

## ▼ Understand the Problem Statement/Case

AI, ML, and DL have been transforming finance and investing. In this project, one has to train a ridge regression model and deep neural network model to predict future stock prices. By accurately predicting stock prices, investors can maximize returns and know when to buy/sell securities. Furthermore, the AI/ML model will be trained using historical stock price data along with volume of transactions. A type of neural nets known as Long Short-Term Memory Networks (LSTM) has been used.

## ▼ Importing Datasets and Libraries

```
from google.colab import drive
drive.mount('/content/drive')
```

☞ Drive already mounted at /content/drive; to attempt to forcibly remount, call

```
import pandas as pd
import plotly.express as px
from copy import copy
from scipy import stats
import matplotlib.pyplot as plt
import numpy as np
import plotly.figure_factory as ff
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from tensorflow import keras
```

```
# Read stock price data
stock_price = pd.read_csv("/content/drive/MyDrive/stock.csv")
stock_price
```

	Date	AAPL	BA	T	MGM	AMZN	IBM	
0	2012-01-12	60.198570	75.510002	30.120001	12.130000	175.929993	180.550003	
1	2012-01-13	59.972858	74.599998	30.070000	12.350000	178.419998	179.160004	
2	2012-01-17	60.671429	75.239998	30.250000	12.250000	181.660004	180.000000	
3	2012-01-18	61.301430	75.059998	30.330000	12.730000	189.440002	181.070007	
4	2012-01-19	61.107143	75.559998	30.420000	12.800000	194.449997	180.520004	
...	...	...	...	...	...	...	...	
2154	2020-08-05	440.250000	174.279999	29.850000	16.719999	3205.030029	125.449997	1
2155	2020-08-06	455.609985	172.199997	29.840000	18.459999	3225.000000	126.120003	1
...	...	...	...	...	...	...	...	

```
# Read the stock volume data
stock_volume = pd.read_csv("/content/drive/MyDrive/stock_volume.csv")
stock_volume
```

	Date	AAPL	BA	T	MGM	AMZN	IBM	TSLA	
0	2012-01-12	53146800	3934500	26511100	17891100	5385800	6881000	729300	3
1	2012-01-13	56505400	4641100	22096800	16621800	4753500	5279200	5500400	4
2	2012-01-17	60724300	3700100	23500200	15480800	5644500	6003400	4651600	3
3	2012-01-18	69197800	4189500	22015000	18387600	7473500	4600600	1260200	5
4	2012-01-19	65434600	5397300	25524000	14022900	7096000	8567200	1246300	12
...	...	...	...	...	...	...	...	...	...
2154	2020-08-05	30498000	46551000	22991700	18914200	3930000	3675400	4978000	1
2155	2020-08-06	50607200	32921600	21908700	35867700	3940600	3417100	5992300	1
...	...	...	...	...	...	...	...	...	...

```
# Sort the data based on Date
stock_price = stock_price.sort_values(by = ['Date'])
stock_price
```

	Date	AAPL	BA	T	MGM	AMZN	IBM	
0	2012-01-12	60.198570	75.510002	30.120001	12.130000	175.929993	180.550003	
1	2012-01-13	59.972858	74.599998	30.070000	12.350000	178.419998	179.160004	
2	2012-01-17	60.671429	75.239998	30.250000	12.250000	181.660004	180.000000	
3	2012-01-18	61.301430	75.059998	30.330000	12.730000	189.440002	181.070007	
4	2012-01-19	61.107143	75.559998	30.420000	12.800000	194.449997	180.520004	
...	...	...	...	...	...	...	...	
2154	2020-08-05	440.250000	174.279999	29.850000	16.719999	3205.030029	125.449997	1
2155	2020-08-06	455.609985	172.199997	29.840000	18.459999	3225.000000	126.120003	1
...	...	...	...	...	...	...	...	

```
# Sort the volume data based on Date
stock_volume = stock_volume.sort_values(by = ['Date'])
stock_volume
```

