



**Trinity
College
Dublin**

The University of Dublin

MCS Dissertation

Predicting Queueing Times Using Queueing Theory & Predictive Analytics

by

Conor Murphy

Supervisor: Prof. Siobhán Clarke

A dissertation submitted in fulfilment of the requirements

*for the degree: **Master in Computer Science***

at the

School of Computer Science & Statistics,

The University of Dublin, Trinity College

Submitted to the University of Dublin, Trinity College, May 2018

Declaration

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

Conor Murphy

Date

Permission to lend

I agree that the Library and other agents of the College may lend or copy this report upon request.

Conor Murphy

Date

Summary

The dissertation's main goal was to attempt to answer the research question: *“How accurate can a system using a combination of queueing theory and predictive analytics be in creating predictions of queueing times in dynamic, single, real-time queues such as those in walk-in food takeaway establishments?”*

Queueing theory is the mathematical study of queues. Predictive analytics involves analysing historical and current data and then feeding that data into a predictive model. This dissertation combined queueing theory and predictive analytics to create an application and web server to act as a system that creates predictions of queueing times of walk-in food takeaway establishments with dynamic, single, real-time queues. Queueing times in this context starts when a customer enters the queue, and finishes when the customer receives their order. To begin, queueing theory was researched thoroughly, with a look at different types of queueing models and queueing disciplines. The algorithms to be used in the predictive model were also researched, with that research present in Chapter 2. Related work was also discussed and critiqued.

This application acted as a client to the web server, sending and receiving HTTP requests. The web server consisted of a queueing model and predictive model. The queueing model was used to simulate a week of wait times to be fed into the predictive model. The predictions created by the system proved to be quite accurate given the test data, with an overall average absolute error of only 2.25 minutes. However, the amount of test data used was not sufficient enough to answer the research question fully. The results of this dissertation may not be enough to answer the research question in a satisfactory manner, but it does give substance to the theory that that it may be possible to create accurate predictions for food establishment queueing times.

Abstract

University of Dublin, Trinity College

School of Computer Science & Statistics

Master in Computer Science

Predicting Queueing Times Using Queueing Theory & Predictive Analytics

Author: Conor Murphy - Supervisor: Prof. Siobhán Clarke

Queueing theory has its origin in research by Agner Krarup Erlang, with his design of models that describe the telephone exchange in Copenhagen, Denmark. Since its inception, queueing theory has been incorporated into telecommunications networks, traffic engineering, and even in the design of retail locations and factories.

In the past few years, predictive analytics as a field has been gaining traction in both research and commercial environments. Predictive analytics involves analysing historical and current data and then feeding that data into a predictive model. The model then runs one or more algorithms, producing a prediction about future or unknown events.

This dissertation aims to utilise a combination of both fields in an attempt to answer the following research question: *“How accurate can a system using a combination of queueing theory and predictive analytics be in creating predictions of queueing times in dynamic, single, real-time queues such as walk-in food takeaway establishments?”*

Acknowledgements

- First and foremost, I would like to express my sincere gratitude to my supervisor, Prof. Siobhán Clarke, for her continued support over the year. Without her advice and guidance, this dissertation would never have been finished.
- All of the staff and faculty members in the School of Computer Science & Statistics, for imparting their vast knowledge to me over the past 5 years. A majority of my current knowledge would be non-existent without their time and dedication.
- My fellow classmates who were in this adventure right beside me.
- My family, girlfriend, and friends, for helping me to keep my head above the water.

Thank you all for your time, advice, knowledge, companionship, and love.

Contents

1	Introduction	1
1.1	Why is this research being carried out?	1
1.2	What's been done before?	1
1.3	What are the objectives?	2
1.4	How did it turn out?	3
2	Research	4
2.1	Related Work	4
2.1.1	Mathematical-based approach	4
2.1.2	Observation-based approach	6
2.1.3	Virtual queues	6
2.1.4	Takeaways	7
2.2	Queueing Theory	7
2.2.1	Kendall's Notation	7
2.2.2	Queueing Disciplines	8
2.2.3	Queueing Models	9
2.2.4	Takeaways	13
2.3	Predictive Analytics	14
2.3.1	Regression	14
2.3.2	Linear Regression	15
2.3.3	Ridge and Lasso Regression	16
2.3.4	Random Forest	16
2.3.5	K-Nearest Neighbours	17
2.3.6	Takeaways	18
2.4	Cloud Hosting	18
2.4.1	SCSSNebula	18

2.4.2	Amazon Web Services	19
2.4.3	Takeaways	20
3	Design & Implementation	21
3.1	Design	21
3.1.1	Overview	21
3.1.2	Application	24
3.1.3	Web Server	28
3.2	Implementation	30
3.2.1	Where to start?	30
3.2.2	Web Server	30
3.2.3	Application	35
3.2.4	Testing	38
4	Evaluation	40
4.1	Results	40
4.1.1	Individual Algorithm Performance	40
4.1.2	Prediction Accuracy	43
4.1.3	Summary of Results	48
4.2	Criticism	49
5	Future Work	51
5.1	Queueing Model Improvements	51
5.2	Further Testing & Tweaking	52
5.3	Branching Out	52
A	Sample Google Places API Response	58
B	MAE Algorithm Choosing Method - Results Graphs	59
C	Sample Function and Unit Test	62
D	Individual Algorithm Comparison Data Tables	63

List of Figures

2.1	Arrival process codes	8
2.2	Service time distribution codes	8
2.3	State space diagram of M/D/1 queue[39]	10
2.4	M/D/c Queue model with 2 servers. The service rate is given as $1/D$, where D is the fixed service rate time.	11
2.5	M/M/1 Queue example	11
2.6	M/M/c Queue model with 2 servers. The queue is the same as an M/D/c queue, with the one exception being that the service rate is not deterministic.	13
2.7	The proxy acts as a “barrier” between the outside world and the VM .	19
2.8	An overview of the Amazon EC2 instance security [8]	20
3.1	The original concept for the application UI	22
3.2	An overview of a user process	23
3.3	Design Overview	23
3.4	Basic overview of application’s function	24
3.5	The initial UI displayed to the user when they start the application . .	24
3.6	Searching by name screen	25
3.7	Searching nearby food establishments screen	26
3.8	The chosen food establishments details before making the request . . .	27
3.9	The submit wait time UI screen	28
3.10	A request’s journey from request to prediction	28
3.11	The inner workings of the web server	30
3.12	A sample of attributes of a Flask request object [4]	31
3.13	The average absolute errors of both methods of choosing algorithms . .	35

3.14	A high-level look at the libraries and frameworks used to create the overall system	38
3.15	The screen displayed to users once a prediction is returned	38
4.1	Individual algorithm results after 0 inputs	41
4.2	Individual algorithm results after 1 input	41
4.3	Individual algorithm results after 2 inputs	42
4.4	Individual algorithm results after 3 inputs	42
4.5	Individual algorithm results after 4 inputs	43
4.6	Cafe queueing times compared to the predictions made by the system .	44
4.7	The absolute error rate of the cafe establishment type	44
4.8	Fast food queueing times compared to the predictions made by the system	45
4.9	The absolute error rate of the fast food establishment type	45
4.10	Takeaway queueing times compared to the predictions made by the system	46
4.11	The absolute error rate of the takeaway establishment type	46
4.12	Deli queueing times compared to the predictions made by the system .	47
4.13	The absolute error rate of the deli establishment type	47
4.14	Ranking the establishment types from most to least accurate	49
A.1	Sample response from Google Places API	58
B.1	Cafe results after choosing predictive algorithms using the MAE method mentioned in Chapter 3	59
B.2	Deli results after choosing predictive algorithms using the MAE method mentioned in Chapter 3	60
B.3	Fast food results after choosing predictive algorithms using the MAE method mentioned in Chapter 3	60
B.4	Takeaway results after choosing predictive algorithms using the MAE method mentioned in Chapter 3	61
D.1	Individual algorithm predictions after 0 inputs	63
D.2	Individual algorithm predictions after 1 input	63
D.3	Individual algorithm predictions after 2 inputs	64
D.4	Individual algorithm predictions after 3 inputs	64
D.5	Individual algorithm predictions after 4 inputs	64

Chapter 1

Introduction

1.1 Why is this research being carried out?

Day to day, there are many times when people find themselves in a situation where they need to queue. Such a queue can eat into people's time, something invaluable in this day and age. In order to make the most of our time, there needs to be a way to predict queueing time, before entering the queue itself. Knowing this would allow for better time management, and ultimately affect a person's decision on whether or not they will enter the queue.

A recent survey stated that on average, people spend 5 hours and 35 minutes a month queueing [21]. This equates to 67 hours, or almost three days queueing annually. In a world that is constantly moving and fast-paced, every minute counts.

The inspiration for this research topic came about in mid-2017, when going for lunch with friends. Upon arrival at the food establishment, the queue was a lot larger than we had originally anticipated. I remember vividly thinking that there must be a better way than pure guess work to predict queue time. Also in 2017, I spent half a year working as an intern in SAP's advanced analytics division. This really forged my own personal interest in predictive analytics.

1.2 What's been done before?

Within related work, there are three approaches taken to attempt to manage queue times. These are observation-based methods, mathematical modelling using queueing theory, and those that utilise virtual queues. The observation-based methods main

strengths come from the fact that they use historical data and display that data to users. While this shows past queue times, it does not create any predictions. This means that the user has to have some ability to analyse the data and make (possibly inaccurate) assumptions. There is also the possibility of human error, due to the fact that all calculations are done by humans. Systems that use mathematical models have the advantage of being able to create these predictions using queueing theory. However, the lack of real-time data can lead to obsolete models which decrease in accuracy over time. Another similar approach as regards handling queues involve the creation of virtual queues. They let the user know how long they will be waiting based on how many people are in front of them. While this approach does allow users to see how long they will be waiting, they have to actually enter the virtual queue before seeing the queue time.

The aforementioned approaches have their strengths, but they also have some fatal weaknesses. They simply display current waiting times to users, but are unable to predict what the queueing time would be in the future. The aim of this dissertation is to create a system that takes advantage of those strengths, while neutralising the weaknesses (or minimising their effect).

In this context, data will be gathered using both user-inputted data, as well as queueing theory systems. Users will use an Android application to communicate with a web server hosted on AWS. The application allows users to search for a specific food establishment or choose from nearby food establishments. Once a user selects a food establishment, the web server then returns a queueing time prediction. Users may also enter in their own queue times, which will have an effect on predictions given to future users.

1.3 What are the objectives?

The objective for this dissertation is simple. The main goal is to create a system that creates accurate predictions for walk-ins to various types of food establishments, using both queueing theory and predictive analytics. The fact that we are dealing with walk-ins means that we will not be dealing with situations where the establishment received an order for collection, which eliminates the need to queue. Secondary objectives also include creating an easy-to-use system. The system should also perform well, i.e. the

prediction should be given in a quick and timely manner.

The system could then see a change in how queues are managed in food establishments, as well as having the possibility to expand into other areas such as doctor's offices, theme parks, post offices, or anywhere else that a queue is regularly formed. This can help users make informed decisions about how best to spend their time, which in turn could have an impact on those businesses or areas where queues are needed.

1.4 How did it turn out?

Following research, design, and implementation of this system, the average absolute error for predictions was 2.25 minutes. This gives what can be seen as a beginning of an answer to the original research question. While the results are promising, no final answer to the research question can be given until even further testing is done. However, the results shown in Chapter 4 - Evaluation show that it is definitely possible to get a somewhat accurate prediction, given the test data used in this dissertation.

Chapter 2

Research

This chapter will discuss the research done in order to craft the system. The chapter begins by examining related work. It will then take a deeper look at queueing theory, and the predictive analytics algorithms incorporated into the solution. Following this, we will take a look at the cloud hosting options for the server, namely SCSSNebula vs. AWS.

2.1 Related Work

As mentioned earlier in this report, there are several approaches that attempt to predict queueing times. These include those that follow a mathematical model using purely queueing theory, whereby data is collected beforehand and then fed into a queueing model. Another approach involves physically counting queues and performing calculations. The results are then displayed. Finally, an approach that is used a lot in the restaurant industry involves creating a virtual queue that users can enter, which will tell them how long they have to wait before they get to the front of the queue.

Both the strengths and weaknesses of these similar applications will have an effect on my overall solution.

2.1.1 Mathematical-based approach

One of the first similar applications examined was a project titled “Restaurant Wait Time Estimation Report” by Segovia, Patel, & Lonneman [30]. For this project, the authors collected data from a local sit-down restaurant and fed it into their queueing model. The queueing model consisted of an M/M/c queue, where the arrival rate (λ)

represented the amount of customers per hour, and service rate (μ) is how long it took to serve each customer. The M/M/c queue represents a single line served by multiple servers (c is greater than 1).

The simulation model involves creating data structures that represent the menu, servers, parties, tables, and waiting list. Two methods were then created to initialize a night at the restaurant and another to run through a step-by-step process of the night. The second method created an arrival time for parties of random sizes, anticipated what the party orders, and measured the total time the party spent in the queue.

The main strengths of this approach lies in the strict use of queueing theory. The model was able to give precise predictions and because of this, the simulation resulted in an overall margin error of +/- 2 minutes [30]. However, the fact that the data was only collected once and not on a continuous, rolling basis means that the queueing models can soon turn stale, possibly resulting in erroneous predictions. Another area in which this approach falls short is the fact that the data collected is one hundred percent simulated, with no real-time data. The authors used a software program known as Arena in an attempt to simulate a real-world restaurant.

Another project that follows this approach is “The Waiting Game: Fast-Food Queuing Theory” by AetherWorks employee Shannon Cody [20]. Cody was inspired by her daily lunch trips, and wanted to see if she could “*eat there more frequently and waste less time stuck in line*” [20]. Unlike the report by Segovia, Patel, & Lonneman, Cody examines multiple restaurants with different queueing styles. These are single server with a single queue, single server with multiple queues, as well as multiple servers in multiple queues. Where the two are similar is in their choice of queueing model. Both use an M/M/c queue to simulate their respective queues. Some of Cody’s test restaurants include Subway, Chipotle, and Starbucks. The author breaks down each station at each restaurant and estimates the time spent at each station. For example, Cody states that at Chipotle there are 5 stations, each taking up different amounts of time. For her queueing model, Cody assumes an arrival rate of 1 customer every 200 seconds. This assumption is one of the main weaknesses of her approach, with the author herself stating that it is “admittedly unrealistic” [20]. The service rate appears to have been observed in the restaurants themselves and noted down by the author.

Overall, the strength of mathematical-based estimations is the accuracy of their pre-

dictions. The fact that the projects could focus on a certain amount of restaurants allowed the authors to fine tune their designs to ensure this high level of accuracy. However, this focus also has drawbacks. Due to the fact that only a few restaurants were selected, the models that were created are not a “one-size-fits-all” solution. The main weakness in both of the aforementioned projects lie in the fact that the data is collected once, before the model is executed. This can lead to obsolete models which, as previously mentioned, could possibly lead to miscalculated predictions.

2.1.2 Observation-based approach

The observation-based approach that will be discussed here is seen at <https://queue-times.com/>. This website draws information from theme parks’ native apps to one place, and then displays them to users. To estimate a queue time, these parks estimate the amount of people in a queue, then figure out how long it takes to fill a ride vehicle, finish the ride, and then leave the line. They also factor in how many people they can fit into a ride vehicle. The website also displays historical data, meaning that users can see the queue times of specific days in the past.

This approach updates over time, at different intervals (dependent on the park’s schedule). This makes this approach quite powerful, as the times can never become obsolete. Users can get quick up-to-date results whenever they want them.

Where this approach fails our current objective is down to how long it takes to update queue times. An update requires recounting the entire queue, an activity that can be time-consuming, especially as the queue gets longer. There is also the possibility of human error when counting the queue, thus leading to incorrect queueing time calculations.

