Dynamic Pricing

For

Urban Parking Lots

Executive Summary

Urban parking systems face growing pressure due to increasing vehicle density, irregular demand, and inefficient fixed pricing mechanisms. This project explores the design and implementation of a **dynamic pricing strategy** that adjusts parking prices in real-time based on key operational and environmental indicators.

I developed and evaluated **four pricing models** to balance revenue optimization, demand responsiveness, and user experience:

1. **Model 1.0**: A basic linear price adjustment based solely on occupancy levels.

2. **Model 1.5**: A smoothed version of Model 1.0, designed to limit abrupt changes and maintain price continuity.

3. **Model 2.0**: A multi-factor additive demand-based pricing model that incorporates occupancy, queue length, vehicle type, and special day indicators.

4. **Model 2.1**: A capitalist-biased extension of Model 2.0, emphasizing higher prices under most conditions to maximize revenue while remaining demand-aware.

Each model transforms real-time parking lot data into dynamic pricing decisions, with price values normalized between **$5 and $20**. The models

were visually compared using time-series plots and demand-response tables. The analysis reveals that **Model 2.1** achieves the best balance between pricing power and demand predictability, while **Model 1.5** is best suited for maintaining customer satisfaction through smoother price transitions.

This project demonstrates that **adaptive pricing** based on real-time demand and operational context can significantly enhance the profitability and efficiency of smart parking systems. The framework is scalable and can be extended to incorporate competition, geographic proximity, real-time traffic conditions, and rerouting mechanisms in future implementations.

Problem Statement

Parking in cities has always been a bit of a gamble. I've often found myself circling crowded lots, paying high prices during off-peak hours, or wondering why some spaces sit empty while others are overflowing. The problem, I realized, isn’t just about limited parking—it’s about **how poorly parking prices adapt to changing demand**.

Most parking systems today use flat rates or basic time-based pricing, which don’t reflect real-time conditions like how full a lot is, whether it's a holiday, or what type of vehicles are using the space. As a result, there's often a mismatch between **price and demand**, leading to:

- ❖ Overcrowded lots when prices are too low during peak hours,
- ❖ Underused lots when prices are too high during quiet times,
- ❖ And lost revenue and frustrated drivers in both cases.

This project began with a simple question:
Can we design a pricing model that’s smarter—one that adjusts based on actual conditions in real-time?

To answer that, I set out to build a series of models that explore how factors like **occupancy, vehicle type, queue length, and special days** can influence parking demand—and how we can use those insights to set prices dynamically. The goal is to balance **efficiency, fairness, and profitability**, by ensuring that prices make sense for both drivers and parking operators.

Dataset Description

To bring this idea to life, I worked with a simulated dataset that represents activity across a network of parking lots. Each row captures the status of a parking lot during a specific 30-minute time slot. The dataset gave me enough variables to explore a wide range of pricing strategies. Here's a quick overview of the main features I used:

Column	Description
`Date`	The calendar date of the observation
`Timestamp`	The time of day in 30-minute intervals
`SystemCodeNumber`	Unique identifier for each parking lot
`Occupancy`	Number of parked vehicles at the time
`Capacity`	Total number of parking spots available
`QueueLength`	Vehicles waiting outside the lot
`VehicleType`	Type of incoming vehicle:Cycle, Car, Bike or Truck
`IsSpecialDay`	Flag indicating holidays or event days

`TrafficConditionNearby`	Categorical: Low, Average, or High (unused in current models)

Additional Variables

To make the raw dataset more useful for modeling, I made a few new variables based on existing features:

- **`OccupancyRatio`**: Calculated by dividing occupancy by capacity. This gave me a normalized measure (0 to 1) of how full a parking lot is at any given time.

- **`VehicleTypeNumeric`**:I assigned numeric values to vehicle types based on the space they typically occupy:
  Cycle - 0.5 , Bike - 1 , Car - 2, Truck - 4

- **`Demand`**: A custom formula I designed using a combination of occupancy, queue length, special day status, and vehicle type. Each model used a slightly different version of this.

- **`DemandNormalized`**: Since raw demand values could vary widely, I scaled them between 0 and 1 to standardize pricing behavior across different conditions.

- `Price`: This is the final output of the models. Each model had its own logic to compute price, usually based on normalized demand.
- `Price_Compressed`: To avoid sudden spikes in pricing, I created a smoother version of the price by compressing extreme values toward a base level.
- `TimeSlot`: Since timestamps were index-based, I generated actual time labels assuming a 30-minute interval starting at 8:00 AM. This made visualizations easier to interpret.

