THE UNIVERSITY OF SYDNEY

QBUS6840 Predictive Analytics (2019S1)

Group Project (Assignment 2)

Due date: Saturday 25 May 2018

Group Number: 48

Group Members:

470066517

470290259

470251663

470059944

470281888

Introduction

The S&P/ASX200 index is an adjusted and market-capitalization weighted stock market index of stocks listed in Australia. It is a benchmark to measure the performance of equities in Australia. In order to forecast the next five daily and monthly indexes, we explored the datasets information and processed them. After that, some benchmark models are used to initially predict the index, such as the Decomposition model and Exponential Smoothing model. It is also important to use some advanced models to forecast. The ARIMA, Feed-forward neural network model and Recurrent Neural Network model (with LSTM method) are used in this report. Finally, we compared the result of each model and chose the best one to forecast the next five daily and monthly indexes.

Exploratory Data Analysis

1. Overall description of raw data

To understand the dataset, firstly, import raw daily and monthly data. By using codes of reading the data, we can observe that there are dates, open price, high price, low price, close price and adjusted close price and close price is chosen to be the sample and prediction target.

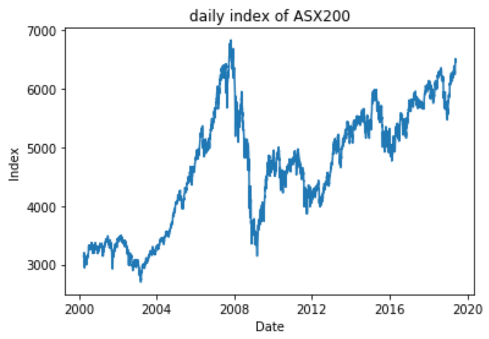
After checking the information of daily and monthly data, there are 42 missing values in daily data. Then we use interpolate function to replace 'NaN' with the average of close price two days before and after the date of the missing value. And the data type of these variables is 'float64'. From the figure below, the mean of daily and monthly data is around 4657 and 4674 respectively and the standard deviation of these two features is huge.

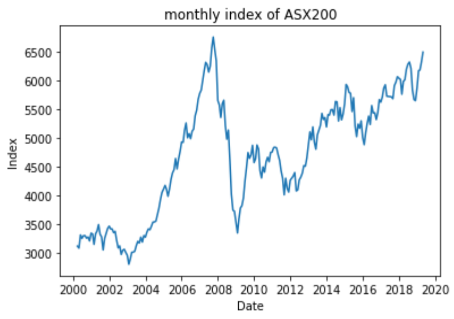
	Daily Close Price	Monthly Close Price
Count	4836.00	230.00
Mean	4657.88	4674.53
Standard Deviation	1017.92	1023.82
Min	2700.40	2800.90
Max	6828.70	6754.10

Description of daily and monthly close price

Patterns and interesting features of data can be discovered through the process of exploratory data analysis; in this way, errors in the modeling process can be reduced to a great extent. The following part shows that some exploratory plots.

2. Price exploratory analysis

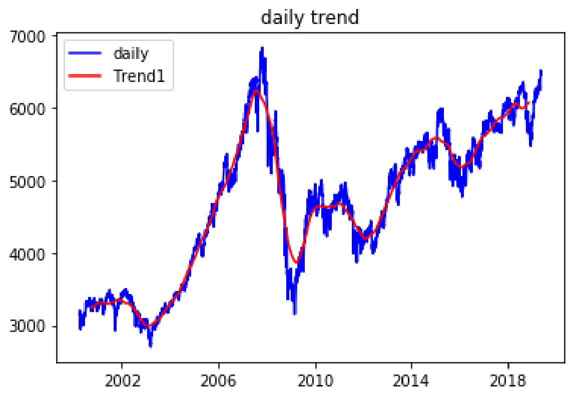




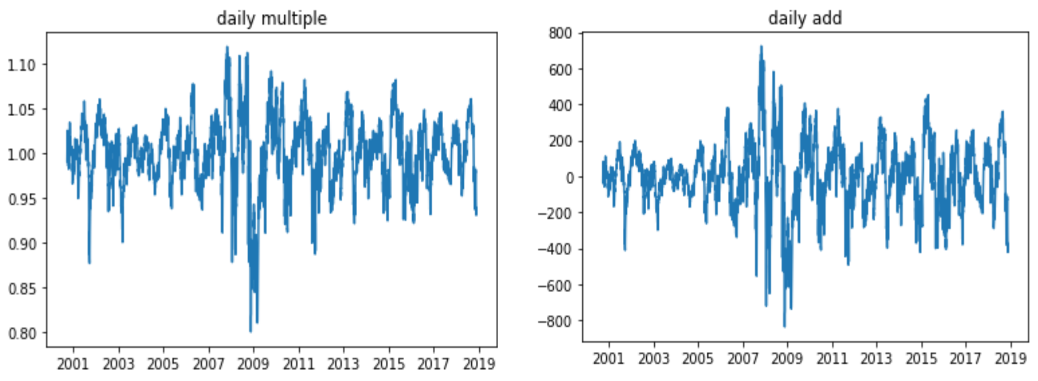
From the figures above, the patterns and trends of monthly and daily data are approximately same. The fluctuation of daily index is more apparent than that of monthly index. The index reached the bottom (about 2700) in 2003 and peaked before 2008 at about 6700 for both daily and monthly index with a significant drop in 2008 largely because of financial crisis. Then the index increased slowly from 2009 to 2019 and reached about 6300 recently. With this long-term trend, we initially decide to choose exponential smoothing, drift and naïve method to be benchmark methods. In the following parts, decomposition is used to detect whether there is a trend in ASX200 index.

The decomposition method could be used to analyze the seasonality of the data. The model assumed the time series is affected by trend, cycle, seasonal variations and irregular fluctuations. There are two forms of decomposition model which are additive model ($y_t = T_t + S_t + C_t + \epsilon_t$) and multiplicative model ($y_t = T_t \times S_t \times C_t \times \epsilon_t$).

In terms of the daily index, there are about 250 working days in a year. Therefore, the CMA-250 could be used for moving average. The plot of the daily index and its trend are shown below.

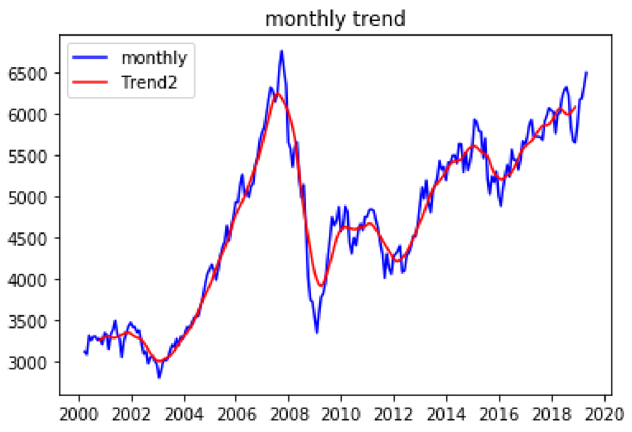


It is hard to observe the magnitude of the seasonal variation, so both the multiplicative and additive models were tried to determine the seasonal index.

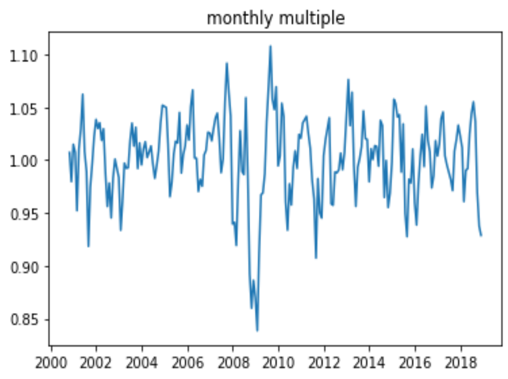


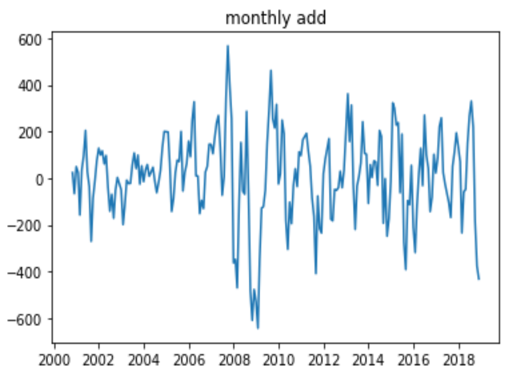
As can be seen from the figure, for the multiplicative model, the seasonal variations fluctuate around 1 and for the additive model, the seasonal variations fluctuate around zero. In addition, we selected the last 4,750 daily indexes and treated them as data for the entire 19 years to calculate the seasonal index. From the numerical point of view, the indexes of the multiplicative model are close to 1, and the indexes of the additive model are close to 0, which implies that the daily index does not have seasonality regardless of multiplicative model or additive model.

Similarly, the trend of monthly data could be received by taking a CMA-12 moving average. The original monthly index and its trend are shown below.



After removing the impact of trend, the seasonal plots. After removing the impact of trend, the seasonal plots could be received as follow. Combined with the values of seasonal indexes, the monthly index also does not have seasonality regardless of multiplicative model or additive model. According to the results, the monthly index also does not have seasonality regardless of multiplicative model or additive model.





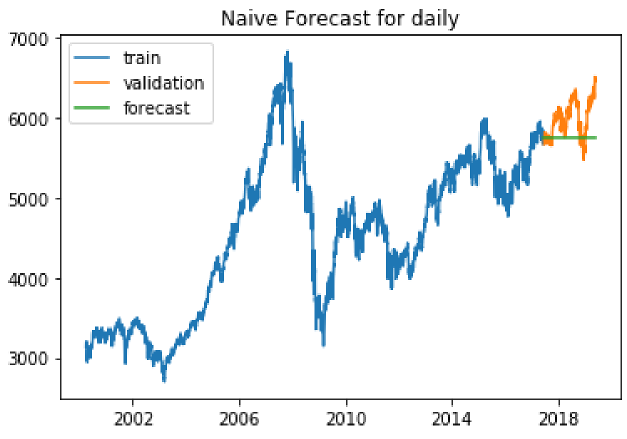
Benchmark Model

Sometimes simple methods generate surprising results. We treat them as a benchmark to measure whether the advanced model is worth using. There are three benchmarks: naïve method, drift method and trend exponential method.

1. Naïve Method

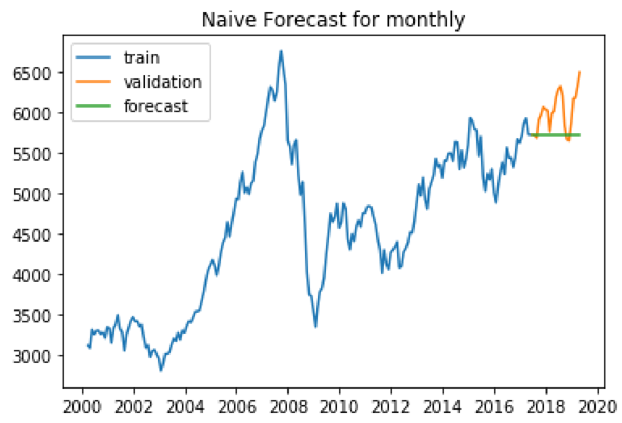
Naïve forecasting method is an estimating technique in which the predicted value of this period is equal to the actual value of the most recent period, without adjusting the data. The formula of the Naïve method is $\hat{Y}_{t+h} = Y_t$.

The model performance for the daily index is shown below.



According to the plot, the fitting result is not good. The forecasting index is much different from the validation set. The mean squared error is about 96,144.21. Under this model, the index for the next five days should be the same as on 24 May 2019, which equals to 6,456.00.

For the monthly index, the performance of Naïve method is shown below.



Similarly, the model could not fit the validation set accurate. The MSE is about 134,668.11. The forecasted index for the next five months is 6491.80.

2. Drift method

2.1. Model descriptions and explanations

The Drift method is a variation of the naïve method. In the drift method, both of the most recent close price and the average change of historical data (drift) are taken into consideration.

The drift method is equivalent to connecting the first and last point of the historical data to draw a line and then extend backward to forecast the new close price. The formula is ：

$$\hat{y}_{t+h} = y_t + \frac{h}{t-1}\sum_{t=2}^{T}(y_t - y_{t-1}) = y_t + h(\frac{y_t - y_1}{t-1})$$

2.2. Motivations

The reason of using the drift method to is that the stock price always fluctuates and cannot be the same as the result predicted by the naïve method. The drift method can capture the information of the trend on the basis of naïve.

2.3. Model fitting and forecasting

Figure 2.3.1 shows the training set, validation set and the close price estimated by the drift method in the daily close price.

Figure 2.3.2 shows the training set, validation set and the close price estimated by the drift method in monthly close price.

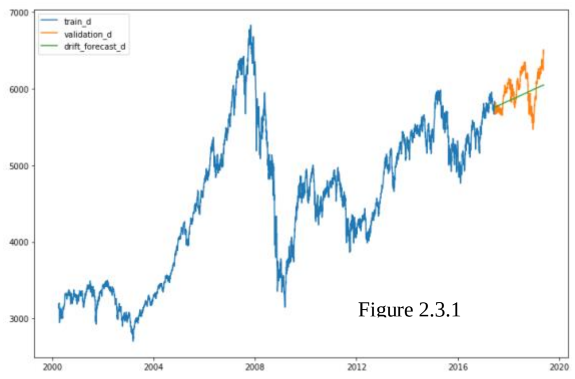


Figure 2.3.1

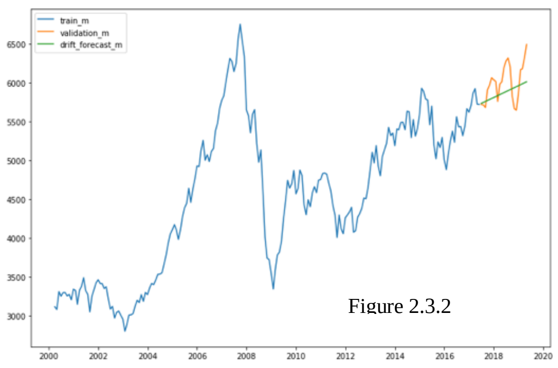


Figure 2.3.2

The MSE for daily close price and monthly close price is 42484.70 and 60228.78.

The forecasted close price of next five days are shown below:

	Forecasting					MSE
Date	2019/5/27	2019/5/28	2019/5/29	2019/5/30	2019/5/31	
Daily data	6456.68	6457.36	6458.04	6458.73	6459.41	42484.70
Monthly data	6506.54	6521.28	6536.02	6550.77	6565.51	60228.78

Table 2.3.1

2.4. Advantages and disadvantages

As a simple model, the drift method is very easy to calculate and easy to understand. Compared to the naïve method, the drift method takes into account the factors of the trend. However, the drift method depends on the first and last known data, ignoring the large amount of data in the middle, which contributes to missing information.

3. Trend corrected exponential smoothing (Holt's linear method)

3.1. Model descriptions and explanations

Exponential smoothing method is a weighted moving average method. The more recent the observation is, the greater the weight. Trend corrected exponential smoothing combines level and trend based on the simple exponential smoothing method.

The Forecast equation in Holt's linear method is:

$$\widehat{y_{t+h|1}}:t = l_t + \mathrm{h}b_t$$

where the level and the trend are:

$$l_t = \alpha y_t + (1-\alpha)\,(l_{t-1} + b_{t-1})$$

$$b_t = \beta\,(l_t - l_{t-1}) + (1-\beta)\,b_{t-1}$$

For one-step forecast, we have h = 1

$$\widehat{y_{t+1|1}}:t = \alpha y_t + (1-\alpha)\,(l_{t-1} + b_{t-1}) + \beta\,(l_t - l_{t-1}) + (1-\beta)\,b_{t-1}$$

For the l_0, b_0, assume:

$$l_0 = y_1$$

$$b_0 = y_2 - y_1$$

$$\text{SSE}\,(\alpha, \beta) = \sum_{t=2}^{n}(y_t - l_{t-1} - b_{t-1})^2$$

According to the formulas, the holts (y, alpha, beta) and SSE (x, y) is defined.

