THE UNIVERSITY OF SYDNEY

QBUS6840 Predictive Analytics (2019S1)

Group Project (Assignment 2)

Due date:	Saturday 25 May 2018
Group Number:	143
Group Members:	470437832
	490451474
	480519021
	490095267

Contents

1 Introduction

Over the last few decades, trading in a stock market became one of the most popular ways to create wealth. According to the survey conducted by ASX, there is about 1/3 of the population in Australia held shares in 2017 (ASX Australian Investor Study, 2017). Furthermore, the stock market performed an important role in the growth of the economy of a country has attracted more people's attention. The future value of a stock is an important index for investors to consider whether they are going to invest in the stock market. However, it is a challenging task to make an accurate forecast for the stock index due to the dynamic nature of stock prices. Therefore, this report makes an effort to find out the best model used to forecast monthly and daily stock index in respectively by using the historical data of S&P/ASX 200 which performed as the benchmark of equity performance on Australian markets. It is also representative to use the data of S&P/ASX 200 to measure Australian stock market due to the high proportion that stocks listed on the ASX accounted for Australia's share market capitalization (ASX200 List, 2019). In this report, monthly data and daily data of S&P/ASX 200 would be collected and used to build Autoregressive Integrated Moving Average model (ARIMA) and Long Short Term Memory model (LSTM). The best model for monthly and daily forecast would be selected in respectively by input the validation set and compare the accuracy with real index. Finally, the future index for monthly and daily be predicted via the outputs of the selected model.

2 Related Work

There are many factors that can affect stock prices such as general reports and earnings announcements. Therefore, it is important to select appropriate algorithms to make the prediction more accurate. The study conducted by Nelson et al (2017) predicted stock prices by using LSTM method and revealed the outcomes of stock index generated by LSTM model is usually more accurate than predicted by other machine learning models. One paper from Ayodele et al (2014) compared the accuracy of ARIMA and ANN model by predicting daily stock prices. The result indicates that the ARIMA model generally makes a better prediction if the time series is linear whereas the ANN model outperforms in dealing with non-linear time series. Another study predicted financial data by applying ARIMA and LSTM methods. The performance of these two models revealed that LSTM is superior to ARIMA model (S. S. Namin & A. S. Namin, 2018). Further Experimental results obtained by Wang et al (2012) indicated a method used in predicting combined different models usually generates more accurate results due to the complicated time series of the stock price.

3 Dataset and Features

The data set used in this experiment is monthly historical stock index starts from "2000-02-29" and daily historical stock index starts from "2000-03-31". The data set contains 7 columns which are date, opening price, high price, low price, closing price, adjusted closing price and volume traded. The opening price is the price that the stock first traded on the trading day. High price and low price represent the highest price and lowest price the stock trades on this trading day. Closing price means the price stock trades when the stock market closes. Adjusted closing price is the closing price which also considers some account factors. In this report, closing price is selected to form a time series with date components together and the forecasts we made would be based on that time series.

4 Exploratory Data Analysis

4.1 Data Preprocessing

Monthly and daily data collected were read through pd function of python and the column of closing price was extracted and converted into a time series with date as index. It can be observed that there is 1 missing value in the monthly data set and 42 missing values in daily data set which need to be addressed through cleaning the data. The removing of the missing values is an essential task before applying any predictive models as if the model consists of missing values it can have an effect on our predictions. By applying dropna function, all missing values are removed, and the descriptive statistics of monthly and daily data are shown as follows. In addition, the original data set is split as training subset and validation subset in order to make a better forecast in the following prediction sections. Validation subset contains the last one year's figures from both monthly data set and daily data set while the rest of data would be the components of training subset for us to build the model. Furthermore, all of rows with the day of week as Sunday have been removed to improve our accuracy as we only make predictions from Monday to Friday.

	Monthly Data	Daily Data
Count	228.00	4779.00
Mean	4659.37	4638.69
Standard Deviation	1015.29	1008.55

Table 1

We perform different kind of preprocessing steps for different reasons. For instance, removing the nan value will clean up and necessary and irrelevant data and we will process the clean data to get better results. Secondly if we do not scale the data before using it for Neural Network we might not get the correct output as we multiply the data with weights and this output is fed into the next layer so if input data is of huge magnitude this output can become really huge after some iterations, the processing time can be huge and the accuracy will be dropped. For ARIMA model we want the data to be stationary as otherwise the Expected value will not be the same and we will not be able to get the mean of the timeseries to model it. Those specific preprocessing steps for different model to make predictions will be explained in their sections.

4.2 Exploring Monthly Time Series

As the time series is combined of trend, cycle, seasonal variations and irregular fluctuations. A moving average smoothing is needed to calculate the initial trend- cycle estimate. As we know our data is monthly data that means m=12, so we must do a "2 by m"-MA, where m is the period. In this case, the period is m=12, so we must do a "2X12 MA". We have done smoothing through the help of pandas rolling. When we run this program, we observe that there are 7 nans in the beginning and 5 nans in the end but we need to have 6 missing values at each end, so to restore this pattern we need to call the shift function to move one step forward.

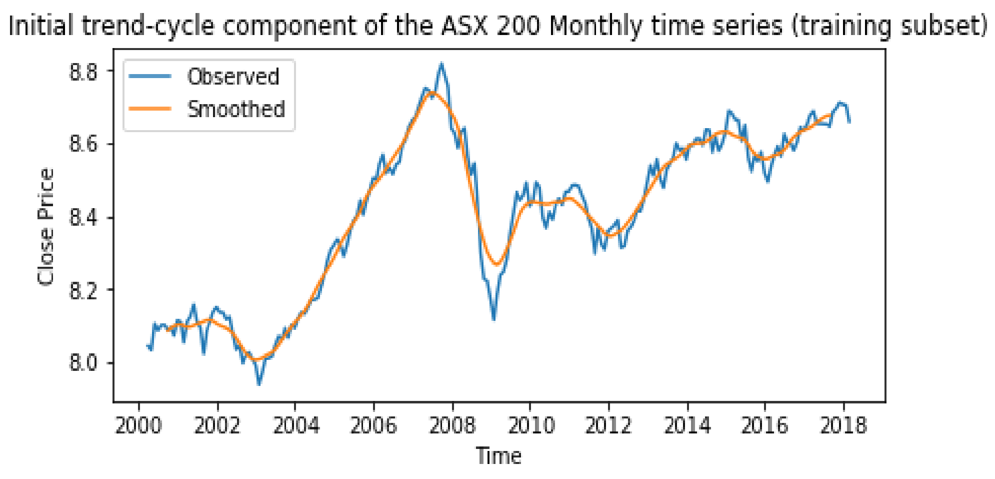


Figure 1

Through figure 7 we observe that there is no specific trend in the time series but it looks more like cyclic component involved in it. If we look into the period between 2008 and 2010 there is a significant downward trend which is due to the Global Financial Crises which occurred during the period 2007-2008.

For the Seasonal Component we observe the time series and break it into different components as result we subtracted the trend cycle from the monthly smoothed series to reveal the Seasonality. Observing the Seasonality in figure 8 we can say that there is an irregular seasonality pattern and we can see there is a significant dip between 2008-2010.

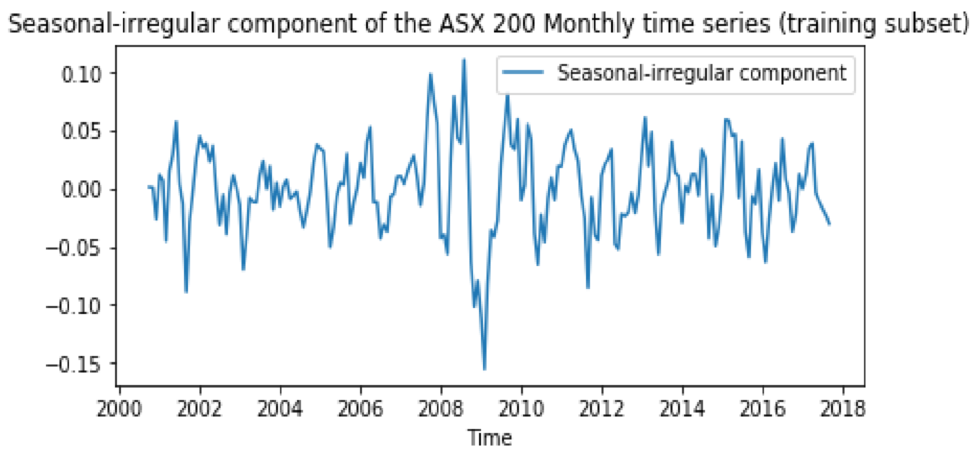


Figure 2

4.3 Exploring Daily Time Series

In order to reveal an initial trend-cycle estimate, a CMA-5 is used by smoothing out noises and seasonality. We assume the seasonal period is 5 as there are 2 days that do not have any data each week.

Through figure 3 (given below) we can observe that the trend of daily-cycle component is really similar to the original time series as we use a very small rolling period for a huge data set. In addition, there is no obviously trend that we can observe visually as cycle might dominate the data set.

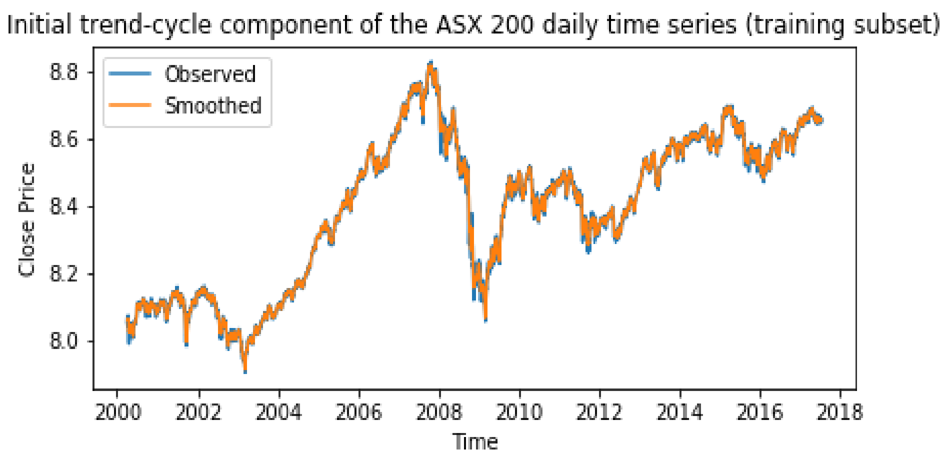


Figure 3

Seasonal component is extracted by subtracting initial trend-cycle estimate from the original time series.

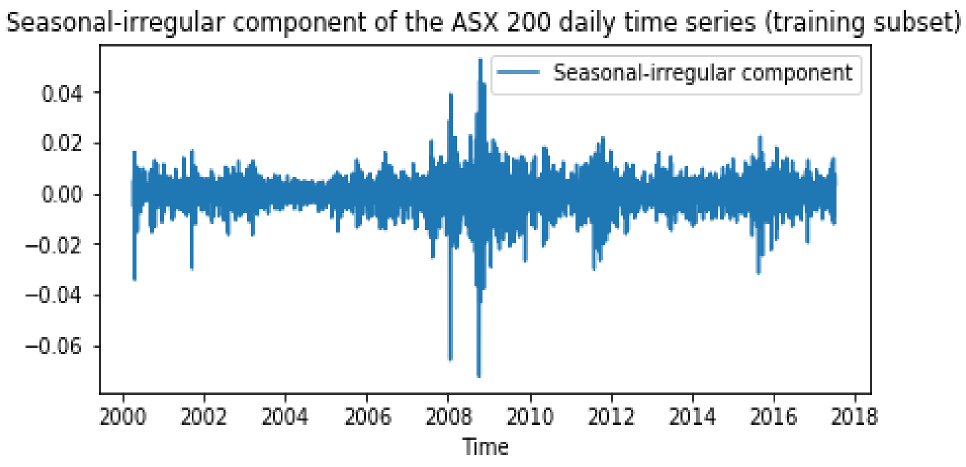


Figure 4

It can be observed that there is an irregular seasonality pattern in daily data set. In another word, it seems that there does not exist any seasonality in the time series. Furthermore, the influence of global financial crises between 2008 and 2010 still can be observed as the fluctuations are significant.

4.4 Differences

The main difference between two graphs can be observed visually is that daily seasonal component shows a higher frequency based on the large number of values. In order to observe more differences between two patterns, we zoom on in the graph of daily seasonal component in 2017 and July of 2015 in respectively. If we look into the seasonal component of 2017, the patterns in shows a trend that goes up very slowly. However, the pattern shows in monthly data is a peak with a significant drop following. The figures in two data set is different in the same period which can also be proved by comparing seasonal component based on one month.

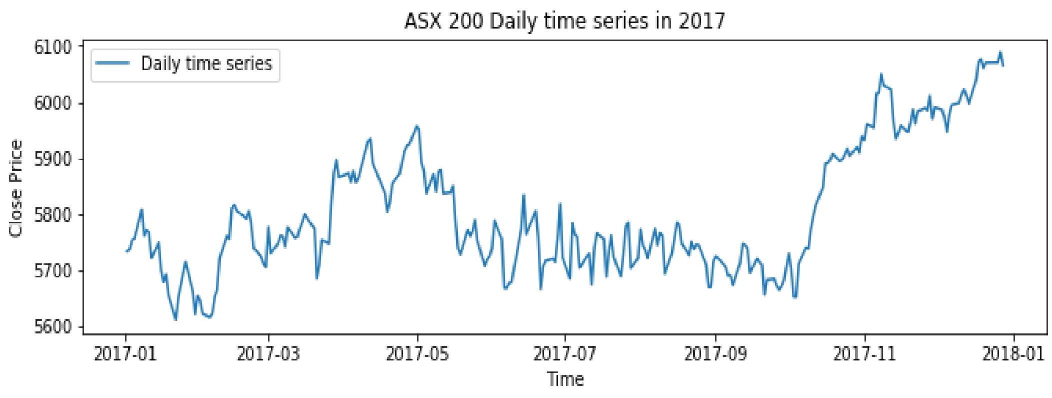


Figure 5

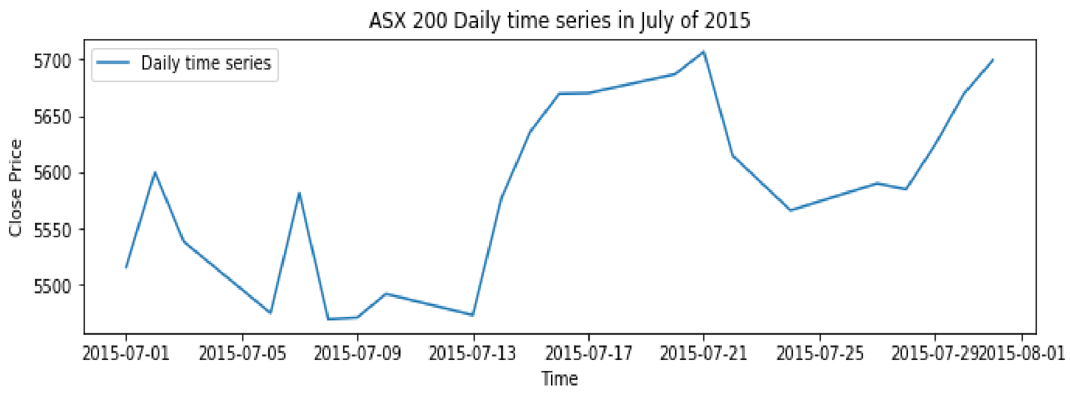


Figure 6

5 Methods

5.1 Decomposition

5.1.1 Model Description

The basic idea for decomposition method is to deconstruct the original time series into several different components which include trend, cycle, seasonal component and irregular fluctuations. By extracting each component and finding the relationships, it can be achieved by applying two different methods which is chosen based on the original time series, one is additive method, and another is multiplicative method. The forecast would be easily created by re-build relationships between decomposed components.

5.1.2 Motivation

There are a variety of patterns that can be revealed through decomposition method. It gives a chance to observe the features and the relationships between different components. Furthermore, the components it revealed would be reliable for us to make predictions as the time period is really broad. Therefore, decomposition is selected as a basic model to make the forecast. The selection of additive and multiplicative method is also important since it affects a lot on final predictions. As multiplicative model is usually used for the time series that seasonal variations are proportional to the trend. Therefore, it seems better to use additive model for the decomposition task since we cannot see any proportion of relationships from the graphs.

5.1.3 Preprocessing Steps

Although it seems that there does not exist a proportion relationship between the trend and seasonal peaks, it is still too subjective if we are going to use additive method. Therefore, log transformation for both of data set is executed to make the time series more stable and fit addictive method better.

5.1.4 Model fitting

The initial trend-cycle estimates $(\widehat{T_t + C_t})$ are calculated by applying moving average. For monthly data, the repeated period is 12. For daily data, we assume the seasonality is 5 as there are 2 days that do not have any data in each week. Therefore, a CMA-12 for monthly data set and a CMA-5 for daily data set are performed by using rolling function of panda. Due to the nature of panda, the number of nan values in the beginning and the end of time series that generated by applying moving average is not symmetry. In order to restore the patterns, the smoothed series has been moved one step forward.

