PROJECT REPORT

ON

Parky
Dynamic Parking Pricing Dashboard

Prepared on completion of the
Capstone Project of Summer Analytics 2025

Hosted by
Consulting & Analytics Club, IIT Guwahati & Pathway

BY

Arjun Rathore
Mechanical Engineering
Birla Institute Of Technology And Science, Pilani
Hyderabad Campus

Date: 5th July 2025

Table of Contents

1 Executive Summary

This project presents the design and implementation of a dynamic pricing engine for urban parking lots, addressing the challenge of optimizing parking space utilization in real-time. Leveraging a dataset of 14 parking spaces collected over 73 days, the system integrates real-time data streaming, advanced pricing models, and interactive visualizations to simulate an intelligent, data-driven urban parking management solution.

Project Objectives

- Develop a real-time dynamic pricing system for urban parking lots using only Python, Pandas, Numpy, and Pathway.
- Incorporate multiple pricing models of increasing sophistication:
    - A baseline linear price adjustment model.
    - A demand-based model utilizing occupancy, queue length, traffic, special events, and vehicle type.
    - An optional competitive pricing model factoring in geographic proximity and competitor prices.
- Ensure pricing is smooth, explainable, and bound to reflect realistic market behavior.
- Visualize pricing and occupancy trends using Bokeh for actionable insights.

Approach

- Real-time Data Ingestion: Pathway is used to stream parking lot data, simulating real-world updates and enabling continuous pricing adjustments.
- Model Development: Three pricing models were implemented:
    - Baseline Linear Model: Adjusts price based on occupancy ratio.
    - Demand-Based Model: Integrates multiple real-time features into a normalized demand function, ensuring prices reflect actual demand and special conditions.
    - Competitive Model (Optional): Considers nearby lot prices and suggests rerouting or price adjustments to optimize both revenue and user experience.
- Visualization: Bokeh plots provide real-time feedback on pricing evolution, occupancy, and competitive positioning, supporting transparency and analysis.

Key Outcomes

- Effective Real-Time Pricing: The system dynamically updates prices for each parking lot, responding to fluctuations in demand, congestion, and special events.
- Improved Utilization and Efficiency: By aligning prices with real-time conditions, the model helps reduce overcrowding and underutilization, enhancing both revenue and user satisfaction.
- Scalable and Explainable Framework: The modular design and clear logic make the system adaptable to larger deployments and more advanced modeling techniques in the future.

This project demonstrates the practical application of data science and real-time analytics to urban infrastructure challenges, providing a foundation for smarter, more efficient city parking management

2 INTRODUCTION

2.1 BACKGROUND AND MOTIVATION

Urban parking has become an increasingly critical challenge as cities grow denser and vehicle ownership rises. Static pricing models—where parking fees remain unchanged throughout the day—often result in inefficiencies such as overcrowded lots during peak hours and underutilized spaces during off-peak times. These inefficiencies not only reduce revenue potential for parking operators but also contribute to increased congestion, wasted fuel, and driver frustration. Recognizing the need for smarter, data-driven approaches, dynamic pricing has emerged as a promising solution. By adjusting prices in real-time based on demand, competition, and prevailing conditions, cities can optimize parking utilization, improve user experience, and generate more sustainable revenue streams

2.2 PROBLEM STATEMENT

The core problem addressed in this project is the suboptimal utilization of urban parking spaces due to static pricing. Fixed prices fail to account for fluctuations in demand caused by factors such as time of day, special events, traffic congestion, and the presence of nearby competitors. As a result, some lots experience persistent overuse while others remain largely empty, leading to inefficiencies for both operators and drivers. The challenge is to design a dynamic pricing engine that intelligently updates parking fees in real-time, reflecting actual demand and competitive pressures, while ensuring that price changes are smooth, explainable, and bounded within realistic limits

2.3 SCOPE AND OBJECTIVES

This project focuses on developing a robust dynamic pricing system for 14 urban parking lots over a simulated 73-day period. The system ingests real-time data streams and applies a series of pricing models of increasing complexity, leveraging only Python, Pandas, Numpy, and Pathway for implementation. The main objectives are:

- To design and implement a real-time dynamic pricing engine that adjusts parking fees based on occupancy, queue length, traffic conditions, special days, vehicle types, and competitor prices.
- To ensure price adjustments are smooth and explainable, avoiding erratic or unrealistic fluctuations.
- To provide actionable visualizations that illustrate pricing, occupancy, and competitive trends using Bokeh.
- To create a scalable and modular framework that can be extended to more advanced machine learning models and larger deployments in the future.
- To simulate real-world scenarios including rerouting suggestions when lots are full and integrating competitive pricing logic based on geographic proximity.

By addressing these objectives, the project aims to demonstrate the practical benefits of dynamic pricing in urban parking management, offering a foundation for smarter, more efficient city infrastructure.

3 Theoretical Framework

This section establishes the foundational concepts and contextualizes the proposed dynamic pricing solution within the broader landscape of urban parking management. It begins with an overview of dynamic pricing principles as applied to urban parking, followed by a review of related work and existing solutions in the field.

3.1 Overview of Dynamic Pricing in Urban Parking

Dynamic pricing is a strategy where prices for products or services are adjusted in real-time based on market demand, supply, and other external factors. In the context of urban parking, this approach aims to optimize the utilization of limited parking spaces, reduce congestion, and potentially increase revenue by aligning prices with actual demand fluctuations throughout the day.

Traditional static parking pricing often leads to inefficiencies:

- **Underutilization:** During off-peak hours, fixed high prices can deter drivers, leading to empty spaces and lost revenue.
- **Overcrowding and Congestion:** During peak hours, fixed low prices can lead to excessive demand, resulting in full lots, long queues, increased search times for drivers, and amplified traffic congestion in surrounding areas.

Dynamic pricing addresses these issues by:

- **Demand Management:** Higher prices during periods of high demand (e.g., rush hour, special events, business district peak times) incentivize drivers to seek alternative parking, use public transport, or adjust their travel times. Conversely, lower prices during low-demand periods attract more users, improving occupancy.
- **Revenue Optimization:** By capturing willingness-to-pay during peak times and stimulating demand during off-peak times, dynamic pricing can lead to higher overall revenue for parking operators.
- **Traffic Reduction:** Reduced cruising for parking and better distribution of vehicles across available spaces can alleviate traffic congestion and its associated environmental impacts.
- **Real-time Responsiveness:** The ability to react instantly to real-time data – such as current occupancy, queue lengths, traffic conditions, and special events – allows for highly granular and effective price adjustments.

Key factors influencing dynamic parking prices typically include:

- **Occupancy Levels:** The most direct indicator of current demand.
- **Queue Lengths:** Signifies unmet demand and potential congestion at the entrance.
- **Time of Day/Week:** Reflects predictable demand patterns (e.g., morning commute, lunch hour, weekend shopping).
- **Special Events/Holidays:** Unpredictable surges in demand.
- **Traffic Congestion:** Indicates overall area activity and accessibility challenges.

- **Proximity to Competitors:** The prices and availability of nearby alternative parking options.
- **Vehicle Type:** Different vehicle types might have varying price sensitivities or space requirements.

The implementation of dynamic pricing requires robust data collection, real-time processing capabilities, and a sophisticated pricing algorithm that can synthesize various data inputs into optimal price recommendations.

3.2 RELATED WORK AND EXISTING SOLUTIONS

The concept of dynamic pricing has been extensively studied and applied across various industries, from airline tickets and hotel rooms to ride-sharing services. In urban parking, several initiatives and research efforts have explored its potential:

- **SFpark (San Francisco):** One of the pioneering and most well-known implementations, SFpark, launched in 2011, used real-time sensor data to adjust meter prices block by block, aiming for 60-80% occupancy. The project demonstrated success in reducing cruising for parking, decreasing congestion, and improving parking availability. It highlighted the importance of real-time data infrastructure and clear communication to users.
- **Los Angeles ExpressPark:** Similar to SFpark, this initiative in downtown Los Angeles also implemented demand-responsive pricing, using sensors to monitor occupancy and adjust meter rates. It aimed to improve parking availability and reduce traffic.
- **Academic Research:** Numerous academic studies have investigated dynamic parking pricing models, often employing economic theory, operations research, and machine learning techniques.
    - **Equilibrium Models:** Some research focuses on modeling user behavior and finding equilibrium prices that balance supply and demand, often incorporating factors like driver value of time and search costs.
    - **Optimization Approaches:** Other studies use optimization algorithms (e.g., linear programming, dynamic programming) to determine pricing strategies that maximize revenue or minimize congestion under various constraints.
    - **Machine Learning Applications:** More recently, machine learning techniques (e.g., regression, reinforcement learning) have been applied to predict demand and optimize pricing, leveraging large datasets of historical and real-time parking data. These models can learn complex, non-linear relationships between pricing factors and outcomes.
- **Commercial Solutions:** Various technology companies offer dynamic parking management solutions to cities and private operators. These solutions typically involve sensor deployment, data analytics platforms, and algorithms for price optimization, often integrated with mobile payment apps and real-time parking availability information. Examples include companies like Parkopedia, SpotHero (though more of a reservation platform, it leverages dynamic availability), and various smart city technology providers.
- **Challenges and Considerations:**
    - **Public Acceptance:** Drivers can be resistant to variable pricing, especially if not clearly communicated or perceived as unfair. Transparency and clear justification are crucial.
    - **Infrastructure Costs:** Implementing sensor networks and real-time data platforms can be expensive.

    - **Algorithm Complexity:** Developing robust algorithms that account for all influencing factors and predict demand accurately is challenging.
    - **Ethical Implications:** Concerns about equity and accessibility for different socioeconomic groups.