α and β are the smoothing parameters of level and trend respectively.

3.2. Motivations

We performed the decomposition in EDA to analyze its seasonality, and we have used the idea of moving average. In the benchmark, we don't use moving average alone because it can be completely replaced by exponential smoothing, exponential smoothing is weighted moving average and more generalized. By setting the parameters α, β, the model can be optimized, which is also superior to moving average.

3.3. Model fitting and forecasting

The range of α and β are set to (0.1, 1.0.1) for literation. The optimal α, β and MSE for daily close price and monthly close price is 0.9,0.1 and 33082.84.

The optimal α, β and MSE for monthly close price and monthly close price is 0.9,0.1 and 17960.64.

The forecasted close price of next five ahead are shown below:

	Forecasting					MSE
Date	2019/5/27	2019/5/28	2019/5/29	2019/5/30	2019/5/31	
Daily data	6470.89	6480.39	6489.89	6499.39	6508.90	70315.06
Monthly data	6527.50	6577.32	6627.15	6676.97	6726.79	17960.64

3.4. Advantages and disadvantages

The advantage is that the method will considerate all the data. In addition, it is more reasonable and easier to interpret than other simple models.

The shortcoming is that giving a relatively large weight in the short-term, it is more appropriate to make a short-term forecast than the long-term.

In general, in our benchmarks, the drift method is the most accurate in forecasting daily close price, and TCES is the best for monthly.

Advanced Model

ARIMA

1. Model Description

1.1. Introduction

ARIMA is an approach to understand the time series and conduct forecasting. ARIMA integrate AR model and MA model, and it indicates that the current value has a linear relationship with previous value, previous and current residual value. The I indicate that the data has been conducted finite difference before fitting the model to ensure stationarity. The AR and MA indicate the current value has a linear relationship with different items, the details are as follow:

	previous value	previous residual	current residual
AR	✓	×	✓
MA	×	✓	✓
ARIMA	✓	✓	✓

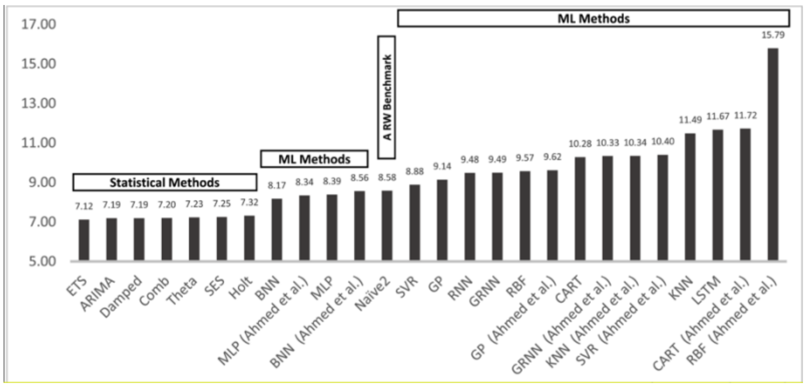
1.2. Model Assumption

ARIMA works on an assumption of stationarity. As ARIMA assume the sample ACF and PACF can convert to population ACF and PACF. Without stationary data, this conversion is impossible to realize. If the data is non-stationary, we should conduct data transformation, such as log transformation and first differencing, to ensure a stationary data.

In addition, for order selection, MA must be invertible and stationary.

1.3. Selection Motivation

According to model comparison, for short-term forecasting, ETS and ARIMA models performed the best in all algorithmic model available to time series. We have conducted ETS above, so in this part, we focus on ARIMA (Sharmistha, 2019)



2. Pre-processing

2.1. Missing value

To keep the continuity of time series, we fill the missing value through interpolation method.

2.2 Validation set split

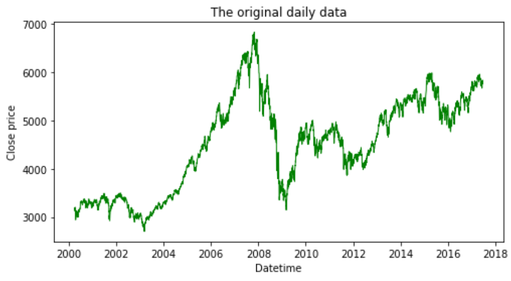
The original data is split into training set and validation set. As we use this data for a short-term forecasting, a large validation set would decrease the accuracy of forecasting. However, an unduly small validation set would generate an overfitting model. After comprehensive consideration, we split 10% of original data as validation set for both daily data and monthly data.

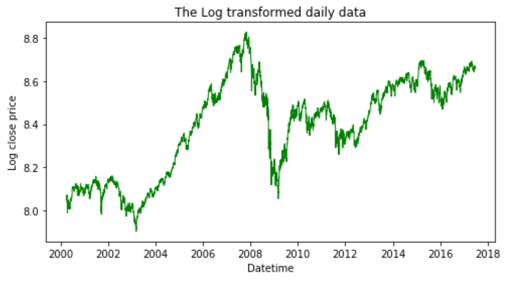
2.3. Data transformation

The purpose of this process is acquiring stationary data

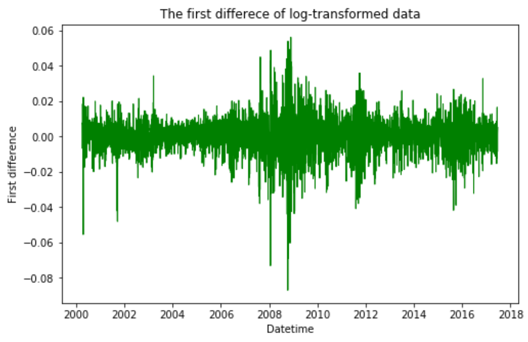
2.3.1. Daily data

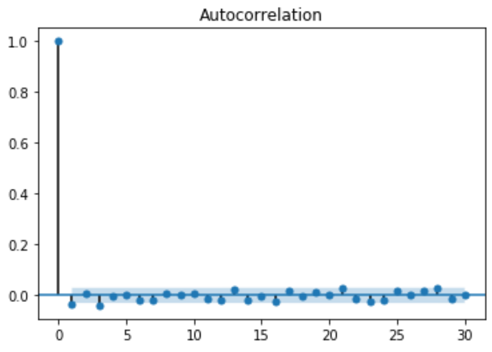
By observing the original data, the data is non-stationary. So, we conduct log-transformation. However, the data is still obviously non-stationary.





So, we conduct first order differencing on the basis of log-transformation. The ACF plot indicates the time series cut off quickly, which is the symbol for stationary series. To further check the stationarity, we also conduct Dickey-Fuller test. The p-value is 0.00 which indicate the data is stationary.

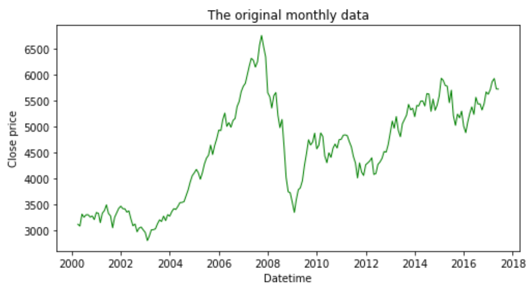


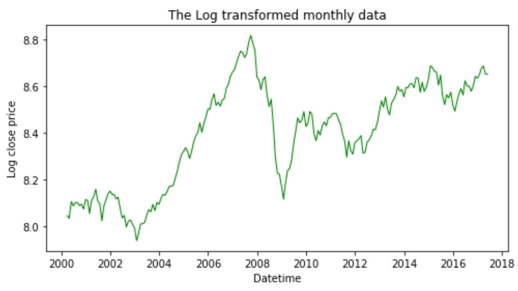


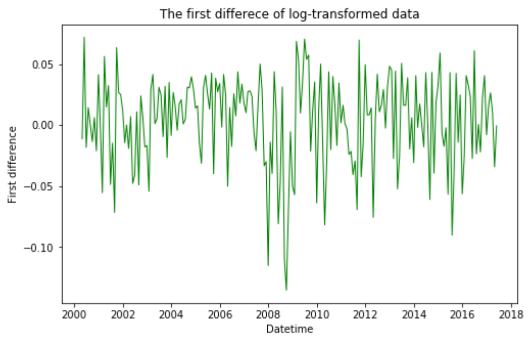
So, the order for I is 1.and p, q can be set according to ACF and PACF, which will be discussed latter.

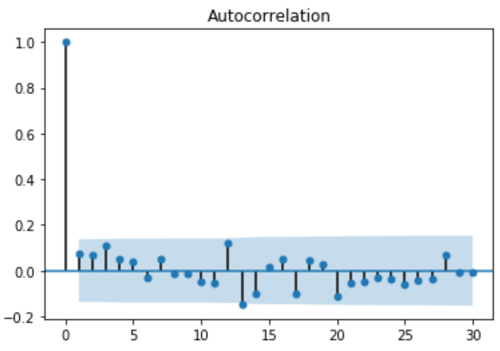
2.3.2. Monthly

Similarly, the original time series is non-stationary. So, we conduct log-transformation and first order differencing. After these transformations, the data seems to be stationary as the ACF plot quickly cut off. For further conformation, we conduct Dickey-Fuller test. The p-value is 0.00 which indicate the data is stationary.









So, the order for I is 1.and p, q can be set according to ACF and PACF, which will be discussed latter.

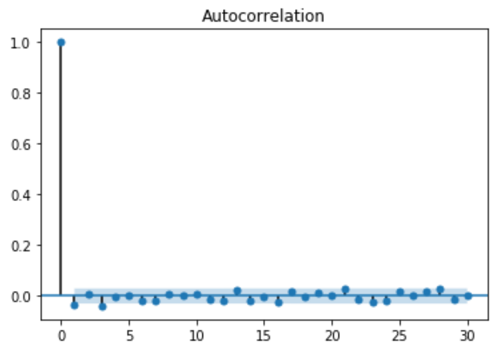
3. ARIMA modeling and analysis

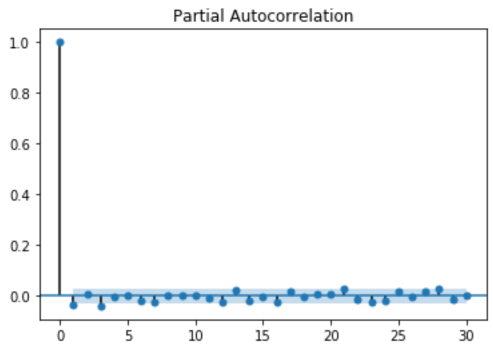
the following we discuss the ARIMA process for daily and monthly data separately.

3.1. Daily Data

3.1.1. Order Selection

We plot ACF and PACF to select the order for MA and AR, respectively. Both plots indicate autocorrelation. However, partial autocorrelation the autocorrelation with the relationships of intervening observations removed. So, ACF plot stands for MA model, and PACF stands for AR model.





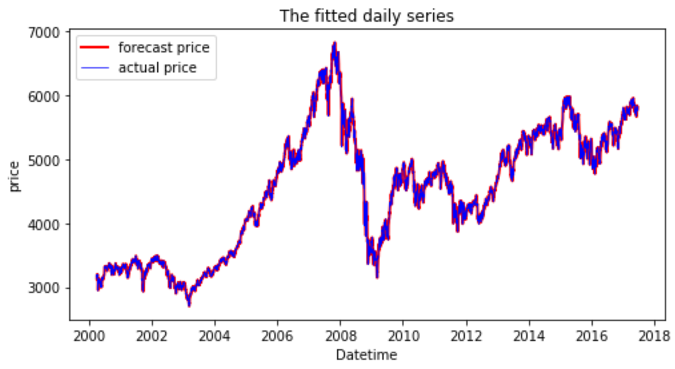
In these plots, both the autocorrelation and partial autocorrelation cut off in lag 3. So, the ARIMA order could be (3, 1, 3). However, it is the order selected artificially, there may be more proper order. Therefore, we use st.arma_order_select_ic code to select the best order automatically according to AIC. We use AIC as criterion to select model rather than MSE, because we choose a lower-than-average validation set ratio, which increases the probability of overfitting, and AIC penalize overfitting.

As an appropriate model tends to have most significant coefficients, highest log-likelihood statistic and lowest AIC. By comparing these factors comprehensively, ARIMA (4,1,3) tends to be the best choice.

	(3,1,3)	**(4,1,3)**
significant coefficients	4	4
log-likelihood	14011.70	14013.60
AIC	-28007.39	-28009.19

3.1.2. Model Fitting

After ARIMA (4,1,3) fit to log-transform data, we can get the fitted series. Notably, we should transform the fitted series into initial form - price. The plot indicates the model fit the series well.



3.1.3. Model check

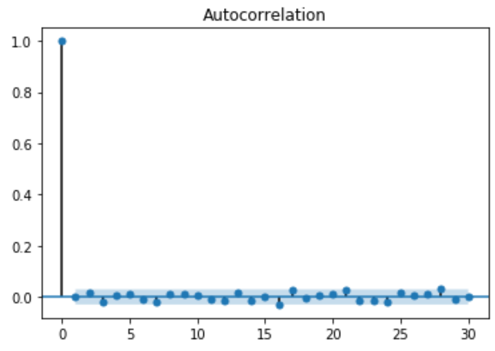
This process focuses on check the assumption of ARIMA and Residual diagnostics.

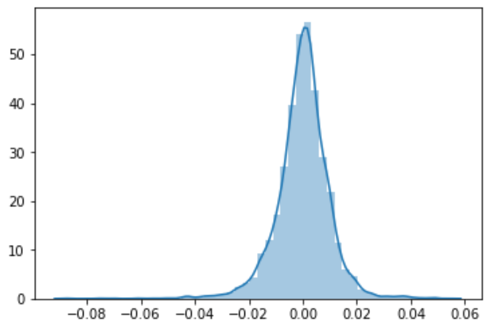
Assumption check

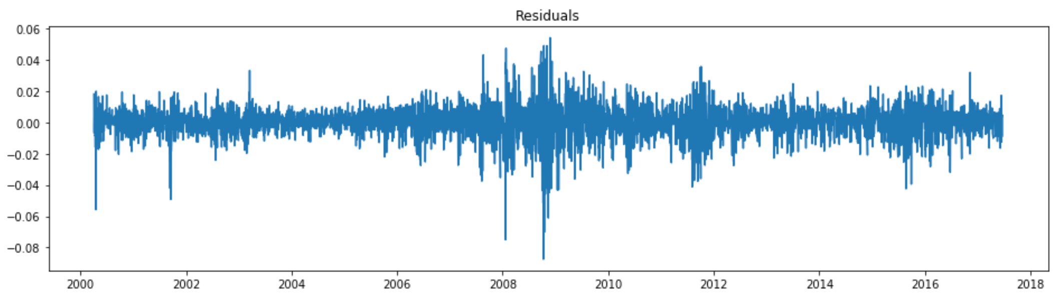
we check the stationarity and invertibility of ARIMA model by using sm.tsa.ArmaProcess().isstationary and sm.tsa.ArmaProcess().isinvertible faction in python. Both results indicate assumptions are complied with.

Residual diagnostics

The residual of a meaningful model should be normally distributed and stationary, namely, the residual should be a white noise.





