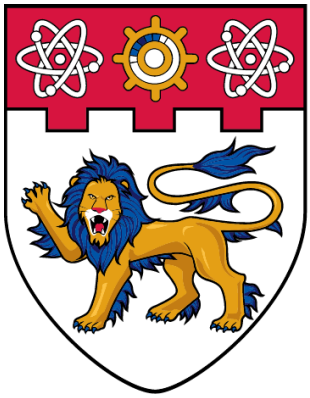


NANYANG TECHNOLOGICAL UNIVERSITY

SINGAPORE

Recommendation System to Improve Driver Performance and Earnings

(Final Report)

Derrick Ng Choon Seng

U2122873F

Supervisors:

Lee Bu Sung, Francis (Associate Professor, Nanyang Technological University)

Seanglidet Yean (Singapore-ETH Center)

Jiazu Zhou (Singapore-ETH Center)

Nanyang Technological University

College of Computing and Data Science

Abstract

The urbanisation of cities brings increasing demand for urban mobility, specifically ride-hailing. As such, to meet the increasing demand, drivers are incentivized by attempting to select optimal trips which maximise earnings and efficiency by reducing idle time. This project seeks to address this by developing a data-driven approach for recommending optimal trips using geospatial-temporal datasets and analysing various aspects of a trip. This approach utilizes algorithm design and the implementation of machine learning techniques such as reinforcement learning to simulate optimal trip selections. Q-learning specifically, was used to train an agent to select the best surge multiplier given certain conditions. The conditions are represented as different states and the surge multiplier values to apply are represented by actions. The final optimal policy which is the Q-table, is then used in the designed algorithm to calculate trip earnings. By taking advantage of the dynamic calculation and better representing driver earnings, recommendations and simulations can represent real-life ride-hailing more accurately. Along with accessibility to predictive data, the system aims to improve drivers' overall performance in the ride-hailing scene. The system results were compared against historical data and achieved at least 37% and up to approximately 163% improvement in driver earnings.

Acknowledgements

I would like to thank Professor Lee Bu Sung and mentors Seanglidet Yean and Zhou Jiazu for their supervision, guidance and encouragement. They have provided me with great insights and constructive feedback throughout this entire project. I would also like thank LTA for the dataset used in this project and Ho Guo Liang Ken for helping with the data extraction.

Table of Contents

1. Introduction

With the introduction of technology, companies such as Grab, Gojek and more have become the bulk of trips taken in Singapore. In January 2024, there were 74,000 street-hail trips on average and 527,000 ride-hail trips made through apps or calls [1] and 'the number of private-hire cars, which includes self-drive rental and ride-hailing, has jumped by more than fourfold from 18,847 to 84,413 in the same period' [2]. It has also been claimed that 'Many consumers today choose booking a car on ride-hailing platforms over taxis as they prefer the certainty of knowing the fare in advance' [2] and that 'The market has moved towards dynamic pricing and the convenience of comparing and booking cars using apps' [3].

From the consumers' point of view, the introduction of ride-hailing services has brought benefits to their lives, which could be the reason behind increasing demands for ride-hailing services and in turn driving this major shift in the market.

With an increasing demand for ride-hailing services, there must also be an increase in the supply to keep the balance. To increase the supply, drivers would need to take up more rides or there needs to be an increase in the number of drivers. This project seeks to seeks to incentivize drivers in order to encourage engagement from them.

2. Scope

This study aims to improve the efficiency and in turn overall earnings of ride-hailing drivers, to encourage the participation of more drivers. This study will focus on ride-hailing trips located within Singapore. The project will be conducted over a period of 9 months and will include the designing of algorithms as well as implementation of existing machine learning methods. Tests and simulations will be conducted to evaluate the system's performance.

3. Research and literature review

3.1. Geospatial Temporal Data

Extensive research has been conducted in relation to geospatial taxi data, route recommendation algorithms as well as pricing systems of ride-hailing platforms. Geospatial data related aspects including graph representation and predictions of demands were also explored as part of this literature review.

Studies have investigated the route recommendation for drivers and the various metrics taken into consideration, such as supply and demand [4], place of interests (POIs) and road conditions. The 'HUNTS Trajectory Recommendation System' expanded on this by considering points of interests (POIs) and scoring routes derived by solving maximum vector problem [5].Forecasting of supply and demand can be implemented by various methodologies ranging from simple machine learning models such as Light Gradient Boosting Machine (LGBM) regression [6,7] to deep learning approaches making use of Spatial-Temporal Tree Convolution Models (STTCM) [8].

3.2. Ride-hailing Earnings

Most ride-hailing platforms such as Grab, Uber and Gojek make use of dynamic pricing to manage trip fares based on real-time road conditions [9,10,11] This is done by implementing surge charges on top of the base fare, which may increase the final fare of the trip, based on some conditions. By managing the fares based on real-time conditions, it aims to ensure that there are rides available for users in any conditions. For example, when there is high demand and low supply, applying a surge charge will increase the fare and as such the earnings of the driver. This would then incentivise drivers to turn on their app, which contributes to the supply rebalances the supply and demand [9]. Another simpler method, which TADA has adopted, is to simply impose surcharges at fixed times, such as during morning and evening peak periods [12].

Key Components

Different platforms consider various factors that are unique to their own pricing algorithm [13,14,15,16,17]. Though there may be slight differences across the different platforms, they all have a few key components which are the main factors in calculating the trip costs:

The trip fare includes:

- Base Fare
- Distance Rate Fare
- Duration Rate Fare
- Booking/Platform Fee

Dynamic surge pricing:

- Surge Multiplier

Dynamic Pricing

With ride-hailing, 'dynamic pricing has become a crucial strategy' and we can implement Q-learning to optimize the surge pricing [18]. By making use of Q-learning, we can train the model to determine the best surge prices to implement depending on the road conditions to help balance out the supply and demand for trips.

Commission Rate

On top of the trip fare and surge pricing, there is a service/commission/operation fee for each trip that the driver must pay. This fee is used to cover operational costs by the company as well as taxes. The commission rate may vary across different companies.

It has been found that Uber Ghana charges driver-partners a fixed service fee/commission rate of 20% [19] and Gojek Singapore charges a fixed 10% [20]. Grab Singapore used to charge a fixed 20.18% but have restructured to charge a variable fee instead of a fixed fee [21]. Tada claims that it is the only platform that has 0% commission for drivers [22].

4. Pilot Study

Readings and code exploration regarding geospatial temporal data have been performed before the start of the project to facilitate familiarization with the data and scope. Readings included local and overseas online papers regarding relevant topics including spatial-temporal data, trajectory and route predictions and taxi data. Reference was also made to the work of a URECA student who worked on a similar topic regarding the demand and supply of taxis in Singapore [6,7].

4.1. Code Exploration

There were two main codes used in the taxi analysis paper [6,7]. The first is to extract a days' worth of data from a monthly dataset, and the second is a prediction model which predicts the gap of different locations in Singapore where gap refers to the difference in supply and demand of drivers.

Minor changes were made to improve the original prediction model used in the taxi analysis paper [6,7]. The training data was sorted by longitude, latitude then time. In the original model, lag features were created by shifting the entire dataset, causing some lag features to be shifted from a different longitude-latitude pair. To maintain structure, shifts were only performed within the same location (longitude-latitude pair) in the new model to ensure that values from different locations will not overlap.

Below are the results for both the original and new models for comparison:

```
MAE in training set: 3.7947207128312637
MAE in test set: 14.560159417818372
```

Figure 1: Original model results

```
Model 2
MAE in training set: 7.615297776195015
MAE in test set: 8.965965609156058
```

Figure 2: New model results

4.2. API

- **OSRM** route and table services [23] were used to calculate routes, distances and duration of trips which were used in the algorithm to select trips.
- **Onemap** [24] was used to obtain the Singapore map and neighbourhood planning areas in Singapore along with its shape and coordinates as multipolygon objects (refer to Appendix A).

4.3. Libraries

Documentations of libraries such as Geopandas, scikit-mobility, MovingPandas, Shapely.Geometry, and Folium, which deal with geospatial data, were read to gain understanding of their functionalities and uses. The use of various APIs they provided were also explored. These libraries were used to store coordinates, route linestrings as well as plot maps for visual analysis.

4.4. Initial algorithm

After there was sufficient knowledge of the relevant data and tools, an algorithm was designed to recommend an optimal trip to take up following each drop off. A separate algorithm is then used to iteratively perform the trip recommendation within a selected time frame to simulate a working day. The total earnings of the simulated working day are then summed up and compared to historical data as an evaluation.

5. Data Processing

The historical data provided will be used as inputs to both the recommendation system as well as the gap prediction model discussed in Section 6. Data processing would have to be performed to clean and restructure the data to be fit for use.

5.1. Dataset Structuring

The dataset provided by LTA is quite large, consisting of months' worth of data, with each month's worth being approximately 4.6 GB. In this project, only a days' worth of data will be used, which requires extraction from the provided dataset. The extracted dataset will be provided with the help of a fellow FYP student who is handling multiprocessing, Ho Guo Liang Ken.