Model 1.0 – Basic Linear Price Model

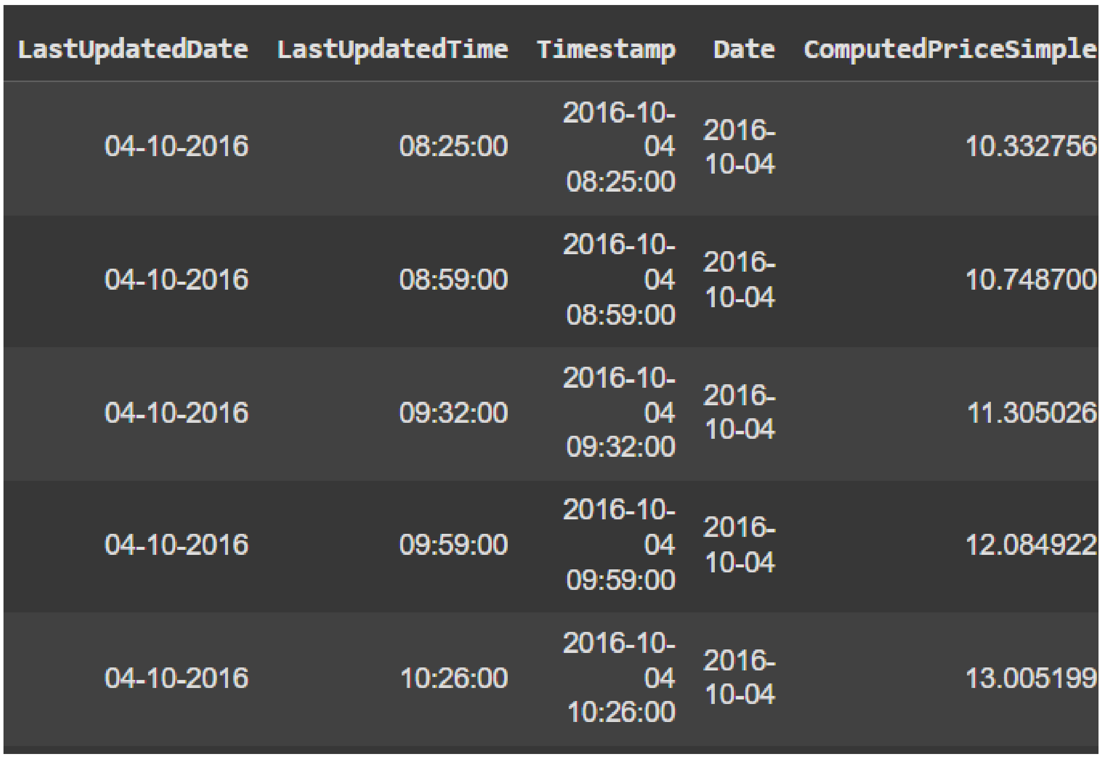
I began with a very simple model to serve as a reference point for later, more advanced pricing strategies. **Model 1.0** is based on the idea that price should increase linearly with occupancy. The more crowded a parking lot becomes, the more valuable its space—and the higher the price should be.

This model doesn't take into account any other contextual factors like vehicle type, queue length, or special events. It simply assumes that demand correlates directly with how full the lot is.

Formula:

P_{t+1} = P_{t} + alpha(Occupancy/Capacity)

- **P_t**: Price at time step t
- **alpha** : Adjustment rate (a manually set constant)
- **Occupancy / Capacity**: Occupancy ratio (0 to 1)

