2020

ELEC 490 CAPSTONE GROUP 12 Traffic Prediction

SUPERVISOR: DR. IL MIN KIM

ANN FERNANDES, SERENA SINCLAIR, LUVIT CHUMBER, SASANKA WICKRAMASINGHE

Executive Summary

This report investigates a potential alternative design to the implementation of RSUs for the prediction of autonomous vehicle traffic patterns. The design comprises of a series of LSTM models that are implemented for multiple intersections. In the problem definition, it is stated that the data must be accurate and real signifying that simulated data generation is not accepted, and the model should successful in predicting the traffic patterns of vehicles. In this solution, the two-design criterion are met – accurate and real data for 6 intersections were gathered and used for training and testing, and the prediction of traffic was achieved with an average accuracy of 97% through the 6 chosen intersections.

When designing the final product, the current solutions regarding traffic prediction was initially researched to understand the options that exist. This research consisted of an analysis on the current infrastructure, environmental, social and economic impact, and current solutions for autonomous vehicles. This was done by finding various scholarly articles from organizations such as Massachusetts Institute of Technology (MIT), The Governor's Highway Safety Association (GHSA), etc. It was determined that there is a clear lack of infrastructure in the industry as it is in its early stages of development.

After these current solutions were defined, the design and approach for the implementation was investigated. Multiple datasets were collected from the City of Victoria located in Melbourne, Australia. This data met the criterion as it was historical traffic data from 2014 that was maintained by the State of Victoria. The different datasets contained information on date, time, traffic volume, location coordinates, accidents, conditions of road, weather conditions, and etc. Preprocessing was completed to extract the correct features required for the model. An LSTM model was then built to predict the traffic patterns of the intersection located at Elizabeth St and Little Collins St. The model was then expanded to 6 intersection to widen the scope of the predictions.

Finally, a stakeholder analysis was completed alongside additional recommendations on scaling the implemented solution to meet greater needs which includes the effects of seasons and reaching wider range of intersections to create an interconnected network.

Table of Contents

Table of Figures

Introduction

With the rise of autonomous vehicles and continued expansion of urban areas, the scale of traffic and the number of vehicles is rapidly increasing. As a result, traffic congestion has evolved into an unavoidable problem. It is necessary to make changes to the current road infrastructure to support these vehicles and help reduce the side effects of increasing traffic.

In recent years, intelligent transportation systems (ITS) have been used and accepted in many countries today, not only to limit traffic congestion but to increase road safety and efficient infrastructure usage [1]. The key component to ITS is the traffic prediction model. It is capable of providing essential information to drivers to allow for better decision making. All while reducing problems such as accidents, congestion and pollution by relaying information to vehicles through roadside units (RSUs) [1]. A study conducted in 2008 reported that 40% of car crashes in the United States have occurred at intersections [2]. Therefore, accurate traffic predictions at intersections is important for maintaining constant mobility and reducing congestion.

There are two primary challenges in predicting traffic, when dealing with intersections. The first one is that traffic prediction is non-linear and dynamic problem [3]. Recurrent traffic patterns can be found over time, however, nonrecurring events (accidents and roadwork) can heavily impact the overall accuracy. Therefore, incorporating such events could potentially increase traffic prediction accuracy. Other factors that further effect the traffic model are day of the week, peak time, time, and seasons. The second challenge is finding precise and abundance of historical data for the model to obtain accurate results. Since non-linear events were to be included, it was decided that multiple sources of data must be analyzed and combined to make our final dataset. Along with traffic volume at intersections, the road accidents and roadworks dataset were used to extract information regarding whether a non-linear event happened near the intersection at a specific time.

After the data was preprocessed using KNIME, the data was then used to train and test our final model. Since the aim of the paper is to increase accuracy of the traffic model using non-linear events, it was determined that a bidirectional long short term memory (LSTM) model would be implemented due to its predictive nature and its ability to “preserve the long-term effect of the data” when using time series data [4]. The model outputs a value from 1 to 3 for each direction of the intersection, with the higher number indicating more congestion.

To evaluate the model, real traffic datasets from 2014 originating from Melbourne, Victoria located in Australia was used. It consisted of intersection traffic volume data, accident data, and roadwork data. When comparing the existing methods to the proposed approach, the results reveal that the proposed idea increased the overall accuracy of predictions.

Background and Motivation

The successful deployment of autonomous vehicle networks depends on the adoption of new wireless technology, the dedicated short-range communications (DSRC). Over the past years, the number of vehicles on the road have been increasing rapidly. Solutions must be developed and implemented to mitigate issues that arise from this large growth. The issues that come to light with this increase include but are not limited to congestion, pollution, accidents, fatalities, etc. As a means to reduce these effects, autonomous vehicles are being implemented worldwide. The Governor’s Highway Safety Association (GHSA) predicts that by year 2050, 40% to 60% of vehicles in the United States operating on the road will be autonomous, as shown in Figure 1 [5].

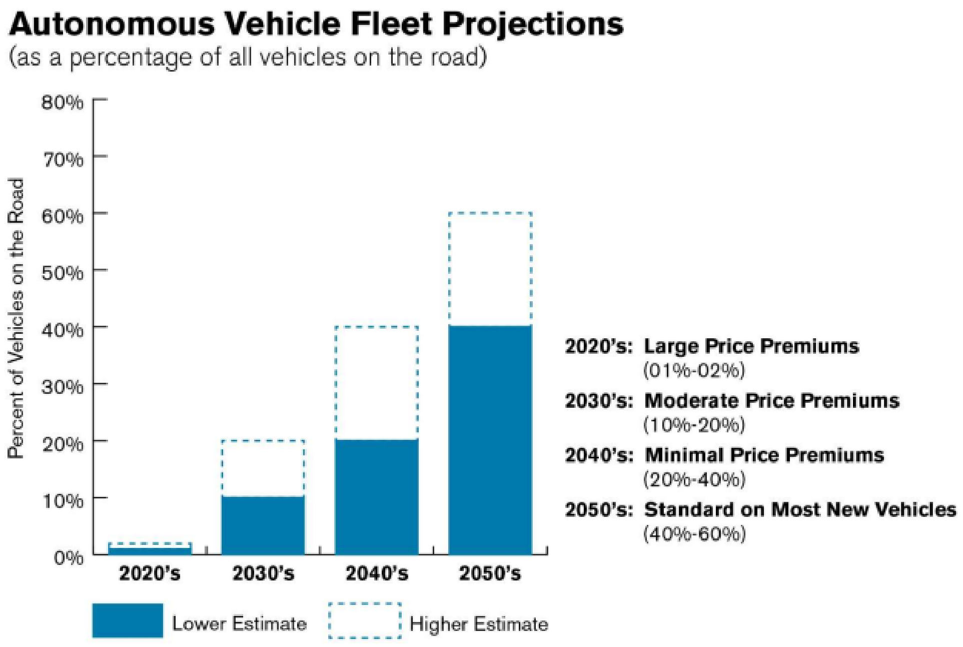


Figure 1 – 30-year projection depicting percentage of autonomous vehicles on the road.

Current infrastructure and solutions

However, with the development of autonomous vehicles being in their early stage cycle, the infrastructure surrounding them have not yet been established. The primary focus in the industry as of now is to help build signs, charging stations, etc. to help setup the foundation for autonomous vehicles. However, to optimize the efficiency of these vehicles, these methods are not ideal. Automakers such as Tesla, Ford, BMW, Mercedes-Benz, etc. utilize cameras to detect road signs, and traffic lights to tell their vehicles about the operations they should perform [6]. However, cameras are not very reliable depending on the situation. They cannot interpret the hand signs of construction workers on the sides of the road. Furthermore, the cameras are constrained by the conditions of the environment. Environmental effects such as fog, rain, snowfall, and strong low-angle sunlight can highly restrict the capabilities in reading road conditions.

Proposed Solution

With RSUs, information regarding the road can be transmitted without restrictions as observed with the implementation of cameras. This creates a more efficient and safer system for both the occupants and other vehicles on the road. By incorporating/adopting machine learning with the RSUs, future traffic predictions can be relayed to the autonomous vehicles. This is accomplished by feeding the machine learning model data of the road conditions, which would then output values regarding the conditions of the road. The information will disclose whether the upcoming areas are busy to redirect the vehicles. The approach is explained further in the subsequent sections of the report.

Impact

The proposed solution solves issues ranging from social to environmental. Being an addition to the current infrastructure will allow for growth in the autonomous vehicle industry. As the space becomes more established, these vehicles will become more safe, reliable, and efficient giving consumers more incentive to switch over from combustion powered vehicles.

Since the roadside units (RSU) will have the ability to provide information to the different autonomous vehicles, there will be a reduction in congestion seen on the roads If there is congestion up ahead, the RSU can transmit this information to redirect the vehicles to where traffic is the least congested. This will ultimately lead to lesser number of cars piling up into congested areas and lower unproductive hours. Furthermore, there will likely be a decrease in accidents/fatalities in the areas as it will allow for a better flow of traffic. A study done by the US Nation Library of Medicine National Institutes of Health found that there was a clear positive linear correlation between traffic volume and total accidents [7]. Another societal impact is the increased incentive for buyers to move from combustion engine powered vehicles to electric vehicles. Currently, there is not much infrastructure for these vehicles, as explained earlier, to make them a safe and reliable source of transportation. This is one of the greatest deterrents for consumers when choosing whether to purchase an autonomous vehicle or not. In 2017, a study conducted by MIT on 3000 participants found that nearly 48% of them were not comfortable with the current standards of autonomous vehicles, hence they would not invest in one. This can be observed on Figure 2 [8]. Another study completed by Cox Automotive determined that consumers' interest for autonomous features were high, but drivers view self-driving cars as less safe. This is evident as 49% of respondents said they would not own a fully autonomous vehicle, known in the industry as Level 5 vehicles [9].