The structure of the dataset is as shown below in Table 1.

A	B	C	D	E	F	G	H	I	J	K	L
	VEHICLE_ID	DRIVER_ID	REQUEST_ID	DT_START	DT_END	LATITUDE_PICKUP	LONGITUDE_PICKUP	LATITUDE_END	LONGITUDE_END	distance_km	duration_s

Table 1: Extracted dataset structure

	DT_START	start_time_window	DT_END	LATITUDE_PICKUP	LONGITUDE_PICKUP	LATITUDE_END	LONGITUDE_END	VEHICLE_ID	distance_km	duration_s	start_week	start_dayofweek	start_hour	start_minute
141228	2022-09-01 07:57:08	15.0	2022-09-01 08:07:24	1.28	103.83	1.29	103.85	0	1.960442	616.0	35	3	7	57

Table 2: Simulation dataset

6. Gap Prediction Model

The gap prediction model used will be adapted from the prediction model used in the taxi analysis paper [6,7] with minor changes to suit this project's use case.

The aim of this model is to predict a general trend of gaps at certain timings and locations, by performing time-series predictions using a Light Gradient-Boosting Machine (LGBM) regression. The model will take in a days' worth data, provided by Ken after extraction, as input. The dataset will then be cleaned, and lag features will be created to prepare the data for time-series predictions.

The output will be a Pandas dataframe which contains the grid coordinates, time features and predicted gap values as seen in Table 3. The predicted gaps dataframe will then be read as inputs for the recommendation system.

index	week	dayofweek	time_window	LONGITUDE	LATITUDE	gap	gap_lag_1	gap_lag_2	gap_lag_3	gap_lag_4	gap_lag_5	gap_lag_6	gap_pred	frame
0	35	3	32.0	103.61	1.24	-1.0	-1.0	-1.0	-3.0	-1.0	-1.0	-1.0	-0.605281	35-3-32.0
1	35	3	35.0	103.61	1.24	-2.0	-1.0	-1.0	-1.0	-3.0	-1.0	-1.0	-0.605281	35-3-35.0
2	35	3	37.0	103.61	1.24	-1.0	-2.0	-1.0	-1.0	-1.0	-3.0	-1.0	-1.526947	35-3-37.0
3	35	3	41.0	103.61	1.24	-1.0	-1.0	-2.0	-1.0	-1.0	-1.0	-3.0	-0.787556	35-3-41.0
4	35	3	46.0	103.61	1.24	-1.0	-1.0	-1.0	-2.0	-1.0	-1.0	-1.0	-0.472593	35-3-46.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
16246	35	3	43.0	104.02	1.32	-9.0	-2.0	-3.0	-1.0	-2.0	1.0	2.0	-1.512314	35-3-43.0
16247	35	3	44.0	104.02	1.32	-4.0	-9.0	-2.0	-3.0	-1.0	-2.0	1.0	-5.071955	35-3-44.0
16248	35	3	45.0	104.02	1.32	-5.0	-4.0	-9.0	-2.0	-3.0	-1.0	-2.0	-5.897678	35-3-45.0
16249	35	3	46.0	104.02	1.32	-3.0	-5.0	-4.0	-9.0	-2.0	-3.0	-1.0	-4.618749	35-3-46.0

Table 3: Predicted gaps dataset

7. System Overview

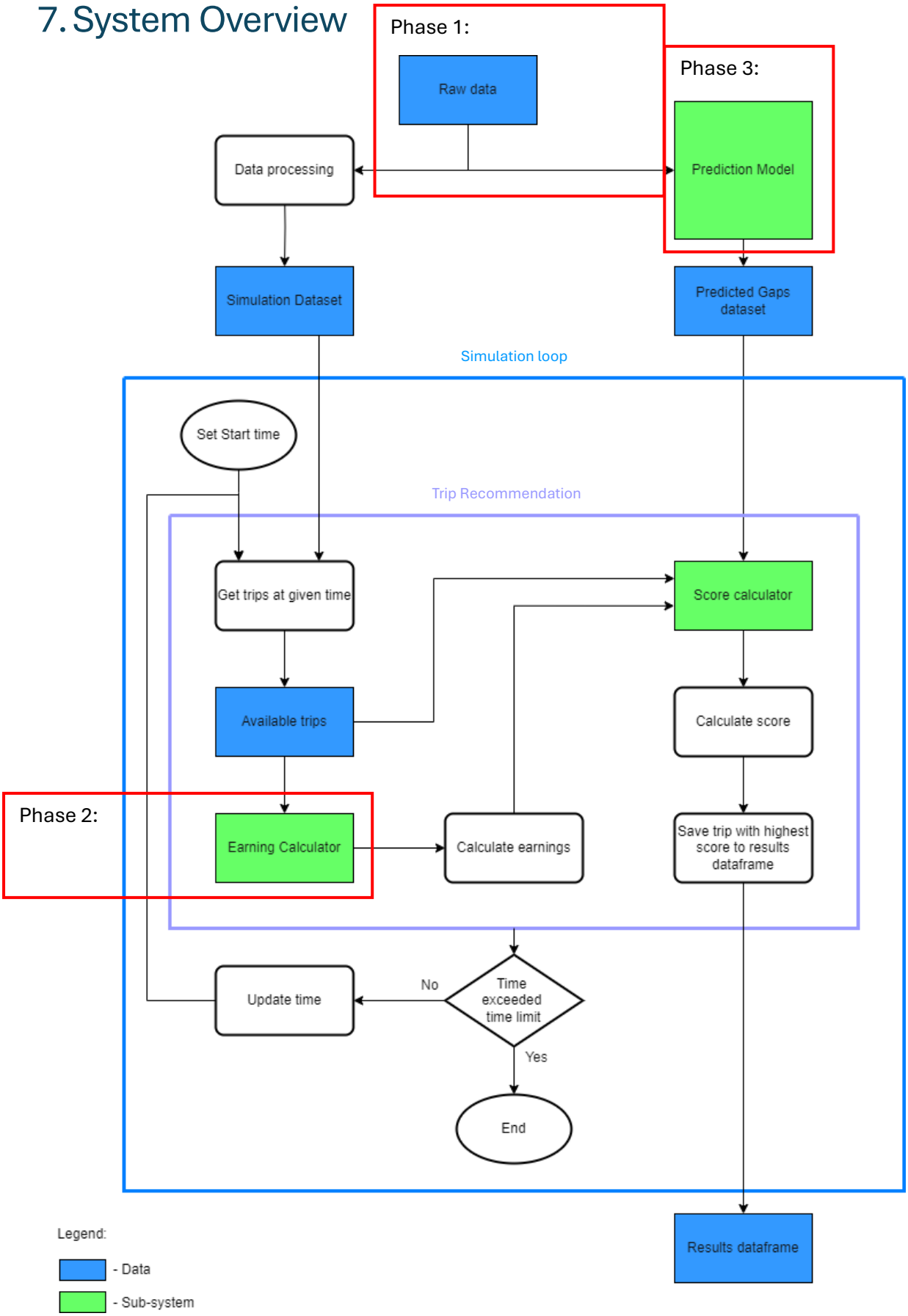


Figure 3: System Design Flowchart

The recommendation system makes use of an algorithm which selects the best available trip, which is then looped from the start to end time to simulate a driver's typical working day. Historical data is used to predict gaps (refer to Table 3), which is then taken as inputs along with a simulation dataset (refer to Table 2) that has also been processed from the same historical dataset. The output of the system is a dataframe which describes the information of all the recommended trips taken, such as the timing, distance, earnings, etc.

Once the base system is working and evaluated, various components of the systems will subsequently be modified to improve it and simulate real-life situations. The modifications aim to improve the system by individually enhancing each component of the system, which will be carried out in different phases. Figure 3 shows the system overview and various components which can be enhanced.

7.1. Trip Selection

The algorithm gets all available trips within 5 minutes of the drop off timing and selects an optimal trip based on the trip score defined below, where 'norm_gap' and 'norm_earnings' are the normalised values of the predicted gap and earnings.

$$Score = (0.7 * norm\_gap) + (0.3 * norm\_earnings)$$

Equation 1: Trip score calculation equation

Predicted gaps or gaps is defined as demand minus supply as per the taxi analysis paper [6,7] and are obtained through the gap prediction model mentioned in Section 6. Larger gaps imply a higher demand and a lower supply.

With a slightly heavier weightage assigned to gap values, trips which lead to locations with high gap would be scored better and more likely be selected. The higher demand increases the probability of a driver taking on a subsequent trip, possibly reducing driver's idle time spent waiting for passengers or driving around to look for passengers. This prioritises the probability of future trips over the immediate earning rewards per trip. This aims to ensure a constant availability of trips for drivers to improve efficiency.

The earnings of each trip is calculated using a naïve distance-based calculation and fares provided by Singapore's official tariff ordinance, referenced from the taxi analysis paper [6,7]. The earning equation is as shown:

$$Earnings = 4.50 + distance * 0.78$$

Equation 2: Earning calculation from 0000-0559

$$Earnings = 4.50 + distance * 0.7$$

Equation 3: Earning calculation from 0600-2359

The routes for selected trips will be calculated using OSRMs' API, which produces the same routes as Google Maps, and saved in the resultant dataframe.

