DYNAMIC PRICING FOR URBAN PARKING LOTS

CAPSTONE PROJECT OF SUMMER ANALYTICS 2025
HOSTED BY CONSULTING & ANALYTICS CLUB, PATHWAY

Suraj Singh Chahar
IIT Roorkee

01

INTRODUCTION

Background and Motivation:

Urban parking spaces represent a critically limited and highly demanded resource within cities. The traditional approach of maintaining static parking prices throughout the day often leads to significant inefficiencies, resulting in either severe overcrowding or underutilization of available spots. To address these challenges and improve the overall utilization of urban parking infrastructure, the implementation of dynamic pricing strategies is crucial. Such strategies should be responsive to real-time conditions, factoring in fluctuating demand and competitive landscapes. This project aims to simulate such a system, focusing on developing intelligent, data-driven pricing for urban parking lots.

Project Objective:

The primary goal of this project is to build a dynamic pricing model for each of 14 urban parking spaces. This model is designed to update prices realistically and in real-time, taking into account several key factors: historical occupancy patterns , current queue length , nearby traffic congestion levels , special day indicators (e.g., holidays, events) , the type of incoming vehicle , and the prices of competitor parking lots. The pricing system will start from a base price of $10 , and a core requirement is that the price variations are smooth and explainable, avoiding erratic fluctuations. Optionally, the system may also suggest rerouting vehicles to nearby lots if the current lot becomes overburdened.

This project utilizes data collected from 14 urban parking spaces over a period of 73 days. The data was sampled at 18 distinct time points per day, with a 30-minute difference between each sample, spanning from 8:00 AM to 4:30 PM on the same day. Participants are tasked with creating an intelligent, data-driven pricing engine using only numpy and pandas libraries for machine learning model development, with the overall simulation and real-time data processing handled by Pathway.

02

METHODOLOGY AND APPROACH

Methodology Overview

We followed a structured approach combining data engineering, exploratory analysis, and modeling:

Data Collection and Integration

- Collected real-time streaming data from 14 parking lots over 73 days.
- Data sampled at 30-minute intervals, resulting in 18 time points per day.
- Included location, capacity, occupancy, queue length, vehicle type, traffic conditions, and special day flags.

Data Cleaning and Preprocessing

- Checked for missing values and duplicates (none detected).
- Parsed timestamps and merged date/time fields.
- Sorted data by lot ID and timestamp for time-series consistency.

Feature Engineering

- Occupancy Rate (normalized occupancy relative to capacity).
- Vehicle Type Encoding (space-based weights for cars, bikes, trucks, cycles).
- Traffic Level Encoding (urgency-based scaling for low, average, high congestion).
- Time Features: Hour, Minute, DayOfWeek, and Hourly window IDs for real-time streaming.

Exploratory Data Analysis (EDA)

- Visualized and analyzed occupancy and queue distributions.
- Examined relationships with traffic conditions, vehicle types, special days, and time of day.
- Identified clear diurnal peaks and traffic-sensitive demand surges requiring dynamic pricing responses.

Modeling Approach

- Developed multiple pricing strategies of increasing sophistication:
  - Model 1: Linear pricing based on occupancy rate.
  - Model 2: Demand score integrating occupancy, queue length, traffic, special events, and vehicle type.
  - Model 3: Advanced model with inter-lot competition and rerouting simulation using Haversine distances.

Real-Time Simulation

- Implemented pricing models in a streaming framework (Pathway).
- Processed data in tumbling hourly windows to generate time-sensitive prices.
- Simulated user rerouting decisions based on dynamic prices and competitor proximity.

03

DATA DESCRIPTION

The dataset contains detailed records collected from 14 urban parking spaces over a period of 73 days. Each day is sampled at 18 time points, with 30-minute intervals ranging from 8:00 AM to 4:30 PM.
Each record captures the state of a parking lot at a specific time and includes:

1. Location Information:
- Latitude and Longitude of each parking space.
- Enables calculating proximity to competitor lots.
2. Parking Lot Features:
- Capacity: Maximum vehicles that can be parked.
- Occupancy: Current number of parked vehicles.
- Queue Length: Number of vehicles waiting to enter.
3. Vehicle Information:
- Type of incoming vehicle: Car, bike, cycle or truck.
4. Environmental Conditions:
- Nearby traffic congestion level.
- Special day indicator (e.g., holidays, events).

Each time step represents the real-time state of the parking lot. Demand varies throughout the day based on these features, allowing us to model how price should respond dynamically.

04

BASIC DATA EXPLORATION

In this section, we summarize the raw dataset's structure and basic statistics to understand its quality and properties before modeling.

Dataset Info
I inspected the dataset using the df.info() method. It showed:

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18368 entries, 0 to 18367
Data columns (total 12 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   ID                      18368 non-null  int64
 1   SystemCodeNumber        18368 non-null  object
 2   Capacity                18368 non-null  int64
 3   Latitude                18368 non-null  float64
 4   Longitude               18368 non-null  float64
 5   Occupancy               18368 non-null  int64
 6   VehicleType             18368 non-null  object
 7   TrafficConditionNearby  18368 non-null  object
 8   QueueLength             18368 non-null  int64
 9   IsSpecialDay            18368 non-null  int64
 10  LastUpdatedDate         18368 non-null  object
 11  LastUpdatedTime         18368 non-null  object
dtypes: float64(2), int64(5), object(5)
memory usage: 1.7+ MB
```

Descriptive Statistics
I ran df.describe() to get summary statistics for numerical features.

df.describe()

	ID	Capacity	Latitude	Longitude	Occupancy	QueueLength	IsSpecialDay
count	18368.000000	18368.000000	18368.000000	18368.000000	18368.000000	18368.000000	18368.000000
mean	9183.500000	1605.214286	25.706547	90.751170	731.084059	4.587925	0.150915
std	5302.529208	1131.153886	1.582749	3.536636	621.164982	2.580062	0.357975
min	0.000000	387.000000	20.000035	78.000003	2.000000	0.000000	0.000000
25%	4591.750000	577.000000	26.140048	91.727995	322.000000	2.000000	0.000000
50%	9183.500000	1261.000000	26.147482	91.729511	568.000000	4.000000	0.000000
75%	13775.250000	2803.000000	26.147541	91.736172	976.000000	6.000000	0.000000
max	18367.000000	3883.000000	26.150504	91.740994	3499.000000	15.000000	1.000000

Missing Values
I checked for null or missing data using df.isnull().sum() -->>

df.isnull().sum()

	0
ID	0
SystemCodeNumber	0
Capacity	0
Latitude	0
Longitude	0
Occupancy	0
VehicleType	0
TrafficConditionNearby	0
QueueLength	0
IsSpecialDay	0
LastUpdatedDate	0
LastUpdatedTime	0

dtype: int64

Duplicate Records
I ran df.duplicated().sum() to check for duplicate entries.