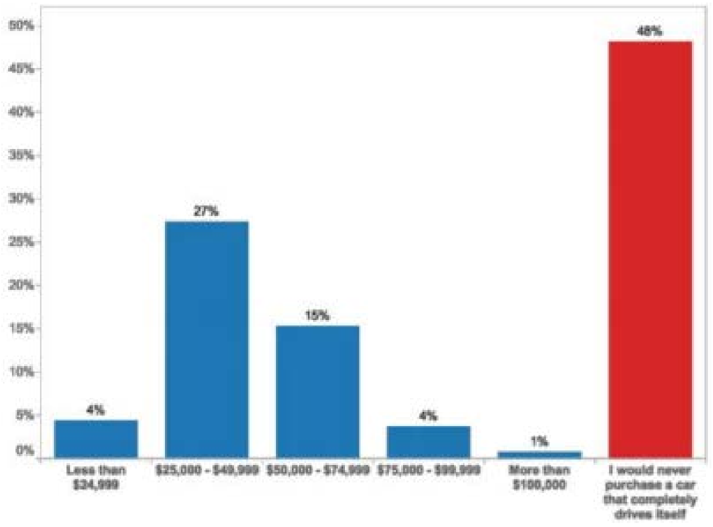


Figure 2 – Number of participants in MIT study that would consider paying autonomous vehicles.

With the increased incentive, there will be indirect impacts environmentally. The biggest impact that can be observed is the reduction of gas consumption. Since majority of autonomous vehicles are electric powered, this would lead to an overall decrease in combustion engine powered vehicles. This is supported by a study done by the United States Department of Energy (DOE) which ascertained there would be a 90% reduction in consumption in transportation from autonomous vehicles [10]. Ultimately, with the scaling down of gas consumption and traffic congestion there will be a rise in economic benefits. Economic benefits include accessibility to cheaper transportation and less time spent on the road spent elsewhere. A SWOT analysis of the solution was completed to gauge the strengths, weaknesses, opportunities, and threats.

Table 1 – SWOT analysis of the solution.

Strengths	Weaknesses
- Less consumption of gas - Lower traffic congestions - Reduction in road accidents and fatalities - Expansion into other features	- Measuring effects of weather - Difficulty finding sufficient data - Many moving pieces literally and figuratively
Opportunities	**Threats**
- Industry is in early stages of life cycle, which is the most attractive time to enter - Integration with other industries of transportation	- Reduction of jobs in transportation sectors - Strong reliance on government policies and laws

Design and Approach

Data Integration and Preprocessing

Datasets

Since one of the main requirements of this project was to use real and accurate traffic data, we made use of multiple data sources provided by the city of Victoria. The datasets chosen are publicly accessed by the Victorian Government Data Directory and maintained by the State of Victoria in Australia [11]. For this project, we used all the data collected in the year 2014. Table 2 gives a summary of the datasets used.

Table 2 – Dataset Summary

Dataset	Time Range Used	Number of Records
Traffic Volume	1 January 2014 – 31 December 2014	35,040 per intersection
Car Accidents	1 January 2014 – 31 December 2014	14,247
Roadworks	1 January 2014 – 31 December 2014	5,637

The main source of data used was the "intersection traffic volume data" set [11], which contains sensor data collected in Melbourne, Victoria located in Australia, along with some suburban areas split into separate files for each day. Each intersection has one sensor installed in every lane which counts the number of cars that pass in 15-minute intervals. It was decided that the project will focus on a small number of intersections located in the downtown district of Melbourne as seen in Figure 3. The following features were extracted from the dataset: date, time, traffic volume, day of the week, and sensor number.

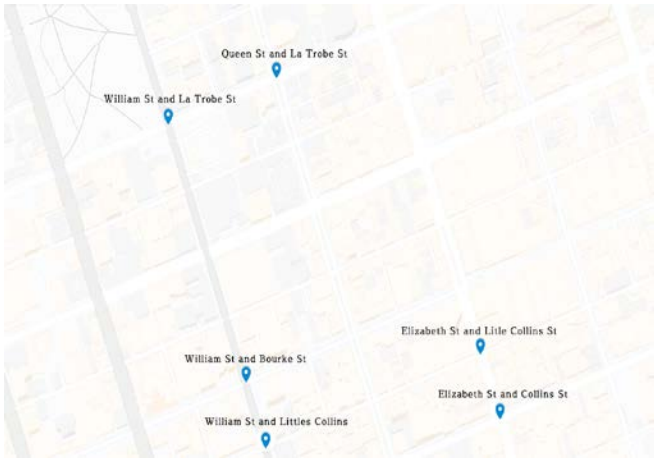

Queen St and La Trobe St
William St and La Trobe St
Elizabeth St and Litle Collins St
William St and Bourke St
Elizabeth St and Collins St
William St and Littles Collins

Figure 3 - Chosen Intersections

The next dataset used was the accident and crashes dataset which contains attributes such as coordinates(longitude and latitude), date, time, accident type (i.e., collision with another vehicle or pedestrian or stationary object), road number (which road the collision occurred), conditions of road surface (i.e., dry or wet), and weather conditions (i.e. raining or clear). The following attributes were extracted from the dataset: date, time, and location coordinates.

The last source was the roadwork and events dataset which contains attributes such as location, type of work (i.e., construction, holiday, events), work start date and time, work end date and time. The following attributes were extracted: location, start and end date and time for the events.

The above datasets are integrated to obtain the input data described in the next subsection. For each intersection, the events from the roadworks and accidents that happen in a one kilometer radius from the intersection are identified using the Haversian distance function and added to the corresponding data entry. The number of events for a data entry are then summed and normalized.

Feature Selection

This section describes the selected features used as input data and why it was chosen. This is an important part since it determines the success of the model. Different features and how they are represented influence the prediction results.

- **Direction of traffic (North, East, South, West):** The count of vehicles travelling in said direction over interval
- **Day of the week:** Traffic patterns differ depending on the day. This feature was one-hot encoded into 7 columns.
- **Peak/off peak:** Set to 1 if the time is between 6:00 to 9:00 and 4:00 to 7:00, and 0 otherwise. This was chosen because traffic is different during peak vs not peak hours
- **Number of Accidents:** The accidents were joined to the original dataset and then normalized into the range [0,1] with any accident count above 3 being treated as 3
- **Number of Events:** The events were joined to the original dataset and then normalized into the range [0,1] with any event count above 30 being treated as 30
- **Time:** The interval at which the data was recorded in minutes, normalized into the range [0, 1] per day
- **Season:** Traffic patterns and road conditions differ depending on the time of year. This feature was one-hot encoded into 4 columns: Winter, Spring, Summer, Fall.

Preprocessing

Data Manipulation

The dataset was manipulated one day at a time to keep processing times fast, but all days were manipulated through the same flow. The dataset originally contained sensor data as an entry and time as a feature. This was transposed as we want to extract features from the sensors not time. Using the info provided to us from VicRoads, the entity that manages the dataset, "Starting from the North-East corner and rotating clockwise around the intersection, number vehicle detectors in the first main road approach" [12]. We were able to aggregate individual sensors into a directional feature. Figure 4 shown below shows the steps required to manipulate the dataset into the basic format containing Day of year, Day of week (number), Time, Peak, latitude, longitude, North, East, South, West. The yellow boxes represent settings that were dependent on intersection info, starting from the top-left to bottom-right these controlled what intersection was selected, longitude and latitude of intersection, and sensor aggregation for each direction.

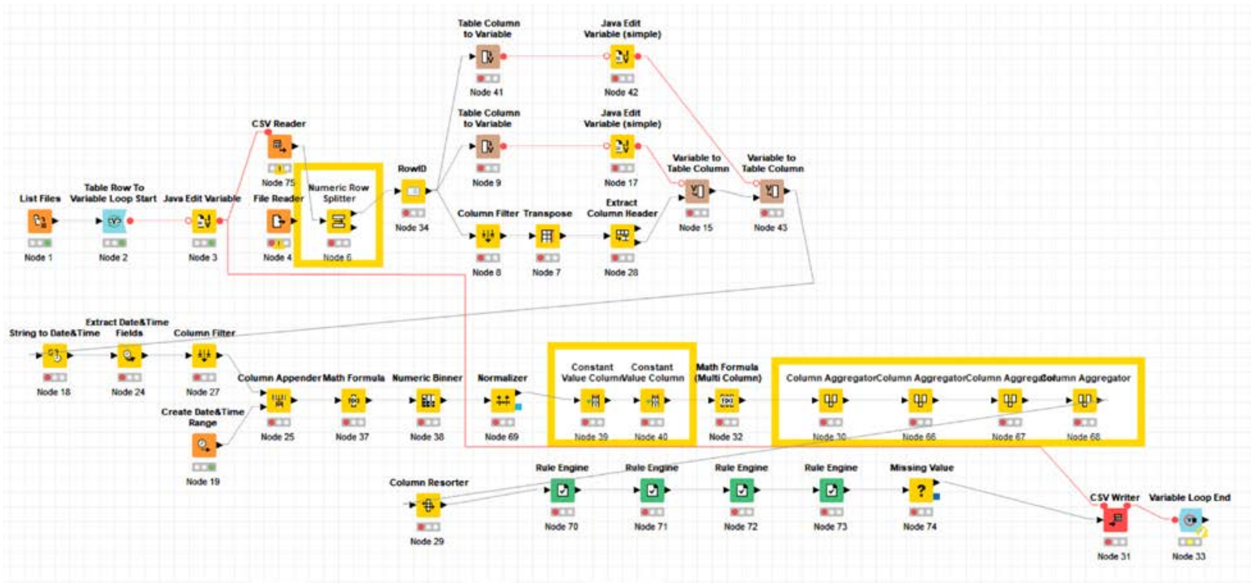


Figure 4 – Initial preprocessing step workflow form Knime

Feature Correction

The dataset was then manipulated into the features required for the models, illustrated in Figure 5 below. The workflow normalizes the traffic counts on a per day per intersection basis into the range [0, 1] and one-hot encodes the seasons and day of week features. The data available after this step was used in training the baseline models.

