DYNAMIC PARKING PRICING USING TRANSACTION DATA

BY

WENHAN LUO

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTERS OF APPLIED SCIENCE
IN CIVIL ENGINEERING

Toronto, Ontario, Canada, 2024

Abstract

Managing on-street parking in dense urban areas poses challenges due to high demand and limited parking space availability, leading to increased congestion and search times for drivers. This thesis explores the efficacy of implementing a dynamic parking pricing policy inside a parking network to mitigate these challenges. Dynamic parking pricing adjusts prices based on parking demands, aiming to balance parking occupancy across different areas. The research investigates the feasibility of utilizing transaction data to predict parking occupancy, eliminating the need for expensive occupancy detection infrastructure. A predictive Neural Network is generated, and a price-setting algorithm is proposed to optimize and change parking prices to increase availability in high-occupied areas and attract drivers to underutilized spaces. The price-adjustment algorithm does iterative testing on an existing parking network and targets areas that need price changes through the number of spaces open on average. All identified parking areas go through a price increase/decrease adjustment phase, and the new occupancy's are updated throughout the entire parking network. The algorithm ends when the parking network sees no more benefits with a price adjustment. Final results demonstrated a substantial increase in available parking spaces in congested parking areas, and a decrease in underutilized parking locations, where more drivers would park at low-density parking areas when prices were reduced. The new parking network had a 17% increase in parking areas opening during the morning weekdays, a 17% increase for afternoon weekdays, a 14% increase for evening weekdays, a 1% increase for morning weekends, a 4% increase for afternoon weekends and 20% increase for evening weekends. Recommendations are provided for parking authorities, suggesting the implementation of dynamic pricing in high-occupancy areas, like downtown cores, to alleviate parking shortages and congestion. This strategy could be gradually introduced to address urban parking challenges effectively.

Dedication

I want to dedicate this thesis to my friends and family that have supported me throughout this journey. I owe them many thanks for the love and words of encouragement in completing this thesis.

Acknowledgements

I would like to thank my supervisor Dr. Mehdi Nourinejad for the endless guidance throughout this research, and giving me the opportunity to work on this fascinating project. I would also like to thank the parking team consisting of Hesam Rashidi, Fatemeh Sadeghi and Raneem Ayoub for the valuable insights and discussion that led to the completion of the this thesis. Additionally, I would like to thank the city of Toronto, Toronto Parking Authority for their collaboration for portions of this work.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AUC Area Under Curve 17

GTA Greater Toronto Area 20

LSTM Long Short-Term Memory 18

MAPE Mean Percent Absolute Error 17

MSE Mean Squared Error 17

NN Neural Network 38

RMSE Root Mean Squared Error 17

TPA Toronto Parking Authority 20

Chapter 1

Introduction

1.1 Motivation

Parking management is complex in dense urban areas due to the high demand for downtown cores and insufficient parking supply. This complexity can cause adverse consequences such as long parking search times and walking distances from parking locations to final destinations. The authorities that manage these parking areas tend to utilize parking management strategies to help increase the performance of their parking system. Parking management refers to various policies and programs (e.g. mobility management, parking regulations, shared parking) that result in more efficient use of parking resources [2]. Parking pricing strategies range from hourly, progressive, time-of-day, and fixed-time pricing. Hourly pricing is widely applied, which charges drivers proportional to their parking duration, making it suitable for short stays associated with shopping and recreation-related trips. Time-of-day pricing adjusts parking rates based on the time of the day, such as the morning and afternoon, subject to the different parking demands. Similar to hourly pricing, progressive parking pricing increases the parking rate if the parking duration surpasses a specified time threshold, while fixed pricing charges drivers a fixed price for a specified duration, often a day or a period of the day, making it suitable for commuting-related trips. Figure 1.1 demonstrates the total parking payment depending on the parking duration in each pricing strategy. The x-axis represents the dwell time a driver stays parked at a location, and each parking payment structure is different depending on the policy.

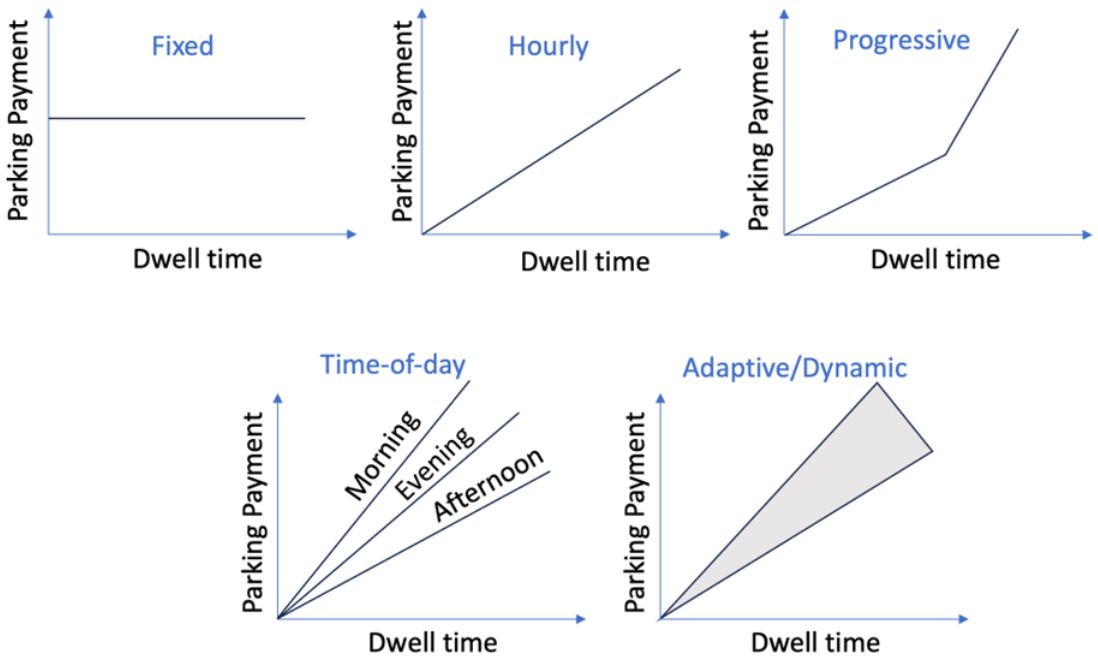


Figure 1.1: Parking Pricing Strategies

However, these existing pricing strategies fail to account for the overcapacity in high-density streets and underutilized parking spaces in low-density streets, as studies conducted in 11 major cities have found that the average search time for a parking space took around 8.1 minutes, and cruising for these spaces accounted for 30% of the city traffic congestion [3]. Also, it is estimated that emissions from parking infrastructure cost between $4- $20 billion annually, or between $6 and $23 per space per year in the US [4]. These increasing issues depend on newer methods of shifting demands of drivers dynamically, as you can adjust parking price changes according to demand for parking spaces.

Dynamic pricing is an alternative pricing strategy designed to be sensitive to demand variations. This policy adjusts parking rates at specified frequencies (e.g., monthly, weekly, or even hourly) according to monitored occupancy levels. The parking rates are adjusted to shift demand from high-occupancy to low-occupancy parking zones by increasing the rate of the former and decreasing that of the latter. Figure 1.2 (a) shows an example of dynamic parking pricing, with two nearby streets priced equivalently. The difference between the two is that the parking spaces in Street 1 are close to places with high foot

traffic (i.e., coffee shops, restaurants, clothing stores), while parking spaces in Street 2 are close to places with low foot traffic (i.e., commercial buildings). Attraction differences lead to a higher occupancy rate in street 1 (100%) than Street 2 (40%). Figure 1.2 (b) shows the two streets after implementing dynamic parking pricing. As the parking price of Street 1 increases to $4 from $3, the occupancy rate decreases as some drivers would not be as enticed to park there. However, as the parking price of Street 2 drops from $3 to $2, street 2 attracts more drivers, increasing the occupancy rate. If dynamic parking pricing is implemented throughout all parking areas, parking authorities could evenly distribute the demand between parking spaces and reduce cruising and traffic congestion.

Figure 1.2: Dynamic Parking Pricing Example

Dynamic pricing is currently implemented within areas of cities in the United States (US), such as San Francisco, Washington, and Los Angeles. In Canada, cities like Victoria (British Columbia), Vancouver, and Calgary have also implemented a form of dynamic pricing; the main difference between the country's approaches is the average period for price change. The average period for price change determines how often the city decides

to change the parking prices, as cities within the US change tend to change the parking prices every 3-4 months, while the Canadian cities change the prices annually. This period difference is due to most US cities having finer parking occupancy data gathered from sensors or cameras, while Canadian cities tend to use annual parking meter occupancy data. Canadian cities call their pricing "dynamic", but this policy seems like a more static approach, where cities just adjust rates annually to tackle demands.In light of the positive outcomes like increased parking availability, decrease in illegal parking, and reduction in parking search times, cities are now taking bold steps to expand dynamic parking pricing across other urban areas. However, the scalability of dynamic parking pricing presents challenges due to the high costs associated with the occupancy detection infrastructure. In the US pilots, cities deploy sensor technologies to gather occupancy level data to fine-tune parking prices within a parking network, this means every parking spot had sensor. This became a challenge, as to scale the parking policy, you would need to have millions of sensors throughout the country. Another challenge posed to scaling the parking policy, is maintaining the sensors, as some experience premature battery failure and and others are affected by electromagnetic interference with other transportation technology [5]. In this thesis the solution of using historical transaction data to predict occupancy rates is researched, as these data sets are rich with occupancy information. This approach eliminates the need for occupancy detection infrastructure and can be easily gathered through existing parking meters. By predicting occupancy levels using historical transaction data, a dynamic parking pricing system can be simulated to change prices according to demand levels within a network. However, there are limits to this approach, as vehicles parking across multiple spaces, illegal parkers, permit holders are not accounted for.

Looking further into the dynamic parking pricing approach, existing cities that have implemented this policy only change parking pricing every 3-4 months. This seems to be more of an "adaptive" approach, as you adapt your parking prices to the occupancy rates measured. However, as this method is championed by cities as "Dynamic Parking Pricing," we will continue to call it that to reduce confusion.

1.2 Research Goal

The goal of this research is to study the effects of using real transaction data to simulate a dynamic parking pricing system and analyze the results of an existing parking network when a dynamic parking pricing policy is enforced. Preliminary analysis is conducted on the raw transaction data to gain insights and trends regarding the demand patterns for different scenarios (time-of-day, parking prices, locations). From the analysis a Neural

Network is modelled to predict future occupancy rates from price changes, using historical transaction data. This Neural Network would be modelled to account for each parking locations' attraction to drivers. Factors such as points of interest, average parking pricing within the area, nearby parking capacity will used to characterize each parking area. An acceptable Neural Network would then be implemented into an algorithm that is built to review the original parking network and adjust parking prices (increase or decrease) depending on the amount of spaces available on average. When a parking price change is made, the effects of the price change is predicted, and the algorithm optimizes an entire parking network to open parking spaces in high-density parking areas and fill up parking spaces in low-density areas.

1.3 Research Objectives

This research sets four main objectives to examine the results of using transaction data for dynamic parking pricing. The four objectives include:

1. Conduct preliminary analysis to gain insights, trends and patterns that could be important to improving the occupancy rate prediction model (neural network).
2. Develop a Neural Network model using transaction data to predict parking occupancy at different on-street zones.
3. Develop a price setting optimization algorithm that incorporates the neural network to make parking price changes based on an original parking network and iterate through parking price changes to improve the network from its original state.
4. Investigate insights from the results of the algorithm and analyze the effects of implementing the parking pricing policy.

The goal of the research is to help parking authorities develop a framework, where they can utilize parking transaction data to determine if using transaction data is a feasible method to gather occupancy level information and implement it into a dynamic parking pricing system.

1.4 Scope & Thesis Organization

Although the developed algorithm can be tailored to any set of parking transaction data, this research's methodology is focused on on-street parking. As on-street parking spaces tend to be more spaced out than off-street parking spaces, the feasibility of using parking transaction data to replace occupancy detection infrastructure to do on-street dynamic parking pricing is within the scope of this research. A flow chart of this research's milestones is seen in Figure 1.3. The figure includes a preliminary data analysis on the transaction data, from cleaning data the initial raw data to data insights through visualization methods, then building an acceptable Neural Network model that predicts occupancy levels based on certain price changes. The next milestone is to develop the price-setting optimization algorithm that uses the neural network's predictions, and makes appropriate parking price changes throughout the entire parking network. The final milestone is to gather conclusions by comparing the original parking network, to the parking network with dynamic parking pricing implemented.

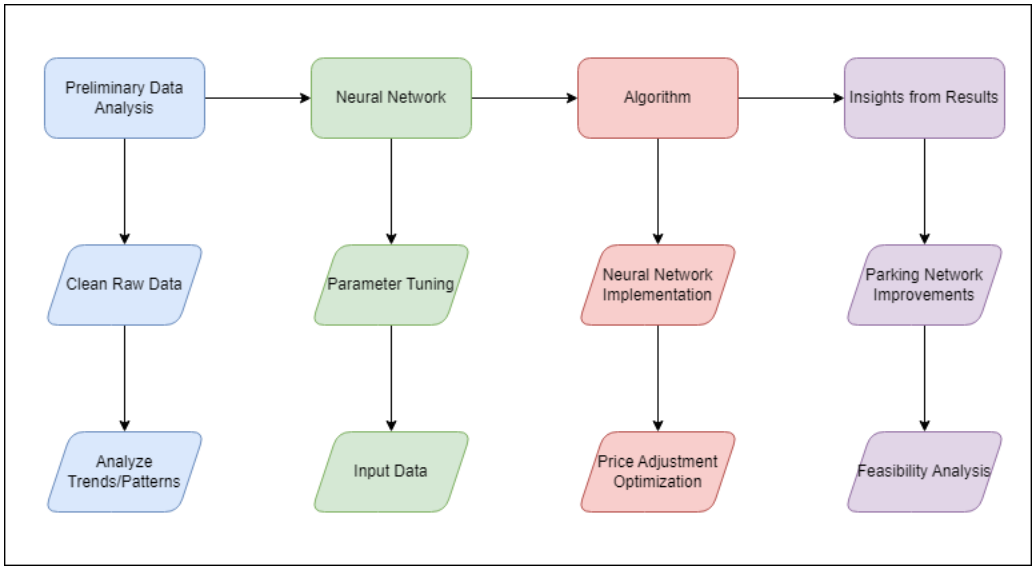


Figure 1.3: Research Milestone Flowchart

Chapter 2

Literature Review

This chapter discusses the literature on dynamic parking pricing. It also provides an overview of parking occupancy data collection methods,dynamic parking pricing pilots, and existing papers that focus on the topic of parking.

2.1 Dynamic Parking Pricing Method

Dynamic parking pricing consists of using a demand-based approach to adjust parking prices of high-demand parking areas (improve turnover) and low-demand parking areas (attract drivers). In this approach, parking rates for each area are adjusted depending on each area's occupancy rate. Prices are adjusted after some time (frequency), while the price change depends on the parking operators. Figure 2.1 illustrates a simple flowchart of how dynamic parking pricing works. First, a set parking price for each parking area is established, and then after a specific period, the gathered occupancy data indicates how occupied an area is. Next, a price adjustment depends on the pricing structure set beforehand. The process continues until all parking areas are deemed efficient.

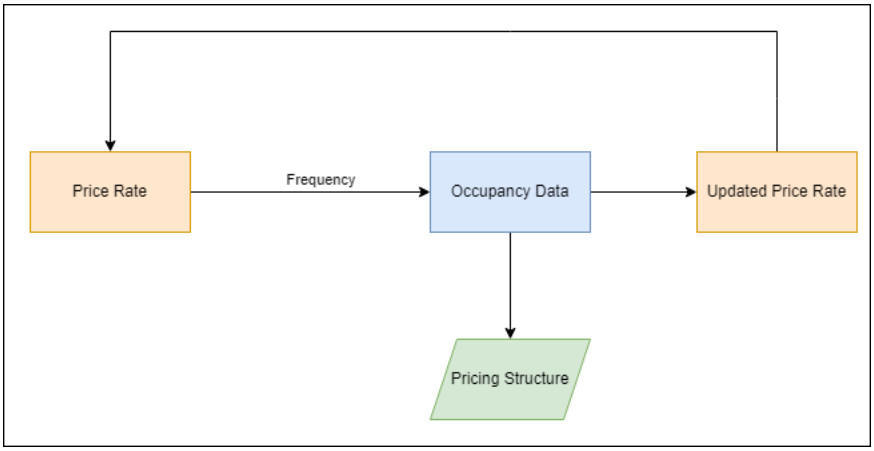


Figure 2.1: Dynamic Parking Pricing Method Flowchart

In the San Francisco Pilot, parking prices were adjusted every three months, and the price changes depends on the occupancy percentage. When occupancy is 80-100%, the hourly rate is raised by \$0.25, 60-80% the hourly rate is not changed, 30-60% the hourly rate is decreased by \$0.25 and less than 30% the hourly rate is lowered by \$0.5. As the city already enforced hourly rates, the prices were changed from the original depending on the parking area's zone. This ranged from \$2.00-\$3.50 [5]. Similarly, the city of Washington changed parking pricing according to occupancy at a frequency of 3 months, using a mixture of parking sensors and cameras to gather the occupancy rates. However, Washington's pilot had a different pricing structure; within the pilot's study area, there was already a district-wide price of \$2.30. As the pilot went on, the price rate structure changed. Figure 2.2 showcases how the rate structure gradually expanded through every price change. From the baseline of one pricing rate, by the end of the pilot (November 2017), the rate structure had nine different parking prices depending on the area, ranging from \$1.00 - \$5.00

PRICE CHANGE	RATE STRUCTURE (HOURLY RATES)								
Baseline				$2.30					
Round 1 *October 2016*			$2.00	$2.30	$2.75				
Round 2 *February 2017*		$1.50	$2.00	$2.30	$2.75	$3.25			
Round 3 *May 2017*	$1.00	$1.50	$2.00	$2.30	$2.75	$3.25	$4.00		
Round 4 *August 2017*	$1.00	$1.50	$2.00	$2.30	$2.75	$3.25	$4.00	$4.75	
Round 5 *November 2017*	$1.00	$1.50	$2.00	$2.30	$2.75	$3.25	$4.00	$4.75	$5.50

Figure 2.2: City of Washington Dynamic Parking Pricing Rate Structure

2.2 Parking Occupancy Data Collection Methods

Parking occupancy refers to the amount of spaces available to drivers for a specific parking area. Each parking area has a capacity, and no more cars can park there when capacity reaches its maximum. With an increase in full parking areas, the parking search time of drivers increases. Parking areas are scarce as the increase of drivers on the road outweighs the number of new parking spaces; this is why discovering this full parking area is necessary, as demands need to be evenly distributed to avoid park cruising. In dynamic parking pricing, parking occupancy determines if a parking area's price needs to be adjusted, depending on how empty or full it is. However, data collection methods are used to determine parking occupancy. These methods are able to give us the exact amount of capacity taken throughout the hours of operation, and this information is used to make pricing adjustments. There are multiple methods to gather this information, the main detection technologies are magnetic sensors, cameras, manual surveys, and transaction data.

2.2.1 Sensors

In-ground magnetic sensors are small pucks that detect the presence of a single vehicle above them. When a vehicle is detected for a spot, it is deemed occupied and will be captured in the data collection. Since the sensors are actively running, all vehicles that occupy spaces are accurately counted. Sensors can detect all types of vehicles. However, it is worth noting that if a larger vehicle takes multiple spaces, the sensor will detect all those

spaces to be occupied. Figure 2.3 (a) shows an example of a magnetic puck installed in the pavement using an adhesive. These sensors are self-powered wireless devices that send occupancy data to sensor management systems via a network of nearby gateways. SF-Park (San Francisco Dynamic Parking Pricing Pilot) is an example of a pilot using magnetic sensors for a dynamic parking pricing, with a total of 8,200 StreetSmart LLC in-ground sensors installed during the pilots lifetime [5].

(a) Parking Sensor

(b) Portable CCTV Camera

(c) Time-Lapse/Fixed Camera

Figure 2.3: Parking Occupancy Detection Methods

During the SF-Park pilot, using sensors encountered two main problems. First was electromagnetic interference caused by overhead lines for buses, light rail vehicles and alternating currents from junction boxes. Electromagnetic interference reduced the accuracy of the in-ground sensors, which affected the vehicle occupation detection. The second issue noticed during the pilot, was early battery failures, as the sensors did not reach their advertised 5-year battery life. Many sensors had to be replaced or troubleshooted, which increased overall maintenance costs. Today, newer sensor technologies are available that use radar and optical readers and have an overall better battery life spans, averaging around 8-10 years before the need of a replacement. Newer sensor models offer adhesive options that are less intrusive to infrastructure. In the Washington DC pilot, Park-DC, multiple puck sensor vendors tested the effectiveness and limitations of their sensors. Park-DC also worked closely with the sensor vendors to identify and solve sensor communications issues and adjusted the sensor ping frequency until it was at an acceptable rate to gather occupancy data [6].

2.2.2 Cameras

Cameras obtain parking occupancy information by detecting vehicles inside the camera's field of view. Three types of cameras can detect occupancy, each with advantages and disadvantages. The first type is portable CCTV Cameras, which capture the footage of a parking area, giving details on vehicle lengths, vehicle counts, vehicle speeds and vehicle classification. Portable CCTV Cameras are mounted on mobile trailers that can be relocated to record data at different locations, see Figure 2.3 (b). From portable CCTV cameras, the footage can be processed in two ways: either manually counting for small-scale data collection or using artificial intelligence to determine the parking occupancy for large-scale data collection. While portable CCTVs offer certain advantages, their potential to block traffic and limited lens angles make them a less viable solution for measuring parking occupancy in urban areas. In dense urban cities, the physical footprint of portable CCTVs can be a concern. These cameras can obstruct essential services or impede traffic when placed, despite their portable design. This issue is further compounded by their limited lens angles, which restrict the field of view and, thus, the effectiveness in monitoring a comprehensive area.

Time-lapse cameras offer an alternative method for occupancy detection (see Figure 2.3 (c)) by capturing images at specific intervals (e.g., five minutes). Time-lapse cameras are mounted on city assets, like streetlights or traffic signal poles and provide a wider lens angle, covering a larger area than a CCTV camera. Despite their potential for greater area coverage, time-lapse cameras are not the ideal solution for urban settings due to their high maintenance costs and the logistics and environmental challenges concerning their power source.

The third type is a fixed camera, which is mounted on streetlights or signal poles, similar to time-lapse cameras, and captures a narrow field of view. The main flaw of fixed cameras is the scarcity of feasible mounting locations that provide an acceptable aerial view, a communications line, and power. The installation requires a permit from the pole operators in charge.

2.2.3 Transaction Data

Access to occupancy information requires a cost-effective, practical, and scalable approach. An often widely available data source that can be used for parking occupancy prediction is the parking transaction data, which has information about each parking payment's date and time of transaction and the price paid. Parking transaction data is readily available in

many cities (existing parking meters) and does not require investment in city infrastructure which can be costly and timely.

The main challenge with transaction data is payment compliance, as occupancy data doesn't capture permit-holders, illegal parkers – those who do not pay at all and those who overstay their payment, and early departers – those who leave earlier than granted. Parking occupancy inference from transaction data decreases in accuracy with more illegal parking and permit holders. Nevertheless, some pilots show reductions in illegal parking when dynamic pricing is implemented since the drivers would have better chances of finding a legal parking space. The Boston Sea Port pilot [7] reported noticeable reductions in double parking after implementing dynamic parking pricing. Due to higher parking availability, drivers are more willing to park legally as vacant parking spots are now open [7].