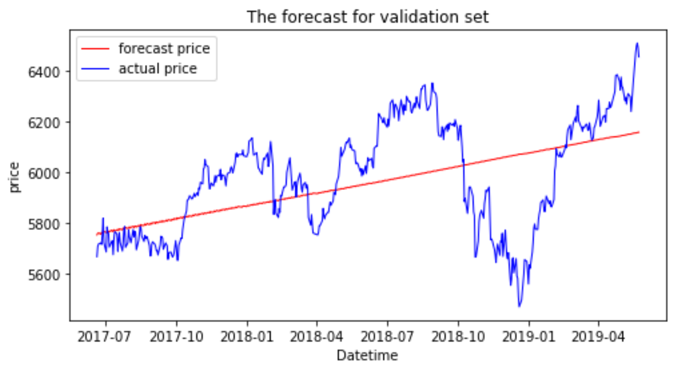


count	mean	std	min	max
4389	0.00	0.01	-0.09	0.05

By observing the above plots, the residual seems to be a white noise. For further conformation, we use Dickey-Fuller and KS test to test the stationarity and normality. Both tests have a p-value of 0.00, so we conform the residual is a white noise, and the ARIMA model is appropriate.

3.1.4. Validation Test

The purpose of this step is calculating the MSE of validation set for ultimate model selection. We can use multiple-step out-of-sample dynamic forecast to get the forecast value. Similarly, these forecasting values should be transformed to initial form. To further select model, we use the forecast value to compare with the real value in validation set and get MSE. MSE can be a criterion to compare different model.



The MSE for validation set is 37248.70, and this result can be compared for further model selection, which will be discussed latter.

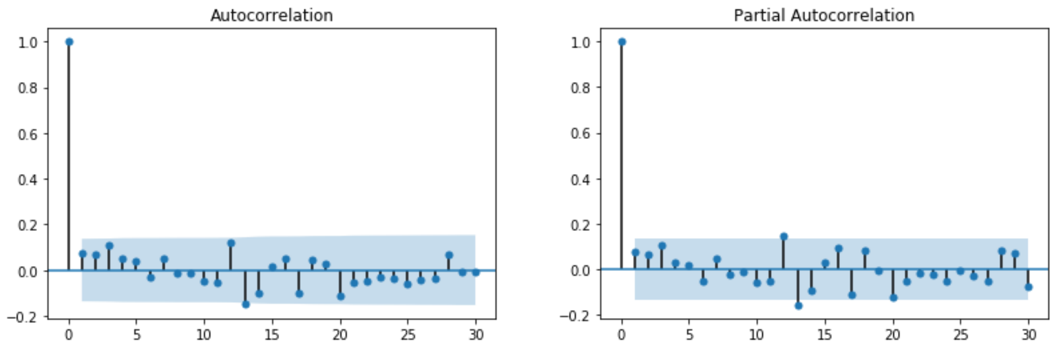
3.1.5. Forecasting

If ARIMA is viewed as the most proper model to forecast, we should model ARIMA by using all the available and conduct 5-step dynamic forecasting. We select the best order of (4,1,3) by using AIC. We have tested the order selection, model assumptions and residuals requirement. All the factors meet requirement. As the words' limitation, we only report the final forecasting as follow:

	Forecasting					MSE
	1	2	3	4	5	
daily data	6463.59	6463.17	6456.62	6465.53	6462.29	37248.70

3.2. Monthly Data

3.2.1. Order Selection

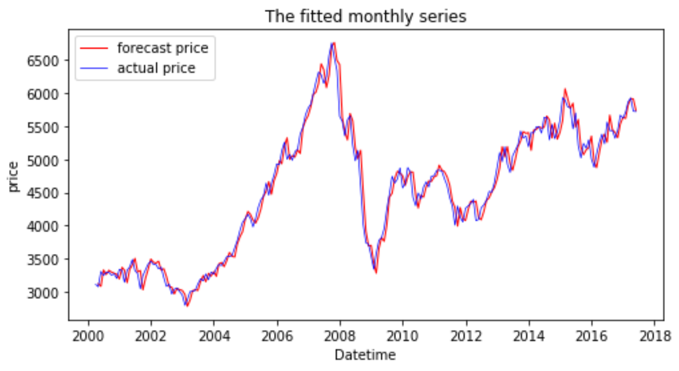


By observing both plots, all the lags seem to fall within significant level after lag 0. So, we select a preliminary tentative order of (0, 1, 0). However, ARIMA (0, 1 ,0) indicates the change of log-price is random walk, which means there is no pattern for time series. So, there must be a more proper order. For a more reasonable selection, we use python to select order based on AIC automatically. The python recommends an order of (5, 1, 2). Comprehensively comparing the statistic factors of two orders, we select (5, 1, 2) as ultimate order.

	(0,1,0)	**(5,1,2)**
significant coefficients	1	5
log-likelihood	384.97	394.012
AIC	-765.94	-770.02

3.2.2. Model Fitting

After ARIMA (5,1,2) fits to log-transform data, we can get the fitted series. Then, we transform them into initial form. the model also fit the original series well.



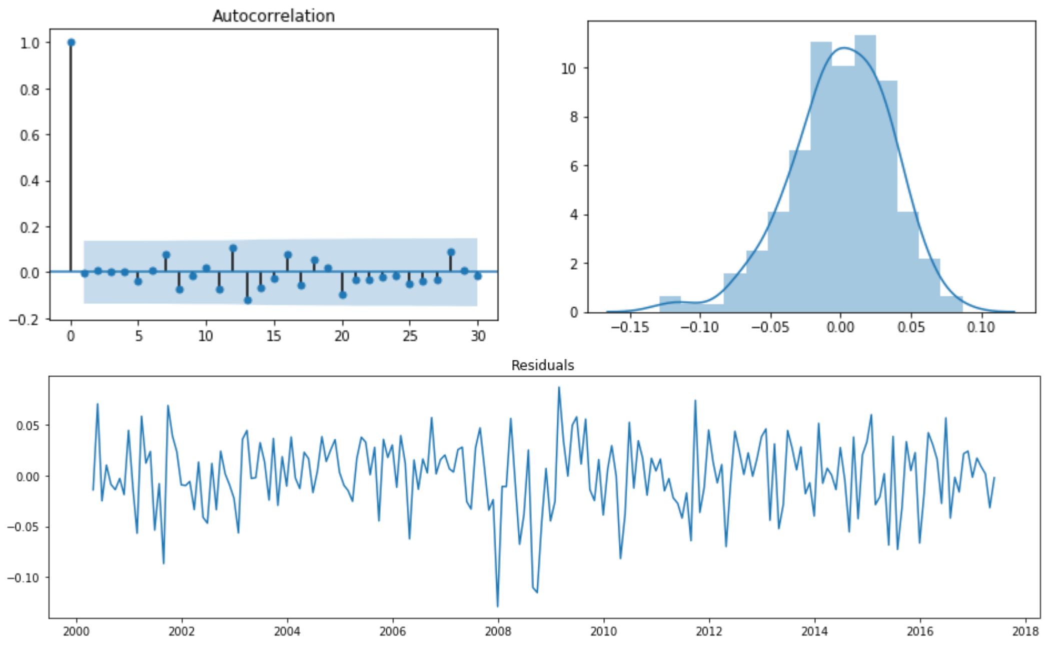
3.2.3. Model check

Assumption check

we check the stationarity and invertibility of ARIMA model by using python function. Both results indicate assumptions are complied with.

Residual diagnostics

Similar as daily data, for an appropriate model, the residual should be approved as a white noise.

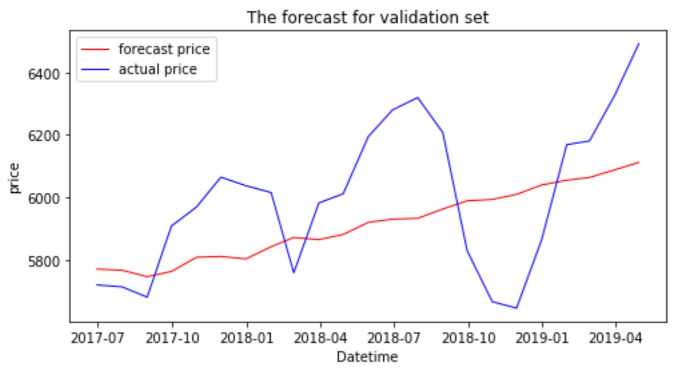


count	mean	std	min	max
206	0.00	0.04	-0.13	0.09

By observing the above plots, the residual seems to be a white noise. For further conformation, we conduct Dickey-Fuller test and KS test to test the stationarity and normality, respectively. Both tests have a p-value of 0.00, so we conform the residual is a white noise, and the ARIMA model is appropriate.

3.2.4. validation

To further select model, we should conduct validation. We use the forecast value to compare with the real value in validation set and get MSE.



The MSE for validation set is 51164.12.

3.2.5. Forecasting

Similar as daily data, after modeling ARIMA using all available data and testing all the assumption and model, we could get the forecast for next 5 months as follow

	Forecasting					MSE
	1	2	3	4	5	
monthly data	6534.20	6623.18	6688.18	6684.35	6662.51	51164.12

4. Advantage & Disadvantage

Advantage: Model is simple and only relies on the endogenous variables.

Disadvantage: Model only captures linear relationship. In addition, model requires stationary data, which might not be applicable for highly volatile data, such as corporate stock price.

5. Conclusion

	Forecasting					MSE
	1	2	3	4	5	
daily data	6463.59	6463.17	6456.62	6465.53	6462.29	37248.70
monthly data	6534.20	6623.18	6688.18	6684.35	6662.51	51164.12

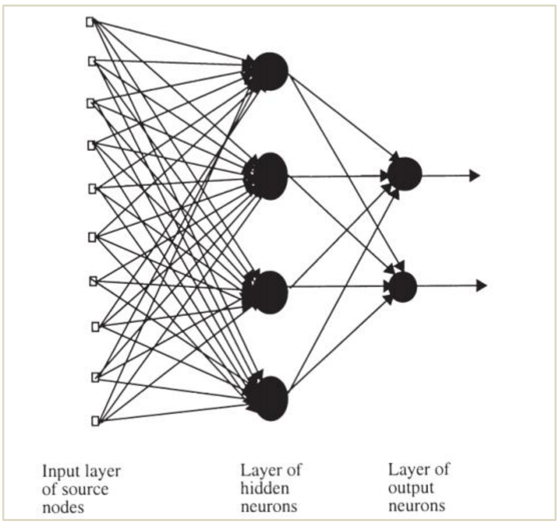
Neural Network model

1. Model Description

1.1. Feed-forward neural network model

1.1.1. Introduction

Neural networks method is originated from simple mathematical models of the brain, which is able to conduct nonlinear relationships between the response and predictors. It includes a network of “neurons” organized in layers, the predictors (or inputs) form the bottom layer, and the forecasts form the top layer. Feed-forward neural network is one of the most common and easiest neural network models, it has intermediate layers with “hidden neurons” where the neurons receive inputs from the previous layers and the output of the node in one layer is the input of the next layer. The model uses a weighted linear combination to combine the inputs of each node and the response are modified by a non-linear function.



1.1.2. Selection Motivation

Neural network model has three main intrinsic capabilities, which is learning from any mapping from input to output, supporting multiple inputs and multiple outputs and extracting patterns from long sequence data (Otexts.com, 2019).

For specific values evaluation such as the stock price, common analytical methods are based on statistics principle, such as the ARMA series (including AR, MA, ARMA, ARIMA). These methods need to satisfy some assumptions (such as stationary assumptions, etc.), but their algorithms are actually severely limited. This is because many scenarios in practice are difficult to satisfy such assumptions. Therefore, this problem is often abstracted into a regression problem. The independent variable is the value in historical time series, while the dependent variable is the value in predicted time. In this case many methods can be used, such as neural networks or even deeper learning methods (LSTM).

1.1.3. Pre-processing

1.1.3.1. Data transformation

Neural networks normally work best with scaled data, especially in the sigmoid or tanh activation function. Therefore **MinMaxScaler()** is used to scale the data to the range of [0,1] with the following formula:

$$scaler = \frac{x - min}{max - min}$$

After scaling the data, we need to transform the dataset into the (n,p) matrix whereas the p is lag, this is for the window size input as the neural network can input multiple data to the hidden layer, for daily data, we use lag =6, this is because for non-seasonal time series, the window size is the optimal number of lags (according to the AIC) for a linear AR(p) model. Based on the previous PACF analysis, we found that that p=6, which means6 day data are related to the new predicted data, therefore we choose 6 as daily data window size. For monthly data, although the lag is 5 in PACF analysis, the greater the window size normally leads to a more precise outcome, but it also doesn't fit for the short term prediction, in this case, we need to make a trade-off , in the end 12 is chosen to be monthly data window size.

1.1.3.2. Validation set split

The scaled data is split into training set and validation set with 10% of original data as validation set for both daily data and monthly data. This is because for short-term forecasting, a large validation set would decrease the accuracy of forecasting. However, an unduly small validation set would generate an overfitting model. After comprehensive consideration, the split method is the same as in ARIMA part.

1.1.4. Feed forward neural network modeling and analysis

In the following part, we discuss the feed forward neural network process for daily and monthly data separately.

1.1.4.1. Daily data

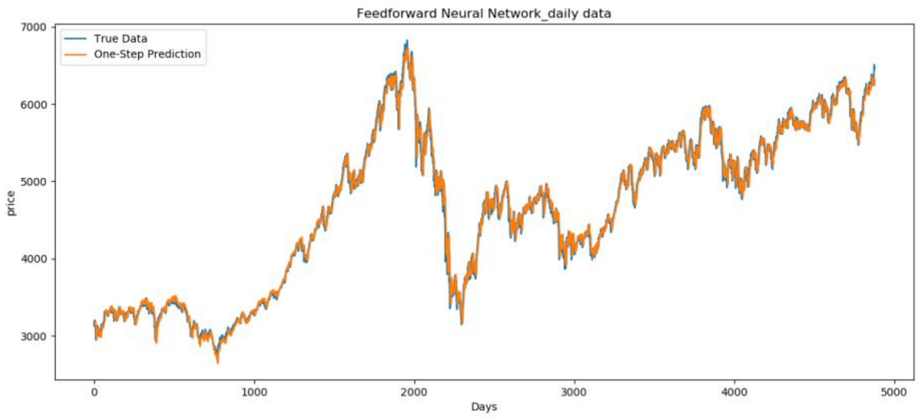
1.1.4.1.1. Parameter tuning

In the feed forward neural network model, a few parameters are essential for model fitting. The first one is the input number, which equals to the window size. The second one is the number of neurons, which also represents the width of the layer, the third one is the batch size, which is the total number of training examples present in a single batch. The fourth one is the number of hidden layers, which is tuned to be single layer, and the last one is the number of epochs, which means the learning frequency. To find the best parameter, we conduct a loop statement to find the best parameter based on the training set MSE, we set the number of neurons from 12 to 20 in 2 steps changes, the number of epochs from 10 to 70 in 10 steps change, the number of batch size in (10,20,40,60,80,100). Below is the best parameter:

Neuron	Epochs	Batch size	Hidden layer
14	30	60	1

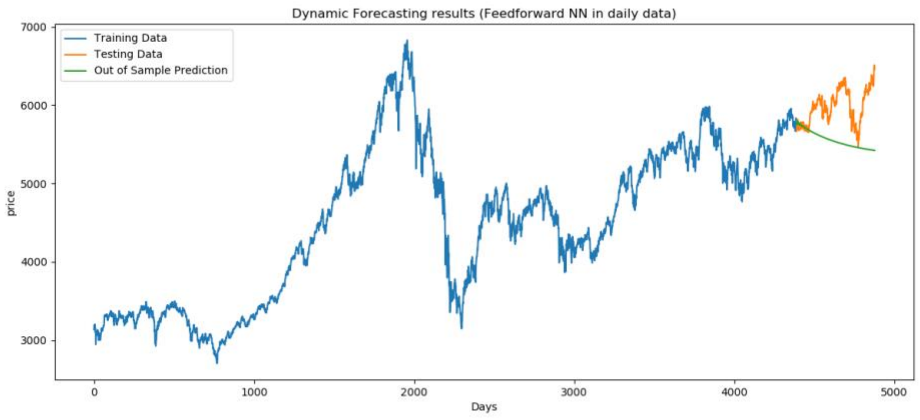
1.1.4.1.2. One step forecast

After fitting the model with the training set, We use one step forecast with whole time series data after data transformation and transform the data into the initial price value, below is the comparison between original piece and forecasted price, we can observe that the fitting level is quite high:



1.1.4.1.3. Validation test

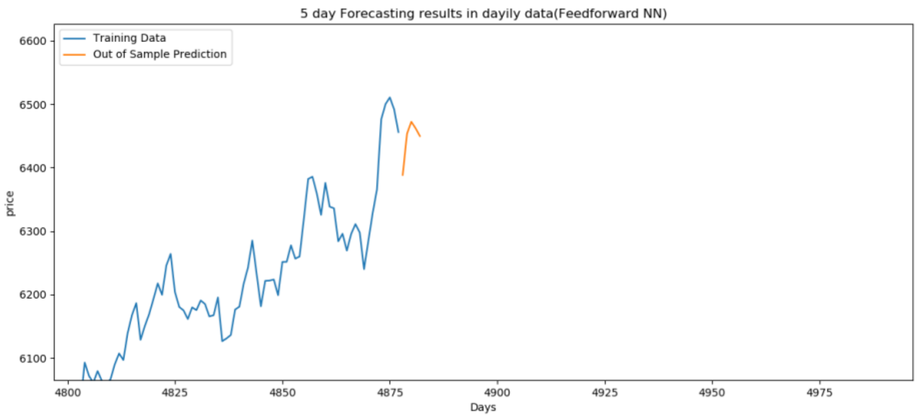
We can use dynamic forecast to get the forecast value. Similarly, these forecasting values should be transformed to initial form. To further select model, we use the forecast value to compare with the real value in validation set and get MSE. Here is the MSE result:



The MSE for validation set is 260806.68, we can observe that there is a downward trend in the forecast data.