2.1.3 Virtual queues

Virtual queues are mainly used by restaurants. Some popular choices are QLess [15] and Nowait [12]. These systems work by creating a virtual queue that customers can then join. This queue assigns each users a number. The lower the number, the closer to the front of the queue you are. Both operate via text message, sending a text to users when they enter the queue, and when they are next in the queue. Once someone in the virtual queue has been served, the restaurant hosts then move the queue along.

This adds a possibility of human error to the system, which could result in incorrect wait times.

The strengths of this approach are clear. Users can immediately see how long they will have to wait once joining the queue. It also eliminates the need to physically queue. However, the weaknesses lie in the reliance on restaurant hosts to move the queue along. Particularly on busy nights, employees may become too flustered to remember to move the queue on every single time. Another issue I found with this approach is that users can only see the wait time once they enter the queue. If a user sees the wait time and is not willing to wait that long, they could throw off the entire queueing operation by leaving the queue or worse, forgetting to leave the queue.

2.1.4 Takeaways

Analysing these similar approaches really helped to shape the overall design of my solution. The solution will attempt to take the accuracy of the mathematical-based approach and combine that with the constant updating of the observation-based approach, while also attempting to emulate the ease-of-use of the virtual queues. The goal is to create a system that requires minimal input from the user while producing accurate and up-to-date predictions.

2.2 Queueing Theory

Queueing theory is the mathematical study of queues. A queueing model is created in order to predict queue lengths and wait times. The field has its origins in research by Agner Krarup Erlang, a Danish mathematician, in the early twentieth century. Queues themselves are crucial economically. In order to eliminate queues, resources would have to be unlimited. Managing queues is also an essential task as excessive wait times can lead to loss of sales, as well as losing custom to competitors.

2.2.1 Kendall's Notation

Queues are generally described using Kendall's notation in the form $A/S/c$, where A is the time between arrivals to the queue, S represents the time it takes to complete service, and c represents the number of servers. Kendall's notation is the standard system used to classify queueing nodes proposed by David George Kendall in 1953.

Below are two figures showing tables of the various symbols used in Kendall's notation [23]:

Symbol	Name	Description
M	Markovian/memoryless	Poisson process (or random) arrival process
M^X	Batch Markov	Poisson process with a random variable X for the number of arrivals at one time
MAP	Markovian arrival process	Generalisation of the Poisson process
BMAP	Batch Markovian arrival process	Generalisation of the MAP with multiple arrivals
MMPP	Markov modulated Poisson process	Poisson process where arrivals are in "clusters"
D	Degenerate distribution	A deterministic or fixed inter-arrival time
E_k	Erlang distribution	An Erlang distribution with k as the shape parameter
G	General distribution	-
PH	Phase-type distribution	Some of the above distributions are special cases of the phase-type, often used in place of a general distribution

Figure 2.1: Arrival process codes

Symbol	Name	Description
M	Markovian/memoryless	Exponential service time
M^Y	Bulk Markov	Exponential service time with a random variable Y for the size of the batch of entities serviced at one time
D	Degenerate distribution	A deterministic or fixed service time
E_k	Erlang distribution	An Erlang distribution with k as the shape parameter
G	General distribution	-
PH	Phase-type distribution	Some of the above distributions are special cases of the phase-type, often used in place of a general distribution
MMPP	Markov modulated Poisson process	Exponential service time distributions, where the rate parameter is controlled by a Markov chain

Figure 2.2: Service time distribution codes

Figure 2.1 shows the arrival process codes, while figure 2.2 shows the service time distribution codes.

2.2.2 Queueing Disciplines

Queueing discipline determines the way the server serves the customers. It tells us whether customers are served consecutively or simultaneously. If consecutively, it also describes the order in which they are served. If simultaneously, it describes how the server capacity is shared among customers. Queueing nodes can follow one of the following service disciplines [27]:

First in first out (FIFO) - A traditional approach to queues whereby the customer who has been waiting the longest is served first.

Last in first out (LIFO) - The opposite approach to FIFO, whereby the customer who has been waiting the least is served first.

Processor sharing - Service capacity is equally shared between customers.

Priority - Customers with high priority are served first.

Shortest job first - The next job to be served is the one with the smallest size.

Shortest remaining processing time - The next job to serve is the one with the smallest remaining processing requirement.

2.2.3 Queueing Models

There are various queueing models that are used in queueing theory. Below is an investigation into these models which will help to pick the appropriate model for the final system. The choice of queueing model is extremely crucial, as the queueing model will be producing the simulated data to be fed into the predictive model. This pipeline is discussed in further detail in Chapter 3 - Design & Implementation.

M/D/1 Queue

The M/D/1 queue represents a queue with a single server (1), a Poisson process arrival distribution (M), with a fixed (deterministic) service time (D). The model was first published by Erlang in 1909, and used in the design of the Copenhagen Telephone Exchange. This initial queueing model was the beginning of the field of queueing theory. An M/D/1 queue is a stochastic process whose state space is the set 0, 1, 2, 3, where the value corresponds to the number of entities in the system, including any currently in service. Arrivals occur at rate λ according to a Poisson process, moving the process from state i to $i + 1$. Service rate is $\mu = 1/D$, where D is the fixed service time. Utilization = $\rho = \lambda/\mu$. The single server serves those in the queue, one at a time (following a FIFO discipline). Once the service is complete, the customer leaves the queue, decreasing the queue length by 1. The queue capacity is infinite.

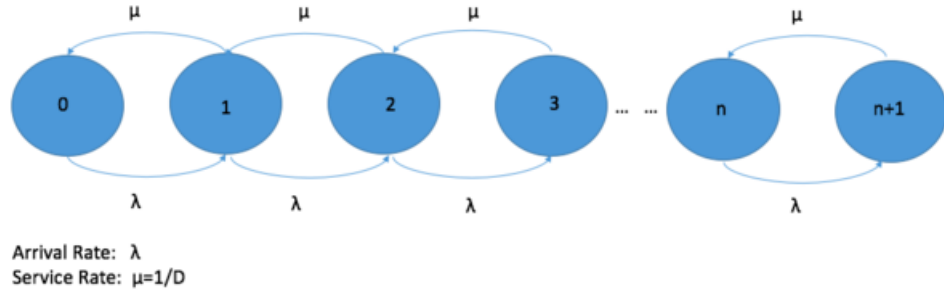


Figure 2.3: State space diagram of M/D/1 queue[39]

Performance Metrics

The waiting time in the system, ω is given by:

$$\omega = \frac{1}{\mu} + \frac{\rho}{2\mu(1-\rho)} \quad (2.1)$$

The waiting time in the queue, ω_Q is given by:

$$\omega_Q = \frac{\rho}{2\mu(1-\rho)} \quad (2.2)$$

M/D/c Queue

This model is an extension of the previously described M/D/1, where the number of servers c is greater than 1. Like M/D/1, this model was first published in 1909 by Erlang. Arrivals are determined by a Poisson process and job service times are fixed (deterministic). An M/D/ c queue is a stochastic process whose state space is the set $0, 1, 2, 3, \dots$, where the value corresponds to the number of customers in the system, including any currently in service. Arrivals occur at rate λ according to a Poisson process and move the process from state i to $i + 1$. Services times are deterministic time D (serving at rate $\mu = 1/D$). The servers, c , serve customers from the front of the queue, according to a FIFO discipline. Once the service is complete, the customer leaves the queue, decreasing the queue length by 1. The queue capacity is infinite.

Performance Metrics

When $\rho = (\lambda D)/c < 1$, the waiting time distribution has distribution $F(y)$ given by:

$$F(y) = \int_0^\infty F(x + y - D) \frac{\lambda^c x^{c-1}}{(c-1)!} e^{-\lambda x} dx, \quad y \geq 0 \quad c \in N. \quad (2.3)$$

Below, L_q represents the number of customers in the queue, while L_s represents the number of customers in the entire system:

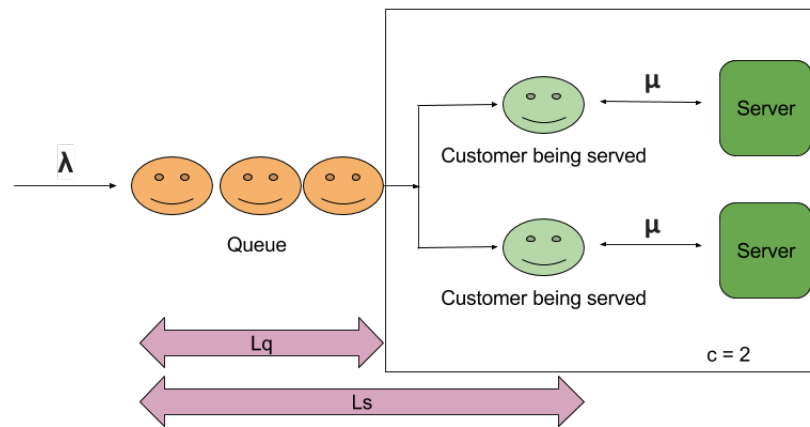


Figure 2.4: M/D/c Queue model with 2 servers. The service rate is given as $1/D$, where D is the fixed service rate time.

M/M/1 Queue

The M/M/1 queueing model follows a Poisson distribution of arrival (λ) and service rate (μ) with a single server [42]. Below, L_q represents the number of customers in the queue, while L_s represents the number of customers in the entire system.

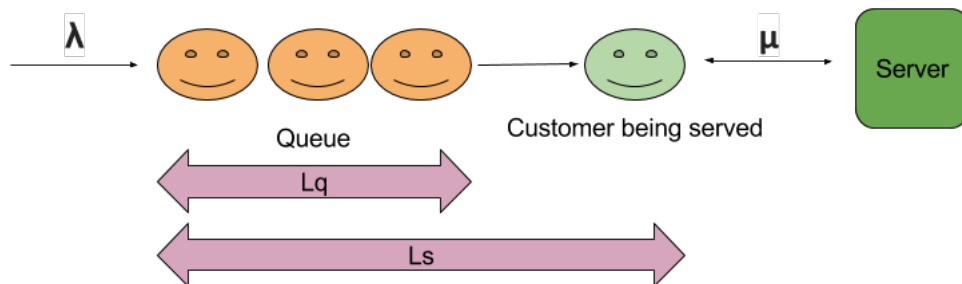


Figure 2.5: M/M/1 Queue example

It is a continuous-time Markov chain, assuming that the M/M/1 queue-size process

starts at state 0 (i.e. no one in the queue). It will remain in state 0 for a period of time that is exponentially distributed with parameter λ then it moves to state 1.

Performance Metrics

The waiting time in the system, ω is given by:

$$\text{Wait in the System} = \omega = \omega_q + \frac{1}{\mu} \quad (2.4)$$

The waiting time in the queue, ω_q is given by:

$$\text{Wait in the Queue} = \omega_q = \frac{W_q}{\lambda} \quad (2.5)$$

where L_q is:

$$\text{Number in the Queue} = L_q = \frac{\rho^2}{1 - \rho} \quad (2.6)$$

where ρ is:

$$\rho = \frac{\lambda}{\mu} \quad (2.7)$$

Average time spent waiting:

$$\frac{\rho}{\mu - \lambda} \quad (2.8)$$

M/M/c Queue

The M/M/c queue is a generalization of the M/M/1 queue to the case of c servers. As in M/M/1, for an M/M/c queue, the buffer is infinite and the arrival process is Poisson with rate λ . Service time of each of the c servers is exponentially distributed with parameter μ . As in the case of M/M/1 we assume that the service times are independent and are independent of the arrival process [42]. Utilization (ρ) for the M/M/c queue model is defined as:

$$\rho = \frac{\lambda}{c * \mu} \quad (2.9)$$

Below, L_q represents the number of customers in the queue, while L_s represents the number of customers in the entire system:

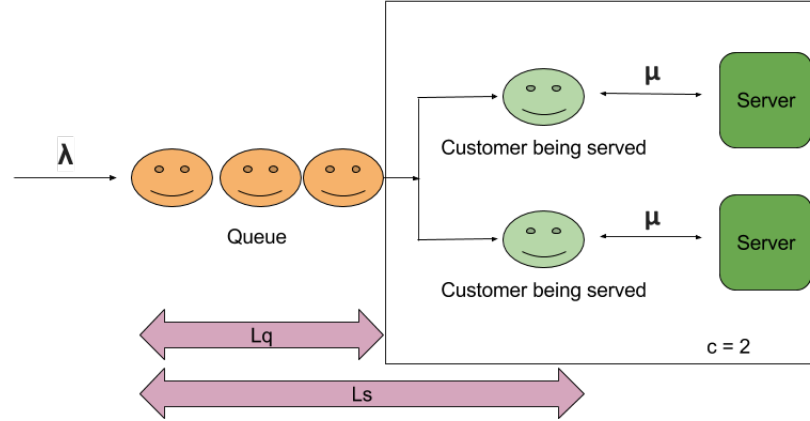


Figure 2.6: M/M/c Queue model with 2 servers. The queue is the same as an M/D/c queue, with the one exception being that the service rate is not deterministic.

Performance Metrics

The probability that an arriving customer is forced to join the queue (all servers are occupied) is given by:

$$C(c, \lambda/\mu) = \frac{\left(\frac{c\rho}{c!}\right)\left(\frac{1}{1-\rho}\right)}{\sum_{k=0}^{c-1} \frac{(c\rho)^k}{k!} + \left(\frac{c\rho}{c!}\right)\left(\frac{1}{1-\rho}\right)} = \frac{1}{1 + (1-\rho)\left(\frac{c!}{(c\rho)^c}\right) \sum_{k=0}^{c-1} \frac{(c\rho)^k}{k!}} \quad (2.10)$$

The average number of customers in the system (in service and in the queue) is given by:

$$\frac{\rho}{1-\rho} C(c, \lambda/\mu) + c\rho \quad (2.11)$$

The average total time a customer spends in both the queue and in service is given by:

$$\frac{C(c, \lambda/\mu)}{c\mu - \lambda} + \frac{1}{\mu} \quad (2.12)$$

2.2.4 Takeaways

Our system is focusing on food establishments, and food establishments generally follow a traditional queueing discipline. For our system, all queues will be utilising a first-in-

first-out discipline.

The arrival rates for our queueing model will be random, due to the fact that it will be determined using Google's Popular Times feature. When it comes to the service rate, there is no way to efficiently determine a dynamic service rate for each type. However, the library used to create our queue, "Ciw" [33], allows us to get a random service rate using its network methods. The service rate will be random, with the average of the sampled service rates equalling the value we pass into the method. This value will be obtained by observing the food establishments. The reasoning for this is that different customers will have different orders of varying size, meaning that the rate at which they are served should be different.

Finally, our number of servers is unknown at this point, so the best choice would be to assume that it could be 1 or more. Therefore the queueing model that meets our needs the greatest is the M/M/c queue, where the arrival and service rate follow a Poisson distribution process, and the number of servers is 1 or more.

2.3 Predictive Analytics

Predictive analytics involves creating predictions about future or unknown events using current or historical data fed into predictive models. These predictive models consist of one or more machine learning algorithms. Predictive analytics is used in healthcare, marketing, insurance, telecommunications, retail, and many more fields. Each prediction depends on multiple factors, for example every characteristic known about a patient [31]. The predictive model that will be used by my web server will involve reading in user inputted wait times as well as simulated wait times created by a queueing model. The predictive model examines this data and looks for patterns in an attempt to produce an accurate prediction. Due to this, the model will incorporate regression-based algorithms.

2.3.1 Regression

Regression is a set of statistical processes for estimating relationships among variables. Modelling refers to the development of mathematical expressions that describe in some sense the behaviour of a random variable of interest [28]. This variable can be anything, be it a student's grade in a module, the price of oil, or in our case, a queue time for a

food establishment. In all cases, this variable is called the dependent variable and is generally denoted by y . Other variables which are thought to provide information on the behaviour of the dependent variable are incorporated into the model as predictor or explanatory variables. These variables are called the independent variables and generally denoted by X [28]. The focus is on the relationship between the dependent variable and one or more independent variables (occasionally referred to as “predictors”) [41].

