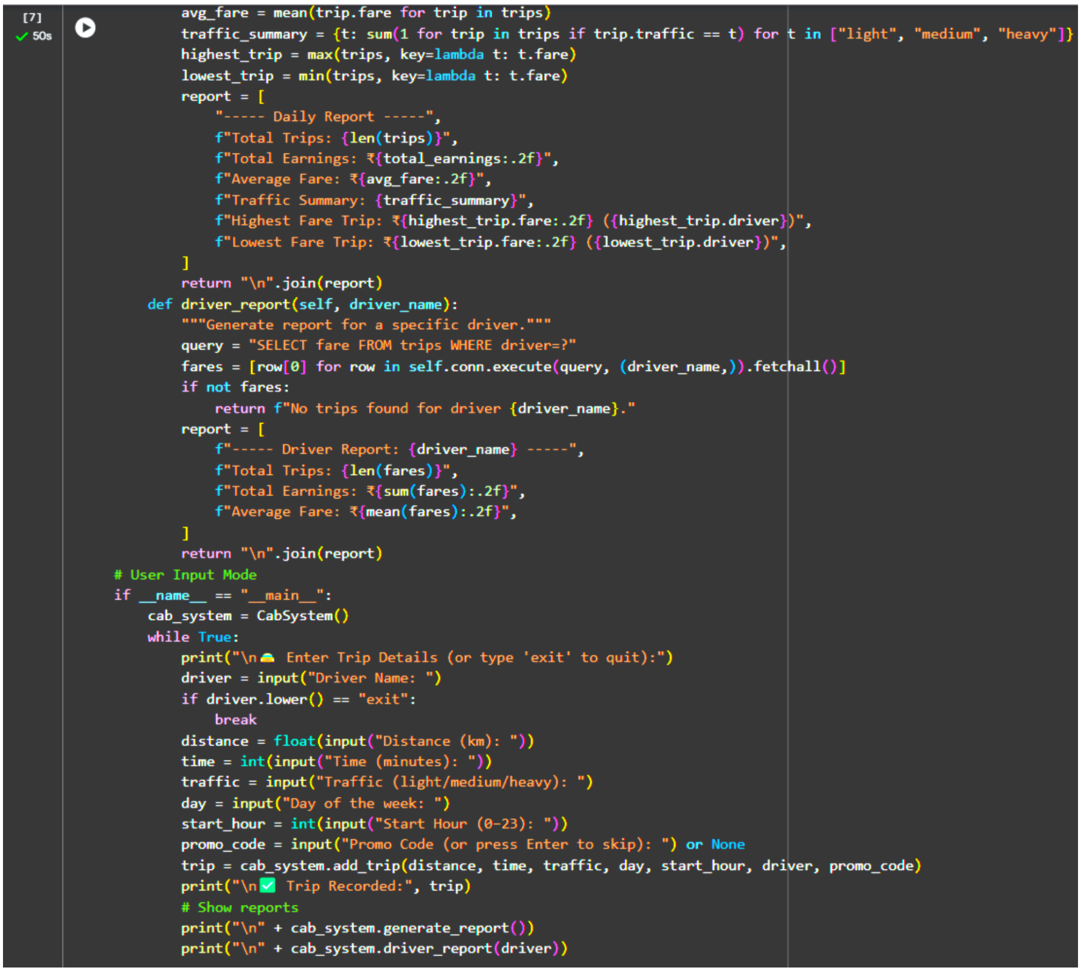
```python
# CAB FARE ESTIMATOR 🚕

import sqlite3
from datetime import datetime
from statistics import mean
class Trip:
    """Represents a single cab ride."""
    def __init__(self, distance, time, traffic, day, start_hour, fare, driver, promo_code=None, timestamp=None):
        self.distance = distance      # in km
        self.time = time              # in minutes
        self.traffic = traffic.lower()
        self.day = day.capitalize()   # e.g., Monday
        self.start_hour = start_hour  # trip start time (0-23)
        self.fare = fare              # final fare
        self.driver = driver
        self.promo_code = promo_code
        self.timestamp = timestamp or datetime.now().isoformat()
    def __str__(self):
        return (
            f"Trip: Driver={self.driver}, Distance={self.distance} km, Time={self.time} min, "
            f"Traffic={self.traffic}, Day={self.day}, Hour={self.start_hour}, "
            f"Promo={self.promo_code}, Fare=₹{self.fare:.2f}, Time={self.timestamp}"
        )
class FareCalculator:
    """Handles dynamic fare calculation logic with surcharges and discounts."""
    # Base pricing
    BASE_FARE = 50
    PER_KM_RATE = 10
    PER_MIN_RATE = 2
    BOOKING_FEE = 20
    # Traffic multipliers
    TRAFFIC_MULTIPLIERS = {
        "light": 1.0,
        "medium": 1.10,
        "heavy": 1.25,
    }
    # Peak hours: 6-9 AM and 6-9 PM
    PEAK_HOURS = set(range(6, 10)) | set(range(18, 22))
    PEAK_MULTIPLIER = 1.20
    # Weekend multiplier
    WEEKEND_MULTIPLIER = 1.15
    # Promo codes
    PROMO_CODES = {
        "NEW50": {"type": "flat", "value": 50},       # flat ₹50 off
        "DISC10": {"type": "percent", "value": 10},   # 10% off
        "SAVE20": {"type": "percent", "value": 20},   # 20% off
```



```python
    @classmethod