8. Dynamic Earning Calculator

As this project focuses on ride-hailing trips, the simple earning equation used for street-hailing is likely unable to accurately represent earnings from ride-hailing. This highlights the need for a more realistic calculation to represent ride-hailing trip costs and earnings.

8.1. System Design

The earning calculator consists of two main components, the trip fare calculation and the surge multiplier Q table. A driver's earning will be calculated by implementing an equation that combines these two components shown below:

$$Earning = (Trip\_fare * Surge\_multiplier) * (1 - Commission\_rate)$$

where

$$Trip\_fare = Base\_fare + Distance\_fare + Duration\_fare + Booking\_fee$$

$$Distance\_fare = Distance * Distance\_rate$$

$$Duration\_fare = Duration * Duration\_rate$$

Equation 4: Driver earnings calculation

A pessimistic approach was taken for this project which assumes the worst case for the drivers and diminished their outcomes. As such, a high commission rate of 20% is assumed, and drivers only earn 80% of the trip cost.

8.1.1. Trip Fare

The trip fare will be the sum of a fixed base fare, the distance-based fare (per km rate), the duration-based fare (per min rate) and a fixed booking fee. Since most ride hailing companies such as Grab, Uber and Gojek are private, they have not released the specific numbers which they use. As such LTA fares will be referenced [25]. However, some components such as the duration-based fare are not provided by LTA, which requires some trials to obtain.

8.1.2. Surge Multiplier

The surge multiplier component relies on Q-learning to get a resultant Q table that determines which multiplier to apply under different conditions.

8.1.2.1. Introduction to Q-learning

Q-learning is a reinforcement machine learning algorithm that consists of an agent (model) and an environment, and interaction between these two can be described by the Markov Decision Process (MDP) [26]. The goal of this algorithm is for the agent (model) to learn the best action to take in each state, to maximise its total rewards over time. The key components to Q-learning are as follows [27]:

- State of the agent and environment, **s**
- Possible actions that can be taken in the current state, **a**
- Reward after taking an action in a particular state, **r**
- Score value of each state-action pair, **Q(s, a)**

The algorithm uses an epsilon-greedy (exploration-exploitation) approach to determine an action given a state, where epsilon refers to the probability of selecting an exploration action. Exploration actions are selected at random, which allows the agent to increase its knowledge, while exploitation actions are chosen using a greedy approach based on the current Q-values to maximise rewards [28].

The Q-table is updated using the Bellman Equation [29], where α is the learning rate and γ is the discount factor:

$$q^{new}(s,a) = (1-\alpha)\underbrace{q(s,a)}_{\text{old value}} + \alpha\overbrace{\left(R_{t+1} + \gamma \max_{a'} q(s',a')\right)}^{\text{learned value}}$$

Equation 5: Q-learning update iteration

The Q-table will be updated iteratively for however many steps defined per episode, and it will repeat the same process for however many episodes are defined.

The resultant Q table can be represented by a 2D matrix, where the rows represent the states, and the columns represent the actions. By simply selecting the column with the highest value for each row, the best action (column) in the state (row) can be obtained.

8.1.2.2. Project Use Case

In this use case, the states are the conditions which would affect the surge pricing, such as the supply and demand (gap), the timing of trips (peak or not) and potentially some other factors. The actions are the surge multiplier values to be applied.

After training the model and obtaining the Q-table, an optimal surge multiplier value (action) can be obtained given the trip conditions (state).

8.2. Methodology

Most of the components used in the calculation will be referenced from LTA [25]. LTA provides the cost for flag-down fares which will be used as the fixed base fare, distance-based fare rate, booking fees, and peak period timings. They also provide location surcharges which provides insights on 'hot' locations to look out for, which could be used as improvements in the system in the future. However, the time-based fare rate provided by LTA is based on waiting time instead of the trip duration which is needed, making the time-based fare an unknown variable in calculating the trip fares.

To determine the time-based fare rate, samples will be collected from the Grab application and reverse engineering analysis will be performed to obtain a reasonable rate that will be used in the calculation of driver earnings. From the samples collected, the following parameters would be recorded and used:

- End location
- Trip cost

The time-based fare rate would then be calculated using the following equation:

$$Duration\_rate = [(Trip\_cost/Surge\_multiplier) - Base\_fare - Booking\_fee - Distance\_based\_fare]/Duration$$

Equation 4: Time-based fare rate calculation

8.3. Implementation

8.3.1. Reverse Engineering to Obtain Duration Fare Rate

50 data samples were collected by inputting random locations into the Grab application and searching for rides. Data samples were collected within a period from 5:15pm to 5:45pm on a Wednesday, and all have the same starting location. For each sample, the location was inputted into Google Maps to obtain its coordinates. The location name, cost of trip, and coordinates of location were then logged down onto a Word document.

69E Lor Melayu, Singapore 416968 - 17.4 - (103.90955, 1.32384)

495F Tampines Street 43, Singapore 525495 - 20.1 - (103.95279, 1.36450)

72 Lor 5 Toa Payoh, #01-576, Singapore 310072 - 11.7 - (103.85207, 1.33383)

262 Boon Lay Dr, Singapore 640262 - 24.2 - (103.70748, 1.34491)

20 Choa Chu Kang Street 62, Singapore 689143 - 23 - (103.74612, 1.40007)

255 Dairy Farm Rd, Singapore 679056 - 21.2 - (103.77623, 1.36543)

14 Tuas Ave 1, Singapore 639499 - 30.20 - (103.66154, 1.32245)

Figure 4: Sample of data collected

An assumption was made throughout the reverse engineering analysis. According to LTA standards [25], the data samples were collected during peak periods. However, due to personal experience and the assumption that most people end work at 6pm, the demands would be low. And as such it would be assumed that the data samples were collected off-peak, and no surge will be applied (surge multiplier value 1).

With the data samples collected, the duration-based fare rates would be obtained following Equation 6. Several values used in the equation will be referenced as per LTA standards [25] as follows:

Type of Fare	Fee ($)
Base Fare	4.40
Booking Fee	2.30
Distance-based Fare Rate	- 0.26 every 400m (1-10km) - 0.26 every 350m (above 10km)

Table 4: LTA referenced values

For each data sample, the trip's duration and distance was calculated using OSRM's API. The distance-based fare can then be calculated by applying the above rates with the calculated distance. The duration-based fare rate can then be derived using Equation 6. All values are saved onto a Pandas dataframe, with each row representing a data sample, excluding the first row which represents the starting location for all trip samples collected.

	End_Location	Coords	Distance (m)	Duration (min)	Total_Fare ($)	Base_Fare ($)	Booking_Fare ($)	Distance_Fare ($)	Duration_Fare ($)	Duration_Rate ($/min)
0	59 Ang Mo Kio Ave 8, Singapore 567752	(103.84868, 1.37052)	0.0	0.000000	0.0	0.0	0.0	0.00	0.000000e+00	0.000000e+00
1	69E Lor Melayu, Singapore 416968	(103.90955, 1.32384)	13112.1	16.480000	17.4	4.4	2.3	8.84	1.860000e+00	1.128641e-01
2	495F Tampines Street 43, Singapore 525495	(103.95279, 1.3645)	18630.3	22.140000	20.1	4.4	2.3	13.00	4.000000e-01	1.806685e-02
3	72 Lor 5 Toa Payoh, #01-576, Singapore 310072	(103.85207, 1.33383)	7277.1	11.490000	11.7	4.4	2.3	4.94	6.000000e-02	5.221932e-03
4	262 Boon Lay Dr, Singapore 640262	(103.70748, 1.34491)	23057.0	26.558333	24.2	4.4	2.3	16.38	1.120000e+00	4.217132e-02
5	20 Choa Chu Kang Street 62, Singapore 689143	(103.74612, 1.40007)	17442.0	21.703333	23.0	4.4	2.3	12.22	4.080000e+00	1.879896e-01
6	255 Dairy Farm Rd, Singapore 679056	(103.77623, 1.36543)	15345.4	20.048333	21.2	4.4	2.3	10.66	3.840000e+00	1.915371e-01
7	14 Tuas Ave 1, Singapore 639499	(103.66154, 1.32245)	31364.4	33.090000	30.2	4.4	2.3	22.62	8.800000e-01	2.659414e-02
8	3 Yung Sheng Rd, Singapore 618499	(103.721154, 1.3347)	22348.0	25.540000	27.2	4.4	2.3	15.86	4.640000e+00	1.816758e-01
9	365b Sembawang Cres, Singapore 751365	(103.81741, 1.44396)	11066.3	15.126667	19.7	4.4	2.3	7.54	5.460000e+00	3.609520e-01

Table 5: Breakdown of trip costs

After obtaining the rates for all 50 samples, the Shapiro-Wilk normality test was performed to determine if the are normally distributed. The p-value obtained was 0.001125, and is lower than the alpha value of 0.05, which means that the null hypothesis that the distribution is normal and can be rejected, suggesting that the data is not normally distributed [30]. It can also be observed from the box and whisker plot in Figure 5 that the data is skewed.

