« **Importing libraries**

```
[1] import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    from sklearn.preprocessing import MinMaxScaler
    from keras.models import Sequential, load_model
    from keras.layers import LSTM, Dense, Dropout
```

We used numpy for scientific operations, pandas to modify our dataset, matplotlib to visualize the results, sklearn to scale our data, and keras to work as a wrapper on low-level libraries like TensorFlow.

« **Uploading the data**

```
[2] from google.colab import files
    dataset = files.upload()
```

Choose Files AAPL.csv

- **AAPL.csv**(application/vnd.ms-excel) - 90346 bytes, last modified: 10/11/2021 - 100% done

```
Saving AAPL.csv to AAPL.csv
```

The files that are to be uploaded were downloaded from yahoo finance. We took the Apple company stock prices from 10/11/2016 to 10/8/2021.

« **Reading the data**

```
[10] df = pd.read_csv('AAPL.csv')
     df.head()
```

| | Date | Open | High | Low | Close | Adj Close | Volume |
|---|---|---|---|---|---|---|---|
| **0** | 2016-10-11 | 29.424999 | 29.672501 | 29.049999 | 29.075001 | 27.267719 | 256164000 |
| **1** | 2016-10-12 | 29.337500 | 29.495001 | 29.187500 | 29.334999 | 27.511553 | 150347200 |
| **2** | 2016-10-13 | 29.197500 | 29.360001 | 28.930000 | 29.245001 | 27.427147 | 140769600 |
| **3** | 2016-10-14 | 29.469999 | 29.542500 | 29.282499 | 29.407499 | 27.579540 | 142608800 |
| **4** | 2016-10-17 | 29.332500 | 29.459999 | 29.195000 | 29.387501 | 27.560787 | 94499600 |

```
[12] df.shape

(1258, 7)
```

We read the csv file into a data frame. The data frame has 1258 rows and 7 columns. The data contains the stock prices for 1258 days.

```
[5]  df = df['Close'].values
     df = df.reshape(-1, 1)
     df.shape

(1258, 1)
```

To make it as simple as possible, we used just one variable which is the "Close" price.

« **Splitting the data into Training and Testing Sets**

```
[6] dataset_train = np.array(df[:int(df.shape[0]*0.8)])
    dataset_test = np.array(df[int(df.shape[0]*0.8):])
    print(dataset_train.shape)
    print(dataset_test.shape)

    (1006, 1)
    (252, 1)
```

The data was split as 80% for training and 20% for testing. So, out of the 1258 rows 1006 are for training and 252 for tresting.

« **Scaling the data**

```
[8] scaler = MinMaxScaler(feature_range=(0,1))
    dataset_train = scaler.fit_transform(dataset_train)
    dataset_test = scaler.transform(dataset_test)
```

We used the MinMaxScaler to scale our data between zero and one.

« **Function to create the datasets**

```
[12] def create_dataset(df):
         x = []
         y = []
         for i in range(50, df.shape[0]):
             x.append(df[i-50:i, 0])
             y.append(df[i, 0])
         x = np.array(x)
         y = np.array(y)
         return x,y
```

« **Creating the training and testing data sets**

```
[31] x_test, y_test = create_dataset(dataset_test)
     x_train, y_train = create_dataset(dataset_train)
     print(x_test.shape,y_test.shape,x_train.shape,y_train.shape,df.shape)

(202, 50) (202,) (956, 50) (956,) (1258, 1)
```

```
[39] x_test

array([[0.84028219, 0.90923652, 0.87861074, ..., 0.94088308, 0.94914275,
        0.93021053],
       [0.90923652, 0.87861074, 0.87944603, ..., 0.94914275, 0.93021053,
        0.94478089],
       [0.87861074, 0.87944603, 0.87499135, ..., 0.93021053, 0.94478089,
        0.9786549 ],
       ...,
       [1.11684195, 1.10022973, 1.1063549 , ..., 1.07860608, 1.04603147,
        1.06431413],
       [1.10022973, 1.1063549 , 1.10839664, ..., 1.04603147, 1.06431413,
        1.07257379],
       [1.1063549 , 1.10839664, 1.10524129, ..., 1.06431413, 1.07257379,
        1.08454561]])
```

We created the testing and training data sets by calling the function for each one. We are taking the timestep as 50. So, the model takes the closing prices from the last 50 days and predicts the closing price for the next day.

First we call the function for the testing data. Basically, we are creating an array of 202 rows and 50 columns, and storing it in x_test. So in the first row of the array x_test we have the closing price for the first 50 days.In the next row, we have the closing price for 50 days, leaving out the first value of the previous row, and so on for the remaining 200 days.

The model looks at the closing price for 50 days and predicts the closing price for the 51st day. This value is stored in the arrray y_test (i.e the value that the model is supposed to predict, is stored in the array y_test). y_test is an array of

202 rows.
The same is done for the training data as well.

« **Reshaping the data**

```
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
print(x_test.shape,x_train.shape)

(202, 50, 1) (956, 50, 1)
```

We then reshaped the data to make it a 3D array in order to use it in LSTM Layer.