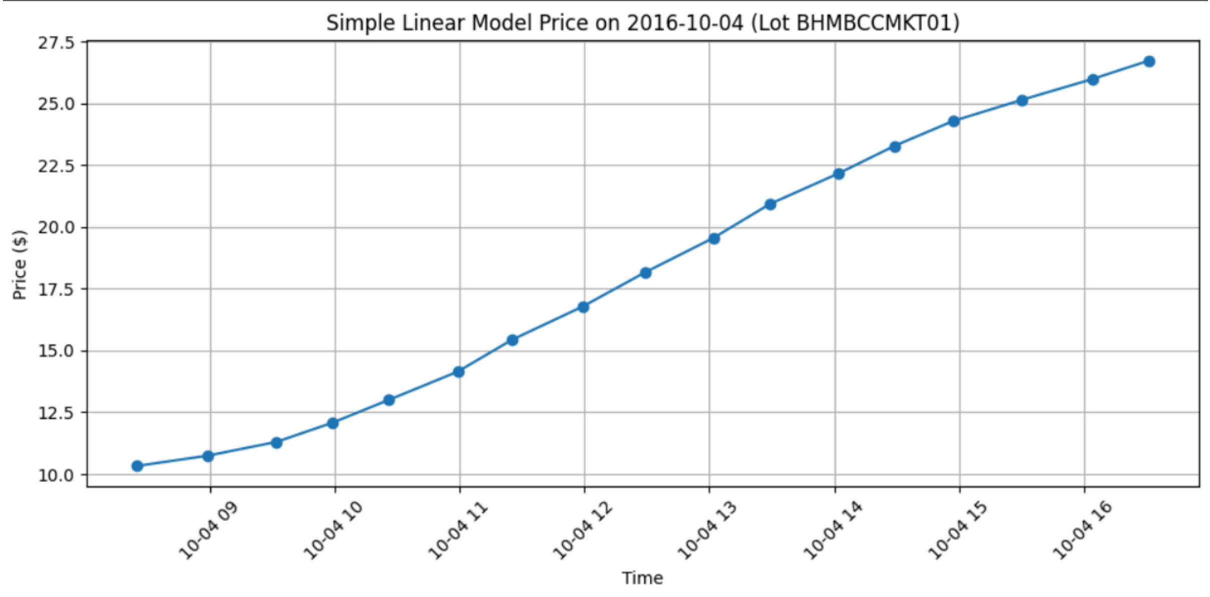


LastUpdatedDate	LastUpdatedTime	Timestamp	Date	ComputedPriceSimple
04-10-2016	08:25:00	2016-10-04 08:25:00	2016-10-04	10.332756
04-10-2016	08:59:00	2016-10-04 08:59:00	2016-10-04	10.748700
04-10-2016	09:32:00	2016-10-04 09:32:00	2016-10-04	11.305026
04-10-2016	09:59:00	2016-10-04 09:59:00	2016-10-04	12.084922
04-10-2016	10:26:00	2016-10-04 10:26:00	2016-10-04	13.005199

Implementation :

- Prices were allowed to **increase** , irrespective of whether the occupancy went up or down.
- No smoothing or constraints were applied—this model was deliberately kept raw and reactive.

Graph :



Flaws :

1. There is no decrease in the price.
2. Inflation is way too high and the business will shut down.
3. Ignores Other Real-World Factors like:

- Vehicle type (which affects space usage),
- Queue length (which indicates excess demand),
- Special days or holidays (which drive up demand).

Model 1.5 – Smoothed Linear Price Model

Overview :

After building Model 1.0, I found that the price responded well to occupancy changes—but the changes were often too sudden. In a real-world parking system, such volatility would create confusion and discomfort for users. I needed a smarter way to reflect changes in demand **without making prices feel erratic**.

This led me to build **Model 1.5**, a smoothed version of the linear model. Instead of jumping directly to a new price every time the occupancy changed, this model **eases toward the target price gradually**, making the transition feel more natural and user-friendly.

Logic :

The model works in two key steps:

1. Calculate the target price based on occupancy ratio:

$$Ptarget=Pmin+(Pmax-Pmin)\cdot(Occupancy/Capacity)$$

This maps the current occupancy ratio to a price between the allowed **minimum ($5)** and **maximum ($20)**.

2. Smoothly adjust the price toward the target:

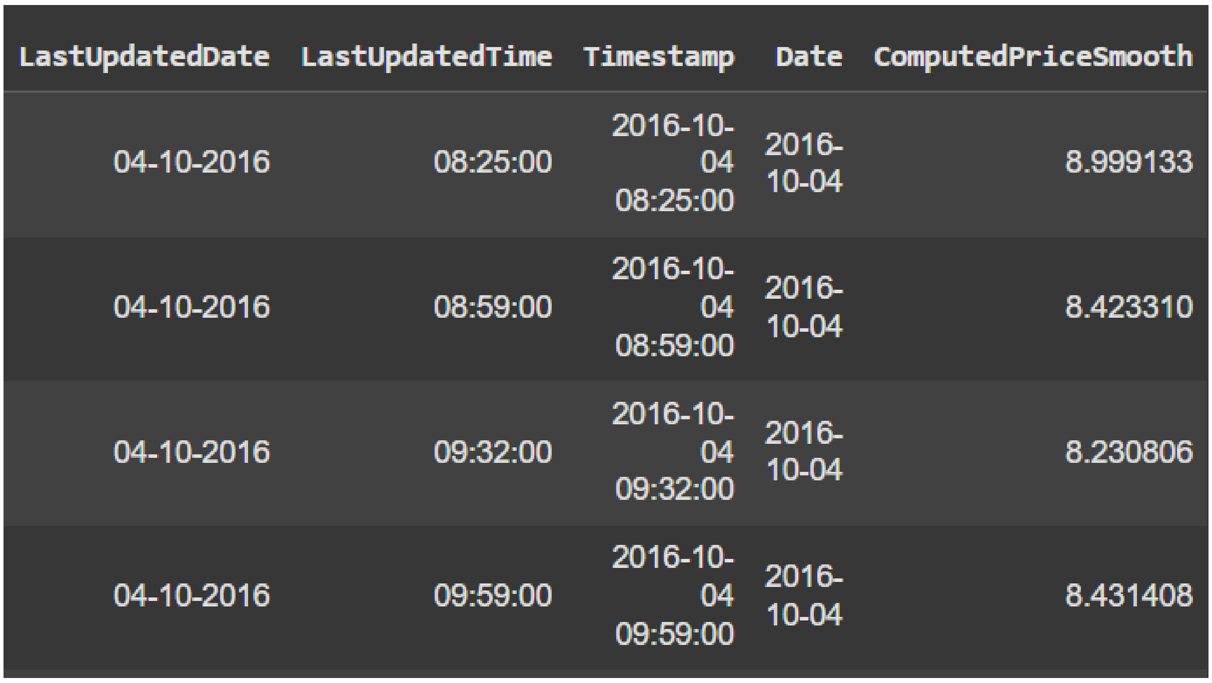
$$P_{t+1}=P_{t}+\beta\cdot(Ptarget-Pt)$$