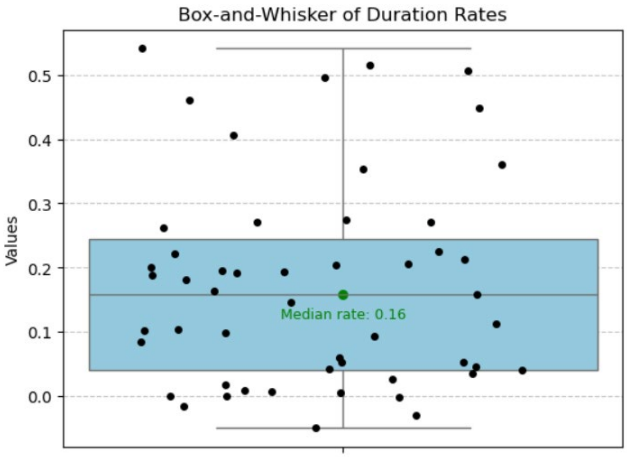


Figure 5: Box and Whisker of values

As such, the median value of 0.16 was used as it is works better with skewed datasets as compared to mean. 10 more samples were collected to evaluate the derived value. The samples were collected within the same period, however on a different day of the week. Using the value of $0.16 per minute as the duration-based fare rates, the total fare for each trip was calculated and compared against the samples collected from Grab. The difference in fares were then plotted as a histogram in Figures 6.1 and 6.2.

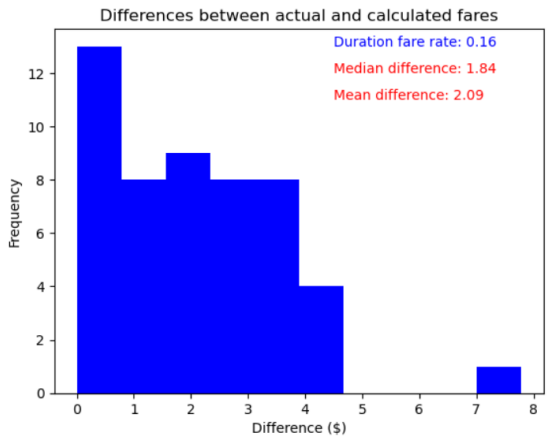


Figure 6.1: Differences in first 50 samples

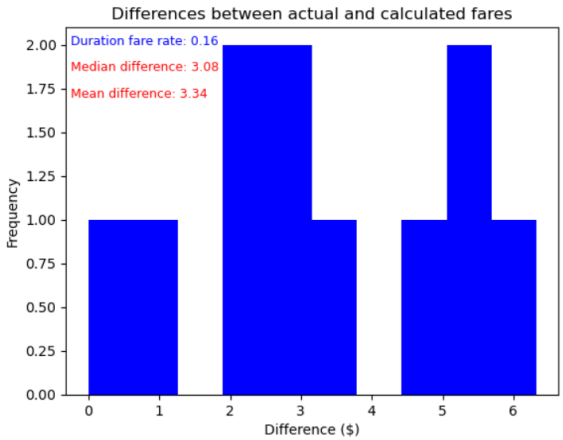


Figure 6.2: Differences in extra 10 samples

In the initial analysis, most of the samples had differences ranging from $0 to $4, with a median value of $1.84 and an outlier of around $7. The evaluation samples had larger differences overall as well as the median which increased to $3.08. This could be due to the collection of data samples on a different day or the sample size being too small.

8.3.2. Dynamic Surge Multiplier

8.3.2.1. Markov Decision Process (MDP) Formulation

The Markov Decision Process (MDP) is a mathematical framework used in reinforcement learning, which models decision-making processes using an agent based on the following components: the set of all possible states (S), the set of all possible actions (A), the transition function (T) and the reward function (R) [31,32].

States

The set of states (environment) will be defined by a 2-dimensional combination of gaps and timing respectively to represent the trip conditions.

Based on the dataset being used, predicted gap ranges from -829 to 494, however only positive gap values will be used since negative gaps imply a higher supply than demand. There is then no need for a surge multiplier, which can be represented by a gap value of 0. The new range of predicted gaps, ranging from 0 to 494, are then split into 10 bins and the median value of each bin will be used. Min-max normalization will then be applied so that it can be used to calculate the rewards. Timing refers to if a specific timestamp is within peak hours or not, and there are only 2 possible values, non-peak and peak, which will be represented by 0 and 1 respectively.

```
(0.0, 0), (0.0, 1), (0.119, 0), (0.119, 1), (0.225, 0), (0.225, 1)
```

Figure 7: Sample of set of possible states

Actions

The set of actions will be defined by a list of values which represent the possible surge multiplier values to implement.

There are 11 possible actions with values ranging from 1 to 2, with an increment of 0.1 between each value. 1 will be the minimum value which represents the cost of a normal trip without surge, and 2 will be the maximum value which represents the maximum surge which will be 2 times the cost of a normal trip. Min-max normalization will also be applied so that it can be used to calculate the rewards.

```
[0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
```

Figure 8: Set of possible actions

Transition Function

The transition rules determine the next state of the agent given a state and an action, and have been defined as follows (refer to Appendix B):

1) If there is no surge (value 0.0) selected as action, the state has a 50% probability of remaining in the same state and a 50% probability of moving to a state with higher gap. This is because without surge pricing, there is no incentives for drivers and as such the gap would remain the same or increase due to higher demand and lower supply.

2) Similarly, if there is surge (value 0.1 to 1.0) selected as action, the state has as 50% probability of remaining in the same state and a 50% probability of moving to a state which has lower gap. This is because surge pricing would incentivize drivers to be active which increases the supply and in turn lowers the gap or the gap could remain the same if demands increase as well.

Rewards

The rewards for each state-action pair are calculated using the normalized gap and action values previously derived, along with the timing (0 or 1) through Equation 7. The timing values are divided by 10 to reduce its significance as it is only a small factor, while the equation focuses strongly on the normalized gap values which are the most important.

$$rewards = \frac{1}{1 + \left| (norm\_gap + \frac{timing}{10}) - norm\_action \right|}$$

Equation 5: Reward calculation equation

The purpose of dynamic pricing and surge multipliers is to balance the demand and supply by increasing trip cost to incentivize drivers to be active, which increases the supply to balance out the high demands. Similarly, when demands are low, trips would

be cheaper which decreases the supply since there are no incentives. As such the equation aims to highly reward state-action pairs that have similar value magnitudes, for example low gap states and low value actions.

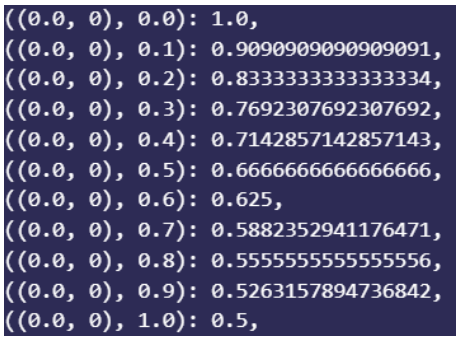


Figure 9.1: Sample of rewards with low gap state

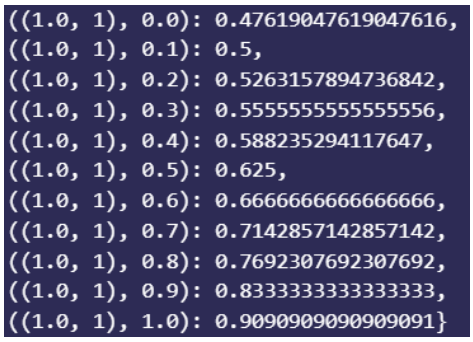


Figure 9.2: Sample of rewards with high gap state

As seen in figure 9.1, when there is low gap, the reward for actions with smaller magnitudes are greater, and the rewards get smaller with for actions with increasing magnitudes as they get further from the low gap magnitude. Similarly in figure 9.2, when there is high gap, the reward for actions with smaller magnitudes are lower, and the rewards get larger for actions with larger magnitudes.

8.3.2.2. Q-learning Training

Bellman's equation was used as the policy to train the agent, and decaying epsilon algorithm was used. In decaying epsilon, the epsilon value decreases over time. As the number of training episodes increases, the probability of exploration actions would then decrease, and the probability of exploitation actions would increase. An initial value of 1 and a minimum value of 0.01 was used in the decaying epsilon algorithm. The training consists of 5000 episodes and 5000 steps per episode, where each step would determine and action and obtain the rewards for the state-action pair. A learning rate of 0.01 and a discount factor of 0.9 was used.

8.3.2.3. Evaluation

The model was evaluated by checking for rewards convergence at the end of the training process. With a maximum reward of 1 per step and 5000 steps per episode, the maximum reward per episode is 5000.

