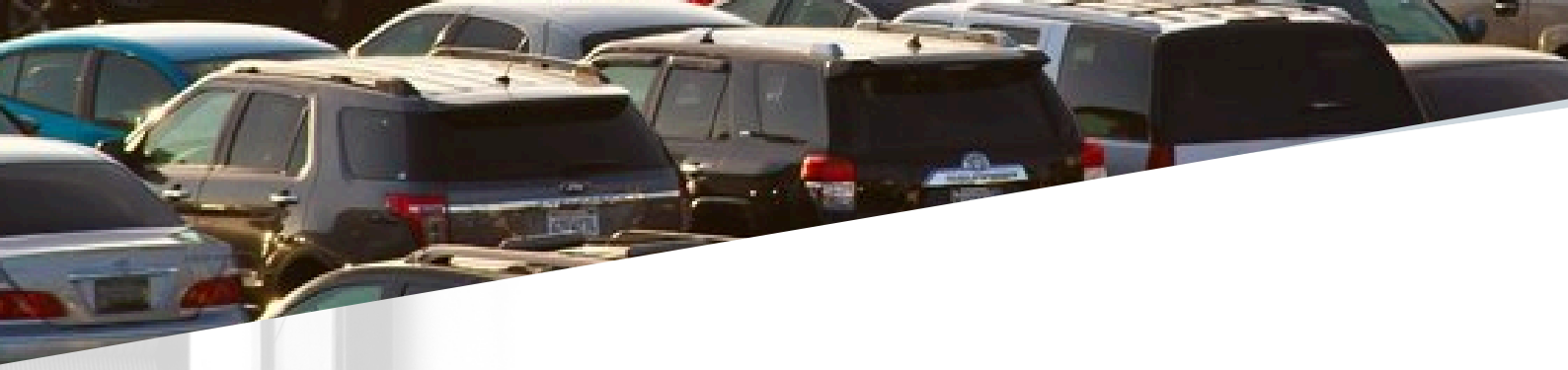


CAPSTONE PROJECT REPORT

DYNAMIC PRICING FOR URBAN PARKING LOTS

-KRITHYAA VIJAYANAND

SUMMER ANALYTICS 2025

This report aims at providing reasoning and additional insights on the implementation of the project

- Urban parking spaces are scarce and face fluctuating demand throughout the day. Static pricing often results in either overcrowding or underutilization of these lots. Dynamic pricing allows for more efficient use by adjusting prices in response to real-time conditions and demand patterns.

- **Project Objective**: To develop an intelligent, data-driven dynamic pricing engine for 14 parking lots using real-time data streams, basic economic principles, and machine learning models implemented from scratch with Python, Pandas, and Numpy.

Methodology
Real-Time Data Simulation
- Used Pathway to simulate real-time streaming of parking data
- Preserved timestamp order to reflect realistic data ingestion
- Enabled continuous feature updates and price predictions

Assumptions:
- Parking capacity doesn't change over time.
- Base price fixed at $10 for all lots initially
- Each row represents a single vehicle.
- Linear weights (α, β, γ, δ, ε) tuned heuristically
- Vehicle Type: Mapped to fixed weights for demand (bike = 0.7, car = 1.0, truck = 1.5).

Pricing Models:
Model 1: Baseline Linear Pricing Model
Logic: simple, intuitive relationship between the parking lot occupancy and the price. The idea is that as the lot fills up, the price should increase linearly to reflect higher demand and discourage excessive congestion.

$$\text{Price}_{t+1} = \text{Price}_t + \alpha \cdot \left(\frac{\text{Occupancy}}{\text{Capacity}} \right)$$

Sensitivity parameter controlling how much the price changes with occupancy

- This model acts as a baseline to show how occupancy directly influences price.
- It is easy to interpret and implement but ignores other important factors like queue length, traffic, or special events.

However, does not take into account change patterns and can lead to sudden jump if there is sharp fluctuation.

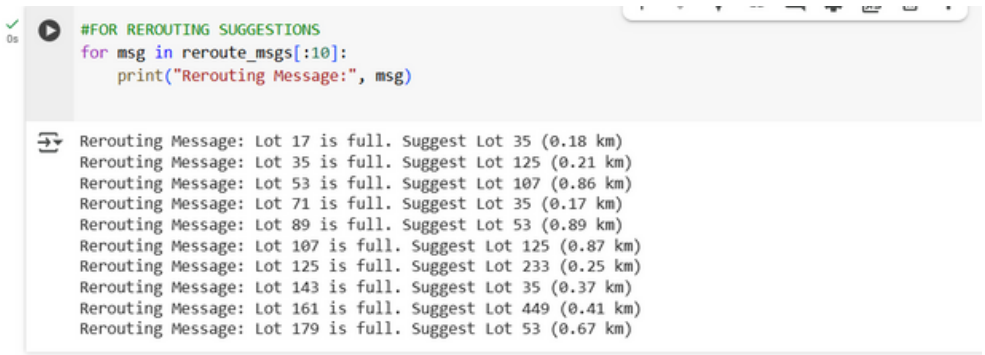
Model 2: Demand-Based Price Function
Logic: To capture more realistic demand dynamics, this model incorporates multiple key factors that influence parking demand and thus pricing. These factors include occupancy rate, queue length (waiting vehicles), traffic congestion, special events, and the type of incoming vehicle.