2.2.4 Hybrid Sources and Data Fusion

Data fusion combines multiple data sources, including magnetic pucks, cameras, and transaction data, to predict parking occupancy. Park-DC implemented data fusion by merging transaction data, pay-by-cell transactions, spatial occupancy sampling and parking citations [6]. Figure 2.4 shows a diagram for the combination of real-time and historical data sources that detect real-time occupancy and availability.

The Park-DC pilot comprised three phases to predict occupancy using data fusion. The first was using temporal data collected from portable CCTV to determine where to place in-ground sensors. The portable nature of the cameras allows them to be relocated to collect data from multiple sites and identify parking locations where demand patterns are unusual, requiring an in-ground sensor to gather the ground truth. The second phase was to refine the sensor occupancy information using data fusion. The combination of real-time and historical occupancy data improved occupancy predictions for dynamic parking pricing. The third phase involved finding the minimal viable sensor coverage. In Park-DC's case, 50% sensor coverage was initially assumed. The city then removed specific components from the data to determine if occupancy could still be accurately predicted. After the testing, Park-DC found that 50% sensor coverage was viable enough for the pilot area. However, this method requires costly parking occupancy infrastructure, which won't be required if a highly accurate model that predicts occupancy is developed.

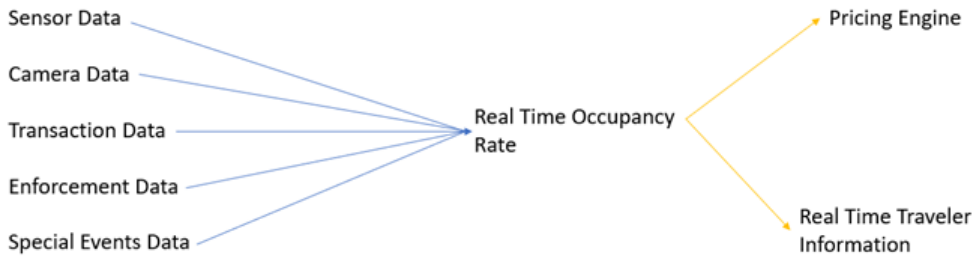


Figure 2.4: Data Fusion Diagram

2.3 City Pilots Overview

Table 2.1 presents the dynamic parking pricing strategies of select cities, highlighting each cities strategy of dynamic parking pricing. Despite shared goals of optimized pricing and occupancy levels, the variations in strategies highlight the lack of a universal solution. The following section reviews the most relevant pilots that implemented dynamic parking pricing. The mentioned pilots are still existing, with cities that have implemented a variation of dynamic parking pricing. The San Francisco, City of Boston and Washington DC pilots are further investigated as they are the cities that generated comprehensive reports on the dynamic parking pricing pilots implemented.

Table 2.1: Dynamic Parking Pricing City Overview

City	Year of Implementation	Occupancy Detection Method	Average Period Taken for Price Change	Target Occupancy	Time-of-day price variations
Seattle [8]	2010	Mix of Surveying, Transaction Data, and Historical Occupancy Data	Annually	70-85%	Three Weekday Periods
San Francisco [5]	2011	In-Ground Sensors	3 Months	60-80%	Three Weekday Periods
Los Angeles [9]	2012	In-Ground Sensors	3-6 Months	70-90%	Three Weekday Periods
Calgary [10]	2013	Unknown	Annually	50-80%	Four Weekday Periods
Washington [6]	2016	Variety of methods, mostly sensors	3 Months	None Stated	Three Weekday Periods
Boston [7]	2017	In-Ground Sensors	2 Months	60-80%	Three Weekday Periods
Baltimore [11]	2017	Surveys	6 Months	75-85%	None

2.3.1 San Francisco (SF-Park)

San Francisco launched SF-Park, a pilot that implemented dynamic parking pricing to better balance the parking supply and demand. About one-quarter of San Francisco's metered parking spaces were monitored with magnetic detection pucks. The parking rates were increased if the occupancy was too high (>85%) and decreased if the occupancy was too low (<60%), intending to keep at least one parking spot open on every block.

As an outcome of the pilot, the chances of experiencing occupancies between 60% to 80% increased by 31%. Moreover, the average park search time decreased by 43%, and vehicle miles travelled reduced by 30% [5]. Some parking meters were upgraded to allow for dynamic parking price changes, with the technology to implement frequent changes to rates remotely, needless of a technician to change them manually.

Upgraded parking meters added more functionality to the existing parking system. Real-time occupancy data could be disseminated through web and mobile apps, using

the aggregated information to help drivers make informed decisions about their travel plans. Upon implementing dynamic pricing, SF-Park installed roadway sensors to evaluate the traffic impacts of the initiative. These sensors gauged traffic volumes and speeds, allowing the city to appraise the impact of dynamic parking pricing on vehicular flow. The assessment helped determine whether the pilot program was beneficial in reducing traffic congestion.

2.3.2 City of Boston Pilot

In 2017, the City of Boston initiated a dynamic parking pricing pilot in the Seaport area, mirroring the objectives of the SF-Park pilot in San Francisco. The primary goal was to maintain a target occupancy rate between 60 – 80%, thereby encouraging drivers to either park in less congested areas or opt for alternative transportation methods. The pilot used sensors mounted on parking meters to monitor parking occupancy. These sensors informed pricing adjustments: rates would increase by 50 cents if occupancy surpassed 80%, with a capped hourly rate ranging between $1 and $4.

Concurrently, Boston explored new mobility strategies to reduce reliance on car parking. The initiatives included allocating more spaces for car-share operators and expanding the bike-share network. These measures were aimed at promoting alternative transportation modes. The pilot's impact assessment revealed notable reductions in illegal parking—12% in Back Bay and 35% in Seaport. Notably, the parking facilities rarely reached full capacity, which implies a higher availability of spaces. However, the pilot's effectiveness in Seaport was less apparent due to ongoing construction, new developments, and fluctuating seasonal demands, leading to minimal congestion reduction relative to operational costs.

Boston's pilot underscored several insights. Foremost was the importance of effectively communicating parking price changes to drivers, enabling better trip planning, and reducing the likelihood of cruising for parking. Additionally, the city identified limitations in the current sensor technology. The sensors provided unreliable occupancy data, thus challenging the scalability of this approach and complicating the assessment of demand levels. Consequently, Boston's experience suggested that alternative methods for measuring parking occupancy should be considered.

2.3.3 Washington DC (Park-DC)

In response to increasing traffic and limited parking, Washington D.C. launched Park-DC. Park-DC had three primary objectives: reducing parking search times, alleviating conges-

tion, and developing effective parking management solutions with minimal investment in new assets.

Park-DC's core strategy was to implement dynamic pricing, adjusted by demand and time of day. For occupancy detection, the city used a combination of parking sensors and surveillance cameras, supplemented by data from parking transactions. In addition to technological solutions, Park-DC focused on effective communication with drivers by installing regulatory signs to inform drivers about price changes and parking regulations. The effectiveness of the signs was assessed through surveys, providing insights into whether further improvements were necessary.

Park-DC also developed a mobile application to assist drivers with parking. The app facilitated the parking process and enabled the collection of valuable data on parking sessions, driver behaviour, and system performance. To evaluate the pilot's impact on congestion reduction and safety, Park-DC used metrics from manual surveys, parking citations, Bluetooth sensor data, census information, and camera footage. Finally, Park-DC explored the scalability of different occupancy detection methods for city-wide implementation, focusing on cost-effective solutions, they concluded that using sensors was the most effective way to gather occupancy data, but future implementations of hybrid data sources has potential.

2.4 Parking Occupancy Prediction Related Literature

2.4.1 Parking Occupancy Prediction Models

Researching the implementation of a dynamic parking pricing system using transaction data as the source of occupancy levels holds relevance within parking occupancy prediction models using machine learning techniques. As prediction frequency (minute, hour, daily) and parking study type (on-street parking group, off-street parking lot, block level, individual parking spaces) impact the methodology used for prediction model application, existing literature provides numerous ideas to designing an accurate parking occupancy prediction model. Table 2.2 shows relevant papers using machine learning techniques to predict/estimate parking occupancy (block level, spaces). Performance metrics are used within each paper to assess the accuracy of each model, and all studies show the use of various machine-learning techniques. [12] combined a gated convolutional neural network and a long-short-term memory to capture spatial and temporal traffic features to predict block-level occupancy 30 minutes in advance in downtown Pittsburgh. Structuring the model's

capability to source heterogeneous structure traffic data sources, the model outperforms other baseline methods with a Mean Percent Absolute Error (MAPE) of 10.6%. It was found that incorporating traffic speed and weather information significantly improved the mode's performance, with weather data particularly improving recreational parking area predicting accuracy. [13] took a hybrid approach of using traffic and parking sensor data to combine with historical occupancy parking data to predict parking garage occupancy. Two models were generated, and the neural network outperformed the random forest model with an Mean Squared Error (MSE) of 7.18 to 7.98. Historical occupancy data was the most important predictive variable in the study, followed by traffic flow data near the study area. Using an economical workflow based on cloud platform elastic computing services [14] was able to develop a system that used a long short-term memory model that estimates the number of available parking spaces every two hours based on live weather and parking sensor occupancy data. The suggested network performed well with an Root Mean Squared Error (RMSE) of 5.42 compared to the traditional back propagation neural network RMSE of 13.87. [15] used heterogeneous clustering and regression learning to predict block-level occupancy with San Francisco parking data (SF-Park). The model combined a convolutional neural network with a long-short term memory and a clustering augmented learning method to predict occupancy with a MAPE of 7.8%.

2.4.2 Parking Occupancy Prediction Factors

Building a feasible model requires key features that help the model output a better solution. Factors that affect parking trends, demands and occupancy are important to include when using a prediction model. Referring back to [15], the model aggregated analogous data (spatio-temporal information) into clusters to affect parking data processing. The model saw improvements to predictions helping predict block-level occupancy. [16] predicted the availability of parking spaces using publicly accessible data. Suitable categories were tested to see the importance of each feature for prediction. From the neural network model, the outputted predictions had an average MSE of 0.16, showing that weekday, time of day, location, and temperature have a significant impact. In contrast, traffic, vacation time and rainfall are only secondary to a neural network model.

2.4.3 Thesis Model

The model presented in the thesis follows related literature by using a neural network to predict the number of spaces open for a specific parking group daily. Results indicate that the model performs with an MSE of 4.25 and an Area Under Curve (AUC) of 0.72.

Table 2.2: Related Literature Information Table

Paper	Paper Goal	Performance Metric	Result	Method
[12]	Predicting block-level parking occupancy 30 minute in advance using a deep learning model	MAE, MAPE	1.39 MAE and 10.6% MAPE	Gated Convolutional Neural Network + Long Short-Term Memory
[13]	Predict Occupancy Rate	MSE, MAE	Neural Network (MSE: 7.18, MAE: 1.39) Random Forest (MSE: 7.98, MAE: 1.92)	Neural Network and Random Forest
[14]	Estimate the number of available parking spots	RMSE	5.42	Long Short-Term Memory and Back Propagation Neural Network
[15]	Predict block-level parking occupancy	MAE	0.878	Convolutional Neural Network and Stacked LSTM Autoencoder
[16]	Estimate the probability of a free parking space within a certain region.	MSE	0.16	Neural Network
This Thesis	Predict # of Spaces Open for a Parking Group (Daily)	MSE	MSE: 4.25	Neural Network

2.5 Research Gap and Study Contributions

Despite existing research on occupancy prediction using transaction data and dynamic parking pricing systems, only some studies focus on using further spatial and temporal data to enhance occupancy prediction. As existing studies tend to focus on block-level parking, predicting individual parking groups allows for parking group price adjustments that can better level the demand for parking spaces. Table 2.3 shows related literature data sources used when training the prediction models, and the presented model includes other significant features such as parking price, neighbouring parking price, month (limited to three months) , capacity weight, and point of interest data (further discussed in section 5.2). These additional data sources further characterize a parking group, as the average nearby parking price explains what pricing scheme is in this nearby area, month creates a

seasonality gap of parking trends, capacity weight dictates how many available spaces are even in the area and point of interest data pinpoints the attractions (restaurant, school, hospital, etc.) nearby which all affect a drivers decision to park in a certain spot. These sources increased the prediction model's accuracy and characterized each parking group further than just historical occupancy. There are also no studies that use occupancy prediction to change parking prices inside a parking network. This thesis uses an algorithm built on a neural network to make appropriate parking price changes in high- and low-density parking groups.

This study contributes to the literature on dynamic parking pricing by using a plethora of data sources to build a prediction model, and formulating a necessary parking price change algorithm based on existing parking network. With the development of such an algorithm, policy and parking authorities will have a proper framework to build a cost-effective, dynamic parking pricing system by using transaction data as an occupancy detection method. This paper will build on the literature and provide an extensive real case study of how dynamic parking pricing could be implemented in Toronto.

Table 2.3: Related Literature Data Sources

Paper	Time-of-Day	Day-of-Week	Historical Occupancy Data	Traffic Flow	Holiday	Event	Weather	Location	Parking Sensor	Average Nearby Parking Price	Month	Capacity Weight	Point of Interest Data
[12]			√	√			√						
[13]	√	√	√	√	√	√	√						
[14]					√		√		√				
[15]	√	√	√	√	√	√	√	√	√				
[16]	√	√	√	√			√	√					
This Thesis	√	√	√				√	√		√	√	√	√

Chapter 3

Study Data

Parking transaction data is vital for building an accurate Neural Network model that can predict occupancy levels. This chapter discusses the parking transaction data used, the study area, and other data sources used in this thesis.

3.1 Toronto Parking Authority Transaction Data

Fine-level transaction data was used in this thesis and was provided by the Toronto Parking Authority (TPA) [17]. Fine-level refers to every single transaction made being tracked within the data and no aggreation was done to the given data. TPA is a municipal parking services company that provides off-street parking, on-street parking, and bicycle sharing to the population of Toronto. As TPA owns most parking areas in the Greater Toronto Area (GTA), conclusions made through the analysis are made with confidence regarding the on-street parking trends inside Toronto.

3.2 Study Area

The study area is mainly inside the downtown core of Toronto but extends north, west, and east. The study area extends to North York on the north end, with most parking groups mainly on Yonge St. In the west direction, the study area extends to Etobicoke, with parking groups mainly on Lake Shore Blvd W and Bloor St W. In the east direction, the study area extends to the East York area, with parking groups being on Danforth Ave and Queen St E. Figure 3.1 represents a map of the study area, with blue dots representing each parking group within the study area.

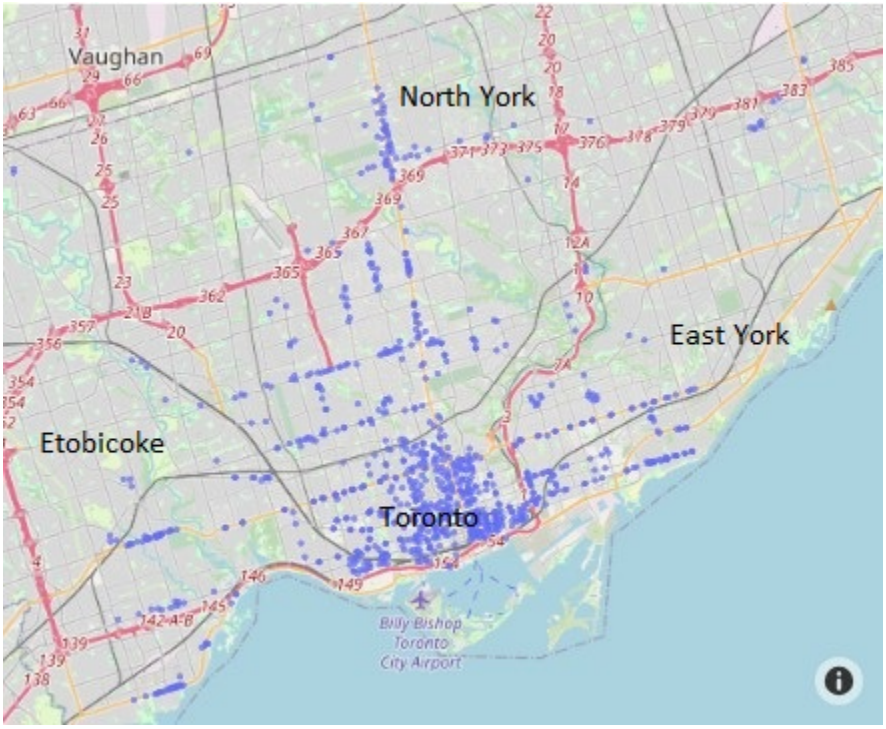


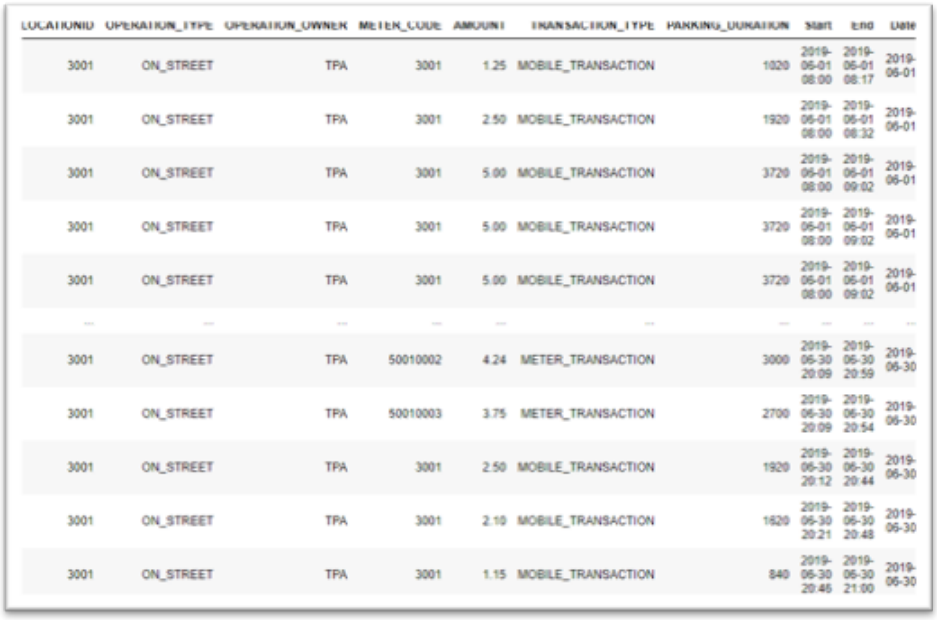
Figure 3.1: Study Area Map

3.3 Raw Data set Descriptions

The data set contains raw transaction data for 940 on-street parking areas throughout the GTA, with most parking locations inside downtown Toronto. TPA only provided us data gathered from the dates ranging May 2019 to July 2019 and May 2022 to July 2022. Figure 3.2 represents what information is given in each dataset. Each row inside the data set represents a specific transaction made by a driver and each column is described as:

- Location ID - Every parking area has a specific location associated with it. The parking area location is linked to a corresponding location id. The method of each location id given is unknown, but ids close in number range tend to be in the same vicinity.
- Operation Type - This is the operation type of the parking area, since the focus is for on-street parking, the entire dataset has the operation type of on-street parking.
- Operation Owner - Every parking area is owned by a different company, the column describes which company owns the parking area. As the data is only TPA data, all operation owner cells are labeled “TPA”.

- Meter Code - In certain parking areas, there can be multiple parking meters installed for that specific area. The meter code describes which meter the transaction was dealt at. If the transaction was through a mobile payment, the meter code equals the location id.
- Transaction Start Time - This column describes the exact transaction start time made by a driver. The outline for a transaction time is in the format of YEAR/MONTH/DAY HOUR/MINUTE/SECOND/AM or PM
- Transaction End Time - Like the start date time transaction column this describes the transaction end time a driver has paid for up until. It is in the same format as YEAR/-MONTH/DAY HOUR/MINUTE/SECOND/AM or PM. Note: This is the time the user has paid for, not when the user leaves the parking area.
- Amount - For every transaction made, the driver must pay a certain amount, either to the meter they are parked at or using the mobile app. This column describes the amount the driver has paid.
- Transaction Type - With the emergence of mobile technologies, there are two types of transactions that can be made. One being at the parking meter at a given parking area. The other is by using the TPA mobile app to pay for a parking spot at a parking area.
- Parking Duration - This column describes the parking duration the driver has paid for in seconds. This is in relation to the "AMOUNT" column, as driver pay a fixed amount for a certain parking duration.



LOCATIONID	OPERATION_TYPE	OPERATION_OWNER	METER_CODE	AMOUNT	TRANSACTION_TYPE	PARKING_DURATION	Start	End	Date
3001	ON_STREET	TPA	3001	1.25	MOBILE_TRANSACTION	1020	2019-06-01 08:00	2019-06-01 08:17	2019-06-01
3001	ON_STREET	TPA	3001	2.50	MOBILE_TRANSACTION	1920	2019-06-01 08:00	2019-06-01 08:32	2019-06-01
3001	ON_STREET	TPA	3001	5.00	MOBILE_TRANSACTION	3720	2019-06-01 08:00	2019-06-01 09:02	2019-06-01
3001	ON_STREET	TPA	3001	5.00	MOBILE_TRANSACTION	3720	2019-06-01 08:00	2019-06-01 09:02	2019-06-01
3001	ON_STREET	TPA	3001	5.00	MOBILE_TRANSACTION	3720	2019-06-01 08:00	2019-06-01 09:02	2019-06-01
...	...	...	...	...	...	...	...	...	...
3001	ON_STREET	TPA	50010002	4.24	METER_TRANSACTION	3000	2019-06-30 20:09	2019-06-30 20:59	2019-06-30
3001	ON_STREET	TPA	50010003	3.75	METER_TRANSACTION	2700	2019-06-30 20:09	2019-06-30 20:54	2019-06-30
3001	ON_STREET	TPA	3001	2.50	MOBILE_TRANSACTION	1920	2019-06-30 20:12	2019-06-30 20:44	2019-06-30
3001	ON_STREET	TPA	3001	2.10	MOBILE_TRANSACTION	1620	2019-06-30 20:21	2019-06-30 20:48	2019-06-30
3001	ON_STREET	TPA	3001	1.15	MOBILE_TRANSACTION	840	2019-06-30 20:46	2019-06-30 21:00	2019-06-30

Figure 3.2: Raw Transaction Data Example

3.4 Data Limitations

Developing an accurate neural network and algorithm requires a large set of data. Chapter 4 describes the data used when building the neural network. However, the data has imperfections, which limits the accuracy of prediction. To begin, the two data sets used have a three-year difference between them, and during this gap, the COVID-19 pandemic happened, changing the demand patterns of all transportation facilities. The 2019 data set does not account for the effects of the pandemic, while the 2022 data includes the aftermath of the pandemic. During these times, many variables can change the demand patterns of certain parking groups, hindering the accuracy of occupancy prediction. Also, during these three years, many parking management changes did occur, such as construction, parking space allocation (CurbTO), and parking price changes. The data was also only limited to three months in the year, meaning the neural network does not account for all seasons throughout the year. With more seasonal data, the neural network would capture the demand patterns better by understanding the demands each month of the year and within each season. Nonetheless, the provided was crucial for the entirety of this thesis.

Chapter 4

Preliminary Analysis

Before a transaction data dynamic parking pricing model is developed, the raw data must be cleaned to ensure the data is properly formatted and can be used for further data analysis. This chapter discusses how the raw transaction data is cleaned, and how preliminary analysis was done. The chapter ends with an overview of the data analysis and how it will structure the neural network model.

