Shay Dineen

6/16/18

Report #1

**Overview**

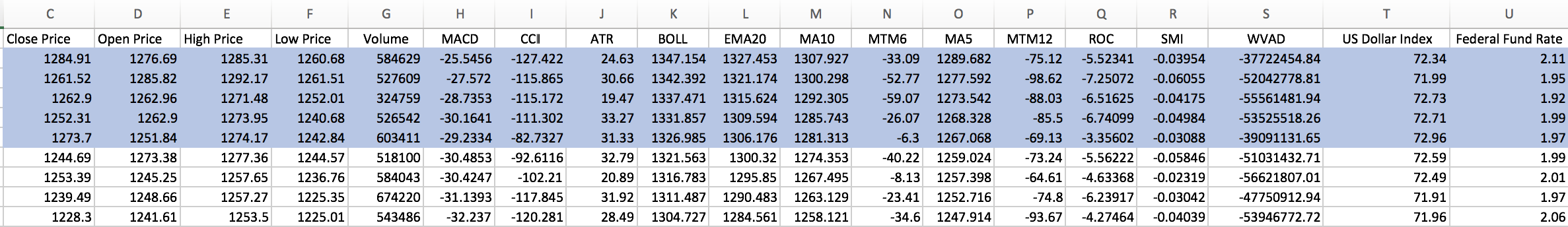
Through the course of my Summer Research, I have been incredibly fortunate to have access to a Nvidia P100 based server. With this capability, I have been able to run a number of experiments to test the credibility of Bao, Yue, and Rao’s deep learning framework for financial time series prediction. In this write up, I hope to emphasize arguably the most important aspect of this validation research. This point deals with the construction of my LSTM. Specifically, I would like to illustrate how I have been shaping the financial data for the LSTM so that the LSTM does not have access to any financial information that it would otherwise not have in the real world.

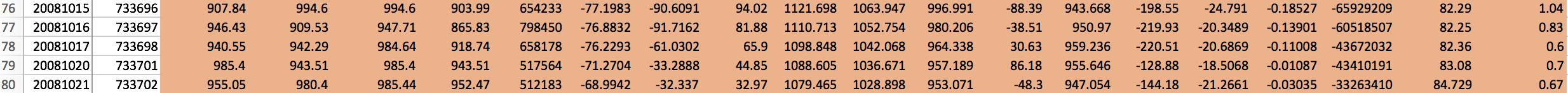
Originally, it seemed that Bao, Yue, and Rao achieved their ‘remarkable’ results because their LSTM was fed inputs that it would otherwise not have access to. For example, Bao, Yue, and Rao’s framework would fail to accurately predict tomorrow’s closing price in the real world if it was fed data about tomorrow’s closing price. Simply put, making a prediction on tomorrow’s closing would be useless if you gave the LSTM tomorrow’s closing price. In this case, the LSTM would not be learning the complicated patterns that we hope it would, but would rather just be ‘cheating.’ When shaping the inputs for the LSTM, I paid careful attention to making sure that I was not making this mistake. To illustrate my methodology, I will walk through the exact steps that I used to shape the input data. To better follow the steps I took, I will exclude the data preprocessing phase where I scale the data between 0 and 1.

**Data Shaping**

To begin, we will first look at the data in its original excel format. As it is possible to deduce that the data is correctly shaped from only a few examples, we will be looking at the first five trading days and the 75th to 79th trading days to ensure that that the data is properly shaped.

Exhibit #1: Exhibit 1 (pictured below) shows the highlighted cells that we will be looking at through the course of the data shaping phase. The first five cells (colored in blue) are the first five trading days and will be used to show that model looks back the correct number of time steps. The five rows colored in orange are the first five days where the predictions will be made.





As you can see there are total of 19 indicators in the S&P 500 dataset that we will be inputting into the LSTM.

I have used two separate methods when testing LSTM performance. One method implements the traditional 80% train and 20% set sets and the other method implements the ‘sliding window’ train/test approach. I have implemented these methods in two different files and used slightly different data shaping techniques for both. Although both methods lead to the correct data shape, I will explain how each of these methods achieve the same, correct result.