1.1.4.1.4. Forecasting

If neural network is viewed as the most proper model to forecast, we should model neural network by using all the available and conduct 5-step dynamic forecasting, here is the final forecasting as follow:



	Forecasting				
	1	2	3	4	5
daily data	6388.38	6453.41	6472.29	6461.96	6449.69

1.1.4.2. Monthly Data

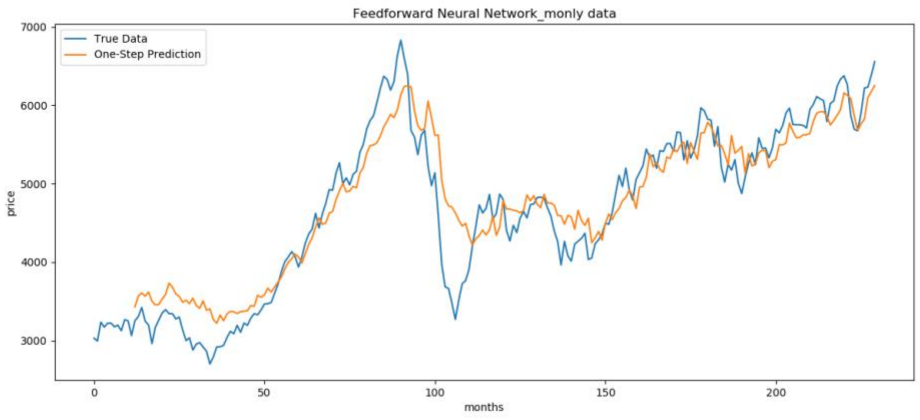
1.1.4.2.1. Parameter tuning

To find the best parameter in monthly data, we also conduct a loop statement based on the training set MSE, we set the number of neurons from 12 to 20 in 2 steps changes, the number of epochs from 10 to 70 in 10 steps change, the number of batch size in (10,20,30,50, 60,70,100). Below is the best parameter:

Neuron	Epochs	Batch size	Hidden layer
20	50	100	1

1.1.4.2.2. One step forecast

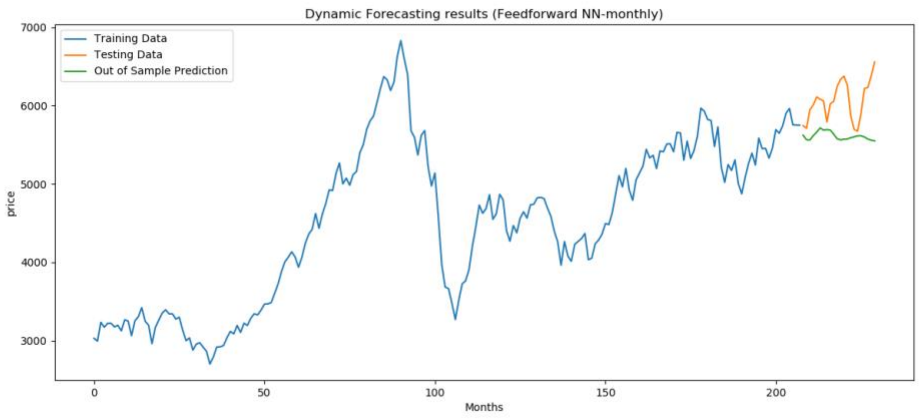
After fitting the model with the training set, we use one step forecast with whole time series data after data transformation and transform the data into the initial price value, below is the comparison between original piece and forecasted price:



We observe that the fitting level is not too high, this may attribute to the low number of monthly data.

1.1.4.2.3. Validation test

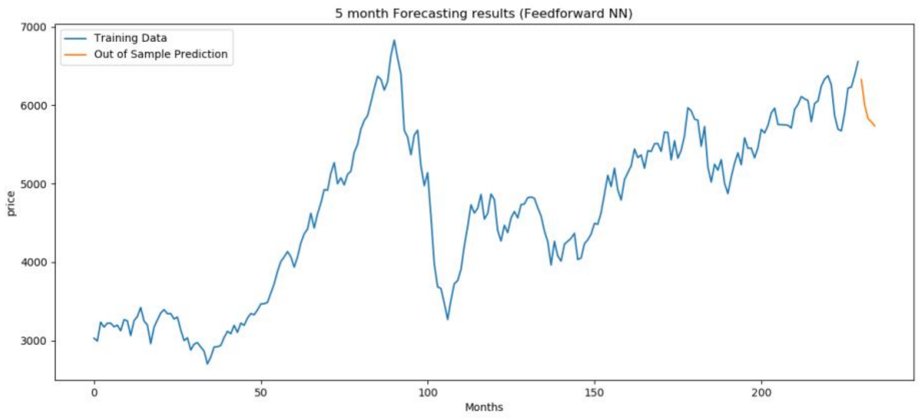
We use dynamic forecast to get the forecast monthly data value and these forecasting values are transformed to initial form. To further select model, we use the forecast value to compare with the real value in validation set and get MSE. Here is the MSE result:



The MSE for validation set is 620410.50, we can observe that the forecasted price is fluctuated but lower than the validation data.

1.1.4.2.4. Forecasting

If neural network is viewed as the most proper model to forecast, we should model neural network by using all the available and conduct 5-step dynamic forecasting, here is the final forecasting as follow:



	Forecasting				
	1	2	3	4	5
monthly data	6328.52	6105.04	5909.68	5709.96	5496.81

1.2. RNN model with long short-term memory

1.2.1. Introduction and Motivation

This feed forward NN model is not optimal for modelling data with time or spatial dependency. This is because each input data feature is independent with each other and the relationship between neurons is also independent ("Recurrent Neural Networks Tutorial, Part 1 – Introduction to RNNs", 2019). The RNN model uses a set of more complex neurons that store state over a sequence of inputs. It accepts sequences of inputs and can produce sequences as outputs or single values. This is ideal for modeling time series data since the input is a set of sequences and we may wish to produce a sequence of outputs or a single output as our prediction. (Tutorial 10, 2019) . However, the vanishing gradient problem has standard RNNs lose its learning long-term dependencies and LSTMs were designed to combat vanishing gradients through a gating mechanism ("A Beginner's Guide to LSTMs and Recurrent Neural Networks", 2019).

1.1.4.3. Daily data

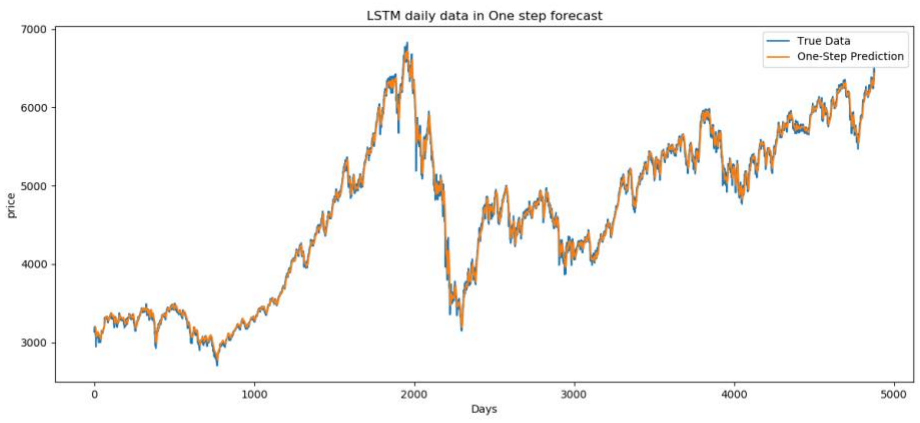
1.1.4.3.1. Parameter tuning

A few parameters are important for RNN model fitting, including the units, which is the dimension of the weight vector inside the LSTM “neurons” or cells., the batch size, the number of epochs. To find the best parameter, we conduct a loop statement to find the best parameter based on the training set MSE, we set the number of unit in 13,14,15, 20, the number of epochs in 10,15, 20,30,40, 50, 60. And the number of batch size is from 60 to140 in 20 steps change. The validation split is set to be = 0.05, early stop patience is set to be 2, below is the best parameter:

Neuron	Epochs	Batch size	Hidden layer
15	15	140	1

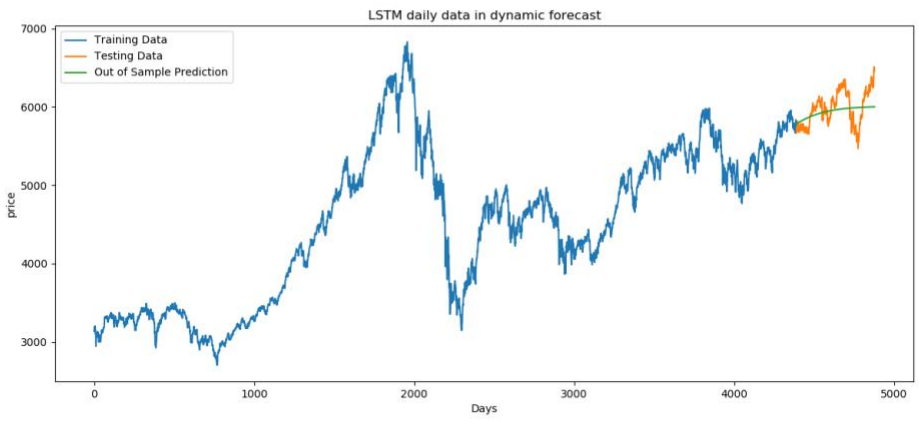
1.1.4.3.2. One step forecast

After fitting the model with the training set, we use one step forecast with whole time series data after data transformation and transform the data into the initial price value, below is the comparison between original piece and forecasted price in LSTM method, We observe that the fitting level is quite high:



1.1.4.3.3. Validation test

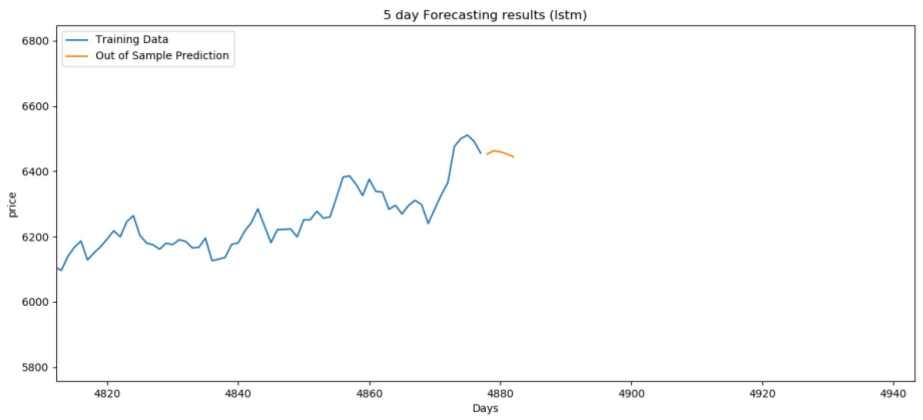
dynamic forecast is used to get the forecast daily data value and these forecasting values are transformed to price form. Here is the MSE result between validation data and forecast data:



The MSE for validation set is 38737.35, we can observe that the forecasted price is quite steady compared with the validation data.

1.1.4.3.4. Forecasting

If LSTM is viewed as the most proper model to forecast, we should model the LSTM by using all the available data and conduct 5-step dynamic forecasting, here is the final forecasting as follow:



	Forecasting				
	1	2	3	4	5
daily data	6452.48	6462.76	6459.5	6452.91	6444.59

1.1.4.4. Monthly Data

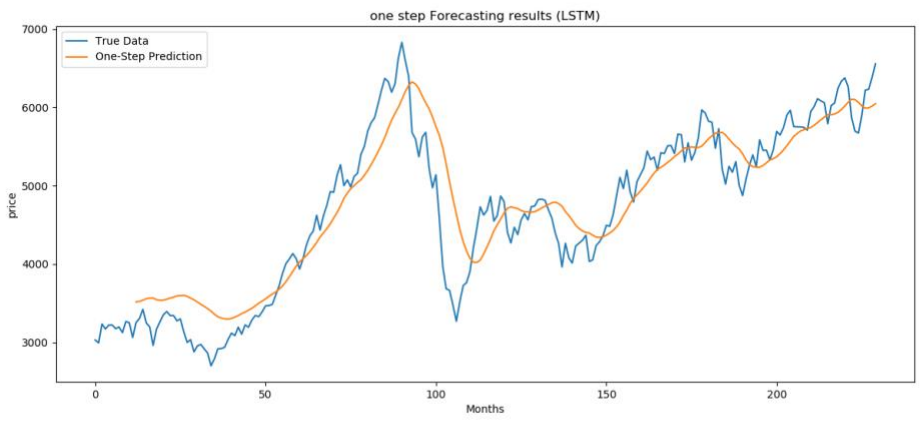
1.1.4.4.1. Parameter tuning

To find the best parameter in monthly data, we conduct a loop statement to find the best parameter based on the training set MSE, we set the number of unit in 15,20,30,40,50,60,70, the number of epochs in 10,15, 20,30,40, 50, 60. And the number of batch size is from 60 to140 in 20 steps change. The validation split is set to be = 0.01, and the early stop patience is set to be 2, below is the best parameter:

Neuron	Epochs	Batch size	Hidden layer
40	10	60	1

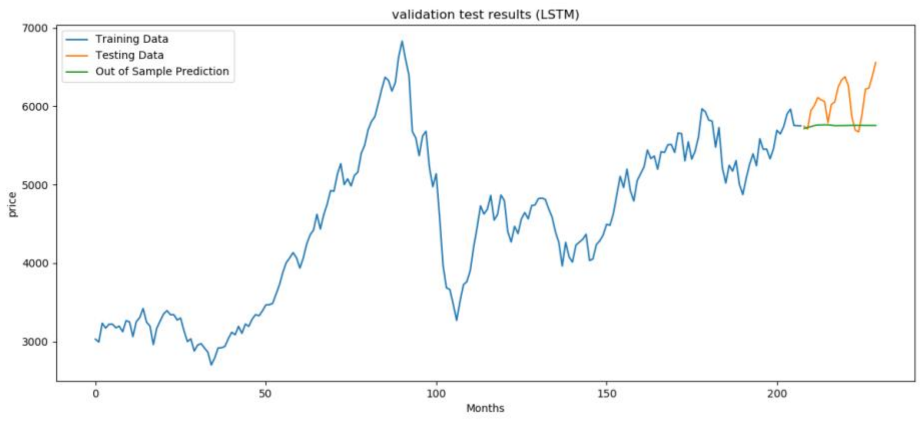
1.1.4.4.2. One step forecast

After fitting the model with the training set, we use one step forecast with whole time series data after data transformation and transform the data into the initial price value, below is the comparison between original piece and forecasted price in LSTM method, We observe that the fitting level is quite low, this may due to low number of the dataset:



1.1.4.4.3. Validation test

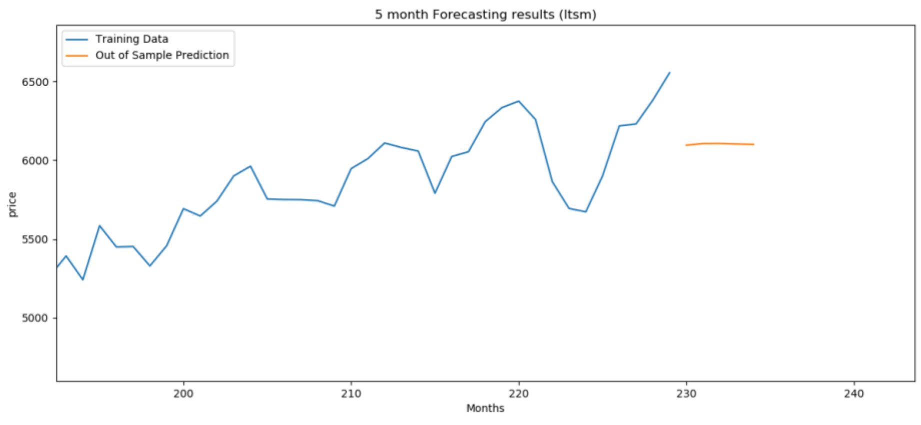
dynamic forecast is used to get the forecast monthly data value and these forecasting values are transformed to price form. Here is the MSE result between validation data and forecast data:



The MSE for validation set is 151370.47, from the diagram above we can observe that the forecasted price is quite steady compared with the validation data.

1.1.4.4.4. Forecasting

If LSTM is viewed as the most proper model to forecast, we should model the LSTM by using all the available data and conduct 5-step dynamic forecasting, here is the final forecasting as follow:



	Forecasting				
	1	2	3	4	5
monthly data	6095.30	6105.6	6106.23	6102.62	6100.41

2. Disadvantage

Disadvantage: these two models are really difficult to find the optimal parameter as there are a great number of parameters in the model. Therefore, it is hard to find the best model.

3. Conclusion

		Forecasting					MSE
		1	2	3	4	5	
Feed-forward NN	daily data	6388.38	6453.41	6472.29	6461.96	6449.69	260806.68
	monthly data	6328.52	6105.04	5909.68	5709.96	5496.81	620410.50
RNN (LSTM)	daily data	6452.48	6462.76	6459.5	6452.91	6444.59	38737.35
	monthly data	6095.30	6105.6	6106.23	6102.62	6100.41	151370.47

In conclusion, the RNN with LSTM method does better performance to the Feed-forward NN Method, this is because in RNN model, each input data feature is dependent with each other and the relationship between neurons is also dependent, which will optimal the time series data prediction performance as it can dig out the potential relationship among data.

Conclusion

In conclusion, the ARIMA model performs the best for the daily index and the Exponential smoothing has the least MSE on the monthly. The outcomes are shown below.

	Forecasting					MSE
	1	2	3	4	5	
daily data	6463.59	6463.17	6456.62	6465.53	6462.29	37248.70
monthly data	6527.50	6577.32	6627.15	6676.97	6726.79	17960.64

For the two Neural network models, the parameters should be improved for better performance.

Reference

11.3 Neural network models | Forecasting: Principles and Practice. (2019). Retrieved from https://otexts.com/fpp2/nnetar.html

A Beginner's Guide to LSTMs and Recurrent Neural Networks. (2019). Retrieved from https://skymind.ai/wiki/lstm

Recurrent Neural Networks Tutorial, Part 1 – Introduction to RNNs. (2019). Retrieved from http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

Sharmistha. (2019). *ARIMA/SARIMA vs LSTM with Ensemble learning Insights for Time Series Data*. Retrieved from https://towardsdatascience.com/arima-sarima-vs-lstm-with-ensemble-learning-insights-for-time-series-data-509a5d87f20a

Tutorial 10. (2019). Lecture.

Appendix A

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
dateparse = lambda dates: pd.datetime.strptime(dates, '%Y-%m-%d')
daily =
pd.read_csv('ASX200Daily.csv',parse_dates=['Date'],index_col='Date',
date_parser=dateparse)
dateparse1 = lambda dates: pd.datetime.strptime(dates, '%Y/%m/%d')
monthly =
pd.read_csv('ASX200Monthly.csv',parse_dates=['Date'],index_col='Date
',date_parser=dateparse)
daily_close = daily['Close']
monthly_close = monthly['Close']
daily.head()
monthly.head()
#%%
print('ASX 200 daily data information:')
print(daily.info())
print('ASX 200 monthly data information:')
print(monthly.info())
print('ASX 200 daily data description:')
print(daily.describe())
print('ASX 200 monthly data description:')
print(monthly.describe())

#%%
daily.isnull().sum()
monthly.isnull().sum()

daily = daily.interpolate()

monthly = monthly.interpolate()
Daily = daily['Close']
Monthly = monthly['Close']
daily = pd.DataFrame(Daily.dropna())
monthly = pd.DataFrame(Monthly.dropna())
#%%
plt.figure()
plt.plot(daily)
plt.title('daily index of ASX200')
plt.xlabel('Date')
plt.ylabel('Index')
plt.show()
#%%
plt.figure()
plt.plot(monthly)
plt.title('monthly index of ASX200')
plt.xlabel('Date')
plt.ylabel('Index')
plt.show()
#
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
dateparse = lambda dates: pd.datetime.strptime(dates, '%Y-%m-%d')
```

```
ASX200Daily = pd.read_csv('ASX200Daily.csv', parse_dates=['Date'],
index_col='Date', date_parser=dateparse)
ASX200Monthly = pd.read_csv('ASX200Monthly.csv',
parse_dates=['Date'], index_col='Date', date_parser=dateparse)
print(ASX200Daily.head())
print(ASX200Monthly.head())
ASX200Daily.isnull().sum()
ASX200Daily = ASX200Daily.interpolate()
ASX200Daily.isnull().sum()
daily = pd.DataFrame(ASX200Daily['Close'])
Monthly = ASX200Monthly['Close']
monthly = pd.DataFrame(Monthly.dropna())
#decompositions
#daily
#1.Initial trend estimate
Trend1 = daily.rolling(2, center = True).mean().rolling(250, center
= True).mean()
plt.figure()
plt.plot(daily, color='blue', label='daily')
plt.plot(Trend1, color='red',label='Trend1')
plt.title('daily trend')
plt.legend()
#2.de-trend:multipul
daily_res1 = daily/Trend1
plt.figure()
plt.plot(daily_res1)
plt.title('daily multiple')
#seasonal index
daily_res_1 = daily_res1.iloc[128:,:]
daily_res_zero_mul = np.nan_to_num(daily_res_1)
daily_S_mul = np.reshape(daily_res_zero_mul,(19,250))
daily_avg_mul = np.mean(daily_S_mul, axis=0)
print(daily_avg_mul)
#2.de-trend:add
daily_res2 = daily-Trend1
plt.figure()
plt.plot(daily_res2)
plt.title('daily add')
#seasonal index
daily_res_2 = daily_res2.iloc[128:,:]
daily_res_zero_add = np.nan_to_num(daily_res_2)
daily_S_add = np.reshape(daily_res_zero_add,(19,250))
daily_avg_add = np.mean(daily_S_add, axis=0)
print(daily_avg_add)
#monthly
#1.Initial trend estimate
Trend2 = monthly.rolling(2, center = True).mean().rolling(12, center
= True).mean()
plt.figure()
plt.plot(monthly, color='blue', label='monthly')
plt.plot(Trend2, color='red',label='Trend2')
plt.title('monthly trend')
plt.legend()
#2.de-trend:multipul
monthly_res1 = monthly/Trend2
plt.figure()
plt.plot(monthly_res1)
plt.title('monthly multiple')
#seasonal index
monthly_res_1 = monthly_res1.iloc[2:,:]
monthly_res_zero_mul = np.nan_to_num(monthly_res_1)
monthly_S_mul = np.reshape(monthly_res_zero_mul,(19,12))
monthly_avg_mul = np.mean(monthly_S_mul, axis=0)
print(monthly_avg_mul)
#2.de-trend:add
monthly_res2 = monthly-Trend2
plt.figure()
plt.plot(monthly_res2)
plt.title('monthly add')
#seasonal index
monthly_res_2 = monthly_res2.iloc[2:,:]
monthly_res_zero_add = np.nan_to_num(monthly_res_2)
monthly_S_add = np.reshape(monthly_res_zero_add,(19,12))
monthly_avg_add = np.mean(monthly_S_add, axis=0)
print(monthly_avg_add)
# split the data set
train_size_daily = int(len(daily['Close'])*0.9)
validation_size_daily = len(daily['Close'])-train_size_daily
daily_train = daily.iloc[:train_size_daily,:]
daily_val = daily.iloc[train_size_daily:,:]
train_size_monthly = int(len(monthly['Close'])*0.9)
validation_size_monthly = len(monthly['Close'])-train_size_monthly
monthly_train = monthly.iloc[:train_size_monthly,:]
monthly_val = monthly.iloc[train_size_monthly:,:]
#
#Naive forecast for daily
nai1 = np.asarray(daily_train['Close'])
y_d = daily_val.copy()
y_d['naive'] = nai1[len(nai1)-1]
plt.figure()
plt.plot(daily_train.index, daily_train['Close'], label='train')
plt.plot(daily_val.index, daily_val['Close'], label='validation')
plt.plot(y_d.index, y_d['naive'], label='forecast')
plt.title('Naive Forecast for daily')
plt.legend()
#MSE
from sklearn.metrics import mean_squared_error
print("MSE_daily_naive:
{0}".format(mean_squared_error(daily_val['Close'], y_d['naive'])))
#
#Naive forecast for monthly
nai2 = np.asarray(monthly_train['Close'])
y_m = monthly_val.copy()
y_m['naive'] = nai2[len(nai2)-1]
plt.figure()
plt.plot(monthly_train.index, monthly_train['Close'], label='train')
plt.plot(monthly_val.index, monthly_val['Close'],
label='validation')
plt.plot(y_m.index, y_m['naive'], label='forecast')
plt.title('Naive Forecast for monthly')
plt.legend()
#MSE
```

```
print("MSE_monthly_naive:
{0}".format(mean_squared_error(monthly_val['Close'], y_m['naive'])))
#
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
#load the data as a time series
dateparse = lambda dates: pd.datetime.strptime(dates, '%Y-%m-%d')
daily =
pd.read_csv('ASX200Daily.csv',parse_dates=['Date'],index_col='Date',
date_parser=dateparse)
monthly =
pd.read_csv('ASX200Monthly.csv',parse_dates=['Date'],index_col='Date
',date_parser=dateparse)
daily.info()
monthly.info()
daily.describe()
monthly.describe()
print(daily)
print(monthly)
#clean data
orig=pd.read_csv('ASX200Daily.csv')
orig['Date']= pd.to_datetime(orig['Date'])
orig.set_index('Date',inplace=True)
orig.isnull().sum()
orig = orig.interpolate()
orig.isnull().sum()
clean_daily=orig['Close']
index_daily=clean_daily.to_frame()
clean_monthly=monthly.dropna()['Close']
index_monthly=clean_monthly.to_frame()
clean_daily.describe()
clean_monthly.describe()
print(clean_daily)
print(clean_monthly)
#train-validation split
train_size1 = int(len(index_daily['Close'])*0.9)
validation_size1 = len(index_daily['Close'])-train_size1
train_daily=clean_daily.iloc[:train_size1]
validation_daily=clean_daily.iloc[train_size1:]
train_size2 = int(len(index_monthly['Close'])*0.9)
validation_size2 = len(index_monthly['Close'])-train_size2
train_monthly=clean_monthly.iloc[:train_size2]
validation_monthly=clean_monthly.iloc[train_size2:]
#Drift Method
#daily
y_hat_drift_d=validation_daily.copy().to_frame()
y_t_d=train_daily.iloc[-1]
y_1_d=train_daily.iloc[0]
y_hat_drift_d['drift_d']=0
for i in range(len(validation_daily)):
     y_hat_drift_d['drift_d'].iloc[i]=y_t_d+((i+1)*(y_t_d-
y_1_d)/(len(train_daily)-1))
     i=i+1
print(y_hat_drift_d['drift_d'] )
```

```
plt.figure(figsize=(12,8))
plt.plot(train_daily, label='train_d')
plt.plot(validation_daily, label='validation_d')
plt.plot(y_hat_drift_d['drift_d'], label='drift_forecast_d')
plt.legend(loc='best')
plt.show()
mse_drift_d=mean_squared_error(validation_daily,y_hat_drift_d['drift
_d'])
print(mse_drift_d)
#prediction for next five days
drift_pre_d=[]
y_t_d1=validation_daily.iloc[-1]
y_1_d1=train_daily.iloc[0]
for m in range(5):
     drift_pre_d.append(y_t_d1+((m+1)*(y_t_d1-
y_1_d1)/(len(clean_daily)-1)))
print(drift_pre_d)
#monthly
y_hat_drift_m= validation_monthly.copy().to_frame()
y_t_m=train_monthly.iloc[-1]
y_1_m=train_monthly.iloc[0]
y_hat_drift_m['drift_m']=0
for k in range(len(validation_monthly)):
     y_hat_drift_m['drift_m'].iloc[k]=y_t_m+((k+1)*(y_t_m-
y_1_m)/(len(train_monthly)-1))
     k=k+1
print(y_hat_drift_m['drift_m'] )
plt.figure(figsize=(12,8))
plt.plot(train_monthly, label='train_m')
plt.plot(validation_monthly, label='validation_m')
plt.plot(y_hat_drift_m['drift_m'], label='drift_forecast_m')
plt.legend(loc='best')
plt.show()
mse_drift_m=mean_squared_error(validation_monthly,y_hat_drift_m['dri
ft_m'])
print(mse_drift_m)
#prediction for next five months
drift_pre_m=[]
y_t_m1=validation_monthly.iloc[-1]
y_1_m1=train_monthly.iloc[0]
for n in range(5):
     drift_pre_m.append(y_t_m1+((n+1)*(y_t_m1-
y_1_m1)/(len(clean_monthly)-1)))
print(drift_pre_m)
#Holt's linear method
#daily
#define function
def holts(y,alpha,beta):
    ll=[y[0]]
    bb=[y[1]-y[0]]
    holts_mannually=[]
    for t in range(len(y)):
        ll.append(alpha*y[t]+(1-alpha)*(ll[t]+bb[t]))
        bb.append(beta*(ll[t+1]-ll[t])+(1-beta)*bb[t])
        holts_mannually.append(ll[t+1]+bb[t+1])
    return holts_mannually,ll[1:],bb[1:]
```