	Date	AAPL	BA	T	MGM	AMZN	IBM	TSLA	
0	2012-01-12	53146800	3934500	26511100	17891100	5385800	6881000	729300	35
1	2012-01-13	56505400	4641100	22096800	16621800	4753500	5279200	5500400	40
2	2012-01-17	60724300	3700100	23500200	15480800	5644500	6003400	4651600	38
3	2012-01-18	69197800	4189500	22015000	18387600	7473500	4600600	1260200	51
4	2012-01-19	65434600	5397300	25524000	14022900	7096000	8567200	1246300	120
...	...	...	...	...	...	...	...	...	...
2154	2020-08-05	30498000	46551000	22991700	18914200	3930000	3675400	4978000	19
2155	2020-08-06	50607200	32921600	21908700	35867700	3940600	3417100	5992300	19
...	...	...	...	...	...	...	...	...	...

```
# Check if Null values exist in stock prices data
stock_price.isnull().sum()
```

```
Date      0
AAPL      0
BA        0
T         0
MGM       0
AMZN      0
IBM       0
TSLA      0
GOOG      0
sp500     0
dtype: int64
```

```
# Check if Null values exist in stocks volume data
stock_volume.isnull().sum()
```

```
Date      0
AAPL      0
BA        0
T         0
MGM       0
AMZN      0
IBM       0
TSLA      0
GOOG      0
sp500     0
dtype: int64
```

```
# Get stock prices dataframe info
stock_price.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2159 entries, 0 to 2158
Data columns (total 10 columns):
#   Column      Non-Null Count  Dtype
---  -
0   Date        2159 non-null   object
1   AAPL        2159 non-null   float64
2   BA          2159 non-null   float64
3   T           2159 non-null   float64
4   MGM         2159 non-null   float64
5   AMZN        2159 non-null   float64
6   IBM         2159 non-null   float64
7   TSLA        2159 non-null   float64
8   GOOG        2159 non-null   float64
9   sp500       2159 non-null   float64
dtypes: float64(9), object(1)
memory usage: 185.5+ KB
```

```
# Get stock volume dataframe info
stock_volume.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2159 entries, 0 to 2158
Data columns (total 10 columns):
#   Column      Non-Null Count  Dtype
---  ---
0    Date        2159 non-null   object
1    AAPL        2159 non-null   int64
2    BA          2159 non-null   int64
3    T           2159 non-null   int64
4    MGM         2159 non-null   int64
5    AMZN        2159 non-null   int64
6    IBM         2159 non-null   int64
7    TSLA        2159 non-null   int64
8    GOOG        2159 non-null   int64
9    SP500       2159 non-null   int64
dtypes: int64(9), object(1)
memory usage: 185.5+ KB
```

```
stock_price.describe()
```

	AAPL	BA	T	MGM	AMZN	IBM
<b>count</b>	2159.000000	2159.000000	2159.000000	2159.000000	2159.000000	2159.000000
<b>mean</b>	140.819823	189.942700	35.162899	23.105743	915.665665	161.853001
<b>std</b>	70.827601	103.678586	3.207490	6.963847	697.838905	25.561938
<b>min</b>	55.790001	67.239998	26.770000	7.140000	175.929993	94.769997
<b>25%</b>	89.165714	124.015000	33.040001	18.545000	316.490005	142.769997
<b>50%</b>	116.599998	142.419998	34.930000	23.780001	676.010010	156.949997
<b>75%</b>	175.019997	297.044998	37.419998	28.430000	1593.645019	185.974998
<b>max</b>	455.609985	440.619995	43.470001	38.029999	3225.000000	215.800003

```
stock_volume.describe()
```