    def calculate_fare(cls, distance, time, traffic, day, start_hour, promo_code=None):
        """Calculate total fare with surcharges and discounts."""
        # Base fare
        fare = cls.BASE_FARE + (distance * cls.PER_KM_RATE) + (time * cls.PER_MIN_RATE) + cls.BOOKING_FEE
        # Traffic multiplier
        fare *= cls.TRAFFIC_MULTIPLIERS.get(traffic.lower(), 1.0)
        # Peak hours
        if start_hour in cls.PEAK_HOURS:
            fare *= cls.PEAK_MULTIPLIER
        # Weekend
        if day.lower() in ["saturday", "sunday"]:
            fare *= cls.WEEKEND_MULTIPLIER
        # Promo code
        if promo_code and promo_code in cls.PROMO_CODES:
            discount = cls.PROMO_CODES[promo_code]
            if discount["type"] == "flat":
                fare -= discount["value"]
            elif discount["type"] == "percent":
                fare *= (1 - discount["value"] / 100)
        return max(round(fare, 2), 0.0)  # never negative
class CabSystem:
    """Manages trips, database persistence, and report generation."""
    def __init__(self, db_name="cab_system.db"):
        self.conn = sqlite3.connect(db_name)
        self.create_table()
    def create_table(self):
        """Create trips table if not exists."""
        query = """
        CREATE TABLE IF NOT EXISTS trips (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            driver TEXT,
            distance REAL,
            time INTEGER,
            traffic TEXT,
            day TEXT,
            start_hour INTEGER,
            fare REAL,
            promo_code TEXT,
            timestamp TEXT
        )
        """
        self.conn.execute(query)
        self.conn.commit()
    def add_trip(self, distance, time, traffic, day, start_hour, driver, promo_code=None):
        """Add a trip to database and return the Trip object."""
```