```
df.duplicated().sum()

np.int64(0)
```

05

FEATURE ENGINEERING AND ENCODING OF CATEGORICAL COLUMNS

Feature Engineering:

I created new features and processed existing ones to improve the predictive power of our pricing model:

- Occupancy Rate

  Calculated as:

  OccupancyRate = Occupancy / Capacity

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

To capture diurnal demand patterns, I derived multiple time-based features from the dataset's timestamps. These features help model parking behavior as it varies by hour of the day, and enable the system to learn and adapt to temporal trends.

Timestamp Combination:

I merged LastUpdatedDate and LastUpdatedTime columns into a single Timestamp:

```
df['Timestamp'] = pd.to_datetime(
    df['LastUpdatedDate'] + ' ' + df['LastUpdatedTime'],
    format='%d-%m-%Y %H:%M:%S'
)
```

- Ensures unified datetime format.
- Essential for time-based analysis, ordering, and streaming

Derived Hour and Minute columns to capture intra-day patterns:

- Hour reflects coarse-grained temporal variation (e.g., morning vs. afternoon).
- Minute adds fine-grained time resolution.

Day of the Week

Created DayOfWeek feature:

- Encodes which day it is (Monday = 0, Sunday = 6).
- Captures weekly cyclic patterns in demand (e.g., weekends vs weekdays).

```
df['Hour'] = df['Timestamp'].dt.hour
df['Minute'] = df['Timestamp'].dt.minute
df['DayOfWeek'] = df['Timestamp'].dt.dayofweek
```

05

FEATURE ENGINEERING AND ENCODING OF CATEGORICAL COLUMNS

Encoding:

- Vehicle Type Encoding

The VehicleType is categorical (car, bike, truck). I converted it into numerical weights:
Car = 1.0
Bike = 0.5
Truck = 1.5
Cycle = 0.1

```
vehicle_weights = {'car': 1.0, 'bike': 0.5, 'truck': 1.5, 'cycle': 0.1}
df['VehicleTypeWeight'] = df['VehicleType'].map(vehicle_weights)
```

- Traffic Encoding

The TrafficConditionsNearby is categorical (car, bike, truck). I converted it into numerical weights:
Low = 0.3
High = 1.0
Avaerage = 0.6