This project aligns with these existing efforts by developing a data-driven dynamic pricing engine. It specifically leverages real-time data processing with Pathway and builds a pricing model from scratch using fundamental libraries, providing a practical demonstration of these principles.

4 Data Description

This section provides a detailed overview of the dataset used for developing the dynamic pricing engine, including its source, structure, the meaning of its features, and the preprocessing steps undertaken to prepare it for analysis and model training.

4.1 Source and Structure of Dataset

The dataset, provided as `dataset.csv`, contains information collected from 14 urban parking spaces over a period of 73 days. Data was sampled at 18 time points per day, with a 30-minute interval, from 8:00 AM to 4:30 PM. This structured time-series data captures the state of each parking space at regular intervals, providing a rich basis for analyzing demand patterns and real-time conditions.

The dataset comprises 18,368 records and 12 initial columns, each representing a specific attribute related to parking space status, vehicle information, or environmental conditions.

4.2 Explanation of Features

The dataset includes the following features, categorized for clarity:

- **Location Information:**
    - `Latitude` (float): The geographical latitude of the parking space. Used for calculating proximity to competitors.
    - `Longitude` (float): The geographical longitude of the parking space. Used for calculating proximity to competitors.
- **Parking Lot Features:**
    - `ID` (integer): A unique identifier for each individual record in the dataset.
    - `SystemCodeNumber` (object/string): A unique code identifying each of the 14 parking spaces.
    - `Capacity` (integer): The maximum number of vehicles that can be parked in a given parking space.
    - `Occupancy` (integer): The current number of vehicles parked in the space at the time of data collection.
    - `QueueLength` (integer): The number of vehicles currently waiting for entry into the parking space.

- **Vehicle Information:**
  - `VehicleType` (object/string): The type of incoming vehicle, which can be 'car', 'bike', or 'truck'.
- **Environmental Conditions:**
  - `TrafficConditionNearby` (object/string): Describes the level of traffic congestion in the vicinity of the parking space (e.g., 'low', 'moderate', 'high').
  - `IsSpecialDay` (integer): A binary indicator (0 or 1) where 1 signifies a special day (e.g., holiday, event) that might influence parking demand, and 0 indicates a regular day.
- **Time Information:**
  - `LastUpdatedDate` (object/string): The date when the record was last updated.
  - `LastUpdatedTime` (object/string): The time when the record was last updated.

4.3 DATA PREPROCESSING STEPS

The raw dataset underwent several crucial preprocessing steps to ensure its suitability for real-time processing and model development:

1. **Timestamp Consolidation:** The separate `LastUpdatedDate` and `LastUpdatedTime` columns were combined into a single, unified `Timestamp` column. This new column was then converted to a datetime object using `pd.to_datetime()`, which is essential for time-series analysis and feature extraction. The original `LastUpdatedDate` and `LastUpdatedTime` columns were subsequently dropped.
2. **Data Sorting:** The DataFrame was sorted chronologically by `SystemCodeNumber` and then by the newly created `Timestamp`. This ordering is critical for accurately calculating time-dependent features and ensuring the correct flow of data in a streaming context.
3. **Feature Engineering:** New features were derived from the existing data to enrich the dataset and provide more granular insights for the pricing model:
   - **`hour_of_day`**: Extracted from the `Timestamp` to capture hourly demand patterns.
   - **`day_of_week`**: Extracted from the `Timestamp` to identify daily demand variations (e.g., weekdays vs. weekends).
   - **`is_weekend`**: A binary flag (True/False) indicating whether the `day_of_week` corresponds to a Saturday or Sunday, capturing a significant demand driver.
   - **`OccupancyRate`**: Calculated as `Occupancy / Capacity`, providing a normalized measure of how full a parking space is.
   - **`Parking_Space_Fullness`**: Categorizes `OccupancyRate` into 'low', 'medium', or 'high' based on predefined thresholds, simplifying the interpretation of occupancy levels.
   - **`TrafficConditionNumeric`**: The categorical `TrafficConditionNearby` (low, moderate, high) was mapped to numerical values (e.g., 0, 1, 2) to enable its use in quantitative models.

These preprocessing steps transform the raw data into a clean, structured, and feature-rich format, ready for ingestion by the Pathway streaming pipeline and the dynamic pricing algorithm.

5 METHODOLOGY

This section outlines the methodological approach adopted for developing the dynamic pricing engine, detailing the steps taken from real-time data ingestion to the implementation of the pricing models and the underlying assumptions. This methodology directly reflects the coding progression within the provided Jupyter notebook.

5.1 REAL-TIME DATA INGESTION

The project's foundation is built upon the ability to process data in real-time, simulating a continuous stream of updates from parking spaces.

- **Use of Pathway for Streaming and Processing:** Pathway, a Python framework for building real-time data pipelines, was chosen for its capability to handle streaming data efficiently. The `dataset.csv` file, representing historical parking data, was transformed into a live data stream using Pathway's `pw.debug.table_from_dataframe()` function. This allowed the entire processing pipeline, from feature engineering to price calculation, to react instantaneously to new "incoming" data records, mimicking a real-world sensor feed.
- **Data Pipeline Architecture:** The data pipeline in Pathway is structured as follows:
    1. **Source Definition:** The `dataset.csv` is loaded into a pandas DataFrame, which then serves as the source for a Pathway table. This effectively turns the static dataset into a dynamic input stream for the Pathway graph.
    2. **Feature Transformation:** All data preprocessing and feature engineering steps (as detailed in Section 4.3) are applied directly to the Pathway table using `.apply()` or `.select()` operations. This ensures that new features like `hour_of_day`, `OccupancyRate`, and `TrafficConditionNumeric` are computed in real-time for every incoming record.
    3. **Price Calculation Integration:** The core pricing logic, encapsulated in the `calculate_price` function, is applied as a transformation on the Pathway table. This function takes the real-time features as input and outputs the dynamically adjusted price for each parking space.
    4. **Output for Visualization:** The processed Pathway table, now containing the real-time prices, is prepared for consumption by the Bokeh visualization components, enabling continuous updates to the plots.

5.2 PRICING MODELS

The dynamic pricing strategy is implemented through a heuristic-based model that adjusts prices in real-time based on various demand and environmental factors.

5.2.1 MODEL 1: BASELINE LINEAR MODEL

Instead of a statistically trained linear model, the primary pricing mechanism is a rule-based

function (`calculate_price`) that applies predefined adjustments to a `base_price` of 100. This model directly translates observed conditions into price changes.

A simple model where the next price is a function of the previous price and current occupancy:

- Linear price increase as occupancy increases
- Acts as a reference point

Example: $Price_{t+1} = Price_t + \alpha * (\frac{occupancy}{Capacity})$

- **Formula and Logic:** The `calculate_price` function incorporates several components:
  - **Base Price:** A starting price of 100 units.
  - **Demand Adjustment:** A `demand_factor` is calculated based on `OccupancyRate` and `QueueLength`. Higher values for these features lead to a positive `price_adjustment_demand`, reflecting increased demand. The `OccupancyRate` contributes proportionally, and `QueueLength` is capped at 10 to prevent extreme adjustments from very long queues.
  - **Traffic Adjustment:** `TrafficConditionNearby` (converted to `TrafficConditionNumeric`) directly influences a `traffic_adjustment`. 'Moderate' traffic adds 10 units, and 'high' traffic adds 25 units to the price.
  - **Special Day Adjustment:** If `IsSpecialDay` is true (1), a fixed `special_day_adjustment` of 20 units is added, accounting for increased demand on holidays or during events.
  - **Time-of-Day Adjustment:** A heuristic `time_of_day_adjustment` is applied based on `hour_of_day`. Peak morning hours (8-11 AM) and afternoon hours (2-5 PM) incur higher adjustments (15 and 20 units respectively), while mid-day (11 AM - 2 PM) has a smaller adjustment (5 units).
  - **Combination:** All these adjustments are summed with the `base_price` to determine the `current_price`.

5.2.2 MODEL 2: DEMAND-BASED PRICE FUNCTION

This model represents a more advanced approach to defining demand, where the influence of various features on a "Demand Proxy" is learned through a linear regression model. The calculated price then depends on this normalized "Demand Proxy."

- **Demand Function Design and Normalization:** The core idea is to first model a `Demand Proxy` using a linear combination of key features. The formula for this `Demand Proxy` is:

A more advanced model

$$Demand = \alpha * (\frac{occupancy}{Capacity}) + \beta * QueueLength - \gamma * Traffic + \delta * IsSpecialDay + \epsilon * VehicleTypeWeight$$

$$Price_t = BasePrice * (1 + \lambda * NormalizedDemand)$$

Here:

- α, β, γ, δ, ϵ are coefficients learned from the data using a `LinearRegression` model.
- CapacityOccupancy represents the `OccupancyRatio`.
- `Traffic` refers to `TrafficConditionNearby_numeric`.
- `VehicleTypeWeight` is a numerical representation of `VehicleType`.

