**Stock price prediction using LSTM**

# **Introduction**

Neural Networks have been demonstrated to be very powerful in solving the real-world problems like imaging and natural language processing. Long Short Term Memory(LSTM) is a special type of Recurrent Neural Network(RNN) which can retain important information over time using memory cells.

This property of LSTMs makes it a wonderful algorithm to learn sequences that are interdependent and can help to build solutions like language translation, sales time series, chatbots, autocorrections, next word suggestions, etc.

Interested in stock price movements, I am therefore motivated and curious about employing nerual nets to approach financial time series problems of its kind.

I will scrape Apple price data from Yahoo Finance using yfinance API.

The data is from **2018-01-01** to **2022-12-31.**

# **Data**

Datasets consist of following columns:

* **Date** Time stamp of when data was collected, being used as index column;
* **Volume** The number of shares traded in AAPL during a given trading day;
* **Open** The price at AAPL upon the opening of an exchange on a given trading day;
* **High** The highest price at which AAPL traded during the course of the day;
* **Low** The lowest price at which AAPL traded during the course of the day;
* **Close** The final price at which AAPL traded during the course of the day; it is also selected as our target data.

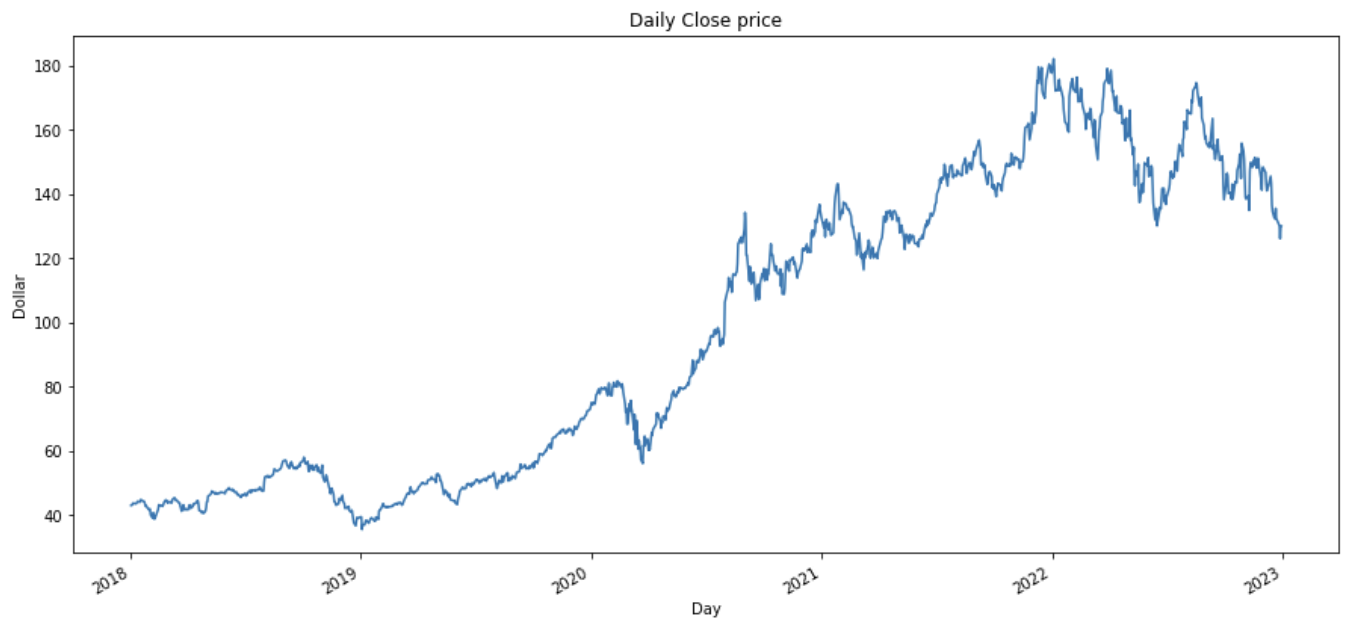
Plus, add a new feature based on the columns above:

* **pct** The percentage of change of close price compared to previous trading day.





**Visualize the five-year daily Close price**



# **Data Preprocessing**

The LSTM model will need data input in the form of X vs y. Where the X will represent the last 60 day’s prices and y will represent the 61th-day price.

