CAP 6615 - Neural Networks
Programming Assignment-3
Recurrent Neural Network

Venkata Vynatheya Jangal, Sohith Raja Buggaveeti, Upendar Penmetcha, Chinmai Sai Eshwar Reddy Kasi and Venkateswarlu Tanneru

University of Florida

Friday, 18th March 2022

Contents

Recurrent Neural Network

1 Abstract:

A Recurrent Neural Network (RNN) is a type of artificial neural network in which the connections between nodes build a directed graph over time. It's a type of generalized feed forward network with internal memory that can handle the input data.
In this assignment, a recurrent neural network (RNN) is created and trained on S&P 500 data. A sliding sampling window of 180 days is used to test the model by predicting the following one, two, three, or four values.
The models are also re-tested on data that has been damaged by noise to see how well they perform. The purpose of this research is to optimize the RNN such that it produces the best results possible.

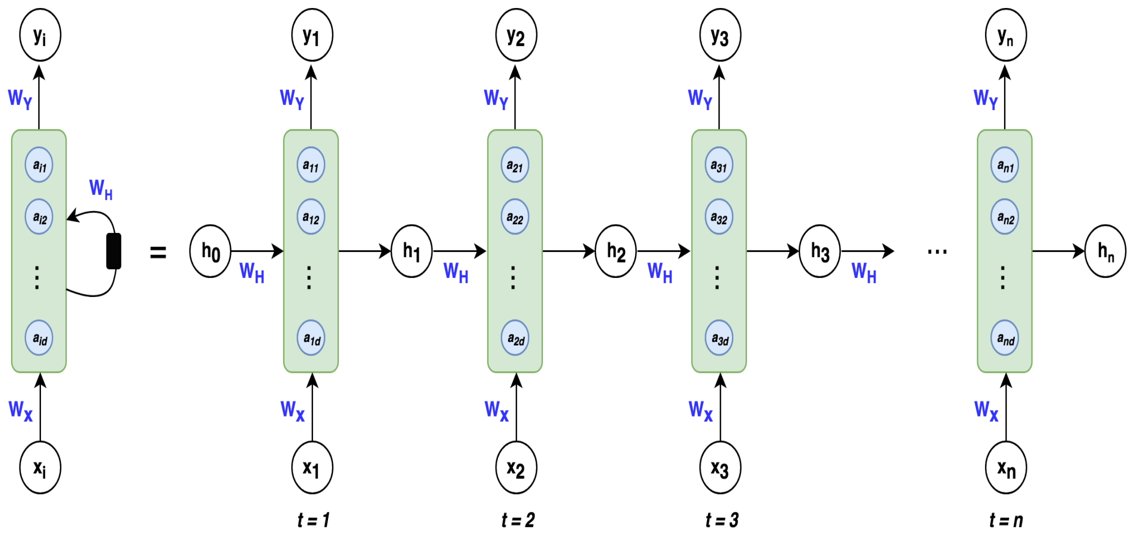
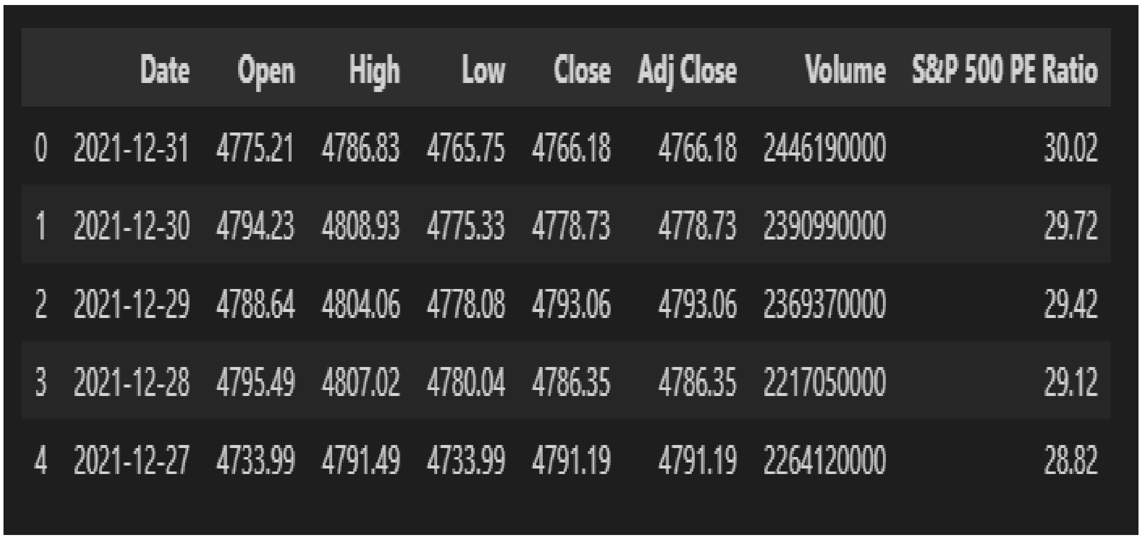


Figure 1: Recurrent Neural Network

2 Dataset Generation:

A S&P 500 dataset between January 4, 1960 and December 31, 2021 was gathered from the Yahoo Finance website to be used as input for this assignment. The table is scraped from the website and was saved as a csv file. One of the standard indicators used to determine whether a market is overpriced, undervalued, or correctly valued is the Shiller P/E ratio. The Shiller-PE website has it available for download. From 1960 to 2021, the Shiller P/E ratio data was gathered monthy. To transform this monthly data to daily data, we apply the linear interpolation approach. The data from the S&P 500 and the Shiller P/E ratio were then combined and saved in S&P500.csv. After then, the data was split into training and test sets.



	Date	Open	High	Low	Close	Adj Close	Volume	S&P 500 PE Ratio
0	2021-12-31	4775.21	4786.83	4765.75	4766.18	4766.18	2446190000	30.02
1	2021-12-30	4794.23	4808.93	4775.33	4778.73	4778.73	2390990000	29.72
2	2021-12-29	4788.64	4804.06	4778.08	4793.06	4793.06	2369370000	29.42
3	2021-12-28	4795.49	4807.02	4780.04	4786.35	4786.35	2217050000	29.12
4	2021-12-27	4733.99	4791.49	4733.99	4791.19	4791.19	2264120000	28.82

Figure 2: Dataset

The resulting dataset is an N-dimensional time series data indicated by the symbols P_0,P_1,...,P_{N-1}. The whole dataset is split into 180 days (6 months) fixed-size windows. As a result, whenever the window is pushed to the right by size W, the data in it is pushed to the right as well. The sliding windows do not overlap in any way. After that, the data is normalized to the range [0,1]. After that, the data is separated into test and training datasets.

