


Open in app 



[Sign up for Medium and get an extra one](#)



Faça login em Medium com o Google



Kaique Valim

kaiquevalim23@gmail.com



Cortes do Ilha

cdoilha@gmail.com

Mais 1 conta



Bee Guan Teo

[Follow](#)

Oct 25, 2021 · 7 min read ·  ·  Listen



Save



Stock Prices Prediction Using Long Short-Term Memory (LSTM) Model in Python



Photo by [Alesia Kozik](#) from [Pexels](#)

Long Short-Term Memory (LSTM) is one type of recurrent neural network which is used to learn order dependence in sequence prediction problems. Due to its capability of storing past information, it is very useful in predicting stock prices. This is because the prediction of a future stock price is dependent on the previous prices.

In this article, we will go through the steps to build a LSTM model to predict the stock prices in **Python**.

Disclaimer: The writing of this article is only aimed at demonstrating the steps to build a LSTM model to predict stock prices in Python. It doesn't serve any purpose of promoting any stock or giving any specific investment advice.

Prerequisite Python Packages

1. **yFinance** — <https://pypi.org/project/yfinance/>
2. **Numpy** — <https://numpy.org/>
3. **Matplotlib** — <https://matplotlib.org/>
4. **Pandas** — <https://pandas.pydata.org/>
5. **Scikit-Learn** — <https://scikit-learn.org/stable/>
6. **Tensorflow** — <https://www.tensorflow.org/>

Github

The original full source codes presented in this article are available on my [Github Repo](#). Feel free to download it (`stock_price_lstm.ipynb`) if you wish to use it to follow my article.

Stock Prices Prediction Using LSTM

1. Acquisition of Stock Data

Firstly, we are going to use *yFinance* to obtain the stock data. *yFinance* is an open-source Python library that allows us to acquire stock data from Yahoo Finance without any cost.

In this case, we are going to acquire the stock prices of AAPL over the last 5 years.

```

1  import math
2  import yfinance as yf
3  import numpy as np
4  import pandas as pd
5  from sklearn.preprocessing import MinMaxScaler
6  import matplotlib.pyplot as plt
7  import tensorflow as tf
8  from tensorflow import keras
9  from tensorflow.keras import layers
10
11 stock_data = yf.download('AAPL', start='2016-01-01', end='2021-10-01')
12 stock_data.head()

```

lstm_part_1.py hosted with ❤ by GitHub

[view raw](#)

Line 1–9: Import all the required libraries.

Line 11–12: Use the *yFinance* download method to acquire the stock data started from 1 Jan 2016 to 1 Oct 2021 and then preview the data.

	Open	High	Low	Close	Adj Close	Volume
Date						
2016-01-04	25.652500	26.342501	25.500000	26.337500	24.286825	270597600
2016-01-05	26.437500	26.462500	25.602501	25.677500	23.678219	223164000
2016-01-06	25.139999	25.592501	24.967501	25.174999	23.214840	273829600
2016-01-07	24.670000	25.032499	24.107500	24.112499	22.235073	324377600
2016-01-08	24.637501	24.777500	24.190001	24.240000	22.352640	283192000

Image Prepared by the Author

2. Visualizing Stock Prices History

Prior to preparing to build a LSTM model, let's take a look at the historical prices movement of AAPL by plotting a line chart.

```
1 plt.figure(figsize=(15, 8))
2 plt.title('Stock Prices History')
3 plt.plot(stock_data['Close'])
4 plt.xlabel('Date')
5 plt.ylabel('Prices ($)')
```

lstm_part_2.py hosted with ❤ by GitHub

[view raw](#)

Line 1–2: Set the plot figure size and title.

Line 3: Use the Matplotlib *plot* method to create a line chart for historical close prices of AAPL.

Line 4–5: Set the x-axis and y-axis labels.

