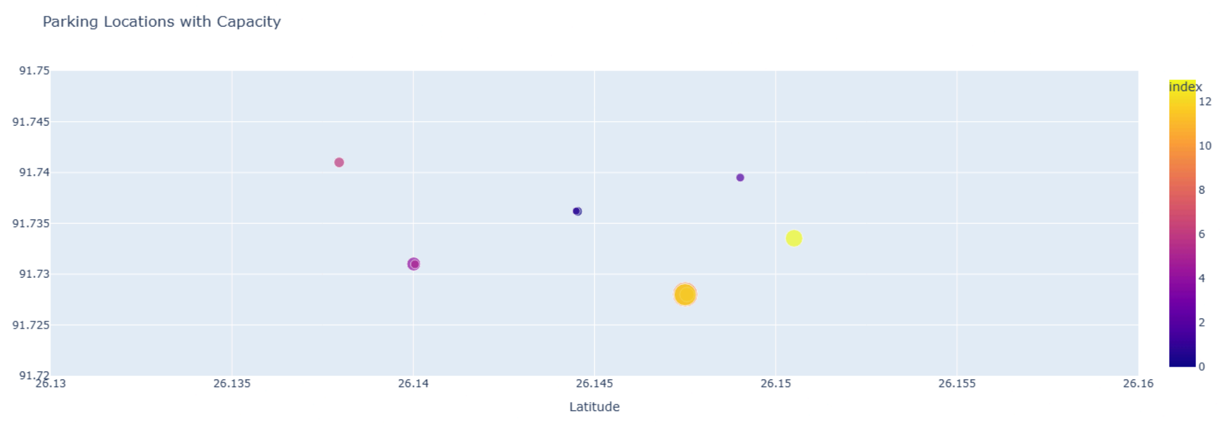
Machine Learning

DYNAMIC PRICING for URBAN PARKING

- Mayank Kumar Jha



Index

The Capstone project shows the implementation of various things in the scope of Data Science, Machine Learning & Deep Learning, I've learnt during the Summer Analytics course by IIT Guwahati during the 5 week period from May to June, 2025.

Problem Statement

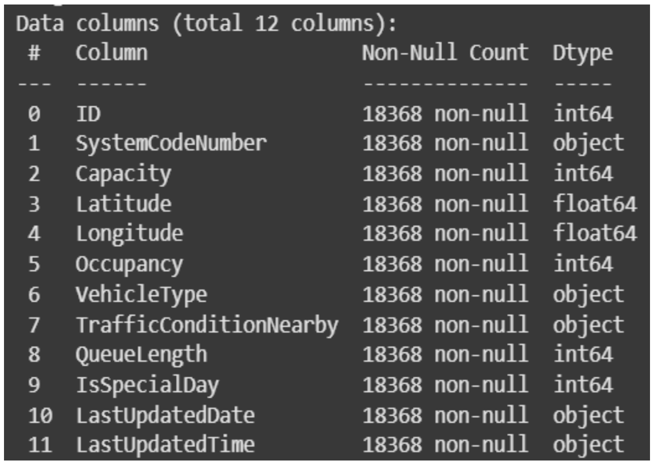
The problem was to design a pricing model for 14 different Parking Spots across an urban space. We needed to design a model which took care of several parameters and made a pricing for a vehicle based on the parameters dynamically. The constraint was to keep the price more than 0.5x the base price and less than 2x the base price which was $10.

It was told to make 3 different models, where each model grew with complexity, first be a baseline linear model, second be a demand based model and third be a competitive model.

Dataset analysis

The initial analysis of data had 14 unique parking spots. Each spot had a specific 'SystemCodeNumber' to it. For each SystemCodeNumber, there were 3 parameters, Latitude, Longitude & Capacity.

Apart from that, there were more parameters like 'TrafficConditionNearby', 'VehicleType', 'QueueLength', 'IsSpecialDay', 'LastUpdatedDate' & 'LastUpdatedTime'.



```
Data columns (total 12 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   ID                      18368 non-null  int64
 1   SystemCodeNumber        18368 non-null  object
 2   Capacity                18368 non-null  int64
 3   Latitude                18368 non-null  float64
 4   Longitude               18368 non-null  float64
 5   Occupancy               18368 non-null  int64
 6   VehicleType             18368 non-null  object
 7   TrafficConditionNearby  18368 non-null  object
 8   QueueLength             18368 non-null  int64
 9   IsSpecialDay            18368 non-null  int64
 10  LastUpdatedDate         18368 non-null  object
 11  LastUpdatedTime         18368 non-null  object
```

We can see the data types of all the columns from the df.info() command. The column 'ID' was just numeric and gave numbers from 1 to 18368 which we can drop for data analysis. There were 14 unique values for the group {'SystemCodeNumber', 'Capacity', 'Latitude', 'Longitude'} so we can store them in a metadata dictionary.

