

Stock Price Prediction Using Time Series Analysis

Machine Learning Project Report
April 8th, 2022

TABLE OF CONTENT

S. No	TOPICS	Page No.
1	INTRODUCTION	03
2	DATASET	03
3	DATA PREPROCESSING	04
4	MOVING AVERAGES	05
4	FEATURE SCALING	06
5	MODEL IMPLEMENTATION	06
6	LSTM MODEL METHODOLOGY	07
7	FUTURE PREDICTION	10
8	LIMITATIONS AND CONCLUSION	11

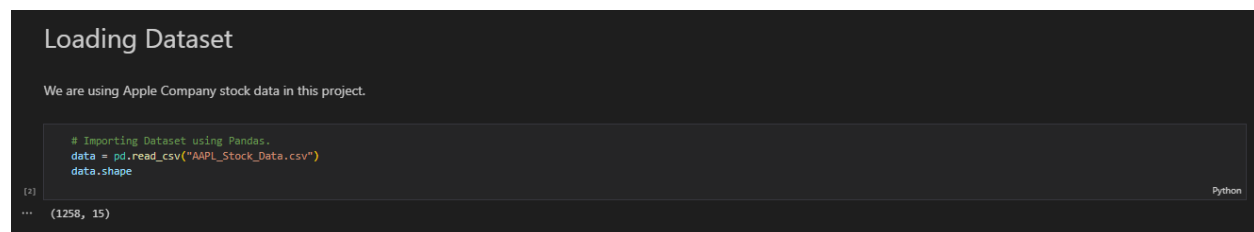
Introduction

The inherent volatility and unpredictability of financial markets make it difficult to forecast stock prices. But, with the development of cutting-edge technology and methodologies, including machine learning and statistical modelling, we may use past stock price data to construct prediction models that might aid in making wise investing selections. To examine historical stock price data and predict future stock prices, our study will investigate several techniques, including time-series analysis, statistical modelling, and machine learning algorithms. To better understand the dynamics of financial markets, we hope to use the power of data-driven methodologies to sift through stock price data to find patterns, trends, and potential signals.

A timeseries dataset of Apple Inc. stock prices will be used in this project to develop a predictive model using LSTM, a type of recurrent neural network (RNN). Due to the complexity and volatility of financial markets, predicting stock prices is a difficult endeavor. To capture temporal dependencies and patterns in timeseries data, LSTM, a potent deep learning approach, has however demonstrated promising results. We seek to forecast Apple's stock prices using an LSTM model trained and tuned with historical stock price data, which could offer insightful information to traders, investors, and financial analysts.

About Dataset

The data set we are using in this project is basically scrapped from yahoo.com. The data set basically contains information of apple company stock prices from the period of 2010 to 2019. We have around 1250 records and 15 features. The most important feature in the data set is basically the closing price of the stock and the time column.



```
Loading Dataset

We are using Apple Company stock data in this project.

# Importing Dataset using Pandas.
data = pd.read_csv("AAPL_Stock_Data.csv")
data.shape

[2]:
... (1258, 15)

Python
```

Data Exploration and Pre-Processing

Resetting the index of a dataset is a standard practice in time series projects to guarantee that the data is arranged sequentially with a consistent frequency, which is frequently required for time-based analytics and modelling.