**80% Train 20% Test**

Exhibit #2: Shows the importing of the S&P 500 dataset. The first two columns of the dataset are excluded as they are just the date and time variables which are unneeded by the LSTM.

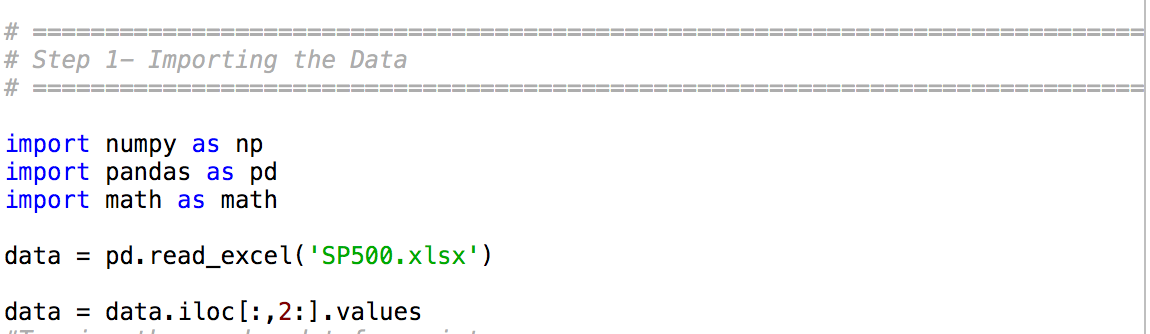


Exhibit #3: As you can see, rows indexed 0 through 4 in the Numpy 2D array correspond to the exact same values that can be found in the blue highlighted rows in the Excel file.

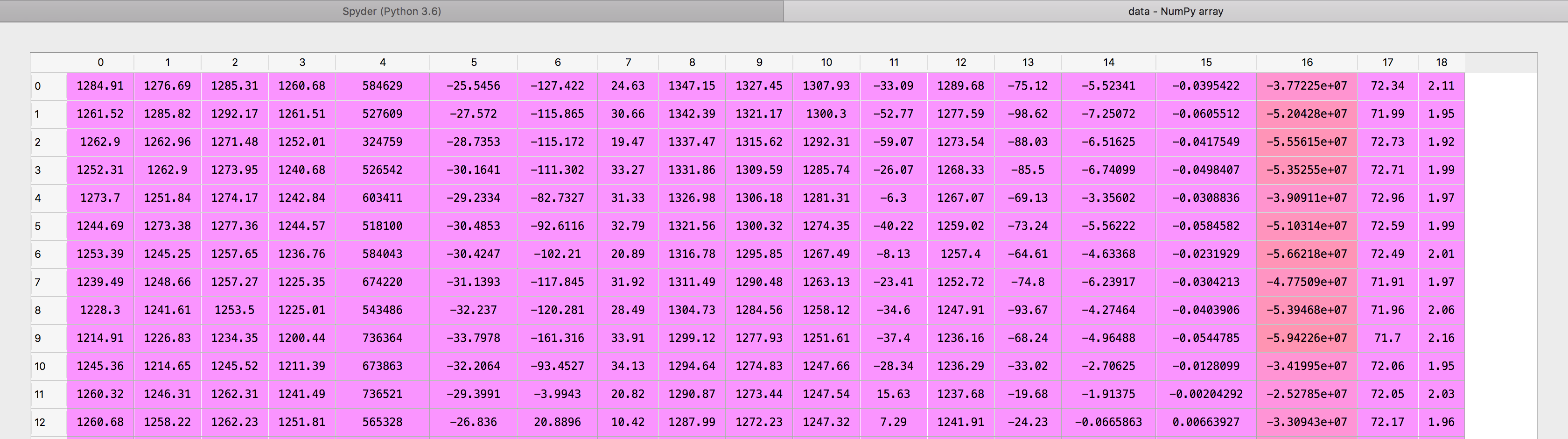


Exhibit #4: Shows the shape of the data and training set. As you can see, the training set contains 1662 trading days which corresponds to 80% of the overall dataset.