	AAPL	BA	T	MGM	AMZN	
<b>count</b>	2.159000e+03	2.159000e+03	2.159000e+03	2.159000e+03	2.159000e+03	2.159000e+03
<b>mean</b>	5.820332e+07	6.419916e+06	2.832131e+07	9.845582e+06	4.102673e+06	4.453091e+06
<b>std</b>	4.568141e+07	9.711873e+06	1.428911e+07	7.295753e+06	2.290722e+06	2.462811e+06
<b>min</b>	1.136200e+07	7.889000e+05	6.862400e+06	9.507000e+05	8.813000e+05	1.193000e+06
<b>25%</b>	2.769930e+07	3.031850e+06	2.002150e+07	5.796450e+06	2.675700e+06	3.111250e+06
<b>50%</b>	4.209420e+07	3.991000e+06	2.485930e+07	7.899800e+06	3.494800e+06	3.825000e+06
<b>75%</b>	7.182480e+07	5.325900e+06	3.210565e+07	1.104055e+07	4.768150e+06	4.937300e+06
<b>max</b>	3.765300e+08	1.032128e+08	1.950827e+08	9.009820e+07	2.385610e+07	3.049020e+07

## ▼ Perform Exploratory Data Analysis and Visualisation

```
# Function to normalize stock prices based on their initial price
```

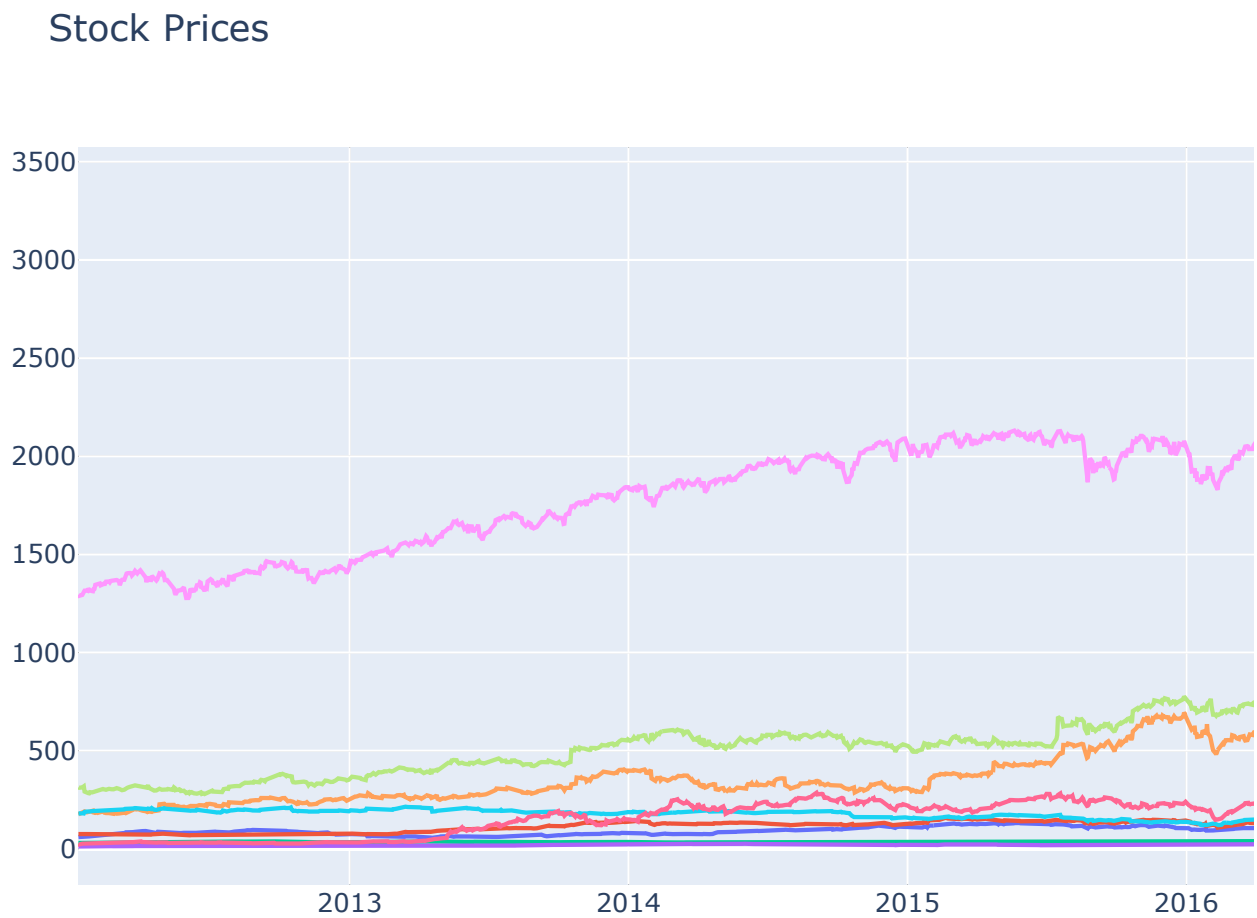
```
def normalize(df):
    x = df.copy()
    for i in x.columns[1:]:
        x[i] = x[i]/x[i][0]
    return x
```

```
# Function to plot interactive plots using Plotly Express
```

```
def interactive_plot(df, title):
    fig = px.line(title = title)
    for i in df.columns[1:]:
        fig.add_scatter(x = df['Date'], y = df[i], name = i)
    fig.show()
```



```
# Plot interactive chart for stocks data
interactive_plot(stock_price, 'Stock Prices')
```