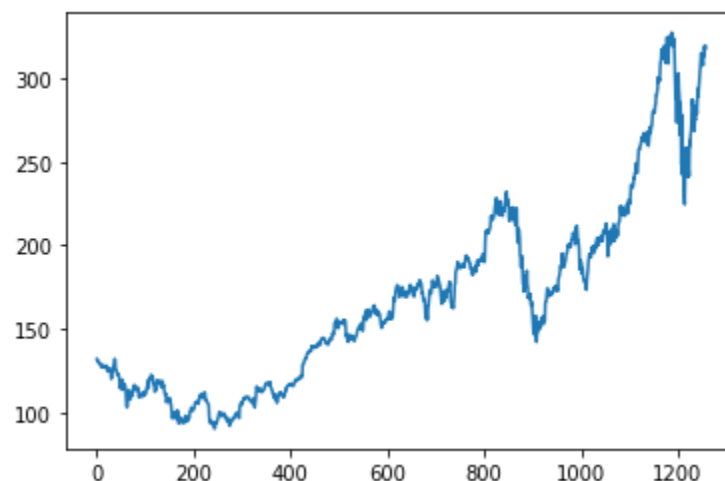
A timestamp or date column frequently acts as the index in time series data. The data might not, however, be evenly spaced in terms of time periods. You can make sure that the data is in sync with a regular time interval, such as daily, monthly, or yearly, by resetting the index. As a result, it may be easier to do time-based analysis including computing rolling averages, spotting trends, and analyzing seasonality patterns.

```
data = data.reset_index()
data.head()
```

index	Unnamed: 0	symbol	date	close	high	low	open	volume	adjClose	adjHigh	adjLow	adjOpen	adjVolume	divCash	splitfactor
0	0	AAPL	2015-05-27 00:00:00+00:00	132.045	132.260	130.05	130.34	45833246	121.682558	121.880685	119.844118	120.111360	45833246	0.0	1.0
1	1	AAPL	2015-05-28 00:00:00+00:00	131.780	131.950	131.10	131.86	30733309	121.438354	121.595013	120.811718	121.512076	30733309	0.0	1.0
2	2	AAPL	2015-05-29 00:00:00+00:00	130.280	131.450	129.90	131.23	50884452	120.056069	121.134251	119.705890	120.931516	50884452	0.0	1.0
3	3	AAPL	2015-06-01 00:00:00+00:00	130.535	131.390	130.05	131.20	32112797	120.291057	121.078960	119.844118	120.903870	32112797	0.0	1.0
4	4	AAPL	2015-06-02 00:00:00+00:00	129.960	130.655	129.32	129.86	33667627	119.761181	120.401640	119.171406	119.669029	33667627	0.0	1.0

Now we can drop unnamed and date columns from the dataset, as we have reset the index and don't need these columns anymore.

Now we are plotting a graph of closing value of the graph to check out the overall trend of closing price of the stock throughout the data.