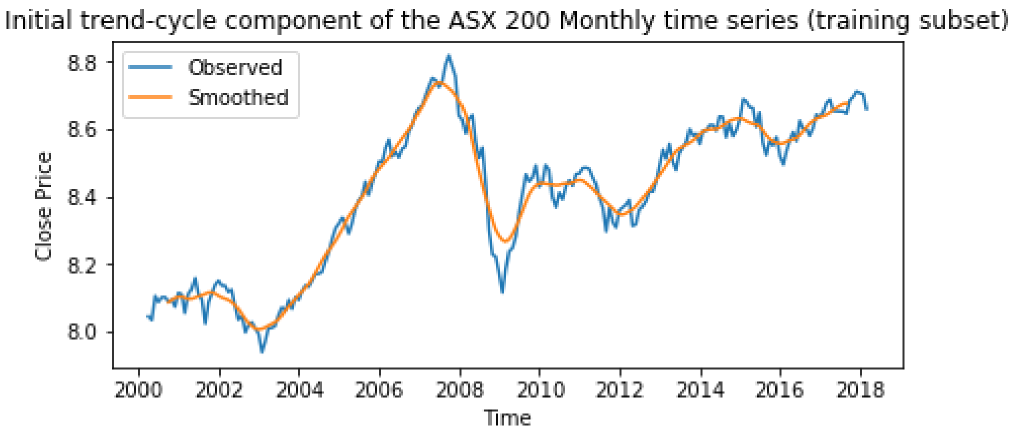


Figure 7

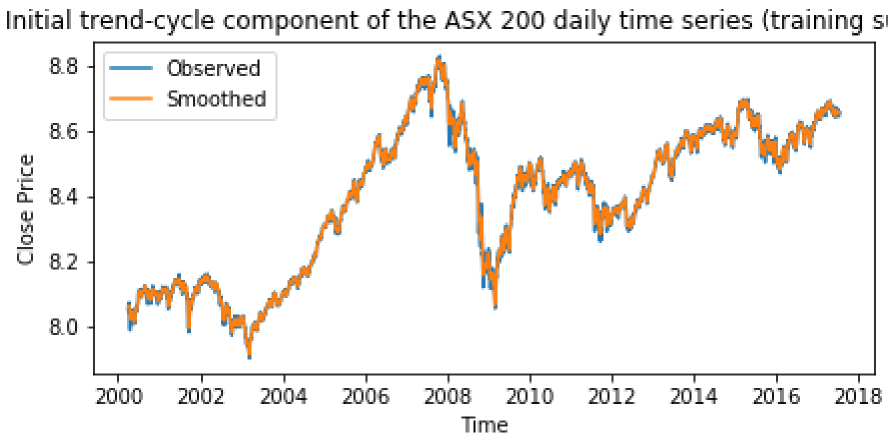


Figure 8

Detrend the time series by subtracting initial trend-cycle estimate from the original time series: $\widehat{S_t + e_t} = y_t - \widehat{T_t + C_t}$. The patterns of seasonal and error components are extracted through conducting the code of "*seasonal = training_subset - trend_cycle*".

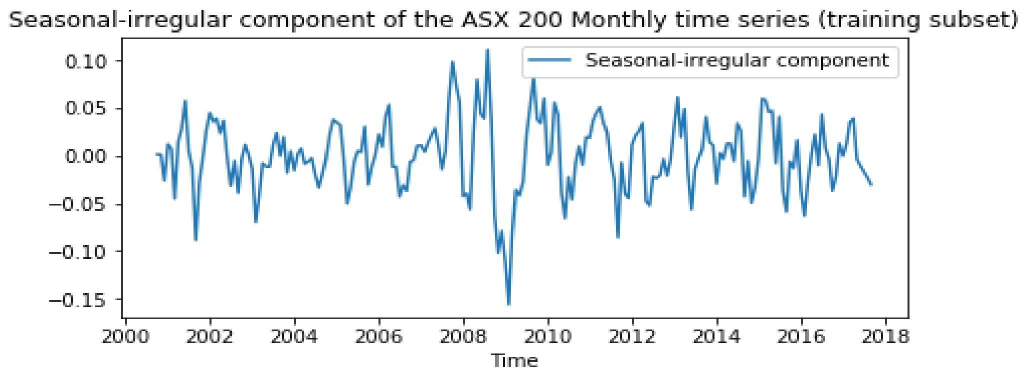


Figure 9

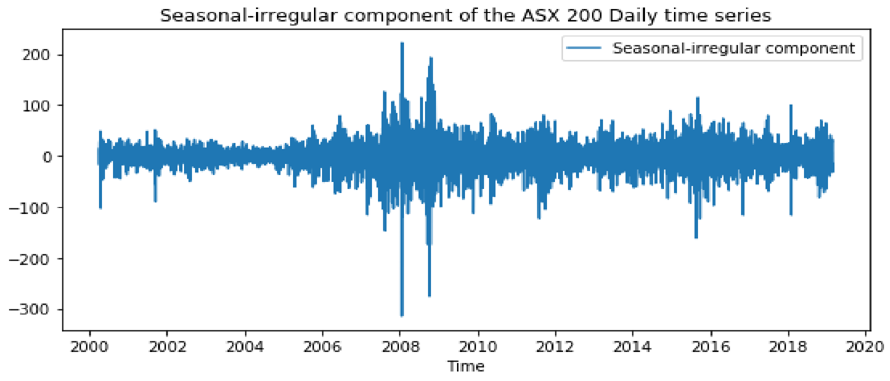


Figure 10

Then a normalization is executed on the detrended time series by making the mean of the seasonal index equal to 0 to get rid of errors and obtain seasonal indices $\hat{S}_t$. Deseasonalize the original time series to get trend-cycle and error components: $d_t = \widehat{T_t + C_t} + e_t = y_t - \hat{S}_t$. A linear regression model is created to re-estimate the trend component. The coefficients and the intercepts of both of the patterns are calculated using the code of "*coef = lm.coef_[0][0] intercept = lm.intercept_[0]*". The outputs for monthly and daily time series are listed as following.

	Monthly	Daily
Coefficient	0.002	0.0001
Intercept	8.11	8.10

Table 2

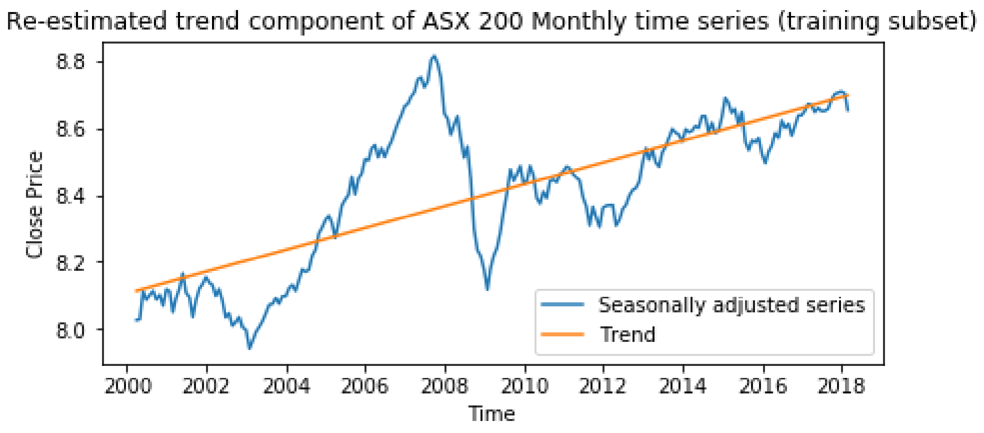


Figure 11

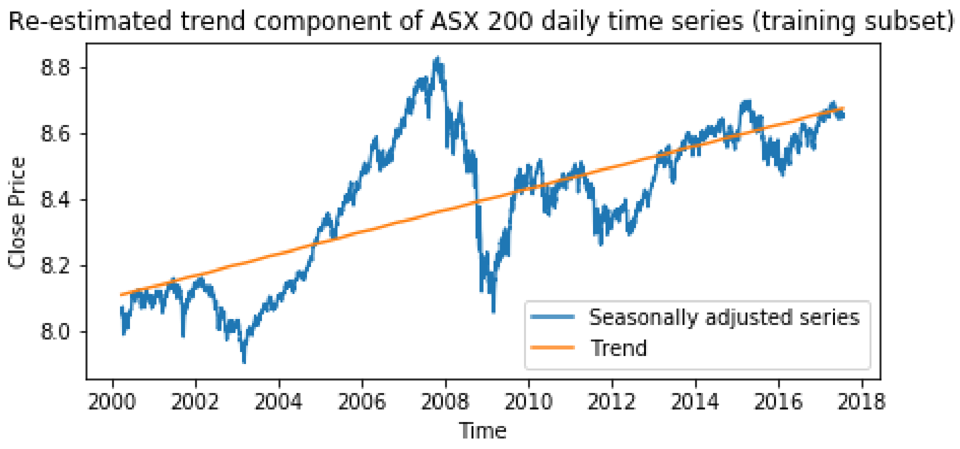


Figure 12

Furthermore, cycle component $\widehat{C}_t$ is calculated by subtracting trend and seasonal components from the original time series: $Y_t - \hat{S}_t - \hat{T}_t$ and the noises of cycle component are removed by doing a moving average. Errors are estimated by removing the trend, cycle, and seasonal components: $\hat{\varepsilon} = \hat{Y}_t - \hat{T}_t - \hat{S}_t - \hat{C}_t$

After doing all steps above, the model is ready to forecast. Closing prices in validation subset are predicted by adding trend and seasonal components to identify the accuracy of this method.

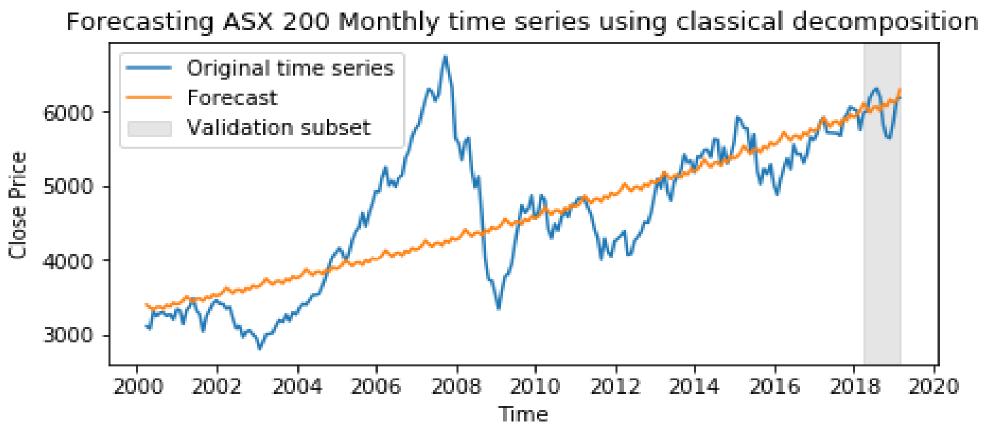


Figure 13

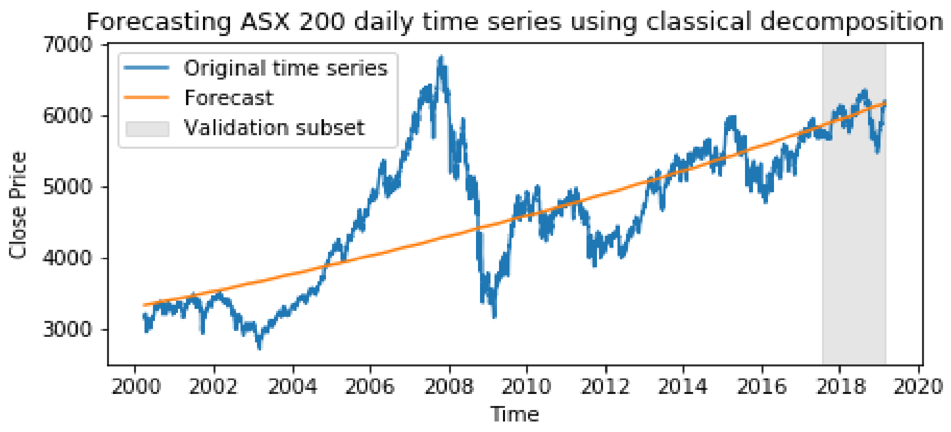


Figure 14

The Results for the Decomposition methods for the Validation and Training subset for Monthly and daily data are as follows.As it's difficult to identify the seasonal components in financial time series, decomposition method cannot provide very accurate results. The RMSE and MAE is huge for both training and validation subset.

Decomposition	RMSE	MAE	MAPE
Validation Subset (Monthly)	258.61	219.14	3.69
Training Subset (Monthly)	689.6	478.09	10.43
Validation Subset (Daily)	201.33	158.8	2.69
Training Subset (Daily)	702.7	494.13	10.81

Table 3

5.2 Exponential Smoothing

5.2.1 Model Description

Exponential smoothing is a method to smooth time series data which assigns exponentially decreasing weights overtime (Holt, 2004). More weights are assigned to newer data because it can be seen as more relevant. There are three main types of exponential smoothing which are simple exponential smoothing, Holt's Linear exponential smoothing and Holt's winter exponential smoothing. Simple exponential smoothing requires a smoothing factor denoted by α which determines the weights for observations. The value of α is usually given between 0 and 1 and the larger α, the more the observation is taken into account to make a prediction. Holt's Linear model reveals trend by adding a factor denoted by b on simple exponential smoothing. Holt's Winter model is an extension of Holt's Linear which add a factor of seasonality denoted by γ. Therefore, it seems exponential smoothing is a model that fits many types of data set and also easy to understand when we make a prediction.

5.2.2 Motivation

Exponential smoothing assigns reasonable weights as it pays more attention to the recent data which is more relevant to our prediction whereas simple moving average the past observations are weighted equally which causes the forecast less accurate. Furthermore, exponential smoothing easy to explain the principle and not much complicated to apply in python. Therefore, it becomes one of our choice to achieve the final forecast. As simple exponential smoothing does not perform well for the time series we want to predict because there is a trend in the data which has already revealed before. Therefore, Holt's Linear model and Holt's Winter model would be used to make the prediction in this report.

5.2.3 Preprocessing Steps

Exponential smoothing is not a complicated model that need much transformation for the data set. Using original data to fit the model is enough.

5.2.4 Model fitting

Holt's Linear Model

Holt's Linear model is built by applied the following code on the training subset:

```
holts_linear = Holt(training_subset).fit()
training_ts = holts_linear.fittedvalues
holts_linear.params
```

Then we use the model to predict in validation subset and plot the forecast for monthly data and daily data in respectively.

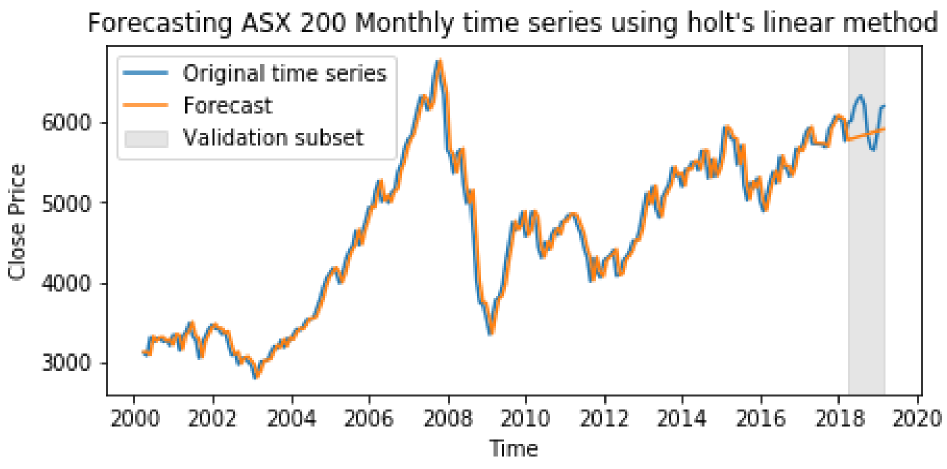


Figure 15

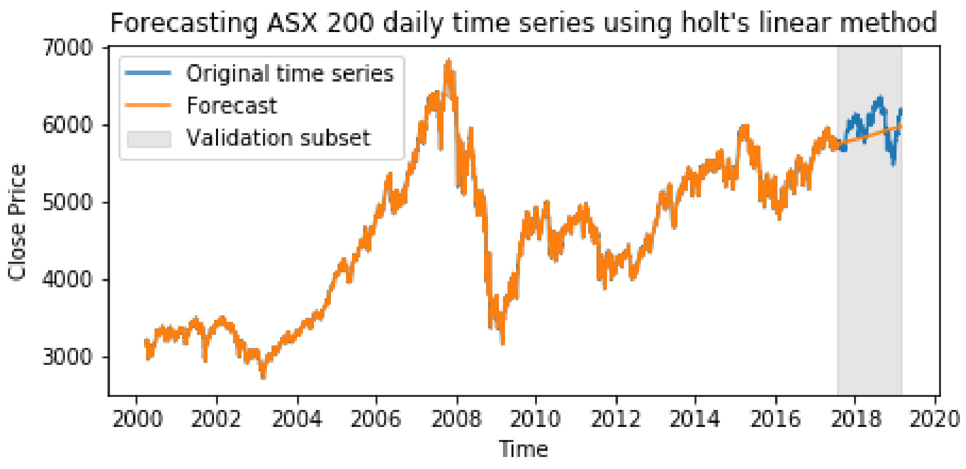


Figure 16

Results For the Holt's Linear Method For Daily and Monthly Data:

Holt's Linear Method	**RMSE**	**MAE**	**MAPE**
Daily Training Subset	48.69	33.33	0.73
Daily Validation Subset	223.02	185.59	3.07
Monthly Training Subset	171.47	133.09	2.91
Monthly Validation Subset	304.31	265.79	4.31

Table 4

Holt's Winter Model

Although it looks like there does not exist any seasonality in the data set as what we revealed in decomposition method. We still try to do both of additive and multiplicative method for Holt's Winter model to see if it can make a better forecast. The periods that used in this model are the same as which we used to do moving average before. For monthly data m is 12 for daily is 5. The following code is performed on training subset to achieve the model.