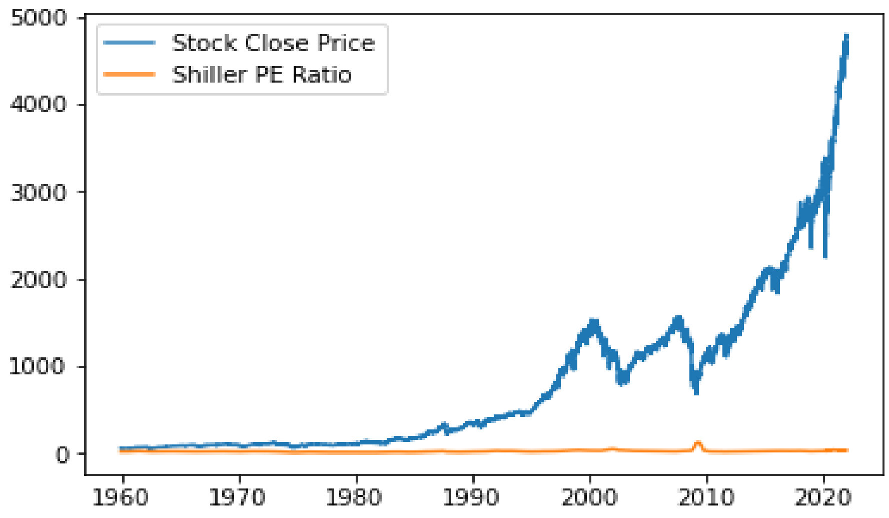


Figure 3: Graph of Dataset with Shiller P/E ratio

3 Network Parameters

1. **Input:** Generated Dataset S&P 500 data from Jan 1^{st},1960 to Dec 31^{st},2021.
2. **Predicted Output feature:** Close price
3. **Activation function:** sigmoid
4. **Epochs:** 1 and 5
5. **Optimizer:** Adam
6. **Evaluation metrics:** prediction error, accuracy
7. **Number of time steps:** 180
8. **Loss metrics:** Mean squared error

The model is trained using data from 1960 to 2000 and then tested using data from 2000 to 2021. The RNN that was created is a regression model that forecasts future Stock Close Prices. The model is then evaluated by comparing the expected price output to the original price values.

No of Epochs	Accuracy
1	92.3
5	89.3

Table 1: Unoptimized RNN Prediction Accuracy

No of Epochs	Accuracy
1	97.3
5	97.3

Table 2: Optimized RNN Prediction Accuracy

The number of epochs was chosen in order to increase the model's performance. While adjusting the number of epochs, the accuracy of the model was calculated for both optimized and unoptimized models. When using an optimizer, the accuracy increased from 92.3 to 97.3 when the number of epochs was set to 1. It has also been noted that when the number of epochs was increased in the case of unoptimized model, the model has become over-fitted, resulting in a significant drop in the model's accuracy.
The optimized RNN generated superior outcomes even with 1 epoch. But it can be noted that the accuracy remained the same even when the epochs were increased to 5. The accuracy of the unoptimized RNN is 89.3 with 5 epochs while the optimized RNN had an accuracy of 97.3 with 5 epochs, which is a suitable accuracy.

4 RNN Model

The LSTM model is the architecture used to deploy this model. Long Short Term Memory (LSTM) is an acronym for Long Short Term Memory. Long Short-Term Memory (LSTM) networks are a modified version of recurrent neural networks that make it simpler to recall information from the past. Here, the RNN's vanishing gradient problem is solved. Given time lags of uncertain duration, LSTM is well-suited to identify, analyse, and predict time series. Back-propagation is used to train the model. Three gates are present in an LSTM network:

(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:** Identify which details in the block should be removed. The sigmoid function determines this. It examines the previous state (h t-1) and the content input(Xt) and outputs a number between 0 and 1 for each number in the cell state C t1.

4.1 Loss Metrics:

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.
One parameter for evaluating classification models is accuracy. Informally, accuracy refers to the percentage of correct predictions made by our model. The following is the formal definition of accuracy:

Accuracy = (Number of correct predictions) / Total Number of predictions

Mean squared Error(MSD): In regression circumstances where your expected and predicted outcomes are real-number values, mean squared error is used.The loss is calculated using a simple formula. It's simply the difference in squares between the expected and predicted values.

$$MSD = \sum_{i=1}^{D}(x_i - y_i)^2$$

4.2 Activation Function:

1. **Sigmoid Function:** We used the sigmoid activation function since each neuron must anticipate a value between [0,1]. The sigmoid function looks like this:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

2. **Softmax Function:**In neural network models that predict a multinomial probability distribution, the softmax function is utilized as the activation function in the output layer.Softmax is used as the activation function for multi-class classification problems requiring class membership on more than two labels.Although less popular, the function can be utilized as an activation function for a hidden layer in a neural network. At a bottleneck or concatenation stage, it may be employed when the model needs to choose or weight numerous separate inputs internally.

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

3. **TanH function:** Tanh is similar to the logistic sigmoid, however it is superior. The tanh function has a range of values between 0 and 1. (-1 to 1). Tanh is also a sigmoidal function (s - shaped).

$$\tanh(\frac{x-g}{h})$$

4.3 Optimizer:

Adam can be thought of as a hybrid of RMSprop and Stochastic Gradient Descent, with the addition of momentum. It, like RMSprop, uses squared gradients to scale the learning rate and takes advantage of momentum by using the gradient's moving average.

- **Adam Optimizer:**The Adam optimizer employs a hybrid of two gradient descent techniques: Momentum: By taking into account the 'exponentially weighted average' of the gradients, this approach is utilized to speed up the gradient descent algorithm.
- **Adamax Optimizer:** The Adamax algorithm is implemented by this optimizer. It's an Adam version that uses the infinite norm. The default parameters are the same as those in the paper. In some cases, Adamax outperforms Adam, particularly in models containing embeddings.
- **SGD Optimizer:**The iterative approach of stochastic gradient descent is used to optimize an objective function with sufficient smoothness criteria. Because it replaces the real gradient with an estimate, it can be considered a stochastic approximation of gradient descent optimization. This minimizes the computing cost, especially in high-dimensional optimization problems, allowing for faster iterations in exchange for a reduced convergence rate.

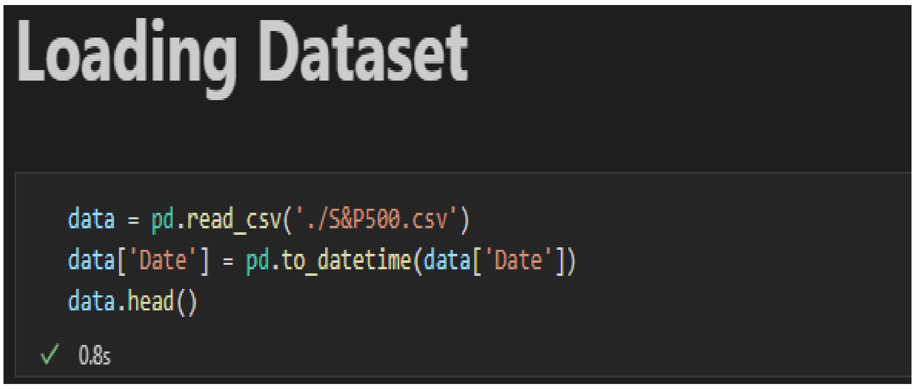
Reason for Choosing Adam Optimizer:

- One interesting and dominant argument about optimizers is that SGD better generalizes than Adam but Adam has a greater speed.
- Picking the right optimizer with the right parameters, can help you squeeze the last bit of accuracy out of your neural network model.
- Adam uses Momentum and Adaptive Learning Rates to converge faster.

5 RNN Model:

5.1 Data set Generation :

Below code represents a code to convert the data obtained from Yahoo finance to data frame using the Pandas.