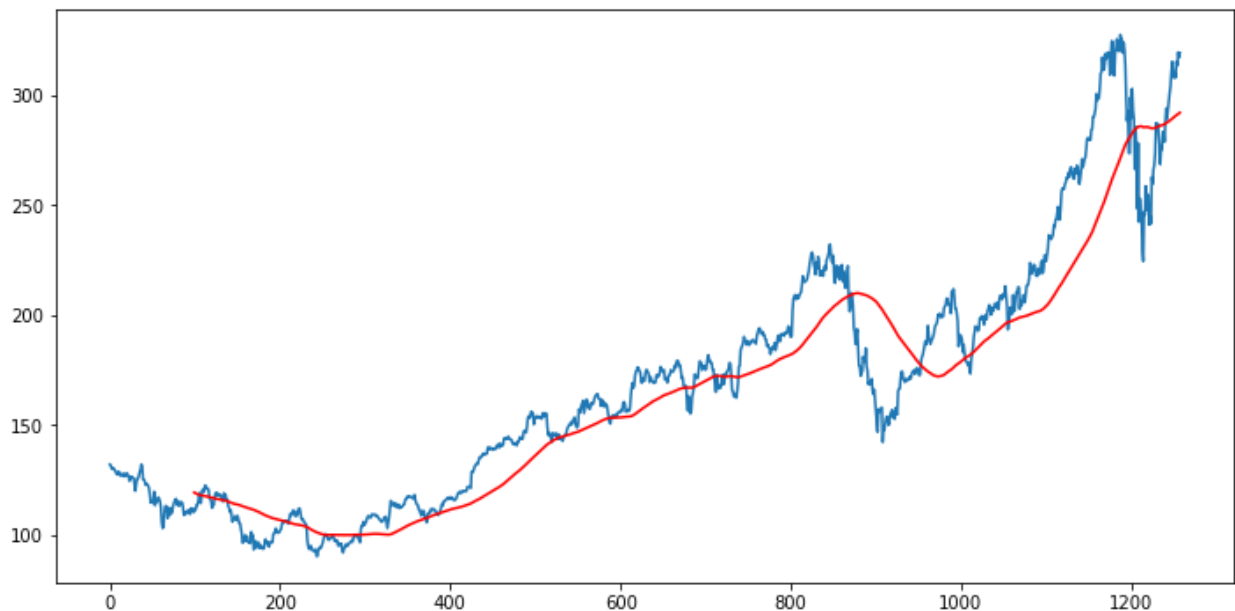


Moving Averages

In time series analysis, moving averages are a frequently employed statistical tool that can provide light on the underlying trends and patterns in the data. They are determined by averaging a predetermined number of data points obtained through a sliding window that "moves" through the time series data.

Moving averages can aid in the detection of trends in time series data by reducing the impact of transient variations. You may determine a trend's direction and strength by calculating moving averages over various window widths.

We are also calculating the 100 days moving average for our data. Which is plotted as



Train – Test Split

The train-test split procedure is used to estimate the performance of machine learning algorithms when they are used to make predictions on data not used to train the model. Here we are splitting our dataset into train dataset and test dataset with the ratio of 70-30% where 70% is our train dataset and 30 is our test dataset.

Train - Test Split

```
[12] # Splitting Data into Train - Test Set

training_data = pd.DataFrame(data['close'][0:int(len(data)*0.70)]) # Creating training dataset with 70% of total values starting from 0.
testing_data = pd.DataFrame(data['close'][int(len(data)*0.70): int(len(data))]) #Remaining 30% pass into test data

Python

[13] print(training_data.shape)
print(testing_data.shape)

Python

... (880, 1)
(378, 1)
```

Scaling Dataset

Feature scaling is the process of converting a dataset's numerical characteristics into a predetermined range or scale to speed up the training of machine learning algorithms and boost their performance. It is a crucial step in the preprocessing process that is frequently carried out on the input characteristics (sometimes referred to as independent variables or predictors) of a machine learning model.

In our project, we are using max scaler function. With this technique, the features are typically scaled to a range between [0, 1]. It modifies the data by dividing it by the range and removing the minimum value from each data point.

```
[14] from sklearn.preprocessing import MinMaxScaler
scaling = MinMaxScaler(feature_range= (0,1))

scaling.fit(training_data)
training_array = scaling.fit_transform(training_data)
training_array

Python
```

Model Implementation

There are several models which can be used in time series analysis such Arima, Arma etc. but the strongest and most robust of them all is the LSTM model of deep neural network.

Long Short-Term Memory

Recurrent neural network (RNN) architectures such as Long Short-Term Memory (LSTM) are frequently used for sequential data processing applications like time series forecasting, speech recognition, and natural language processing. Because of the disappearing or inflating gradient problem, typical RNNs have difficulty capturing long-term dependencies in sequential data. LSTMs are made to address this issue.

The use of a memory cell, which enables LSTMs to transfer information across many time steps in the sequence, is their main innovation. Specialized gates, such as the input gate, forget gate, and output gate, which are governed by sigmoid and tanh activation functions, are used to update the memory cell. The LSTM can learn when to update and retain information based on the input data thanks to these gates, which control the flow of information into and out of the memory cell.

LSTMs are a powerful type of recurrent neural network architecture that can effectively model long-term dependencies and complex patterns in sequential data.

