DYNAMIC PRICING FOR URBAN PARKING LOTS

CAPSTONE PROJECT OF SUMMER ANALYTICS 2025

HOSTED BY CONSULTING & ANALYTICS CLUB × PATHWAY

NIDHI CHAHAR
IIT KGP

INTRODUCTION

Urban parking spaces are a scarce and highly sought-after resource in modern cities. With increasing vehicle density and limited parking infrastructure, static parking prices—set uniformly throughout the day—often lead to inefficiencies. At times, parking lots become overcrowded during peak hours while remaining underutilized during off-peak periods. This mismatch between demand and supply not only frustrates drivers but also reduces the revenue potential of parking operators.

To address these challenges, dynamic pricing has emerged as a viable solution. By adjusting parking rates in real time based on factors like current occupancy, traffic conditions, queue lengths, and special events, operators can better manage demand, improve space utilization, and ensure a more equitable distribution of vehicles across available lots.

This project aims to build a data-driven, intelligent dynamic pricing engine for 14 urban parking lots using real-time data streams. The system is designed to calculate and update parking prices continuously based on both historical patterns and real-time conditions, while keeping the price evolution smooth and interpretable. The models are implemented using Python, Pandas, Numpy, and Pathway, and results are visualized using real-time Bokeh plots.

Through this work, we explore how economic principles, demand modeling, and real-time data processing can come together to make urban parking more efficient, fair, and responsive to changing conditions.

PROBLEM STATEMENT

Urban parking lots experience significant fluctuations in demand throughout the day due to factors such as peak office hours, special events, traffic congestion, and vehicle types. Fixed, static parking prices fail to account for these variations, often resulting in overcrowding during peak periods and underutilization during off-peak times. Such inefficiencies lead to frustrated drivers, increased traffic congestion, and lost revenue opportunities for parking operators.

To tackle this challenge, a dynamic, real-time pricing strategy is required — one that adjusts prices intelligently based on current and historical demand indicators, while ensuring that the price changes remain smooth and understandable to users.

This project simulates such a system by using real-world-inspired data streams from 14 parking lots over 73 days, and designing pricing models that respond to occupancy patterns, queue lengths, traffic conditions, special events, and vehicle types.

OBJECTIVES

The primary objectives of this project are:

- To design and implement dynamic pricing models for 14 urban parking lots, updated in real-time based on demand indicators.
- To develop three progressively sophisticated pricing models:
  - Model 1: A baseline linear model that increases prices proportionally to occupancy.
  - Model 2: A demand-based model that incorporates multiple factors such as queue length, traffic conditions, special days, vehicle type, and temporal trends.
  - Model 3 (Optional): A competitive pricing model that considers the prices and proximity of nearby parking lots to adjust its own pricing and suggest rerouting when needed.
- To ensure the price variation is smooth, interpretable, and bounded (e.g., not more than twice or less than half of the base price).
- To process and visualize real-time pricing updates using Pathway for data streaming and Bokeh for interactive visualizations.
- To demonstrate how real-time data and economic principles can improve the efficiency and utilization of scarce urban parking resources.

DATA DESCRIPTION

The dataset consists of records collected from 14 urban parking lots over a period of 73 days, with observations recorded every 30 minutes during operational hours (8:00 AM to 4:30 PM), resulting in 18 time points per day.
Each record represents the state of a parking lot at a given timestamp, with the following attributes:

1. Location Features:
- SystemCodeNumber — unique identifier for each parking lot.
- Latitude & Longitude — geographic coordinates (not explicitly used in current models but relevant for competitive pricing).

2. Parking Lot Features:
- Capacity — maximum number of vehicles that can park in the lot.
- Occupancy — number of vehicles parked at that time.
- QueueLength — number of vehicles waiting to enter the lot.

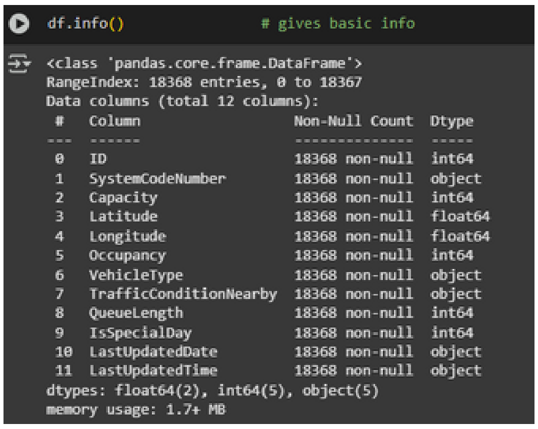
3. Vehicle Information:
- VehicleType — type of incoming vehicle (car, bike, truck, cycle).

4. Environmental & Temporal Factors:
- TrafficConditionNearby — traffic level near the lot (low, medium, high).
- IsSpecialDay — binary indicator for holidays or special events.
- LastUpdatedDate & LastUpdatedTime — combined to form a full Timestamp.

BASIC DATA EXPLORATION

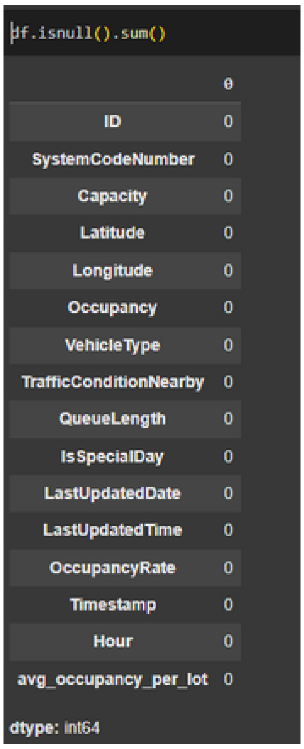
In this section, I explored the dataset to understand its structure, quality, and contents. My goal was to check for missing values, duplicates, and basic distributions before proceeding to deeper analysis.

1. Checking Dataset Information
I loaded the dataset into a pandas DataFrame and used the .info() method to inspect the column names, data types, and the number of non-null entries.



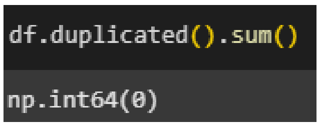
```
df.info()                # gives basic info

<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
```

2.Checking for Missing Values
I applied .isnull().sum() on the dataset and found that there were no missing values in any of the columns.



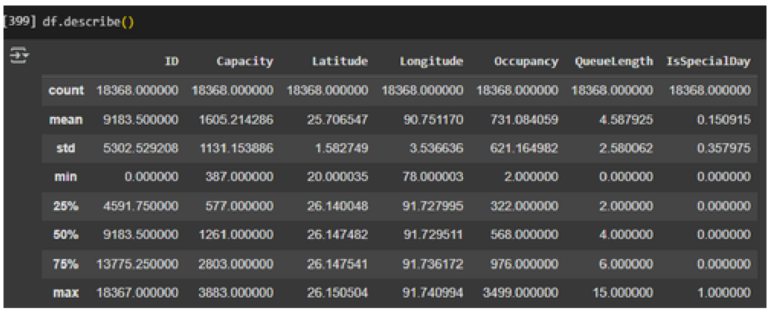
```
df.isnull().sum()

                        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
OccupancyRate           0
Timestamp               0
Hour                    0
avg_occupancy_per_lot   0

dtype: int64
```

3.Checking for Duplicates
I also checked for duplicate rows using .duplicated().sum() and verified that no duplicate records existed in the dataset.



