**Recurrent Neural Network**

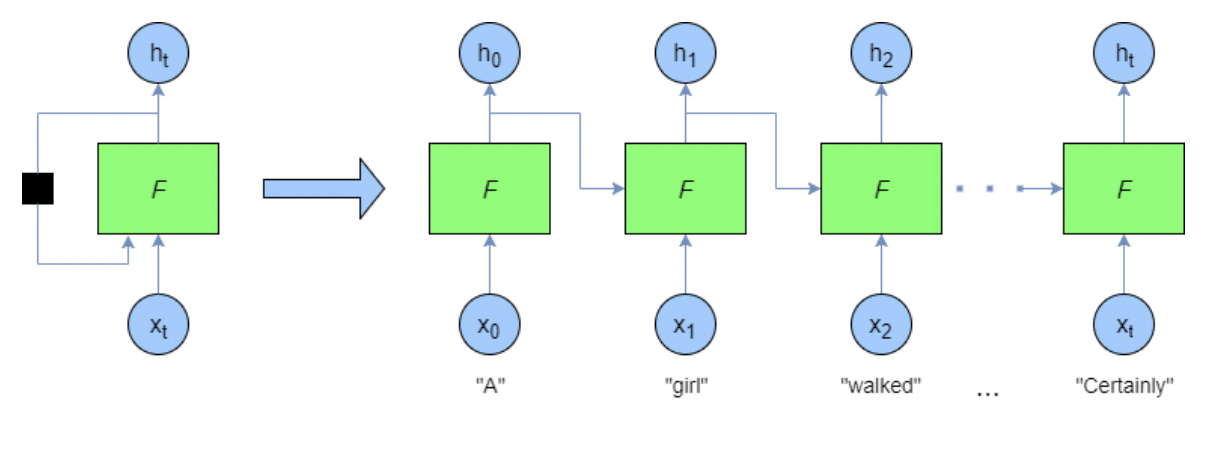
# 

**ABSTRACT**

A Recurrent Neural Network (RNN) is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. It is a form of generalized feed forward network that has internal memory which can be used to process the input data.

In this assignment, a recurrent neural network (RNN) is designed and trained on S&P 500 data. The model is tested by predicting the next one,two,three or four values using a sliding sampling window of width 180 days.

The models are also re-tested on *noise-corrupted* data to observe the performance. The goal of this project is to optimize the RNN to give the best possible results.



**Figure 1:** Sample Recurrent Neural Network

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **CONTENTS** | **PAGE NUMBER** |
| 1 | **Dataset Generation** | 4 |
| 2 | **Network Parameters** | 7 |
| 3 | **RNN Model** | 9 |
| 4 | **Python code for RNN** | 14 |
| 5 | **Performance evaluation for unoptimized and optimized RNN** | 19 |
| 6 | **Optimized RNNs Output graphs for noisy data** | 20 |
| 7 | **Discussion** | 23 |
| 8 | **Appendix** | 25 |

**1. DATASET GENERATION**

The S&P 500 dataset is collected from the [*Yahoo Finance website*](https://finance.yahoo.com/quote/%5EGSPC/history/) between the time frame of January 4, 1960 to December 31, 2020. The table from the website is scraped and stored into a csv file. Shiller P/E ratio is one of the standard metrics used to evaluate whether a market is overvalued, undervalued, or fairly-valued. It can be downloaded from *the* [*Shiller-PE website*](https://www.multpl.com/shiller-pe).

The Shiller P/E ratio data was collected monthly from 1960 to 2020. We use linear interpolation method to convert this monthly data to daily data. The formula for linear interpolation is as follows:

**y = y1 + ((x – x1) / (x2 – x1)) \* (y2 – y1)**where,

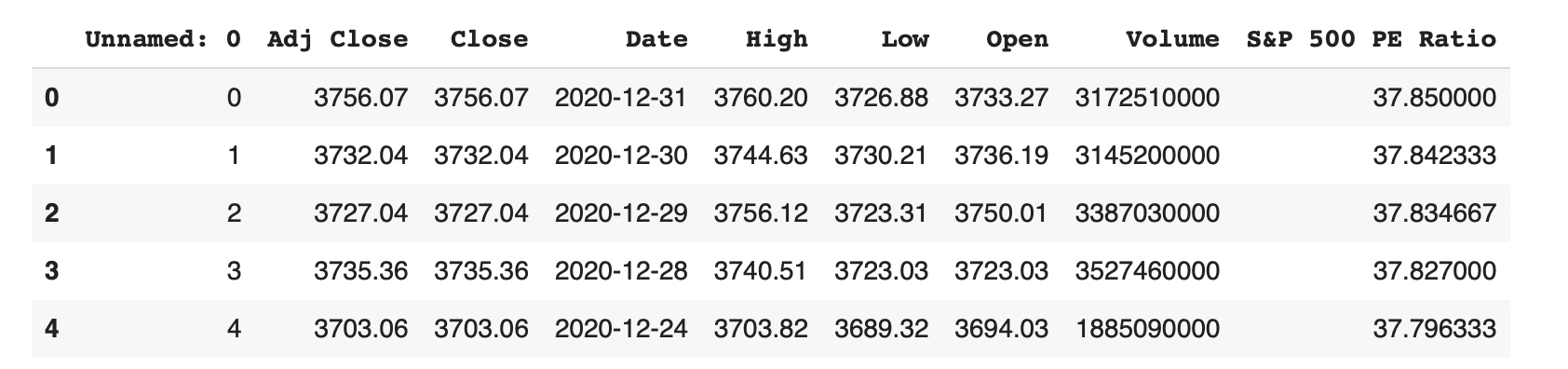
X = known value

y= unknown value

(x1,y1) = coordinates below the known value

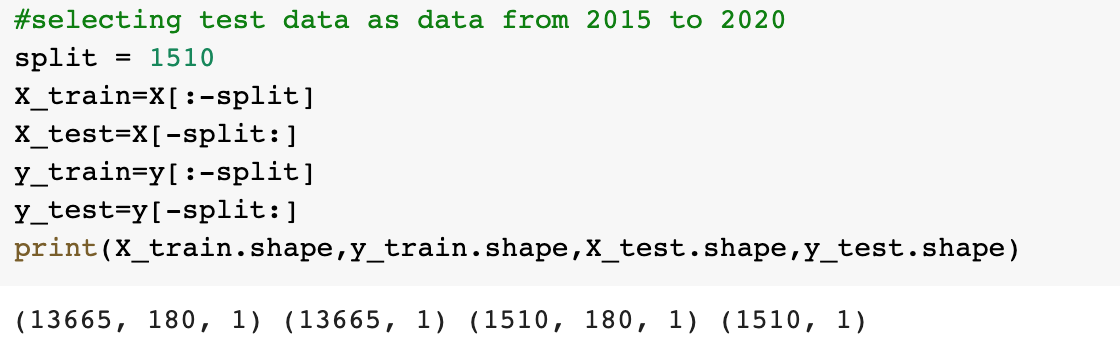
(x2,y2) = coordinates above the known value

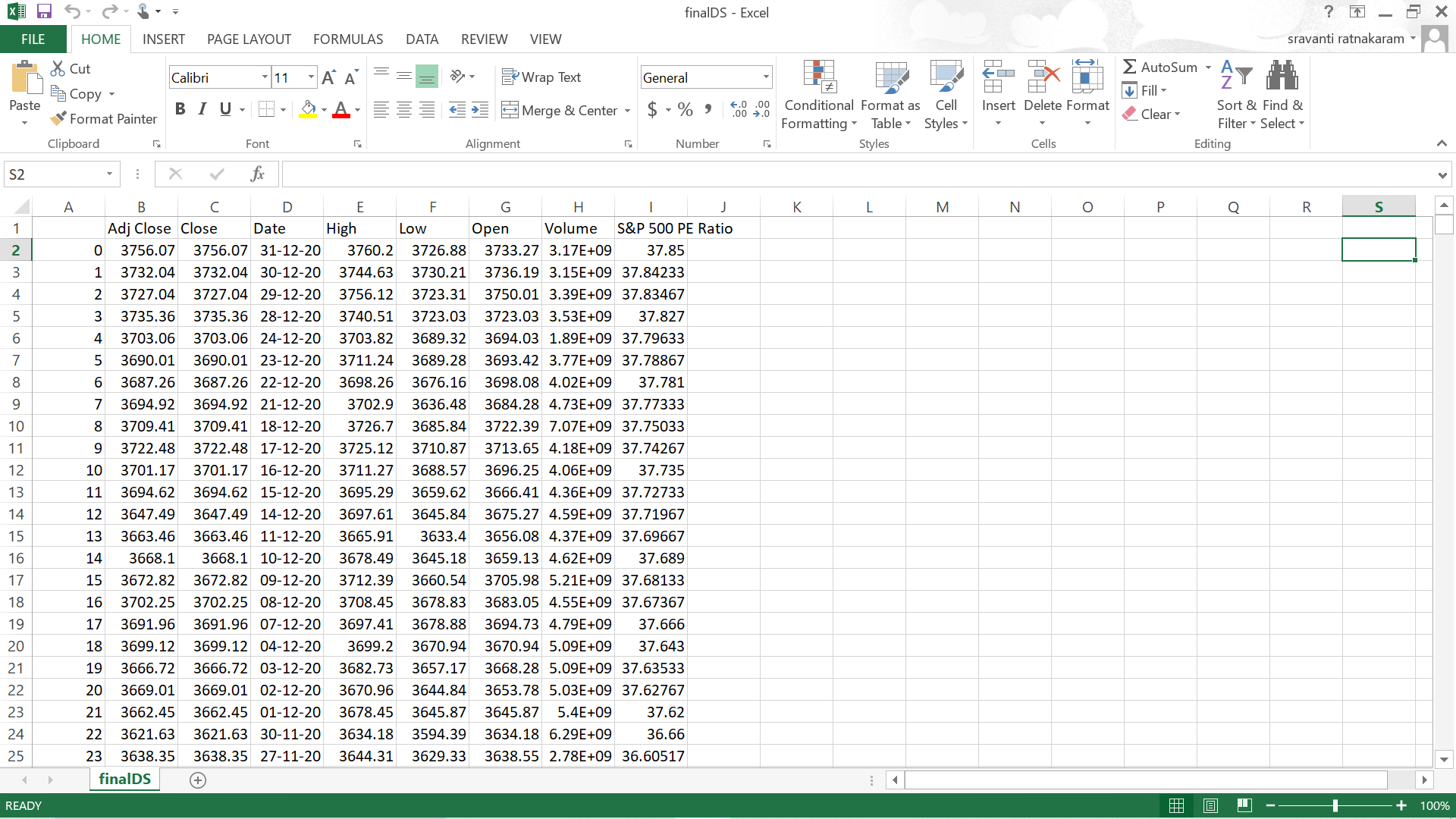
S&P 500 data and Shiller P/E ratio data were then merged and stored in *finalDS.csv.* The data was then divided into training and test sets.



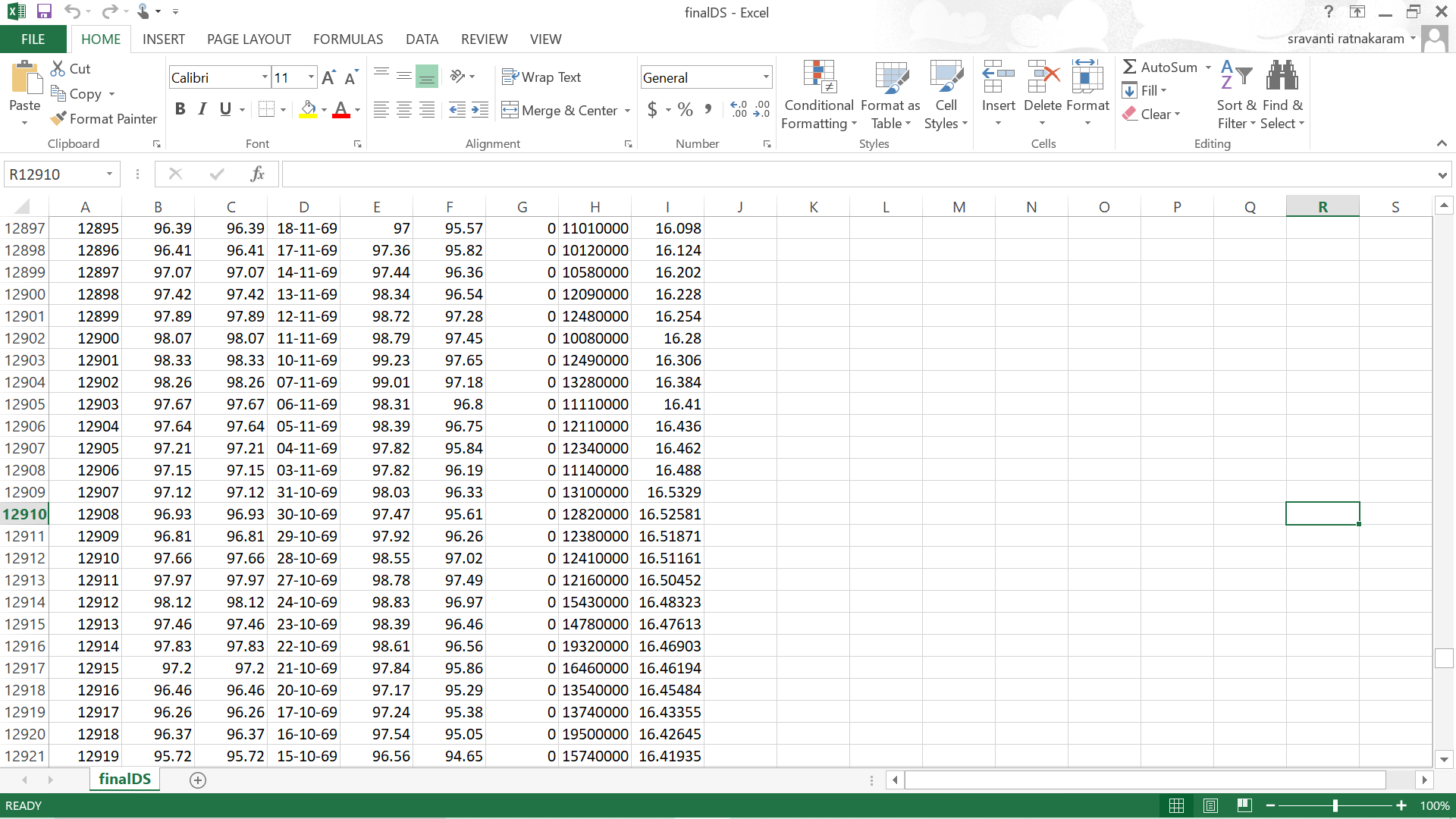
**Figure 2:** snapshot of dataset

The generated dataset is a time series data of length N, defined as CP0,CP1,…,CPN−1 in which CPi is the close price on day ‘i’, *0≤i<N*. The entire dataset is divided into fixed window size of W = 180. Therefore, whenever the window is moved to the right by size W , there is no overlapping between data in all the sliding windows. Then the data values are normalized within range of [0,1]. Then the data values are split into test and train datasets.