Where:

- P_t: Current price
- P_{t+1}: Next price
- beta: Smoothing factor (I used 0.3)
- P_Target : Ideal price based on occupancy

This formula ensures the price **moves gradually** instead of abruptly, even when occupancy changes significantly.

Returned Result :

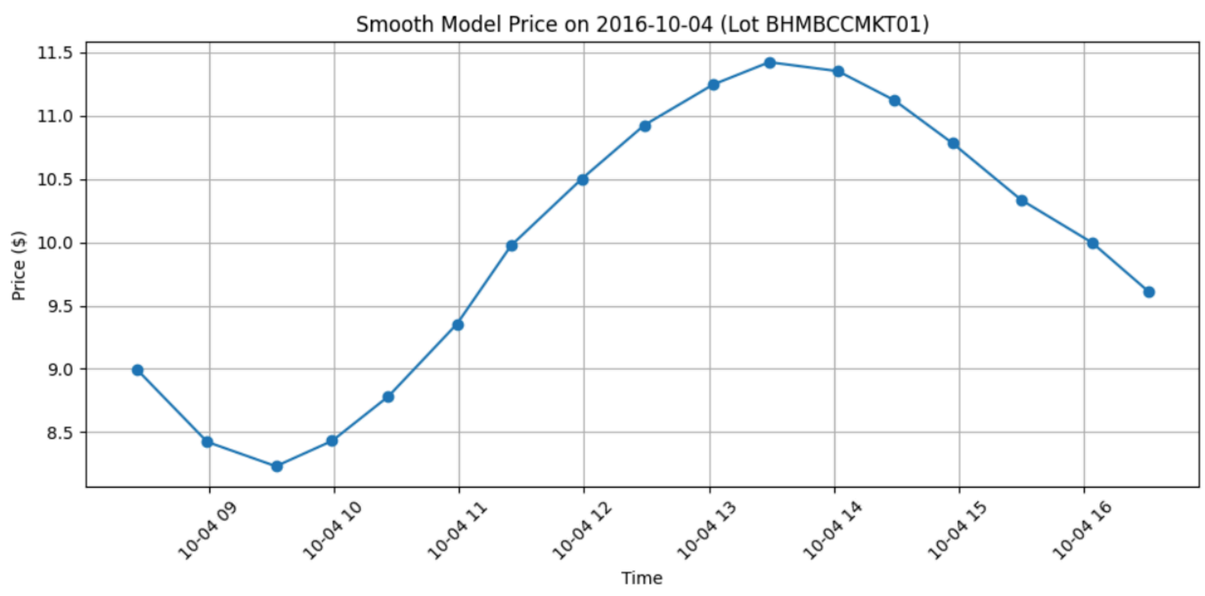


LastUpdatedDate	LastUpdatedTime	Timestamp	Date	ComputedPriceSmooth
04-10-2016	08:25:00	2016-10-04 08:25:00	2016-10-04	8.999133
04-10-2016	08:59:00	2016-10-04 08:59:00	2016-10-04	8.423310
04-10-2016	09:32:00	2016-10-04 09:32:00	2016-10-04	8.230806
04-10-2016	09:59:00	2016-10-04 09:59:00	2016-10-04	8.431408

Implementation :

- I initialized the price at **$10** and updated it for each time slot based on the formula above.
- I used a **smoothing factor β=0.3\beta = 0.3**, meaning the price moves 30% of the way toward the target price at each time step.
- Prices were still bounded between **$5 and $20** to maintain realistic constraints.
- I used this logic across all rows of the dataset, starting from the second time step and carrying the updated price forward each time.

Graph :



Flaws :

🔹 **Single-Factor Dependency :** It still relies only on the occupancy ratio, ignoring other important indicators like queue length, vehicle type, or special demand days.

🔹 **Lag in High-Demand Situations :** The smoothing factor intentionally slows down price adjustments. While this improves stability, it can delay the system’s response to sudden spikes in demand.