## ▼ Prepare The Data Before Training The AI/ML Model

```
# Function to concatenate the date, stock price, and volume in one dataframe
def individual_stock(price_df, volume_df, name):
    return pd.DataFrame({'Date': price_df['Date'], 'Close': price_df[name], 'Vol
```

```
# Function to return the input/output (target) data for AI/ML Model
# Note that our goal is to predict the future stock price
# Target stock price today will be tomorrow's price
def trading_window(data):
    n = 1
    data['Target'] = data[['Close']].shift(-n)
    return data
```

```
# Let's test the functions and get individual stock prices and volumes for AAPL
price_volume = individual_stock(stock_price, stock_volume, 'AAPL')
price_volume
```

	Date	Close	Volume	
0	2012-01-12	60.198570	53146800	
1	2012-01-13	59.972858	56505400	
2	2012-01-17	60.671429	60724300	
3	2012-01-18	61.301430	69197800	
4	2012-01-19	61.107143	65434600	
...	...	...	...	
2154	2020-08-05	440.250000	30498000	
2155	2020-08-06	455.609985	50607200	
2156	2020-08-07	444.450012	49453300	
2157	2020-08-10	450.910004	53100900	
2158	2020-08-11	437.500000	46871100	

2159 rows x 3 columns

```
price_volume_target = trading_window(price_volume)
price_volume_target
```

	Date	Close	Volume	Target
0	2012-01-12	60.198570	53146800	59.972858
1	2012-01-13	59.972858	56505400	60.671429
2	2012-01-17	60.671429	60724300	61.301430
3	2012-01-18	61.301430	69197800	61.107143
4	2012-01-19	61.107143	65434600	60.042858
...	...	...	...	...
2154	2020-08-05	440.250000	30498000	455.609985
2155	2020-08-06	455.609985	50607200	444.450012
2156	2020-08-07	444.450012	49453300	450.910004
2157	2020-08-10	450.910004	53100900	437.500000
2158	2020-08-11	437.500000	46871100	NaN

2159 rows × 4 columns

```
# Remove the last row as it will be a null value
price_volume_target = price_volume_target[:-1]
price_volume_target
```

	Date	Close	Volume	Target
0	2012-01-12	60.198570	53146800	59.972858
1	2012-01-13	59.972858	56505400	60.671429
2	2012-01-17	60.671429	60724300	61.301430
3	2012-01-18	61.301430	69197800	61.107143
4	2012-01-19	61.107143	65434600	60.042858
...	...	...	...	...
2153	2020-08-04	438.660004	43267900	440.250000
2154	2020-08-05	440.250000	30498000	455.609985
2155	2020-08-06	455.609985	50607200	444.450012
2156	2020-08-07	444.450012	49453300	450.910004
2157	2020-08-10	450.910004	53100900	437.500000