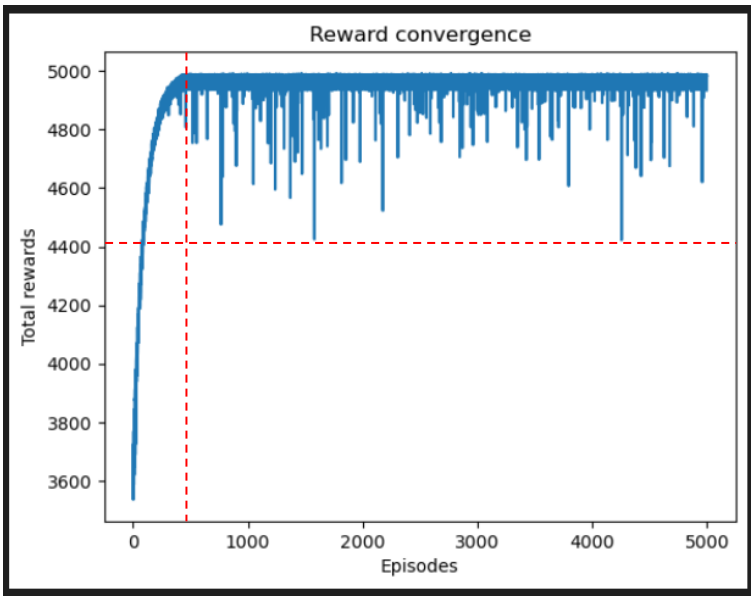


Figure 10: Total rewards convergence

As seen in figure 10, the rewards converge towards the maximum reward of 5000 after approximately 500 episodes. However, there are still some fluctuations in several episodes, which could be due to the epsilon greedy algorithm whereby an exploration action could have been taken which would not maximise the rewards. Even with the fluctuations, the rewards have a value of at least 4400.

Since the rewards still converge overall, it shows that the agent is learning well and that it is statistically sound.

9. Simulation and Evaluation

Simulation

The trip selection process is repeated from the start time of 7am to the end time of 7pm, with a 1.5-hour break from 11:30am to 1pm, according to the algorithm in Figure 11. In the simulation, the starting location is selected based on the location with the highest

gap at the starting time. Results are then saved in a Pandas dataframe with each row representing a trip, except the first row which is the starting point (refer to Appendix C). The overall route is plotted using Folium for a visual representation of the recommended trips (refer to Appendices D and E).

Algorithm 1 Looping Algorithm

```
counter ← 1
Initialise time_limit
while True do
    Get trips at selected_time
    wait_time ← 5
    for i in range(wait_time) do
        new_time = selected_time + i
        Get trips at new_time
        Concatenate trips to trip dataframe
    end for
    Filter trip dataframe for rows with most commonly appeared neighbourhood
    for each trip do
        Calculate distance, duration
        Calculate expected arrival time, earning
        Calculate predicted gap at arrival time
        Append all calculated values to trip dataframe
    end for
    Calculate trip score for all trips in trip dataframe
    Select trip based on best trip score
    if counter == 1 then
        Create new route dataframe with same columns as trip dataframe
    end if
    Concatenate selected trip into route dataframe
    Get linestring object of route for selected trip and append to route dataframe
    Get arrival_time of last row of route dataframe
    if arrival_time ≥ time_limit then
        Break
    end if
```

Figure 11: Pseudocode for recommendation system

Initial Evaluation

The results of the simulation using the naïve distance-based calculator yields a total earning of approximately $394, with 23 trips taken as seen in Figure 12. The results were then compared against top earners in the dataset used, seen in Figure 13.

```
Weight distribution: 0.7 - predicted gap, 0.3 - expected earnings
Total earnings: $ 393.6599999999999
Number of trips:  23
```

Figure 12: Simulation results (naïve calculation)

```
Top performer:
 Total number of trips:  18
 Total distance:  335.63 km
 Total duration:  382 minutes
 Total earnings: $ 315.941
```

Figure 13: Top performer from historical data

The initial algorithm proves to be effective, having the recommendation system performing slightly better than the top performer, by increasing the number of trips within the same time period and overall earning. It achieves the goal of improving a driver's overall performance and earnings.

With the recommendation system evaluated and achieving its proposed goal, enhancements to individual components within the system can be made to further improve the recommendation system.

Final Evaluation

After the dynamic earning calculator was implemented to replace the naïve distance-based calculator, the simulation was run again. Similar to the initial evaluation, the results were recorded and compared against historical data performance. Calculating the earnings for the entire dataset is too time-consuming, as such an assumption was made that more trips indicated more earnings and top performers were decided based on the number of trips.

```
Weight distribution: 0.7 - predicted gap, 0.3 - expected earnings
Total earnings: $ 592.4699999999999
Number of trips:  30
```

Figure 14: Simulation results (dynamic calculation)

VEHICLE_ID	Earning	No_of_trips
12098	359.89	27
23620	339.93	28
1037	310.12	27
30000	309.91	27
3419	304.12	26
19741	302.86	27
22057	298.03	26
2696	293.03	27
4295	283.06	27
40855	263.79	27

Figure 15: Top 10 performers from historical data

As seen in Figure 15, top performers from the dataset have earnings ranging from around $264 to 360, with 26 to 28 trips. The system yields earnings of around $592 as seen in Figure 14, which improves earnings by at least 64% and up to 124%.

Further tests were conducted to verify the system's performance given different conditions such as starting points and timings. Firstly, different starting locations were used given the same time frame from 7am to 7pm with a break from 11:30am to 1pm. From the top performers in Figure 15, the starting location of each vehicle was obtained and used as the starting point for the simulation. The second time frame that was tested starts at 12pm to 12am with a break from 6pm to 7:30pm. Similarly, the starting locations of top performers within the time frame was used as the starting point for the simulation. The results were then recorded along with its performance against the historical top performers.

	VEHICLE_ID	Starting_point	System_earnings	No_of_trips	Historical_earnings	Percentage_increase
0	12098	(103.84, 1.28)	536.02	29	359.89	48.94
1	23620	(103.75, 1.38)	464.27	26	337.14	37.71
2	1037	(103.84, 1.29)	553.71	29	310.12	78.55
3	30000	(103.86, 1.3)	551.60	30	309.91	77.99
4	3419	(103.84, 1.31)	568.47	31	303.70	87.18
5	19741	(103.75, 1.37)	569.97	29	302.15	88.64
6	22057	(103.77, 1.34)	557.09	29	298.03	86.92
7	2696	(103.78, 1.32)	544.88	27	293.95	85.36
8	4295	(103.91, 1.32)	546.31	30	283.06	93.00
9	40855	(103.72, 1.36)	571.96	31	262.15	118.18

Figure 16: Different start locations from 7am to 7pm

	VEHICLE_ID	Starting_point	System_earnings	No_of_trips	Historical_earnings	Percentage_increase
0	3093	(103.75, 1.35)	640.93	39	396.94	61.47
1	22609	(103.84, 1.34)	635.53	38	328.99	93.18
2	26910	(103.97, 1.37)	668.23	41	312.87	113.58
3	12082	(103.88, 1.33)	668.31	47	307.02	117.68
4	34601	(103.87, 1.32)	602.45	34	287.00	109.91
5	34584	(103.97, 1.36)	661.87	46	283.03	133.85
6	23015	(103.72, 1.35)	642.46	42	282.07	127.77
7	35265	(103.88, 1.35)	643.93	43	258.74	148.87
8	32573	(103.85, 1.27)	663.35	44	252.09	163.14
9	20805	(103.85, 1.32)	608.96	35	242.75	150.86

Figure 17: Different start locations from 12pm to 12am

From Figures 16 and 17, an increase in performance is still noted despite the different starting locations and time frames. These improvements show that the implementation of the dynamic calculator can optimise the recommendation system as it is able to improve the system's decision-making by including more factors for consideration which allows it to make more informed decisions.

10. Discussion

10.1. Limitation

Though results demonstrate improvements in performance, it is only a simple representation that considers the demand and supply gap of drivers and does not account for other factors such as holidays, events or weather which could affect road conditions and as such the surge pricing.

10.2. Future Work

Future work could include the implementation of the taxi data as a graph as it would be a better representation of the 3-dimensional geospatial-temporal data. Graph Convolutional Networks (GCNs) can then be used to implement the gap predictions as they are known to capture relationships and dependencies within graphs, which could improve prediction accuracy as compared to the Light Gradient Boosting model which is currently used on the tabular data representation. The complexity of the recommendation system could also be increased by including a wider range of dates used, which could account for trends over the different days or months. More road conditions could be factored into the implementation of the MDP and Q-learning algorithm to increase its complexity and to better represent the road environments.

11. Conclusion

This project aimed to increase ride-hailing drivers' earnings by designing and improving a recommendation system.

A simple system was initially developed, using a simple algorithm to compare and select an optimal trip after considering the immediate reward of trip earning and future reward of another possible trip. Earning calculations were performed using a naïve distance-based calculations. The system was then improved with the implementation of a dynamic earning calculation to replace the naïve calculation component. The Q-learning algorithm was implemented to enable more adaptive decision-making, which aims to

better represent real-life ride-hailing scenarios where prices can fluctuate based on road conditions. Certain essential values that were not readily available were derived from experiments and tests conducted.

