FINAL REPORT

A DATA SCIENCE APPROACH TO OPTIMIZE THE PROFIT FOR BATTERY HOLDERS IN AUSTRALIA

Fangyue Chen (z5175111), Yumeng Dong (z5183940), Jiaying Liu (z5215876),
Shiding Zhang (z5183938),Qinqin Zhu (z5211029)

School of Mathematics and Statistics
UNSW Sydney

November 2020

Submitted in partial fulfilment of the requirements of
the capstone course DATA3001

Plagiarism statement

I declare that this thesis is my own work, except where acknowledged, and has not been submitted for academic credit elsewhere.

I acknowledge that the assessor of this thesis may, for the purpose of assessing it:

- Reproduce it and provide a copy to another member of the University; and/or,
- Communicate a copy of it to a plagiarism checking service (which may then retain a copy of it on its database for the purpose of future plagiarism checking).

I certify that I have read and understood the University Rules in respect of Student Academic Misconduct, and am aware of any potential plagiarism penalties which may apply.

By signing this declaration I am agreeing to the statements and conditions above.

Signed: Yumeng Dong Date: 21/11/20

Signed: Cassie Liu Date: 21/11/20

Signed: Fangyue Chen Date: 21/11/2020

Signed: QINQIN ZHU Date: 21/11/2020

Signed: 张小丁 Date: 21/11/2020

Acknowledgements

As we near completion of our project, we would like to thank everyone who supported our group in finishing this capstone project.

First of all, we are grateful to the School of Mathematics and Statistics for successfully running the Data Science and Decisions in Practice capstone course for us. We also take this opportunity to express our deep sense of gratitude to all the staff in DATA3001 for creating a safe online learning environment and a positive and harmonious atmosphere in the midst of the COVID-19. It was an invaluable experience and gave us a better understanding of the practicalities of analysing our data.

We are really fortunate that we met Dr Michele de Nadai, who guided our project. We would like to thank him for his advice and patience during the construction of the model. In spite of some problems, we ended up with a satisfactory result. During the semester, Dr Michele de Nadai met with us twice a week to analyse the difficulties we encountered and lead us to valuable directions at key points. We are very grateful for this. He also helped us point out problems within the project we have. His advice on data restructuring has significantly improved the effectiveness of our models. We feel really lucky to work with Dr Michele de Nadai.

We would also like to thank our capstone sponsor Oliver Nunn from Australian Energy Market Commission. First of all, we are very grateful to him for providing us with such a meaningful and interesting project, which made an unforgettable experience for us. His guidance has given us a deeper understanding of the profitability process for battery holders and allowed us to plan our projects clearly. In addition, we would like to thank him for taking time out of his busy schedule to hold weekly meetings with our group.

Abstract

The electricity price in the Australian National Electricity Market changes every 5 minutes, where the battery owners can make profit through these price fluctuations. In this project, an algorithm is developed for the battery owners to maximize their profit. The principle of this algorithm is to deduce a proper charging strategy based on the predicted electricity price that is generated by a Dense Neural Network Model. Throughout this project, a practical procedure was designed and implemented to make revenue. Ultimately, the model made a profit of 11086 AUD over 3 months. This highlights the huge commercial potential of this model as it represents 72% of the real benefit benchmark.

Contents

Chapter 1

Introduction

The National Electricity Market (NEM)[1] is a cross-state wholesale market where generators and retailers trade electricity in Australia. It interconnects the six eastern and southern states, and holds around 80% of total electricity trading in Australia. The NEM not only facilitates the transaction of electricity between generators and retailers, but also stimulates competition in market prices. Since Australia has one of the most expensive electricity prices in the world, using batteries to store electricity generated by renewable resources such as wind and solar has become more commonly used by not only households but also large industries. To make electricity prices more responsive to sudden changes in the amount of electricity generated from renewable energy sources, the Australian Energy Market Commission (AEMC) develops the rule of updating the electricity every five minutes from July 2022. This would also benefit battery owners since they can buy electricity at low prices and sell at higher prices.

This project will help the quantitative team of AEMC consider the business opportunities arising from five-minute fluctuating electricity prices from the standpoint of battery owners who play the role of retailers in the electricity supply chain. The battery used in this project is assumed to be 2 Megawatt-hours (MWh), 1MW charging rate. It will lose around 19% electricity while charging into battery due to efficiency. Battery cannot discharge when it is empty, or charge it when the battery is full capacity. As the electricity price fluctuates, the most profitable method is to discharge at the high price and charge when the price is low. To achieve that, a forecasting model using time series structure data that can predict the energy price for each state under the current five minutes bidding system needs to be developed. Then suggestions would be provided on when to charge or discharge the battery to increase the battery owner's profit.

Chapter 2

Literature Review

The report from Australia energy regulator[2] gives us a better understanding of the current situation in the Australian electricity market. Coal, gas, solar, water and water are main sources of generators to produce electricity. Then the electricity will be transmitted to the energy retail interface through the transmission networks and distribution networks. Authorised or licensed energy retailers buy electricity from generators and sell to energy users including large retail customers, householders and embedded network customers. The market price of electricity is not constant, and fluctuations in the price of electricity can be caused by changes in demand from consumers and changes in the amount of electricity generated. As the rising proportion of renewable energy in the source of the generator, more low inertia and unstable voltage will appear. Thirty-minute quotations hardly reflect the production costs of electricity. To solve this issue, the settlement period of electricity quotes will change from thirty-minute to five minutes from July 2022.

To attain the objective of forecasting the prices or demand, fundamental analysis and technical analysis techniques are used by a number of researchers. The most efficient way to forecast the future is to understand the present scenarios. Carter Bouley [3] found that electricity prices follow daily, weekly and seasonal patterns under time series structure. This report sets data from 2013 to 2018 as training data and uses the 2019 data to assess how well the model performs on unknown data. The predictions for randomly selected days have good performance. The neural network model uses one week data to predict one day ahead. In order to let the neural network understand the importance of the order of time, the sliding window matrix is the key dimension as the new matrix is built every hour. Then transfers the price prediction of 24 hours ahead into a supervised learning problem. Moreover, Carter also tried a Long-Short Term Memory (LSTM), Gate Recurrent Unit (GRU) with convolutional layers, as well as classical machine learning, regression trees and ARIMA forecasting to select the best model by comparing price predicting results. Besides that, a home battery is also simulated with 14KWH capacity and 5 KW of power and the charging hour is set as three hours for simplicity. The model will help them select 3 cheapest hours to charge and 3 most expensive hours to discharge during 24 hour prediction.

Besides predicting the electricity price, Jakub and Rafał[4] computed electricity spot price prediction intervals using quantile regression and forecast averaging. They proposed that using multiple methods to construct empirical prediction intervals (PI) is better at price forecasting than using a single method. Since multiple methods can provide more information on future price evolution. For individual models,

he mentioned a lot, such as autoregressive models (AR + MA, ARX), spike preprocessed autoregressive models (p-AR,p-ARX), threshold autoregressive models (TAR, TARX) and two classes of semi-parametric autoregressive models. They also propose a new method Quantile Regression Averaging (QRA) for constructing prediction intervals, which utilizes the concept of quantile regression and a pool of point forecasts of individual models. For comparison, they evaluated the quality of the forecasts interval by comparing the nominal coverage to the true coverage. It provided additional information on the evolution of future prices. In particular, they allowed for a better assessment of future uncertainty and for planning of different strategies for the range of possible outcomes indicated by the forecast interval.

While many researchers utilize predictive models to perform the predicting electricity price, the author Jesus, Fjo and Bart[5] mentioned that the area of deep learning algorithms is a good way to explore. Thus they proposed deep learning approaches and empirical comparison of traditional algorithms for forecasting spot electricity prices. They propose four different deep learning models (a DNN as an extension to the traditional MLP, a hybrid LSTM-DNN structure, a hybrid GRU-DNN structure and a CNN model) for predicting electricity prices and they found it led to improvements in predictive accuracy.The Recurrent Neural Network they used is a type of network that builds additional mappings to hold relevant information from past inputs and that are suitable for modeling time series data, e.g. electricity prices. They also try to establish an extensive benchmark for commonly used forecasters to predict the electricity prices.

Chapter 3

Materials

3.1 The Software

The softwares used in this project are Jupyter Notebook for Python, RStudio for R, Github and Google Colab. Data cleaning and elementary analysis are implemented in Python, where 'Pandas' and 'Numpy' libraries are used for data cleaning, 'Matplotlib' library for data visualization. Observing data trends and modeling based on time series structure are implemented in both R and Python, since the cooperation between the two software programs can speed up adjustments of SARIMA modeling. The machine learning toolkits 'Keras' and 'tensorflow' are used for building Neural Network models . Finally, Gurobi Optimizer with python interfaces is used to solve mathematical optimization problems since it is widely used in the industry.

3.2 The Data

The given data contains four types of files: ' biddayoffer', 'bidperoffer', 'dispatchprice' and 'dispatchregionsum'. As the goal of the model is to forecast the price of the electricity, the strongest indicators are Total Demand and RRP - as such only the 'dispatchregionsum' and 'dispatchprice' datasets are chosen to be used.

Initially, the 'dispatchregionsum' and 'dispatchprice' data sets had both a one year record (around 500MB) as well as historical 10 year record (around 2.1GB). Within these records there are many useless or empty columns in the original files which make the data quite messy. After data cleaning, the data is collated into one file of 6311536 rows, 4 columns and 297MB in size. The key variables are SettlementDate, RegionID, Regional Reference Price (RRP) and TotalDemand.

A brief description of the key variables are provided below:

SETTLEMENTDATE: Settlement point for electricity spot prices
REGION: The six region: New South Wales (NSW), Queensland (QLD), South Australia (SA), Western Australia (WA), Tasmania (TAS) and Victoria (VIC)
RRP: Regional Reference Price of electricity per kilowatt
TOTALDEMAND: Total electricity demand of region

Data used in this report includes electricity total demand and regional reference price (RRP) in six states of Australian over the twelve year period. The observed data spans from July 2008 to July 2020 and recorded at five minute intervals.

3.3 Data Cleaning

Python is used to clean data and create relevant variables for model building. In this case, SETTLEMENTDATE, RRP, TOTALDEMAND are selected. Variables 'year', 'month', 'day' are added to the data because all of these variables contain the information to account for the potential seasonality in this time series data.

Figure 3.1: Data Description

	SETTLEMENTDATE	REGIONID	RRP	TOTALDEMAND	time	year	month	day
0	2008-07-01 00:00:00	NSW1	27.43703	9027.49	00:00:00	2008	7	1
6	2008-07-01 00:05:00	NSW1	27.82000	9067.65	00:05:00	2008	7	1
11	2008-07-01 00:10:00	NSW1	27.82000	9115.77	00:10:00	2008	7	1
16	2008-07-01 00:15:00	NSW1	27.82000	9049.75	00:15:00	2008	7	1
21	2008-07-01 00:20:00	NSW1	27.74008	8935.08	00:20:00	2008	7	1

Chapter 4

Exploratory Data Analysis

Firstly, a visualization of one day's RRP and TOTALDEMAND was done to check the possible relationship between them. As the Figure 4.1 shows, there is a weak relationship between them, both of them have the similar trend in time interval (800,2400). To further analyse, this report will develop a deeper analysis.