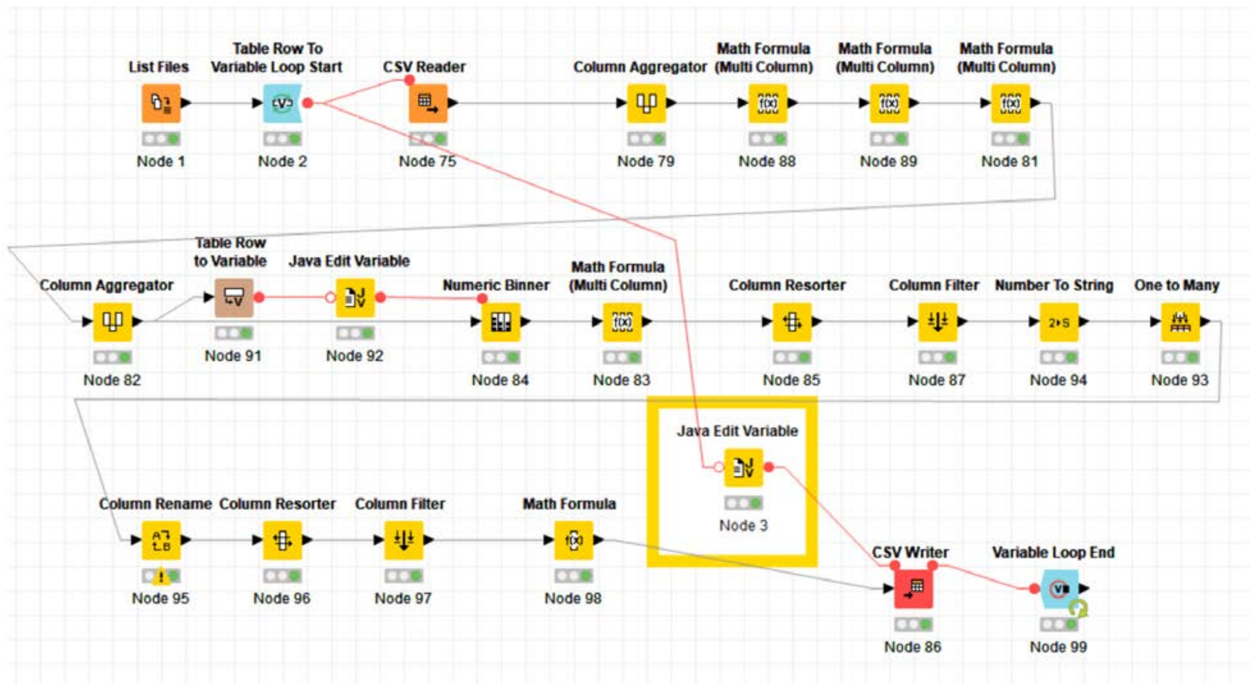


Figure 5 – Baseline model dataset creation workflow from Knime

Events and Accidents Manipulation

The accidents and events are then joined to matched entries through the use of the FindAccidents and FindEvents metanodes in Figure 6 below. These metanodes can be further expanded and viewed in Figure 7 respectively. The added columns are then normalized into the range [0,1] with a threshold value described in Feature Selection above.

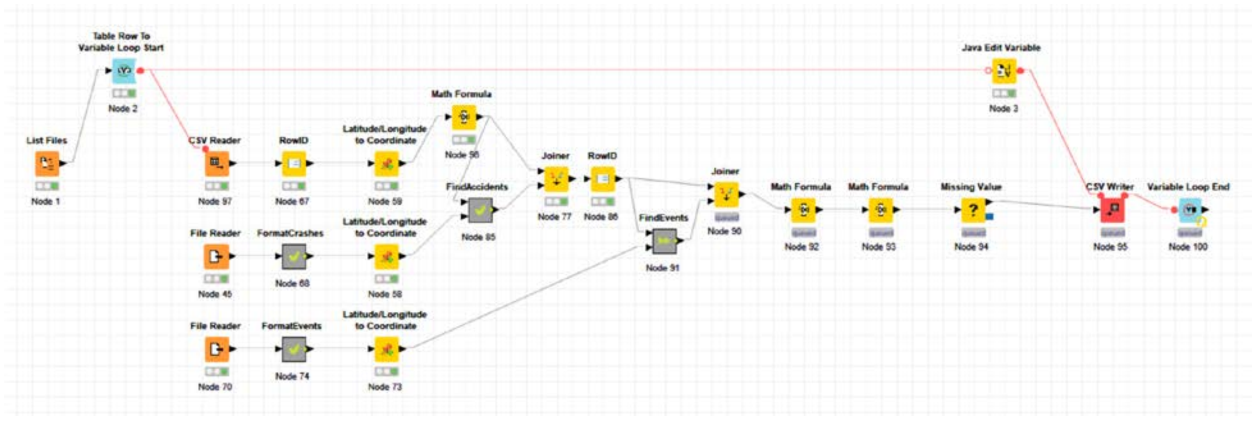


Figure 6 – Final data processing step workflow from Knime

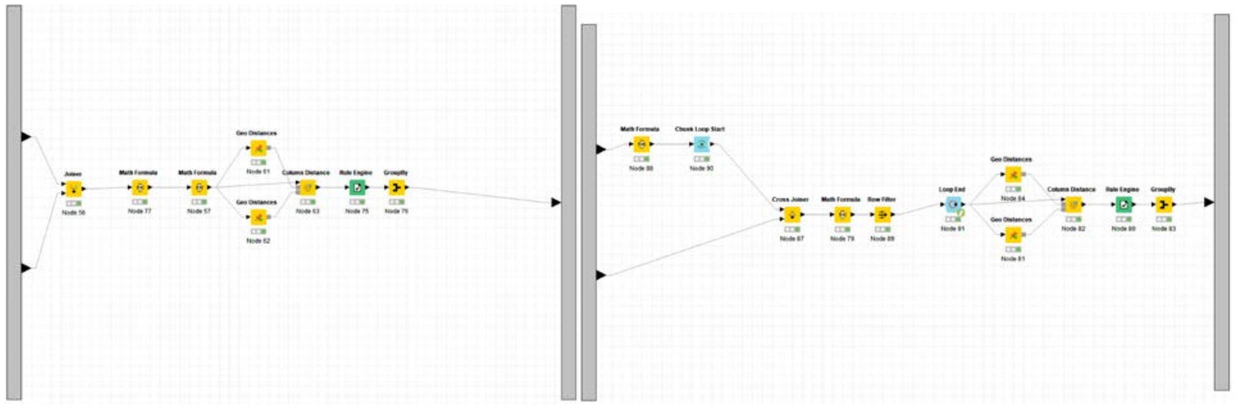


Figure 7 – Metanode Expansion, FindAccidents (Left) and FindEvents (Right)

LSTM Model

Neural networks are great in prediction because of its “unique non-linear adaptive processing ability” [13]. The LSTM layer consists of a series of LSTM units chained together creating the overall LSTM model. The LSTM has the ability to remove/forgot or add information to each unit using three types of memory units or gates as seen in Figure 8. The first type is the input gate which is used to memorize some information from the past. The second type is the output gate which is used for the output, and the last gate, the forget gate, is used to forget some information from the past.