Results demonstrate that the system and enhancements contributed to increased trip efficiency and earnings compared to historical data, which would not only benefit the drivers, but also the private hiring organisations that are receiving commissions.

References

[1] E. Loi and W. William, “A ‘vital niche’, but flagging a cab is getting harder,” The Straits Times, https://www.straitstimes.com/singapore/transport/a-vital-niche-but-flagging-a-cab-is-getting-harder

[2] D. Ng, “Can taxis still compete with ride-hailing? Yes, say industry players,” CNA, https://www.channelnewsasia.com/singapore/taxis-versus-ride-hailing-drivers-surcharge-deep-dive-podcast-4490136#:~:text=Many%20consumers%20today%20choose%20booking,compare%20prices%20between%20different%20apps

[3] C. Tan, “Street-hail rides dwindle as Singapore’s taxi population continues to shrink,” The Straits Times, https://www.straitstimes.com/singapore/transport/street-hail-rides-dwindle-as-singapore-s-taxi-population-continues-to-shrink

[4] D. Shao, W. Wu, S. Xiang and Y. Lu, "Estimating Taxi Demand-Supply Level Using Taxi Trajectory Data Stream," 2015 IEEE International Conference on Data Mining Workshop (ICDMW), Atlantic City, NJ, USA, 2015, pp. 407-413, doi: 10.1109/ICDMW.2015.250.

[5] Y. Ding, S. Liu, J. Pu, and L. Ni, “HUNTS: A Trajectory Recommendation System for Effective and Efficient Hunting of Taxi Passengers,” SMU, https://ink.library.smu.edu.sg/sis_research/3472/

[6] W. H. Ling, “Taxi Demand and Supply in Singapore,” URECA NTU, https://ethzchsec.sharepoint.com/:w:/r/sites/URECA2023-P2Pdata/_layouts/15/Doc2.aspx?action=edit&sourcedoc=%7B0f485000-22d2-4fa7-b8c1-41d487baeb49%7D&wdOrigin=TEAMS-MAGLEV.teamsSdk_ns.rwc&wdExp=TEAMS-TREATMENT&wdhostclicktime=1730368221910&web=1

[7] EnergizeStatistics, “Taxi-demand-supply-gap-prediction”, GitHub, energizestatistics/taxi-demand-supply-gap-predictionhttps://github.com/EnergizeStatistics/taxi-demand-supply-gap-prediction/blob/master/preprocessing.ipynb

[8] L. Jianbo, L. Zhiqiang, M. Zhaobin, W. Xiaotong, and X. Zhihao, “Optimization of spatial-temporal graph: A taxi demand forecasting model based on spatial-temporal tree,” Science Direct, https://www.sciencedirect.com/science/article/pii/S1566253523004943

[9] Grab, “Why fares go up when it rains,” Grab, https://www.grab.com/sg/inside-grab/stories/surge-dynamic-pricing-explained

[10] Uber, “How Uber’s dynamic pricing model works,” Uber, https://www.uber.com/en-GH/blog/uber-dynamic-pricing

[11] Gojek, “What is dynamic pricing?,” Gojek, https://www.gojek.com/sg/help/driver/fares-and-charges/what-is-dynamic-pricing

[12] Z. Abdullah, “New ride-hailing firm Tada enters market, says it will not impose dynamic pricing,” The Straits Times, https://www.straitstimes.com/singapore/transport/new-ride-hailing-firm-tada-enters-market-says-it-will-not-impose-dynamic-pricing

[13] P. Abedinpour, “How Uber Calculates Your Ride Fare: The Inside Scoop on Their Pricing Strategy!,” Medium, https://medium.com/@peymaan.abedinpour/how-uber-calculates-your-ride-fare-the-inside-scoop-on-their-pricing-strategy-bee21e050a2f

[14] Grab, “Driver’s guide to earnings,” Grab, https://www.grab.com/sg/driver/drive/earnings/

[15] Gojek, “How much does a GoCar trip cost?,” Gojek, https://www.gojek.com/sg/help/customer/gocar/how-much-does-a-gocar-trip-cost

[16] Uber, “How much does a ride with Uber cost?,” Uber, https://www.uber.com/global/en/price-estimate/

[17] tolga özdemir, “The Dynamics of Surge Pricing: A Deep Dive for Product Managers,” Medium, https://medium.com/agileinsider/the-dynamics-of-surge-pricing-a-deep-dive-for-product-managers-a326f017b983

[18] A. Badawy, “Harnessing the Power of Dynamic Pricing in Ride-Hailing Apps Using Reinforcement Learning,” Medium, https://medium.com/@alibadawy96/harnessing-the-power-of-dynamic-pricing-in-ride-hailing-apps-using-reinforcement-learning-4e6952b7d98

[19] Uber, “Tracking your earnings in Ghana,” Uber, https://www.uber.com/gh/en/drive/basics/tracking-yor-earnings/

[20] Gojek, “Gojek driver incentives: Only 10% service fees and upsized incentives,” Gojek, https://www.gojek.com/sg/blog/dp-sg-gojek-driver-looking-ahead-incentives-updates-2023

[21] Y. Kok, “Grab to introduce variable commission rate, says move will make driver compensation fairer,” The Straits Times, https://www.straitstimes.com/singapore/transport/grab-to-introduce-variable-commission-rate-says-move-will-make-driver-compensation-fairer#:~:text=With%20the%20new%20driver%20compensation,%2Dup%20point%2C%20Grab%20said

[22] Tada, “TADA for DRIVER,” Tada, https://tada.global/app/driver

[23] "OSRM API v5.5.1 Documentation," Project OSRM, https://project-osrm.org/docs/v5.5.1/api/#route-service

[24] “Onemap API Documentation”, Onemap, https://www.onemap.gov.sg/apidocs/maps

[25] LTA, “Taxi Fares & Payment Methods,” Land Transport Authority, https://www.lta.gov.sg/content/ltagov/en/getting_around/taxis_private_hire_cars/taxi_fares_payment_methods.html

[26] J. Nuer, “Q-Learning - An introduction,” Medium, https://medium.com/@jereminuerofficial/q-learning-an-introduction-f1392738bcf

[27] A. H. Nababa, “Understanding Q-Learning in Reinforcement Learning,” Medium, https://medium.com/@alaminhnab4/understanding-q-learning-in-reinforcement-learning-3b0e10223ae5

[28] “Epsilon-Greedy Algorithm in Reinforcement Learning,” geeksforgeeks, https://www.geeksforgeeks.org/epsilon-greedy-algorithm-in-reinforcement-learning/

[29] A. Amine, “Q-Learning Algorithm: From Explanation to Implementation,” Medium, https://towardsdatascience.com/q-learning-algorithm-from-explanation-to-implementation-cdbeda2ea187

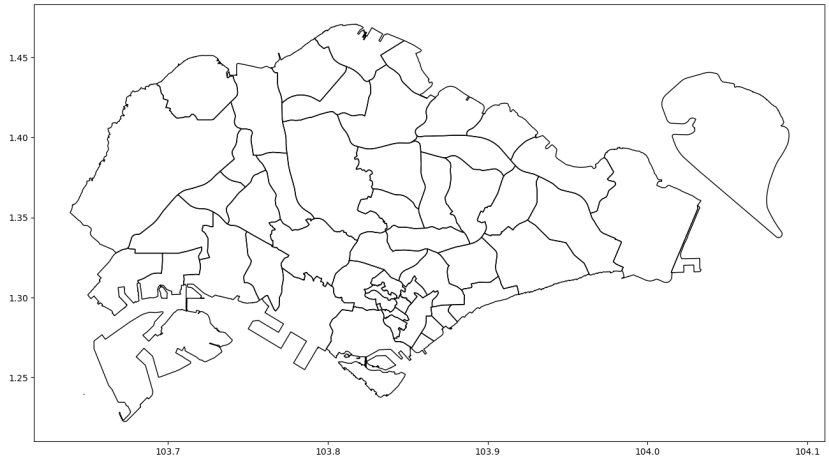
[30] A. Humaizi, “10.0 Shapiro-Wilk test,” Medium, https://medium.com/@maizi5469/10-0-shapiro-wilk-test-5be38fd3c2a6

[31] R. Jagtap, “Understanding the Markov Decision Process (MDP),” builtin, https://builtin.com/machine-learning/markov-decision-process

[32] L. Bonanni, “An Introduction to Markov Decision Processes,” Medium, https://medium.com/@lorenzobonanni/an-introduction-to-markov-decision-processes-835abc4dea56.