4.1 Cleaning the Data

A cleaning procedure is done to process the data and extract only the crucial information. As discussed in Chapter 3, fine-level transaction data by the minute and unnecessary data columns are removed from the primary data frame. The columns kept were:

- Location ID - The location id is important because it helps distinguish which parking group this transaction data is from. Keeping the location ID will help find trends for every specific parking group.
- Transaction Start Date Time - The transaction start data time is the transaction data information needed to know when drivers start parking.
- Transaction End Date Time - Like the start date time, this transaction data information,represents how long the driver has paid to park for. The information can help estimate the occupancy.

Three columns of data are used to determine the occupancy count. Once the data frame is shortened, the raw date times are turned into an object datetime using the datetime Python

library. The DateTime library allows changing the transaction date time into a date object that can be further manipulated. A date object helps quickly produce graphs or statistics in a (YYYY-mm-dd HH:MM:SS) format. Y means year, m means month, d means day, H means hour, M means minute, and S means second.

4.2 Occupancy Count

After the raw data is cleaned, valuable insights can be gathered by manipulating the cleaned data. The most critical data to extract is the occupancy count. The occupancy count refers to the number of vehicles that are counted inside a parking group throughout an entire day. This count involves a vehicle's transaction start time and end time. The primary assumption of this occupancy count is that every vehicle enters the parking group at the start and leaves at the end. In a real-world scenario, drivers can leave earlier or later, and there are illegal drivers with parking permits that this occupancy count does not account for. Figure 4.1 shows a simple diagram of how the occupancy count function works. At the start of the day (8:00 am on Monday-Saturday, 11:00 am on Sundays), a counter counts the number of vehicles entering and leaving a parking group based on the transaction start and end time. This count is done throughout the day and ends at the end of the day (9:00 pm). With a start and end, the counter will always begin at 0 and end at 0. The count is done for each parking group every day for three months. It is important to note that the vehicle transaction start time does not coincide perfectly with the end time as drivers select a period of time to park throughout the day.

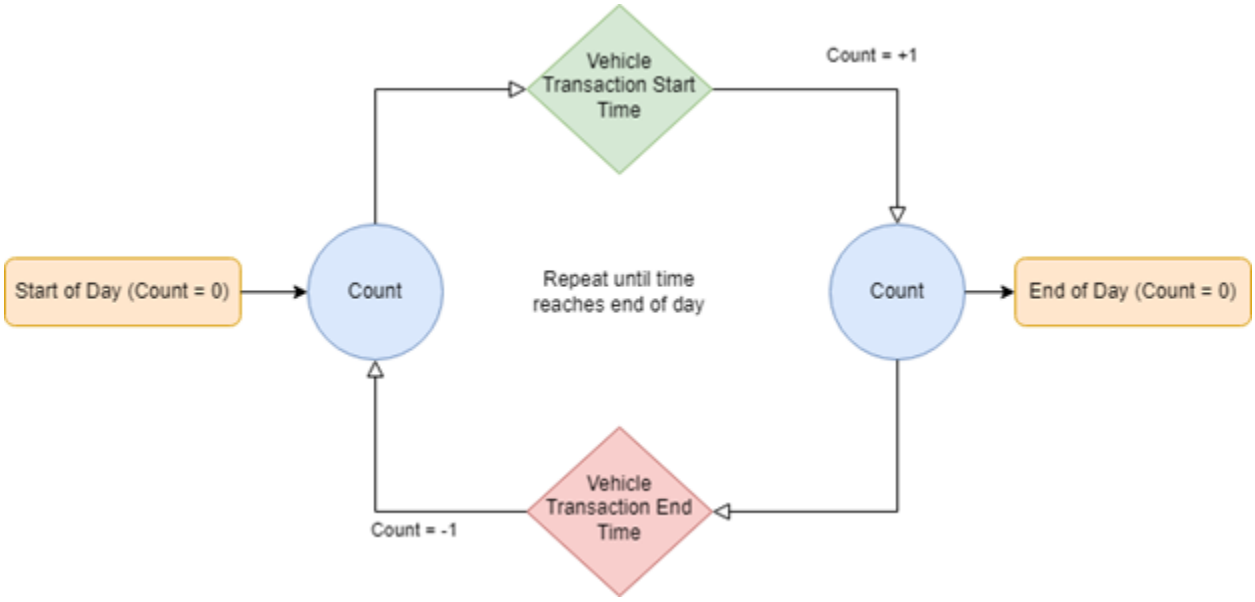


Figure 4.1: Occupancy Count Flow Diagram

Before manipulating the occupancy count data, it is also explained how graphs will be formatted and presented. The characteristics of the graphs are explained to avoid confusion before conclusions are made from the analysis. Figure 4.2 is an example of the graph format that will be presented in the data analysis. For the graph, the y-axis is the occupancy count (Count), while the x-axis represents the time (Hour) throughout a car park's hours of operation. Here, the maximum capacity (gathered through TPA capacity data) of the car park is also displayed and indicated with a horizontal red line. The title will also give specific details of what each graph represents. The example graph represents an occupancy count plot for location ID 3001 on 2019-06-21 with 1-minute intervals. Other graphs will have a similar format and be further explained to clarify the data represented by each graph.

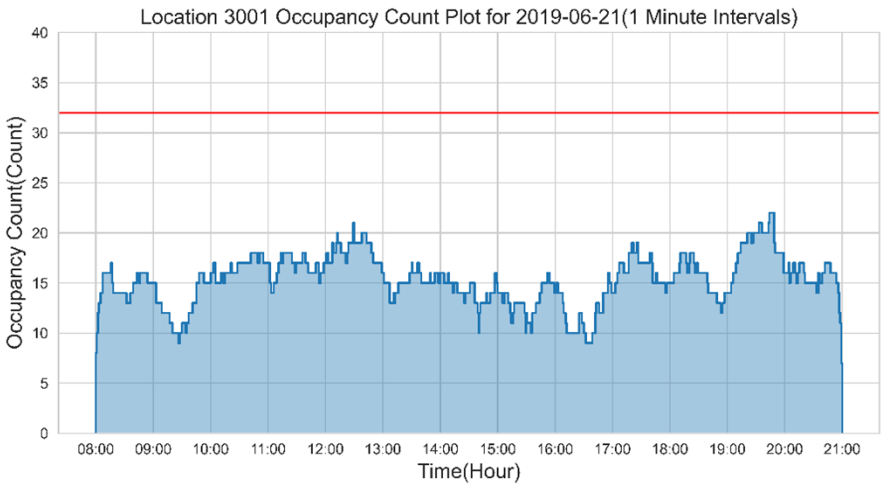


Figure 4.2: Example Graph Format

4.3 Time Intervals

To display meaningful data, the visuals must be clear and concise. Displaying occupancy count graphs on specific time intervals becomes a concern, as the data displayed can't be too fine but also not too coarse. A time interval represents the frequency of an occupancy count often made throughout the day. Three-time intervals were investigated, and the selected time interval represented all graphs. Figure 4.3 - Figure 4.5 compare the three time intervals of 1 minute, 15 minutes and 30 minutes. When comparing the three, the 1-minute time interval graph looks fine as it counts the occupancy by the minute. This makes it harder to find trends or patterns from our graph. The thirty-minute interval count graph shows that the occupancy count might sometimes be above the occupancy capacity. However, this is not the case, as from the 1-minute interval, the count never goes above capacity, making the 30-minute interval data too coarse. The 15-minute interval occupancy count represents the data well, as the trends are apparent within each hour of operation, unlike the 1-minute interval. In contrast, the data is not so coarse that the data inflates too high. From the conclusions, the graphs will be displayed by 15-minute interval counts.

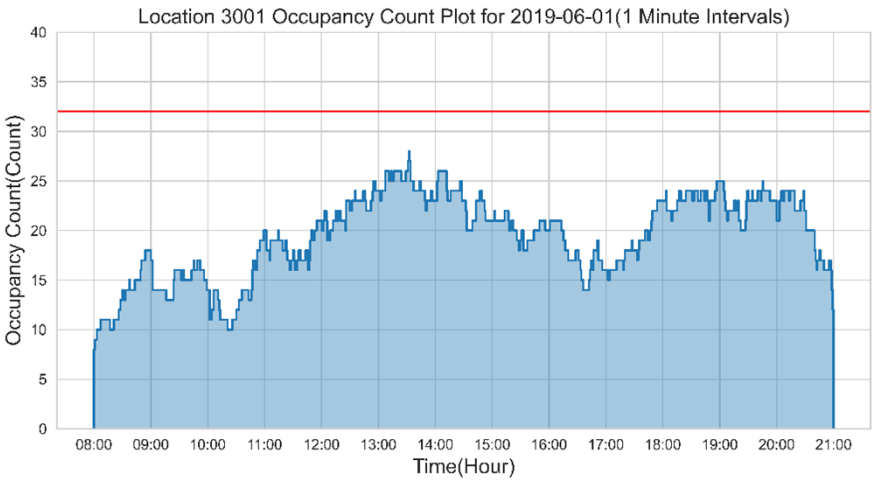


Figure 4.3: Occupancy Plot with 1 Minute Intervals

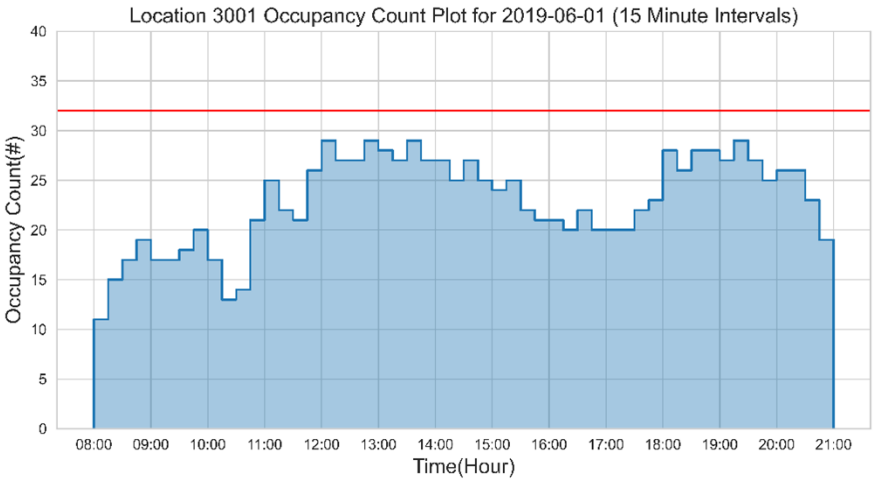


Figure 4.4: Occupancy Plot with 15 Minute Intervals

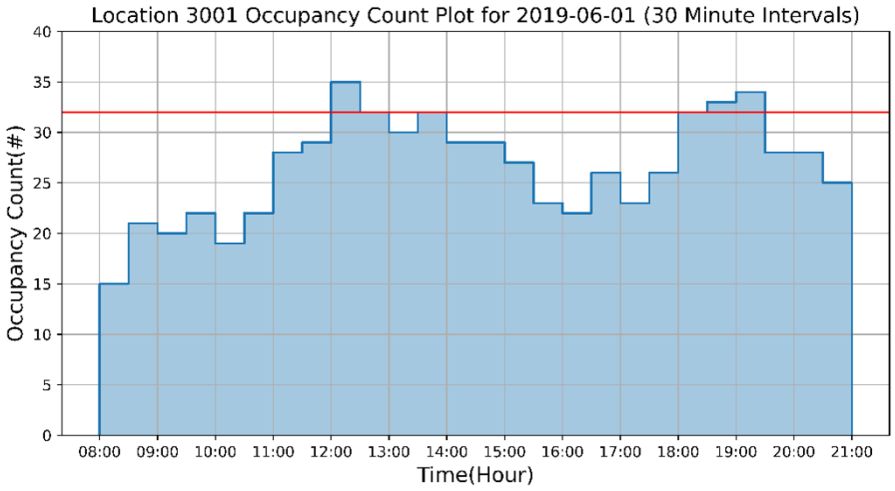


Figure 4.5: Occupancy Plot with 30 Minute Intervals

4.4 Data Analysis (Real Street Comparison Example)

The following section provides an overview of real sample scenarios highlighting the potential of dynamic parking pricing to optimize utilization—addressing underused parking locations in some areas while mitigating congestion in others. Figure 4.6 and 4.7 (15 Minute Intervals) showcases a case study involving Toronto St (Location ID 4111) and The Esplanade (Location ID 4131). Despite their proximity there is merely a 7-minute walk apart—the occupancy rates between the two streets are similar, while a $1 discrepancy in parking rates exists. However, both locations maintain occupancy within the ideal range of 60-80% (this percentage was used in the SFPark and ParkDC pilots [5] [6], suggesting that current pricing is effective and does not necessitate adjustments.

Toronto St. (4111):

- 25 Spaces
- $5.00/hr
- Average Occupancy = 46.85%

The Esplanade (4131):

- 29 Spaces

- $4.00/hr
- Average Occupancy = 49.38%

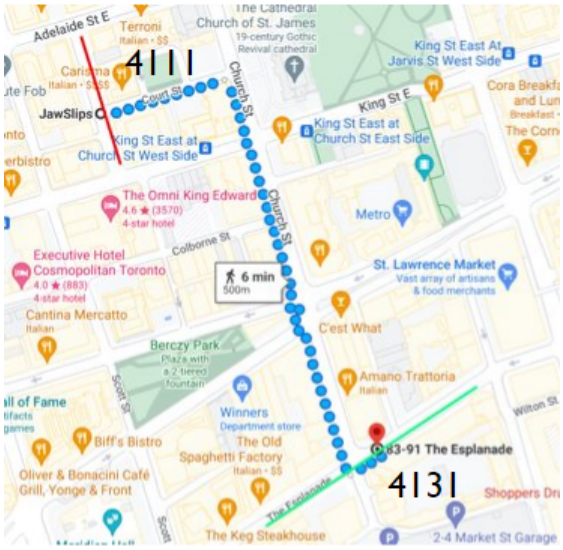


Figure 4.6: Toronto St vs The Esplanade Walking Distance Map

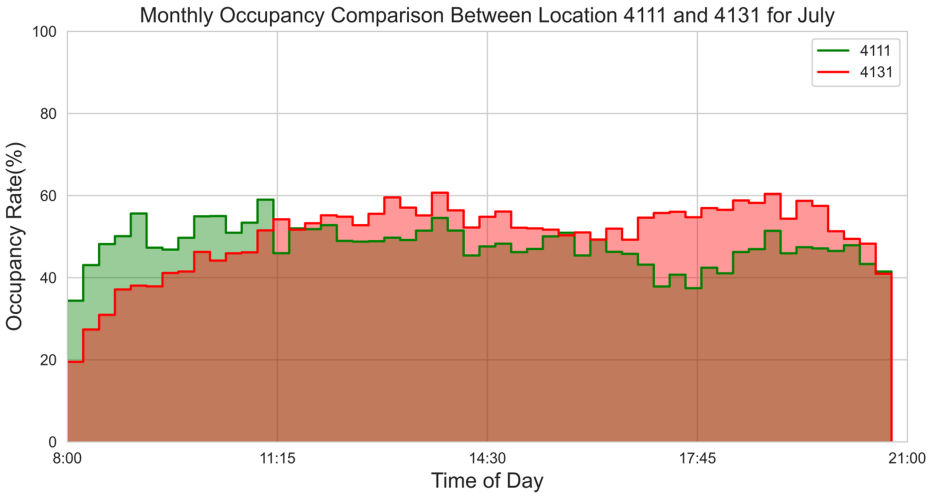


Figure 4.7: Toronto St vs The Esplanade Occupancy Graph

Conversely, Figure 4.8 and 4.9 (15 Minute Intervals) examines two adjacent parking areas on Wellington St, locations ID 4120 and 4121. Despite their close vicinity, there is a marked disparity in their occupancy rates. Location ID 4120, with a higher rate of $5/hr, achieves 20% occupancy, whereas Location ID 4121, at $4/hr, frequently experiences high occupancy. This disparity indicates a clear opportunity for dynamic pricing: reducing rates at location ID 4120 or increasing rates at location ID 4121 could balance occupancy levels more effectively.

Wellington St E (4121):

- 14 Spaces
- $4.00/hr
- Average Occupancy = 66.18%

Wellington St E (4120):

- 27
- $5.00/hr
- Average Occupancy = 11.51%

Figure 4.8: Wellington St E Walking Distance Map

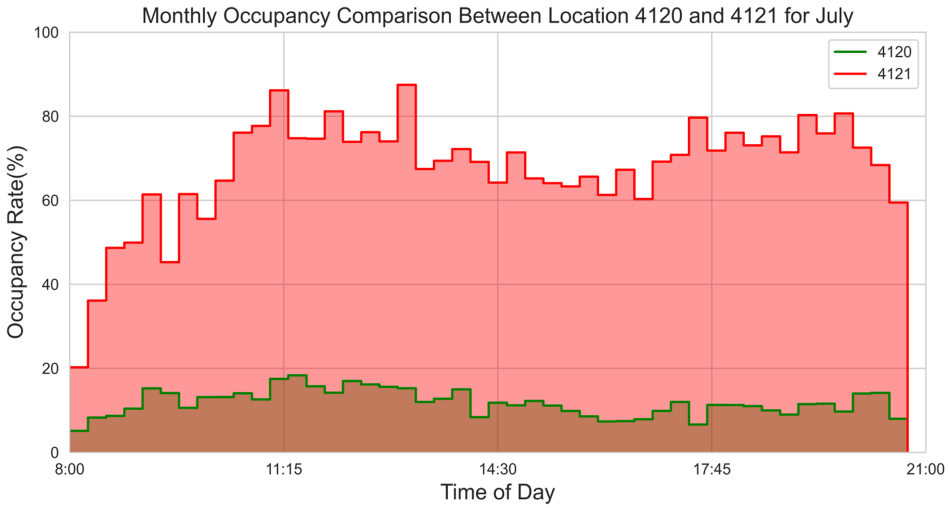


Figure 4.9: Wellington St E Occupancy Graph

4.5 Day-of-Week and Time-of-Day Analysis

Understanding different demand patterns is essential to creating an accurate neural network, as labelling these patterns helps the neural network understand these patterns before making a prediction. The day of the week is essential when predicting occupancy, as each day has different demands. Figure 4.10 compares the average occupancy rate between the weekday and weekend times for all locations. There are noticeable differences during the peak times of days, with more occupancy parking during the weekday mornings and higher occupancies during the afternoon and evening peaks for the weekend. Figure 4.11 further incorporates the time of day when comparing weekdays to weekends, and the results show that from Monday through Friday, work-related activities drive the morning occupancies higher compared to the weekend. Social events influenced an increase in occupancy during the afternoon and evenings on Saturdays and an overall decrease in occupancy rate during all times on Sundays.

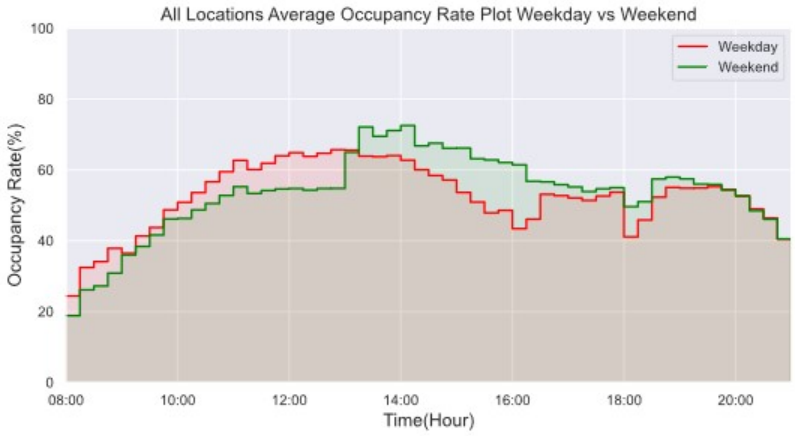


Figure 4.10: All Locations Average Occupancy Rate Plot Weekday vs Weekend

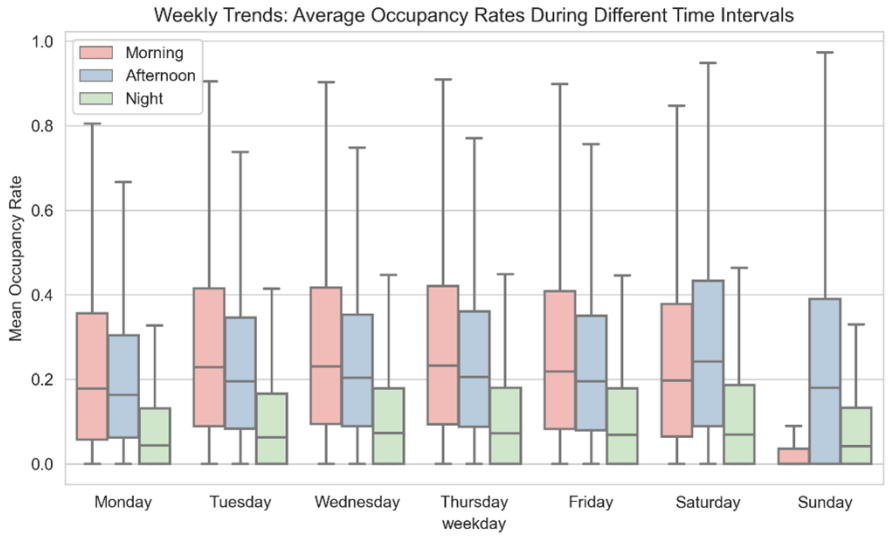


Figure 4.11: Weekly Trends, Average Occupancy Rate Comparisons for Different Time Intervals

4.6 Parking Rate Analysis

From section 4.5, the day-of-week and time-of-day exhibited importance in occupancy demand patterns, with each having significant changes to occupancy depending on what drivers are more likely to do at a specific time of day. Parking rates are another main factor in determining if a driver wants to park at a specific location, as their total trip cost is increased or decreased based on the location's pricing. However, the transaction data suggest that higher parking rates do not always lead to lower occupancy rates. Figure 4.12 - Figure 4.14 show boxplots that compare the occupancy rates to the price per hour of all parking locations, with each figure having higher occupancy rates as the price increases. This observation hints at a spatial dimension inside the transaction data, where parking behaviour sustains high parking prices due to the parking location. Figure 4.15 illustrates this connection by mapping out and colouring each parking location based on the existing price. The prices highlight that the downtown core has the highest rates for parking, meaning drivers need to pay a higher price if they are to park within the downtown area.