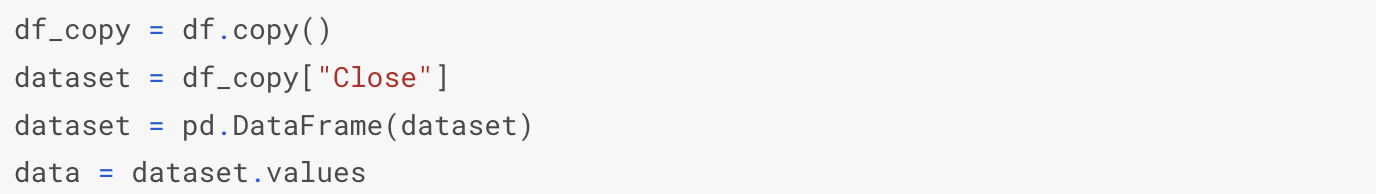
By looking at a lot of such examples from the past 5 years, the LSTM will be able to learn the movement of prices. Hence, when we pass the last 60 days of the price it will be able to predict tomorrow’s stock close price.

Since LSTM is a Neural network-based algorithm, standardizing or normalizing the data is mandatory for a fast and more accurate fit.

## **1. Scale features**

The input and output data that go into model need to be scaled.

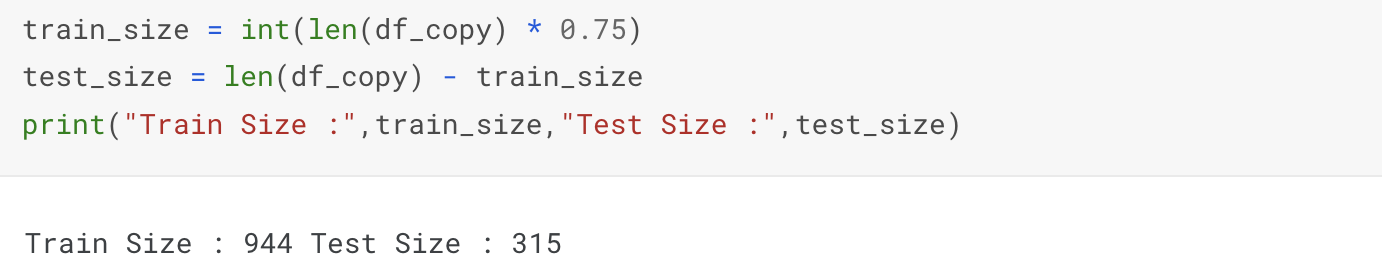
Use preprocessing.MinMaxScaler() function in scikit-learn library to scale data.





## **2. Split data to train and test set**

Split data into training (75%) and test sets (25%).





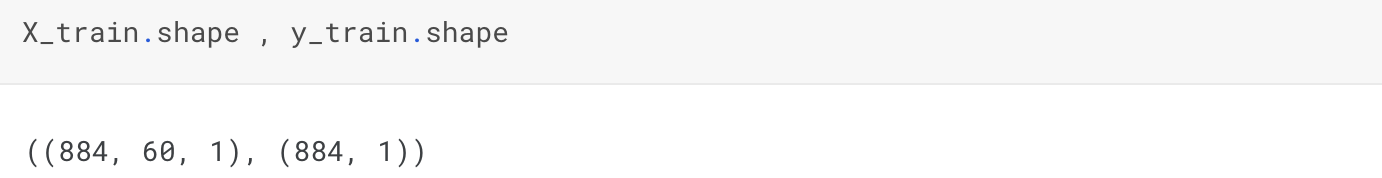


## **3. Convert to Numpy array**



## **4. Reshaping the input to (n\_samples, time\_steps, n\_feature)**





# **Model LSTM**

Look at the use of the LSTM function instead of Dense to define the hidden layers. The output layer has one neuron as we are predicting the next day price, if we want to predict for multiple days, then change the input data and neurons equal to the number of days of forecast.

In the below code snippet I have used three hidden LSTM layers and one output layer. We can choose more layers if we don’t get accuracy for your data. Similarly we can increase or decrease the number of neurons in the hidden layer.