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

np.int64(0)
```

4.Descriptive Statistics
Finally, I used .describe() to compute summary statistics for the numerical columns. This gave me insights into the ranges and distributions of features like Capacity, Occupancy, and QueueLength.



```
[399] 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

FEATURE ENGINEERING

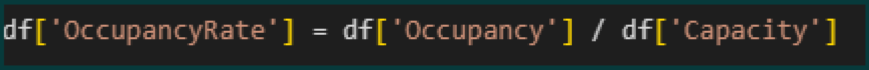
To make the dataset more suitable for modeling and to create features that better capture the dynamics of parking demand, I engineered several new features based on the raw data.

1. Occupancy Rate

Since lots have different capacities, comparing Occupancy alone was not meaningful.

Therefore, I created a new feature called OccupancyRate, calculated as:

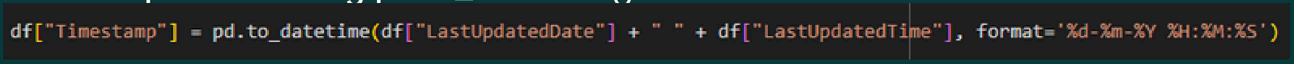
$$\text{OccupancyRate} = \text{Occupancy} / \text{Capacity}$$



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

2. Timestamp

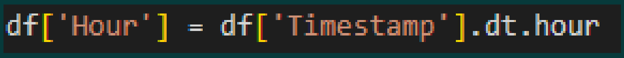
The dataset included separate columns for date (LastUpdatedDate) and time (LastUpdatedTime). To enable time-based analysis, I combined them into a single Timestamp column using pd.to_datetime().



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

3. Hour of the Day

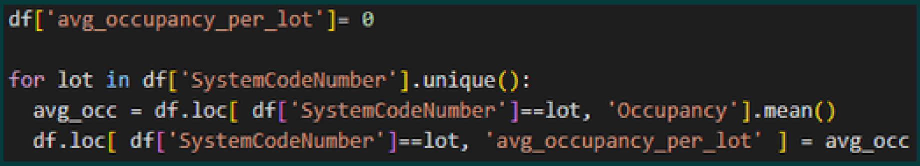
To study hourly trends in occupancy, I extracted the hour component from the Timestamp and stored it in a new column called Hour.



```
df['Hour'] = df['Timestamp'].dt.hour
```

4. Average Occupancy per Lot

For each parking lot, I calculated its overall average occupancy across all records and stored this in a new column avg_occupancy_per_lot. This feature was useful for computing the Temporal Coefficient in pricing models.



```
df['avg_occupancy_per_lot']= 0

for lot in df['SystemCodeNumber'].unique():
  avg_occ = df.loc[ df['SystemCodeNumber']==lot, 'Occupancy'].mean()
  df.loc[ df['SystemCodeNumber']==lot, 'avg_occupancy_per_lot' ] = avg_occ
```

5. Temporal Coefficient

During modeling, I computed the Temporal Coefficient for each observation, defined as:

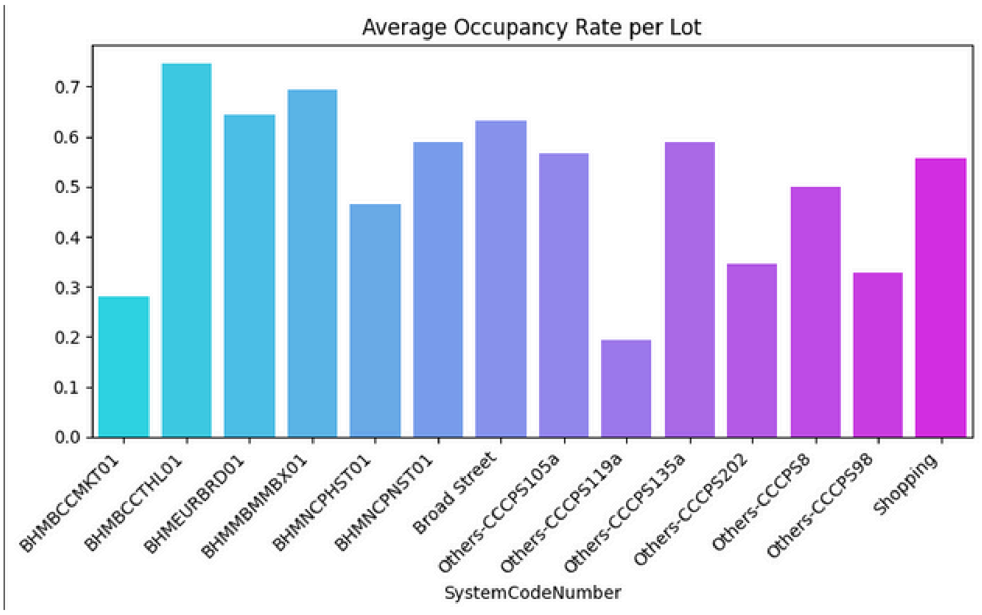
Temporal Coefficient= Occupancy at current hour/ Average Occupancy of the lot

This feature helped capture how busy the lot was at a given time compared to its norm.

EXPLORATORY DATA ANALYSIS (EDA)

After cleaning and engineering the dataset, I performed EDA to uncover patterns and relationships among the variables.

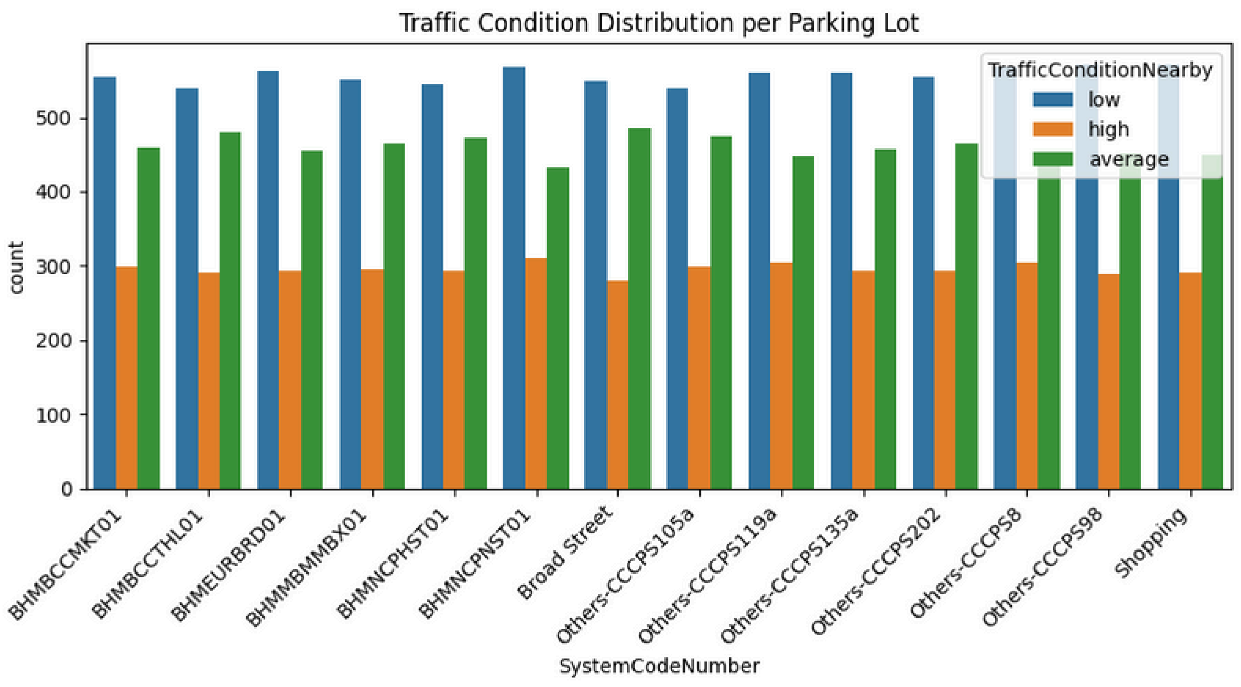
Starting with studying average occupancy rate of different Parking Lots:



- There is significant variation in the average occupancy rates across different parking lots, with some lots exceeding 70% occupancy while others are below 30%.
- The lot labeled "BHMEURBRD01" has the highest average occupancy rate, indicating it may experience high demand or limited capacity.
- Several lots, such as "Others-CCCP5119a" and "Others-CCCP598," show notably low occupancy rates, suggesting underutilization or potential for repurposing.
- The "Shopping" lot has a moderate occupancy rate, which could reflect fluctuating demand based on shopping hours or events.
- The distribution of occupancy rates suggests opportunities for demand balancing, such as redirecting vehicles from highly occupied lots to those with more available space.

EXPLORATORY DATA ANALYSIS (EDA)

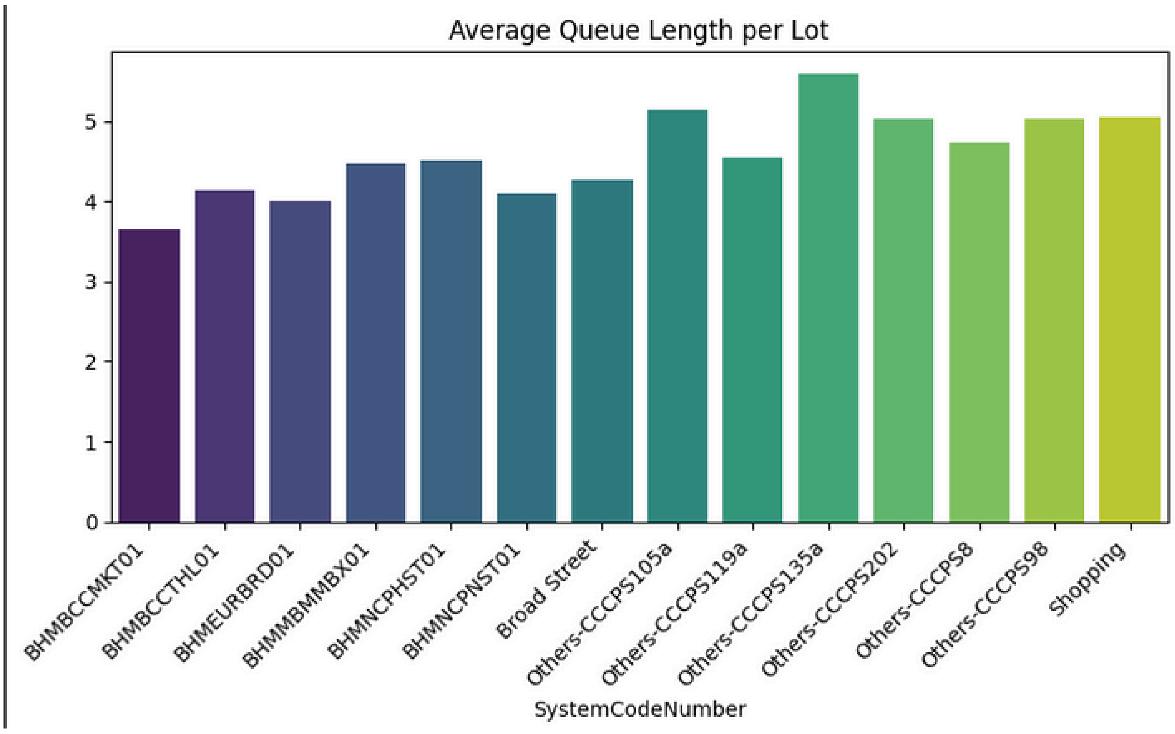
Now distribution of Traffic conditions in different parking lots :



- Across all parking lots, the 'low' traffic condition is the most frequently observed, consistently showing the highest count compared to 'average' and 'high' traffic conditions.
- The 'high' traffic condition category has the lowest count in every parking lot, indicating that high congestion is relatively rare across the system.
- There is a remarkable consistency in the distribution pattern for each lot: 'low' is always the highest, followed by 'average', then 'high', suggesting a stable traffic environment.
- Some lots, such as "BHMBCCMKTO1" and "Broad Street," have slightly higher counts in the 'average' category compared to others, which may warrant further investigation into their traffic flow patterns.
- The similar distribution across diverse parking lots, including "Shopping" and "Others-CCCP" series, implies that external factors (e.g., time of day or special events) may have a limited impact on overall traffic condition variability.

EXPLORATORY DATA ANALYSIS (EDA)

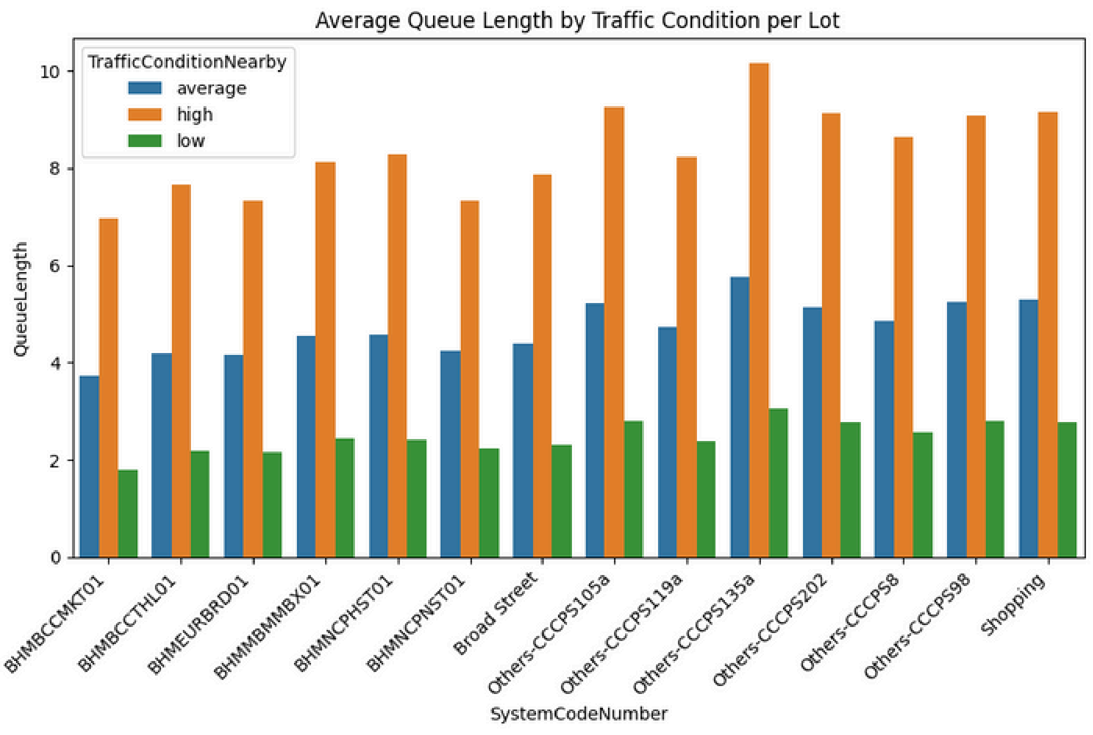
Now we will look at Variation in Queue Length wrt Parking lots:



- Queue lengths vary notably across parking lots, with the lowest average around 3.7 and the highest exceeding 5.5, indicating differing levels of congestion and wait times.
- The lot labeled "Others-CCCP5135a" exhibits the highest average queue length, suggesting it may be a bottleneck or experience peak demand more frequently than other lots.
- Several lots, such as "BHMBCCMKTO1" and "BHMEURBRD01," have consistently lower average queue lengths, which may reflect more efficient traffic flow or lower usage.
- The "Shopping" lot maintains a relatively high average queue length, comparable to the busiest lots, possibly due to fluctuating visitor volumes or event-driven surges.
- The cluster of "Others-CCCP" lots generally shows higher queue lengths, indicating that these locations may require targeted interventions, such as improved signage or dynamic routing, to alleviate congestion.

EXPLORATORY DATA ANALYSIS (EDA)

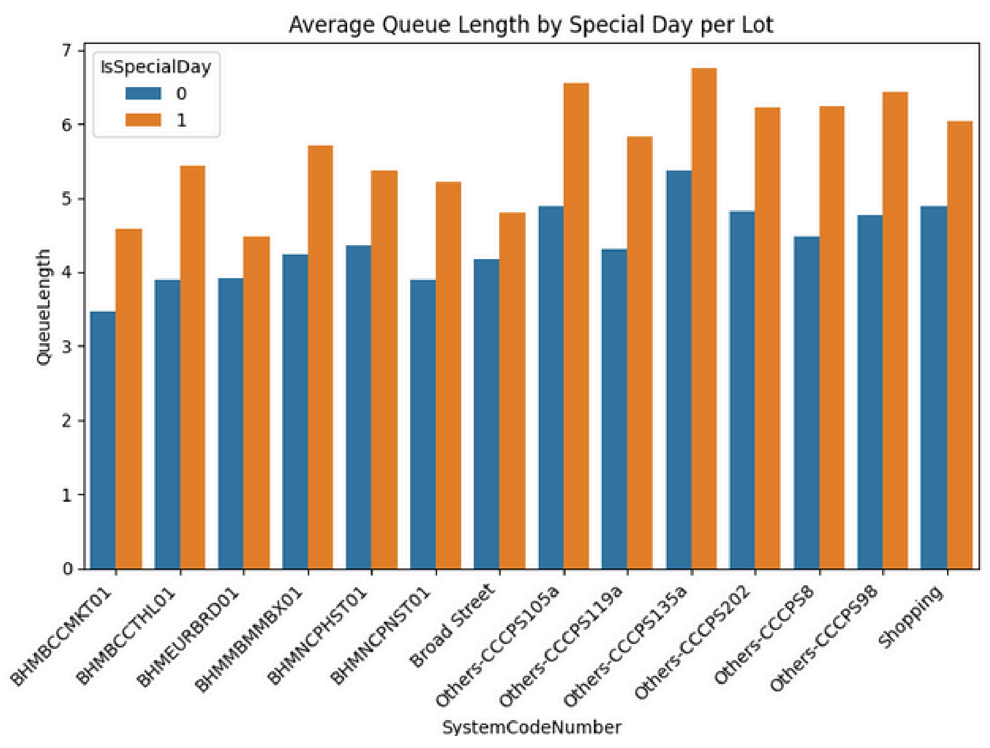
To understand how traffic levels interact with queue length:



- Queue lengths rise with traffic: Every lot shows longer queues as traffic conditions worsen, with 'high' traffic producing the longest queues.
- Consistent effect: The impact of traffic is similar across all lots—queue lengths under 'high' traffic are much greater than under 'low' traffic.
- Some lots are more affected: Lots like "Others-CCCP5135a" and "Others-CCCP5202" see especially high queues during 'high' traffic.
- Low traffic keeps queues short: Under 'low' traffic, queue lengths remain minimal, rarely exceeding 3 vehicles.
- Real-time monitoring is key: Since queue lengths respond sharply to traffic changes, dynamic management could help reduce delays.

EXPLORATORY DATA ANALYSIS (EDA)

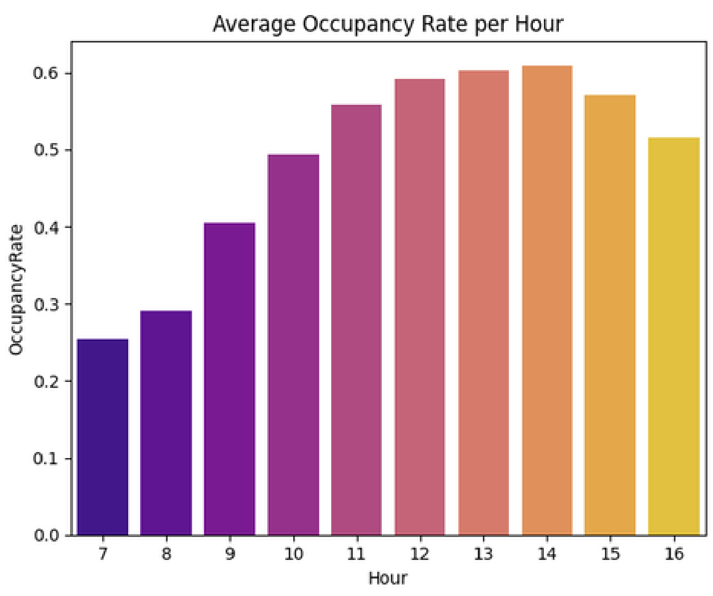
I investigated whether special days (e.g., holidays or events) affected queues:

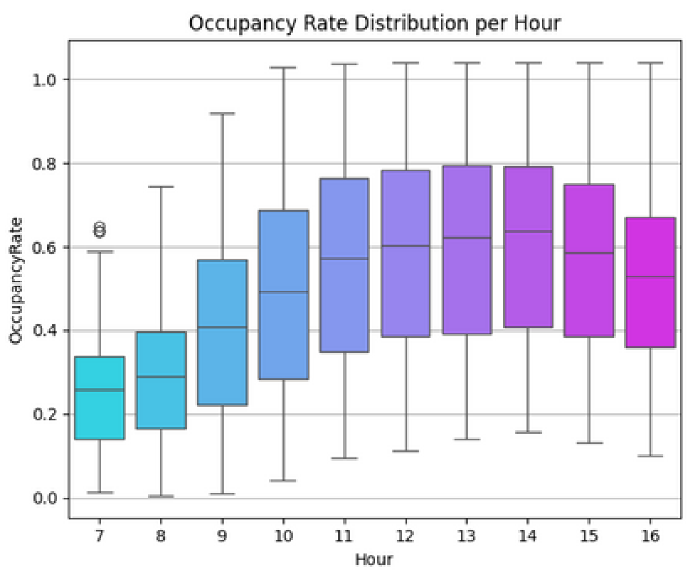


- Queue lengths are consistently higher on special days: Every parking lot shows a noticeable increase in average queue length when it is a special day compared to regular days.
- The difference between regular and special days is substantial: In some lots, such as "Others-CCCP5105a" and "Others-CCCP5135a," the increase in queue length on special days exceeds 1.5 vehicles, highlighting the impact of special events or holidays.
- All lots are affected, but the magnitude varies: While every lot experiences longer queues on special days, lots like "BHMBCCMKTO1" and "BHMEURBRD01" see a smaller increase, suggesting differences in event-driven demand or capacity.
- Shopping and 'Others-CCCP' lots are especially impacted: These lots exhibit some of the highest queue lengths on special days, indicating they may be popular destinations during events or holidays.
- Operational planning should consider special days: The clear spike in queue lengths on special days emphasizes the need for dynamic resource allocation or traffic management to handle increased demand efficiently.