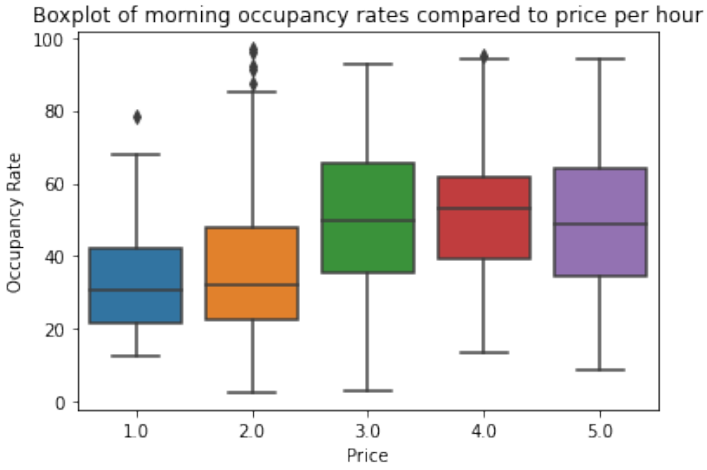


Figure 4.12: Boxplot of Morning Occupancy Rates Compared to Price per Hour

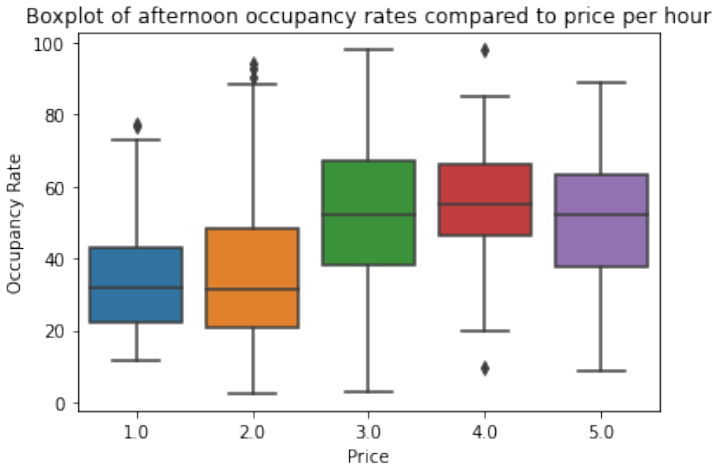


Figure 4.13: Boxplot of Afternoon Occupancy Rates Compared to Price per Hour

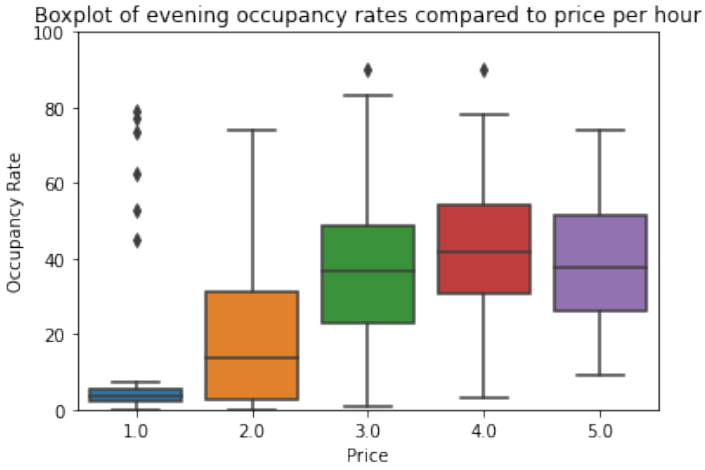


Figure 4.14: Boxplot of Evening Occupancy Rates Compared to Price per Hour

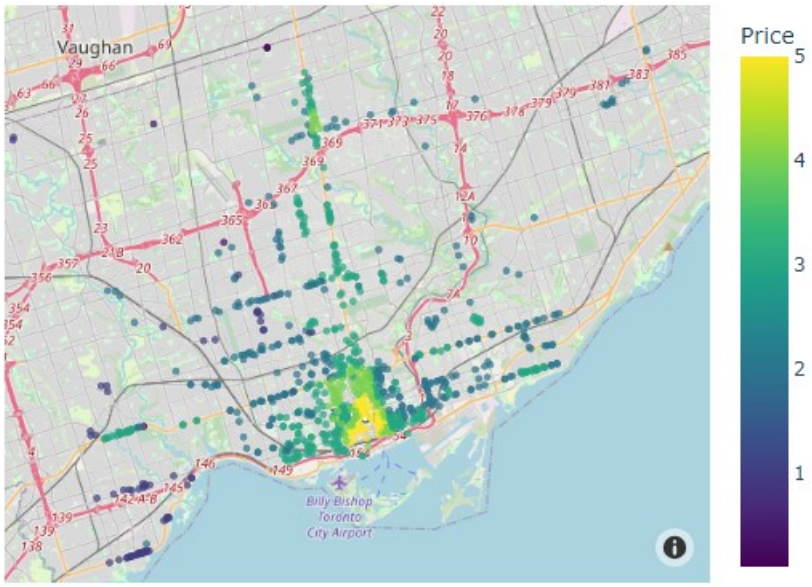
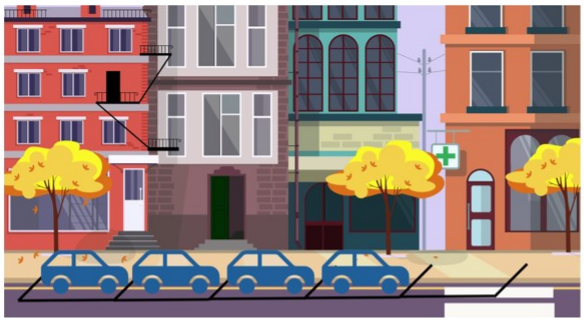


Figure 4.15: Parking Network Price Map

4.7 Spaces Open vs Occupancy Rate

Up to this point of the thesis, the term occupancy rate has been used to determine how open a parking area is. However, the occupancy rate needs to showcase how open or full a parking area is. An example can be seen in Figure 4.16; both (a) and (b) parking locations have an occupancy rate of 80%, but parking location (a) has only one space open compared to location (b) with two spaces open. With each location having different parking capacities, the occupancy rate fails to show the actual density for each parking area. For the remainder of the thesis, spaces open will be used to determine how open a parking area is, as it better indicates the density levels of each parking group.

(a)



Spaces Open: 1/5

Occupancy Rate: 80%

(b)

Spaces Open: 2/10

Occupancy Rate: 80%

Figure 4.16: Occupancy Rate vs Spaces Open Diagram

Chapter 5

Neural Network

As the data is cleaned and valuable insight is gained from the analysis of the raw transaction data, the next step is to use the data to predict daily spaces open using a neural network. (Neural Network (NN)) are multi-layer networks of neurons that can predict a variable based on specific inputs. NN has been used in previous works and has successfully predicted spaces open in different scenarios as discussed in section 2.4. In this section, the background of NN will be explained, and the NN generated in this thesis will be further discussed.

5.1 Background

A neural network is made up of a series of algorithm with the goal to find the underlying relationships in a set of data through a process that mimics that of a human brain, hence referring to a system of neurons, either organic or artificial [18]. By adapting to changes in the input data, the network generates the best possible outcomes without the need to adjust the output criteria. NN are used in many useful applications, and has high capabilities when used in forecasting research. Our research focuses on a similar problem of predicting occupancy's based on historical occupancy data. NN are composed of layers of nodes, with each node acting like a neuron inside the brain. Each node has an impact on the generated output, and effects the overall prediction of the NN.

NN architecture (Figure 5.1) consists of three types of layers:

1. Input Layer - where initial data is inputted for the neural network.
2. Hidden Layer - the layer/layers where computation is done to find relationships within the data.

3. Output Layer - produces the result from the given input layer.

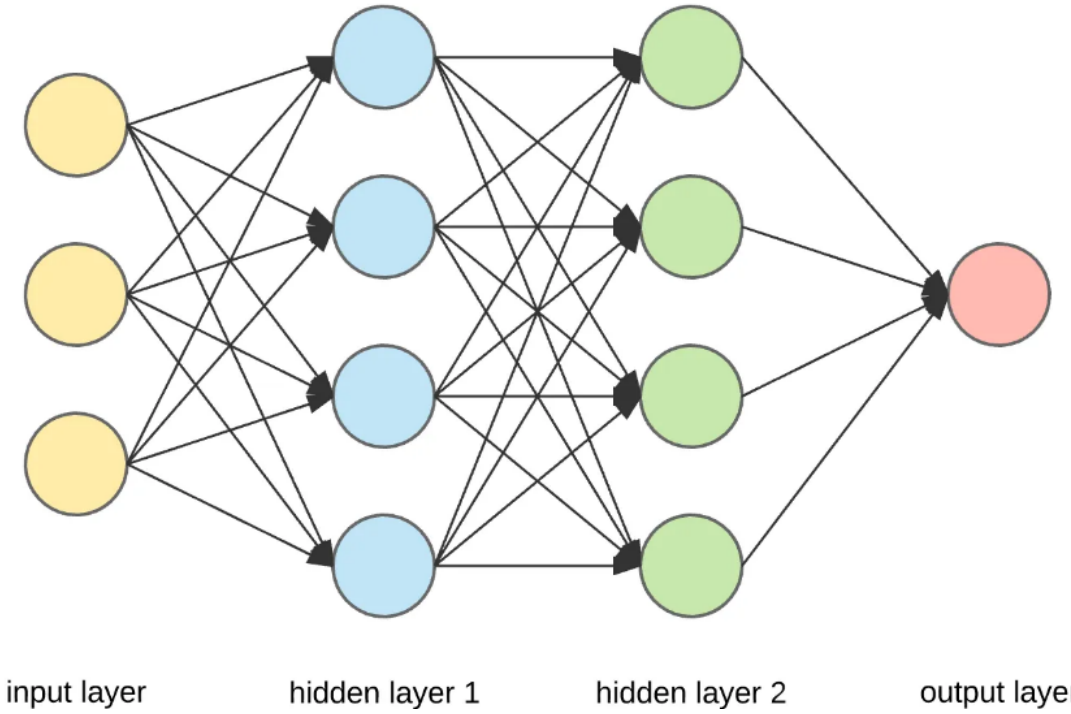


Figure 5.1: Neural Network Architecture [1]

Within the layers is made up of neurons, and each individual neuron has an impact on the output. Each neuron takes inputs and multiplies them by their weights. Weights are determined by how important each input is, with the neuron's goal to adjust these weights based on the given inputs and outputs [19]. Equation 5.1, 5.1 and Figure 5.2 [20] shows an example of how two inputs are used to calculate the hidden layers, which are then used to generate the output. With x representing specific input data, w representing a weight determined by the NN training, h determined by the solution of the input and weight, and y representing the output solution. The inputs are multiplied by the weights to get a vector, which is then multiplied by ones to calculate the result. Every layer's output becomes the following layer's input.

$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} = \begin{bmatrix} x_1 w_{11} + x_2 w_{21} \\ x_1 w_{12} + x_2 w_{22} \\ x_1 w_{13} + x_2 w_{23} \end{bmatrix}' = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix}' \tag{5.1}$$

$$\begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix}' \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = h_1 + h_2 + h_3 = y \tag{5.2}$$

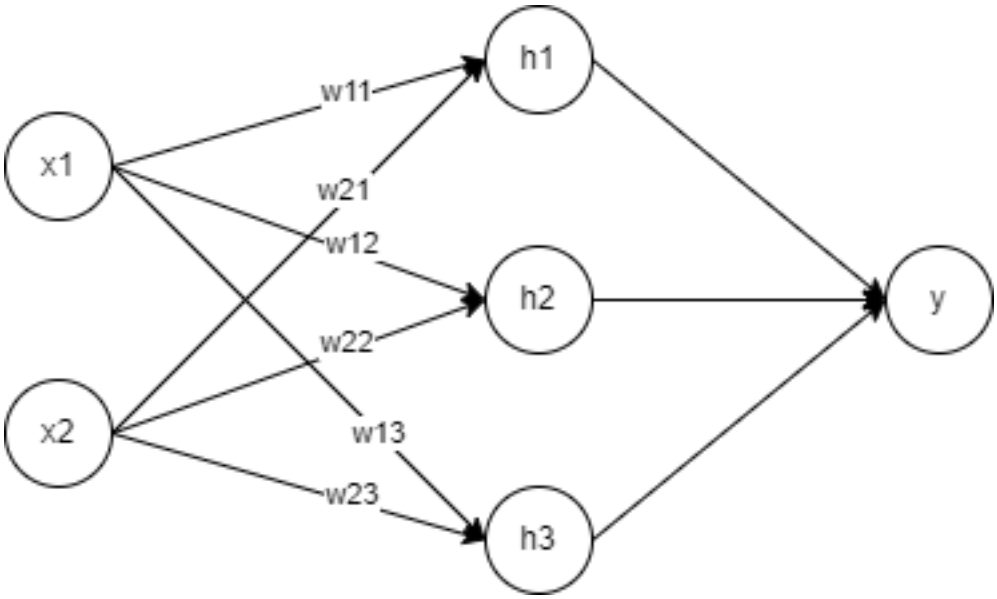


Figure 5.2: Neural Network Example Visualization

Each type of NN has a unique structure, within this thesis a feed-forward neural network is used, meaning the information travels in one-way only. No feedback or loops exist within the neural network, meaning the output of a layer is not affected by its previous layers. For the generated NN in this thesis, the input data consists of historical occupancy data, and multiple features that help characterize a parking location such as: Parking Price per Hour, Average Parking Price of Vicinity, Average Temperature, Ward Data, Points of Interest, Capacity Weight (Nearby Vicinity), and Month.

5.2 Features (Input Data)

In this section the features inside the input data are discussed. Every parking location in the parking network must be characterized inside the NN to help the NN learn and recognize patterns

within the data to make better predictions. Features characterize the input data and gives each location individual properties, which are used as inputs for the ML model. These features are essential to predict parking occupancy with higher accuracy. For the spaces open prediction model, the features are all numerical and represent one parking location.

5.2.1 Parking Price per Hour

Parking pricing is the main contributor to shifting the demand levels of all parking groups, as this factor determines the total travel cost of a driver. This is the most important feature inside the neural network as we change the price per hour of parking groups to make our predictions. The TPA sets the initial parking prices, and parking price changes are made to see the results of dynamic parking pricing.

5.2.2 Average Parking Price

As parking price is the main contributor to justifying a driver's decision to park at a location, another contributor is the parking prices around the driver's destination. Average parking pricing is a necessary feature, as it accounts for a parking group's attractiveness compared to nearby parking groups. This is an important feature, as we predict nearby parking by changing the average parking pricing value after a target parking group price change. The maximum distance threshold is set to 150m when determining the nearby neighbours. Equation 5.3 is used to find the average parking price of the parking location, with A denoting the average parking price, T denoting the target parking location's parking price, N being the neighbouring locations' parking prices and n representing the number of nearby neighbours.

$$A = \frac{T + \sum N}{1 + n} \tag{5.3}$$

5.2.3 Average Temperature

Inputting the average temperature for a day is an essential feature as it distinguishes the seasonality of the parking transaction. For example, parking occupancies follow different demand trends in the winter compared to summer, as drivers are more likely to pay for a premium parking spot for a shorter travel time. Inside the neural network, the average temperature has a more negligible impact on prediction, as the raw transaction data only ranges in the May, June, and July months, meaning different seasons of the year will not be accounted for.

5.2.4 Wards

The study area covers much of the GTA, making the parking areas disperse. This means the data needs to be categorized based on the parking location. Ward data is inputted as a feature to solve this problem, as they are labelled based on neighbourhoods inside Toronto. Labelling each parking group a ward number categorizes the areas to account for demand trends. For example, the average parking occupancy rate in the Etobicoke area (Parkdale-High Park Ward) is lower than the downtown core (Toronto Centre Ward). Through distinguishing wards, the neural network can account for demand differences. Figure 5.3 represents the Toronto municipal ward map, labelling each ward with a number. Each parking group is labelled inside the neural network with a ward number corresponding to the location.

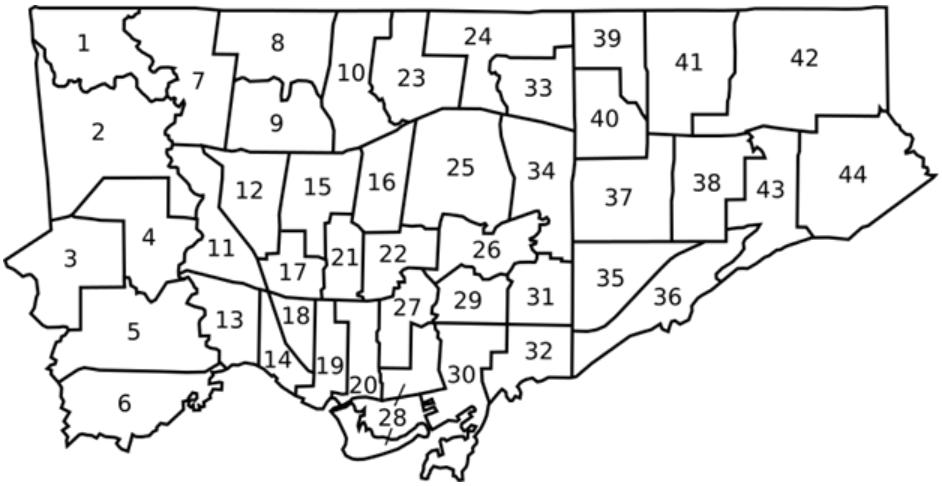


Figure 5.3: Toronto Municipal Ward Map [**?**]

5.2.5 Points of Interest

Helping the neural network understand each parking group's characteristics increases the accuracy of occupancy predictions. Integrating point-of-interest data becomes imperative to enhance the effectiveness of neural network predictions. Point of interest refers to a parking location's nearby attractions that appeal to drivers or a statistic that gives insight into the location. These attractions give a reason for drivers to park at a specific location, and inputting point of interest data will help distinguish parking groups. The point of interest data is sorted into two major categories. Amenities and statistics. For the amenities, a numbered score is given to each parking group; each number indicates how many points of interest are at the nearby location, with a distance threshold set to 150 m. The statistic point of interest category indicates the amount nearby for a given statistic. Each point of interest category will be explained further below [21]:

- Recreational (Amenity) - Recreational amenities include places that drivers go for leisure, whether it be social, cognitive, or physical leisure's, all are considered for this category. Examples include bars, cafes, food courts, pubs, and restaurants.
- Education (Amenity) - Education amenities are places for drivers that progress or acquire knowledge. Examples include colleges, universities, primary/secondary school, libraries, extra curricular schools.
- Bicycle (Amenity) - The bicycle category includes nearby bike share systems. This point of interest is highly related to parking, as it includes another means of transportation within a certain area.
- Public (Amenity) - Public points of interest include public buildings that service citizens. Examples include courthouse, fire station, police station, post offices, and townhall.
- Financial (Amenity) - The financial category for amenities equates to sources where drivers can either exchange, withdraw or deposit money. Examples include ATMs, banks, currency exchange stores and self payment kiosks.
- Health (Amenity) - Health amenities are locations where drivers can seek healthcare. Examples include clinics, dentists, hospitals, nursing homes, pharmacies, social facilities, and veterinaries.
- Parking (Amenity) - The parking category refers to nearby off-street parking locations. This category has high importance, as this effects the total capacity of parking spaces within the vicinity.
- Population 2022 (Statistic) - This statistic represents the amount of people living within the vicinity. This number is generated from 2022 population data.
- Occupation (Statistic) - The occupation POI represents the amount of jobs inside each ward.
- Worked at Home (Statistic) - This statistic represents the estimated number of people who worked at home near the parking location, and the importance amounts to the number of drivers that might have a parked car at the location due to working from home.
- Average Total Household Income (Statistic) - The average total household income statistic helps distinguish the wealth of families within the vicinity. This affects the demands for parking when parking prices are changed.

5.2.6 Capacity Weight

The parking capacity within a vicinity affects the behaviour of parking demand patterns, as drivers can park further away if a parking area is full. However, the distance between parking groups affects how willing a driver is to park further away from their destination. Using total capacity within an area as a feature ensures that by changing parking prices, drivers can still park elsewhere. The capacity is weighted to account for distances between parking groups. This numerical weight is calculated by taking the average capacity weight of nearby neighbours. The capacity weight is the relationship between distance and maximum capacity from a neighbour parking group to the main parking group. Equation 5.4 represents the equation to calculate capacity weight. C denotes the capacity weights, N represents the nearby neighbors' capacity, and D represents the distance between the main location and the neighbor.

$$C = \frac{\sum N * D}{\sum D} \tag{5.4}$$

5.2.7 Month

As important feature inputs help the neural network predict spaces open at higher accuracy, monthly data help the neural network learn trends and patterns that happen throughout the months of the year. Figure 5.4 showcases each month's spaces open on average. The May and June results are similar, while in July, there is more demand, leading to fewer spaces being open on average. Indicating which month each row of data represents will help the neural network understand the monthly parking trends inside the data.

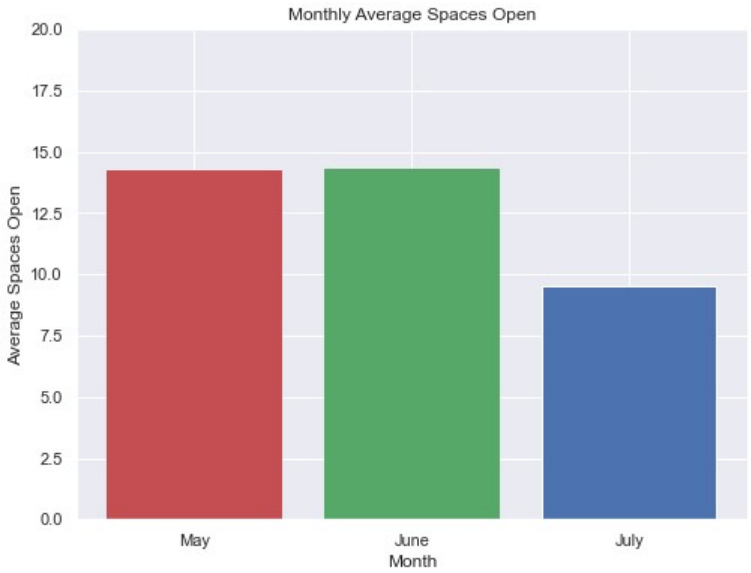


Figure 5.4: Monthly Average Spaces Open for All Locations

5.2.8 Feature Importance

Selecting the correct features helps the neural network learn the data more efficiently, resulting in more accurate predictions. Feature importance is a technique in machine learning that calculates a score for all input features of a given model [**?**]. A higher score means that a specific feature significantly affects the model when predicting a variable (spaces open). The important features are outputted to understand the data better and improve the model. Since a score is given to each feature, irrelevant features are removed from the model to increase predicting efficiency. Algorithm 1 represents how the permutation feature importance is done on an existing data set [22]. Figure 5.5 shows a feature importance graph that ranks the importance of each feature inside the neural network. The graph shows that the capacity weight and recreational data rank highest, while the average temperature and specific ward information rank the lowest. The average temperature feature importance score being low was concluded to be from the limitations of the data, as our data only ranges from May to June, and the daily average temperature is similar throughout these times. Looking at the results from the feature importance test, average temperature data was removed due to a low score.

