# Machine Learning Engineer Nanodegree

Prepared by: Chun Wang Chan
12 December 2018

## Report

Investment and Trading Capstone Project
Build a Stock Price Indicator using Machine Learning techniques

# I. Definition

## Project Overview

A company completed an initial public offering then its shares will become public. As a result, the shares can be traded on a stock market. The stock market is a venue for buyers and sellers of shares to trade.

The price of a stock can represent the value of the company. The share price is affected by changes in the stock supply and demand. So if there are more buyers interested to buy at a given moment than seller selling it then the price moves up. On the other hand, if more shareholders intended to sell than people to buy then the price will fall. But this is the direct and simple reason for stock price movement. There are much more complex or indirect hidden factors affecting shareholders make decision resulting price change. Thus, the problem of predicting future share price is a challenge.

In this project, I will explore the problem of stock price prediction using machine learning methods. The machine learning methods include supervised learning method and deep learning method. The supervised learning method to examine is using XGBoost to tackle the problem. The deep learning method will be using a RNN LSTM to predict the stock price.

## Project Statement

Problem definition: To predict the adjusted close price of the day immediately after the end of the given training end date, of a given stock as accurate as possible using historical stock prices. The project will be undertaken in the order below.

Outline of tasks:

1. Import dataset from source.
2. Explore the data by printing out the data frame and visualise it by plotting into graphs.
3. Pre-processing the dataset
4. Split dataset into training and testing dataset.
5. Implement benchmark model and record benchmark result
6. Implement, train and test XGBoost and RNN LSTM.
7. Plot and summarize final results.

## Metrics

The formula of RMSE:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

The evaluation metric to be used to quantify the performance is the Root Mean Square Error or RMSE. This metric is very popular for the regression problem and continuous values. It can provide prediction accuracy of the model and it is said to be computationally simple. The characteristic of RMSE is that it penalises higher difference more than metric like Mean Absolute Error or MAE. This factor is important and useful for the problem. Very inaccurate prediction should be avoided and heavily penalised. So the model can minimise chances to lose huge money by following very inaccurate prediction.

# II. Analysis

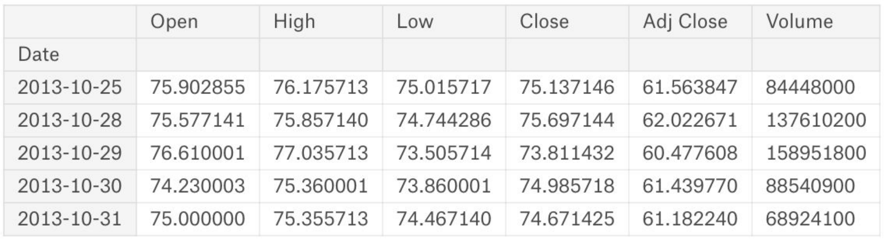
## Data Exploration

Dataset Origin: The dataset is obtained from the Yahoo Finance API. It contains historical stock prices which can be used for training a machine learning model or a deep neural network to predict future stock prices. For each stock the Yahoo Finance dataset will provide features or values including open, high, low, close, volume and adjusted close. The dataset contains data for stocks traded in most major stock exchanges internationally.

But I will only use selected US stock in this project. These stocks including Apple, VISA and Tesla. Each of these stock are traded in different industry sectors. Apple is in the computer hardware, software and consumer electronics sector. VISA is in the financial service sector and Tesla is in the automobile sector. I chose these three stocks from different sectors because it can introduce some variety in the problem and result. This is because different stocks from different sectors might act quite differently in price movement.

The input data is the data feature of adjusted closing price. The adjusted closing price is closing price after amended any distributions and corporate actions occurred before next trading day opens. The API can provide price data from the first day of trading the stock to current date. But for this project we only will be using most recent 5 years of price data. It should be reasonable amount for building model. The output is for each prediction it will predict the 1 day ahead of adjusted closing price for given stock.