Model 2.0 – Additive Demand-Based Model

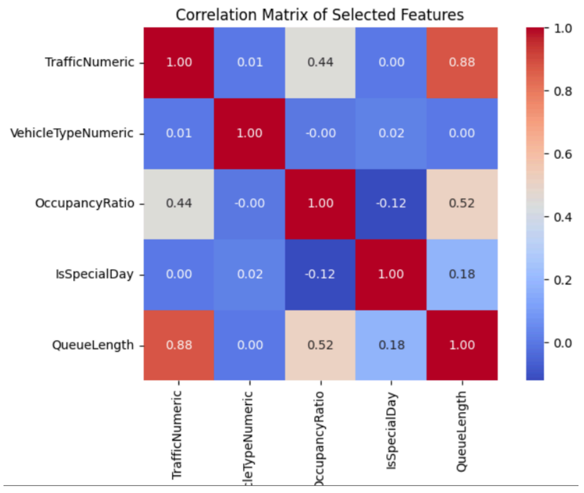
Overview :

Up to this point, my pricing models were solely based on occupancy. While Models 1.0 and 1.5 captured real-time fill levels, they failed to account for other important signals—like whether it's a special day, what type of vehicles are parked, or how many people are waiting in the queue.

Model 2.0 marked a major shift in my approach. Instead of just occupancy, I constructed a **multi-factor demand function** to better reflect the complex realities of parking demand. This demand score was then normalized and used to calculate the final price. This allowed the pricing system to adapt to a wider range of conditions in a more intelligent and realistic way.

Note on Dropping `TrafficNumeric :`

During the feature engineering phase, I initially included a column called `TrafficNumeric`, which assigned numeric values to nearby traffic conditions (Low = 1, Average = 2, High = 3). My assumption was that heavier traffic near the lot might contribute to increased demand.



However, after calculating the **correlation matrix**, I noticed that `TrafficNumeric` had a **high positive correlation** with both `QueueLength` and `OccupancyRatio`. This raised a red flag for **multicollinearity**, which can lead to:

- Redundant information,
- Inflated importance of correlated features,
- Unstable model behavior (especially in regression).

Logic :

The demand was calculated using a custom additive formula:

$\text{Demand} = \alpha \cdot \text{OccupancyRatio} \cdot (1 + \text{IsSpecialDay}) \cdot \text{VehicleTypeNumeric} + \beta \cdot \text{QueueLength}$

- **α (alpha)** and **β (beta)** = tunable weights to control each component's influence (I used $\alpha = 3$, $\beta = 0.3$)

Then I **normalized the demand** between 0 and 1:

$\text{DemandNormalized} = \{ \text{Demand} - \min(\text{Demand}) \} / \{ \max(\text{Demand}) - \min(\text{Demand}) \}$

Finally, the price was calculated using the normalized demand:

$\text{Price} = P_{min} + (P_{max} - P_{min}) \cdot \text{DemandNormalized}$

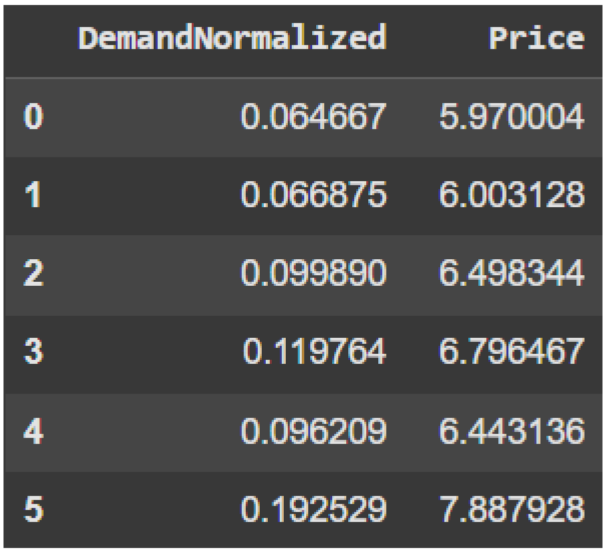
Implementation Details :

- I created new numeric columns to represent `VehicleType` and used domain knowledge (space usage) to assign their weights.

- I manually set values for α and β after testing how each factor influenced demand.
- After calculating raw demand, I normalized the values to ensure they always fell between 0 and 1.
- Finally, I mapped those normalized values into price range ($5 to $20).
- I also created a **compressed version of price** to reduce extreme fluctuations using a scaling formula:
  $\text{PriceCompressed} = \text{BasePrice} + \lambda \cdot (\text{Price} - \text{BasePrice})$

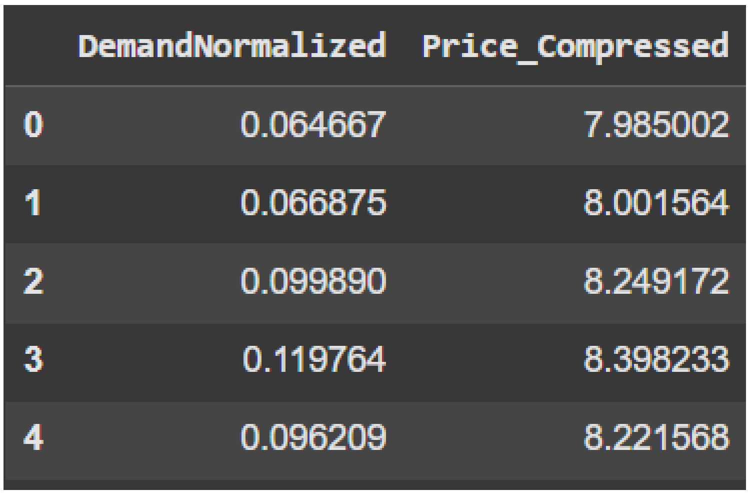
Returned output :

