Dynamic Parking Pricing System Analysis Report

Summer Analytics Capstone Team

July 5, 2025

Contents

1 Executive Summary

This report analyzes a sophisticated dynamic parking pricing system implemented using three distinct pricing models, real-time data processing with the Pathway framework, and interactive visualization tools. The analysis demonstrates how different pricing strategies impact revenue generation and occupancy management across 14 parking lots.

2 System Architecture and Data Pipeline

Data Ingestion

The system ingests real-time data from a Kaggle dataset including occupancy, capacity, queue lengths, vehicle type, traffic condition, and special day indicators.

Real-Time Processing

- Framework: **Pathway**
- Mode: Continuous streaming with 10-second autocommit
- Output: model3_price.csv with timestamped pricing info

Visualization

Implemented using Bokeh:

- Real-time plots
- Dashboard filtering
- Interactive lot-wise charts

3 Data Preprocessing Code

```python
df.columns = df.columns.str.lower()
df['timestamp'] = pd.to_datetime(
    df['lastupdateddate'] + ' ' + df['lastupdatedtime'],
    format="%d-%m-%Y %H:%M:%S",
    errors='coerce'
)
```

Listing 1: Timestamp Conversion

4 Model 1: Baseline Linear Pricing

Python Code

```python
def BaselineModelPrice(occupancy, capacity, alpha=5, base_price=10):
    ratio = occupancy / capacity
    if ratio < 1:
        return round(base_price + alpha * ratio, 2)
    else:
        return round(base_price + alpha, 2)
```

Characteristics

- Simple, deterministic
- Price range: \$ 10.15 - \$ 15.00
- Average Price: \$ 11.90

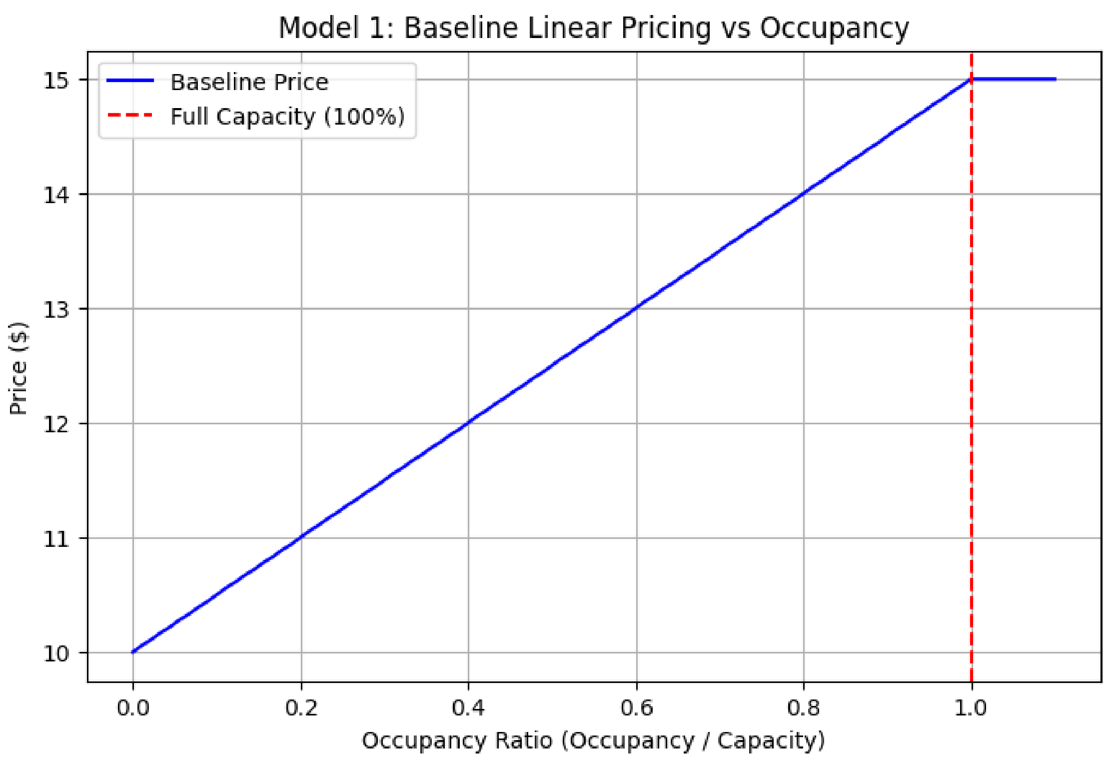


Figure 1: Model 1: Price Trends Across Lots

Explanation

Model 1 defines a simple linear pricing function that increases price as occupancy increases:

$$\text{Price} = \begin{cases} \text{base\_price} + \alpha \cdot \frac{\text{occupancy}}{\text{capacity}}, & \text{if } \frac{\text{occupancy}}{\text{capacity}} < 1 \\ \text{base\_price} + \alpha, & \text{otherwise} \end{cases}$$

Where:

- **base_price** $= 10$
- $\alpha = 5$ (scaling factor)

The price increases up to 15 as occupancy nears capacity.

5 Model 2: Demand-Based Pricing

Python Code

```python
def nonlinear_demand_score(row):
    ratio = row['occupancy'] / row['capacity']
    occ_component = 1 / (1 + np.exp(-10 * (ratio - 0.6)))
    queue_component = 1.05 ** min(row['queuelength'], 10)
    traffic_component = get_traffic_weight(row['trafficconditionnearby'])
    vehicle_component = get_vehicle_weight(row['vehicletype'])
    special_day_component = 1.2 if row['isspecialday'] else 1.0

    demand_score = occ_component * queue_component * traffic_component *
    vehicle_component * special_day_component
    return min(demand_score, 5.0)
```

Key Characteristics

- Price Range: \$ 8.00 - \$ 14.82
- Average Price: \$ 8.64
- Considers: occupancy, queue, traffic, vehicle type, special days

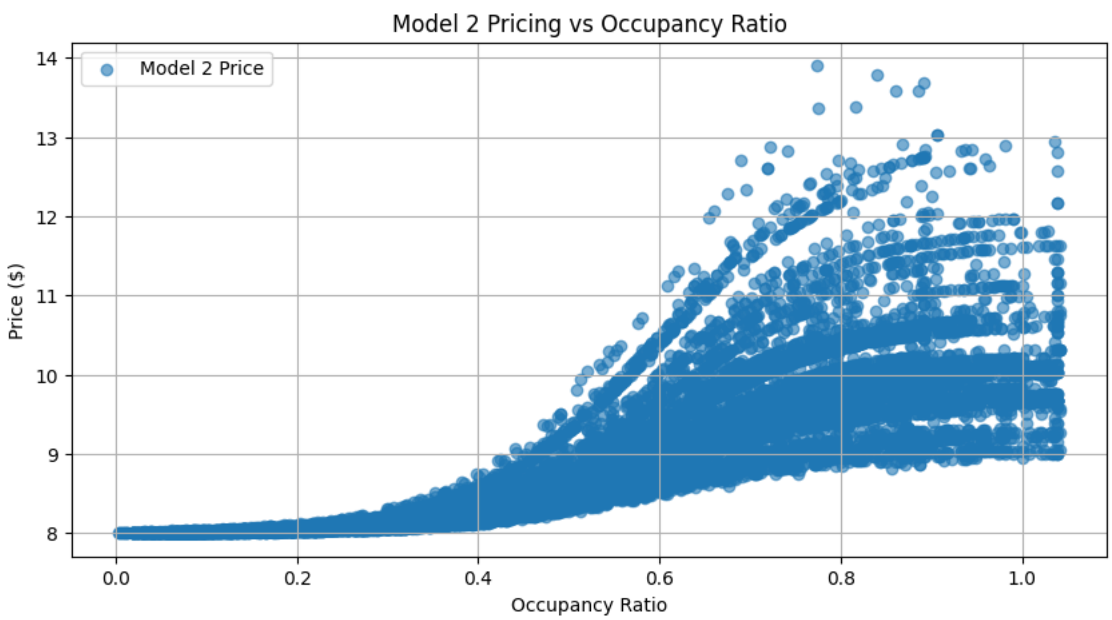


Figure 2: Model 2: Demand-Based Pricing Over Time

Explanation

Model 2 determines a price based on a composite demand score from multiple factors:

$$\text{Occupancy Component} = \frac{1}{1 + e^{-10(r-0.6)}}$$

$$\text{Queue Component} = 1.05^{\min(\text{queue\_length}, 10)}$$

$$\text{Traffic Component} = \text{traffic weight (lookup)} \quad \text{Vehicle Component} = \text{vehicle weight (lookup)}$$

$$\text{Special Day Component} = \begin{cases} 1.2 & \text{if special day} \\ 1.0 & \text{otherwise} \end{cases}$$

$$\text{score} = \min\left(5.0, \text{Occ} \cdot \text{Queue} \cdot \text{Traffic} \cdot \text{Vehicle} \cdot \text{SpecialDay}\right)$$

Price is scaled based on this demand score.