$$\text{Demand} = \alpha \cdot \left(\frac{\text{Occupancy}}{\text{Capacity}} \right) + \beta \cdot \text{QueueLength} - \gamma \cdot \text{Traffic} + \delta \cdot \text{IsSpecialDay} + \varepsilon \cdot \text{VehicleTypeWeight}$$

- Models multiple factors affecting demand instead of just occupancy.
- Allows for a smooth, bounded price that adapts dynamically and is more reflective of real-world conditions.
- The inclusion of vehicle type and special day captures nuanced effects on demand.

Rerouting suggestion to nearby lots if current lot is overburdened.

This result shows next suggested lot for a particular lot and also shows the distance of it from current lot



```
#FOR REROUTING SUGGESTIONS
for msg in reroute_msgs[:10]:
    print("Rerouting Message:", msg)
```

```
Rerouting Message: Lot 17 is full. Suggest Lot 35 (0.18 km)
Rerouting Message: Lot 35 is full. Suggest Lot 125 (0.21 km)
Rerouting Message: Lot 53 is full. Suggest Lot 107 (0.86 km)
Rerouting Message: Lot 71 is full. Suggest Lot 35 (0.17 km)
Rerouting Message: Lot 89 is full. Suggest Lot 53 (0.89 km)
Rerouting Message: Lot 107 is full. Suggest Lot 125 (0.87 km)
Rerouting Message: Lot 125 is full. Suggest Lot 233 (0.25 km)
Rerouting Message: Lot 143 is full. Suggest Lot 35 (0.37 km)
Rerouting Message: Lot 161 is full. Suggest Lot 449 (0.41 km)
Rerouting Message: Lot 179 is full. Suggest Lot 53 (0.67 km)
```

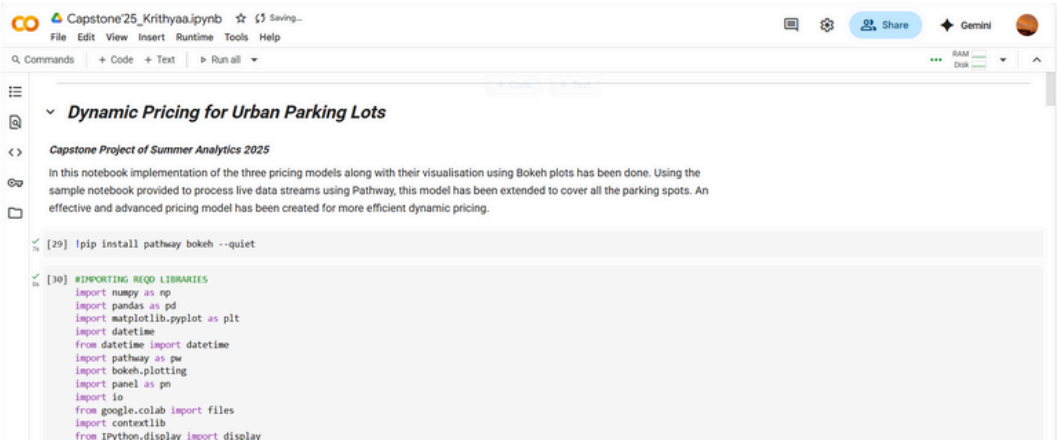
Price change with Demand and Competition