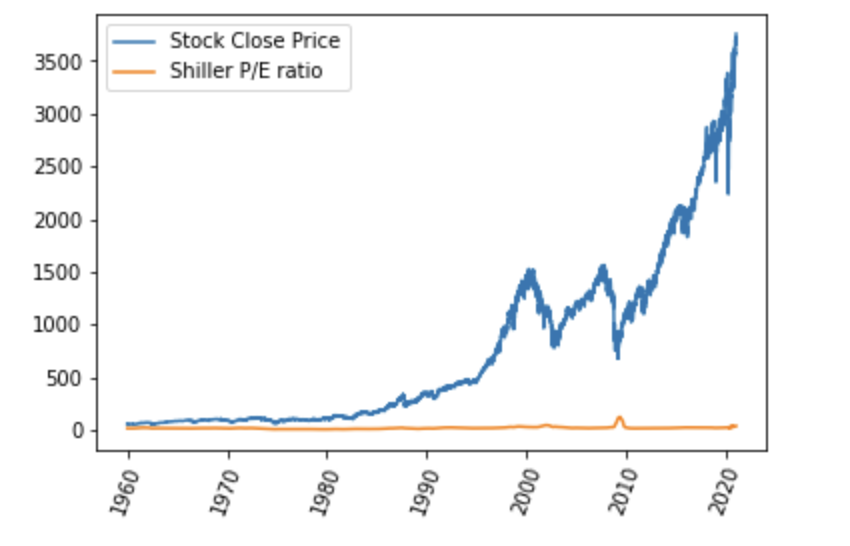


****

**Figure 3:** Portion of the test data



**Figure 4:** Portion of the train data



**Figure 5:** Graphical representation of data

**2. NETWORK PARAMETERS**

1. **Input :** generated dataset

2. **Predicted Output feature:** predicted ‘Close’ price

3. **Activation function:** sigmoid, tanh

4. **Epochs**:1,10,100.(tabulated below)

5. **Optimizer:** Adam

6. **Evaluation metrics:** prediction error, accuracy

7. **Number of timesteps:** 180

8. **Loss metrics:** Mean squared error

The model is trained on the entire data set from 1960 to 2014 and tested on data from 2015 to 2020. The RNN built is a regression model that predicts the future Stock Close Price. The resulting output of the predicted price is then compared to the original price values to evaluate the model.

|  |  |
| --- | --- |
| **Number of iterations**  **(epochs)** | **Accuracy** |
| 1 | 97.64 |
| 10 | 90.68 |
| 100 | 95.14 |

**Table 1**: Unoptimized RNN accuracy tabulations

|  |  |
| --- | --- |
| **Number of iterations**  **(epochs)** | **Accuracy** |
| 1 | 90.49 |
| 10 | 95.54 |
| 100 | 97.72 |

**Table 2**: Optimized RNN accuracy tabulations

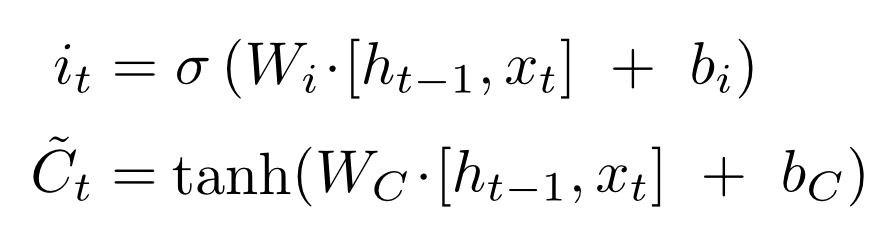
The number of **epochs** were chosen in a manner to improve the performance of the model. The accuracy of the model for the optimized and the unoptimized models were calculated while changing the number of epochs. It was observed that when the **number of epochs was 1**, the accuracy dropped from **97.64 to 90.49** when an optimizer was used. It has also been observed that as the number of epochs have increased drastically, it led to **over-fitting** of the model which led to a great decrease in the accuracy of the model.

Hence, the number of epochs was chosen in such a way that the loss was minimum. In the case of **10 epochs**, the optimized RNN produced better results. As the accuracy increased from **90.68 to 95.14.** In the case of **100 epochs**, The accuracy for the unoptimized RNN is **95.14** and for optimized it is **97.72** which is a suitable accuracy. So, **100 epochs** was **chosen** as it provided the best accuracy in the case of both optimized and unoptimized.

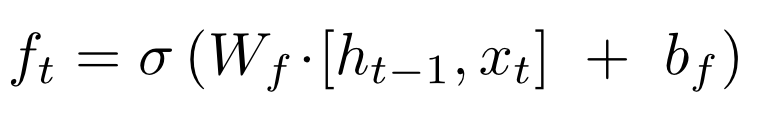
**3. RNN Model**

The architecture employed to deploy this model is the **LSTM model**. ***LSTM*** stands for Long Short Term Memory. Long Short-Term Memory (LSTM) networks are a ***modified version*** of recurrent neural networks, which makes it easier to remember past data in memory. The ***vanishing gradient*** problem of RNN is resolved here. LSTM is well-suited to classify, process and predict time series given time lags of unknown duration. It trains the model by using back-propagation. In an LSTM network,**three gates** are present:

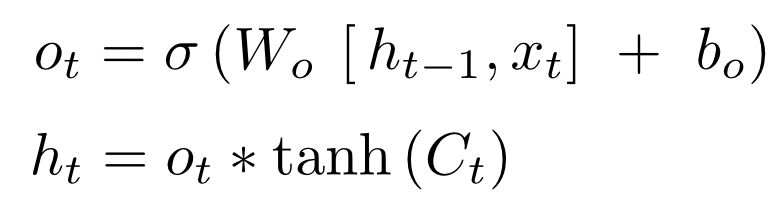
**i. Input gate** — discover which value from input should be used to modify the memory. **Sigmoid** function decides which values to let through **0,1.** and **tanh** function gives weightage to the values which are passed deciding their level of importance ranging from**-1** to **1**

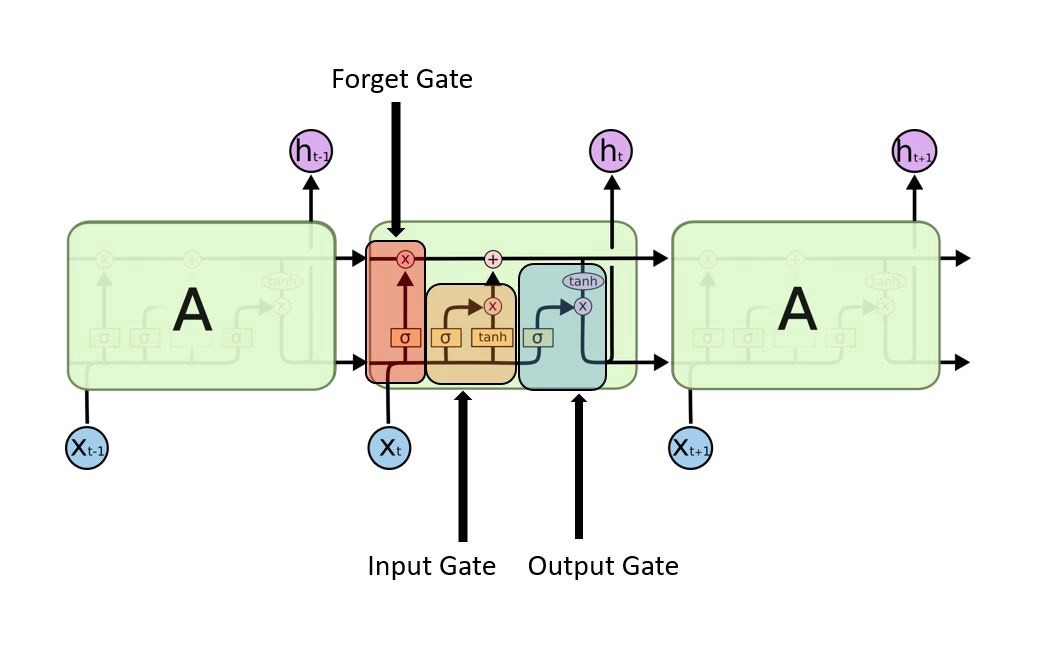
****

**ii. Forget gate** — discover what details to be discarded from the block. It is decided by the **sigmoid function.** it looks at the previous state(**ht-1**) and the content input(**Xt**) and outputs a number between **0(***omit this*)and **1(***keep this***)**for each number in the cell state **Ct−1**.

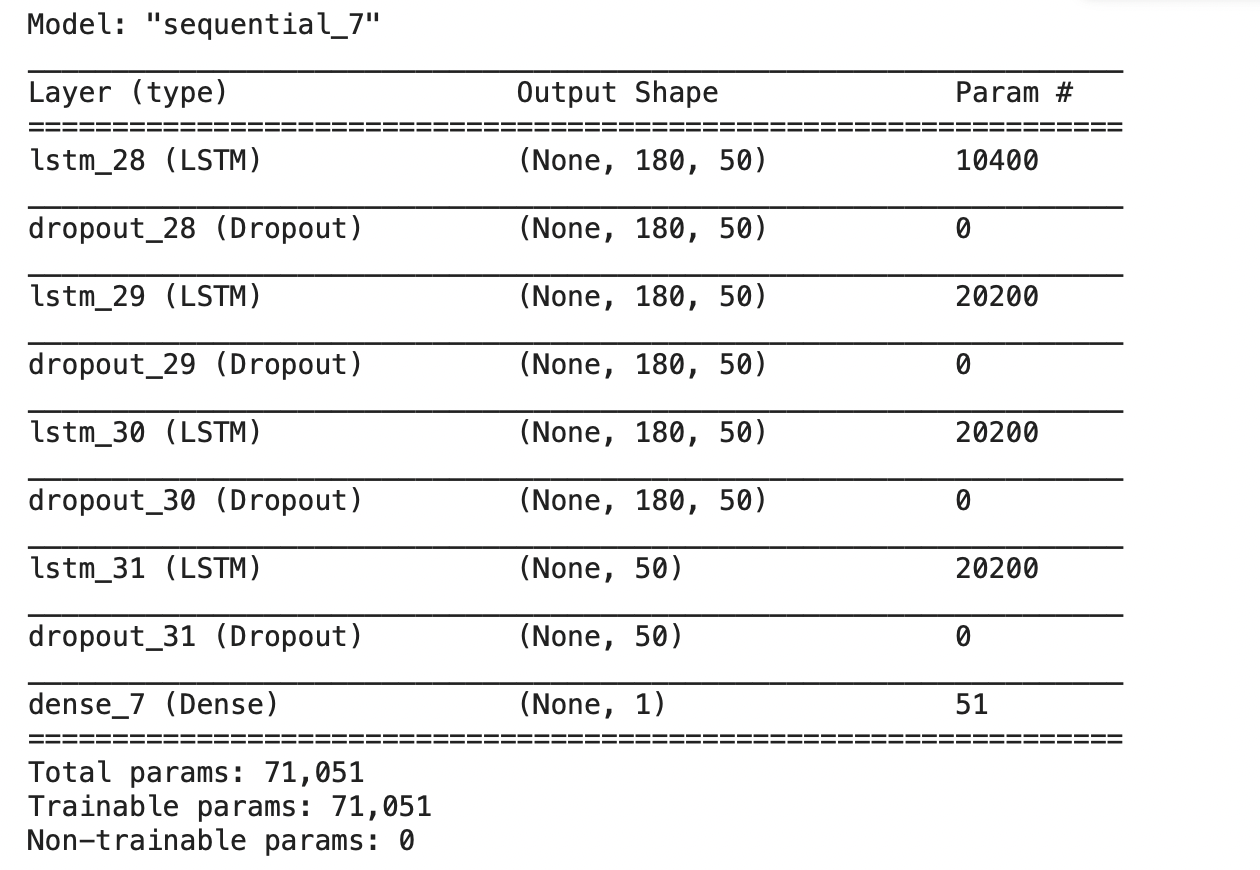
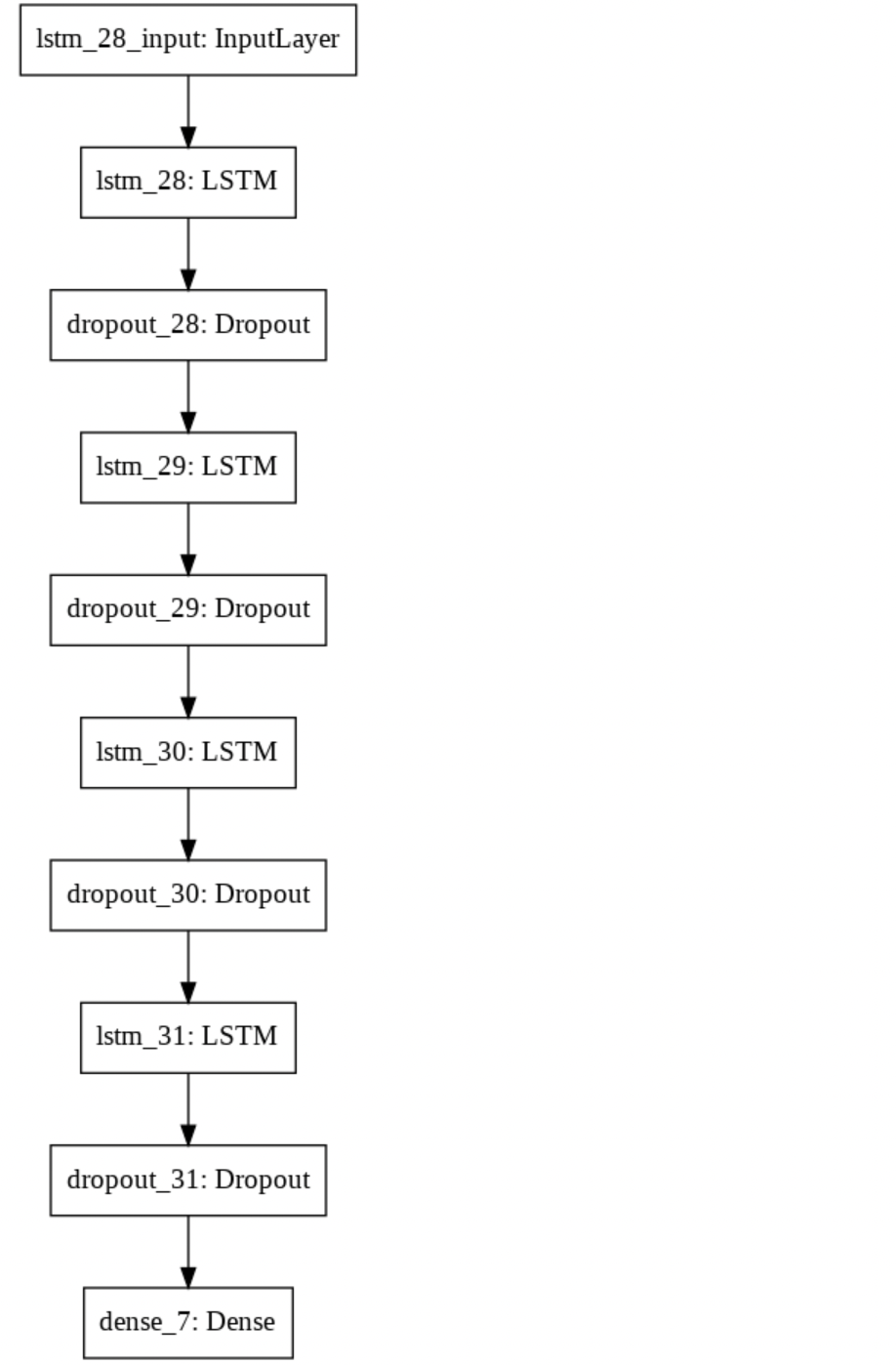