2.3.2 Linear Regression

Linear regression is one of the most common statistical tools for modelling relationships between independent variables and some desired value (our dependent variable). When there is only one explanatory variable, the algorithm used is referred to as “simple linear regression”. Simple linear regression can be defined by the following equation:

$$h_{\theta}(x) = \theta_0 + \theta_1 x \quad (2.13)$$

The goal of the equation is to select θ_0 and θ_1 that minimises $J(\theta_0, \theta_1)$ where $J(\theta_0, \theta_1)$ is the cost function and is defined by:

$$J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 \quad (2.14)$$

The above is the simplest form of linear regression, that does not see much use in the real world. Most real world uses utilise multiple linear regression, where there is more than one predictor variable. Multiple linear regression’s hypothesis can be defined by:

$$h_{\theta}(x) = \theta^T x \quad (\text{with } \theta, x \text{ is now } n + 1 - \text{dimensional vectors}) \quad (2.15)$$

As before, the goal is to select θ that minimises $J(\theta)$ where $J(\theta)$ is the cost function and is defined by:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 \quad (2.16)$$

Linear regression is extremely useful if the goal is prediction, or forecasting, or error reduction. It can be used to fit a predictive model to an observed data set of y and X values. After developing such a model, if an additional value of X is then given without its accompanying value of y , the fitted model can be used to make a prediction of the value of y [38].

2.3.3 Ridge and Lasso Regression

Ridge Regression is the name given to Tikhonov Regularization in the statistical realm. It gets its name from the diagonal of ones in the correlation matrix, as it may be thought of as a ridge. It is a technique for analysing multiple regression data that suffer from multicollinearity. Multicollinearity, or collinearity, is the existence of near-linear relationships among the independent variables [25].

Lasso regression is quite similar to ridge regression. It performs both variable selection and regularization in order to enhance prediction accuracy. It was introduced by Robert Tibshirani in 1996 based on Leo Breiman's non-negative garrotte. It was originally created for use in least squares models, but is easily extended to a wide variety of statistical models.

Given a linear regression with standardized predictors x_{ij} and centred response values y_i for $i=1,2,\dots,N$ and $j = 1,2,\dots,p$, the lasso solves the problem of finding $\beta = \{\beta_j\}$ to minimize [34]:

$$\sum_{i=1}^N (y_i - \sum_j x_{ij} \beta_j)^2 + \lambda \sum_{j=1}^p |\beta_j|. \quad (2.17)$$

2.3.4 Random Forest

Random forests are an ensemble learning method that operate by constructing a multitude of decision trees and outputting the mean prediction of the individual trees [22]. The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners [40]. Given a training set $X = x_1, \dots, x_n$ with responses $Y = y_1, \dots, y_n$, bagging repeatedly (B times) selects a random sample with replacement of the training set and fits trees to these samples [40]:

For $b = 1, \dots, B$:

1. Sample, with replacement, n training examples from X, Y ; call these X_b, Y_b .
2. Train a classification or regression tree f_b on X_b, Y_b .

After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x' [40]:

$$f = \frac{1}{B} \sum_{j=1}^B f_b(x') \quad (2.18)$$

The number of trees necessary for good performance grows with the number of predictors. The best way to determine how many trees are necessary is to compare predictions made by a forest to predictions made by a subset of a forest. When the subsets work as well as the full forest, you have enough trees [24].

2.3.5 K-Nearest Neighbours

K-nearest neighbours (KNN) is a pattern recognition method used for both regression and classification. In KNN regression, the output is the property value for the object. This value is the average of its k nearest neighbours [37]. KNN has been used in statistical estimation since the beginning of 1970s [29]. KNN can be quite easy to fit, and one only needs to estimate a small number of coefficients. The KNN method is as follows [32]:

1. Assume a value for the number of nearest neighbours K and a prediction point x_0
2. KNN identifies the training observations N_0 closest to the prediction point x_0
3. KNN estimates $f(x_0)$ using the average of all the responses in N_0 , i.e.:

$$f(x_0) = \frac{1}{K} \sum_{x_i \in N_0} y_i \quad (2.19)$$

A commonly used distance metric for continuous variables is Euclidean distance, which is defined as the straight-line distance between two points. Euclidean distance can be defined as:

$$\sqrt{\sum_{i=1}^k (x_i - y_i)^2} \quad (2.20)$$

Choosing K can be a challenging task. A small value for K provides the most flexible fit, which will have low bias but high variance. This variance occurs because the prediction in a given region is dependent on just one observation. Contrarily to this, choosing a large value for K will provide a smoother and less variable fit. The prediction in a region is an average of several points, and so changing one observation has a smaller effect.

2.3.6 Takeaways

Each of the described algorithms all have their strengths, and can provide fast and timely calculations for our system. Each of the algorithms will be used to calculate predictions. The average of each algorithm's prediction will give us our overall prediction. The reasoning behind this choice will be discussed in Chapter 3 - Design & Implementation.

2.4 Cloud Hosting

The user application will communicate with a server that will perform the necessary operations to create a prediction. In order for this communication to take place, the server needs to be hosted on a virtual machine, on the cloud. Cloud hosting will allow for easy access and take some of the heavier operations off the hands of the application. Two options came to mind straight away when thinking of cloud hosting. These are SCSSNebula and Amazon Web Services (AWS).

2.4.1 SCSSNebula

SCSSNebula is an enterprise cloud and data centre virtualisation Infrastructure as a Service (IaaS) platform for use in research and teaching in the School of Computer Science and Statistics. It is based on OpenNebula, a cloud solution founded by Ignacio M. Llorente and Ruben S. Montero in 2005. Some of the design principles of OpenNebula are openness of the architecture, flexibility, portability, stability, scalability, and simplicity. Following years of research and development in efficient and scalable management of virtual machines on large-scale distributed infrastructures, OpenNebula has now become one of the leading supplier of cloud hosting solutions. OpenNebula currently have numerous high-profile clients such as Hitachi, BBC, Unity, and Deloitte [6]. SCSSNebula could be accessed via the college network using an online Sunstone portal, available at <https://nimbusselfservice.scss.tcd.ie>.

In order to create a new virtual machine (VM), users can simply choose a predefined template from the templates menu, give the VM a name and define the number of instances they wish the VM to have. Any images associated with these VMs are non-persistent i.e. all changes will be lost once the VM is deleted. Each new VM is given an IP address, only routable via the college network. In order to communicate with

the VM outside of the college network, a proxy must be configured.

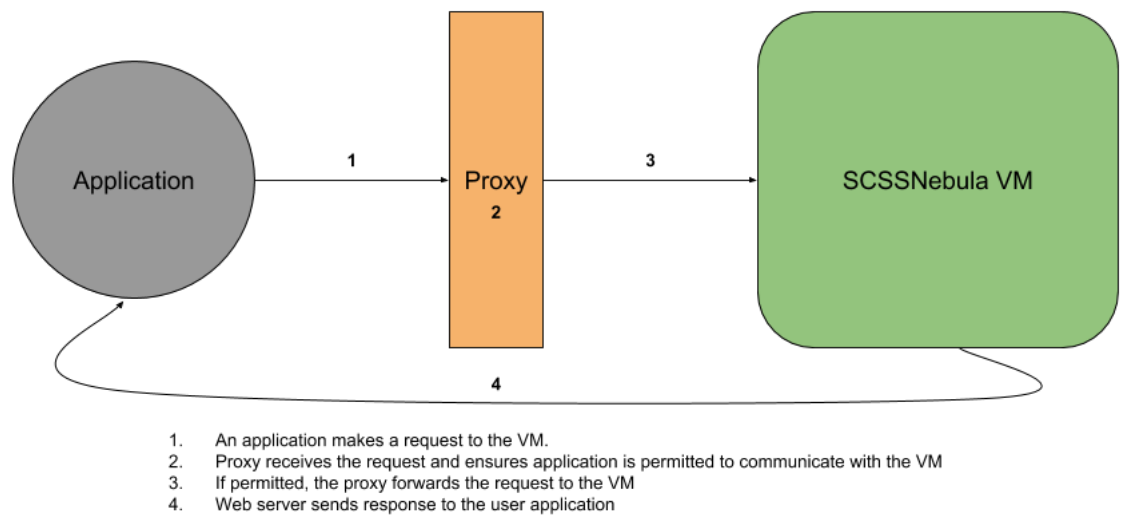


Figure 2.7: The proxy acts as a “barrier” between the outside world and the VM

2.4.2 Amazon Web Services

Amazon Web Services (AWS) is a subsidiary of Amazon.com that provides on-demand cloud computing platforms to both individual and commercial users. The service was launched as a public beta in March 2006. Within AWS, Amazon Elastic Compute Cloud (Amazon EC2) provides scalable computing. EC2 removes the need to implement hardware, meaning applications can be built and deployed quickly and efficiently. EC2 provide virtual computing environments, known as instances. Amazon EC2 reduces the time required to obtain and boot new server instances to minutes, allowing users to quickly scale capacity. Similar to SCSSNebula, predefined templates allow users to choose the operating system of their VM. Users may also choose different amounts of storage and memory for their VMs, and authentication is used via “*key pairs*” (AWS keeps track of the public key while the user keeps the private key in a secure location [2]). Security is also customisable, with users having the ability to create “*security groups*”. Within these groups, users can control access permissions, meaning total control over who can communicate with their VM [2].

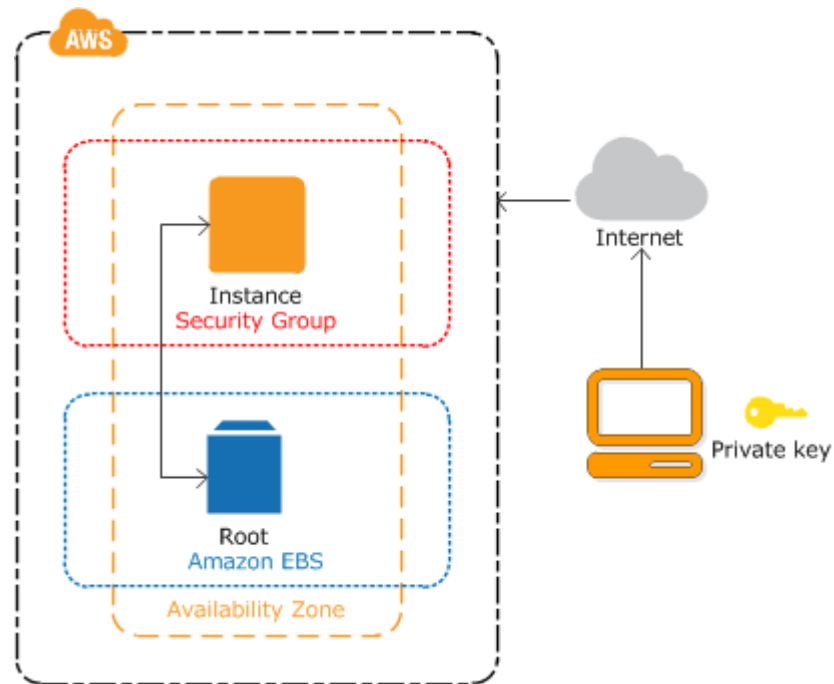


Figure 2.8: An overview of the Amazon EC2 instance security [8]

Amazon themselves moved their retail site to AWS and EC2 instances in November 2010. Like OpenNebula, they boast a wide array of successful customers, namely Netflix, Lamborghini, NASA, and AirBnB among them [2].

2.4.3 Takeaways

The choice of which service to use here was a difficult one. However, the decision was made to go with AWS, as it is used more widely than SCSSNebula. The use of AWS means that there is not the need to configure proxy allowances for the client and server to communicate. AWS also offers better security, including AWS Shield [1].

Chapter 3

Design & Implementation

In this chapter, we will take a look at the design and implementation of the solution devised following the research mentioned in Chapter 2. We will start by taking a look at the original plan for the solution that was theorized before the research was complete. We can then follow the evolution of the idea right up to the final implementation.

Within this chapter and beyond, there are several words whose definitions may be unclear or unfamiliar. These words are used in this dissertation with the following definitions:

Queue time - The time from when the customer enters the queue, to when they receive their food.

Cafe - Food establishment that serves coffee and light snacks e.g. Starbucks/Costa.

Deli - Food establishment in which servers go step by step with customers when preparing orders e.g. Centra Sandwich Counter/Burrito restaurant.

Takeaway - Food establishment that produces food that is generally eaten outside the establishment e.g. Chinese Takeaway/Chip shop.

Fast food - Food establishment that produces fast food e.g. McDonald's/KFC.

3.1 Design

3.1.1 Overview

Initially, the plan was to create an application that users can use to see queueing times for nearby food establishments. The application would take in a user's input and pass it to a predictive model that would then run various algorithms to produce a queue time prediction. The overall UI of the application was originally planned to have a

similar style to Google Maps [9], whereby users could see themselves on a large map and also see nearby food establishments. Users could then click on a food establishment they wanted to get a prediction for, enter in a few more clarifying details and then be presented with a queueing time prediction.

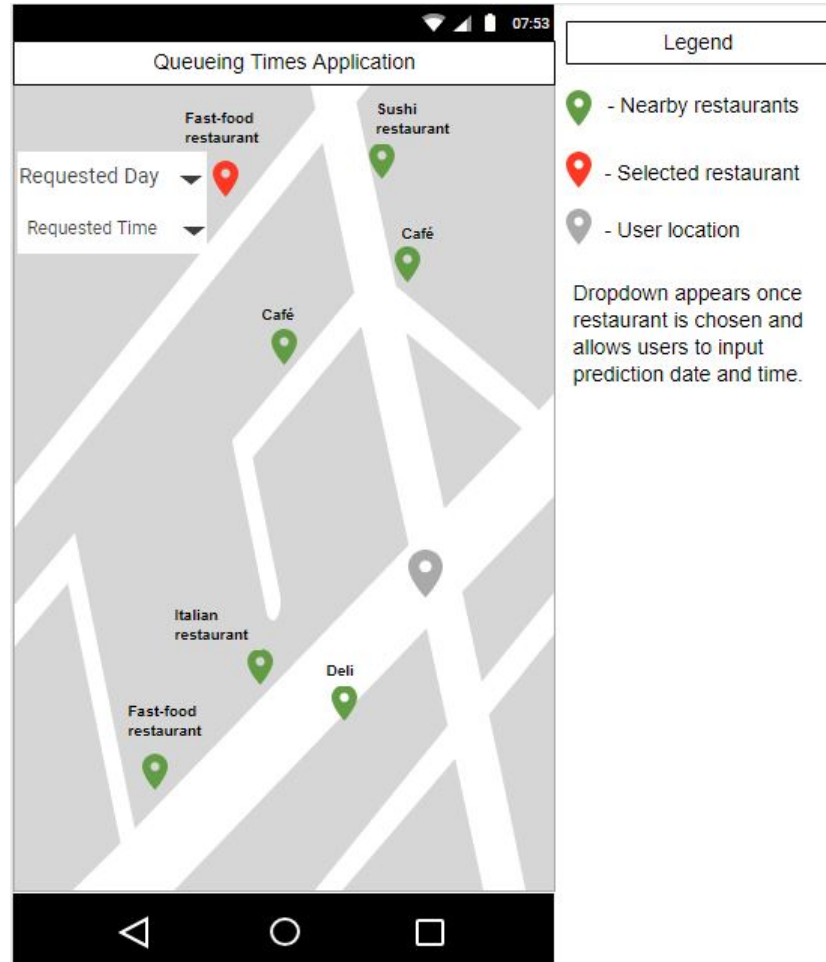


Figure 3.1: The original concept for the application UI

The entire operation would be performed on the user's smartphone. The application would be developed as an Android application, due to my background in Java development. Another reason for choosing Android is that of the 432 million smartphones sold in the final quarter of 2016, 81.7% ran Android [35]. This would give the application greater exposure to potential users.