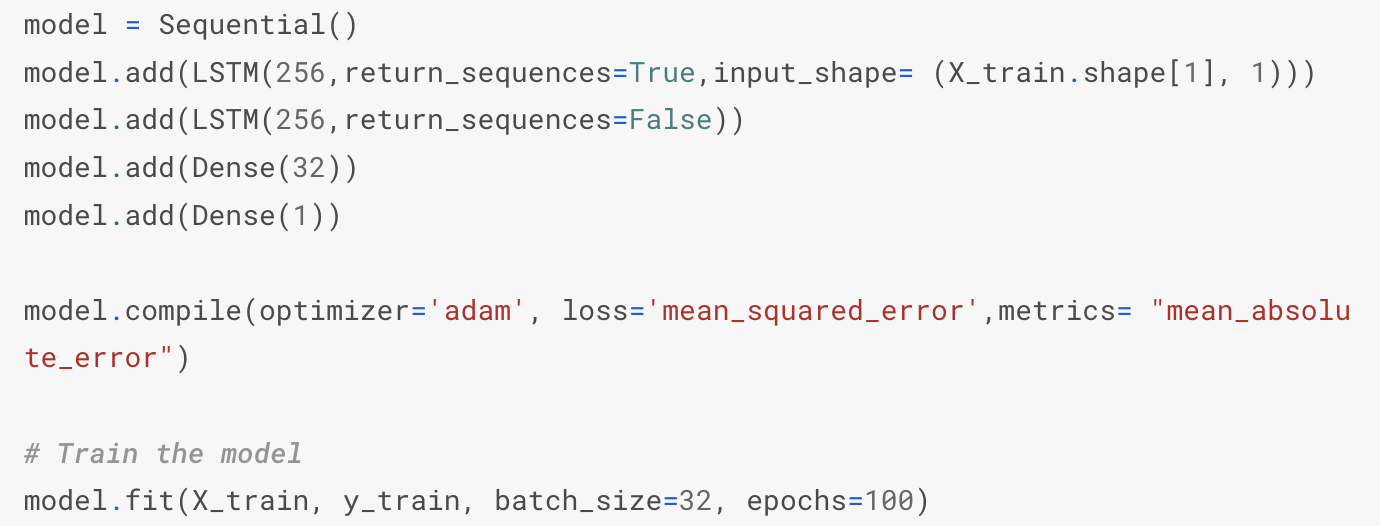
Just keep in mind, the more neurons and layers you use, the slower the model becomes. Because now there are many more computations to be done.

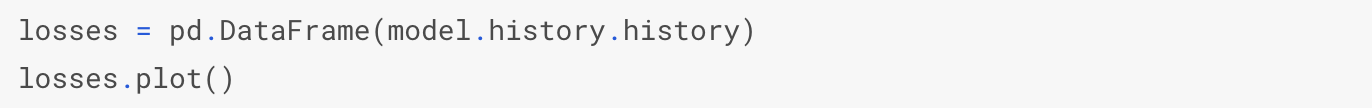
Each layer has some hyperparameters which needs to be tuned.

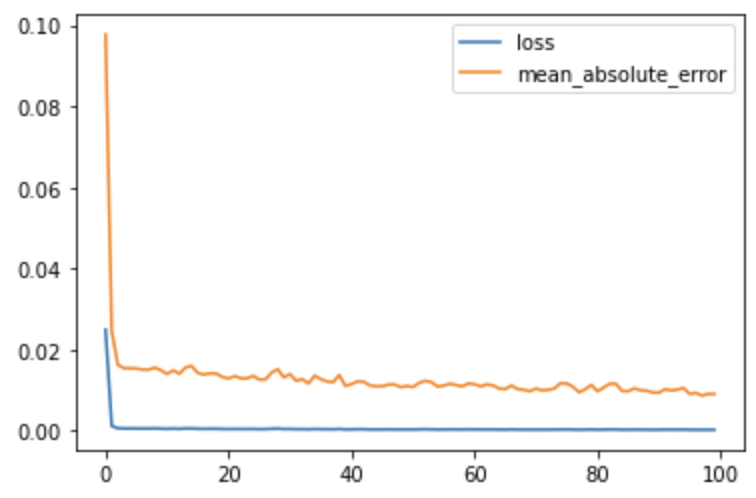
Take a look at some of the important hyperparameters of LSTM below :