Algorithm 1 Permutation Feature Importance [22]

At first, the MSE, and accuracy is calculated using the existing neural network model with all features denoted as metric M
for each feature F **do**
    Shuffle the values of F
    Make a prediction with shuffled values and calculate the chosen metric M_f
    Calculate the difference D_f between M and M_f for the feature F and save it
end for
Sort the D_f differences in order to get important features

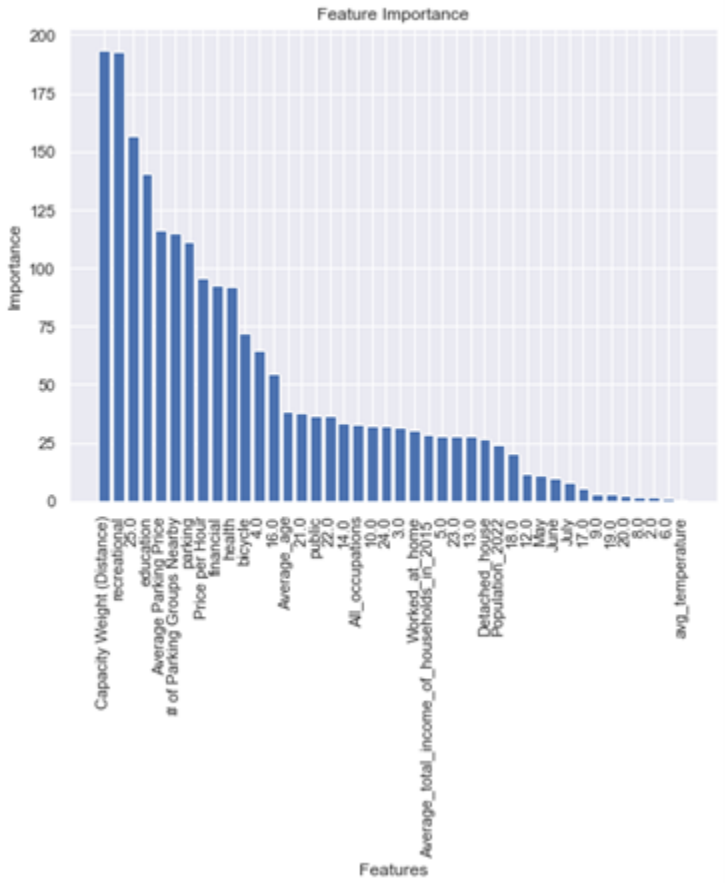


Figure 5.5: Feature Importance Graph

5.2.9 Neural Network Model Properties

Designing the neural network model involves interpreting the problem you aim to solve. Predicting the number of spaces open for a specific parking group involves a regression model, as we are predicting a continuous numerical value. The first step to modelling the NN is to decide on the architecture used. As discussed in Section 5.1 a feed-forward NN is used for the model in this thesis, and base parameters are chosen. These parameters are then tuned into making a more accurate model. Parameter tuning is a standard method used to decide the best NN architecture. It involves performing a grid search test to find the best-performing model based on set parameters. A grid search is a tuning technique that attempts to compute the optimum values of hyperparameters, with hyperparameters being how the learning process is influenced within a NN. Multiple sets of parameters are compared with each other and the best model is

selected based on a performance metric. This metric measures the effectiveness of the combined hyperparameters done within the grid search. Each model will be trained using a seventy to thirty percent split for the train and test data. The seventy percent train data is used to generate the model and the thirty percent test data is used to check if the predictions are feasible.

In this thesis the performance metric used is the mean squared error (MSE) metric, a risk function that measures the square of errors. As the target (occupancy rate) is seen to be normally distributed, MSE penalizes more significant errors compared to smaller ones [23]. MSE can be defined as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y - y^{\wedge})^2 \tag{5.5}$$

where n is denoted as the number of samples, y is the observed value, and ŷ is the predicted value. A grid search is performed using the GridSearchCV library inside Python to test the optimal number of neurons and hidden layers to choose for the neural network. From the results, the hyperparameters are set to appropriate values to get the best-performing model for prediction. GridSearchCV performs a process of hyperparameter tuning, which determines the optimal values for hyperparameters for a given model [24]. Specific parameters are predefined, and the grid search iterates through each set of parameters to find the best model. Figure 5.6 illustrates the grid search results inside a heat map, comparing the different MSE results of each set of parameters. The scale represents the range of scores depicted by a colour, with a lower score showcasing a poor MSE and a higher score representing a favourable MSE. The x-axis represents the number of neurons inside the NN, while the y-axis represents the number of hidden layers inside the NN. The range for the hidden layers are from 2 to 6 and the neurons selected to test are 32,64 and 128. These values are chosen as they fit in with the complexity of the problem, and testing past a certain amount of hidden layers and neurons results in over fitting [25].From the results of the heat map, it indicates that the best-performing model is a three-layer neural network with 64 neurons, and Table 5.1 shows results of each combination of parameters tested. A negative MSE is showcased as using an error function with the Python (sklearn) library, results with higher scores are ranked higher, which is the opposite case with MSE. The MSE is negative to easily output our best model inside a grid search.

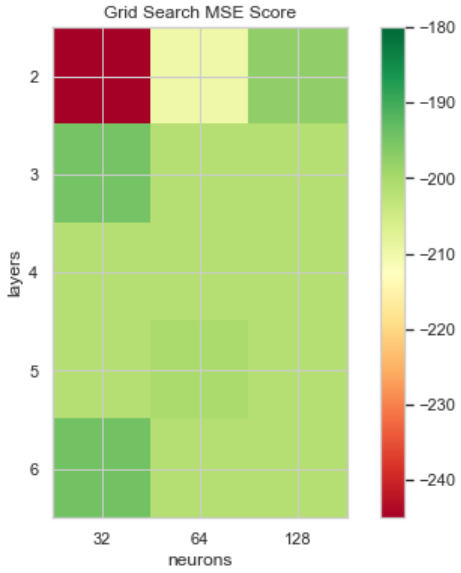


Figure 5.6: Gridsearch MSE Heat Map

Table 5.1: Gridsearch MSE Results

Layers	Neurons	MSE
2	32	-265.44
2	64	-209.89
2	128	-197.40
3	32	-201.37
3	64	-194.42
3	128	-201.36
4	32	-201.36
4	64	-201.39
4	128	-201.38
5	32	-201.43
5	64	-200.23
5	128	-201.38
6	32	-194.55
6	64	-201.41
6	128	-201.43

After finding the best-performing model through the grid search, two neural network training hyperparameters, batch size and epochs are performed using a grid search to find the best-performing model training settings. Batch size indicates the number of samples a neural network works through until the internal parameters are updated. At the same time, the epochs define the number of times the neural network will iterate through the training dataset before a final model [26]. It is also important to note that Minibatch stochastic gradient descent is used, as it can trade off convergence speed and computation efficiency, helping the model perform better in Python [27]. The grid search tested the batch size numbers of 16, 32 and 64, while the epochs tested are 100 and 150. These values are selected through a trial and error process that involves looking at a range of values to prevent under-fitting and over-fitting. From the grid search results in Table 5.2, the best neural network training hyperparameters use a batch size 16 and epochs of 150.

Table 5.2: Batch Size and Epoch Grid Search Results

Batch Size	Epochs	MSE Score
16	100	-18.41
16	150	-17.36
32	100	-18.34
32	150	-18.53
64	100	-19.13
64	150	-19.02

After fine-tuning all the hyper parameters from the grid search, the specifications of the final model are:

- Data Preprocessing Pipeline: Normalization: $\rightarrow$ StandardScaler
- Neural Network Model: Dense1 $\rightarrow$ ReLU $\rightarrow$ Dense2 $\rightarrow$ ReLU Dense3 $\rightarrow$ ReLU with dimensions of Dense1(64), Dense2(32), Dense3(1)
- Optimizer: ADAM Learning Rate (0.001), Weight decay (L2 regularization):1e-3

The final model starts with data preprocessing, and the input data is normalized using a StandardScaler which standardizes the features by removing the mean and scaling to a specific variance. As variables that are measured at different scales do not contribute equally to the model fitting, which might create a bias [28]. Standardization is a scaling technique wherein it makes the data scale-free by converting the statistical distribution of the data into the below format equation 5.6 [29]:

$$z = \frac{x - \mu}{\sigma} \tag{5.6}$$

where x is the observation, μ is the mean and σ is the standard deviation ($\mu = 0$,$\sigma = 1$).

The second step is the neural network model; three dense layers are used, each activating a rectified linear unit (ReLU). Dense layers are superficial neurons that receive all neurons from previous layers [30]. ReLU equips a deep learning model with non-linearity while resolving the problem of vanishing gradients and is the most popular activation function in deep learning [31]. The equation represents the ReLU:

$$f(x) = max(0, x) \tag{5.7}$$

where x is the observation, the ReLU will return a zero if any negative outputs are present. This generates feasible results from the neural network, as the spaces open can not be negative.

The last step is optimizing the neural network. An adaptive moment estimation (ADAM) is an iterative optimization algorithm used to minimize the loss function during the training of a neural network [32]. Also, the neural network is optimized using an L2 regularization weight decay, which addresses over-fitting. Over-fitting happens when a model tailors closely to a particular data set, making a model unable to generalize other data sets. L2 regularization tackles this by adding a "squared magnitude" of the coefficient as the penalty term to the loss function [33]. Below is the equation for L2 regularization:

$$L2 = (wx + b + y)^2 + \lambda w^2 \tag{5.8}$$

where x is the independent variable, y is the response variable, w is the weight, b is the bias, and lambda is the amount of shrinkage applied to the model.

5.3 Neural Network Prediction Examples

Compared to all other models, the NN is not flawless. The algorithm implements a design to discard any suggested price adjustments from the NN that do not align with expected economic principles — as price increases/decreases, demand should accordingly decrease/increase. For instance, Figure 5.7 shows a scenario where the algorithm accepts the NN's prediction. After a \$0.5 rate increase at Location 3702, marked by a yellow node, the result indicates a rise in open parking spaces—validating the prediction and aligning with the strategic goal to alleviate congestion in that area. The red line on the graph indicates the maximum capacity of location 3702; the green line represents the historical transaction data of the occupancy count (\$3); the

yellow line represents the prediction of occupancy for location 3702 at its original price of $3, and the blue line represents the prediction of location 3702 when the parking price increases to $3.50. From this example's current price, there are days where the average open spaces reaches 0 (completely full), and with a price increase the location repulses some vehicles away, increasing the available spaces.Similarly, Figure 5.8 presents a prediction for Location 5824, represented as a purple node, where a decrease in price by $0.5 leads to fewer open spaces. The yellow line represents the old price ($2) prediction compared to the new price prediction ($1.5), as the amount of open spaces on average is decreased. The prediction matches the intended outcome of drawing more vehicles to underutilized spaces.

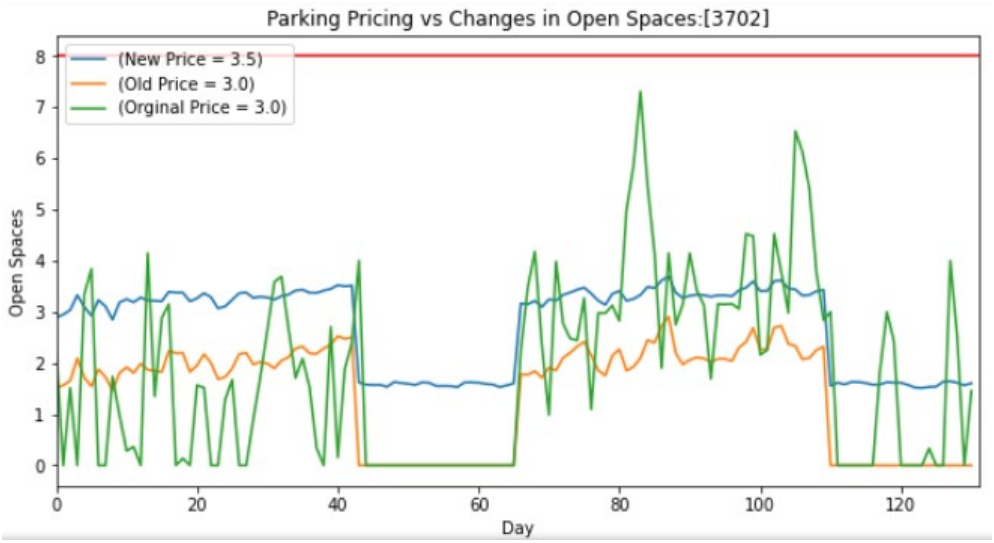


Figure 5.7: Price Increase Prediction

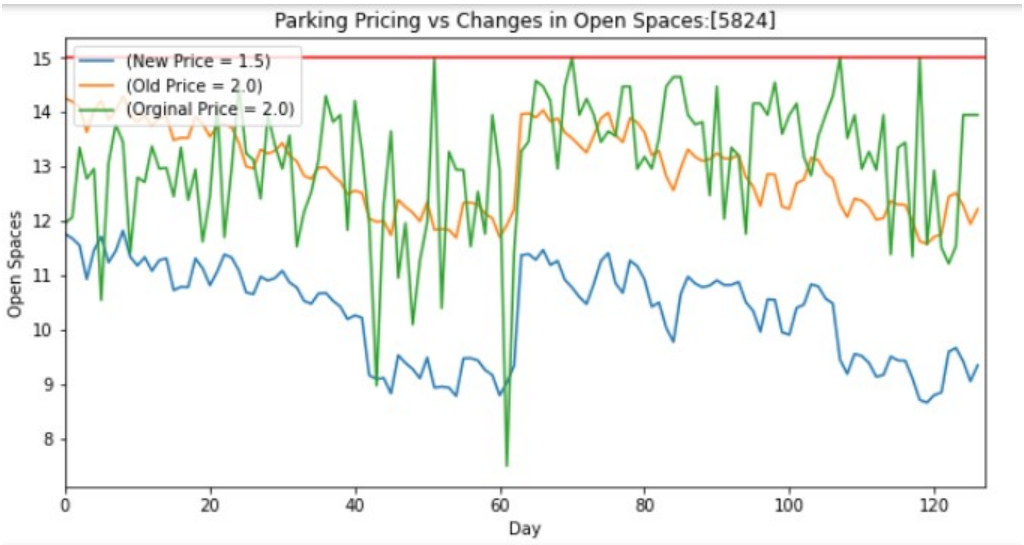


Figure 5.8: Price Decrease Prediction

Chapter 6

Algorithm

An algorithm was developed for a dynamic parking pricing system. This chapter discusses, in order, an overview of the algorithm, algorithm design, and results expected from the algorithm.

6.1 Overview of Algorithm

Inside a dynamic parking pricing system, the spaces open are required to update parking prices and make further pricing changes based on how drivers respond. When using transaction data to do dynamic parking pricing, an algorithm was developed to predict new occupancy based on parking price changes. Inside a dynamic parking pricing system, we must be able to obtain occupancy (spaces open), update parking prices and make further pricing changes based on how drivers respond. When using transaction data to do dynamic parking pricing, an algorithm must be developed as predicted occupancy (spaces open) is used to make parking pricing changes. The first step of the algorithm is to find the parking groups that need change; these are either parking locations where the number of spaces open is zero or close to max capacity. Next, that location will have a parking price change (increase/decrease), affecting the occupancy of the targeted location and nearby parking locations (neighbours). These open spaces are then predicted using the neural network, and the algorithm makes parking price changes until no more changes are necessary.

6.2 Algorithm Work Flow

In this section, the algorithm's workflow will be explained. Figure 6.1 illustrates the algorithm's workflow, which starts by initializing a default parking network. Each parking group is linked

to its neighbours (within 150m), allowing the algorithm to extract spatial information such as parking price levels and occupancy capacity. The algorithm then classifies each parking group based on its existing occupancy level as a coloured node: a location with an average of fewer than one space open is marked red; those with spaces open between one and three are marked yellow; underused parking locations that have an average of less than one space taken is marked blue, and those with spaces taken between one and three are marked purple. All other parking locations are marked green, as these represent the locations where the amount of spaces open is acceptable.

Target candidates (parking locations) are identified and sorted into a list. The first candidate on this list is proposed with a rate change of $0.50, depending on the node's colour (increase if yellow/red and decrease if purple/blue). After the price change, the number of spaces open is predicted using the neural network. If the prediction is logical (increase in price equates to increase in spaces open or a decrease in price equates to a decrease in spaces open) then we keep the iteration and the number of spaces open for neighboring nodes are also updated. The parking location price change is skipped if the prediction is not logical, and the next target candidate is chosen. The process iterates until all appropriate adjustments are made, culminating in a network that reflects optimized pricing across the parking landscape.

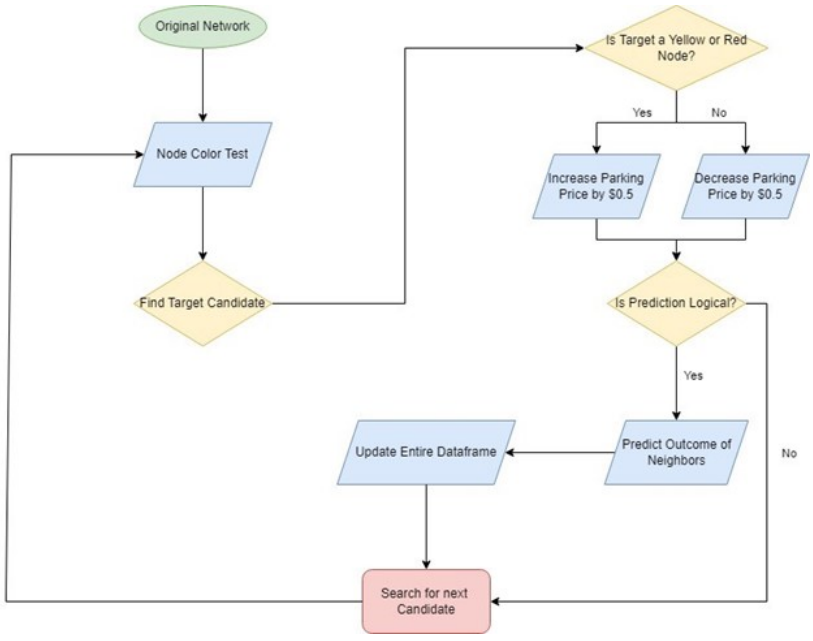


Figure 6.1: Algorithm Work Flow Chart

6.3 Algorithm Design

After building the neural network in section 5, the algorithm is designed to change parking prices to high/low-density parking groups to level the occupancy. Algorithm 2 describes how the node colour test is done; this test helps determine which parking groups require a price change. After the test is done, the entire parking network is coloured. Algorithm 3 represents the next step of making parking price changes to yellow, red, purple and blue nodes. The price change is set at \$0.5, whether an increase or decrease, with a maximum cap of \$1 in price change for every parking node. The final step is to make occupancy predictions on price-changed nodes, and Algorithm 4 represents how it is implemented. If a prediction is logical, the neighbouring parking occupancy is also predicted. The algorithm finishes with an entirely updated parking network.

Algorithm 2 Node Colour Test

At first, each parking groups' average spaces open A is gathered into a data frame , the data frame is iterated through to colour each group as a coloured node. Note: Parking groups with less than 5 parking capacity are exempted. *mincap* is set to 5.

for all A in data frame **do**

Denote o as average spaces open, t as average spaces taken and c as maximum capacity.

if $1 < o \leqq 3$ and $c > mincap$ **then**

Yellow Node

else if $o < 1$ and $c > mincap$ **then**

Red Node

else if $1 < t \leqq 3$ and $c > mincap$ **then**

Purple Node

else if $t < 1$ and $c > mincap$ **then**

Blue Node

else

Green Node

end if

end for

Algorithm 3 Price Change

As each parking node is given a colour , a list l of yellow, red, purple and blue nodes are iterated through and a price change is made.Note: Nodes without neighbors are exempt from price change.

for all nodes in list l **do**

Denote o as original average spaces open, p as price, c as change in price and P as predicted average spaces open.

if $o < 3$ **then**

c = 0.5

p + c ▷ Price Increase as High Density Node

else

c = -0.5

p + c ▷ Price Decrease as Low Density Node

end if

end for

Algorithm 4 Prediction and Neighbor Prediction

A list L of parking nodes that have changed prices will be iterated through to predict new average spaces open, and the prediction will be accepted or rejected due to logic of change.

for all nodes in list L **do**

Denote c as change in price, o as original average spaces open,p as predicted average spaces open, and P as predicted neighbour average spaces open.

if $c > 0.5$ **then**

Make prediction p using neural network. ▷ Price Increase Prediction

if $p > o$ **then**

Make predictions of neighbouring parking groups P

Updated entire data frame

else

Skip node as prediction is not logical

end if

else

Make prediction p using neural network ▷ Price Decrease Prediction

if $p < o$ **then**

Make predictions of neighbouring parking groups P

Updated entire data frame

else

Skip node as prediction is not logical

end if

end if

end for

Chapter 7

Results

7.1 Neural Network Model

From Section 4, the preliminary analysis revealed significant discrepancies in the demand levels of parking groups at different times and days of the week. To account for these patterns, the neural network models are split up into their respective categories, and the results of each neural network model can be seen in Table 7.1. All models perform well, with the weekend models showing poor results due to a smaller data set. Both training and testing MSE are shown to prove that overfitting does not occur within the models, and all models have accurate performing predictions [34]. The models with AUC ¿ 0.7 are considered acceptable [35], with the morning and evening weekend models underperforming. AUC provides an aggregate measure of performance across all possible classification thresholds [36] to rank the predictions of the models.

Table 7.1: Neural Network Models Results

Neural Network Model	Training MSE	Testing MSE	AUC
Morning Weekday	4.25	4.30	0.722
Afternoon Weekday	4.16	4.20	0.734
Evening Weekday	4.48	4.56	0.724
Morning Weekend	6.61	6.85	0.66
Afternoon Weekend	5.16	5.28	0.72
Evening Weekend	4.76	5.19	0.69

7.2 Algorithm

In this section each model's algorithm results will be shown and discussed. A total of six algorithm results are shown to see the performance of the algorithm on each existing parking network.

7.2.1 Weekday Algorithm

7.2.1.1 Morning Weekday Algorithm

The algorithm's application to morning pricing adjustments led to a 32% reduction in moderately occupied nodes (yellow) and an 8.3% reduction in nodes with a moderate level of free spaces (purple), as shown in Table 7.2. The algorithm increased the availability of parking spaces, as reflected by the rise in green nodes. These adjustments contributed to increased highly available parking spaces, represented by green nodes.

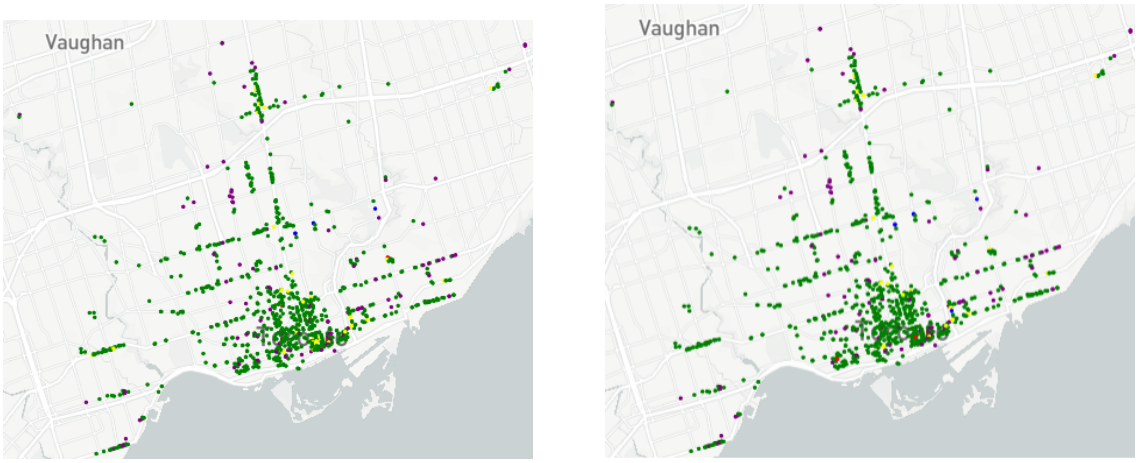
As presented in Table 7.3, analysis of the parking locations indicates that the majority required a minor adjustment, with a price change of $0.5 to achieve the desired occupancy levels. Figure 7.2 visually compares the parking network before and after implementing the algorithm. It highlights a shift in the distribution of yellow nodes, particularly within the downtown core, where the concentration of yellow nodes was initially highest. The after-image shows the change in occupancy rate in these nodes, further demonstrating the algorithm's effectiveness in aligning prices with demand during the morning period.

