VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui

School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

Predicting Public Transport Loadings using a Prediction Model

Michael Behan

Supervisors: Andrew Lensen and Andrew Myers

Submitted in partial fulfilment of the requirements for
Bachelor of Engineering with Honours.

Abstract

Passengers utilising the transport network run by Metlink are unable to know how full a vehicle is. Metlink is unable to provide the occupancy data in real-time due to limitations for when the data gets uploaded. This project developed a system that takes historical data and processes it so it can be used to make a prediction. The system takes the average occupancy based on properties that contribute to the occupancy trend such as COVID level and day. This is important information to know for passengers so they can safely socially distance for the respected COVID level.

Contents

Chapter 1

Introduction

Metlink is responsible for running the public transport network across the Greater Wellington Region. This network incorporates Buses, Ferries and Trains to get their passengers from point A to point B. Passengers can plan and track their journey on the Metlink website, app and display boards. Delays, cancellations and vehicle positions are available in real-time to the passenger to estimate when the bus will arrive at their location.

This project aims to present another factor that passengers can gain extra insight into their journey. There is no way for a passenger to know if they will get onto a vehicle that is empty, a few seats left, standing room only or if they have to wait for the next service due to no room at all. A popular request Metlink has been receiving is that they would like to know if there will be enough space on the vehicle they intend to board, this request has grown in popularity due to the need to know if passengers can safely socially distance due to COVID.

1.1 Problem

Real-time data is not available, instead, the use of historical data provided by Metlink is needed to predict the occupancy of a vehicle. The past six months of transport data entry is available and can be accessed to assist in making an informed prediction. Included in the data is the value of how many passengers got in and out at a particular station at a particular time. This data is only uploaded when a train arrives at the Wellington Station and connects to an internet connection which can sometimes be the following morning.

When making an informed prediction, the program needs to factor in public holidays, weather and past data trends. Another factor to consider is this data does include occupancy values during each COVID level; therefore, a group of data may be treated as outlying data as it is a lot different when comparing various months.

1.1.1 Data

There are two file types that are provided by Metlink that are relevant to this project: APC (Automatic Passenger Counting) and Consist files. The APC is responsible for storing detailed information regarding when, where and the number of people going in and out of each door. There is a data entry for each door, the trains that run on the Metlink network all have eight doors, therefore there are eight data entries each holding the number of people going in and out of that door. The consists file contains information that states the journey and what units the train consists of. A train unit represents two carriages with a total number of 8 doors, each door has a data row in the APC files that has a property "Train_number" that can be used to relate to the Consists file that has the property "Unit Id". These files are stored for every train journey represented in an excel file, at the end of each day the excel files are compiled into a zip folder where they are kept for 6 months on the Metlink server.

The format of the data is explained further in Chapter 2.

1.1.2 Occupancy values

In this data format, the occupancy value is not related to any other entry other than the eight doors at that time and station. The occupancy value is calculated using the total number of people going in minus the total number of people going out at that station. The problem is that there is no data value of how many people are currently on the train. Because the system needs to predict how full at train will be at a station, this value is required.

The total occupancy value can be calculated by iterating through each train stop along the journey and update the value accordingly. The process to obtain the current occupancy at each station is:

- Assign each APC unit (Combine the eight doors together) to a journey by matching with Consists files.
- Iterate through the units journey stops and calculate the total occupancy by adding total in and subtracting total out.
- At the end of the journey, total in should be ignored and should expect the result of subtracting total in to be zero.
- At the beginning of the next journey, total in is assigned to be the starting total occupancy.

Throughout this project, this process is where the most time and effort has gone into. The lack of any documentation for the data made it very difficult to understand how and when the data is inserted and what the data represents. Multiple conversations with GWRC (Greater Wellington Regional Council) and multiple testing approaches were developed to form a greater understanding.

As seen in Figure 1.1.2, displaying all data entries for a journey going from Johnsonville to Wellington, there are inaccurate values present. If the previous occupancy is wrong it will have a cumulative effect where the error is carried over for all the following occupancy values, that is why we see a lot of negative values. Therefore, it is important to make sure that the occupancy values are calculated correctly, otherwise, if the data being used to predict is wrong, it will result in inaccurate predictions.

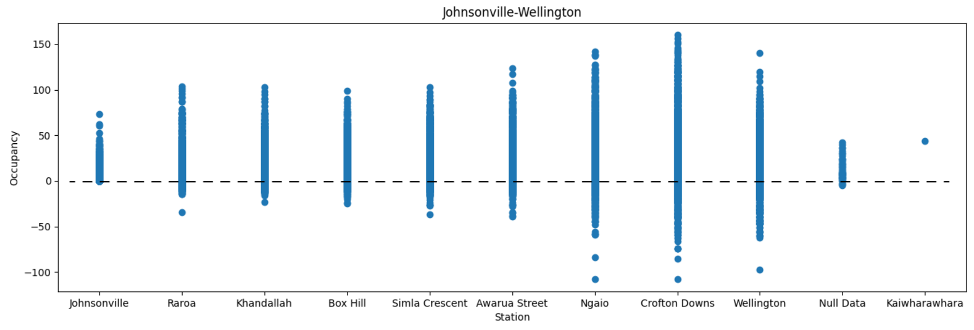


Figure 1.1: Single journey occupancy example

1.2 Requirements

The following requirements have been analysed and evaluated based on the project brief and the available data:

R1 The system needs to produce a prediction file regarding the occupancy values for every Train journey based on the network's timetable.

R2 The system must preprocess the data to have a data object that represents a train with its total occupancy when it arrives/leaves the station.

R3 The system needs to output the predictions to be parsed and displayed on both the Metlink website and the display boards at each station.

R4 The system must have an easy architecture for additional features to be included that could affect the occupancy prediction.

R5 The system must automate the process to import the latest data and features used to make new predictions.

1.3 Contributions

Throughout the period of this project the following contributions have been made:

C1 The design and implementation of a system to calculate and generate a list of predictions regarding the occupancy values of a trains journey.

C2 The design and implementation of a process to parse and format data into data objects representing a train vehicle and it's total occupancy for each stop along it's journeys.

C3 A quantitative and qualitative evaluation of the performance of the predictions calculated.

C4 A proof of concept of a system that demonstrates the operation of this design.

Chapter 2

Background and Related Work

After researching and reading about different projects in this area, there is not a project that has the same data access and the same aim as this project. It appears that most train networks provide this data in real-time based on the data calculated from the sensors, so there is no need for a prediction model [1]. The train vehicles that run throughout Metlink's network are unable to provide real-time occupancy data due to the only way to upload data to the servers is to connect to the access point at the Wellington station. Every train would have to have a wireless access point to upload the occupancy data when the passenger counting sensors have new data, this project is created to avoid the cost of installing multiple access points.

In some countries, passenger counting sensors are not found in the transport network, so projects have been developed to create crowd-sourced data [4]. The crowd-sourced data is used in the prediction model to attempt to deliver the passengers information about the vehicle. But if there isn't enough data then the predictions are not reliable [4]. Prediction models have many real-world applications. Some of these applications present similar aspects to the predictions planned to be made in this project.