The 'VehicleType' column had 4 unique entries, 'Cycle', 'Bike', 'Car', 'Truck' in increasing order of size useful for pricing models. 'TrafficConditionNearby' was also categorical with data like 'low', 'average', 'high'. 'QueueLength' and 'Occupancy' gave more insights about space available and congestion. 'IsSpecialDay' recorded 0 and 1 for No and Yes. The last two columns gave the date and time of the update.

Feature Engineering

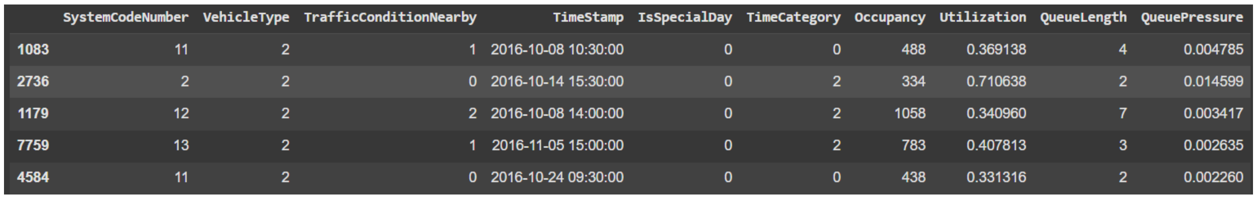
Some of the easier things to do while feature engineering were to reduce the 4 columns of SystemCodeNumber, Capacity, Latitude & Longitude into one single column which was encoded into numbers from 0 to 13 where each number was the index for these data in a metadata dictionary. I also made an id_map to store indices with their corresponding SystemCodeNumber string.

The next step was to convert the last two columns of LastUpdatedDate and LastUpdatedTime into one single column, 'TimeStamp' which should be a datetime object. Also according to the problem, data was collected every 30 minutes in a day, so we made the time rounded to 18 unique time stamps from 8:00 to 16:30. Also we had data for 73 consecutive days, so this reduced the dimensionality of data hugely for analysis further.

I introduced a new column of 'TimeCategory' which divided the 18 unique time stamps of a day into 3 categories, morning (0), noon (1) and afternoon (2) for better analysis.

I also encoded the 'VehicleType' into the format {'Cycle' : 0, 'Bike' : 1, 'Car' : 2, 'Truck' : 3}.

Since the 'Occupancy' column didn't give a clear picture about congestion at a spot, neither did 'QueueLength'. So I made two new columns, 'Utilization' = 'Occupancy'/'Capacity' for a spot. 'QueuePressure' = 'QueueLenght'/'EmptySpots' where 'EmptySpots' = 'Capacity' - 'Occupancy'.



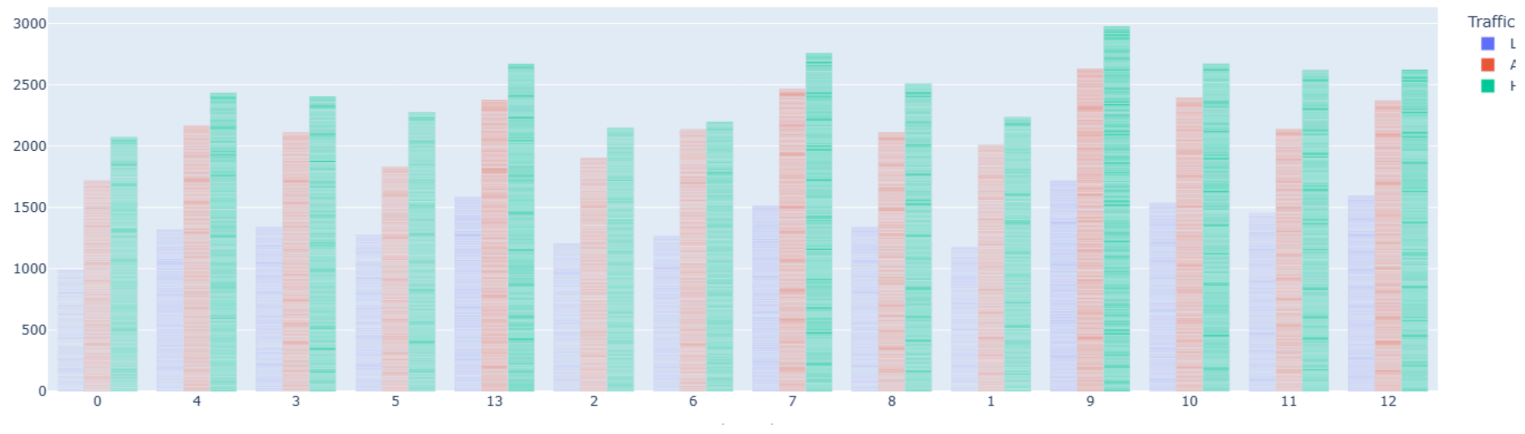
	SystemCodeNumber	VehicleType	TrafficConditionNearby	TimeStamp	IsSpecialDay	TimeCategory	Occupancy	Utilization	QueueLength	QueuePressure
1083	11	2	1	2016-10-08 10:30:00	0	0	488	0.369138	4	0.004785
2736	2	2	0	2016-10-14 15:30:00	0	2	334	0.710638	2	0.014599
1179	12	2	2	2016-10-08 14:00:00	0	2	1058	0.340960	7	0.003417
7759	13	2	1	2016-11-05 15:00:00	0	2	783	0.407813	3	0.002635
4584	11	2	0	2016-10-24 09:30:00	0	0	438	0.331316	2	0.002260