2158 rows x 4 columns

```
# Scale the data
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
price_volume_target_scaled = sc.fit_transform(price_volume_target.drop(columns =
```

```
price_volume_target_scaled
```

```
array([[0.01102638, 0.11442624, 0.01046185],
       [0.01046185, 0.12362365, 0.01220906],
       [0.01220906, 0.13517696, 0.01378478],
       ...,
       [1.          , 0.10747163, 0.97208751],
       [0.97208751, 0.10431171, 0.98824476],
       [0.98824476, 0.11430054, 0.95470465]])
```

```
price_volume_target_scaled.shape
```

```
(2158, 3)
```

```
# Creating Feature and Target
X = price_volume_target_scaled[:, :2]
y = price_volume_target_scaled[:, 2:]
```

```
X.shape, y.shape
```

```
((2158, 2), (2158, 1))
```

```
# Splitting the data this way, since order is important in time-series
# Note that we did not use train test split with it's default settings since it
split = int(0.65 * len(X))
X_train = X[:split]
y_train = y[:split]
X_test = X[split:]
y_test = y[split:]
```

```
X_train.shape, y_train.shape
```

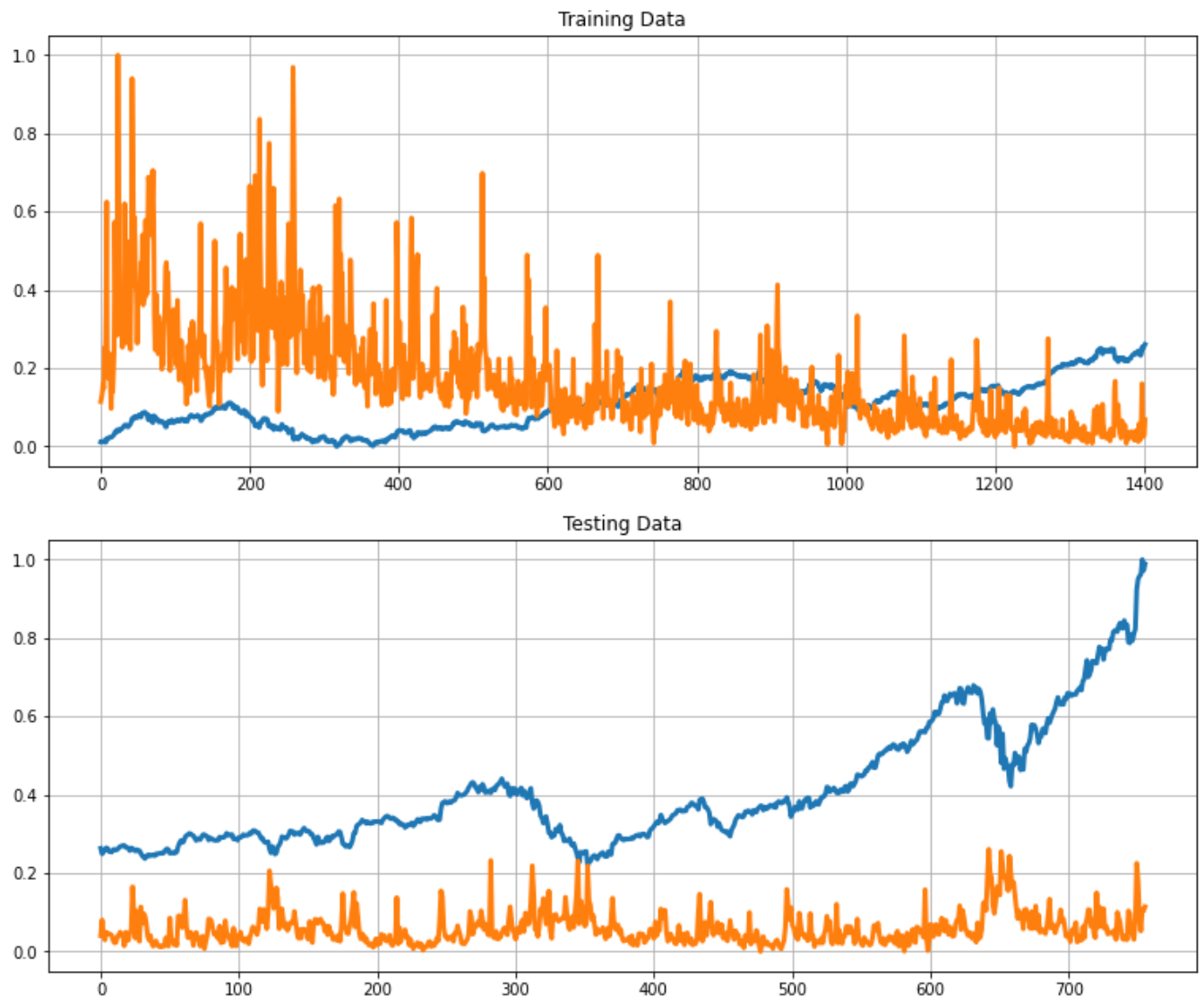
```
((1402, 2), (1402, 1))
```

```
X_test.shape, y_test.shape
```

```
((756, 2), (756, 1))
```

```
# Define a data plotting function
def show_plot(data, title):
    plt.figure(figsize = (13, 5))
    plt.plot(data, linewidth = 3)
    plt.title(title)
    plt.grid()

show_plot(X_train, 'Training Data')
show_plot(X_test, 'Testing Data')
```