Figure 4.1: One day plot of RRP and Totaldemand

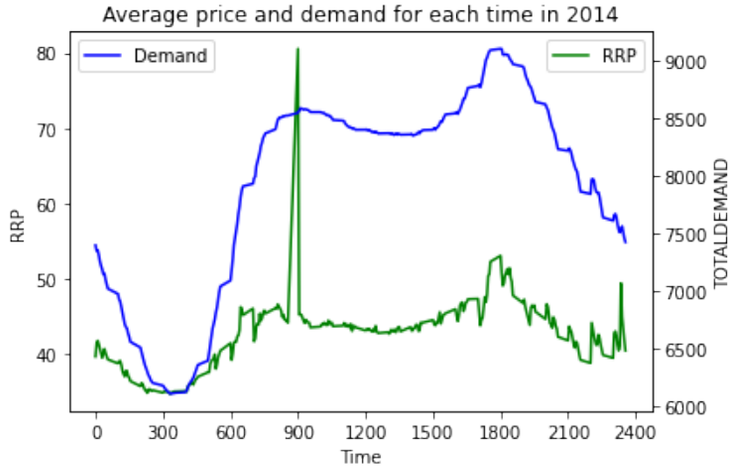
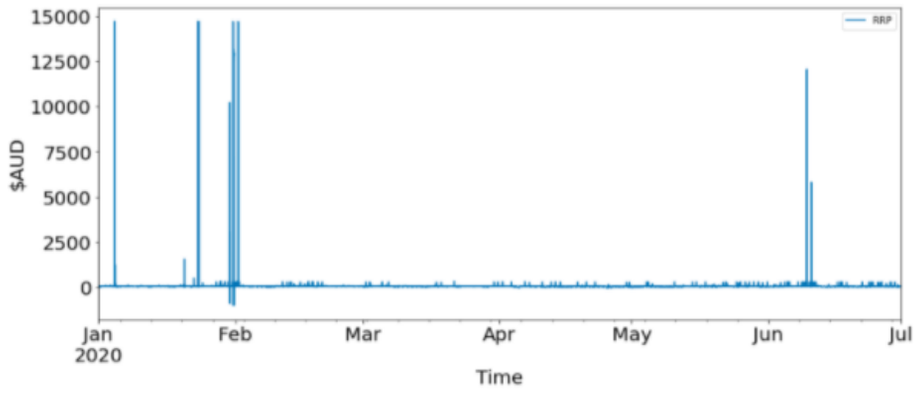


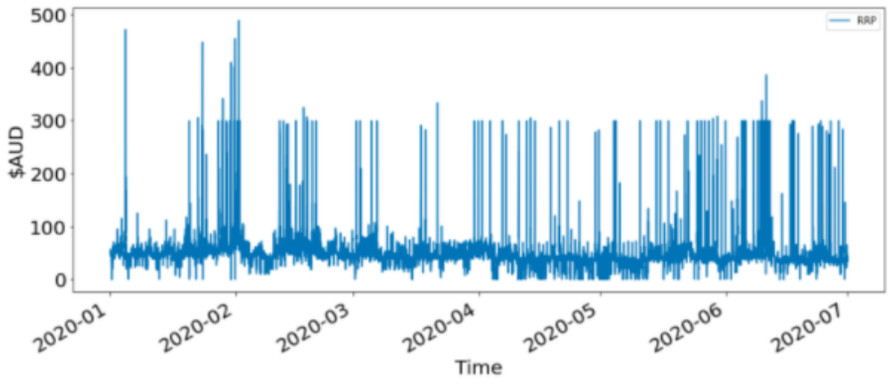
Figure 4.2: One year of RRP with Outliers



To further analyse the data, all the RRP in 2020 have been plotted (Fig 4.2) and there are some extremely high points of RRPs. These outliers are not caused by recording error, there are proper reasons why these prices are so high. For example, the outlier in early January was caused by the bush fires in NSW. To avoid the impact of these outliers on model accuracy, a lower and upper limit of [5, 500] was

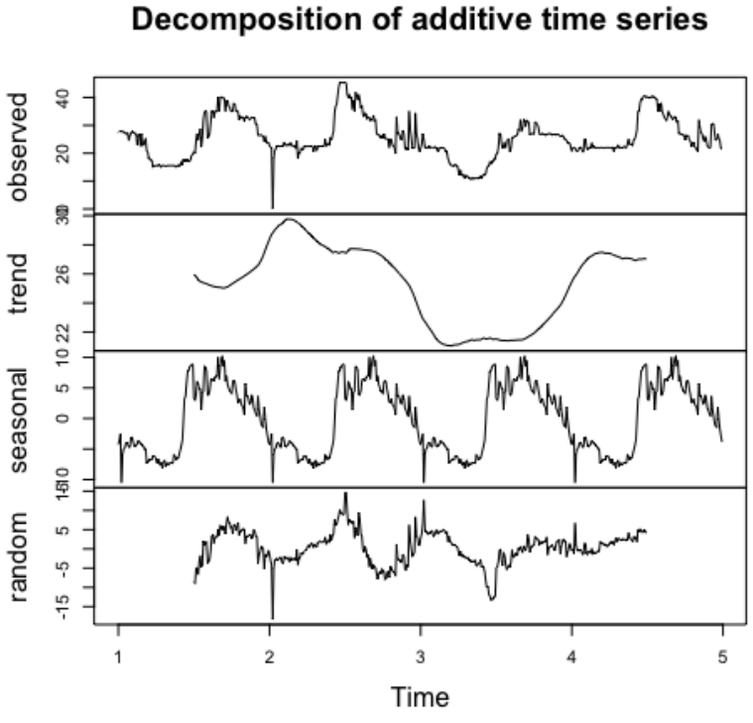
set on the data.

Figure 4.3: RRP after Removing Outliers



The Fig 4.3 is now a better plot to understand the trend and seasonality of RRP. Since the data is a time series data, firstly it is important to check our data's potential trend. The Fig 4.4 shows trends in time, seasonal trends and also the potential for Random error at a given time. According to this graph we could figure out that there is a significant seasonal trend. Therefore, seasonal factors should be taken into account in later model building.

Figure 4.4: Decomposition



Chapter 5

Methods and Techniques

5.1 Multiple Linear Regression Model

Multiple Linear Regression model is our first attempt to predict the response variable **RRP**. As mentioned in Chapter 4 of this report, apart from the seasonality of **RRP**, past **TOTALDEMAND** should also be considered as a potential variable. After analysing the data, we noticed that the pattern of **RRP** in a day varies between weekdays, weekends and seasons. Hence, 'weekday' and 'season' are related to '**RRP**', and these two variables were added into the model.

Figure 5.1: Slice of data with additional variables

SETTLEMENTDATE	RRP	TOTALDEMAND	year	month	day	weekday	time1	season
2008-07-01 00:05:00	27.82000	9027.49	2008	7	1	2	5	4
2008-07-01 00:10:00	27.82000	9067.65	2008	7	1	2	10	4
2008-07-01 00:15:00	27.82000	9115.77	2008	7	1	2	15	4
2008-07-01 00:20:00	27.74008	9049.75	2008	7	1	2	20	4

Firstly, setting the highest price to \$500 and the lowest price to \$5 to remove some outliers and normalizing our data to improve the performance of this model. Then using sklearn in python, a multiple linear regression model with explanatory variables 'TOTALDEMAND', 'Month', 'Weekday', 'Time', 'Season' is built. Using the data from 01/01/2020 to 31/05/2020 as training data for this Multiple Linear Regression Model gives us the below equation:

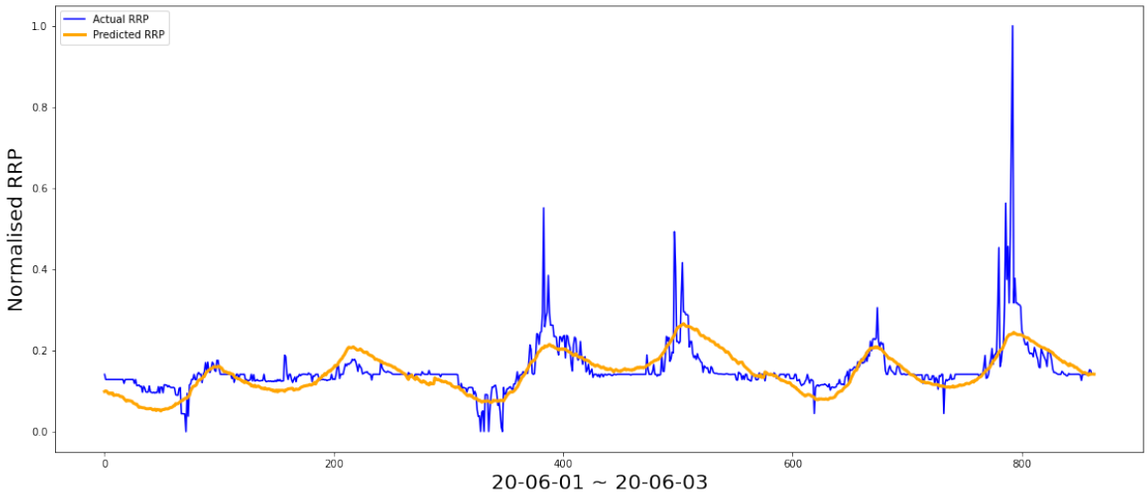
$$\begin{aligned} RRP =& 0.0533 + 0.2224 * Demand - 0.0319 * month + 0.0033 * weekday \\ & - 0.015 * time + 0.0104 * season \end{aligned}$$

R-square is 0.3220.
R-square is a coefficient of determination which represents the explanatory power[6] of the statistical model in the form of a scale with range [0 , 1]. As such R-square can be considered a measure of accuracy - the closer to 1 the better fit of the model.(Jeffrey M.,2008,p200) Clearly, the 0.3220 here means this model did not perform well.

Figure 5.2 is a good validation of the previously mentioned argument, it represents the predictions of RRP in the next three days using the model, which is from

01/06/2020 to 03/06/2020. From the predictions output by the model, it can barely catch the peaks. This is because from the equation above, the coefficients for the variables are relevantly small and fitting time series data into a linear regression is not the best model to use in this project.

Figure 5.2: Predicted RRP vs Actual RRP , From 01/06/2020 to 03/06/2020



5.2 SARIMA Model

Since there is a seasonal trend of RRP as mentioned before, SARIMA[7] model is more suitable for predicting future RRP. Compared with the ARIMA model, SARIMA could support time series data that has a seasonal trend. It contains three hyper-parameters to tune: Auto-Regression(AR), Integrated (I) Moving Average (MA) and an additional parameter for the period of the seasonality.

This report will follow four steps to do the prediction by SARIMA modelling.

1.Stabilize non-stationary time series data.

2.Hypothesis testing to determine whether the residual sequence is stable. The white noise condition will also be avoided.

3.Build the corresponding time series model by using the most suitable parameters.

4.Visualize and judge modeling effects by comparing predictions with test sets.

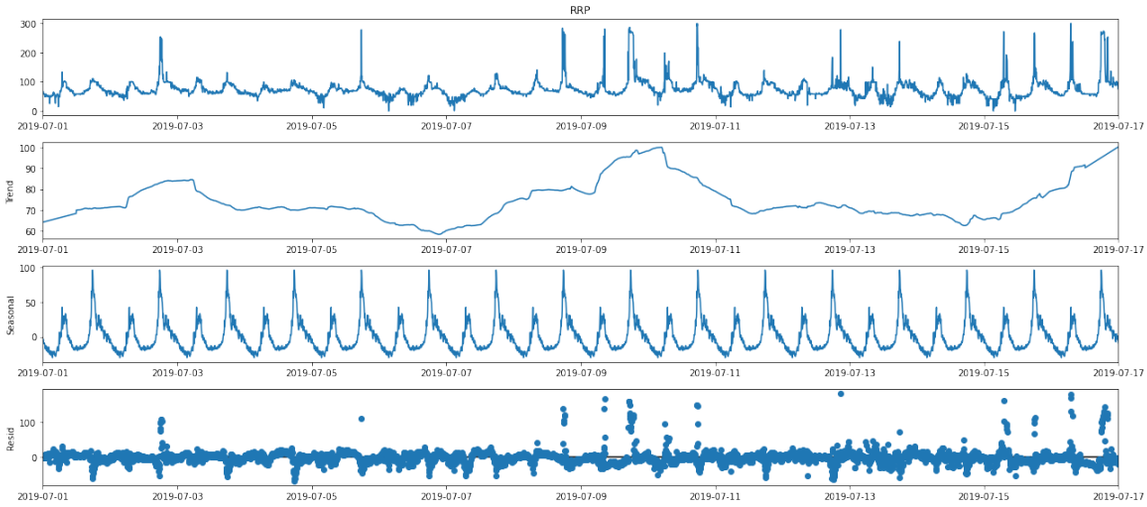
Step 1: Stabilize the Data

First of all, the data is divided into a training set and a testing set, where the testing set is used as a measure of the model's accuracy. To increase the running speed, firstly take the data samples from '01/07/2019 00:00:00' to '17/07/2019 00:00:00' as the training set and let next day records as the testing set.