Loading Dataset

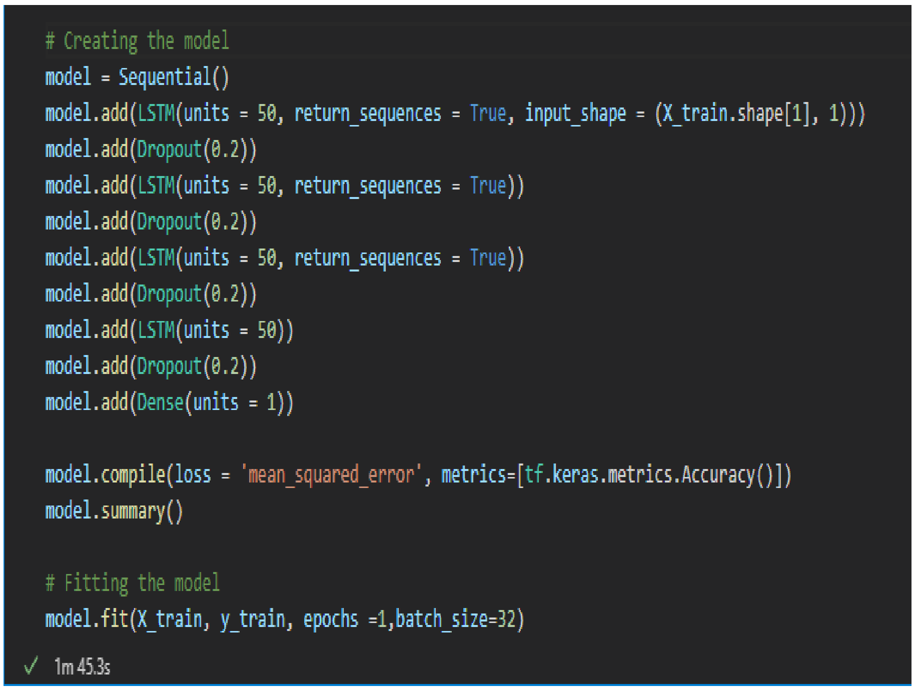
```
data = pd.read_csv('./S&P500.csv')
data['Date'] = pd.to_datetime(data['Date'])
data.head()
```

✓ 0.8s

Figure 4: Dataset Generation

5.2 Initialization and Training Model :

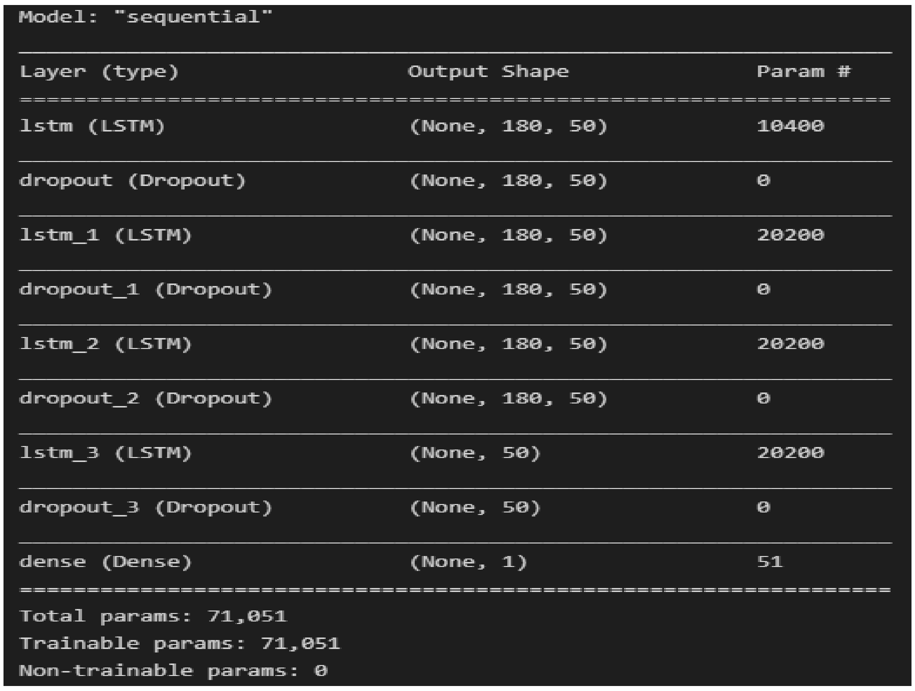
The model has been initialized with one LSTM layer, followed by three other LSTM layers that also use dropout to boost efficiency in the code below. A Dense layer follows the final layer.Here, we have the sliding window of size 180. Mean square error is used and we have also used Adam Optimizer. Below code represents a model that is initialized by LSTM layer



```
# Creating the model
model = Sequential()
model.add(LSTM(units = 50, return_sequences = True, input_shape = (X_train.shape[1], 1)))
model.add(Dropout(0.2))
model.add(LSTM(units = 50, return_sequences = True))
model.add(Dropout(0.2))
model.add(LSTM(units = 50, return_sequences = True))
model.add(Dropout(0.2))
model.add(LSTM(units = 50))
model.add(Dropout(0.2))
model.add(Dense(units = 1))

model.compile(loss = 'mean_squared_error', metrics=[tf.keras.metrics.Accuracy()])
model.summary()

# Fitting the model
model.fit(X_train, y_train, epochs =1,batch_size=32)
```

✓ 1m 45.3s

Figure 5: Model

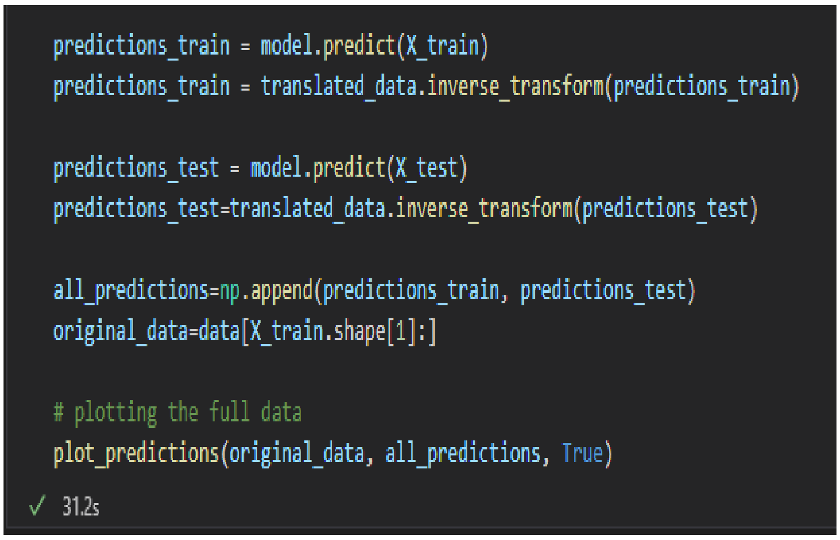


```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 180, 50)           10400
_________________________________________________________________
dropout (Dropout)            (None, 180, 50)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 180, 50)           20200
_________________________________________________________________
dropout_1 (Dropout)          (None, 180, 50)           0
_________________________________________________________________
lstm_2 (LSTM)                (None, 180, 50)           20200
_________________________________________________________________
dropout_2 (Dropout)          (None, 180, 50)           0
_________________________________________________________________
lstm_3 (LSTM)                (None, 50)                20200
_________________________________________________________________
dropout_3 (Dropout)          (None, 50)                0
_________________________________________________________________
dense (Dense)                (None, 1)                 51
=================================================================
Total params: 71,051
Trainable params: 71,051
Non-trainable params: 0
```