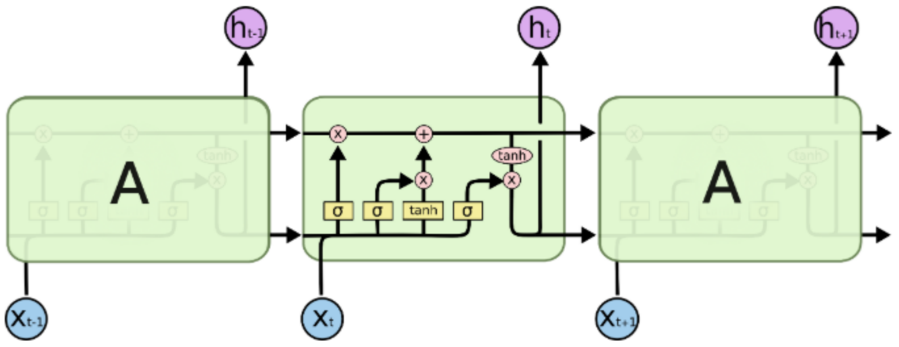


Figure 8 – LSTM model

Model and Implementation

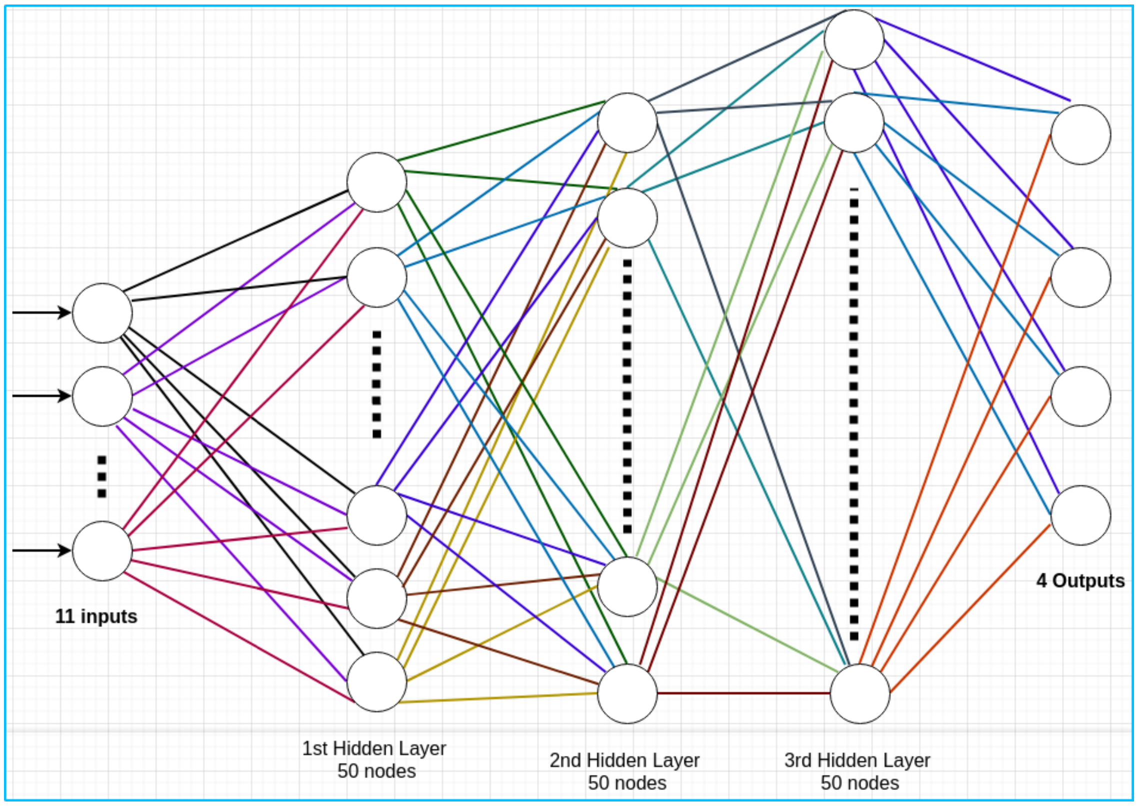


Figure 9 – The final model of LSTM with no events

For our LSTM model, a bidirectional design was used with three hidden layers. A bidirectional model was selected because of the access it has to both past and future data simultaneously. This design has an advantage over a unidirectional model, since a unidirectional model only has access to past events.

In this case, having non-live data, it was possible to use a bidirectional design, because future states are accessible from the current state.

Selecting the size of the model was done partially by research into optimal models for the type of data we have (time series) and partially by trial and error. Initially, a model with three hidden layers with 20 nodes in each layer was chose. Experiments were done by training models with added/reduced layer and node sizes and the model which gave us the best result was selected. This 'trial and error' approach is a common approach used in adaptive algorithms for building neural networks [14].

The activation function used was the hyperbolic-tangent (tanh), since because of the gates in LSTM neural networks, the vanishing-gradient issue does not exist. Using a non-linear activation function allows the model to learn more complex structures in the data [15], which is the reason why tanh was selected. The optimizer used was the Adam optimizer, because of the boost in training speed it is able to achieve. This optimizer used an adaptive learning rate for each parameter, allowing the model to converge quickly [16].

We trained a model per intersection with and without events, meaning for the 6 intersections, we trained a total of 12 models. The model without events has 13 features for the training data; day of week (7 fields total, each field is either 0 or 1), time of day (floating point), season (4 fields total, each either 0 or 1), peak (0 or 1) as seen in Figure 9. For the model trained with events, the training data has 15 features, having accidents in range (floating point) and events in range (floating point) also added to the features. There are 4 total predicted outputs traffic in the North, East, South, West directions of the intersection. The prediction is a classification, with possibilities of 1 (low traffic), 2 (medium traffic), 3 (high traffic) as an output.

The LSTM models were both trained in Google Colab's online Jupyter notebook environment and then exported to a python file to work with the wrapper. The notebook was set to use the GPU hardware accelerator option to speed up the training time reduce the resource usage. These models use Python 3.

Results

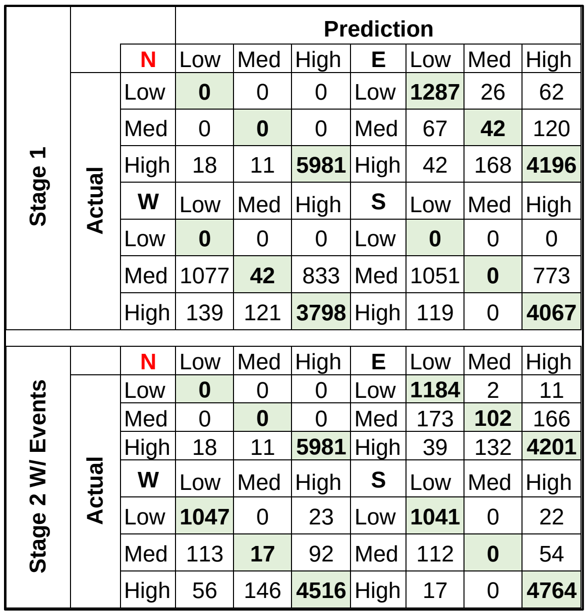
For training and testing purposes, we used the full dataset for 2014. The dataset was further broken down to 80% for training and 20% for testing with the records shuffled so that the model did not overfit. Training for both models took close to an hour with little difference in time between the models. The most time-consuming part was the preprocessing, which took over two hours per intersection for the events to be correlated to the correct data entries. The dataset which did not include events only took 30 minutes to generate.

In the following section, we compare the accuracy of the proposed models: events and no events for six intersections located in the downtown core of Melbourne, Australia. From Table 3 it can be seen that by incorporating events in the data processing set the accuracy of the model increases greatly. This shows that non-recurrent patterns have a great impact on traffic and should be used to help increase accuracy.

Table 3 - Accuracy of Model

Intersection ID	**Accuracy**	
	No Events	With Events
2922	99.0%	99.0%
2924	86.5%	99.0%
4580	71.0%	98.5%
4589	85.0%	95.0%
4591	87.5%	95.0%
4600	88.5%	96.0%

Figure 10 is a confusion matrix for the intersection ID 4589. The top half is the results without events while the bottom half is with events. Each block includes the results for each direction and everything in the diagonals is what the model was able to predict correctly (green), the rest are wither false positive or true negatives. From the figure the model did not do well recognizing low congestion for the directions West and South, but when events were used, the model was able to categorize most of the low congestion element and increased the number of correct predictions on the diagonals.



		Prediction							
		N	Low	Med	High	E	Low	Med	High
Stage 1	Actual	Low	**0**	0	0	Low	**1287**	26	62
		Med	0	**0**	0	Med	67	**42**	120
		High	18	11	**5981**	High	42	168	**4196**
		W	Low	Med	High	S	Low	Med	High
		Low	**0**	0	0	Low	**0**	0	0
		Med	1077	**42**	833	Med	1051	**0**	773
		High	139	121	**3798**	High	119	0	**4067**
Stage 2 W/ Events		N	Low	Med	High	E	Low	Med	High
	Actual	Low	**0**	0	0	Low	**1184**	2	11
		Med	0	**0**	0	Med	173	**102**	166
		High	18	11	**5981**	High	39	132	**4201**
		W	Low	Med	High	S	Low	Med	High
		Low	**1047**	0	23	Low	**1041**	0	22
		Med	113	**17**	92	Med	112	**0**	54
		High	56	146	**4516**	High	17	0	**4764**

Figure 10 - Confusion Matrix of Intersection ID 4589

Stakeholders and Ethics

The analysis of stakeholders for a project is crucial in assessing its viability. There are a wide range of stakeholders that must be considered for this project. They include but are not limited to automotive manufacturers, regulators/legislators, professional transportation drivers, energy and fuel companies, auto repair shops, insurance companies and etc. The primary stakeholders are the automakers, energy and fuel companies, and drivers. These stakeholders are key in the deployment of this system into the real world. The secondary stakeholders include the regulators/legislators, insurance companies, auto repair shops, ride hailing companies, etc. These stakeholders have an indirect influence on the deployment however are considered since their response to the solution will affect the behavior of other consumers.