Figure 5.3 illustrates price fluctuations, trend changes, seasonal fluctuation trends and the distribution of residual values of RRP. The residuals of the original data are not tightly clustered on the Y= 0 line, as can be seen from the diagram of the residuals in the bottom row. It shows that the data is random and irregular, predictive models will be difficult to build. Therefore, first-order differential should

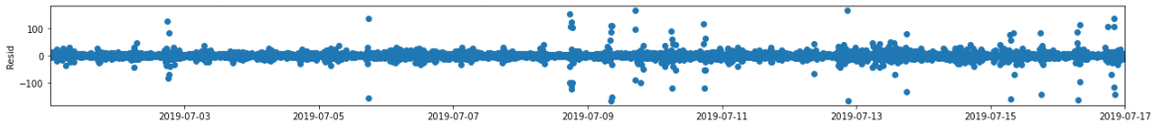
be done to stabilize original non-stationary data.

Figure 5.3: Decomposition of RRP



Step 2: Testing

Figure 5.4: Residual plot after First-order Differentiation



Compared with the original residual plot (Fig 5.3), the data with first-order differential was a more stable residual plot, as it had a more stable residual plot (Fig 5.4).

Figure 5.5: ADF test results after First-order Differentiation

```
Results of Dickey-Fuller Test:
Test Statistic                  -1.707080e+01
p-value                          7.828018e-30
#Lags Used                       2.400000e+01
Number of Observations Used      4.582000e+03
Critical Value (1%)             -3.431778e+00
Critical Value (5%)             -2.862171e+00
Critical Value (10%)            -2.567106e+00
```

Moreover, the Augmented Dickey Fuller test (ADF Test) is a common statistical test used to test whether a given time series is stationary or not. According to the result (Fig 5.5), the p-value here is very significant, so after differentiating, the data is stationary. And there is also no white noise. Subsequent analysis will use the processed data as regularized historical data usually results in highly accurate predictions.

Step 3: Model Development

Through observing the graph of the Auto-correlation function(ACF) (Fig 5.6), the AR order is 1 and the Partial auto-correlation function (PACF) (Fig 5.7) of MA order is 5.

Figure 5.6: Plot of ACF

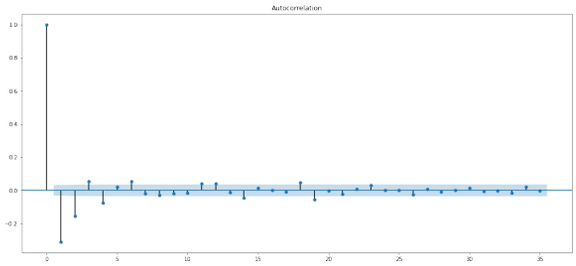
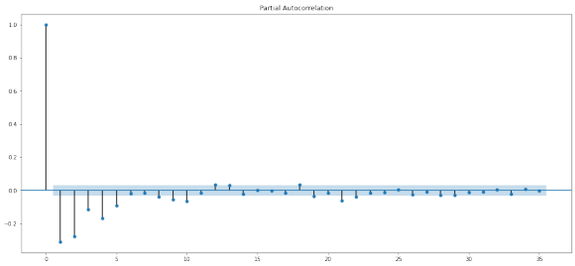
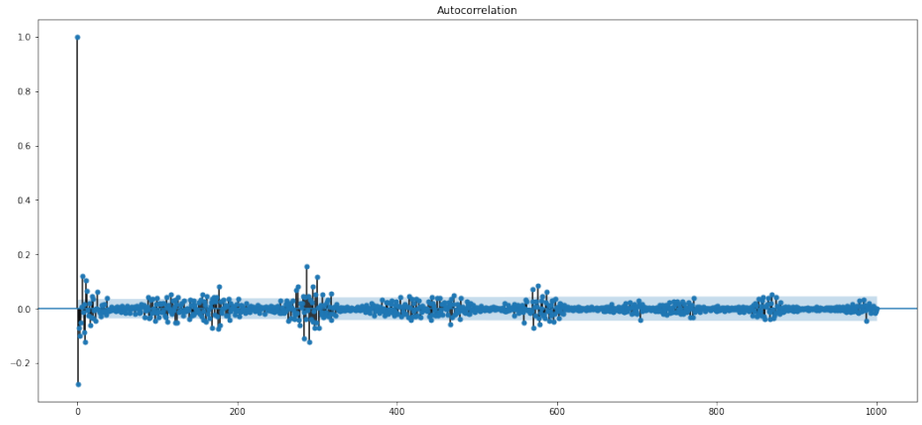


Figure 5.7: Plot of PACF



Then combining the auto.arima in R to make sure the best parameters (p, d, q) (P, D, Q) while 'p' means trend auto-regression order, 'd' represents trend difference order and 'q' is trend moving average order. Similarly, 'P' means seasonal auto-regressive order, 'D' represents trend difference order and 'Q' is trend moving average order. And '[288]' is the number of time steps for a single seasonal period where 288 is one day's records. As you can see from the Figure 5.8, there is a peak at every 288 points, so 288 is a reasonable value to take. Moreover, the corresponding variable 'weekday' is added into the SARIMAX model as an additional variable X.

Figure 5.8: ACF showing daily seasonality(288)

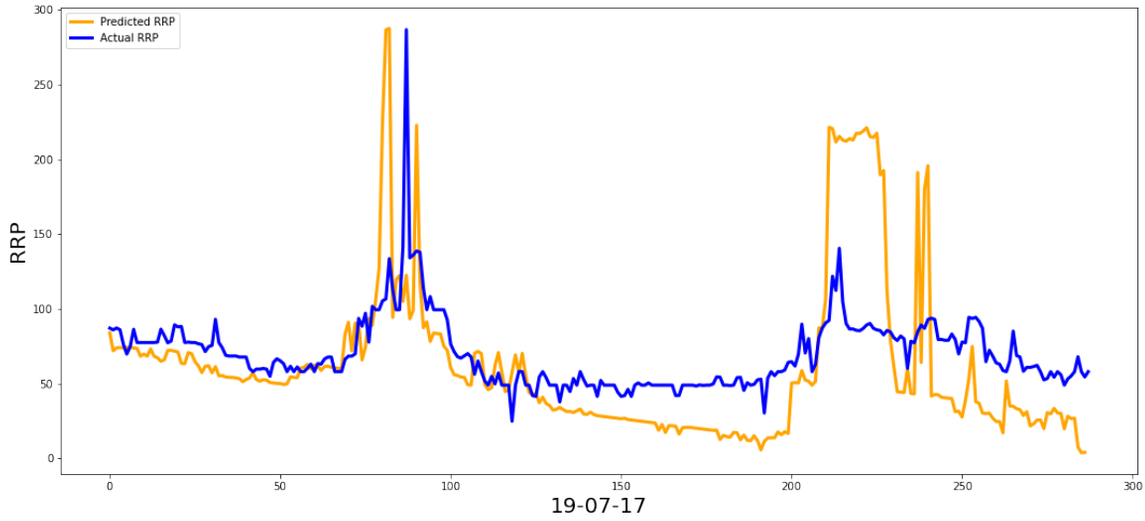


Step 4: Model Evaluation

Finally, combining the two methods of selecting SARIMA orders, the hyper-parameters chosen for the SARIMAX model is SARIMA (5, 1, 2)(0, 1, 1,[288]) which has the smallest AIC. As shown in Fig 5.9, the orange line is the predicted RRP and the blue one is the actual RRP of on 17th of July 2019 (288 records), clearly the prediction did not meet previous expectations. This model barely predicts precisely

the RRP at each point, only the trend is caught. For example, looking at the data points after 120, there is a significant bias of our prediction. Moreover, around 220 there is a high peak of our prediction which is not fit to the facts. In order to get more accurate predictions, and according to experience of relevant projects mentioned in literature review, a neural network model is the next step to find the most appropriate model.

Figure 5.9: The predicted RRP vs the real RRP: SARIMA(5, 1, 2)(0, 1, 1,[288])



5.3 Neural Network Model

In general Neural Networks are like the human brain, they are built from many individual interconnected nodes which all perform a sub-process. This section of building the Neural Network model mainly follows two steps. Firstly, training the Neural Network models. Then choosing the best one from the trained models.

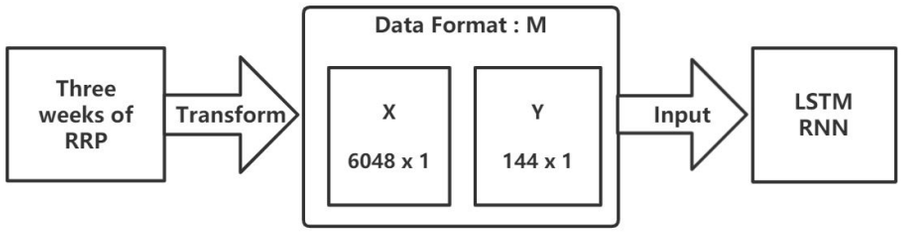
Step 1: Model Training

The Recurrent Neural Network (RNN)[8] was the first model to be tried, since RNN is known as a model for handling time series data and predicting the future. A second attempt was the Long-Short Term Memory (LSTM)[9], which is a special RNN that is explicitly designed to avoid long-term dependency problems. The last model is the Dense Neural Network (DNN), which can handle large data volumes well. DNN[10] is a neural network which belongs to a type of supervised learning. It trains the model from labeled training data and can be used for regression and classification. The layers are fully connected by the neurons in a network layer. Relevant information from past inputs are held and therefore it is suitable for modeling time series data. All of these models will use three weeks RRP as input to predict the next 12 hours RRP in one go, where the reason for choosing the next 12 hours to predict is explained in Chapter 6.

The training and validation data used in this step is the data between 01/07/2008 and 31/03/2020, on the other hand the testing data to be predicted is the data varied from 01/04/2020 to 30/06/2020.

There are two versions of the input training data, along with the process of model improvements. The first version is to use the data of three weeks (without variation) as the input for RNN and LSTM. This flowchart below (Fig 5.10) shows how the raw data is transformed and then fed into the LSTM or RNN model. Here the vector X is of size 6048*1, which equals to 12(number of intervals per hour)*24(hours)*7(days)*3(weeks)*1. The Y vector is the next 12 hours.

Figure 5.10: Data transforming for LSTM/RNN



The second version data is using three weeks with transformation as input for DNN. From Chapter 4 discussed before, it can be visualized that the time series data follows a weekly pattern, i.e. this Monday's RRP data has a strong relationship with the last Monday's RRP data. Therefore, a reasonable way that would improve the forecasting performance would be to reshape the input data to contain the information of data in the same weekdays and then sliding for three weeks. The implementation is in the function 'GetNewData1' (see Appendix). The explanation can be visualized from the figures below.

Figure 5.11: Data transforming for DNN

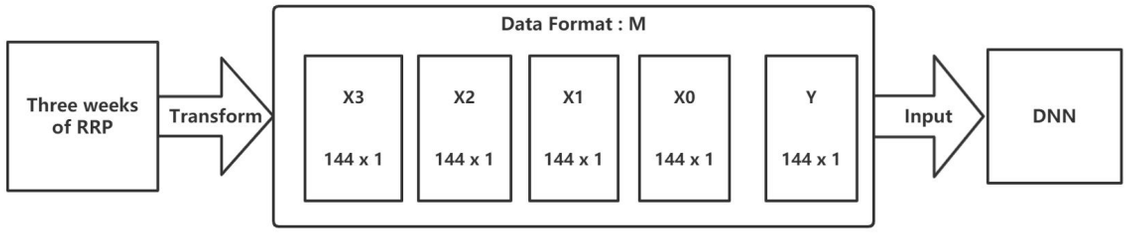


Figure 5.11 is the flowchart of how the data is transformed and then as input fed into the DNN model. The three weeks ahead data is passed into the 'GetNewData1' function, in which the raw data is transformed into four X vectors and one Y vector. Since the data points are formatted in 5 minute intervals, for each hour, there will be 12 data points. Therefore, for 12 hours, there will be 144 data points.