## ▼ Build and Train a Ridge Linear Regression Model

```
from sklearn.linear_model import Ridge
# Note that Ridge regression performs linear least squares with L2 regularization
# Create and train the Ridge Linear Regression Model
regression_model = Ridge()
regression_model.fit(X_train, y_train)
```

```
Ridge()
```

```
# Test the model and calculate its accuracy
lr_accuracy = regression_model.score(X_test, y_test)
print("Linear Regression Score: ", lr_accuracy)
```

```
Linear Regression Score: 0.7950028030821766
```

```
# Make Prediction
predicted_prices = regression_model.predict(X)
predicted_prices
```

```
array([[0.03466412],
       [0.03374627],
       [0.03451936],
       ...,
       [0.81048342],
       [0.78876033],
       [0.80091324]])
```

```
# Append the predicted values into a list
Predicted = []
for i in predicted_prices:
    Predicted.append(i[0])
```

```
len(Predicted)
```

```
2158
```

```
# Append the close values to the list
close = []
for i in price_volume_target_scaled:
    close.append(i[0])
```


```
# Create a dataframe based on the dates in the individual stock data
df_predicted = price_volume_target[['Date']]
df_predicted
```

	Date
0	2012-01-12
1	2012-01-13
2	2012-01-17
3	2012-01-18
4	2012-01-19
...	...
2153	2020-08-04
2154	2020-08-05
2155	2020-08-06
2156	2020-08-07
2157	2020-08-10

2158 rows × 1 columns




```
# Add the close values to the dataframe  
df_predicted['Close'] = close  
df_predicted
```

	Date	Close	
0	2012-01-12	0.011026	
1	2012-01-13	0.010462	
2	2012-01-17	0.012209	
3	2012-01-18	0.013785	
4	2012-01-19	0.013299	
...	...	...	
2153	2020-08-04	0.957606	
2154	2020-08-05	0.961583	
2155	2020-08-06	1.000000	
2156	2020-08-07	0.972088	
2157	2020-08-10	0.988245	