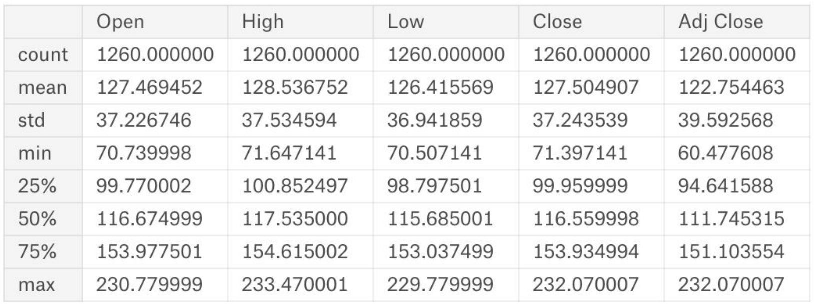
Apple Inc Dataset Sample:



| | Open | High | Low | Close | Adj Close | Volume |
|---|---|---|---|---|---|---|
| Date | | | | | | |
| 2013-10-25 | 75.902855 | 76.175713 | 75.015717 | 75.137146 | 61.563847 | 84448000 |
| 2013-10-28 | 75.577141 | 75.857140 | 74.744286 | 75.697144 | 62.022671 | 137610200 |
| 2013-10-29 | 76.610001 | 77.035713 | 73.505714 | 73.811432 | 60.477608 | 158951800 |
| 2013-10-30 | 74.230003 | 75.360001 | 73.860001 | 74.985718 | 61.439770 | 88540900 |
| 2013-10-31 | 75.000000 | 75.355713 | 74.467140 | 74.671425 | 61.182240 | 68924100 |

Comments: The data sample shows five days data for Apple Inc. and each row is a day. There are five feature columns, including open, high, low, close, adj close and volume. The first column on the left is actually the index of the dataset and it is the date of the stock data. For example, the row with date of '2013-10-25' will contain price and trade volume of the stock on that date. The rest of the columns are other data including open, high, low, close and adjusted close prices. Also the volume or amount of trade on the date of the index. These are existed data from the dataset.

Basic Statistic:



| | Open | High | Low | Close | Adj Close |
|---|---|---|---|---|---|
| count | 1260.000000 | 1260.000000 | 1260.000000 | 1260.000000 | 1260.000000 |
| mean | 127.469452 | 128.536752 | 126.415569 | 127.504907 | 122.754463 |
| std | 37.226746 | 37.534594 | 36.941859 | 37.243539 | 39.592568 |
| min | 70.739998 | 71.647141 | 70.507141 | 71.397141 | 60.477608 |
| 25% | 99.770002 | 100.852497 | 98.797501 | 99.959999 | 94.641588 |
| 50% | 116.674999 | 117.535000 | 115.685001 | 116.559998 | 111.745315 |
| 75% | 153.977501 | 154.615002 | 153.037499 | 153.934994 | 151.103554 |
| max | 230.779999 | 233.470001 | 229.779999 | 232.070007 | 232.070007 |

Comments: Here is the print out statistic for the Apple Inc. dataset. The count for all features are equal. The adjusted closing price mean is lower than the mean of open and closing prices. As we can see from min to 75% row the adjusted close price have lower values than open and closing price. The max of adjusted closing price same as close price max and it is higher than open price max.

Characteristics of the dataset:

| **Data Name** | **Data Type** | **Example** | **Description** |
|---|---|---|---|
| Open | Float64 | 34.37 | The first traded price of stock upon the opening of an exchange on a trading day. |
| High | Float64 | 35.89 | The highest traded price of the stock for a given trading day. |
| Low | Float64 | 34.12 | The lowest traded price of the stock for a given trading day. |
| Close | Float64 | 35.83 | The final and most up-to-date stock price traded on a given trading day. It does not reflect on after-hours and adjusted close price. |
| Adjusted Close | Float64 | 35.83 | A stock's closing price of given trading date after amended any distributions and corporate actions occurred before next trading day opens. |
| Volume | Float64 | 28720000.0 | The total quantity of shares traded for given stock. |

Comment: Here is a overview of the table showing the characteristics of the Apple Inc dataset. The description provides the meaning or purpose of the features. It also provides the data type and example of the features. This is for understanding the dataset as a whole but only feature to be used is the adjusted closing price.

***aapl_data.info() print out***

```
DatetimeIndex: 1497 entries, 2013-01-02 to 2018-12-11
Data columns (total 6 columns):
High        1497 non-null float64
Low         1497 non-null float64
Open        1497 non-null float64
Close       1497 non-null float64
Volume      1497 non-null float64
Adj Close   1497 non-null float64
dtypes: float64(6)
memory usage: 81.9 KB
```

Comment: The figure above is the printed output of the dataset with information about the features. The index are date time indexes containing 1260 entries. The stock price data ranged from 2013-01-02 to 2018-12-11. So there are around 5 years of stock price data from the date of printing it. I would say it is not a big dataset to work with. The adjusted closing price is type float which acknowledge the fact that it is a regression prediction problem.