Sample of the transformed dataset after Feature Enginnering

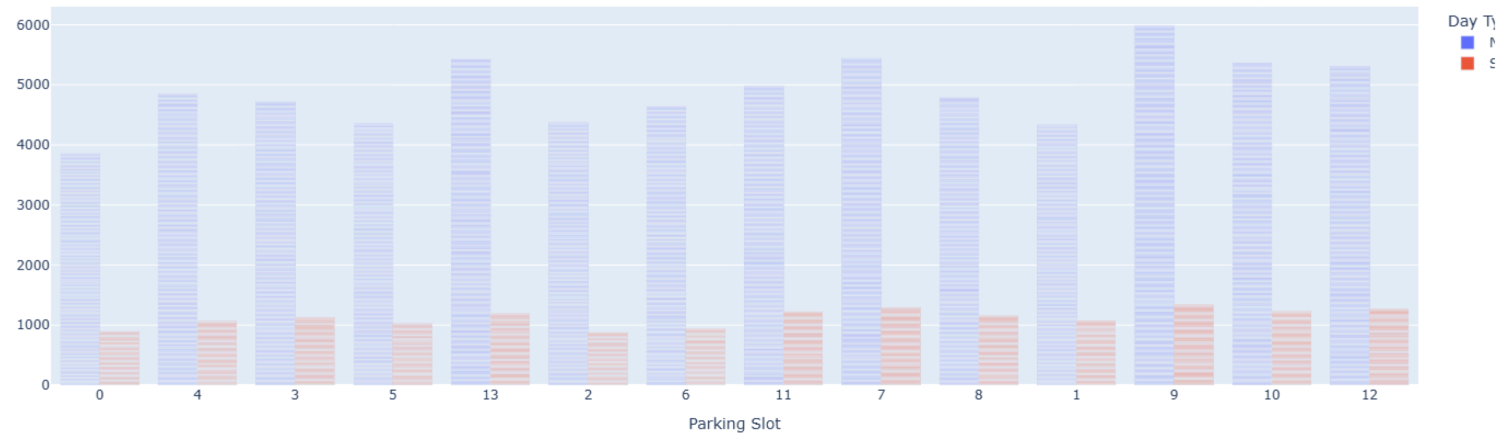
Exploratory Data Analysis (EDA)

During EDA, I explored while plotting the Average QueueLength along with many different features to analyse their effects for better modelling. The plots for the same are pasted below.