```
def sse(x,y):
    return np.sum(np.power(x-y,2))
sse_d=[]
alphas=np.arange(0.1,1,0.1)
betas=np.arange(0.1,1,0.1)
for a in alphas:
    for b in betas:
        smoothed_d,ll_d,bb_d=holts(clean_daily,alpha=a,beta=b)
        sse_d.append(sse(smoothed_d[:-1],clean_daily.values[1:]))
#optimal alpha and beta
optimal_d = np.argmin(sse_d)
#def com()
com=[]
alphas=np.arange(0.1,1,0.1)
betas=np.arange(0.1,1,0.1)
for p in alphas:
    for q in betas:
        com.append((p,q))
print(com[optimal_d])
#calculte MSE
holts_train_d,lll,bbb=holts(train_daily,alpha=0.9,beta=0.1)
holts_val_d=[]
for day in range(len(validation_daily)):
    holts_val_d.append(lll[-1]+(day+1)*bbb[-1])
print(holts_val_d)
mse_holts_d=mean_squared_error(holts_val_d,validation_daily)
print(mse_holts_d)
#forecast
holts_train_dd,l,b=holts(clean_daily,alpha=0.9,beta=0.1)
holts_fore_d=[]
for da in range(5):
    holts_fore_d.append(l[-1]+(da+1)*b[-1])
print(holts_fore_d)
#monthly
#define function
sse_m=[]
alphas=np.arange(0.1,1,0.1)
betas=np.arange(0.1,1,0.1)
for a in alphas:
    for b in betas:
        smoothed_m,ll_m,bb_m=holts(clean_monthly,alpha=a,beta=b)
        sse_m.append(sse(smoothed_m[:-1],clean_monthly.values[1:]))
#optimal alpha and beta
optimal_m = np.argmin(sse_m)
print(com[optimal_m])
#calculte MSE
holts_train_m,lll,bbb=holts(train_monthly,alpha=0.9,beta=0.1)
holts_val_m=[]
for month in range(len(validation_monthly)):
    holts_val_m.append(lll[-1]+(month+1)*bbb[-1])
print(holts_val_m)
mse_holts_d=mean_squared_error(holts_val_m,validation_monthly)
print(mse_holts_m)
#forecast
holts_train_mm,l,b=holts(clean_monthly,alpha=0.9,beta=0.1)
holts_fore_m=[]
```

```
for mon in range(5):
    holts_fore_m.append(l[-1]+(mon+1)*b[-1])
print(holts_fore_m)
#%%-------------------------------------------------------------------
-------
#                              ARIMA
#--------------------------------------------------------------------
---------
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
import statsmodels.api as sm
from pandas import Series
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.api import qqplot
import warnings
warnings.filterwarnings('ignore')

#------------------daily data and IDA-------------------------

orig=pd.read_csv('ASX200Daily.csv')
orig['Date']= pd.to_datetime(orig['Date'])
orig.set_index('Date',inplace=True)

# clean the data
#filling missing value
orig.isnull().sum()
orig = orig.interpolate()
orig.isnull().sum()


# split the data set
train_size = int(len(orig['Close'])*0.9)
validation_size = len(orig['Close'])-train_size

train_d = orig.iloc[:train_size,:]
validation_d = orig.iloc[train_size:,:]


#%%
#----------------------data prepocessing----------------------

#check initial training data
#plot original data
D=train_d.index
p=train_d['Close']
plt.figure(figsize=(8,4))
lines=plt.plot(D,p)
plt.setp(lines[0],linewidth=1,color='green')
plt.title('The original daily data')
plt.xlabel('Datetime')
plt.ylabel('Close price')
```

```
#log transform
train_d['close_log']=np.log(train_d['Close'])
D=train_d.index
p=train_d['close_log']
plt.figure(figsize=(8,4))
lines=plt.plot(D,p)
plt.setp(lines[0],linewidth=1,color='green')
plt.title('The Log transformed daily data')
plt.xlabel('Datetime')
plt.ylabel('Log close price')

#first differencing
close_log=train_d['close_log']
close_log_diff=train_d['close_log'].diff()
close_log_diff.dropna(inplace=True)
D=train_d.index[1:]
x=close_log_diff
plt.figure(figsize=(8,5))
lines=plt.plot(D,x)
plt.setp(lines[0],linewidth=1,color='green')
plt.title('The first differece of log-transformed data')
plt.xlabel('Datetime')
plt.ylabel('First difference')
plt.show()

#test stationarity after log and first difference
#plot ACF & PACF
plt.figure(figsize=(20,8))
acf=sm.graphics.tsa.plot_acf(close_log_diff,lags=30, alpha = 0.05)
pacf=sm.graphics.tsa.plot_pacf(close_log_diff,lags=30, alpha = 0.05)

#Dickey-Fuller test
def test_stationarity(timeseries):
    print('Results of Dickey-Fuller test')
    result=adfuller(timeseries,autolag='AIC')
    print('ADF Statistic: %f' % result[0])
    print('p-value: %f' % result[1])
    print('Lages Used: %f' % result[2])
    print('Number of Observations Used: %f' % result[3])
    print('Critical Values:')
    for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value))
test_stationarity(close_log_diff)
#stationary, so tentative order (3,i,3)=(3,1,3)

#%%
#----------------------order selection----------------------
#Select order by AIC
import statsmodels.tsa.stattools as st
order
=st.arma_order_select_ic(close_log_diff,max_ar=6,max_ma=6,ic=['aic']
)
print(order.aic_min_order)
p=order.aic_min_order[0]
q=order.aic_min_order[1]
```

```
#%%
#------------------------model fitting----------------------------
# ARIMA(6,1,6) model
from statsmodels.tsa.arima_model import ARIMA
model1 = ARIMA(close_log, order=(4,1,3))
result1 = model1.fit(disp=-1)
print(result1.summary())

# ARIMA(3,1,3) model
from statsmodels.tsa.arima_model import ARIMA
model2 = ARIMA(close_log, order=(3, 1, 3))
result2 = model2.fit(disp=-1)
print(result2.summary())
#%%
#------------------------residual diagonostic-----------------
# plot residuals
residuals_d = pd.DataFrame(result1.resid)
plt.figure(figsize=(16,4))
plt.plot(residuals_d)
plt.title('Residuals')
plt.show()

#plot ACF for residual
plt.figure()
acf_r=sm.graphics.tsa.plot_acf(residuals_d,lags=30, alpha = 0.05)

# plot distribution of residual
sns.distplot(residuals_d)

#test the stationary of residuals
#Dickey-Fuller test
test_stationarity(residuals_d.loc[:,0])

#test the normality of residuals
#KS test
from scipy.stats import kstest
x=kstest(residuals_d, 'norm')

#the overall description of residual
residuals_d.describe()

#%%
#-------------invertability and stationarity test for MA------
ma = np.array([3])
arma_process = sm.tsa.ArmaProcess(ma)
print("MA Model is{0}stationary".format(" "if
arma_process.isstationary else "not"))
print("MA Model is{0}invertible".format(" "if
arma_process.isinvertible else "not"))

#%%
#-----------------get fitted series and training MSE---------------
# Get Fitted Series in initial form
forecast_t = result1.predict(typ = 'levels', dynamic =False)
train_d['forecast']=None
```

```
train_d.iloc[1:,7]=np.exp(forecast_t)
train_d.head()

# plot fitted series
D=train_d.index
p=train_d['Close']
f=train_d['forecast']
plt.figure(figsize=(8,4))
lines=plt.plot(D,f,p)
plt.setp(lines[0],linewidth=2,color='red',linestyle='--')
plt.setp(lines[1],linewidth=0.7,color='blue')
plt.title('The fitted daily series')
plt.xlabel('Datetime')
plt.ylabel('price')
plt.legend(loc='upper left')
plt.show()

# caculate MSE
from sklearn.metrics import mean_squared_error
train_d_mse=mean_squared_error(train_d.iloc[1:,7],train_d.iloc[1:,3]
)

print('ARIMA train daily data MSE:{0:2f}'.format(train_d_mse))


#%%
#-------------- forecast for validation data set------------------
validation_d['forecast']=None
validation_d['forecast']=np.exp(result1.forecast(steps=validation_si
ze)[0])

#visulazition
D=validation_d.index
p=validation_d['Close']
f=validation_d['forecast']
plt.figure(figsize=(8,4))
lines=plt.plot(D,f,p)
plt.setp(lines[0],linewidth=1,color='red')
plt.setp(lines[1],linewidth=1,color='blue')
plt.title('The forecast for validation set')
plt.xlabel('Datetime')
plt.ylabel('price')
plt.legend(loc='upper left')
plt.show()

# Validation MSE
validation_d_mse=mean_squared_error(validation_d['forecast'],validat
ion_d['Close'])
print('ARIMA validation daily data
MSE:{0:2f}'.format(validation_d_mse))
#%%
#------------------ 5 day forecast---------------------
orig['close_log']=np.log(orig['Close'])
from statsmodels.tsa.arima_model import ARIMA
model3 = ARIMA(orig['close_log'], order=(4,1,3))
result3 = model3.fit(disp=-1)
```

```
print(result3.summary())

# save as csv
x=np.around(np.exp(result3.forecast(steps=5)[0]), decimals=2)
dt=pd.date_range(start='2019-05-27', end='2019-05-31')
pre=pd.DataFrame({'Predictions':x,'Dates':dt})
pre.set_index('Dates',inplace=True)
pre.to_csv('prediction_ARIMA.csv')


#%% -----------------------monthly data---------------------------
--------
orig_m=pd.read_csv('ASX200Monthly.csv')
orig_m['Date']= pd.to_datetime(orig_m['Date'])
orig_m.set_index('Date',inplace=True)

# clean the data
#filling missing value
orig_m.isnull().sum()
orig_m= orig_m.interpolate()
orig_m.isnull().sum()


# split the data set
train_size_m = int(len(orig_m['Close'])*0.9)
validation_size_m = len(orig_m['Close'])-train_size_m

train_m = orig_m.iloc[:train_size_m,:]
validation_m = orig_m.iloc[train_size_m:,:]


#%%
#----------------------data prepocessing----------------------

#check initial training data
#plot original data
D=train_m.index
p=train_m['Close']
plt.figure(figsize=(8,4))
lines=plt.plot(D,p)
plt.setp(lines[0],linewidth=1,color='green')
plt.title('The original monthly data')
plt.xlabel('Datetime')
plt.ylabel('Close price')

#log transform
train_m['close_log']=np.log(train_m['Close'])
D=train_m.index
p=train_m['close_log']
plt.figure(figsize=(8,4))
lines=plt.plot(D,p)
plt.setp(lines[0],linewidth=1,color='green')
plt.title('The Log transformed monthly data')
plt.xlabel('Datetime')
plt.ylabel('Log close price')
```

```
#first differencing
close_log_m=train_m['close_log']
close_log_diff_m=train_m['close_log'].diff()
close_log_diff_m.dropna(inplace=True)
D=train_m.index[1:]
x=close_log_diff_m
plt.figure(figsize=(8,5))
lines=plt.plot(D,x)
plt.setp(lines[0],linewidth=1,color='green')
plt.title('The first differece of log-transformed data')
plt.xlabel('Datetime')
plt.ylabel('First difference')
plt.show()

#test stationarity after log and first difference
#plot ACF & PACF
plt.figure(figsize=(20,8))
acf=sm.graphics.tsa.plot_acf(close_log_diff_m,lags=30, alpha = 0.05)
pacf=sm.graphics.tsa.plot_pacf(close_log_diff_m,lags=30, alpha =
0.05)

#Dickey-Fuller test
def test_stationarity(timeseries):
    print('Results of Dickey-Fuller test')
    result=adfuller(timeseries,autolag='AIC')
    print('ADF Statistic: %f' % result[0])
    print('p-value: %f' % result[1])
    print('Lages Used: %f' % result[2])
    print('Number of Observations Used: %f' % result[3])
    print('Critical Values:')
    for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value))
test_stationarity(close_log_diff_m)
#stationary, so tentative order (3,i,3)=(3,1,3)

#%%
#----------------------order selection----------------------
#Select order by AIC
import statsmodels.tsa.stattools as st
order
=st.arma_order_select_ic(close_log_diff_m,max_ar=6,max_ma=6,ic=['aic
'])
print(order.aic_min_order)
p=order.aic_min_order[0]
q=order.aic_min_order[1]


#%%
#-----------------------model fitting----------------------------
# ARIMA(6,1,6) model
from statsmodels.tsa.arima_model import ARIMA
model1_m = ARIMA(close_log_m, order=(5,1,2))
result1_m = model1_m.fit(disp=-1)
print(result1_m.summary())
#%%
# ARIMA(3,1,3) model
```

```
from statsmodels.tsa.arima_model import ARIMA
model2_m = ARIMA(close_log_m, order=(0, 1, 0))
result2_m = model2_m.fit(disp=-1)
print(result2_m.summary())
#%%
#-------------------------residual diagonostic-----------------
# plot residuals
residuals_m = pd.DataFrame(result1_m.resid)
plt.figure(figsize=(16,4))
plt.plot(residuals_m)
plt.title('Residuals')
plt.show()

#plot ACF for residual
plt.figure()
acf_r=sm.graphics.tsa.plot_acf(residuals_m,lags=30, alpha = 0.05)

# plot distribution of residual
sns.distplot(residuals_m)

#test the stationary of residuals
#Dickey-Fuller test
test_stationarity(residuals_m.loc[:,0])

#test the normality of residuals
#KS test
from scipy.stats import kstest
x=kstest(residuals_m, 'norm')

#the overall description of residual
residuals_m.describe()

#%%
#-------------invertability and stationarity test for MA------
ma = np.array([2])
arma_process = sm.tsa.ArmaProcess(ma)
print("MA Model is{0}stationary".format(" "if
arma_process.isstationary else "not"))
print("MA Model is{0}invertible".format(" "if
arma_process.isinvertible else "not"))

#%%
#-----------------get fitted series and training MSE---------------
# Get Fitted Series in initial form
forecast_t_m = result1_m.predict(typ = 'levels', dynamic =False)
train_m['forecast']=None
train_m.iloc[1:,7]=np.exp(forecast_t_m)
train_m.head()

# plot fitted series
D=train_m.index
p=train_m['Close']
f=train_m['forecast']
plt.figure(figsize=(8,4))
lines=plt.plot(D,f,p)
plt.setp(lines[0],linewidth=1,color='red',label='forecast price')
plt.setp(lines[1],linewidth=0.7,color='blue',label='actual price')
plt.title('The fitted monthly series')
plt.xlabel('Datetime')
plt.ylabel('price')
plt.legend(loc='upper left')
plt.show()

# caculate RMSE
train_m_mse=mean_squared_error(train_m.iloc[1:,7],train_m.iloc[1:,3]
)

print('ARIMA train daily data RMSE:{0:2f}'.format(train_m_mse))


#%%
#-------------- forecast for validation data set------------------
validation_m['forecast']=None
validation_m['forecast']=np.exp(result1_m.forecast(steps=validation_
size_m)[0])

#visulazition
D=validation_m.index
p=validation_m['Close']
f=validation_m['forecast']
plt.figure(figsize=(8,4))
lines=plt.plot(D,f,p)
plt.setp(lines[0],linewidth=1,color='red',label='forecast price')
plt.setp(lines[1],linewidth=1,color='blue',label='actual price')
plt.title('The forecast for validation set')
plt.xlabel('Datetime')
plt.ylabel('price')
plt.legend(loc='upper left')
plt.show()