**iii. Output gate** — the input and the memory of the block is used to decide the output. **Sigmoid** function decides which values to let through **0,1.** and **tanh** function gives weightage to the values which are passed deciding their level of importance ranging from**-1** to **1** and multiplied with output of **Sigmoid.**

****

****

**Figure 6:** Sample LSTM architecture



**Figure 7:**RNN architecture of the optimized model

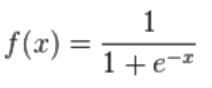
**3.1 LOSS METRIC**

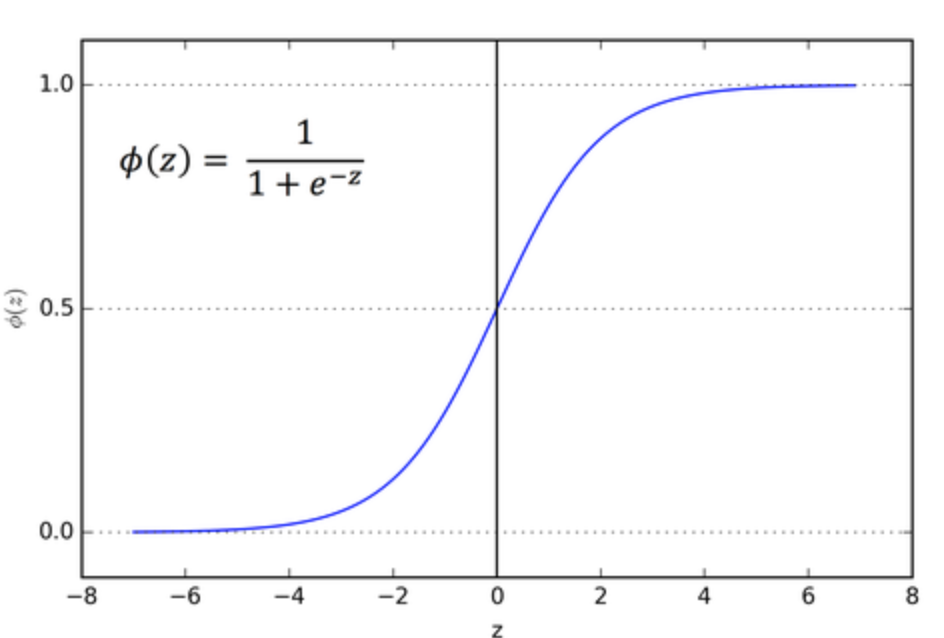
The model uses ***mean squared error*** as a loss metric. Mean squared error is used to tell how close a regression line is to a set of points. It also gives more weight to larger differences.

**3.2 ACTIVATION FUNCTION**

**Sigmoid Function:**

Since every neuron must predict a value within the range of [0,1], we chose sigmoid activation function. The sigmoid function can be represented as:

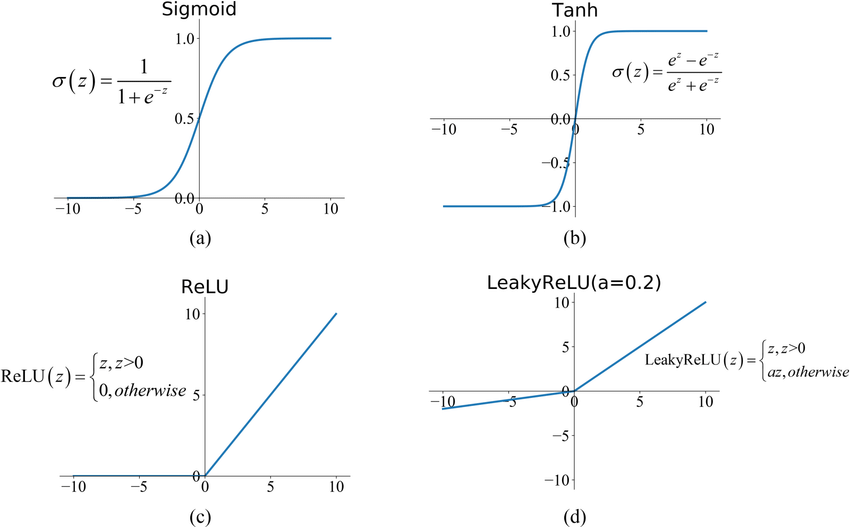




**Figure 8:** Sigmoid Activation function

**Tanh Function:**

Tanh is also like logistic sigmoid but better. The range of the tanh function is from (-1 to 1). tanh is also sigmoidal (s - shaped).



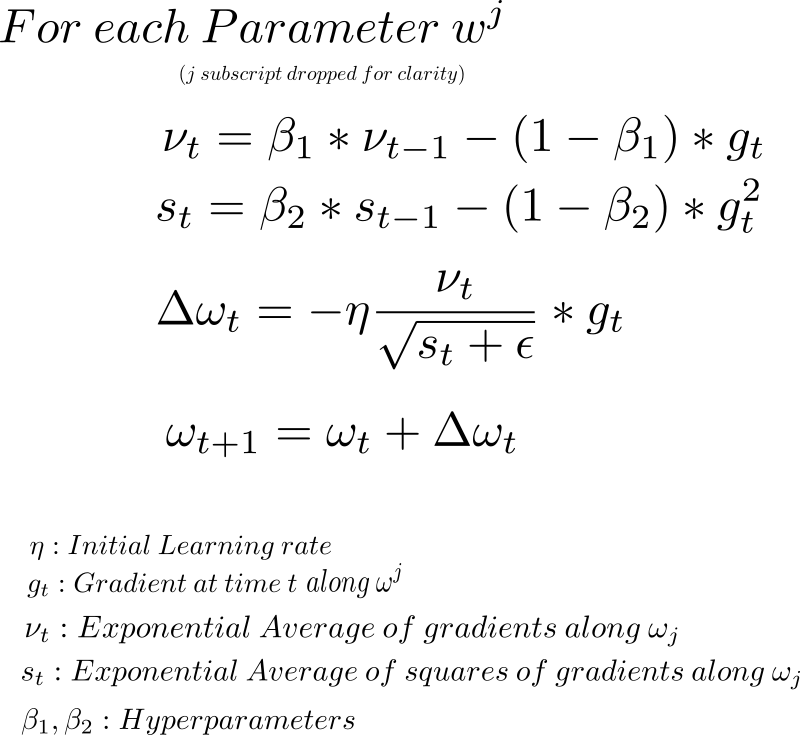
**Figure 9:** Tanh Activation function

**3.3 OPTIMIZER**

The model uses ***Adam*** optimizer.

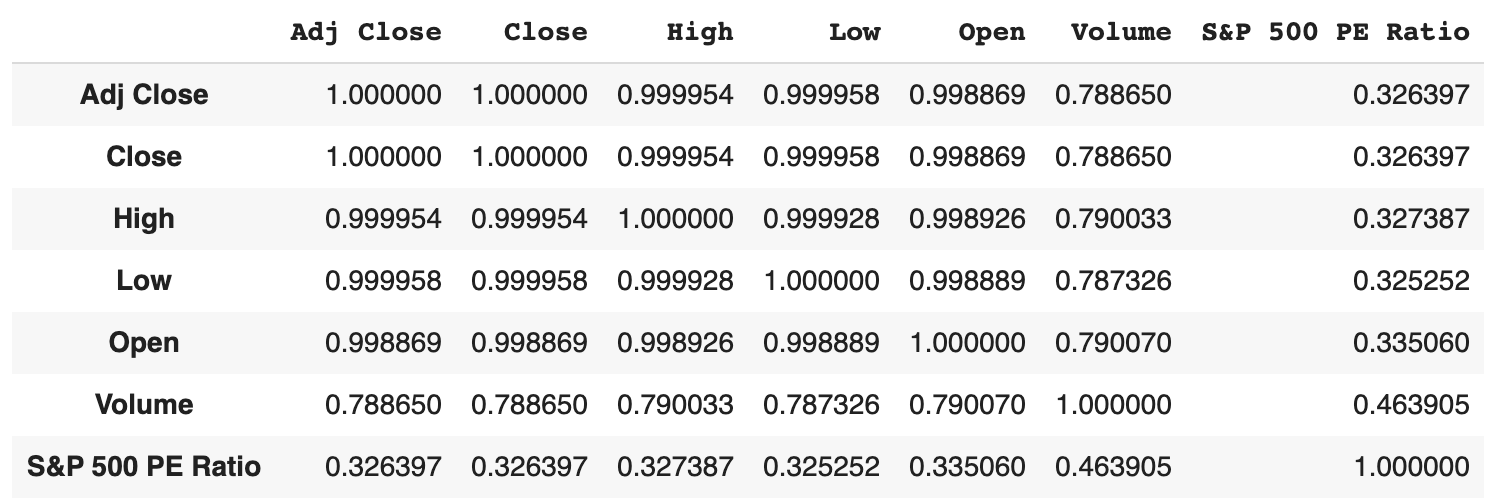
* **Adam optimizer:**

**Adam** can be looked at as a combination of RMSprop and Stochastic Gradient Descent with momentum. It uses the squared gradients to scale the learning rate like RMSprop and it takes advantage of momentum by using the moving average of the gradient.



**3.4 SELECTING INPUT FEATURES**

Initially, the model was trained on High, Low, Open, Close, S&P 500 PE Ratio, Volume, Adj Close. The accuracy of the model turned out to be around 53%. So, correlation between the features were calculated and can be seen below.

****

**Figure 10:** Correlation table

We observe that the Shiller P/E ratio is very weakly correlated to the close price. Hence, we remove it from our feature list.

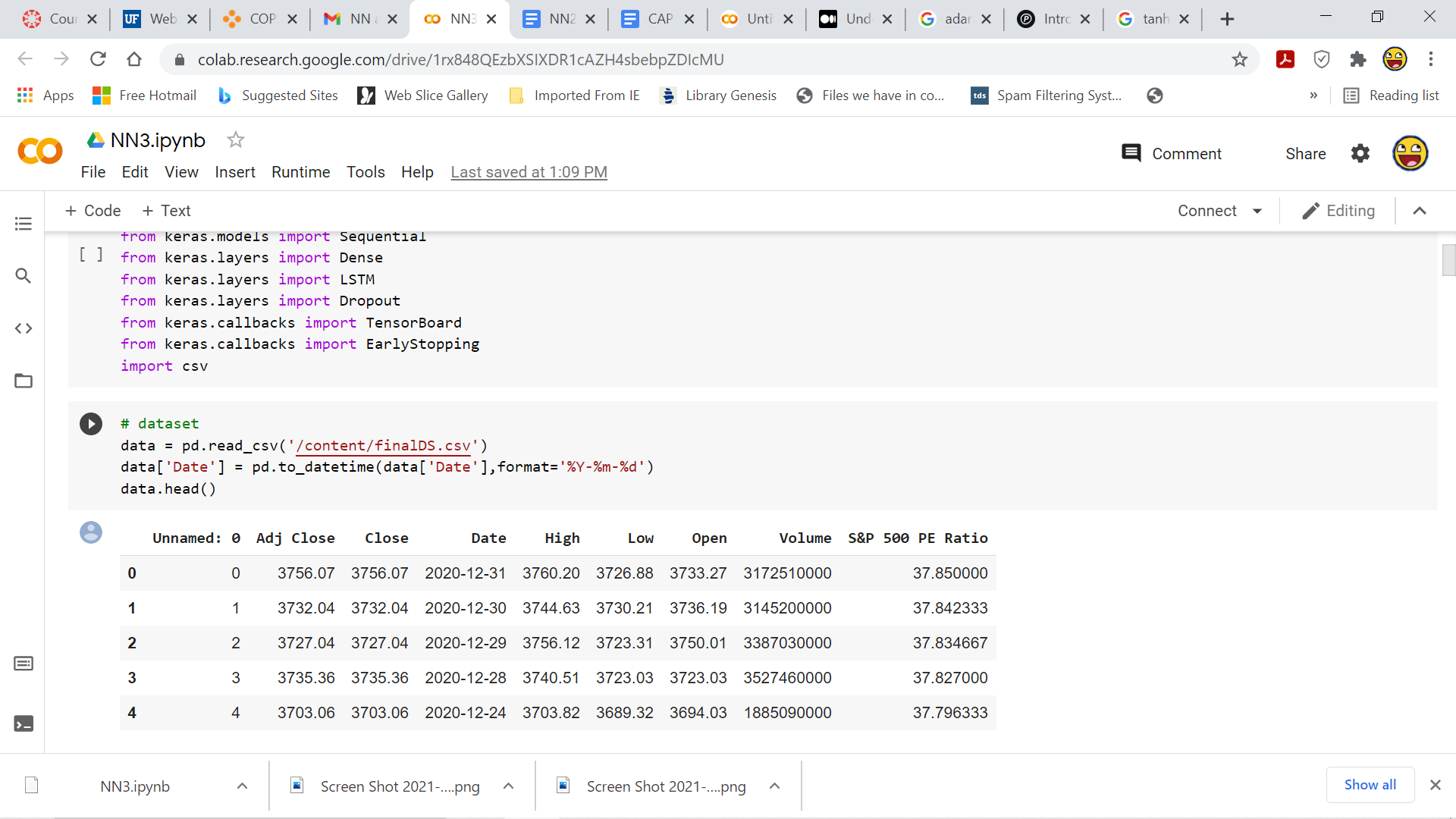
We can see that Adj Close, High, Low and Open are strongly correlated to the close price. Hence, we add another column average where we calculate the mean of High and Low prices. Now, we trained the RNN on Adj Close, Average and Open. The accuracy of the model increased by 4%.