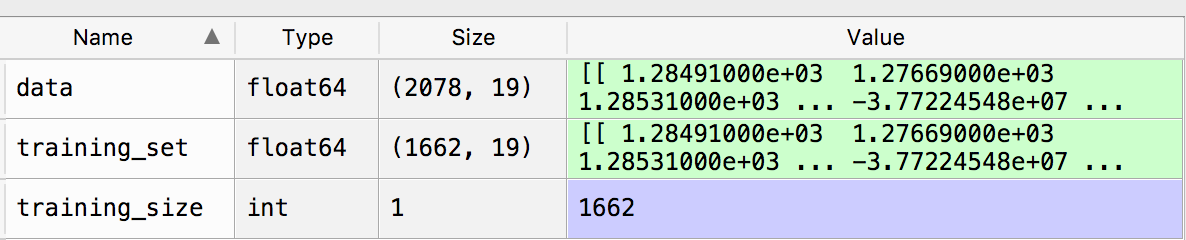


Exhibit #5: I created two Python lists to which the data will be appended to. Although the number of time steps in this report is set to 75, the number of time steps can easily be changed and will have no impact on the correctness of the shape being fed into the LSTM.

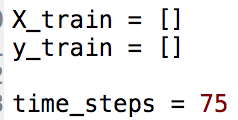
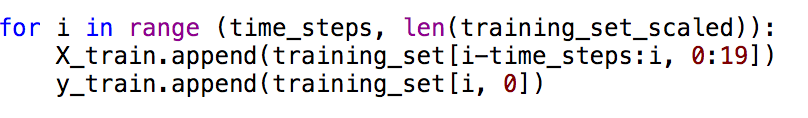


Exhibit #6: Shows the for-loop in which the data is correctly appended to the ***X\_train*** and ***y\_train*** lists



In the above code snippet, I have started the for-loop at the number of time steps as we will have to collect data from ***i*** minus the number ***time steps*** all the way up to day ***i-1***. Since ***i*** is the upper bound, it excluded in Python so this way we will only get data from ***i-timesteps*** to ***i-1*** in the ***X\_train*** list. We will be using the information from ***i-timesteps*** to ***i-1*** in ***X\_train*** to be able to make a prediction for day ***i***. In addition, we will be using all 19 indicators for prediction so we include them in the 0:19 part of the ***X\_train*** append statement.

Lastly, the ***y\_train.append()*** statement correctly appends the correct closing price for day ***i***. To do this, the closing price on day ***i*** is appended to ***Y\_train.*** We will be using the previous 75 days’ worth of data to make a prediction for day ***i***.

The idea behind the for-loop is that any given index in ***X\_train*** will correspond to day ***i-1*** and that same index in ***y\_train*** will correspond to the closing price on day ***i***.

An important note is that although the code snippet above says ***training\_set\_scaled*** in the beginning of the for-loop, the ***training\_set\_scaled*** Numpy array and the ***training\_set*** Numpy array are the same length. I am not using the ***training\_set\_scaled*** data for this report for readability.

Exhibit #7: The below code snippet shows how ***X\_train*** and ***y\_train*** are turned from lists to Numpy arrays and also shows how ***X\_train*** is then reshaped into the 3rd degree tensor that the Keras’ LSTM expects.

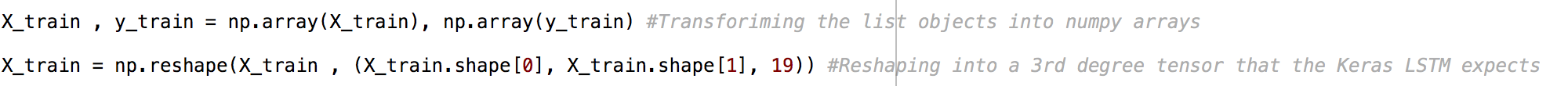


Exhibit #8: The below screenshot shows the shapes of ***X\_train*** and ***y\_train*** after the for-loop and reshaping. Since we are using 75 time steps, ***X\_train*** and ***y\_train*** will only contain 1587 rows (the difference between 1662 and 75). ***X\_train*** has the shape that we would expect- 1587 training points, a column value of 75 which corresponds to the number of time steps, and a depth of 19 which corresponds to the indicators for a day in row **X** and column **Z. y\_train**is also a column vector of length 1587, which corresponds to the closing price that we are trying to predict.