Figure 6: Model Data

5.3 Generating Predictions :

The above-mentioned trained model was used to create predictions, which were then utilized to assess the model's performance.



```
predictions_train = model.predict(X_train)
predictions_train = translated_data.inverse_transform(predictions_train)

predictions_test = model.predict(X_test)
predictions_test=translated_data.inverse_transform(predictions_test)

all_predictions=np.append(predictions_train, predictions_test)
original_data=data[X_train.shape[1]:]

# plotting the full data
plot_predictions(original_data, all_predictions, True)
```
✓ 31.2s

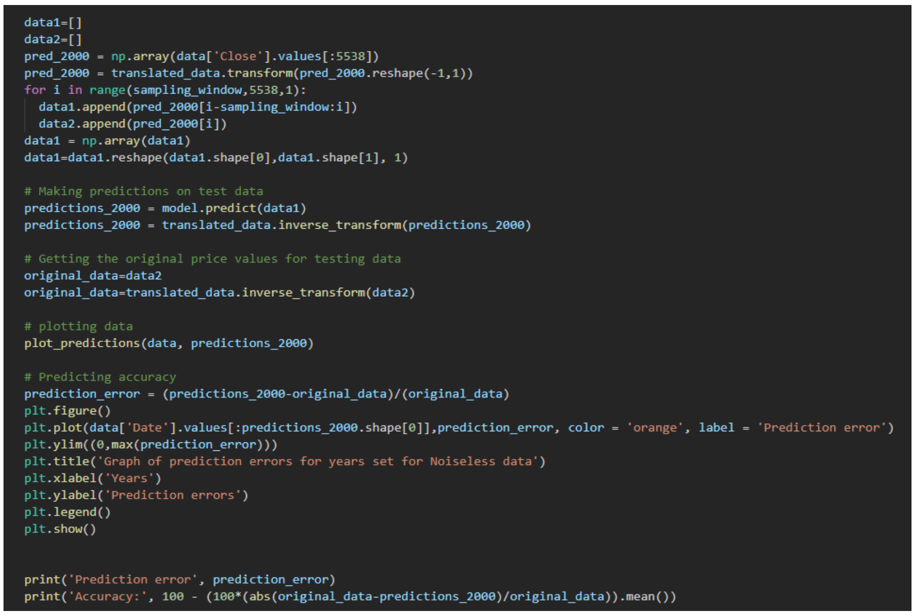
Figure 7: Model Data

5.4 Calculation of performance metrics :

Prediction error and accuracy are the performance metrics evaluated. The error in prediction is calculated as follows:

prediction_error = (predicted_price-original_price)/original_price

Prediction error is calculated and is a plotted as a graph against timestamps as shown below:

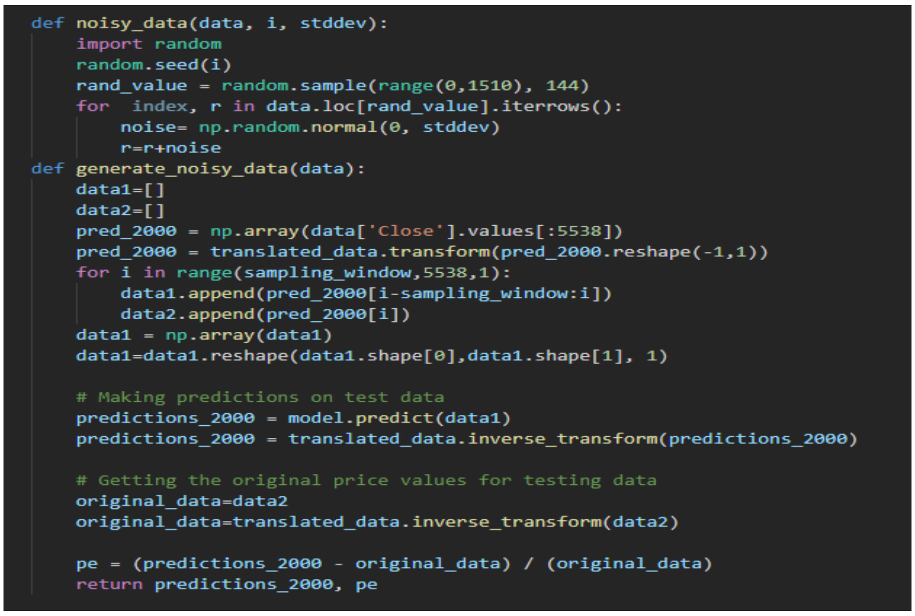


```python
data1=[]
data2=[]
pred_2000 = np.array(data['Close'].values[:5538])
pred_2000 = translated_data.transform(pred_2000.reshape(-1,1))
for i in range(sampling_window,5538,1):
  data1.append(pred_2000[i-sampling_window:i])
  data2.append(pred_2000[i])
data1 = np.array(data1)
data1=data1.reshape(data1.shape[0],data1.shape[1], 1)

# Making predictions on test data
predictions_2000 = model.predict(data1)
predictions_2000 = translated_data.inverse_transform(predictions_2000)

# Getting the original price values for testing data
original_data=data2
original_data=translated_data.inverse_transform(data2)

# plotting data
plot_predictions(data, predictions_2000)

# Predicting accuracy
prediction_error = (predictions_2000-original_data)/(original_data)
plt.figure()
plt.plot(data['Date'].values[:predictions_2000.shape[0]],prediction_error, color = 'orange', label = 'Prediction error')
plt.ylim((0,max(prediction_error)))
plt.title('Graph of prediction errors for years set for Noiseless data')
plt.xlabel('Years')
plt.ylabel('Prediction errors')
plt.legend()
plt.show()


print('Prediction error', prediction_error)
print('Accuracy:', 100 - (100*(abs(original_data-predictions_2000)/original_data)).mean())
```

Figure 8: Model Data

5.5 Adding noise to input data :

By applying additive gaussian noise to an input data, it is perturbed.The following code snippets demonstrate how to do so. Each standard deviation's prediction error has been calculated and tabulated.



```python
def noisy_data(data, i, stddev):
    import random
    random.seed(i)
    rand_value = random.sample(range(0,1510), 144)
    for  index, r in data.loc[rand_value].iterrows():
        noise= np.random.normal(0, stddev)
        r=r+noise
def generate_noisy_data(data):
    data1=[]
    data2=[]
    pred_2000 = np.array(data['Close'].values[:5538])
    pred_2000 = translated_data.transform(pred_2000.reshape(-1,1))
    for i in range(sampling_window,5538,1):
        data1.append(pred_2000[i-sampling_window:i])
        data2.append(pred_2000[i])
    data1 = np.array(data1)
    data1=data1.reshape(data1.shape[0],data1.shape[1], 1)

    # Making predictions on test data
    predictions_2000 = model.predict(data1)
    predictions_2000 = translated_data.inverse_transform(predictions_2000)

    # Getting the original price values for testing data
    original_data=data2
    original_data=translated_data.inverse_transform(data2)

    pe = (predictions_2000 - original_data) / (original_data)
    return predictions_2000, pe
```