```
traffic_map = {'low': 0.3, 'average': 0.6, 'high': 1}
df['TrafficConditionNearby'] = df['TrafficConditionNearby'].map(traffic_map)
```

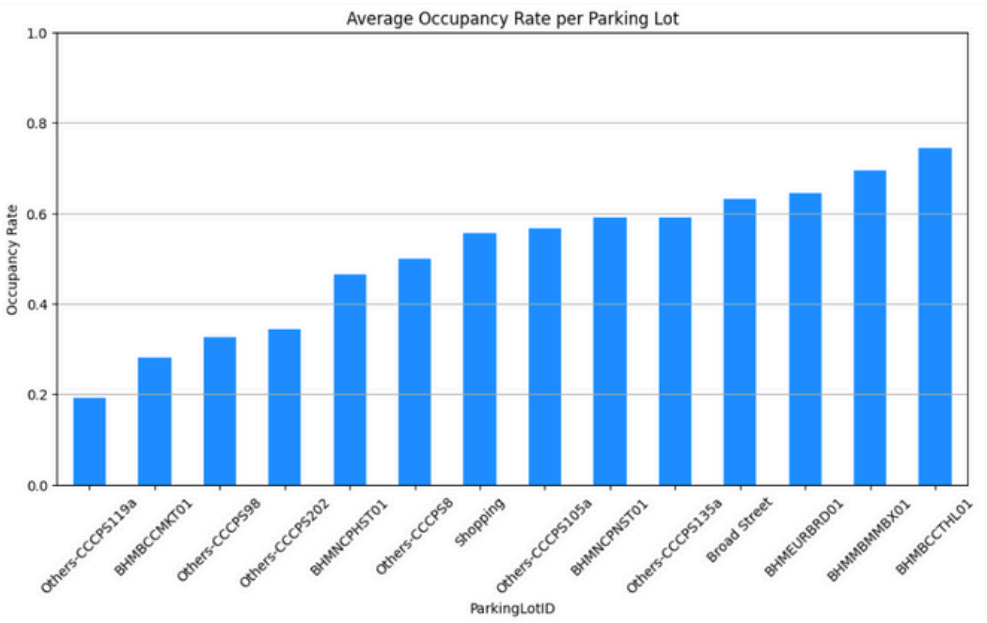
To integrate categorical variables into the pricing model in a meaningful, interpretable way, I assigned numerical weights that reflect their real-world impact on parking demand.
For VehicleType, weights were chosen to represent the relative space each vehicle occupies: Cycle (0.1) for minimal space, Bike (0.5) as smaller than a car, Car (1.0) as the baseline, and Truck (1.5) as occupying significantly more space. This scaling allows the model to price proportionally for larger vehicles that reduce available capacity faster.
For TrafficConditionNearby, weights were designed to reflect increasing demand pressure with congestion: Low (0.3) indicating easier, less urgent parking demand; Average (0.6) suggesting moderate competition for spaces; and High (1.0) capturing the strong likelihood that users will pay more to secure nearby parking when congestion is severe.
These encodings ensure the model can account for these qualitative factors in a smooth, quantitative way that drives sensible pricing behavior.

06

EXPLORATORY DATA ANALYSIS (EDA)

Exploratory Data Analysis (EDA)
This section explores the dataset to uncover patterns, understand variability across parking lots, and identify the key factors influencing parking demand and pricing.
EDA was performed through descriptive statistics, visualizations, and feature relationships.

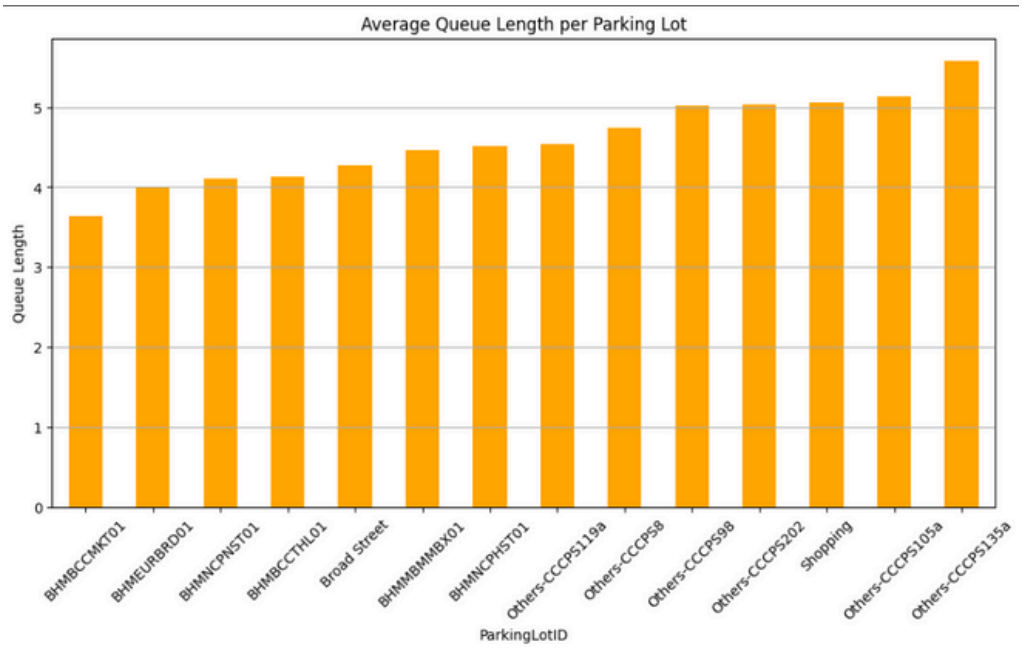
Starting with studying average occupancy rate of different Parking Lots:



- Occupancy rates vary substantially between lots, reflecting differences in location, capacity, and demand.
- Some lots like BHMBCCCTHL01 and BHMMBMBOX01 maintain consistently high occupancy (above 70–80%), suggesting strong demand and potential pricing power.
- Others, such as Others-CCCPS119a and BHMBCCMKT01, show much lower average occupancy (below 30%), indicating underutilization.
- This variation highlights the importance of location-specific dynamic pricing rather than a uniform rate across all lots.
- Understanding these differences is critical for designing demand-responsive pricing strategies that maximize revenue while balancing utilization.

06
EXPLORATORY DATA ANALYSIS (EDA)

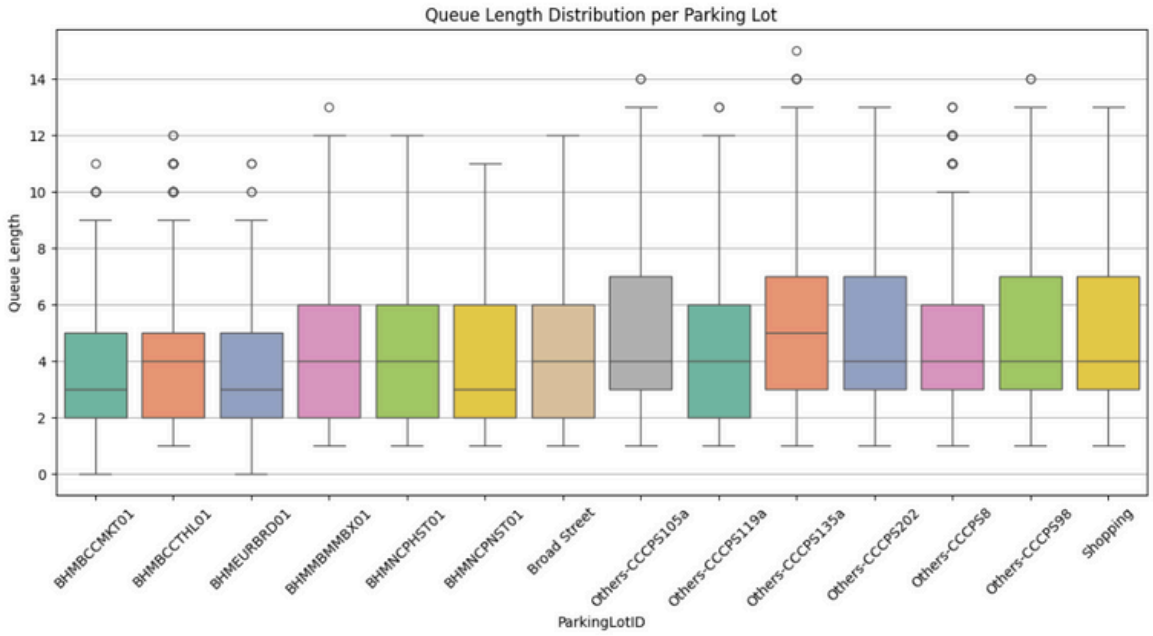
Now we will look at Variation in Queue Length wrt Parking lots :



- The queue lengths vary narrowly but meaningfully across locations.
- Lots like Others-CCCPS153a and Others-CCCPS105a exhibit the longest average queues (above 5.0), indicating persistent congestion and high demand pressure.
- In contrast, BHMBCCMKT01 and BHMEURBRD01 show shorter queues (around 3.6–4.0), suggesting either lower demand or more efficient entry/exit handling.