2.1 Occupancy Prediction Models for Transport

There are existing applications using machine learning algorithms to predict occupancy values for transport methods. An interesting project named the iRail project in Belgium aims to predict Train occupancy values based on crowd-sourced data [4]. There is no official occupancy data available for the Belgium train network, so a crowd-sourced dataset is used to train the model. This is an exciting idea as it results in a possible real-time solution to the problem if the passengers classify the occupancy while they are travelling. For crowd-sourced data to prove to be successful, it needs a lot of backing and adoption from users. This has not happened. The predictions being made are not possible due to the lack of data.

Many projects talk about the different techniques used to solve prediction problems. A study presents a new method based on Deep Learning with Recurrent Neural Networks to predict the occupancy of a car park [2]. This project focuses on the use of Deep Learning using the Birmingham Car Park Occupancy dataset that is updated every 30 minutes to benchmark the performance of different RNN algorithms and configurations. The idea of improving predictions for real situations by updating the model periodically with updated data so the model can adjust.

As pointed out in the iRail project, a limited amount of data is good at predicting trains with a low occupancy which is outside peak times, so it is easy to predict [4]. The dataset used to train and the reliability is essential.

2.2 ML Approaches for Time Series

For this project, the historical data being processed is time-series data. Therefore, I concluded that it was worth exploring articles about making predictions with different approaches for time series data [9].

There are multiple different techniques and methods to approach forecasting time series data, such as Symbolic Regression, Linear Regression and Feed-Forward Neural Networks. These can all be valid approaches that make accurate predictions and some can even be combined to improve results. After researching, I concluded there is no right algorithm that will fit the project. I will have to experiment and evaluate different techniques. For this project, we need a method that can be expanded to factor in multiple parameters such as weather and also be able to improve after new data has been imported.

In an article [9], I found that experiments with time-series data in depth. They focused on combining ML techniques with framing data by windows. This is to achieve an accurate prediction. It reshapes the information to frame a given point and will check the previous values within the window.

2.3 Metlink Train data format

The data format for the two file types that Metlink create is presented by the two tables: Table 2.3 and Table 2.3. The APC data format (Table 2.3) contains information that represents the data of the measured sensor for each door, the door number is defined by the "door_number" property where the value ranges from zero to seven, specifying each door within the train unit. A new entry is created for every door when they are closed, the entry created contains information from the period of when the door opens until it is shut. Each door measures the number of people in and out and are assigned to the respected property for the defined door number, all other property key share the same values between the doors as they are part of the same unit. Total_in and total_out values represents the value of the combined doors, the occupancy value represents the total_in minus the total_out.

The consists data format (Table 2.3) contains the information that represents what units the train consists of. Each row defines a unit and the units journey, the journey id and final destination properties are used to identify the trip and where the unit ends its journey. The date properties: Dep Date, Actual Dep Date, Arr Date and Actual Arr Date are defined to match with the APC file, for a Unit id that equals a Train_number has a start_date_time between the consists Dep date and Arr date.

Key	**Description**	**Value example**
id	Identifier for unit data entry	1051874873
train_number	Unit id number	4374
entry_number	Data entry count	1
start_date_time	Date unit arrives	202109212327
end_date_time	Date unit departs	202109212332
dwell	Time train dwells at station	300
type	Type of stop	Station
name1	Station name	Raroa
name2	Station secondary name	
door_release_codes		1
total_in	People in across all doors	11
total_out	People out across all doors	3
occupancy	total_in minus total_out	9
door_number	Door number (0-7)	0
people_in	People in for door	2
people_out	People out for door	0
door_statuses		7

Table 2.1: Train APC Data format

Key	**Description**	**Value example**
Train Id	Service number	6364
Train Date	Date	20210821
Unit Id	Front half of Unit	FP4374
Unit Selcall	“1” + Front half of Unit Num	14374
Trail Unit Id	Back half of Unit	FT4374
Trail Unit Selcall	“2” + Back half of Unit num	24374
Seq	Unit sequence num in Train	0
Manual Linked Vehicle	Driving car	
Journey Id	Identifier for Journey	379210
Final Dest	Going to location	WIKNE
Dep Date	Time expected to depart	202109212314
Actual Dep Date	Time departed from station	202109212314
Arr Date	Time expected to arrive	202109220014
Actual Arr Date	Time arrived at terminal	202109220017
Status		CMP
No Vehicles		0

Table 2.2: Train Consists Data format

Chapter 3

Design

This section discusses the design of the deliverable system that will automate the data input and produce an output consisting of train occupancy prediction values.

3.1 Database

The data is found as a zip folder for each day containing multiple excel files, some consisting of more than 900,000 rows. Each row represents a train door and how many people went in and out of it. The total number of data entries after removing duplicates is more than eight million for a six month period, therefore it is best to process and import every data row into a database. Without a database, the system will need to make multiple different queries; due to a large amount of data, the system can't unzip the files and just store them in RAM, by adding them to a database, the data can be easily accessed, managed and updated.

3.1.1 Database model

The data format is unique so an existing model could not be reused to fit this project, therefore a custom model had to be designed. I have declared and designed four models for the system that is responsible for storing the data to allow for easy management and accessibility.

The first model (Figure 3.1.1(A)), which is responsible for storing all the APC data. Every property column found in the excel file format is present in the model with two additional properties: zip_file_name and date_created. These two are added to avoid reimporting the same data, the import script should check if the zip file has been imported previously, if it has it should check if the date of the zip folder has been modified since it has been previously imported. The next model (Figure 3.1.1(B)) is similar but has a few modifications in comparison to the excel file format. In addition to zip_file_name and date_created properties being appended to the model, the is also unit_ids and train_numbers. A train consists file can contain multiple rows, with each row storing each unit that is part of the same train with the same journey. The array contains each unit identifier within the train.

The remaining two models (Figure 3.1.1(C) & (D)) defined are designed to relate the two data format models together, the "Station" model (Figure 3.1.1(C)) is designed to relate an APC unit to a journey and assign the total_occupancy based on the previous occupancy value within the journey. Then the "JourneyTrain" model (Figure 3.1.1(D)) is created to combine Station objects that are units part of the same train along the journey. The JourneyTrain model has the properties of prev_occupancy and occupancy, where occupancy is the total occupancy of all the units that consist in the train and prev_occupancy is the total occupancy