Figure 5.12: Timeline of vectors inside Matrix M

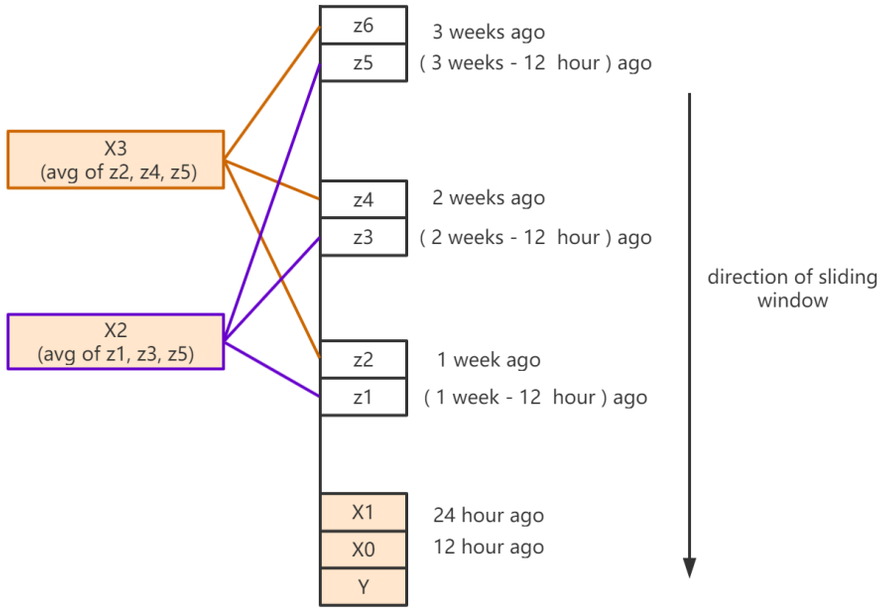


Figure 5.12 shows how each of the four X vectors are calculated, where Y vector is the data to be predicted, and *X0* is the data from 12 hours ago and *X1* is the data from 24 hours ago. To calculate *X2*, firstly the data gained at the same weekday and time but one week ago is denoted as *z1* . The 12 hours ago data as *z2* and next is to find the same two data frames for two weeks ago. Next, compute the average of *z1*, *z3*,*z5* and store in *X2*, compute the average of *z2*, *z4*,*z6* and store in *X3*.

Figure 5.13: Sliding windows Visualization

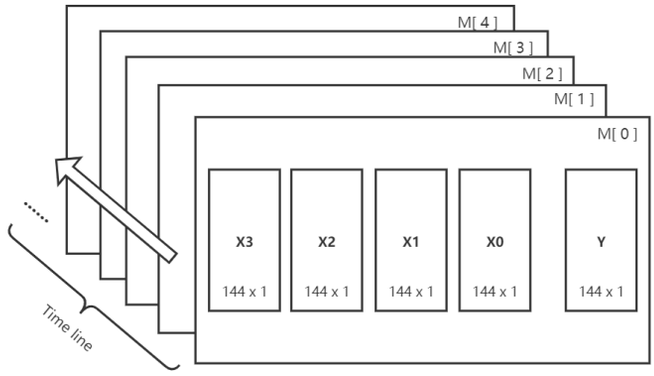


Figure 5.13 shows the procedure for each matrix M. Here each matrix M can be considered as a window. By sliding the windows through the timeline from 2008, the whole data format is built, which is how the DNN model is trained.

Step 2: Model Selection

F1 score[11] is known as a measure of accuracy for model evaluation. It conveys the balance of precision and recall. F1 score ranges from 0 to 1, where the larger the value, the better the result. Below is the mathematical formula for F1 score.

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

A number of experiments were done before the type of model being finalized. The table below presents the best performance for each model and F1 score is chosen to be the evaluation criteria.

Figure 5.14: Comparisons between 3 Neural Network models

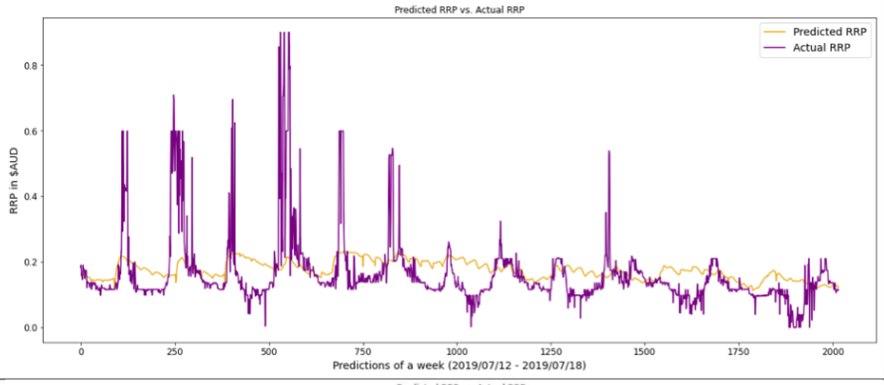
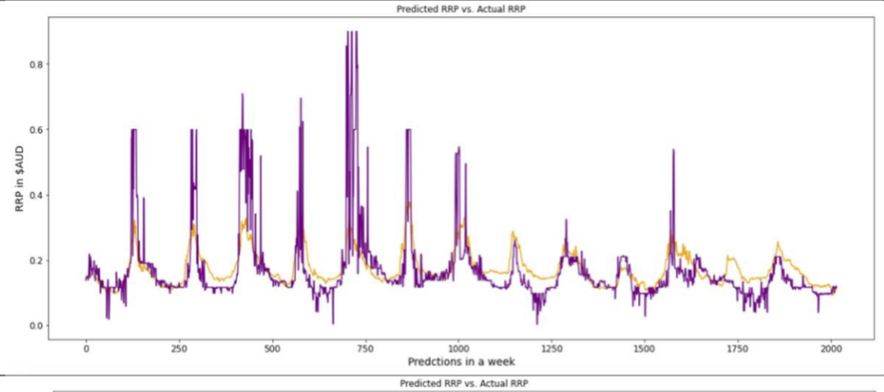
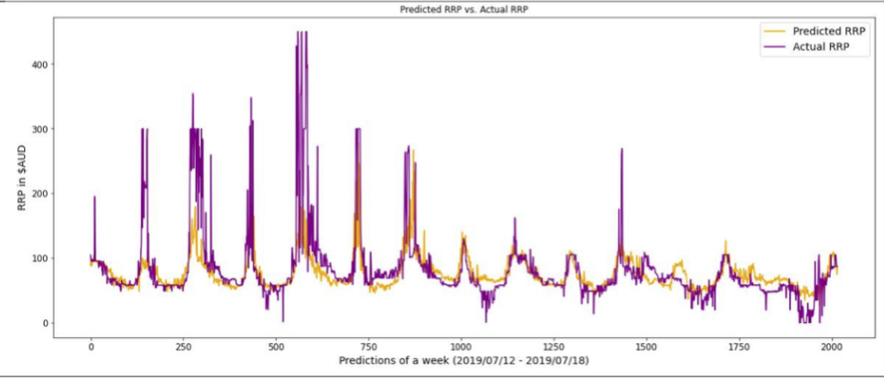
Model	Plot of Prediction RRP vs. Real RRP	F1 Score
Long-Short Term Memory		0.12
Recurrent Neural Network		0.33
Dense Neural Network		0.54

Figure 5.14 shows the weekly performance for each of the three models. The purple line is the real testing data and the yellow line represents the predicted results. It can be clearly seen that LSTM has the most unsatisfactory performance, with the lowest F1 score. The weekly prediction has a slight trend and does not match the troughs. RNN, DNN has a more preferable performance, which matches both the peaks and troughs well. Moreover, the F1 score for DNN is higher than for RNN, which suggests that DNN has a better performance generally.

Figure 5.15: Predicted RRP vs. Actual RRP on 17/07/2019

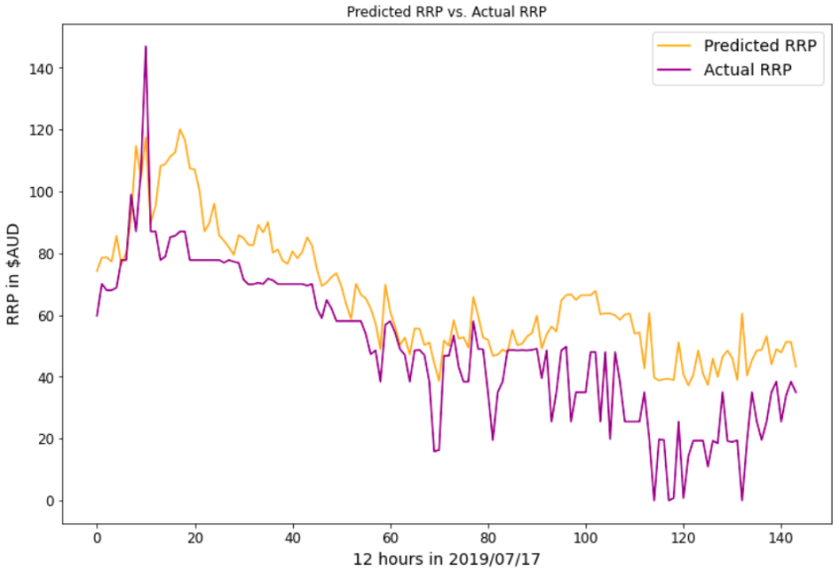


Figure 5.15 shows the performance of the DNN model within a 12 hour time period. It seems that the predicted results match most of the fluctuations correctly but overall the predicted data is a bit higher than the real one.

Figure 5.16: Training Loss vs. Validation Loss

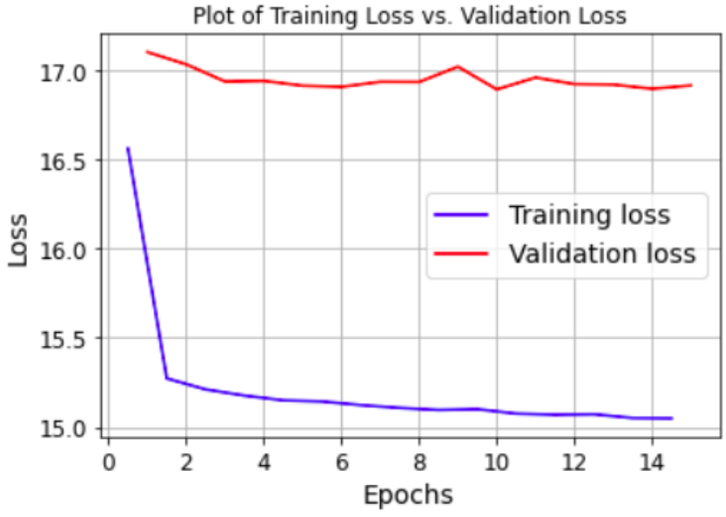


Figure 5.16 shows the plot of training loss and validation loss, it is clear that the validation loss is higher than the training loss throughout the procedure. For training loss, there is a huge drop from the first to the second epoch, then goes down slowly later on.

After testing on different parameters (i.e. epoch, loss function, activation function, number of layers), the final model is a DNN model with three layers, with Selu as the activation function, Log-cosh as the loss function, and Adam as the optimizer. Figure 5.17 is the summary of model.

