Capstone Project Report

# Dynamic Pricing for Urban Parking Lots

# Capstone Project - Summer Analytics 2025

# Import required libraries

!pip install pathway bokeh --quiet  
import pandas as pd  
import numpy as np  
from math import radians, cos, sin, asin, sqrt  
from bokeh.plotting import figure, show, output\_notebook  
from bokeh.models import ColumnDataSource  
from bokeh.layouts import column  
import pathway as pw

# Load dataset

df = pd.read\_csv("dataset.csv")  
df['Timestamp'] = pd.to\_datetime(df['LastUpdatedDate'] + ' ' + df['LastUpdatedTime'], format='%d-%m-%Y %H:%M:%S')  
df = df.rename(columns={'SystemCodeNumber': 'parking\_lot\_id'})

# Constants

# --- Model 1: Baseline Linear Pricing ---

# Simulate Model 1

# Visualize Model 1

BASE\_PRICE = 10  
ALPHA = 2.0 # for Model 1  
  
def baseline\_linear\_price(prev\_price, Occupancy, Capacity): # Corrected capitalization  
 return prev\_price + ALPHA \* (Occupancy / Capacity)  
  
price\_history\_1 = []  
prices = {lot\_id: BASE\_PRICE for lot\_id in df['parking\_lot\_id'].unique()}  
for t in sorted(df['Timestamp'].unique()): # Changed 'time' to 'Timestamp'  
 current\_df = df[df['Timestamp'] == t] # Changed 'time' to 'Timestamp'  
 new\_prices = {}  
 for \_, row in current\_df.iterrows():  
 lot\_id = row['parking\_lot\_id']  
 prev\_price = prices.get(lot\_id, BASE\_PRICE) # Use .get with a default  
 price = baseline\_linear\_price(prev\_price, row['Occupancy'], row['Capacity'])  
 new\_prices[lot\_id] = price  
 price\_history\_1.append({'time': t, 'parking\_lot\_id': lot\_id, 'price': price})  
 prices = new\_prices  
  
price\_df1 = pd.DataFrame(price\_history\_1)  
plots = []  
for lot\_id in price\_df1['parking\_lot\_id'].unique():  
 lot\_data = price\_df1[price\_df1['parking\_lot\_id'] == lot\_id]  
 source = ColumnDataSource(lot\_data)  
 p = figure(title=f"Model 1 - Lot {lot\_id} Price Over Time", x\_axis\_type="datetime", width=800, height=250)  
 p.line(x='time', y='price', source=source, line\_width=2)  
 plots.append(p)  
show(column(\*plots))

<div id="c7bfb1b1-f6c1-4e55-afe3-fc506375afbf" data-root-id="73ae384f-b106-4060-b81d-3495d71601ec" style="display: contents;"></div>

# --- Model 2: Demand-Based Pricing ---

# Simulate Model 2

# Map categorical fields - TrafficConditionNearby to numeric level

# Visualize Model 2

def get\_vehicle\_weight(vehicle\_type):  
 return {'car': 1, 'bike': 0.5, 'truck': 1.5}.get(vehicle\_type, 1)  
  
def demand\_function(row):  
 alpha, beta, gamma, delta, epsilon = 1.2, 0.5, 0.3, 2.0, 1.0  
 demand = (  
 alpha \* (row['Occupancy'] / row['Capacity']) +  
 beta \* row['QueueLength'] -  
 gamma \* row['TrafficConditionNearby'] +  
 delta \* row['IsSpecialDay'] +  
 epsilon \* get\_vehicle\_weight(row['VehicleType'])  
 )  
 return demand  
  
def demand\_based\_price(demand):  
 demand = np.clip(demand, 0, 10)  
 price = BASE\_PRICE \* (1 + 0.1 \* (demand / 10))  
 return np.clip(price, 5, 20)  
  
price\_history\_2 = []  
prices = {lot\_id: BASE\_PRICE for lot\_id in df['parking\_lot\_id'].unique()}  
  
traffic\_level\_map = {'low': 1, 'average': 2, 'high': 3}  
  
  
for t in sorted(df['Timestamp'].unique()):  
 current\_df = df[df['Timestamp'] == t].copy() # Create a copy to avoid SettingWithCopyWarning  
 current\_df['TrafficConditionNearby'] = current\_df['TrafficConditionNearby'].map(traffic\_level\_map)  
 # Handle potential missing values after mapping  
 current\_df['TrafficConditionNearby'] = current\_df['TrafficConditionNearby'].fillna(current\_df['TrafficConditionNearby'].median())  
  