For additive seasonality:

```
holt_winters_add = ExponentialSmoothing(training_subset,
seasonal_periods = 12, trend = 'add', seasonal = 'add').fit()
training_ts_add = holt_winters_add.fittedvalues
holt_winters_add = ExponentialSmoothing(training_subset,
seasonal_periods = 5, trend = 'add', seasonal = 'add').fit()
training_ts_add = holt_winters_add.fittedvalues
```

Same is repeated for for multiplicative seasonality as well.

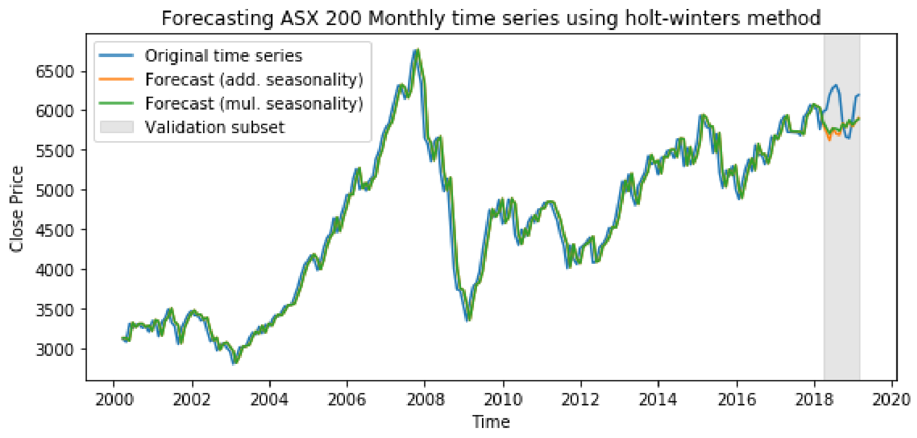


Figure 17

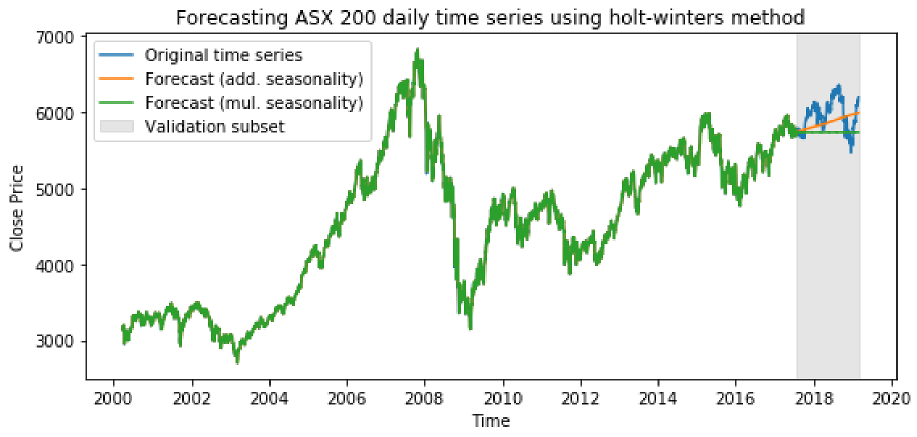


Figure 18

Results for the Holt's Winter Method with the Additive and multiplicative seasonality for Monthly and Daily Data:

		RMSE	**MAE**	**MAPE**
Additive Seasonality	Daily Training Subset	48.66	33.35	0.73
	Daily Validation Subset	219.05	182.69	3.02
	Monthly Training Subset	176.84	133.82	2.95
	Monthly Validation Subset	365.53	306.53	5.01

Multiplicative Seasonality	Daily Training Subset	48.63	33.38	0.73
	Daily Validation Subset	305.49	250.17	4.1
	Monthly Training Subset	164.94	128.8	2.86
	Monthly Validation Subset	335.66	283.87	4.59

Table 5

Holt's Winter method also involves seasonal components. Therefore, the RMSE and MSE are huge for validation subset.

5.3 ARIMA

5.3.1 Model Description

Autoregression Integrated Moving Average (ARIMA) is an advanced statistical model which is widely used in analysing and forecasting stationary time series data. However, if the data is non-stationary, ARIMA still can be used if it can be made to be stationary by one or more times of differencing steps. ARIMA is based on the regression analysis which measures the strength of the dependent and the independent variable. AR means autoregression which is a process of regressing a variable on past values of itself. MA is moving average which involves constructing a linear combination of error terms whose values occurred contemporaneously and at various times in the past. I is integrated indicates that the data values have been replaced with the difference between their values and the previous values (Meyler et al,1998). SARIMA is a extension to ARIMA model which supports to analyse and predict for the time series that contains seasonal components.

SARIMA is usually denoted by $(p,d,q)(P,D,Q)_m$ where:
p: measures the number of lag observations in the model.
d: is the total number of times the data is differenced.
q: represents the order of the moving average.
(P, D, Q) are seasonal part of the model where P, D and Q refer to the order of seasonal AR, I and MA parts while m is the number of seasonal periods.

5.3.2 Motivation

Advanced models usually make a good prediction in sample training. Nevertheless, the prediction that they generate for test data in practice may not perform better than simpler models due to their nature of complexity. SARIMA is more flexible than the simple statistical models such as linear regression or exponential smoothing to capture the relationships in the time series. However, it is still easy enough and will not lead to overfitting. Therefore, SARIMA becomes one of our choices to predict the financial time series.

5.3.3 Preprocessing Steps

Firstly, a training and validation subset is split. We use the last one year's data from daily and monthly data set as validation subset. The rest of data would be contained in training subset to build the model.

```
# Splitting data into training and validation subsets
```

```
training_subset = daily_ts[:-365]
validation_subset = daily_ts[-365:]
```

In addition to this, we check the stationary of data set as ARIMA is only able to deal with stationary time series. Monthly and daily time series are plotted to capture any non-stationary features visually. It can be observed that the patterns shown in monthly and daily diagrams are not centered around fixed set of value. In addition, it also can be observed the that mean of the time series is not equals to 0. Both of these two features are evidences of non-stationary time series.

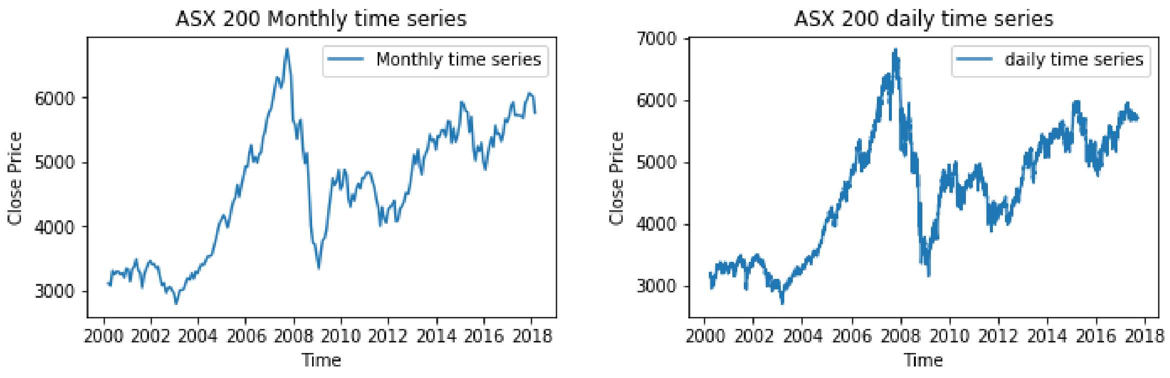


Figure 19

Then autocorrelation function (ACF) and partial autocorrelation function (PACF) are plotted to verify the conclusion we demonstrated above. Figure 17 shows the ACF and PACF for monthly time series and figure 18 shows ACF and PACF for daily time series. It can be observed that diagrams of ACF die down extremely slow which proves the conclusion we obtained visually that both of monthly and daily time series are non-stationary.

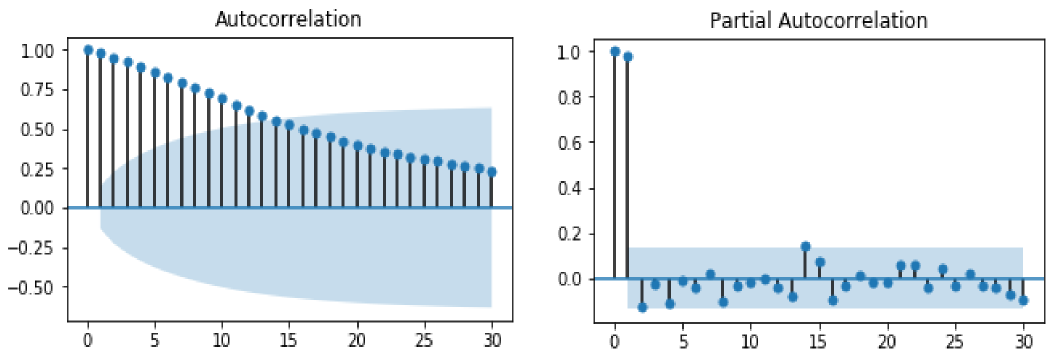


Figure 20

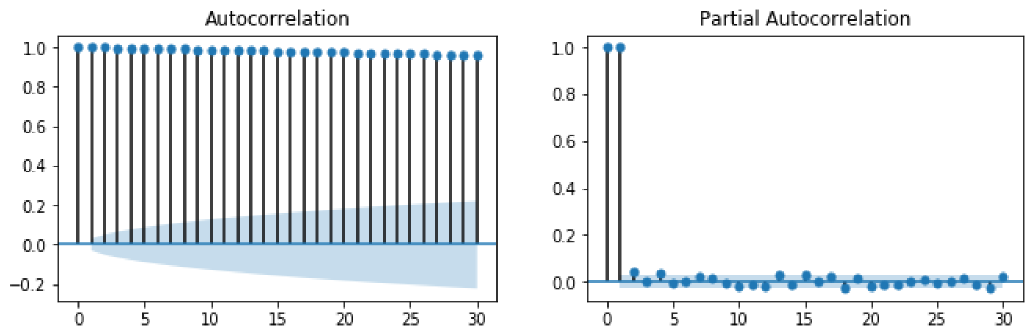


Figure 21

In addition, adfullers test is applied to check the stationary in a mathmatical way: *result = adfuller(ts).* P-value that we get from the output is 0.12 which proves that the time series is non-stationary since p-value is larger than 0.05.

In order to fit the model, data transformation is required. We take a log transformation for time series to stabilize the variance and then try first order differencing to create new time series. Two new time series are plotted as figure 19. It can be observed that the data is not centered around mean. It seems they are stationary after the first transformation.

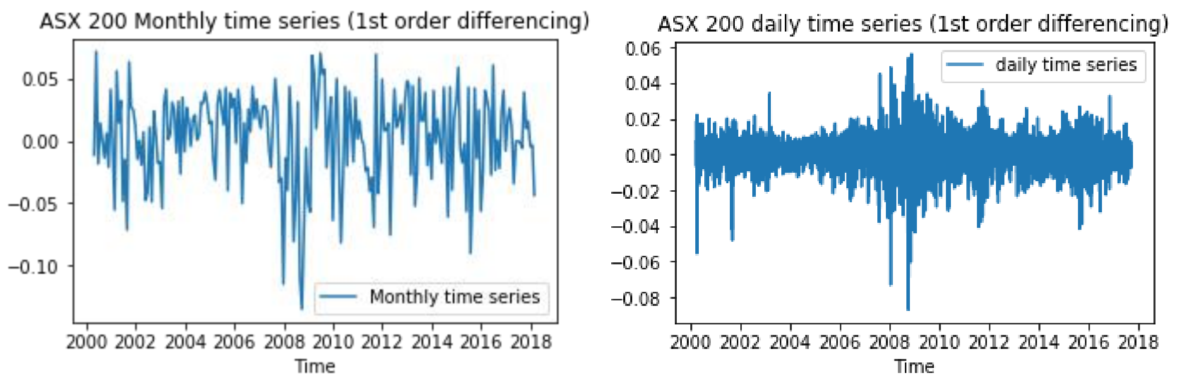


Figure 22

Nevertheless, we still plotted ACF and PACF for both of new time series to verify whether their stationarity. Figure 20 and figure 21 demonstrate ACF and PACF of monthly and daily time series after transformation in respectively. Both ACF and PACF cut down fairly quickly which means the time series is stationary after transformation. In addition, it is important to check the orders at which ACF and PACF cut off. The order of ACF will be lag q of ARIMA and the order of PACF will be the lag p of ARIMA.

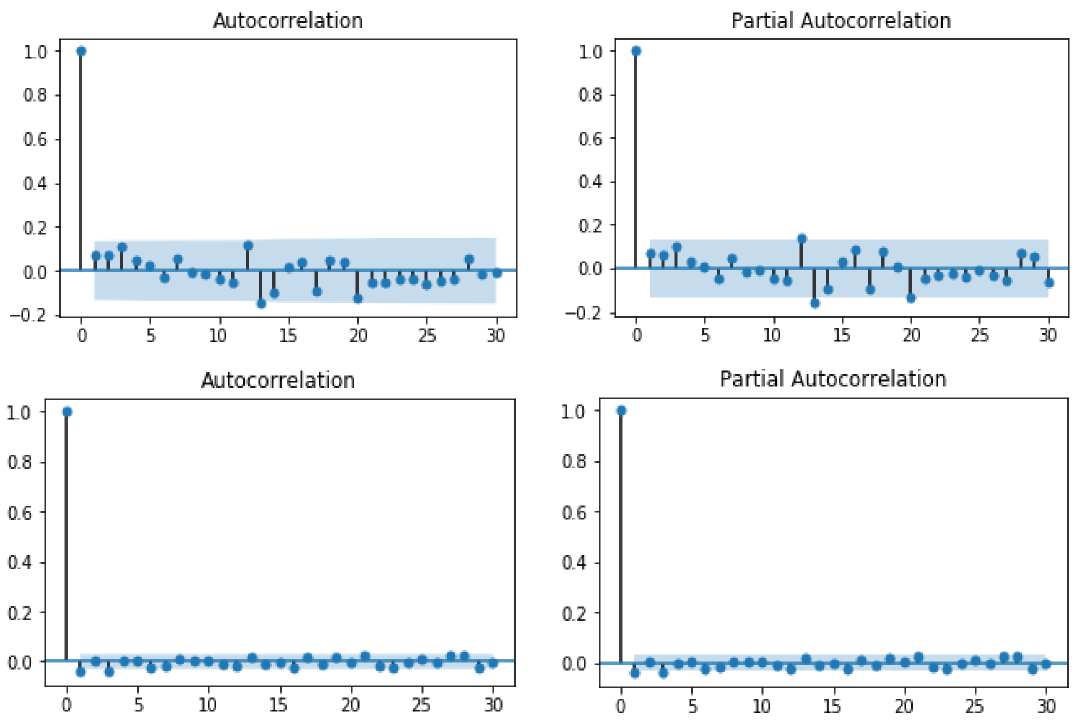


Figure 23

Furthermore, adfuller is applied again to check stationarity in the mathematical way. P-value becomes 0 this time. The time series is stationary and ready for next step experiment.

5.3.4 Model fitting

In the experimental of building model, two models are built as we use two different methods to tune parameters of SARIMA. One model is subjective as parameters of the first model would be chosen by observing ACF and PACF diagrams visually. Another model is more objective as it depends on auto ARIMA packages and all parameters are automatically tuned to optimize the minimum AIC and BIC.

