Tse I Ching Eugene (UID: 3035608215)

Ningoo Siddhant Darpan (UID: 3035577973)

DoDiligence: AI Stock Predictor

1. Project objectives and description[1]

Using AI/Machine Learning techniques, we have created an application that primarily aims to assist investors' decision making when trading securities, by attempting to predict the closing prices of a chosen NASDAQ stock based on the historically traded closing prices. Our program's results may also help inform users' understanding of the current market situation.

To train DoDiligence to predict stock prices, the Long Short Term Memory (LSTM) model is used. We used Google Colab to work on this program in the form of a Jupyter notebook. Furthermore, we used Pandas for data reading, numpy and sklearn for the preprocessing of the data, and Keras for the LSTM model. The final program is executable from Jupyter notebooks, where user friendly documentation allows users to understand the code in detail.

2. Highlights of DoDiligence

- DoDiligence is an AI based application that predicts the closing stock price of any NASDAQ listed stock based on the past closing prices.

- DoDiligence is a LSTM based AI model that receives the users input of the stock that requires prediction and searches the web for the historical key stock metrics (eg. Open, Close, High, Low, Volume, Adjusted Close). It uses the Closing Price for the model training and fits the model to adjust per a month's worth of data.

- The final results are not as expected, as we realized towards the end that, ultimately we are only giving the model old prices which do not have any relation to the future prices. For future improvements, we should work towards predicting the stock price based on market news and sentiments which would be performed via data mining through numerous online forums, financial articles and using this data to read for their emotions (a topic that was covered in COMP3359 Lectures). We would then integrate this with the current model.

[1] This section is altered from our Interim Report.

3. Tasks achieved

3.1 Model implementation[2]

DoDiligence mainly relies on the Long Short Term Memory model, a commonly used RNN model that is great at learning the long term dependencies and is said to be one of the more accurate models when it comes to time series data.

We have built our model in a 4 layer approach using the Keras Sequential model with LSTM layers. The first 2 layers are the LSTM layers, with 50 neurons. Moving forward, we needed to implement the dense layer network for the model to learn the dataset in depth. Hence, we implemented the next 2 layers being the dense layer network with 25 neurons in the first and 1 in the second.

Once the architecture was completed, we focused on compiling it and picking the most suitable loss evaluator, which is the Mean Squared Error (MSE). This is a necessary step as we have to guide the model and let it know where it is going wrong and where it isn't when comparing its learning along with the actual values. Following this, we need to implement an optimizer that would accordingly advise the model on changing the weights of the model and so on. Therefore, in order to use MSE to optimize the model, we chose the Adam Optimizer.

When it came to fitting the model on our training dataset, we were initially confused as there were huge errors in the predictions, but we realized that increasing the batch size and epochs was leading to rather consistent predictions. Therefore, for the fitting, after a bit of experimentation, we came to the conclusion that the values for Batch_size are ideal to be 5, while the value for Epochs to be 7.

```
[ ]  #Building the layers to the LSTM model
     model = Sequential()
     model.add(LSTM(50, return_sequences = True, input_shape = (x_train.shape[1], 1)))
     model.add(LSTM(50, return_sequences = False))
     model.add(Dense(25))
     model.add(Dense(1))
```

```
     #compiling and fitting the model to the dataset
     model.compile(optimizer = 'adam', loss = 'mse')
     model.fit(x_train, y_train, batch_size = 64, epochs = 50)
```

Figure 1.

[2] This section is altered from our Interim Report.

During the second phase of this, when implementing a new feature of predicting over a time horizon, we had come across some less favorable results, and after deeper experimentation, we found that it would be better if the model adjusted weights based on each month's data (i.e. Batch Size = 64).

For implementing the feature for predicting over a time horizon, we referenced Krish C Naik's[3] work. We only referenced the way he had predicted over the span of the next 30 days. He had implemented it in a way that it would predict for t+1, and append that t+1 prediction to the dataset, and with the updated dataset he would retrieve t+2, and follow up until t+30. We used this same method to work on our feature of predicting over a time horizon of X days (where X is an input provided by the user).

We first reshape the data here to prepare for prediction.

```
[26] lhd = len(test_data) - 100
     x_input=test_data[lhd:].reshape(1,-1)
     final_input=list(x_input)
     final_input=final_input[0].tolist()
```

Prediction is performed here using the LSTM model, and the user can now view the predicted price for the *xth* day.

```
[27] # prediction for next x days
     pred_output=[]
     n_steps=100
     i=0
     while(i<days):

         if(len(final_input)>100):
             x_input = np.array(final_input[1:])
             x_input = x_input.reshape(1,-1)
             x_input = x_input.reshape((1, n_steps, 1))
             pred_unfinished = model.predict(x_input)
             final_input.extend(pred_unfinished[0].tolist())
             final_input = final_input[1:]
             pred_output.extend(pred_unfinished.tolist())
         else:
             x_input = x_input.reshape((1, n_steps,1))
             pred_unfinished = model.predict(x_input)
             final_input.extend(pred_unfinished[0].tolist())
             pred_output.extend(pred_unfinished.tolist())
         i += 1
     print("The price predicted by DoDiligence is:")
     print(scaler.inverse_transform(pred_output)[-1][0])
```

Figure 2.