We also trained the RNN with Adj Close price as the only. We observe that the univariate model behaved far better with an accuracy of 97.7%. Hence, we chose this univariate model as our Optimized RNN. Please refer to the appendix for more detailed explanation.

**4. PYTHON CODE FOR RNN**

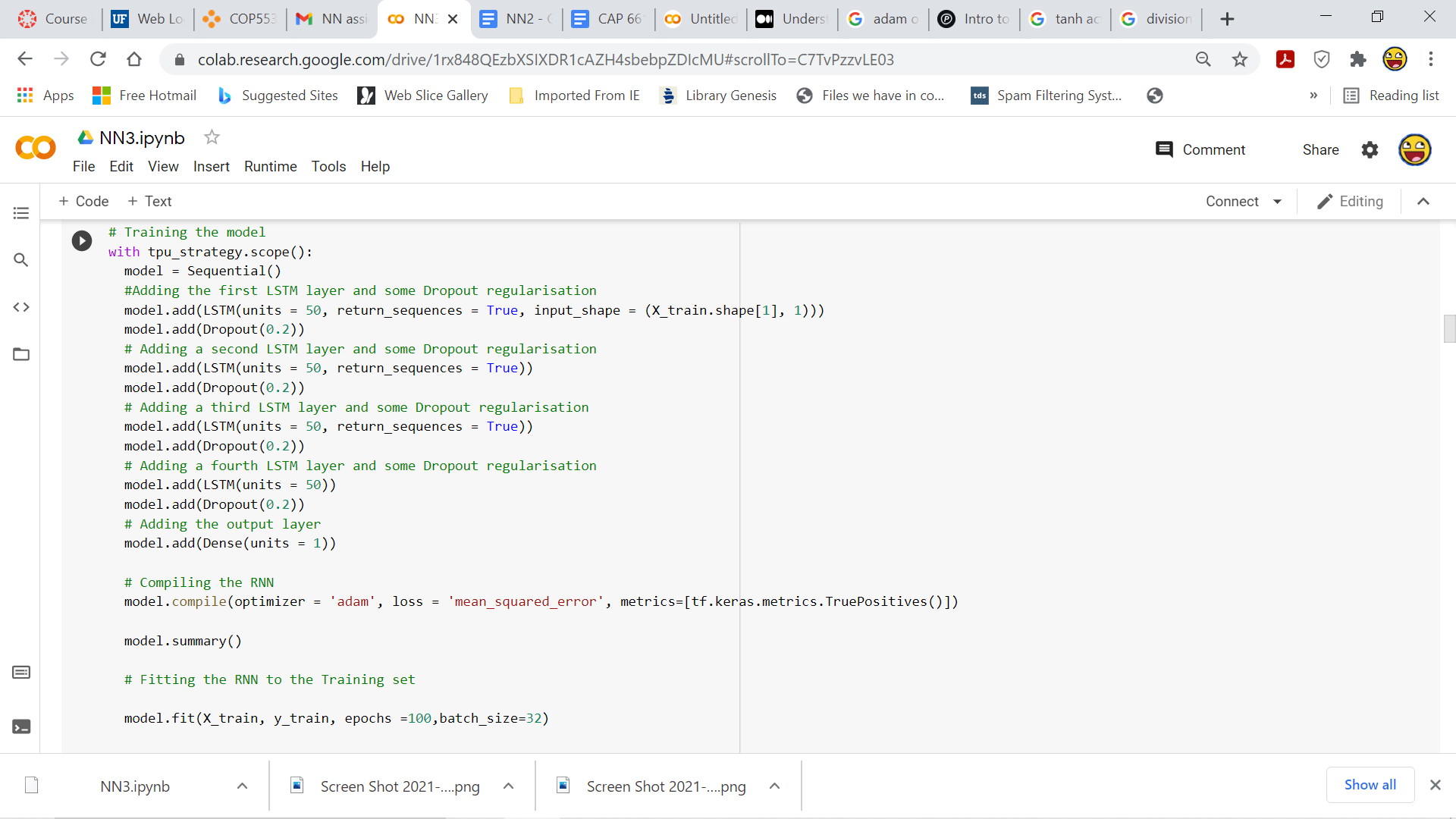
***Dataset Generation***

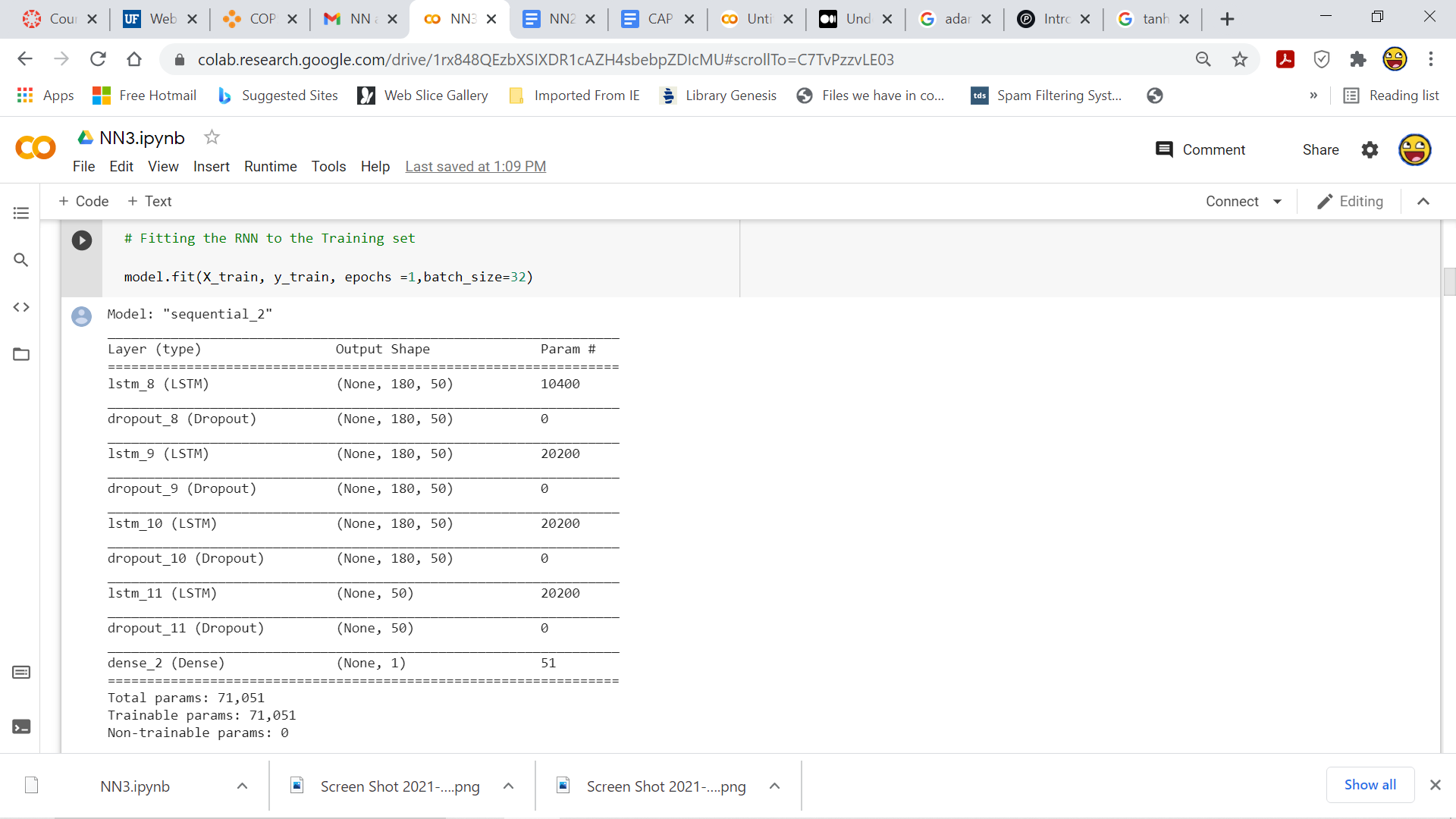
The following code snippet depicts the conversion of the data obtained from the yahoo finance website into a dataframe using Pandas library.

.

***Model Initialization and training***

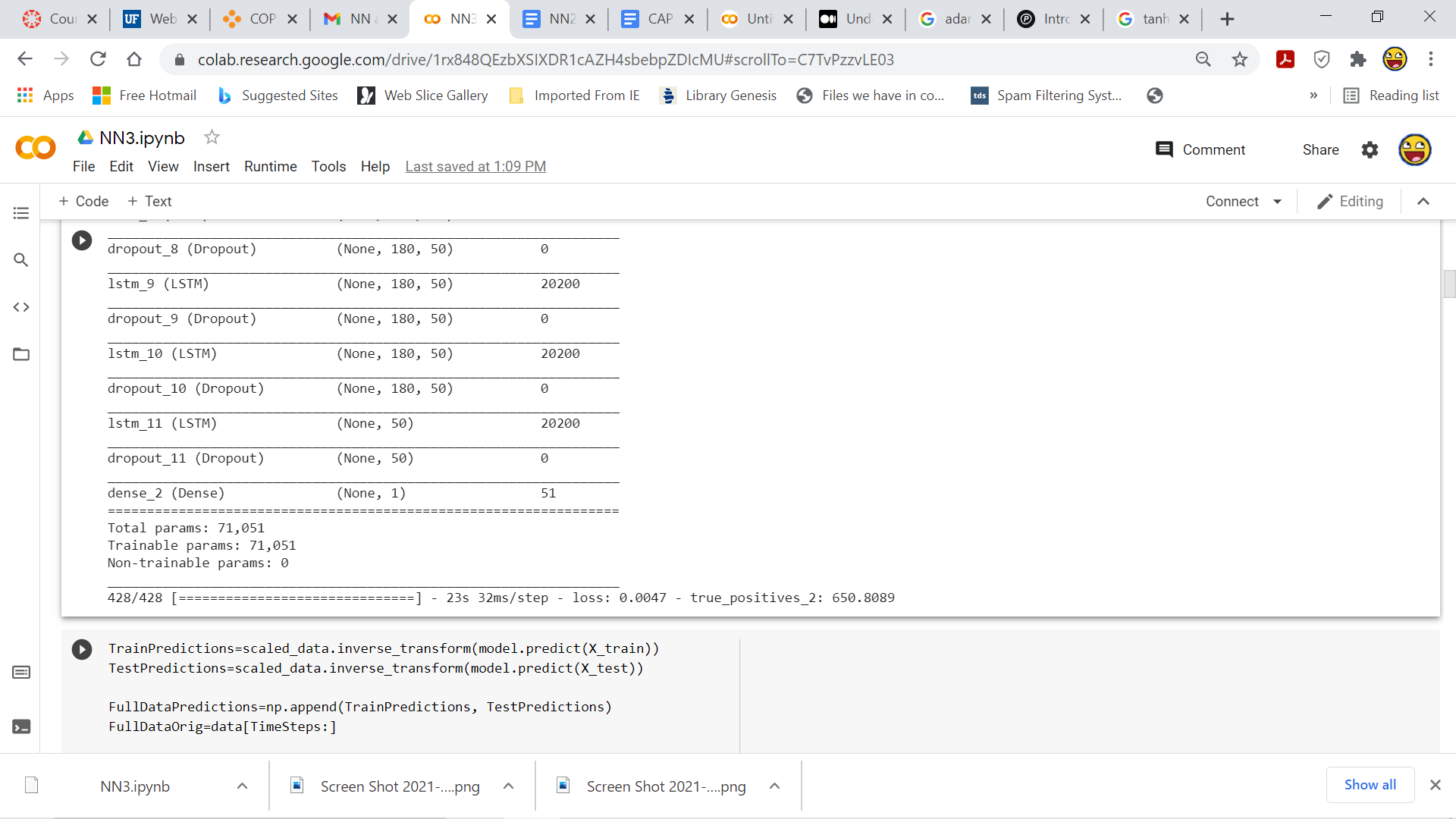
In the code snippet below the model has been initialized with an LSTM layer followed by three other LSTM layers which also employ dropout to improve efficiency. The last layer is followed by a Dense layer. The sliding window is of size 180. The loss used is mean squared error loss.Adam optimizer has also been used.