Manually fitting model

For monthly time series, as we have already done a first order differencing and the data became stationary after our transformation. That means the correct order differencing has been selected as 1(ie d=1). In addition, p and q of monthly time series have been selected as 0 since all the subsequent values after lag 0 are mostly 0's. Furthermore, it can be observed that both ACF and PACF have a lag that out of the shadow period. That means, the monthly data set has seasonality although we cannot observe it directly from the seasonal plot. As P is determined by PACF and Q is determined by ACF, the values of both P and Q are chosen as 1. Therefore, the model of SARIMA $(0,1,0)(1,0,1)_{12}$ would be performed on monthly data set by executing code of:

```
model_ARIMA = ARIMA(training_subset_log, order = (0, 1, 0),(1,0,1)[12]
```

For daily time series, the value of d is also 1 as we have done the first order differencing to make the time series stationary. In addition, p and q of daily time series is selected as 3 since both of ACF and PACF entirely cut down after lag 3. Furthermore, we cannot observe any seasonal component from the ACF and PACF plots. Therefore, the model of ARIMA (3, 1,3) is selected to make a forecast on daily time series which can be done by:

```
model_ARIMA = ARIMA(training_subset_log, order = (3, 1, 3)
```

After the model is fitted, the residual of time series is plotted to check whether the model captures all features from the data. As there is no pattern in the residual plots, the models we built before are considered capturing all features from data.

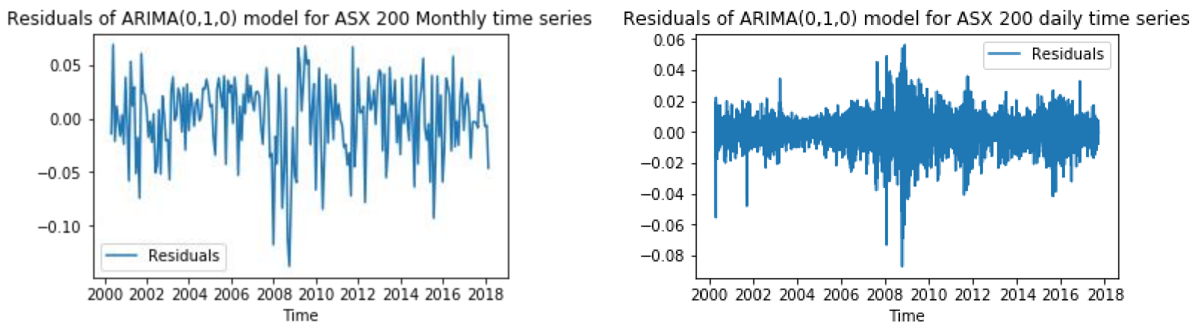


Figure 24

Finally, the validation set is fit in the model we built to make forecasts and measure errors. It must be noted that as a log transformation is performed before we build the model, it is important to take an inverse log on the outputs which generated by predicting the validation set.

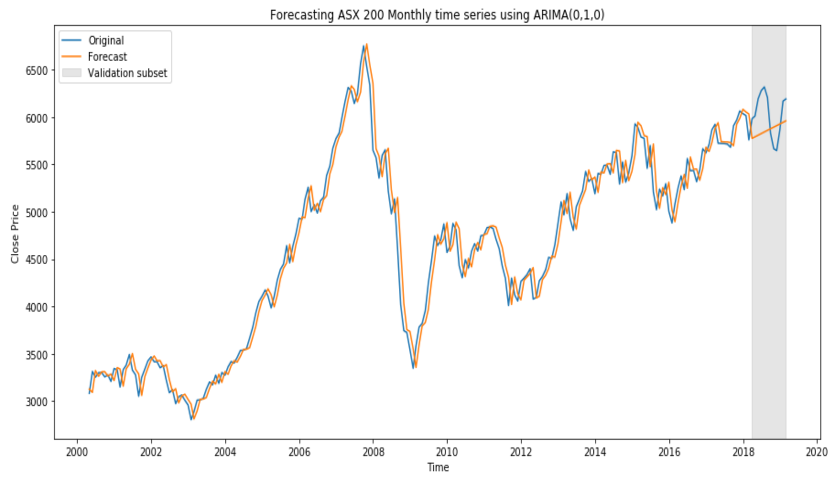


Figure 25

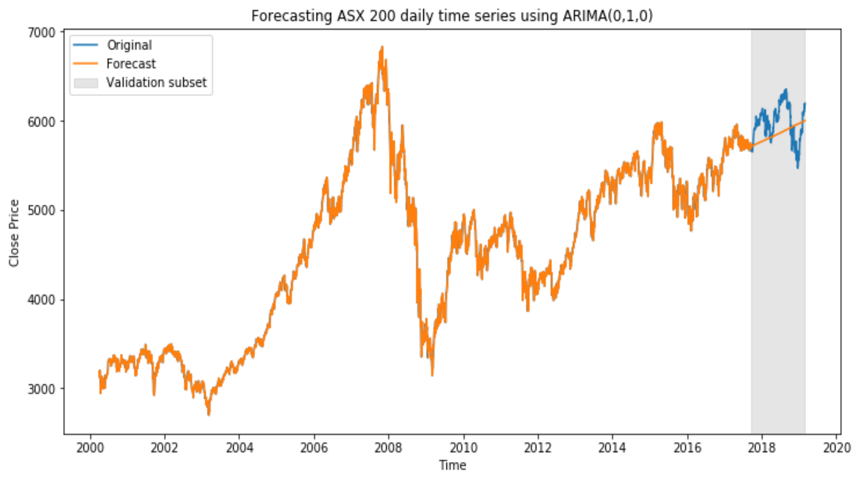


Figure 26

Auto-fitting model

Through the function of auto ARIMA packages, a list of parameters were generated with values of AIC and BIC following. The best model chosen by auto ARIMA for monthly and daily are respectively(0,1,1)(0,0,1)[12] and (1,1,0) which we have shown a highlight in the table of outcomes. The model is automatically built by executing code of:
`train_model = autoARIMA_model.fit(training_subset_log)`

Monthly		Daily	
Order	**Seasonal Order**	**Order**	**Seasonal Order**
(0, 1, 0)	(0, 0, 0, 12)	(0, 1, 0)	(0, 0, 0, 5)
(1, 1, 0)	(1, 0, 0, 12)	(1, 1, 0)	(1, 0, 0, 5)
(0, 1, 1)	(0, 0, 1, 12)	(0, 1, 1)	(0, 0, 1, 5)
(0, 1, 1)	(1, 0, 1, 12)	(1, 1, 0)	(0, 0, 0, 5)
(0, 1, 1)	(0, 0, 0, 12)	(1, 1, 0)	(0, 0, 1, 5)
(0, 1, 1)	(0, 0, 2, 12)	(1, 1, 0)	(1, 0, 1, 5)
(0, 1, 1)	(1, 0, 2, 12)	(2, 1, 0)	(0, 0, 0, 5)
(1, 1, 1)	(0, 0, 1, 12)	(1, 1, 1)	(0, 0, 0, 5)
(0, 1, 0)	(0, 0, 1, 12)	(2, 1, 1)	(0, 0, 0, 5)

(0, 1, 2)	(0, 0, 1, 12)		
(1, 1, 2)	(0, 0, 1, 12)		

Table 6

Validation subset is used to make forecasts and measure errors. However, it is important to convert the predictions of validation subset into time series and then take an inverse log of the time series. The forecasts of time series is plotted as figure 25.

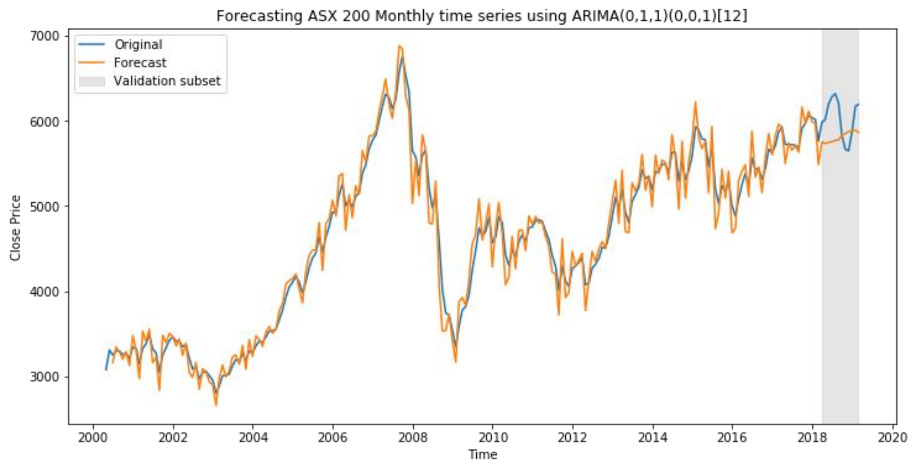


Figure 27

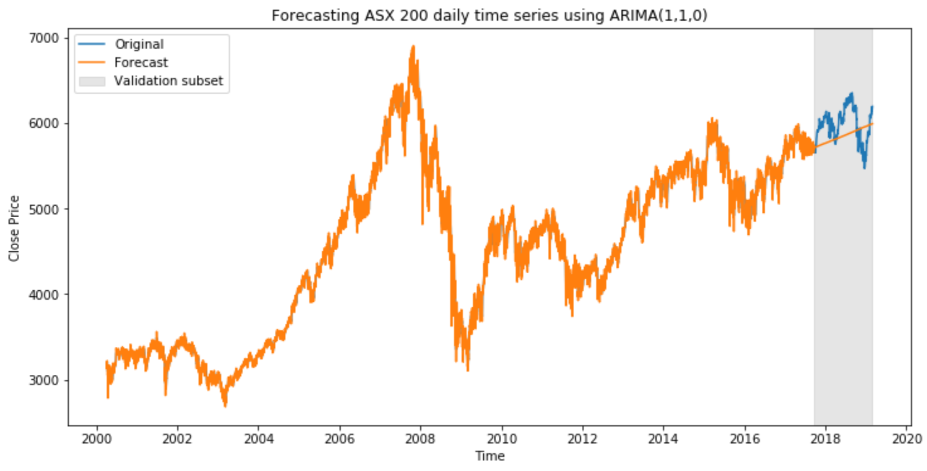


Figure 28

Results for the Manual Fitting and Auto Fitting ARIMA model For Monthly and Daily Data:

		RMSE	**MAE**	**MAPE**
Manually Fitting	Daily Training Subset	46.35	32.15	0.7
	Daily Validation Subset	245.21	214.78	3.56
	Monthly Training Subset	172.16	133.88	2.93
	Monthly Validation Subset	292.45	262.16	4.35
Auto Filling	Daily Training Subset	46.14	32.11	0.7
	Daily Validation Subset	246.48	215.82	3.58
	Monthly Training Subset	165.22	128.65	2.83
	Monthly Validation Subset	337.94	291.97	4.76

Table 7

5.4 LSTM

5.4.1 Model Description

Long short-term memory (LSTM) is a special kind of recurrent neural network (RNN) which is able to avoid long-term dependency problem in sequence prediction (Yan & Ouyang, 2018). The LSTM unit contains a cell which stores or reads information. In addition, there are also three gates in LSRM unit called input gate, output gate and forget gate which controls whether information will flow into or out of the cell. The program usually generates the best outputs based on the accumulated experience that it learnt automatically.

5.4.2 Motivation

As the financial time series has dynamic nature which is really complicated, it is not appropriate to reflect the noises simply by analytical equations with parameters if high accuracy is required. Therefore, it is not much reliable if we use traditional prediction model.

Moreover, our traditional models follow a linear pattern to analyse the data whereas NN will have a non-linear (because of the activation functions used) to analyze the data and present the output. Neural networks are noticed as they are good at processing non-linear and non-stationary data. Neural networks consist many different models such as deep neural networks (DNN), recurrent neural networks (RNN), long short-term memory (LSTM). It has been indicated that the learning process of DNN and RNN always contains all the historical information of the previous time series which becomes difficult for these two models to learn long-term sequence (Ergen & Kozat, 2018). LSTM contains memory modules which have a selection on remembering patterns to achieve better forecast for long durations time series. Therefore, LSTM is considered prior to other models to process our financial data set and make a forecast.

5.4.3 Preprocessing Steps

In order to fit Keras neural network, the data set is transformed from data frame into array. In addition, the data is scaled to the range between 0 and 1 which are done by using the following code as neural networks perform best with scaled data especially when we use the sigmoid or tanh activation function.

```
scaler = MinMaxScaler(feature_range=(0, 1))
data = scaler.fit_transform(data)
```

In addition, both of monthly data set and daily data set are split into training subset and test subset. We use 70% of data for training and 30% of data for test. Furthermore, the size of time window is decided in preprocessing steps. The value we give to time window of daily data set is 15 which means using last 15 values to predict the next one. It can be given any value, but it is better to assign a value that larger than the seasonal period of time series because it seems more reliable when we use more than one seasonal period to predict the next value. For monthly data set, we use 4 as the size of time window. After choosing a value of time window, As both training subset and test subset are in the time series form, we split training block in the size of time window and picked up the next value as prediction. Then the split data is reshaped as we are going to use Keras package to build our model. All these steps can be easily done by executing the following code (We use daily data set as sample).

```
train_size = int(len(data)*0.7)
test_size = len(data) - train_size
Dtrain, Dtest = data[0:train_size,:], data[train_size:len(data),:]
time_window =15
Xtrain, Ytrain = [], []
for i in range(len(Dtrain) - time_window -1):
    Xtrain.append(Dtrain[i:(i+time_window), 0])
    Ytrain.append(Dtrain[i+time_window, 0])
Xtrain = np.array(Xtrain)
Ytrain = np.array(Ytrain)
```

5.4.4 Model fitting

For the LSTM models firstly, we need to install the Keras packages, which is a high-level approach to build, train and predict using the neural networks. Keras models are built on libraries like Tensorflow or Theano which assists us to prototype the networks quickly.

After that, we set the random seed as 70 so that we can get the same results every time. Then we load the data and convert the data by (.values) to an array data type because the Keras operates on the array data type and not on the series or dataframe.

Then we need to scale the data using the Min-Max Scaler function so that the data ranges between 0-1, and then converting the data into the train and testing subset. The process is detailedly explained in the preprocessing part.After that, we define our model and choose the parameters to see the training and testing errors. Various Parameters which needs to be tuned for the LSTM model are as follows:

Batch Size: Batch size refers to the total number of data existing in a single batch. For our model, the Batch size is 4779 for the daily data and 228 for the daily data.
Number of Batches: Number of Batches refers to the total amount of batches we are using in the model. For our model, we have chosen 1 batch because the data is limited.

Time Window*:* refers to the time span which is taken to predict one step ahead forecast. For example, if we are taking the time window for 15 days, it means that the past 15 days data is used for predicting the 1 step ahead prediction. In our report, we have taken different time windows for the month and the daily data to see the effect on the training and test error.

Neurons: The functions for the neuron is to take in some inputs and it fires an output. We need to apply for a trade-off theory in taking the number of neurons as increasing the number of neurons can lead to overfitting, decreasing can lead to poor accuracy. In our Report, we have taken different values for the neurons to see the effects.

Epoch: is a forward/backward pass of all batches. In our model, we have tried to see the effect of the Epoch. We observed increasing the epoch usually take longer time but we saw a bit of accuracy in our models as it refines the model parameters each time.

Dropouts: (Browniee ,2017), explained that the drop out parameters are used where the input and recurrent connections to LSTM units are probabilistically excluded from activation and the assigned weights are updated when we train the model. Different values for dropouts were checked in the report to see the effects of overfitting and accuracy.

Optimiser: Different Optimiser like Adam, Sgd and Adadelta were tuned in the model.

Sgd refers to Stochastic gradient descent optimizer. The purposes for the SGD optimiser is to include support for momentum, learning rate decay, and Nesterov momentum. Adadelta is based on a moving window of gradient updates and not considering the accumulation of all the past gradients. Adam Optimiser is considered as the default optimiser.In our model we tried different optimiser to see the differences. We need to mention the optimiser we are choosing in our model creation with the following code: model.compile(optimizer = 'adam', loss = 'mean_squared_error',metrics=['accuracy']).

The Training and the Test subsets are then divided into the Validation Split, for our model we did the Validation Split of 0.05.

Keeping the above parameters in mind we built an algorithm for the model and changed the various parameters to see the effects.After that we transformed the Min-Max Scaler values to the original values and run them in loop to make the one step prediction and finally plot the predictions. The following tables shows different values for the parameters with their respective test and training errors for the daily and the monthly data.