* **units=256**: This means we are creating a layer with 256 neurons in it.
* **input\_shape = (TimeSteps, TotalFeatures)**: The input expected by LSTM is in 3D format. Our training data has a shape of (884, 60, 1) this is in the form of (number of samples, time steps, number of features). This means we have 884 examples to learn in training data, each example looks back 60-steps in time like what was the stock price yesterday, the day before yesterday so on till last 60 days. This is known as Time steps. The last number ‘1’ represents the number of features. Here we are using just one column ‘Closing Stock Price’ hence its equal to ‘1’
* **kernel\_initializer=’uniform’**: When the Neurons start their computation, some algorithm has to decide the value for each weight. This parameter specifies that. You can choose different values for it like ‘normal’ or ‘glorot\_uniform’.
* **activation=’relu’**: This specifies the activation function for the calculations inside each neuron. You can choose values like ‘relu’, ‘tanh’, ‘sigmoid’, etc.
* **return\_sequences=True:**LSTMs backpropagate thru time, hence they return the values of the output from each time step to the next hidden layer. This keeps the expected input of the next hidden layer in the 3D format. This parameter is False for the last hidden layer because now it does not have to return a 3D output to the final Dense layer.
* **optimizer=’adam’:**This parameter helps to find the optimum values of each weight in the neural network. ‘adam’ is one of the most useful optimizers, another one is ‘rmsprop’
* **batch\_size=32**: This specifies how many rows will be passed to the Network in one go after which the SSE calculation will begin and the neural network will start adjusting its weights based on the errors.  
  When all the rows are passed in the batches of 32 rows each as specified in this parameter, then we call that 1-epoch. Or one full data cycle. A small value of batch\_size will make the LSTM look at the data slowly, like 2 rows at a time or 4 rows at a time which could lead to overfitting, as compared to a large value like 20 or 50 rows at a time, which will make the LSTM look at the data fast which could lead to underfitting. Hence a proper value must be chosen using hyperparameter tuning.
* **Epochs=100**: The same activity of adjusting weights continues for 100 times, as specified by this parameter. In simple terms, the LSTM looks at the full training data 100 times and adjusts its weights.

**1.Create Model LSTM**

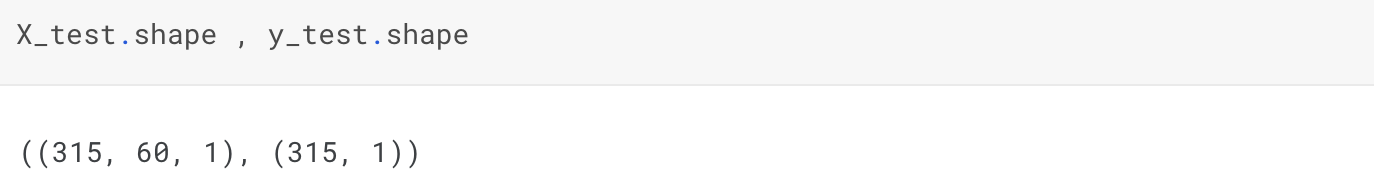






## **2. Creating a testing set with 60 time-steps and 1 output**

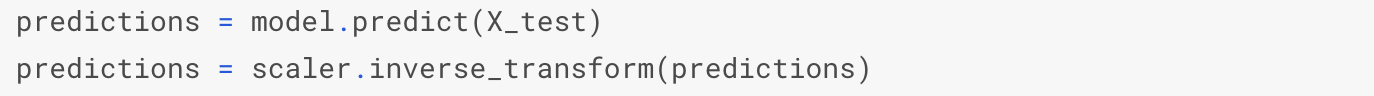




## **3. Prediction**

Now we use the trained model, we are checking if the predicted prices are close to the actual prices or not.

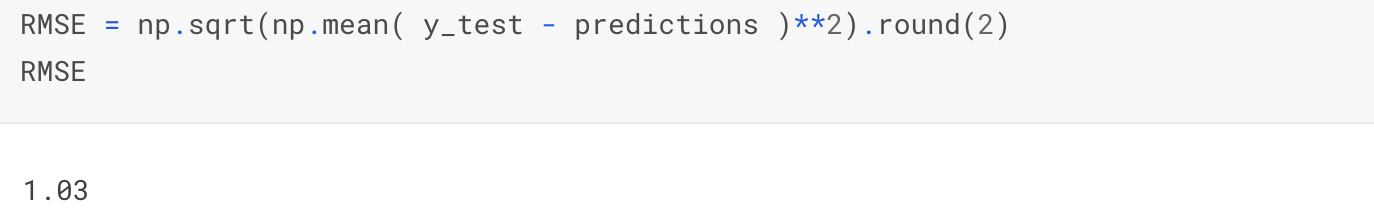
Notice the inverse transform of the predictions. Since we normalized the data before the model training, the predictions on testing data will also be normalized, hence the inverse transformation will bring the values to the original scale.