- High average queue lengths, especially when combined with high occupancy rates, signal the need for priority pricing interventions at specific locations.
- These observations also support the importance of queue-aware pricing logic in the model, helping to balance load and discourage excess waiting.

06
EXPLORATORY DATA ANALYSIS (EDA)

BoxPlot for Queue Length wrt Parking lots :

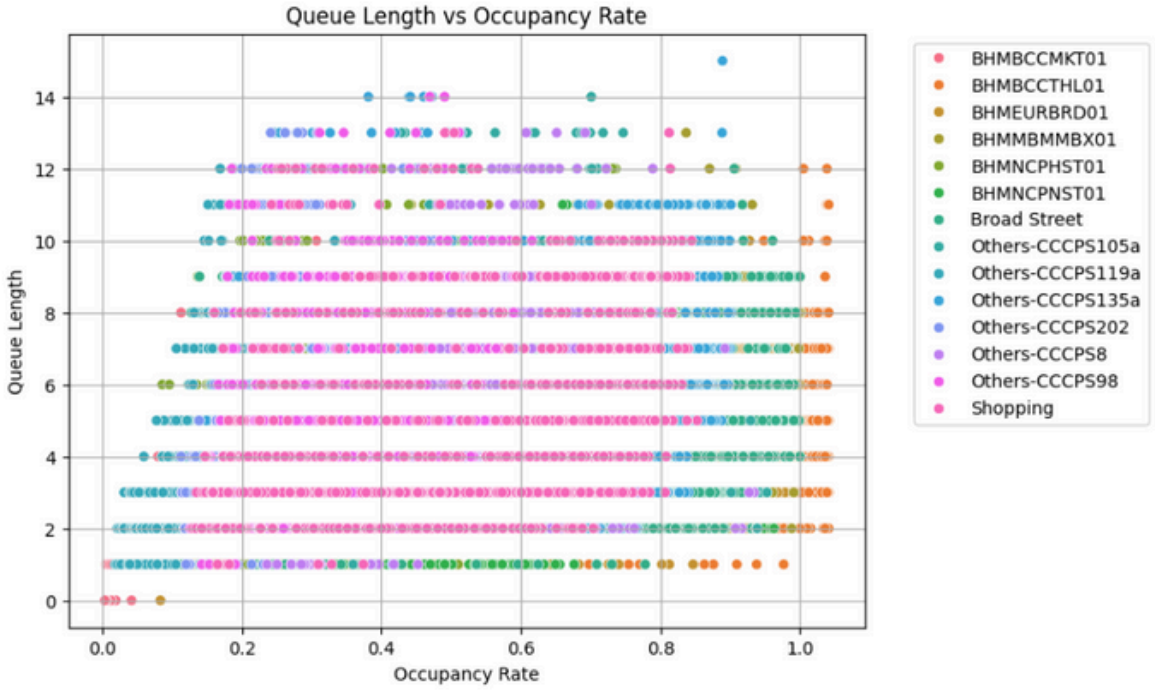


- Median queue lengths are similar across many lots, generally between 3 and 5 vehicles.
- However, variability is substantial, with several lots showing wide interquartile ranges—indicating inconsistent demand.
- Multiple lots exhibit high-end outliers, with queue lengths exceeding 10 or even 14 vehicles. These spikes suggest episodic congestion that could strain capacity.
- Lots such as Others-CCCPS105a and Others-CCCPS153a have both higher medians and more extreme outliers, highlighting them as candidates for dynamic pricing interventions to manage surges.
- The clear presence of outliers across many lots supports the need for queue-aware, demand-responsive pricing to mitigate wait times and distribute demand more evenly.

06

EXPLORATORY DATA ANALYSIS (EDA)

Now Queue Length and OccupancyRate wrt Parking lots :

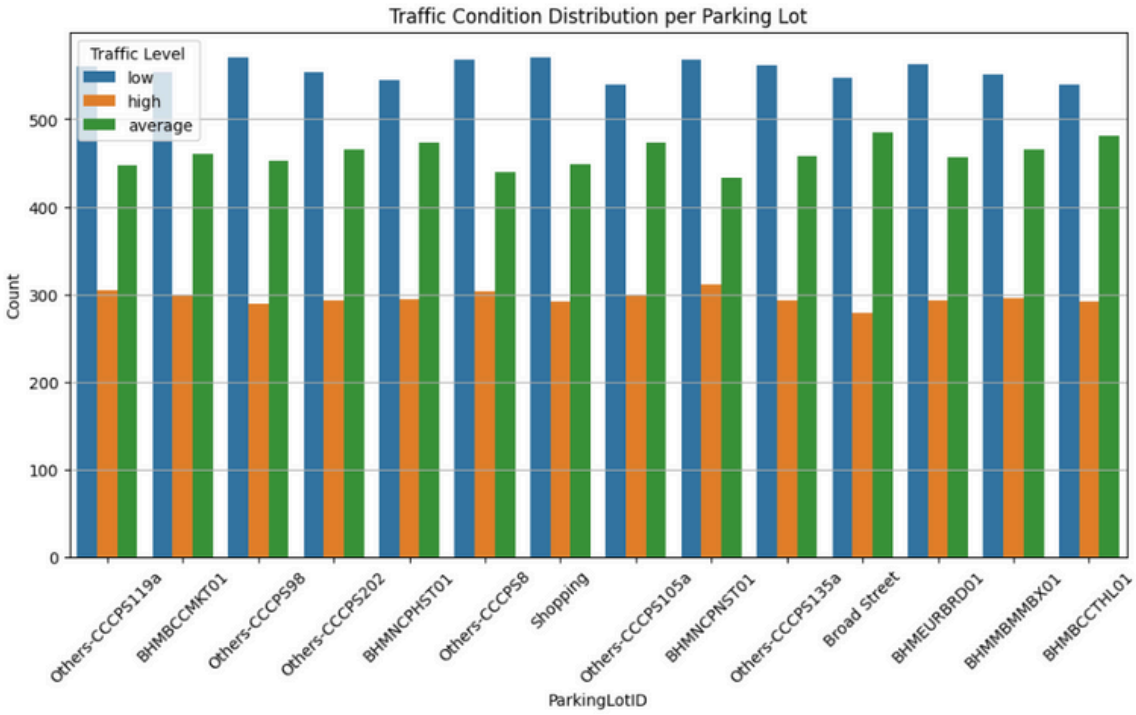


- A general positive trend is visible: higher occupancy rates tend to be associated with longer queues. This aligns with expectations—fuller lots force vehicles to wait.
- However, the relationship is noisy and dispersed, suggesting that occupancy rate alone doesn't fully predict queuing behavior.
- Some lots exhibit high queue lengths even at moderate occupancy rates (e.g., 40–60%), indicating local factors like lot layout, entry delays, or nearby demand spikes.
- This variability suggests that other features, such as traffic conditions, special events, and time of day, also play critical roles in driving queues.
- The pattern validates including queue length as a separate pricing input in the model, rather than assuming it's fully explained by occupancy.

06

EXPLORATORY DATA ANALYSIS (EDA)

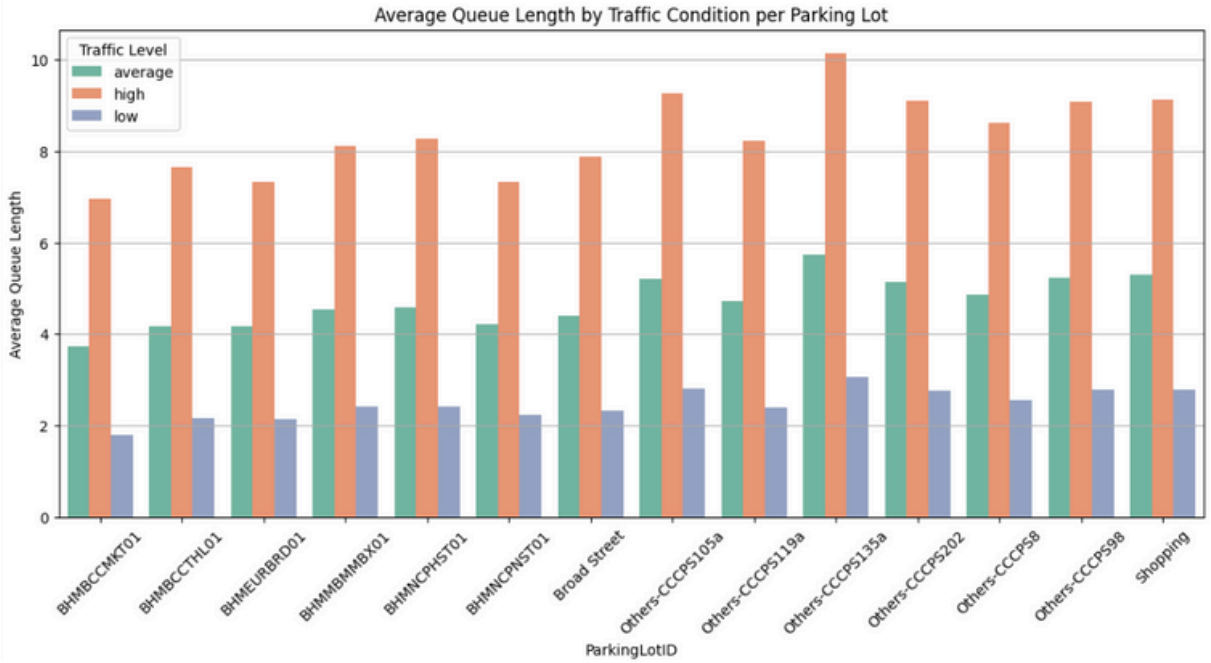
Now we shift our focus towards distribution of Traffic conditions in different parking lots :



- Across all parking lots, low traffic is the most frequent condition, suggesting generally smooth vehicle flow in most areas.
- However, average traffic conditions are also common and quite consistent across lots, implying moderate congestion is the norm in the urban setting.
- High traffic conditions, while less frequent, are present in all locations—most notably in lots such as Others-CCCPS105a, BHMMCPNST01, and Others-CCCPS119a, which show slightly elevated counts.
- This variability confirms that traffic pressure differs across locations and should be factored into pricing models to account for user urgency.
- Lots under frequent high congestion may justify premium pricing due to reduced rerouting flexibility for users.

