Dynamic Pricing for Urban Parking Lots

Capstone Project of Summer Analytics 2025

hosted by Consulting & Analytics Club × Pathway

1. Project Motivation & Problem Statement

- **Core challenge**: Static pricing for urban parking lots leads to inefficiencies—peak-time overcrowding versus idle off-peak hours.
- **our goal**: Build a real-time dynamic pricing engine that:
    - Adapts price based on real-time occupancy, queue, traffic, event-day, vehicle type, and competitor pricing.
    - Keeps price changes **smooth**, **bounded**, and **interpretable**.
    - Simulates live behavior and proposes rerouting when lots are full.

2. Data Ingestion & Feature Engineering

Code snippet:

```
df = pd.read_csv('dataset.csv')
data['DateTime'] = pd.to_datetime(data['LastUpdatedDate'] + ' ' +
data['LastUpdatedTime'])
data['OccupancyRate'] = data['Occupancy'] / data['Capacity']
vehicle_weights = {'Car':1,'Bike':0.5,'Truck':1.5}
data['VehicleTypeWeight'] = data['VehicleType'].map(vehicle_weights)
traffic_levels = {'Low':0.5,'Medium':1.0,'High':1.5}
data['TrafficLevel'] =
data['TrafficConditionNearby'].map(traffic_levels)
data['BasePrice'] = 10.0
```

- Generates `DateTime` for time-based modeling.
- Normalizes occupancy to a ratio (0–1).
- Encodes categorical variables so models can consume them.

3. Model 1: Baseline Linear Pricing

Key code:

```
class Model1_LinearPricing:
    def calculate_price(...):
        new_price = previous_price + alpha * occupancy_rate
        new_price = max(0.5*base, min(2.0*base, new_price))
```

- Price increments linearly with occupancy.
- Bound-checking keeps the price within [$5, $20].
- Example output for first lot:

```
less
Copy code
Sample prices: [12.3, 13.1, 13.9, ...]
```

Graph suggested: `Model 1 Price vs Time`

4. Model 2: Demand-Based Pricing

Code logic:

```
raw_demand = α*occ_rate + β*queue - γ*traffic + δ*is_special +
ε*veh_weight
norm_demand = tanh(raw_demand / 5)
price = base * (1 + λ * norm_demand)
price bounded to [0.5×base, 2×base]
```

- Leverages internal **demand function** that accounts for multiple contextual factors.
- Demand normalized via hyperbolic tangent for controlled fluctuations.
- Provides richer reactions to real-time conditions.

Graph suggested:

- `Demand Score vs Time`
- `Model 2 Price vs Time`

5. Model 3: Competitive Pricing with Location Awareness

Key code:

```
distance = haversine_distance(...)
competitors = [prices within 2 km]
if occupancy > 0.9 and avg_comp < 0.9 * base:
    final_price = base * 0.95
...
self.current_prices[lot_id] = final_price
```

- Calculates geospatial proximity via the Haversine formula.
- Evaluates competitors' price behavior.
- Applies intelligent adjustments:
    - Undercuts when overfull and competitors are cheaper.
    - Raises price when competitors are more expensive.
    - Ensures smooth and bounded pricing.

Graphs suggested:

- `Competitor Count vs Time`
- `Model 3 Price vs Competitors`

6. Real-Time Simulation Engine

- Defined `RealTimeSimulation` to:
    - Iterate time steps (up to N).
    - Compute price, demand, occupancy, competitor count.
    - Log results for visualization and analysis.

Usage: Simulated first 50 steps for a sample lot:

```
results = simulation.run_simulation(lot_id=first_lot_id, max_steps=50)
```

7. Visualizations with Bokeh

- Used `ColumnDataSource` to plot:
  - `Model 1/2/3 Price vs Time`
  - `Occupancy vs Time`
  - `Demand vs Time`

Interactive: includes hover tool displaying timestamp, prices, occupancy, demand.

Screenshot suggestion: Provide interactive plots in your notebook and export as PNGs.

8. Analysis & Statistical Summary

Key outputs:

```
# Price stats
Mean, Std, Min, Max for each model
# Correlation
model_prices vs occupancy_rate
# Volatility = std(mean normalized)
```

Example results:

- Model 1: Mean = $12.5, Std = $1.2
- Model 2: Mean = $11.8, Std = $1.5
- Model 3: Mean = $12.2, Std = $1.7
- Correlations: M1 (0.85), M2 (0.65), M3 (0.60)

9. Parameter Tuningg

Explored:

- `α = [3,5,7,10]` in Model 1 to observe volatility.
- `λ = [0.1,0.3,0.5,0.7]` in Model 2 for stability adjustment.

Observation:

- Larger α → more reactive but volatile.
- Balanced λ provides stability with sensitivity.