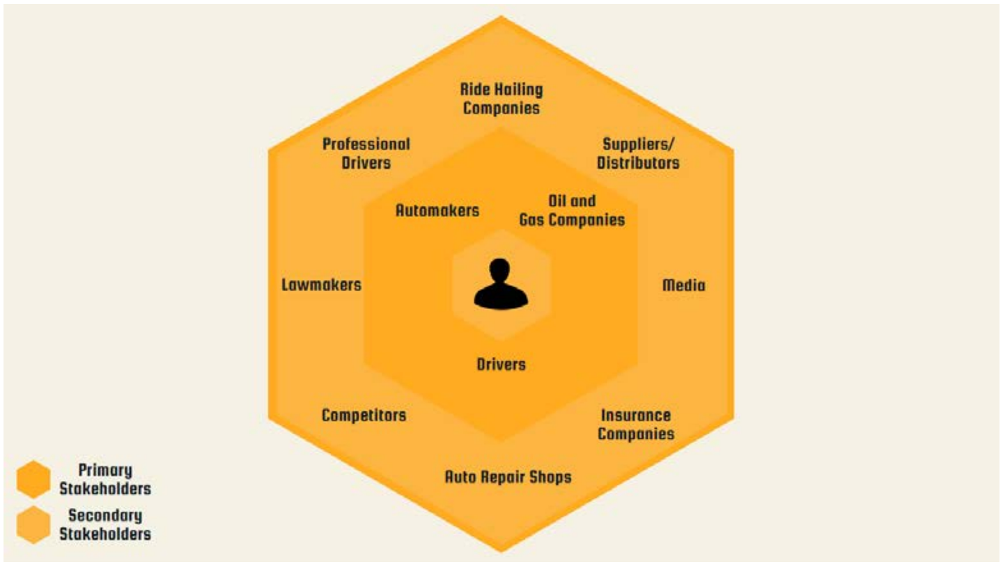


Figure 11 – Stakeholder map depicting the primary stakeholders and secondary stakeholders.

Safety and privacy are common issues seen with the any implementation of autonomous vehicles. As explained earlier the increased safety on the roads will allow occupants to feel less anxious due to the combination of less time spent on the road and in congested areas where accidents are inevitable. Additionally, it will also give rise to opportunities for elderly people and those with disabilities to travel safer without the need to be behind the wheel. Privacy concerns were also taken into consideration. Since the model does not collect any personal information of the occupants in the vehicles, this solution does not add to the privacy risks with accessing sensitive information.

Some ethical problems were considered when looking into the possibility of this solution. Because of the automation of driving, professional transportation drivers will see a decrease in jobs since companies will be able to manage their own fleet of vehicles. This is clearly evident in the application of taxis and semi-trucks where the RSUs will ultimately help these vehicles communicate their way to delivering.

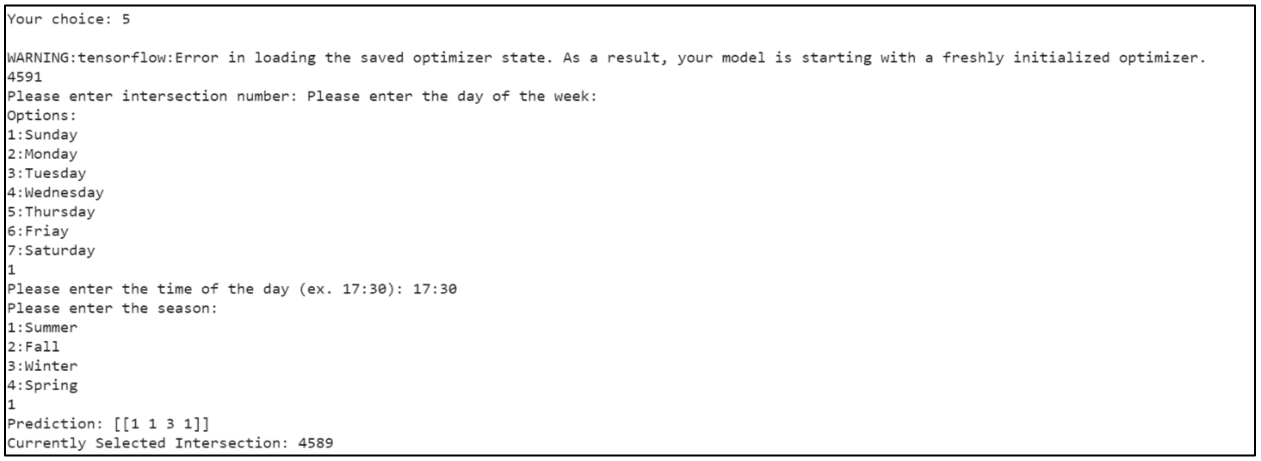
The cost and manufacturability of this project is low due to the fact that the solution is primarily software based. If the model is expanded into hundreds of intersections, more computing power may be required to speed the process of training the models for each of the intersections as well as storing each model. This may lead to implementing a more intensive GPU and memory which ranges in cost depending on a number of factors. However, the cost is relatively low compared to the gain of predicting traffic predictions.

Performance Specification

Table 4 – Performance specifications from blueprint

	Specification	Specification met?
1	**Functional Requirements**	
1.1	Accept traffic data for any location and map	No
1.2	Analyze traffic data and use it to train the neural net	Yes
1.3	Use a customized deep learning algorithm to analyze data and determine future traffic patterns.	Yes
1.4	Expanding algorithm to identify future traffic patterns for larger locations.	Yes
2	**Interface requirements**	
2.1	Console view	Yes
2.2	Output text file containing future traffic predications for the location and time span chosen	Yes
2.3	Output map image with traffic patterns identified (ex: green for low traffic, red for high traffic) for a specific time window.	No
3	**Performance requirements**	
3.1	Determine future traffic patterns given a location and traffic data. +75% accuracy	Yes

The final product includes a console view which enables the user to choose one of the six intersections to train and test using a combination of traffic data, road accidents, and roadworks as seen in Figure 13. This was implemented by creating a wrapper over the initial model, allowing it to be user friendly to train and test. The console accepts the data to train the model with and saves snapshots, this way the user does not have to repeatedly train the model for that particular intersection. Furthermore, the wrapper is capable of route check, where the user inputs a day of the week, season, time, peak, and number of events and the program will predict at what level the traffic is currently at in the intersection for a particular direction.



```
Your choice: 5

WARNING:tensorflow:Error in loading the saved optimizer state. As a result, your model is starting with a freshly initialized optimizer.
4591
Please enter intersection number: Please enter the day of the week:
Options:
1:Sunday
2:Monday
3:Tuesday
4:Wednesday
5:Thursday
6:Friay
7:Saturday
1
Please enter the time of the day (ex. 17:30): 17:30
Please enter the season:
1:Summer
2:Fall
3:Winter
4:Spring
1
Prediction: [[1 1 3 1]]
Currently Selected Intersection: 4589
```

Figure 12 - Console output from generated sample prediction

The final product meets most of the specifications outlined in the initial design phase as seen in Table 4. In total, only two specifications out of the nine were not met. However, that did not hinder the final performance of the project. Many of these specifications were established before a formal neural network or deep learning course was taken, thus some are not applicable. The first specification not met was accepting traffic data and map. The console only accepts traffic data following a certain convention seen in the features section, but no map was used as the focus is to train a specific intersection. The second specification not met was outputting a map with the traffic patterns identified. We decided this was not needed since the goal of this project is to prove that by incorporating nonlinear elements, we could improve the accuracy of the model. The other requirements were met since real traffic data was used to train and test our customized model and the model was expanded to include 6 intersections in the downtown area of Melbourne, Australia. The accuracy of our model was always higher than 75% as described in the results section.

Conclusion

Accurate traffic predictions are an important part for ITS and DSRC deployment. This paper investigated the problem of traffic predictions at intersections by designing a nonlinear neural network model. Since accidents and roadworks greatly affect the traffic flow at intersections, multiple sources of data were used and combined as training and testing data for the model. The data was then used to train a LSTM model and the final results where compared to the results of the same model without the events feature. The result showed that by incorporating nonlinear events, we were able to increase the accuracy of the model greatly. Real-world datasets were used and collected which further show that our model is applicable to real world situations. In future work, we plan to introduce new events such as road conditions and visibility conditions, while also experimenting with different neural and deep learning networks.

Overall Effort

Table 5 – Effort Overall

Name	Percentage
Ann Fernandes	100%
Luvit Chumber	100%
Serena Sinclair	100%
Sasanka Wickramasinghe	100%

The whole team put an equal effort in the project overall. Because we were able to modularize the final project, we each took ownership of a certain aspect while still receiving support from our group members. Luvit and Sasanka took charge of the preprocessing stage, while Serena and Ann focused on designing the model. Additionally, finding the dataset was a team effort. Overall, we were on schedule for the milestone, with a little delay due to the COVID-19 outbreak situation.

References

[1] "ITS Research Fact Sheets - Benefits of Intelligent Transportation Systems," [Online]. Available: https://www.its.dot.gov/factsheets/benefits_factsheet.htm. [Accessed 02 April 2020].

[2] "National HighwayTrafficSafetyAdministration. Crash Factors in Intersection- Related Crashes: An On-Scene Perspective, 2018," [Online]. Available: http://www-nrd.nhtsa.dot.gov/Pubs/811366.pdf . [Accessed 02 April 2020].

[3] W. Z. S. W. a. Y. W. W. Alajali, "Intersection Traffic Prediction Using Decision Tree Models," *Symmetry,* vol. 10, no. 9, p. 386, 2018.

[4] "Colah," 27 August 2018. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accessed 02 April 2020].

[5] J. Hedlund, "Autonomous Vehicles Meet Human Drivers: Traffic Safety Issues for States," Governors Highway Safety Association.

[6] "oem1stop," 2010. [Online]. Available: https://www.oem1stop.com/sites/default/files/2013-12_Fahrerassistenzsysteme_EN.PDF.

[7] A. E. B. O. Retallack, "Current Understanding of the Effects of Congestion on Traffic Accidents," 2019.

[8] B. R. B. S. C. F. B. Hillary Abraham, "Consumer Interest in Automation: Preliminary," 2017. [Online]. Available: https://agelab.mit.edu/sites/default/files/MIT%20-%20NEMPA%20White%20Paper%20FINAL.pdf.