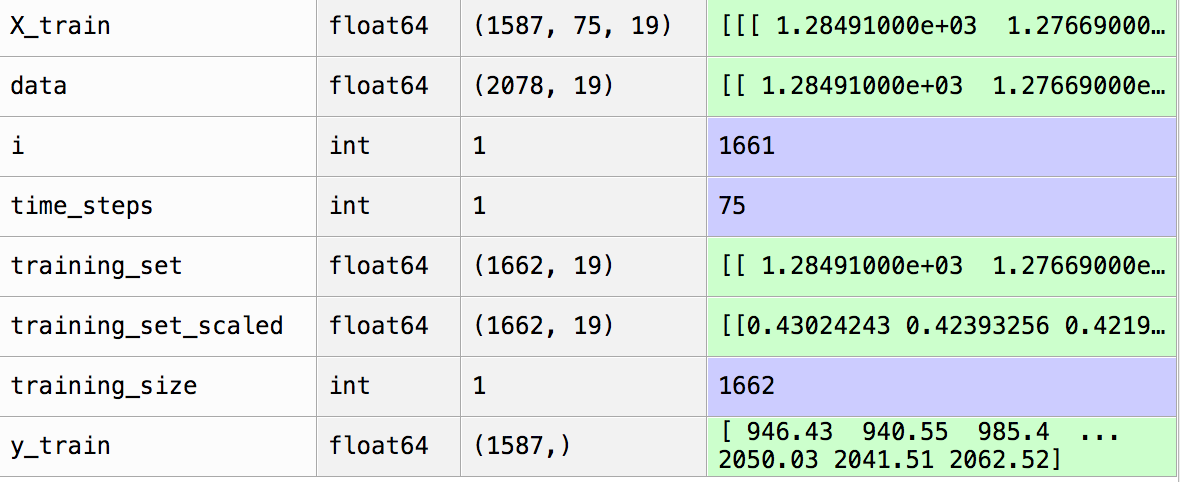
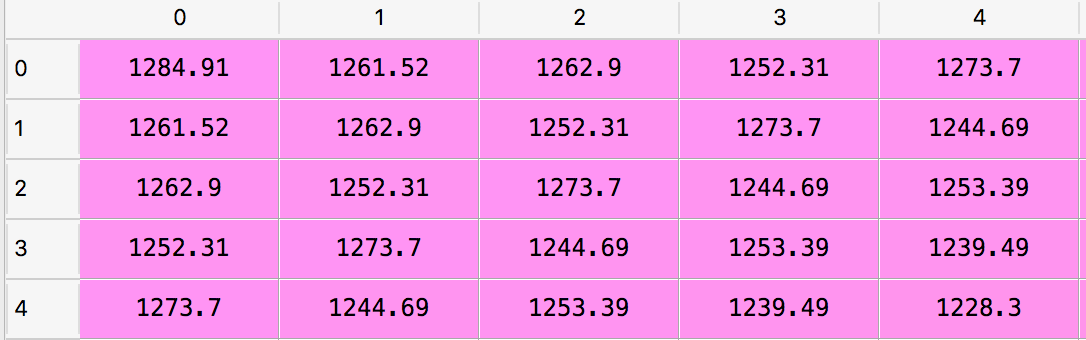
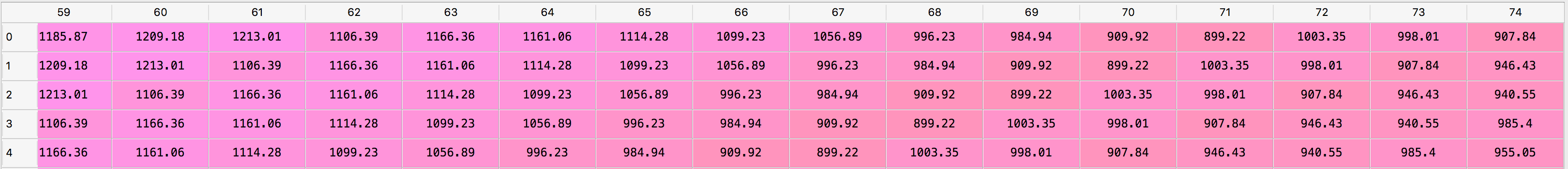


