

BA Salary Growth India

Ayo Raj, this is straight-up 🔥 — you've got the setup for a **banger capstone project** that blends real-time systems, economic logic, and machine learning vibes... all with the flavor of city chaos and parking drama. Let's unpack this beast and build a **roadmap from scratch** — like a pro data analyst who drinks data for breakfast ☕📊.

💡 **Project Title: Dynamic Pricing Engine for Urban Parking Spaces**

🔍 **Step 1: Understand the Problem**

You're building a **dynamic pricing model** for 14 parking lots, working with **real-time data** across:

- Time (30-min intervals, 8AM–4:30PM)
- Location (lat, long)
- Features like:
 - Occupancy
 - Queue Length
 - Traffic congestion
 - Special event days
 - Vehicle type (car, bike, truck)
 - Competitor prices

🧠 **Goal?**

- Price starts at \$10
- Adjusts *smoothly* in real time
- Based on demand, competition, and congestion
- Bonus: reroute vehicles if needed (like Google Maps but for parking)

Step 2: Peek at the Dataset

Let's explore `/mnt/data/dataset.csv` — we'll load it and look at a sample:

Sample Load Code (Colab)


```
python

import pandas as pd

df = pd.read_csv('/mnt/data/dataset.csv')
df.head()
```

Look for columns like:

- `timestamp`
- `lot_id`
- `latitude`, `longitude`
- `capacity`, `occupancy`, `queue_length`
- `traffic_level`
- `special_day`
- `vehicle_type`
- `competitor_prices`

 **First Task:** Clean the data. Handle missing values, parse timestamps, encode categorical stuff (vehicle types, etc.)

Step 3: Build the Models — Like Levels in a Game

Model 1: Baseline Linear Model

Simple linear relationship:

```
python
```

```
next_price = current_price + alpha * (occupancy / capacity)
```

- `alpha` can be tuned (start with 1)
- Bound the price between \$5 and \$20

🔄 Use a rolling or lag feature to simulate “real-time” pricing.

🧠 Model 2: Demand-Based Price Function

Build a demand score:

```
python
```

```
demand =  $\alpha$  * (occupancy / capacity) +  $\beta$  * queue -  $\gamma$  * traffic +  $\delta$  * special_day +  $\epsilon$  * vehicle_weight
```

Map demand to price:

```
python
```

```
price = base_price * (1 +  $\lambda$  * normalized_demand)
```

Normalize the demand between 0 and 1. Ensure smooth variation using rolling averages.

🥷 Model 3: Competitive Pricing Model (Boss Level)

Add geo-intelligence:

- Compute distance between parking lots using lat/long (Haversine formula)
- Compare your lot's price vs. nearby lots

Rules:

- Nearby cheaper lots? Maybe lower your price
- You're near expensive lots? Raise your price a bit
- Too full? Suggest rerouting to cheaper lots nearby

Step 4: Real-Time Simulation with Pathway

You'll simulate streaming data:

- Use **Pathway** to simulate delayed data streams
- Insert your pricing logic as a hook

You'll need to follow [Pathway's guide](#) to ingest and process time-ordered events.

Step 5: Visualize It Real-Time (with Bokeh)

Plot:

- Line chart of price vs. time per lot
- Occupancy & demand trends
- Comparison of your lot vs. competitors nearby

You'll need:

```
python
```

```
from bokeh.plotting import figure, show
```

Animate the charts in near real-time in your Colab notebook!

Step 6: Reporting and Justification

Create a clean Google Colab report with:








- Description of models
- Assumptions
- Price behavior analysis
- Real-time screenshots
- Explainable ML logic

- Creativity in rerouting logic

Bonus Tips:

- Use `groupby(['lot_id', 'timestamp'])` to handle multi-lot time-series
- One-hot encode vehicle type: car, bike, truck = 1, 0.5, 2 (weights)
- Use rolling mean or EWMA to smooth price transitions

TL;DR — Your BA/DA Roadmap

Phase	What You Do
 Data Wrangling	Clean + explore the dataset
 Model 1	Linear pricing logic based on occupancy
 Model 2	Demand-based pricing using all features
 Model 3	Smart pricing with competitor intelligence
 Pathway	Simulate streaming inputs
 Bokeh	Real-time visuals
 Report	Justify models, show insights, shine ✨

Want me to help you **start building Model 1**, or maybe design the **demand function** for Model 2 together? Say the word, Raj — we'll build this capstone into a résumé gem 💎

Alright Raj, let's start this capstone journey with a bang — step by step, from **dataset dissection** to **model-building mastery**, and oh yes, making this entire experience *classic* like vinyl records and timeless flow 🎧📊

🌟 Step 1: The Data Awakens — Exploration & Understanding

We cracked open the dataset and here's the vibe:

Dataset Summary

- 14 unique parking lots (`SystemCodeNumber`)
- 18,368 records, covering 73 days at 18 time points per day
- Every 30 min from 8 AM to 4:30 PM — consistent, beautiful time series
- Columns include:

Feature	Type	Notes
<code>Latitude</code> , <code>Longitude</code>	float	For geo-analysis in Model 3 🌐
<code>Capacity</code> , <code>Occupancy</code>	int	Core for demand modeling 🚗
<code>QueueLength</code>	int	Reflects demand pressure
<code>VehicleType</code>	object	Car, bike, truck, or others
<code>TrafficConditionNearby</code>	object	Categorical: low, medium, high
<code>IsSpecialDay</code>	int	1 = holiday/event day, 0 = normal
<code>LastUpdatedDate</code> & <code>Time</code>	object	Must be parsed to datetime

This data is a *living pulse* of a city's parking chaos — and we're about to make sense of it.