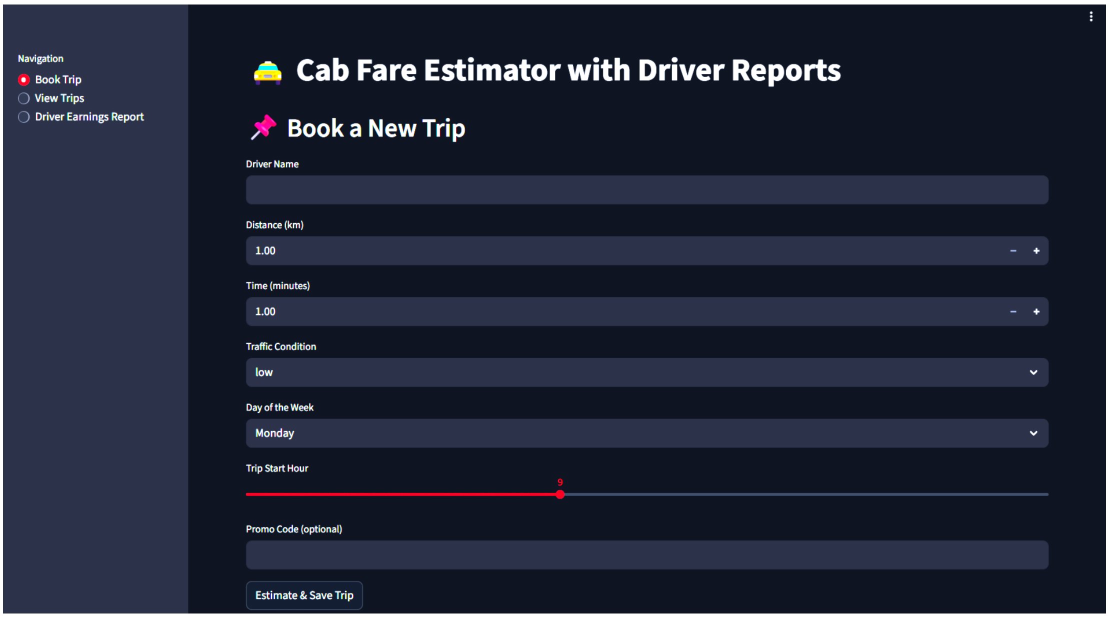


```
    def add_trip(self, distance, time, traffic, day, start_hour, driver, promo_code=None):
        """Add a trip to database and return the Trip object."""
        fare = FareCalculator.calculate_fare(distance, time, traffic, day, start_hour, promo_code)
        trip = Trip(distance, time, traffic, day, start_hour, fare, driver, promo_code)
        query = """
        INSERT INTO trips (driver, distance, time, traffic, day, start_hour, fare, promo_code, timestamp)
        VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)
        """
        self.conn.execute(
            query,
            (driver, distance, time, traffic, day, start_hour, fare, promo_code, trip.timestamp),
        )
        self.conn.commit()
        return trip
    def fetch_trips(self):
        """Retrieve all trips as Trip objects."""
        query = "SELECT driver, distance, time, traffic, day, start_hour, fare, promo_code, timestamp FROM trips"
        rows = self.conn.execute(query).fetchall()
        return [
            Trip(
                distance=row[1],
                time=row[2],
                traffic=row[3],
                day=row[4],
                start_hour=row[5],
                fare=row[6],
                driver=row[0],
                promo_code=row[7],
                timestamp=row[8],
            )
            for row in rows
        ]
    def generate_report(self):
        """Generate overall system report."""
        trips = self.fetch_trips()
        if not trips:
            return "No trips recorded yet."
        total_earnings = sum(trip.fare for trip in trips)
        avg_fare = mean(trip.fare for trip in trips)
        traffic_summary = {t: sum(1 for trip in trips if trip.traffic == t) for t in ["light", "medium", "heavy"]}
        highest_trip = max(trips, key=lambda t: t.fare)
        lowest_trip = min(trips, key=lambda t: t.fare)
        report = [
            "----- Daily Report -----",
            f"Total Trips: {len(trips)}",
            f"Total Earnings: ₹{total_earnings:.2f}",
```



```
        avg_fare = mean(trip.fare for trip in trips)
        traffic_summary = {t: sum(1 for trip in trips if trip.traffic == t) for t in ["light", "medium", "heavy"]}
        highest_trip = max(trips, key=lambda t: t.fare)
        lowest_trip = min(trips, key=lambda t: t.fare)
        report = [
            "----- Daily Report -----",
            f"Total Trips: {len(trips)}",
            f"Total Earnings: ₹{total_earnings:.2f}",
            f"Average Fare: ₹{avg_fare:.2f}",
            f"Traffic Summary: {traffic_summary}",
            f"Highest Fare Trip: ₹{highest_trip.fare:.2f} ({highest_trip.driver})",
            f"Lowest Fare Trip: ₹{lowest_trip.fare:.2f} ({lowest_trip.driver})",
        ]
        return "\n".join(report)
    def driver_report(self, driver_name):
        """Generate report for a specific driver."""
        query = "SELECT fare FROM trips WHERE driver=?"
        fares = [row[0] for row in self.conn.execute(query, (driver_name,)).fetchall()]
        if not fares:
            return f"No trips found for driver {driver_name}."
        report = [
            f"----- Driver Report: {driver_name} -----",
            f"Total Trips: {len(fares)}",
            f"Total Earnings: ₹{sum(fares):.2f}",
            f"Average Fare: ₹{mean(fares):.2f}",
        ]
        return "\n".join(report)
# User Input Mode
if __name__ == "__main__":
    cab_system = CabSystem()
    while True:
        print("\n🚖 Enter Trip Details (or type 'exit' to quit):")
        driver = input("Driver Name: ")
        if driver.lower() == "exit":
            break
        distance = float(input("Distance (km): "))
        time = int(input("Time (minutes): "))
        traffic = input("Traffic (light/medium/heavy): ")
        day = input("Day of the week: ")
        start_hour = int(input("Start Hour (0-23): "))
        promo_code = input("Promo Code (or press Enter to skip): ") or None
        trip = cab_system.add_trip(distance, time, traffic, day, start_hour, driver, promo_code)
        print("\n✅ Trip Recorded:", trip)
        # Show reports
        print("\n" + cab_system.generate_report())
        print("\n" + cab_system.driver_report(driver))
```

