# Dynamic Pricing for Urban Parking Lots

# Summer Analytics 2025 – Final Submission

import pandas as pd

import numpy as np

from sklearn.preprocessing import MinMaxScaler

from datetime import datetime

from math import radians, sin, cos, sqrt, atan2

# Load and preprocess data

df = pd.read\_csv("/mnt/data/dataset.csv")

df["Timestamp"] = pd.to\_datetime(df["LastUpdatedDate"] + " " + df["LastUpdatedTime"],

                                   format="%d-%m-%Y %H:%M:%S")

df["OccupancyRate"] = df["Occupancy"] / df["Capacity"]

df["TrafficIndex"] = df["TrafficConditionNearby"].map({"low": 0, "medium": 0.5, "high": 1})

df["VehicleWeight"] = df["VehicleType"].map({"bike": 0.5, "car": 1.0, "truck": 1.5})

# Normalize features

scaler = MinMaxScaler()

df[["occ\_rate", "QueueLength", "TrafficIndex", "IsSpecialDay", "VehicleWeight"]] = scaler.fit\_transform(

    df[["OccupancyRate", "QueueLength", "TrafficIndex", "IsSpecialDay", "VehicleWeight"]])

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

def price\_baseline(prev\_price, occ\_rate, alpha=5.0, rho=0.3, lower=5.0, upper=20.0):

    raw = prev\_price + alpha \* occ\_rate

    capped = max(lower, min(upper, raw))

    return rho \* prev\_price + (1 - rho) \* capped

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

weights = np.array([0.4, 0.25, -0.2, 0.1, 0.45])

lambda\_factor = 1.2

def demand\_function(row):

    features = np.array([

        row["occ\_rate"], row["QueueLength"], row["TrafficIndex"],

        row["IsSpecialDay"], row["VehicleWeight"]

    ])

    return np.dot(weights, features)

def price\_demand(prev\_price, row, base=10.0, rho=0.3, lower=5.0, upper=20.0):

    demand = demand\_function(row)

    raw = base \* (1 + lambda\_factor \* demand)

    capped = max(lower, min(upper, raw))

    return rho \* prev\_price + (1 - rho) \* capped

# --- Model 3: Haversine Distance (for Competitive Pricing) ---

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

    c = 2 \* atan2(sqrt(a), sqrt(1 - a))

    return R \* c

# Example simulation over time

lot\_prices = {}

lot\_history = []

for \_, row in df.iterrows():

    lot = row["SystemCodeNumber"]

    prev\_price = lot\_prices.get(lot, 10.0)

    # Choose model

    new\_price = price\_demand(prev\_price, row)

    lot\_prices[lot] = new\_price

    lot\_history.append({

        "lot": lot,

        "timestamp": row["Timestamp"],

        "price": new\_price

    })

# Convert history to DataFrame for visualization or export

history\_df = pd.DataFrame(lot\_history)

# Save for plotting

history\_df.to\_csv("/mnt/data/lot\_price\_history.csv", index=False)

print("Simulation completed and pricing history exported.")