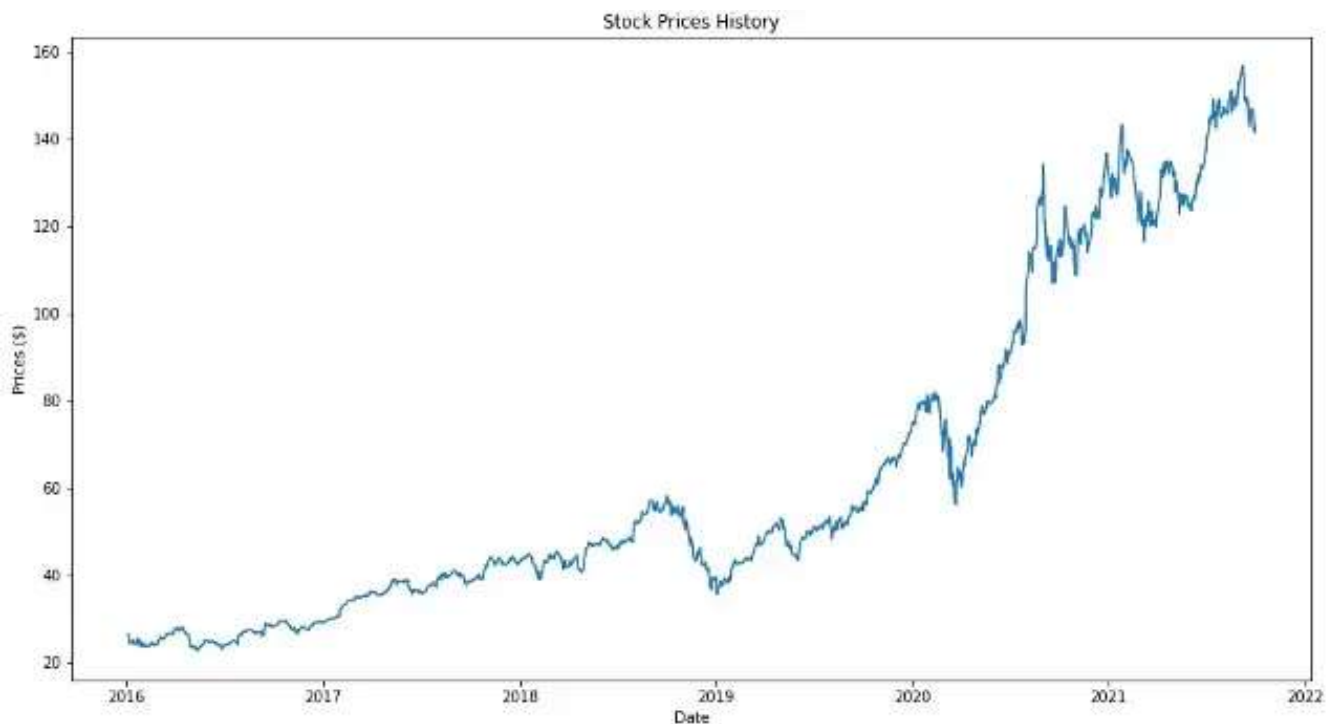


Image Prepared by the Author

The AAPL shows an upward trend over the last five years. This could be one of the potential assets worth our investment consideration.

3. Data Preprocessing

To build a LSTM model, we need to separate our stock prices data into a training set and a test set. Besides, we will also normalize our data so that all the values are ranged from 0 to 1.

3.1 Preparation of training set

Here we will only need the closing prices from our dataset to train our LSTM model. We are going to extract 80% of the closing prices from our acquired stock data as our training set.

```
1 close_prices = stock_data['Close']
2 values = close_prices.values
3 training_data_len = math.ceil(len(values)* 0.8)
4
5 scaler = MinMaxScaler(feature_range=(0,1))
6 scaled_data = scaler.fit_transform(values.reshape(-1,1))
7 train_data = scaled_data[0: training_data_len, :]
8
9 x_train = []
10 y_train = []
11
12 for i in range(60, len(train_data)):
13     x_train.append(train_data[i-60:i, 0])
14     y_train.append(train_data[i, 0])
15
16 x_train, y_train = np.array(x_train), np.array(y_train)
17 x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
```

lstm_part_3.py hosted with ❤ by GitHub

[view raw](#)

Line 1–2: Extract the closing prices from the acquired stock data and convert it to a number series.

Line 3: Calculate the data size for 80% of the dataset. The *math.ceil* method is to ensure the data size is rounded up to an integer.

Line 5–6: Use the *Scikit-Learn MinMaxScaler* to normalize all our stock data ranging from 0 to 1. We also reshape our normalized data into a two-dimensional array.

Line 7: Set apart the first 80% of the stock data as the training set.

Line 9–10: Create an empty list for a sequence of feature data (*x_train*) and a sequence of label data (*y_train*).

Line 12–14: Create a 60-days window of historical prices (*i-60*) as our feature data (*x_train*) and the following 60-days window as label data (*y_train*).

Line 16–17: Convert the feature data (*x_train*) and label data (*y_train*) into *Numpy array* as it is the data format accepted by the Tensorflow when training a neural

network model. Reshape again the x_{train} and y_{train} into a three-dimensional array as part of the requirement to train a LSTM model.

3.2 Preparation of test set

Next, we will proceed to prepare a test set.

```
1 test_data = scaled_data[training_data_len-60: , : ]
2 x_test = []
3 y_test = values[training_data_len:]
4
5 for i in range(60, len(test_data)):
6     x_test.append(test_data[i-60:i, 0])
7
8 x_test = np.array(x_test)
9 x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
```

lstm_part_4.py hosted with ❤ by GitHub

[view raw](#)

Line 1: Extract the closing prices from our normalized dataset (the last 20% of the dataset).