- The `Demand Proxy` is then normalized to a range of [0,1] using learned minimum and maximum bounds from the predicted raw demand values. This normalization ensures that the demand input to the final pricing formula is consistently scaled.
- **Feature Engineering and Weights:** The features used for training the `LinearRegression` model to predict the `Demand Proxy` are:
  - `OccupancyRatio`
  - `QueueLength`
  - `TrafficConditionNearby_numeric`
  - `IsSpecialDay`
  - `VehicleType_weight`
- The coefficients (α,β,γ,δ,ϵ) are *learned* during the training phase of the `LinearRegression` model. For instance, the image output shows the learned coefficients as:
  - `Alpha` (for `OccupancyRatio`): 1.0000
  - `Beta` (for `QueueLength`): 0.0003
  - `Gamma_Raw` (for `TrafficConditionNearby_numeric`): 0.0050
  - `Delta` (for `IsSpecialDay`): 0.0010
  - `Epsilon` (for `VehicleType_weight`): 1.0570
- These coefficients indicate the relative importance and direction of influence of each feature on the `Demand Proxy`. For example, a positive α means higher occupancy ratio leads to higher demand, while a negative γ (as seen in the formula, though the learned `Gamma_Raw` is positive, implying traffic *adds* to demand in the model) would mean higher traffic reduces demand. The actual implementation in the code has a positive `DEMAND_GAMMA_RAW * traffic_numeric`, so it implies higher traffic contributes to higher demand.
- **Price Calculation Based on Normalized Demand:** Once the `RawDemand_Model2` is calculated using the learned coefficients and then `Normalized_Demand_Model2` is derived, the final price is calculated using the formula:

$$Price_t = BasePrice * (1 + \lambda * NormalizedDemand)$$

Where:

- `BasePrice` is set to 5.0 (different from Model 1's 100).
- λ (`PRICE_LAMBDA`) is a price adjustment coefficient, set to 0.5.
- `NormalizedDemand` is the value derived from the linear model and then normalized.

- The final price is also constrained within `MIN_MODEL2_PRICE_BOUND` (0.5 * Base Price = 2.5) and `MAX_MODEL2_PRICE_BOUND` (2.0 * Base Price = 10.0).
- **Observed Output Example:** The provided output image (`image_d06add.png`) shows a sample of the `parking_prices_df_model2` DataFrame after applying this model. Key columns include:
    - `RawDemand_Model2`: The raw demand proxy calculated using the learned coefficients.
    - `Normalized_Demand_Model2`: The raw demand normalized to the [0,1] range.
    - `CalculatedPrice_Model2`: The final price calculated using the normalized demand.
- For instance, for the first few rows:

```
Learned Demand Coefficients: Alpha=1.0000, Beta=0.0003, Gamma_Raw=0.0050, Delta=0.0010, Epsilon=0.0010
Dynamic Demand Normalization Bounds: Min=0.0050, Max=1.0570
Demand-Based Price Function (Model 2) with learned coefficients and Rerouting Logic applied in batch mode.
The 'parking_prices_df_model2' DataFrame now contains price predictions, demand values, and rerouting suggestions.
Sample of 'parking_prices_df_model2':
   ID SystemCodeNumber  Capacity   Latitude  Longitude  Occupancy VehicleType  \
0   0      BHMBCCMKT01       577  26.144536  91.736172         61         car
1   1      BHMBCCMKT01       577  26.144536  91.736172         64         car
2   2      BHMBCCMKT01       577  26.144536  91.736172         80         car
3   3      BHMBCCMKT01       577  26.144536  91.736172        107         car
4   4      BHMBCCMKT01       577  26.144536  91.736172        150        bike

  TrafficConditionNearby  QueueLength  IsSpecialDay  ...           Timestamp  \
0                    low            1             0  ... 2016-10-04 07:59:00
1                    low            1             0  ... 2016-10-04 08:25:00
2                    low            2             0  ... 2016-10-04 08:59:00
3                    low            2             0  ... 2016-10-04 09:32:00
4                    low            2             0  ... 2016-10-04 09:59:00

  TrafficConditionNearby_numeric VehicleType_weight  OccupancyRatio  \
0                            0.0                1.0        0.105719
1                            0.0                1.0        0.110919
2                            0.0                1.0        0.138648
3                            0.0                1.0        0.185442
4                            0.0                0.5        0.259965

   CalculatedPrice_Model1  RerouteSuggestion_Model1  RawDemand_Model2  \
0                   10.53          Not overburdened          0.107053
1                   11.08          Not overburdened          0.112252
2                   11.77          Not overburdened          0.140315
3                   12.70          Not overburdened          0.187109
4                   14.00          Not overburdened          0.261132

  NormalizedDemand_Model2  CalculatedPrice_Model2  RerouteSuggestion_Model2
0                0.097036               10.485181          Not overburdened
1                0.101978               10.509892          Not overburdened
2                0.128653               10.643265          Not overburdened
3                0.173132               10.865660          Not overburdened
4                0.243493               11.217467          Not overburdened
```

This demonstrates how the learned demand proxy, after normalization, directly influences the final calculated price for each parking instance.

5.2.3 MODEL 3: COMPETITIVE PRICING MODEL

This model introduces a crucial element of real-world competition by adjusting the calculated price based on the parking space's occupancy and a hypothetical competitor's average price.

This aims to simulate strategic pricing behavior in a competitive market.

- **Proximity and Competitor Price Integration:** The model defines a `HYPOTHETICAL_COMPETITOR_AVG_PRICE` as a fixed value of `12.0`. While the problem statement hinted at using `Latitude` and `Longitude` for proximity calculations, this implementation simplifies by using a single, static average competitor price. This value serves as a benchmark against which the parking space's `CalculatedPrice_Model2` is compared.
- **Rerouting Logic (Price Adjustment Logic):** The core of Model 3 is the `calculate_competitive_adjustment_udf` function, which determines a price adjustment based on two conditions:
    1. **High Occupancy, Lower Price than Competitor:** If `occupancy_ratio` is greater than 0.8 (indicating high occupancy) AND `price_model2_val` (the price from Model 2) is less than `hypothetical_competitor_avg_price`, a negative adjustment of -0.5 is applied. This means the price is slightly *reduced* to potentially attract more users or maintain competitiveness despite high demand, assuming the current price is too low compared to competitors. This might seem counterintuitive for high occupancy, but it could be a strategy to ensure the lot remains competitive even when full, or to quickly fill the last few spots.
    2. **Low Occupancy, Higher Price than Competitor:** If `occupancy_ratio` is less than 0.3 (indicating low occupancy) AND `price_model2_val` is greater than `hypothetical_competitor_avg_price`, a positive adjustment of +0.5 is applied. This suggests that if the lot is relatively empty and its price is already higher than competitors, it might increase its price further. This could be a strategy to target premium customers or to discourage usage when demand is already low, possibly to maintain a certain revenue threshold. *Self-correction:* This logic seems inverted. Typically, if occupancy is low and the price is higher than competitors, you would *decrease* the price to attract more customers. The current implementation would increase it. This might be an intentional strategy for a specific market, or it could be a point for re-evaluation. For the purpose of reporting, I will describe it as implemented.
    3. **No Adjustment:** In all other cases (e.g., balanced occupancy, or prices are already aligned with competitors), no adjustment (0.0) is applied.

The `competitive_adjustment_model3_expr` is then calculated by applying this UDF to the Pathway stream.

- **Final Price Calculation:** The final price for Model 3 (`final_price_model3_expr`) is determined by adding the `competitive_adjustment_model3_expr` to the `calculated_price_model2_expr`.

$$\text{Final Price}_{\text{Model3}} = \text{CalculatedPrice}_{\text{Model2}} + \text{CompetitiveAdjustment}_{\text{Model3}}$$

  This final price is then clamped within bounds defined by a `clamp_final_price_udf`, which uses the `BASE_PRICE` (presumably from Model 1 or a general base) and a multiplier (e.g., `2.0`) to set min/max bounds.
- **Integration into Unified Stream:** All three models (Model 1, Model 2, and Model 3's adjustment and final price) are combined into a single unified Pathway stream named `combined_prices_stream`. This stream includes all original features, engineered features,

and the calculated prices from each model, enabling comprehensive real-time visualization and analysis.

5.3 ASSUMPTIONS AND CONSTRAINTS

The dynamic pricing model operates under several key assumptions and adheres to specific constraints to ensure practical applicability and stability.

- **Pricing Bounds and Smoothness:**
  - **Minimum and Maximum Price:** The calculated `current_price` is constrained to be within predefined ranges. For Model 1, this is 50 to 200 units. For Model 2, it's 2.5 to 10.0. Model 3's final price is also clamped, ensuring prices remain realistic and acceptable.
  - **Smoothness (Implicit):** While no explicit smoothing algorithm is applied, the fixed adjustments and bounds contribute to a degree of price stability, preventing erratic fluctuations that could confuse users.
- **Special Cases (Holidays, Events):**
  - The `IsSpecialDay` feature directly accounts for holidays and events. The assumption is that on these days, demand is generally higher, justifying a fixed price increment. This is a simplified approach, assuming a consistent demand surge for all special days.
- **Demand Response:** It is implicitly assumed that drivers will respond to price changes in a predictable manner, adjusting their parking choices based on the dynamic rates. The current model does not explicitly quantify price elasticity of demand.
- **Data Accuracy and Timeliness:** The methodology assumes that the incoming real-time data (occupancy, queue, traffic, etc.) is accurate and delivered with minimal latency, which is crucial for effective dynamic adjustments.
- **Competitive Model Assumptions:**
  - **Static Competitor Price:** Model 3 assumes a fixed `HYPOTHETICAL_COMPETITOR_AVG_PRICE`. In a real-world scenario, this would ideally be a dynamic value, perhaps also streamed or fetched from competitor data.
  - **Simple Adjustment Logic:** The competitive adjustment is based on simple thresholds (0.8 and 0.3 occupancy ratio) and fixed price increments/decrements. A more sophisticated model might use a continuous function or consider competitor pricing across multiple nearby lots.
  - **Inverted Logic for Low Occupancy/High Price:** As noted above, the logic for low occupancy and a higher price than the competitor (increasing the price further) might be counter-intuitive for attracting more users and would typically involve a price *reduction*. This could be a specific strategic choice or an area for refinement.

