In this Colab, we will use a keras Long Short-Term Memory (LSTM) model to predict the stock price of Tata Global Beverages

Here are some imports we need to make: numpy for scientific computation, matplotlib for graphing, and pandas for manipulating data.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Load training data set with the "Open" and "High" columns to use in our modeling.

```
url = 'https://raw.githubusercontent.com/mwitiderrick/stockprice/master/NSE-TATAGLOBAL.csv'
dataset_train = pd.read_csv(url)
training_set = dataset_train.iloc[:, 1:2].values
```

Let's take a look at the first five rows of our dataset

```
dataset_train.head()
```

	Date	Open	High	Low	Last	Close	Total Trade Quantity	Turnover (Lacs)
0	2018-09-28	234.05	235.95	230.20	233.50	233.75	3069914	7162.35
1	2018-09-27	234.55	236.80	231.10	233.80	233.25	5082859	11859.95
2	2018-09-26	240.00	240.00	232.50	235.00	234.25	2240909	5248.60
3	2018-09-25	233.30	236.75	232.00	236.25	236.10	2349368	5503.90
4	2018-09-24	233.55	239.20	230.75	234.00	233.30	3423509	7999.55

Import MinMaxScaler from scikit-learn to scale our dataset into numbers between 0 and 1

```
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range=(0,1))
training_set_scaled = sc.fit_transform(training_set)
```

We want our data to be in the form of a 3D array for our LSTM model. First, we create data in 60 timesteps and convert it into an array using NumPy. Then, we convert the data into a 3D array with X_train samples, 60 timestamps, and one feature at each step.

```
X_train = []
y_train = []
for i in range(60, 2035):
    X_train.append(training_set_scaled[i-60:i, 0])
    y_train.append(training_set_scaled[i, 0])
X_train, y_train = np.array(X_train), np.array(y_train)

X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
```

Make the necessary imports from keras

```
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dropout
from keras.layers import Dense
```

Add LSTM layer along with dropout layers to prevent overfitting. After that, we add a Dense layer that specifies a one unit output. Next, we compile the model using the adam optimizer and set the loss as the mean_squarred_error

```
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(optimizer='adam',loss='mean_squared_error')

model.fit(X_train,y_train,epochs=100,batch_size=32)
```

```
62/62 [==============================] - 6s 102ms/step - loss: 8.6388e-04
Epoch 73/100
62/62 [==============================] - 6s 93ms/step - loss: 7.2068e-04
Epoch 74/100
62/62 [==============================] - 6s 102ms/step - loss: 8.7690e-04
Epoch 75/100
62/62 [==============================] - 6s 95ms/step - loss: 6.9299e-04
Epoch 76/100
62/62 [==============================] - 10s 158ms/step - loss: 7.2301e-04
Epoch 77/100
62/62 [==============================] - 7s 106ms/step - loss: 6.6359e-04
Epoch 78/100
62/62 [==============================] - 6s 96ms/step - loss: 6.4085e-04
Epoch 79/100
62/62 [==============================] - 6s 97ms/step - loss: 7.6720e-04
Epoch 80/100
62/62 [==============================] - 6s 93ms/step - loss: 7.3517e-04
Epoch 81/100
62/62 [==============================] - 6s 102ms/step - loss: 7.5006e-04
Epoch 82/100
62/62 [==============================] - 6s 97ms/step - loss: 6.1277e-04
Epoch 83/100
62/62 [==============================] - 6s 102ms/step - loss: 8.9060e-04
Epoch 84/100
62/62 [==============================] - 6s 93ms/step - loss: 7.4062e-04
Epoch 85/100
62/62 [==============================] - 6s 98ms/step - loss: 7.9677e-04
Epoch 86/100
62/62 [==============================] - 6s 89ms/step - loss: 8.0788e-04
Epoch 87/100
62/62 [==============================] - 6s 99ms/step - loss: 7.1714e-04
Epoch 88/100
62/62 [==============================] - 6s 94ms/step - loss: 6.6731e-04
Epoch 89/100
62/62 [==============================] - 6s 105ms/step - loss: 0.0010
Epoch 90/100
62/62 [==============================] - 6s 102ms/step - loss: 7.4019e-04
Epoch 91/100
62/62 [==============================] - 6s 102ms/step - loss: 6.6945e-04
Epoch 92/100
62/62 [==============================] - 6s 90ms/step - loss: 7.2526e-04
Epoch 93/100
62/62 [==============================] - 6s 101ms/step - loss: 7.1585e-04
Epoch 94/100
62/62 [==============================] - 6s 95ms/step - loss: 5.9162e-04
Epoch 95/100
62/62 [==============================] - 6s 104ms/step - loss: 5.8136e-04
Epoch 96/100
62/62 [==============================] - 6s 98ms/step - loss: 6.0083e-04
Epoch 97/100
62/62 [==============================] - 7s 109ms/step - loss: 6.5794e-04
Epoch 98/100
62/62 [==============================] - 6s 95ms/step - loss: 6.6987e-04
Epoch 99/100
62/62 [==============================] - 6s 96ms/step - loss: 7.1517e-04
Epoch 100/100
62/62 [==============================] - 6s 94ms/step - loss: 6.2353e-04
<keras.callbacks.History at 0x78b375636fb0>
```