[9] B. Reimer, "Consumer Interest in Automation: Change over One Year," 2018.

[10] "Study of the potential energy consumption impacts of connected and automated vehicles," U.S Energy Information Administration, Washington, 2017.

[11] "Victorian Government Data Directory," [Online]. Available: https://www.data.vic.gov.au/ . [Accessed 20 October 2019].

[12] C. E. Consultant, "ETS Enquiry #501674143 "email"," 2019.

[13] W. H. W. a. H. M. Wei, "Wei, Wangyang, Honghai Wu and Huadong Ma. “An AutoEncoder and LSTM-Based Traffic Flow Prediction Method.," *Sensors,* 2019.

[14] G. D. a. M. N. V. Magoulas, "Magoulas, George D. and Michael N. Vrahatis. “Adaptive Algorithms for Neural Network Supervised Learning: a Deterministic Optimization Approach," *I. J. Bifurcation and Chaos,* vol. 16, pp. 1929-1950, 2006.

[15] S. Sharma, "Activation Functions in Neural Networks," Towards Data Science, 6 September 2017. [Online]. Available: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6. [Accessed 02 April 2020].

[16] "ICLR 2015 Conference for Neural Networks," 7 May 2015. [Online]. Available: https://www.iclr.cc/archive/www/doku.php%3Fid=iclr2015:main.html. [Accessed 5 March 2020].

[17] J. Levy, "Evaluation of the public health impacts of traffic congestion: a health risk assessment," 2010. [Online]. Available: https://ehjournal.biomedcentral.com/articles/10.1186/1476-069X-9-65.

Appendix A: Model Code

Data_processing.py

```
from
sklearn.model_selection
import train_test_split
from sklearn.preprocessing import MinMaxScaler
import numpy as np

sc = MinMaxScaler(feature_range=(0, 1))
features_idx_events =   [6, 7, 8, 9, 10, 11, 12, 13, 16, 17, 18,
19, 20, 26, 27]
#features_idx_events =   [6, 7, 8, 9, 10, 11, 12, 13, 16, 17, 18,
19, 20, -2, -1]
features_idx =          [6, 7, 8, 9, 10, 11, 12, 13, 16, 17, 18,
19, 20]
labels_idx = [-13, -12, -11, -10]
#labels_idx = [21, 22, 23, 24]


def preprocessing(df, events=True, verbose=False):
        features = df.iloc[:, features_idx]
        labels = df.iloc[:, labels_idx]
        if events:
                # TODO: update logic to drop last 2 columns
                features = df.iloc[:, features_idx_events]

        if verbose:
                print(features)
                print(labels)

        x_train, x_test, y_train, y_test =
train_test_split(features, labels, test_size=0.2, shuffle=True,

                                    random_state=1997)

        x_train, y_train = np.array(x_train), np.array(y_train)
        x_train = np.reshape(x_train, (x_train.shape[0],
x_train.shape[1], 1))

        x_test, y_test = np.array(x_test), np.array(y_test)
        x_test = np.reshape(x_test, (x_test.shape[0],
x_test.shape[1], 1))
```

```
        return x_train, x_test, y_train, y_test


def confusion_matrix(predict, actual, Verbose=False):
        '''               p Low | p Med | p High
        a Low      |
        a Med      |
        a High    |
        '''

        W = [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0,
0]]
        S = [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0,
0]]
        N = [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0,
0]]
        E = [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0,
0]]
        for i in range(len(predict)):
                pred = predict[i]
                pred = [int(round(p, 0)) for p in pred]
                actu = actual[i]
                actu = [int(round(a, 0)) for a in actu]
                # print(actu)
                W[actu[0]][pred[0]] += 1
                S[actu[1]][pred[1]] += 1
                N[actu[2]][pred[2]] += 1
                E[actu[3]][pred[3]] += 1

        N[0][0] = ''
        N[0][1] = 'p Low'
        N[0][2] = 'p Med'
        N[0][3] = 'p High'
        N[1][0] = 'a Low'
        N[2][0] = 'a Med'
        N[3][0] = 'a High'

        E[0][0] = ''
        E[0][1] = 'p Low'
        E[0][2] = 'p Med'
        E[0][3] = 'p High'
        E[1][0] = 'a Low'
```

```
E[2][0] = 'a Med'
E[3][0] = 'a High'

S[0][0] = ''
S[0][1] = 'p Low'
S[0][2] = 'p Med'
S[0][3] = 'p High'
S[1][0] = 'a Low'
S[2][0] = 'a Med'
S[3][0] = 'a High'

W[0][0] = ''
W[0][1] = 'p Low'
W[0][2] = 'p Med'
W[0][3] = 'p High'
W[1][0] = 'a Low'
W[2][0] = 'a Med'
W[3][0] = 'a High'

if Verbose == True:
        print()
        print("North Confusion Matrix")
        print(N[0])
        print(N[1])
        print(N[2])
        print(N[3])
        print()
        print("East Confusion Matrix")
        print(E[0])
        print(E[1])
        print(E[2])
        print(E[3])
        print()
        print("South Confusion Matrix")
        print(S[0])
        print(S[1])
        print(S[2])
        print(S[3])
        print()
        print("West Confusion Matrix")
        print(W[0])
        print(W[1])
        print(W[2])
```

```
		print(W[3])

	return N, S, E, W


def read_intersections(file_loc, verbose=False):
	intersections = list()
	f = open(file_loc, "r")

	keys = f.readline().rstrip('\n').split(',')
	if verbose:
		print("Keys:", keys)
		print("Intersections Data:")

	lines = f.readlines()
	for line in lines:
		line = line.rstrip('\n')
		line = line.split(',')
		intersection_dict = dict()
		for i, x in enumerate(line):
			intersection_dict[keys[i]] = x
		if verbose:
			print(intersection_dict)
		intersections.append(intersection_dict)

	f.close()
	return intersections


def save_intersections(intersections, file_loc):
	print("Saving updated intersections file")
	f = open(file_loc, "w")

	keys = intersections[0].keys()
	keys = ",".join(keys) + "\n"
	f.write(keys)

	for line in intersections:
		line = ",".join(line.values()) + "\n"
		f.write(line)

	f.close()
```

```
def print_intersections(intersections):
        print("List of Trained Intersections:")
        keys = intersections[0].keys()
        for line in intersections:
                for k, v in line.items():
                        if k == "name":
                                print(f"{k}: {v}")
                        else:
                                print(f"\t{k}: {v}")
                print()
```

dict_bin_search.py

"""This module implements a binary search through the method search(). Given a

key, the method will search through a previously sorted list of dictionaries

and return the index of the first matching value.

Author: Luvit Chumber

Date: 2020-03-23

"""

```
from dict_quad_sorts import insertion_sort


def search(data, key, search_val):
    """Searches through a particular key from a list of dictionaries."""
    insertion_sort(data, key)

    low = 0
    high = len(data) - 1

    while low <= high:
        mid = (high + low) // 2
        mid_dict = data[mid]

        if mid_dict[key] == search_val:
            return mid
        elif search_val < mid_dict[key]:
            high = mid - 1
        else:
            low = mid + 1
        return None

    """
```

dict_quad_sorts.py

"""This module implements 4 different quadratic sorts which works on a list of dictionaries and sorts a given key.
Author: Luvit Chumber
Date: 2020-03-23
"""

```
def insertion_sort(data, sort_on_key):
    """Sorts dictionary on key using insertion sort algorithm."""
    for i in range(1, len(data)):
        first_dict = data[i - 1]
        second_dict = data[i]
        while i > 0 and first_dict[sort_on_key] >
second_dict[sort_on_key]:
            data[i - 1], data[i] = data[i], data[i - 1]
            i = i - 1
            first_dict = data[i - 1]
            second_dict = data[i]


def bubble_sort(data, sort_on_key):
    """Sorts dictionary on key using bubble sort algorithm."""
    swap_flag = True
    n = len(data)
    while swap_flag:
        swap_flag = False
        for i in range(1, n):
            first_dict = data[i - 1]
            second_dict = data[i]

            if first_dict[sort_on_key] > second_dict[sort_on_key]:
                data[i - 1], data[i] = data[i], data[i - 1]
                swap_flag = True


def bubble_sort_opt(data, sort_on_key):
    """sorts dictionary on key using optimized bubble sort
algorithm."""
    swap_flag = True
    n = len(data)
    while swap_flag:
        swap_flag = False
        for i in range(1, n):
            first_dict = data[i - 1]
            second_dict = data[i]

            if first_dict[sort_on_key] > second_dict[sort_on_key]:
                data[i - 1], data[i] = data[i], data[i - 1]
                swap_flag = True
        n -= 1


def selection_sort(data, sort_on_key):
    """Sorts dictionary on key using selection sort algorithm."""
    n = len(data)
    for i in range(n - 1):
        min_idx = i  # assume first index is min to start
        min_dict = data[min_idx]
        for j in range(i + 1, n):
            check_dict = data[j]
            if check_dict[sort_on_key] < min_dict[sort_on_key]:
                min_idx = j
                min_dict = data[min_idx]
        if min_idx != i:
            data[i], data[min_idx] = data[min_idx], data[i]
```