Step 2: Data Preprocessing

Before modeling, we'll:

- 🧠 Convert `LastUpdatedDate` + `LastUpdateTime` → single datetime
- 🎨 One-hot encode `VehicleType` and `TrafficConditionNearby`
- 📊 Normalize continuous features: `Occupancy` , `QueueLength` , etc.

Coming next:

```
python
```

```
df['timestamp'] = pd.to_datetime(df['LastUpdatedDate'] + ' ' + df['LastUpdatedTime'])
```



Step 3: Model 1 – Baseline Linear Pricing

🎯 Logic: Price increases linearly with occupancy

```
python
```

```
price_t+1 = price_t +  $\alpha$  * (occupancy / capacity)
```

- Start with a base price = \$10
- Bound price between \$5 and \$20
- α = scaling factor (start with 2 or tune it)
- Smooth with a rolling average (optional)

✨ **Purpose:** Establish baseline & visual patterns



Step 4: Model 2 – Demand-Driven Pricing

🎯 Create a demand function:

```
python
```

```
Demand =  $\alpha$  · (Occupancy / Capacity) +  $\beta$  · QueueLength -  $\gamma$  · TrafficLevel +  $\delta$  · IsSpecialDay +  $\epsilon$  · VehicleWeight
```

Then scale price:

```
python
```

```
Price = Base * (1 +  $\lambda$  × NormalizedDemand)
```

👉 You'll need to:

- Assign numeric weights for `VehicleType` (e.g., car=1, bike=0.5, truck=1.5)
- Convert traffic levels: low=0, medium=1, high=2
- Normalize final demand between 0 and 1

✨ **Purpose:** Adaptive, intelligent pricing model

Step 5: Model 3 – Competitive Pricing (Boss Level)

🎯 Add location-based price adjustment:

- Use haversine formula to find nearby lots
- Compare their prices
- Adjust yours if you're too expensive or too crowded

Also:

- Suggest rerouting if:

```
python
```

```
occupancy >= 0.9 * capacity and cheaper lot nearby
```

✨ **Purpose:** Strategic pricing, real-world realism

Step 6: Real-Time Simulation with Pathway

You'll simulate a streaming setup:

- Load data with delays
- Trigger your pricing model
- Output predicted price in time-order

✨ Use:

- [Pathway Streaming Guide](#)



Step 7: Visualization with Bokeh

🎨 Plot:

- Price vs. time (line chart)
- Competitor comparison per lot
- Real-time demand heatmaps?

✨ Purpose: Justify your price model visually in real-time



Step 8: Reporting – Tell the Story

Include in final notebook:

- Each model logic + formula
 - Demand function derivation
 - Assumptions and smoothing logic
 - Tradeoffs and why your pricing is “smart”
-



Ready to Ride? Next Steps:

Let's start with **Model 1** implementation:

- Clean timestamp
- Compute base price
- Plot how it evolves with occupancy

Wanna dive in? Shall we code the **cleaning + Model 1 logic** together now? Let's build the *rhythm of revenue* 📊📈

Yesss Raj! 🎉 We just dropped Model 1 — and oh boy, it's running smooth like butter on a hot stack of pancakes.

Let me walk you through this beauty like it's a lyrical tour 🎵:

🔥 MODEL 1: Baseline Linear Pricing — *The Foundation Flow*

📌 What We Did

✅ Combined the Date & Time:

We made a beautiful `timestamp` column, converting `"04-10-2016"` + `"07:59:00"` into full-fledged datetime.

```
python
```

```
df['timestamp'] = pd.to_datetime(df['LastUpdatedDate'] + ' ' + df['LastUpdatedTime'])
```

Now everything is aligned in time — just like a real-time traffic clock 🚦

✅ Calculated the Occupancy Rate:

We now know **how packed each lot is**, every 30 mins:

```
python
```

```
df['OccupancyRate'] = df['Occupancy'] / df['Capacity']
```

So if a lot is 80% full, it's `0.8` — and that becomes fuel for our price engine. 🧠⚙️

✅ Implemented Linear Pricing Logic:

We used the formula:

```
python
```

```
price_next = price_now + α * occupancy_rate
```

- **Base Price** = \$10
- **Alpha (α)** = 2 (you can tune it later)
- **Clamped** between \$5 and \$20

It's elegant, explainable, and rising just like your BA career 📈

🔍 Sample Output for Lot: BHMBCCMKT01

Time	OccupancyRate	Model1_Price
07:59 AM	10.6%	\$10.00
08:25 AM	11.1%	\$10.22
09:59 AM	25.9%	\$11.38
11:59 AM	44.9%	\$14.52
12:29 PM	46.1%	\$15.44

So the more packed it gets, the more the price climbs. And if a lot chills out? Prices won't spike — they'll glide like logic.

🗺️ What's Next?

We've set the **baseline rhythm**.

Now, shall we:

1. 🎯 **Plot** this pricing over time for visual satisfaction?
2. 🔬 Move to **Model 2 (Demand-Based)** where we build a real demand score using traffic, queue, and vibes?
3. 🚀 Set up **real-time simulation** using a delay loop + plotting?

Your call, commander! 📁💡

