Dynamic Urban Parking Pricing System

Capstone Project – Summer Analytics 2025, Consulting & Analytics Club IIT Guwahati
Author : Kangkan Kalita
Email : *kalitakangkan.239@gmail.com*
Google Colab notebook *(Link)*

Table of Contents

1. Introduction

Cities worldwide wrestle with two extremes in urban parking:

- **Overcrowding** during rush hours, leading to long queues and frustrated drivers
- **Underutilization** at off-peak times, leaving revenue unrealized

Static, time-invariant pricing exacerbates both problems. In contrast, a **dynamic pricing engine** adjusts rates in real time in response to demand, capacity, and contextual factors improving both utilization and fairness.

In this project, we design, implement, and validate such an engine for a network of 14 parking lots, using only Python, Pandas, NumPy, and Pathway for streaming simulation. We compare three models of increasing sophistication and deliver both static and interactive real-time dashboards.

2. Problem Statement & Objectives

Goal: Predict an optimal price for each lot at each 30-minute interval, starting from a base price of $10, such that the lot stays neither empty nor overcrowded.

Core objectives :

1. **Implement three pricing strategies** from scratch:
   - Baseline Linear
   - Demand-Based Nonlinear
   - (Optional) Competitive
2. **Simulate real-time ingestion** using Pathway's replay mechanism.
3. **Visualize** historical and static pricing in Google Colab and via a Panel dashboard.
4. **Validate** that prices remain smooth, bounded (50%–200% of base), and correlate sensibly with occupancy and queue data.

---

3. Dataset and Pre-processing

We leverage a 73-day observational dataset from 14 distinct parking lots, sampled every 30 minutes between 8 AM and 4:30 PM. Each record includes :

Feature	Type	Description
`Timestamp`	datetime	Date & time of the observation
`SystemCodeNumber`	string	Unique ID for each parking lot
`Capacity`	int	Maximum number of vehicles
`Occupancy`	int	Current parked vehicles
`QueueLength`	int	Vehicles waiting
`VehicleType`	categorical	Incoming vehicle: car, bike, cycle, truck
`TrafficConditionNearby`	categorical	low, average, high congestion level
`IsSpecialDay`	binary	Holiday or event flag
`Latitude`, `Longitude`	float	Geographic coordinates (for competitive model)

3.1 Pre-processing Steps

1. **Parse & sort** `Timestamp`
2. **Compute**:
   - **OccupancyRate** = Occupancy / Capacity
   - **TrafficLevel** ∈ {0.3, 0.6, 0.9}
   - **VehicleWeight** ∈ {0.5…1.3}

- **BasePrice** = $10 constant

3. **Handle missing** values (none present in primary features).

Code snippet :

```
df['Timestamp'] = pd.to_datetime(df['Date'] + ' ' + df['Time'])
df.sort_values('Timestamp', inplace=True)
df['OccupancyRate'] = df.Occupancy / df.Capacity
traffic_map = {'low':0.3,'average':0.6,'high':0.9}
df['TrafficLevel'] = df.TrafficConditionNearby.map(traffic_map)
veh_wt = {'car':1.0,'bike':0.7,'cycle':0.5,'truck':1.3}
df['VehicleWeight'] = df.VehicleType.map(veh_wt)
df['BasePrice'] = 10.0
```

---

4. Pricing Models

4.1 Baseline Linear Model

A per-lot, time-series update:

$$\text{Price}_{t+1} = \text{Price}_t + \alpha\cdot(\text{Occupancy}/\text{Capacity}),\ \alpha=1.0$$

• **Clipped** to [0.5×Base, 2×Base]
• Fast, interpretable, but doesn't account for queues or events

4.2 Demand-Based Nonlinear Model

A more expressive demand function :

$$\text{raw} = \alpha\cdot o + \beta\cdot(q/C) - \gamma\cdot t + \delta\cdot 1_{\text{Special}} + \epsilon\ \text{veh\_wt}$$
$$\text{score} = \tanh(\text{raw}),\quad \text{Price} = \text{Base}\,[1 + \lambda\cdot\text{score}]$$

Coefficient	**Interpretation**
$\alpha = 0.6$	Occupancy weight
$\beta = 0.3$	Queue weight
$\gamma = 0.2$	Traffic penalty
$\delta = 0.4$	Special day bonus
$\lambda = 0.8$	Price sensitivity

- **tanh** bounds score to (−1,1), so variation ≤ ±80% of base.
- **Clipping** ensures price ∈ [$5,$20].

4.3 Competitive Pricing Model (Optional)

Incorporates nearby lots within a 2 km radius:

- Computes Haversine distance
- Weights competitor prices ∝1/(1+dist)
- Adjusts up or down when your lot is over- or under-priced

---

5. Smoothing & Two-Stage CSV Exports

5.1 `parking_stream_full.csv`

- Contains raw features + timestamp (for Pathway replay)
- Columns: `Timestamp, SystemCodeNumber, Capacity, Occupancy, QueueLength, TrafficConditionNearby, VehicleType, IsSpecialDay, Latitude, Longitude`

5.2 `parking_stream_final.csv`

- Enriched with computed:
  - `OccupancyRate, BaselinePrice, DemandScore, DemandPrice, SmoothedDemandPrice`
- **SmoothedDemandPrice** via EMA (span = 5) to reduce "zig-zag" noise

---

6. Visualization & Dashboard

6.1 Static Bokeh Plot

- Line chart of Baseline vs. (Smoothed) Demand prices over time
- Verified per-lot patterns and range bound
- Histograms of DemandScore distribution

6.2 Interactive Panel Dashboard (Cell 7)

- **Tabs** for each `SystemCodeNumber`
- **Time-series** plot (live-style) + **Daily-avg** bar chart
- Widget controls could be extended to dropdowns or sliders

---

7. Results & Analysis

- **Baseline**
  - Mean price ≈ \$20.0, very little variation (clipping dominates)
  - Low correlation with OccupancyRate (≈ 0.02)
- **Demand Model**
  - Mean price ≈ \$12.6, range \$9.6, \$16.3\$9.6, \\$16.3

    - Correlation with OccupancyRate ≈ 0.63 (stronger responsiveness)
- **Smoothing**
    - EMA effectively removes short-term spikes while preserving trends
- **Daily Slice**
    - Demonstrates smooth peak pricing midday, lower off-peak rates

---

8. Discussion & Future Work

1. **Hyperparameter Tuning**
    - GridSearchCV on α, β, γ, λ using historical occupancy & revenue as objective.
2. **Feature Extensions**
    - Weather data, event schedules, dynamic vehicle-type weighting.
3. **True Real-Time Deployment**
    - Move from CSV replay to real Kafka ingestion via Pathway's connectors.
4. **User Interaction**
    - Allow drivers to request rerouting suggestions when queues exceed thresholds.
5. **Model A/B Testing**
    - Deploy variants in parallel for live evaluation.

---

9. Reproducibility & Deployment

1. Clone repo & `cd` into project.
2. `pip install -r requirements.txt` (pins `pathway`, `bokeh`, `panel`, `pandas`, etc.).
3. Open `Dynamic_Parking_Pricing.ipynb` in Google Colab or Jupyter.
4. Run cells **1 through 8** in order.
5. View interactive dashboard in-cell or via `panel serve`.

---

10. References

1. Pathway documentation (Link)
2. Pathway Github repository (Link)
3. Bokeh & Panel user guides (Link)
4. My Google Colab notebook for reference (Link)