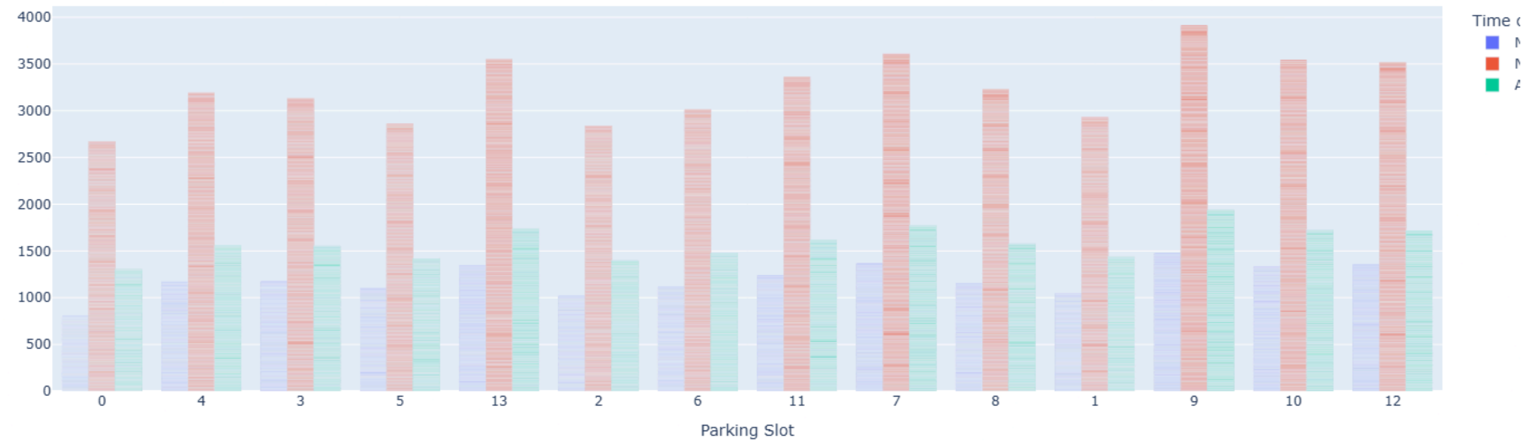
Avg QueueLength per Parking Slot by Traffic Condition



Avg QueueLength per Parking Slot by Special Day



Avg QueueLength per Parking Slot by Time of Day



Predictive Modelling

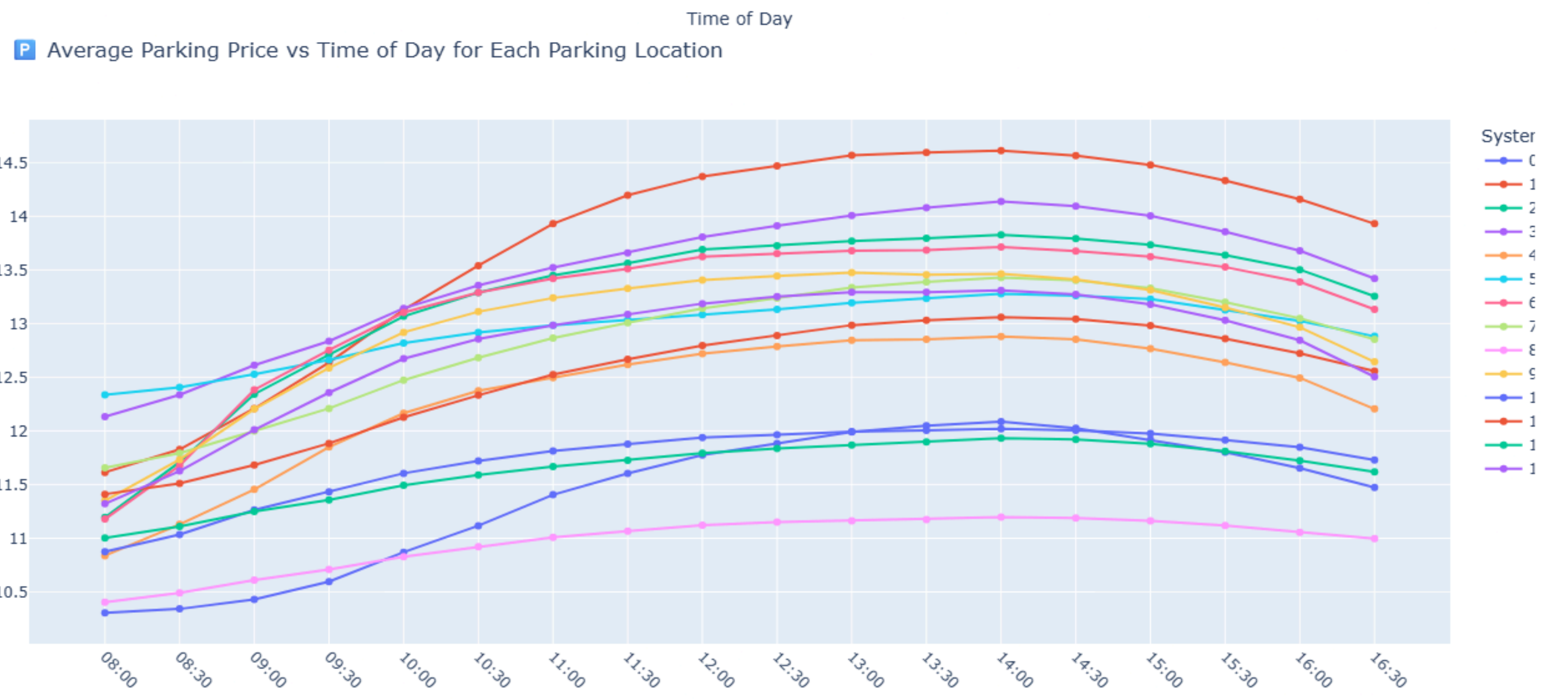
1. Baseline Linear Model

In this model, I used a very simple price function which is a linear factor of Utilization.

$$price = base\_price + \alpha * utilization$$

Here, base_price was $10 and we put α as 5 for the model and it runs well.