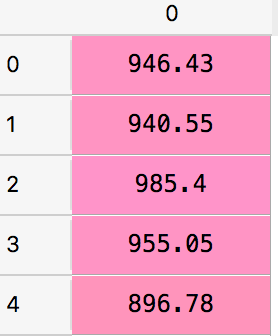
Exhibit #9: The screenshot below shows ***i-time\_steps*** in the first five rows of ***X\_train***. As you can see, index (0,0) corresponds to the first trading day’s closing price as shown in the blue highlighted cells in Exhibit #1. Index (0, 1) corresponds to the second trading day’s closing price as shown in the blue highlighted cells. This pattern continues to index (0, 74). Index (1,0) corresponds to the second trading day’s closing price since we have moved one day forward in the training set. Moving one day forward means that we exclude the closing price found in index (0,0). Likewise, index (0, 2) corresponds to third trading day in the blue highlighted cells and the closing price at (0,1) has been omitted as we have shifted another day forward.



The above screen shot is used to show that the for-loop correctly looked back the correct number of time-steps for each training example in ***X\_train***.

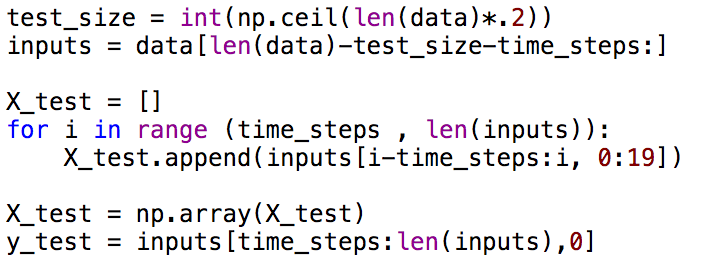
Exhibit #10: Below you will find screen shots of ***X\_train*** and ***y\_train*** at their first observations. These screenshots are used to show that LSTM does not have access to information that it normally would not have.





The closing price shown at index (0,74) corresponds to the closing price in the first row of the orange highlighted data. Similarly, ***y\_train*** at row 0 corresponds to the second row’s closing price in the orange highlighted cells. The first row in ***X\_train*** contains the first 75 trading days’ information to be able to predict the 76th day’s closing price. The closing price shown at index (1,74) now corresponds to the closing price found in the second row of the orange highlighted data. The second row of ***y\_train*** corresponds to the third row’s closing price in the orange highlighted cells. The second row in ***X\_train*** contains trading information from the second observation in the excel spreadsheet to the the 76th observation. This information is used to predict the 77th day’s closing price. Since we are only using the past 75 days’ observations to predict the next day’s closing price, the LSTM is unable to ‘cheat.’

Exhibit #11: This screenshot shows the code that is used to create ***X\_test*** and ***y\_test.***



In exhibit #11, ***X\_test*** and ***y\_test*** consist of 20% of the overall training days, leading us to have 416 observations in each set. Due to the time step constraint, both ***X\_test*** and ***y\_test*** use information on trading days that are used in ***X\_train*** and ***y\_train.*** For example***, X\_test*** starts at index 1662 in the original dataset, however; ***X\_test*** still needs to look back 75 trading days to be able to make a prediction. Because of this, ***X\_test*** will be using information on the last 75 trading days in ***X\_train*** to be able to make its first prediction. It was for this reason that I created a new variable called ***inputs***.

The ***inputs*** variable contains all the trading days that will be needed for the LSTM to make test predictions. The for-loop used in the above code snippet is very similar to the one used for shaping ***X\_train*** and ***y\_train*** except that ***y\_test*** is omitted in this loop. ***y\_test*** is created by taking the trading days in ***inputs*** from the number of time steps to length of inputs.