The main flaw of this approach is the fact that all calculations and functionality is carried out on the user's smartphone itself. This places too much responsibility on the user's device, with less powerful devices taking more time to create a prediction. This also makes the application quite large, which again can alienate certain users who do not have the storage available to download large applications. In order to combat this,

the decision was made to create a web server that will handle the calculations and that the application will communicate with. This decision to create a web server effectively turns the mobile application into a basic HTTP client application.

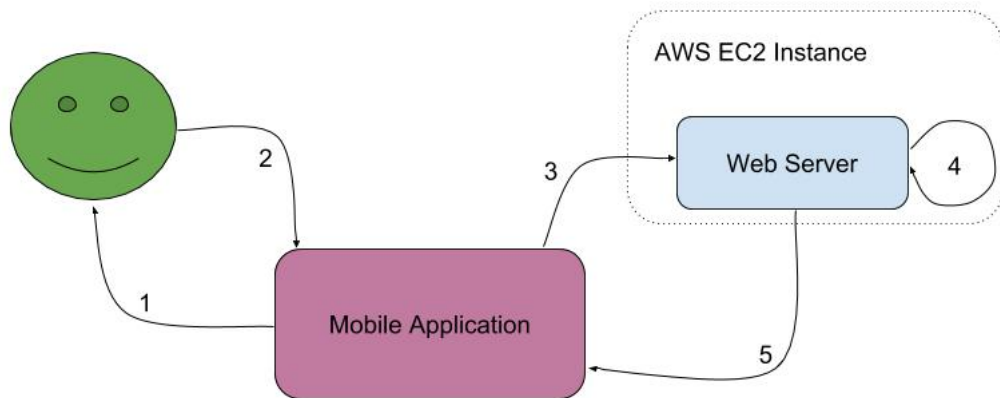


Figure 3.2: An overview of a user process

The above shows the following:

1. The mobile application displays the UI to the user. This can be the initial requests screen, or the calculated prediction.
2. The user makes a request using the application UI.
3. The application takes the user input and formats it into a structure that can be read by the web server. It then forwards the request to the web server.
4. The web server scrapes the necessary information from the request and feeds it into the predictive model.
5. The web server returns the prediction to the application, to be displayed with step 1 to the user.

The application allows users to get a prediction from the web server and submit their own queueing time for later use in the predictive model.

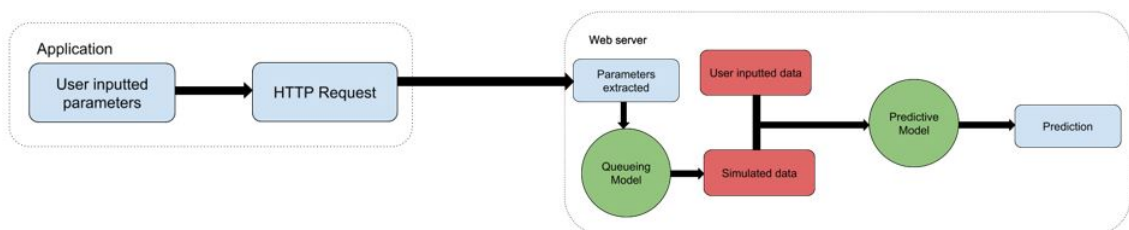


Figure 3.3: Design Overview

3.1.2 Application

As previously discussed, the application was originally planned to perform all calculations. Once the decision was made to create a web server, the goal of the application changed quite dramatically. It now acted as a middle man between the user and the web server. On start-up, the UI displays several options for the user. Users can choose between searching for a food establishment by name, searching nearby food establishments, or submitting their own wait time.

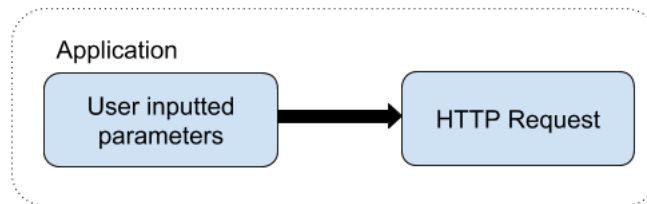


Figure 3.4: Basic overview of application's function

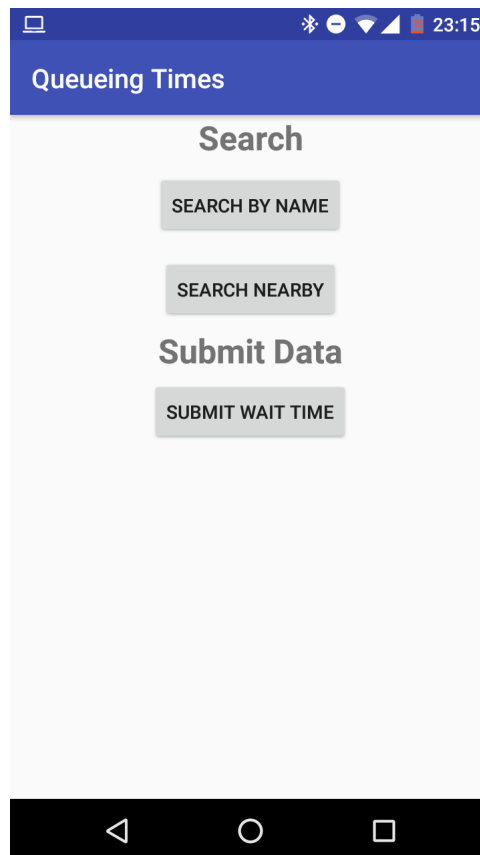


Figure 3.5: The initial UI displayed to the user when they start the application

Once users select a button, the relevant UI screen is displayed to the user. If the user selects the “Search by name”, the user is prompted to enter the food establishment

name, the type of food establishment (from a drop-down containing “cafe”, “deli-style”, “fast food”, or “takeaway”), the time they wish to get a prediction for, and a day they wish to get a prediction for. Users can then press the “Get times” button in the bottom right. Once the button is pressed, the information is formatted and sent to the web server to create a prediction.

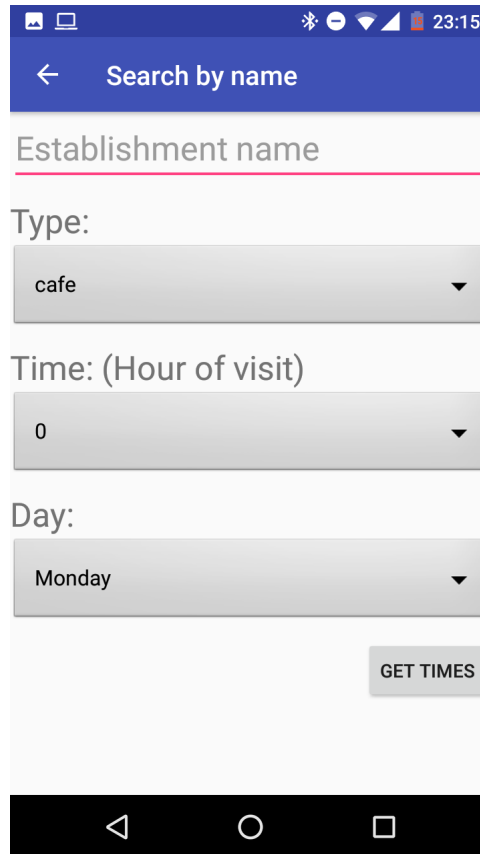


Figure 3.6: Searching by name screen



Figure 3.7: Searching nearby food establishments screen

Selecting the button “Search Nearby” brings the user to the screen shown in Fig 3.7. Once this option is chosen, a request is sent to the Google Places API [10] to retrieve a JSON formatted response that contains all food establishments within a 500m radius of the user’s location. A sample JSON response is located in Appendix A. The nearby food establishments are displayed with their name and full address in various buttons. Once a button is chosen, the relevant information is displayed to the user on a screen that is similar to Fig 3.6, with food establishment name, time, and day filled in already. The only field the user has to input is the establishment type, in order to help the queueing model on the web server.

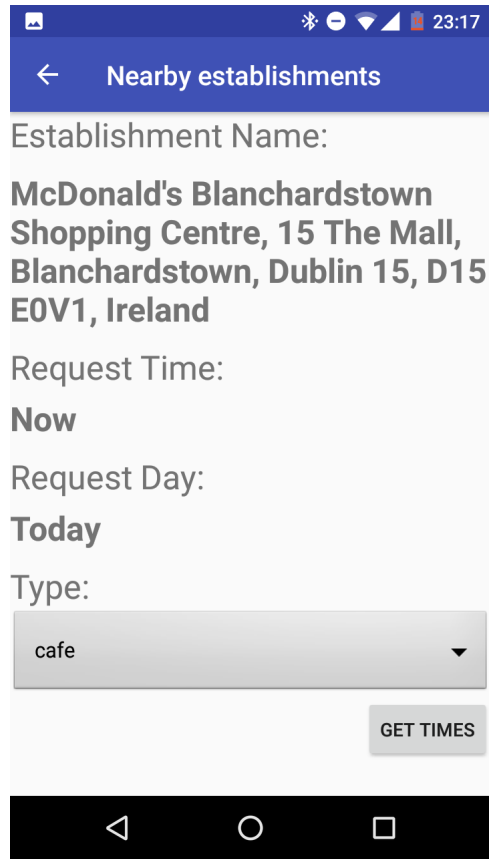


Figure 3.8: The chosen food establishments details before making the request

If a user selects the “Submit wait time” button, the application displays the screen seen below. Users input the name of the food establishment, the time they began waiting, the day they waited, and how long (in minutes) they waited in the queue. Once submitted, the application sends the formatted information to the web server to be stored for later use in creating predictions.

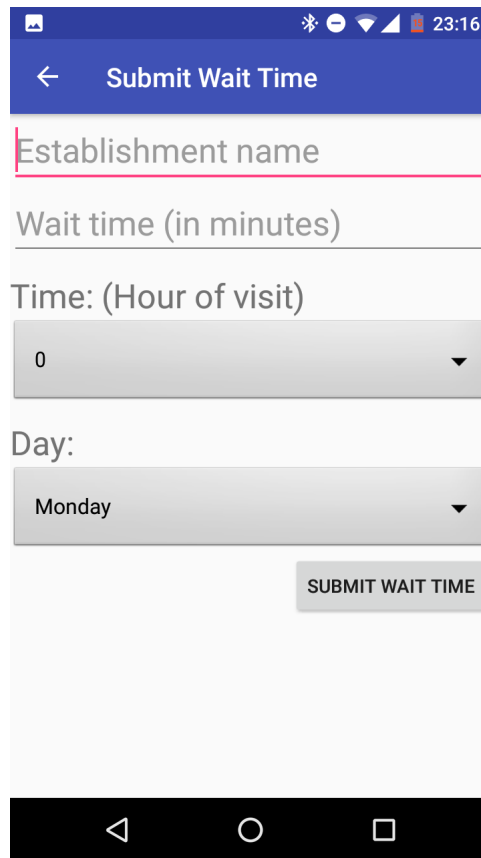


Figure 3.9: The submit wait time UI screen

3.1.3 Web Server

The web server can be further split into two sections, the queueing model and the predictive model. The queueing model creates the simulated data that is combined with user-inputted data that will be passed into the predictive model.

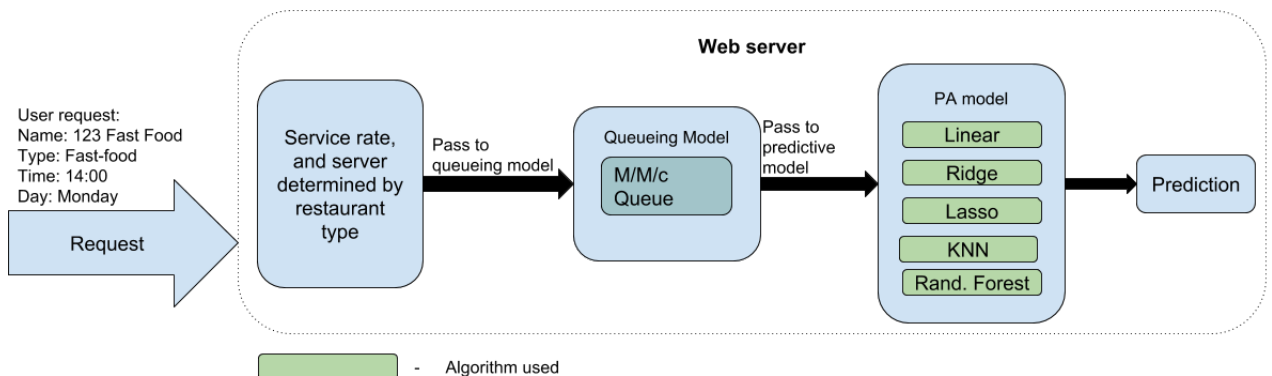


Figure 3.10: A request's journey from request to prediction

Queueing Model

As stated in Chapter 2 of this dissertation, there are many types of queueing models that can be used to describe a queue. Due to there being several different types of food establishment, the decision was made to create a queueing model that used the M/M/c queue. This was due to the versatility of this type of queue. Arrival times and service rates are given in a Poisson process (random). The number of servers can vary, with somewhere like a deli possibly having only one server, but somewhere like a fast-food food establishment having several servers available. Arrival times are determined using Google Places popularity feature, which tracks how popular a location is at any given time. The average value of the service rate is determined by the web server looking at the establishment type chosen by the user. This was added to compensate for the different service speeds of different types of food establishments (e.g. a deli would have a faster service rate than a takeaway). Originally, the responsibility of setting the number of servers lay with the user, but after careful consideration, it was decided that this number would be based on the establishment type as well. Admittedly, this can be flawed, but it allows a generic number of servers be chosen for each establishment type.

Predictive Model

The predictive model contained a variety of regression algorithms, which are discussed at length in Chapter 2 of this dissertation. The idea here was to create a model that would be quick and efficient in returning a prediction to the user. Another goal was to be as accurate as possible. In order to achieve these goals, all types (cafe, takeaway, deli-style, and fast food) use a combination of linear regression, lasso regression, ridge regression, k-nearest neighbours (KNN) regression, and random forest (RF) regression. The reasoning behind this choice will be further discussed later on in this chapter, within the Implementation section.

The data that is fed into this predictive model is a combination of the data simulated by the queueing model and historical data inputted by the user.

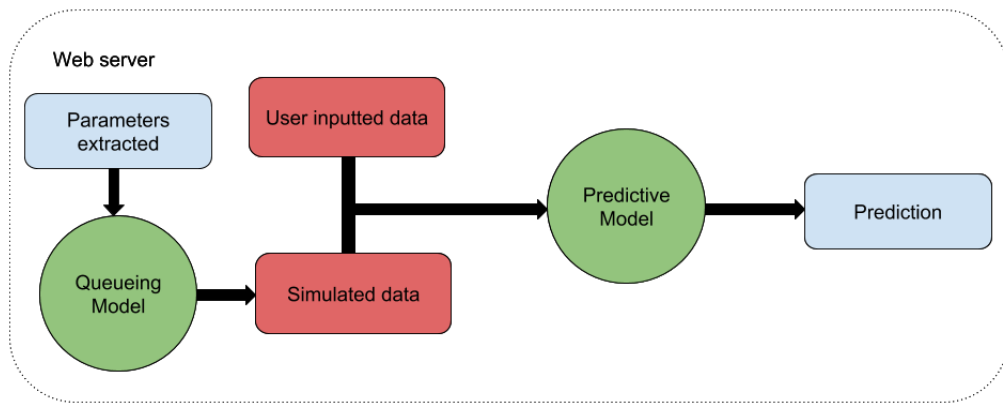


Figure 3.11: The inner workings of the web server

3.2 Implementation

This section will cover how the solution was created, including a look at the libraries chosen and techniques used to build the final product. The section will also include a look at the process followed in completing the project.