06
EXPLORATORY DATA ANALYSIS (EDA)

I also investigated the Queue lengths and how they are affected by different traffic conditions in different parking lots :

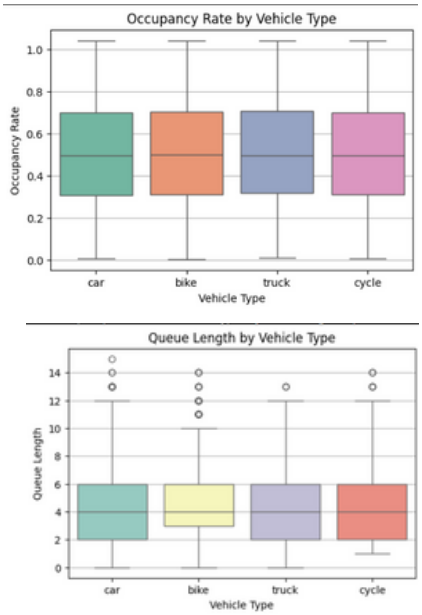


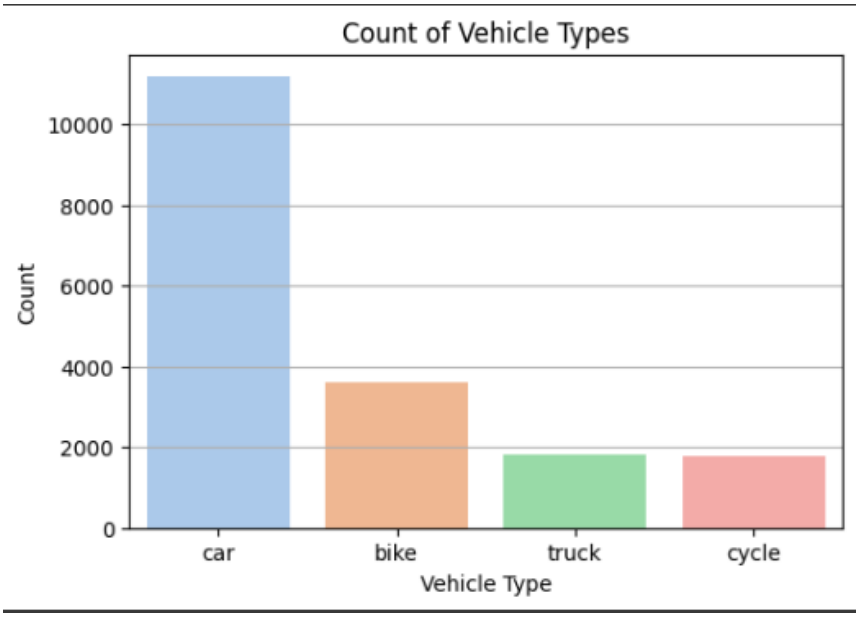
- Across all parking lots, queue lengths increase consistently with traffic level.
- Under low traffic conditions, average queues remain short—generally around 2 to 3 vehicles.
- For average traffic, queues rise to ~4–5 vehicles.
- Under high traffic conditions, queues often exceed 7–10 vehicles, with some lots like Others-CCCPS119a and Others-CCCPS105a peaking over 10.
- This clear monotonic trend confirms traffic congestion is a strong driver of queuing behavior.
- It provides solid evidence for including traffic level as a key input in demand-based dynamic pricing models, enabling price increases during high congestion to manage demand and reduce queuing pressure.

06

EXPLORATORY DATA ANALYSIS (EDA)

Now we study the distribution of Vehicle types and how that affects occupancy rates adn queue lengths



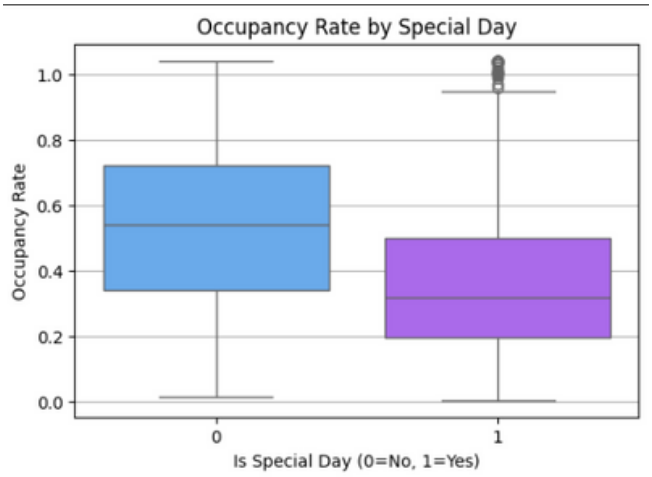


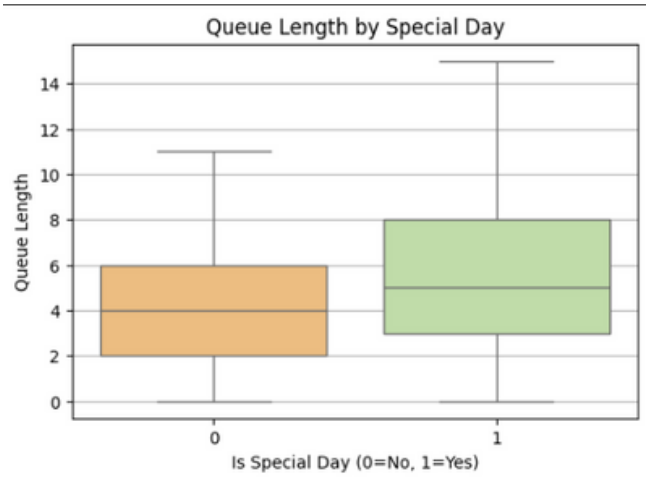
1. Vehicle Type Distribution
   - Cars are the most prevalent vehicle type in the dataset, far outnumbering bikes, trucks, and cycles.
   - Bikes form the next largest category, while trucks and cycles are much less frequent and nearly equal in count.
   - The dominance of cars suggests that any parking or traffic analysis will be heavily influenced by car-related patterns.
2. Occupancy Rate by Vehicle Type
   - The occupancy rate is distributed similarly across all vehicle types.
   - Median occupancy rates for cars, bikes, trucks, and cycles are all near the center of the possible range, indicating moderate lot utilization regardless of vehicle type.
   - The spread and outliers of occupancy rates are comparable across categories, showing no significant difference in how vehicle types use the parking lots.
3. Queue Length by Vehicle Type
   - Queue lengths for cars, bikes, trucks, and cycles all show similar patterns, with medians around 4–5 vehicles.
   - All vehicle types have outliers, occasionally exceeding 10 vehicles in line, pointing to sporadic congestion events.
   - No single vehicle type stands out as being more affected by queuing, indicating uniform congestion experience.

06

EXPLORATORY DATA ANALYSIS (EDA)

Now we study Special Days and their effects :



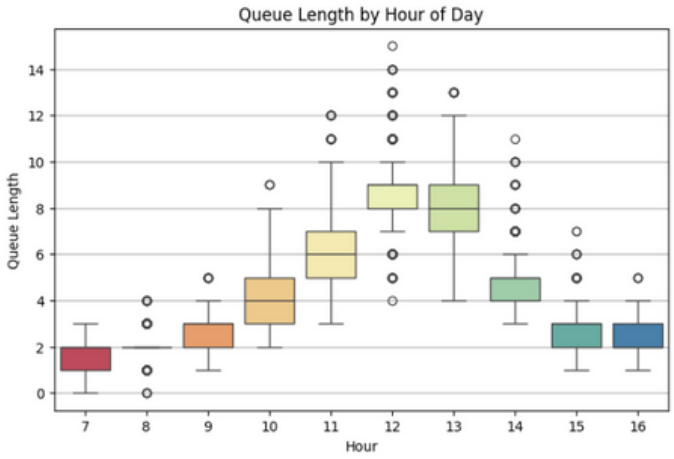


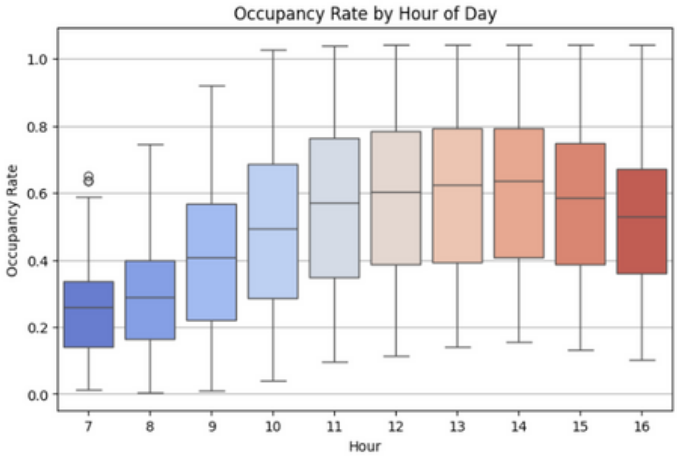
1. Distribution of Special vs Normal Days
   - Normal days (where the special day indicator is 0) are much more frequent in the dataset compared to special days (indicator 1).
   - The count of normal days is significantly higher, suggesting that most operational data reflects routine conditions rather than event-driven or holiday scenarios.