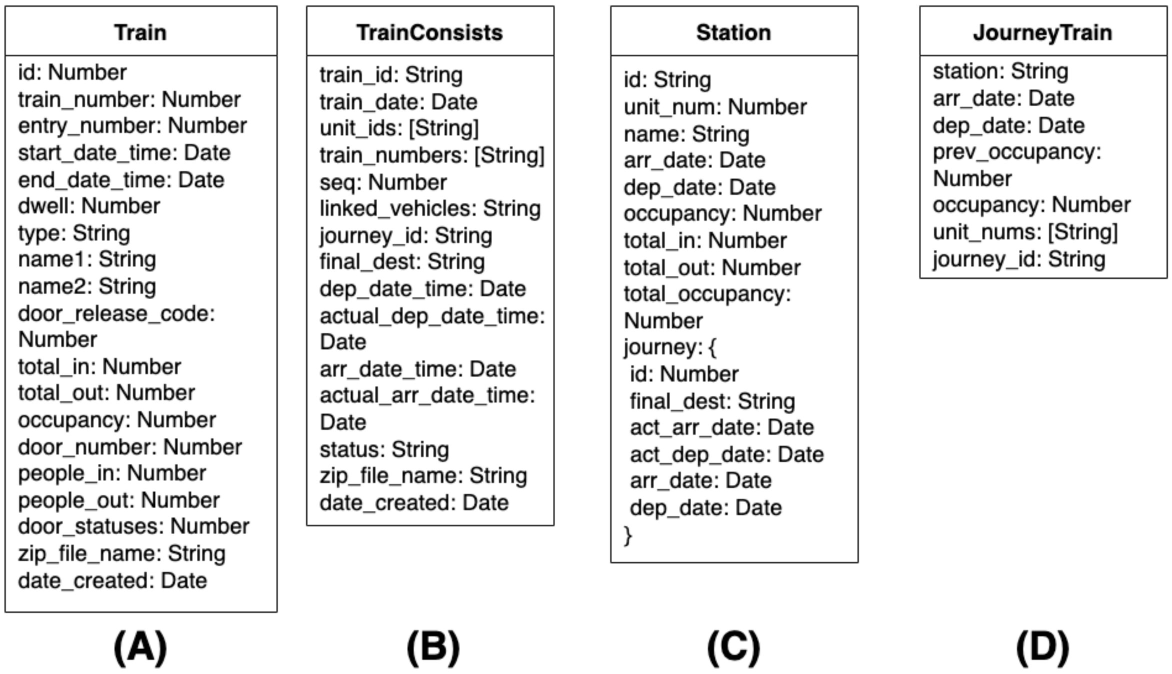


Figure 3.1: Datebase models diagram

of the train at the previous station.

It was important to provide and store as many data features that may be relevant for calculating and displaying a prediction. When designing this database model the main idea kept in mind is what properties would someone looking at the prediction want to see. A key idea is that this system should be a white box and therefore needs to display all the data that is used to make a prediction to backtrace exactly how the prediction was made.

3.2 Program Structure

The requirement of **R4** states that the program needs to have the ability to add extra features for future improvement therefore the structure of a program needs to be designed in a way that the system can be easily understood and extended. Adding features like weather and sports events to help the prediction is an aim for the future of this system. The program is designed to follow the functional programming paradigm. Functional programming is achieved when the program consists entirely of functions, a function receives input and delivers an output and no global variable will change [6]. The main benefit of the design of a function having no effect other than to compute its results is its modularity [6]. It makes it very easy to develop a small module and insert it anywhere within the structure, a module can be reused in multiple places and can be separated out and tested independently [6].

For this project, there are many stages and complex calculations that can make debugging a difficult process. Because the system is modularly designed, each function can be separated out and tested independently, this makes debugging much easier. By separating the program into parts and having them all come together to form the final result allows the addition of new modules and provides the ability to test every individual part before it is attached [6].

This design makes use of small functions/modules to be reused in multiple aspects of the program. The figure below presents each file of the program and how each function works with surrounding functions, there are multiple "helper" files that contain small functions that are reused to compute results needed for multiple modules. For example, the importHelper.js file contains two functions: importData() and zipImport(), these are used for both importing APC files and Consists files. The design of this is to make the import stage generic, that is why two different data formats can use the same function. Therefore it should be easy to implement another data format into the system using the same function.

3.3 Pre-processing

This section discusses the process flow of the pre-processing stage within the system. To calculate a prediction that is based on the historical data, the data needs to be pre-processed to ensure that the values are correct and relevant. The diagram below presents the pre-processing process flow, this stage has been designed to package the data into a new format that is used to predict future occupancy values (Figure 3.3).

The design can be described as a pipeline made up of stages, the first stage is to import the excel file and insert them in the database, the next is to take the documents from the database and assign the calculated current occupancy value of the unit throughout the journey. After the occupancy values are calculated they are parsed as a Station model object and inserted into the database where they are used to create a complete train based on the connected units and formatted as a JourneyTrain model object and inserted into the database. These JourneyTrain objects are created to make it easy to assign flags and features based on the dates e.g. lockdown periods. Once the flags are assigned to the corresponding objects, all the data is exported to a JSON file. The final format has the property keys: station, occupancy, secondsSinceMidnight, weekday, weekend, date and COVID_LVL. These properties can all be derived from the imported data apart from COVID level. This prediction feature can be obtained from the government website that presents the history timeline of the alert level periods [5].

To satisfy the requirement to have the ability to add extra features for future improvement (**R4**), the design has the key idea to be modular and easily extended. Extra features can be added in the future to provide further insight into the occupancy predictions. The pipeline design also provides the flexibility to insert a new stage anywhere along the process.