3.2.1 Where to start?

Once the appropriate research was completed and the design drawn out, there was a decision to be made. The first step to building this solution was crucial as it was desirable that the project began as it meant to go on - productively. The application was going to be the main point of interaction for end users, so there was an argument to be made to get that up and running first. Testing the functionality would be pointless, however, due to the need for the application to communicate with a server and react to the various responses it would receive. Due to this, the decision was made to make a start on the web server, as this would be easy to test with a dummy HTTP client.

3.2.2 Web Server

The web server began as a simple HTTP server using the Flask library [7]. The server would listen on port 80 for incoming requests. Flask allows application developers to determine the type of HTTP requests that are accepted by the server, which deals with the risk of inappropriate usage. The first implementation of the server accepted GET requests only. These requests contained parameters that the server would pass to the queueing and predictive model. Within Flask, the incoming request is stored

in a variable named “request” with application developers being able to access various attributes of that variable.

Attribute	Description
method	The request method (e.g. “GET” or “POST”)
url	The reconstructed current URL
base_url	Like url but without the querystring
args	The parsed URL parameters (the part in the URL after the question mark)
form	The form parameters
values	A combination of args and form

Figure 3.12: A sample of attributes of a Flask request object [4]

With this first implementation, the incoming requests were sent using the Postman application [14]. This application allows for the testing of APIs and servers, allowing developers to customise requests and send them to any URL they wish. The first requests sent to the web server contained five different variables:

1. restaurant_name - The name (including address) of the requested food establishment
2. service_rate - The rate of service of the food establishment (e.g. slow, medium, fast)
3. request_time - The time that the prediction should be made for
4. request_day - The day that the prediction should be made for
5. servers - The number of servers at the requested food establishment

The first request sent via Postman to the web server was a GET request in the following form:

```
server_address:80?restaurant_name=Centra Westland Row, Dublin 1&
service_rate=fast&request_time=14&request_day=Thursday&servers=1
```

The server sees that the request is a GET request and passes it into the “if” clause in

the `start()` function in our server's code. Using the `request.values` attribute, the parameters sent in the request are extracted and placed into their own variables. Following this, the code begins to format the queueing model variables.

As mentioned in Chapter 2 - Research, in the "Queueing Models" subsection, there were many choices for a queueing model. It was decided that an M/M/c queue would be used. Within this model, the arrival rate (λ) is random, the service rate (μ) is random, and the number of servers is ≥ 1 . The arrival rate was determined using python libraries "*python-google-places*" [16] and "*populartimes*" [11]. To get the arrival rate, an API request is made using "*python-google-places*". This request returns the raw JSON response returned by the Google Places API. The ID of the place is then extracted and used in a "*populartimes*" request. This request returns a JSON response that is quite similar to the "*python-google-places*" response, with the addition of an array that contains the popular times for every single hour in every single day. This is a JSON array that consists of each day and its corresponding popular times. The popular times are returned as an array of 24 integers representing the 24 hours in the corresponding day. These integers range between 0 and 100, where 0 indicates nobody being present and 100 indicates an extremely busy period. The service rate was obtained using Ciw's network function, whereby we set the average value of the service rate. Sampled rates give an average of the set value. This was used in order to simulate real-world establishments, as no two orders are the exact same or follow the same process.

In order to prevent needless requests being sent to the Google Places API, once the initial request is made, the popular times are written to .csv files stored in a "times/createddata/" directory. Subsequent arrival rates can be obtained by reading a local .csv file. This will cut down on execution time and increase overall throughput. Once the popular times are obtained, the popular time allocated to the request's time is taken. This value is converted to a percentage value (e.g. a value of 46 would result in an arrival rate of 0.46).

The service rate was originally left to the user to decide. This can be confusing and vary from user to user. The service rate could be "fast", "medium", or "slow". This choice affects the service rate that is passed into the queueing model. This was also the case for the number of servers. This can cause an issue if a user is attempting to get a prediction for a location they have never visited before. In this scenario, the user

will not be able to correctly enter the number of servers, which could possibly result in an incorrect prediction.

Due to this, a new parameter was introduced in an attempt to stifle the aforementioned confusion. Rather than sending the service rate and number of servers, the request would contain an establishment type that could be used to determine the service rate and number of servers. The establishment type can be one of the following: “cafe”, “deli”, “fast food”, or “takeaway”. These types result in an average service rate of 0.25, 0.5, 0.41, or 0.41 respectively. These times were chosen as service rate can be converted to minutes by dividing 1 by the service rate. For our types, the average amount of minutes for service was 4 minutes, 2 minutes, 2.5 minutes, and 2.5 minutes respectively. These service rates averages were chosen based on real-world visits to each establishment types. Originally, there was also a fifth establishment type: “restaurant”. This type was removed as in most cases for traditional sit-down restaurants, there is no queue for tables and a majority of the time customers have a reservation, thus voiding the need to queue. These people would also be included in Google’s Popular Times feature, adding erroneous values to the arrival rate. As stated in the introduction, the goal was to create accurate predictions for walk-ins to food establishments. For all types the number of servers is set to 1.

Once the arrival rate, service rate, and number of servers are determined, a variable is created to represent the queue. This variable is an instance of the “Queue” class that was created for this system. This class is quite simple, with attributes representing the arrival rate, service rate, and number of servers. The class also has a function that is used to generate a simulated queue, called “gen_queue()”. This function utilises the “Ciw” queue network simulation library [33]. This library requires the creation of a network that incorporates the arrival rate, service rate, and number of servers. The network then creates different arrival rates and service rates for the simulated queues, ensuring that the averages are equal to the queue variables attributes. The network fully describes the food establishment. A simulation object is then created, which will act as the engine that will run the simulation. We then run the simulation for an equivalent of 70 minutes, using Ciw’s “simulate_until_max_time()” function. The simulation is run for 70 minutes so that we have an observation period of 60 minutes, with a 5 minute warm-up and a 5 minute cool-down. The warm-up ensures that the simulation is up and running and “in the swing of it” by the time that the observa-

tion period begins, while the cool-down ensures that no simulated customers are stuck in the simulation when we collect the results [18]. We then extract the wait times from the results and calculate the average wait time. This process is then repeated 99 more times in order to create an adequate amount of data to feed our predictive model. This process is repeated for each hour and day of the week, in order to get sufficient data to feed into the predictive model. The original idea was to just run the previously described process for the request's day, but the decision was made to simulate an entire week, so as to get a big enough dataset to feed into the predictive model.

The next step that the code takes involves feeding data into the predictive model. The predictive model in this system uses regression algorithms, due to the fact that we will be attempting to get a single value by observing past data. The predictive model takes in the simulated wait times, as well as the establishment type, requested day, requested time, and food establishment name. The server then checks to see if there is any user-inputted data that can be used in the prediction. These regression algorithms were ran using the scikit-learn library [26]. The results and time details of each entry are placed into separate lists, to be turned into NumPy[13] arrays for use in regression algorithms. The prediction was created by getting the average of the predictions produced by linear regression, ridge regression, lasso regression, random forest regression, and k-nearest neighbours regression. This was due to the fact that some algorithms produced outliers with their predictions. The use of the other four algorithms acted as a sort of damage control against these outliers. This approach proved to be quite accurate. The graphs of the overall results of this approach, as well as graphs comparing the individual performance of each algorithm can be found in Chapter 4 - Evaluation.

However, due to the fact that the amount of test data was quite small, it was decided that a mathematical approach should also be considered for choosing which algorithms can be used. Mean Absolute Error (MAE) was chosen as the metric that would allow us to choose which algorithms would perform best. MAE measures the average magnitude of the errors in a set of predictions. It's the average over the test sample of the absolute differences between predictions and actual observation where all individual differences have equal weight [36]. The dataset would be split using k-folds validation and each fold would have its MAE calculated. Each algorithm was fitted with the training data and predictions were compared to the test data. This gave the choice of

algorithms concrete proof of being the best choice for creating predictions, based on each algorithms MAE.

Both of the previous configurations of the predictive model were tested against real-world algorithms. Both performed quite well, with the “Average of all algorithms” approach having an average absolute error of 2.25 minutes, and the “MAE” approach having an average absolute error of 2.75. The average absolute error for each type can be seen in the following graph:

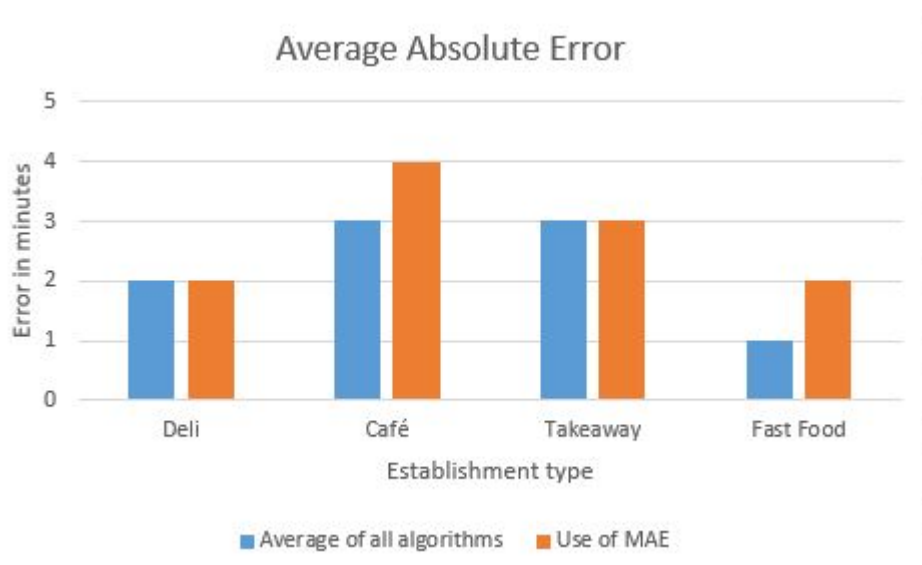


Figure 3.13: The average absolute errors of both methods of choosing algorithms

As you can see from the graph, the “Average of all algorithms” approach is the superior choice. This meant that the configuration that used the average method would be the configuration used going forward. The graphs displaying the results of the MAE methodology can be found in Appendix B.

3.2.3 Application

The client application involved perhaps the most difficult choice to make, as the language and style of application was debated for a long period of time. Originally, as stated in the previous section, the client application was simplified to utilising Postman [14]. The decision was made to go with an Android application [3]. The main reason for choosing Android was due to the fact that 81.7% of new mobile phones at the end of 2016 were Android devices [35].

The first step in creating this application was to create an application that could communicate with a server. This application needed to both send requests as well as receive

and display responses. The application was created using Java with XML bindings. The frontend would be written using XML with each XML element having an ID that would allow it to be manipulated by the Java code that powered the backend. As previously stated, the goal was to keep the required computational power of the application low. In order to send HTTP requests, the application used Volley [17]. Volley is a HTTP library that makes networking with Android apps easy and straightforward. The application displayed a form to users in which they could enter in a food establishment name, establishment type, a request time, and a request day. Using the IDs of the various input forms, the data that was inputted by the user could be found and placed in various Java variables. These variables were then used to compile a HTTP request. Beneath the form, there is a button that the user presses to send their request. Once this button is pressed, Volley takes the HTTP request and sends it to the server. By default, Volley has a retry policy whereby if a response is not received within 2.5 seconds, Volley performs the request again, and so on and so forth until either a response is received or the maximum amount of retries is reached. Due to the fact that the server needs to perform some time-consuming calculations, the retry policy time limit needed to be increased, so therefore the time was set to one minute.

Following the completion of this, the next feature to be implemented was the “Search Nearby” feature of the application. As mentioned in the previous section “Design”, the original idea for this feature was to create a “Google Maps-esque” screen where users could see nearby food establishments on a map and select their desired food establishment. This idea was quickly changed when it became apparent that the speed and performance of the application would be entirely dependent on the user’s smartphone. This would involve real-time rendering of the map, as well as constantly tracking the user’s location. Not only would this use up the phone’s computing resources completely, it would also be a drain on the smartphone’s battery. By reimagining the feature and moving the heavier calculations to the web server, the speed and performance becomes more uniform across all users, regardless of the hardware running the client application.

The new design of the feature meant a reinvented main screen for the application. Once users launched the application, they are now met with a home screen with two buttons stating: “Search by name”, “Search nearby”, respectively. Selecting “Search by name” takes the user to the form described previously. Selecting “Search nearby” activates the new implementation of the feature discussed in the previous paragraph.

This new implementation involves getting the user's location and performing a request (using Volley again) to the Google Maps API to locate nearby food establishments. This dealt with the issue of continuously tracking user's locations by only noting the location once, when the "Search nearby" button is selected. By "nearby" the decision was made to search a radius of 500m, as this seems like a reasonable "nearby" distance to travel to a food establishment, as it would take roughly 5 minutes. Once a response is received, a screen is created with buttons that display nearby food establishments' names with their formatted addresses placed under the name. Users can then select a food establishment that they wish to get a prediction for. This then launches a screen that contains a form that is similar to the screen shown when users select "Search by name". The main difference between the forms is that for nearby food establishments, there is an assumption that the user wants the prediction at the current time. Therefore the only editable fields in this form are the establishment type, as the name, time and day have already been determined. Users can then select the "Get times" button, which invokes a similar Volley-based method that sends a request to the web server. The results screens for both are the exact same.

Another important feature of the application and overall system is providing the ability for users to add their own wait times to be included in later predictions. This involved adding a new section to the application's main screen under a new "Submit Data" section. A new button is displayed stating "Submit Wait Time". Once this is selected, a screen is shown where users can input the food establishment's name, the time and day they waited, and how long they waited for. This new data is sent as a POST request using Volley. This involved adding new functionality to the web server to be able to handle POST requests. Upon receiving a POST request, the server extracts the parameters from the request (using the request.values attribute) and stores them in unique variables. These wait times are then stored in .csv files for use in later predictions. This required the changing of how the prediction model got its data. The new method involved combining the user data with the simulated data created by the queueing model. The server then sends a thank you response to the client application that gets displayed to the user.

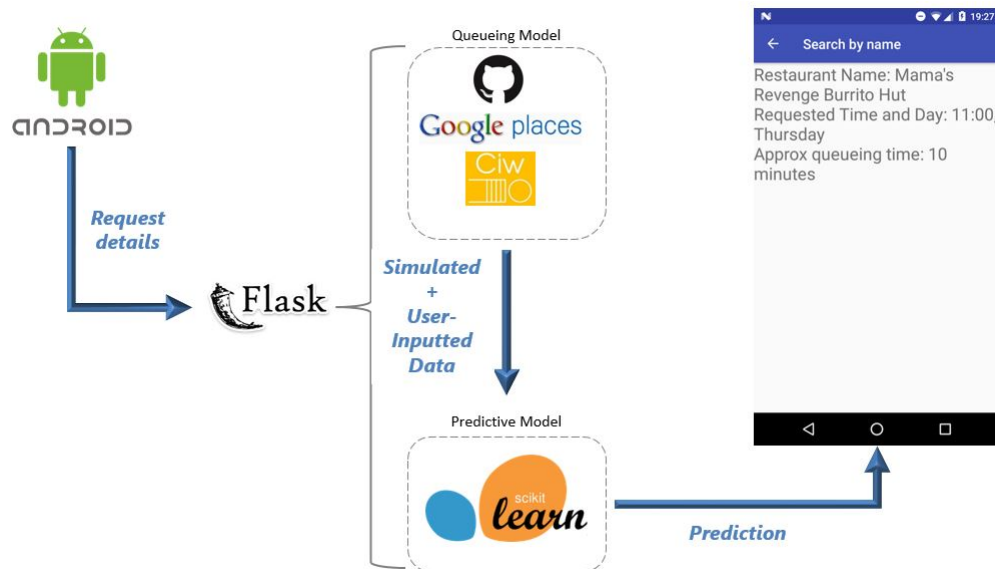


Figure 3.14: A high-level look at the libraries and frameworks used to create the overall system

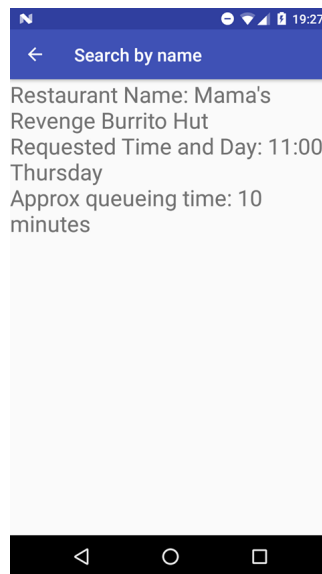


Figure 3.15: The screen displayed to users once a prediction is returned

3.2.4 Testing

The main method of testing for the web server was unit testing. Unit testing is a software testing method that involves testing individual sections of code (in this case individual functions) and determining if they perform as they were designed to. A unit is considered the smallest testable part of any software. It follows a method known as “White Box Testing”, whereby the tester knows the inner workings of the item being tested [19]. The tester chooses the unit’s inputs and determines the appropriate

outputs. The method is named so due to the fact that the software being tested, in the eyes of the tester, is like a transparent or white box, into which the tester can clearly see. The main advantage of White Box Testing is that it is very thorough, meaning that almost all paths can be covered. A disadvantage that critics of the method argue is that the test script maintenance can be difficult and cumbersome if the software changes regularly [19]. The contrast to this method is “Black Box Testing”, whereby the tester does not know the inner workings of the item being tested [5]. The code located in Appendix C is an example of a function used in the web server, as well as the corresponding unit test for that function. The unit test ensures that the correct output is produced when given certain inputs.