EXPLORATORY DATA ANALYSIS (EDA)

To capture the daily temporal pattern:

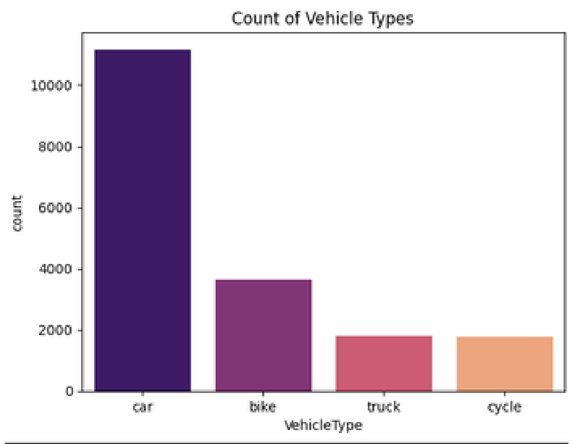




- Daily occupancy follows a clear pattern: Rates start low in the early morning, rise through midday (peaking 12–2 PM), then decline in the afternoon.
- Peak and off-peak hours are distinct: Highest demand occurs midday, while early morning and late afternoon see lower occupancy.
- Hour-to-hour variability: Even during peak times, occupancy rates vary widely, with some hours showing much higher or lower values than the average.
- Outliers are present: Occasional surges or drops in occupancy, especially in the early hours, highlight variability not captured by averages.
- Operational takeaway: The combination of rising averages and broad distributions suggests dynamic management (like flexible pricing) could improve utilization during peak and variable periods.

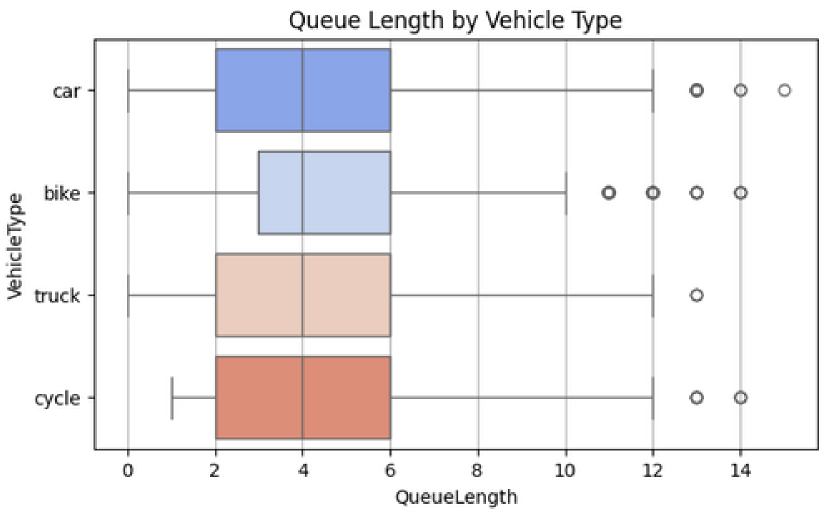
EXPLORATORY DATA ANALYSIS (EDA)

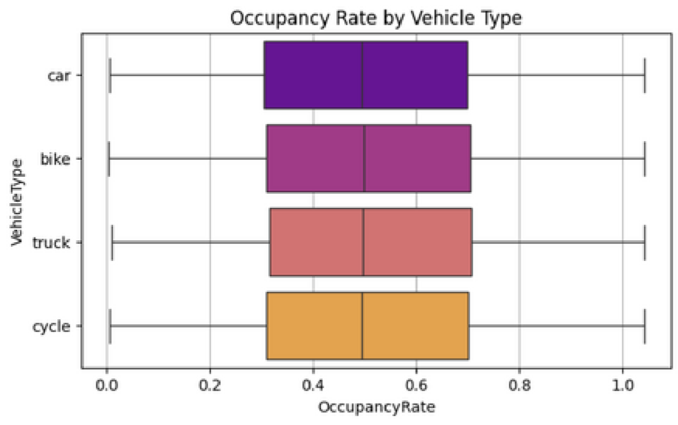
To understand the composition of incoming vehicles and its relation with other columns:



- Cars are the most common vehicle type by a large margin, with counts far exceeding those of bikes, trucks, and cycles, indicating that cars dominate the vehicle population in the dataset.

- Queue lengths vary by vehicle type: Cars and trucks tend to have longer queue lengths and more outliers, while bikes and cycles generally experience shorter, more consistent queues. This suggests that larger vehicles may contribute more to congestion and queue variability at parking or checkpoint locations.

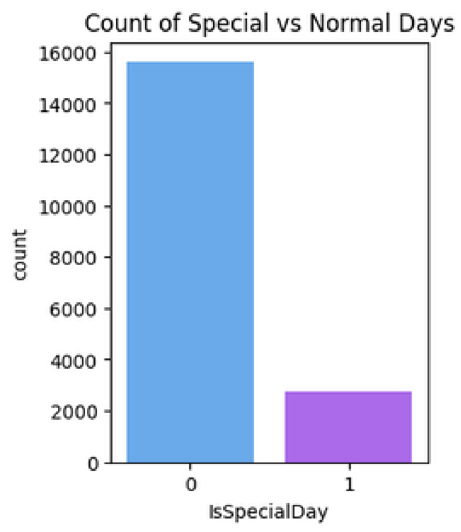


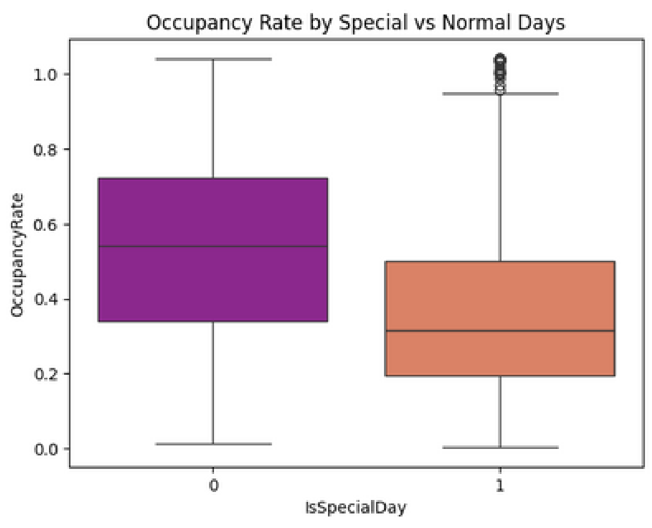


- Occupancy rates are similar across all vehicle types, with cars, bikes, trucks, and cycles each showing a wide but comparable distribution, indicating no single vehicle type consistently dominates or lags in occupancy.