Figure 9: Code for adding Noise

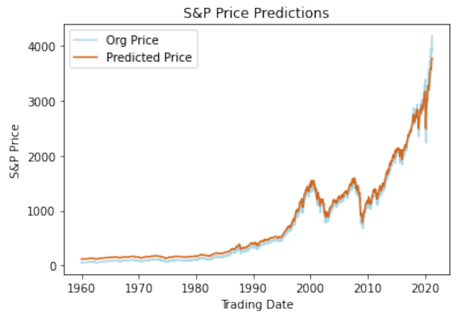
6 Performance evaluation for Optimized and UnOptimized RNN on Noiseless Data

In comparison to the unoptimized model, the prediction error is minimal due to the use of the Adam optimizer and a sufficient number of epochs, which in our model is 100. The accuracy of the unoptimized RNN is 95.14 epochs, whereas the optimized RNN is 97.72 epochs.

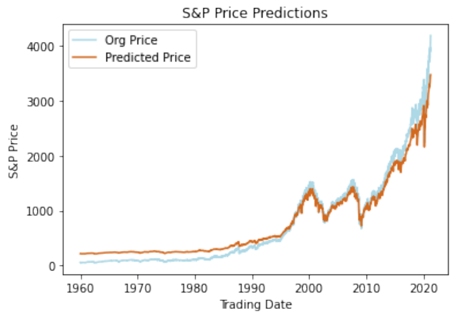
6.1 Unoptimized Model Output Graphs:

The data predictions are done in 2 ways where we predicted the data by training model with total data but in other case we just predict from the years after 2000.

6.1.1 Prediction made on all years:



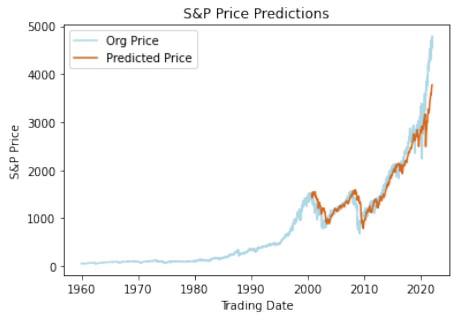
(a) Predicted Data 1 Epoch - All Years



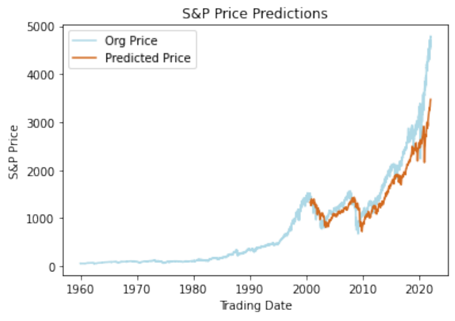
(b) Predicted Data 5 Epoch - All Years

Figure 10: Unoptimized Model data Prediction

6.1.2 Prediction made for years after 2000:



(a) Predicted Data 1 Epoch for years after 2000

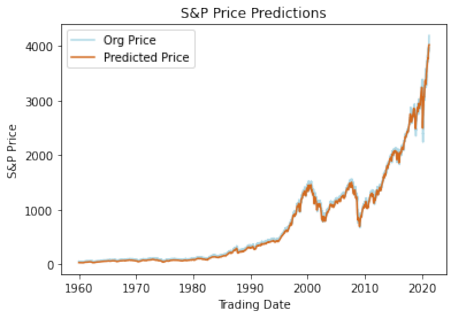


(b) Predicted Data 5 Epoch for years after 2000

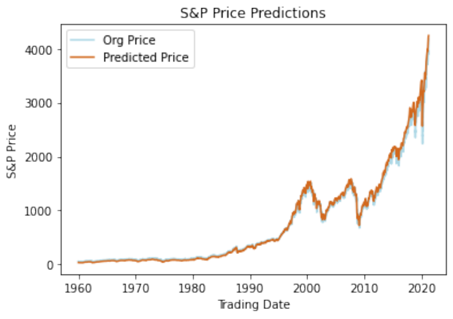
Figure 11: Unoptimized Model data Prediction

6.2 Optimized Model Output Graphs:

6.2.1 Prediction made on all years:



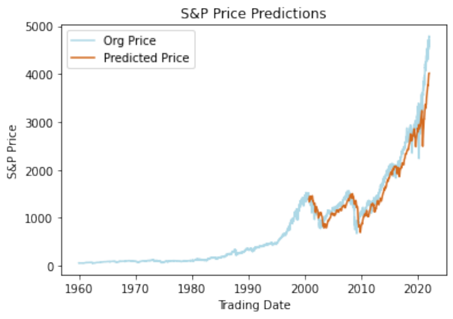
(a) Predicted Data 1 Epoch - All Years



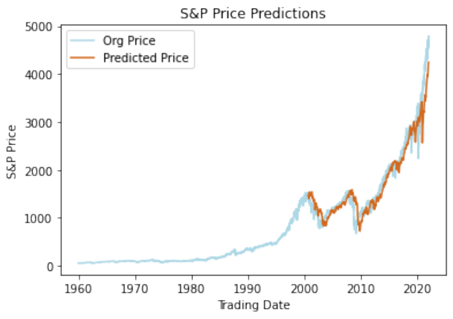
(b) Predicted Data 5 Epoch - All Years

Figure 12: Optimized Model data Prediction

6.2.2 Prediction made for years after 2000:



(a) Predicted Data 1 Epoch for years after 2000



(b) Predicted Data 5 Epoch for years after 2000

Figure 13: Optimized Model data Prediction

7 Tabulating the prediction error:

Table on Prediction Error data from years 2000 to 2021,This table contains various outputs of noised infused data at different standard deviations.

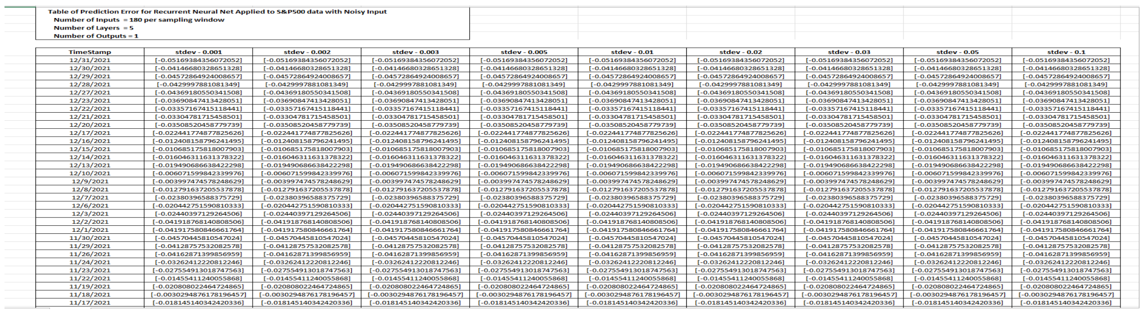
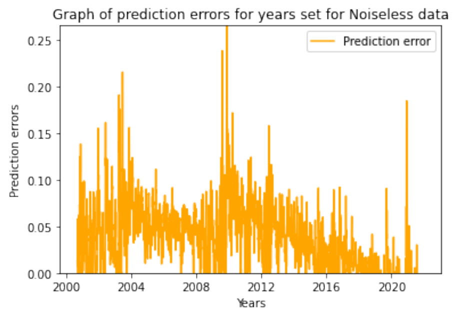