[3] We have only referenced his work for predicting over the next X days. We have not referenced any other part from his work.

One key characteristic that we would like to highlight about our model is its reusability. As we would only be taking in the data for the required ticker as specified by the user, the model would get the data from the web source while running the application, meaning that without the data, the model remains untrained.

Similarly, the model would only be training when the user input is received, meaning that the model would be training every time whenever the application is called. Due to this, it takes some time to get the prediction from start to finish, in comparison to a program that has downloaded and saved all the required static data from other sources into a CSV file, and has a model that has been previously trained and requires no subsequent training.

3.2 Data engineering[4]

In order to retrieve, sort and effectively use the data necessary for our program, we needed to use libraries such as datetime, pandas and numpy and SK Learn.

We decided to obtain the data ranging from 01/01/2012 up until the day before the running of the model. This is further explained in the following sections. Hence, in order to get the date of the day before the running, we had to use the DateTime library (Figure 3).

As there were several online price quoting websites on the web from where we could easily get the data, we used the Pandas DataReader to directly get the data from Yahoo.

Another item that we require input from the user is their time horizon of interest that they would like DoDiligence to predict the price over. For example, if the user would like to find out the closing price of a certain stock 15 days later, the user would input 15 when prompted.

Next, we retrieve yesterday's date (the day prior to running the program).

```
[2]  #Get the date before the day of running the program
     yesterday = (datetime.now() - timedelta(days = 1)).strftime('%Y-%m-%d')
```

Figure 3.

[4] This section is altered from our Interim Report.

```
#Get the stock that needs to be predicted
print("Please input the NASDAQ stock TICKER for the stock you would like to predict:")
stockpick = input()
#Get the time horizon that the user would like to predict over
print("Please input time horizon you'd like DD to predict over")
days = int(input())
#Retrieve data from Yahoo Finance through Pandas datareader
df = web.DataReader(stockpick, data_source='yahoo', start='2012-01-01', end= yesterday)
```

Figure 4.

Once the data has been collected, we need to normalize the data and scale it within the range of 0 to 1. This was then done by using the preprocessing feature MinMaxScaler in the SK Learn library.

```
#normalization of range of stock prices
scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(dataset)
```

Figure 5.

Now the data is ready to be split into different datasets for different purposes. Originally we had mentioned that we would be splitting the data in 4, namely Training, Train-Dev, Dev and Test Datasets, however during the implementation we found that using just Training and Testing Dataset is sufficing the model and further splitting is not required.

During implementation, we also found that using data that ranges from 2016 to 2019 may not be enough for training, hence we decided to extend our data range from 01/01/2012 to the most recent day (i.e. the day before running the Application). Hence for the splitting we decided to maintain the training dataset to contain the first 80% of the whole data, while the testing dataset will be using the most recent 20%.

After the splitting and assigning of the dataset, we would then convert the dataset to a numpy array and reshape the dataset such that it can be taken in as an argument into the model.

```
#Group the data as per their train/test size
train_data = scaled_data[0:trainin_data_length , :]
#Append the corresponding data to the train set
x_train = []
y_train = []
for i in range (60, len(train_data)):
  x_train.append(train_data[i-60 : i, 0])
  y_train.append(train_data[i , 0])
```

```
[259] #Convert to numpy array so that other functions can be done
      x_train, y_train = np.array(x_train), np.array(y_train)
```

```
[260] #The model would only take a 3D array, hence reshaping is required
      x_train = np.reshape(x_train, (x_train.shape[0],x_train.shape[1], 1))
```

Figure 6.

```
[263] #Append the corresponding dataset to the correct arrays
      test_data = scaled_data[trainin_data_length-60 : , :]
      x_test = []
      y_test = dataset[trainin_data_length: , :]
      for i in range (60, len(test_data)):
        x_test.append(test_data[i-60 : i, 0])
```

Before proceeding to LSTM model prediction, the data is then converted to numpy array and reshaped.

```
[264] #Reshape the array to proceed to LSTM model prediction
      x_test = np.array(x_test)
      x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
```

Figure 7.

