Capstone Project Report

Title: Urban Parking System Optimization

Author: Tarushi Agarwal
Github: Tarushiiiii
Repo: SA-Capstone-Project–Tarushi
Date: July 2025
Technologies: Python, Pathway, Bokeh & Panel, Pandas, GitHub

---

1. Project Objective

To design a **real-time, data-driven pipeline** that:

1. **Dynamically prices** parking based on occupancy and demand.
2. **Recommends optimal parking locations for users based on** current conditions.
3. Simulates **streaming data workflows** using Pathway.

---

2. Repository Structure

```
├── Model_1_SA_Capstone_Project.ipynb
├── bokeh_plot_model-1.png
├── Model_2_SA_Capstone_Project.ipynb
├── bokeh_plot_model-2.png
├── Model_3_SA_Capstone_Project.ipynb
├── bokeh_plot_model-3.png
├── model_1_sa_capstone_project.py
├── model_2_sa_capstone_project.py
├── model_3_sa_capstone_project.py
├── project_report.pdf
└── README.md
```

Each model includes both a colab/jupyter notebook and a Python script for reproducibility, along with outputs and visuals.

---

3. Model 1: Baseline Linear Pricing

Logic: Adjusts price incrementally using the below mentioned logic:

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

Working:

- ParkingSchema1 is defined to store the data in the table based on the appropriate datatype.
- The model identifies the maximum capacity and the occupancy in real time and adjusts the price accordingly.
- If the slots the already occupied, the price increases subsequently row-wise.

Implementation Highlights:

- Uses **Pathway's reducer** to track price per lot.
- Streams parking data via CSV replay.
- Produces output visualization plotted in real-time panel extension and saved as bokeh_plot_model-1.png in the repository.

---

4. Model 2: Demand-Based Pricing

Logic: Predicts the parking price based on the inputs:

- Capacity: Total capacity of the parking lot.
- Occupancy: Number of lots occupied.
- Queue Length: Number of vehicles in the waiting queue
- Traffic: Based on the traffic congestion
- Special Day: Flag for marking if it is any special occasion.
- Vehicle Type: Higher price for heavier vehicles and lower for lighter.

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

Use this demand value to adjust prices:

$$\text{Price}_t = \text{BasePrice}\cdot(1+\lambda\cdot\text{NormalizedDemand})$$

Working:

- The data is pre-processed based on UDFs to make it consistent and easier for model predictions.
- ParkingSchema2 is defined to store the data in the table based on the appropriate datatype.

- The model uses input parameters and hyperparameters—alpha, beta, gamma, delta, and epsilon—to predict demand-based pricing.
- It is based on the idea that the price increases by alpha times as availability decreases, and by beta times as waiting times grow. The price decreases if there is heavy traffic to accommodate mass, and it increases on special days like national holidays or concerts to maximize profits. Heavier vehicles are charged higher prices.
- Finally, the overall price is normalized to stay within 0.5x to 2x the base price (10 Rs).

Implementation Highlights:

- User-defined functions to encode vehicle type, traffic score, and IsSpecialDay flag.
- Real-time demand calculation and normalization.
- Produces output visualization plotted in real-time panel extension and saved as bokeh_plot_model-2.png in the repository.

---

5. Model 3: Competitive Pricing Model : Routing & Recommendation

Logic:

Score = w1 * (1 - Occupancy / Capacity)

- w2 * QueueLength

- w3 * Traffic

- w4 * HaversineDistance

The model recommends the lot with the **maximum score per timestamp**.

Working:

- The data is pre-processed based on UDFs to make it consistent and easier for model predictions.
- ParkingSchema3 is defined to store the data in the table based on the appropriate datatype.
- The model uses input parameters and hyperparameters—w1, w2, w3, and w4—for developing competitive pricing model.
- The haversine function computes the straight-line distance (in kilometres) between two geographic points using latitude and longitude.
- The compute_score function combines availability, queue length, traffic level, and distance using weighted subtraction to rank parking lots.
- Higher scores indicate better options for routing, balancing space, proximity, congestion, and wait time.
- The recommendation groups parking data by timestamp and computes the lot with the highest routing score using Pathway's argmax reducer, returning the recommended coordinates and associated score per time group.

Implementation Highlights:

- Parses Latitude, Longitude, and computes Haversine distance to a fixed user location.
- Calculates composite scores using UDFs.
- Groups and selects the top lot with pw.reducers.argmax.
- Produces output visualization plotted in real-time panel extension and saved as bokeh_plot_model-3.png in the repository.

---

6. Summary of Achievements

Model	Key Feature	Output
1	Stateful occupancy pricing	Time-series prices per lot
2	Demand-based pricing using features	Dynamic price per row
3	Routing with geographic & cost logic	Lot recommendation per time

---

7. How to Run

Prerequisites: Python 3.10+, install via pip install pathway bokeh panel pandas.

1. **Clone the repository**

```bash
git clone https://github.com/Tarushiiiii/SA-Capstone-Project--Tarushi.git
cd SA-Capstone-Project--Tarushi
```

2. **Run a specific model notebook or script**

```bash
python model_1_sa_capstone_project.py
```

3. **Or open the respective `.ipynb` file in Google Colab or Jupyter**.
4. View **outputs and Bokeh plots** in saved PNGs or serve Panel widgets.

8. Conclusion

This capstone demonstrates end-to-end real-time analytics:

- **Streaming data ingestion**
- **Stateful and stateless ML**
- **Geospatial routing**
- **Interactive visualizations**

It’s a robust foundation for smart-city parking solutions—dynamic, scalable, and extension-ready.