Figure 5.17: Summary of the DNN model

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, None, 288)         1440
_________________________________________________________________
dense_1 (Dense)              (None, None, 188)         54332
_________________________________________________________________
dense_2 (Dense)              (None, None, 1)           189
=================================================================
Total params: 55,961
Trainable params: 55,961
Non-trainable params: 0
_________________________________________________________________
```

Chapter 6

Analysis and Discussion

6.1 Optimization

The software Gurobi is a tool to maximize or minimize a mathematical expression. In this project, the optimization object is defined as maximizing the product of X·P, where P means the vector of RRP, and X represents the vector that can be adjusted by Gurobi[12]. Then adding the constraints into the Gurobi program, Gurobi can find the optimal X to maximize X·P.

Figure 6.1: Contrast of actual RRP and Actions made by Gurobi

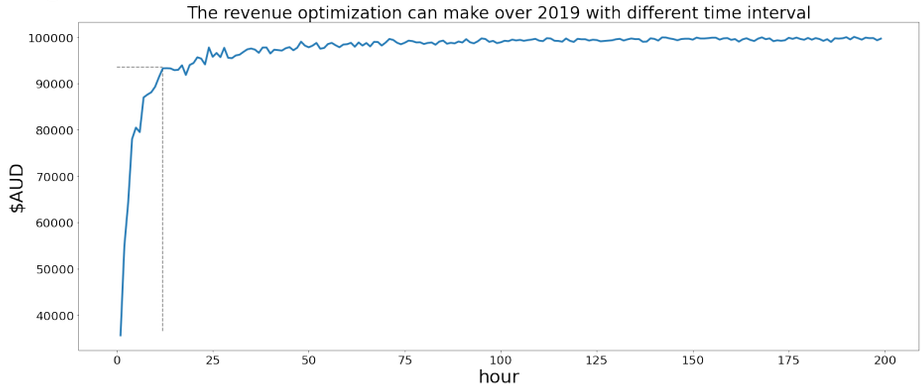
In Figure 6.1, one day RRP (01/04/2020) were used as the input in the Gurobi. The X-axis represents the points of each 5 minutes. For example, when x is equal to 24, it means the 24th 5 minutes on this day, which is the 120th minute of the day, is 2AM. The red line is RRP, the green line is the action induced by Gurobi optimization, which means the amount of electricity charged or discharged . The green line is discharging when it is greater than 0, charging when it is smaller than 0, doing nothing when it is 0. The green line ranges 1/12 to -1/12 and is due to the charging rate being 1MW/H, equivalently 1/12MWH every five minutes

In Figure 6.2, the initial battery information setting was 2000 $AUD, 1MWH electricity left in the battery. The X-axis is in hour, Y-axis is in $AUD, with this plot showing how much cash revenue the Gurobi program can make over 2019 under different time lengths. Firstly, the real 2019 RRP data from New South Wales was used. The data for the year was then equally divided into sequential slices. Secondly, a Gurobi function is programmed that uses the battery information and a

list of price, P, as input. It was then processed by linear programming to produce an output of the battery information and a list of battery actions X(Discharge or Charge battery). Thirdly, the working procedure - for each time length, the first slice of RRP and initial battery information were put into the Gurobi program, it would then use the output battery information and the next slice of RRP as the input for the next round. Eventually, the last slice will produce the final result of how much revenue this time length can make over 2019.

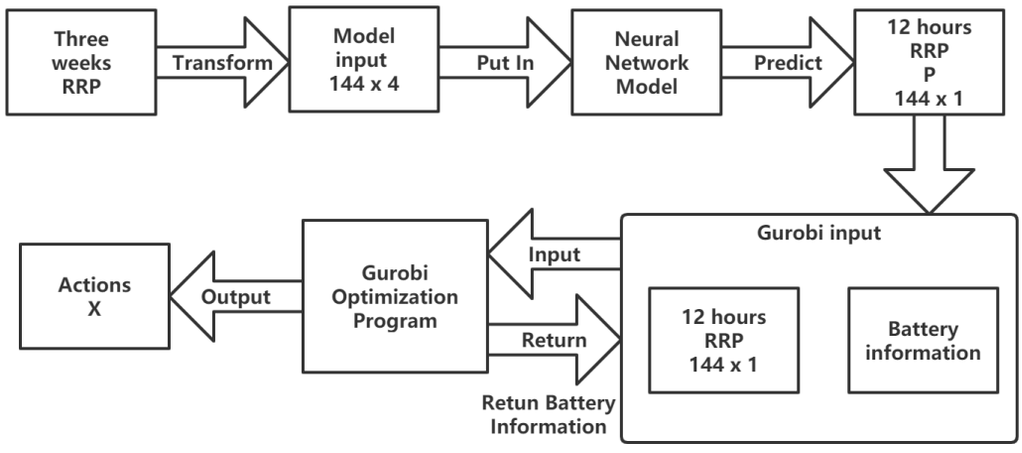
From this logarithm-like line in Figure 6.2, it can be derived that the long time intervals have a positive effect on making revenue, but the marginal advantage of time length is going down with the length increases. The time consumption of doing one 12 hours Gurobi calculation is 0.084s, 100 hours Gurobi is 5.460s, 200 hours will explosively grow to 23.348s. Thus, the 12 hour is the efficiency-performance balance point.

Figure 6.2: Revenue made with different length of time intervals



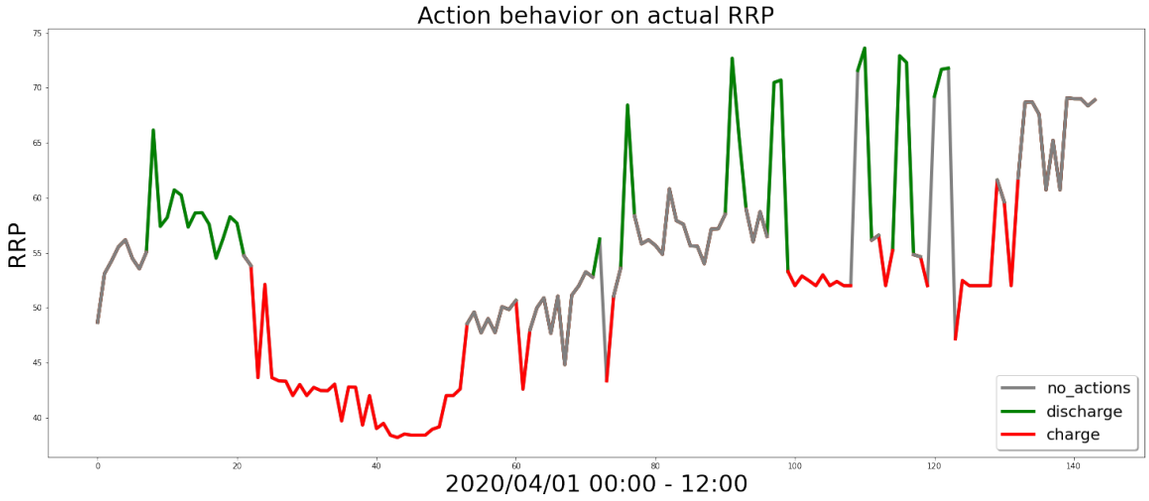
In the optimization part of this project, past three weeks real data are used as input for the neural network model, then the model produces a 12 hour RRP prediction. The RRP predictions will be the source material to do the optimization. Finally the suggestion of when to charge or discharge the battery will be provided from the Gurobi optimization program. The flow diagram Figure 6.3 represents the whole process.

Figure 6.3: Process of Neural Network Model combined with Gurobi Optimization



The benchmark actions of this project is the output generated by the Gurobi optimization program on actual RRP. Figure 6.4 represents how different actions were taken based on the RRP. It is clear to see that the Gurobi optimization program managed to generate a strategy of discharging at higher prices and charging at lower prices.

Figure 6.4: Benchmark Actions suggested by Gurobi Optimization in terms of RRP



6.2 Analysis of Results

Three controlled variables experiments with Gurobi optimization were taken to check the performances of Gurobi with the Dense Neural Network Model.

The experiments were done based on data over three months (04/2020 - 06/2020). The controlled variables are time lengths that were used by the Gurobi optimization program, real RRP and predicted RRP are the independent variables, the dependent variable is the profit at the given time. Note that the assumption the battery owner has $2000 on hand at the beginning is made.

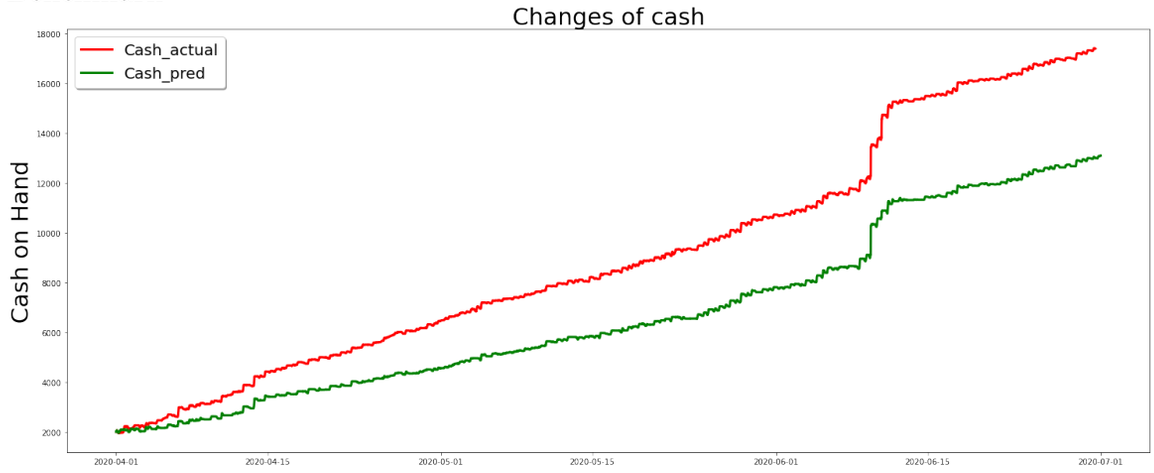
Experiment 1 — twelve hour period with predicted RRP (SARIMA)
Since running the optimization program with the SARIMA model takes too long, which is 285 times slower than the Neural Network model. Hence SARIMA model is not suitable for this project in a timely manner.

Experiment 2 — twelve hour period with actual RRP (Benchmark)
Running the optimization program with the time period 12 hours with actual RRP can make $15383.

Experiment 3 — twelve hour period with predicted RRP(DNN)
Running the optimization program with the time period 12 hours with predicted RRP from Neural Network can make $11086.

Figure 6.5 represents the comparison of change in cash produced by Experiment 2 and Experiment 3. It is clear to see that the optimization program can make around $4000 more profit with actual RRP than predicted RRP. Because the same Gurobi

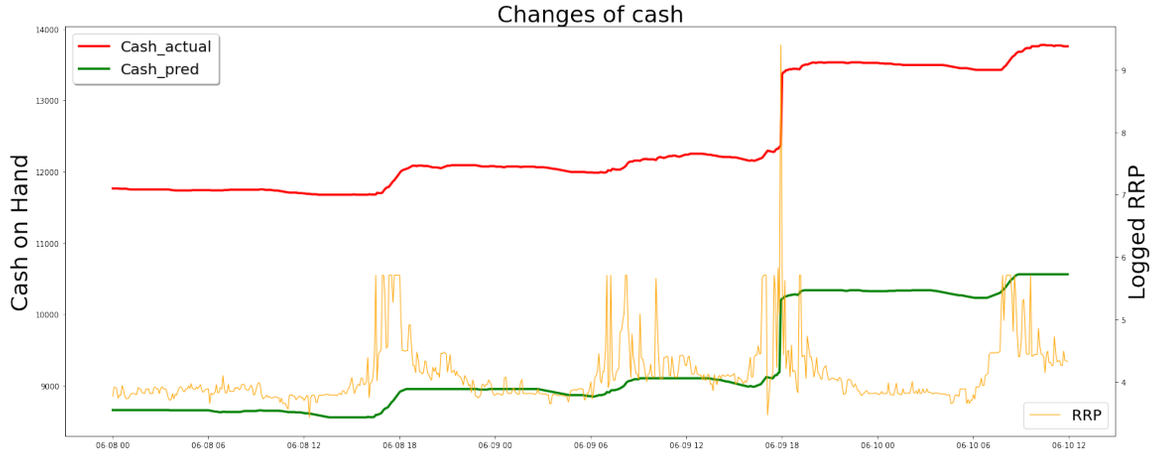
Figure 6.5: Cash Increase over 3 months based on predicted RRP compared with Benchmark



optimization program was used, the gap between these two profits is caused by the slightly less accurate predicted RRP produced by the Neural Network model.

There is a relatively sharp increase in cash on 9th of June 2020. As the zoomed in plot shown in Figure 6.6 that this is caused by an extremely high RRP during that time, and this plot also shows that cash on hand is increased when the RRP reaches the peak times.

Figure 6.6: Influence of RRP on the change of revenue



Chapter 7

Conclusion and Further issues

In conclusion, a forecasting model and optimization program have been built and combined as an algorithm to maximize the profit for the battery owners. As a result of different models analysis, the Neural Network Model provides the best RRP prediction. Moreover, using the predicted RRP into the Gurobi optimization program allows battery owners to make a revenue of $11086, which is 72% revenue compared to the benchmark.

However there are also some limitations in this project. Data out of range from 5 to 500 are considered as outliers during data exploration and SARIMA modelling. This might lead to some bias in the prediction. Moreover, due to limited computer processing power, only the past three weeks' RRP were put in the Neural Network model. That might also reduce the accuracy.

For further improvement, some research could be done to improve the model's prediction accuracy, adding other variables that could related to our RRP like weather, as well as increasing the input length. Besides, future research could also be done using multiple models such as a hybrid LSTM-DNN structure's model and a hybrid GRU-DNN structure's model to see if they can create a model that has a greater true predictive performance than the ones created in this report.

References

[1] E. Australian Government Department of Industry, Science and Resources, "National electricity market." https://www.energy.gov.au/government-priorities/energy-markets/national-electricity-market-nem, 2017.

[2] A. E. Regulator, *State of the energy market 2020*. Energy, Australian Competition and Consumer Commission 2020, 2020.

[3] C. Bouley, "Recurrent neural networks for electricity price prediction." https://towardsdatascience.com/recurrent-neural-networks-for-electricity-price-prediction-a26f8411ea44.

[4] R. W. Jakub Nowotarski, "Computing electricity spot price prediction intervals using quantile regression and forecast averaging," *Computational Statistics*, vol. 30, no. 3, pp. 791–803, 2014.

[5] A. Einstein, "Forecasting spot electricity prices: Deep learning approaches and empirical comparison of traditional algorithms," *Jesus Lago, Fjo De Ridder, Bart De Schutter*, vol. 221, pp. 386–405, 2018.

[6] J. M. Wooldridge, *Introductory econometrics : a modern approach*. Econometrics, Mason, OH : Thomson/South-Western, 2009.

[7] A. M. K.W Hipel, *Time Series Modelling of Water Resources and Environmental Systems*. Elsevier, 1994.

[8] O. V. G. B. Wojciech Zaremba, Ilya Sutskever, "Recurrent neural network regularization." https://arxiv.org/pdf/1409.2329.pdf, 2015.

[9] J. Brownlee, *Long Short-Term Memory Networks With Python: Develop Sequence Prediction Models with Deep Learning*. Machine Learning Mastery, 2017.

[10] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," 2018.

[11] K. P. Shung, "Accuracy, precision, recall or f1." https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9, 2018.

[12] P. Santos, "Tutorial: Linear programming." https://www.gurobi.com/resource/mathematical-programming-tutorial-linear-programming/.

Appendix A

A.1 Data Cleaning