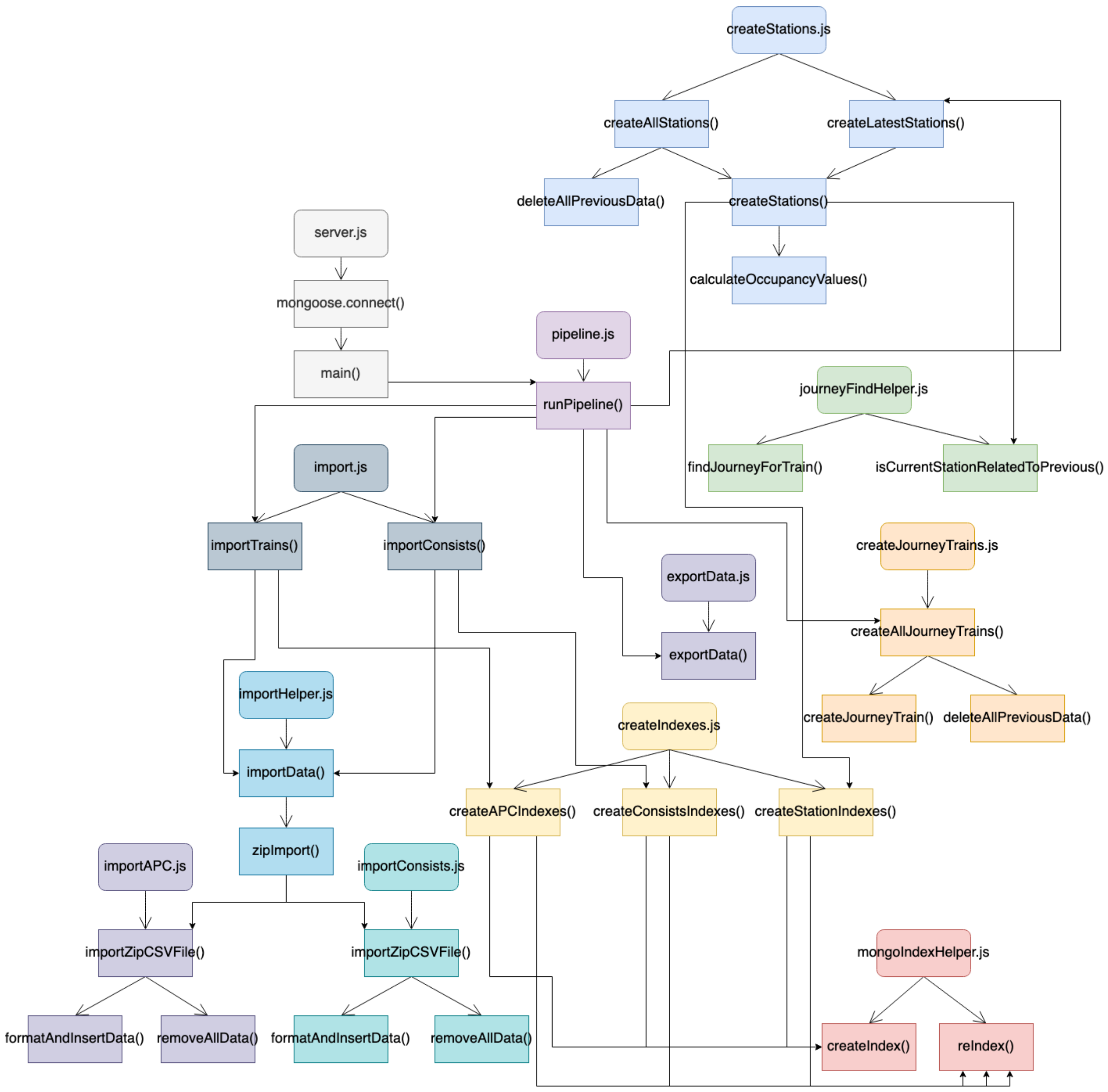


Figure 3.2: Diagram of program structure

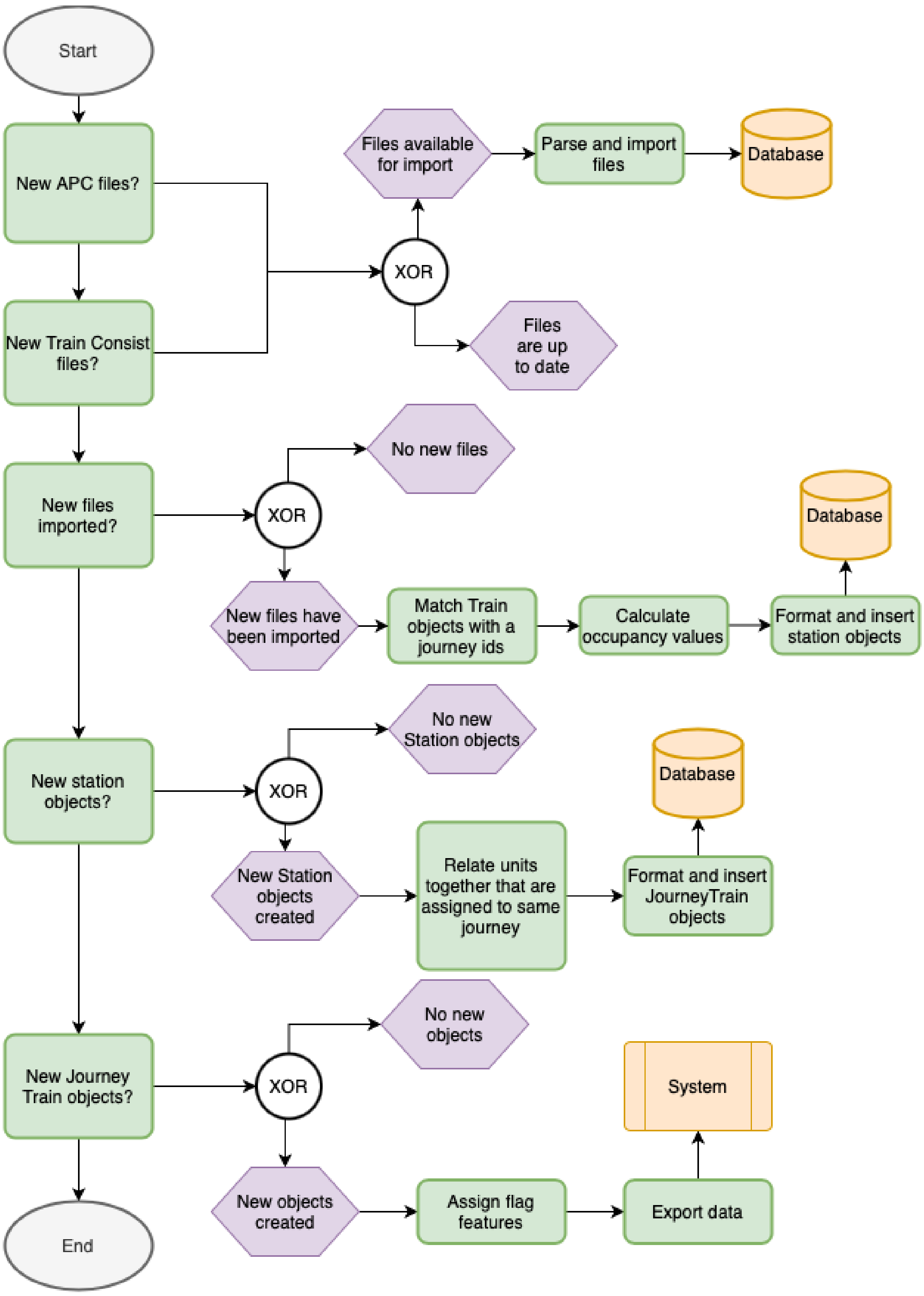

Start
New APC files?
New Train Consist files?
XOR
Files available for import
Parse and import files
Database
Files are up to date
New files imported?
XOR
No new files
New files have been imported
Match Train objects with a journey ids
Calculate occupancy values
Format and insert station objects
Database
New station objects?
XOR
No new Station objects
New Station objects created
Relate units together that are assigned to same journey
Format and insert JourneyTrain objects
Database
New Journey Train objects?
XOR
No new objects
New objects created
Assign flag features
Export data
System
End

Figure 3.3: Flowchart of the pre-processing stage

Chapter 4

Implementation

4.1 Technology

The technology choices have all been made after researching, examining and experimenting, all choices have been made to best suit the system for the specified project. The main technology aspects that needed a decision were which database framework, programming language and prediction model.

4.1.1 Database selection

For selecting which database technology is most suitable for this project, it was between two categories: relational and NoSQL databases. An example of each type of database that I considered is MySQL for relational and the NoSQL database, MongoDB.

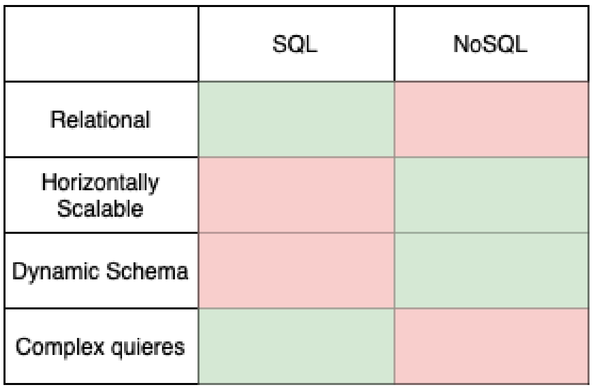
The concept of SQL relational databases is based around the idea of related data tables, while NoSQL is not modelled by relational tables but rather stores data in the database via documents. This is the key difference between the two, the use of a NoSQL database will result in the best performance for big data and a SQL database has the best performance when data needs to relate over different tables [7]. Another feature that NoSQL provides is the ability to have a dynamic schema, therefore attribute names can be dynamically defined at runtime for each document [8].