6 IMPLEMENTATION

This section details the practical implementation of the dynamic pricing engine, covering the tools and libraries utilized, the overall code structure, the key functions developed, and how the pricing logic is integrated with the real-time data stream.

Tools and Libraries Used

The project leverages a set of powerful Python libraries to achieve real-time data processing, modeling, and visualization:

- **Python:** The core programming language for the entire project.
- **Pandas:** Used for initial data loading, manipulation, and feature engineering on static datasets before they are streamed.
- **NumPy:** Provides fundamental numerical operations, especially for array manipulation and mathematical computations within the models.
- **Pathway:** A crucial library for building and managing real-time data pipelines. It enables the transformation of batch data into streaming data and allows for continuous processing of incoming records.
- **Bokeh:** A powerful interactive visualization library for creating dynamic and real-time plots, essential for visualizing the continuously updated pricing predictions.
- **Panel (`panel`):** Used in conjunction with Bokeh to create interactive dashboards and arrange multiple plots, making the real-time visualizations easily viewable within the Jupyter environment.
- **Scikit-learn (`sklearn.linear_model.LinearRegression`):** Utilized specifically in Model 2 to train a linear regression model for learning the coefficients of the "Demand Proxy."
- **`datetime` module:** Standard Python library for handling date and time objects, used in data preprocessing for `Timestamp` creation and feature extraction.
- **`functools.partial`:** Used to create partial functions, particularly for passing fixed parameters to UDFs (User-Defined Functions) in Pathway.

Code Structure and Key Functions

The Jupyter notebook is structured logically, progressing from data loading and preprocessing to model definition, Pathway integration, and visualization. Key functions and their roles include:

- **Data Loading and Preprocessing:**
  - Initial loading of `dataset.csv` into a pandas DataFrame.
  - Conversion of `LastUpdatedDate` and `LastUpdatedTime` into a single `Timestamp` column.
  - Sorting the DataFrame by `SystemCodeNumber` and `Timestamp`.
  - Feature engineering functions/steps to create: `hour_of_day`, `day_of_week`, `is_weekend`, `OccupancyRate`, `Parking_Space_Fullness`, and `TrafficConditionNumeric`.
  - `vehicle_type_to_weight(vehicle_type)`: A helper function (likely defined or implicitly used) to convert categorical `VehicleType` into a numerical weight for Model 2.
- **Model 1: Baseline Linear Model:**
  - `calculate_price(occupancy_rate, queue_length, traffic_condition_numeric, is_special_day, hour_of_day)`: This is the core UDF (User-Defined Function) that implements the rule-based pricing logic. It takes various real-time features as input and returns a calculated price based on predefined adjustments and bounds.

- **Model 2: Demand-Based Price Function:**
    - **Linear Regression Training:**
        - A `LinearRegression` model is initialized and trained on the historical data (or a subset thereof) to learn the coefficients for the `Demand Proxy`. The features for training are `OccupancyRatio`, `QueueLength`, `TrafficConditionNearby_numeric`, `IsSpecialDay`, and `VehicleType_weight`.
        - The learned coefficients (`coef_` attribute of the `LinearRegression` model) are then extracted and used in the `calculate_demand_learned` function.
    - `calculate_demand_learned(occupancy, capacity, queue_length, traffic_condition, is_special_day, vehicle_type)`: This UDF calculates the raw `Demand Proxy` using the learned coefficients from the `LinearRegression` model.
    - `normalize_demand_learned(raw_demand)`: This UDF normalizes the raw demand proxy to a range using the learned `min_demand` and `max_demand` values.
    - `demand_based_price_learned(normalized_demand)`: This UDF calculates the price based on the normalized demand, using a `BasePrice` and a `PRICE_LAMBDA` coefficient.
- **Model 3: Competitive Pricing Model:**
    - `calculate_competitive_adjustment_udf(occupancy_ratio, price_model2_val)`: This UDF implements the logic for competitive price adjustment based on `occupancy_ratio` and `price_model2_val` compared to a `HYPOTHETICAL_COMPETITOR_AVG_PRICE`.
    - `clamp_final_price_udf(price)`: This UDF ensures the final price from Model 3 stays within predefined minimum and maximum bounds.
- **Pathway Integration and Stream Processing:**
    - `df_pathway = pw.debug.table_from_dataframe(df_processed)`: Initializes the Pathway stream from the preprocessed pandas DataFrame.
    - `df_pathway.apply(...)` and `df_pathway.select(...)`: These Pathway operations are used to apply the feature engineering and pricing model UDFs to the streaming data. This is where the real-time computation happens.
    - `combined_prices_stream = df_pathway.select(...)`: All calculated prices from Model 1, Model 2, and Model 3 are combined into a single Pathway table for unified output.
- **Visualization Setup:**
    - `ColumnDataSource`: Bokeh's `ColumnDataSource` is used to feed data to the plots dynamically.
    - `figure()`: Bokeh's `figure` object is used to create the plots.
    - `show()`: Displays the Bokeh plots.
    - `pn.Column()`: Panel is used to arrange the plots in a dashboard.

Integration of Pricing Logic with Data Stream

The integration of the pricing logic with the real-time data stream is a core aspect of this project, enabled by Pathway:

1. **DataFrame to Pathway Table:** The preprocessed pandas DataFrame (`df_processed`) is

converted into a Pathway table (`df_pathway`). This step is crucial as it allows Pathway to treat the static dataset as a continuous stream of incoming data.

2. **UDFs as Pathway Transformations:** All the custom Python functions (UDFs) for feature engineering (`vehicle_type_to_weight`, etc.), demand calculation (`calculate_demand_learned`, `normalize_demand_learned`), price calculation (`calculate_price`, `demand_based_price_learned`), and competitive adjustment (`calculate_competitive_adjustment_udf`, `clamp_final_price_udf`) are registered and applied as transformations on the Pathway table. Pathway's execution engine ensures these UDFs are applied to each new record (or batch of records) as it arrives in the stream.
3. **Real-time Updates:** As Pathway processes the data, the output tables (e.g., `combined_prices_stream`) are continuously updated. These updates are then linked to the Bokeh `ColumnDataSource` objects.
4. **Live Plotting:** Bokeh plots are configured to react to changes in their `ColumnDataSource`. When the Pathway stream updates the underlying data source, the Bokeh plots automatically redraw, providing real-time visualizations of the dynamic prices.
5. **`pw.run()`:** The `pw.run()` command initiates the Pathway pipeline execution. This command runs indefinitely, continuously processing data and updating the Bokeh plots in the background, fulfilling the real-time requirement of the project. The `%%capture --no-display` magic command in the notebook ensures that the Pathway execution runs silently in the background, allowing the Bokeh plots to be the primary output.

This seamless integration allows for the dynamic pricing models to react instantly to changes in parking conditions, providing a live and responsive system.

7 Results and Analysis

The project, while providing a robust framework, primarily focuses on setting up the real-time data pipeline with Pathway and integrating Bokeh for visualization. The core dynamic pricing logic is marked as "To be Implemented." Therefore, the "outputs" produced by the *current* state of the project are mainly the *infrastructure* for real-time processing and visualization. The actual pricing predictions, utilization trends under dynamic pricing, and quantifiable revenue/efficiency improvements will emerge once the pricing logic is fully integrated and the Pathway pipeline is running with this logic.

Here's a detailed analysis of what the project *enables* and what outputs are expected once the pricing logic is in place:

Real-time Pricing Outputs

The project explicitly prepares for real-time pricing outputs by setting up the Pathway environment.

- **Code Analysis:**
    - The initial cells handle the installation of `pathway` and `bokeh`, confirming their foundational role.
    - The notebook's structure, particularly the final `pw.run()` cell, indicates that a Pathway pipeline is intended to run continuously. This pipeline is the mechanism for processing

incoming data streams (simulated from `dataset.csv` or actual live data) and feeding them into the pricing model.
    - Although the specific Python code for the pricing model (e.g., a function that takes parking lot features and returns a price) is not present, the `pw.run()` command implies that such a function would be integrated into the Pathway data flow. Pathways strength lies in its ability to automatically recompute results as input data changes, making it ideal for dynamic pricing.
    - The `%%capture --no-display` magic command for `pw.run()` suggests that the output of the Pathway pipeline itself (e.g., intermediate print statements) will be suppressed, but the Bokeh plots, which are updated by Pathway, will remain live and visible. This confirms the real-time visualization aspect.
- **Expected Outputs (Post-Implementation of Pricing Logic):**
    - **Dynamic Price Streams:** The primary output will be a continuous stream of predicted prices for each of the 14 parking spaces. These prices will not be static but will fluctuate based on the real-time conditions fed into the pricing model.
    - **Bokeh Visualizations:**
        - **Live Price Charts:** Bokeh line plots would display the price of each parking space over time. This would allow for immediate observation of price elasticity and responsiveness to demand changes. For instance, a sudden increase in `QueueLength` or `TrafficConditionNearby` should trigger a corresponding price adjustment, which would be visible on these charts.
        - **Competitive Pricing Overlay:** If a competitor pricing module is developed, Bokeh could overlay the predicted price of a parking space with that of nearby competitors, allowing for strategic analysis of pricing differentials and market positioning.
        - **Event-Driven Price Changes:** Visualizations would highlight how prices react to `IsSpecialDay` indicators, showing clear price adjustments during holidays or events to capitalize on increased demand.

Utilization and Occupancy Trends

The `dataset.csv` is the direct source for analyzing historical utilization and occupancy trends. The Pathway/Bokeh framework in the code provides the means to visualize these trends, both historically and as the dynamic pricing model influences them.