   - This imbalance should be considered when modeling, as special day effects may be less represented but potentially impactful.
2. Queue Length Patterns by Special Day Status
   - Queue lengths are consistently higher on special days compared to normal days.
   - The median queue length, as well as the upper quartile and maximum values, are noticeably greater on special days.
   - On normal days, most queue lengths are moderate, with fewer extreme values.
   - The wider spread and higher median on special days indicate increased demand and congestion during holidays or events.

06

EXPLORATORY DATA ANALYSIS (EDA)

Now we come to the most important part of EDA where I investigated how features like occupany rates and queue lengths vary wrt time





1. Number of Records per Hour
   - Data coverage is consistent throughout the day except for early morning (7 AM) and late afternoon (4 PM), where record counts are noticeably lower.
   - Peak data collection occurs between 8 AM and 3 PM, indicating these hours are the primary focus for analysis and likely represent the busiest periods.
2. Occupancy Rate by Hour
   - Occupancy rates increase steadily from morning to early afternoon, with median values rising from below 0.4 at 7–8 AM to above 0.7 by 1–2 PM.
   - Peak occupancy is sustained between 12 PM and 2 PM, with high variability, suggesting these hours experience the highest demand and potential for full lots.
   - Early and late hours show lower and more variable occupancy, indicating underutilization during these times.
3. Queue Length by Hour
   - Queue lengths rise sharply mid-morning, peaking around noon to 1 PM, with medians reaching 6–8 vehicles and frequent outliers above 10.
   - Early hours (7–8 AM) have minimal queuing, while queue lengths drop again after 2 PM as demand tapers off.
   - Congestion is most severe during midday, highlighting critical periods for dynamic pricing intervention and demand management.

07

MODEL 1 : LINEAR PRICING BASED ON OCCUPANCY

OBJECTIVE :
To create a simple, interpretable baseline pricing strategy that adjusts parking price in direct proportion to occupancy rate.

Model Description

- This model assumes demand pressure is driven solely by how full the lot is.
- As occupancy rises, the price increases linearly, encouraging price-sensitive users to choose lower-demand periods or lots.
- Helps reduce congestion by spreading demand more evenly.

Formula :

$$\text{Price}_{t+1} = \text{Price}_t + \alpha \times \text{OccupancyRate}$$

- BasePrice: Default starting price (e.g., ₽10).
- α \ alpha: Sensitivity coefficient controlling price responsiveness.
- Occupancy / Capacity: Normalized occupancy rate between 0 and 1.

Schema Design (Model 1 – Pathway Stream)
The ParkingSchema defines a minimal, clean schema tailored specifically for Model 1, which relies only on occupancy rate as a pricing signal.