Line 2–6: Similar to the training set, we will have to create feature data (x_{test}) and label data (y_{test}) from our test set.

Line 8–9: Convert the feature data (x_{test}) and label data (y_{test}) into *Numpy array*. Reshape again the x_{test} and y_{test} into a three-dimensional array

3.3 Setting Up LSTM Network Architecture

Now, we are ready to use an open-source machine learning library, *Tensorflow*, to set up our LSTM network architecture.

```
1 model = keras.Sequential()
2 model.add(layers.LSTM(100, return_sequences=True, input_shape=(x_train.shape[1], 1)))
3 model.add(layers.LSTM(100, return_sequences=False))
4 model.add(layers.Dense(25))
5 model.add(layers.Dense(1))
6 model.summary()
```

lstm_part_5.py hosted with ❤ by GitHub

[view raw](#)

Line 1: Define a *Sequential* model which consists of a linear stack of layers.

Line 2: Add a *LSTM layer* by giving it 100 network units. Set the *return_sequence* to true so that the output of the layer will be another sequence of the same length.

Line 3: Add another LSTM layer with also 100 network units. But we set the *return_sequence* to *false* for this time to only return the last output in the output sequence.

Line 4: Add a densely connected neural network layer with 25 network units.

Line 5: At last, add a densely connected layer that specifies the output of 1 network unit.

Line 6: Show the summary of our LSTM network architecture.

```
Model: "sequential"
Layer (type)                 Output Shape              Param #
=====
lstm (LSTM)                   (None, 60, 100)          40800
lstm_1 (LSTM)                 (None, 100)              80400
dense (Dense)                 (None, 25)               2525
dense_1 (Dense)               (None, 1)                26
=====
Total params: 123,751
Trainable params: 123,751
Non-trainable params: 0
```

Image Prepared by the Author

3.4 Training LSTM Model

At this stage, we are almost ready to train our LSTM model by fitting it with the training set. Prior to that, we have to set an optimizer and a loss function for our model.

```
1 model.compile(optimizer='adam', loss='mean_squared_error')
2 model.fit(x_train, y_train, batch_size= 1, epochs=3)
```

lstm_part_6.py hosted with ❤ by GitHub

[view raw](#)

Line 1: Adopt “*adam*” optimizer and set the *mean square error* as loss function.

Line 2: Train the model by fitting it with the training set. We can try with `batch_size` of 1 and run the training for 3 epochs.

```
Epoch 1/3
1098/1098 [=====] - 36s 30ms/step - loss: 2.2327e-04
Epoch 2/3
1098/1098 [=====] - 33s 30ms/step - loss: 1.6478e-04
Epoch 3/3
1098/1098 [=====] - 33s 30ms/step - loss: 1.5542e-04
<keras.callbacks.History at 0x7f5e8c650>
```

Image Prepared by the Author

3.5 Model Evaluation

Our next task is to evaluate our trained LSTM model with the test set and then apply the **root mean square error (RMSE)** metric to examine the performance of the model.

```
1 predictions = model.predict(x_test)
2 predictions = scaler.inverse_transform(predictions)
3 rmse = np.sqrt(np.mean(predictions - y_test)**2)
4 rmse
```

lstm_part_7.py hosted with ❤ by GitHub

[view raw](#)

Line 1: Apply the model to predict the stock prices based on the test set.

Line 2: Use the *inverse_transform* method to denormalize the predicted stock prices.

Line 3–4: Apply the RMSE formula to calculate the degree of discrepancy between the predicted prices and real prices (*y_test*) and display the result.

```
1.7354735536146328
```

Image Prepared by the Author

The result shows that the RMSE is only as low as about 1.74. The model is seemingly working well.

3.6 Visualizing the Predicted Prices

It is always helpful to visualize the predicted prices in a graphical way. Here we are going to plot our predicted stock price and the real stock price using the *Python Matplotlib* again.


```
1 data = stock_data.filter(['Close'])
2 train = data[:training_data_len]
3 validation = data[training_data_len:]
4 validation['Predictions'] = predictions
5 plt.figure(figsize=(16,8))
6 plt.title('Model')
7 plt.xlabel('Date')
8 plt.ylabel('Close Price USD ($)')
9 plt.plot(train)
10 plt.plot(validation[['Close', 'Predictions']])
11 plt.legend(['Train', 'Val', 'Predictions'], loc='lower right')
12 plt.show()
```

lstm_part_8.py hosted with ❤ by GitHub

[view raw](#)

Line 1: Use the filter method to only retain the closing price column in the dataframe.