```
import pandas as pd
import csv
import numpy as np
from math import log

dispatchprice_09 = pd.read_csv(wk_dir + 'dispatchprice_fy2009-2019.csv')
dispatchprice = pd.read_csv(wk_dir + 'dispatchprice_fy2020.csv')

dispatchprice = dispatchprice[dispatchprice['INTERVENTION'] == 0]
dispatchprice_09 = dispatchprice_09[dispatchprice_09['INTERVENTION'] == 0]

dispatchprice = dispatchprice[['SETTLEMENTDATE', 'REGIONID', 'RRP']]
dispatchprice_09 = dispatchprice_09[['SETTLEMENTDATE', 'REGIONID', 'RRP']]

dispatchprice = dispatchprice_09.append(dispatchprice)
dispatchprice = dispatchprice.reset_index()
dispatchprice = dispatchprice.drop(columns = ['index'])
del dispatchprice_09

dispatchprice.to_csv(wk_dir + 'cleaned_dispatchprice.csv', index = False)
del dispatchprice


dispatchregionsum_09 = pd.read_csv(wk_dir + 'dispatchregionsum_rrponly_fy2009-2019.csv')
dispatchregionsum = pd.read_csv(wk_dir + 'dispatchregionsum_fy2020.csv')

dispatchregionsum = dispatchregionsum[dispatchregionsum['INTERVENTION'] == 0]
dispatchregionsum_09 = dispatchregionsum_09[dispatchregionsum_09['INTERVENTION'] == 0]

dispatchregionsum = dispatchregionsum[['SETTLEMENTDATE', 'REGIONID', 'TOTALDEMAND']]
dispatchregionsum_09 = dispatchregionsum_09[['SETTLEMENTDATE', 'REGIONID', 'TOTALDEMAND']]

dispatchregionsum = dispatchregionsum_09.append(dispatchregionsum)
dispatchregionsum = dispatchregionsum.reset_index()
```

```
dispatchregionsum = dispatchregionsum.drop(columns = ['index'])
del dispatchregionsum_09

dispatchregionsum.to_csv(wk_dir + 'cleaned_dispatchregionsum.csv',
    index = False)
del dispatchregionsum

dispatchprice = pd.read_csv(wk_dir + 'cleaned_dispatchprice.csv')
dispatchregionsum = pd.read_csv(wk_dir + 'cleaned_dispatchregionsum
    .csv')

data = pd.merge(dispatchprice, dispatchregionsum, how = 'inner', on
    = ['SETTLEMENTDATE', 'REGIONID'])
del dispatchprice
del dispatchregionsum

data = data.drop_duplicates(keep = 'first')

data.to_csv(wk_dir + 'cleaned_data.csv', index = False)
del data

from some_api import plot_the_graph, one_day_time

wk_dir = '/home/zhang/comp/Data3001/data_source/'

data = pd.read_csv(wk_dir + 'cleaned_data.csv')
data['SETTLEMENTDATE'] = pd.to_datetime(data['SETTLEMENTDATE'])

data['time'] = data['SETTLEMENTDATE'].dt.time
data['year'] = data['SETTLEMENTDATE'].dt.year
data['month'] = data['SETTLEMENTDATE'].dt.month
data['day'] = data['SETTLEMENTDATE'].dt.day

data[data['REGIONID'] == 'NSW1']
```

A.2 Data Exploration

```
def avg_plot(region = 'NSW1', year = 0, month = 0, day = 0, x_gap =
    300):
    #select data
    data1 = data[data['REGIONID'] == region]
    if year != 0:
        data1 = data1[data1['SETTLEMENTDATE'].dt.year == year]

    if month != 0:
        data1 = data1[data1['SETTLEMENTDATE'].dt.month == month]

    if day != 0:
        data1 = data1[data1['SETTLEMENTDATE'].dt.day == day]

    data1['SETTLEMENTDATE'] = data1['SETTLEMENTDATE'].dt.hour * 100
    + data1['SETTLEMENTDATE'].dt.minute

    grouped_rrp = data1['RRP'].groupby(data1['SETTLEMENTDATE'])
    rrp = grouped_rrp.mean()

```

```
    grouped_demand = data1['TOTALDEMAND'].groupby(data1['
    SETTLEMENTDATE'])
    demand = grouped_demand.mean()

    time_line = day_time

    plot_the_graph(rrp, demand, time_line, x_gap)

avg_plot(year = 0, month = 2, day = 0)
```

A.3 Linear Model

```
import pandas as pd
import csv
import numpy as np
import datetime

from numpy import mean
from numpy import median
from numpy import std
import matplotlib.pyplot as plt
from scipy.stats import pearsonr
from scipy.optimize import curve_fit
import statsmodels.api as sm
from statsmodels.formula.api import ols
import seaborn as sns
from scipy import stats

from matplotlib.pyplot import MultipleLocator
from math import log

from sklearn import linear_model

data = pd.read_csv('./cleaned_data_with_time.csv')
data['SETTLEMENTDATE'] = pd.to_datetime(data['SETTLEMENTDATE'])
data['weekday'] = (data['SETTLEMENTDATE'].dt.weekday + 1)

data = data[data['REGIONID'] == 'NSW1']

#set dummy
data['month'].astype('category')
data['day'].astype('category')
data['weekday'].astype('category')

#set season variable
#spring 1
#summer 2
#autumn 3
#winter 4
data['season'] = 0
data.loc[(data['month'] >= 9) & (data['month'] <= 11), 'season'] =
    1
data.loc[(data['month'] == 12) | (data['month'] == 1) | (data['
    month'] == 2), 'season'] = 2
data.loc[(data['month'] >= 3) & (data['month'] <= 5), 'season'] = 3
data.loc[(data['month'] >= 6) & (data['month'] <= 8), 'season'] = 4
```

```
data['season'].astype('category')

data

#get previous 5 minutes
data['TOTALDEMAND'] = data['TOTALDEMAND'].shift(+1)
data = data.iloc[1:]
data.loc[(data['month'] == 1) & (data['day'] == 12) & (data['time1'] == 500)]


data_2019 = data[data['year']==2019]
#model 2++
X201 = data_2019[['TOTALDEMAND','month','weekday','time1','season']]
Y201 = data_2019['RRP']

# with sklearn
regr201 = linear_model.LinearRegression()
regr201.fit(X201, Y201)
print('Intercept: \n', regr201.intercept_)
print('Coefficients: \n', regr201.coef_)

# prediction with sklearn
new_month = 1
new_weekday = 7
new_time1 = 500
new_season = 2
new_demand = 5897.17
print ('Predicted RRP: \n', regr201.predict([[new_demand, new_month, new_weekday, new_time1, new_season]]))

# with statsmodels
X201 = sm.add_constant(X201) # adding a constant

model201 = sm.OLS(Y201, X201).fit()
predictions201 = model201.predict(X201)

print_model201 = model201.summary()
print(print_model201)

#create residual vs. predictor plot for 'assists'
fig = plt.figure(figsize=(12,8))
fig = sm.graphics.plot_regress_exog(model201, 'time1', fig=fig)

#remove outliers
data = data[data['RRP'] < 1000]
data

fit = ols('RRP ~ TOTALDEMAND+ C(month)+C(weekday)+C(time1)+C(season)', data=data).fit()
fit.summary()

#create residual vs. predictor plot for 'assists'
fig = plt.figure(figsize=(12,8))
fig = sm.graphics.plot_regress_exog(fit, 'TOTALDEMAND', fig=fig)
```

A.4 SARIMA—R

```
library(readr)
data <- read_csv("D:/DATA3001/data/cleaned_data_with_time.csv")
//choose data of NSW from 2018/07/01 to 2018/07/01 as a test sample
NSW <- subset(data,REGIONID=="NSW1")
ocneday_NSW<-subset(NSW, Year =='2008' & Month =="7"& (Day == 1 |
    Day == 2 | Day == 3 | Day == 4 | Day == 5 | Day == 6 | Day == 7)
    )
oneday_NSW<-oneday_NSW[4]
summary(oneday_NSW)

ts_oneday <- ts(oneday_NSW,frequency = 288) //change data into time
     series format
library('forecast')
arima1<-auto.arima(ts_oneday,trace=T) //use auto arima to figure
    out the parameter