For this projects use case, MongoDB has shown to be a great solution due to the fast set-up and performance shown with big data. The aim of this project is to improve prediction accuracy by validating the model with new data. The database's size will keep increasing over time and a NoSQL database can scale and store a large volume of data [7]. Although there does require some relating of data between tables, which MySQL would outperform NoSQL. I believe for the long term this NoSQL database will scale the best. The implementation of an SQL database will increase the speed of the processing of related tables, for this project, reliability is more important than speed.

4.1.2 Programming language

The bulk of this project consists of data processing, some great candidates for selecting a programming language are Java, R, Python and JavaScript. The requirements that have been defined to be the best fit for this system are to first be great for data processing, work well with MongoDB, have a rich library ecosystem and be straightforward to manage and manipulate JSON documents.

To use MongoDB, most languages are supported, a datastore should be independent of what programming language is chosen. However, MongoDB stores data as JSON docu-



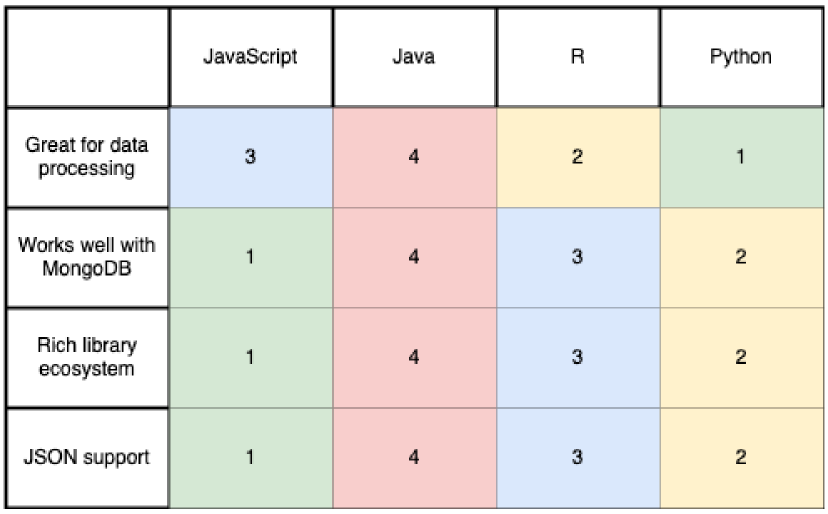
	SQL	NoSQL
Relational	🟩	🟥
Horizontally Scalable	🟥	🟩
Dynamic Schema	🟥	🟩
Complex quieres	🟩	🟥

Figure 4.1: Database technologies comparisons

ments, therefore the ability and performance of serialising and deserialising JSON need to be considered. JSON was born out of JavaScript, therefore it seems like the obvious choice to manage and manipulate JSON documents [10]. JavaScript objects, values and array syntax is similar to the JSON format, where it is easy to convert JSON data into JavaScript objects [10].

Python and R are both the tool of choice for many data scientists, both have very active communities that create tools crafted for data analysis and processing [3]. R has been created and designed to the exact purpose of being used for data science and statistics, therefore this language isn't suitable for this project as the language needs to support multiple purposes [3].

Below is a table where the language candidates have been ranked for each requirement that is needed to support the project (Figure 4.1.2). The ranking decisions are made by making conclusions based on reading papers and previous experience with the language.



	JavaScript	Java	R	Python
Great for data processing	3	4	2	1
Works well with MongoDB	1	4	3	2
Rich library ecosystem	1	4	3	2
JSON support	1	4	3	2

Figure 4.2: Programming languages comparison diagram

Choosing both JavaScript and Python as the programming languages allowed the flexibility and power of Python with the access of great libraries such as Matplotlib and Pandas while also having JavaScript to process and format the data from MongoDB.

4.1.3 Prediction model

The implemented method that the system uses to calculate predictions is written in Python, the process of this method is:

1. Taking arguments that specify station, COVID level, day and time.
2. Calculate if the day is a weekend or not.
3. Calculate the average occupancy of all the data that satisfy the argument values.
4. The average is calculated by first generating a query using the Python library Pandas, which creates a group of data that equals the arguments. The query has the parameters of: station, COVID_LVL, weekday, weekend, secondsSinceMidnight.
5. For the argument of time, this query is defined as ((time - 500) < secondsSinceMidnight < (time + 500)). The times in the dataset are when the train arrives at the station, therefore the vehicle isn't always on the exact same time so a buffer of plus and minus 500 seconds is added on each side so the data entry is still obtained.
6. Once a Pandas dataframe (Group of data) is created, consisting of all data that satisfy the argument values, the occupancy values for each entry is summed to a total value.
7. The summed occupancy total value is then divided by the total number in the dataframe. This result is returned and assigned to the variable representing the average.
8. The next step in the process is to get the average occupancy for just the station and COVID level argument values.
9. These two averages are added together with the weights by multiplying the first average value by 0.8 and multiplying the station, COVID average by 0.2.
10. The result of the added weighted averages represents the prediction.

This implementation can be improved to make better, more accurate predictions. The averaging function is designed to take multiple property values that can be swapped in and out to calculate the value that results in the best prediction. This implementation also can weigh different averages, the current weights defined need more tweaking based on tests.

The prediction implementation method that has been defined demonstrates the core of the system, the results that the system can produce are used to be the baseline for future work to improve on.

4.2 Validation

This section of the report outlines the implementation of validating the data that is imported and created. The purpose of data validation is to ensure the data is clean and reliable, this project deals with over eight million entries of data, therefore it's impossible to guarantee that the data that has been imported is correct just by looking at it. Throughout the development period of this project, this validation stage has been crucial for debugging. The validation stages that have been implemented are for importing APC files and validating that the correct journey id has been assigned to the Train object. These two aspects of the system are where the most complexity is, that is why time was invested in developing a validation system. The complexity makes it very difficult to debug and therefore it's difficult to really know if the data is correct without this validation system.

4.2.1 Train door sensors

When the calculated occupancy values were encountered to be negative, a hypothesis that was explored was the possibility that the sensors on the train doors were faulting or installed incorrectly. Multiple validation heuristics were developed to check for any unexpected behaviour of the data measured by the sensors located at each door throughout the train that counts the number of people going in and out of the vehicle.

Heuristic 1

This heuristic aims to see if there are one or more sensors located on the doors of the train carriage that have been installed incorrectly. If the sensor is installed incorrectly, the value for people_out is actually the value for people_in therefore when the calculation subtracts people_out, a negative value occurs.

1. Find a journey that results in a negative occupancy
2. Swap the first door data values, so people_in is the value of people_out
3. Recalculate the journey with the door sensor flipped to see if this change results in a non-negative occupancy value
4. Repeat this process for every combination of door sensor swap

At the end of execution, the script displays every combination with the occupancy at the end of the journey. Those combinations that result in a positive value are tested with the same combination of flipped door sensors for the same unit on a different journey. The test aims to see if this combination works for all journeys that this unit has gone on, if the occupancy values are always positive with this combination the conclusion can be made that the set of door sensors that have been switched have been installed incorrectly.

This conclusion was never made as no combination ever resulted in a positive value for all journeys for any unit.