Chapter 4

Evaluation

This chapter will take a look at just how accurate the predictions created by our web server are. The predictions will be compared to real-world examples, whereby the establishment in question is visited and the queueing time taken and compared to our prediction. Each establishment type was represented by one selected venue. These venues were visited during the same hour on the same day for a number of weeks (a different hour and day for each type). Predictions were given prior to entering each establishment. The queueing time recording began as soon as the queue was entered. The time was stopped when the food was received. After getting the real-world queueing time, the time was then entered into our system and noted for future calculations. This meant that the following week, the prediction given would take into account the previous weeks' actual queue time.

4.1 Results

This section will talk about exactly how accurate the predictions produced by the web server are. The predictions were produced using the queueing model and predictive model in the web server. We will then take a look at each individual establishment type and measure just how accurate the predictions were, from no user inputs up to four user inputs.

4.1.1 Individual Algorithm Performance

This subsection contains graphs that display each algorithm's individual performance when creating a prediction for each establishment type. The graphs begin with no user

inputs, and increase by one user input until four user inputs are reached.

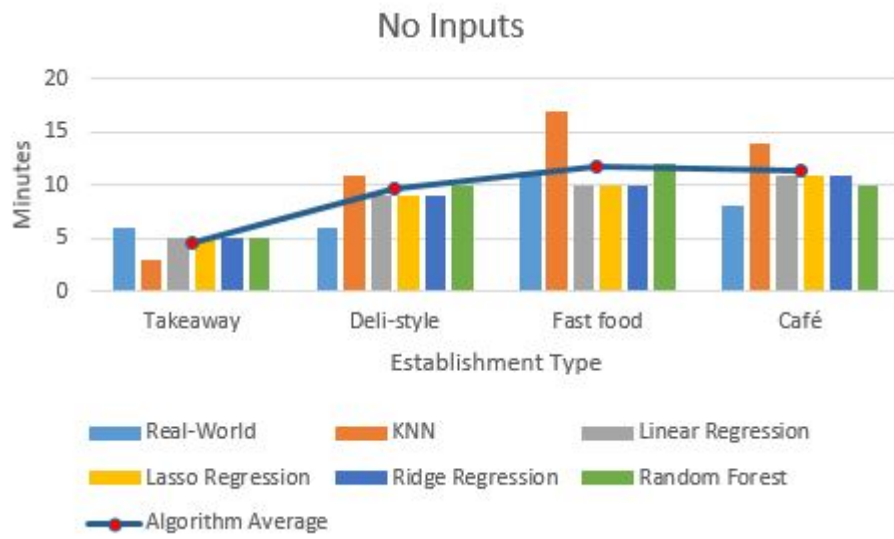


Figure 4.1: Individual algorithm results after 0 inputs

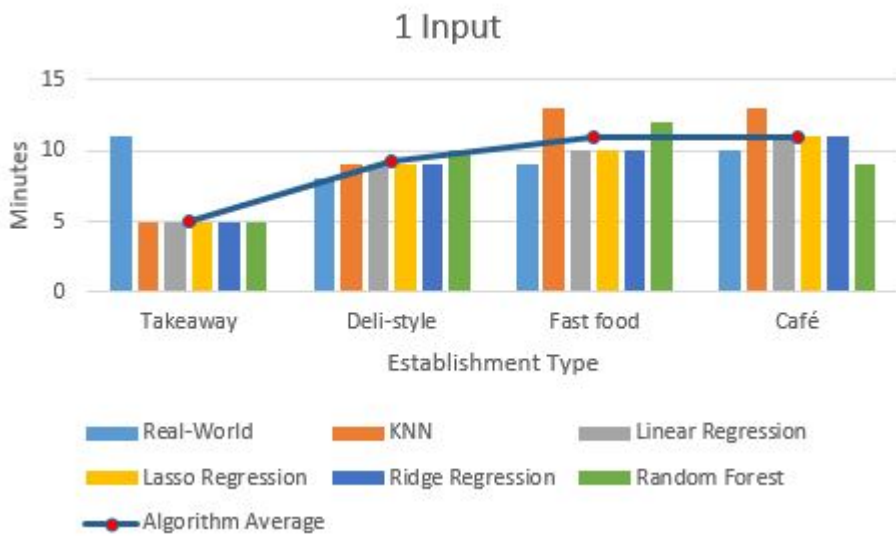


Figure 4.2: Individual algorithm results after 1 input

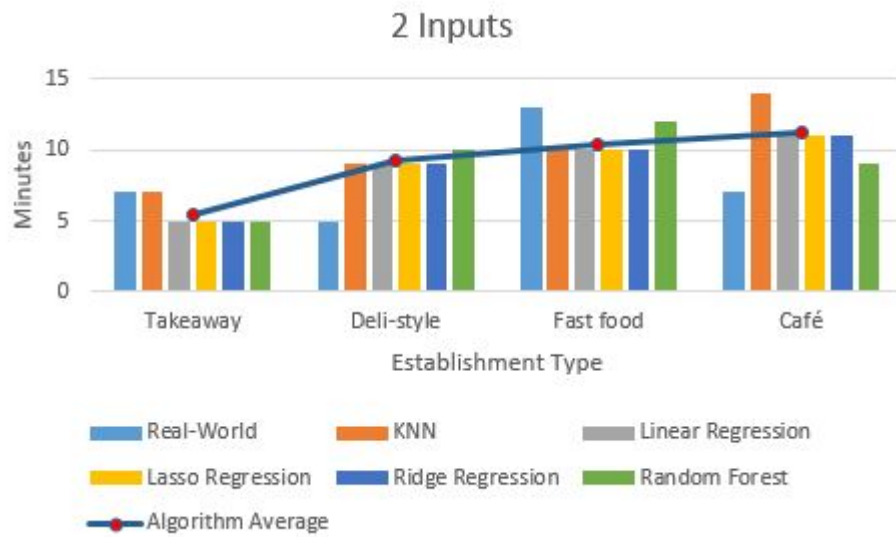


Figure 4.3: Individual algorithm results after 2 inputs

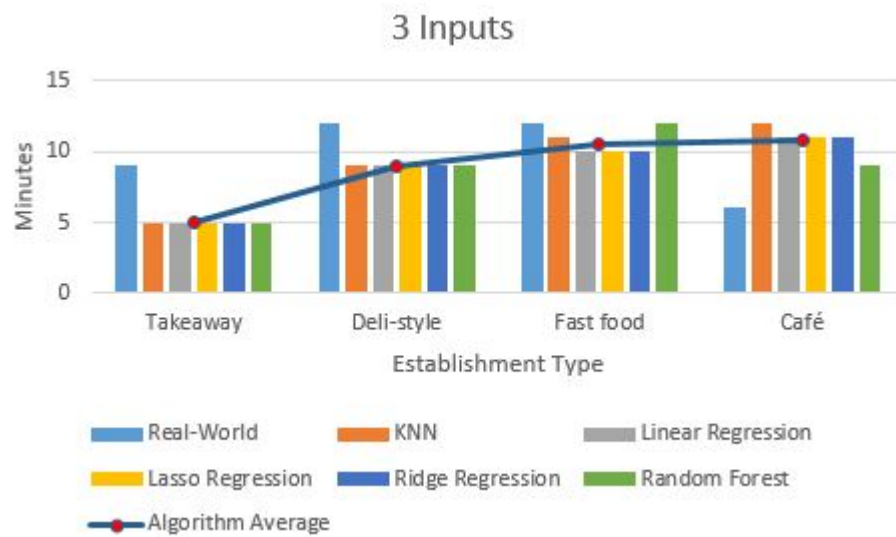


Figure 4.4: Individual algorithm results after 3 inputs

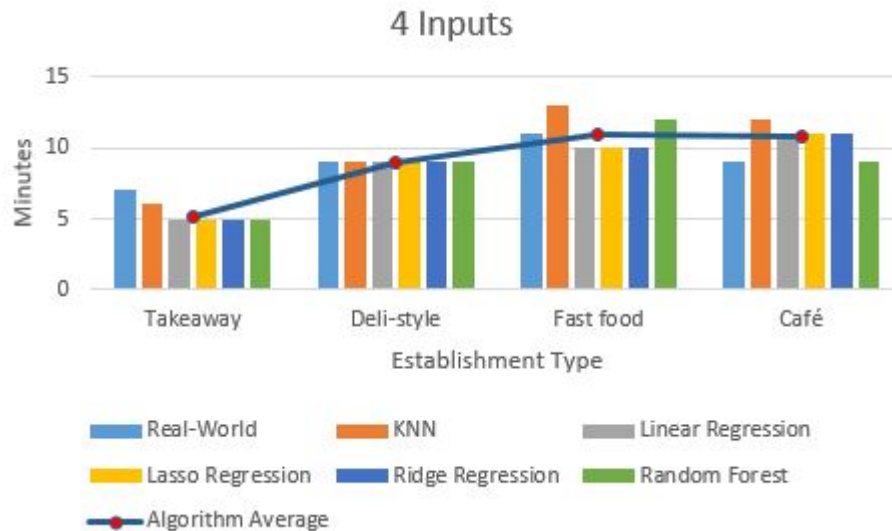


Figure 4.5: Individual algorithm results after 4 inputs

From these graphs, it can be seen that the choice to average out the five algorithms was a good one. As you can see in 4.1 in the “Fast food” section, KNN produces a result of 17 minutes. This is a massive outlier and may have caused even more inaccuracy in the prediction. However, the other algorithms’ predictions bring the average down, resulting in an error of merely 1 minute. The tables containing the data displayed in the previous graphs are located in Appendix D.

4.1.2 Prediction Accuracy

Cafe

The chosen venue for the cafe type was a local branch of an international cafe chain. The venue was chosen due to the fact that it is generally very busy and always has a queue. The chosen time and day for this venue was 3pm on a Sunday. The real-world queueing time was measured from the time the queue was entered, to the time that the coffee was received. The below graph shows the results obtained after no user inputs, up to four user inputs:

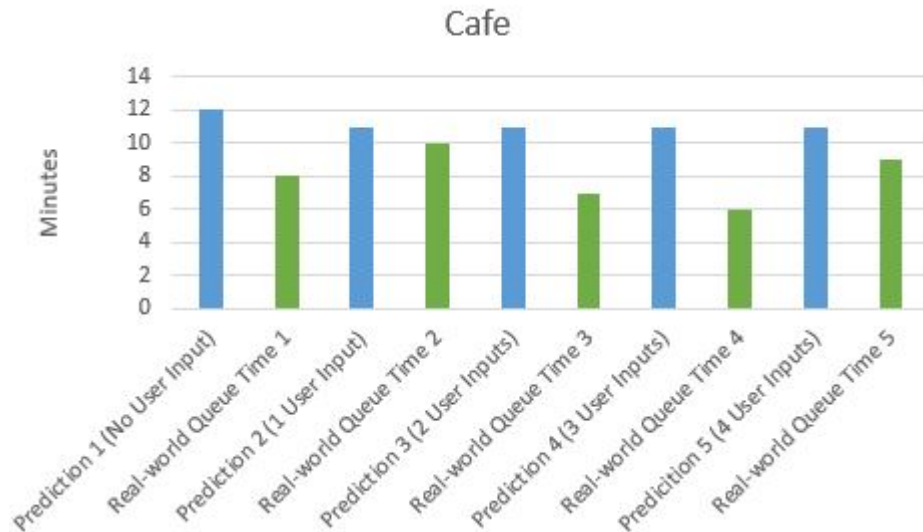


Figure 4.6: Cafe queuing times compared to the predictions made by the system

As you can see from the graph, all of the predictions produced results that turned out to be longer than the actual queueing time. While users may be satisfied with the results, where the actual queue time is shorter than the prediction, in terms of accuracy it is not as desired. The below table shows the errors in minutes:

	Prediction 1 (No User Inputs)	Prediction 2 (1 User Input)	Prediction 3 (2 User Inputs)	Prediction 4 (3 User Inputs)	Prediction 5 (4 User Inputs)
Error (in minutes)	4	1	4	5	2

Average error: 3 minutes

Figure 4.7: The absolute error rate of the cafe establishment type

The average error in minutes was 3.2 minutes, rounded out to 3 minutes. This error is most likely due to the various changes in the real-world queueing times, which are something that cannot be controlled. Adding in more user inputs should, in theory, provide a more accurate prediction.

Fast Food

The fast food venue was a local branch of an international fast food restaurant. Like previous choices, this venue was chosen due to it being rather popular and having a rather large queue (the majority of the time). The real-world queue time was measured from when the queue was first entered, to when the food was received. The chosen time and day for this venue was 4pm on a Saturday. The following graph shows the

results obtained after no user inputs, up to four user inputs:

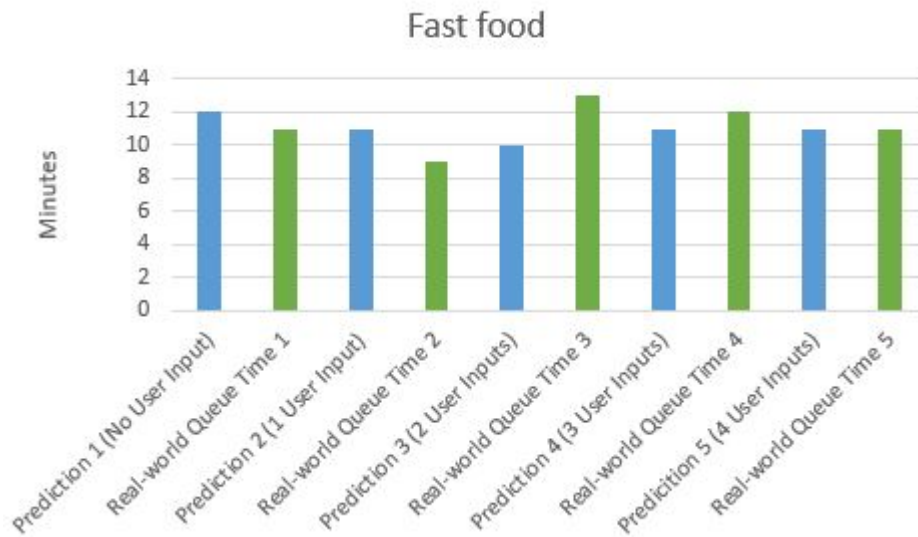


Figure 4.8: Fast food queuing times compared to the predictions made by the system

This establishment type allowed us to see how much user inputs can affect the system's predictions. Following one user input, the system's prediction lowered by one minute. After the second input is entered, the system's prediction is lowered once again. This is due to the system making adjustments to the data being fed into the model in accordance with the user inputs. The following two inputs are then off by 1 and 0 minutes respectively. These results can be seen in the table below:

	Prediction 1 (No User Inputs)	Prediction 2 (1 User Input)	Prediction 3 (2 User Inputs)	Prediction 4 (3 User Inputs)	Prediction 5 (4 User Inputs)
Error (in minutes)	1	2	3	1	0

Average error: 1 minutes

Figure 4.9: The absolute error rate of the fast food establishment type

Here, the average error is 1.4 minutes, which can be rounded out to 1 minute. This average error is quite low and as such is quite impressive. As stated previously, these results showcase how much of an effect user inputs can have on incorrect predictions.

