Dynamic Pricing for Urban Parking Lots — Project Report

Overview

This project implements a **real-time dynamic pricing system for urban parking lots** using:

- **Pathway** for streaming data ingestion and processing
- **Pandas** for preprocessing and cleaning
- A **demand-based pricing model** incorporating multiple factors
- Output aggregation in daily windows

The goal is to simulate how parking prices can dynamically adapt to occupancy, queue lengths, traffic, vehicle type, and special days.

Objectives

- Preprocess historical parking data and clean inconsistencies
- Simulate a streaming data pipeline
- Compute daily demand and prices for each parking lot
- Export results for analysis

Step 1 — Data Cleaning and Preparation

Rationale

The original dataset contained:

- Timestamps split across two columns

- Textual labels in numeric fields (e.g., `"low"`, `"high"`, `"average"`)
- Potential mixed types

Cleaning was essential to avoid parse errors during ingestion.

Actions

- Combined `LastUpdatedDate` and `LastUpdatedTime` into a single `Timestamp`
- Mapped `VehicleType` (e.g., car, truck) to numeric weights
- Replaced text labels with numeric equivalents:
  - `"low"` → `0.2`
  - `"medium"` → `0.5`
  - `"high"` → `0.8`
  - `"average"` → `0.5`
- Converted all numeric columns (`Occupancy`, `Capacity`, `QueueLength`, `TrafficLevel`, `IsSpecialDay`, `VehicleTypeWeight`) to numeric types
- Filled any missing values with `0`

Outcome: Clean, sorted CSV ready for streaming.

Step 2 — Pathway Schema Definition

Rationale

Pathway requires a **strict schema** for ingestion. Each column must have a consistent type.

Schema:

Column	Type	Description
Timestamp	str	Combined date and time of record
SystemCodeNumber	str	Unique parking lot identifier
Occupancy	int	Current occupancy
Capacity	int	Maximum capacity
QueueLength	int	Vehicles waiting
TrafficLevel	float	Nearby traffic condition (0–1)
IsSpecialDay	int	Flag for holiday/special event
VehicleTypeWeight	float	Multiplier for vehicle type impact

Step 3 — Data Ingestion via Pathway Streaming

Rationale

We simulated real-time data ingestion with `pw.demo.replay_csv`, enabling incremental processing.

This allowed:

- Continuous updates
- Rolling window calculations

Input rate: 500 rows per second (configurable)

Step 4 — Timestamp Parsing and Feature Extraction

We extracted:

- `t`: Timestamp for event time

- `day`: The start of the day (used for daily grouping)

This supports **tumbling windows** per day.

Step 5 — Tumbling Window Aggregation

Rationale

Dynamic pricing is calculated **per day per parking lot**.

Windowing Model:

- **Tumbling Window:** Daily interval
- **Instance Key:** `SystemCodeNumber + "_" + day`

Aggregates computed per window:

- `occ_avg`: Average occupancy
- `queue_avg`: Average queue length
- `traffic_avg`: Average traffic level
- `special`: Max special day flag
- `vehicle_avg`: Average vehicle type weight
- `cap`: Max capacity

This aggregation prepares the input for the pricing model.

Step 6 — Dynamic Pricing Model

Model Justification

Inspired by economic demand-based pricing:

$$\text{Demand} = \alpha \cdot \text{capacity/occupancy} + \beta \cdot \text{queue length} - \gamma \cdot \text{traffic level} + \delta \cdot \text{special day} + \epsilon \cdot \text{vehicle weight}$$

Parameters used:

- **ALPHA = 2.0:** Emphasizes occupancy relative to capacity
- **BETA = 0.5:** Moderate impact of queue length
- **GAMMA = 1.0:** Reduces price if traffic is congested
- **DELTA = 1.5:** Increases price on special days
- **EPSILON = 1.0:** Vehicle type impact
- **LAMBDA = 1.0:** Demand sensitivity scaling
- **BASE_PRICE = 10.0:** Reference base price

Normalization:

norm_demand=Demand/5.0

Raw price calculation:

raw_price=BASE_PRICE×(1+LAMBDA×norm_demand)

Bounding:
Prices were capped between **0.5x** and **2.0x** the base price.

Step 7 — Export of Computed Results

Rationale

Pathway does not provide `to_pandas()` or `to_csv()` methods on tables, so we used:

JSONLines sink:

python
CopyEdit

```
pw.io.jsonlines.write(
```

```
    delta_with_price,
    "parking_prices_output",
    overwrite=True
)
```

Post-run merging:

python
CopyEdit

```
import json, glob, pandas as pd

files = glob.glob("parking_prices_output/*.jsonl")
rows = []
for f in files:
    with open(f) as ff:
        for line in ff:
            rows.append(json.loads(line))
df_result = pd.DataFrame(rows)
df_result.to_csv("daily_parking_prices.csv", index=False)
```

Final Output

The final CSV contains:

- `t`: Day end timestamp
- `occ_avg`, `queue_avg`, `traffic_avg`, `special`, `vehicle_avg`, `cap`, `lot`: Aggregated inputs
- `demand`: Computed demand score
- `norm_demand`: Normalized demand
- `raw_price`: Pre-capped price
- `price`: Final bounded price per day per lot

This output can be visualized, analyzed, or further processed.

Conclusion and Justification

This project demonstrates:

- Real-time streaming data processing
- Dynamic pricing model integrating multiple features
- Robust data cleaning
- Clean export for downstream applications

Each step was carefully chosen to ensure:

- Clean numeric inputs
- Clear separation of aggregation and pricing logic
- Compatibility across Pathway versions

Potential Enhancements

- Visualization in Bokeh/Panel dashboards
- Real-time API endpoint for live prices
- Extended demand models (e.g., seasonal adjustments)