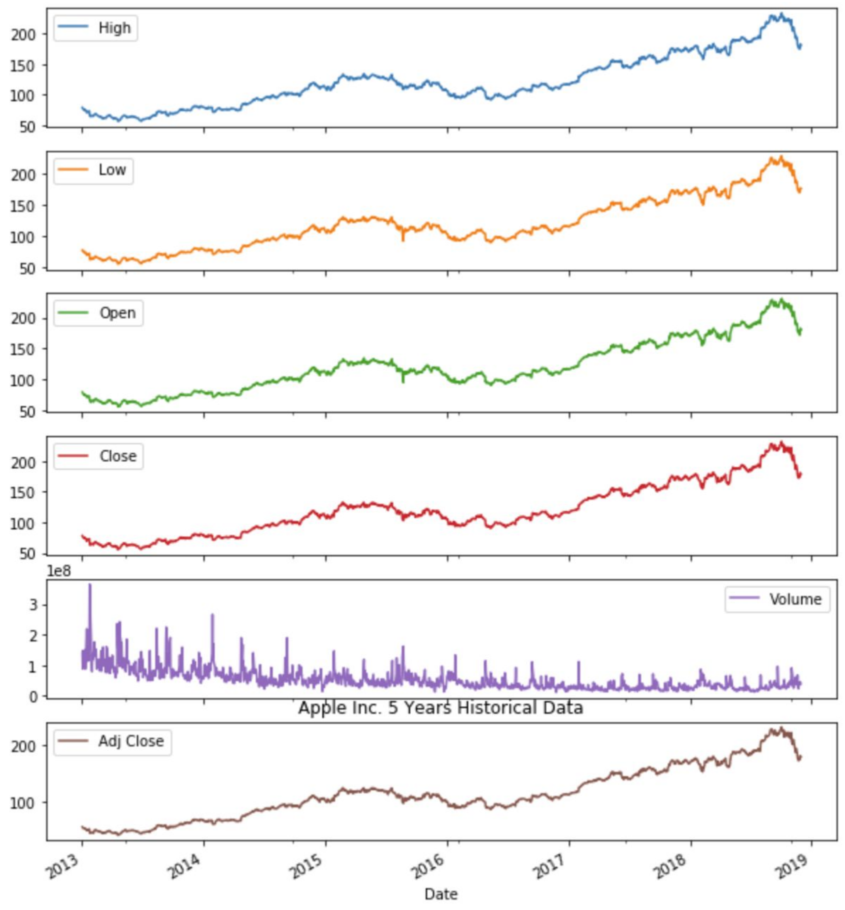
Abnormalities or interesting qualities:
The data only contained working days so there was no need to clean empty fields from data for the weekend or non-trading days.

Backtest is a test simulation of trading strategy to check its performance if it were traded historically. When training the time series dataset, we have to be careful of in-sample backtesting. In-sample backtesting is when we test the trading strategy or model using the same time period data as we trained it. This problem can be related to in machine learning where we must not use the training data for testing the model. The result can never be trusted. So we cannot use the same date range for training and testing the model.

Another issue working with time series data is to avoid look-ahead bias. A look-ahead bias happens when using data in model training that would not have been known or exist during the period being trained. As this will affect the accuracy of the result. So we cannot train the model with a date range ( e.g. future dates) and test the model with a historical date range. For example, the model was trained using 2014 data then we cannot test with 2008-2013 data. This will cause a look-ahead bias.