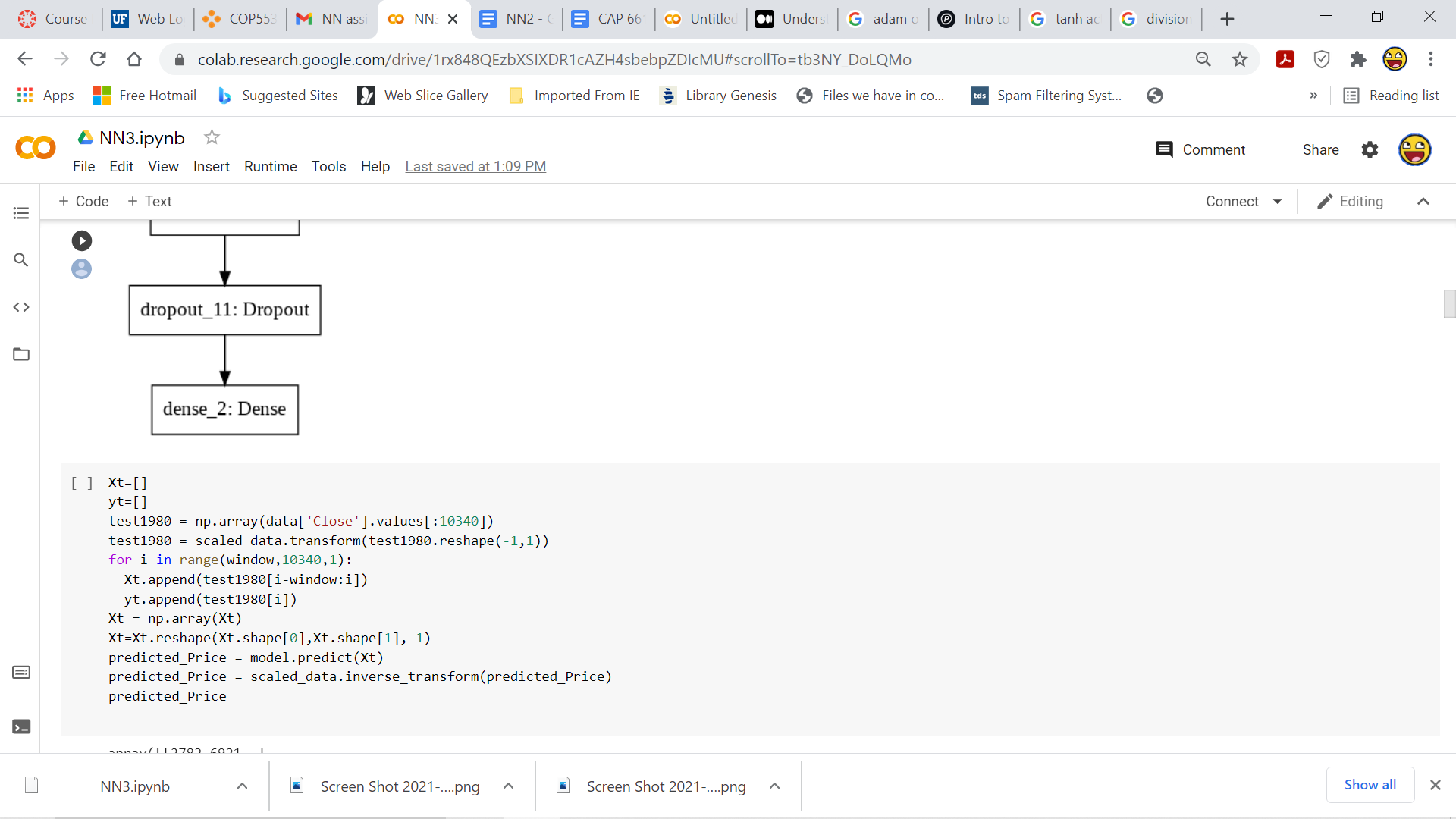


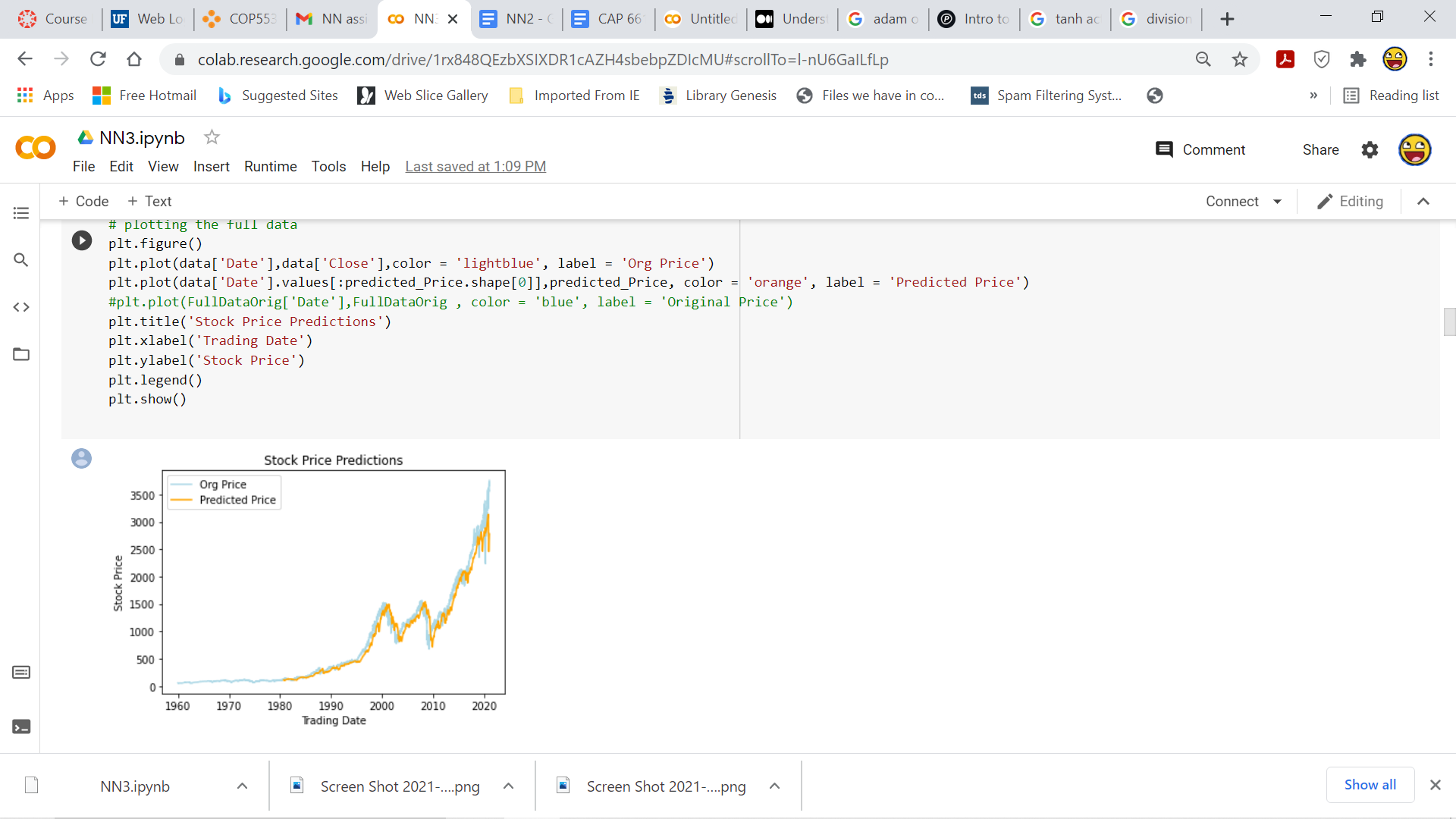
******

***Generating predictions***

The above trained model has been used to generate predictions which are then used to evaluate the performance of the model.

******

******

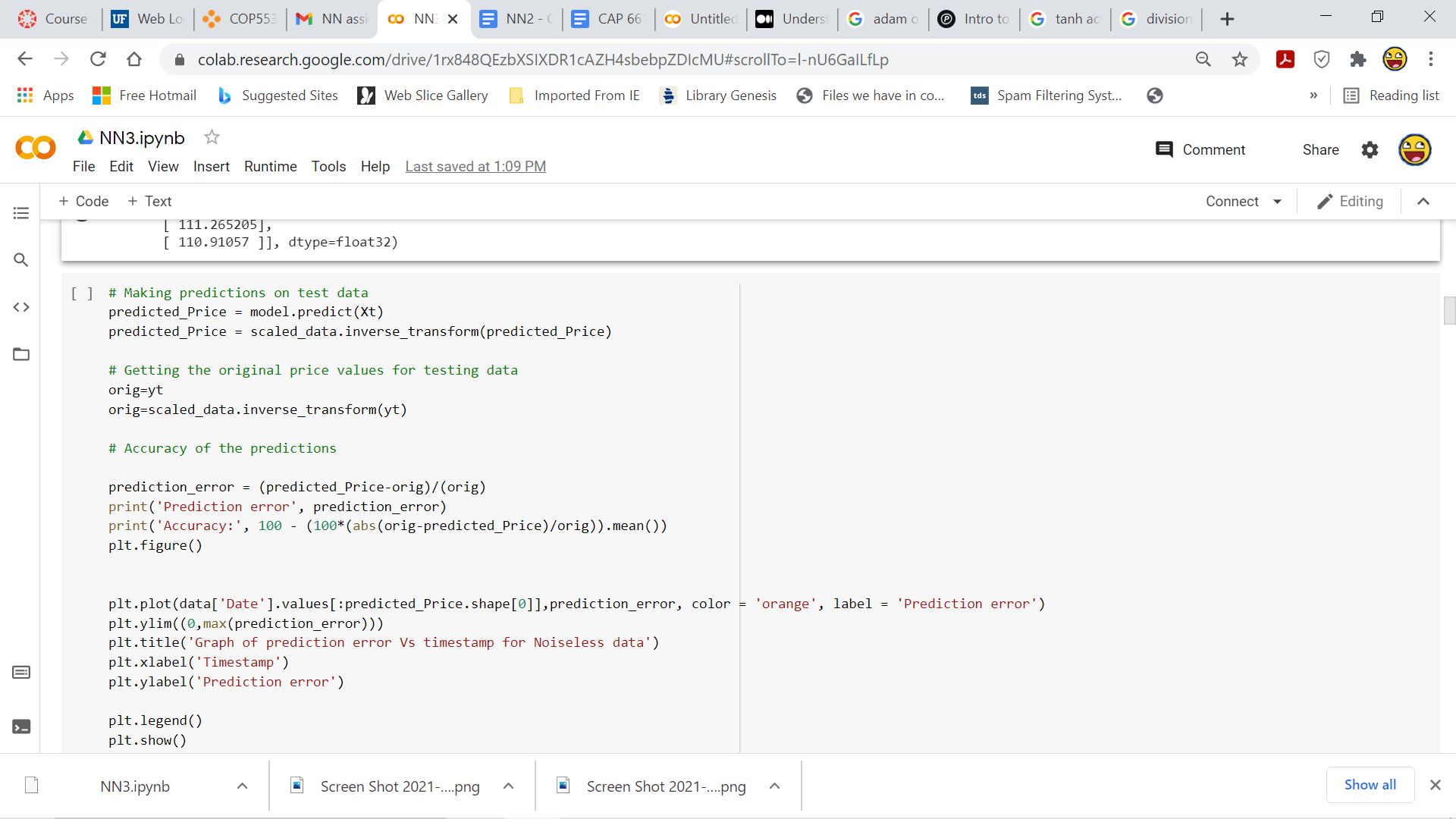
******

***Calculation of performance metrics***

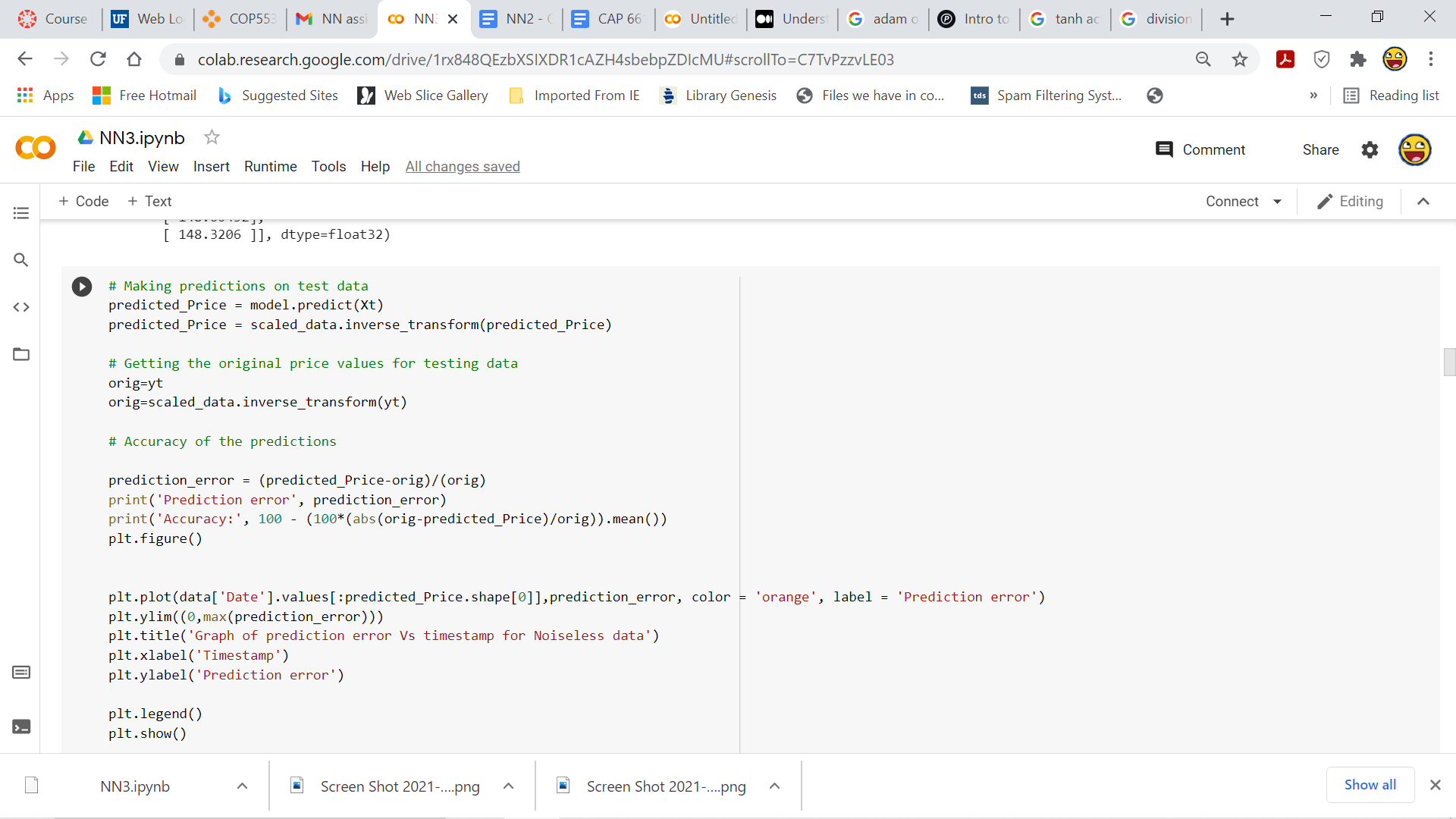
The performance metrics used are prediction error and accuracy. The prediction error is calculated as :

***prediction error= (predicted price - original price)/original price.***

The prediction error that is calculated is plotted as a graph against the timestamps which is displayed in **Figure 11**.