EXPLORATORY DATA ANALYSIS (EDA)

To understand the composition of IsSpecialDay and its impact on occupancy rate:





- The dataset has many more records for normal days than special days, indicating class imbalance.
- Median occupancy is higher on normal days, while special days show more variability and occasional spikes.

MODEL 1

OBJECTIVE

The goal of this model was to calculate a simple, interpretable dynamic price that increases linearly with occupancy and accounts for temporal demand patterns. This model ensures that prices rise during busy periods and remain low during off-peak times, while being easy to explain and implement.

Pricing Function

I used the following linear pricing equation:

$$\text{Price}=10+\alpha\cdot\text{CapacityOccupancy}+\beta\cdot\text{Temporal Coefficient}$$

Where:

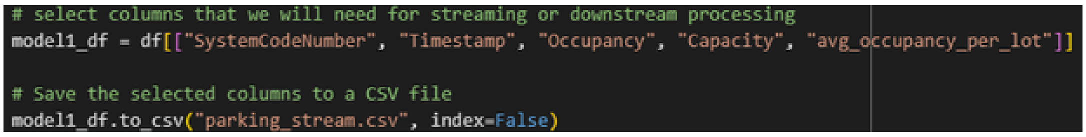
- Base price = 10 (as given in the problem statement)
- $\alpha=5$: weight given to the occupancy rate
- $\beta=2$: weight given to the temporal coefficient
- Occupancy rate = Occupancy/Capacity
- Temporal coefficient = Occupancy at current hour/Average occupancy of the lot

This formula ensures that the price increases both as the lot fills up and during peak hours when the occupancy exceeds the lot's average.

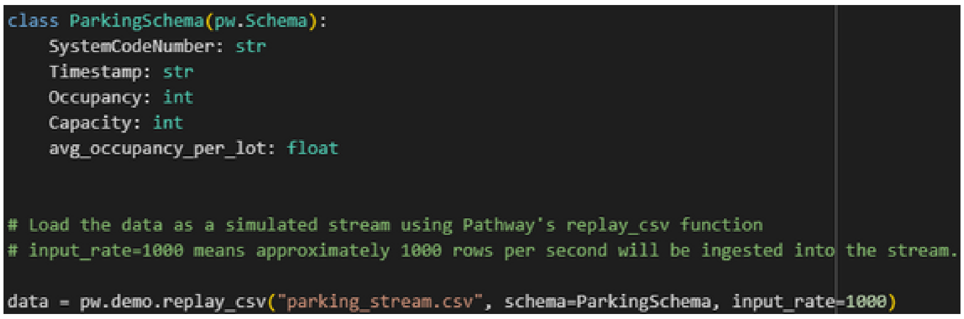
Data Preparation

To implement this model:

- I selected the necessary columns: SystemCodeNumber, Timestamp, Occupancy, Capacity, and avg_occupancy_per_lot.
- I saved this subset of the data into a CSV file (parking_stream.csv) for streaming.
- I defined a Pathway schema for the streaming data, matching these columns.



```python
# select columns that we will need for streaming or downstream processing
model1_df = df[["SystemCodeNumber", "Timestamp", "Occupancy", "Capacity", "avg_occupancy_per_lot"]]

# Save the selected columns to a CSV file
model1_df.to_csv("parking_stream.csv", index=False)
```



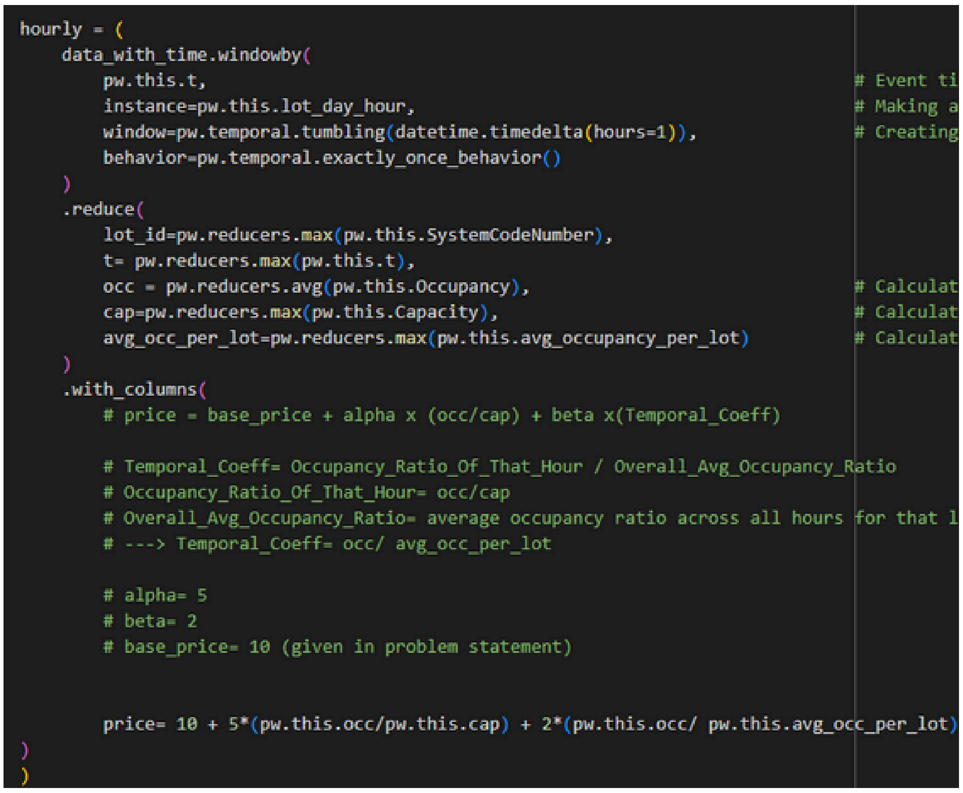
```python
class ParkingSchema(pw.Schema):
    SystemCodeNumber: str
    Timestamp: str
    Occupancy: int
    Capacity: int
    avg_occupancy_per_lot: float


# Load the data as a simulated stream using Pathway's replay_csv function
# input_rate=1000 means approximately 1000 rows per second will be ingested into the stream.

data = pw.demo.replay_csv("parking_stream.csv", schema=ParkingSchema, input_rate=1000)
```

MODEL 1

Real-Time Processing

Since the project required simulating real-time pricing, I used Pathway to:

- Load the dataset as a simulated stream.
- Define tumbling 1-hour windows per lot per day to aggregate hourly statistics.
- Compute the average occupancy and capacity in each window.
- Calculate the temporal coefficient and finally the price using the formula above.



```
hourly = (
    data_with_time.windowby(
        pw.this.t,                                                          # Event ti
        instance=pw.this.lot_day_hour,                                      # Making a
        window=pw.temporal.tumbling(datetime.timedelta(hours=1)),           # Creating
        behavior=pw.temporal.exactly_once_behavior()
    )
    .reduce(
        lot_id=pw.reducers.max(pw.this.SystemCodeNumber),
        t= pw.reducers.max(pw.this.t),
        occ = pw.reducers.avg(pw.this.Occupancy),                           # Calculat
        cap=pw.reducers.max(pw.this.Capacity),                              # Calculat
        avg_occ_per_lot=pw.reducers.max(pw.this.avg_occupancy_per_lot)      # Calculat
    )
    .with_columns(
        # price = base_price + alpha x (occ/cap) + beta x(Temporal_Coeff)

        # Temporal_Coeff= Occupancy_Ratio_Of_That_Hour / Overall_Avg_Occupancy_Ratio
        # Occupancy_Ratio_Of_That_Hour= occ/cap
        # Overall_Avg_Occupancy_Ratio= average occupancy ratio across all hours for that l
        # ---> Temporal_Coeff= occ/ avg_occ_per_lot

        # alpha= 5
        # beta= 2
        # base_price= 10 (given in problem statement)


        price= 10 + 5*(pw.this.occ/pw.this.cap) + 2*(pw.this.occ/ pw.this.avg_occ_per_lot)
)
)
```

Visualization

To visualize the output of this model:

- I used Bokeh to create a real-time line plot of the computed price for a selected lot (BHMBCCMKT01).
- The x-axis represented time, and the y-axis showed the dynamically computed price.
- The plot updated in real-time as the data stream progressed.

MODEL 1

Observations
From the real-time plot of Model 1:

- Prices increased steadily during the peak hours (around 12–2 PM), reflecting higher occupancy and temporal demand.
- During early morning and late afternoon, prices remained close to the base level as demand was lower.
- The model responded well to changing occupancy and provided a smooth, interpretable price trajectory.

This baseline linear model demonstrated the feasibility of using occupancy and temporal demand to adjust parking prices dynamically. In MODEL 2, I extended this idea further by incorporating additional features such as queue length, traffic, vehicle type, and special day effects into a more comprehensive demand-based model.

MODEL 2

In this section, I implemented a more sophisticated pricing model that incorporates multiple real-world factors influencing parking demand. This model builds on the baseline linear model by introducing a demand score, which captures not just occupancy but also queue length, traffic conditions, vehicle type, and special events.

Objective
The aim of this model was to account for a broader set of factors that influence parking demand and adjust the price accordingly. Unlike Model 1, which only considered occupancy and time-of-day effects, this model uses a mathematical demand function to reflect the true complexity of urban parking dynamics.

Demand Function
I designed the following demand function:
DemandScore = 0.3 × OccupancyRatio + 0.3 × NormQueueLength + (-0.15) × IsSpecialDay + 0.15 × VehicleTypeWeight + 0.2 × TrafficConditionWeight + 0.2 x hour_weight
Where:

- OccupancyRate: Current occupancy divided by capacity.
- NormalizedQueueLength: Queue length normalized (maximum queue length assumed to be 15).
- IsSpecialDay: Indicator (1 for special days, 0 otherwise).
- VehicleTypeWeight: Weight assigned based on vehicle type (car > bike > cycle, etc.).
- TrafficConditionWeight: Weight assigned based on nearby traffic (high > medium > low).
- TemporalCoefficient: Current hour's occupancy compared to lot's average.

I empirically chose the following weights for the coefficients:

- alpha = 0.3, beta = 0.3, gamma = -0.15, delta = 0.15, epsilon = 0.2, zeta = 0.2

These values ensured that occupancy and queue had the most influence, while special days slightly lowered demand (assuming some reduced work-related parking), and vehicle and traffic types provided finer adjustments.

Pricing Function
Once the demand score was computed, I normalized it (bounded between 0 and 1) and applied it in the following pricing function:
Price = BasePrice × (1 + lambda × NormalizedDemand^2)where lambda = 0.8, ensuring that prices were bounded and smooth, never exceeding twice the base price or dropping below half.

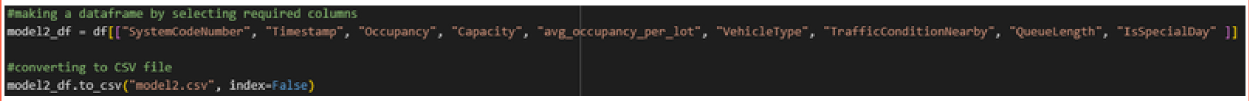
MODEL 2

Data Preparation

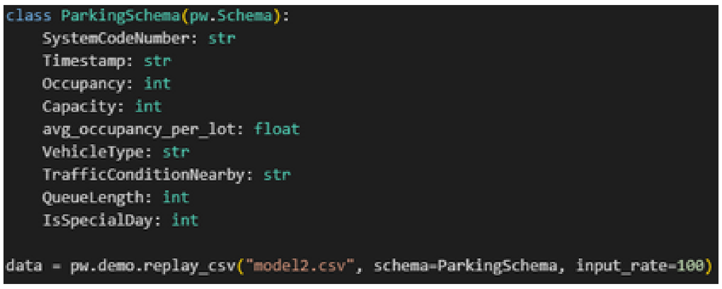
For this model, I included more columns from the dataset:

- VehicleType, TrafficConditionNearby, QueueLength, and IsSpecialDay in addition to the columns used in Model 1.
- I saved this enriched dataset into a CSV file (model2.csv) for streaming.

I also defined a Pathway schema to include all these fields.



```
#making a dataframe by selecting required columns
model2_df = df[["SystemCodeNumber", "Timestamp", "Occupancy", "Capacity", "avg_occupancy_per_lot", "VehicleType", "TrafficConditionNearby", "QueueLength", "IsSpecialDay" ]]

#converting to CSV file
model2_df.to_csv("model2.csv", index=False)
```

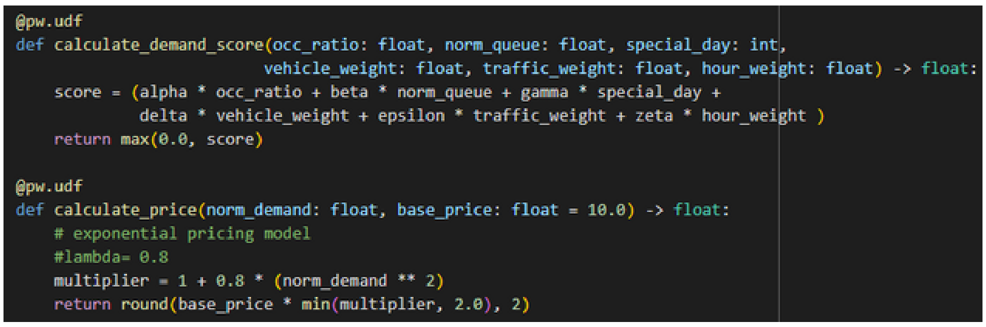


```
class ParkingSchema(pw.Schema):
    SystemCodeNumber: str
    Timestamp: str
    Occupancy: int
    Capacity: int
    avg_occupancy_per_lot: float
    VehicleType: str
    TrafficConditionNearby: str
    QueueLength: int
    IsSpecialDay: int

data = pw.demo.replay_csv("model2.csv", schema=ParkingSchema, input_rate=100)
```