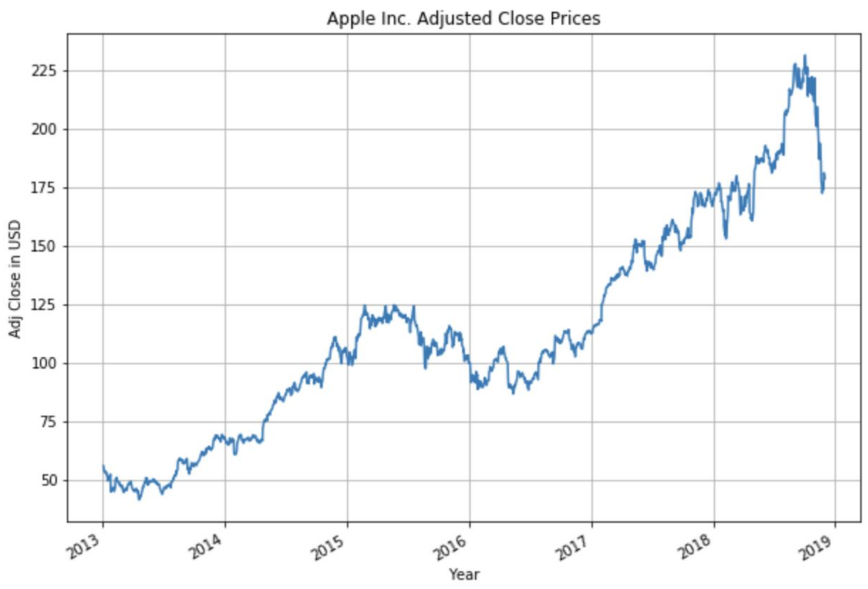
## Exploratory Visualization

Plot Apple Inc. dataset:



Comment: The plot above shows the visualization of each column or features in the Apple Inc dataset. It is just a overview of all the columns in the dataset. The first four and last subplots are about the pricing of the stock. They represent different aspect of the pricing but the visualization appeared very similar. The fifth subplot displays the volume of the stock which shows the amount of trade in the period.

Plot adjusted closing price:



Comment: The plot above is the adjusted closing price of Apple Inc. The adjusted closing price is the main feature or column to be used in the project. It will be used to train the model and we will have to make prediction on the adjusted closing price of 1 day ahead to the future as our target goal. The simple observation of the plot was that the price range around 50 to 225 USD during the 5 years period. The price is overall ascending trend in the 5 years.

## Algorithms and Techniques

I intended to use supervised learning and deep learning algorithms in the project. Supervised learning regression algorithms can predict continuous values. Deep learning can also be used to predict this type of values. They are suitable for problem and characteristic of the dataset. The problem is to predict future stock price using historical stock prices which is in the form of time series continuous data.

### Supervised Learning - XGBoost

Extreme Gradient Boosting (XGBoost) is an implementation of gradient boosting machine. It is being quite popular in many machine learning competitions. It is faster and performs better than naive gradient boosting in general. Native gradient boosting is also generally slower because it sequentially generates the next weak learner tree. But XGBoost has improved on this by having parallel tree building to speed up the process. It is able to solve regression and prediction problems in fast speed and great results. For using a supervised learning method, the XGBoost is a good option to explore.

How dataset will be handled by XGBoost: XGBoost uses the gradient boosting decision tree algorithm. In Ensemble method, we will use a group of predictors to help us make the final prediction result. When training the model it uses a ensemble technique called boosting. Boosting is a technique where we start with a initial weak model with accuracy close to random guesses. Then it iteratively add next new model which corrects the error or residual of previous existed weak model. We calculate the error value of the difference between the correct target value we want and the weak model’s prediction value. This process continues until no further improvement can be made or it has reached how far we wanted the process to be. The gradient boosting technique is where the next new model is created to predict the residual of previous models. Then using gradient descent technique to reduce the loss for the next model. This making the collection of model or final model moving towards better accuracy. Then it will be added to the collection of models to make final prediction result.

Deep Learning - RNN LSTM
Deep learning architectures are capable of regression and prediction task. Two architectures to be considered are MLP and RNN. RNN has been used for prediction tasks such as stock market prediction. It uses sequential of inputs and takes consideration of historical inputs. But other neural network architectures only treat input independently. This characteristic may be beneficial for detecting correlation and related patterns in the price movement. A particular extension of RNN to consider using is the Long-Short Term Memory or LSTM. RNN has internal memory for remembering the inputs it received. The LSTM extension is to extend the internal memory capacity. As a result, it helps to learn important experiences or patterns over a longer period of time.

How dataset will be handled by LSTM: The LSTM has a chain of repeating modules of nuruel network like other RNN but with more nuruel network layers. The cell state can be seen as a connection line going through the whole chain of cells in the LSTM RNN. The information going through the cell state can be removed (forgotten) and stored (remember) by something called gates in each cell. The information can be completely or partially forgotten or stored. So the LSTM RNN can control when to remember and when to forget historical information.