2158 rows × 2 columns

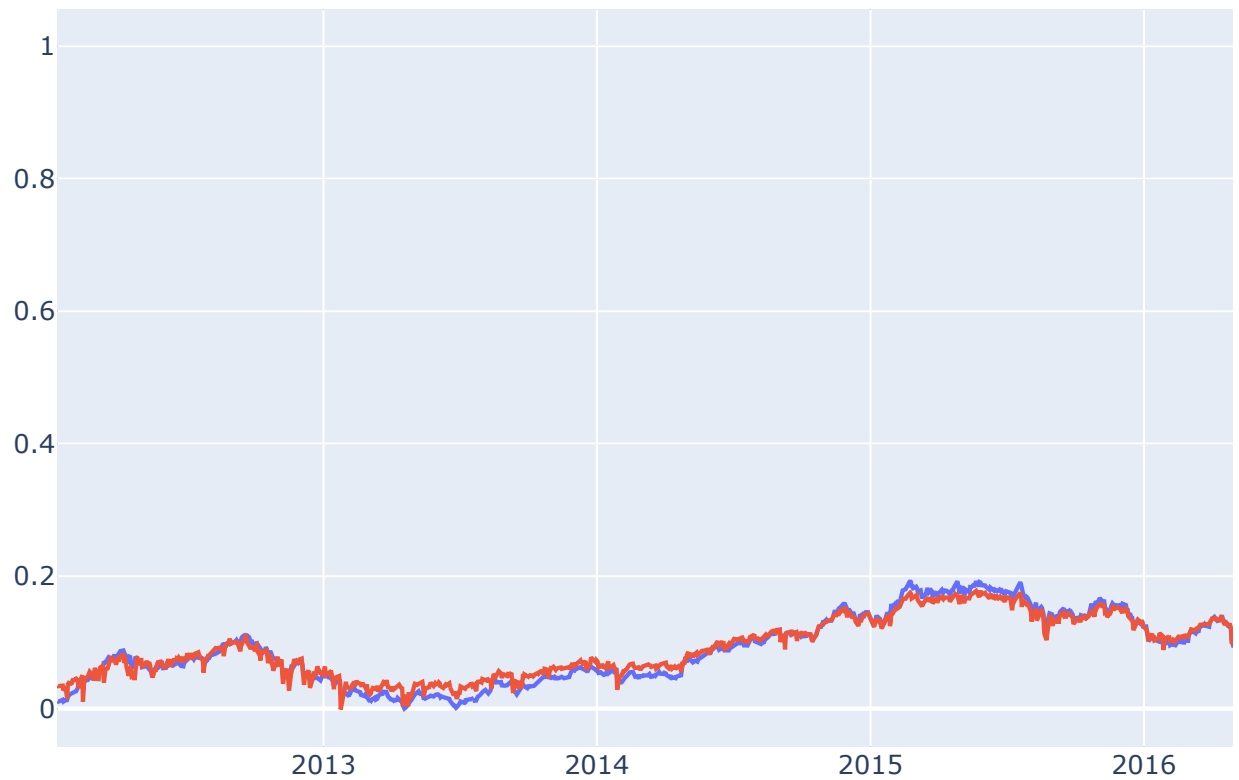
```
# Add the predicted values to the dataframe  
df_predicted['Prediction'] = Predicted  
df_predicted
```

	Date	Close	Prediction	
0	2012-01-12	0.011026	0.034664	
1	2012-01-13	0.010462	0.033746	
2	2012-01-17	0.012209	0.034519	
3	2012-01-18	0.013785	0.034556	
4	2012-01-19	0.013299	0.034707	
...	...	...	...	
2153	2020-08-04	0.957606	0.778280	
2154	2020-08-05	0.961583	0.783205	
2155	2020-08-06	1.000000	0.810483	
2156	2020-08-07	0.972088	0.788760	
2157	2020-08-10	0.988245	0.800913	

2158 rows × 3 columns

```
# Plot the results  
interactive_plot(df_predicted, "Original Vs. Prediction")
```

Original Vs. Prediction



## ▼ Train an LSTM Time Series Model

```
# Let's test the functions and get individual stock prices and volumes for AAPL
price_volume = individual_stock(stock_price, stock_volume, 'sp500')
price_volume
```