« **Building the Model**

```
model = Sequential()
model.add(LSTM(units=96, return_sequences=True, input_shape=(x_train.shape[1], 1)))
model.add(Dropout(0.2))
model.add(LSTM(units=96, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(units=96, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(units=96))
model.add(Dropout(0.2))
model.add(Dense(units=1))
```

We created a stacked LSTM model.

« **Compiling the Model**

```
model.compile(loss='mean_squared_error', optimizer='adam')
```

We used loss='mean squared error' because it is a regression problem, and the adam optimizer to update network weights iteratively based on training data.

« **Training**

```
model.fit(x_train, y_train, epochs=50, batch_size=32)
model.save('stock_prediction.h5')
```

```
Epoch 1/50
30/30 [==============================] - 12s 164ms/step - loss: 0.0142
Epoch 2/50
30/30 [==============================] - 5s 162ms/step - loss: 0.0027
Epoch 3/50
30/30 [==============================] - 5s 163ms/step - loss: 0.0023
Epoch 4/50
30/30 [==============================] - 5s 163ms/step - loss: 0.0021
Epoch 5/50
30/30 [==============================] - 5s 163ms/step - loss: 0.0023
Epoch 6/50
30/30 [==============================] - 5s 165ms/step - loss: 0.0027
Epoch 7/50
30/30 [==============================] - 5s 163ms/step - loss: 0.0018
Epoch 8/50
30/30 [==============================] - 5s 163ms/step - loss: 0.0017
Epoch 9/50
30/30 [==============================] - 5s 163ms/step - loss: 0.0016
Epoch 10/50
30/30 [==============================] - 5s 163ms/step - loss: 0.0015
Epoch 11/50
30/30 [==============================] - 5s 164ms/step - loss: 0.0014
Epoch 12/50
30/30 [==============================] - 5s 163ms/step - loss: 0.0016
Epoch 13/50
30/30 [==============================] - 5s 165ms/step - loss: 0.0015
Epoch 14/50
30/30 [==============================] - 5s 166ms/step - loss: 0.0028
Epoch 15/50
30/30 [==============================] - 5s 163ms/step - loss: 0.0021
```

We used 50 epochs and set the batch size to 32.

« **Prediction**

```
[49] predictions = model.predict(x_test)
```

## « Reverse scaling

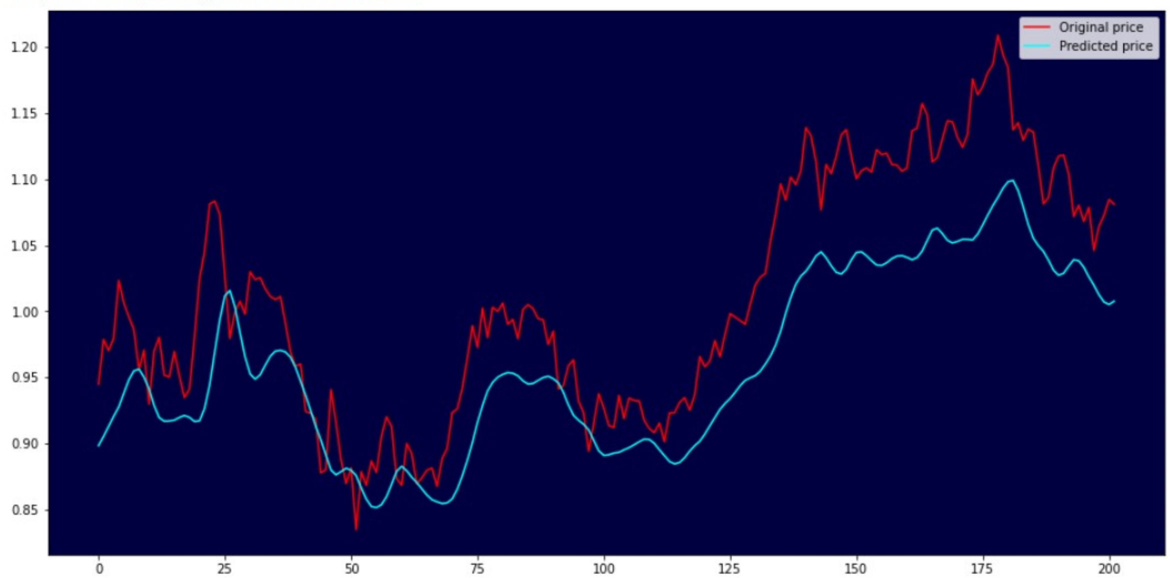
```python
predictions = scaler.inverse_transform(predictions)
y_test_scaled = scaler.inverse_transform(y_test.reshape(-1, 1))
```

We had scaled the data between 0 and 1. Therefore, we had to do the reverse scaling.

## « Visualising the results

```python
fig, ax = plt.subplots(figsize=(16,8))
ax.set_facecolor('#000041')
ax.plot(y_test_scaled, color='red', label='Original price')
plt.plot(predictions, color='cyan', label='Predicted price')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7efc3e9d9410>
```




We then plotted the results. The predicted price which is stored in predictions is plotted in blue, and the actual price which is stored in y_test_scaled is plotted in red.