### Benchmark
There are two benchmark models to be used in this project. A simple linear regression model to give a point of reference to be compared to my chosen solution models. Another one is to have a persistence model to create a forecast performance baseline. A model performing at or below this baseline should be better adjusted or avoided.

In the linear regression benchmark model, a linear regression model will be trained and tested using the stock price data.  Then my chosen solution models will be trained and tested using the same data (e.g. date range, stock symbol). Finally, using the metric root-mean-square error to compare the results of the two models.

**Benchmark RMSE Score Results:**

| **Linear Regression (AAPL\|VISA\|TESLA)** | **Persistence Model (AAPL\|VISA\|TESLA)** |
|---|---|
| 2.549, 1.504, 9.030 | 2.545, 1.494, 9.012 |

## III. Methodology

### Data Preprocessing
For supervised learning XGBoost: Since supervised learning consist input values (x) and predict the output value (y). But the dataset we have is time series with one sequence of prices arranged by date time. So we need to alter the time series data into a supervised learning problem. We need to use time (t) as input and predict the next timestep (t+1) price. For normalization, since the XGBoost I choose to use the default form of tree so it was not necessary for this process. Unless if the XGBoost with tree has poor result then I can try XGBoost with linear model and I will have to do so.

For deep learning LSTM: Similarly to supervised learning, we need to preprocess the sequence of adjusted closing price into input (x) and output (y) values. The input (x) column will be the adjusted closing price of time (t) and the target prediction value (y) will be price at time (t+1).

In deep learning, neural networks such as LSTM can be affected by the scale of input value. Therefore it is necessary to normalise the input data for the LSTM. The data will be scaled into range between -1 to 1. This is because the LSTM default activation function is the hyperbolic tangent or tanh. The scaling can be done using the MinMaxScaler to scale the input dataset into the desired range. When scaling the dataset it might expose some insight from the test dataset into the training set. This is going to make the model untrusted because it may have a look-ahead problem. As a result, we use the MinMaxScaler which during the process will calculate the min and max values on the training dataset and they are used to scale the test dataset and prediction results.

Make dataset into stationary: The time series can be non-stationary. The dataset is non-stationary meaning that there is a structure in the data that is dependent on the time. Known as trend. So the dataset has to be transformed into stationary state by removing the trend in the data. The trend can be removed from the dataset by calculating the differences between previous (t-1) and current (t) timestep data. The differences can be obtained by using diff() function from the pandas library. This will return a stationary version of the dataset for the model to be trained.

For LSTM model the input must be reshaped to a specific format. More specifically the x column or input column will be reshaped to samples, timesteps and features. The input data has to be reshaped because when defining the LSTM layer we must specified the batch input shape. So for the amount of observations in each batch, the amount of time steps and how many features to read. After the input X is reshaped into the tuple format then it can be feed into the LSTM model for fitting it.

## Implementation

Linear Regression Benchmark -

1. Import the Linear Regression.

First of all is to import the linear regression model from the sklearn linear model package. This has to be done before we can initialise, train and test the linear regression model for benchmarking.

2. Initialise a linear regression model using no parameters.

As this is for a baseline benchmark so the model will be initialised using no custom parameters. This means the model will use default parameters to train and predict data. The parameters for initialization are the fit_intercept, normalize, copy_X and n_jobs. The default of fit_intercept is 'true', meaning it will calculate the intercept for this model. The normalize default is 'false', meaning it will not normalize the model. The copy_X default is 'true', meaning input x will be copied and not overwritten. Finally, the n_jobs default is 'none', which represents the number of jobs to use for the computation.

3. Fit the linear regression model with training data.

The train X input is in a sequence of 1 dimensional so it needs to be reshaped into 2 dimensional using reshape(-1,1). The train Y has no problem of this so it does not require reshaping. The train X and Y input are given as parameters for the fit() function of the linear regression model object.

4. Make prediction using linear regression model

After the model has been fitted we can use the model to make prediction on testing data. This can be done by using the predict() function of the model. The test X input data also needs to be reshaped from 1 dimensional sequence to 2 dimensional. Using test X input to predict the y values or next timestep (t+1) values. Storing the predicted values into a list variable named predictions.

5. Get the RMSE score of the prediction obtained from linear regression model

There is a mean_squared_error() function available from the sklearn metrics package. This can calculate the mean squared error by providing two parameters. We provide the test Y values containing the correct values and the predictions list containing the predicted values. Using these two sets to check against each other and calculate the error amount. Then finally, using the sqrt() from the math package on the calculated mean squared error value to get the RMSE score of this linear regression model performance.