Takeaway

The takeaway venue was a local takeaway. The venue is a busy establishment with

both delivery and over-the-counter options. As stated previously, the system was designed to deal with walk-ins, so therefore the over-the-counter choice was used. The real-world queue time was measured from when the queue was first entered, to when the food was received. The chosen time and day for this venue was 9pm on a Tuesday. The following graph shows the results obtained after no user inputs, up to four user inputs:

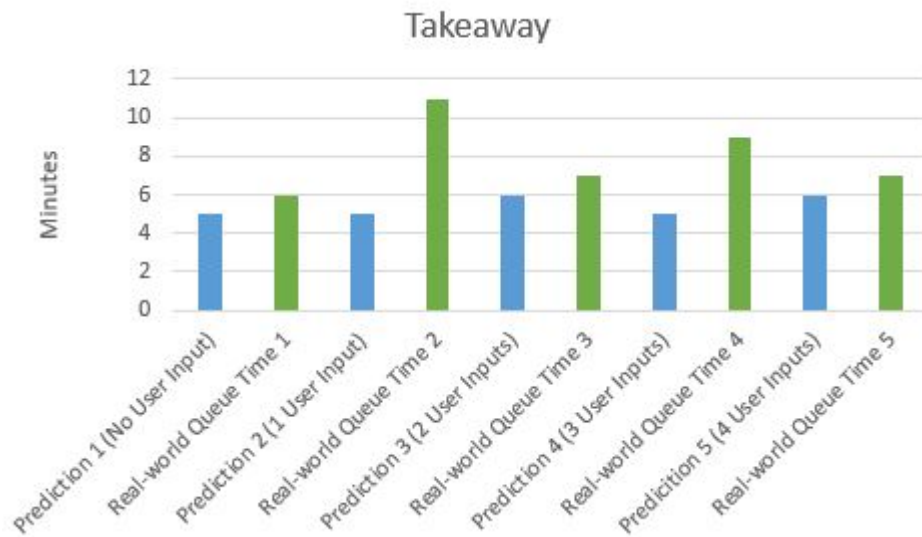


Figure 4.10: Takeaway queuing times compared to the predictions made by the system

	Prediction 1 (No User Inputs)	Prediction 2 (1 User Input)	Prediction 3 (2 User Inputs)	Prediction 4 (3 User Inputs)	Prediction 5 (4 User Inputs)
Error (in minutes)	1	6	1	4	1

Average error: 3 minutes

Figure 4.11: The absolute error rate of the takeaway establishment type

The average error here is 2.6 minutes, which can be rounded out to 3 minutes. The main cause of errors here could be down to the fact that the venue is also taking delivery orders as well as over-the-counter orders. This means that when a person enters a queue of, for example, 3 people, they may also be behind several orders that have been placed for delivery. This possible delay could be taken into consideration and may be something that could be investigated in future work.

Deli

The deli venue chosen for this establishment type was a burrito restaurant popular

with students in the vicinity of Trinity College. This venue’s notorious queues were one of the inspirations for this entire dissertation, as choosing the hour of visiting had to be done carefully in order to avoid longer queues. The venue is consistently busy and fits the mould of a deli-style establishment needed for this system’s testing. The real-world queue time was measured from when the queue was first entered, to when the food was received. The chosen time and day for this venue was 11am on a Thursday. The following graph shows the results obtained after no user inputs, up to four user inputs:

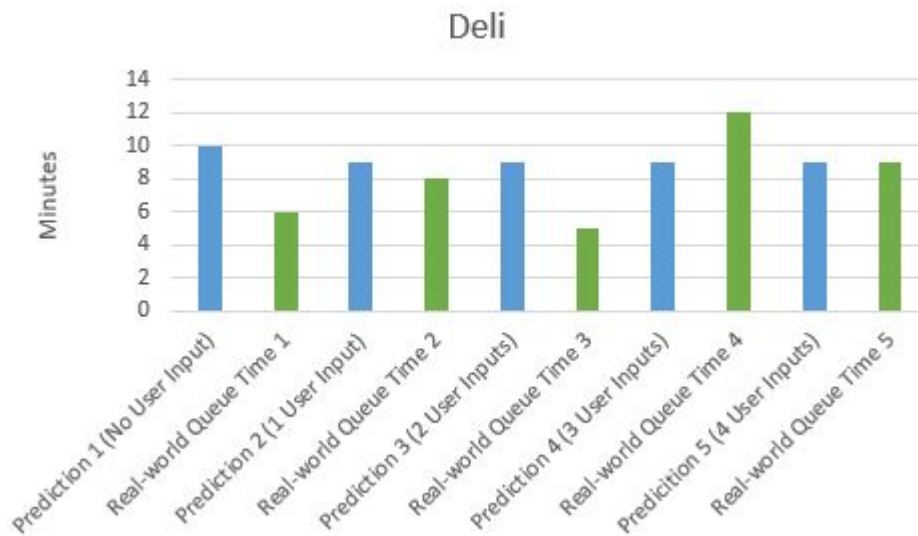


Figure 4.12: Deli queueing times compared to the predictions made by the system

User inputs play a small part in shaping the system’s predictions here, too. The prediction is lowered by 1 minute after “Real-world Queue Time 1”, and then remains the same when a user input of similar queue time is entered. The fact that the queue time is a float number rounded out to an integer is the reason that the prediction doesn’t lower after “Real-world Queue Time 3”. The prediction was most likely a value that was greater than or equal to 8.5, thus resulting in a prediction of 9 minutes. The below table displays the errors in minutes:

	Prediction 1 (No User Inputs)	Prediction 2 (1 User Input)	Prediction 3 (2 User Inputs)	Prediction 4 (3 User Inputs)	Prediction 5 (4 User Inputs)
Error (in minutes)	4	1	4	3	0

Average error: 2 minutes

Figure 4.13: The absolute error rate of the deli establishment type

The average error here is only 2 minutes, which is quite accurate. A deli-style establishment has different order sizes, which aren't taken into consideration here. This could be the reasoning behind the errors.

4.1.3 Summary of Results

After all calculations, the average absolute error was 2.25 minutes, which can be rounded to 2 minutes. This result shows that the system is quite accurate, even given a small number of user inputs. When there are no inputs, the average error is 2.2 minutes, which can be rounded to 2 minutes. Once the system gets introduced to user inputs, the average error increases to 2.375 minutes, which can be rounded to 2 minutes. While this would make it seem that the system works best without user input, keeping the data fresh and up-to-date via user input is important in preventing obsolete models (as mentioned in Chapter 2 - Research). As more inputs are given, the system is likely to become even more accurate.

This absolute error rate is similar to that of “Restaurant Wait Time Estimation Report” by Segovia, Patel, & Lonneman [30]. The advantage that the system created for this dissertation has over the system created for “Restaurant Wait Time Estimation Report” is the fact that the data is constantly updated, meaning the models will not become obsolete. The report mentioned is also only designed for one particular restaurant, whereas the system created for this dissertation uses a more general approach applied to various food establishment types.

The most accurate establishment type was the fast food type, with an average absolute error of just 1 minute, which is quite impressive. The least accurate establishment type was tied between the takeaway and the cafe type. These types produced an average absolute error of 3 minutes, which is not too poor given that the average wait time for these types were both 8 minutes. The full ranking from most to least accurate can be seen in the following table:

Rank	Establishment Type	Average Absolute Error (Minutes)	Comment
1	Fast Food	1	Most accurate
2	Deli	2	-
3	Takeaway	3	May have been influenced by delivery orders not being considered in Google's Popular Times feature
3	Cafe	3	Least accurate

Figure 4.14: Ranking the establishment types from most to least accurate

4.2 Criticism

As a reminder, the research question was:

“How accurate can a system using a combination of queueing theory and predictive analytics be in creating predictions of queueing times in dynamic, real-time queues such as walk-in food takeaway establishments?”

As mentioned in the previous section, the average absolute error was 2.25 minutes. This result means that the system was quite accurate, given that the average real-world wait time was 8.8 minutes. Despite this, the question cannot be answered quite yet. The amount of testing for the system is not at a high enough level to allow us to give a satisfactory answer. If we were to base it on this dissertation alone with the small amount of test data used, we would most definitely say that the research question has been answered with a resounding “very accurate”. Unfortunately, due to time and financial constraints, the establishments could not be visited as many times as needed to prove this.

Rather than explicitly answering the question, this dissertation has indeed given substance to the theory that it may be possible to create accurate predictions for food establishment queueing times. Just how accurate these predictions can be has also been hinted at in the research carried out in this report, but as mentioned, significantly more testing needs to be carried out before guaranteed accuracy can be claimed.

Where this dissertation could have been improved (as well as what I would change if I were to start again) is mainly focused on getting more test data (real-world queue-

ing times). In designing the system, I assumed that I would be able to get sufficient data to answer the research question. As I stated above, more data would have made answering the research question a lot easier. This data could have been obtained by using volunteers who could have simply collected data on their lunch break.

Another area that could have been improved is in the queueing model. Every time a request is made, a new set of simulated data is created with the queueing model. To save time and increase performance, one could implement a way of storing the simulated data. When a request is made, the simulated data could be checked for “freshness”. For example, if the data is more than a week old the model could be run again and fresh data simulated.

The application is another area that could be improved. Instead of getting a user to select an establishment from a list of nearby establishments, the queueing times could be instantly displayed to the user when “Search nearby” is selected. There could also be a change made that removes the need for users to select the establishment type every single time (e.g. a learning model that associates similar restaurants to one type).

Overall, I find that this dissertation has been a success. Our goals were to create a system that was simple and straightforward to use while producing accurate results. The system is very easy to use, and the predictions created were proven to be accurate, given the test data used.

Chapter 5

Future Work

This chapter will take a look at the possible work that could be carried out as a continuation to this dissertation. The main focus of possible future work would be improving the queueing model for better predictions, further testing (and tweaking if necessary) of the created system, as well as the extension of the idea to other establishments with queues including theme parks, doctor's offices, post offices and more.

5.1 Queueing Model Improvements

One area that could be explored in future work inspired by this dissertation is in regards to the queueing model. The current model relies on Google's Popular Times in order to get the arrival rate, issues arise in areas where two establishments of the same name (i.e. a franchise) are. In this scenario, the queueing times are unable to be predicted as no arrival rate can be determined.

The queueing model could also be improved by creating a method to create dynamic service rates. At present, the system uses service rates that were simulated while the system ran. Future work could be carried out to attempt to obtain service rates in a similar manner to arrival rates. This could lead to improved accuracy due to more accurate service rates. These changes could spur the creation of a completely new queueing model that may be a better fit for establishments with similar queues to the test food establishments used in this dissertation.

As mentioned previously, one improvement would be to implement a way of storing the simulated data. When a request is made, the simulated data could be checked for "freshness". If the data is more than a week old, for example, the model could be run again and fresh data simulated. This could improve overall performance and make for

a smoother overall process.

5.2 Further Testing & Tweaking

As mentioned in the previous chapter while evaluating our system, the amount of test data used in evaluating this system was small. This meant that the research question could not be answered in a satisfactory manner. If someone wished to get a solid answer to the research question, it would be a good idea to attempt to use volunteers to gather queue times. This would allow the system to be fully tested and confirm or deny the ability to predict accurate queue times.

Further tweaking to the system could also be done. As mentioned in the Implementation section of Chapter 3, there were two methods of choosing algorithms to use in creating predictions. One utilised all five available algorithms, the other used the three with the lowest Mean Absolute Error (MAE) values. After testing both methods, the decision was made to go with the usage of all five algorithms. This choice may have produced accurate results given the test data, but further testing with vast amounts of data could be done to see if there is a superior method of choosing algorithms. This could involve testing the use of all five, testing the choice of the three algorithms with the lowest MAE values, or a method based on another metric or parameter entirely (e.g. RMSE).

As mentioned in the Criticism section of the previous chapter, the application could be changed to instantly display nearby establishment wait times when users select “Search Nearby”.

Another area that could be considered would be improving the scalability of the current system, if needed. At present, it is unknown how well the system will perform given high levels of traffic or massive datasets.

5.3 Branching Out

The work carried out in the completion of this dissertation was focused on food establishments with dynamic, real-time queues. The system could be extended to predict queue times in other establishments that have dynamic, real-time queues such as doctor’s offices, post offices, theme parks etc. The idea could also be extended to establishments with multiple queues such as supermarkets. This would mean intro-

ducing different queueing models to the system as well as potentially exploring more algorithms to be used in creating predictions for those establishment types.

Bibliography

- [1] Aws shield - managed ddos protection. <https://aws.amazon.com/shield/> [Accessed 15th April 2018].
- [2] (2018). Amazon ec2. <https://opennebula.org/users/featuredusers/> [Accessed 18th April 2018].
- [3] (2018). Android developers. <https://developer.android.com/> [Accessed 8th May 2018].
- [4] (2018a). Api - flask 1.0.2 documentation. <http://flask.pocoo.org/docs/1.0/api/#flask.Request/> [Accessed 4th May 2018].
- [5] (2018). Black box testing - software testing fundamentals. <http://softwaretestingfundamentals.com/black-box-testing/> [Accessed 9th May 2018].
- [6] (2018). Featured users - opennebula. <https://aws.amazon.com/ec2/> [Accessed 18th April 2018].
- [7] (2018b). Flask website. <http://flask.pocoo.org/> [Accessed 17th April 2018].
- [8] (2018). Getting started with amazon ec2 linux instances. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html [Accessed 20th April 2018].
- [9] (2018a). Google maps. <https://www.google.ie/maps/> [Accessed 20th April 2018].
- [10] (2018). Google places api. <https://developers.google.com/places/> [Accessed 2nd April 2018].
- [11] (2018). m-wrzs/populartimes. <https://github.com/m-wrzs/populartimes> [Accessed 25th March 2018].

- [12] (2018). Nowait website. <http://nowait.com/> [Accessed 17th April 2018].
- [13] (2018). Numpy numpy. <http://www.numpy.org/index.html> [Accessed 5th April 2018].
- [14] (2018). Postman — api development environment. <https://www.getpostman.com/> [Accessed 1st May 2018].
- [15] (2018). Qless website. <https://www.qless.com/> [Accessed 17th April 2018].
- [16] (2018b). slimkrazy/python-google-places: Simple wrapper around the new 'experimental' google places api. <https://github.com/slimkrazy/python-google-places> [Accessed 25th March 2018].
- [17] (2018). Transmitting network data using volley. <https://developer.android.com/training/volley/index.html> [Accessed 17th April 2018].
- [18] (2018). Tutorial iv: Trials, warm-up & cool-down. https://ciw.readthedocs.io/en/latest/Tutorial-I/tutorial_iv.html [Accessed 10th February 2018].
- [19] (2018). White box testing - software testing fundamentals. <http://softwaretestingfundamentals.com/white-box-testing/> [Accessed 9th May 2018].
- [20] Cody, S. (2013). The waiting game: Fast-food queuing theory. *Aetherstore Blog*.
- [21] Hedaux, S. (2017). The psychology of queueing. <https://www.linkedin.com/pulse/psychology-queueing-simon-hedaux> [Accessed: 16th April 2018].
- [22] Ho, T. K. (1995). Random decision forests. *Proceedings of the 3rd International Conference on Document Analysis and Recognition*.
- [23] Kendall, D. (1953). Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded markov chain. *Institute of Mathematical Statistics*.
- [24] Liaw, A. and Wiener, M. (2002). Classification and regression by randomforest.
- [25] NCSS.com. *NCSS Statistical Software, Chapter: Ridge Regression*.