Results for the Daily Data are as follows:

No. of Neurons	Time Window	Drop Out	Optimiser	Epoch	Rmse Train	Rmse Test
100	3	0.6	Sgd	300	55.32	58.92
80	60	0.8	Sgd	100	80.07	75.74
50	60	0.4	Adam	30	105.47	176.08
100	18	0.8	Adam	100	61.43	74.66
200	60	0.2	Adadelta	50	57.89	58.06
80	6	0.6	Adam	50	49.12	49.16
100	18	0.8	Sgd	30	0.02	0.02
50	15	0.4	Adadelta	30	60.44	65.35

Table 8

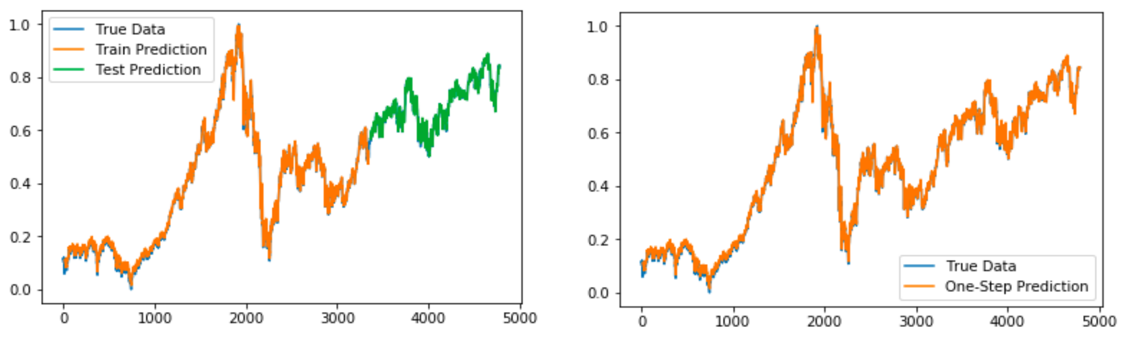
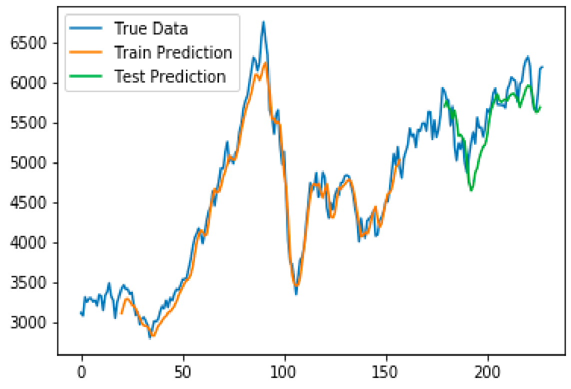


Figure 29

Results for the Monthly Data are as follows:

No. of Neurons	Time Window	Drop Out	Layers	Optimiser	Epoch	Rmse Train	Rmse Test
100	10	0.2	2	Adadelta	30	261.87	389.56
50	20	0.4	2	Adam	200	165.31	276.31
50	12	0.2	2	Sgd	20	481.4	442.81
200	10	0.8	2	Adam	300	196.77	276.54
70	4	0.3	4	Adam	50	327.06	638.6
150	7	0.8	3	Sgd	50	924.86	1360.09
70	10	0.3	4	Adam	50	233.48	250.21

Table 9



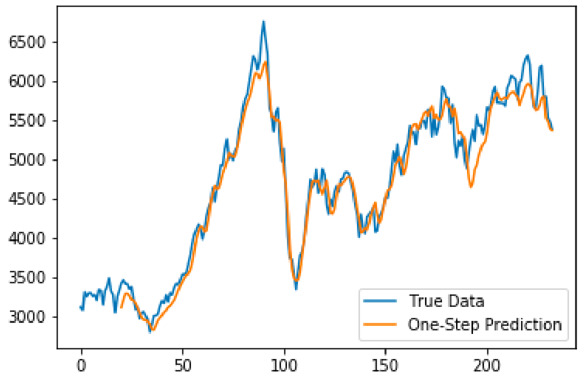


Figure 30

6 Results

6.1 Model Selection

The model is mainly selected based on the performance on RMSE and MAE. Nevertheless, we also analyzed the nature of financial time series and considered whether the model is well-suited for our time series since it will never make a good prediction if we cannot make a multiple consideration on the model.

According to our analysis, LSTM gives the lowest RMSE and MAE for both daily and monthly time series. In addition, the stock index is easily influenced by financial statements, corporate announcements etc. Due to the complicated and dynamic nature of financial time series it is very difficult to capture features that dominated the time series. LSTM selects the useful information automatically and form the relationships among them better and better by using accumulated experiences. It determines the next value that does not rely much on the last value. Therefore, it is decided to be the best model to predict financial time series for both daily and monthly data set.

6.2 Daily and Monthly Forecasts

	June	July	August	September	October
Monthly	6289.56	6299.56	6354.45	6374.22	6344.06

	27/05	28/05	29/05	30/05	31/05
Daily	6456.97	6457.93	6458.9	6459.86	6460.83

Table 10

7 Conclusion

In this report, a series of prediction analysis is done by applying different models. Finally, we do find some features in financial time series and make results based on the best model we selected. However, financial time series is not simply influenced by features which can be seen from the diagram visually.

We explored a lot of various techniques to predict the daily and monthly financial time series. We managed to improve upon our results by tuning the hyperparameters and apply advanced models such as ARIMA and LSTM. But the fact there is no clear seasonality and so much variation in the stock exchange dataset that even the advanced models could not improve the prediction accuracy beyond a certain point. Further research is required if a very accurate result is needed.

8 References

- ASX200 List. (2019). *ASX Top 200 Companies*. Retrieved from https://www.asx200list.com/

- Australian Securities Exchange. (2017). *ASX Australian Investor Study*. Retrieved from https://www.asx.com.au/education/2017-asx-investor-study.htm

- Ayodele, A. A., Adewumi, A. O., & Charles, K. A. (2014). Comparison of ARIMA and artificial neural networks models for stock price prediction. *Journal of Applied Mathematics*. http://dx.doi.org.ezproxy1.library.usyd.edu.au/10.1155/2014/614342

- Browniee, J. (2017). How to Use Dropout with LSTM Networks for Time Series Forecasting [Blog]. Retrieved from https://machinelearningmastery.com/use-dropout-lstm-networks-time-series-forecasting/

- Ergen, T., & Kozat, S. (2018). Efficient Online Learning Algorithms Based on LSTM Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 29(8), 3772–3783. https://doi.org/10.1109/TNNLS.2017.2741598

- Holt, C. (2004). Forecasting seasonals and trends by exponentially weighted moving averages. (Author Abstract). *Journal of Economic and Social Measurement*, 29(1 3), 123–125. https://doi.org/10.3233/JEM-2004-0211

- Hyndman, R.J., & Athanasopoulos, G. (2018) Forecasting: principles and practice, 2nd edition, OTexts: Melbourne, Australia. OTexts.com/fpp2

- Meyler, A., Kenny, G.& Terry,T. (1998). Forecasting irish inflation using ARIMA models. *Central Bank and Financial Services Authority of Ireland Technical Paper Series*. Retrieved from https://mpra.ub.uni-muenchen.de/id/eprint/11359

- Namin, S.S.& Namin, A.S. (2018). Forcasting economic and financial time series: ARIMA vs LSTM. Retrieved from https://arxiv.org/ftp/arxiv/papers/1803/1803.06386.pdf

- Nelson, D. & Pereira, A. & de Oliveira, R. (2017). Stock market's price movement prediction with LSTM neural networks. Retrieved from https://www.researchgate.net/publication/318329563_Stock_market's_price_movement_prediction_with_LSTM_neural_networks

- Wang, J., Wang, J., Zhang, Z., & Guo, S. (2012). Stock index forecasting based on a hybrid model. *Omega*, 40(6), 758–766. https://doi.org/10.1016/j.omega.2011.07.008

- Yan, H. & Ouyang, H. (2018). Financial Time Series Prediction Based on Deep Learning. *Wireless Pers Commun*, 102(2), 683-700. https://doi-org.ezproxy1.library.usyd.edu.au/10.1007/s11277-017-5086-2