3.3 Experimentation [5]

Whenever we had tried adjusting the below variables in our application, we made sure not to change any other variables concurrently as we would want to know the difference that specific change makes to the application.

[5] This section is altered from our Interim Report.

3.3.1 Vary the number of Batch size & Epochs

Originally we had left the Batch Size and Epochs to be set as 1 and 1 respectively, and whenever we reran the model (meaning that we would also have to retain the model), we were getting big differences in the prediction value and in the RMSE value. When looking deeper into the model implementation, we noticed this and decided to fix it. After several rounds of trial and error and comparing the predictions and the RMSE, we decided to stick with 5, 7 for batch size and epochs respectively. These values were also providing us with rather consistent results between the reruns.

However, when we were implementing the new feature of predicting the closing stock price over a specified time horizon, we found that the MSE loss was still very large. In order to fix it we ran more tests and experimented with the different combinations of the epochs and batch size, where the model would adjust its weights based on a week's worth of data (batch size = 80, epochs=70) to half a decade even (batch size=20, epochs=5). We later came to the conclusion that the loss is much smaller when the models readjust the weights based on a month's data, hence bringing the batch size to be 64 while the epochs to be 50 (as shown in Figure 1 as above). This is further justified by the historical movements of Stocks in NASDAQ where it takes a month for an upwards or downwards trend to settle in. Due to this, we found it valid enough to be the final fitting.

3.3.2 Vary the number of neurons in LSTM & Dense Layers

We attempted to change the number of neurons in both LSTM and dense layers and carried out several trial rounds for each test value. However, due to there not being any substantial change in results, we deemed this experiment to be inconclusive.

3.3.3. Frequency of days (e.g. closing prices of alternating days instead of every day)

As we had initially mentioned in our proposal, we would be using the data from 01/01/2016. However when we were evaluating our model, we found that the results were very inconsistent and in order to fix that we would require to increase batch size and epochs. In order to do that, we considered that increasing the range of data that we would be using would be most ideal. Once this was done we started seeing more consistent results.

3.3.4 Graph Plotting

During the refinement process, we experimented on the graphs to see if there was a better way to refine how we presented the data. However, we ended up providing one additional graph which specifically shows the relationship between the real price and the predictions during the testing phase (see Figure 8).

```
[30] #Comparing the model's predictions with actual values (zoomed in)
     valid = data[trainin_data_length :]
     valid['Predictions'] = predictions
     plt.figure(figsize=(16,8))
     plt.title('DoDiligence - Report Generated for ' + stockpick)
     plt.xlabel('Date', fontsize = 18)
     plt.ylabel('Close Price USD', fontsize=18)
     plt.plot(valid[['Close', 'Predictions']])
     plt.legend(['Real Value', 'Predictions'], loc = 'upper left')
     plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  This is separate from the ipykernel package so we can avoid doing imports until
```

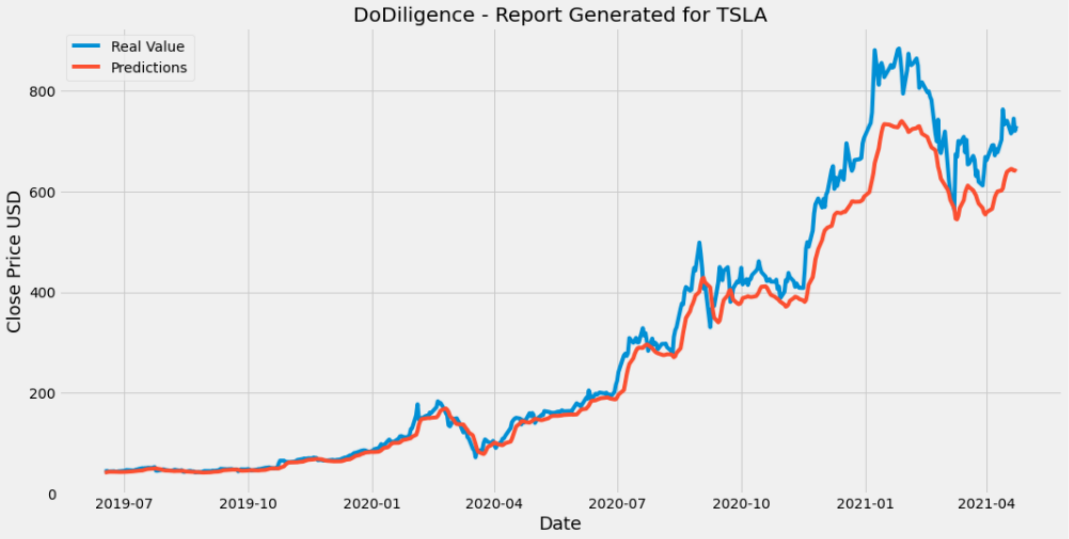


Figure 8.