Line 2–4: Split our stock data into three plotting regions: training, validation and prediction.

Line 5–12: Configure the chart figure size, title, x-axis & y-axis label and legends.

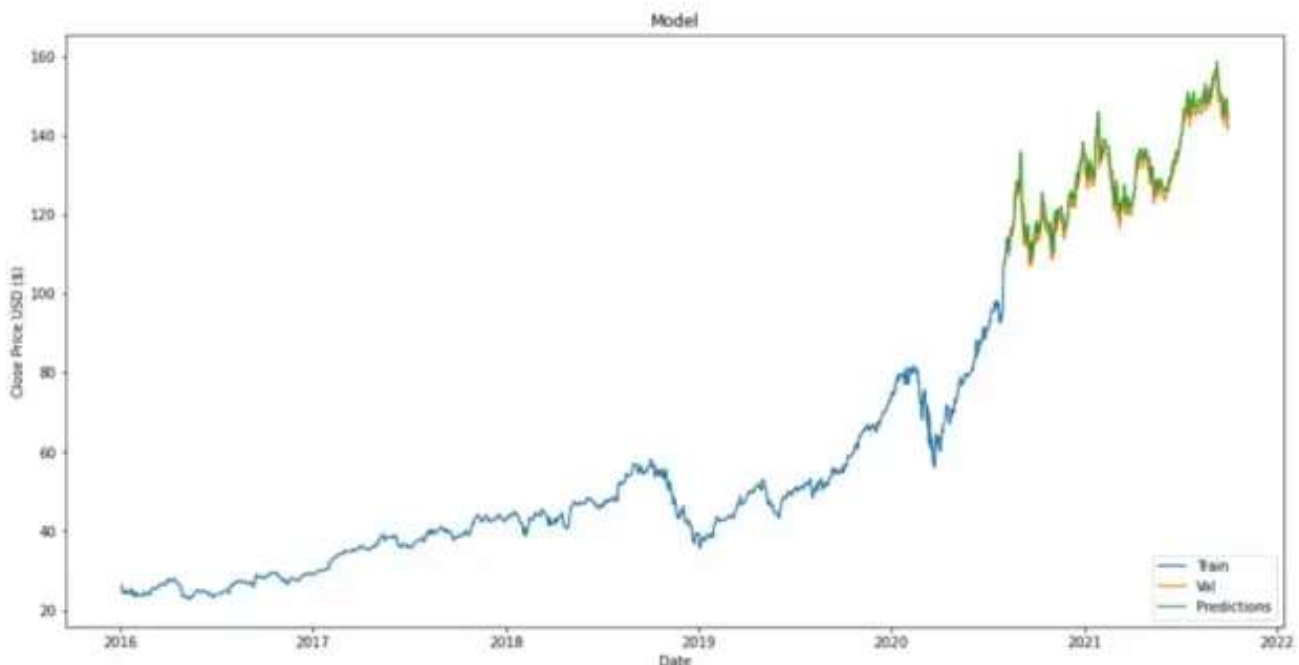


Image Prepared by the Author

From the resulting chart above, we can see the predicted stock prices follow the trend of the real stock prices closely. This shows the effectiveness of the LSTM to work with time series or sequential data like the stock prices.

Conclusions

LSTM can be another great tool for stock price prediction. However, this is important to note that the predicted stock prices shall not be used as a solely definitive guide to make an investment decision without further analysis. This is because the **prediction is only based on the historical prices movement that usually won't be the only factor that affects the future price movement.**

The main limitation of using any machine learning algorithm in predicting stock prices is that we can only backtest the historical data but the price movement does not necessarily follow the historical trend in various unforeseen circumstances. That's the reason a further fundamental/market analysis is required here to support our investment decision making.

I wish you enjoy reading this article.

Subscribe to Medium

If you like my article and would like to read more similar articles from me or other authors, feel free to subscribe to Medium. Your subscription fee will partially go to me. This can be a great support for me to produce more articles that can benefit the community.

References

1. https://en.wikipedia.org/wiki/Long_short-term_memory
2. <https://www.ibm.com/cloud/learn/recurrent-neural-networks>

Enjoy the read? Reward the writer.^{Beta}

Your tip will go to Bee Guan Teo through a third-party platform of their choice, letting them know you appreciate their story.

Give a tip

Sign up for The Handbook of Coding in Finance

By The Handbook of Coding in Finance

Everything about financial data analytics and machine learning. [Take a look.](#)

Your email



Get this newsletter

By signing up, you will create a Medium account if you don't already have one. Review our [Privacy Policy](#) for more information about our privacy practices.

[About](#) [Help](#) [Terms](#) [Privacy](#)

Get the Medium app