Exhibit #12: Shows the shape of ***X\_test*** and ***y\_test*** after data shaping. Exhibit #12 shows that the data is the correct shape that we would expect.



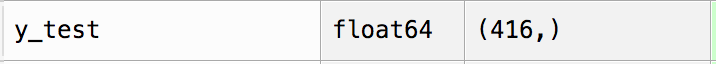
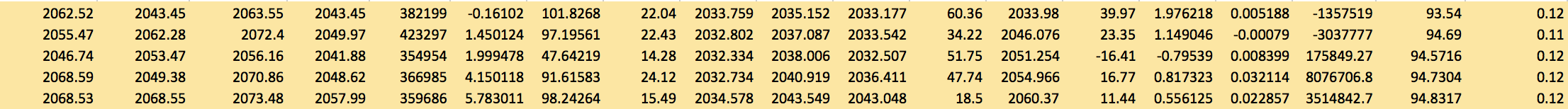
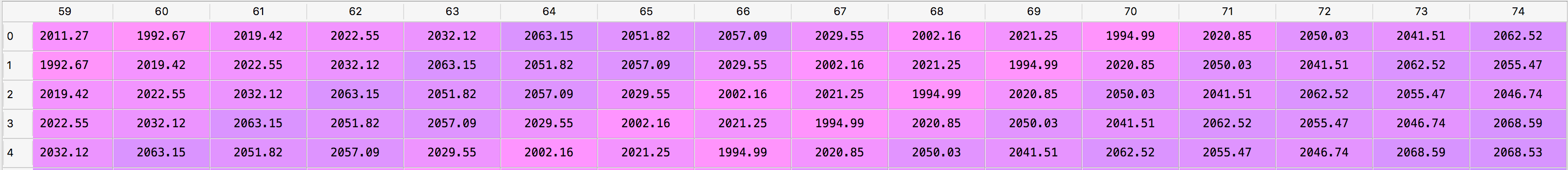


Exhibit #13: Shows that X\_test does not have access to the closing price that it is trying to predict





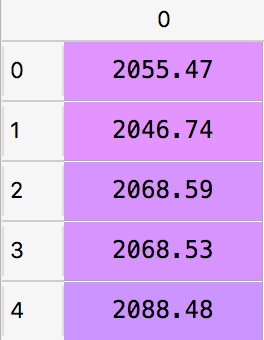
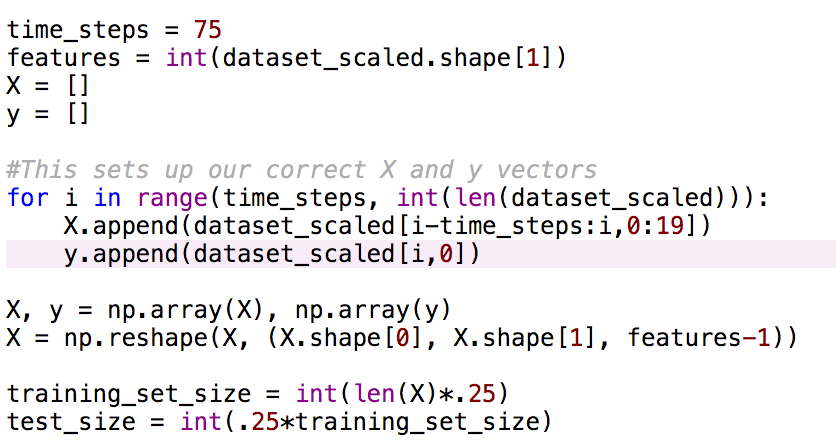


Exhibit #13 shows that at any given test observation, ***X\_test*** contains information on trading days up to ***i-1*** and ***y\_train*** contains the closing price at day ***i***. To help illustrate this, I have included a screenshot of the S&P 500 dataset in its original Excel format. This screenshot corresponds to the trading days found in the screenshots of ***X\_test*** and ***y\_test***. Index (0,74) in ***X\_test*** corresponds to the closing price found in the first column, first row in the highlighted Excel spreadsheet. ***y\_test*** at index (0,0) corresponds to the next day’s closing price that the LSTM is trying predict in ***X\_test*** at index (0,:). Likewise, ***y\_test*** at index (1,0) corresponds to the next day’s closing price that the LSTM is trying to predict in ***X\_test*** at index (1,:). Since ***X\_test*** and ***y\_test*** are shaped like this, the LSTM is unable to ‘cheat’ as it does not have access to information that it would otherwise not have in the real world.