STREAMLIT SCREENSHOTS



🚕 Cab Fare Estimator with Driver Reports

📌 Book a New Trip

Driver Name

Distance (km)
1.00

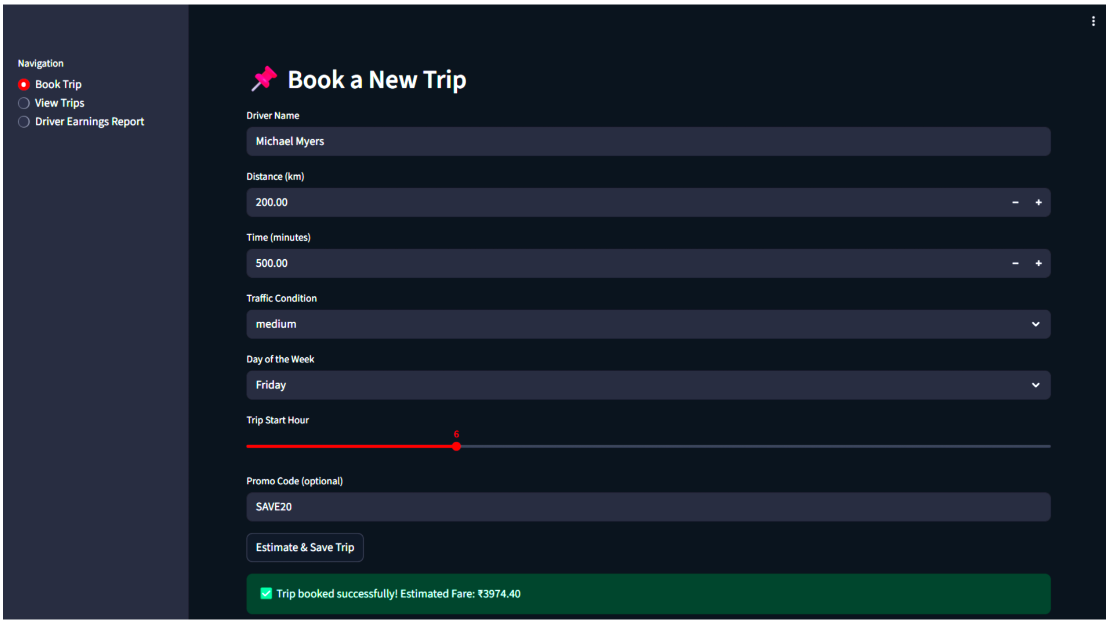
Time (minutes)
1.00

Traffic Condition
low

Day of the Week
Monday

Trip Start Hour
9

Promo Code (optional)