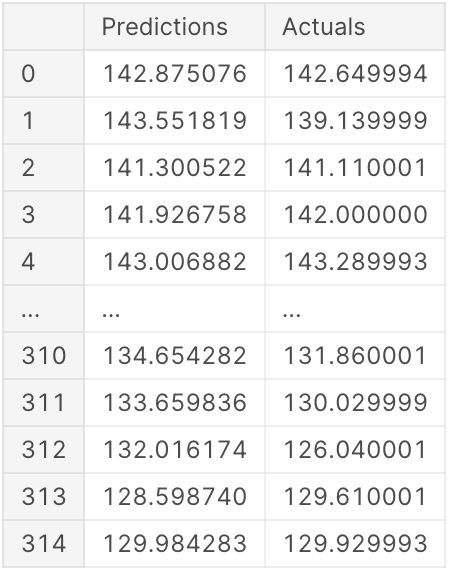


## **4. Model Evaluation**

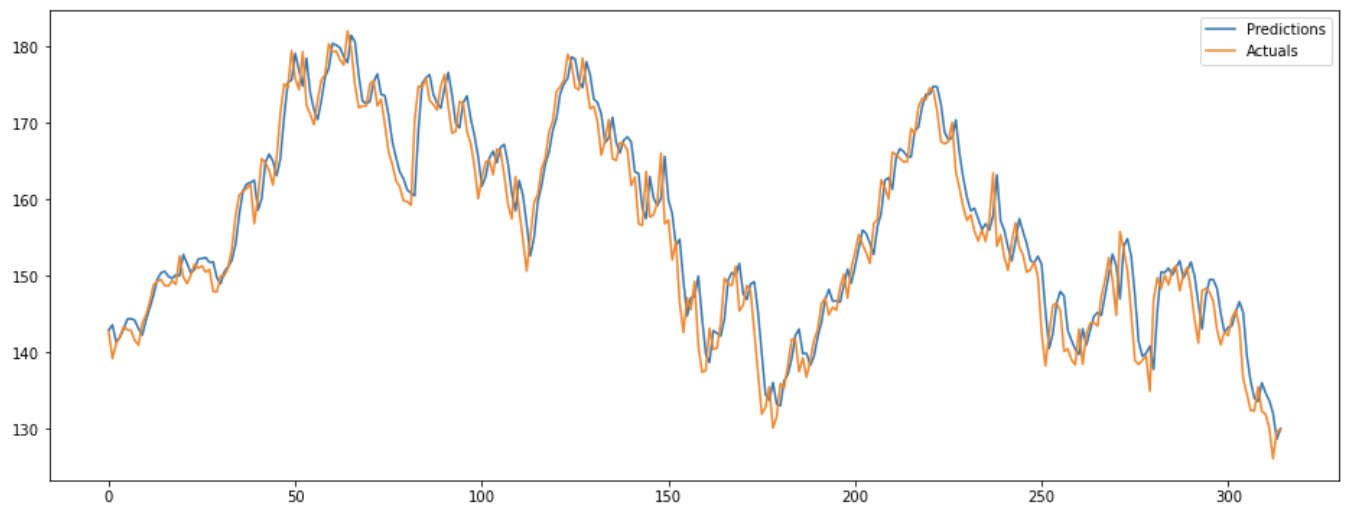
Get the root mean squared error (RMSE)



## **5. Compare the Actual values and Prediction values**



## **6. Plot the Actual values and Prediction values**



## **7. Plot Train data, Test data and Prediction values**

The model shows that LSTM can capture the pattern of Apple’s stock prices. However, there are some gaps between predicted and true movements. We can certainly improve the performance by further tuning the model, but the assumption of the stable status in stock market mechanism makes it hard to get significantly better results.

Since investor behavior is highly impacted by various factors, the future idea is to add sentiment analysis to better understand investors' pyschology. Therefore, we can capture more price catalysts in the market and achieve a better prediction with a holistic view.