I tried to plot the model's price vs time of day using plotly express. Different colours are for different parking spots.



2. Demand Based Pricing Model

In this model, I made use of various parameters for making the demand function work most efficiently.

First we calculate

$$raw\_demand = \alpha * Utilization + \beta * QueueLength - \gamma * TrafficConditionNearby + \delta * IsSpecialDay + \varepsilon * VehicleType$$

Then I normalised this rawDemand between 0 and 1 using MinMax normalization.

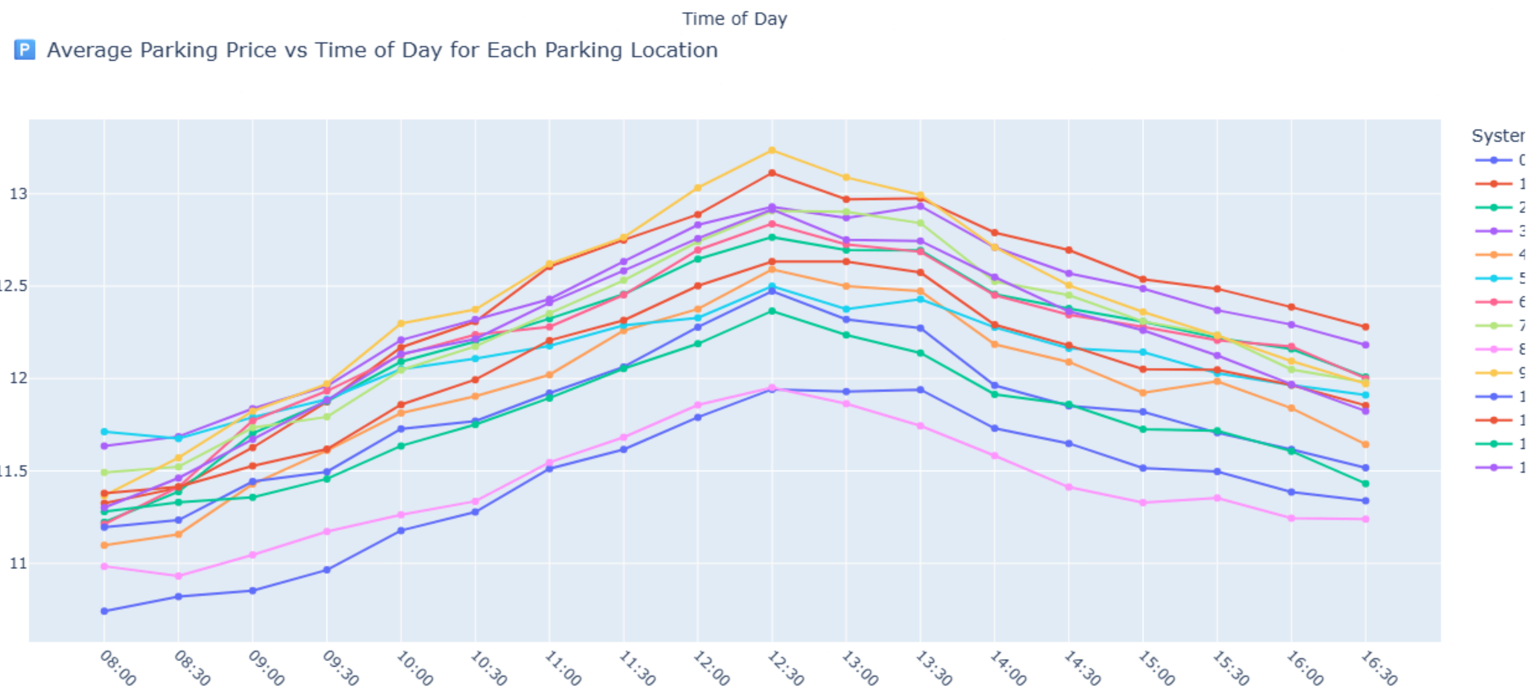
$$norm\_demand = (raw\_demand - min\_demand)/(max\_demand - min\_demand + 10^{6})$$

Now, use this normalised demand to calculate the price with base price.

$$price = base\_price * (1 + \lambda * norm\_demand)$$

Here, base_price=10, α=3, β=0.3, γ=0.5, δ=1, ε=0.5, λ=0.5

The plot for this Model is shown below.