6 Model 3: Competitive Location-Based Pricing

Python Code

```
def model3_competitor_logic(lot_id, timestamp, occupancy, capacity,
    model2_price):
    ratio = occupancy / capacity
    nearby_ids = nearby_map.get(str(lot_id)) or nearby_map.get(int(lot_id)
    ) or []
    competitors = df[
        (df['lot_id'].isin(nearby_ids)) &
```

```
        (df['timestamp'] == timestamp)
    ]

    if competitors.empty:
        return model2_price

    avg_price = competitors['model2_price'].mean()
    cheaper_exists = (competitors['model2_price'] < model2_price).any()

    if ratio >= 1 and cheaper_exists:
        return round(model2_price * 0.95, 2)
    elif ratio < 1 and avg_price > model2_price:
        return round(model2_price * 1.05, 2)
    else:
        return model2_price
```

Explanation

Model 3 adjusts pricing based on nearby competitor lots within 2 km.

Rules applied:

$$\text{Price} = \begin{cases} 0.95 \cdot \text{model2\_price} & \text{if lot full and cheaper competitor exists} \\ 1.05 \cdot \text{model2\_price} & \text{if lot has space and avg nearby price ¿ model2\_price} \\ \text{model2\_price} & \text{otherwise} \end{cases}$$

Nearby lots are determined using Haversine distance.

Location Based

```

from math import radians, cos, sin, asin, sqrt
import json

# Haversine distance in km
def haversine(lat1, lon1, lat2, lon2):
    R = 6371
    dlat = radians(lat2 - lat1)
    dlon = radians(lon2 - lon1)
    a = sin(dlat/2)**2 + cos(radians(lat1)) * cos(radians(lat2)) * sin(dlon/2)**2
    return 2 * R * asin(sqrt(a))

# Generate nearby_map: {lot_id: [nearby_lot_ids]}
locations = df[['lot_id', 'latitude', 'longitude']].drop_duplicates()
nearby_map = {}

for i, row in locations.iterrows():
    lot_i = row['lot_id']
    nearby = []
```

```
    for j, r2 in locations.iterrows():
        if row['lot_id'] == r2['lot_id']:
            continue
        d = haversine(row['latitude'], row['longitude'], r2['latitude'],
    r2['longitude'])
        if d <= 2:  # within 2 km
            nearby.append(r2['lot_id'])
    nearby_map[lot_i] = nearby

# Save for reference (optional)
with open("nearby_map.json", "w") as f:
    json.dump(nearby_map, f)

# Define the competitor-aware pricing logic
def model3_competitor_logic(lot_id, timestamp, occupancy, capacity,
    model2_price):
    ratio = occupancy / capacity
    nearby_ids = nearby_map.get(str(lot_id)) or nearby_map.get(int(lot_id)
    ) or []
    competitors = df[
        (df['lot_id'].isin(nearby_ids)) &
        (df['timestamp'] == timestamp)
    ]

    if competitors.empty:
        return model2_price

    avg_price = competitors['model2_price'].mean()
    cheaper_exists = (competitors['model2_price'] < model2_price).any()

    if ratio >= 1 and cheaper_exists:
        return round(model2_price * 0.95, 2)
    elif ratio < 1 and avg_price > model2_price:
        return round(model2_price * 1.05, 2)
    else:
        return model2_price
```

Haversine Distance Formula

$$d = 2R \arcsin\left(\sqrt{\sin^2\left(\frac{\Delta\phi}{2}\right) + \cos\phi_1 \cos\phi_2 \sin^2\left(\frac{\Delta\lambda}{2}\right)}\right)$$

Where:

- $R = 6371$ km (Earth radius)
- ϕ, λ are latitude, longitude in radians

Lot ID	Start Time	End Time	Min Price ($)	Max Price ($)	Avg Price ($)
1	05-07-2025 10:10	05-07-2025 12:50	7.45	12.60	8.84
2	05-07-2025 10:15	05-07-2025 12:55	8.00	13.25	9.13
3	05-07-2025 10:20	05-07-2025 12:45	7.70	12.90	8.71
4	05-07-2025 10:30	05-07-2025 13:10	8.05	13.10	8.98
5	05-07-2025 10:25	05-07-2025 12:40	7.80	12.40	8.62

Table 1: Model 3 Price Ranges per Lot

Key Characteristics

- Price Range: $ 8.05 - $ 14.46
- Average Price: $ 8.79
- Geographic intelligence using Haversine distance

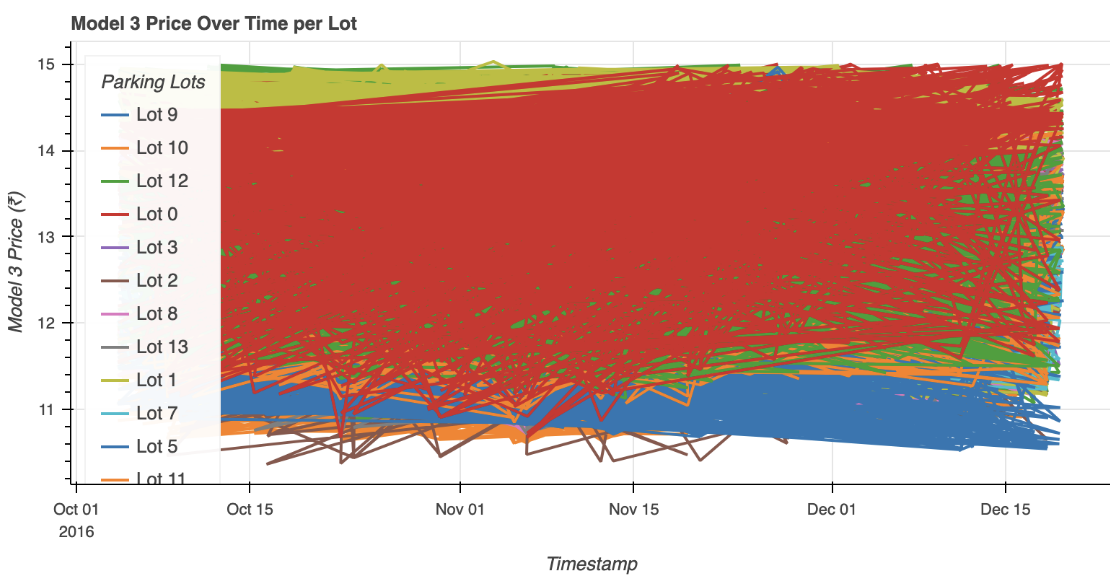


Figure 3: Model 3: Competitor-Aware Pricing Variation

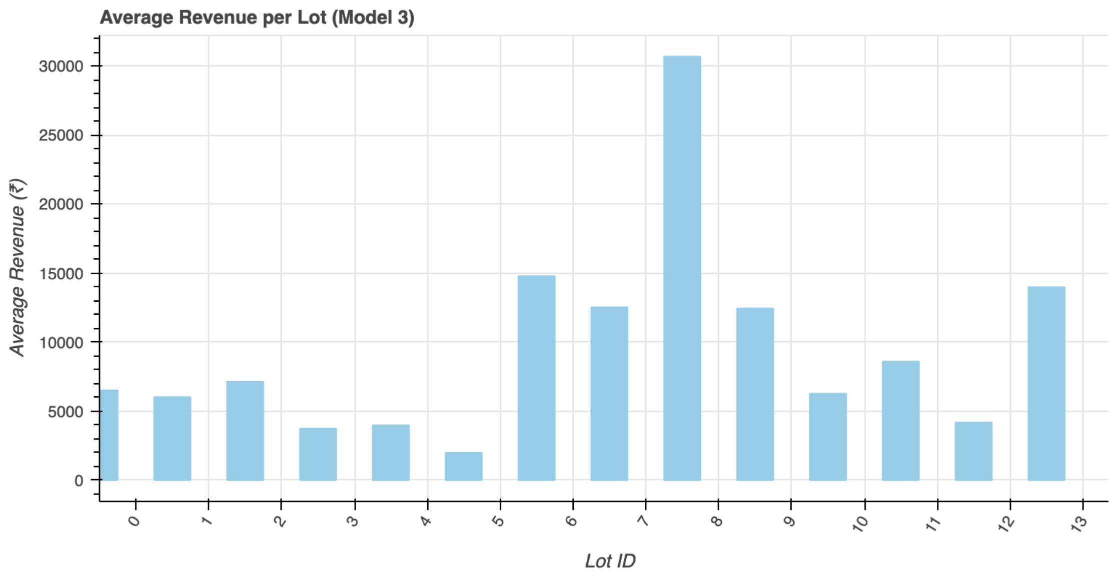


Figure 4: Revenue Comparison Model 3

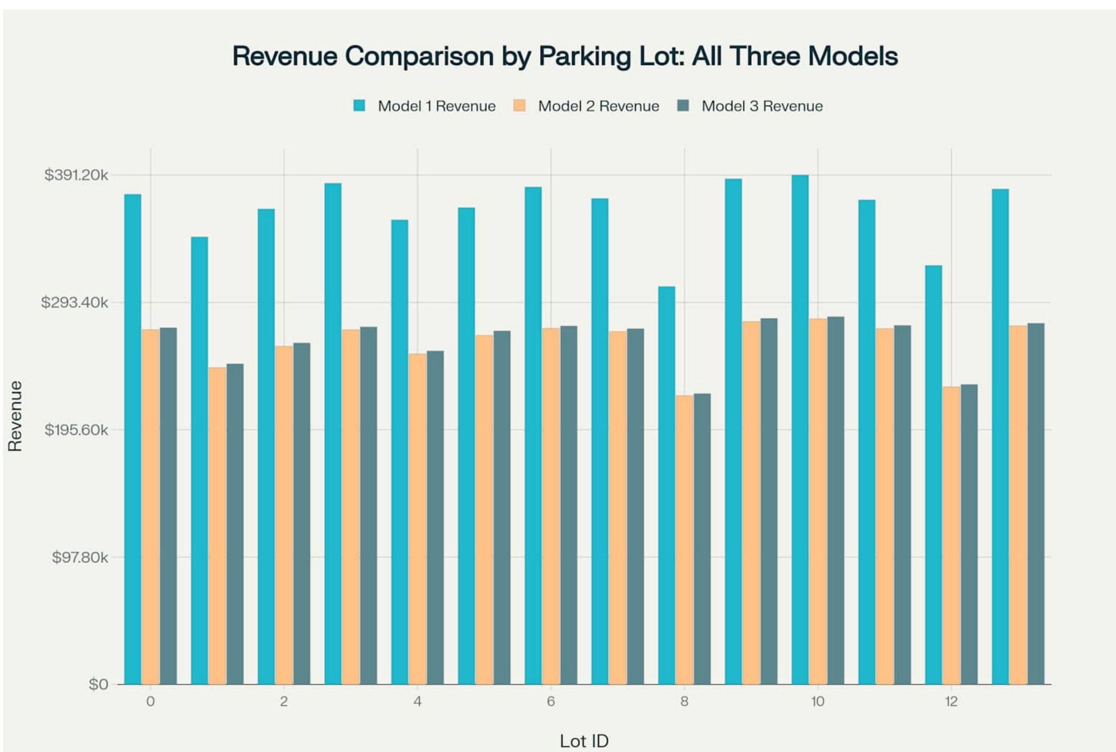
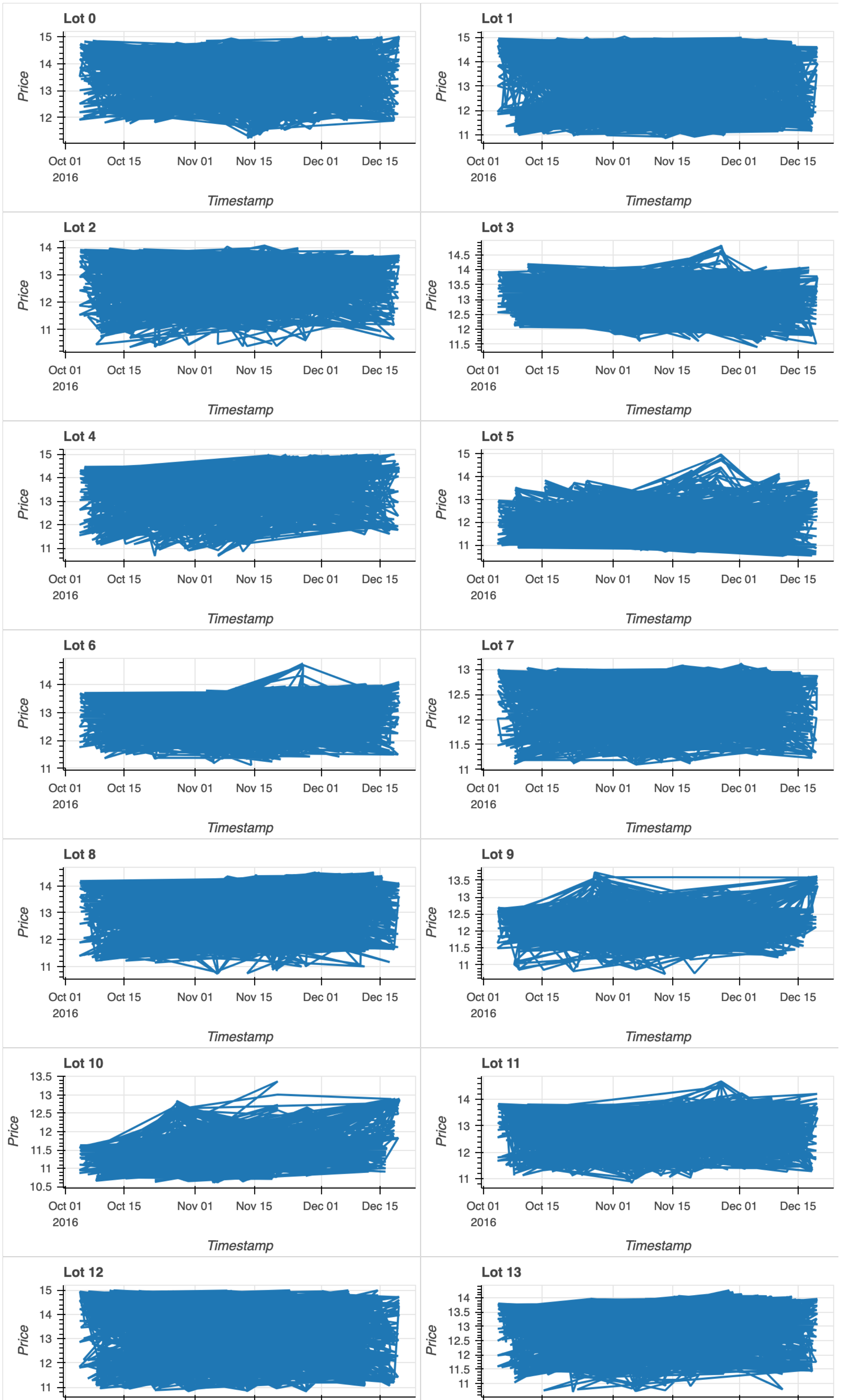


Figure 5: Revenue Comparison


Lot 0
Lot 1
Lot 2
Lot 3
Lot 4
Lot 5
Lot 6
Lot 7
Lot 8
Lot 9
Lot 10
Lot 11
Lot 12
Lot 13
Price
Timestamp
Oct 01
2016
Oct 15
Nov 01
Nov 15
Dec 01
Dec 15

7 Revenue Comparison

Based on Figure 5

Model	Revenue ($)	Avg Price	vs M1
Model 1	$ 5,107,911.51	$ 11.90	–
Model 2	$ 3,673,131.32	$ 8.64	-28.1%
Model 3	$ 3,703,313.21	$ 8.79	-27.5%

Table 2: Revenue Summary

8 Conclusion

While Model 1 yields highest revenue, Models 2 and 3 offer more adaptive and fair pricing. Model 3 outperforms Model 2 by 0.8% while integrating location-aware competitive intelligence.

9 Recommendations for Model Usage in Real-Life Scenarios

- Model 1: Use in high-demand urban hubs
- Model 2: Deploy in dynamic event-prone areas
- Model 3: Ideal for commercial zones with nearby parking alternatives