Lstm_model.py

```
#'''
from keras.models import Sequential, load_model, save_model
from keras.layers import TimeDistributed, Dense, LSTM, Activation, RepeatVector,
Dropout
from keras.callbacks import ModelCheckpoint
from keras.utils import  to_categorical
#'''
import numpy as np



# steps depends on amount of data to be predicted
```

```
class LSTMModel:
        def __init__(self, x_train_shape, y_train_shape, intersection_list=None,
num_epochs=50, l_rate=.001, batch_size=50):
                self.model = None
                self.models = []
                self.multi_model = False
                self.current_idx = 0
                self.input_length = x_train_shape
                self.steps = y_train_shape
                self.epochs = num_epochs
                self.l_rate = l_rate
                self.batch_size = batch_size
                if intersection_list:
                        self.intersections=intersection_list
                        self.multi_model = True
                        self.load_network()



        def get_batch(self, x_data, y_data):
                x = np.zeros(self.batch_size)
                y = np.zeros(self.batch_size)
                while True:
                        if self.current_idx >= len(self.x_data):
                                # reset the index back to the start of the dataset
                                self.current_idx = 0
                        x = x_data[self.current_idx:self.current_idx +
self.batch_size]
                        y_tmp = y_data[self.current_idx:self.current_idx +
self.batch_size]
                        y = to_categorical(y_tmp, num_classes=self.possible_classes)
                        # convert all of temp_y into a one hot representation
                        self.current_idx += self.batch_size
                        yield x, y

        def save_network(self, filepath):
                save_model(self.model, filepath, overwrite=True,
include_optimizer=True)

        def load_network(self, filepath=''):
                if self.multi_model:
                        for i in self.intersections:
                                self.models.append(load_model("./model/" + i +
".hdf"))
```

```
              else:
                     self.model = load_model(filepath, compile=True)

       def init_network(self, hidden_size, activation='relu', optimizer='adam',
loss='mean_squared_error', verbose=False):
              model = Sequential()
              model.add(LSTM(units=hidden_size, return_sequences=True,
input_shape=(self.input_length, 1)))
              model.add(LSTM(units=hidden_size, return_sequences=True))
              model.add(LSTM(units=hidden_size, return_sequences=True))
              model.add(LSTM(units=hidden_size))
              model.add(Dense(units=self.steps, activation=activation))
              model.compile(optimizer=optimizer, loss=loss)

              if verbose:
                     print(model.summary())

              if self.multi_model:
                     self.models.append(model)
              else:
                     self.model = model



       def train(self, x_train, y_train, file_loc, validation_split=0.2,
save=True):
              check_pointer = ModelCheckpoint(filepath=file_loc, verbose=1)
              callbacks_list = [check_pointer]
              if self.multi_model:
                     print("Training is not possible with multi model mode!")
              else:
                     self.model.fit(x_train, y_train, epochs=self.epochs,
batch_size=self.batch_size,
                                       validation_split=validation_split,
callbacks=callbacks_list)
              if save:
                     print("Saving Trained Model")
                     self.save_network(file_loc)

       def get_accuracy(self, x_data, y_data, verbose=False):
              if self.multi_model:
                     try:
                            test_output =
self.model[self.intersections.index(x_data[0][0])].predict(x_data)
```

```
                except:
                        print("Model for intersection does not exist!")
        else:
                test_output = self.model.predict(x_data)
        test_output = np.around(test_output, 0).astype(int)
        corr = 0
        incorr = 0
        for pred, label in zip(test_output, y_data):
                pred = pred.astype(int)
                label = label.astype(int)
                for i, j in zip(pred, label):
                        if i == j:
                                corr = corr + 1
                        else:
                                incorr = incorr + 1
                        if verbose == 1:
                                print("Label:", label)
                                print("Prediction:", pred)

        res = corr / (corr + incorr) * 100
        print("Accuracy rate: {res}")
        return test_output

def predict(self, x_data, intersection=0):
        test_output = 0
        if self.multi_model:
                try:
                        test_output =
self.models[self.intersections.index(str(intersection))].predict(x_data)
                except:
                        print("Model for intersection does not exist!")
        else:
                test_output = self.model.predict(x_data)
        test_output = np.around(test_output, 0).astype(int)
        return test_output
```

Main.py

```
from
data_proces
sing import
*
        from lstm_model import LSTMModel
```

```python
from dict_bin_search import search
import pandas as pd
import numpy as np
import os.path
import menu
import re
import sys

def get_dataset(sel_intersection, intersections):
	intersection_file = search(intersections, 'name', sel_intersection)
	intersection_file = intersections[intersection_file]['dataset']
	df = pd.read_csv(intersection_file.strip(' '))
	return df


def main():
	"""Program execution starts here."""
	intersections = read_intersections("intersections.data")
	intersection_list = []

	for i in range(len(intersections)):
		intersection_list.append(intersections[i]['name'].strip(' '))

	#sel_intersection = intersections[0]['name']
	sel_intersection = "4589"

	df = get_dataset(sel_intersection, intersections)
	x_train, x_test, y_train, y_test = preprocessing(df)

	model_file = search(intersections, 'name', sel_intersection)

	main_menu = [
		"CAPSTONE TRAFFIC PREDICTION",

		"Select Intersection",
		"List of Intersections",
		"Train Intersection",
		"Test Intersection (Accuracy)",
		"Route Check"
	]
	train_menu = [
		"Train Mode:",
                "Train from scratch w/ events",
                "Train from file w/ events",
                "Train from scratch",
                "Train from file",
        ]
        route_check_menu = [
                "Train Mode:",

                "Train from scratch w/ events",
                "Train from file w/ events",
                "Train from scratch",
                "Train from file",
        ]

        while True:
                print("Currently Selected Intersection:", sel_intersection)
                choice = menu.do_menu(main_menu)
                model = LSTMModel(x_train.shape[1], y_train.shape[1])
                if choice is None:
                        return  # Exit main() (and program).
                if choice == 1:
                        # Select Intersection
                        temp_menu = ["Please Select a New Intersection"]

                        for line in intersections:
                                option = line["name"] + ": " + line["street"]
                                temp_menu.append(option)

                        choice = menu.do_menu(temp_menu)
                        if choice is not None:
                                sel_intersection = intersections[choice -
1]['name']

                                print(sel_intersection, "set as current
intersection.")


                                df = get_dataset(sel_intersection,
intersections)
                                x_train, x_test, y_train, y_test =
preprocessing(df)


                                model_file = search(intersections, 'name',
sel_intersection)
                                if model_file is not None:
                                    model_file =
intersections[model_file]['model']
                                    #try:

       model.load_network('./model/'+str(sel_intersection)+'.hdf')
                                    #except:
                                    #      print("File for loading model is
not found! Creating a new model from scratch")
                                    #
       model.init_network(hidden_size=50)

              elif choice == 2:
                     # List Intersections
                     print_intersections(intersections)
              elif choice == 3:
                     model = LSTMModel(x_train.shape[1], y_train.shape[1])
                     # Train Intersections
                     # TODO test each option
                     choice = menu.do_menu(train_menu)
                     if choice == 1:
                            x_train, x_test, y_train, y_test =
preprocessing(df, events=True)
                            model = LSTMModel(x_train.shape[1],
y_train.shape[1])

                            intersection_idx = search(intersections,
'name', sel_intersection)
                            model_file = "model/" + sel_intersection +
".hdf"
                            #if os.path.exists(model_file):
                                   #os.remove(model_file)
                            model.init_network(hidden_size=50)
                     elif choice == 2:
                            x_train, x_test, y_train, y_test =
preprocessing(df, events=True)
                            model = LSTMModel(x_train.shape[1],
y_train.shape[1])

                            intersection_idx = search(intersections,
'name', sel_intersection)
                            model_file = "model/" + sel_intersection +
".hdf"
                            if os.path.exists(model_file):
                                model.load_network(model_file)
                        else:
                                print("Model does not exist, starting
from scratch")
                                model.init_network(hidden_size=50)
                elif choice == 3:
                        x_train, x_test, y_train, y_test =
preprocessing(df, events=False)
                        model = LSTMModel(x_train.shape[1],
y_train.shape[1])

                        intersection_idx = search(intersections,
'name', sel_intersection)
                        model_file = "model/" + sel_intersection +
".hdf"
                        if os.path.exists(model_file):
                                os.remove(model_file)
                        model.init_network(hidden_size=50)
                elif choice == 4:
                        x_train, x_test, y_train, y_test =
preprocessing(df, events=False)
                        model = LSTMModel(x_train.shape[1],
y_train.shape[1])

                        intersection_idx = search(intersections,
'name', sel_intersection)
                        model_file = "model/" + sel_intersection +
".hdf"
                        if os.path.exists(model_file):
                                model.load_network(model_file)
                        else:
                                print("Model does not exist, starting
from scratch")
                                model.init_network(hidden_size=50)
                try:
                        e = int(input("Enter Epochs to train for
(Default=50): "))
                        model.epochs = e
                except ValueError:
                        print("Invalid number entered, using default
value")
                model.train(x_train, y_train, './'+ model_file)
```

```
                        intersections[intersection_idx]['model'] = model_file
                        save_intersections(intersections,
"intersections.data")

                elif choice == 4:
                        # Test Intersections
                        model_file = "model/" + sel_intersection + ".hdf"
                        model = LSTMModel(x_train.shape[1], y_train.shape[1])
                        if os.path.exists(model_file):
                                model.load_network(model_file)
                                test_output = model.get_accuracy(x_test,
y_test)
                                N, S, E, W = confusion_matrix(test_output,
y_test, True)
                        else:
                                print("Please train intersection first")

                elif choice == 5:
                        model = LSTMModel(x_train.shape[1], y_train.shape[1],
intersection_list)
                        # Route Check
                        x_data = []
                        day_week = [1,2,3,4,5,6,7]
                        season=[1,2,3,4]
                        time_str = ''
                        flag = 0
                        x_data = []
                        peak = 0

                        while flag == 0:
                                num_inter = input("Please enter intersection
number: ")
                                if num_inter in intersection_list:
                                        flag = 1
                                        num_inter = int(num_inter)
                                        #x_data.append(int(num_inter))
                                else:
                                        print("Intersection not found!")

                        flag = 0
                        while flag == 0:
                                week = [0,0,0,0,0,0,0]
```

```
            num_day = input("Please enter the day of the
week:\nOptions:\n1:Sunday\n2:Monday\n3:Tuesday\n4:Wednesday\n5:Thursday\n6:F
riay\n7:Saturday\n")
            num_day = int(num_day)
            if num_day in day_week:
                flag = 1
                week[num_day-1] = 1
                x_data = x_data + week
            else:
                print("Day of the week not found!")

        flag = 0
        while flag == 0:
            time_day = input("Please enter the time of the
day (ex. 17:30): ")
            temp = time_day.split(':')
            if len(temp) == 2:
                hour = int(temp[0]) * 60
                if hour > 9 and hour < 17:
                    peak = 1
                time = hour + int(temp[1])
                time_d = float(time) / float(1440)
                if time_d > 1.0:
                    print("Please enter time in the
proper format!")

                else:
                    x_data.append(time_d)
                    flag = 1
            else:
                print("Please enter time in the proper
format!")

        flag = 0
        while flag == 0:
            seasons = [0,0,0,0]
            season_input = input("Please enter the
season:\n1:Summer\n2:Fall\n3:Winter\n4:Spring\n")
            season_input = int(season_input)

            if season_input in season:
                flag = 1
                seasons[season_input-1] = 1
                x_data = x_data + seasons
                else:
                        print("Season not found!")

            # add [0,0] for events
            x_data.append(peak)
            x_data = x_data + [0,0]
            x_test = np.array([x_data])
            x_test = np.reshape(x_test, (x_test.shape[0],
x_test.shape[1], 1))
            res = model.predict(x_test, num_inter)
            print("Prediction: " + str(res))

main()
```