- [26] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.
- [27] Penttinen, A. (1999). Introduction to teletraffic theory.
- [28] Rowlings, J. O., Pantula, S. G., and Dickey, D. A. (1998). *Applied Regression Analysis: A Research Tool, Second Edition*. Springer.
- [29] Sayad, S. K nearest neighbors - regression. http://www.saedsayad.com/k_nearest_neighbors_reg.htm [Accessed: 20th April 2018].
- [30] Segovia, E. V., Patel, A. R., and Lonneman, Z. W. Restaurant wait time estimation. Technical report.
- [31] Siegel, E. (2013). Predictive analytics: Harnessing the power of big data. *Competitive Edge*.
- [32] Steorts, R. C. (2017). Comparison of linear regression with k-nearest neighbors.
- [33] The Ciw library developers (2018). Ciw: v1.1.5.
- [34] Tibshirani, R. (1996). Regression shrinkage and selection via the lasso: a retrospective. *Journal of the Royal Statistical Society*.
- [35] Vincent, J. (2017). 99.6 percent of new smartphones run android or ios. <https://www.theverge.com/2017/2/16/14634656/android-ios-market-share-blackberry-2016> [Accessed 22th April 2018].
- [36] Wesner, J. (2016). Mae and rmse - which metric is better? *Medium.com*. <https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d> [Accessed 10th May 2018].
- [37] Wikipedia. k-nearest neighbors algorithm. https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm [Accessed: 18th April 2018].
- [38] Wikipedia. Linear regression. https://en.wikipedia.org/wiki/Linear_regression [Accessed 18th April 2018].

-
- [39] Wikipedia. M/d/1 queue. https://en.wikipedia.org/wiki/M/D/1_queue [Accessed: 16th February 2018].
- [40] Wikipedia. Random forest. https://en.wikipedia.org/wiki/Random_forest [Accessed: 15th April 2018].
- [41] Wikipedia. Regression analysis. https://en.wikipedia.org/wiki/Regression_analysis [Accessed: 18th April 2018].
- [42] Zukerman, M. (2013). Introduction to queueing theory and stochastic teletraffic models.

Appendix A

Sample Google Places API Response

```
{
  "html_attributions": [],
  "next_page_token": "CVC06gEAAIU_eTvxvMUDgshA3n2jec09UJazkVIInsRIItfFoylg31agLvdlwNlFUV44IH-Y-yIy720xjKDea85tAsH-zzBR-
  l8ypfXU4UAF0Cce88vM2DyJy80q1xj_By92iHozURYG0j4xkx4xbggccLalLhqC3zBUkhzjH3_KaqdgrR8CgTS0EpgAh1DtlFw6GpIdm7JoyHu17j6FIaXshR_R78uEiyoYk7bZ8L8p2EipALxS4aFDf48r1dDt0e2vanA00x0007g6cvmV50rXNA-
  Em5PLt37h7f9g48HwAlOlyYh1cySDXK0Ww4xvPufsc01xPo185E0FLW5Tohedumc97EwRoq6sRmG6tbeIeHfE1gVwZ9GPlmooX-RSE56jS8M9-khZxFE7V2QvRUDvAaB0CEzVePi6GLDon32AehPri-Lr2fhyCuE9UrxkDVA--q1uXUcWR7D70h1T0l0qV9vK0A0q52awHmRXJ_UlUfpaC44P-55n88fAXX96yT0
  YXNlVlD0d04770g3jUP0KdsvvCchS58rR7Gd7TqIVV02iHawbK3duIDEFnoYku3iNpVhRZ-Hc79kxkF0V0Kvew4Gj3R83Pwv7hLxanOVU45GLU76rvngBwE5EKto9S5agvD69Hemh0K4eKwFLvg4d8HEmeEqJd49L76cK0Cq",
  "results": [
    {
      "formatted_address": "17 Main St, Blanchardstown, Dublin 15, Ireland",
      "geometry": {
        "location": {
          "lat": 53.3869196,
          "lng": -6.377462400000001
        },
        "viewport": {
          "northeast": {
            "lat": 53.38828452989272,
            "lng": -6.376089770107279
          },
          "southwest": {
            "lat": 53.38558487010727,
            "lng": -6.378789429892723
          }
        }
      },
      "icon": "https://maps.gstatic.com/mapfiles/place_api/icons/restaurant-71.png",
      "id": "5137b44613445426ec5246e8c7b35c7091ebbb9",
      "name": "Rudys Restaurant",
      "opening_hours": {
        "open_now": false,
        "weekday_text": []
      },
      "photos": [
        {
          "height": 3024,
          "html_attributions": [
            {
              "url": "https://maps.google.com/maps/contrib/114311554281961623687/photos/\u003eArIk_Portnoy\u003cA\u003e"
            }
          ],
          "photo_reference": "CmRAAAAAM8UP6KFTThgc41UyQnFV3PAHFENGn4RSfyYEVK1g1j1bYH5WGIa0qm8Zc68JcsRL5uXU-p5sA2_gmp13iN5e0c5Pab460h8BvRVY15zt2xmw49v9Lcxp0R000IEHBRZix2yRGlT062t07HwMhGR7-r0KTHfGtveTGY0zrFeA1j4LA",
          "width": 4032
        }
      ],
      "place_id": "ChI32x_EvFANZ0gRqW1I-XoJ0ic",
      "rating": 4.2,
      "reference": "CmRBAAAAsZYVNP31U4Phs27Hjn1d74-Ds/n_VQ1Xono8PTX52H16qNqjje020m1c54x235aCJ1cue4q2epayufd7-BuFaog167Dj1h1LX0KPEKjJbFne98EsTAU5m3khEHAFFpB91XoDprceaQv6Ee2_gH54hP0iKg4xxiZ5350heSaqr7Vd1g",
      "types": [ "restaurant", "food", "point_of_interest", "establishment" ]
    },
    {
      "formatted_address": "2 Liffey Valley Shopping Centre, Fonthill Rd, Dublin 22, Ireland",
      "geometry": {
        "location": {
          "lat": 53.352709,
          "lng": -6.394558
        },
        "viewport": {
          "northeast": {
            "lat": 53.35405882989272,
            "lng": -6.393208170107278
          },
          "southwest": {
            "lat": 53.35135917010728,
            "lng": -6.395907829892722
          }
        }
      }
    }
  ]
}
```

Figure A.1: Sample response from Google Places API

Appendix B

MAE Algorithm Choosing Method - Results Graphs

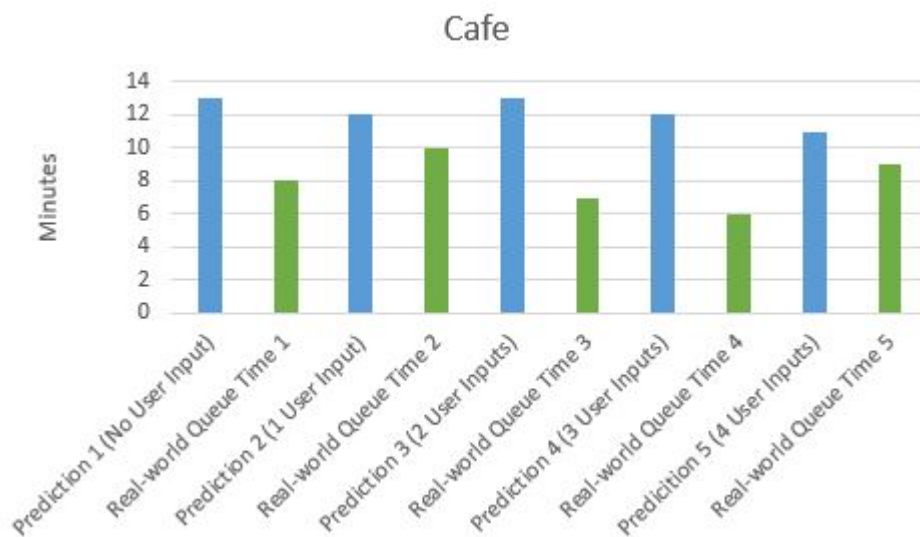


Figure B.1: Cafe results after choosing predictive algorithms using the MAE method mentioned in Chapter 3

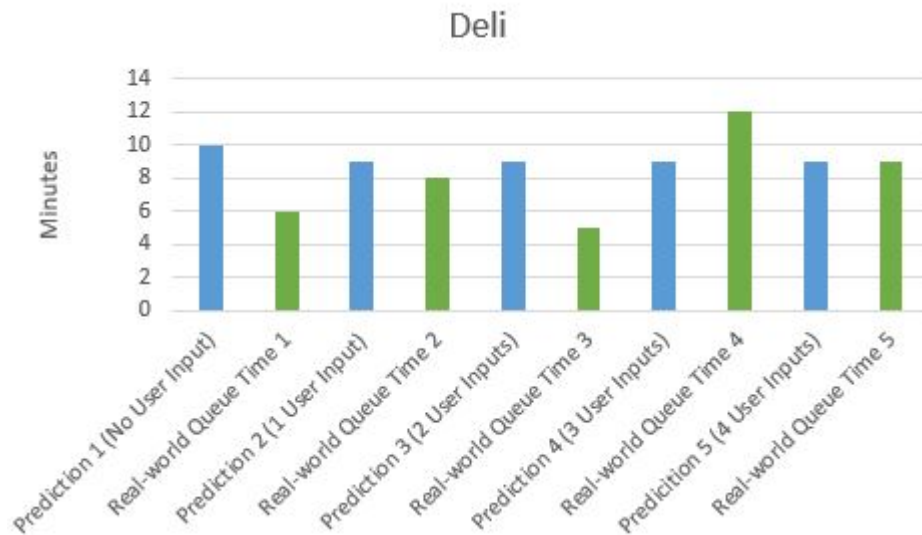


Figure B.2: Deli results after choosing predictive algorithms using the MAE method mentioned in Chapter 3

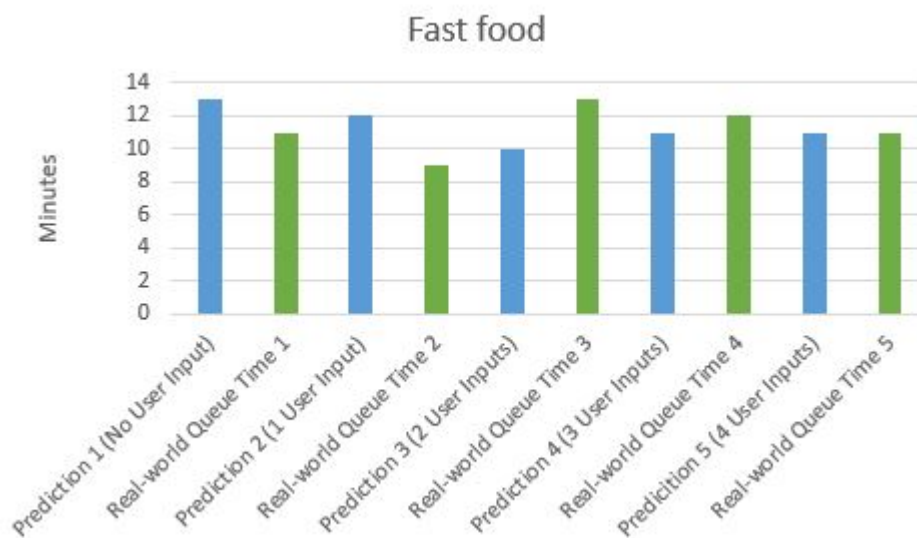


Figure B.3: Fast food results after choosing predictive algorithms using the MAE method mentioned in Chapter 3

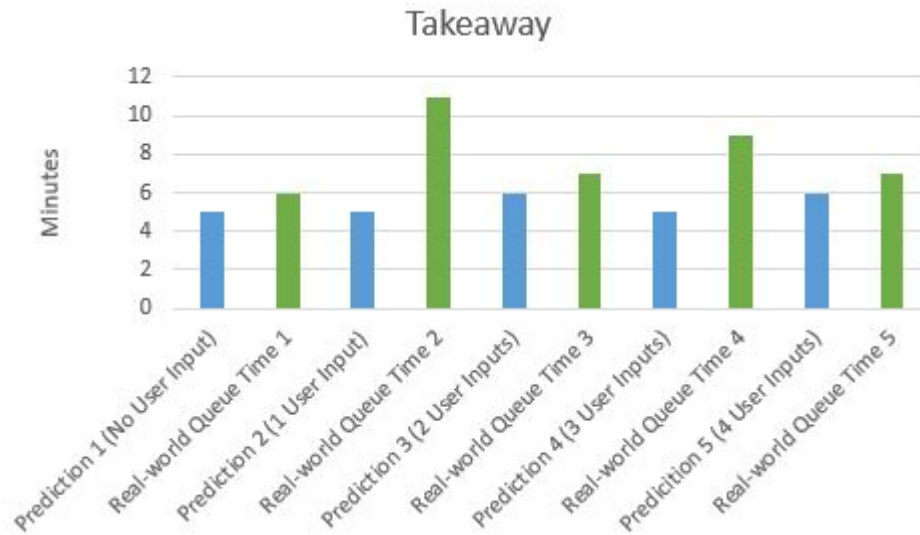


Figure B.4: Takeaway results after choosing predictive algorithms using the MAE method mentioned in Chapter 3

Appendix C

Sample Function and Unit Test

```
# Sample function to be tested
def day_to_number(day):
    for i in range(len(test_vals.days)):
        if day == test_vals.days[i]:
            return i

# Sample unit test for above function
def test_day_to_number(self):
    self.assertEqual(application.day_to_number('Monday'), 0)
    self.assertNotEqual(application.day_to_number('Monday'), 10)
    self.assertEqual(application.day_to_number('Sunday'), 6)
    self.assertNotEqual(application.day_to_number('Sunday'), 205)
```


Appendix D

Individual Algorithm Comparison Data Tables

	Real-World	KNN	Linear Regression	Lasso Regression	Ridge Regression	Random Forest	Algorithm Average
Takeaway	6	3	5	5	5	5	5
Deli-style	6	11	9	9	9	10	10
Fast food	11	17	10	10	10	12	12
Café	8	14	11	11	11	10	11

Values given in minutes

Figure D.1: Individual algorithm predictions after 0 inputs

	Real-World	KNN	Linear Regression	Lasso Regression	Ridge Regression	Random Forest	Algorithm Average
Takeaway	11	5	5	5	5	5	5
Deli-style	8	9	9	9	9	10	9
Fast food	9	13	10	10	10	12	11
Café	10	13	11	11	11	9	11

Values given in minutes

Figure D.2: Individual algorithm predictions after 1 input

	Real-World	KNN	Linear Regression	Lasso Regression	Ridge Regression	Random Forest	Algorithm Average
Takeaway	7	7	5	5	5	5	5
Deli-style	5	9	9	9	9	10	9
Fast food	13	10	10	10	10	12	10
Café	7	14	11	11	11	9	11

Values given in minutes

Figure D.3: Individual algorithm predictions after 2 inputs

	Real-World	KNN	Linear Regression	Lasso Regression	Ridge Regression	Random Forest	Algorithm Average
Takeaway	9	5	5	5	5	5	5
Deli-style	12	9	9	9	9	9	9
Fast food	12	11	10	10	10	12	11
Café	6	12	11	11	11	9	11

Values given in minutes

Figure D.4: Individual algorithm predictions after 3 inputs

	Real-World	KNN	Linear Regression	Lasso Regression	Ridge Regression	Random Forest	Algorithm Average
Takeaway	7	6	5	5	5	5	5
Deli-style	9	9	9	9	9	9	9
Fast food	11	13	10	10	10	12	11
Café	9	12	11	11	11	9	11

Values given in minutes

Figure D.5: Individual algorithm predictions after 4 inputs