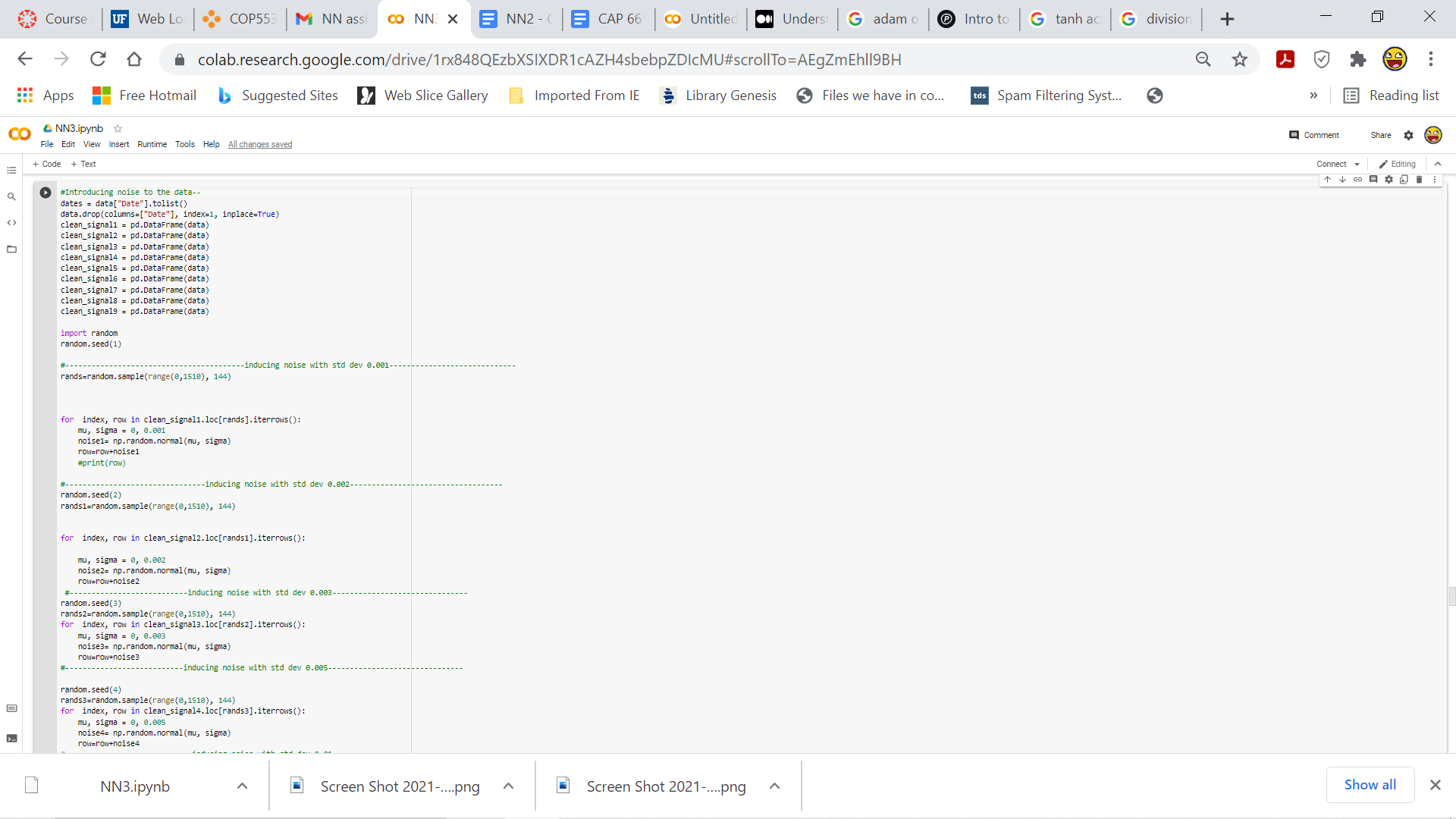


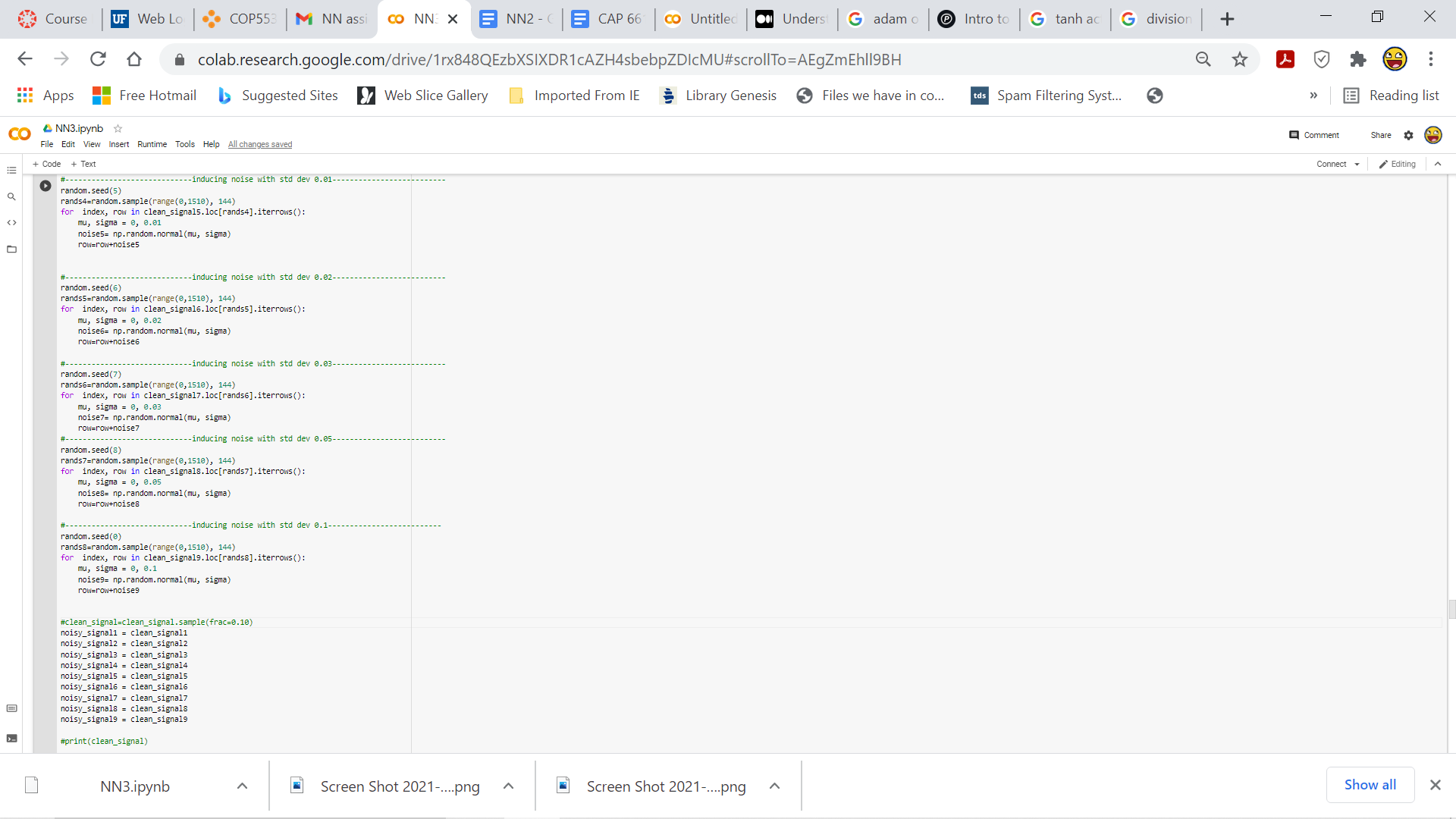
***Plotting the Prediction error graph***

****

***Introducing noise to input data***

The input data are perturbed by adding additive gaussian noise. The following code snippets implements that. The prediction error for each standard deviation is calculated and has been tabulated in **Figure 15**

******



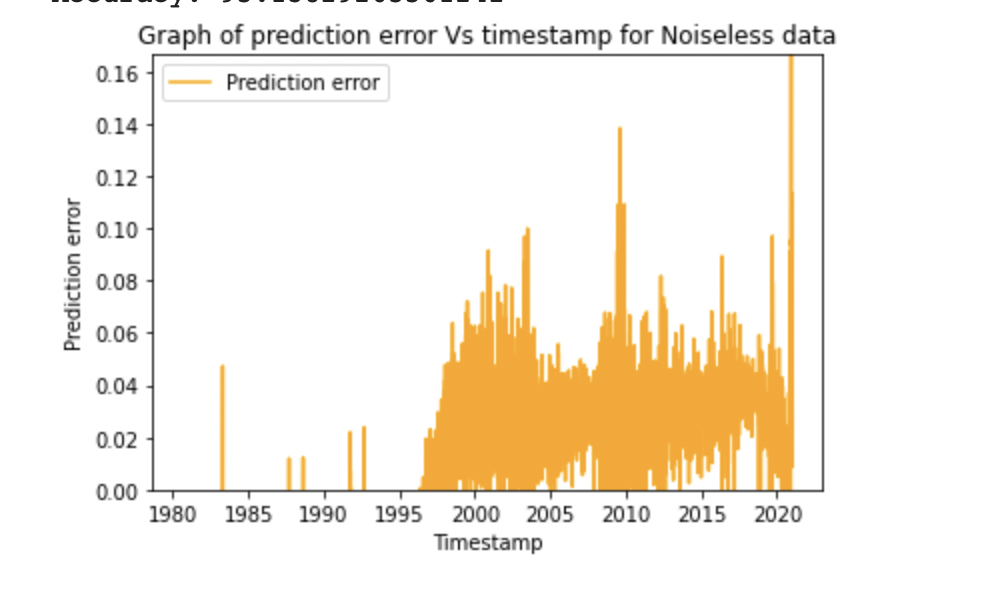
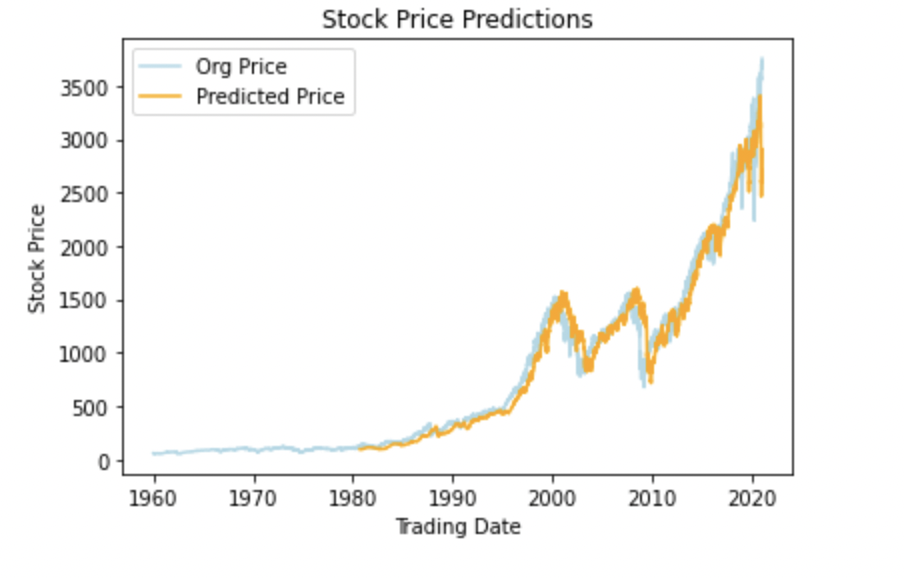
***Tabulating the prediction error***

The prediction error has been tabulated in an excel file. As the table has a large dimension, only a portion of the **table** has been displayed in **Figure** **15**