Menu.py

```
"""This is the menu module.
Functions:
    do_menu(title, choices)
        Print a text menu, return an int representing the user's choice.
Author: R. Linley
Date: 2019-01-06
Revisions: 2019-12-18, 2020-02-14
"""


def do_menu(menu_in):
    title = menu_in[0]
    choices = menu_in[1:]
    """Display a text menu of choices and return the user's choice.
    Loop on invalid choices.
    All but the last choice are customizeable and are numbered 1, 2, and
    so on.  The last choice is always "X. Exit."  For example:
        My Menu
        1. Do something
        2. Do something else
        3. Do some other thing
        X. Exit
        Your choice:
    Parameters:
        title: A str representing a title for the menu.  (In the example
        above, "My Menu" is the value of title.) If title is the empty
        string, it won't be printed.
        choices: A list of strings representing the menu choices,
    excluding "Exit".  These are presented in order of occurrence in
    the list.  As menu item numbers are generated automatically,
    these should not appear in the list.  (In the example above, the
    list ["Do something", "Do something else",
    "Do some other thing"] is the value of choices.)
Returned value:
    An integer representing the user's choice if the user selects a
    numbered choice, or None if the user selects "x" or "X" to exit.
Note:
    Raises an exception if title is not a string or if choices is
    not a list of strings.
"""
# Check parameter types.
# title is supposed to be a str. Raise an exception if it's not.
if not isinstance(title, str):
    raise TypeError("Error: do_menu() - First argument must be a string.")
# choices is supposed to be a list of strings. Raise an exception if
# it's not.
if not isinstance(choices, list):
    raise TypeError("Error: do_menu() - Second argument must be a list "\
                    "of strings.")
for choice in choices:
    if not isinstance(choice, str):
        raise TypeError(f"Error: do_menu() - {choice} in your second "\
                        "argument is not a string.")
num_choices = len(choices)
# Make a list of the valid choice numbers.
valid_choices = list(range(1, num_choices+1))
while True: # Loop until the user makes a valid choice.
    # Only print the title if it isn't the empty string.
    if title != "":
        title_border = "*" * (len(title) + 4)
        # print menu title with asterisks border
        print(f"\n{title_border}\n* {title} *\n{title_border}\n")
    # print numbered choices
    for choice_num in valid_choices:
        print(f"{choice_num}. {choices[choice_num-1]}")
    print("\nX. Exit\n")
    choice = input("Your choice: ")
    print()
    try:
        choice = int(choice) # Non-int input throws a ValueError.
        if (choice in valid_choices):
                return choice
            except ValueError: # Execution branches here on non-int input.
                pass # Take no action on non-int input.
            if choice in ["x","X"]: # If so...
                return None # Done here. The user wants out.
            # Still here? User made an invalid choice, so...
            print("\nInvalid choice.")
            print("Valid choices: ",end="")
            if num_choices > 0:
                print(f"{', '.join(str(i) for i in valid_choices)} and ", end="")
            print("X (to exit).\nTry again.")


if __name__ == "__main__":
    # Module testing.

    # Trip the built-in exception-raising with bad arguments.
    # First, a non string for the title...
    try:
        do_menu(6, [])
    except TypeError as err:
        print(err)
        # Error: do_menu() - First argument must be a string.
    # Then a non-list for the choices...
    try:
        do_menu("", 6)
    except TypeError as err:
        print(err)
        # Error: do_menu() - Second argument must be a list of strings.
    # Then a non-string in the choices list...
    try:
        do_menu("", ["This", "is", 1, "test"])
    except TypeError as err:
        print(err)
        # Error: do_menu() - 1 in your second argument is not a string.

    print("--- End of exception-raising tests. ---")

    # Test with an empty title, empty menu to make sure it won't break
    # and to show that no blank lines appear for an empty title.
    while True:
        c = do_menu("", [])
        if c is None:
```

```
        break
    else:
        # Since there are no choices, execution should
        # never be able to get here.
        print(f"Problem: a choice of {c} was returned from an empty menu.")

# Test with a single item menu.
m = ["Only choice"]
while True:
    c = do_menu("Testing!", m)
    if c is None:
        break
    else:
        print(f"\nYou chose {c}.")

# Test with multiple menu items.
m = ["This", "That", "The other thing"]
while True:
    c = do_menu("test Menu", m)
    if c is None:
        break
    print("\nYou chose:", c)

print("\nTests complete.")
```

Requirements.txt

```
absl-py==0.9.0
astor==0.8.1
attrs==19.3.0
backcall==0.1.0
bleach==3.1.4
cachetools==4.0.0
certifi==2019.11.28
chardet==3.0.4
colorama==0.4.3
decorator==4.4.2
defusedxml==0.6.0
entrypoints==0.3
gast==0.2.2
google-auth==1.12.0
google-auth-oauthlib==0.4.1
google-pasta==0.2.0
grpcio==1.27.2
```

```
h5py==2.10.0
idna==2.9
importlib-metadata==1.5.2
ipykernel==5.2.0
ipython==7.13.0
ipython-genutils==0.2.0
ipywidgets==7.5.1
jedi==0.16.0
Jinja2==2.11.1
joblib==0.14.1
jsonschema==3.2.0
jupyter==1.0.0
jupyter-client==6.1.2
jupyter-console==6.1.0
jupyter-core==4.6.3
Keras==2.3.1
Keras-Applications==1.0.8
Keras-Preprocessing==1.1.0
Markdown==3.2.1
MarkupSafe==1.1.1
mistune==0.8.4
nbconvert==5.6.1
nbformat==5.0.4
notebook==6.0.3
numpy==1.18.2
oauthlib==3.1.0
opt-einsum==3.2.0
pandas==1.0.3
pandocfilters==1.4.2
parso==0.6.2
pickleshare==0.7.5
prometheus-client==0.7.1
prompt-toolkit==3.0.5
protobuf==3.11.3
pyasn1==0.4.8
pyasn1-modules==0.2.8
Pygments==2.6.1
pyrsistent==0.16.0
python-dateutil==2.8.1
pytz==2019.3
pywin32==227
pywinpty==0.5.7
PyYAML==5.3.1
```

```
pyzmq==19.0.0
qtconsole==4.7.2
QtPy==1.9.0
requests==2.23.0
requests-oauthlib==1.3.0
rsa==4.0
scikit-learn==0.22.2.post1
scipy==1.4.1
Send2Trash==1.5.0
six==1.14.0
tensorboard==2.1.1
tensorflow==2.1.0
tensorflow-estimator==2.1.0
termcolor==1.1.0
terminado==0.8.3
testpath==0.4.4
tornado==6.0.4
traitlets==4.3.3
urllib3==1.25.8
wcwidth==0.1.9
webencodings==0.5.1
Werkzeug==1.0.0
widgetsnbextension==3.5.1
wrapt==1.12.1
zipp==3.1.0
```

Intersections.data

```
name,street,dataset,model,long,lat
2922, Elizabeth and LaTrobe St, data/2922.csv,,144.96129,-37.81034
2924, Willams St and LaTrobe St,data/2924.csv,,144.95642,-37.81174
4580, Elizabeth St and Collins St,data/4580.csv,,144.964018,-37.81622
4589, Elizabeth St and Little Collins St,data/4589.csv,model/4589.hdf,144.96354,-37.81525
4591, William St and Little Collins St,data/4591.csv,,144.95866,-37.81105
4600, William St and Bourke St, data/4600.csv,,144.9583,-37.81571
```