Heuristic 2

The next heuristic developed to test the door sensors involves getting a train unit at peak time in the morning at the Wellington Station and display each door value. At peak time there is an expectation that there are more people going out of the train compared to in. If this is not the case and there is a door with a high number of people going in, next check another day for the same unit and see if the same door has more people in than out at the Wellington station. If this trend repeats for multiple days the conclusion can be made that the sensor on the door is measuring the wrong way.

Heuristic 3

The last heuristic that was done consisted of printing out each doors occupancy value (people_in minus people_out) for every stop along the unit's journey. The printed format makes it easy to see any invalid trends that occur throughout the trains journey. The main trend that this heuristic looks for is a repeating door's occupancy value of zero. This would indicate that the sensor isn't measuring at all and ignores every passenger walking in that door. This pattern was observed multiple times but not constant for every journey that the unit travelled. Therefore, this heuristic could not make a conclusion that the door sensors are faulty.

By executing these heuristics didn't allow the conclusion of detecting that the sensors are faulty. Therefore, they did prove that the sensor data is correct and the problem of the occupancy values happens within the systems program. It was very important that the data could be proven to be accurate as the process of calculating the occupancy values is very complex and has many aspects that can result in negative values, it made it easy that the data that is invalid can be ruled out.

The problem of calculating occupancy values is explored further in Chapter 5.

4.2.2 APC Data importing

The process of validating the APC data is the excel file directory path is passed as a parameter to the script that executes to check that each entry has been added to the database and all the property values match. A great example to explain how effective this stage can be is there was a problem occurring when calculating a prediction for a specified time. For a prediction request at a peak time, it is expected to reflect an occupancy value at peak, but it did not. There are multiple factors that can cause this to happen within the implemented system. Only after backtracking in the code, the validation system started development used first to validate the APC data, where it throws an error if the date property value is matched incorrectly. The library that is used to parse the date strings added a timezone by default, therefore all date times were 12 hours off. If the validation stage was implemented to execute straight after it has been imported, the time spent debugging would've significantly decreased.

4.2.3 Journey identifier matching

A lot of time was spent debugging the creation of Station objects, where units are assigned a journey id and the occupancy is calculated. A problem arrived when relating units with the same journey id into a JourneyTrain object where the total occupancy of the whole train is summed. It is expected that all the units found in the consists file should have the same journey id and combine to create the JourneyTrain object. The validation system developed checks this and it showed that this was not the case, there were missing units that hadn't been assigned a journey id or assigned an incorrect id. This debugging process of tweaking parameters of how a journey is assigned and then running the validation script allowed a fast iteration process to know how the changes affected the result.

Chapter 5

Evaluation

The evaluation of the prediction system for predicting occupancy values for the Metlink Train network was done using performance as a metric. This section discusses methods to measure performance and what good performance means in the context of this project.

5.1 Methods

Different methods have been used to measure how well the model performs, this performance is measured by comparing the calculated prediction result with the expected value fetched from the dataset.

Method 1

This first method is executed to visually see how the prediction system performs for a single journey. The evaluation method utilises functions developed in the system designed specifically for evaluation purposes.

The method executes by first taking a user input that defines the day, next the program will run and calculate a prediction for every station within each journey that occurred on the selected date. Data is removed for all entries greater than and equal to the selected date, the remaining data is the training set used to calculate a prediction. The function will iterate through each journey calculating each prediction for each station with each journey, at the end of each iteration, a line graph will be presented. The graph shows visually how the prediction compares to the actual value retrieved from the data. On the x-axis, each station name is labelled and the occupancy values are increasing up the y-axis, there is a line for the prediction and the actual values.

This method essentially simulates predicting the occupancy values for tomorrow. It provides an easy understanding of how the prediction model performs. The indication for the best performance is when the two lines are overlapping.

Method 2

This method results in an actual numeric performance measure, the idea is to split the dataset into a training and a test set. Then we use the training set to calculate a prediction for every entry in the test set and validate if the prediction is correct using the actual value.

The first step for this method is to split the data, the size of the test set is defined as 10% of the entire dataset, the data is first shuffled before the split using a random state value. Each entry in the test set is parsed into the predict method where each property is used to get the average based of the dataset of the training set. After each prediction is calculated it is matched with a classifier, the classification is defined as 0 if the occupancy is less than 25 and 1 if less than 50, the classifier continues to increment by 25 until 150. The classified prediction is matched with the classified actual value, if there is a difference of 1 or less, this is considered as a correct prediction and adds one to the correct predictions variable. The accuracy of the model is calculated by taking the correct predictions divided by the total. This will produce a percentage value of how accurate the system is.

5.2 Results

The results using method 1 are displayed below for three different journeys randomly selected.

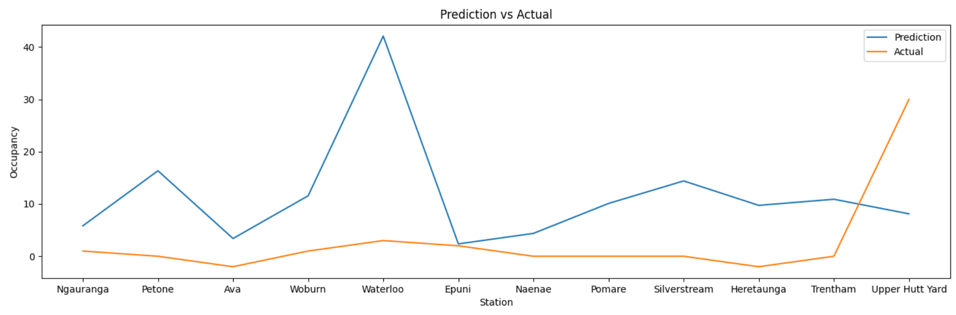


Figure 5.1: Prediction vs Actual for a journey from Wellington to Upper Hutt

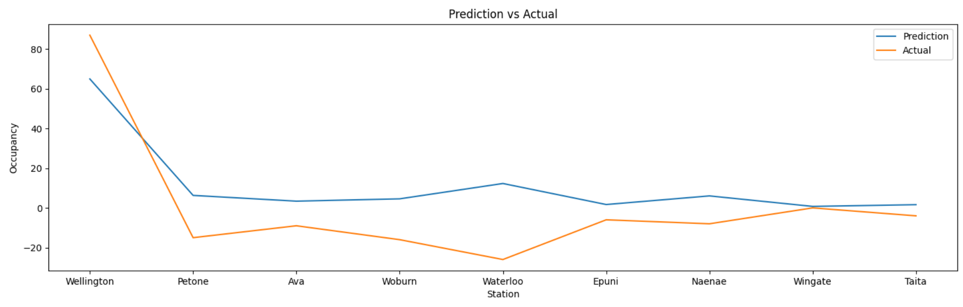


Figure 5.2: Prediction vs Actual for a journey from Wellington to Taita

The results using method 2 is displayed below in a line graph showing the accuracy along the y-axis and across the x-axis is each random state value starting at 0 and increments until 10.