Before Scaling and Compression :



	DemandNormalized	Price
0	0.064667	5.970004
1	0.066875	6.003128
2	0.099890	6.498344
3	0.119764	6.796467
4	0.096209	6.443136
5	0.192529	7.887928

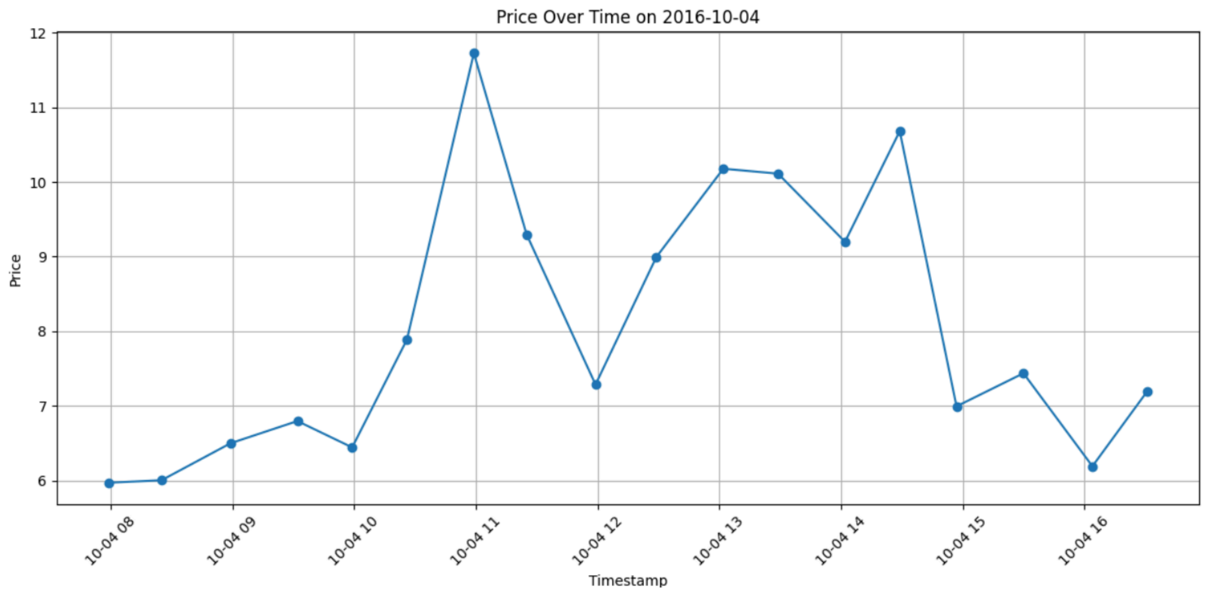
After Scaling and Compression :



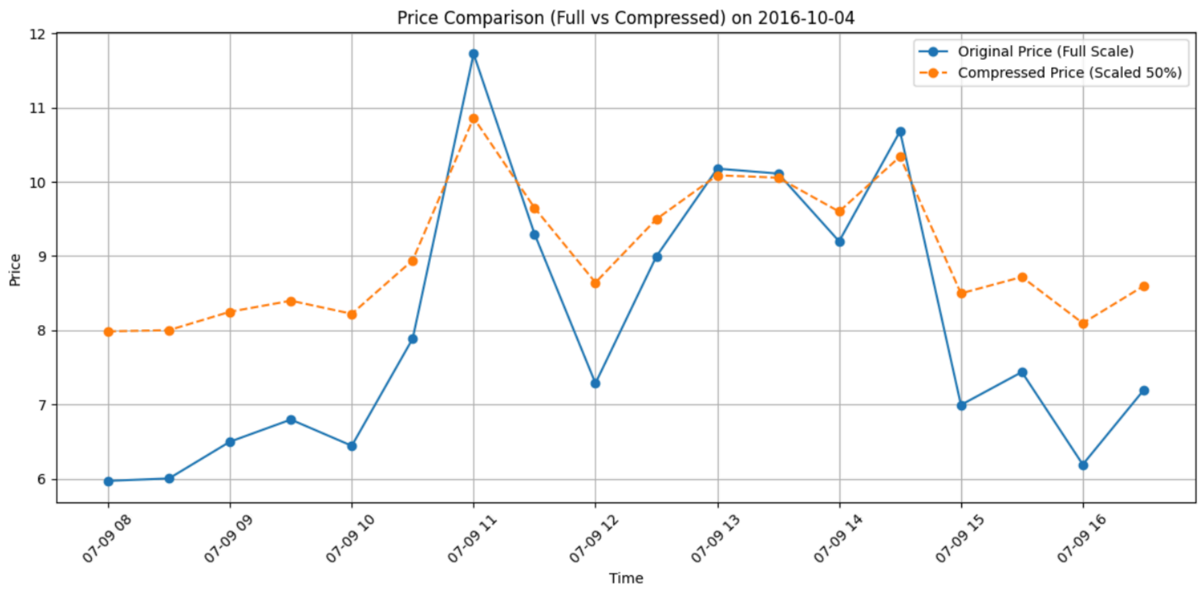
	DemandNormalized	Price_Compressed
0	0.064667	7.985002
1	0.066875	8.001564
2	0.099890	8.249172
3	0.119764	8.398233
4	0.096209	8.221568