```
df[["ParkingLotID","Timestamp", "Occupancy", "Capacity"]].to_csv("parking_stream.csv", index=False)


class ParkingSchema(pw.Schema):
    Timestamp: str
    ParkingLotID: str
    Occupancy: int
    Capacity: int


data = pw.demo.replay_csv("parking_stream.csv", schema=ParkingSchema, input_rate=100)
```

Rationals:

- Only includes the essential fields needed for computing OccupancyRate.
- Keeps the stream lightweight and efficient, avoiding unnecessary overhead.
- Ideal for a baseline model, where interpretability and speed are prioritized.

07

MODEL 1 : LINEAR PRICING BASED ON OCCUPANCY

Windowing in Pathway

- Data ingested as a real-time stream.
- Tumbling hourly windows were defined using:

```
fmt = "%Y-%m-%d %H:%M:%S"

# Now we Add new columns to the data stream:
data_with_time = data.with_columns(
    t = data.Timestamp.dt.strptime(fmt),
    lot_day_hour = data.ParkingLotID + "_" + data.Timestamp.dt.strptime(fmt).dt.strftime("%Y-%m-%d %H:00:00")
)
```

- Prices computed once per hour per lot, ensuring stable, time-consistent updates.

```
model = (
    data_with_time.windowby(
        pw.this.t,
        instance=pw.this.lot_day_hour,
        window=pw.temporal.tumbling(datetime.timedelta(hours=1)),
        behavior=pw.temporal.exactly_once_behavior()
    ).reduce(
        t=pw.reducers.max(pw.this.t),
        lot_id=pw.reducers.max(pw.this.ParkingLotID),
        occupancy=pw.reducers.avg(pw.this.Occupancy),
        capacity=pw.reducers.max(pw.this.Capacity),
    ).with_columns(
        price= 10 + 5*(pw.this.occupancy/pw.this.capacity)
)
)
```

- Defines non-overlapping 1-hour windows
- Each window aggregates 30-minute sampling intervals into a single hourly price.
- Supports clear, scheduled price updates for users.
- Groups data by a unique identifier combining lot ID, day, and hour.
- Ensures prices are calculated separately for each lot-hour pair.
- Guarantees that each record is processed once, preventing duplication or missed updates in the streaming pipeline.
- t: Latest timestamp in the hour (for accurate reporting).
- lot_id: Confirms grouping consistency.
- occupancy: Averages occupancy across all 30-minute intervals within the hour.
- capacity: Kept constant (maximum in window).
- Dynamically increases price as the occupancy rate rises, encouraging demand balancing

07

MODEL 1 : LINEAR PRICING BASED ON OCCUPANCY

Visualization of Model 1 Pricing Output:
To demonstrate real-time price adjustments, we implemented an interactive visualization using Bokeh and Panel in Python.
Key Design Choices:

- Focused on one parking lot at a time (e.g., BHMBCCMKT01) to maintain clarity and real-time responsiveness.
- Used Pathway's .plot() integration to stream pricing data directly into the visualization.
- Employed Bokeh for rich, interactive plotting with time-aware axes and hover tools.

Implementation Highlights:

- Defined a reusable create_lot_plotter function to generate plots for any selected lot ID.
- Used Pathway's plot() method to bind live data to the visualization, ensuring prices appear in real time as the stream updates.
- Limited to one lot at a time to avoid clutter and to match the real-time streaming behavior, which would not suit static plots of all 14 lots simultaneously.

Key Features of Model 1 :

- Simple Linear Formula: Prices increase proportionally with occupancy rate.
- Real-Time Streaming: Uses Pathway's hourly tumbling windows for smooth, consistent updates.
- Baseline Control Model: Serves as a benchmark for evaluating more advanced pricing strategies.

Limitations of Model 1 :

- Ignores other demand drivers like queue length, traffic congestion, vehicle type, and special days.
- Assumes occupancy alone predicts demand, which may fail during surges or events.
- Uniform sensitivity (same alpha) across all lots may not capture location-specific dynamics.
- No competitive pricing or rerouting logic to account for nearby lot availability and pricing.

08

MODEL 2 : DEMAND BASED PRICING

OBJECTIVE :

To design a more sophisticated, realistic pricing model that captures multiple demand drivers beyond simple occupancy, enabling nuanced, fair, and effective dynamic pricing in real-time.

Design Overview :

Model 2 calculates a composite demand score for each hourly window using weighted contributions from:

- Occupancy Rate: Higher occupancy → higher price.
- Queue Length: Longer queues → higher demand signal.
- Traffic Condition Nearby: Higher congestion → lower demand score (users may avoid the lot).
- Special Day Flag: Adjusts for holidays/events.
- Vehicle Type Weight: Larger vehicles increase demand pressure.

```
df = df.sort_values('Timestamp').reset_index(drop=True)


required_columns = ["Timestamp", "ParkingLotID", "Occupancy", "Capacity",
                    "QueueLength", "TrafficConditionNearby", "IsSpecialDay", "VehicleType","VehicleTypeWeight"]



df[required_columns].to_csv("parking_stream_model2.csv", index=False)


class ParkingSchemaModel2(pw.Schema):
    Timestamp: str
    ParkingLotID: str
    Occupancy: int
    Capacity: int
    QueueLength: int
    TrafficConditionNearby: float
    IsSpecialDay: int
    VehicleType: str
    VehicleTypeWeight: float


data = pw.demo.replay_csv("parking_stream_model2.csv", schema=ParkingSchemaModel2, input_rate=50)
```

Demand Score Formula :

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

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

Explanation of Weights:

- α\alphaα (0.6): Emphasizes occupancy as the primary driver.
- β\betaβ (0.3): Captures queue pressure.
- γ\gammaγ (0.1): Penalizes demand for high traffic (users may avoid congested routes).
- δ\deltaδ (0.2): Boosts demand on special days.
- ϵ\epsilonϵ (0.15): Adjusts for larger vehicles occupying more space.

08

MODEL 2 : DEMAND BASED PRICING

A richer schema than Model 1 was defined to support the multi-factor model.

Windowing Strategy in Pathway

- Data is streamed using tumbling hourly windows.
- Grouped by lot_day_hour (unique per lot and hour).
- Guarantees exactly-once processing for reliable, repeatable pricing updates.