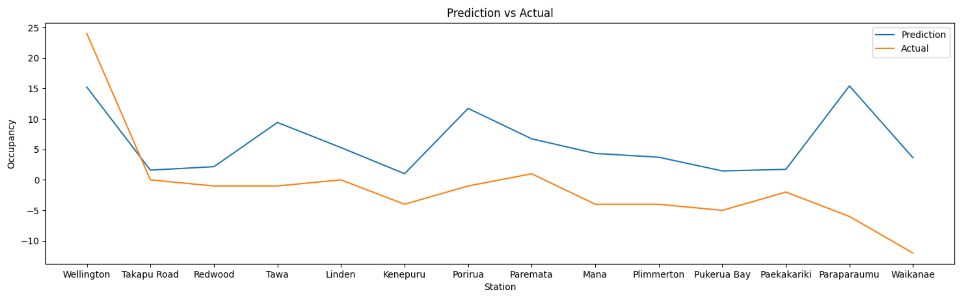


Figure 5.3: Prediction vs Actual for a journey from Wellington to Waikanae

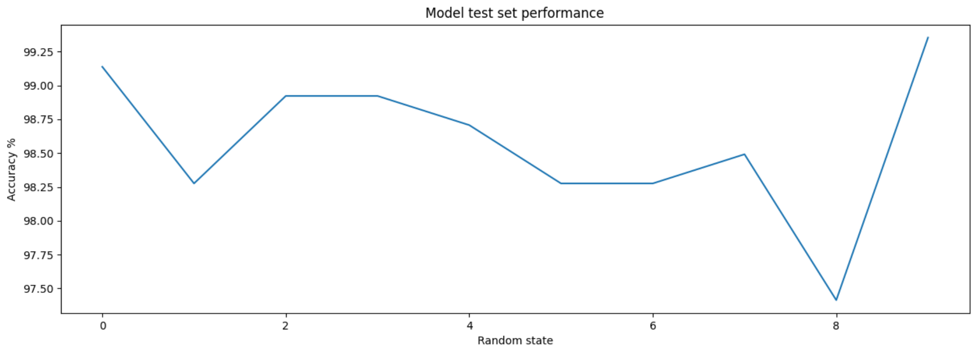


Figure 5.4: Accuracy measurements for shuffled test sets

5.3 Discussion

The performance evaluation of the system has presented a great insight into how the predictions compare to the actual data. Based on the three graphs made by method 1, it has shown that the predictions closely resemble the actual occupancy values. There are stations that the difference between prediction and the actual value is further away. However, this is only relative and the value seen in Figure 5.2 is only overestimating the value by 28. The margin of error defined in the method is the classifier value can be plus or minus 1, and a classification is done in an increment of 25. Therefore, this is an incorrect prediction based on the requirements of the method but it is only overestimating by 3 people. To fix the overestimation, the system can be improved in multiple ways, but a key improvement could be to add a new feature to the model that can be used with the calculation.

Method 2, provided a better measurement to present the accuracy of the system, in the graph (Figure 5.3) the accuracy is presented for a range of different test sets. It shows that the system is able to make a prediction that has an accuracy higher than 97% using 90% of the shuffled dataset. Without shuffling the dataset, the evaluation could be invalid as it could be based on an exact configuration, where for this project the dataset is changing so would not be valid for this use case.

A key factor that can affect the validity of the evaluation methods outlined previously is they are based around the idea that the data and the data processing is of quality. Because the methods involve matching up with the processed data to validate that the prediction is correct, this evaluation is not valid if the data is incorrect to begin with. To fix this, an evaluation method would be included to physically count the number of passengers entering and exiting the train and to match that value with the prediction instead. This also has the assumption that the physical count is of quality and is accurate.

The figures created using method 1 shows that the actual occupancy values provided by the dataset (Orange line) are close to zero throughout the journey. In some cases the occupancy is less than zero, this is due to values not being included within the pre-processing stage. In this stage there are rules that try and mitigate this error, however, this needs improving. I think that the occupancy calculation process needs to be rewritten and validation scripts need to be developed that execute throughout the process to ensure that the data is correctly calculated.

When taking an average of all data for each COVID level it shows a clear relative difference between each level, however, the average occupancy of all stations for level 1 is only a little higher than five (Figure 5.3). The value is expected to be higher, there is a possibility this problem is caused by the mitigation technique to remove negative values by setting them to zero. Therefore the average calculation includes multiple zero values to lower the average.

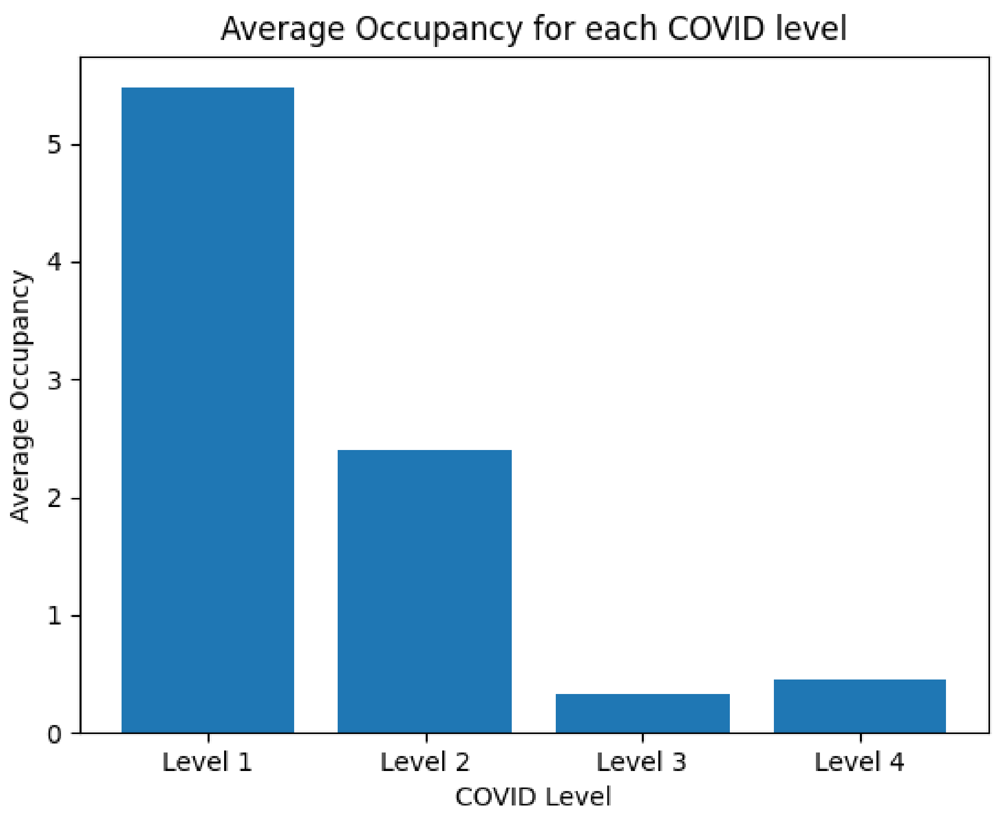


Figure 5.5: COVID levels occupancy average comparison

The prediction performance is measured in Figure 5.3 over a period of a week, this graph is created using method 2 and adjusting it to calculate a prediction for every station stop for every day. The performance ranges from 90% to 100%, this is within the area that is expected to make reliable predictions. In Figure 5.3, the classification class distribution is presented, as defined in method 2 the prediction is matched with a classifier. A classifier value of 0 is assigned to an occupancy value less than 25, Figure 5.3 displays that the system has an unbalanced class distribution, the classifier value of 0 is the result of the majority of calculated predictions. Figure 5.3 has been zoomed in to be able to see the other classification values as they are really small in comparison to the classification of 0.