Appendix

A: Plot of neighbourhoods that make up the Singapore map



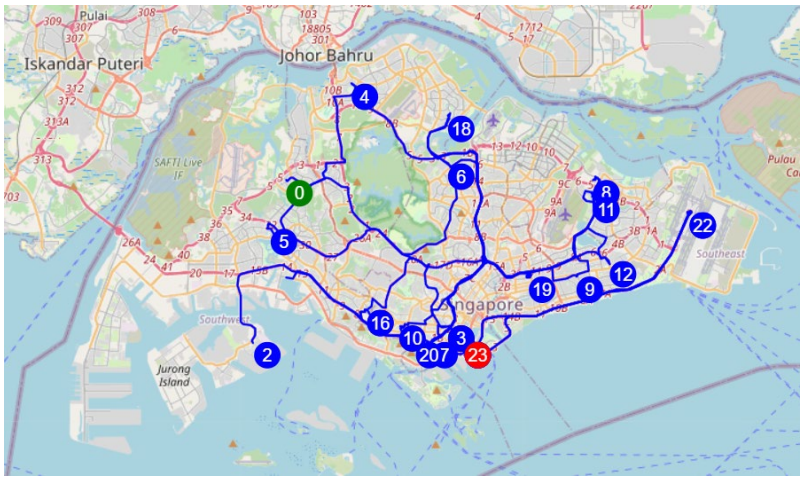
B: Code for transition function

```python
# Define state transition rules
def transition_state(state, action):

    size = len(states)

    # Get index of current state
    for i in range(size):
        if states[i] == state:
            break

    # Minimum gap possible
    if i == 0 or i == 1:
        # No surge -> gap may increase or remain same
        if action == norm_actions[0]:
            next_state = random.choice([states[i], states[i+2]])
        # Surge -> gap can only remain since it's already the lowest
        else:
            next_state = states[i]

    # Maximum gap possible
    elif i == size-1 or i == size-2:
        # No surge -> gap can only remain since it's already the highest
        if action == norm_actions[0]:
            next_state = states[i]
        # Surge -> gap may decrease or remain same
        else:
            next_state = random.choice([states[i-2], states[i]])

    # All other gaps
    else:
        # No Surge -> gap may increase or remain same
        if action == norm_actions[0]:
            next_state = random.choice([states[i], states[i+2]])
        # Surge -> gap may decrease or remain same
        else:
            next_state = random.choice([states[i-2], states[i]])

    return next_state
```

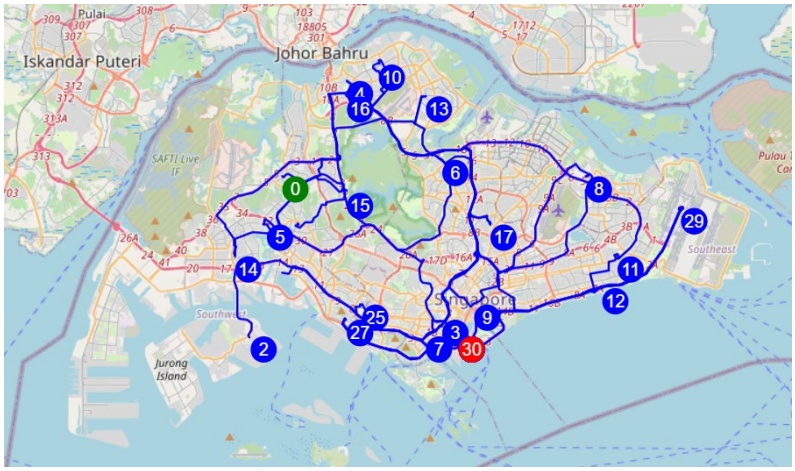
C: Sample of output dataframe containing recommended trips from simulation

	PLN_AREA_END	coordinates	distance	duration	current_timestamp	current_time_window	expected_timestamp	expected_time_window	expected_earnings	predicted_gap	norm_expected_earnings	norm_predicted_gap	score	route
0	PUNGGOL	(103.91401, 1.404536)	0.0	0	2022-01-09 07:00:00	14.0	2022-01-09 07:00:00	14.0	-1000.0	-1000.0	0.0	0.0	0.0	NaN
1	BUKIT MERAH	(103.8078637, 1.2902975)	26.17	29	2022-01-09 07:04:00	14.0	2022-01-09 07:33:00	15.0	22.82	51.669172	1.0	1.0	1.0	LINESTRING (103.91403 1.40511, 103.91425 1.405...
2	SENGKANG	(103.8929675, 1.394057)	22.32	25	2022-01-09 07:36:00	15.0	2022-01-09 08:01:00	16.0	20.12	52.39126	1.0	0.996674	0.997672	LINESTRING (103.81025 1.28876, 103.81025 1.288...
3	PUNGGOL	(103.9105, 1.403427)	4.09	7	2022-01-09 08:03:00	16.0	2022-01-09 08:10:00	16.0	7.36	58.87736	0.979598	1.0	0.993879	LINESTRING (103.88879 1.38904, 103.88875 1.388...
4	SINGAPORE RIVER	(103.8358913, 1.2896087)	19.08	23	2022-01-09 08:14:00	16.0	2022-01-09 08:37:00	17.0	17.86	131.922876	0.99519	1.0	0.998557	LINESTRING (103.9145 1.39944, 103.91442 1.3995...
5	ORCHARD	(103.8302632, 1.3049497)	2.49	5	2022-01-09 08:38:00	17.0	2022-01-09 08:43:00	17.0	6.24	69.398162	0.987149	1.0	0.996145	LINESTRING (103.838 1.29076, 103.83788 1.29047...
6	BUKIT MERAH	(103.82255, 1.265484)	5.25	8	2022-01-09 08:45:00	17.0	2022-01-09 08:53:00	17.0	8.18	10.221772	0.988606	1.0	0.996582	LINESTRING (103.83325 1.29901, 103.83325 1.299...
7	SINGAPORE RIVER	(103.8323129, 1.292821)	4.0	6	2022-01-09 08:54:00	17.0	2022-01-09 09:00:00	18.0	7.3	101.366011	0.988314	1.0	0.996494	LINESTRING (103.82389 1.27108, 103.8239 1.2711...
8	TOA PAYOH	(103.859056, 1.3349651)	8.86	14	2022-01-09 09:00:00	18.0	2022-01-09 09:14:00	18.0	10.7	57.037734	0.992186	1.0	0.997656	LINESTRING (103.82894 1.29029, 103.82949 1.290...
9	OUTRAM	(103.844832, 1.285452)	6.97	9	2022-01-09 09:18:00	18.0	2022-01-09 09:27:00	18.0	9.38	126.647183	0.989811	1.0	0.996943	LINESTRING (103.86 1.32726, 103.8599 1.32723, ...

D: Overall route of recommended trips from initial evaluation (Corresponds to results in Figure 12)



E: Overall route of recommended trips from final evaluation (Corresponds to results in Figure 14)