Appendix A

```
# Task 1 & 2 (Daily EDA)

# In[1]:


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
from matplotlib.pylab import rcParams
from sklearn.linear_model import LinearRegression


# In[2]:


# Reading data
daily_data = pd.read_csv('ASX200Daily.csv')
print(daily_data.head())


# In[3]:


# Converting to a time series
daily_data['Date'] = pd.to_datetime(daily_data['Date'])
daily_data.set_index('Date', inplace = True)
daily_ts = pd.Series(daily_data['Close'])
print(daily_ts.head())


# In[4]:


# Checking data for missing values
print('Before data cleaning ->')
print('No. of missing values in daily data:', sum(daily_ts.isna()))


# In[5]:


# Removing missing values
daily_ts.dropna(inplace=True)


# In[6]:


# Checking data for missing values again
print('After data cleaning ->')
print('No. of missing values in daily data:', sum(daily_ts.isna()))
```

```
# In[7]:


# Descriptive statistics
print('Daily data ->')
print(daily_ts.describe())


# In[8]:


# Setting figure width and height
rcParams['figure.figsize'] = 15, 7

# Plotting time series
plt.plot(daily_ts, label = "Daily time series")
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.title('ASX 200 Daily time series')
plt.legend()


# #### Zooming in on data to see if there are any significant
patterns

# In[9]:


# Plotting a random year of the dataset
plt.plot(daily_ts['2017'], label = "Daily time series")
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.title('ASX 200 Daily time series')
plt.legend()


# In[10]:


# Plotting a random year-month of the dataset
plt.plot(daily_ts['2015-07'], label = "Daily time series")
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.title('ASX 200 Daily time series')
plt.legend()


# In[11]:


# Smoothing data by using CMA-5 to calculate the initial trend-cycle
estimate
trend_cycle = daily_ts.rolling(5, center=True).mean()
```

```
plt.plot(daily_ts, label = "Observed")
plt.plot(trend_cycle, label = "Smoothed")
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.title('Initial trend-cycle component of the ASX 200 Daily time
series')
plt.legend()


# In[12]:


# Calculating the seasonal-irregular component
seasonal = daily_ts - trend_cycle

plt.plot(seasonal, label = "Seasonal-irregular component")
plt.xlabel('Time')
plt.title('Seasonal-irregular component of the ASX 200 Daily time
series')
plt.legend()




# Task 1 & 2 (Monthly EDA)

# In[1]:


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
from matplotlib.pylab import rcParams
from sklearn.linear_model import LinearRegression


# In[2]:


# Reading data
monthly_data = pd.read_csv('ASX200Monthly.csv')
print(monthly_data.head())


# In[3]:


# Converting to a time series
monthly_data['Date'] = pd.to_datetime(monthly_data['Date'])
monthly_data.set_index('Date', inplace = True)
monthly_ts = pd.Series(monthly_data['Close'])
print(monthly_ts.head())


# In[4]:
```

```
# Checking data for missing values
print('Before data cleaning ->')
print('No. of missing values in monthly data:',
sum(monthly_ts.isna()))


# In[5]:


# Removing missing values
monthly_ts.dropna(inplace=True)


# In[6]:


# Checking data for missing values again
print('After data cleaning ->')
print('No. of missing values in monthly data:',
sum(monthly_ts.isna()))


# In[7]:


# Descriptive statistics
print('Monthly data ->')
print(monthly_ts.describe())


# In[8]:


# Setting figure width and height
rcParams['figure.figsize'] = 15, 7

# Plotting time series
plt.plot(monthly_ts, label = "Monthly time series")
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.title('ASX 200 Monthly time series')
plt.legend()


# In[9]:


# Smoothing data by using CMA-12 to calculate the initial trend-
cycle estimate
trend_cycle = monthly_ts.rolling(2, center=True).mean().rolling(12,
center=True).mean().shift(-1)

plt.plot(monthly_ts, label = "Observed")
plt.plot(trend_cycle, label = "Smoothed")
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.title('Initial trend-cycle component of the ASX 200 Monthly time
series')
plt.legend()


# In[10]:


# Calculating the seasonal-irregular component
seasonal = monthly_ts - trend_cycle

plt.plot(seasonal, label = "Seasonal-irregular component")
plt.xlabel('Time')
plt.title('Seasonal-irregular component of the ASX 200 Monthly time
series')
plt.legend()




# Task 3 (Daily Decomposition)

# In[1]:


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
from matplotlib.pylab import rcParams
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error


# In[2]:


# Reading data
daily_data = pd.read_csv('ASX200Daily.csv')


# In[3]:


# Converting to a time series
daily_data['Date'] = pd.to_datetime(daily_data['Date'])
daily_data.set_index('Date', inplace = True)
daily_ts = pd.Series(daily_data['Close'])
print(daily_ts.head())


# In[4]:
```

```
# Removing missing values
daily_ts.dropna(inplace=True)


# In[5]:


# Days of the week with closing price given
daily_ts.index.dayofweek.unique()


# In[6]:


# Removing rows with day of the week = 6 (Sunday)
# Only taking weekdays into account will give better predictions for
this assignment
daily_ts = daily_ts[daily_ts.index.dayofweek!=6]


# In[7]:


# Transforming time series by taking log
# It reduces variation in data by penalizing larger values more than
the smaller ones
daily_ts = np.log(daily_ts)


# In[8]:


# Splitting data into training and validation subsets
training_subset = daily_ts[:-365]
validation_subset = daily_ts[-365:]


# In[9]:


# Smoothing data by using CMA-5 to calculate the initial trend-cycle
estimate
trend_cycle = training_subset.rolling(5, center=True).mean()

# Setting figure width and height
rcParams['figure.figsize'] = 15, 7

plt.plot(training_subset, label = "Observed")
plt.plot(trend_cycle, label = "Smoothed")
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.title('Initial trend-cycle component of the ASX 200 daily time
series (training subset)')
plt.legend()
```

```
# In[10]:


# Calculating the seasonal-irregular component
seasonal = training_subset - trend_cycle

plt.plot(seasonal, label = "Seasonal-irregular component")
plt.xlabel('Time')
plt.title('Seasonal-irregular component of the ASX 200 daily time
series (training subset)')
plt.legend()


# In[11]:


# Calculating (un-normalized) seasonal indexes
seasonal_indexes = np.empty(5)
for i in range(0,5):
    seasonal_indexes[i] =
seasonal[seasonal.index.dayofweek==i].mean()

seasonal_indexes


# In[12]:


# Calculating normalized seasonal indexes
norm_seasonal_indexes = seasonal_indexes - seasonal_indexes.mean()
norm_seasonal_indexes


# In[13]:


# Repeating normalized seasonal indexes for the entire duration
seasonal_components = np.empty(training_subset.size)
for i,j in enumerate(training_subset.index.dayofweek):
    seasonal_components[i] = norm_seasonal_indexes[j]


# In[14]:


# Calculating seasonally adjusted series
seasonally_adjusted = training_subset - seasonal_components

plt.plot(training_subset, label = "Observed")
plt.plot(seasonally_adjusted, label = "Seasonally adjusted")
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.title('Seasonally adjusted ASX 200 daily time series (training
subset)')
plt.legend()
```

```
# In[15]:


# Dataframe for linear regression
daily_df = pd.DataFrame()
daily_df['Time'] = range(1, training_subset.size+1)
daily_df['Close'] = seasonally_adjusted.values
daily_df.head()


# In[16]:


# Creating a linear regression model
lm = LinearRegression()
lm.fit(daily_df[['Time']], daily_df[['Close']])


# In[17]:


# Coefficient and intercept values
coef = lm.coef_[0][0]
intercept = lm.intercept_[0]
print('Coefficient:', coef)
print('Intercept:', intercept)


# In[18]:


# Re-estimating trend
reestimated_trend = pd.Series(data=(intercept +
daily_df['Time']*coef).values, index=seasonally_adjusted.index)

plt.plot(seasonally_adjusted, label = "Seasonally adjusted series")
plt.plot(reestimated_trend, label = "Trend")
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.title('Re-estimated trend component of ASX 200 daily time series
(training subset)')
plt.legend()


# In[19]:


# Estimating cyclic component
trend_seasonal = reestimated_trend + seasonal_components
cyclic_component = training_subset - trend_seasonal

# Removing noise by smoothing data (MA-3)
cyclic = cyclic_component.rolling(3, center=True).mean()
```

```
plt.plot(cyclic_component, label = "Cyclic component")
plt.plot(cyclic, label = "Smoothed")
plt.xlabel('Time')
plt.title('Cyclic component of ASX 200 daily time series (training
subset)')
plt.legend()


# In[20]:


# Calculating remainder
remainder = training_subset - (cyclic + reestimated_trend +
seasonal_components)

plt.plot(remainder, label = "error/residual")
plt.xlabel('Time')
plt.title('Remainder (residual) of ASX 200 daily time series
(training subset)')
plt.legend()


# In[21]:


# Forecasting 'Close' price for next 365 days (validation subset)

training_ts = reestimated_trend + seasonal_components

t = np.empty(validation_subset.size)
for i in range(0,validation_subset.size):
    t[i] = intercept + (training_subset.size+1+i)*coef

seasonal = daily_ts - trend_cycle
seasonal_components = np.empty(daily_ts.size)
for i,j in enumerate(daily_ts.index.dayofweek):
    seasonal_components[i] = norm_seasonal_indexes[j]

predictions = validation_subset.copy()
for i in range(0,predictions.size):
    predictions[i] = t[i] +
seasonal_components[training_subset.size+i]


# In[22]:


# Appending fitted values with the predicted values
predicted_ts = training_ts.append(predictions)


# In[23]:


# Returning data to its original form by taking inverse of log
predicted_ts = np.exp(predicted_ts)
```

```
daily_ts = np.exp(daily_ts)
training_subset = np.exp(training_subset)
training_ts = np.exp(training_ts)
validation_subset = np.exp(validation_subset)
predictions = np.exp(predictions)


# In[24]:


plt.plot(daily_ts, label = "Original time series")
plt.plot(predicted_ts, label = "Forecast")
plt.axvspan(validation_subset.index[0],
validation_subset.index[validation_subset.size-1], color='grey',
alpha=0.2, label="Validation subset")
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.title('Forecasting ASX 200 daily time series using classical
decomposition')
plt.legend()


# In[25]:


# Measuring errors of training subset
RMSE = np.sqrt(mean_squared_error(training_subset, training_ts))
MAE = mean_absolute_error(training_subset, training_ts)
MAPE = np.mean(np.abs((training_subset - training_ts) /
training_subset)) * 100

print('Training subset ->')
print('Root Mean Squared Error (RMSE):', RMSE)
print('Mean Absolute Error (MAE):', MAE)
print('Mean Absolute Percentage Error (MAPE):', MAPE)


# In[26]:


# Measuring errors of validation subset
RMSE = np.sqrt(mean_squared_error(validation_subset, predictions))
MAE = mean_absolute_error(validation_subset, predictions)
MAPE = np.mean(np.abs((validation_subset - predictions) /
validation_subset)) * 100

print('Validation subset ->')
print('Root Mean Squared Error (RMSE):', RMSE)
print('Mean Absolute Error (MAE):', MAE)
print('Mean Absolute Percentage Error (MAPE):', MAPE)




# Task 3 (Monthly Decomposition)
# In[1]:


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
from matplotlib.pylab import rcParams
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
import math


# In[2]:


# Reading data
monthly_data = pd.read_csv('ASX200Monthly.csv')


# In[3]:


# Converting to a time series
monthly_data['Date'] = pd.to_datetime(monthly_data['Date'])
monthly_data.set_index('Date', inplace = True)
monthly_ts = pd.Series(monthly_data['Close'])
print(monthly_ts.head())


# In[4]:


# Removing missing values
monthly_ts.dropna(inplace=True)


# In[5]:


# Transforming time series by taking log
# It reduces variation in data by penalizing larger values more than
the smaller ones
monthly_ts = np.log(monthly_ts)


# In[6]:


# Splitting data into training and validation subsets
training_subset = monthly_ts[:-12]
validation_subset = monthly_ts[-12:]


# In[7]:
```

```
# Smoothing data by using CMA-12 to calculate the initial trend-
cycle estimate
trend_cycle = training_subset.rolling(2,
center=True).mean().rolling(12, center=True).mean().shift(-1)

# Setting figure width and height
rcParams['figure.figsize'] = 15, 7

plt.plot(training_subset, label = "Observed")
plt.plot(trend_cycle, label = "Smoothed")
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.title('Initial trend-cycle component of the ASX 200 Monthly time
series (training subset)')
plt.legend()


# In[8]:


# Calculating the seasonal-irregular component
seasonal = training_subset - trend_cycle

plt.plot(seasonal, label = "Seasonal-irregular component")
plt.xlabel('Time')
plt.title('Seasonal-irregular component of the ASX 200 Monthly time
series (training subset)')
plt.legend()


# In[9]:


# Calculating (un-normalized) seasonal indexes
seasonal_indexes = np.empty(12)
for i in range(0,12):
    seasonal_indexes[i] = seasonal[seasonal.index.month==i+1].mean()

seasonal_indexes


# In[10]:


# Calculating normalized seasonal indexes
norm_seasonal_indexes = seasonal_indexes - seasonal_indexes.mean()
norm_seasonal_indexes


# In[11]:


# Repeating normalized seasonal indexes for the entire duration
seasonal_components = np.empty(training_subset.size)
```

```
for i in range(0,training_subset.size):
    seasonal_components[i] = 
norm_seasonal_indexes[seasonal.index.month[i]-1]

plt.plot(seasonal_components, label = "Normalized")
plt.xlabel('Time (months)')
plt.title('Normalized seasonal indexes of ASX 200 Monthly time 
series')
plt.legend()


# In[12]:


# Calculating seasonally adjusted series
seasonally_adjusted = training_subset - seasonal_components

plt.plot(training_subset, label = "Observed")
plt.plot(seasonally_adjusted, label = "Seasonally adjusted")
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.title('Seasonally adjusted ASX 200 Monthly time series (training 
subset)')
plt.legend()


# In[13]:


# Dataframe for linear regression
monthly_df = pd.DataFrame()
monthly_df['Time'] = range(1, training_subset.size+1)
monthly_df['Close'] = seasonally_adjusted.values
monthly_df.head()


# In[14]:


# Creating a linear regression model
lm = LinearRegression()
lm.fit(monthly_df[['Time']], monthly_df[['Close']])


# In[15]:


# Coefficient and intercept values
coef = lm.coef_[0][0]
intercept = lm.intercept_[0]
print('Coefficient:', coef)
print('Intercept:', intercept)


# In[16]:
```

```
# Re-estimating trend
reestimated_trend = pd.Series(data=(intercept +
monthly_df['Time']*coef).values, index=seasonally_adjusted.index)

plt.plot(seasonally_adjusted, label = "Seasonally adjusted series")
plt.plot(reestimated_trend, label = "Trend")
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.title('Re-estimated trend component of ASX 200 Monthly time
series (training subset)')
plt.legend()


# In[17]:


# Estimating cyclic component
trend_seasonal = reestimated_trend + seasonal_components
cyclic_component = training_subset - trend_seasonal

# Removing noise by smoothing data (MA-3)
cyclic = cyclic_component.rolling(3, center=True).mean()

plt.plot(cyclic_component, label = "Cyclic component")
plt.plot(cyclic, label = "Smoothed")
plt.xlabel('Time')
plt.title('Cyclic component of ASX 200 Monthly time series (training
subset)')
plt.legend()


# In[18]:


# Calculating remainder
remainder = training_subset - (cyclic + reestimated_trend +
seasonal_components)

plt.plot(remainder, label = "error/residual")
plt.xlabel('Time')
plt.title('Remainder (residual) of ASX 200 Monthly time series
(training subset)')
plt.legend()


# In[19]:


# Forecasting 'Close' price for next 12 months (validation subset)

training_ts = reestimated_trend + seasonal_components

t = np.empty(validation_subset.size)
for i in range(0,validation_subset.size):
    t[i] = intercept + (training_subset.size+1+i)*coef
```

```
seasonal = monthly_ts - trend_cycle
seasonal_components = np.empty(monthly_ts.size)
for i in range(0,monthly_ts.size):
    seasonal_components[i] =
norm_seasonal_indexes[seasonal.index.month[i]-1]

predictions = validation_subset.copy()
for i in range(0,predictions.size):
    predictions[i] = t[i] +
seasonal_components[training_subset.size+i]


# In[20]:


# Appending fitted values with the predicted values
predicted_ts = training_ts.append(predictions)


# In[21]:


# Returning data to its original form by taking inverse of log
predicted_ts = np.exp(predicted_ts)
monthly_ts = np.exp(monthly_ts)
training_subset = np.exp(training_subset)
training_ts = np.exp(training_ts)
validation_subset = np.exp(validation_subset)
predictions = np.exp(predictions)


# In[22]:


plt.plot(monthly_ts, label = "Original time series")
plt.plot(predicted_ts, label = "Forecast")
plt.axvspan(validation_subset.index[0],
validation_subset.index[validation_subset.size-1], color='grey',
alpha=0.2, label="Validation subset")
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.title('Forecasting ASX 200 Monthly time series using classical
decomposition')
plt.legend()


# In[23]:


# Measuring errors of training subset
RMSE = np.sqrt(mean_squared_error(training_subset, training_ts))
MAE = mean_absolute_error(training_subset, training_ts)
MAPE = np.mean(np.abs((training_subset - training_ts) /
training_subset)) * 100
```

```
print('Training subset ->')
print('Root Mean Squared Error (RMSE):', RMSE)
print('Mean Absolute Error (MAE):', MAE)
print('Mean Absolute Percentage Error (MAPE):', MAPE)


# In[24]:


# Measuring errors of validation subset
RMSE = np.sqrt(mean_squared_error(validation_subset, predictions))
MAE = mean_absolute_error(validation_subset, predictions)
MAPE = np.mean(np.abs((validation_subset - predictions) /
validation_subset)) * 100

print('Validation subset ->')
print('Root Mean Squared Error (RMSE):', RMSE)
print('Mean Absolute Error (MAE):', MAE)
print('Mean Absolute Percentage Error (MAPE):', MAPE)




# Task 3 (Daily Exponential Smoothing)

# In[1]:


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
from matplotlib.pylab import rcParams
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from statsmodels.tsa.holtwinters import ExponentialSmoothing, Holt


# In[2]:


# Reading data
daily_data = pd.read_csv('ASX200Daily.csv')


# In[3]:


# Converting to a time series
daily_data['Date'] = pd.to_datetime(daily_data['Date'])
daily_data.set_index('Date', inplace = True)
daily_ts = pd.Series(daily_data['Close'])
print(daily_ts.head())
```

```
# In[4]:


# Removing missing values
daily_ts.dropna(inplace=True)


# In[5]:


# Days of the week with closing price given
daily_ts.index.dayofweek.unique()


# In[6]:


# Removing rows with day of the week = 6 (Sunday)
# Only taking weekdays into account will give better predictions for
this assignment
daily_ts = daily_ts[daily_ts.index.dayofweek!=6]


# In[7]:


# Splitting data into training and validation subsets
training_subset = daily_ts[:-365]
validation_subset = daily_ts[-365:]


# ### Holt's Linear Method

# In[8]:


holts_linear = Holt(training_subset).fit()
training_ts = holts_linear.fittedvalues
holts_linear.params


# In[9]:


# Forecasting values of next 365 days
predictions = holts_linear.forecast(365)
predictions.index = validation_subset.index


# In[10]:


predicted_ts = training_ts.append(predictions)

# Setting figure width and height
rcParams['figure.figsize'] = 15, 7
```

```
plt.plot(daily_ts, label = "Original time series")
plt.plot(predicted_ts, label = "Forecast")
plt.axvspan(validation_subset.index[0],
validation_subset.index[validation_subset.size-1], color='grey',
alpha=0.2, label="Validation subset")
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.title("Forecasting ASX 200 daily time series using holt's linear
method")
plt.legend()


# In[11]:


# Measuring errors of training subset
RMSE = np.sqrt(mean_squared_error(training_subset, training_ts))
MAE = mean_absolute_error(training_subset, training_ts)
MAPE = np.mean(np.abs((training_subset - training_ts) /
training_subset)) * 100

print('Training subset ->')
print('Root Mean Squared Error (RMSE):', RMSE)
print('Mean Absolute Error (MAE):', MAE)
print('Mean Absolute Percentage Error (MAPE):', MAPE)


# In[12]:


# Measuring errors of validation subset
RMSE = np.sqrt(mean_squared_error(validation_subset, predictions))
MAE = mean_absolute_error(validation_subset, predictions)
MAPE = np.mean(np.abs((validation_subset - predictions) /
validation_subset)) * 100

print('Validation subset ->')
print('Root Mean Squared Error (RMSE):', RMSE)
print('Mean Absolute Error (MAE):', MAE)
print('Mean Absolute Percentage Error (MAPE):', MAPE)


# ### Holt-Winters Method

# In[13]:


# For additive seasonality
holt_winters_add = ExponentialSmoothing(training_subset,
seasonal_periods = 5, trend = 'add', seasonal = 'add').fit()
training_ts_add = holt_winters_add.fittedvalues

# For multiplicative seasonality
holt_winters_mul = ExponentialSmoothing(training_subset,
seasonal_periods = 5, trend = 'add', seasonal = 'mul').fit()
```

```
training_ts_mul = holt_winters_mul.fittedvalues


# In[14]:


# Holt-winters additive seasonality parameters
holt_winters_add.params


# In[15]:


## Holt-winters multiplicative seasonality parameters
holt_winters_mul.params


# In[16]:


# Forecasting values of next 365 days

predictions_add = holt_winters_add.forecast(365)
predictions_add.index = validation_subset.index

predictions_mul = holt_winters_mul.forecast(365)
predictions_mul.index = validation_subset.index


# In[17]:


predicted_ts_add = training_ts.append(predictions_add)
predicted_ts_mul = training_ts.append(predictions_mul)

plt.plot(daily_ts, label = "Original time series")
plt.plot(predicted_ts_add, label = "Forecast (add. seasonality)")
plt.plot(predicted_ts_mul, label = "Forecast (mul. seasonality)")
plt.axvspan(validation_subset.index[0],
validation_subset.index[validation_subset.size-1], color='grey',
alpha=0.2, label="Validation subset")
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.title("Forecasting ASX 200 daily time series using holt-winters
method")
plt.legend()


# In[18]:


# Measuring errors of training subset (additive seasonality)
RMSE = np.sqrt(mean_squared_error(training_subset, training_ts_add))
MAE = mean_absolute_error(training_subset, training_ts_add)
MAPE = np.mean(np.abs((training_subset - training_ts_add) /
training_subset)) * 100
```

```
print('Training subset (additive seasonality) ->')
print('Root Mean Squared Error (RMSE):', RMSE)
print('Mean Absolute Error (MAE):', MAE)
print('Mean Absolute Percentage Error (MAPE):', MAPE)


# In[19]:


# Measuring errors of validation subset (additive seasonality)
RMSE = np.sqrt(mean_squared_error(validation_subset,
predictions_add))
MAE = mean_absolute_error(validation_subset, predictions_add)
MAPE = np.mean(np.abs((validation_subset - predictions_add) /
validation_subset)) * 100

print('Validation subset (additive seasonality) ->')
print('Root Mean Squared Error (RMSE):', RMSE)
print('Mean Absolute Error (MAE):', MAE)
print('Mean Absolute Percentage Error (MAPE):', MAPE)


# In[20]:


# Measuring errors of training subset (multiplicative seasonality)
RMSE = np.sqrt(mean_squared_error(training_subset, training_ts_mul))
MAE = mean_absolute_error(training_subset, training_ts_mul)
MAPE = np.mean(np.abs((training_subset - training_ts_mul) /
training_subset)) * 100

print('Training subset (multiplicative seasonality) ->')
print('Root Mean Squared Error (RMSE):', RMSE)
print('Mean Absolute Error (MAE):', MAE)
print('Mean Absolute Percentage Error (MAPE):', MAPE)


# In[21]:


# Measuring errors of validation subset (multiplicative seasonality)
RMSE = np.sqrt(mean_squared_error(validation_subset,
predictions_mul))
MAE = mean_absolute_error(validation_subset, predictions_mul)
MAPE = np.mean(np.abs((validation_subset - predictions_mul) /
validation_subset)) * 100

print('Validation subset (multiplicative seasonality) ->')
print('Root Mean Squared Error (RMSE):', RMSE)
print('Mean Absolute Error (MAE):', MAE)
print('Mean Absolute Percentage Error (MAPE):', MAPE)
```

```
# Task 3 (Monthly Exponential Smoothing)

# In[1]:


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
from matplotlib.pylab import rcParams
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from statsmodels.tsa.holtwinters import ExponentialSmoothing, Holt


# In[2]:


# Reading data
monthly_data = pd.read_csv('ASX200Monthly.csv')


# In[3]:


# Converting to a time series
monthly_data['Date'] = pd.to_datetime(monthly_data['Date'])
monthly_data.set_index('Date', inplace = True)
monthly_ts = pd.Series(monthly_data['Close'])
print(monthly_ts.head())


# In[4]:


# Removing missing values
monthly_ts.dropna(inplace=True)


# In[5]:


# Splitting data into training and validation subsets
training_subset = monthly_ts[:-12]
validation_subset = monthly_ts[-12:]


# ### Holt's Linear Method

# In[6]:


holts_linear = Holt(training_subset).fit()
training_ts = holts_linear.fittedvalues
holts_linear.params
```

```
# In[7]:


# Forecasting values of next 12 months
predictions = holts_linear.forecast(12)


# In[8]:


predicted_ts = training_ts.append(predictions)

# Setting figure width and height
rcParams['figure.figsize'] = 15, 7

plt.plot(monthly_ts, label = "Original time series")
plt.plot(predicted_ts, label = "Forecast")
plt.axvspan(validation_subset.index[0],
validation_subset.index[validation_subset.size-1], color='grey',
alpha=0.2, label="Validation subset")
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.title("Forecasting ASX 200 Monthly time series using holt's
linear method")
plt.legend()


# In[9]:


# Measuring errors of training subset
RMSE = np.sqrt(mean_squared_error(training_subset, training_ts))
MAE = mean_absolute_error(training_subset, training_ts)
MAPE = np.mean(np.abs((training_subset - training_ts) /
training_subset)) * 100

print('Training subset ->')
print('Root Mean Squared Error (RMSE):', RMSE)
print('Mean Absolute Error (MAE):', MAE)
print('Mean Absolute Percentage Error (MAPE):', MAPE)


# In[10]:


# Measuring errors of validation subset
RMSE = np.sqrt(mean_squared_error(validation_subset, predictions))
MAE = mean_absolute_error(validation_subset, predictions)
MAPE = np.mean(np.abs((validation_subset - predictions) /
validation_subset)) * 100

print('Validation subset ->')
print('Root Mean Squared Error (RMSE):', RMSE)
print('Mean Absolute Error (MAE):', MAE)
print('Mean Absolute Percentage Error (MAPE):', MAPE)


# ### Holt-Winters Method

# In[11]:


# For additive seasonality
holt_winters_add = ExponentialSmoothing(training_subset,
seasonal_periods = 12, trend = 'add', seasonal = 'add').fit()
training_ts_add = holt_winters_add.fittedvalues

# For multiplicative seasonality
holt_winters_mul = ExponentialSmoothing(training_subset,
seasonal_periods = 12, trend = 'add', seasonal = 'mul').fit()
training_ts_mul = holt_winters_mul.fittedvalues


# In[12]:


# Holt-winters additive seasonality parameters
holt_winters_add.params


# In[13]:


## Holt-winters multiplicative seasonality parameters
holt_winters_mul.params


# In[14]:


# Forecasting values of next 12 months
predictions_add = holt_winters_add.forecast(12)
predictions_mul = holt_winters_mul.forecast(12)


# In[15]:


predicted_ts_add = training_ts.append(predictions_add)
predicted_ts_mul = training_ts.append(predictions_mul)

plt.plot(monthly_ts, label = "Original time series")
plt.plot(predicted_ts_add, label = "Forecast (add. seasonality)")
plt.plot(predicted_ts_mul, label = "Forecast (mul. seasonality)")
plt.axvspan(validation_subset.index[0],
validation_subset.index[validation_subset.size-1], color='grey',
alpha=0.2, label="Validation subset")
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.title("Forecasting ASX 200 Monthly time series using holt-
winters method")
plt.legend()


# In[16]:


# Measuring errors of training subset (additive seasonality)
RMSE = np.sqrt(mean_squared_error(training_subset, training_ts_add))
MAE = mean_absolute_error(training_subset, training_ts_add)
MAPE = np.mean(np.abs((training_subset - training_ts_add) /
training_subset)) * 100

print('Training subset (additive seasonality) ->')
print('Root Mean Squared Error (RMSE):', RMSE)