# Validation RMSE
validation_m_mse=mean_squared_error(validation_m['forecast'],validat
ion_m['Close'])
print('ARIMA validation daily data
MSE:{0:2f}'.format(validation_m_mse))
#%%
#------------------ 5 day forecast---------------------
orig_m['close_log']=np.log(orig_m['Close'])
from statsmodels.tsa.arima_model import ARIMA
model3_m = ARIMA(orig_m['close_log'], order=(5,1,2))
result3_m = model3_m.fit(disp=-1)
print(result3_m.summary())
print(np.exp(result3_m.forecast(steps=5)[0]))
#NN
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import math
from sklearn.model_selection import GridSearchCV
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from keras.layers.core import Dense
from keras.models import Sequential
#%%
# ignore warning
import warnings
warnings.simplefilter('ignore')
# set figsize
from pylab import rcParams
rcParams['figure.figsize'] = 16,6
np.random.seed(1)
data_d = pd.read_csv('ASX200Daily.csv',
parse_dates=['Date'],index_col='Date')
data_m = pd.read_csv('ASX200Monthly.csv',
parse_dates=['Date'],index_col='Date')
data_d_cls= data_d['Close']
data_m_cls= data_m['Close']
data_d_cls=pd.to_numeric(data_d_cls,errors='coerce')
data_m_cls=pd.to_numeric(data_m_cls,errors='coerce')
data_d_cls=data_d_cls.interpolate()
data_m_cls=data_m_cls.dropna()
data_m_cls=data_m_cls.values.reshape((len(data_m_cls),1))
data_d_cls=data_d_cls.values.reshape((len(data_d_cls),1))
scaler = MinMaxScaler(feature_range=(0, 1))
#%%
data_m_cls = scaler.fit_transform(data_m_cls)
data_d_cls = scaler.fit_transform(data_d_cls)
# define window size = 6, daily
time_window_d = 6
#%%

Xall_d, Yall_d = [], []
for i in range(time_window_d, len(data_d_cls)):
    Xall_d.append(data_d_cls[i-time_window_d:i, 0])
    Yall_d.append(data_d_cls[i, 0])

Xall_d = np.array(Xall_d)
Yall_d = np.array(Yall_d)
train_size = int(len(Xall_d) * 0.9)
test_size = len(Xall_d) - train_size

# split training set
Xtrain = Xall_d[:train_size,:] #
Ytrain = Yall_d[:train_size]
Xtest = Xall_d[-test_size:,:]  #
Ytest = Yall_d[-test_size:]     #
#%%
####   grid search start
best_score = 100000
for a in [60]:
    for b in [30]:
        for c in [12,14]:
            model = Sequential()
            model.add(Dense(c, input_dim = time_window_d, activation
= 'relu'))
            model.add(Dense(1))
            model.compile(loss = 'mean_squared_error', optimizer =
'adam')
            np.random.seed(1)
            model.fit(Xtrain, Ytrain, epochs = b, batch_size = a,
verbose = 2, validation_split = 0.05) # use 95% for training and 5%
for testing
            # predict all y values and transform back to normal
scale
            allPredict = model.predict(Xall_d)
            allPredictPlot = scaler.inverse_transform(allPredict)
            trainScore =
mean_squared_error(scaler.inverse_transform(data_d_cls[:train_size])
, allPredictPlot[:train_size,0])
            print('Training Data MSE: {0:.2f}'.format(trainScore))
            if trainScore < best_score:
                best_score = trainScore
                best_parameters =
{'epochs':b,'batch_size':a,'neauon':c}
            else :
                best_score =best_score

print("Best score:{:.2f}".format(best_score))
print("Best parameters:{}".format(best_parameters))

#%%

#%%
# add two layers (1 hidden + 1 output)
model_d = Sequential()
model_d.add(Dense(14, input_dim = time_window_d, activation =
'relu'))
model_d.add(Dense(1))
# loss function: MSE
# optimization method: ADAM
model_d.compile(loss = 'mean_squared_error', optimizer = 'adam')
np.random.seed(1)


#%%
model_d.fit(Xtrain, Ytrain,
          epochs = 30,
          batch_size = 60,
          verbose = 2,
          validation_split = 0.05) # use 95% for training and 5% for
testing
#%%
allPredict = model_d.predict(Xall_d)
allPredictPlot = scaler.inverse_transform(allPredict)

#%%
# visualization

#%%
plt.figure()
plt.plot(scaler.inverse_transform(data_d_cls), label='True Data')
plt.plot(np.arange(time_window_d, len(data_d_cls)), allPredictPlot, 
label='One-Step Prediction')
plt.title('Feedforward Neural Network daily data in One step 
forecast')
plt.legend();
plt.xlabel('Days')
plt.ylabel('price')


#%%
trainScore = mean_squared_error(Ytrain, allPredict[:train_size,0])
print('Training Data MSE: {0:.8f}'.format(trainScore))
#%%
#dynamic forcast
dynamic_prediction = np.copy(data_d_cls[:len(data_d_cls) - 
test_size])

for i in range(len(data_d_cls) - test_size, len(data_d_cls)):
    last_feature = np.reshape(dynamic_prediction[i-time_window_d:i], 
(1,time_window_d))
    next_pred = model_d.predict(last_feature)
    dynamic_prediction = np.append(dynamic_prediction, next_pred)

dynamic_prediction = dynamic_prediction.reshape(-1,1)
dynamic_prediction = scaler.inverse_transform(dynamic_prediction)


#%%
plt.figure()
plt.plot(scaler.inverse_transform(data_d_cls[:len(data_d_cls) - 
test_size]), label='Training Data')
plt.plot(np.arange(len(data_d_cls) - test_size, len(data_d_cls), 1), 
         scaler.inverse_transform(data_d_cls[-test_size:]), 
         label='Testing Data')
plt.plot(np.arange(len(data_d_cls) - test_size, len(data_d_cls), 1), 
         dynamic_prediction[-test_size:], 
         label='Out of Sample Prediction')
plt.legend(loc = "upper left")
plt.xlabel('Days')
plt.ylabel('price')
plt.title('Dynamic Forecasting results (Feedforward NN in daily 
data)');

#%%
testScore = mean_squared_error(scaler.inverse_transform(data_d_cls[-
test_size:]), 
                                         dynamic_prediction[-
test_size:])
print('Dynamic Forecast MSE: {0:.2f}'.format(testScore))


#%%
trainScore = 
mean_squared_error(scaler.inverse_transform(data_d_cls[:train_size])
, allPredictPlot[:train_size,0])
print('Training Data MSE: {0:.2f}'.format(trainScore))
```

```
#%%
origin = np.copy(data_d_cls)
prediction_d = origin

#%%
for i in range(len(prediction_d)-5,len(prediction_d)):
    if  i<=4883:
          last_feature_d= np.reshape(prediction_d[-
time_window_d:len(prediction_d)], (1,time_window_d))
          next_pred_d= model_d.predict(last_feature_d)
          prediction_d = np.append(prediction_d, next_pred_d)
    else:
        print(prediction_d)
prediction_d = prediction_d.reshape(-1,1)
prediction_d = scaler.inverse_transform(prediction_d)
#%%
plt.figure()
plt.plot(scaler.inverse_transform(data_d_cls), label='Training
Data')
plt.plot(np.arange(len(prediction_d) - 5, len(prediction_d), 1),
prediction_d[-5:], label='Out of Sample Prediction')
plt.legend(loc = "upper left")
plt.title('5 day Forecasting results in dayily data(Feedforward
NN)');
plt.xlabel('Days')
plt.ylabel('price')
#%%
print(prediction_d[-5:].round(2))

#%%

#lstn

from keras.layers.recurrent import LSTM
from keras.layers.core import Dense, Activation
#%%
# define time window
time_window_d2= 6

# reset training / test data for X and Y
Xall, Yall = [], []

for i in range(time_window_d2, len(data_d_cls)):
    Xall.append(data_d_cls[i-time_window_d2:i, 0])
    Yall.append(data_d_cls[i, 0])

# Convert them from list to array
Xall = np.array(Xall)
Yall = np.array(Yall)

train_size = int(len(Xall) * 0.9)
test_size = len(Xall) - train_size

Xtrain = Xall[:train_size, :]
Ytrain = Yall[:train_size]
```

```
Xtest = Xall[-test_size:, :]
Ytest = Yall[-test_size:]

Xtrain = np.reshape(Xtrain, (Xtrain.shape[0], time_window_d2, 1))
Xtest = np.reshape(Xtest, (Xtest.shape[0], time_window_d2, 1))


#%%
from keras.callbacks import EarlyStopping
#%%
####   grid search start
best_score = 100000
for d in [ 60, 80, 100,120,140]:
    for e in [10,15, 20,30,40, 50, 60]:
        for f in [13,14,15,20]:
            model = Sequential()
            model.add(LSTM(input_shape=(None,
1),units=f,return_sequences=False))
            model.add(Dense(output_dim=1))
            model.add(Activation("linear"))
            model.compile(loss = "mse", optimizer = "rmsprop")
            early_stop = EarlyStopping(monitor = 'loss', patience =
2, verbose = 1)
            np.random.seed(1)
         # model fitting
            model.fit(Xtrain,Ytrain,batch_size = d,nb_epoch =
e,validation_split = 0.05)
            allPredict = model.predict(np.reshape(Xall,
(len(Xall),time_window_d2,1))
            )
            allPredict = scaler.inverse_transform(allPredict)
            allPredictPlot = np.empty_like(data_d_cls)
            allPredictPlot[:, :] = np.nan
            allPredictPlot[time_window_d2:, :] = allPredict
            trainScore =
mean_squared_error(scaler.inverse_transform(data_d_cls[:train_size])
, allPredict[:train_size,0])
            print('Training Data MSE: {0:.2f}'.format(trainScore))
            if trainScore < best_score:
                best_score = trainScore
                best_parameters =
{'epochs':e,'batch_size':d,'unit':f}
            else :
                best_score =best_score

print("Best score:{:.2f}".format(best_score))
print("Best parameters:{}".format(best_parameters))
#%%
model = Sequential()


model.add(LSTM(input_shape=(None,
1),units=15,return_sequences=False))
model.add(Dense(output_dim=1))
model.add(Activation("linear"))
model.compile(loss = "mse", optimizer = "rmsprop")
#%%
early_stop = EarlyStopping(monitor = 'loss', patience = 2, verbose =
1)
#%%
np.random.seed(1)

# model fitting
model.fit(Xtrain,
          Ytrain,
          batch_size = 140,
          nb_epoch = 15,
          validation_split = 0.05)
#%%
# predict all y values and transform back to the original scale
allPredict = model.predict(np.reshape(Xall,
(len(Xall),time_window_d2,1)))
allPredict = scaler.inverse_transform(allPredict)
# prepare variables for visualization
allPredictPlot = np.empty_like(data_d_cls)
allPredictPlot[:, :] = np.nan
allPredictPlot[time_window_d2:, :] = allPredict
#%%
# visualization results
plt.figure()
plt.plot(scaler.inverse_transform(data_d_cls), label='True Data')
plt.plot(allPredictPlot, label='One-Step Prediction')
plt.legend()
plt.title('LSTM daily data in One step forecast')
plt.xlabel('Days')
plt.ylabel('price')
plt.show()
#%%# MSE score
trainScore =
mean_squared_error(scaler.inverse_transform(data_d_cls[:train_size])
,
                                                     allPredict[:train_size,0])
print('Training Data MSE: {0:.2f}'.format(trainScore))

#%%
#dynamic forcast
dynamic_prediction = np.copy(data_d_cls[:len(data_d_cls) -
test_size])

for i in range(len(data_d_cls) - test_size, len(data_d_cls)):
    last_feature = np.reshape(dynamic_prediction[i-
time_window_d2:i], (1,time_window_d2,1))
    next_pred = model.predict(last_feature)
    dynamic_prediction = np.append(dynamic_prediction, next_pred)

# dynamic prediction results, and transform back to the original
scale
dynamic_prediction = dynamic_prediction.reshape(-1,1)
dynamic_prediction = scaler.inverse_transform(dynamic_prediction)
#%%
```

```
# visualization results
plt.figure()
plt.plot(scaler.inverse_transform(data_d_cls[:len(data_d_cls) -
test_size]), label='Training Data')
plt.plot(np.arange(len(data_d_cls) - test_size, len(data_d_cls), 1),
         scaler.inverse_transform(data_d_cls[-test_size:]),
         label='Testing Data')
plt.plot(np.arange(len(data_d_cls) - test_size, len(data_d_cls), 1),
         dynamic_prediction[-test_size:],
         label='Out of Sample Prediction')
plt.legend(loc = "upper left")
plt.title('LSTM daily data in dynamic forecast')
plt.xlabel('Days')
plt.ylabel('price')
plt.show();
#%%
# MSE score
testScore = mean_squared_error(scaler.inverse_transform(data_d_cls[-
test_size:]),
                                             dynamic_prediction[-
test_size:])
print('Dynamic Forecast MSE: {0:.2f}'.format(testScore))

#%%
origin = np.copy(data_d_cls)
prediction_d2 = origin

#%%
for i in range(len(prediction_d2)-5,len(prediction_d2)):
    if  i<=4883:
          last_feature_d2= np.reshape(prediction_d2[-
time_window_d2:len(prediction_d2)], (1,time_window_d2,1))
          next_pred_d2= model.predict(last_feature_d2)
          prediction_d2 = np.append(prediction_d2, next_pred_d2)
    else:
        print(prediction_d2)
prediction_d2 = prediction_d2.reshape(-1,1)
prediction_d2 = scaler.inverse_transform(prediction_d2)
#%%
print(prediction_d2[-5:].round(2))

#%%
plt.figure()
plt.plot(scaler.inverse_transform(data_d_cls), label='Training
Data')
plt.plot(np.arange(len(prediction_d2) - 5, len(prediction_d2), 1),
prediction_d2[-5:], label='Out of Sample Prediction')
plt.legend(loc = "upper left")
plt.title('5 day Forecasting results (lstm)')
plt.xlabel('Days')
plt.ylabel('price');
#%%
#month data nn mothod
time_window_m = 12
Xall_m, Yall_m = [], []
for i in range(time_window_m, len(data_m_cls)):
    Xall_m.append(data_m_cls[i-time_window_m:i, 0])
    Yall_m.append(data_m_cls[i, 0])

Xall_m = np.array(Xall_m)
Yall_m = np.array(Yall_m)
train_size = int(len(Xall_m) * 0.9)
test_size = len(Xall_m) - train_size

# split training set
Xtrain = Xall_m[:train_size,:] #
Ytrain = Yall_m[:train_size]
Xtest = Xall_m[-test_size:,:]  #
Ytest = Yall_m[-test_size:]     #
#%%
#grid search start
best_score = 100000
for g in [10, 20, 40, 60, 80, 100]:
    for h in [10, 20,30,50, 60,70,100]:
        for i in [12,14,16,18,20]:
            model_m= Sequential()
            model_m.add(Dense(i, input_dim = time_window_m,
activation = 'relu'))
            model_m.add(Dense(1))
            model_m.compile(loss = 'mean_squared_error', optimizer =
'adam')
            np.random.seed(1)
            model_m.fit(Xtrain, Ytrain, epochs = h, batch_size = g,
verbose = 2, validation_split = 0.05) # use 95% for training and 5%
for testing
            # predict all y values and transform back to normal
scale
            allPredict = model_m.predict(Xall_m)
            allPredictPlot = scaler.inverse_transform(allPredict)
            trainScore =
mean_squared_error(scaler.inverse_transform(data_m_cls[:train_size])
, allPredictPlot[:train_size,0])
            print('Training Data MSE: {0:.2f}'.format(trainScore))
            if trainScore < best_score:
                best_score = trainScore
                best_parameters =
{'epochs':h,'batch_size':g,'neauon':i}
            else :
                best_score =best_score

print("Best score:{:.2f}".format(best_score))
print("Best parameters:{}".format(best_parameters))

#%%
#%%
# add two layers (1 hidden + 1 output)
model_m1 = Sequential()
model_m1.add(Dense(20, input_dim = time_window_m, activation =
'relu'))
model_m1.add(Dense(1))
# loss function: MSE
# optimization method: ADAM
```