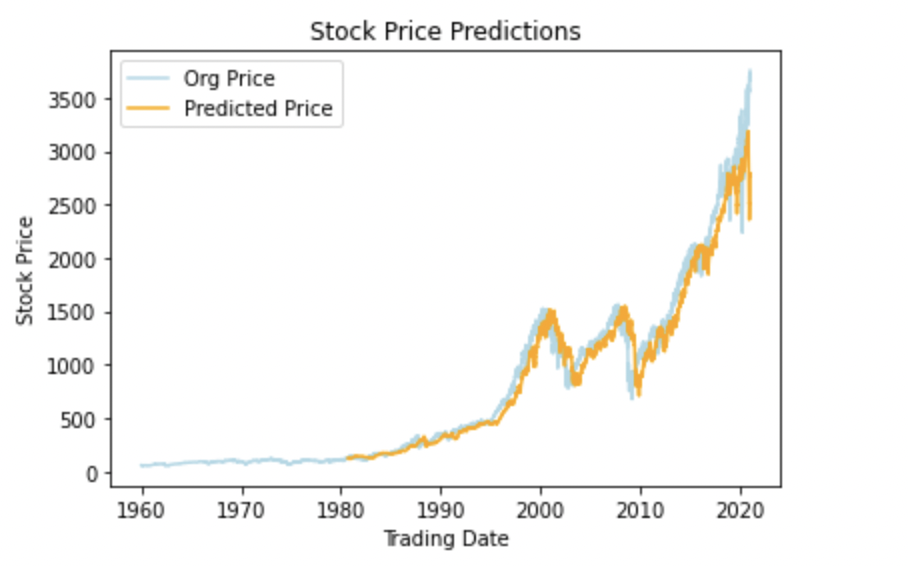
**5. PERFORMANCE EVALUATION FOR UNOPTIMIZED AND OPTIMIZED RNN**

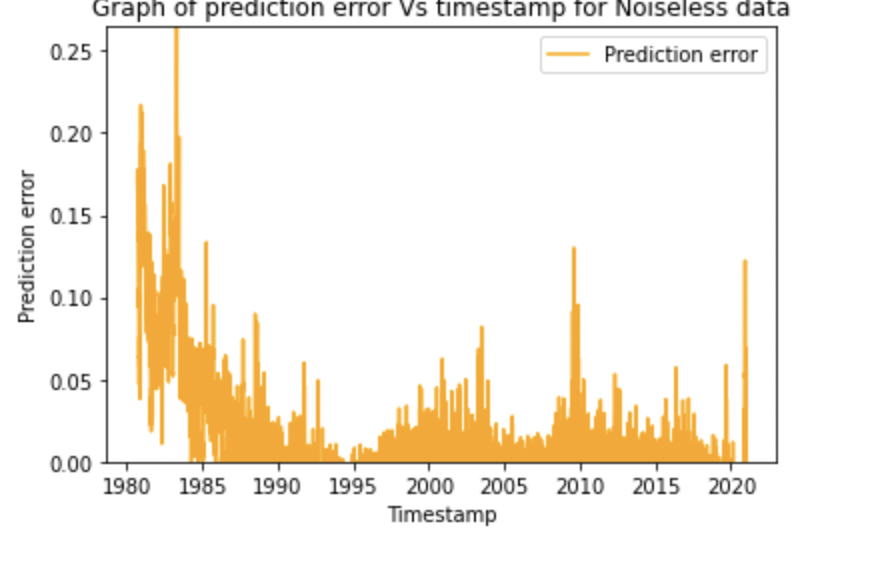
Due to the usage of Adam optimizer and suitable number of epochs which is **100** in our model, the prediction error is low compared to the unoptimized model. In the case of **100 epochs**, the accuracy for the unoptimized RNN is **95.14** and for optimized it is **97.72** .

**Unoptimized RNNs Output Graphs for noiseless data**



**Figure 11:**Prediction error vs timestamp **Figure 12:** stock price predictions

**Optimized RNNs Output Graphs for noiseless data**



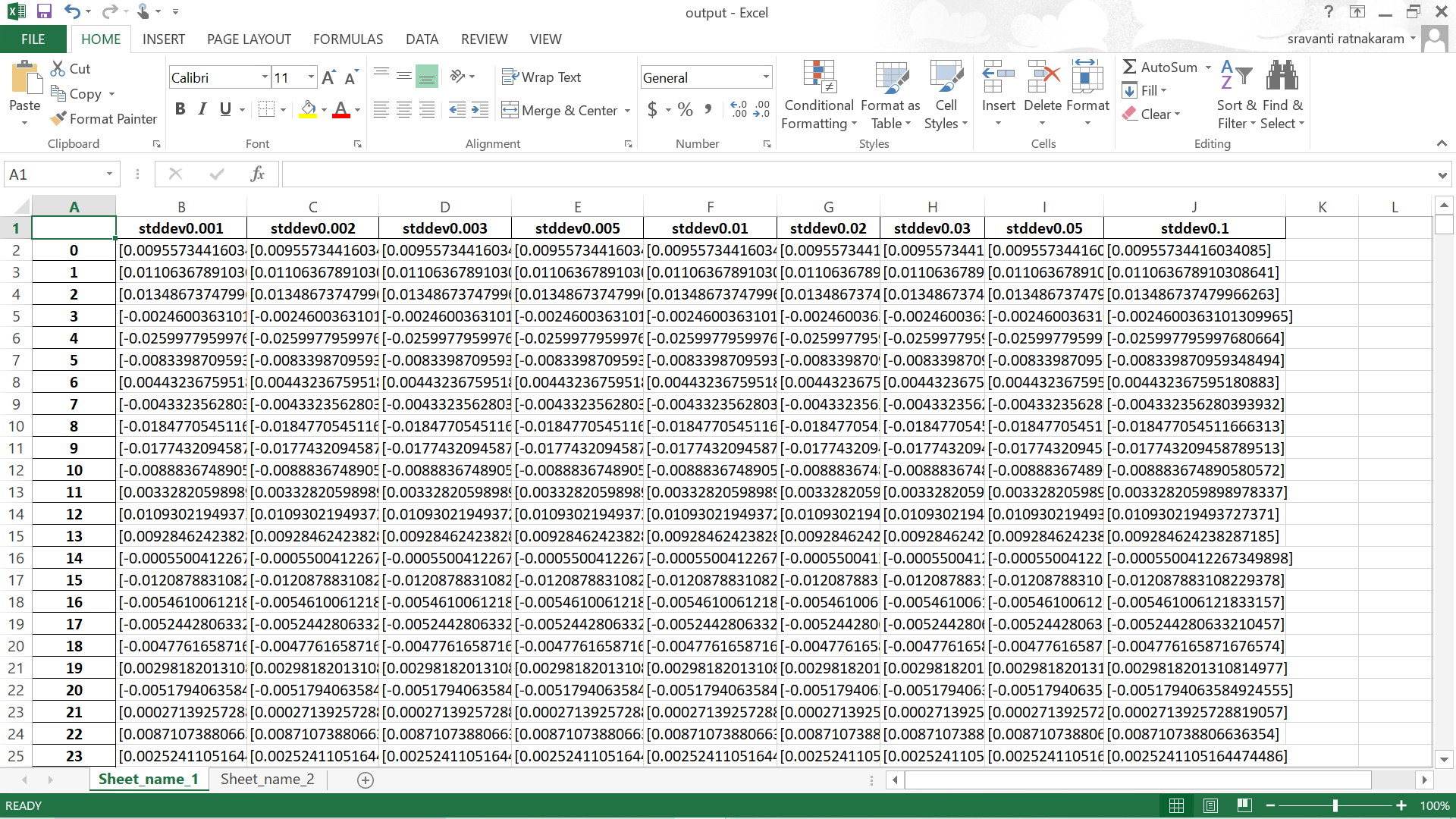
**Figure 13:**Prediction error vs timestamp **Figure 14:** Stock price predictions

**6. Optimized RNNs Output Graphs for noisy data**

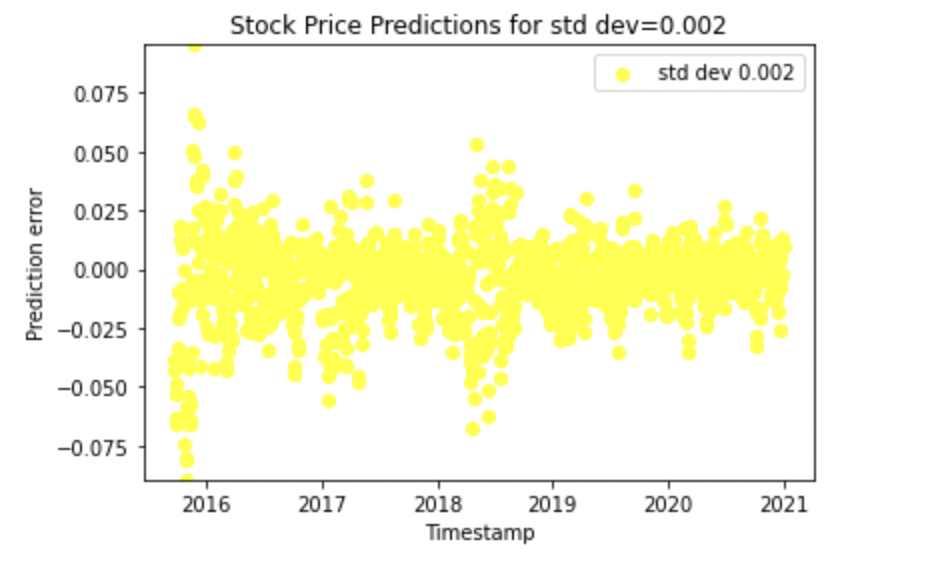
The images were then perturbed with gaussian noise and then the images were then tested. Following which **prediction error** and **timestamp** values for each standard deviation were calculated and tabulated. 9 graphs were also plotted with **timestamp** on the **abscissa** and the **prediction error** on the **ordinate** axis for each **error level.** As it was asked to plot prediction error for each error level in one graph.

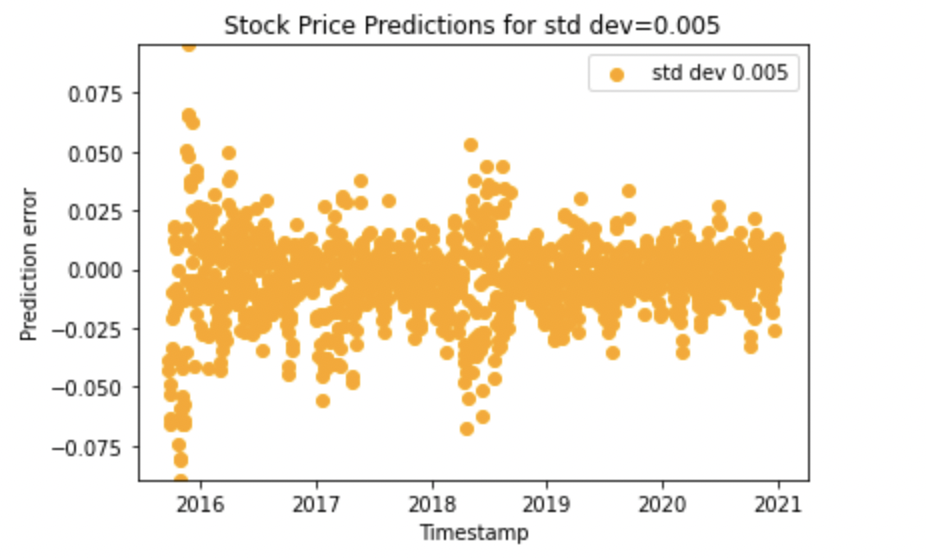
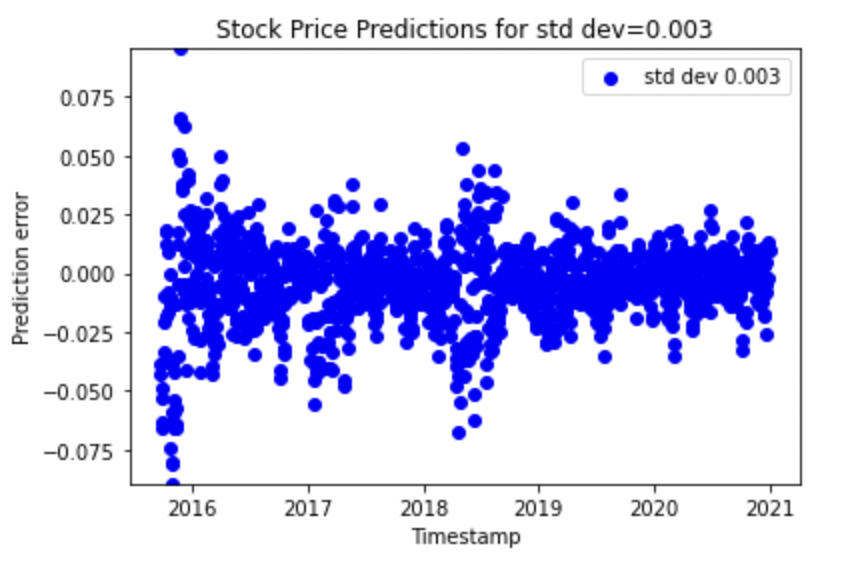
Although different levels of noises were added it was observed that there were miniscule changes in the prediction error values. Furthermore, as the standard deviation of noise level increased, the prediction error values had minute changes which possibly indicates that the changes in the stock prices were so minute.

The values of the prediction with respect to each standard deviation were tabulated.As the table has a large dimension, only a portion of the **table** has been displayed in **Figure** **15.**

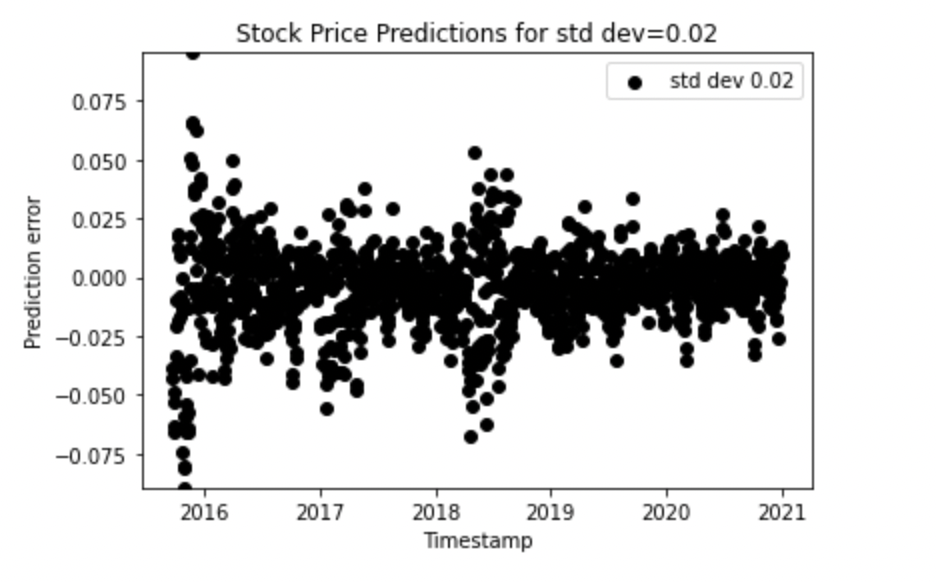
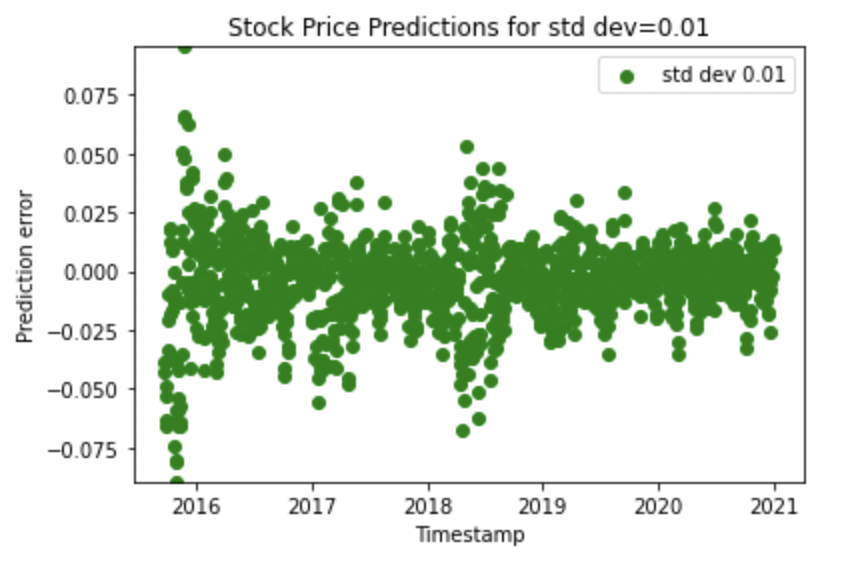