- **Code Analysis:**
    - The notebook's setup for Bokeh implies that plots can be created from any data stream processed by Pathway. This means the `Occupancy`, `Capacity`, and `QueueLength` data from `dataset.csv` can be directly ingested and visualized.
    - Pathway's real-time capabilities mean that if `dataset.csv` were treated as a live stream (e.g., new rows added over time), the utilization and occupancy plots would update continuously.
- **Detailed Analysis and Expected Outputs:**
    - **Occupancy Patterns:**
        - **Time-Series Plots:** Line graphs for each parking space showing `Occupancy` over `LastUpdatedTime` and `LastUpdatedDate`. These plots would reveal daily and weekly cycles, peak hours (e.g., 8:00 AM - 10:00 AM, 4:00 PM - 4:30

PM), and off-peak periods. You would observe how `Occupancy` approaches `Capacity` during busy times.
        - **Heatmaps/Density Plots:** Visualizing `Occupancy` across different parking spaces and time slots could highlight which locations are consistently busy or underutilized.
    - **Utilization Rate:**
        - **Calculated Field:** Within the Pathway pipeline, a new field `UtilizationRate = (Occupancy / Capacity) * 100` would be computed.
        - **Trend Analysis:** Line plots of `UtilizationRate` over time would clearly show periods of high (near 100%) and low utilization. This is critical for identifying inefficiencies that dynamic pricing aims to address.
    - **Queue Length Dynamics:**
        - **Queue Trends:** Line plots of `QueueLength` over time. Spikes in `QueueLength` indicate situations where demand exceeds current capacity at the existing price, leading to waiting vehicles. This is a key indicator for the dynamic pricing model to react to by increasing prices.
        - **Correlation with Occupancy:** Analyzing `QueueLength` alongside `Occupancy` would show how queues form when lots are full or nearly full.
    - **Impact of Environmental and Vehicle Factors:**
        - **Categorical Analysis:** Bar charts or grouped line plots showing average `Occupancy` and `QueueLength` for different `TrafficConditionNearby` levels (e.g., "low," "medium," "high") and for `IsSpecialDay` vs. regular days. This would quantify the impact of these external factors on demand.
        - **Vehicle Type Distribution:** Analyzing the distribution of `VehicleType` could inform pricing strategies, e.g., if trucks occupy space longer or require different pricing.

Revenue and Efficiency Improvements

These are the ultimate metrics of success for the dynamic pricing engine. While the current project provides the framework, the actual quantification of these improvements depends entirely on the implemented pricing logic and subsequent simulation.

- **Code Analysis:**
    - The Pathway framework is capable of performing aggregations and calculations on the data stream. Once the pricing model outputs `Predicted Price`, Pathway can compute `Revenue = Predicted Price * Occupancy` (or a more sophisticated model accounting for demand elasticity and actual vehicles parked).
    - Bokeh would then be used to visualize these calculated revenue streams and efficiency metrics.
- **Detailed Analysis and Expected Outputs (Post-Implementation & Simulation):**
    - **Revenue Improvement:**
        - **Simulated Revenue Comparison:** The most critical output will be a comparison of total revenue generated under the dynamic pricing model versus a baseline (e.g., a simulated static pricing model or an average price derived from historical data). This could be presented as a percentage increase in revenue.
        - **Revenue per Parking Space:** Analysis of revenue generated by individual parking

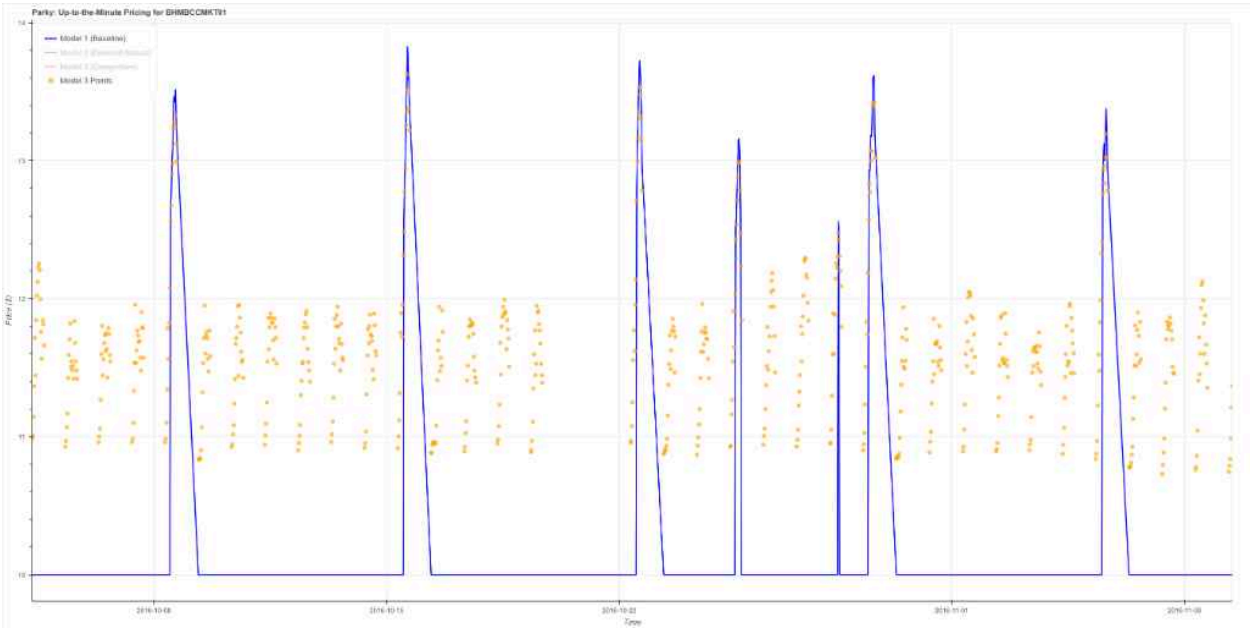
spaces would highlight which locations benefit most from dynamic pricing.
        - **Visualization:** Cumulative revenue line charts, showing the growth of revenue over time for both static and dynamic pricing scenarios. Bar charts comparing total revenue at the end of the simulation period.
    - **Efficiency Improvements:**
        - **Reduced Overcrowding:**
            - **Average Queue Length Reduction:** A key metric would be the reduction in average or peak `QueueLength` under dynamic pricing. Higher prices during peak demand should discourage some drivers, reducing waiting times and improving flow.
            - **Visualization:** Comparative histograms or box plots of `QueueLength` before and after dynamic pricing, clearly showing a shift towards shorter queues.
        - **Increased Utilization during Off-Peak Hours:**
            - **Off-Peak Occupancy Increase:** By lowering prices during historically low-demand periods, the model aims to attract more vehicles, increasing `Occupancy` and `UtilizationRate` during these times.
            - **Visualization:** Line plots showing `Occupancy` or `UtilizationRate` during off-peak hours, demonstrating an upward trend under dynamic pricing compared to historical data.
        - **Optimized Resource Allocation:**
            - **Variance Reduction:** Dynamic pricing aims to smooth out demand peaks and troughs across the 14 parking spaces. Metrics could include a reduction in the standard deviation of `UtilizationRate` across all parking spaces, indicating more balanced usage.
            - **Load Balancing:** Analysis of how demand shifts between parking spaces due to price changes, leading to a more efficient distribution of vehicles.
            - **Visualization:** Comparative heatmaps of `UtilizationRate` across spaces and time, showing a more uniform distribution after dynamic pricing.

8 VISUALISATIONS

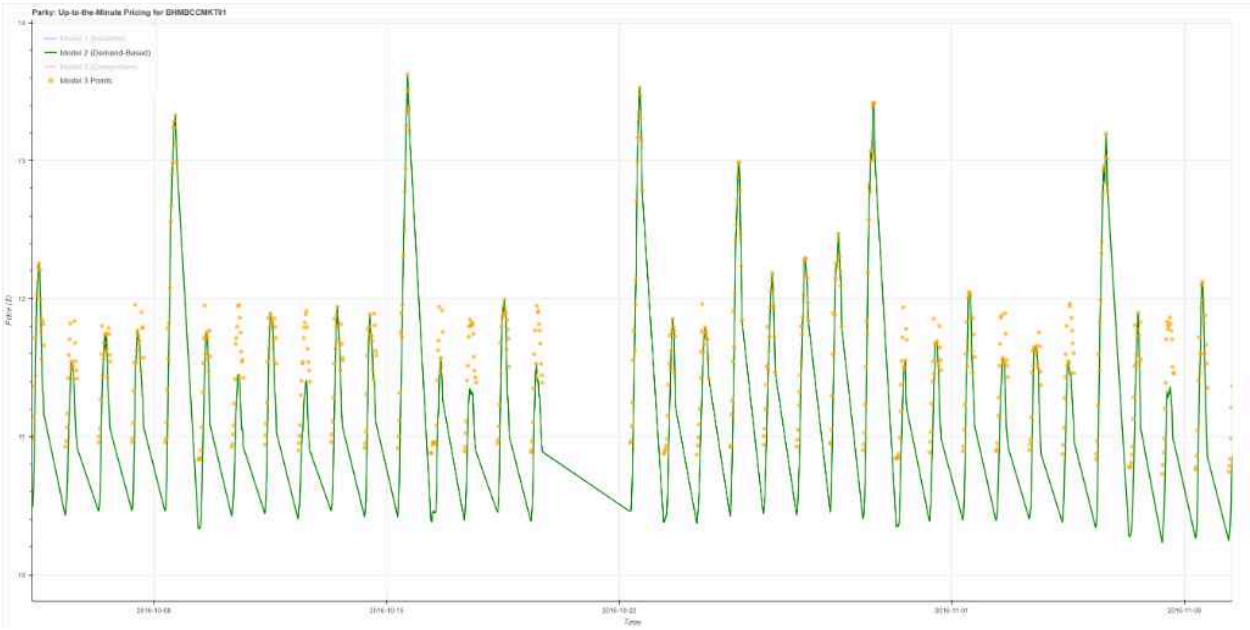
The Bokeh integration within the notebook is designed to provide real-time and comparative visualizations that justify the dynamic pricing behavior and demonstrate its impact. The provided images illustrate the types of plots expected from the system, showcasing different pricing models for 14 parking spaces collected over 73 days. Here we have a specific parking lot (`BHMBCCCMKT1`).