**Sliding Window Training/Testing**

Exhibit #14: Shows the code for data shaping for the sliding window training and testing method



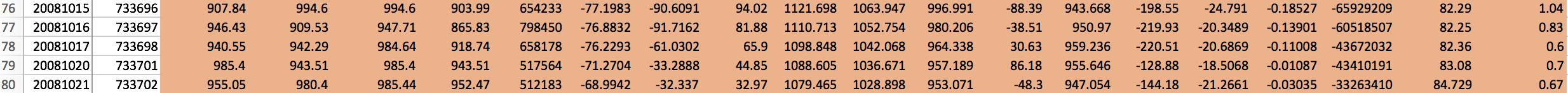
In the above code snippet, I have created two python lists, X and y. This data shaping method presents a cleaner way of shaping the data into the respective X and y tensors. After having shaped X and y, it is very easy to split the data into train and test sets. I use this data shaping method over the method that I had previously discussed in this report because of its more simplistic nature. With this being said, I included the first method in this report as I ran many of my initial experiments using it and I want to give all of my results validity.

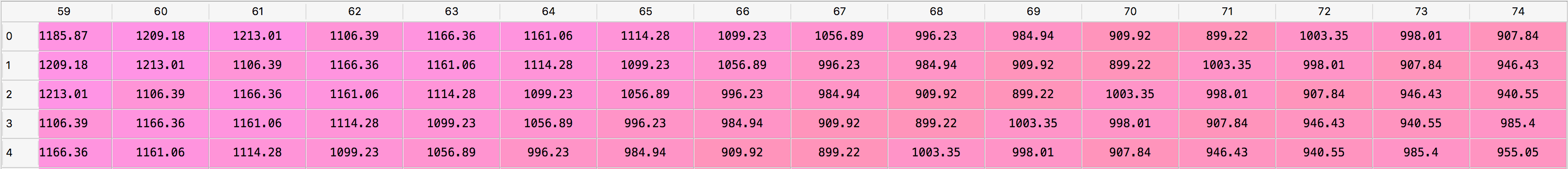
Exhibit #15: Shows that both X and y have the expected shape





Exhibit #16: Illustrates that the LSTM is unable to ‘cheat’





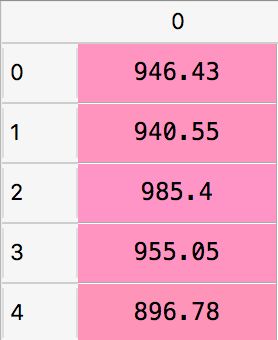


Exhibit 13 shows that when ***X*** and ***y*** are reshaped into their respective tensors, the LSTM is not given information that it would otherwise not have in the real world. The closing price in the orange highlighted spreadsheet found in row one, column one corresponds to the closing price found in X at index (0,74). Y at index (0,0) corresponds to the next day’s closing price found in the second row, first column in the orange highlighted spreadsheet. Because of this shaping, the LSTM will use the information in ***X*** at index (0,:) to predict the next day’s closing price found in the ***y*** tensor’s first index. Likewise, ***X*** tensor at index (1,74) is the second day’s closing price which is then used to predict the next day’s closing price found in the ***y*** tensor at index (1,0)

**Summary**

This report serves as the basis of credibility for all past and future experimental results. Had I been feeding the LSTM inputs that were shaped in such a way that the LSTM would be able to cheat, all of my initial experiments would be invalid. However, the inputs, as described in this report, are shaped in the same way that you would find them shaped in the real world. Because of this, I am fairly certain that my initial results have been achieved through the LSTM learning patterns in financial data rather than cheating.