3. Competitive Price Model (with Location-Aware Adjustments)

In this model, we used the Latitude and Longitudinal data and a self made Linear Regression model to train the dataset for pricing prediction.

For working in this model, I need to define some of the maths which is going to be used.

Min-Max normalization of columns using below functions

$$x_i^{\text{norm}} = \frac{x_i - \min(x)}{\max(x) - \min(x) \ + \ 10^6}$$

Linear Regression Function (X,y)

Here, we calculate the weights of each feature using below functions

$$\mathbf{w} = \left(X_{\text{bias}}^{\top} X_{\text{bias}}\right)^{-1} X_{\text{bias}}^{\top} \mathbf{y}$$

where, $X_{\text{bias}} = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1n} \\ 1 & x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{m1} & x_{m2} & \dots & x_{mn} \end{bmatrix}$

Haversine function to calculate the great circle distance between two points

Convert points to radians, $\phi_1 = \text{radians}(lat_1), \quad \phi_2 = \text{radians}(lat_2)$

Find delta between them, $\Delta\phi = \phi_2 - \phi_1, \quad \Delta\lambda = \text{radians}(lon_2 - lon_1)$

Then calculate the haversine function and finally the distance

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

$$d = 2R \cdot \arcsin\left(\sqrt{a}\right)$$

Now, we normalize the QueueLength feature of the data using min-max normalization.

$$Q_i^{\text{norm}} = \frac{Q_i - \min(Q)}{\max(Q) - \min(Q)}$$

$$\text{If } Q_i \geq 10 \text{ and } P_i \geq 1.0, \quad \text{then: } p_i = p_{\text{base}} \cdot \alpha$$

Then we calculate handcrafted prices for all the rows by applying random weights to all the required features.

$$\text{mult}_i = 1 + \left(0.35 \cdot u_i + 0.25 \cdot P_i + 0.1 \cdot Q_i^{\text{norm}} + 0.1 \cdot \frac{V_i}{3} + 0.05 \cdot \frac{T_i}{2} + 0.1 \cdot \frac{C_i}{2} + 0.05 \cdot S_i\right)$$

$$p_i^{\text{temp}} = p_{\text{base}} \cdot \text{mult}_i$$

Now use the matrix shown beside for linear regression model training.

$$X = \begin{bmatrix} u_1 & P_1 & Q_1^{\text{norm}} & V_1 & T_1 & S_1 & C_1 \\ u_2 & P_2 & Q_2^{\text{norm}} & V_2 & T_2 & S_2 & C_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_m & P_m & Q_m^{\text{norm}} & V_m & T_m & S_m & C_m \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} p_1^{\text{temp}} \\ p_2^{\text{temp}} \\ \vdots \\ p_m^{\text{temp}} \end{bmatrix}$$

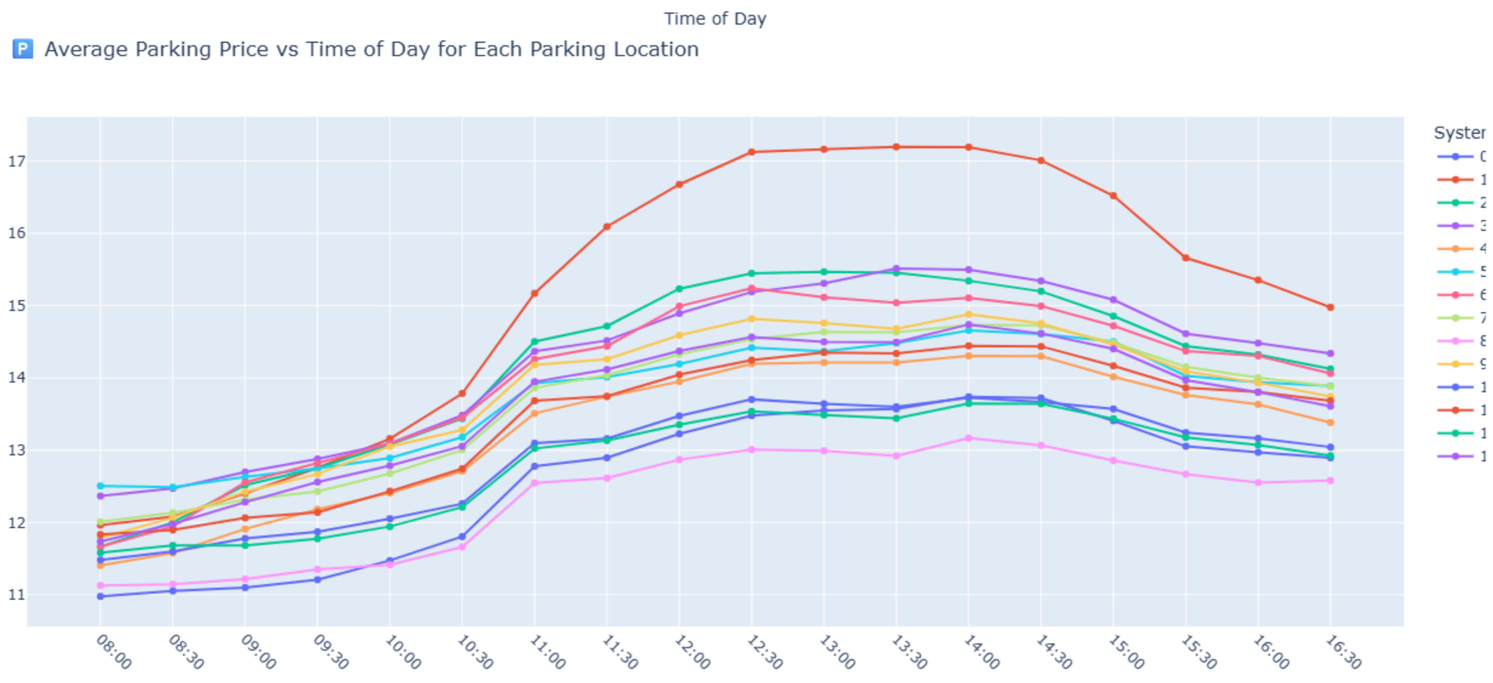
We get, $\mathbf{w} = (X_{\text{bias}}^\top X_{\text{bias}})^{-1} X_{\text{bias}}^\top \mathbf{y}$