We are using dense layer LSTM models in our project with dense layers. For activation function at first, we are using 'relu' activation function and after that also using 'gelu' and 'elu' functions. We are also using a dropout layer to avoid overfitting of data and during fitting of model we are using 'adam' optimizer with mean square error loss function.

```
Implementing LSTM Model

model = Sequential()
model.add(LSTM(units = 70, activation = 'relu', return_sequences = True, input_shape = (trainX.shape[1],1)))
model.add(Dropout(0.1))

model.add(LSTM(units = 80, activation = 'relu', return_sequences = True))
model.add(Dropout(0.15))

model.add(LSTM(units = 90, activation = 'relu', return_sequences = True))
model.add(Dropout(0.2))

model.add(LSTM(units = 100, activation = 'relu', return_sequences = True))
model.add(Dropout(0.3))

model.add(LSTM(units = 110, activation = 'relu', return_sequences = True))
model.add(Dropout(0.4))

model.add(LSTM(units = 120, activation = 'relu'))
model.add(Dropout(0.5))

model.add(Dense(units = 1))
```

```
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
model.fit(trainX, trainy, epochs= 40)
```

1- Activation Function

In neural networks, particularly LSTM (Long Short-Term Memory) models, activation functions play a significant role. The neural network may learn intricate patterns and relationships in the data thanks to activation functions, which add non-linearity to the model. In LSTM models, activation functions are used at many stages, including the input, forget, and output gates as well as the LSTM cells' output. In our model, we are using relu, elu and gelu activation's function because they solve vanishing gradient issue in network.

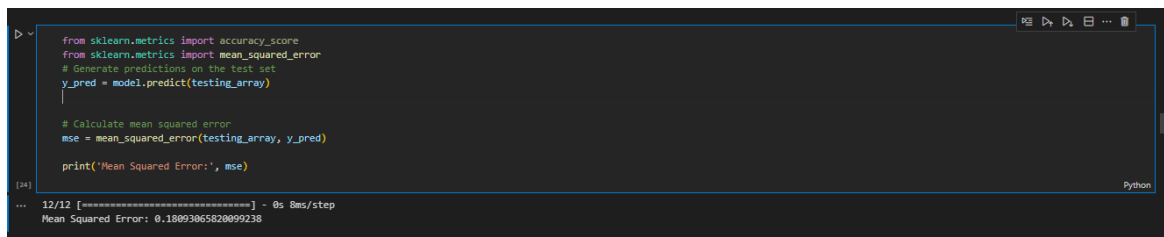
We are getting minimum loss function using ELU activation function. The ELU activation function is one of variation of the Rectified Linear Unit (ReLU) activation function, which is another popular activation function.

2- Optimizer

The optimizer is a crucial part of an LSTM (Long Short-Term Memory) model that is used to enhance learning during training. Based on the gradient of the loss function with respect to the model's parameters, the optimizer chooses how to update the model's weights. The LSTM model's overall performance and speed of convergence can be affected by the optimizer selection.

3- Mean Squared Error (MSE)

When using LSTM models for time series forecasting, the Mean Squared Error (MSE) loss function is frequently employed. The average squared difference between the expected and actual values is what is measured. The MSE loss function is used in time series forecasting to measure the discrepancy between the target variable's expected and actual values at each time step.



```
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error
# Generate predictions on the test set
y_pred = model.predict(testing_array)

# Calculate mean squared error
mse = mean_squared_error(testing_array, y_pred)
print('Mean Squared Error:', mse)
```