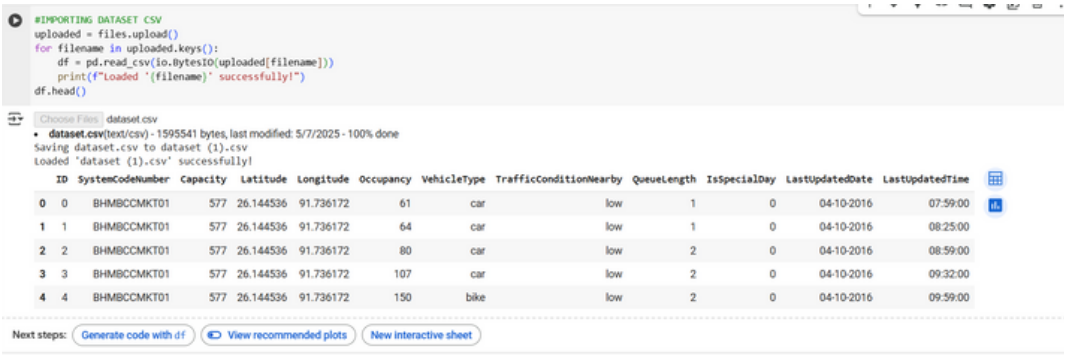
- **Model 1:**
  - Price rises smoothly with average occupancy.
  - Not sensitive to spikes, volatility, or queue build-up

- **Model 2:**
  - Occupancy mainly drives price but queue and vehicle type can amplify it.
  - Special Days increase price by design.
  - Vehicle types like trucks increase the price more than bikes.
  - Prices are capped to avoid instability: [0.5, 2.0].

Visual Results

- Colab working code along with comments

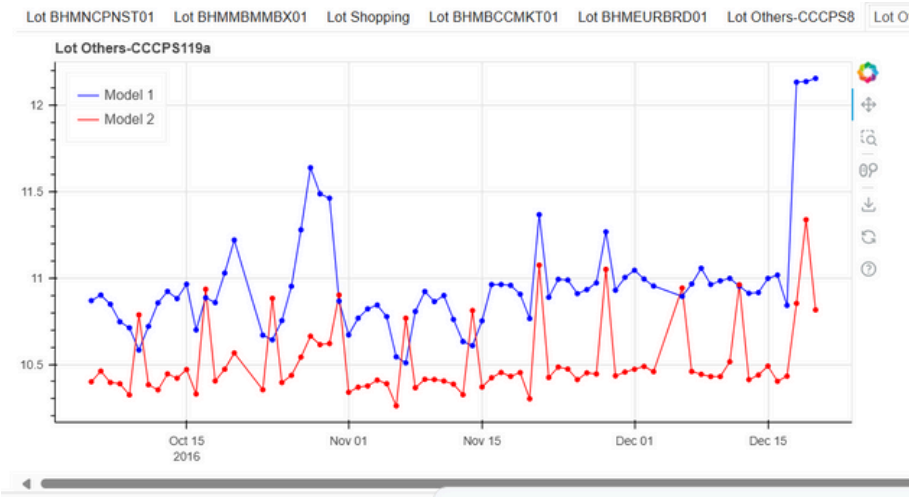




	ID	SystemCodeNumber	Capacity	Latitude	Longitude	Occupancy	VehicleType	TrafficConditionNearby	QueueLength	IsSpecialDay	LastUpdatedDate	LastUpdatedTime
0	0	BHMBCCMKT01	577	26.144536	91.736172	61	car	low	1	0	04-10-2016	07:59:00
1	1	BHMBCCMKT01	577	26.144536	91.736172	64	car	low	1	0	04-10-2016	08:25:00
2	2	BHMBCCMKT01	577	26.144536	91.736172	80	car	low	2	0	04-10-2016	08:59:00
3	3	BHMBCCMKT01	577	26.144536	91.736172	107	car	low	2	0	04-10-2016	09:32:00
4	4	BHMBCCMKT01	577	26.144536	91.736172	150	bike	low	2	0	04-10-2016	09:59:00

- Different tabs to view different parking lots and their trends and how different pricing models works in each

- Comparison of Model 1 and Model 2 for parking lot, Others-CCCPS119a



- Out of the 14 parking lots results can be viewed for any based on user choice
- All of it displayed in colab using inline tabs

Model Comparison
Model 1 (Baseline linear model) vs Model 2 (Demand-based)

Feature	Model 1	Model 2
Parameters Used	Occupancy, Capacity	Occupancy, Capacity, Queue Length, Traffic, Special Day, Vehicle Type
Sensitivity to Demand	Linear, proportional to occupancy	Multifactor, non-linear, more flexible
Pricing behaviour	Stable, predictable pricing	More volatile but bounded, captures real-time urgency and context better.
Customization	Not customized	Can reflect type of vehicle (bike, truck, car)
Use case	Quick benchmark/reference pricing	Sophisticated pricing for revenue optimization

Conclusion:

Model 1 is simple, effective for benchmarking or where data is sparse. However, when more features are provided to our aid implementing Model 2 which calculates price based on demand taking into consideration multiple key factors are what makes this project a truly dynamic pricing for urban parking lots system.
Through this project real-time dynamic pricing and streaming data using Pathway was successfully implemented.