Where, $X_{\text{bias}} = \begin{bmatrix} 1 & u_1 & P_1 & Q_1^{\text{norm}} & V_1 & T_1 & S_1 & C_1 \\ 1 & u_2 & P_2 & Q_2^{\text{norm}} & V_2 & T_2 & S_2 & C_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & u_m & P_m & Q_m^{\text{norm}} & V_m & T_m & S_m & C_m \end{bmatrix}$

Thus we get the price column as $\hat{p}_i = X_{\text{bias}} \cdot \mathbf{w}$

Then we clip the prices so that it doesn't go too high.

Now, let's see the plot of the prices against the time of the day



Rerouting Logic

Now we look upon the logic to reroute the vehicles if a parking spot is too crowded. We use the haversine distance to locate the nearest parking locations to reroute.

$$d(s_c, s_r) = 2R_E \cdot \arcsin\left(\sqrt{\sin^2\left(\frac{\Delta\phi}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\Delta\lambda}{2}\right)}\right)$$

To limit rerouting unnecessary, we take care of some conditions like minimum pricing difference & queue pressure at that parking spot.

$$d(s_c, s_r) \leq 2.5 \text{ km} \qquad q_c > q_{\text{high}} \quad (\text{e.g., } 1.0) \qquad q_r < q_c$$

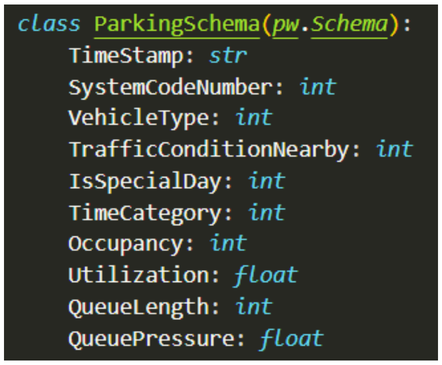
$$q_{\text{moderate}} \leq q_c \leq q_{\text{high}} \quad (\text{e.g, } 0.8 \leq q_c \leq 1.0)$$

$$(p_c - p_r) \geq \delta_p \quad (\text{e.g., } 2.0)$$

Find best reroute option : $\text{argmin}_{s_r}\,(q_r,\ p_r,\ d(s_c, s_r))$

We can see by calculating rerouting percentages that the QueuePressure is most at the parking spot with SystemCoded as 1 with more than 19% of the vehicles needing to be rerouted. But still the logic ensures most of the data was not suggested reroute.

Data Streaming using Pathway



```
class ParkingSchema(pw.Schema):
    TimeStamp: str
    SystemCodeNumber: int
    VehicleType: int
    TrafficConditionNearby: int
    IsSpecialDay: int
    TimeCategory: int
    Occupancy: int
    Utilization: float
    QueueLength: int
    QueuePressure: float
```

Now we look at the logic used for data streaming using Pathway.