```
model_m1.compile(loss = 'mean_squared_error', optimizer = 'adam')
np.random.seed(1)

#%%
model_m1.fit(Xtrain, Ytrain,
          epochs = 50,
          batch_size = 100,
          verbose = 2,
          validation_split = 0.05) # use 95% for training and 5% for
testing
#%%
# predict all y values and transform back to normal scale
allPredict = model_m1.predict(Xall_m)
allPredictPlot = scaler.inverse_transform(allPredict)

#%%
# visualization
plt.figure()
plt.plot(scaler.inverse_transform(data_m_cls), label='True Data')
plt.plot(np.arange(time_window_m, len(data_m_cls)), allPredictPlot,
label='One-Step Prediction')
plt.title('Feedforward Neural Network_monly data')
plt.xlabel('months')
plt.ylabel('price')
plt.legend();
#%%
# MSE
trainScore = mean_squared_error(Ytrain, allPredict[:train_size,0])
print('Training Data MSE: {0:.8f}'.format(trainScore))
#%%
#dynamic forcast
dynamic_prediction = np.copy(data_m_cls[:len(data_m_cls) -
test_size])

for i in range(len(data_m_cls) - test_size, len(data_m_cls)):
    last_feature = np.reshape(dynamic_prediction[i-time_window_m:i],
(1,time_window_m))
    next_pred = model_m1.predict(last_feature)
    dynamic_prediction = np.append(dynamic_prediction, next_pred)

dynamic_prediction = dynamic_prediction.reshape(-1,1)
dynamic_prediction = scaler.inverse_transform(dynamic_prediction)


#%%
plt.figure()
plt.plot(scaler.inverse_transform(data_m_cls[:len(data_m_cls) -
test_size]), label='Training Data')
plt.plot(np.arange(len(data_m_cls) - test_size, len(data_m_cls), 1),
         scaler.inverse_transform(data_m_cls[-test_size:]),
         label='Testing Data')
plt.plot(np.arange(len(data_m_cls) - test_size, len(data_m_cls), 1),
         dynamic_prediction[-test_size:],
         label='Out of Sample Prediction')
plt.legend(loc = "upper left")
plt.xlabel('Months')
plt.ylabel('price')
plt.title('Dynamic Forecasting results (Feedforward NN-monthly)');

#%%
testScore = mean_squared_error(scaler.inverse_transform(data_m_cls[-
test_size:]),
                                         dynamic_prediction[-
test_size:])
print('Dynamic Forecast MSE: {0:.2f}'.format(testScore))


#%%
trainScore =
mean_squared_error(scaler.inverse_transform(data_m_cls[:train_size])
, allPredictPlot[:train_size,0])
print('Training Data MSE: {0:.2f}'.format(trainScore))

#%%
origin = np.copy(data_m_cls)
prediction_m = origin

#%%
for i in range(len(prediction_m)-5,len(prediction_m)):
    if  i<=236:
          last_feature_m= np.reshape(prediction_m[-
time_window_m:len(prediction_m)], (1,time_window_m))
          next_pred_m= model_m1.predict(last_feature_m)
          prediction_m = np.append(prediction_m, next_pred_m)
    else:
        print(prediction_m)
prediction_m = prediction_m.reshape(-1,1)
prediction_m = scaler.inverse_transform(prediction_m)
#%%
print(prediction_m[-5:].round(2))
#%%
plt.figure()
plt.plot(scaler.inverse_transform(data_m_cls), label='Training
Data')
plt.plot(np.arange(len(prediction_m) - 5, len(prediction_m), 1),
prediction_m[-5:], label='Out of Sample Prediction')
plt.legend(loc = "upper left")
plt.xlabel('Months')
plt.ylabel('price')
plt.title('5 month Forecasting results (Feedforward NN)');

#%%
#lstn monthly data

from keras.layers.recurrent import LSTM
from keras.layers.core import Dense, Activation
#%%
# define time window
time_window_m2= 12

# reset training / test data for X and Y
Xall_m, Yall_m = [], []
```

```
for i in range(time_window_m2, len(data_m_cls)):
    Xall_m.append(data_m_cls[i-time_window_m2:i, 0])
    Yall_m.append(data_m_cls[i, 0])

# Convert them from list to array
Xall_m = np.array(Xall_m)
Yall_m = np.array(Yall_m)

train_size = int(len(Xall_m) * 0.9)
test_size = len(Xall_m) - train_size

Xtrain = Xall_m[:train_size, :]
Ytrain = Yall_m[:train_size]

Xtest = Xall_m[-test_size:, :]
Ytest = Yall_m[-test_size:]

Xtrain = np.reshape(Xtrain, (Xtrain.shape[0], time_window_m2, 1))
Xtest = np.reshape(Xtest, (Xtest.shape[0], time_window_m2, 1))


#%%
from keras.callbacks import EarlyStopping
#%%
####   grid search start
best_score = 100000
for x in [ 60, 80, 100,120,140]:
    for y in [10,15, 20,30,40, 50, 60]:
        for z in [15,20,30,40,50,60,70]:
            model_m2 = Sequential()
            model_m2.add(LSTM(input_shape=(None,
1),units=z,return_sequences=False))
            model_m2.add(Dense(output_dim=1))
            model_m2.add(Activation("linear"))
            model_m2.compile(loss = "mse", optimizer = "rmsprop")
            early_stop = EarlyStopping(monitor = 'loss', patience =
2, verbose = 1)
            np.random.seed(1)
            model_m2.fit(Xtrain,Ytrain,batch_size = x,nb_epoch =
y,validation_split = 0.01)
            allPredict = model_m2.predict(np.reshape(Xall_m,
(len(Xall_m),time_window_m2,1))
            )
            allPredict = scaler.inverse_transform(allPredict)
            allPredictPlot = np.empty_like(data_m_cls)
            allPredictPlot[:, :] = np.nan
            allPredictPlot[time_window_m2:, :] = allPredict
            trainScore =
mean_squared_error(scaler.inverse_transform(data_m_cls[:train_size])
, allPredict[:train_size,0])
            print('Training Data MSE: {0:.2f}'.format(trainScore))
            if trainScore < best_score:
                best_score = trainScore
                best_parameters =
{'epochs':y,'batch_size':x,'unit':z}
```

```
            else :
                best_score =best_score
print("Best score:{:.2f}".format(best_score))
print("Best parameters:{}".format(best_parameters))
#%%
model_m3= Sequential()
model_m3.add(LSTM(input_shape=(None,
1),units=40,return_sequences=False))
model_m3.add(Dense(output_dim=1))
model_m3.add(Activation("linear"))

model_m3.compile(loss = "mse", optimizer = "rmsprop")
#%%
early_stop = EarlyStopping(monitor = 'loss', patience = 2, verbose =
1)
#%%
np.random.seed(1)
# model fitting
model_m3.fit(Xtrain,
          Ytrain,
          batch_size = 60,
          nb_epoch = 10,
          validation_split = 0.01)
#%%
allPredict = model_m3.predict(np.reshape(Xall_m,
(len(Xall_m),time_window_m2,1)))
allPredict = scaler.inverse_transform(allPredict)
allPredictPlot = np.empty_like(data_m_cls)
allPredictPlot[:, :] = np.nan
allPredictPlot[time_window_m2:, :] = allPredict
#%%
# visualization results
plt.figure()
plt.plot(scaler.inverse_transform(data_m_cls), label='True Data')
plt.plot(allPredictPlot, label='One-Step Prediction')
plt.legend()
plt.show()
#%%# MSE score
trainScore =
mean_squared_error(scaler.inverse_transform(data_m_cls[:train_size])
,
                                          allPredict[:train_size,0])
print('Training Data MSE: {0:.2f}'.format(trainScore))

#%%
#dynamic forcast
dynamic_prediction = np.copy(data_m_cls[:len(data_m_cls) -
test_size])

for i in range(len(data_m_cls) - test_size, len(data_m_cls)):
    last_feature = np.reshape(dynamic_prediction[i-
time_window_m2:i], (1,time_window_m2,1))
    next_pred = model_m3.predict(last_feature)
    dynamic_prediction = np.append(dynamic_prediction, next_pred)
```

```
# dynamic prediction results, and transform back to the original
scale
dynamic_prediction = dynamic_prediction.reshape(-1,1)
dynamic_prediction = scaler.inverse_transform(dynamic_prediction)
#%%
# visualization results
plt.figure()
plt.plot(scaler.inverse_transform(data_m_cls[:len(data_m_cls) -
test_size]), label='Training Data')
plt.plot(np.arange(len(data_m_cls) - test_size, len(data_m_cls), 1),
         scaler.inverse_transform(data_m_cls[-test_size:]),
         label='Testing Data')
plt.plot(np.arange(len(data_m_cls) - test_size, len(data_m_cls), 1),
         dynamic_prediction[-test_size:],
         label='Out of Sample Prediction')
plt.legend(loc = "upper left")
plt.show();
#%%
# MSE score
testScore = mean_squared_error(scaler.inverse_transform(data_m_cls[-
test_size:]),
                                            dynamic_prediction[-
test_size:])
print('Dynamic Forecast MSE: {0:.2f}'.format(testScore))

#%%
origin = np.copy(data_m_cls)
prediction_m2 = origin
#%%
for i in range(len(prediction_m2)-5,len(prediction_m2)):
    if  i<=236:
          last_feature_m2= np.reshape(prediction_m2[-
time_window_m2:len(prediction_m2)], (1,time_window_m2,1))
          next_pred_m2= model_m3.predict(last_feature_m2)
          prediction_m2 = np.append(prediction_m2, next_pred_m2)
    else:
        print(prediction_m2)
prediction_m2 = prediction_m2.reshape(-1,1)
prediction_m2 = scaler.inverse_transform(prediction_m2)

#%%
plt.figure()
plt.plot(scaler.inverse_transform(data_m_cls), label='Training
Data')
plt.plot(np.arange(len(prediction_m2) - 5, len(prediction_m2), 1),
prediction_m2[-5:], label='Out of Sample Prediction')
plt.legend(loc = "upper left")
plt.title('5 month Forecasting results (ltsm)');
```

Appendix B

THE UNIVERSITY OF SYDNEY

TEAM TASK MEETING AGENDA

TEAM MEETING AGENDA

Group 48

Meeting to be held ABS Seminar Building

12 May 2019

10:00 a.m.-12:30 a.m.

Chairperson: Xue Xia

Minute-Taker: Rui Chen

1. Apologies: None
2. Confirmation of agenda (150 minutes)(XUE XIA)
3. Confirmation of minutes of 12 May (Date) (150 minutes)(XUE XIA)
4. Business arising from minutes of 12 May (Date) (150 minutes)(XUE XIA)
5. Items

   (1) read and analyze the question of the task

   (2) Developed a thesis framework

   (3) Determine the model

   (4) assign the task to each person
6. Any other business: None (XUE XIA)
7. Forward agenda items: None (XUE XIA)
8. Next meeting: 18 May (XUE XIA)

THE UNIVERSITY OF SYDNEY

MINUTES TEMPLATE

Minutes of meeting for Group 48

Date: 12 May 2019 Time: 10:00 a.m.-12:30 a.m. Location: ABS Building

Chairperson: Rui Chen

Minute-Taker: Xue Xia

Document tabled: Rui Chen

Present: Zhiyuan Li, Yusheng Zhou, Xue Xia, Mengxing Zhao, Rui Chen

Apologies: None

Agenda Item	Key Points	Action	By Whom	When	Communication Strategy
1. The structure of the report 2. Assign task for each person	* read and analyze the question of the task * Developed a thesis framework * Determine the model * assign the	* read and analyze the question of the task * Developed a framework of the report * Determine the model * assign the	Mengxing Zhao Zhiyuan Li Yusheng Zhou Rui Chen, Xue	10:00 10:40 11:40 12:20	Discusstion

	task to each person	task to each person	Xia		

Souce: TAFE Access Division "Communication for Business", 2000

TEAM TASK MEETING AGENDA

TEAM MEETING AGENDA

_____________Group 48_____________(Company Name)

Meeting to be held _____**ABS Building**__________(Where)

________**18 May, 2019**__________ (Date)

________**10.00-11.42 am**__________(Time)

Chairperson: __**Xue Xia**____________

Minute-Taker: _____**Yusheng Zhou**____________

9. Apologies: None
10. Confirmation of agenda (102 minutes)(Xue Xia)
11. Confirmation of minutes of __**18 May, 2019**__(Date) (102 minutes)(Xue Xia)
12. Business arising from minutes of _**18 May, 2019** (Date) (102 minutes)(Xue Xia)
13. Items

    (1) Improving the way to handle null values

    (2) Determining the method of splitting train validation

    (3) Taking ARIMA's order
14. Any other business (Xue Xia)
15. Forward agenda items (Xue Xia)
16. Next meeting 24 May, 2019 (Xue xia)

MINUTES TEMPLATE

Minutes of meeting for ____________Group 48____________________________ (Company Name)

Date: _____**18 May, 2019**_________ Time: ____**10.00-11.42 am**________ Location: ______ **ABS Building**______________________

Chairperson: _______**Xue Xia**____

Minute-Taker: _________**Yusheng Zhou**____

Document tabled: ______**Zhiyuan Li**____

Present: **Xue Xia, Yusheng Zhou, Zhiyuan Li, Rui Chen, Mengxing Zhao**

____

Apologies:

______**None**____

____

Agenda Item	Key Points	Action	By Whom	When	Communication Strategy
1. Discuss the doubts about the parts that have already been done	* Improving the way to handle null values	* Improving the way to handle null values	Xue Xia	10.10 am	Discussion
	* Determining the method of splitting train validation	* Determining the method of splitting train validation	Mengxing Zhao	10.31 am	Discussion
2. Arrange what to do next	* Taking ARIMA's order	* Taking ARIMA's order	Zhiyuan Li, Rui Chen	11.00 am	Discussion
					Discussion

	*Adjusting parameters of RNN model	*Adjusting parameters of RNN model	Yusheng Zhou	11.20 am	

Souce: TAFE Access Division “Communication for Business”, 2000

THE UNIVERSITY OF SYDNEY

TEAM TASK MEETING AGENDA

TEAM MEETING AGENDA

____________Group 48________(Company Name)

Meeting to be held ___Sydney University Regiment Building___(Where)

_________24/05/2019_________ (Date)

___________13:00______________(Time)

Chairperson: _Mengxing Zhao__Yusheng Zhou

Minute-Taker: ______Rui Chen________

1. Apologies:
2. Confirmation of agenda (10 minutes)(Mengxing Zhao)
3. Confirmation of minutes of __24/05/2019___ (Date) (5 minutes)(Mengxing Zhao)
4. Business arising from minutes of _24/05/2019 (Date) (10 minutes)(Mengxing Zhao)
5. Items

   (1) Collect outcomes of each member.

   (2) Use the latest data to fit the model
6. Any other business (5 minutes) (Mengxing Zhao)
7. Forward agenda items (10 minutes)(Mengxing Zhao)
8. Next meeting (Mengxing Zhao)

THE UNIVERSITY OF SYDNEY

MINUTES TEMPLATE

Minutes of meeting for Group 48 **(Company Name)**

Date: 24/05/2019 Time: 13:00 Location: Sydney University Regiment Building

Chairperson: Mengxing Zhao Yusheng Zhou

Minute-Taker: Rui Chen

Document tabled: Mengxing Zhao

Present: Mengxing Zhao Yusheng Zhou Ruichen Xue Xia Zhiyuan Li

Apologies:

Agenda Item	Key Points	Action	By Whom	When	Communication Strategy
1. Collect outcomes of each member. 2. Use the latest data to fit the model	*EDA *Benchmark *ARIMA *RNN * * * * * * * * *		1. Mengxing Zhao 2. Yusheng Zhou	1. 13:30 2. 16:00	Television PowerPoint Discussion

Souce: TAFE Access Division “Communication for Business”, 2000