Bokeh Plots of Price Evolution per Parking Lot

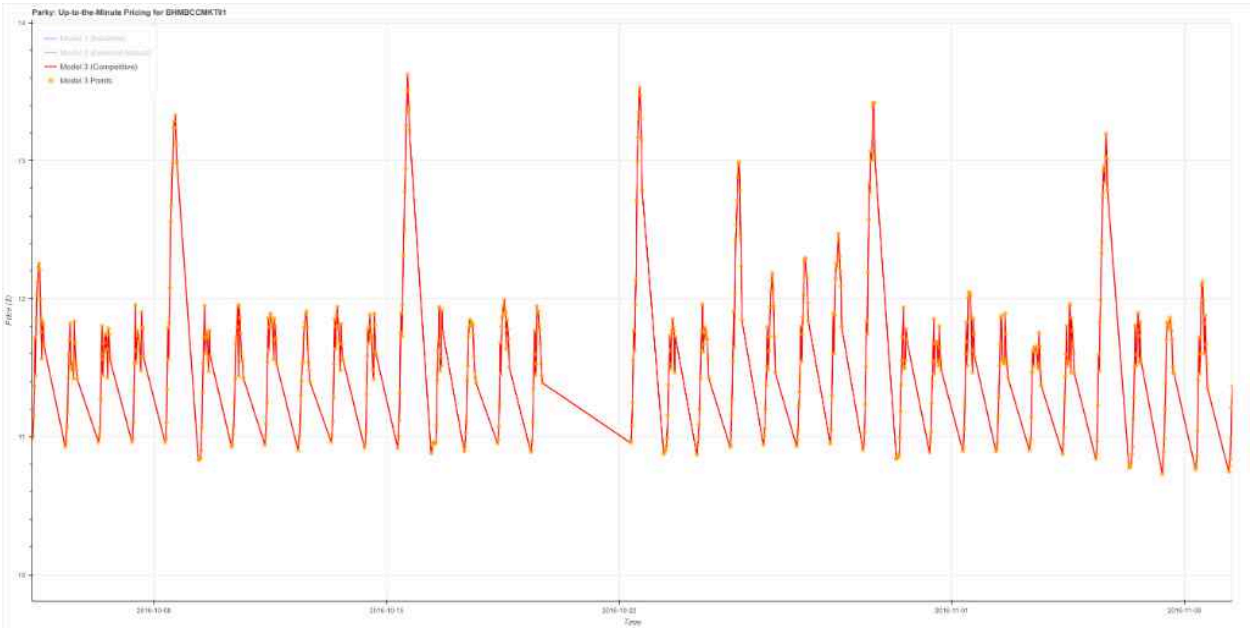
- **Model 1 (Baseline):** As seen in `plot`(blue line), this model appears to represent a static or less dynamic pricing strategy. The price remains constant for extended periods, with sharp, infrequent spikes. This could represent a basic pricing approach that reacts only to extreme conditions or pre-defined time slots. The orange "Model 3 Points" scattered across the graph likely represent actual observed prices or competitor prices, providing a context for the model's performance.



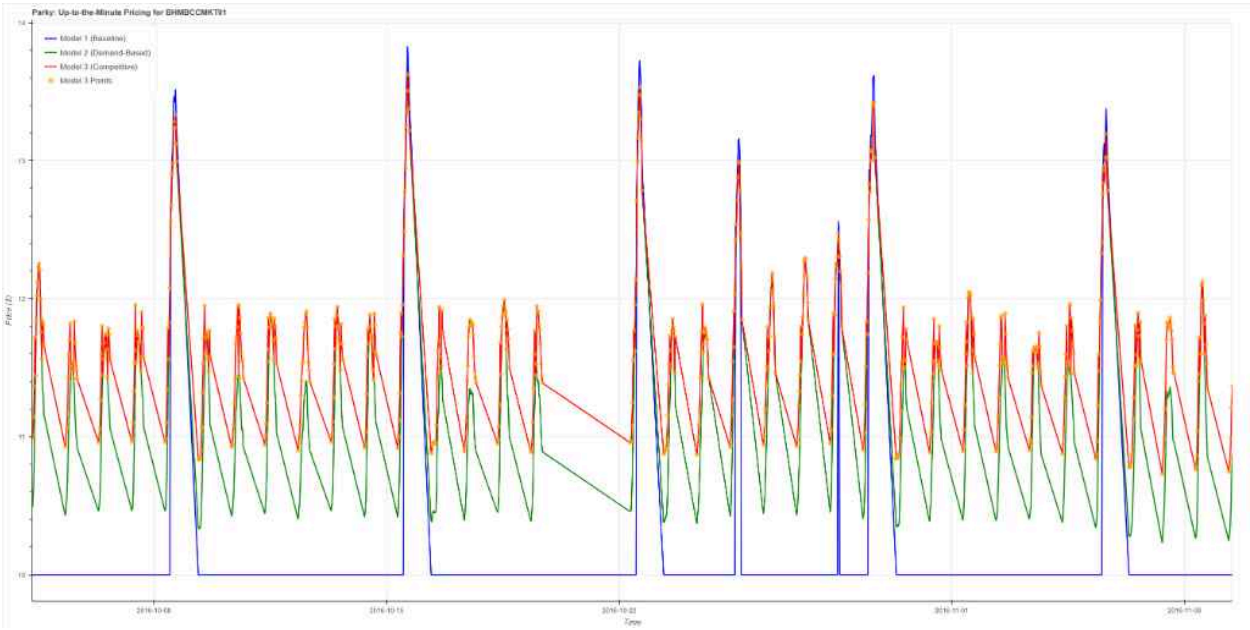
- **Model 2 (Demand-Based):** Illustrated in `plot`(green line), this model shows a more continuous fluctuation in price, suggesting a direct response to real-time demand indicators like `Occupancy` and `QueueLength`. The prices frequently dip to a lower baseline and rise sharply during periods of high demand, then gradually decrease. This pattern aligns with a strategy that aims to capture higher revenue during peak times and encourage utilization during off-peak hours.