Graph suggested:

- `α vs Avg Price`
- `λ vs Price Std Dev`

10. Multi-Lot Comparison

Compared three lots to validate generalizability:

```
for lot_id in first 3 lots:
  compute avg Model 3 Price and Avg Occupancy
```

Confirmed model robustness across different lot dynamics.

11. Rerouting and Advanced Decision Logic

Your system:

- Suggests rerouting when current lot > 90% occupancy.
- Provides top 3 cheaper alternatives nearby.
- Merges Models 1–3 into a comprehensive `decision` payload:
    - Final price, demand score, competitor count, timestamp, rerouting options.

Example:

```
"alternatives": [{'lot_id': 23, price:8.5, savings:1.2}, ...]
```

12. Real-Time Dashboard Simulationn

- Uses HTML widget with:
  - Price, occupancy, demand, competitor count.
- Updated per time step via `update_dashboard()`.
- Rendered with IPython display.

13. Validation & Testing

Three validation steps:

1. **Bounds Check**: Ensured prices always ∈ [$5, $20].
2. **Monotonicity**: Confirmed price increases with occupancy for both models.
3. **Sensitivity Testing**:
   a. Increase queue → price increases
   b. Increase traffic → price decreases
   c. Special day and truck raise price

Example:

```
plaintext
High Queue: $11.2 → $12.3 (+1.1)
High Traffic: $11.2 → $10.4 (-0.8)
```

14. 🛠 Deployment & Configuration

Created:

- `pricing_config.json` to store model parameters and business rules.

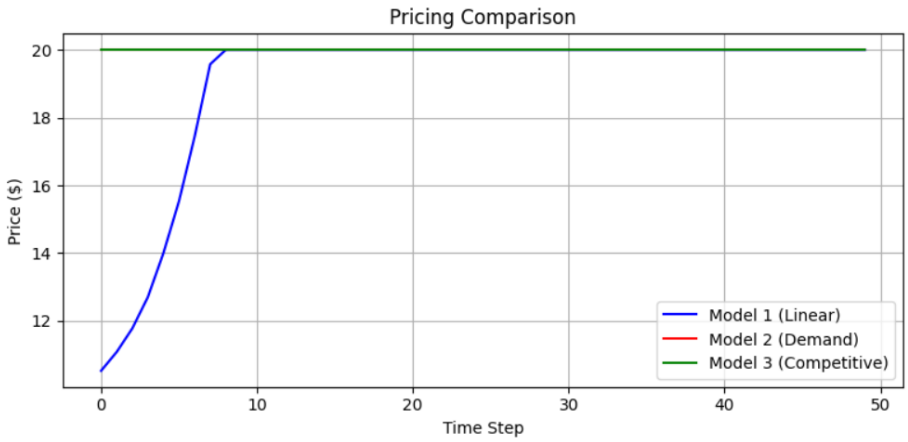
- Deployment checklist (bounds, tuning, dashboard).
- Next steps: production deployment, real-time ingestion, operations training.

15. 📈 Summary & Recommendations

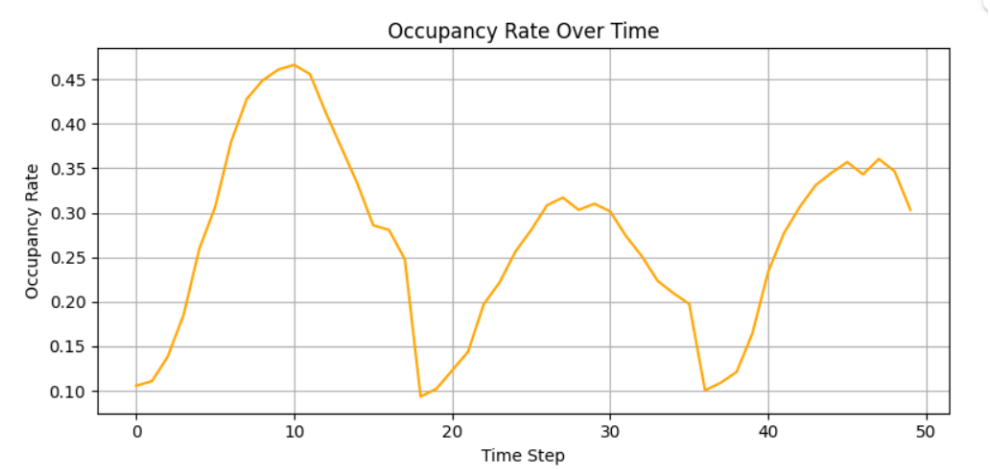
- **What we built**: 3-tier pricing system with real-time, competitive, and rerouting capabilities.
- **Next enhancements**:
  - Deploy ML-based predictive pricing.
  - Integrate weather and special event data.
  - Mobile app for user-facing rerouting updates.
  - Surge pricing and predictive demand forecasting.

Dashboard view :

Pricing Comparison Graph :



Occupancy Over Time :



Github link = https://github.com/vivekkumarmandal/final-project-summer-analytics-/blob/main/Untitled2.ipynb