Additionally, we also included a final predicted price chart that shows the movement of the predicted price over the specified time horizon. Initially we made this chart to append the predicted prices to the whole historic price, but later changed it to only show the predicted price as we found that would be more intuitive as this graph is supposed to highlight the predicted prices only.

```
plt.figure(figsize=(16,8))
plt.title('DoDiligence - Charted predictions over time horizon of ' + str(days) + ' days for ' + stockpick)
plt.xlabel("Date")
plt.ylabel('Close Price USD', fontsize=18)
plt.plot(scaler.inverse_transform(pred_output))
plt.show()
```

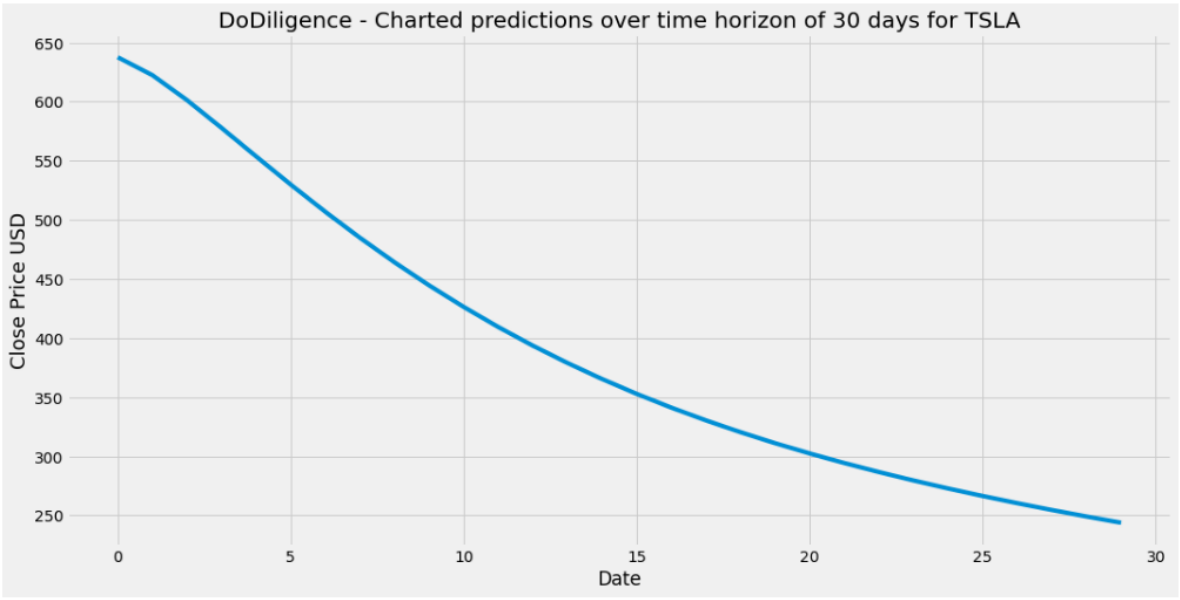


Figure 9.

4. Results of model evaluation & issue analysis

We have witnessed that due to the nature of our application, and how our model needs to be retrained every time before getting the actual price prediction, the predictions are generated inconsistently. Hence, it gives varying results for the same input when run multiple times.

For instance, when running the application for the stock AAPL with a time horizon prediction for 30 days, with an **additional change** we made by changing the date of the most recent closing stock price to 22-May-2020 instead of “yesterday”. Under these circumstances, it had predicted the price as 67, 87, and 97 (in no chronological sequence), with one of the three predictions being a pessimistic prediction.

Below we have delved into the results the model gives for a future stock price prediction, as well the prediction it gave for a date in the past (in order to see how the application is performing) and compared it with the results with that of Naik.

4.1 Results for May 23, 2021 AAPL Prediction

Multiple runs were made for this set of input, where we received the predicted prices as: 63.24, 48.54, 82.39, 64, 57. This is in contrast to the market consensus price target of 125.
Below are the results that were obtained for the time it predicted the price to be 57 in 30 days.