**Figure 15:** Portion of Table for prediction error for standard deviations

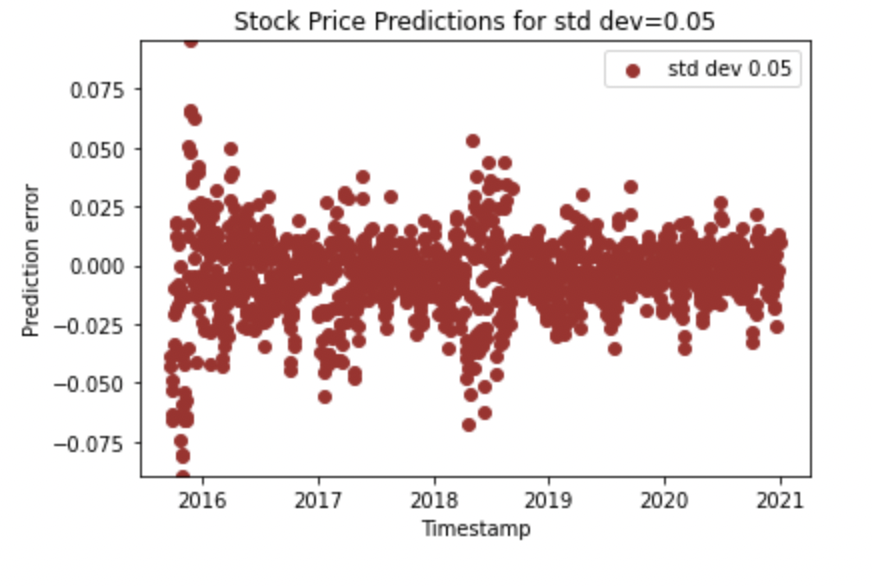
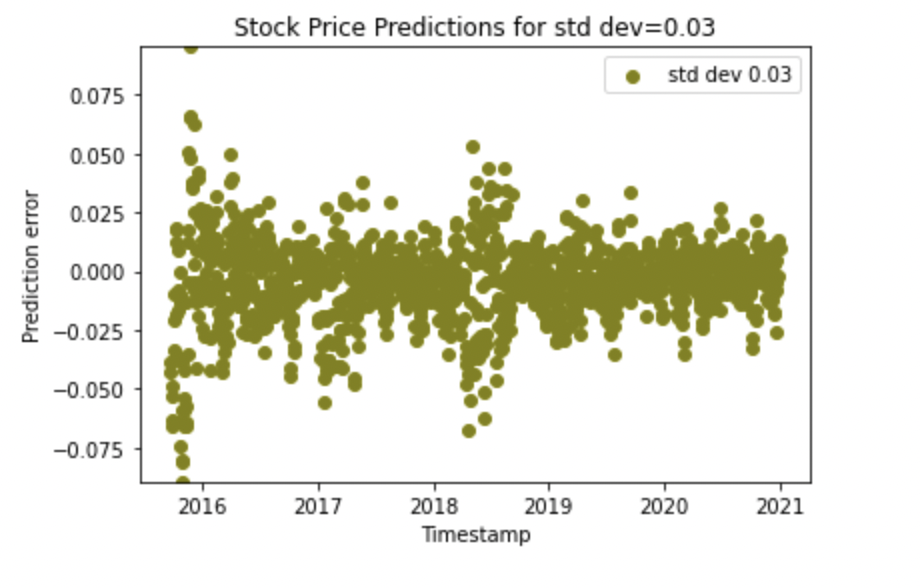
****

**Figure 16:** Prediction error vs timestamp graph for **Figure 17:** Prediction error vs timestamp graph for standard deviation=0.002 standard deviation=0.001

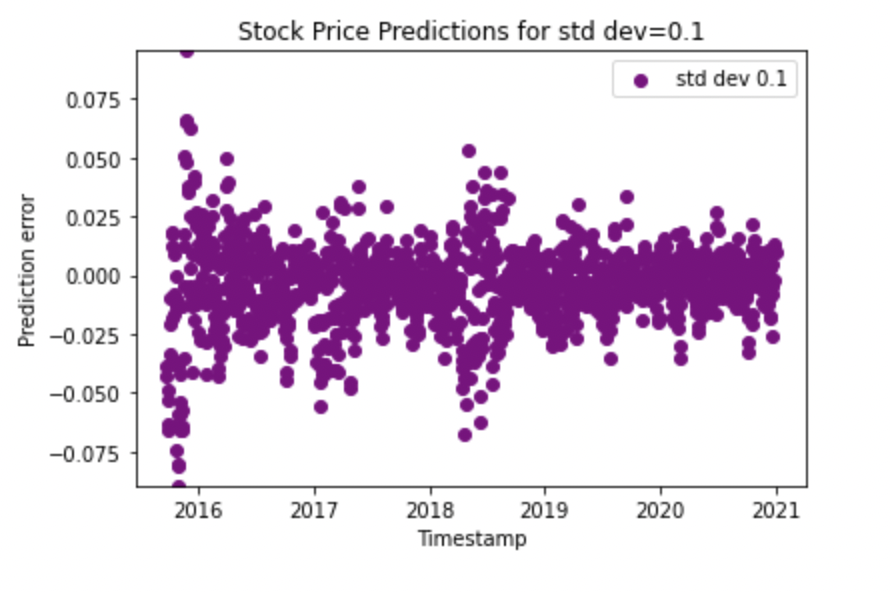
**Figure 18:** Prediction error vs timestamp graph for **Figure 19:** Prediction error vs timestamp graph for standard deviation=0.003 standard deviation=0.005



**Figure 20:** Prediction error vs timestamp graph for **Figure 21:** Prediction error vs timestamp graph for standard deviation=0.01 standard deviation=0.02



**Figure 22:** Prediction error vs timestamp graph for **Figure 23:** Prediction error vs timestamp graph for standard deviation=0.03 standard deviation=0.05



**Figure 24:** Prediction error vs timestamp graph for standard deviation=0.1

**7. DISCUSSION**

**Univariate Model**

This focuses on a single dependent variable. The basic assumption behind the univariate prediction approach is that the value of a time-series at time-step t is closely related to the values at the previous time-steps t-1, t-2, t-3 and so on.

Univariate models are easier to develop than multivariate models. The dependent variable in stock market forecasting is usually the closing or opening price of a finance asset. A forecasting model that is trained solely on the basis of price developments attempts.

**Multivariate Model**

Time series forecasting is about estimating the future value of a time series on the basis of past data. Many time series problems can be solved by looking at a single step in the future. Multi-step time series prediction models the distribution of future values of a signal over a prediction horizon. This approach predicts multiple output values at the same time which is forecasting approach to predict the further course of gradually rising sine wave. In addition, many of these variables are interdependent, which makes statistical modeling even are more complex.

From the above overlay in ***Figure 3*** it implies that shiller P/E ratio is weakly correlated. So, the LSTM model is built using ‘Close’. Univariate and 2 multivariate models are trained. In the multivariate model which used average and adjacent close as input features, the accuracy of the model was **53.4** percent which is shown in the appendix.The input features used for this model were average and adjacent close. Average feature was calculated as the midpoint of high and low attributes.In the multivariate model which used all the input features, the accuracy of the model was **53.9** percent which is shown in the appendix.

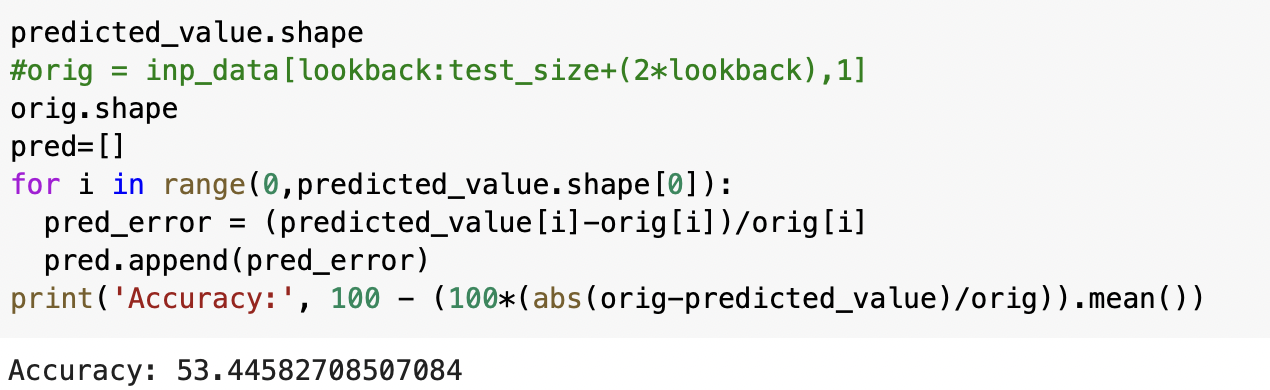
It can be observed from ***figure 10*** that the **correlation** between the feature columns is not very high. Since this **correlation** is **less**, the **resulting model** that has been built does **not** have a **high accuracy**. As this correlation affects the accuracy of the model. So, the **model** here is an **univariate model** which uses the **‘Close’** column input feature. Compared to the other correlation of other columns, Close, Adj.Close, High have **relatively higher correlation** and the accuracy of the model was **97.72** percent.

Furthering the discussion, the performance of the model has been evaluated for noisy data. Additive gaussian noise has been introduced at different standard deviation levels and the model was tested on this noisy data. The predictions for different noise levels were similar as the noise introduced was small. So, very minute changes were observed. To improve this, perhaps a different noise could have been introduced. Also, to improve the accuracy, a sliding window of a different size can be used as well. Also, a multivariate implementation can be used rather than the univariate model. An attempt was made to use the multivariate model which is discussed in the **Appendix**.

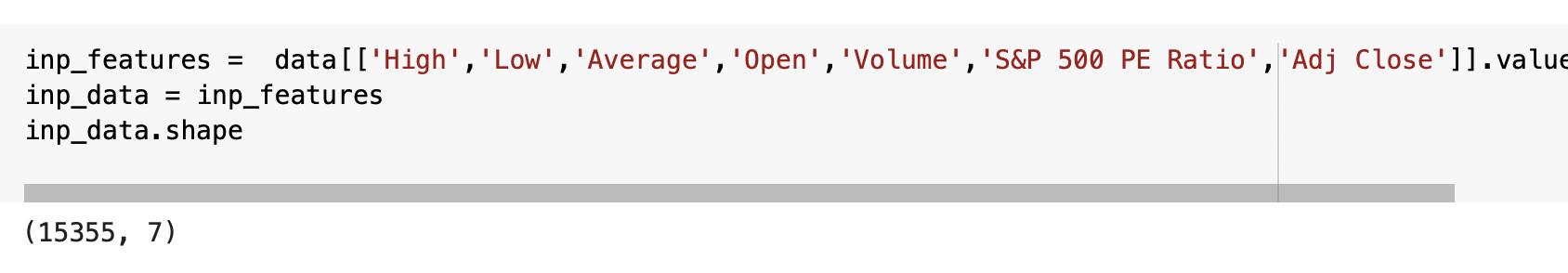
**8. Appendix**

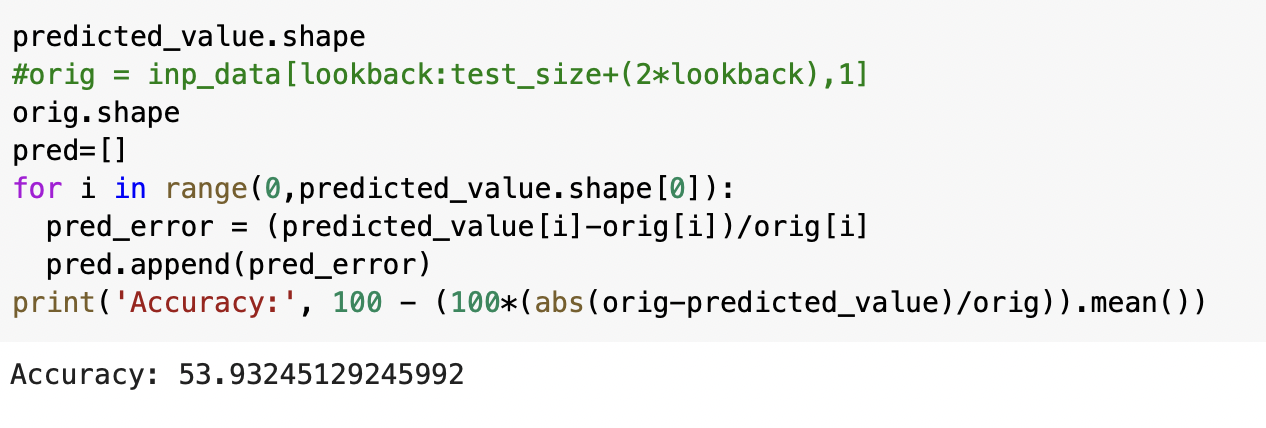
**Multivariate model built using Average and Adjacent close as input features**

****

****

**Multivariate model built using all the input features in the dataset**

****

****