  
 new\_prices = {}  
 for \_, row in current\_df.iterrows():  
 lot\_id = row['parking\_lot\_id']  
 prev\_price = prices.get(lot\_id, BASE\_PRICE) # Use .get with a default  
 demand = demand\_function(row)  
 price = demand\_based\_price(demand)  
 new\_prices[lot\_id] = price  
 price\_history\_2.append({'time': t, 'parking\_lot\_id': lot\_id, 'price': price})  
 prices = new\_prices  
  
price\_df2 = pd.DataFrame(price\_history\_2)  
plots = []  
for lot\_id in price\_df2['parking\_lot\_id'].unique():  
 lot\_data = price\_df2[price\_df2['parking\_lot\_id'] == lot\_id]  
 source = ColumnDataSource(lot\_data)  
 p = figure(title=f"Model 2 - Lot {lot\_id} Price Over Time", x\_axis\_type="datetime", width=800, height=250)  
 p.line(x='time', y='price', source=source, line\_width=2)  
 plots.append(p)  
show(column(\*plots))

<div id="fe14ca3f-2831-43a6-a45a-d8cd4a98f2c0" data-root-id="4336b7f5-c03e-43fd-8110-e707ed2627bf" style="display: contents;"></div>

# --- Model 3: Competitive Pricing ---

# Simulate Model 3

# Map categorical fields - TrafficConditionNearby to numeric level

# Visualize Model 3

# Plot for a few parking lots to avoid overwhelming the output

def haversine(lon1, lat1, lon2, lat2):  
 lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])  
 dlon = lon2 - lon1  
 dlat = lat2 - lat1  
 a = sin(dlat/2)\*\*2 + cos(lat1) \* cos(lat2) \* sin(dlon/2)\*\*2  
 c = 2 \* asin(sqrt(a))  
 r = 6371  
 return c \* r  
  
def find\_nearest\_lot(row, current\_prices):  
 dists = df[df['parking\_lot\_id'] != row['parking\_lot\_id']].groupby('parking\_lot\_id').first().reset\_index()  
 dists['distance'] = dists.apply(lambda r: haversine(  
 row['Longitude'], row['Latitude'], r['Longitude'], r['Latitude']), axis=1)  
 nearest = dists.sort\_values('distance').iloc[0]  
 return nearest['parking\_lot\_id'], current\_prices.get(nearest['parking\_lot\_id'], BASE\_PRICE)  
  
def competitive\_price(row, current\_prices):  
 base = demand\_based\_price(demand\_function(row))  
 lot\_id = row['parking\_lot\_id']  
 price = base  
 if row['Occupancy'] >= row['Capacity']:  
 nearest\_id, nearest\_price = find\_nearest\_lot(row, current\_prices)  
 if nearest\_price < base:  
 price = base - 1.0  
 return max(5, min(price, 20))  
  
price\_history\_3 = []  
prices = {lot\_id: BASE\_PRICE for lot\_id in df['parking\_lot\_id'].unique()}  
  
traffic\_level\_map = {'low': 1, 'average': 2, 'high': 3}  
  
  
for t in sorted(df['Timestamp'].unique()):  
 current\_df = df[df['Timestamp'] == t].copy() # Create a copy to avoid SettingWithCopyWarning  
 # Map categorical fields - TrafficConditionNearby to numeric level  
 current\_df['TrafficConditionNearby'] = current\_df['TrafficConditionNearby'].map(traffic\_level\_map)  
 # Handle potential missing values after mapping  
 current\_df['TrafficConditionNearby'] = current\_df['TrafficConditionNearby'].fillna(current\_df['TrafficConditionNearby'].median())  
  
  
 new\_prices = {}  
 for \_, row in current\_df.iterrows():  
 lot\_id = row['parking\_lot\_id']  
 price = competitive\_price(row, prices)  
 new\_prices[lot\_id] = price  
 price\_history\_3.append({'time': t, 'parking\_lot\_id': lot\_id, 'price': price})  
 prices = new\_prices  
  
price\_df3 = pd.DataFrame(price\_history\_3)  
plots = []  
for i, lot\_id in enumerate(price\_df3['parking\_lot\_id'].unique()):  
 if i >= 3: # Limit to 3 plots  
 break  
 lot\_data = price\_df3[price\_df3['parking\_lot\_id'] == lot\_id]  
 source = ColumnDataSource(lot\_data)  
 p = figure(title=f"Model 3 - Lot {lot\_id} Price Over Time", x\_axis\_type="datetime", width=800, height=250)  
 p.line(x='time', y='price', source=source, line\_width=2)  
 plots.append(p)  
show(column(\*plots))

<div id="c4994f43-a6fb-420d-9369-d16a9ec80d75" data-root-id="0f27f0cf-ce41-44f6-9115-96a9b72a9ff7" style="display: contents;"></div>