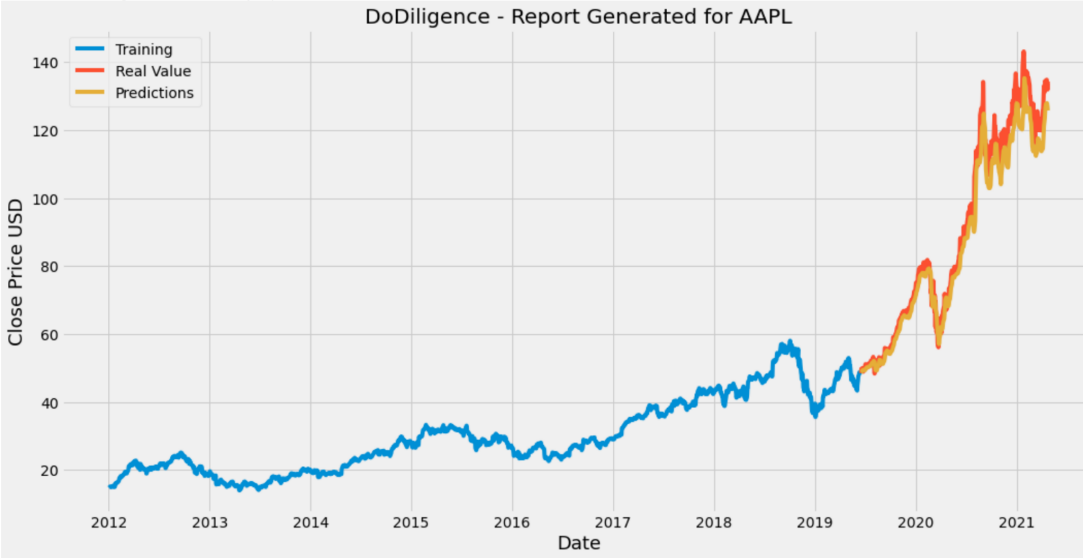
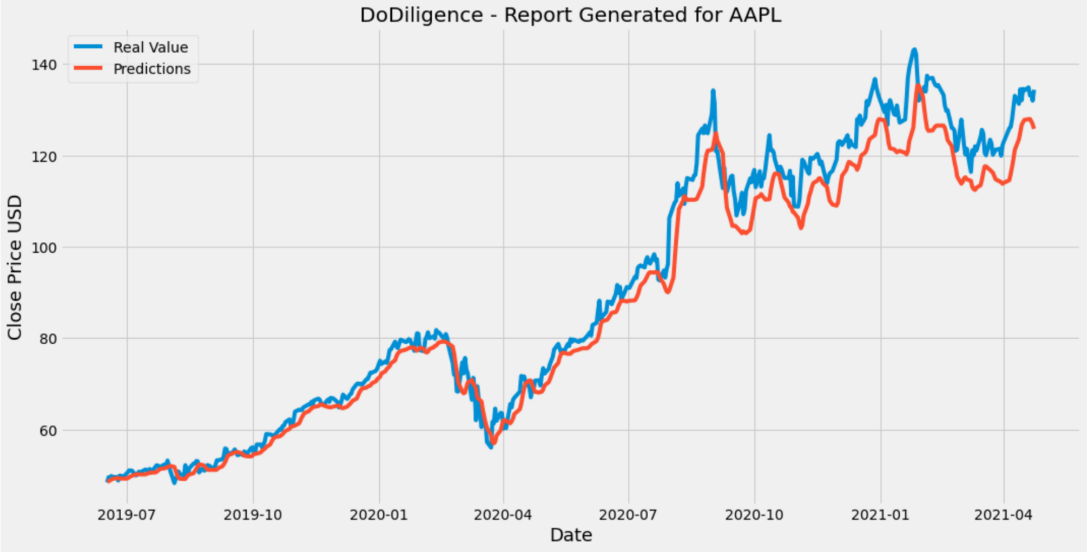


Figure 10.




```
The price predicted by DoDiligence is:
57.75052561507488
```

Figure 12.