Table of Prediction Error for Recurrent Neural Net Applied to S&P500 data with Noisy Input
Number of Inputs = 180 per sampling window
Number of Layers = 5
Number of Outputs = 1

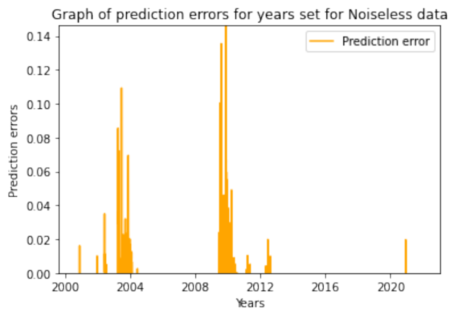
TimeStamp	stdev - 0.001	stdev - 0.002	stdev - 0.003	stdev - 0.005	stdev - 0.01	stdev - 0.02	stdev - 0.03	stdev - 0.05	stdev - 0.1
12/31/2021	[-0.05169384356072052]	[-0.05169384356072052]	[-0.05169384356072052]	[-0.05169384356072052]	[-0.05169384356072052]	[-0.05169384356072052]	[-0.05169384356072052]	[-0.05169384356072052]	[-0.05169384356072052]
12/30/2021	[-0.04146680328651328]	[-0.04146680328651328]	[-0.04146680328651328]	[-0.04146680328651328]	[-0.04146680328651328]	[-0.04146680328651328]	[-0.04146680328651328]	[-0.04146680328651328]	[-0.04146680328651328]
12/29/2021	[-0.04572864924008657]	[-0.04572864924008657]	[-0.04572864924008657]	[-0.04572864924008657]	[-0.04572864924008657]	[-0.04572864924008657]	[-0.04572864924008657]	[-0.04572864924008657]	[-0.04572864924008657]
12/28/2021	[-0.0429997881081349]	[-0.0429997881081349]	[-0.0429997881081349]	[-0.0429997881081349]	[-0.0429997881081349]	[-0.0429997881081349]	[-0.0429997881081349]	[-0.0429997881081349]	[-0.0429997881081349]
12/27/2021	[-0.04369180550341508]	[-0.04369180550341508]	[-0.04369180550341508]	[-0.04369180550341508]	[-0.04369180550341508]	[-0.04369180550341508]	[-0.04369180550341508]	[-0.04369180550341508]	[-0.04369180550341508]
12/23/2021	[-0.03690847413428051]	[-0.03690847413428051]	[-0.03690847413428051]	[-0.03690847413428051]	[-0.03690847413428051]	[-0.03690847413428051]	[-0.03690847413428051]	[-0.03690847413428051]	[-0.03690847413428051]
12/22/2021	[-0.03357167415118441]	[-0.03357167415118441]	[-0.03357167415118441]	[-0.03357167415118441]	[-0.03357167415118441]	[-0.03357167415118441]	[-0.03357167415118441]	[-0.03357167415118441]	[-0.03357167415118441]
12/21/2021	[-0.03304781715458501]	[-0.03304781715458501]	[-0.03304781715458501]	[-0.03304781715458501]	[-0.03304781715458501]	[-0.03304781715458501]	[-0.03304781715458501]	[-0.03304781715458501]	[-0.03304781715458501]
12/20/2021	[-0.03508520458779739]	[-0.03508520458779739]	[-0.03508520458779739]	[-0.03508520458779739]	[-0.03508520458779739]	[-0.03508520458779739]	[-0.03508520458779739]	[-0.03508520458779739]	[-0.03508520458779739]
12/17/2021	[-0.022441774877825626]	[-0.022441774877825626]	[-0.022441774877825626]	[-0.022441774877825626]	[-0.022441774877825626]	[-0.022441774877825626]	[-0.022441774877825626]	[-0.022441774877825626]	[-0.022441774877825626]
12/16/2021	[-0.012408158796241495]	[-0.012408158796241495]	[-0.012408158796241495]	[-0.012408158796241495]	[-0.012408158796241495]	[-0.012408158796241495]	[-0.012408158796241495]	[-0.012408158796241495]	[-0.012408158796241495]
12/15/2021	[-0.010685175818007903]	[-0.010685175818007903]	[-0.010685175818007903]	[-0.010685175818007903]	[-0.010685175818007903]	[-0.010685175818007903]	[-0.010685175818007903]	[-0.010685175818007903]	[-0.010685175818007903]
12/14/2021	[-0.016046311631378322]	[-0.016046311631378322]	[-0.016046311631378322]	[-0.016046311631378322]	[-0.016046311631378322]	[-0.016046311631378322]	[-0.016046311631378322]	[-0.016046311631378322]	[-0.016046311631378322]
12/13/2021	[-0.019490686638422298]	[-0.019490686638422298]	[-0.019490686638422298]	[-0.019490686638422298]	[-0.019490686638422298]	[-0.019490686638422298]	[-0.019490686638422298]	[-0.019490686638422298]	[-0.019490686638422298]
12/10/2021	[-0.006071599842339976]	[-0.006071599842339976]	[-0.006071599842339976]	[-0.006071599842339976]	[-0.006071599842339976]	[-0.006071599842339976]	[-0.006071599842339976]	[-0.006071599842339976]	[-0.006071599842339976]
12/9/2021	[-0.003997474578248629]	[-0.003997474578248629]	[-0.003997474578248629]	[-0.003997474578248629]	[-0.003997474578248629]	[-0.003997474578248629]	[-0.003997474578248629]	[-0.003997474578248629]	[-0.003997474578248629]
12/8/2021	[-0.012791637205537878]	[-0.012791637205537878]	[-0.012791637205537878]	[-0.012791637205537878]	[-0.012791637205537878]	[-0.012791637205537878]	[-0.012791637205537878]	[-0.012791637205537878]	[-0.012791637205537878]
12/7/2021	[-0.02380396588375729]	[-0.02380396588375729]	[-0.02380396588375729]	[-0.02380396588375729]	[-0.02380396588375729]	[-0.02380396588375729]	[-0.02380396588375729]	[-0.02380396588375729]	[-0.02380396588375729]
12/6/2021	[-0.020442751590810333]	[-0.020442751590810333]	[-0.020442751590810333]	[-0.020442751590810333]	[-0.020442751590810333]	[-0.020442751590810333]	[-0.020442751590810333]	[-0.020442751590810333]	[-0.020442751590810333]
12/3/2021	[-0.02440397129264506]	[-0.02440397129264506]	[-0.02440397129264506]	[-0.02440397129264506]	[-0.02440397129264506]	[-0.02440397129264506]	[-0.02440397129264506]	[-0.02440397129264506]	[-0.02440397129264506]
12/2/2021	[-0.041918768140808506]	[-0.041918768140808506]	[-0.041918768140808506]	[-0.041918768140808506]	[-0.041918768140808506]	[-0.041918768140808506]	[-0.041918768140808506]	[-0.041918768140808506]	[-0.041918768140808506]
12/1/2021	[-0.041917580846661764]	[-0.041917580846661764]	[-0.041917580846661764]	[-0.041917580846661764]	[-0.041917580846661764]	[-0.041917580846661764]	[-0.041917580846661764]	[-0.041917580846661764]	[-0.041917580846661764]
11/30/2021	[-0.04570445810547024]	[-0.04570445810547024]	[-0.04570445810547024]	[-0.04570445810547024]	[-0.04570445810547024]	[-0.04570445810547024]	[-0.04570445810547024]	[-0.04570445810547024]	[-0.04570445810547024]
11/29/2021	[-0.04128757532082578]	[-0.04128757532082578]	[-0.04128757532082578]	[-0.04128757532082578]	[-0.04128757532082578]	[-0.04128757532082578]	[-0.04128757532082578]	[-0.04128757532082578]	[-0.04128757532082578]
11/26/2021	[-0.04162871399856959]	[-0.04162871399856959]	[-0.04162871399856959]	[-0.04162871399856959]	[-0.04162871399856959]	[-0.04162871399856959]	[-0.04162871399856959]	[-0.04162871399856959]	[-0.04162871399856959]
11/24/2021	[-0.03262412220812246]	[-0.03262412220812246]	[-0.03262412220812246]	[-0.03262412220812246]	[-0.03262412220812246]	[-0.03262412220812246]	[-0.03262412220812246]	[-0.03262412220812246]	[-0.03262412220812246]
11/23/2021	[-0.027554913018747563]	[-0.027554913018747563]	[-0.027554913018747563]	[-0.027554913018747563]	[-0.027554913018747563]	[-0.027554913018747563]	[-0.027554913018747563]	[-0.027554913018747563]	[-0.027554913018747563]
11/22/2021	[-0.01455411240055868]	[-0.01455411240055868]	[-0.01455411240055868]	[-0.01455411240055868]	[-0.01455411240055868]	[-0.01455411240055868]	[-0.01455411240055868]	[-0.01455411240055868]	[-0.01455411240055868]
11/19/2021	[-0.020808022464724865]	[-0.020808022464724865]	[-0.020808022464724865]	[-0.020808022464724865]	[-0.020808022464724865]	[-0.020808022464724865]	[-0.020808022464724865]	[-0.020808022464724865]	[-0.020808022464724865]
11/18/2021	[-0.0030294876178196457]	[-0.0030294876178196457]	[-0.0030294876178196457]	[-0.0030294876178196457]	[-0.0030294876178196457]	[-0.0030294876178196457]	[-0.0030294876178196457]	[-0.0030294876178196457]	[-0.0030294876178196457]
11/17/2021	[-0.018145140342420336]	[-0.018145140342420336]	[-0.018145140342420336]	[-0.018145140342420336]	[-0.018145140342420336]	[-0.018145140342420336]	[-0.018145140342420336]	[-0.018145140342420336]	[-0.018145140342420336]