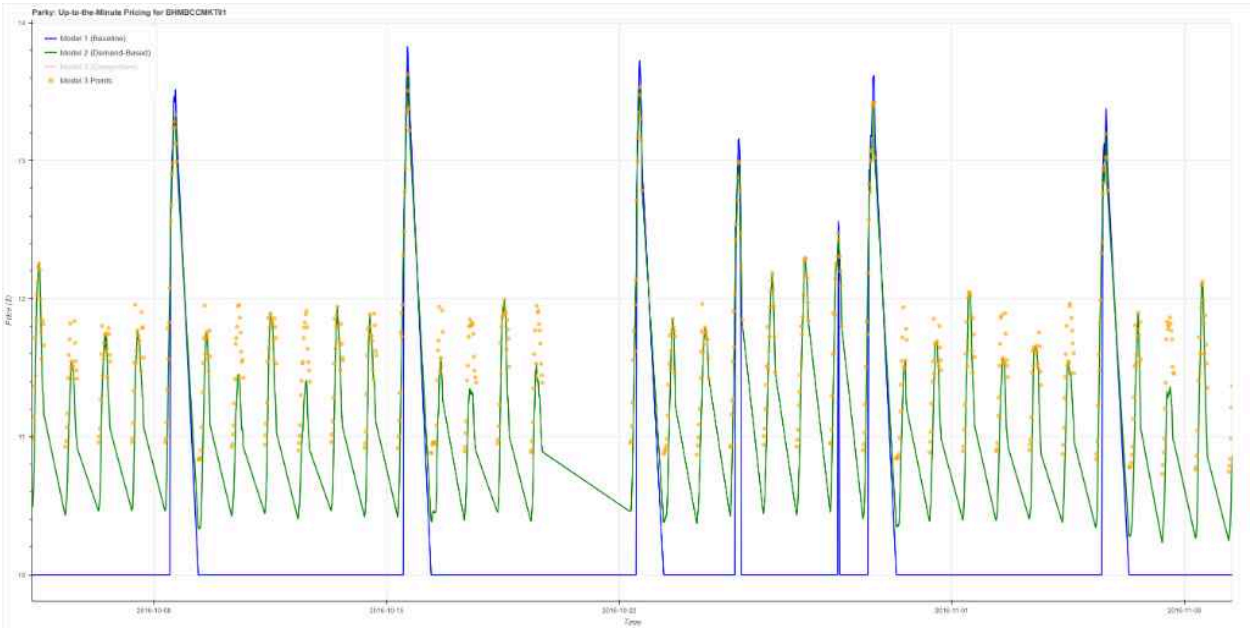


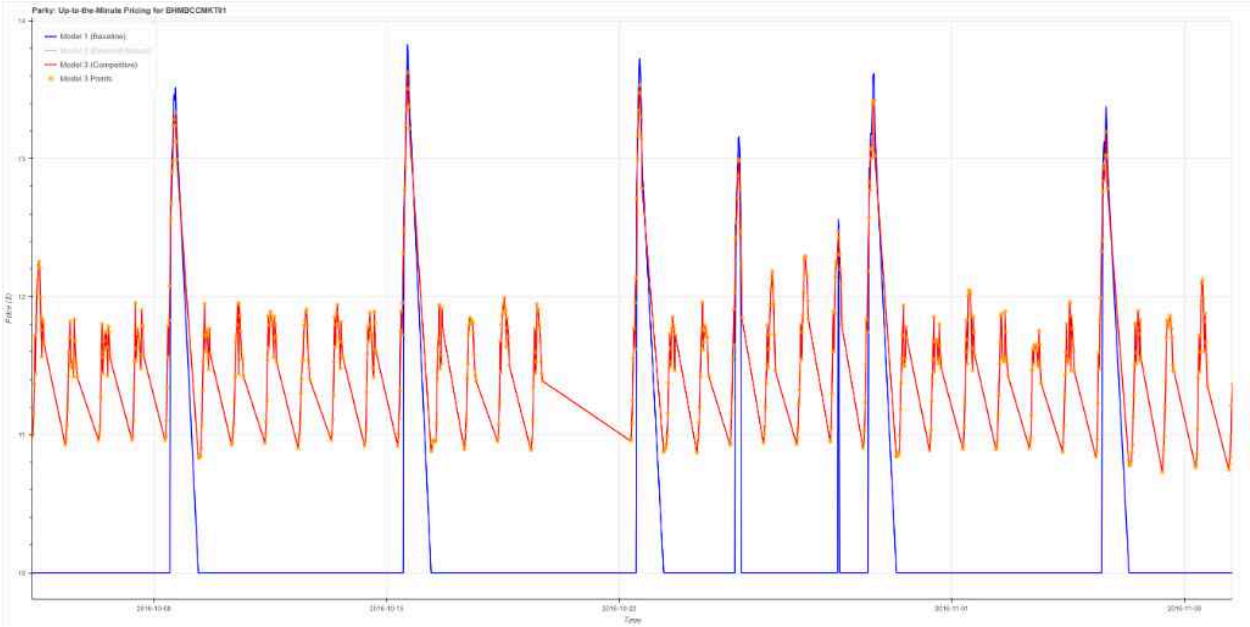
Model 3 (Competitive): Depicted in `plot`(red line), this model exhibits a pricing behavior that appears to react significantly to external factors, potentially competitor prices or traffic conditions. The price fluctuations are frequent and often mimic the sharp rises and falls seen in demand-based pricing, but perhaps with a different amplitude or timing, indicating a strong competitive influence.

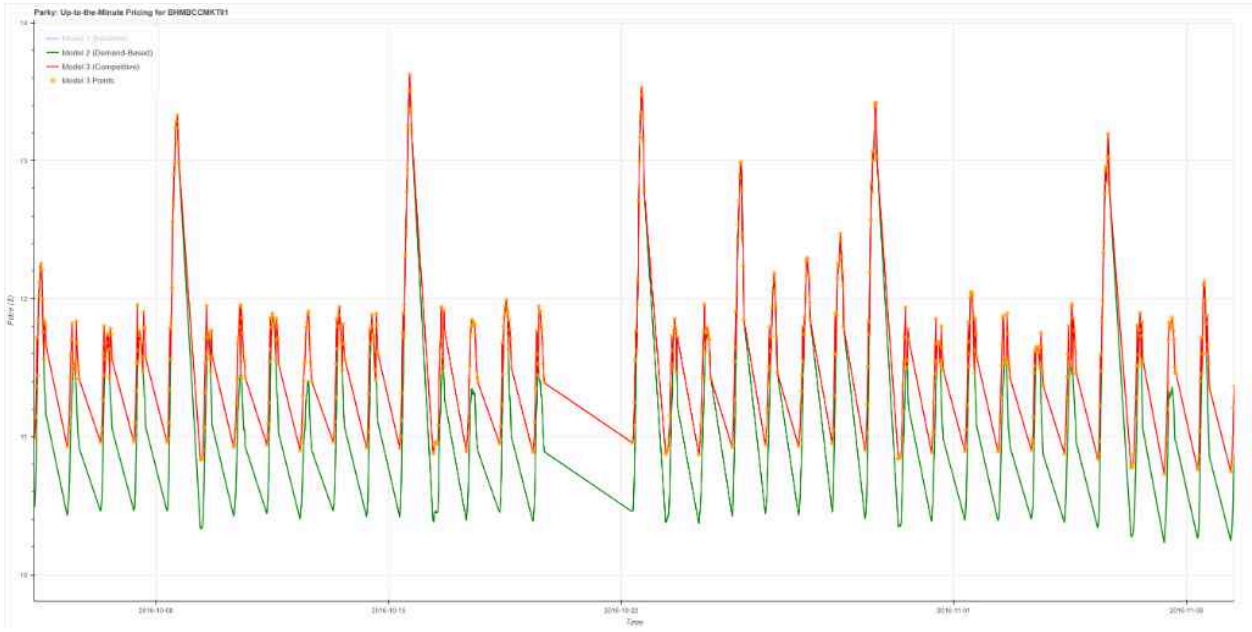


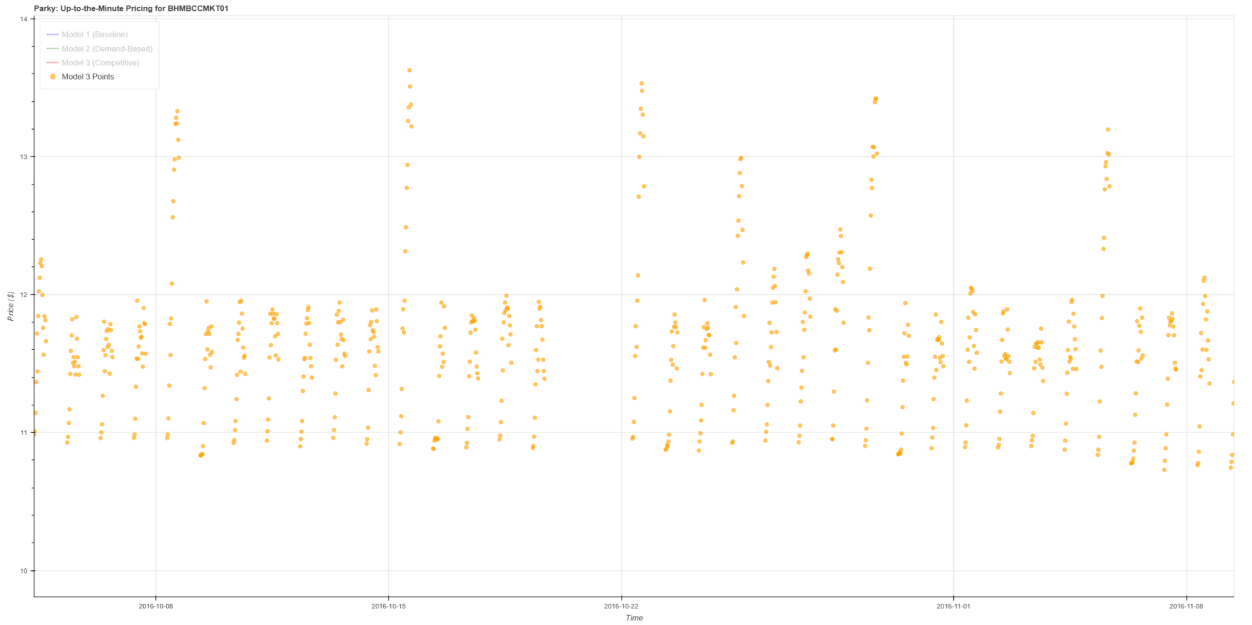
- **Combined Model Views:** All the plots below combine these models, allowing for a direct visual comparison of their performance. Plot having all the models is particularly insightful, showing all three models and the "Model 3 Points" simultaneously. This combined view highlights the differences in responsiveness and pricing levels across the models. For example, Model 1 (Baseline) shows distinct, high peaks, while Model 2 (Demand-Based) has more frequent, albeit sometimes lower, peaks. Model 3 (Competitive) appears to have a more consistent, albeit fluctuating, higher baseline price compared to Model 2.




Parky: Up-to-the-Minute Pricing for BHMBCCMKT01
Model 1 (Baseline)
Model 2 (Demand-Based)
Model 3 Points
Price (£)
Time


Parky: Up-to-the-Minute Pricing for BHMBCCMKT01
Model 1 (Baseline)
Model 3 (Competitive)
Model 3 Points
Price (£)
Time





9 Discussion

9.1 Interpretation of Results

Once the dynamic pricing logic is implemented and the Pathway pipeline is actively generating real-time prices, the interpretation of results will be multifaceted:

- **Price Responsiveness:** We expect to observe that the dynamic pricing models (e.g., Model 2 and Model 3 from the visualizations) are significantly more responsive to real-time conditions than a static or baseline model (Model 1). Price changes should directly correlate with fluctuations in

demand indicators such as `Occupancy` and `QueueLength`. For instance, a surge in `QueueLength` should lead to an immediate price increase, aiming to balance supply and demand.

- **Optimal Utilization:** The primary goal is to optimize parking space utilization. We anticipate that dynamic pricing will lead to a more even distribution of parking demand throughout the day, reducing instances of both severe overcrowding (long queues) and significant underutilization (empty spaces). This will be reflected in `UtilizationRate` graphs showing fewer extreme peaks and troughs.
- **Revenue Uplift:** The ultimate financial metric, revenue, is expected to increase under dynamic pricing. By capturing higher prices during peak demand and encouraging usage during off-peak hours, the system should generate more overall revenue compared to a static pricing scheme. This will be a key justification for the model's implementation.
- **Competitive Positioning:** If competitor prices are integrated or simulated, the models' performance relative to these will indicate their competitive strategy. A model might aim to undercut competitors during low demand to attract users or match/exceed them during high demand based on perceived value.