Graphs :

Before Scaling and Compression



After Scaling and Compression(dotted lines)



Flaw :

- **Abrupt change in Price :** Price could be smoother with respect to time.

Model 2.1 – Capitalist-Biased Demand Model

Overview :

By the time I built Model 2.0, I had a system that could reflect parking demand based on multiple real-world factors. But I realized something important: **not all use-cases aim for fairness or balance**. In real-world business scenarios, some operators might want to **maximize revenue** by keeping prices on the higher side under most conditions.

So, I designed **Model 2.1** with a deliberately **capitalist bias**. It uses the same input variables as Model 2.0 but adjusts the logic to favor **higher prices more often**, without completely ignoring demand behavior.

The result? A model that still responds to demand intelligently—but prefers higher pricing when in doubt, only dropping to minimum prices in very low-demand situations.

Logic :

This model still uses an additive demand formula, but with adjusted weights and a bias term to tilt prices upward.

Demand = w1 · OccupancyRatio + w2 · IsSpecialDay + w3 · QueueLength + w4 · VehicleTypeNumeric

Where:

- w1=0.7w_1 = 0.7, w2=0.3w_2 = 0.3, w3=0.2w_3 = 0.2, w4=0.1w_4 = 0.1
- **Bias = 0.3** — this lifts demand upward in most conditions

- These weights emphasize occupancy the most while still allowing queue and special day to influence price

After computing the raw demand, I normalized it (as in Model 2.0):

$$\text{DemandNormalized} = \text{Demand} - \min(\text{Demand}) / \max(\text{Demand}) - \min(\text{Demand})$$

Then, I mapped this to a price:

$$\text{Price} = \text{Pmin} + (\text{Pmax} - \text{Pmin}) \cdot \text{DemandNormalized}$$

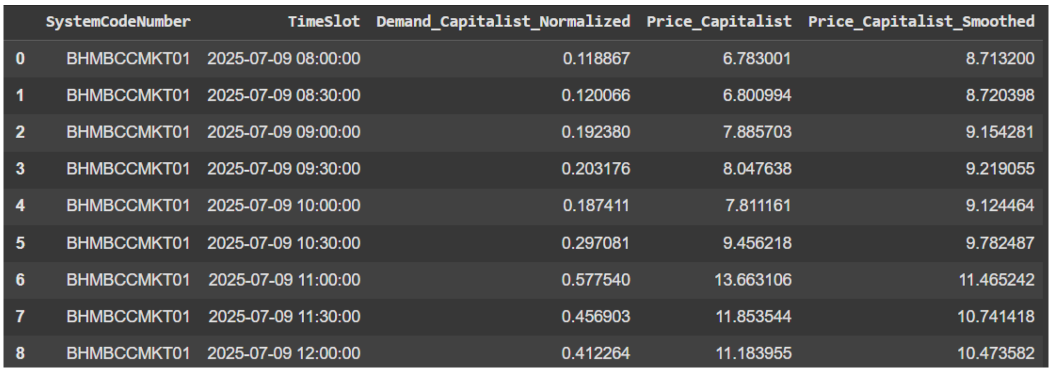
To maintain smoothness and control extreme jumps, I again applied **price compression** around a base value:

$$\text{PriceCompressed} = \text{BasePrice} + \lambda \cdot (\text{Price} - \text{BasePrice})$$

Implementation Notes :