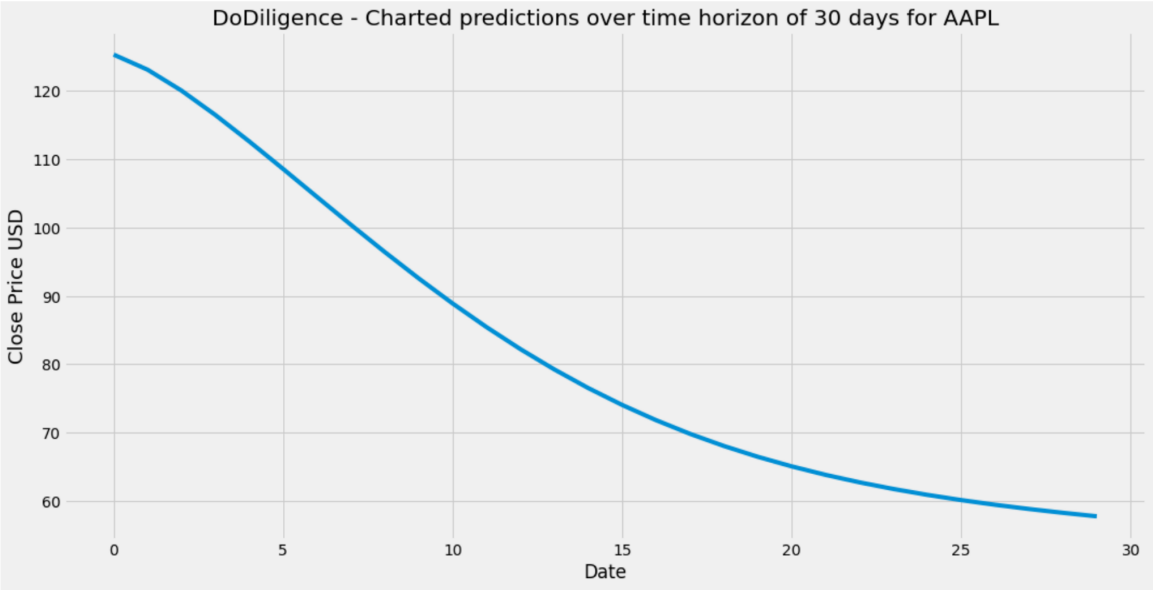


Figure 13.

4.2 Results for Jun 22, 2020 AAPL Prediction

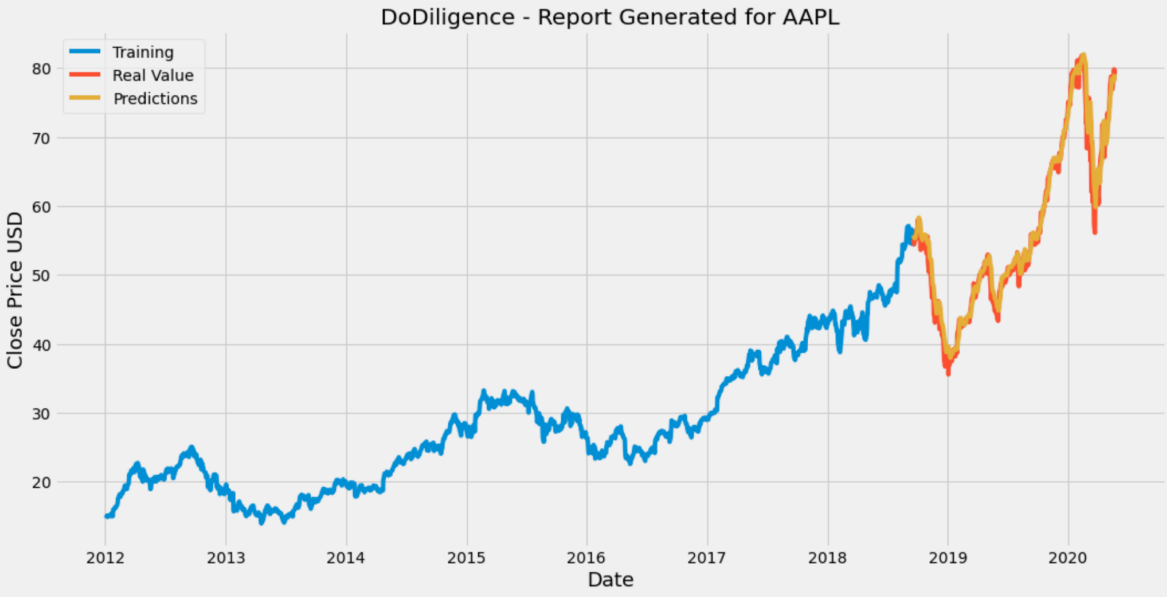


Figure 14.