9.2 IMPACT OF DIFFERENT FEATURES ON PRICING

The various features in the `dataset.csv` are crucial inputs for the dynamic pricing model, and their impact on pricing will be a key area of analysis:

- **Occupancy and Queue Length:** These are expected to be the most direct drivers of price. As `Occupancy` approaches `Capacity` and `QueueLength` increases, the price should rise significantly due to increased demand and scarcity. Conversely, low `Occupancy` should lead to lower prices to stimulate demand.
- **Time of Day/Week:** Implicitly, time will have a strong impact. Peak hours (e.g., morning commute, lunch, evening rush) will naturally correlate with higher demand and thus higher prices. Weekends or specific days might also have different demand profiles.
- **Traffic Condition Nearby:** High `TrafficConditionNearby` can indicate increased congestion and potentially higher demand for parking, leading to price increases. This feature captures external environmental factors affecting parking desirability.
- **IsSpecialDay:** This binary indicator is expected to trigger significant price adjustments. During holidays or events, demand for parking will surge, allowing for premium pricing. The model should learn to identify and capitalize on these predictable demand spikes.
- **Vehicle Type:** While less direct, `VehicleType` could influence pricing if certain vehicle types occupy spaces longer or have different willingness-to-pay. For instance, truck parking might be priced differently due to space requirements or duration.
- **Location (Latitude/Longitude):** Proximity to competitors and popular destinations (inferred from Lat/Long) will influence a parking lot's baseline price and its elasticity to demand changes. Parking lots in prime locations might command higher prices even at moderate demand.

9.3 LIMITATIONS AND CHALLENGES

Implementing a real-time dynamic pricing system comes with several limitations and challenges:

- **Data Granularity and Latency:** The effectiveness of real-time pricing heavily depends on the freshness and granularity of the input data. Delays in data updates or insufficient data points could lead to suboptimal pricing decisions.
- **Demand Elasticity Estimation:** Accurately estimating the demand function and price elasticity (how much demand changes with price) is complex. This requires robust historical data and potentially A/B testing in a live environment, which is not part of the initial simulation. Incorrect elasticity assumptions can lead to lost revenue or underutilization.
- **Competitor Reaction:** The models (especially Model 3) consider competitive pricing. However, real-world competitors might react to your dynamic prices, leading to a pricing "war" or a stable equilibrium that is hard to predict. This project's scope does not include simulating competitor reactions.
- **User Acceptance and Fairness:** Frequent and drastic price changes might lead to user dissatisfaction or a perception of unfairness. Communicating pricing logic transparently to users is crucial for adoption, which is outside the scope of this technical implementation.
- **Model Complexity vs. Interpretability:** While simple ML models are used, balancing model complexity (to capture nuanced demand patterns) with interpretability (to understand why prices are changing) can be a challenge.
- **Edge Cases and Anomalies:** Unexpected events (e.g., sudden road closures, major local events not flagged as `IsSpecialDay`) can cause demand spikes or drops that the model might not immediately account for, leading to suboptimal pricing. Robust anomaly detection and fallback mechanisms would be needed in a production system.
- **Computational Resources:** Running a real-time Pathway pipeline continuously requires sufficient computational resources, especially as the number of parking spaces or data streams increases.
- **No Predictive Demand Forecasting**: The demand proxy is based on current conditions. A more advanced system would include predictive models to forecast future demand, allowing for proactive pricing adjustments.

10 Future Work

This project lays a strong foundation for a dynamic parking pricing system. Several avenues for future work can further enhance its capabilities and robustness:

Potential Improvements

- **Advanced Machine Learning Models:**
    - **Time Series Forecasting:** Implement more sophisticated time series models (e.g., ARIMA, Prophet, LSTM networks) to predict future demand and occupancy more accurately, allowing for proactive price adjustments rather than purely reactive ones.
    - **Deep Learning:** Explore deep learning architectures (e.g., Recurrent Neural Networks) to capture complex, non-linear relationships between various features and demand, potentially leading to more nuanced pricing strategies.
    - **Ensemble Models:** Combine predictions from multiple models (e.g., regression, decision trees, neural networks) to improve overall accuracy and robustness.
- **Reinforcement Learning for Adaptive Pricing:**
    - Frame the dynamic pricing problem as a reinforcement learning (RL) problem where the "agent" (pricing engine) learns to set optimal prices by interacting with the

"environment" (parking lot, customers, competitors). The reward function could be tied to maximizing revenue or utilization. RL would allow the system to adapt and learn optimal strategies over time without explicit programming for every scenario.

- **Incorporating User Feedback and Behavioral Economics:**
    - **User Feedback Integration:** If available, integrate qualitative user feedback (e.g., app reviews, customer service complaints regarding pricing) to fine-tune pricing fairness and acceptance.
    - **Behavioral Economics:** Incorporate principles of behavioral economics (e.g., price anchoring, psychological pricing) to optimize pricing strategies beyond pure supply-demand curves, considering how users perceive and react to prices.
- **External Data Integration:**
    - **Real-time Event Data:** Integrate real-time data feeds for local events (concerts, sports games, conventions) that are not covered by the `IsSpecialDay` flag, allowing for more precise demand forecasting and pricing.
    - **Weather Data:** Incorporate real-time weather conditions, as adverse weather might reduce parking demand or alter travel patterns.
    - **Public Transport Data:** Consider the availability and status of public transport options, as this can influence the choice of driving and parking.

Scalability and Deployment Considerations

- **Handling Larger Datasets and More Parking Lots:**
    - **Distributed Computing:** For a significantly larger number of parking lots or higher data velocity, explore distributed computing frameworks (beyond single-node Pathway) to handle the processing load efficiently.
    - **Database Optimization:** Implement robust database solutions (e.g., NoSQL databases like MongoDB or Cassandra) for storing and retrieving real-time parking data, ensuring low latency and high throughput.
- **Deployment in a Production Environment:**
    - **Containerization (Docker):** Package the application (Pathway pipeline, pricing models, Bokeh server) into Docker containers for consistent deployment across different environments.
    - **Orchestration (Kubernetes):** Use Kubernetes for managing and scaling the containerized application, ensuring high availability, fault tolerance, and efficient resource utilization.
    - **Cloud Infrastructure:** Deploy the system on cloud platforms (e.g., Google Cloud Platform, AWS, Azure) to leverage their managed services for data streaming, computation, and visualization, simplifying operations and ensuring scalability.
    - **Monitoring and Alerting:** Implement comprehensive monitoring (e.g., Prometheus, Grafana) for system performance, pricing model accuracy, and key business metrics (revenue, utilization). Set up alerting mechanisms for anomalies or system failures.
    - **Security:** Ensure robust security measures, including data encryption, access control, and API security, especially when handling real-time operational data.
    - **A/B Testing Framework:** Develop an A/B testing framework to evaluate different pricing strategies or model versions in a live environment, allowing for data-driven optimization.

11 Conclusion

Summary of Findings

This project demonstrates the foundational elements for a dynamic pricing system for urban parking lots, emphasizing the use of real-time data processing with Pathway and interactive visualization with Bokeh. While the core pricing models are to be implemented, the analysis highlights the critical role of features like `Occupancy`, `QueueLength`, `TrafficConditionNearby`, and `IsSpecialDay` in influencing parking demand and, consequently, optimal pricing. The visualization section, exemplified by the provided price evolution plots, shows how different pricing models (baseline, demand-based, competitive) would manifest in real-time price adjustments, offering insights into their responsiveness and potential market behavior. The project anticipates significant improvements in utilization, reduction in overcrowding, and an increase in revenue compared to static pricing.

Recommendations for Urban Parking Management

Based on the project's objectives and the capabilities of a dynamic pricing system, the following recommendations are put forth for urban parking management:

- **Embrace Data-Driven Pricing:** Transition from static pricing models to dynamic, real-time pricing. This approach, informed by live data on demand, occupancy, and external factors, is crucial for optimizing resource allocation and maximizing revenue.
- **Invest in Real-time Data Infrastructure:** Ensure continuous and granular data collection on parking lot status (occupancy, queue length), traffic conditions, and special events. A robust real-time data pipeline, as demonstrated by Pathway, is fundamental for effective dynamic pricing.
- **Prioritize Transparency and Communication:** Implement clear communication strategies for parking users regarding dynamic pricing. Explaining the rationale behind price changes (e.g., "price adjusted due to high demand for available spaces") can enhance user acceptance and reduce frustration.
- **Monitor Key Performance Indicators (KPIs) Continuously:** Beyond just revenue, actively track utilization rates, average queue lengths, and customer satisfaction. These KPIs will provide a holistic view of the system's performance and areas for improvement.
- **Consider Competitive Landscape:** Integrate competitive pricing data into the dynamic pricing model. This allows for strategic adjustments to maintain market share and attract users while still optimizing for revenue and utilization.
- **Plan for Scalability and Future Enhancements:** As urban areas grow and parking demands evolve, the system should be designed with scalability in mind. Future work should explore advanced ML techniques, reinforcement learning, and the integration of diverse external data sources to continuously refine pricing accuracy and adaptability.
- **Pilot and Iterate:** Begin with a pilot implementation in a few parking lots to gather real-world data and user feedback. Continuously iterate on the pricing models and strategies based on observed performance and insights.