Table 7.2: Morning Weekday Algorithm Results

Network	Green Nodes	Yellow Nodes	Red Nodes	Purple Nodes	Blue Nodes
Original	826	37	4	60	2
Updated	843	25	4	55	2

Table 7.3: Morning Weekday Location Price Change Results

Locations with one price change	Locations with two price changes
3006, 3501, 3702, 4102, 4109, 4125, 5308, 5512, 6011, 7920, 7919, 8203, 8412, 4124, 7105, 4362, 4630, 3902, 3202	3810, 3107, 3812, 4359



(a) Morning Weekday Original Network (b) Morning Weekday Updated Network

Figure 7.2: Morning Weekday Parking Network Original vs Updated

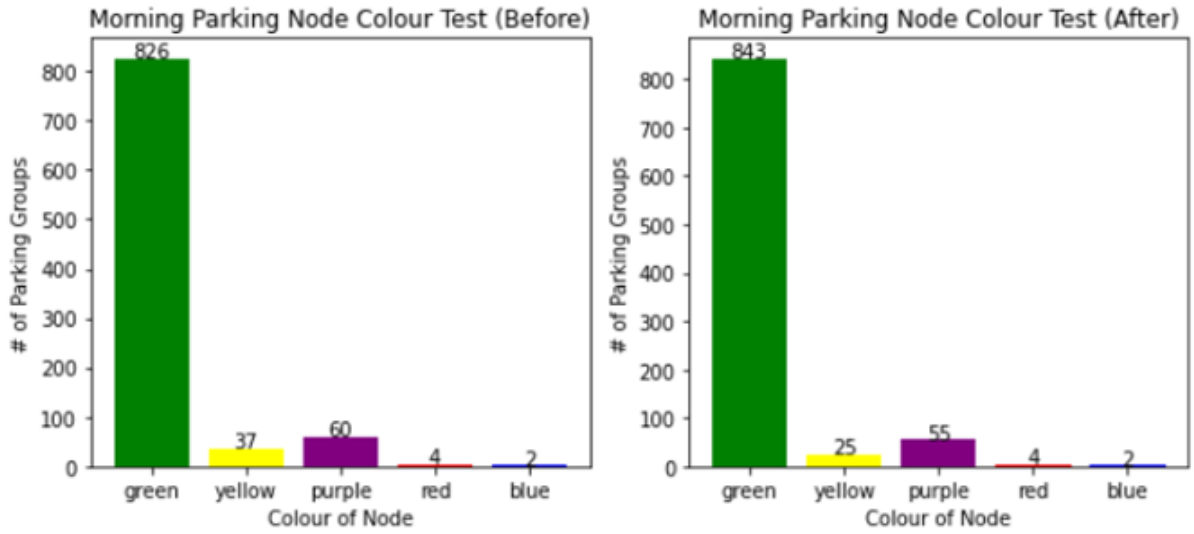


Figure 7.1: Morning Weekday Result Graph

7.2.1.2 Afternoon Weekday Algorithm

Applying the pricing algorithm to the afternoon pricing scheme resulted in a 32% reduction of moderately occupied nodes (yellow), while the highly occupied (red) nodes remained unaffected.

There was a 7.8% decrease in the purple nodes and no change in the least occupied (blue) nodes. As indicated in Table 7.5, a $0.5 rate adjustment was sufficient for all 16 evaluated parking locations to transition into well-occupied (green) nodes.

Figure 7.3 illustrates the parking network's layout before and after the rate adjustments, showing the shift in node colours. These adjustments contributed to a more efficient use of the parking network, alleviating congestion in high-demand areas and increasing utilization in less-occupied zones. Figure 7.4 illustrates the parking network before and after the algorithm is run. The downtown core and Yonge St. show the best improvements, as several yellow nodes turn green after running the algorithm.

Table 7.4: Afternoon Weekday Algorithm Results

Network	Green Nodes	Yellow Nodes	Red Nodes	Purple Nodes	Blue Nodes
Original	839	38	2	51	4
Updated	855	26	2	47	4

Table 7.5: Afternoon Weekday Location Price Change Results

Locations with one price change	Locations with two price changes
3014, 3810, 4122, 4310, 5307, 5308, 5310, 5535, 7913, 8203, 7105, 3107, 5823, 4633, 5824	None

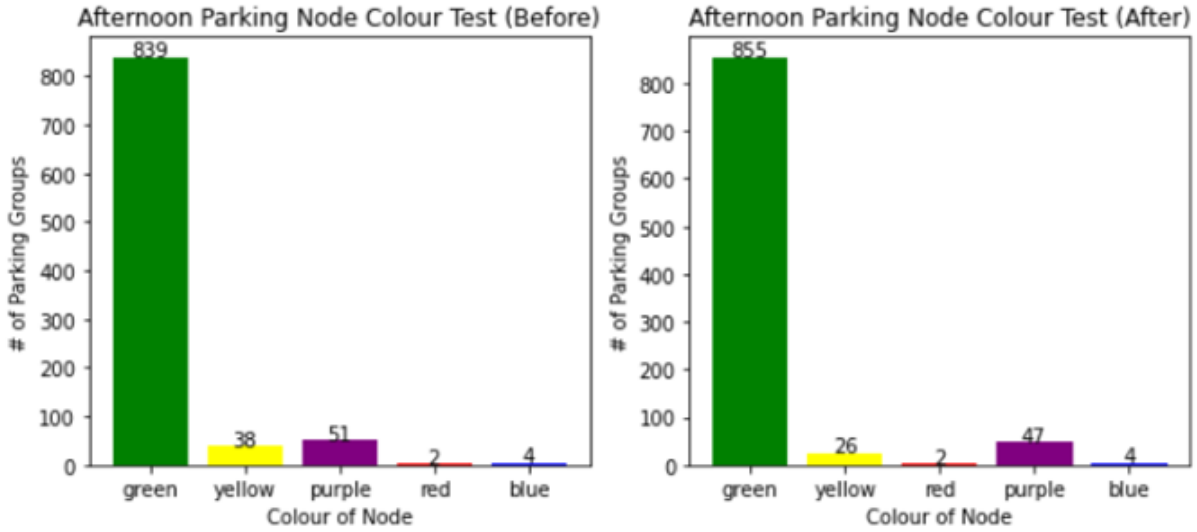
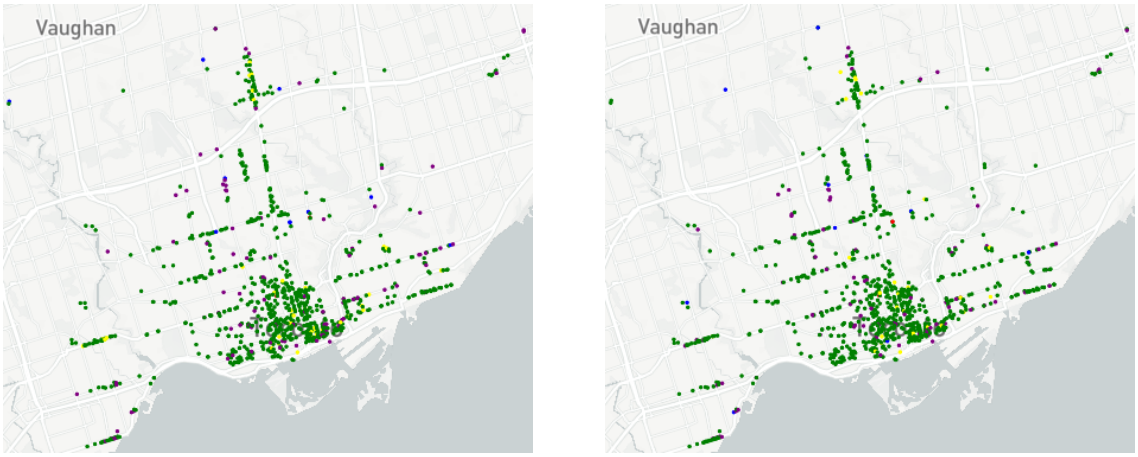


Figure 7.3: Afternoon Weekday Result Graph



(a) Afternoon Weekday Original Network (b) Afternoon Weekday Updated Network

Figure 7.4: Afternoon Weekday Parking Network Original vs Updated

7.2.1.3 Evening Weekday Algorithm

The evening algorithm presents different results from the morning and afternoon, as the parking network has more low-occupied parking areas in the evening. Table 7.6 shows that no yellow and red nodes are turned green after the algorithm. However, the main changes happened to the low-density parking groups, as the purple and blue nodes had a noticeable reduction. 13 purple nodes turned green, and 11 blue nodes turned green, improving the utilization of these parking groups. Figure 7.6 shows that the changes to nodes happen throughout the entire parking network, with each area of the map having a reduction in purple and blue nodes.

Table 7.6: Evening Weekday Algorithm Results

Network	Green Nodes	Yellow Nodes	Red Nodes	Purple Nodes	Blue Nodes
Original	708	4	0	91	75
Updated	731	4	1	78	64

Table 7.7: Evening Weekday Location Price Change Results

Locations with one price change	Locations with two price changes
4122, 7920, 5006, 8703, 7603, 6014, 7805, 3806, 3802, 7601, 6708, 8106, 5215, 3604, 3506, 7605, 3711, 3605, 4103, 4102, 7803, 5313, 6817, 3402, 4120, 7503	6706, 3902, 4006

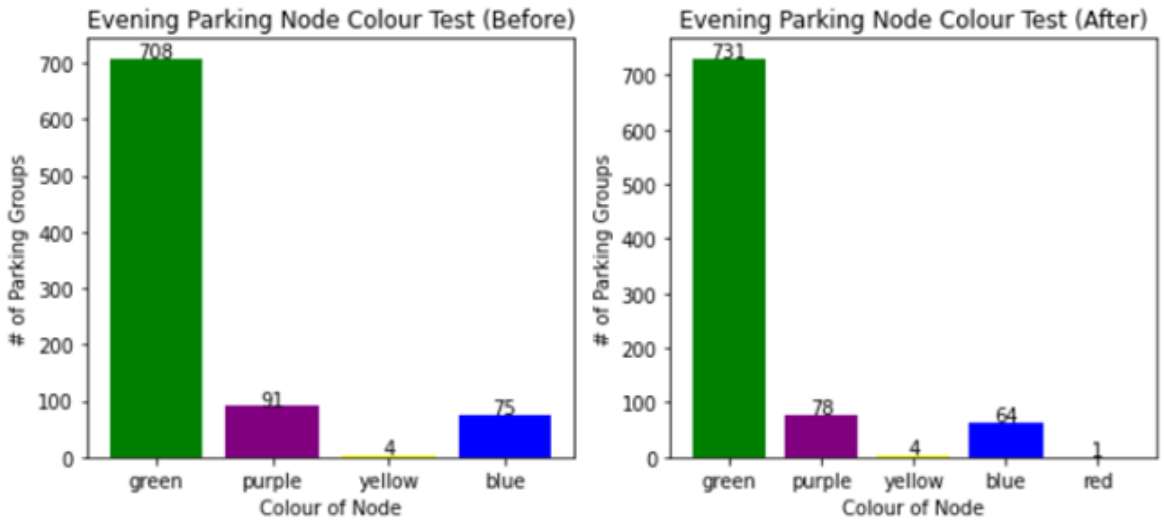
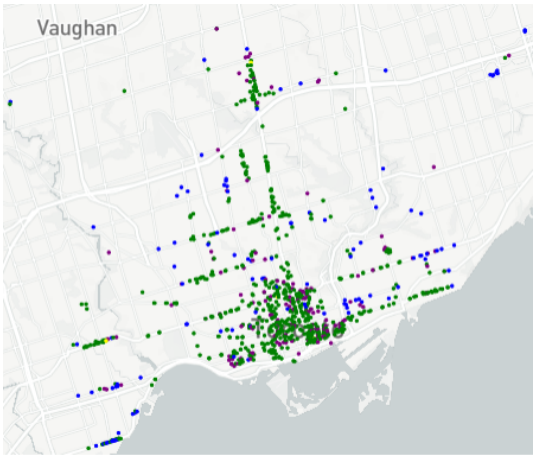
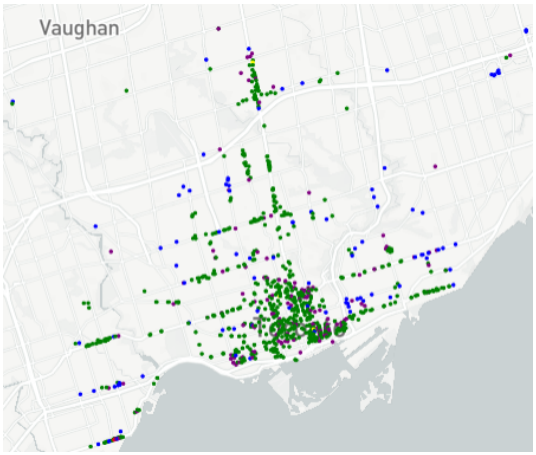


Figure 7.5: Evening Weekday Result Graph



(a) Evening Weekday Original Network



(b) Evening Weekday Updated Network

Figure 7.6: Evening Weekday Parking Network Original vs Updated

7.2.2 Weekend Algorithm

7.2.2.1 Morning Weekend Algorithm

Applying the pricing algorithm to the morning weekend pricing scheme resulted in a minimal reduction of moderately unoccupied nodes (purple), while the low-occupied (blue) nodes increased. There was a 4.9% decrease in the purple nodes. The changes were marginal even with multiple locations having price changes as indicated in Table 7.9, as analyzed in Section 4.5, the mornings during the weekend tend to see the lowest occupancy, this hints towards price adjustments, not having a significant impact on driver parking patterns.

Table 7.8: Morning Weekend Algorithm Results

Network	Green Nodes	Yellow Nodes	Red Nodes	Purple Nodes	Blue Nodes
Original	834	6	2	81	3
Updated	835	6	2	77	6

Table 7.9: Morning Weekend Location Price Change Results

Locations with one price change	Locations with two price changes
3613, 4630, 3721, 3709, 4103, 4102, 7105, 6023, 3203, 3205, 5825, 5818	3202

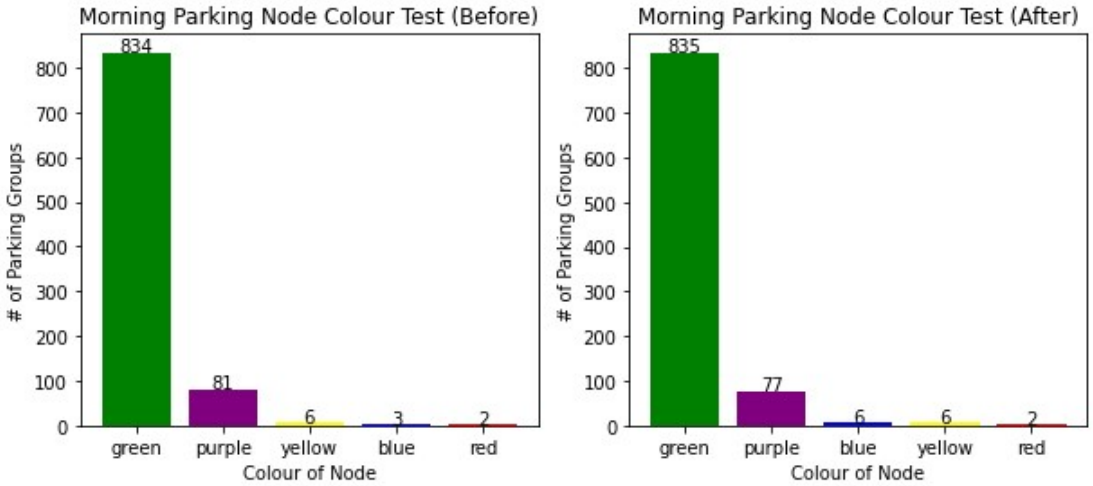
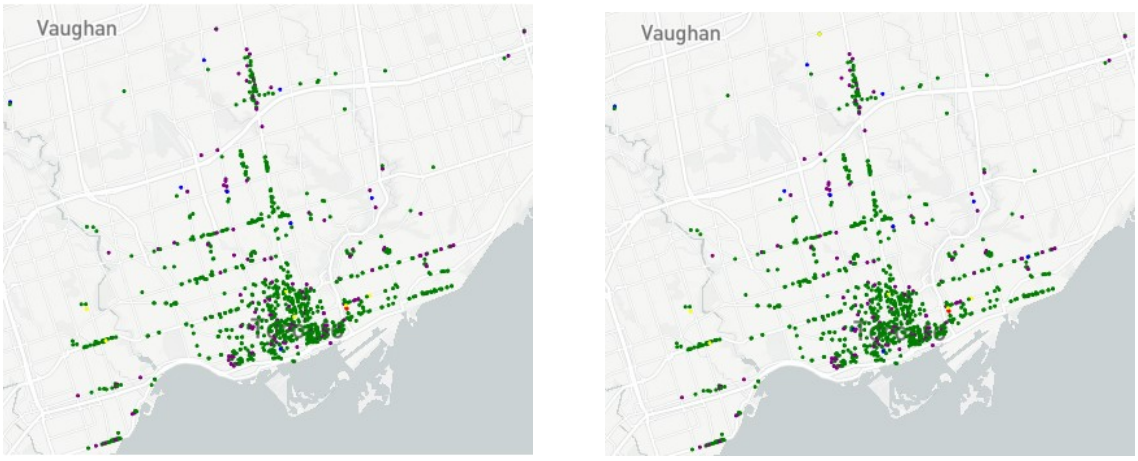


Figure 7.7: Morning Weekend Graph



(a) evening Original Network (b) evening Updated Network

Figure 7.8: Morning Weekend Parking Network Original vs Updated

7.2.2.2 Afternoon Weekend Algorithm

The algorithm's application to afternoon weekend pricing adjustments led to a 3.3% reduction in moderately occupied nodes (yellow) and a 10.9% reduction in nodes with a moderate level of free spaces (purple), as shown in Table 7.10. The algorithm increased the availability of parking spaces, as reflected by the slight rise in green nodes. Like the morning weekend model, the algorithm adjusted prices to many locations, but slight improvement was seen. This indicates that price adjustments rarely impact drivers' demands for high-density parking areas on weekends.

Table 7.10: Afternoon Weekend Algorithm Results

Network	Green Nodes	Yellow Nodes	Red Nodes	Purple Nodes	Blue Nodes
Original	799	60	6	55	3
Updated	807	58	5	49	4

Table 7.11: Afternoon Weekend Location Price Change Results

Locations with one price change	Locations with two price changes
3811, 3908, 4324, 5328, 3703, 4630, 6708, 4633, 5504, 7807, 4634	4632, 7805

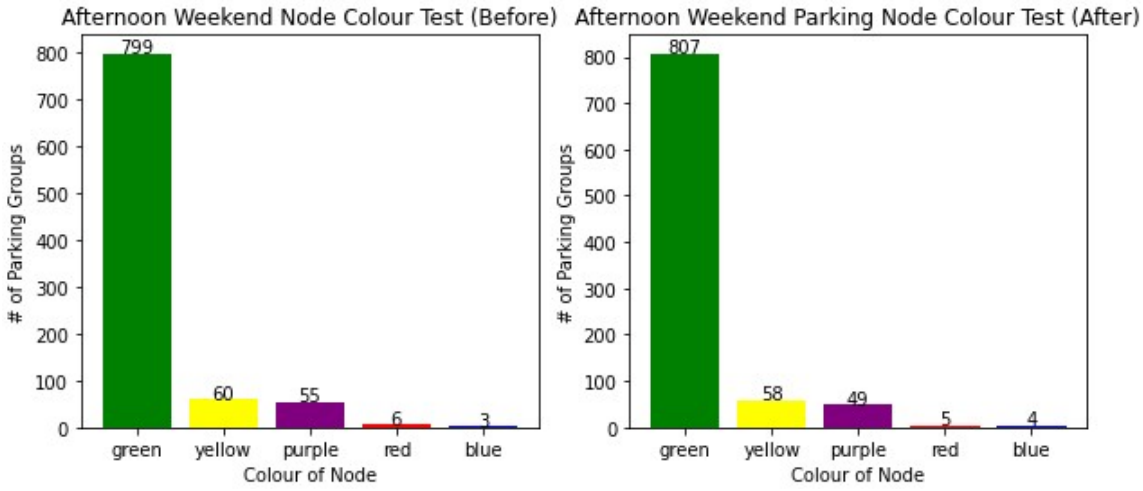
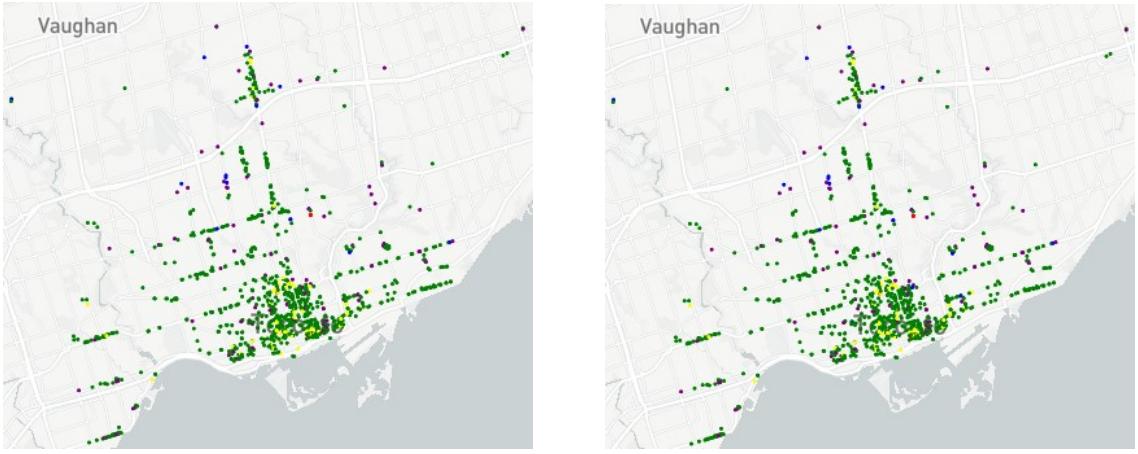


Figure 7.9: Afternoon Weekend Graph



(a) Afternoon Weekend Original Network (b) Afternoon Weekend Updated Network

Figure 7.10: Afternoon Parking Network Original vs Updated

7.2.2.3 Evening Weekend Algorithm

Finally, the algorithm's application to evening weekend pricing adjustments led to a 29.48% reduction in moderately open nodes (purple) and an 8.57% reduction in nodes in highly open nodes (blue), as shown in Table 7.12. The algorithm increased the demand levels for parking

spaces, as reflected by the rise in green nodes. These adjustments contributed to the increased usage of underutilized parking groups. The evening model performed the best out of the three-weekend models, accounting for most location price changes shown in Table 7.13.

Figure 7.12 visually compares the parking network before and after implementing the algorithm. It highlights a shift in the distribution of purple throughout the parking network. The after-image shows the change in occupancy rate in these nodes, further demonstrating the algorithm's effectiveness in aligning prices with demand during the evening weekend period.