📌 Book a New Trip

Driver Name
Michael Myers

Distance (km)
200.00

Time (minutes)
500.00

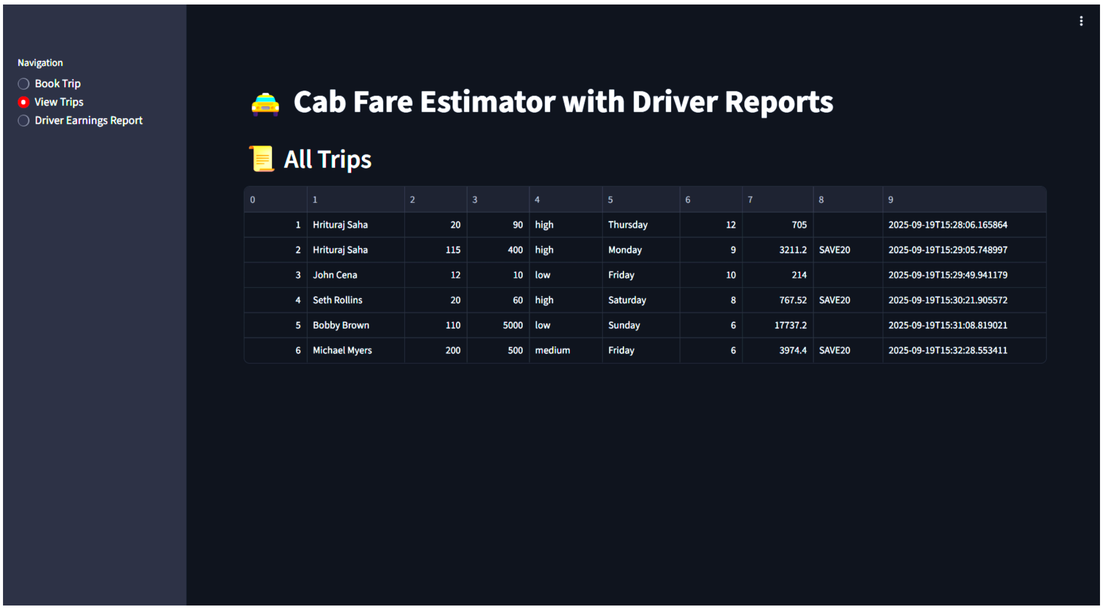
Traffic Condition
medium

Day of the Week
Friday

Trip Start Hour
6

Promo Code (optional)
SAVE20

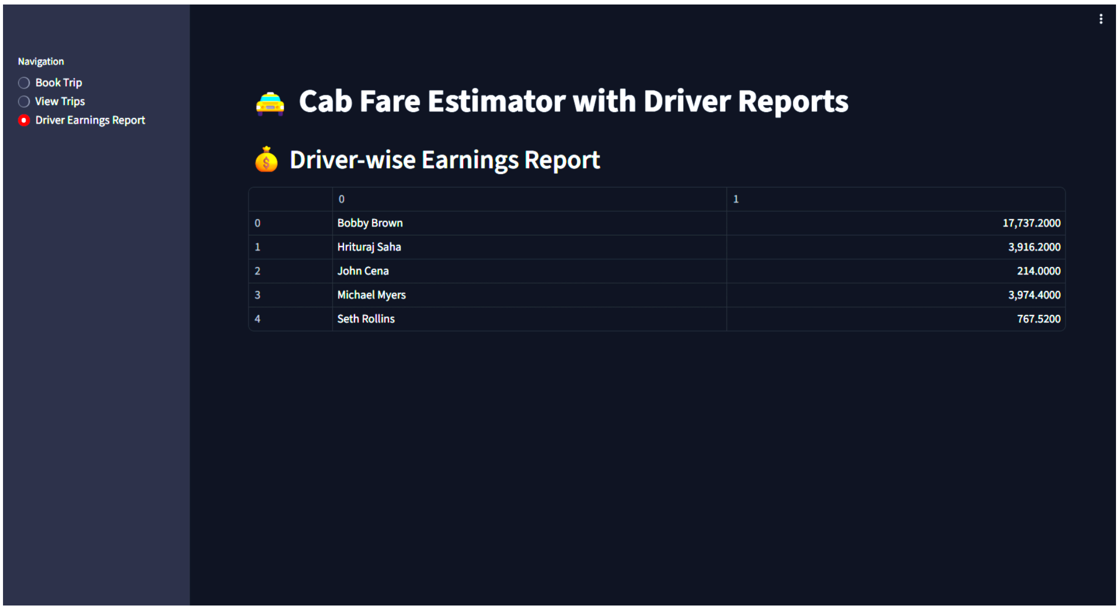
☑ Trip booked successfully! Estimated Fare: ₹3974.40



🚕 Cab Fare Estimator with Driver Reports

📜 All Trips

0	1	2	3	4	5	6	7	8	9
1	Hrituraj Saha	20	90	high	Thursday	12	705		2025-09-19T15:28:06.165864
2	Hrituraj Saha	115	400	high	Monday	9	3211.2	SAVE20	2025-09-19T15:29:05.748997
3	John Cena	12	10	low	Friday	10	214		2025-09-19T15:29:49.941179
4	Seth Rollins	20	60	high	Saturday	8	767.52	SAVE20	2025-09-19T15:30:21.905572
5	Bobby Brown	110	5000	low	Sunday	6	17737.2		2025-09-19T15:31:08.819021
6	Michael Myers	200	500	medium	Friday	6	3974.4	SAVE20	2025-09-19T15:32:28.553411



🚕 Cab Fare Estimator with Driver Reports

💰 Driver-wise Earnings Report

	0	1
0	Bobby Brown	17,737.2000
1	Hrituraj Saha	3,916.2000
2	John Cena	214.0000
3	Michael Myers	3,974.4000
4	Seth Rollins	767.5200

🛠️ CHALLENGES & SOLUTIONS

1. Dynamic pricing rules
    - Solved by using modular multipliers in FareCalculator.
2. Persistent storage
    - Switched from in-memory list → SQLite database for real-world application.
3. Ngrok deployment
    - Added process cleanup + logging to avoid port conflicts in Colab.

📌 REAL-WORLD RELEVANCE

- Simulates real ride-hailing pricing models.
- Demonstrates OOP + database integration.
- Interactive dashboards show data visualization & reporting.
- Extensible for -
    - Surge pricing based on demand.
    - Rider/driver authentication.
    - Payment gateway integration.

🔮 FUTURE ENHANCEMENTS

- Rider & driver authentication system.
- REST API for mobile app integration.
- Advanced ML-based fare prediction.
- Geo-location based surge pricing.
- Export reports to Excel/PDF.