We first took the transformed data into a csv file ready to be streamed.

Then we created a schema using pw, then made some user defined functions (@pw.udf) for normalizing columns and used the weights from the Model 3 linear model for streaming the data.

Then we create a 1-Day tumbling window and also take out the output_price_stream.csv

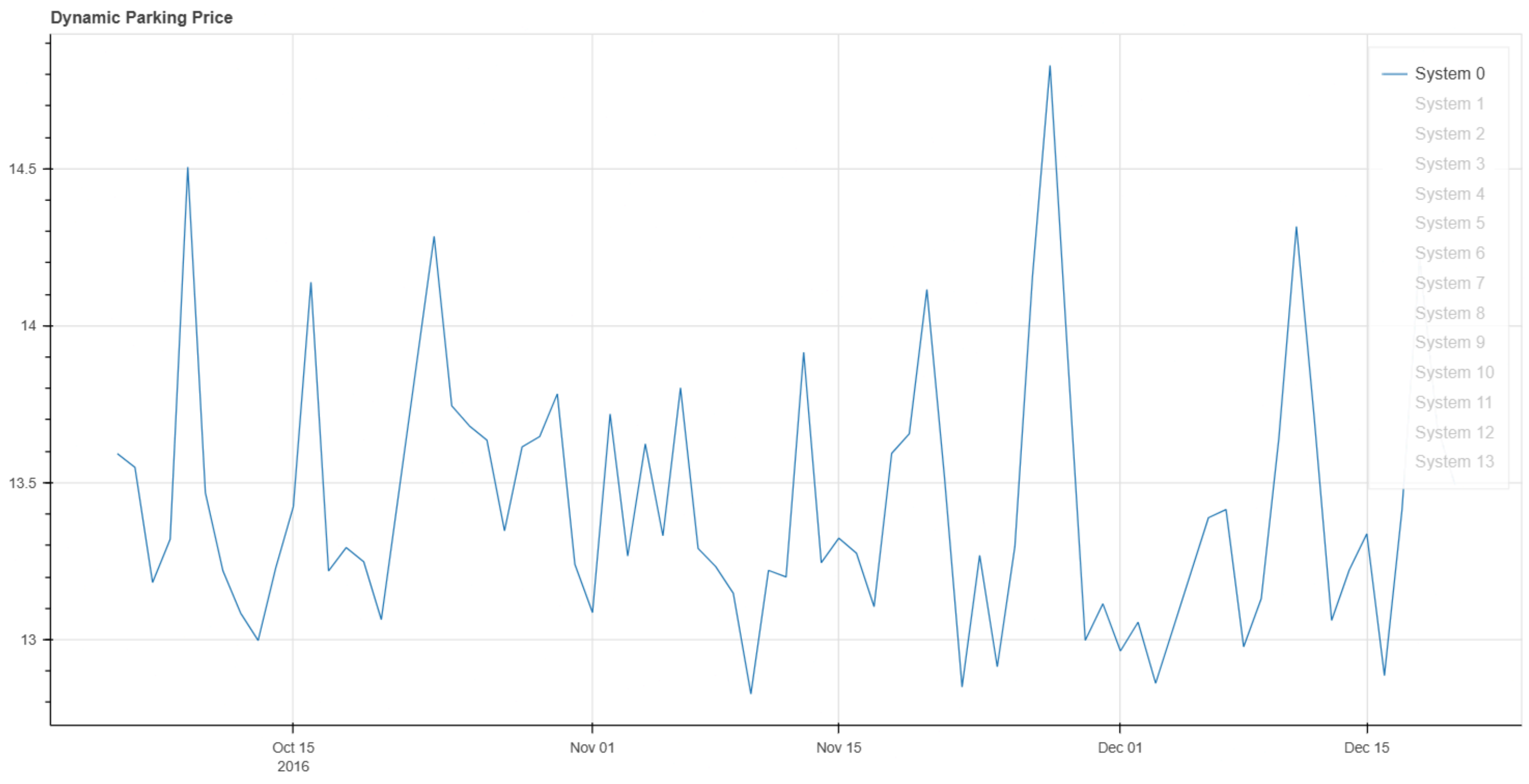
Then we run the code cell with pw.run() to start streaming of data with 100 inputs per second.

Bokeh Plot

Now we make the bokeh plots using the streaming data from pathway.

We also gave a dropdown for selecting the parking spot whose price stream we want to see. We can see the data of multiple parking spots at a time to compare their prices for the 73 day timeline given in the quesiton.

You can see a sample plot in next page of report.



THANK YOU :)

Useful links :

- [Problem Statement & Dataset]()
- [Github Repository]()
- [Colab Notebook]()