Real-Time Processing

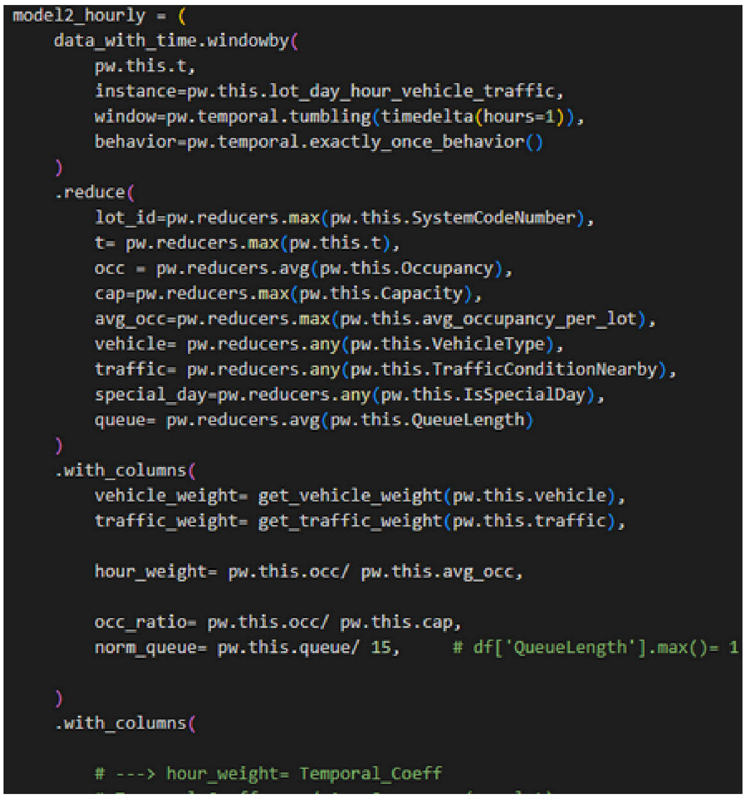
Using Pathway, I:

- Streamed the enriched data in real-time.
- Grouped records into hourly tumbling windows, uniquely identified by lot, hour, vehicle type, and traffic condition.
- Computed all intermediate variables needed for the demand function, including occupancy rate, normalized queue length, vehicle and traffic weights, and temporal coefficient.
- Calculated the demand score using a Pathway UDF (user-defined function).
- Computed the final price based on the demand score and base price.

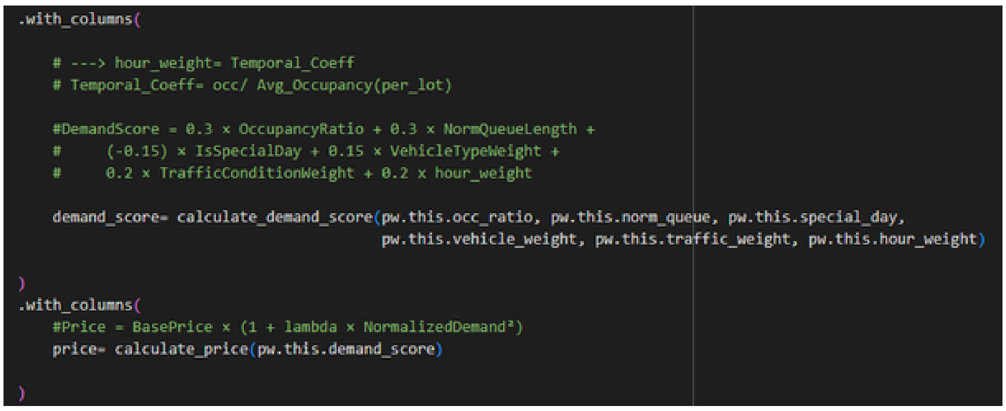


```
@pw.udf
def calculate_demand_score(occ_ratio: float, norm_queue: float, special_day: int,
                           vehicle_weight: float, traffic_weight: float, hour_weight: float) -> float:
    score = (alpha * occ_ratio + beta * norm_queue + gamma * special_day +
             delta * vehicle_weight + epsilon * traffic_weight + zeta * hour_weight )
    return max(0.0, score)

@pw.udf
def calculate_price(norm_demand: float, base_price: float = 10.0) -> float:
    # exponential pricing model
    #lambda= 0.8
    multiplier = 1 + 0.8 * (norm_demand ** 2)
    return round(base_price * min(multiplier, 2.0), 2)
```

MODEL 2



```
model2_hourly = (
    data_with_time.windowby(
        pw.this.t,
        instance=pw.this.lot_day_hour_vehicle_traffic,
        window=pw.temporal.tumbling(timedelta(hours=1)),
        behavior=pw.temporal.exactly_once_behavior()
    )
    .reduce(
        lot_id=pw.reducers.max(pw.this.SystemCodeNumber),
        t= pw.reducers.max(pw.this.t),
        occ = pw.reducers.avg(pw.this.Occupancy),
        cap=pw.reducers.max(pw.this.Capacity),
        avg_occ=pw.reducers.max(pw.this.avg_occupancy_per_lot),
        vehicle= pw.reducers.any(pw.this.VehicleType),
        traffic= pw.reducers.any(pw.this.TrafficConditionNearby),
        special_day=pw.reducers.any(pw.this.IsSpecialDay),
        queue= pw.reducers.avg(pw.this.QueueLength)
    )
    .with_columns(
        vehicle_weight= get_vehicle_weight(pw.this.vehicle),
        traffic_weight= get_traffic_weight(pw.this.traffic),

        hour_weight= pw.this.occ/ pw.this.avg_occ,

        occ_ratio= pw.this.occ/ pw.this.cap,
        norm_queue= pw.this.queue/ 15,     # df['QueueLength'].max()= 1

    )
    .with_columns(

        # ---> hour_weight= Temporal_Coeff
```



```
.with_columns(

    # ---> hour_weight= Temporal_Coeff
    # Temporal_Coeff= occ/ Avg_Occupancy(per_lot)

    #DemandScore = 0.3 × OccupancyRatio + 0.3 × NormQueueLength +
    #     (-0.15) × IsSpecialDay + 0.15 × VehicleTypeWeight +
    #     0.2 × TrafficConditionWeight + 0.2 x hour_weight

    demand_score= calculate_demand_score(pw.this.occ_ratio, pw.this.norm_queue, pw.this.special_day,
                                         pw.this.vehicle_weight, pw.this.traffic_weight, pw.this.hour_weight)

)
.with_columns(
    #Price = BasePrice × (1 + lambda × NormalizedDemand²)
    price= calculate_price(pw.this.demand_score)

)
```

MODEL 2

Visualization

As in Model 1, I visualized the results with Bokeh, producing a real-time line plot of prices for the selected lot (BHMBCCMKT01).

This plot clearly showed how the price responded more dynamically than in Model 1 — rising faster when multiple demand factors aligned (e.g., high occupancy, heavy traffic, long queue) and remaining moderate when demand was lower.

Observations

- Compared to Model 1, prices in Model 2 responded more sharply to sudden changes in demand (e.g., when queue length spiked or traffic worsened).
- The inclusion of vehicle type and traffic weights helped differentiate demand conditions more finely.
- On special days, the price occasionally adjusted downward, reflecting potentially lower work-related parking demand.
- The model maintained smooth, bounded price evolution and avoided erratic jumps.

This model demonstrated the benefits of incorporating multiple demand drivers into the pricing logic, providing a more realistic and effective dynamic pricing strategy for urban parking lots.

CONCLUSION

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.

REFERENCES

- 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.

THANK YOU !!!!