//do the forecast based on parameter from auto.arima and plot it
forecast=forecast(arima1, h=288, level = c(99.5))
plot(forecast)

Box.test(ts_oneday,type="Ljung-Box",lag=6)
Box.test(ts_oneday,type="Ljung-Box",lag=12)
Box.test(ts_oneday,type="Ljung-Box",lag=18)
Box.test(ts_oneday,type="Ljung-Box",lag=24)

```
```

A.5 SARIMA—Python

```
import warnings
import itertools
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import statsmodels.api as sm
import matplotlib
import pmdarima as pm

data = pd.read_csv('./cleaned_data_with_time.csv')
data = data[data['REGIONID']=='NSW1']

df = data[data['year']== 2020]
df = df[['SETTLEMENTDATE', 'RRP']]
df['SETTLEMENTDATE'] = pd.to_datetime(df['SETTLEMENTDATE'])
df.isnull().sum()
df = df.dropna()
df.isnull().sum()

df = df.set_index('SETTLEMENTDATE')
train = df['2020-06-01 00:00:00':'2020-06-08 00:00:00']
train = train['RRP']

#SEASONAL
from pylab import rcParams
```
```

```
rcParams['figure.figsize'] = 18, 8
decomposition = sm.tsa.seasonal_decompose(train, model='additive',
    extrapolate_trend='freq', period=288)
#decomposition = sm.tsa.seasonal_decompose(train, model='
    multiplicative')
fig = decomposition.plot()
plt.show()

from statsmodels.tsa.stattools import adfuller

#ADF: if the p-value is less than the critical value(0.05), the
    series is stationary

def adf_test(timeseries):
    #Perform Dickey-Fuller test:
    print ('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-
    value','#Lags Used','Number of Observations Used'])
    for key,value in dftest[4].items():
       dfoutput['Critical Value (%s)'%key] = value
    print (dfoutput)

print(adf_test(train))

#difference

diff = train - train.shift(1)
diff = diff.dropna()
diff.plot()

print(adf_test(diff))
#p-value < 0.05
#good

# SARIMA example
#from statsmodels.tsa.statespace.sarimax import SARIMAX
mod = sm.tsa.arima.ARIMA(diff, order=(5, 1, 2), seasonal_order=(0,
    1, 1, 288))
res = mod.fit(method='innovations_mle', low_memory=True, cov_type='
    none')

print(res.summary())

test = df['2020-06-08 00:05:00':'2020-06-09 00:00:00']
test = test['RRP']
#test

#test = df['2019-01-08 00:00:00':'2019-01-09 00:00:00']
diffed_test = test - test.shift(1)
#diffed_test.dropnah()
#diffed_test

start_index = test.index.min()
end_index = test.index.max()

```

```
#Predictions
#predictions = model_fit.forecast(steps=283, exog = test)
predictions = res.predict(start=(train.shape[0]), end=(train.shape
    [0] + 288))

predictions = predictions.reset_index()
#predictions

prediction = predictions[0]
pred_sum = prediction.cumsum()
pred = pred_sum.fillna(0) + 63.99995
#pred

pt_testy = test
pt_testy = pt_testy.reset_index()
#pt_testy

plt.plot(pred, color = 'orange')
plt.plot(pt_testy['RRP'], color = 'blue')

plt.xticks(())
plt.yticks(())

plt.show()
```

A.6 Neural Network

```
import sys
assert sys.version_info >= (3, 5)

import sklearn
assert sklearn.__version__ >= "0.20"


# TensorFlow  2 .0 is required
import tensorflow as tf
from tensorflow import keras
assert tf.__version__ >= "2.0"


# Common imports
import numpy as np
import os

np.random.seed(42)
tf.random.set_seed(42)

# To plot pretty figures
# %matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)

from google.colab import drive
```

```
import pandas as pd
import seaborn as sns
from keras import Sequential
from keras.layers import Dense, LSTM
from keras.callbacks import EarlyStopping

from sklearn.preprocessing import StandardScaler, MinMaxScaler
import tensorflow.keras.mixed_precision as mixed_precision

scaler = MinMaxScaler()

drive.mount('/content/drive')
path = '/content/drive/My Drive/DATA3001/cleaned_data.csv'
data = pd.read_csv(path)

data['SETTLEMENTDATE'] = pd.to_datetime(data['SETTLEMENTDATE'])
data['weekday'] = (data['SETTLEMENTDATE'].dt.weekday + 1)

data = data[data['REGIONID'] == 'NSW1']
data = data[data['SETTLEMENTDATE'].dt.year >= 2016]
data.reset_index(inplace=True)
data

data = data[['RRP']]

RRP_MAX = 500
RRP_MIN = 0
data.RRP[data.RRP > RRP_MAX] = RRP_MAX
data.RRP[data.RRP < RRP_MIN] = RRP_MIN

values = data.values

groups = [0]
i = 1

# plot each column
plt.figure(figsize = (25, 10))
for group in groups:
  plt.subplot(len(groups), 1, i)
  plt.plot(values[:, group])
  plt.title(data.columns[group], y=0.5, loc='right')
  i += 1
plt.show()

n_steps = 288 * 4
pred_time = 12 * 12
n = 144
k = 8

data1 = data['RRP']
data1[:5]

data

############### GetNewData1 ###############

```

```
def getNewData1(data,n,k):
  wk = k-3
  S1_NewData = []
  for i in range(0, int(len(data))-int(288*7*(k/2-1))-int(2*n)):
    index = i
    M = []
    for j in range(k):
      M.append(data[index: index + n])
      if (j==k-3):
        pre = index + 288 * 7 - 2*n
        M.append(data[pre: pre + n])

      if (j%2==0):
        index = index + n
      else:
        index = index + 288 * 7 - n

    NEW = np.array(M).T
    S1_NewData.append(NEW)

  S1_NewData = np.array(S1_NewData)
  print(S1_NewData.shape)

  new = []

  new.append((S1_NewData[:,:,0] + S1_NewData[:,:,2] + S1_NewData
    [:,:,4])/3)
  new.append((S1_NewData[:,:,1] + S1_NewData[:,:,3] + S1_NewData
    [:,:,5])/3)

  new.append(S1_NewData[:,:,6])
  new.append(S1_NewData[:,:,7])
  new.append(S1_NewData[:,:,8])

  new = np.array(new)
  new = np.transpose(new, axes=(1, 2, 0))

  return new

neww = getNewData1(data1,n,8)
neww.shape


X_train, Y_train, X_test, Y_test, X_valid, Y_valid = [], [], [],
    [], [], []

new = neww

k=5

for i in range(0, int(len(new) * 0.6)):
    X_train.append(new[i][:,0:k-1])
    Y_train.append(new[i][:,k-1:k])

for i in range(int(len(new) * 0.6), int(len(new) * 0.8)):
    X_valid.append(new[i][:,0:k-1])
```

```
    Y_valid.append(new[i][:,k-1:k])

for i in range(int(len(new) * 0.8), len(new) - n_steps - pred_time):
    X_test.append(new[i][:,0:k-1])
    Y_test.append(new[i][:,k-1:k])

X_train, Y_train, X_test, Y_test, X_valid, Y_valid = np.array(X_train), np.array(Y_train), np.array(X_test), np.array(Y_test), np.array(X_valid), np.array(Y_valid)

"""// Data clean end //"""

############ DNN ############

np.random.seed(42)
tf.random.set_seed(42)

size = 64*20
epoch = 15
activation = 'relu'
model = Sequential()
model.add(Dense(input_shape=[None, 4], units=288, activation=activation))
model.add(Dense(units=188, activation=activation))
model.add(Dense(units=1, activation=activation))
model.compile(loss='logcosh', optimizer="adam")
values = model.fit(X_train, Y_train, batch_size=size, epochs=epoch, validation_data=(X_valid, Y_valid))
model.summary()

def plot_learning_curves(loss, val_loss):
    plt.plot(np.arange(len(loss)) + 0.5, loss, "b-", label="Training loss")
    plt.plot(np.arange(len(val_loss)) + 1, val_loss, "r-", label="Validation loss")
    plt.gca().xaxis.set_major_locator(mpl.ticker.MaxNLocator(integer=True))
    #plt.axis([0, 15, 10, 30])
    plt.legend(fontsize=14)
    plt.title("Plot of Training Loss vs. Validation Loss")

    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.grid(True)

plot_learning_curves(values.history["loss"], values.history["val_loss"])
plt.show()

#func 4 k=8
Y_pred = model.predict(X_test)
Y_pred.shape

def plot_pred_curves(pred, test):
    plt.figure(figsize = (20, 8))
```

```
    plt.plot(np.arange(len(pred)),pred,  color = 'orange', label = "Predicted RRP")
    plt.plot(np.arange(len(test)), test, color = 'purple', label = "Actual RRP")
    plt.legend(fontsize=14)
    plt.title("Predicted RRP vs. Actual RRP")
    plt.xlabel("Prediction for 15/07/2019")
    plt.ylabel("RRP in $AUD")
    plt.show()

plot_pred_curves(Y_pred[2666], Y_test[2666])
plt.show()

################ F1 score ###############

from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import f1_score
Y_pred2 = Y_pred[:,0]
Y_test2 = Y_test[:,0]
Y_pred2 = np.array(Y_pred2).T
Y_test2 = np.array(Y_test2).T
updated_Y_pred2 = np.nan_to_num(Y_pred2)
updated_Y_pred2 = np.squeeze(updated_Y_pred2)

Y_pred3 = updated_Y_pred2.tolist()

updated_Y_test2 = np.squeeze(Y_test2)
Y_test3 = updated_Y_test2.tolist()
Y_test4 = [round(num, 0) for num in Y_test3]
Y_pred4 = [round(num, 0) for num in Y_pred3]

from sklearn.metrics import accuracy_score, f1_score, precision_score
print("F1 Score is:", (f1_score(Y_test4, Y_pred4,average='weighted')))


################### RNN ###################

size = 64*20
epoch = 15

tf.random.set_seed(42)
model6 = keras.models.Sequential([
  keras.layers.SimpleRNN(288, return_sequences=True, input_shape=[None, 4]),
  keras.layers.SimpleRNN(188, return_sequences=True),
  keras.layers.SimpleRNN(1, return_sequences=True)
  ])

model6.compile(loss='mape', optimizer="adam")
values = model6.fit(X_train, Y_train, batch_size=size, epochs=epoch, validation_data=(X_valid, Y_valid))
model6.summary()

```

```
237
238 np.random.seed(42)
239 tf.random.set_seed(42)
240
241 prediction = model6.predict(X_test)
242
243
244 def plot_pred_curves(pred, test):
245     plt.figure(figsize = (20, 8))
246     plt.plot(np.arange(len(pred)),pred,  color = 'orange', label =
    "Predicted RRP")
247     plt.plot(np.arange(len(test)), test, color = 'purple', label =
    "Actual RRP")
248
249     plt.legend(fontsize=14)
250     plt.title("Predicted RRP vs. Actual RRP")
251     plt.xlabel("Prediction for 15/07/2019")
252     plt.ylabel("RRP in $AUD")
253     plt.show()
254
255 plot_pred_curves(prediction[2666], Y_test[2666])
256 plt.show()
257
258 from sklearn.tree import DecisionTreeClassifier
259 from sklearn.naive_bayes import GaussianNB
260 from sklearn.metrics import f1_score
261
262 from sklearn.utils.multiclass import type_of_target
263 type_of_target(prediction)
264 num_classes = len(np.unique(Y_train))
265 prediction = keras.utils.to_categorical(prediction, num_classes)
266 Y_test = keras.utils.to_categorical(Y_test, num_classes)
267 type_of_target(prediction)
268
269 score = f1_score(prediction, Y_test)
270
271 ################### LSTM ###################
272
273 start_lr = 0.00001
274 min_lr = 0.00001
275 max_lr = 0.00005 * tpu_strategy.num_replicas_in_sync
276 rampup_epochs = 5
277 sustain_epochs = 0
278 exp_decay = .8
279
280 def lrfn(epoch):
281   if epoch < rampup_epochs:
282     return (max_lr - start_lr)/rampup_epochs * epoch + start_lr
283   elif epoch < rampup_epochs + sustain_epochs:
284     return max_lr
285   else:
286     return (max_lr - min_lr) * exp_decay**(epoch-rampup_epochs-
    sustain_epochs) + min_lr
287
288 lr_callback = tf.keras.callbacks.LearningRateScheduler(lambda epoch
    : lrfn(epoch), verbose=True)
```