Persistence Model Benchmark -
The concept of a persistence model is that it uses the previous time step (t-1) to predict the next time step value (t). This require the dataset to be formatted into two columns. It should contain the x column consisting the previous timestep value (t-1) and the y column of next time step value (t). This dataset is the lagged set created in the preprocessing stage mentioned earlier in the report. The basic implementation is to return the input value x as the output prediction value. So in the actually prediction process, we want to use the test X set to predict using the persistence model. But essentially, the text X set will become the predictions set and it will be compared against the test Y with expected output. Finally, this comparison is to calculate the RMSE score from the predictions set and test Y set.

XGBoost -

1. Import the XGBRegressor.

We have to import the XGBRegressor from the XGBoost library. This has to be done before we can initialise, train and test the xgbregressor model.

2. Initialize XGBRegressor model

The initialization of the XGBRegressor modell will be done without defining any parameters. That said, the model will use default parameters to train and predict the data. This is because I want to observe the native or pure performance output of the model at the beginning. Then later in refinement stage will explore some parameters to test for further performance increase. There are many parameters for initializing the model. But a few notable parameters are max_depth, learning_rate and n_estimators. The default values for these parameters are 3, 0.1 and 100 respectively.

3. Fit the XGBRegressor model with training data.

The training set for the XGBRegressor model was the same used for the linear regression model benchmark. Therefore, the train X input is still a sequence of 1 dimensional and it needs to be reshaped into 2 dimensional using reshape(-1,1). Whereas the train Y set has no need for reshaping just like before. The train X and Y input are given as parameters for the fit() function of the XGBRegressor model.

4. Make prediction using XGBRegressor model

Similar to making prediction using the linear regression model. The trained XGBRegressor model will be used to make the prediction based the testing data. The XGBRegressor model has the predict() function which require the test set of input value X to make prediction of y. Again, the X set input data needs to be reshaped from 1 dimensional sequence to 2 dimensional for the model. Then predict the y values or next timestep (t+1) values and store into a list variable named predictions.

5. Get the RMSE score of the prediction obtained from XGBRegressor model

For the RMSE score we first use the mean_squared_error() function from the sklearn metrics package. We provide the test Y values and the predictions list from the model's prediction values to the function. Finally, using the sqrt() function from the math package to calculate the square-root of the mean squared error to get the RMSE score.

LSTM -

1. Import the LSTM model.

The Keras package has included the LSTM model which we can import into the project. We can import it from the keras layers package. Again, this must be done before we can initialise, train and test the LSTM model. As we are designing a sequential model so we need to import 'Sequential' from keras models package. Also the 'Dense' and 'Activation' from keras layers package.

2. Initialize LSTM model

As this is for deep learning so setting up the model is slightly different to the process for supervised learning model like the XGBRegressor. First we must create a sequential model using the Sequential() then we can add layers to it. The layer to be added to this sequential model is the LSTM layer. The initialization of the LSTM layer model consist with minimal parameters and the rest of unspecified parameters will be set to default. For the units or neurons I chose the value of 1 as minimal for observing its naive performance. The parameter 'stateful' is set to true meaning the last state of sample in each batch will be used as the initial state for the next batch. The batch_size is also set to 1 for minimal or simple setup for the LSTM model. Moreover, the default activation function is the hyperbolic tangent or 'tanh'. At the end we add a Dense layer to the model and compile the model with setting the loss function to be mean_squared_error and optimizer to be 'adam'. This loss function is closest to our metric of using RMSE.

3. Fit the LSTM model with training data.

The input X must be reshaped into the tuple format which the match the batch input shape. The y value can be untouched and pass to the fit() function. The batch size to start with is set to 1 as I wanted to test the native performance. But this can be tuned to explore further potentials. The epoch value is also set to 1 which together with batch size will affect the speed of the LSTM learns. We want the LSTM to learn over each states so we don't want it to shuffle samples. We set the shuffle parameter for the fit() function to false.

4. Make prediction using LSTM model

The prediction part again is slightly differ from the supervised learning model. After we fitted the LSTM model we can use it to make prediction using the predict() function. This predict() function will need the input data and batch size. We first have to reshape the X input value from 2 into 3 dimensional array. This was done by using the reshape(1, 1, len(X)). After we reshape the X input then it can be passed into the predict() function. The batch size was set to 1 in this stage, but it can be adjusted in later refinement stage.