Therefore, the accuracy values presented in the results are unrealistic as the classification of 0 outweighs the other classes. If a classification ML model was implemented to replace the current prediction model, the algorithm will omit the smaller classes and will again increase the accuracy performance unrealistically.

The evaluation methods presented are only valid if the data is accurate to begin with, the results that the method produced gave great insight into the problems within the system. After fixing and improving the system, these methods can be used again to evaluate the improvements made to justify that the changes made have actually helped the predictions.

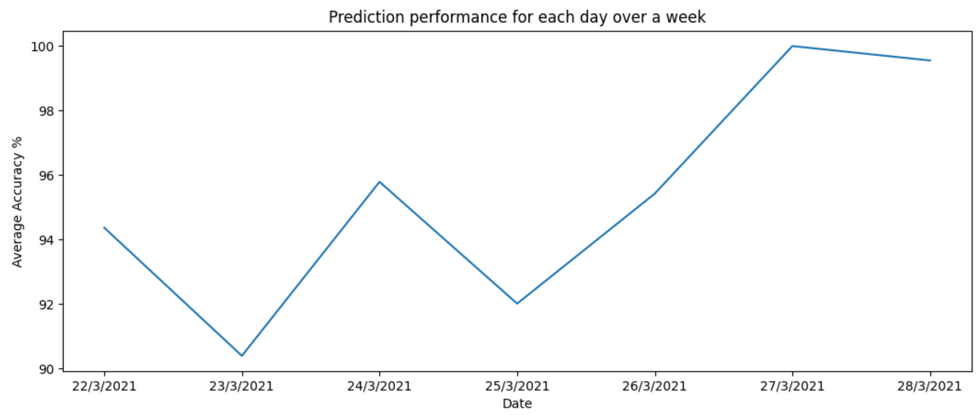


Figure 5.6: Accuracy performance for each day over a week period

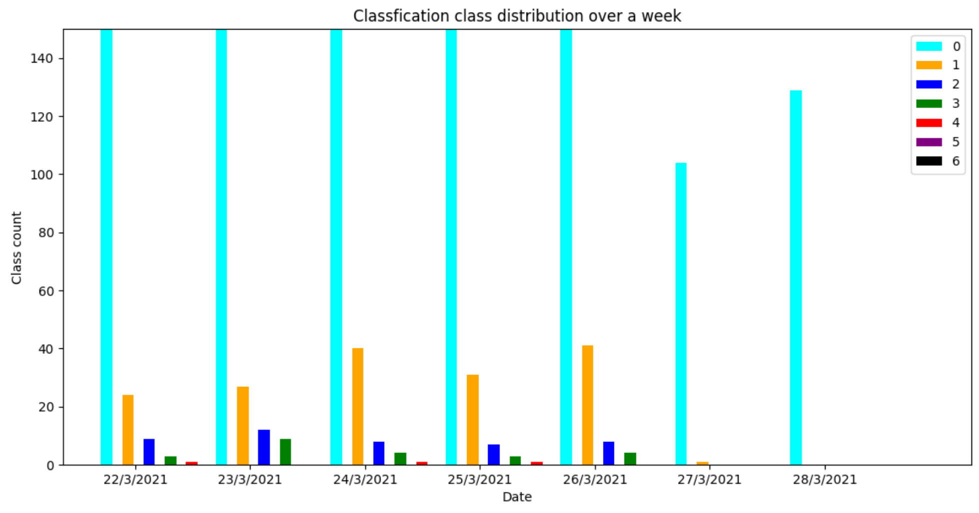


Figure 5.7: Class distribution for each day over a week period

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The aim of this project was to develop a system that can calculate predictions regarding the occupancy count of a train that a passenger intends to get on. The majority of requirements for this project have been accomplished, the remaining requirements will be expanded on in future work. The system does not produce a prediction file containing the occupancy values (R1), this file will be used to meet R3 where the occupancy values will be displayed on both the Metlink website and display boards. These requirements have not been met due to the complexity of the pre-processing stage (R2), this requirement was achieved, the system processes the data and creates data objects that represent a train with its total occupancy. The total occupancy values that are calculated have been evaluated and are believed to be incorrect, the limited time of the project has resulted in this stage being part of future work. The architecture of the system allows for additional features to be used to calculate the prediction (R4), this requirement not only allows for future improvement but helped a lot for debugging purposes.

Requirement R5 has been met through the pipeline architecture design, the system can run automatically to import the latest data and calculate predictions. This requirement is very important for the systems use case as it should run without a person to manually execute it, the system needs to reliably run every day to calculate the predictions with the latest data for future train journeys.

In summary, the contributions made throughout the period of this project are:

- The design and implementation of a system to calculate and generate a list of predictions regarding the occupancy values of a trains journey.
- The design and implementation of a process to parse and format data into data objects representing a train vehicle and it's total occupancy for each stop along it's journeys.
- A quantitative and qualitative evaluation of the performance of the predictions calculated.
- A proof of concept of a system that demonstrates the operation of this design.

6.2 Future work

The occupancy prediction system can be improved in many aspects, the first priority task is to fix the calculation process of occupancy values. Another goal in the future is to automate tests on the system to validate that the predictions created are of value. Once the occupancy values have been evaluated and have shown to be accurate the next goal is to integrate the system with the Metlink website and display boards. Another improvement to the system is to integrate different features into the prediction system, for example, weather and public holidays, these features can have an impact on the occupancy of the vehicle and should be part of the prediction to make it more accurate.

Future goals have been in discussion to use this system to integrate real-time data that is in progress to being added to the vehicles in the future, this will result in more accurate occupancy values for the vehicle that is approaching the station that the passenger is waiting for. When the real-time data isn't available the predicted value is presented instead, the real-time data can also be used to update the prediction for the upcoming journeys. In case of an event that the prediction didn't take into account, the real-time data can assist the prediction to make it up to date with the latest features known.

Bibliography

[1] ALSTOM. Optimet real-time train occupancy: A smoother passenger flow on platforms.

[2] CAMERO, A., TOUTOUH, J., STOLFI, D. H., AND ALBA, E. Evolutionary deep learning for car park occupancy prediction in smart cities. In *International Conference on Learning and Intelligent Optimization* (2018), Springer, pp. 386–401.

[3] DALE, K. *Data visualization with python and javascript: scrape, clean, explore & transform your data.* " O'Reilly Media, Inc.", 2016.

[4] GILLES, V., AND PIETER, C. Predicting train occupancies based on query logs and external data sources.

[5] GOVERNMENT, N. Z. History of the covid-19 alert system.

[6] HUGHES, J. Why functional programming matters. *The computer journal 32*, 2 (1989), 98–107.

[7] JOSE, B., AND ABRAHAM, S. Performance analysis of nosql and relational databases with mongodb and mysql. *Materials Today: Proceedings 24* (2020), 2036–2043. International Multi-conference on Computing, Communication, Electrical & Nanotechnology, I2CN-2K19, 25th & 26th April 2019.

[8] LI, Y., AND MANOHARAN, S. A performance comparison of sql and nosql databases. In *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)* (Aug 2013), pp. 15–19.

[9] PABLO, R. Ml approaches for time series.

[10] SEVERANCE, C. Discovering javascript object notation. *Computer 45*, 4 (April 2012), 6–8.