print('Mean Absolute Error (MAE):', MAE)
print('Mean Absolute Percentage Error (MAPE):', MAPE)


# In[17]:


# Measuring errors of validation subset (additive seasonality)
RMSE = np.sqrt(mean_squared_error(validation_subset,
predictions_add))
MAE = mean_absolute_error(validation_subset, predictions_add)
MAPE = np.mean(np.abs((validation_subset - predictions_add) /
validation_subset)) * 100

print('Validation subset (additive seasonality) ->')
print('Root Mean Squared Error (RMSE):', RMSE)
print('Mean Absolute Error (MAE):', MAE)
print('Mean Absolute Percentage Error (MAPE):', MAPE)


# In[18]:


# Measuring errors of training subset (multiplicative seasonality)
RMSE = np.sqrt(mean_squared_error(training_subset, training_ts_mul))
MAE = mean_absolute_error(training_subset, training_ts_mul)
MAPE = np.mean(np.abs((training_subset - training_ts_mul) /
training_subset)) * 100

print('Training subset (multiplicative seasonality) ->')
print('Root Mean Squared Error (RMSE):', RMSE)
print('Mean Absolute Error (MAE):', MAE)
print('Mean Absolute Percentage Error (MAPE):', MAPE)


# In[19]:


# Measuring errors of validation subset (multiplicative seasonality)
```

```
RMSE = np.sqrt(mean_squared_error(validation_subset,
predictions_mul))
MAE = mean_absolute_error(validation_subset, predictions_mul)
MAPE = np.mean(np.abs((validation_subset - predictions_mul) /
validation_subset)) * 100

print('Validation subset (multiplicative seasonality) ->')
print('Root Mean Squared Error (RMSE):', RMSE)
print('Mean Absolute Error (MAE):', MAE)
print('Mean Absolute Percentage Error (MAPE):', MAPE)




# Task 4 (Daily ARIMA)

# In[1]:


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
from matplotlib.pylab import rcParams
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
import statsmodels as sm
import statsmodels.api as smt
from statsmodels.tsa.arima_model import ARIMA
from pmdarima.arima import auto_arima


# In[2]:


# Reading data
daily_data = pd.read_csv('ASX200Daily.csv')


# In[3]:


# Converting to a time series
daily_data['Date'] = pd.to_datetime(daily_data['Date'])
daily_data.set_index('Date', inplace = True)
daily_ts = pd.Series(daily_data['Close'])
print(daily_ts.head())


# In[4]:


# Removing missing values
daily_ts.dropna(inplace=True)
```

```
# In[5]:


# Splitting data into training and validation subsets
training_subset = daily_ts[:-365]
validation_subset = daily_ts[-365:]


# In[6]:


# Setting figure width and height
rcParams['figure.figsize'] = 15, 7

# Plotting time series
plt.plot(training_subset, label = "daily time series")
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.title('ASX 200 daily time series')
plt.legend()


# In[7]:


# ACF plot of the original data
smt.graphics.tsa.plot_acf(training_subset, lags = 30, alpha = 0.05)
plt.show()

# PACF plot of the original data
smt.graphics.tsa.plot_pacf(training_subset, lags = 30, alpha = 0.05)
plt.show()


# As we can see from the plots above, ACF does not die down quickly
which means that the time series is non-stationary. We will apply
transformations to the data to make it stationary.

# In[8]:


# Transforming time series by taking log
# It reduces variation in data by penalizing larger values more than
the smaller ones
training_subset_log = np.log(training_subset)


# In[9]:


# Plotting log transformed time series
plt.plot(training_subset_log, label = "daily time series")
plt.xlabel('Time')
plt.title('ASX 200 daily time series (log transformed)')
plt.legend()
```

```
# In[10]:


# Applying first order differencing to make the data stationary
daily_ts_diff = training_subset_log.diff()
daily_ts_diff.dropna(inplace = True)

# Plotting time series
plt.plot(daily_ts_diff, label = "daily time series")
plt.xlabel('Time')
plt.title('ASX 200 daily time series (1st order differencing)')
plt.legend()


# In[11]:


# ACF plot of the first order differenced data
smt.graphics.tsa.plot_acf(daily_ts_diff, lags = 30, alpha = 0.05)
plt.show()

# PACF plot of the first order differenced data
smt.graphics.tsa.plot_pacf(daily_ts_diff, lags = 30, alpha = 0.05)
plt.show()


# ### Manually fitting ARIMA model

# After applying log transformation and first order differencing, we
can see that the ACF cuts off fairly quickly. This means that the
data can now be considered stationary.

# As per the lecture slides: When both ACF and PACF die down
quickly, check the orders at which ACF or PACF die down. The order
of ACF will be the lag q of the ARIMA and the order of PACF will be
the lag p of the ARIMA, and the order of difference will be d.

# In this case: ACF order = 0, PACF order = 0, d = 1

# In[12]:


# ARIMA(0,1,0) model
model_ARIMA = ARIMA(training_subset_log, order = (0, 1, 0))

# Fitting the data
fit_ARIMA = model_ARIMA.fit(disp = -1)


# In[13]:


# Checking the residuals
residuals_ARIMA = pd.DataFrame(fit_ARIMA.resid)
```

```
plt.plot(residuals_ARIMA, label = "Residuals")
plt.xlabel('Time')
plt.title('Residuals of ARIMA(0,1,0) model for ASX 200 daily time
series')
plt.legend()


# In[14]:


# Forecasting 'Close' price for next 365 days (validation subset)
results_ARIMA = fit_ARIMA.predict(end=training_subset_log.size+364,
typ='levels', dynamic=False)


# In[15]:


# Returning data to its original form by taking inverse of log
predictions_ARIMA = np.exp(results_ARIMA)


# In[16]:


# Converting results into a time series
predictions_ARIMA.index = daily_ts.index[1:]


# In[17]:


# Plot of Actual vs Forecasted data
plt.plot(daily_ts[1:], label="Original")
plt.plot(predictions_ARIMA, label = "Forecast")
plt.axvspan(validation_subset.index[0],
validation_subset.index[validation_subset.size-1], color='grey',
alpha=0.2, label="Validation subset")
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.title('Forecasting ASX 200 daily time series using
ARIMA(0,1,0)')
plt.legend()


# In[18]:


# Measuring errors of training subset
RMSE = np.sqrt(mean_squared_error(training_subset[1:],
predictions_ARIMA[:-365]))
MAE = mean_absolute_error(training_subset[1:], predictions_ARIMA[:-
365])
MAPE = np.mean(np.abs((training_subset[1:] - predictions_ARIMA[:-
365]) / training_subset[1:])) * 100
```

```
print('Training subset ->')
print('Root Mean Squared Error (RMSE):', RMSE)
print('Mean Absolute Error (MAE):', MAE)
print('Mean Absolute Percentage Error (MAPE):', MAPE)


# In[19]:


# Measuring errors of validation subset
RMSE = np.sqrt(mean_squared_error(validation_subset, 
predictions_ARIMA[-365:]))
MAE = mean_absolute_error(validation_subset, predictions_ARIMA[-
365:])
MAPE = np.mean(np.abs((validation_subset - predictions_ARIMA[-365:]) 
/ validation_subset)) * 100

print('Validation subset ->')
print('Root Mean Squared Error (RMSE):', RMSE)
print('Mean Absolute Error (MAE):', MAE)
print('Mean Absolute Percentage Error (MAPE):', MAPE)


# ### Auto-ARIMA model

# In[20]:


# Using auto-ARIMA to tune hyperparameters
autoARIMA_model = auto_arima(training_subset_log, start_p=0, 
start_q=0, m=5,
                          start_P=0, start_Q=0, seasonal=True, d=1, 
trace=True,
                          error_action='ignore',
                          suppress_warnings=True,
                          stepwise=True)

autoARIMA_model.summary()


# The best model chosen by auto-ARIMA is ARIMA(1,1,0)

# In[21]:


# ARIMA(1,1,0) model
train_model = autoARIMA_model.fit(training_subset_log)


# In[22]:


# Calculating fitted values
fitted_values = training_subset_log + train_model.resid()
```

```
# In[23]:


# Forecasting 'Close' price for next 365 days (validation subset)
results_autoARIMA = train_model.predict(n_periods=365)


# In[24]:


# Converting results into a time series
results_autoARIMA = pd.Series(results_autoARIMA,
index=daily_ts.index[-365:])


# In[25]:


# Appending fitted values with the predicted values
predictions_autoARIMA = fitted_values.append(results_autoARIMA)


# In[26]:


# Returning data to its original form by taking inverse of log
predictions_autoARIMA = np.exp(predictions_autoARIMA)


# In[27]:


predictions_autoARIMA = predictions_autoARIMA[1:]

# Plot of Actual vs Forecasted data
plt.plot(daily_ts[1:], label="Original")
plt.plot(predictions_autoARIMA, label = "Forecast")
plt.axvspan(validation_subset.index[0],
validation_subset.index[validation_subset.size-1], color='grey',
alpha=0.2, label="Validation subset")
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.title('Forecasting ASX 200 daily time series using
ARIMA(1,1,0)')
plt.legend()


# In[28]:


# Measuring errors of training subset
RMSE = np.sqrt(mean_squared_error(training_subset[1:],
predictions_autoARIMA[:-365]))
MAE = mean_absolute_error(training_subset[1:],
predictions_autoARIMA[:-365])
```

```
MAPE = np.mean(np.abs((training_subset[1:] -
predictions_autoARIMA[:-365]) / training_subset[1:])) * 100

print('Training subset ->')
print('Root Mean Squared Error (RMSE):', RMSE)
print('Mean Absolute Error (MAE):', MAE)
print('Mean Absolute Percentage Error (MAPE):', MAPE)


# In[29]:


# Measuring errors of validation subset
RMSE = np.sqrt(mean_squared_error(validation_subset,
predictions_autoARIMA[-365:]))
MAE = mean_absolute_error(validation_subset, predictions_autoARIMA[-
365:])
MAPE = np.mean(np.abs((validation_subset - predictions_autoARIMA[-
365:]) / validation_subset)) * 100

print('Validation subset ->')
print('Root Mean Squared Error (RMSE):', RMSE)
print('Mean Absolute Error (MAE):', MAE)
print('Mean Absolute Percentage Error (MAPE):', MAPE)




# Task 4 (Monthly ARIMA)

# In[1]:


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
from matplotlib.pylab import rcParams
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
import statsmodels as sm
import statsmodels.api as smt
from statsmodels.tsa.arima_model import ARIMA
from pmdarima.arima import auto_arima


# In[2]:


# Reading data
monthly_data = pd.read_csv('ASX200Monthly.csv')


# In[3]:
```

```
# Converting to a time series
monthly_data['Date'] = pd.to_datetime(monthly_data['Date'])
monthly_data.set_index('Date', inplace = True)
monthly_ts = pd.Series(monthly_data['Close'])
print(monthly_ts.head())


# In[4]:


# Removing missing values
monthly_ts.dropna(inplace=True)


# In[5]:


# Splitting data into training and validation subsets
training_subset = monthly_ts[:-12]
validation_subset = monthly_ts[-12:]


# In[6]:


# Setting figure width and height
rcParams['figure.figsize'] = 15, 7

# Plotting time series
plt.plot(training_subset, label = "Monthly time series")
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.title('ASX 200 Monthly time series')
plt.legend()


# In[7]:


# ACF plot of the original data
smt.graphics.tsa.plot_acf(training_subset, lags = 30, alpha = 0.05)
plt.show()

# PACF plot of the original data
smt.graphics.tsa.plot_pacf(training_subset, lags = 30, alpha = 0.05)
plt.show()


# As we can see from the plots above, ACF does not die down quickly
which means that the time series is non-stationary. We will apply
transformations to the data to make it stationary.

# In[8]:
```

```
# Transforming time series by taking log
# It reduces variation in data by penalizing larger values more than
the smaller ones
training_subset_log = np.log(training_subset)


# In[9]:


# Plotting log transformed time series
plt.plot(training_subset_log, label = "Monthly time series")
plt.xlabel('Time')
plt.title('ASX 200 Monthly time series (log transformed)')
plt.legend()


# In[10]:


# Applying first order differencing to make the data stationary
monthly_ts_diff = training_subset_log.diff()
monthly_ts_diff.dropna(inplace = True)

# Plotting time series
plt.plot(monthly_ts_diff, label = "Monthly time series")
plt.xlabel('Time')
plt.title('ASX 200 Monthly time series (1st order differencing)')
plt.legend()


# In[11]:


# ACF plot of the first order differenced data
smt.graphics.tsa.plot_acf(monthly_ts_diff, lags = 30, alpha = 0.05)
plt.show()

# PACF plot of the first order differenced data
smt.graphics.tsa.plot_pacf(monthly_ts_diff, lags = 30, alpha = 0.05)
plt.show()


# ### Manually fitting ARIMA model

# After applying log transformation and first order differencing, we
can see that the ACF cuts off fairly quickly. This means that the
data can now be considered stationary.

# As per the lecture slides: When both ACF and PACF die down
quickly, check the orders at which ACF or PACF die down. The order
of ACF will be the lag q of the ARIMA and the order of PACF will be
the lag p of the ARIMA, and the order of difference will be d.

# In this case: ACF order = 0, PACF order = 0, d = 1

# In[12]:
```

```
# ARIMA(0,1,0) model
model_ARIMA = ARIMA(training_subset_log, order = (0, 1, 0))

# Fitting the data
fit_ARIMA = model_ARIMA.fit(disp = -1)


# In[13]:


# Checking the residuals
residuals_ARIMA = pd.DataFrame(fit_ARIMA.resid)

plt.plot(residuals_ARIMA, label = "Residuals")
plt.xlabel('Time')
plt.title('Residuals of ARIMA(0,1,0) model for ASX 200 Monthly time 
series')
plt.legend()


# In[14]:


# Forecasting 'Close' price for next 12 months (validation subset)
results_ARIMA = fit_ARIMA.predict(end=training_subset_log.size+11, 
typ='levels', dynamic=False)


# In[15]:


# Returning data to its original form by taking inverse of log
predictions_ARIMA = np.exp(results_ARIMA)


# In[16]:


# Plot of Actual vs Forecasted data
plt.plot(monthly_ts[1:], label="Original")
plt.plot(predictions_ARIMA, label = "Forecast")
plt.axvspan(validation_subset.index[0], 
validation_subset.index[validation_subset.size-1], color='grey', 
alpha=0.2, label="Validation subset")
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.title('Forecasting ASX 200 Monthly time series using 
ARIMA(0,1,0)')
plt.legend()


# In[17]:
```

```
# Measuring errors of training subset
RMSE = np.sqrt(mean_squared_error(training_subset[1:],
predictions_ARIMA[:-12]))
MAE = mean_absolute_error(training_subset[1:], predictions_ARIMA[:-
12])
MAPE = np.mean(np.abs((training_subset[1:] - predictions_ARIMA[:-
12]) / training_subset[1:])) * 100

print('Training subset ->')
print('Root Mean Squared Error (RMSE):', RMSE)
print('Mean Absolute Error (MAE):', MAE)
print('Mean Absolute Percentage Error (MAPE):', MAPE)


# In[18]:


# Measuring errors of validation subset
RMSE = np.sqrt(mean_squared_error(validation_subset,
predictions_ARIMA[-12:]))
MAE = mean_absolute_error(validation_subset, predictions_ARIMA[-
12:])
MAPE = np.mean(np.abs((validation_subset - predictions_ARIMA[-12:])
/ validation_subset)) * 100

print('Validation subset ->')
print('Root Mean Squared Error (RMSE):', RMSE)
print('Mean Absolute Error (MAE):', MAE)
print('Mean Absolute Percentage Error (MAPE):', MAPE)


# ### Auto-ARIMA model

# In[19]:


# Using auto-ARIMA to tune hyperparameters
autoARIMA_model = auto_arima(training_subset_log, start_p=0,
start_q=0, m=12,
                          start_P=0, start_Q=0, seasonal=True, d=1,
trace=True,
                          error_action='ignore',
                          suppress_warnings=True,
                          stepwise=True)

autoARIMA_model.summary()


# The best model chosen by auto-ARIMA is ARIMA(0,1,1)(0,0,1)[12]

# In[20]:


# ARIMA(0,1,1)(0,0,1)[12] model
train_model = autoARIMA_model.fit(training_subset_log)
```

```
# In[21]:


# Calculating fitted values
fitted_values = training_subset_log + train_model.resid()


# In[22]:


# Forecasting 'Close' price for next 12 months (validation subset)
results_autoARIMA = train_model.predict(n_periods=12)


# In[23]:


# Converting results into a time series
results_autoARIMA = pd.Series(results_autoARIMA,
index=monthly_ts.index[-12:])


# In[24]:


# Appending fitted values with the predicted values
predictions_autoARIMA = fitted_values.append(results_autoARIMA)


# In[25]:


# Returning data to its original form by taking inverse of log
predictions_autoARIMA = np.exp(predictions_autoARIMA)


# In[26]:


predictions_autoARIMA = predictions_autoARIMA[1:]

# Plot of Actual vs Forecasted data
plt.plot(monthly_ts[1:], label="Original")
plt.plot(predictions_autoARIMA, label = "Forecast")
plt.axvspan(validation_subset.index[0],
validation_subset.index[validation_subset.size-1], color='grey',
alpha=0.2, label="Validation subset")
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.title('Forecasting ASX 200 Monthly time series using
ARIMA(0,1,1)(0,0,1)[12]')
plt.legend()


# In[27]:
```

```
# Measuring errors of training subset
RMSE = np.sqrt(mean_squared_error(training_subset[1:],
predictions_autoARIMA[:-12]))
MAE = mean_absolute_error(training_subset[1:],
predictions_autoARIMA[:-12])
MAPE = np.mean(np.abs((training_subset[1:] -
predictions_autoARIMA[:-12]) / training_subset[1:])) * 100

print('Training subset ->')
print('Root Mean Squared Error (RMSE):', RMSE)
print('Mean Absolute Error (MAE):', MAE)
print('Mean Absolute Percentage Error (MAPE):', MAPE)


# In[28]:


# Measuring errors of validation subset
RMSE = np.sqrt(mean_squared_error(validation_subset,
predictions_autoARIMA[-12:]))
MAE = mean_absolute_error(validation_subset, predictions_autoARIMA[-
12:])
MAPE = np.mean(np.abs((validation_subset - predictions_autoARIMA[-
12:]) / validation_subset)) * 100

print('Validation subset ->')
print('Root Mean Squared Error (RMSE):', RMSE)
print('Mean Absolute Error (MAE):', MAE)
print('Mean Absolute Percentage Error (MAPE):', MAPE)



# Task 4 (Daily LSTM)

# In[ ]:



import time
import math
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error

from keras.layers.core import Dense, Activation, Dropout
from keras.layers.recurrent import LSTM
from keras.models import Sequential

# We want to be able to repeat the experiment so we fix to a random
seed
np.random.seed(70)
```

```
# Let us loading data
data=pd.read_csv("ASX200Daily.csv",usecols=['Close'])##usecols=['Dat
e', 'Close'])
data = data.dropna()  # Drop all Nans
data = data.values  # Convert from DataFrame to Python Array
                    # You need to make sure the data is type of
float
                    # you may use data = data.astype('float32') if
your data are integers

# Prepare data .....


# In[ ]:




""" Scaling ...
Neural networks normally work well with scaled data, especially when
we use
the sigmoid or tanh activation function. It is a good practice to
scale the
data to the range of 0-to-1. This can be easily done by using
scikit-learn's
MinMaxScaler
"""
scaler = MinMaxScaler(feature_range=(0, 1))
data = scaler.fit_transform(data)

"""  Splitting ...
We are going to use a time window of k = 3, so we will split the
time series as
   x_1, x_2, x_3,     x_4  [prediction]
   x_2, x_3, x_4,     x_5
   x_3, x_4, x_5,     x_6
   ....
"""
train_size = int(len(data)*0.7)
test_size = len(data) - train_size
Dtrain, Dtest = data[0:train_size,:], data[train_size:len(data),:]

time_window =15
Xtrain, Ytrain = [], []
for i in range(len(Dtrain) - time_window -1):
    Xtrain.append(Dtrain[i:(i+time_window), 0])
    Ytrain.append(Dtrain[i+time_window, 0])
Xtrain = np.array(Xtrain)
Ytrain = np.array(Ytrain)


Xtest, Ytest = [], []
for i in range(len(Dtest) - time_window -1):
    Xtest.append(Dtest[i:(i+time_window), 0])
    Ytest.append(Dtest[i+time_window, 0])
Xtest = np.array(Xtest)
```

```
Ytest = np.array(Ytest)


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


# In[ ]:


# Define our model .....
model = Sequential()


MyBatchSize = 1

model.add(LSTM(100, input_shape = (time_window,1),
batch_size=MyBatchSize,
        return_sequences=False))   # Many-to-One model

# It seems without using batch_size = 1 here causes some problems, a
bug?

model.add(Dropout(0.2))            # Impose sparsity as we have so
many hidden neurons
# As we will have 100 outputs from LSTM at each time step, we will
use a linear
# layer to map them to a single "prediction" output
model.add(Dense(
        output_dim=1))
model.add(Activation("linear"))

# Compiling model for use
start = time.time()
model.compile(loss="mse", optimizer="Adadelta")
print("Compilation Time : ", time.time() - start)


# Training
model.fit(
         Xtrain,
         Ytrain,
         batch_size=MyBatchSize,
         nb_epoch=50,   # You increase this number
         validation_split=0.05)

# Predicting
# make predictions
trainPredict = model.predict(Xtrain,batch_size=MyBatchSize)
testPredict = model.predict(Xtest,batch_size=MyBatchSize)
# invert predictions due to scaling
trainPredict = scaler.inverse_transform(trainPredict)
Ytrain = scaler.inverse_transform(Ytrain[:,np.newaxis])
testPredict = scaler.inverse_transform(testPredict)
Ytest = scaler.inverse_transform(Ytest[:,np.newaxis])
# calculate root mean squared error
```

```
trainScore = math.sqrt(mean_squared_error(Ytrain,
trainPredict[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(Ytest, testPredict[:,0]))
print('Test Score: %.2f RMSE' % (testScore))


# In[ ]:


# Plotting results
# shift train predictions for plotting
trainPredictPlot = np.empty_like(data)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[time_window:len(trainPredict)+time_window, :] =
trainPredict
# shift test predictions for plotting
testPredictPlot = np.empty_like(data)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(trainPredict)+(time_window*2)+1:len(data)-1, :]
= testPredict
# plot baseline and predictions
plt.plot(scaler.inverse_transform(data), label='True Data')
plt.plot(trainPredictPlot, label='Train Prediction')
plt.plot(testPredictPlot, label='Test Prediction')
plt.legend()
plt.show()


# In[ ]:


## Overall Predictions
Xall = []
predictions = []

for i in range(6):
    Xall = []
    for j in range(len(data) - time_window):
        Xall.append(data[j:(j+time_window), 0])   # pick up the
section in time_window size
    Xall = np.array(Xall)    # Convert them from list to array
    Xall = np.reshape(Xall, (Xall.shape[0], time_window, 1))

    allPredict = model.predict(Xall,batch_size=MyBatchSize)
    data=np.append(data,allPredict[-1])
    data=data[:,np.newaxis]
    allPredict = scaler.inverse_transform(allPredict)
    predictions.append(allPredict[-1])


data=data[:len(data)-1]

allPredictPlot = np.empty_like(data)
allPredictPlot[:, :] = np.nan
allPredictPlot[time_window:, :] = allPredict
```

```
plt.figure()
plt.plot(scaler.inverse_transform(data), label='True Data')
plt.plot(allPredictPlot, label='One-Step Prediction')
plt.legend()
plt.show()




# Task 4 (Monthly LSTM)

# In[ ]:



import time
import math
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error

from keras.layers.core import Dense, Activation, Dropout
from keras.layers.recurrent import LSTM
from keras.models import Sequential

# We want to be able to repeat the experiment so we fix to a random
seed
np.random.seed(70)

# Let us loading data
data=pd.read_csv("ASX200Monthly.csv",usecols=['Close'])##usecols=['D
ate', 'Close'])
data = data.dropna()  # Drop all Nans

data = data.values  # Convert from DataFrame to Python Array
                    # You need to make sure the data is type of
float
                    # you may use data = data.astype('float32') if
your data are integers
# Prepare data .....


# In[ ]:





# In[ ]:



""" Scaling ...
```

```
Neural networks normally work well with scaled data, especially when
we use
the sigmoid or tanh activation function. It is a good practice to
scale the
data to the range of 0-to-1. This can be easily done by using
scikit-learn's
MinMaxScaler
"""
scaler = MinMaxScaler(feature_range=(0, 1))
data = scaler.fit_transform(data)

"""  Splitting ...
We are going to use a time window of k = 3, so we will split the
time series as
   x_1, x_2, x_3,     x_4  [prediction]
   x_2, x_3, x_4,     x_5
   x_3, x_4, x_5,     x_6
   ....
"""
train_size = int(len(data)*0.7)   # Use the first 2/3 data for
training
test_size = len(data) - train_size # the remaining for testing
Dtrain, Dtest = data[0:train_size,:], data[train_size:len(data),:]
# Both Xtrain and Xtest are in time series form, we need split them
into sections
# in time-window size 4
time_window = 4
Xtrain, Ytrain = [], []
for i in range(len(Dtrain) - time_window -1):
    Xtrain.append(Dtrain[i:(i+time_window), 0])   # pick up the
section in time_window size
    Ytrain.append(Dtrain[i+time_window, 0])       # pick up the next
one as the prediction
Xtrain = np.array(Xtrain)    # Convert them from list to array
Ytrain = np.array(Ytrain)


Xtest, Ytest = [], []
for i in range(len(Dtest) - time_window -1):
    Xtest.append(Dtest[i:(i+time_window), 0])   # pick up the
section in time_window size
    Ytest.append(Dtest[i+time_window, 0])       # pick up the next
one as the prediction
Xtest = np.array(Xtest)    # Convert them from list to array
Ytest = np.array(Ytest)

# We are going to use keras package, so we must reshape our data to
the keras required format
# (samples, time_window, features)  we are almost there, but need to
reshape into 3D array
# For time series, the feature number is 1 (one scalar value at each
time step)
Xtrain = np.reshape(Xtrain, (Xtrain.shape[0], time_window, 1))
Xtest = np.reshape(Xtest, (Xtest.shape[0], time_window, 1))
```

```
# In[ ]:



# Define our model .....
##model = Sequential()
MyBatchSize = 1

model = Sequential()
model.add(LSTM(units = 70, return_sequences = True, input_shape = 
(time_window,1)))
model.add(Activation('relu'))
model.add(Dropout(0.3))

model.add(LSTM(units = 70, return_sequences = True))
model.add(Activation('relu'))
model.add(Dropout(0.3))

model.add(LSTM(units = 70, return_sequences = True))
model.add(Dropout(0.3))
model.add(Activation('relu'))


model.add(LSTM(units = 70, return_sequences = False))
model.add(Dropout(0.3))
model.add(Activation('relu'))



##model.add(LSTM(units = 50))
##model.add(Dropout(0.2))
model.add(Dense(units = 1))
model.compile(optimizer = 'adam', loss = 
'mean_squared_error',metrics=['accuracy'])

###regressor.fit(X_train, y_train, epochs = 100, batch_size = 32)


# Training
model.fit(
          Xtrain,
          Ytrain,
          batch_size=MyBatchSize,
          nb_epoch=50,   # You increase this number
          validation_split=0.05)

# Predicting
# make predictions
trainPredict = model.predict(Xtrain,batch_size=MyBatchSize)
testPredict = model.predict(Xtest,batch_size=MyBatchSize)
# invert predictions due to scaling
trainPredict = scaler.inverse_transform(trainPredict)
Ytrain = scaler.inverse_transform(Ytrain[:,np.newaxis])
```

```
testPredict = scaler.inverse_transform(testPredict)
Ytest = scaler.inverse_transform(Ytest[:,np.newaxis])
# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(Ytrain,
trainPredict[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(Ytest, testPredict[:,0]))
print('Test Score: %.2f RMSE' % (testScore))


# In[ ]:


# Plotting results
# shift train predictions for plotting
trainPredictPlot = np.empty_like(data)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[time_window:len(trainPredict)+time_window, :] =
trainPredict
# shift test predictions for plotting
testPredictPlot = np.empty_like(data)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(trainPredict)+(time_window*2)+1:len(data)-1, :]
= testPredict
# plot baseline and predictions
plt.plot(scaler.inverse_transform(data), label='True Data')
plt.plot(trainPredictPlot, label='Train Prediction')
plt.plot(testPredictPlot, label='Test Prediction')
plt.legend()
plt.show()




# In[ ]:


## Overall Predictions
Xall = []
predictions = []

for i in range(6):
    Xall = []
    for j in range(len(data) - time_window):
        Xall.append(data[j:(j+time_window), 0])   # pick up the
section in time_window size
    Xall = np.array(Xall)    # Convert them from list to array
    Xall = np.reshape(Xall, (Xall.shape[0], time_window, 1))

    allPredict = model.predict(Xall,batch_size=MyBatchSize)
    data=np.append(data,allPredict[-1])
    data=data[:,np.newaxis]
    allPredict = scaler.inverse_transform(allPredict)
    predictions.append(allPredict[-1])
```

```
data=data[:len(data)-1]

allPredictPlot = np.empty_like(data)
allPredictPlot[:, :] = np.nan
allPredictPlot[time_window:, :] = allPredict

plt.figure()
plt.plot(scaler.inverse_transform(data), label='True Data')
plt.plot(allPredictPlot, label='One-Step Prediction')
plt.legend()
plt.show()


# In[ ]:


predictions#
```