```
289
290
291 def create_model(X_train):
292   model = Sequential()
293   model.add(LSTM(units=288, return_sequences=True, input_dim=
       X_train.shape[-1], input_length=X_train.shape[1]))
294   model.add(LSTM(units=188, return_sequences=True))
295   model.add(LSTM(units=1))
296   model.compile(optimizer='adam',loss="logcosh")
297   return model
298
299 model = create_model(X_train)
300 model.summary()
301
302 epochs = 15
303 batch_size = 64*20
304 history = model.fit(X_train, Y_train, validation_split = 0.3,
       epochs=epochs, batch_size=batch_size, callbacks=[lr_callback],
       verbose=1)
305
306 prediction = model.predict(X_test)
307
308 plt.figure(figsize = (20, 8))
309 plt.plot(prediction[2666], color = "orange", label="Predicted RRP")
310 plt.plot(Y_test[2666], color = "purple", label="Actual RRP")
311
312 plt.title("Predicted RRP vs. Actual RRP")
313 plt.xlabel("Prediction for 15/07/2019")
314 plt.ylabel("RRP in $AUD")
315
316 plt.show()
317
318 from sklearn.metrics import f1_score
319 Y_test2 = Y_test[:288*7,-1]
320 prediction2 = prediction[:288*7]
321
322 Y_test2 = np.array(Y_test2)
323 prediction2 = np.array(prediction2)
324 f1_score(Y_test3, prediction3, average='macro')
325
326 Y_test2.shape
327 Y_test3 = Y_test2.ravel()
328
329 Y_test3.shape
330
331 prediction2.shape
332
333 prediction3 = prediction2.ravel()
334
335 prediction3.shape
336
337 #func 4 k=6
338 Y_pred = model.predict(X_test)
339 Y_pred.shape
340
341 plt.figure(figsize = (20, 8))
```

```
plt.plot(Y_pred[:1000,-1], color = "orange")
plt.plot(Y_test[:1000,-1], color = "purple")



#func 3
Y_pred = model.predict(X_test)
Y_pred.shape

Y_test.shape

plt.figure(figsize = (20, 8))
plt.plot(Y_pred[:1000,-1], color = "orange")
plt.plot(Y_test[:1000,-1], color = "purple")

#END
```

A.7 Optimization

```
import matplotlib.pyplot as plt
from matplotlib.pyplot import MultipleLocator
import pandas as pd
import numpy as np
import gurobipy

def optimization(price_list, battery = 1, wallet = 20, price = 25, len=10, efficiency = 0.81):
    # build a model
    c = price_list
    p = np.tri(len).tolist()
    input_worth = price * battery + wallet

    MODEL = gurobipy.Model("make_money")
    #MODEL.setParam('Threads', 16)
    MODEL.setParam('OutputFlag', 0)

    # add var
    x = MODEL.addVars(len, lb=-1/12, ub=1/12, name='x')
    charge = MODEL.addVars(len, lb=-1/12, ub=1/12, name='charge')

    # update
    MODEL.update()

    # target func
    battery_money = price * (battery - charge.prod(p[-1]))
    wallet_money = x.prod(c) + wallet
    MODEL.setObjective(battery_money + wallet_money, gurobipy.GRB.MAXIMIZE)

    # add constraints
    #battery constraint
    MODEL.addConstrs(charge.prod(p[i]) <= battery for i in range(len))
    MODEL.addConstrs(charge.prod(p[i]) >= (battery - 2) for i in range(len))

```

```
    for j in range(len):
        MODEL.addGenConstrPWL(x[j], charge[j], [-1, 0, 1], [-
    efficiency, 0, 1])

    # execute
    MODEL.optimize()
    dict = {}
    dict['worth'] = MODEL.ObjVal
    dict['input_worth'] = input_worth
    dict['len'] = len
    dict['battery'] = battery - charge.prod(p[-1]).getValue()
    dict['wallet'] = x.prod(c).getValue() + wallet
    dict['price'] = sum(c)/len


    vars = MODEL.getVars()
    dict['action'] = [vars[i].x for i in range(len)]

    return dict
```

A.8 Experiment 2

```

import gurobipy

import pandas as pd
import csv
import numpy as np
from math import log

import matplotlib.pyplot as plt

from some_api import optimization

wk_dir = 'Your_data_dir'
data = pd.read_csv(wk_dir + 'cleaned_data.csv')
data['SETTLEMENTDATE'] = pd.to_datetime(data['SETTLEMENTDATE'])
data = data[data['REGIONID'] == 'NSW1']
data = data[data['SETTLEMENTDATE'].dt.year == 2020]
data = data[data['SETTLEMENTDATE'].dt.month >= 3]

data = data.reset_index().drop(['index'], axis=1)

list1 = data['RRP'].loc[0:12 * 100]
list1 = list1.tolist()
op_output = optimization(list1, battery = 1, wallet = 2000, price =
    65, len=len(list1), efficiency = 0.81)

list_actions = op_output['action']
rrp_input = 80.5
price = 50
wallet = 2000
battery = 1
period = 12 * 3
historical = data['RRP'].loc[288 * 10 - 143 : 288 * 31].values.
    tolist()
```

```
prediction = data['RRP'].loc[288 * 10 - 143 : 288 * 31].values.tolist()

def redo_prediction(dict_in):
    op_list = dict_in['historical']
    op_output = optimization(op_list, battery = dict_in['battery'], wallet = dict_in['wallet'], price = dict_in['price'], len=len(op_list), efficiency = dict_in['efficiency'])
    dict_out = dict_in
    dict_out['actions'] = op_output['action']
    dict_out['charge'] = op_output['action'][0]

    if dict_out['charge'] < 0:
        charge_in = dict_in['efficiency'] * dict_out['charge']
        battery_room = 2 - dict_in['battery']
        if abs(charge_in) < battery_room:
            dict_out['battery'] -= charge_in
        else:
            dict_out['battery'] = 2
            dict_out['charge'] = -battery_room
    else:
        battery_room = dict_in['battery']
        if dict_out['charge'] > battery_room:
            dict_out['charge'] = battery_room
            dict_out['battery'] = 0
        else:
            dict_out['battery'] -= dict_in['charge']

    dict_out['wallet'] += dict_in['historical'][0] * dict_out['charge']

    if dict_in['historical'][0] > 150:
        dict_out['price'] = dict_in['price'] * 90/100 + 15
    else:
        dict_out['price'] = dict_in['price'] * 90/100 + dict_in['historical'][0] * 10/100

    return dict_out


def update_historical(list_his, rrp_input):
    list_his.pop(0)
    list_his.append(rrp_input)
    return list_his

def give_output(dict_in):
    dict_return = redo_prediction(dict_in)
    dict_in['historical'] = update_historical(dict_in['historical'], dict_in['rrp_input'])
    return dict_return

l = []
l1 = []
l2 = []
l3 = []
dict = {}
```

```
dict['efficiency'] = 0.81
dict['actions'] = list_actions
dict['rrp_input'] = rrp_input
dict['historical'] = data['RRP'].loc[288 * 31 : 288 * 31 + 144].
    values.tolist()
dict['battery'] = 1
dict['wallet'] = 2000
dict['price'] = 43.5
dict['charge'] = 0
for i in data['RRP'].loc[288 * 31 + 144: 288 * 122].tolist():#122
    dict['rrp_input'] = i
    dict = give_output(dict)
    l.append(dict['charge'])
    l1.append(i)
    l2.append(dict['wallet'])
    l3.append(dict['battery'])
```

A.9 Experiment 3

```


import gurobipy

import pandas as pd
import csv
import numpy as np
from math import log

import matplotlib.pyplot as plt
s
from some_api import optimization

import tensorflow as tf
import keras

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

wk_dir = 'Your_data_dir'

data = pd.read_csv(wk_dir + 'cleaned_data.csv')
data['SETTLEMENTDATE'] = pd.to_datetime(data['SETTLEMENTDATE'])
data = data[data['REGIONID'] == 'NSW1']
data = data[data['SETTLEMENTDATE'].dt.year == 2020]
data = data[data['SETTLEMENTDATE'].dt.month >= 3]
data = data.reset_index().drop(['index'], axis=1)
list1 = data['RRP'].loc[0:12 * 100]
list1 = list1.tolist()

op_output = optimization(list1, battery = 1, wallet = 2000, price =
    65, len=len(list1), efficiency = 0.81)

list_actions = op_output['action']
rrp_input = 80.5
price = 50
wallet = 2000
```

```
battery = 1
period = 12 * 3
historical = data['RRP'].loc[288 * 10 - 143 : 288 * 31].values.tolist()
prediction = data['RRP'].loc[288 * 10 - 143 : 288 * 31].values.tolist()

def redo_prediction(dict_in):
    model_input = getNewData12(dict_in['historical'], 144, 8)
    pred_rrp = model.predict(model_input)

    op_list = np.transpose(pred_rrp).tolist()[0]
    op_list.insert(0, dict_in['rrp_input'])
    op_output = optimization(op_list, battery = dict_in['battery'], wallet = dict_in['wallet'], price = dict_in['price'], len=len(op_list), efficiency = dict_in['efficiency'])
    dict_out = dict_in
    dict_out['prediction'] = pred_rrp
    dict_out['actions'] = op_output['action']
    dict_out['charge'] = op_output['action'][0]

    if dict_out['charge'] < 0:
        charge_in = dict_in['efficiency'] * dict_out['charge']
        battery_room = 2 - dict_in['battery']
        if abs(charge_in) < battery_room:
            dict_out['battery'] -= charge_in
        else:
            dict_out['battery'] = 2
            dict_out['charge'] = -battery_room
    else:
        battery_room = dict_in['battery']
        if dict_out['charge'] > battery_room:
            dict_out['charge'] = battery_room
            dict_out['battery'] = 0
        else:
            dict_out['battery'] -= dict_in['charge']

    dict_out['wallet'] += dict_in['rrp_input'] * dict_out['charge']

    if dict_in['rrp_input'] > 150:
        dict_out['price'] = dict_in['price'] * 90/100 + 15
    else:
        dict_out['price'] = dict_in['price'] * 90/100 + dict_in['rrp_input'] * 10/100

    return dict_out


def update_historical(list_his, rrp_input):
    list_his.pop(0)
    list_his.append(rrp_input)
    return list_his

def give_output(dict_in):
    dict_in['historical'] = update_historical(dict_in['historical'], dict_in['rrp_input'])
```

```
87      return redo_prediction(dict_in)
88
89  l = []
90  l1 = []
91  l2 = []
92  l3 = []
93  l4 = []
94  dict = {}
95  dict['efficiency'] = 0.81
96  dict['prediction'] = data['RRP'].loc[288 * 10 - 143 : 288 * 31].
        values.tolist()
97  dict['actions'] = list_actions
98  dict['rrp_input'] = rrp_input
99  dict['historical'] = data['RRP'].loc[288 * 10 - 143 : 288 * 31].
        values.tolist()
100 dict['battery'] = 1
101 dict['wallet'] = 2000
102 dict['price'] = 60
103 dict['charge'] = 0
104 for i in data['RRP'].loc[288 * 31: 288 * 122].tolist():
105     dict['rrp_input'] = i
106     dict = give_output(dict)
107     l.append(dict['charge'])
108     l1.append(i)
109     l2.append(dict['wallet'])
110     l3.append(dict['battery'])
111     l4.append(dict['prediction'][0,0])
```