Figure 14: Table on Error prediction

8 Prediction Error Graphs for Unoptimized Model:



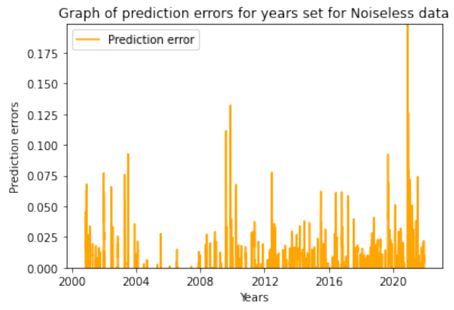
(a) Predicted Error Graph years after 2000



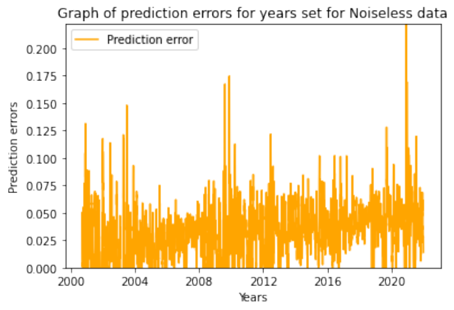
(b) Prediction Error Graph years after 2000

Figure 15: Unoptimized Model Error Prediction Graph

9 Prediction Error Graphs for Optimized Model:



(a) Predicted Error Graph years after 2000



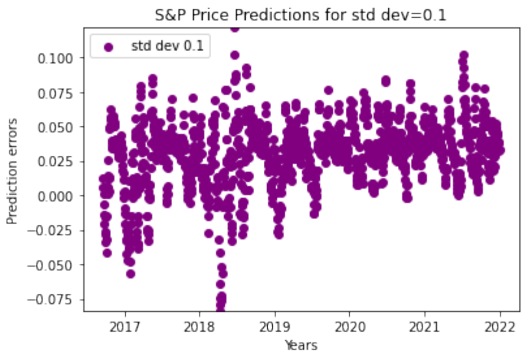
(b) Prediction Error Graph years after 2000

Figure 16: Optimized Model Error Prediction Graph

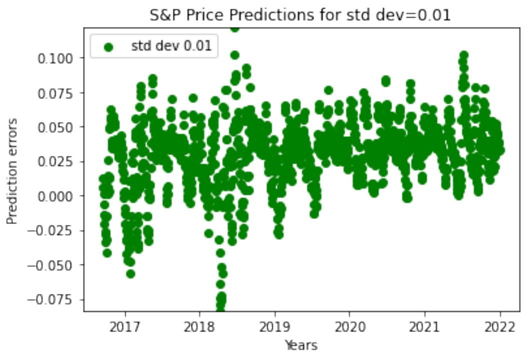
10 Output Graphs of Optimized RNN's for noisy data :

The images are disturbed with the introduction of guassian noise and then they are tested. For each standard deviation calculated and tabulated Prediction error and timestamp is calculated.For each error level, 9 graphs were drawn with the timestamp on the abscissa and the prediction error on the ordinate axis. As it was asked to plot prediction error for each error level in one graph. There are minimal change in prediction error values for different level of noises introduced.Prediction error values had minimal changes even if the standard deviation of the noise level increased. Predition values didn't change much which indicate stock price is minimum.

