Capstone Project Report: Dynamic Parking Pricing System

1. Introduction

This report presents a real-time dynamic parking pricing engine that integrates data ingestion, feature engineering, pricing strategies, and live visualization. The current focus is on **Model 2: Demand-Based Pricing** and **Model 3: Competitive Pricing**. A dedicated section, **“Your Demand Function,”** has been introduced to explain the demand score formula, underlying assumptions, and how prices adapt based on demand and competitor activity.

2. Data Preparation and Preprocessing

2.1 Loading and Cleaning

- The dataset (`dataset.csv`) is loaded using `pandas`.
- A unified `Timestamp` is created by combining `LastUpdatedDate` and `LastUpdatedTime`, followed by sorting the dataframe chronologically.

2.2 Feature Engineering

Feature	Derivation	Rationale
`TrafficLevel`	`{'low': 0.3, 'average': 0.6, 'high': 1.0}`	Standardizes textual traffic data into numerical form.
`VehicleWeight`	`{'bike': 0.3, 'car': 0.6, 'truck': 1.0, 'cycle': 0.2}`	Encodes vehicle size, reflecting price sensitivity.
`OccupancyRate`	`Occupancy / Capacity`	Normalizes occupancy across lots of varying sizes.

The enhanced dataset is saved as `parking_stream.csv` for use in simulated streaming.

3. Streaming Schema and Pipeline Setup

```
class ParkingSchema(pw.Schema):
    Timestamp: str
    Occupancy: int
    Capacity: int
    OccupancyRate: float
    QueueLength: int
    TrafficLevel: float
    IsSpecialDay: int
    VehicleWeight: float
    Latitude: float
    Longitude: float
```

- The stream is simulated using `pw.demo.replay_csv(..., input_rate=100)`, generating ~100 rows per second.
- Two additional columns are created:
  - `t` – a true datetime object.
  - `day` – a truncated timestamp for daily aggregations.

4. Pricing Models

4.1 Model 1 – Enhanced Linear

A weighted linear model with output clipped between 5 and 20 currency units.

4.2 Model 2 – Demand-Based (Primary Model)

Detailed in Section 5.

4.3 Model 3 – Competitive Pricing

Incorporates competitor pricing using **Haversine distance**.

- If a competitor within **500 meters** offers a **lower price** and **your lot's occupancy > 90%**, the price is set to `competitor_price - 1`.
- Otherwise, Model 2's demand-based price is retained.

5. Your Demand Function

5.1 Formula

```
demand = 0.4 * OccupancyRate
      + 0.2 * (QueueLength / 10)
      + 0.2 * TrafficLevel
      + 0.1 * VehicleWeight
      + 0.1 * IsSpecialDay

norm_demand = clip(demand, 0, 1)
price       = 10 * (1 + norm_demand)
final_price = clip(price, 5, 20)
```

5.2 Assumptions

1. **Linear combination** – Each feature independently contributes to the demand score.
2. **Weighted contributions** – Occupancy carries the highest weight (40%) due to its direct correlation with scarcity.
3. **Queue normalization** – Queue length is scaled by 10 to fit within [0,1].
4. **Traffic approximation** – Maps `low/average/high` to `0.3/0.6/1.0` to reflect increasing urgency.
5. **Special days** – A binary flag boosts the score by 10% to account for holidays and events.
6. **Vehicle type** – Heavier vehicles suggest higher willingness to pay, influencing demand slightly.
7. **Pricing elasticity** – The linear relationship between demand and price is a simplification and could be replaced with more nuanced, non-linear models in future iterations.

5.3 Price Behavior Examples

Scenario	OccupancyRate	Queue	Traffic	Vehicle	SpecialDay	norm_demand	Final Price
Quiet	0.2	1	low	bike	0	0.18	≈ ₹12
Busy	0.9	6	high	car	1	0.78	≈ ₹18
Peak	1.0	10	high	truck	1	1.00	₹20

- Prices rise nearly linearly with demand, up to the cap of ₹20.
- Under Model 3, if a nearby competitor offers a lower price and your lot is >90% full, your price will be reduced to `competitor_price − 1`, overriding demand-based escalation.

6. Real-Time Visualization

- A `bokeh` plot with line and circle markers displays price fluctuations over time.
- `panel` is used to wrap the visualization for serving.
- Pathway's `.plot()` function enables live updates, streaming new data continuously.

7. Running the Pipeline

Calling `pw.run()` initiates the full pipeline:

- Real-time data ingestion
- Feature computation
- Demand scoring
- Price determination
- Competitive adjustment
- Visualization update

8. Conclusion

This capstone project demonstrates a complete, end-to-end real-time analytics pipeline for dynamic parking pricing:

- Cleans and enriches raw data.
- Calculates a **transparent, interpretable demand score**.
- Adjusts price responsively within bounds of ₹5–₹20.

- Integrates **competitive pricing logic** to retain customers during high demand.
- Offers **live visualization** to monitor pricing behavior.