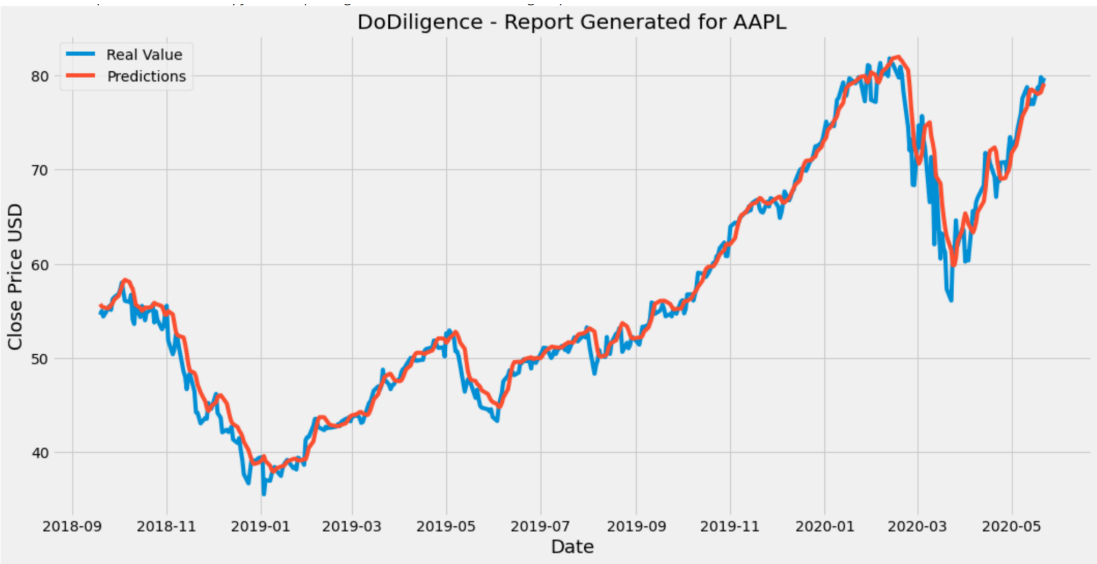


Figure 15.

```
The price predicted by DoDiligence is:
86.51123646130372
```

Figure 16.

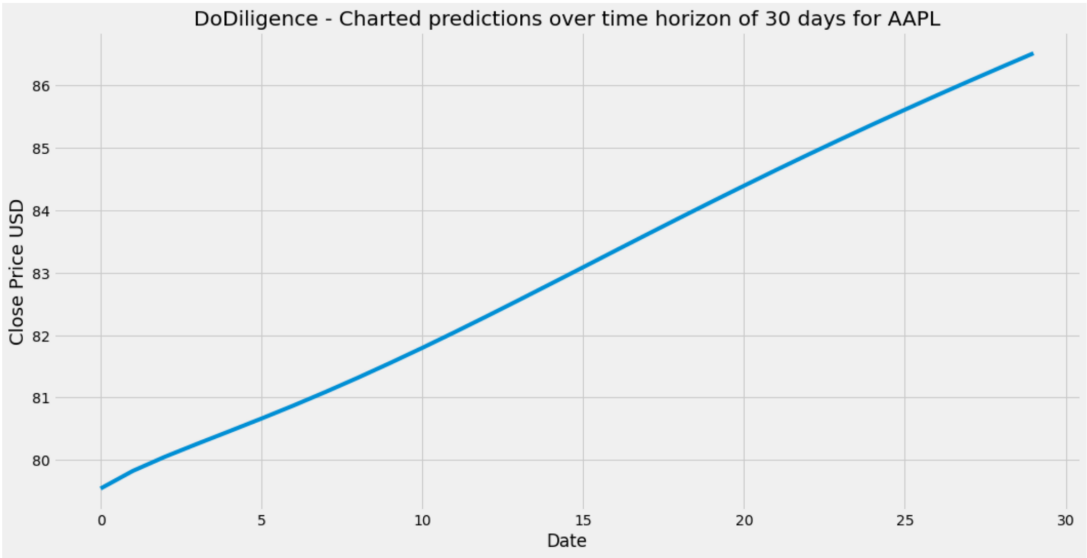


Figure 17.

Whereas the actual price was 89.72 (according to Yahoo Finance):

Jun 22, 2020	87.83	89.86	87.79	89.72	89.27	135,445,200

Figure 18.

4.3 Results obtained by Naik for Jun 22, 2020 AAPL Prediction

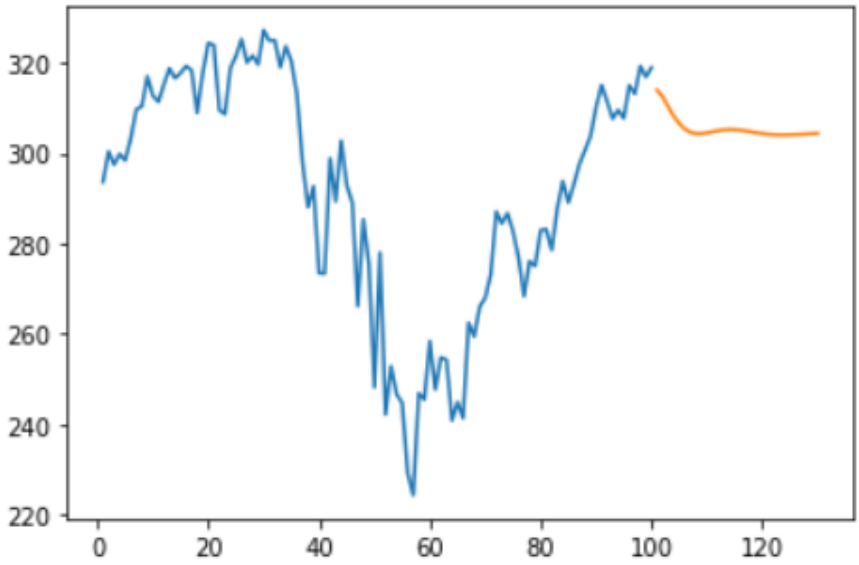


Figure 19.