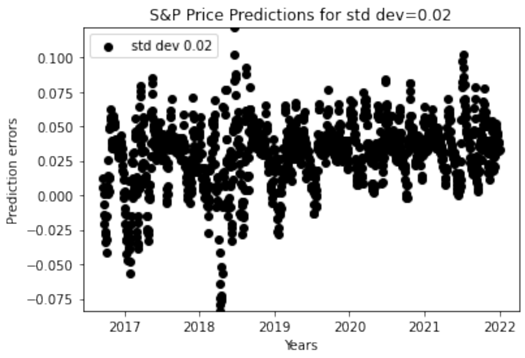
10.1 Noisy Data Prediction Graphs:



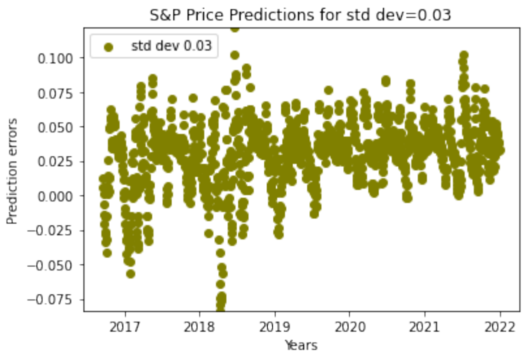
(a) Graph at Std Dev = 0.1



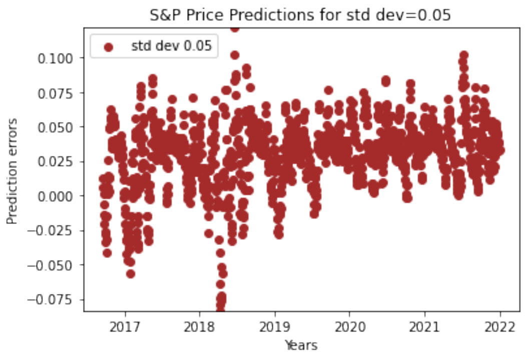
(b) Graph at Std Dev = 0.01



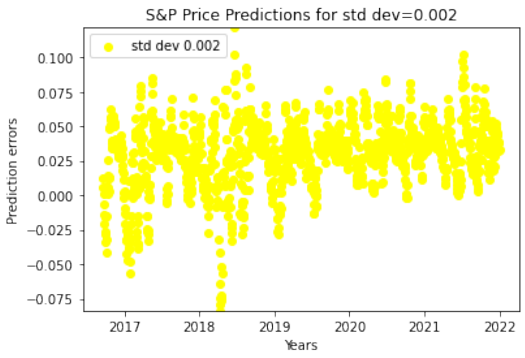
(a) Graph at Std Dev = 0.02



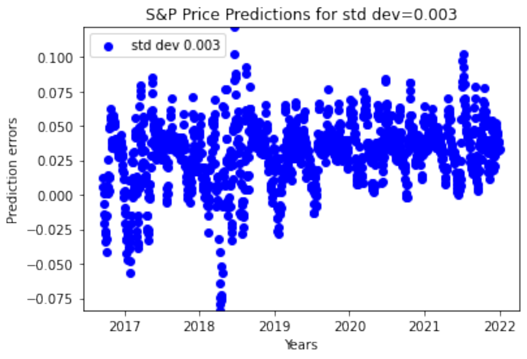
(b) Graph at Std Dev = 0.03



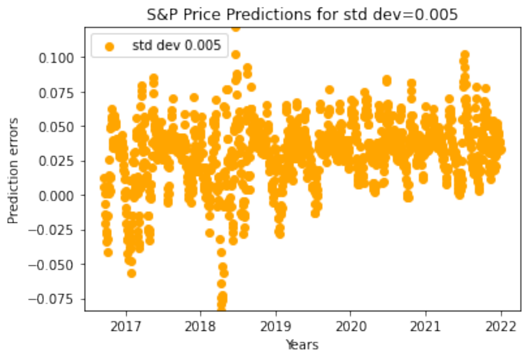
(a) Graph at Std Dev = 0.05



(b) Graph at Std Dev = 0.002



(a) Graph at Std Dev = 0.003

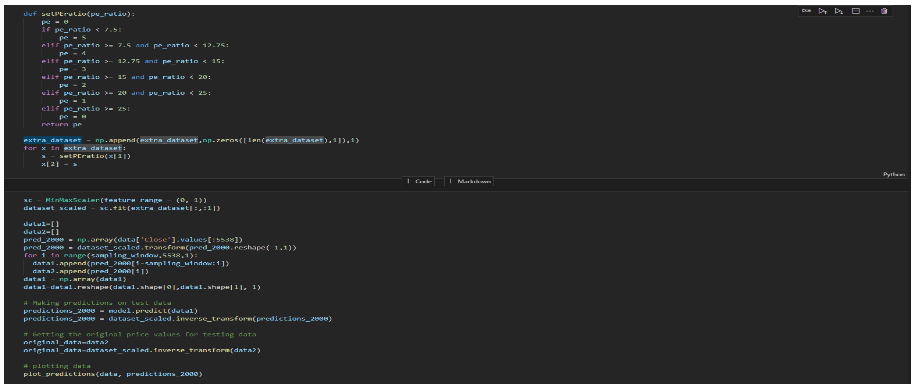


(b) Graph at Std Dev = 0.005

11 Appendix:

11.1 Schiller P/E data to adjust Predictions:

Here we gave a code adjusted the monthly P/E data to interpolate to day to day data,then we used this data and our orginal S&P 500 to adjust the predictions accurately on Daily Data.



```
def setPEratio(pe_ratio):
    pe = 0
    if pe_ratio < 7.5:
        pe = 5
    elif pe_ratio >= 7.5 and pe_ratio < 12.75:
        pe = 4
    elif pe_ratio >= 12.75 and pe_ratio < 15:
        pe = 3
    elif pe_ratio >= 15 and pe_ratio < 20:
        pe = 2
    elif pe_ratio >= 20 and pe_ratio < 25:
        pe = 1
    elif pe_ratio >= 25:
        pe = 0
    return pe

extra_dataset = np.append(extra_dataset,np.zeros([len(extra_dataset),1]),1)
for x in extra_dataset:
    s = setPEratio(x[1])
    x[2] = s
```

```
sc = MinMaxScaler(feature_range = (0, 1))
dataset_scaled = sc.fit(extra_dataset[:,:1])

data1=[]
data2=[]
pred_2000 = np.array(data['Close'].values[:5538])
pred_2000 = dataset_scaled.transform(pred_2000.reshape(-1,1))
for i in range(sampling_window,5538,1):
  data1.append(pred_2000[i-sampling_window:i])
  data2.append(pred_2000[i])
data1 = np.array(data1)
data1=data1.reshape(data1.shape[0],data1.shape[1], 1)

# Making predictions on test data
predictions_2000 = model.predict(data1)
predictions_2000 = dataset_scaled.inverse_transform(predictions_2000)

# Getting the original price values for testing data
original_data=data2
original_data=dataset_scaled.inverse_transform(data2)

# plotting data
plot_predictions(data, predictions_2000)
```

Figure 21: Schiller P/E adjusted data

11.2 Graph of S&P with Schiller P/E data:

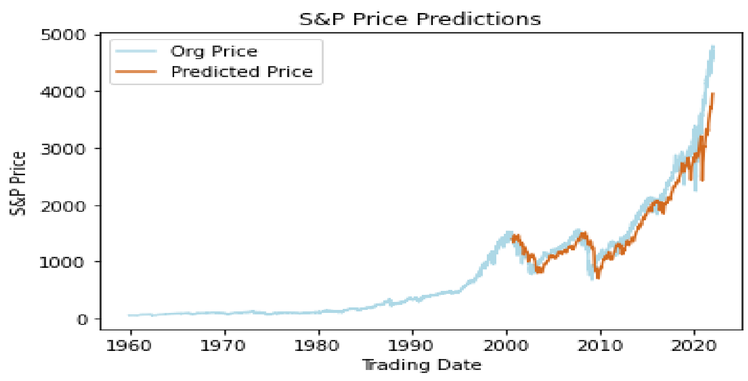


Figure 22: Graph of S&P 500 and Schiller P/E data