```
import datetime

model_2 = (
    data_with_time.windowby(
        pw.this.t,
        instance=pw.this.lot_day_hour,
        window=pw.temporal.tumbling(datetime.timedelta(hours=1)),
        behavior=pw.temporal.exactly_once_behavior()
    )
    .reduce(
        t=pw.reducers.max(pw.this.t),
        lot_id=pw.reducers.max(pw.this.ParkingLotID),
        current_occupancy=pw.reducers.avg(pw.this.Occupancy),
        current_capacity=pw.reducers.max(pw.this.Capacity),
        avg_queue_length=pw.reducers.avg(pw.this.QueueLength),
        max_queue_length=pw.reducers.max(pw.this.QueueLength),
        avg_traffic_level=pw.reducers.avg(pw.this.TrafficConditionNearby),
        special_day=pw.reducers.any(pw.this.IsSpecialDay),
        avg_vehicle_weight=pw.reducers.avg(pw.this.VehicleTypeWeight),
        data_points=pw.reducers.count()
    )
)
```

```
demand_pricing_model = (
    model_2
    .with_columns(
        occupancy_rate=pw.this.current_occupancy / pw.this.current_capacity,
        base_price=10.0
    ).with_columns(
        raw_demand=pw.apply(
            DemandBasedPricingLogic.calculate_demand_score,
            pw.this.current_occupancy,
            pw.this.current_capacity,
            pw.this.avg_queue_length,
            pw.this.max_queue_length,
            pw.this.avg_traffic_level,
            pw.this.special_day,
            pw.this.avg_vehicle_weight,
        )
    ).with_columns(
        normalized_demand=pw.apply(
            DemandBasedPricingLogic.normalize_demand,
            pw.this.raw_demand,
            -0.5,
            1.5
        )
    ).with_columns(
        raw_price=pw.apply(
            DemandBasedPricingLogic.calculate_price_from_demand,
            pw.this.normalized_demand,
            pw.this.base_price,
            1.0
        )
    ).with_columns(
        price=pw.apply(
            DemandBasedPricingLogic.apply_price_bounds,
            pw.this.raw_price,
            10.0,
            2.0,
            0.5
        )
    )
)
```

08

MODEL 2 : DEMAND BASED PRICING

Aggregation in Streaming :

Within each hourly window, the following were computed:

- Average occupancy and queue length.
- Maximum capacity and queue length for normalization.
- Average traffic level and vehicle type weight.
- Boolean flag for any special day.
- Count of data points for diagnostics.

```
demand_pricing_model = (
    model_2
    .with_columns(
        occupancy_rate=pw.this.current_occupancy / pw.this.current_capacity,
        base_price=10.0
    ).with_columns(
        raw_demand=pw.apply(
            DemandBasedPricingLogic.calculate_demand_score,
            pw.this.current_occupancy,
            pw.this.current_capacity,
            pw.this.avg_queue_length,
            pw.this.max_queue_length,
            pw.this.avg_traffic_level,
            pw.this.special_day,
            pw.this.avg_vehicle_weight,
        )
    ).with_columns(
        normalized_demand=pw.apply(
            DemandBasedPricingLogic.normalize_demand,
            pw.this.raw_demand,
            -0.5,
            1.5
        )
    ).with_columns(
        raw_price=pw.apply(
            DemandBasedPricingLogic.calculate_price_from_demand,
            pw.this.normalized_demand,
            pw.this.base_price,
            1.0
        )
    ).with_columns(
        price=pw.apply(
            DemandBasedPricingLogic.apply_price_bounds,
            pw.this.raw_price,
            10.0,
            2.0,
            0.5
        )
    )
)
```

08
MODEL 2 : DEMAND BASED PRICING

Visualization of Model 2 Pricing Output :
Key Design Features:

- Built using Bokeh and Panel for interactive, notebook-friendly plotting.
- Plots prices over time on a datetime x-axis, matching the hourly windowing strategy used in streaming.
- Shows real-time evolution of pricing for any selected parking lot.

Implementation Highlights:

- The create_lot_plotter() function is reusable: any parking lot can be visualized by changing lot_id.
- Default example used 'BHMBCCMKT01', but operators can quickly inspect any lot individually.
- Data is filtered in the stream for the chosen lot using Pathway's .filter(), ensuring that the plot only shows relevant pricing history.
- Used Pathway's .plot() method to bind the live data stream to the visualization in real time.
- Wrapped with pn.Column(viz).servable() for smooth integration in notebook dashboards.

Key Features :

- Multi-factor demand model incorporating occupancy, queues, traffic, events, vehicle types.
- Real-time hourly pricing windows tailored per lot.
- Modular, interpretable design with tunable weights.
- Price bounds to ensure user fairness.
- Deployable in a live streaming environment via Pathway.

Limitations :

- Weight parameters are heuristically set, not automatically optimized.
- Assumes linear additive relationship among demand drivers.
- Traffic impact modeled as simple penalty; doesn't capture rerouting effects.
- No spatial competition or user rerouting logic yet—handled in Model 3.
- Requires more input data than Model 1, increasing system complexity.

09
CONCLUSION AND BUSINESS IMPLICATIONS

Business Implications :

Implementing a demand-based dynamic pricing system for urban parking offers clear, actionable benefits for operators and city planners:

Revenue Optimization

- *Prices adjust in real time based on true demand pressure.*
- *Higher rates during peak periods capitalize on willingness to pay.*
- *Lower rates during off-peak hours attract more users, reducing idle capacity.*

Demand Management

- *Dynamic pricing smooths demand curves, reducing congestion in overused lots.*
- *Encourages users to consider lower-demand lots or off-peak times.*
- *Reduces queues, wait times, and user frustration.*

Better User Experience

- *Transparent, real-time pricing signals help users plan.*
- *Avoids static, unfair "one-size-fits-all" pricing that doesn't match demand.*
- *Supports more efficient, equitable use of city infrastructure.*

Operational Insights

- *Streaming analytics provide operators with real-time visibility into usage patterns.*
- *Helps identify consistently over- or underutilized lots.*
- *Informs infrastructure planning, staffing, and marketing decisions.*

Strategic Value:

By adopting demand-based dynamic pricing, parking operators and cities can make smarter use of scarce urban space, reduce congestion, improve revenue, and deliver better service to drivers.

This data-driven approach represents an important step toward more sustainable, efficient, and user-friendly urban mobility systems.

10 REFERNCES :

- *Pathway Official Documentation*
- *Pathway Docs*
  - *Comprehensive guide to Pathway's streaming data framework, including schema design, windowing, and deployment.*
- *Pathway GitHub Repository*
- *Pathway on GitHub*
  - *Open-source codebase with examples and integrations.*
- *Pathway Examples & Tutorials*
- *Pathway Examples Hub*
  - *Practical tutorials on real-time data pipelines, streaming joins, and windowing strategies.*
- *Bokeh Visualization Library*
- *Bokeh Docs*
  - *Interactive visualization library used for real-time plotting of prices.*
- *Panel for Python Dashboards*
- *Panel Docs*
  - *High-level dashboarding framework for deploying interactive Bokeh visualizations in notebooks and apps.*