5. Get the RMSE score of the prediction obtained from LSTM model

This process will be identical when we get the RMSE score for the supervised learning models. We calculate the MSE value of the test Y values and predictions by using the mean_squared_error() function from the sklearn metrics package. Then using the sqrt() function from the math package to calculate the square-root of the mean squared error to get the RMSE score of this LSTM model's result.

## Refinement

The initial solutions for the XGBRegressor and LSTM models are using default parameters. This was to see the original or initial performance of these models without any tuning. The refinement solutions will try to tune the parameters to seek for any improvement in performance.

XGBRegressor -
The method for improving the XGBregressor is to tune the parameters using GridSearchCV. The GridSearchCV() function is imported from the sklearn model selection package. This grid search approach allow us to define a range of parameter values for each parameter and find the best combination out of them. Using this grid search can help to explore the good parameter values for the model to perform better. Thus it might be able to increase the performance of the model. The parameters to test on grid search including the learning_rate, max_depth, min_child_weight, subsample, colsample_bytree and the n_estimators. The GridSearchCV() function will find the best parameters, inform the best parameters and output the model initialised with these best parameters. Then we can use this improved model to fit and predict our dataset in the project. The prediction result will be used to calculate the RMSE score and see if there was any improvement.

LSTM-
Personally I think one challenge of tuning deep learning model is that the speed of each test run can be very slow based on the running platform. Therefore, it was too time consuming and so many advanced methods (if any) cannot be used. Also I have lack of experience working with LSTM tuning. Thus for the LSTM model, I was tuning the model without using prewritten function like GridSearchCV. The epoch, neurons and batch size parameters are being tuned for better performance. It was a simple refinement solution but it is an option for me to tune the LSTM. After some trial-and-error to find the best or better parameters, the model will be used to make prediction and calculate the RMSE score for reference.

## IV. Results

### Model Evaluation and Validation

Fundamentally the target models chosen for the project are both suitable and reasonable. This is because the problem of the project is a kind of regression problem. So a supervised learning model for regression and RNN were great candidates.

The initial or default parameters performed very close to the benchmark models. The first time implementing the models, the persistence model performed significantly worse than others. But this last run before submission was actually better than linear regression model. Actually, the XGBRegressor performed very bad in the AAPL dataset. It got more worse score than the benchmark models. But overall, both target models with default or simple parameters have underperformed the benchmark. This was disappointing as I expected to have better performance. The reason can be caused by many factors such as data preprocessing and parameter tuning. The tuned or improved target models have slightly increase their scoring in most cases. But most of them have very tight improvement and some case even got worst score. For the improved XGBRegressor, the three models running for each datasets have used some common parameters. The learning_rates used were 0.06 and 0.08 but not lower in 0.04 or higher in 0.1. The n_estimators used were 50 and 5 but not 25 and 100. For the improved LSTM, some parameters were also common among them. All three sets used batch_size of 1. Both AAPL and VISA used 23 neurons for improved result but TESLA used 16 neurons. The epoch values were different for the three sets, they were 1, 7 and 5 respectively.

For validation purpose, benchmark models and target models have been tested with datasets of three different stock. This was to give a diverse and widen sets of data to test how the models will perform. The three stock datasets each represents a different industry. Their volatility and trending the price are different. Also the company stability, performance and length of business history are also different.

### Justification

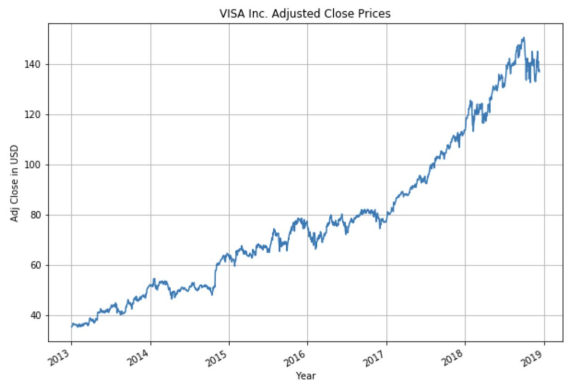
| RMSE score/ Models | Linear Regression | Persistence Model | XGBRegressor | LSTM |
|---|---|---|---|---|
| **AAPL** | 2.549 | 2.545 | 2.577, 2.545 (Improved) | 2.558, 2.543 (Improved) |
| **VISA** | 1.504 | 1.494 | 1.501, 1.485, (Improved) | 1.520, 1.480 (Improved) |
| **TESLA** | 9.030 | 9.012 | 9.224, 9.013 (Improved) | 9.036, 9.044 (Improved) |