Table 7.12: Evening Weekend Algorithm Results

Network	Green Nodes	Yellow Nodes	Red Nodes	Purple Nodes	Blue Nodes
Original	611	10	0	78	37
Updated	636	12	1	55	32

Table 7.13: Evening Weekend Location Price Change Results

Locations with one price change	Locations with two price changes
4122, 6014, 3806, 5538, 3809, 6708, 4630, 4633, 4632, 3721, 3407, 3701, 7502, 8201, 4634, 5216, 7906, 6101, 5312, 3406, 3412, 3719, 3801, 4130, 6406, 3720, 5108, 8205, 4909, 3711, 6017, 4803	3726, 3304, 3212

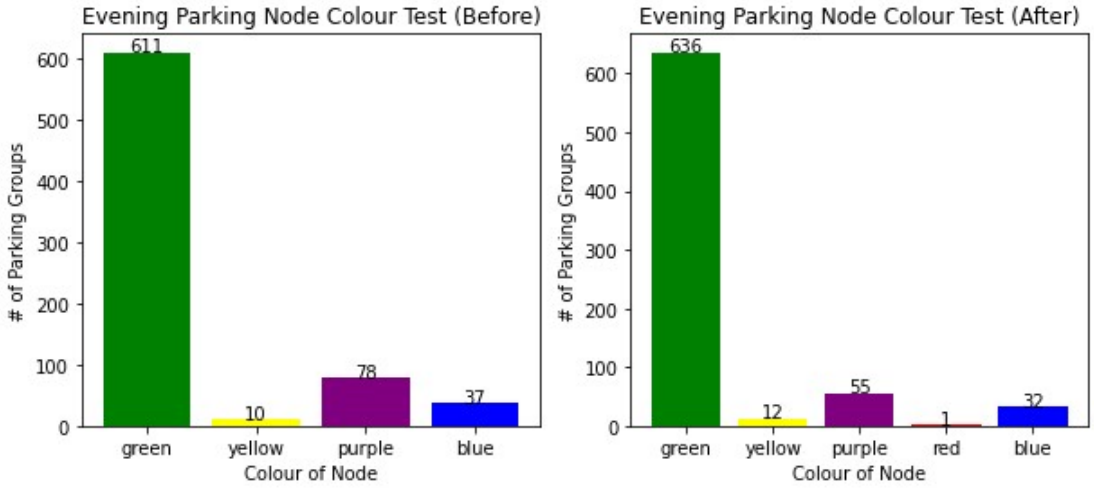
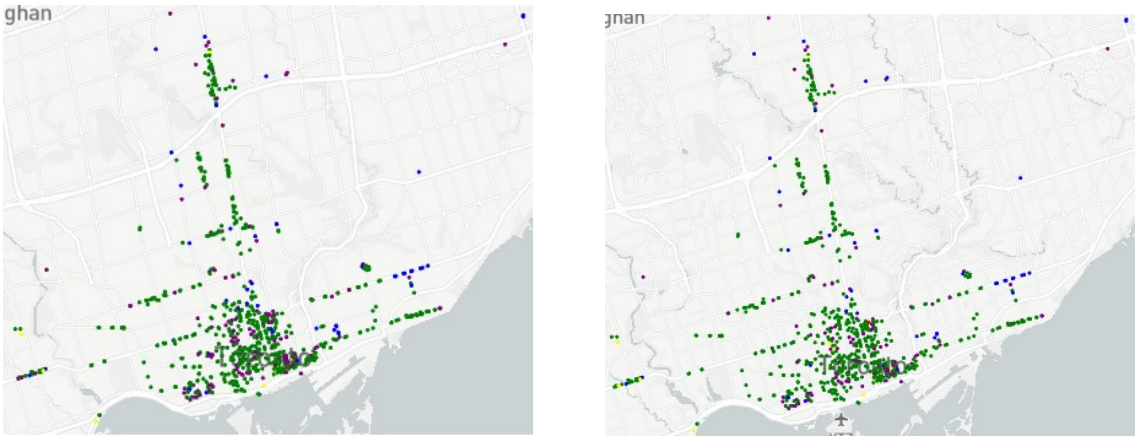


Figure 7.11: Evening Weekend Graph



(a) Evening Weekend Original Network (b) Evening Weekend Updated Network

Figure 7.12: Evening Weekend Parking Network Original vs Updated

Implementing a dynamic parking pricing policy algorithm positively affected the existing parking network, with a 17% increase in parking areas opening during the morning weekdays, a 17% increase for afternoon weekdays, a 14% increase for evening weekdays, a 1% increase for morning weekends, a 4% increase for afternoon weekends and 20% increase for evening weekends.

Chapter 8

Conclusions

8.1 Summary

Managing on-street parking in densely populated urban areas is a multifaceted challenge due to the high demand for limited parking spaces. With parking areas becoming full, this prolongs the search time for drivers, adding congestion on the road. Implementing a dynamic parking pricing policy is an innovative approach to address this complexity. Dynamic parking pricing levels the occupancy of parking groups by adjusting parking prices according to demand to ensure parking spaces are open in popular parking areas while also attracting drivers to underutilized locations. Cities like San Francisco, Washington DC, and Los Angeles have seen success in dynamic parking pricing pilots with improvement to parking availability, reduction in parking search times, and an overall decrease in congestion. From the positive outcomes of the pilots, cities are now expanding the parking pricing policy within their cities. However, scaling dynamic parking pricing is expensive due to the cost of occupancy detection infrastructure. Cities often deploy sensor technologies to gather occupancy rates for price adjustments. Maintenance of these sensors becomes a concern, as some sensors see early battery failure, and electromagnetic interference, reducing occupancy rate accuracy. Using historical transaction data to predict parking occupancy becomes a viable solution to the dynamic occupancy detection infrastructure problem, as this data is widely available and manipulable. Installing occupancy detection infrastructure can be avoided, as a historical transaction data prediction model can generate future parking occupancy based on price changes. These predictions can then be used to further adjust parking prices dynamically. The successful impacts of a parking pricing policy would entail a parking network that has its parking occupancy balanced within a certain occupancy rate range to ensure that every parking location is being fully utilized. In this thesis, the scope is to use historical transaction data to predict parking occupancy within a parking network, and make suitable parking pricing adjustments to balance the supply and demand of all parking locations.

Preliminary analysis was conducted on the data to gain insight into trends and patterns within the transaction data. The analysis revealed that the day of the week impacted demands, as drivers tended to park at different locations depending on whether it was a weekday or weekend. Demand levels were higher during weekends than on weekdays, suggesting the need to create different prediction models for the weekdays and weekends. This also suggested that parking prices would also have to be adjusted according to demand levels for a specific day (weekend or weekday). Another factor that affected demand patterns was the specific time of day, as demand levels varied during morning, afternoon, and evening hours. These peak hours changed the demand levels within a day, suggesting the need to account for different demand levels at a certain time of day. Consequently, the prediction models are segmented into specific day of week and time of day models, to increase the accuracy of occupancy rate prediction.

A neural network was generated that accounted for existing trends found in the preliminary analysis along with different features that categorize each parking location inside the parking network. The features helped characterize each parking location's trends demand levels, to improve overall prediction accuracy. Features like nearby parking price, capacity weight, and points of interest helped the neural network understand each locations properties. Each neural network model produced a AUC greater than 0.7 which confirms acceptable occupancy predictions.

The final algorithm mimics a dynamic parking pricing system, where the price adjustments are optimized to increase the parking price in high-occupied parking areas and decrease the price in low-occupied parking areas. A node test determines the candidates that require a price change. Each parking group is labeled a colour depending on the average parking occupancy. A high-occupied parking group is labelled red when the number of spaces open on average is less than one and labelled yellow when the number of spaces open on average is between one and three. A low-occupied parking group is labelled blue when the number of spaces taken on average is less than one and labelled purple when the number of spaces taken on average is between one and three. A parking group is labelled green if the number of spaces open on average is greater than three spaces taken and three spaces open. The algorithm modifies the price of a target candidate at each step and predicts the resulting average open spaces. As the target price change impacts surrounding parking groups, we additionally predict the average number of open spaces for nearby parking groups. The algorithm operated through iterations, that made a parking price change for every iteration. The algorithm ended when all possible changes were made. Results showcase the overall effects of implementing the dynamic parking pricing policy into the network.

Overall, implementing a dynamic parking pricing policy algorithm positively affected the existing parking network, with a 17% increase in parking areas opening during the morning weekdays, a 17% increase for afternoon weekdays, a 14% increase for evening weekdays, a 1% increase for morning weekends, a 4% increase for afternoon weekends and 20% increase for evening weekends. A recommendations is made to parking authorities based on the results of the research completed. The encouragement to parking authorities is to consider implementing a dynamic parking pricing policy scheme inside the areas of cities where parking occupancy is high, and

where parking occupancy are low. A suitable location for the City of Toronto is in the downtown core where on-street parking occupancy tends to be high. Based on the implementations of using such a pricing system, a strategy can be developed to slowly implemented to tackle these areas, and with the cost-effectiveness of using historical transaction data for occupancy rates, this parking policy could be further scaled into different areas that deal with parking management problems.

8.2 Literature Contributions

The algorithm presents a dynamic parking pricing system using transaction data and changes parking prices according to the occupancy levels of all parking groups. Using transaction data distinguishes this study from the reviewed literature as it uses a cost-effective occupancy detection method. Despite existing neural network models that predict occupancy, few studies focus on using descriptive temporal and spatial data, like point of interest data, average neighbour parking price and nearby capacity limits. This thesis aims to contribute further to the literature on predicting parking occupancy using transaction data and build an algorithm that mimics a dynamic parking pricing system. With the development of the algorithm, parking management authorities will have a framework that can be followed to implement dynamic parking pricing in cities.

8.3 Research Limitations

The thesis used all the fine-level transaction data given by the TPA. However, the data did have some limitations that limited the study. Building an accurate neural network model requires good input data to help make better predictions. As mentioned earlier in Section 3.4, the data dated through May - July 2019 and May - July 2022. During this timeline, many changes to parking infrastructure were made to account for the COVID-19 pandemic. The city of Toronto implemented CurbTO to help small businesses stay open by turning on-street parking spaces into outside patios for dining [37]. Parking groups were removed, impacting the entire parking network, as less capacity equates to more demand at other locations. These parking demand patterns would affect the neural network's overall accuracy, as the network would not be able to account for these patterns.

Secondly, the data set has a three-month range, limiting the neural network prediction to only the summer season. With data ranging throughout the year, the neural network would be trained to account for seasonal trends. Ideally, a more extensive set of raw data would further the validity of the analysis.

8.4 Future Work

Multiple paths can be taken to further the topic discussed in the thesis. First, the prediction model could be improved, as the performance metrics indicate a moderately acceptable model. Other prediction models like LSTM, Convolutional Neural Networks, and Graph Neural Networks could be further explored to output a more accurate model. The input data could also be investigated to help the prediction model better understand the complexities within every parking network. The use of off-street parking data could also help a neural network better understand on-street parking trends as the amount of nearby total capacity increases, with price competition also becoming a factor. Merging transaction data and parking sensor data could also be examined to see how effective actual occupancy data could be in increasing the accuracy of transaction data prediction models. Even taking a more cost-effective approach, using existing payment citation data could be a strategy to fill the compliance rate gaps that are in transaction data.

In addition, for future studies, using transaction data to predict occupancy could lead to developing an on-street parking reservation system, where drivers can reserve a spot to park at a specific on-street parking location. If a cost-effective approach exists, that gives accurate real-time occupancy information, a reservation parking system could be implemented to reduce cruising and overall congestion in dense urban areas, as drivers could reserve a parking space before arriving at the destination. The information of accurate parking occupancy forecasts brings a wealth of opportunities in the parking management industry, as the parking trends and patterns could be investigated to introduce preventative measures to congestion and illegal parkers.

References

[1] G. Ognjanovski, "Everything you need to know about Neural Networks and Backpropagation — Machine Learning Made Easy...," June 2020.

[2] T. A. Litman, "Parking management," Mar 2023.

[3] R. C. Hampshire and D. Shoup, "What share of traffic is cruising for parking?," *Journal of Transport Economics and Policy (JTEP)*, vol. 52, p. 184–201, Jul 2018.

[4] M. Chester, A. Horvath, and S. Madanat, "Parking infrastructure and the environment," *ACCESS Magazine*, vol. 1, p. 28–33, Oct 2011.

[5] SFPark, "SFpark Pilot Program," Jan. 2013. Publisher: San Francisco Municipal Transportation Agency.

[6] Washington, "PENN QUARTER/CHINATOWN PARKING PRICING PILOT," 2019.

[7] Boston, "Performance Parking pilot," Nov. 2016.

[8] Seattle, "Paid Parking Rates - Transportation | seattle.gov," 2017.

[9] LosAngeles, "LA Express Park," May 2012.

[10] Calgary, "Calgary On-Street Parking Rates," 2022.

[11] Baltimore, "Demand-Based Parking Meter Rate Setting," Nov. 2017.

[12] S. Yang, W. Ma, X. Pi, and S. Qian, "A deep learning approach to real-time parking occupancy prediction in transportation networks incorporating multiple spatio-temporal data sources," *Transportation Research Part C: Emerging Technologies*, vol. 107, p. 248–265, Oct. 2019.

[13] J. C. Provoost, A. Kamilaris, L. J. J. Wismans, S. J. van der Drift, and M. van Keulen, "Predicting parking occupancy via machine learning in the web of things," *Internet of Things*, vol. 12, p. 100301, Dec. 2020.

[14] J. Li, J. Li, and H. Zhang, "Deep learning based parking prediction on cloud platform," in *2018 4th International Conference on Big Data Computing and Communications (BIGCOM)*, p. 132–137, IEEE, Aug. 2018.

[15] S. S. Ghosal, A. Bani, A. Amrouss, and I. El Hallaoui, "A deep learning approach to predict parking occupancy using cluster augmented learning method," in *2019 International Conference on Data Mining Workshops (ICDMW)*, p. 581–586, Nov. 2019.

[16] C. Pflügler, T. Köhn, M. Schreieck, M. Wiesche, and H. Krcmar, "Predicting the availability of parking spaces with publicly available data," Sept. 2016.

[17] TPA, "Toronto Parking Authority," Sept. 2021. Archive Location: Toronto, Ontario, Canada Publisher: City of Toronto Scroll: yes.

[18] J. Chen, "What Is a Neural Network?," Apr. 2023.

[19] A. Obuchowski, "Understanding neural networks 1: The concept of neurons," Apr. 2020.

[20] Turing, "Understanding Feed Forward Neural Networks in Deep Learning," Jan. 2023.

[21] R. Olbricht and M. Paulmann, "Overpass API," 2015. Publisher: FOSS@HFT.

[22] E. Zvornicanin, "What is feature importance in machine learning? — baeldung on computer science," Aug. 2022.

[23] A. Gupta, "Mean squared error: Overview, examples, concepts and more — simplilearn," Feb. 2023.

[24] G. L. Team, "Hyperparameter tuning with gridsearchcv," May 2023.

[25] M. Uzair and N. Jamil, "Effects of Hidden Layers on the Efficiency of Neural networks," in *2020 IEEE 23rd International Multitopic Conference (INMIC)*, pp. 1–6, Nov. 2020. ISSN: 2049-3630.

[26] J. Brownlee, "Difference Between a Batch and an Epoch in a Neural Network," Aug. 2022.

[27] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, "Dive into Deep Learning," June 2021. arXiv:2106.11342 [cs].

[28] S. Loukas, "How Scikit-Learn's StandardScaler works," July 2023.

[29] S. Mulani, "Using StandardScaler() Function to Standardize Python Data | DigitalOcean," Aug. 2022.

[30] G. Dumane, "Introduction to Convolutional Neural Network (CNN) using Tensorflow," Mar. 2020.

[31] B. Krishnamurthy, "ReLU Activation Function Explained | Built In," Oct. 2022.

[32] N. Vishwakarma, "What is Adam Optimizer?," Sept. 2023.

[33] A. N. Nagpal, "L1 and L2 Regularization Methods, Explained | Built In," Jan. 2022.

[34] K. Pham, "Overfitting, Generalization, & the Bias-Variance Tradeoff," May 2023.

[35] J. N. Mandrekar, "Receiver Operating Characteristic Curve in Diagnostic Test Assessment," *Journal of Thoracic Oncology*, vol. 5, pp. 1315–1316, Sept. 2010.

[36] Google, "Classification: ROC Curve and AUC | Machine Learning," July 2022.

[37] Toronto, "City of Toronto launches CurbTO plan to address more than 100 hot spots," Apr. 2020. Archive Location: Toronto, Ontario, Canada Publisher: City of Toronto Scroll: yes.

[38] S. S. Ghosal, A. Bani, A. Amrouss, and I. El Hallaoui, "A deep learning approach to predict parking occupancy using cluster augmented learning method," in *2019 International Conference on Data Mining Workshops (ICDMW)*, pp. 581–586, 2019.

[39] M. Balmer, R. Weibel, and H. Huang, "Value of incorporating geospatial information into the prediction of on-street parking occupancy – a case study," *Geo-spatial Information Science*, vol. 24, p. 438–457, July 2021.

[40] B. Pishue, "2022 inrix global traffic scorecard," 2022.

[41] M. Thompson, "Gridlock traffic congestion," 2014.

[42] EuroCarGT, *English: A municipal ward map with ward numbers for the city of Toronto.* Nov. 2014.

[43] E. Canada, "Weather information - environment canada," Apr. 2013. Last Modified: 2023-10-24.

[44] T. Shin, "Understanding feature importance in machine learning — built in," Nov. 2023.

APPENDICES

Appendix A

Python Code Developed

A.1 Packages Used

```
#Imports
import os
import base64
import datetime
import json
import os
import statistics
import numpy
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn.metrics
import networkx as nx
import math
import calendar
import networkx as nx
import haversine as distance
from statistics import mean
from geopy.geocoders import Nominatim
from matplotlib.ticker import FuncFormatter
import matplotlib.dates as mdates
```

```
from matplotlib.dates import DateFormatter
from matplotlib.backends.backend_pdf import PdfPages
# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow import keras
from sklearn import preprocessing
from sklearn import metrics
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from scikeras.wrappers import KerasClassifier
from sklearn.model_selection import GridSearchCV, KFold
from sklearn.neural_network import MLPClassifier
from collections import Counter
from sklearn.metrics import mean_absolute_percentage_error as mape
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import mean_squared_error as mse
from matplotlib.pyplot import cm
from tensorflow.keras import regularizers
from sklearn.metrics import roc_auc_score
from math import radians, sin, cos, sqrt, atan2
#from sklearn.externals import joblib
import pickle
import joblib
import mplleaflet
import numpy as np
from scipy.stats import ttest_ind
from matplotlib import pyplot
pd.options.mode.chained_assignment = None  # default='warn'

import plotly.graph_objects as go
```

A.2 Functions used

```
def calc_distance(lat1, lon1, lat2, lon2): #Calculates the distance between two p
    R = 6371  # Earth radius in kilometers
```

```
    lat1, lon1, lat2, lon2 = map(radians, [lat1, lon1, lat2, lon2])
    dlat = lat2 - lat1
    dlon = lon2 - lon1
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
    c = 2 * atan2(sqrt(a), sqrt(1-a))
    return R * c

def average(num): #Calculates the average for a list
    sum_num = 0
    for t in num:
        sum_num = sum_num + t

    avg = sum_num / len(num)
    return avg
```

A.3 Neural Network

```
#Split data into features and labels
from tensorflow.keras.layers import Dense
from tensorflow.keras.regularizers import l2

occupancy_data = morningtrainmodel
features = occupancy_data.drop(columns = ['Spaces_Open','LOCATIONID'], axis=1)
labels = occupancy_data['Spaces_Open']

#Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.3, random_state=42)
inverse_x = X_test
inverse_y = y_test
train_inverse_x = X_train
train_inverse_y = y_train
# Normalize the features using StandardScaler
scaler = StandardScaler()
#X_train.columns = X_train.columns.astype(str)
#X_test.columns = X_test.columns.astype(str)
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

f = X_train.shape
g = f[1]
#Create a simple model with a single dense layer and a single neuron
model = Sequential([
    Dense(64, activation='relu', kernel_regularizer=l2(0.001),input_shape=((g),)),
    Dense(32, activation='relu'),
    Dense(1, activation='relu'),
])
# Compile the model using mean squared error loss and the Adam optimizer
model.compile(optimizer='adam', loss='mse', metrics=['mean_squared_error','AUC'])

# Save the scaler
# Train the model
epochs = 50
batch_size = 64
hist = model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size, verbose=1)
#model.save("morning_model")
joblib.dump(scaler, 'scaler.pkl')
#Evaluate the model
# test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)
# print(f'Test Loss: {test_loss}')
```

```
# print(f'Test Accuracy: {test_accuracy}')
```