The X-axis is the last 100 days of data obtained, with the addition of the 30 days of closing predicted (the orange plotted line).

The Y-axis is the closing price recorded in USD. Here we see a range of prices from 220 to 320, whereas we have a range of 60 to 80 – this is because after Naik published his work, AAPL stock underwent a split of 4:1, so all the price graphs shifted by 4 times.

4.4 Issue Analysis

4.4.1 Inconsistent predictions

As mentioned in the beginning of this section, our application outputs inconsistent results, due to running as a rerun every time when used.

4.4.2 Undervalued results

When looking at the predictions for 23 May 2021, we see that there is a range of predictions where this range is nowhere near the price target set out in the market. Looking deeper into the graph of test data set predictions plotted against the real values, we see that the prediction line is consistently below the real values. As for the predictions, the prediction for the T+30th day is based on the predictions of T-70 days up until T+29th day. Because of the test predictions being undervalued, there is a high likelihood that the prediction values have been undervalued as well.

Hence, this leads to the huge difference between the price target and the prediction made by DoDiligence.

When looking at the predictions for 22 June 2020, we see that the test data set predictions are in line with the real values, and hence the predictions as well are in line with the actual value that AAPL closed on 22 June 2020. When comparing this with the result that Naik had obtained, we see that our prediction is on par, whereas his is less accurate.

However, after evaluating, we still face the outstanding issue about why the model works fine for the 22 June 2020 prediction but for the 23 May 2021 prediction it is not performing satisfactorily.

5. Possible limitations and remaining issues

5.1 Remaining issues

5.1.1 Under-predicting due to batch size and epochs

We believe that the reason why the model is not performing satisfactorily for the 23 May 2021 prediction is because of the unfavorable epoch size and batch size. We see that for the 22 June 2020 prediction the test data set prediction is perfectly in with the real values, as the model has just the right ranges for itself to adjust the weights, but if we look at the same graph for the 23 May 2021 test, we see that the adjustments are following a bit later than needed. Our hypothesis is that the batch size and epochs are perfectly fit for the 22 June 2020 prediction, but it is not matching for the data set for 23 May 2021 as there is almost an additional whole year's worth of data – hence the fit is not right.

5.1.2 Inconsistent Predictions

In order to fix this, we believe the best way would be to make the whole application less user-input dependent. This means that everything would be prepared beforehand, including all the data needed instead of using pandas web reader dataframe. Due to this, we would be able to pre-train the models before the user uses the application, meaning that the model would remain constant. In order to implement this, all the listed tickers data would be needed in a CSV format that is stored in the same repository as the Jupyter notebook. After that multiple models would be required to be trained as each and every single model needs to be exclusive to one and only one ticker. Based on the static data obtained with a fixed length of data we would be able to perfect the model.

5.2 Limitations and proposed improvements

We realized that our approach of using historic closing prices to predict future closing prices is not the most optimal. As the historic closing prices do not sufficiently reflect the current market factors including company financials, the economic situation, market sentiments, and so on. In order to get a true representation of this, we would have to predict the stock price based on the aforementioned “true” factors. This would have to be done via mining for written data that includes sentiments and financials that reflects a general consensus of stakeholders of a stock. Such data would be found on online forums, financial articles, and so on. After obtaining this, we would have to segregate the data based on type (i.e. Sentiment or Financial).

As a further step, we could utilise supervised learning by integrating this model with our current model to find a correlation between historic market sentiments/financials of a firm and historic prices of that firm’s stock. Based on supervised learning principles, past sentiments and financials of a firm would act as “inputs” and past stock closing prices would act as “outputs”. By learning the relationship between the two, we would be able to predict future closing prices more accurately by utilizing the current market sentiments and financials as a support to further enhance the prediction of the closing stock price in the future.

6. Referenced materials

Naik, K. (2020). krishnaik06/Stock-MArket-Forecasting. Retrieved 4 April 2021, from https://github.com/krishnaik06/Stock-MArket-Forecasting/blob/master/Untitled.ipynb

“Stock Price Prediction App Using Machine Learning Models Optimized by Evolution” - explores recurrent neural networks, Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU) and more. Source: https://www.cse.ust.hk/~rossiter/fyp/103_RO4_Final_201819.pdf

7. Video link

The following is a Google Drive link to our video submission:
https://drive.google.com/file/d/1IYM7JPV9JSKnbM8c7MoaivmbFX3YfKwN/view?usp=sharing