Import the test set for the model to make predictions on

```
url = 'https://raw.githubusercontent.com/mwitiderrick/stockprice/master/tatatest.csv'
dataset_test = pd.read_csv(url)
real_stock_price = dataset_test.iloc[:, 1:2].values
```

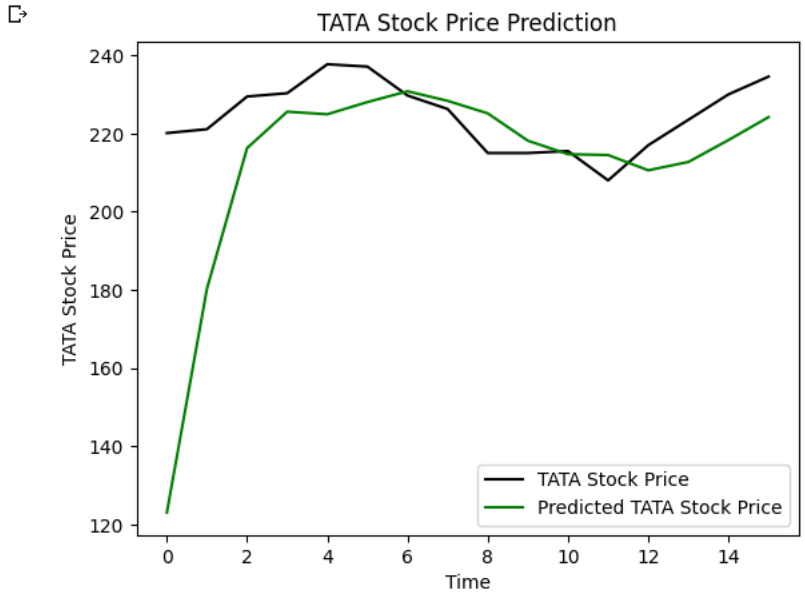
Before predicting future stock prices, we have to manipulate the training set; we merge the training set and the test set on the 0 axis, set the time step to 60, use minmaxscaler, and reshape the dataset as done previously. After making predictions, we use inverse_transform to get back the stock prices in normal readable format.

```
dataset_total = pd.concat((dataset_train['Open'], dataset_test['Open']), axis = 0)
inputs = dataset_total[len(dataset_total) - len(dataset_test) - 60:].values
inputs = inputs.reshape(-1,1)
inputs = sc.transform(inputs)
X_test = []
for i in range(60, 76):
    X_test.append(inputs[i-60:i, 0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
predicted_stock_price = model.predict(X_test)
predicted_stock_price = sc.inverse_transform(predicted_stock_price)
```

```
1/1 [==============================] - 2s 2s/step
```

Plot our predicted stock prices and the actual stock price

```
plt.plot(real_stock_price, color = 'black', label = 'TATA Stock Price')
plt.plot(predicted_stock_price, color = 'green', label = 'Predicted TATA Stock Price')
plt.title('TATA Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('TATA Stock Price')
plt.legend()
plt.show()
```



✓ 0s completed at 23:38