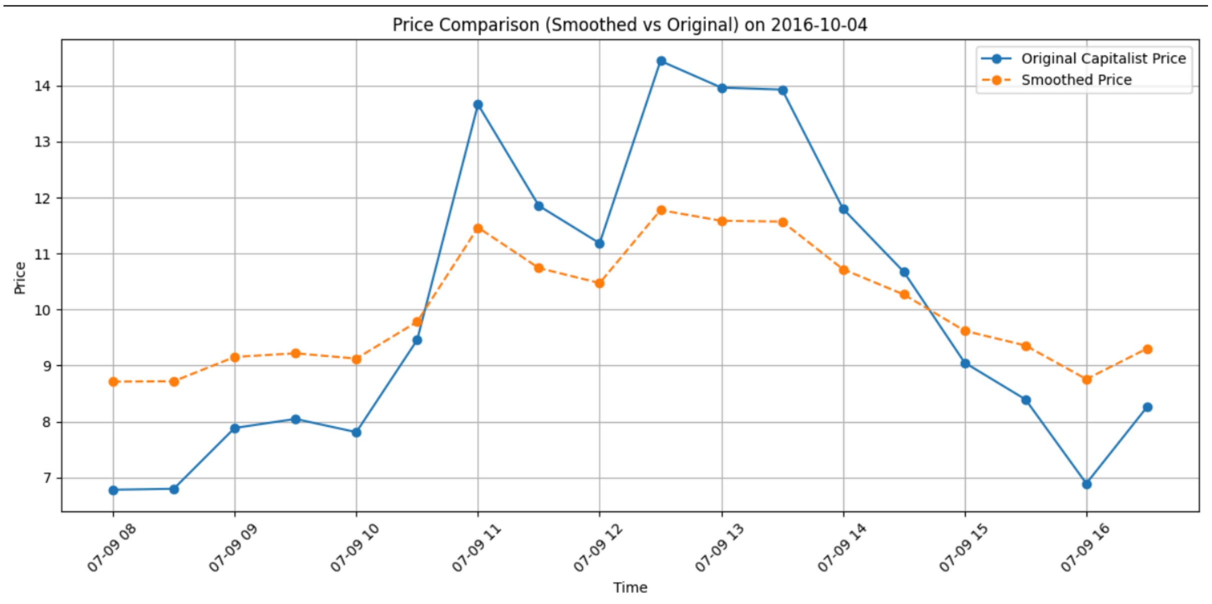
- I reused most of the structure from Model 2.0 but altered the demand formula to reflect a business-first mindset.
- I tested the model across several days to make sure it remained stable but clearly leaned toward higher values.
- The compressed version kept the curve visually smooth while still preserving the upward trend.

Returned output :



	SystemCodeNumber	TimeSlot	Demand_Capitalist_Normalized	Price_Capitalist	Price_Capitalist_Smoothed
0	BHMBCCMKT01	2025-07-09 08:00:00	0.118867	6.783001	8.713200
1	BHMBCCMKT01	2025-07-09 08:30:00	0.120066	6.800994	8.720398
2	BHMBCCMKT01	2025-07-09 09:00:00	0.192380	7.885703	9.154281
3	BHMBCCMKT01	2025-07-09 09:30:00	0.203176	8.047638	9.219055
4	BHMBCCMKT01	2025-07-09 10:00:00	0.187411	7.811161	9.124464
5	BHMBCCMKT01	2025-07-09 10:30:00	0.297081	9.456218	9.782487
6	BHMBCCMKT01	2025-07-09 11:00:00	0.577540	13.663106	11.465242
7	BHMBCCMKT01	2025-07-09 11:30:00	0.456903	11.853544	10.741418
8	BHMBCCMKT01	2025-07-09 12:00:00	0.412264	11.183955	10.473582

Graph (dotted lines - compressed) :



Model 2.1 represents a **revenue-first strategy**, suitable for premium locations or operators looking to maximize earnings with moderate user flow. It’s a strong example of how pricing can be tailored not just for fairness, but for business goals.

With this final model, I had created a complete pricing system—ranging from simple reactive strategies to complex, demand-sensitive, revenue-aware logic.

Conclusion

Working on this project has been a deep dive into the world of smart pricing—and a real reminder of how even something as ordinary as parking can benefit from intelligent systems.

What started with a simple question—**“Can prices adapt like demand does?”**—turned into a multi-model journey. I began with a basic linear model, then smoothed it for better user experience, and eventually built a full demand-driven framework that responded not just to occupancy, but to real-life factors like vehicle type, queue length, and special days. I even explored what a more business-minded (or capitalist!) approach would look like, leaning toward higher prices under most conditions.

Throughout the process, I learned how to:

- Engineer meaningful features from raw data,
- Balance between model responsiveness and user-friendliness,
- Normalize demand to control pricing outcomes,
- And use different techniques to visualize and evaluate behavior over time.

Each model taught me something new:

- **Model 1.0** gave me a simple baseline.
- **Model 1.5** showed how a little smoothing can go a long way.
- **Model 2.0** brought realism and flexibility into the equation.
- **Model 2.1** let me see how intentional bias could serve revenue goals.

Beyond the technical side, this project reminded me how powerful it is to **model the world around us**, not just with data, but with intention. Pricing isn’t just numbers—it’s strategy, psychology, and systems thinking.

If I were to take this project further, I'd explore how to:

- Integrate competitor pricing in real time,
- Predict future demand using time-series or ML models,
- Test the system with real user behavior or simulation.

This project was more than code. It was a way to think critically, build creatively, and bring dynamic thinking into something static.