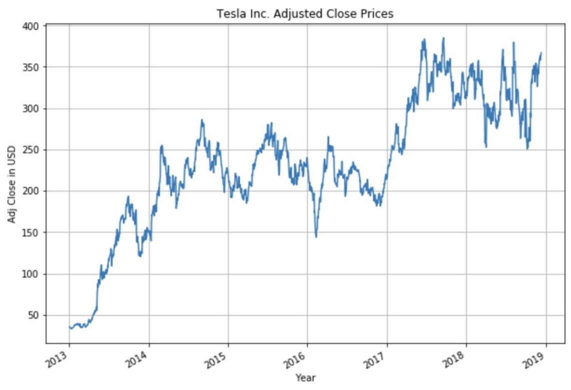
Comment: Green block is the winner among the two, XGBRegressor and LSTM (included improved version)

I used the above table to compare the result and between the two target models which got more wins in RMSE scoring. Although for not improved version models many case underperformed the benchmarks. The improved LSTM model has better score on AAPL and VISA tests than all other models. And in those two cases, the improved LSTM won the XGBRegressor if compared between themselve. The improved XGBRegressor model won the LSTM in the TESLA set, but it is still beaten by the persistence model by very little. So out of the three test cases, the LSTM won ⅔ and XGBRegressor won ⅓. I would say the LSTM has more potential but more testing is needed to be firm. I believe the final solution, as I am aware before the project started, is significant enough to give a trend indication or suggestion on the future price movement. But I am not confident enough for my final solution to solve the problem in real life scenario. However, in real life investment banks and hedge funds does use machine learning models for their auto trading system and price prediction. So in terms of real application is definitely it is significant enough as it is already been used, but of course their model is much more sophisticated.

## V. Conclusion

### Free-Form Visualization





The two graphs above are to compare the two stocks which the final improved LSTM performed very differently. The graph on the left is the VISA stock prices, which the model performed the best of itself and outperformed others. But the right TESLA stock, the improved model lost to the XGBRegressor and even worse than its own score before tuning parameter. From the visual observation, the VISA dataset has a much clearer and simple trend or movement. But the TESLA set has much greater movement and unpredictable trend progress. So a basic thought of this was that the LSTM memory was able to learn the movement from the VISA set but very unhelpful or difficult for the TESLA set.

## Reflection

Summary of the full process used in this project are in the following order:

1. Find, analyse and select the problem domain for the project.
2. Research and find dataset for the project.
3. Setup environment for making the project.
4. Data exploration and pre-processing.
5. Creating the benchmark models, and get baseline score for reference.
6. Create the XGBRegressor and LSTM models, train and test.
7. Evaluate and review the final scores.

The 1st stage finding the problem domain was not too difficult. I had a big and rough areas of which domain I am interested in to solve. Then with some research I was able to narrow them down and selected this stock/ finance domain. The 2nd stage was not hard either as there are many information on the internet about the domain and many API or resource for find dataset needed for this project. The 3rd stage I had some trouble with the platform Kaggle. I was not able to use the Yahoo Finance API because I disable the internet option and data cannot be downloaded into the notebook. In 4th stage, the pre-processing is less straightforward when the dataset has to consider the LSTM characteristic. For example, I have to scale the data into a certain range. The 6th stage is definitely the most works and hard. This was because there are two models to be configured and trained. The LSTM is much more complicated to set up when compared to the XGBRegressor.

## Improvement

There are improvements I believe can be made to this problem domain or project.

- Ensemble Model, is one suggestion from the proposal stage. Maybe it will have great performance to combine the power of supervised learning model and deep learning model. So maybe I could have somehow make the XGBRegressor and LSTM models work together for better performance.
- The LSTM can add more layers or involve other deep learning model such as MLP and NLP. Meaning there are many other models and approach that can be explored. Also other supervised learning model such as Random Forest or SVM for example.
- With better running platform or environment, I can test the LSTM or deep learning model more thoroughly. Also it will able me to train the LSTM or deep learning model which is much more sophisticated.