A.4 Algorithm

```
alist = []
tlist = []
changes = {}
pvalues = []
prices = coord.copy()
prices = prices.copy()
algoupdatedf = morningtrainmodel.copy()
test = {}
avg1list = []
avg2list = []
p = []
location1 = []
location2 = []
#T-TEST LIST
locnearby = []
loc1list = []
loc2list = []
tstatlist = []
pvaluelist = []
beforesolist = []
aftersolist = []
beforerevlist = []
afterrevlist = []
beforeocclist = []
afterocclist = []
locid = algoupdatedf['LOCATIONID'].unique().tolist()
coord = newcoord.copy()
coord = coord.dropna()
coord['Lat'].astype(float)
coord['Long'].astype(float)
coords = locid.copy()
updatedcoord = coord[coord['LOCATIONID'].isin(coords)]
iteration = 0
dist = 0.15
solist = []
list4 = []
alpha = 1e-100
duplist = []
maxdegreecount = 10
spacesopen = 3
mincap = 6
maxcap = 9
oschangegraph = []
revchangegraph = []
orchangegraph = []
time = 5 * 132
#distribution(morningtrainmodel)
#rev(morningtrainmodel,spacemerge)

averages = algoupdatedf.groupby(['LOCATIONID'])['Spaces Open'].mean()
averages = averages.reset_index()
list1 = updatedcoord['Lat'].tolist()
list2 = updatedcoord['Long'].tolist()
#Turns coordinates into a list of coordinates of (Lat, Long)
for x in range(len(list1)):
    list3 = [list1[x],list2[x]]
    list4.append(list3)
locations = updatedcoord['LOCATIONID'].tolist()
avglist = averages['Spaces Open'].tolist()
#Marks each node as yellow(Full) or Green(Open) depending on spaces open on average for the 6 months
solist.clear()
yellow = averages.merge(spacemerge,how = 'left')

averages = averages.merge(spacemerge, how = 'left')
averages['Spaces Taken'] = averages['SPACES'] - averages['Spaces Open']
for a, b in averages.iterrows():
    so = b[1]
    capacity = b[2]
```

```
    st = b[3]
    if 1 < so <= spacesopen and capacity > mincap:
        solist.append("yellow")
    elif so < 1 and capacity > mincap:
        solist.append("red")
    elif st < 1 and capacity > maxcap:
        solist.append("blue")
    elif 1 < st <= spacesopen and capacity > maxcap:
        solist.append("purple")
    else:
        solist.append("green")

yellow_count = solist.count('yellow')
green_count = solist.count('green')
red_count = solist.count('red')
purple_count = solist.count('purple')
blue_count = solist.count('blue')
print(f'Number of blue nodes is {blue_count}, number of purple nodes is {purple_count}, number of green nodes is {green_co

iteration = 1

for it in range(50):
    changelist = []
    maxchanges = Counter(dlist)
    for key, value in maxchanges.items():
        if value > 1:
            changelist.append(key)
    averages = algoupdatedf.groupby(['LOCATIONID'])['Spaces Open'].mean()
    averages = averages.reset_index()
    highlist = averages.merge(spacemerge,how = 'left')
    highlist = highlist[highlist['Spaces Open'] <= spacesopen]
    highlist = highlist[highlist['SPACES'] > mincap]
    hightlist = highlist.sort_values("Spaces Open")
    highlist = highlist['LOCATIONID'].tolist()
    lowlist = averages.merge(spacemerge,how = 'left')
    lowlist['Spaces Taken'] = lowlist['SPACES'] - lowlist['Spaces Open']
    lowlist = lowlist[lowlist['Spaces Taken'] <= spacesopen]
    lowlist = lowlist[lowlist['SPACES'] > maxcap]
    lowlist = lowlist.sort_values("Spaces Taken")
    lowlist = lowlist['LOCATIONID'].tolist()
    targetlist = highlist+lowlist
    targetlist = [elem for elem in targetlist if elem not in changelist]
    print(f'Iteration {iteration} begins')
    iteration = iteration + 1
    if targetlist == []:
        break
    for i in range(len(targetlist)):
        target = [targetlist[i]]
        neighbors = []
        for i in target:
            loc1data = algoupdatedf[algoupdatedf["LOCATIONID"] == i]
            loc1datalist = loc1data['Spaces Open'].tolist()
            average1 = round(average(loc1datalist),2)
            matching_id = updatedcoord[updatedcoord['LOCATIONID'] == i]
            row_values = matching_id.iloc[0][['Long', 'Lat']].values.tolist()
            point1 = row_values
            for a ,row in updatedcoord.iterrows():
                distance1 = calc_distance(point1[1], point1[0], row['Lat'], row['Long'])
                location = int(row['LOCATIONID'])
                if 0 < distance1 < dist:
                    neighbors.append(location)
        if neighbors == []:
            print(f'{target} has no neighbors')
            dlist.append(int(average(target)))
            dlist.append(int(average(target)))
            continue
        check = averages[averages['LOCATIONID'] == int(average(target))]
        if check['Spaces Open'].mean() < spacesopen:
            change = 0.5
        else:
            change = -0.5
        loc2data = algoupdatedf[algoupdatedf['LOCATIONID'].isin(neighbors)]
        loc2datalist = loc2data['Spaces Open'].tolist()
        average2 = round(average(loc2datalist),2)
        targetlocation = target
        #GET EXISTING TRANSACTION DATA OF LOCATION
        locationtest = algoupdatedf[algoupdatedf['LOCATIONID'].isin(targetlocation)]
```

```
locationncpred = locationtest.copy()
locationpred = locationtest.copy()
locationncpred = locationncpred.drop(columns = ['LOCATIONID','Spaces Open'])
locationpred = locationpred.drop(columns = ['LOCATIONID','Spaces Open'])
#DO A PRICE CHANGE
locationpred['Price per Hour'] = locationpred['Price per Hour']+change
newprice = locationpred['Price per Hour'].mean()
oldprice = locationncpred['Price per Hour'].mean()
#DO PREDICTION OF BOTH (PRICE CHANGE VS NO PRICE CHANGE)
input_data_1 = locationncpred
input_transform_1 = scaler.transform(input_data_1)
predictions_1 = model.predict(input_transform_1)
predictions_1 = predictions_1.flatten()
y2 = predictions_1.tolist()
locationncpred['Spaces Open'] = y2
input_data = locationpred
input_transform = scaler.transform(input_data)
predictions = model.predict(input_transform)
predictions = predictions.flatten()
y1 = predictions.tolist() #price is changed
locationpred['Spaces Open'] = y1
locpgwcopy = locpgw.copy()
locpgwcopy = locpgwcopy.merge(spacemerge, on='LOCATIONID', how = 'left')
locpgwcopy = locpgwcopy.drop(columns = ['Average Parking Price','# of Parking Groups Nearby'])
locationncpred = locationncpred.merge(locpgwcopy, on='Capacity Weight (Distance)', how = 'left')
locationpred = locationpred.merge(locpgwcopy, on='Capacity Weight (Distance)', how = 'left')
#CHECK TO SEE IF AVERAGE SPACES OPEN PREDICTED > CAPACITY -> AVERAGE SPACES = CAPACITY
for index, row in locationncpred.iterrows():
    if row['Spaces Open'] > row['SPACES']:
        capacity_val = row['SPACES'].mean()
        locationncpred.at[index,'Spaces Open'] = capacity_val
loc = locationncpred.pop('LOCATIONID')
aso = locationncpred.pop('Spaces Open')
locationncpred.insert(0, 'LOCATIONID', loc)
locationncpred.insert(1, 'Spaces Open', aso)
# #locationncpred = locationncpred[locationncpred['LOCATIONID'] == location]
for index, row in locationpred.iterrows():
    if row['Spaces Open'] > row['SPACES']:
        capacity_val = row['SPACES'].mean()
        locationpred.at[index,'Spaces Open'] = capacity_val
loc = locationpred.pop('LOCATIONID')
aso = locationpred.pop('Spaces Open')
locationpred.insert(0, 'LOCATIONID', loc)
locationpred.insert(1, 'Spaces Open', aso)
#locationpred = locationpred[locationpred['LOCATIONID'] == location]
locationncpred = locationncpred[locationncpred['LOCATIONID'] == target[0]]
locationpred = locationpred[locationpred['LOCATIONID'] == target[0]]
maxspaces = locationpred['SPACES'].mean()
locationncpred = locationncpred.drop(columns = ['SPACES'])
locationpred = locationpred.drop(columns = ['SPACES'])
locationorg = morningtrainmodel[morningtrainmodel['LOCATIONID'] == target[0]]
orgprice = locationorg['Price per Hour'].mean()
#CHECK TO SEE DIFFERENCE BETWEEN PREDICTIONS
org = locationorg['Spaces Open'].tolist()
actual = locationncpred['Spaces Open'].tolist()
pred = locationpred['Spaces Open'].tolist()
diff = [pred[idx]-actual[idx] for idx in range(len(pred))]
positive_count = sum(x > 0 for x in diff)
positive_ratio = positive_count / len(diff)
average3 = round(average(locationpred['Spaces Open']),2)
average5 = round(average(locationncpred['Spaces Open']),2)
avgpred = average(pred)

if change > 0 and average3 > average1 and avgpred < (maxspaces-3):
    print(f'{targetlocation} location meets requirements with an increase of ${change}')
    duplist.append(int(average(targetlocation)))
    comparison(pred,actual,org,newprice,oldprice,orgprice,maxspaces)
    changelocation = targetlocation
    #STEP 2
    #LOOKS AT THE TARGET LOCATION PRICE CHANGE AND CHANGES THE AVERAGE PARKING PRICE OF NEARBY PARKING GROUPS
    #Gets the average prices and average surrounding areas inside a 0.3 km radius
    mask = prices["LOCATIONID"] == changelocation[0]
    prices.loc[mask, ["Price"]] = newprice

    loc1list = []
    avgp = []
    avgc = []
```

```
avgd = []
distlist = []
spacelist = []
avgplist = []
avgllist = []
avgclist = []
targetloclist = []

# Define the location ID you are looking for
for x in ids:
    ID_to_match = x
    matching_id = prices[prices['LOCATIONID'] == ID_to_match]
    row_values = matching_id.iloc[0][['Long', 'Lat']].values.tolist()
    point1 = row_values
    loc1list.append(ID_to_match)
    avgp.clear()
    avgc.clear()
    avgd.clear()
    avgp.append(newprice)
    # # Iterate over the dataframe and calculate the distance between each city and the two points
    for i, row in prices.iterrows():
        pph = row['Price']
        location = row['LOCATIONID']
        spaces = row['SPACES']
        location = int(location)
        distance1 = calc_distance(point1[1], point1[0], row['Lat'], row['Long'])

    # Check if both distances are less than a certain threshold (e.g. 0.3km)
        if 0 < distance1 < dist:
            avgp.append(pph)
            avgc.append(spaces)
            avgd.append(distance1)
            if len(avgp) > 0:
                a = average(avgp)
                b = len(avgp)-1
            else:
                a = 0
                b = 0
            distances = sum(avgd)
            captimesdistance = sum(multiply_lists_recursive(avgc,avgd))
            c = captimesdistance/distances
        if (targetlocation[0] == ID_to_match)&(0 < distance1 < dist):
            targetloclist.append(location)
    avgplist.append(a)
    avgllist.append(b)
    avgclist.append(c)
algonewprice = pd.DataFrame()
algonewprice['LOCATIONID'] = loc1list
algonewprice['Average Parking Price'] = avgplist
algonewprice['# of Parking Groups Nearby'] = avgllist
algonewprice['Capacity Weight (Distance)'] = avgclist
#STEP 3
#Turn the data into one clean dataframe
aa = targetlocation[0]
locapp = algonewprice[algonewprice['LOCATIONID'] == aa]
locappp = locapp['Average Parking Price'].mean()
locationpred['Average Parking Price'] = locappp
a = algonewprice.drop(columns = ['# of Parking Groups Nearby','Capacity Weight (Distance)'])
algoupdatedf = algoupdatedf.drop(columns = ['Average Parking Price'])
algoupdatedf = algoupdatedf.merge(a, on='LOCATIONID', how = 'right')
b = algoupdatedf.pop("Average Parking Price")
algoupdatedf.insert(4,"Average Parking Price", b)
neighborpredictions = pd.DataFrame()
itbreak = 0
for i in targetloclist:
    neighborpreds = algoupdatedf[algoupdatedf['LOCATIONID'].isin([i])]
    neighborpreds = neighborpreds.drop(columns = ['Spaces Open'])
    neighborpreds= neighborpreds.drop(columns = ['LOCATIONID'])
    input_data = neighborpreds
    input_transform = scaler.transform(input_data)
    algopredictions = model.predict(input_transform)
    algopredictions = algopredictions.flatten()
    neighborpreds['Spaces Open'] = algopredictions
    Occ = neighborpreds.pop('Spaces Open')
    neighborpreds.insert(0, 'Spaces Open', Occ)
    algopred = algopredictions.tolist() #price is changed
    neighborpreds['Spaces Open'] = algopred
```

```
            neighborpreds['LOCATIONID'] = i
            neighborpreds = neighborpreds.merge(spacemerge, how = 'left')
            for index, row in neighborpreds.iterrows():
                if row['Spaces Open'] > row['SPACES']:
                    capacity_val = row['SPACES'].mean()
                    neighborpreds.at[index,'Spaces Open'] = capacity_val
            loc = neighborpreds.pop('LOCATIONID')
            aso = neighborpreds.pop('Spaces Open')
            neighborpreds.insert(0, 'LOCATIONID', loc)
            neighborpreds.insert(1, 'Spaces Open', aso)
            #neighborpreds['Capacity'] = neighborpreds['SPACES']
            averagetest = average(neighborpreds['Spaces Open'].tolist())
            spaces = average(neighborpreds['SPACES'].tolist())
            neighborpreds = neighborpreds.drop(columns = ['SPACES'])
            if averagetest == spaces:
                print(f' {i} does not meet neighbor requirements')
                itbreak = itbreak+1
                break
            else:
                neighborpredictions = pd.concat([neighborpredictions,neighborpreds])
        if itbreak > 0:
            dlist.append(int(average(targetlocation)))
            dlist.append(int(average(targetlocation)))
            break
        #UPDATE THE DATAFRAME WITH THE NEW PREDICTIONS AND PRICE CHANGES
        frames = [locationpred,neighborpredictions]
        average4 = round(average(neighborpredictions['Spaces Open']),2)
        if average4 < 3:
            print('neighbor predictions do not meet requirements')
            dlist.append(int(average(targetlocation)))
            dlist.append(int(average(targetlocation)))
            break
        result = pd.concat(frames)
        newlocdata = result['LOCATIONID'].unique()
        newlocdata = newlocdata.tolist()
        remove = algoupdatedf[~algoupdatedf['LOCATIONID'].isin(newlocdata)]
        newframes = [remove,result]
        algoupdatedf = pd.concat(newframes)
        alist.append(changelocation)
        tlist.append(changelocation)
        beforesolist.append(average1)
        aftersolist.append(average3)
        beforerevlist.append((maxspaces-average5)*time*oldprice)
        afterrevlist.append((maxspaces-average3)*time*newprice)
        beforeocclist.append(((maxspaces-average5)/maxspaces)*100)
        afterocclist.append(((maxspaces-average3)/maxspaces)*100)
        algoupdatedf = algoupdatedf[algoupdatedf['LOCATIONID'].isin(locations)]
        avg1list.append(average5)
        avg2list.append(average3)
        changes[int(average(targetlocation))] = changes.get(int(average(targetlocation)), 0) + 1
        print(f'DateFrame Updated Updated Spaces Open Averages for Target Location
        {targetlocation}:{average1} -> {average3} and Neighbors:{average2} ->
        {average4}')
        price = algoupdatedf.groupby(['LOCATIONID'])['Price per Hour'].mean()
        price = price.reset_index()
        averages = algoupdatedf.groupby(['LOCATIONID'])['Spaces Open'].mean()
        averages = averages.reset_index()
        averages = averages.merge(spacemerge,on = 'LOCATIONID', how = 'left')
        averages = averages.merge(price,on = 'LOCATIONID', how = 'left')
        averages['Revenue'] = (averages['SPACES']-averages['Spaces Open']) * time * averages['Price per Hour']
        averages['Occupancy Ratio'] = ((averages['SPACES']-averages['Spaces Open'])/averages['SPACES'])*100
        oschangegraph.append(averages['Spaces Open'].sum())
        revchangegraph.append(averages['Revenue'].sum())
        orchangegraph.append(average(averages['Occupancy Ratio']))
        dlist.append(int(average(targetlocation)))
        print(targetloclist)
        break
    elif change < 0 and average3 < average1 and avgpred > 0 and avgpred < maxspaces:
        print(f'{targetlocation} location meets requirements with an decrease of ${change}')
        duplist.append(int(average(targetlocation)))
        comparison(pred,actual,org,newprice,oldprice,orgprice,maxspaces)
        changelocation = targetlocation
        #STEP 2
        #LOOKS AT THE TARGET LOCATION PRICE CHANGE AND CHANGES THE AVERAGE PARKING PRICE OF NEARBY PARKING GROUPS
        #Gets the average prices and average surrounding areas inside a 0.3 km radius
        mask = prices["LOCATIONID"] == changelocation[0]
        prices.loc[mask, ["Price"]] = newprice
```

```
loc1list = []
avgp = []
avgc = []
avgd = []
distlist = []
spacelist = []
avgplist = []
avgllist = []
avgclist = []
targetloclist = []

# Define the location ID you are looking for
for x in ids:
    ID_to_match = x
    matching_id = prices[prices['LOCATIONID'] == ID_to_match]
    row_values = matching_id.iloc[0][['Long', 'Lat']].values.tolist()
    point1 = row_values
    loc1list.append(ID_to_match)
    avgp.clear()
    avgc.clear()
    avgd.clear()
    avgp.append(newprice)
    ## Iterate over the dataframe and calculate the distance between each city and the two points
    for i, row in prices.iterrows():
        pph = row['Price']
        location = row['LOCATIONID']
        spaces = row['SPACES']
        location = int(location)
        distance1 = calc_distance(point1[1], point1[0], row['Lat'], row['Long'])

    # Check if both distances are less than a certain threshold (e.g. 0.3km)
        if 0 < distance1 < dist:
            avgp.append(pph)
            avgc.append(spaces)
            avgd.append(distance1)
            if len(avgp) > 0:
                a = average(avgp)
                b = len(avgp)-1
            else:
                a = 0
                b = 0
            distances = sum(avgd)
            captimesdistance = sum(multiply_lists_recursive(avgc,avgd))
            c = captimesdistance/distances
        if (targetlocation[0] == ID_to_match)&(0 < distance1 < dist):
            targetloclist.append(location)
    avgplist.append(a)
    avgllist.append(b)
    avgclist.append(c)
algonewprice = pd.DataFrame()
algonewprice['LOCATIONID'] = loc1list
algonewprice['Average Parking Price'] = avgplist
algonewprice['# of Parking Groups Nearby'] = avgllist
algonewprice['Capacity Weight (Distance)'] = avgclist
#STEP 3
#Turn the data into one clean dataframe
aa = targetlocation[0]
locapp = algonewprice[algonewprice['LOCATIONID'] == aa]
locappp = locapp['Average Parking Price'].mean()
locationpred['Average Parking Price'] = locappp
a = algonewprice.drop(columns = ['# of Parking Groups Nearby','Capacity Weight (Distance)'])
algoupdatedf = algoupdatedf.drop(columns = ['Average Parking Price'])
algoupdatedf = algoupdatedf.merge(a, on='LOCATIONID', how = 'right')
b = algoupdatedf.pop("Average Parking Price")
algoupdatedf.insert(4,"Average Parking Price", b)
neighborpredictions = pd.DataFrame()
itbreak = 0
for i in targetloclist:
    neighborpreds = algoupdatedf[algoupdatedf['LOCATIONID'].isin([i])]
    neighborpreds = neighborpreds.drop(columns = ['Spaces Open'])
    neighborpreds= neighborpreds.drop(columns = ['LOCATIONID'])
    input_data = neighborpreds
    input_transform = scaler.transform(input_data)
    algopredictions = model.predict(input_transform)
    algopredictions = algopredictions.flatten()
    neighborpreds['Spaces Open'] = algopredictions
```

```
            Occ = neighborpreds.pop('Spaces Open')
            neighborpreds.insert(0, 'Spaces Open', Occ)
            algopred = algopredictions.tolist() #price is changed
            neighborpreds['Spaces Open'] = algopred
            neighborpreds['LOCATIONID'] = i
            neighborpreds = neighborpreds.merge(spacemerge, how = 'left')
            for index, row in neighborpreds.iterrows():
                if row['Spaces Open'] > row['SPACES']:
                    capacity_val = row['SPACES'].mean()
                    neighborpreds.at[index,'Spaces Open'] = capacity_val
            loc = neighborpreds.pop('LOCATIONID')
            aso = neighborpreds.pop('Spaces Open')
            neighborpreds.insert(0, 'LOCATIONID', loc)
            neighborpreds.insert(1, 'Spaces Open', aso)
            #neighborpreds['Capacity'] = neighborpreds['SPACES']
            averagetest = average(neighborpreds['Spaces Open'].tolist())
            spaces = average(neighborpreds['SPACES'].tolist())
            neighborpreds = neighborpreds.drop(columns = ['SPACES'])
            if averagetest == spaces:
                print(f' {i} does not meet neighbor requirements')
                itbreak = itbreak+1
                break
            else:
                neighborpredictions = pd.concat([neighborpredictions,neighborpreds])
        if itbreak > 0:
            dlist.append(int(average(targetlocation)))
            dlist.append(int(average(targetlocation)))
            break
        #UPDATE THE DATAFRAME WITH THE NEW PREDICTIONS AND PRICE CHANGES
        frames = [locationpred,neighborpredictions]
        average4 = round(average(neighborpredictions['Spaces Open']),2)
        if average4 < 3:
            print('neighbor predictions do not meet requirements')
            dlist.append(int(average(targetlocation)))
            dlist.append(int(average(targetlocation)))
            break
        result = pd.concat(frames)
        newlocdata = result['LOCATIONID'].unique()
        newlocdata = newlocdata.tolist()
        remove = algoupdatedf[~algoupdatedf['LOCATIONID'].isin(newlocdata)]
        newframes = [remove,result]
        algoupdatedf = pd.concat(newframes)
        alist.append(changelocation)
        tlist.append(changelocation)
        beforesolist.append(average1)
        aftersolist.append(average3)
        beforerevlist.append((maxspaces-average5)*time*oldprice)
        afterrevlist.append((maxspaces-average3)*time*newprice)
        beforeocclist.append(((maxspaces-average5)/maxspaces)*100)
        afterocclist.append(((maxspaces-average3)/maxspaces)*100)
        algoupdatedf = algoupdatedf[algoupdatedf['LOCATIONID'].isin(locations)]
        avg1list.append(average5)
        avg2list.append(average3)
        changes[int(average(targetlocation))] = changes.get(int(average(targetlocation)), 0) + 1
        print(f'DateFrame Updated, Updated Spaces Open Averages for Target Location {targetlocation}:{average1} -> {a
        price = algoupdatedf.groupby(['LOCATIONID'])['Price per Hour'].mean()
        price = price.reset_index()
        averages = algoupdatedf.groupby(['LOCATIONID'])['Spaces Open'].mean()
        averages = averages.reset_index()
        averages = averages.merge(spacemerge,on = 'LOCATIONID', how = 'left')
        averages = averages.merge(price,on = 'LOCATIONID', how = 'left')
        averages['Revenue'] = (averages['SPACES']-averages['Spaces Open']) * time * averages['Price per Hour']
        averages['Occupancy Ratio'] = ((averages['SPACES']-averages['Spaces Open'])/averages['SPACES'])*100
        oschangegraph.append(averages['Spaces Open'].sum())
        revchangegraph.append(averages['Revenue'].sum())
        orchangegraph.append(average(averages['Occupancy Ratio']))
        dlist.append(int(average(targetlocation)))
        print(targetloclist)
        break
    else:
        print(f'{targetlocation} does not meet requirements when price is changed by ${change}')
        dlist.append(int(average(targetlocation)))
        dlist.append(int(average(targetlocation)))
    #Draws the network
averages = algoupdatedf.groupby(['LOCATIONID'])['Spaces Open'].mean()
averages = averages.reset_index()
list1 = updatedcoord['Lat'].tolist()
```

```
list2 = updatedcoord['Long'].tolist()
#Turns coordinates into a list of coordinates of (Lat, Long)
for x in range(len(list1)):
    list3 = [list1[x],list2[x]]
    list4.append(list3)
locations = updatedcoord['LOCATIONID'].tolist()
avglist = averages['Spaces Open'].tolist()
#Marks each node as yellow(Full) or Green(Open) depending on spaces open on average for the 6 months
solist.clear()
averages = averages.merge(spacemerge, how = 'left')
averages['Spaces Taken'] = averages['SPACES'] - averages['Spaces Open']
for a, b in averages.iterrows():
    so = b[1]
    capacity = b[2]
    st = b[3]
    if 1 < so <= spacesopen and capacity > mincap:
        solist.append("yellow")
    elif so < 1 and capacity > mincap:
        solist.append("red")
    elif st < 1 and capacity > maxcap:
        solist.append("blue")
    elif 1 < st <= spacesopen and capacity > maxcap:
        solist.append("purple")
    else:
        solist.append("green")

yellow_count = solist.count('yellow')
green_count = solist.count('green')
red_count = solist.count('red')
purple_count = solist.count('purple')
blue_count = solist.count('blue')
print(f'Number of blue nodes is {blue_count}, number of
purple nodes is {purple_count},
number of green nodes is {green_count},
number of yellow nodes is {yellow_count},
and number of Red Nodes is {red_count}.')
```

A.5 Visualizations

```
plt.title('Comparison of location parking prices in the morning')
plt.xlabel('Location Parking Price per Hour ($)')
plt.ylabel('Occupancy Rate (%)')
plt.savefig('morningboxplot.png')
plt.show()
```

```
#Generating the parking network map
import pandas as pd
import numpy as np
import plotly.express as px
import random

#creating random data

newcoord['size'] = 0.1

fig = px.scatter_mapbox(newcoord,
lat='Lat',
lon='Long',
center={'lat':43.7,
```

```
        'lon':-79.388752},
size = 'size',
size_max = 3,
color_continuous_scale=px.colors.sequential.Viridis,
width = 600,
height = 500,
zoom=9.7)

fig.update_layout(mapbox_style='open-street-map')

plt.title('Comparison of location parking prices in the morning')
plt.xlabel('Location Parking Price per Hour ($)')
plt.ylabel('Occupancy Rate (%)')
plt.savefig('morningboxplot.png')
plt.show()

plt.title('Comparison of location parking prices in the morning')
plt.xlabel('Location Parking Price per Hour ($)')
plt.ylabel('Occupancy Rate (%)')
plt.savefig('morningboxplot.png')
plt.show()
```