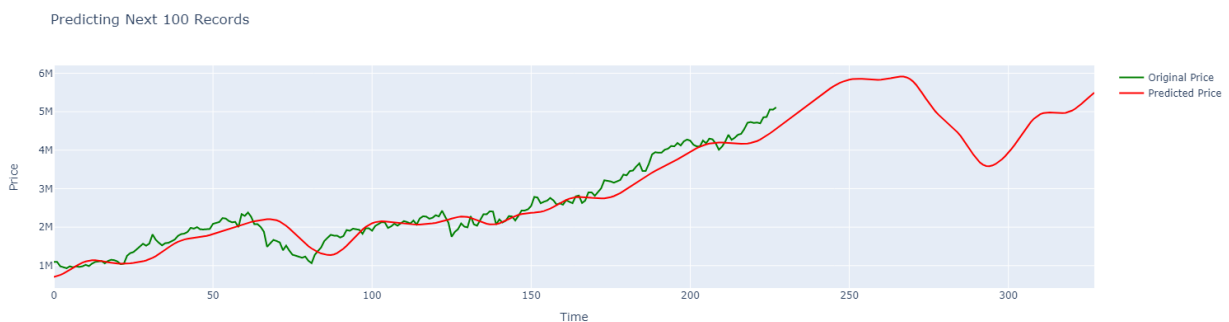
... 12/12 [-----] - 0s 8ms/step
Mean Squared Error: 0.18093865820099238

Now as we have our model saved, we are trying to predict stock value for the next 100 days using this model. But before that we are predicting stock price of the full data to check the performance of our model by checking difference between actual price and the predicted price of the model.



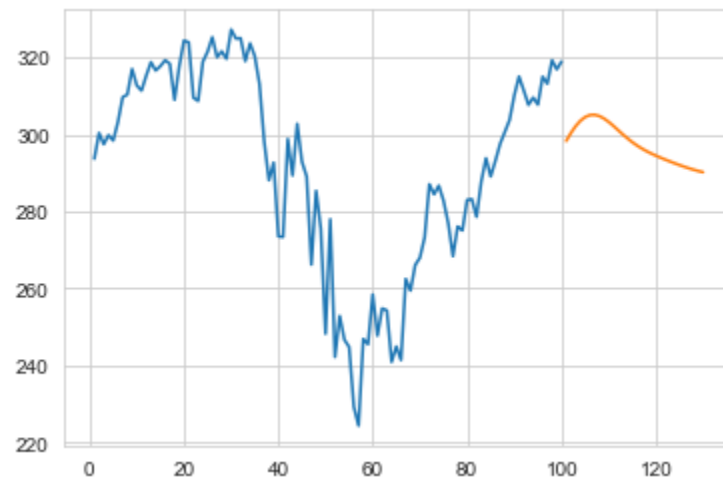
From the graph we can observe that our model predicts price quite similar to the actual price of stock. Even the trend in the data is correctly predicted by the model. Now we can also implement a model to predict the future price.

Similarly, now we are predicting prices for future days. Here in our model, we predict price by learning first 250 records and the doing predictions onward 250.



Predicting Price for Next 30 Days

Similarly, now we are using our model to predict the price for the next 30 days of the data. As we can see in the graph, that now model is predicting a downward trend in the data.



Conclusion

In conclusion, using historical stock data, our project used LSTM (Long Short-Term Memory) models to forecast Apple Inc. stock values. We used a deep architecture with several LSTM layers, each followed by a regularization Dropout layer. We trained and optimized the model using the Mean Squared Error (MSE) loss function. The training and testing sets were divided to assess the performance of the model after feature scaling was used to standardize the input data. The model's non-linearity was added using the ELU (Exponential Linear Unit) activation function. Our LSTM model demonstrated good results in predicting Apple Inc.'s stock prices after rigorous testing and fine-tuning, highlighting the potential of LSTM models in stock price forecasting. To further increase the precision and dependability of the stock price predictions, new features, improved hyperparameters, and the use of ensemble methods might all be investigated.

Limitations And Discussion

- 1- We have used data from the year 2010 to 2019. For future work, we can also use data for larger and broader periods so we can get maximum number of data and better learning of model.
- 2- Although LSTM is the one of the best forecasting models, we can also use other time series models like Arima etc.
- 3- We must use Apple company stock data; the same process can also be applied to any other company's data.
- 4- Many external factors, like the state of the global economy, current political events, and market mood, have an impact on stock prices, which can result in significant levels of volatility. LSTM models could have trouble capturing and forecasting such quick changes in the market.