Appendix B

MEETING MINUTES 1

Minutes of meeting for QBUS6840 Predictive Analytics Group Assignment

Date: 03/05/2019 Time: 3:00 pm Location: Fisher Library Duration: 30 mins

Present: 470437832, 490451474, 480519021, 490095267

Apologies:

Agenda Item	Key Points	Action	By Whom	When	Communication Strategy	Review
1. Task 1, 2 & 3 coding	• Data Preprocessing • Exploratory Data Analysis • Benchmark models including classical decomposition model and exponential moving average	Write code in Jupyter Notebook and upload it in the shared drive	470437832 & 490451474	Till 10/05/2019	WhatsApp Group & Shared Google Drive	All work completed on time
2. Report Writing	• Writing introduction of the report • Writing the related work of the report	Write in MS Word and upload it in the shared drive	490095267 & 480519021			

MEETING MINUTES 2

Minutes of meeting for QBUS6840 Predictive Analytics Group Assignment

Date: 10/05/2019 Time: 5:00 pm Location: ABS Duration: 60 mins

Present: 470437832, 490451474, 480519021, 490095267

Apologies:

Agenda Item	Key Points	Action	By Whom	When	Communication Strategy	Review
3. Task 3 & 4 coding	• Discussion on which models to build • Benchmark models + advanced models including ARIMA and Neural Network	Write code in Jupyter Notebook and upload it in the shared drive	470437832 & 490451474	Till 17/05/2019	WhatsApp Group & Shared Google Drive	All work completed on time
4. Report Writing	• Writing about the data sets and features • Writing about the benchmark models	Write in MS Word and upload it in the shared drive	490095267 & 480519021			

MEETING MINUTES 3

Minutes of meeting for QBUS6840 Predictive Analytics Group Assignment

Date: 17/05/2019 Time: 3:00 pm Location: ABS Duration: 90 mins

Present: 470437832, 490451474, 480519021, 490095267

Apologies:

Agenda Item	Key Points	Action	By Whom	When	Communication Strategy	Review
5. Task 4 coding 6. Report Writing	• Discussion on implementing different variations of ARIMA model • Discussing regarding LSTM model • Explaining code to group members for report writing purposes	Write code in Jupyter Notebook and upload it in the shared drive Write in MS Word and upload it in the shared drive	470437832 & 490451474 490095267 & 480519021	Till 24/05/2019	WhatsApp Group & Shared Google Drive	All work completed on time