F: Various functions used in the system

```python
# Returns a dataframe of all trips within the same grid at a given time

def get_simulation_trips(timing, location):
    # Get longitude and latitude of starting point
    long = round(location[0], 2)
    lat = round(location[1], 2)

    # Breakdown of starting timestamp
    #week = timing.isocalendar().week
    #dayofweek = timing.dayofweek
    hour = timing.hour
    minute = timing.minute

    # Obtain all points that satisfy longitude and latitude at starting time
    start_pts = df[(df['start_hour'] == hour) & (df['start_minute'] == minute+1) & (df['LONGITUDE_PICKUP'].round(2) == long) & (df['LATITUDE_PICKUP'].round(2) == lat)].copy()
    start_pts.reset_index(drop=True, inplace=True)

    # Get coordinates as Points in a list and add coordintaes to DataFrame
    goem_start = gpd.points_from_xy(start_pts['LONGITUDE_PICKUP'], start_pts['LATITUDE_PICKUP'])
    geom_end = gpd.points_from_xy(start_pts['LONGITUDE_END'], start_pts['LATITUDE_END'])

    start_pts.insert(5, 'geometry pickup', goem_start)
    start_pts.insert(8, 'geometry end', geom_end)

    # Add columns for neighbourhoods
    start_pts.insert(6, 'PLN_AREA_PICKUP', '')
    start_pts.insert(10, 'PLN_AREA_END', '')

    droplist = []
    for i in range(len(start_pts)):
        pickpt = start_pts['geometry pickup'][i]
        droppt = start_pts['geometry end'][i]

        pickarea = 0
        droparea = 0

        for index, rows in pln_area_gdf.iterrows():
            area = rows['geometry']

            if area.contains(pickpt) & (pickarea == 0):
                start_pts.loc[i, 'PLN_AREA_PICKUP'] = rows['PLN_AREA_N']
                pickarea = 1

            if area.contains(droppt) & (droparea == 0):
                start_pts.loc[i, 'PLN_AREA_END'] = rows['PLN_AREA_N']
                droparea = 1

        # If either pickup or dropoff not within plannign area, drop the row
        if (pickarea == 0) | (droparea == 0):
            droplist.append(i)

    start_pts = start_pts.drop(droplist)

    return start_pts
```

```python
# Given a dataframe of trips, filter out trips that do not start in the most common neighbourhood

def filter_simulation_trips(simulated_trips_df):
    nbhd_start = simulated_trips_df['PLN_AREA_PICKUP'].value_counts().index[0]

    filtered_df = simulated_trips_df[simulated_trips_df['PLN_AREA_PICKUP'] == nbhd_start]
    filtered_df.set_index(['PLN_AREA_PICKUP', 'PLN_AREA_END'], inplace=True)

    return filtered_df
```

```python
# Returns multilist of starting area, possible trips area name and their end coordinates

def get_trips_data(start_df):
    coords_list = []
    start_name = start_df.index[0][0]
    start_coords = start_df['geometry pickup'].values[0]
    start_area = [start_name, (start_coords.x, start_coords.y)]
    coords_list.append(start_area)

    for i in range(len(start_df)):
        name = start_df.index[i][1]
        coords = start_df['geometry end'].values[i]
        area = [name, (coords.x, coords.y)]
        coords_list.append(area)

    return coords_list
```

```python
# Returns 2 lists (distances and durations to each area from the starting area)

def get_dist_dur(coords_list):
    # OSRM server URL
    osrm_url = "http://router.project-osrm.org/table/v1/driving/"

    # Construct the coordinates string for request
    coordinates = ";".join([f"{lon},{lat}" for lon, lat in coords_list])
    url = f"{osrm_url}{coordinates}?annotations=distance,duration"

    # Send request
    response = requests.get(url)
    data = response.json()

    # Check if request is successful
    if (response.status_code == 200) and (data["code"] == "Ok"):
        # Extract distance and duration matrices
        distance_matrix = data['distances']
        duration_matrix = data['durations']

        # Convert matrices to DataFrames
        #distance_df = pd.DataFrame(distance_matrix, columns=total_list, index=total_list)
        #duration_df = pd.DataFrame(duration_matrix, columns=total_list, index=total_list)

        # Get list of distances and durations
        distance_list = distance_matrix[0]
        duration_list = duration_matrix[0]

        '''
        print("Distance (meters)")
        print(distance_list)
        print("\nDuration (seconds)")
        print(duration_list)
        '''

        return distance_list, duration_list
    else:
        print("Error")
        print(data)
```

```python
# Returns the time window of the current time

def get_time_window(current_time):
    # Break down timestamp
    week = current_time.isocalendar().week
    dayofweek = current_time.dayofweek
    hour = current_time.hour
    minute = current_time.minute

    # Calaulate time window
    time_slice = 30 # in minutes
    time_window = np.floor((hour*60 + minute)/time_slice)

    return time_window
```

```python
# Returns linestring object of the route between 2 points

def get_route(start_coords, end_coords):
    # Define the OSRM server URL (replace with your own server URL if running locally)
    osrm_server_url = "http://router.project-osrm.org"

    # Construct the route request URL
    route_url = f"{osrm_server_url}/route/v1/driving/{start_coords[0]},{start_coords[1]};{end_coords[0]},{end_coords[1]}?overview=full"

    # Send the GET request to the OSRM server
    response = requests.get(route_url)

    # Check if the request was successful
    if response.status_code == 200:
        # Parse the JSON response
        route_data = response.json()

        # Extract route information
        route = route_data['routes'][0]
        distance = route['distance']  # in meters
        duration = route['duration']  # in seconds
        geometry = route['geometry']  # polyline string

        '''
        print(f"Distance: {distance} meters")
        print(f"Duration: {duration} seconds")
        print(f"Geometry: {geometry}")
        '''

    else:
        print(f"Error: {response.status_code}")
        print(response.text)

    # Decode the polyline to a list of (longitude, latitude) tuples
    coordinates = polyline.decode(geometry) # points are in (lat, long) format
    points = [(lon, lat) for lat, lon in coordinates]

    # Create a LineString from the coordinates
    line = LineString(points)

    return line
```

```python
def calculate_trip_fare(distance, duration):

    # Distance fare calculation
    if distance == 0:
        dist_fare = 0
    elif distance <= 10000:
        dist_fare = math.ceil(distance/400)*0.26
    else:
        excess = distance - 10000
        dist_fare = math.ceil(excess/350)*0.26 + 25*0.26

    # Duration fare calculation
    if duration == 0:
        dur_fare = 0
    else:
        dur_fare = duration * 0.16

    # Calculate trip fare
    fare = 4.4 + 2.3 + dist_fare + dur_fare

    return fare

```

```python
# Map absolute gap to closest normalised gap
gaps = [4.48530340401537, 61.86875350595568, 113.56325093775368, 167.48764015609962, 221.49231357253228, 274.73802318069613, 333.3926348745983, 357.5102228426902, 427.9675137783816, 488.5101557132261]
norm_gaps = [0.0, 0.119, 0.225, 0.337, 0.448, 0.558, 0.68, 0.729, 0.875, 1.0]

def map_gap(gap):
    mapped_gap = dict(zip(gaps, norm_gaps))

    # Find closest value to current gap
    closest = min(gaps, key = lambda x: abs(x - gap))

    # Map to normalised gap
    norm_closest = mapped_gap[closest]

    return norm_closest
```

```python
# Determine if timestamp is withing peak hours or not

def is_peak(timestamp):
    # Break down timestamp
    dayofweek = timestamp.dayofweek
    hour = timestamp.hour
    minute = timestamp.minute

    # Calaulate time window
    time_slice = 30 # in minutes
    time_window = np.floor((hour*60 + minute)/time_slice)

    # Weekday peak hours: 7-9:30am, 5-11:59pm
    if dayofweek <5:
        if ((time_window >= 12) & (time_window <= 18)) | ((time_window >= 34) & (time_window <= 47)):
            return 1
        else:
            return 0
    # Weekend peak hours: 10am-1:59pm, 5-11:59pm
    else:
        if ((time_window >= 20) & (time_window <= 27)) | ((time_window >= 34) & (time_window <= 47)):
            return 1
        else:
            return 0
```

```python
# Dynamic pricing based on distance, duration and time
# distance in meters, duration in minutes

def get_dynamic_earnings(distance, duration, expected_gap, current_timestamp):
    # Get basic trip fare
    trip_fare = calculate_trip_fare(distance, duration)

    # Apply multiplier
    gap = map_gap(expected_gap)
    peak = is_peak(current_timestamp)

    state = (gap, peak)

    for i in range(len(states)):
        if states[i] == state:
            break

    q_table_row = q_table[i]
    for j in range(len(q_table_row)):
        if q_table_row[j] == q_table_row.max():
            break

    multiplier = actions[j]

    trip_cost = trip_fare * multiplier

    # Apply commission fee (pessimistic approach)
    earnings = trip_cost*(1 - 0.2)

    return round(earnings,2)
```

```python
# Returns the predicted gap value for a given time window and location

def get_predicted_gap(time_window, location):
    # Get longitude and latitude of starting point
    long = round(location[0], 2)
    lat = round(location[1], 2)

    # Look for values that meet conditions in predicted_gap DataFrame
    predicted_df = predicted_gap[(predicted_gap['time_window'] == time_window) & (predicted_gap['LONGITUDE'] == long) & (predicted_gap['LATITUDE'] == lat)]


    # Return predicted gap value if dataframe is not empty
    if len(predicted_df) != 0:
        return predicted_df['gap_pred'].values[0]
    else:
        return 0
```

```python
def get_neighbours(coords):
    x = coords[0]
    y = coords[1]
    neighbours = []

    # Get neighbouring grid coordinates
    for i in [-0.01, 0, 0.01]:
        for j in [-0.01, 0, 0.01]:
            if (i == 0) & (j == 0):
                continue
            neighbours.append((round(x+i, 2), round(y+j, 2)))
    return neighbours
```

```python
def get_neighbours_gap(time_window, coords):
    # Get list of neighbours
    neighbours = get_neighbours(coords)

    # Create DataFrame to store time_window, coordinates and predicted gap
    nbdf = pd.DataFrame(neighbours, columns = ['longitude', 'latitude'])
    nbdf.insert(0, 'time_window', time_window)
    nbdf['predicted_gap'] = None

    # Get predicted gap for each neighbour
    for index, row in nbdf.iterrows():
        nbdf.loc[index, 'predicted_gap'] = get_predicted_gap(row['time_window'], (row['longitude'], row['latitude']))

    return nbdf
```

```python
def move_to_grid(selected_df):
    time = selected_df['expected_timestamp'][0]
    time_window = selected_df['expected_time_window'][0]
    coords = selected_df['coordinates'][0]

    # Get rows with at gap of at least 10 excluding the current grid
    new = predicted_gap[(predicted_gap['time_window'] == time_window) & (predicted_gap['gap_pred'] > 10)  & (predicted_gap['LONGITUDE'] != coords[0]) & (predicted_gap['LATITUDE'] != coords[1])].copy()
    new.insert(2, 'time', time)
    new.insert(3, 'dist', None)
    new.reset_index(drop=True, inplace=True)

    for index, row in new.iterrows():
        row['LATITUDE'], row['LONGITUDE']
        new.loc[index, 'dist'] = np.sqrt((coords[0] - row['LONGITUDE'])**2 + (coords[1] - row['LATITUDE'])**2)

    move = new[new['dist'] == new['dist'].min()]
    print('\nNo trips, move to grid:', move['LONGITUDE'].values[0], move['LATITUDE'].values[0], 'at', time)

    move_coords = [coords, (move['LONGITUDE'].values[0], move['LATITUDE'].values[0])]
    dist, dur = get_dist_dur(move_coords)
    dur = round(dur[1]/60)

    return move, dur
```