	Date	Close	Volume	
0	2012-01-12	1295.500000	4019890000	
1	2012-01-13	1289.089966	3692370000	
2	2012-01-17	1293.670044	4010490000	
3	2012-01-18	1308.040039	4096160000	
4	2012-01-19	1314.500000	4465890000	
...	...	...	...	
2154	2020-08-05	3327.770020	4732220000	
2155	2020-08-06	3349.159912	4267490000	
2156	2020-08-07	3351.280029	4104860000	
2157	2020-08-10	3360.469971	4318570000	
2158	2020-08-11	3333.689941	5087650000	

2159 rows × 3 columns

```
# Get the close and volume data as training data (Input)
training_data = price_volume.iloc[:, 1:3].values
training_data
```

```
array([[1.29550000e+03, 4.01989000e+09],
       [1.28908997e+03, 3.69237000e+09],
       [1.29367004e+03, 4.01049000e+09],
       ...,
       [3.35128003e+03, 4.10486000e+09],
       [3.36046997e+03, 4.31857000e+09],
       [3.33368994e+03, 5.08765000e+09]])
```

```
# Normalize the data
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
training_set_scaled = sc.fit_transform(training_data)
```

```
# Create the training and testing data, training data contains present day and p
X = []
y = []
for i in range(1, len(price_volume)):
    X.append(training_set_scaled [i-1:i, 0])
    y.append(training_set_scaled [i, 0])

# Convert the data into array format
X = np.asarray(X)
y = np.asarray(y)

# Split the data
split = int(0.7 * len(X))
X_train = X[:split]
y_train = y[:split]
X_test = X[split:]
y_test = y[split:]

# Reshape the 1D arrays to 3D arrays to feed in the model
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))
X_train.shape, X_test.shape

((1510, 1, 1), (648, 1, 1))
```

```
# Create the model
inputs = keras.layers.Input(shape=(X_train.shape[1], X_train.shape[2]))
x = keras.layers.LSTM(150, return_sequences= True)(inputs)
x = keras.layers.Dropout(0.3)(x)
x = keras.layers.LSTM(150, return_sequences=True)(x)
x = keras.layers.Dropout(0.3)(x)
x = keras.layers.LSTM(150)(x)
outputs = keras.layers.Dense(1, activation='linear')(x)

model = keras.Model(inputs=inputs, outputs=outputs)
model.compile(optimizer='adam', loss="mse")
model.summary()
```

Model: "model\_1"

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 1, 1)]	0
lstm_3 (LSTM)	(None, 1, 150)	91200
dropout_2 (Dropout)	(None, 1, 150)	0
lstm_4 (LSTM)	(None, 1, 150)	180600
dropout_3 (Dropout)	(None, 1, 150)	0
lstm_5 (LSTM)	(None, 150)	180600
dense_1 (Dense)	(None, 1)	151

```
=====  
Total params: 452,551  
Trainable params: 452,551  
Non-trainable params: 0  
=====
```

```
# Train the model
history = model.fit(
    X_train, y_train,
    epochs = 20,
    batch_size = 32,
    validation_split = 0.2
)
```

```
Epoch 1/20
38/38 [=====] - 7s 46ms/step - loss: 0.0317 - val_
Epoch 2/20
38/38 [=====] - 0s 12ms/step - loss: 0.0074 - val_
Epoch 3/20
38/38 [=====] - 0s 11ms/step - loss: 6.8593e-04 -
Epoch 4/20
38/38 [=====] - 0s 11ms/step - loss: 3.8687e-04 -
Epoch 5/20
38/38 [=====] - 0s 12ms/step - loss: 3.7438e-04 -
Epoch 6/20
38/38 [=====] - 0s 11ms/step - loss: 3.0908e-04 -
Epoch 7/20
38/38 [=====] - 0s 12ms/step - loss: 3.4145e-04 -
Epoch 8/20
38/38 [=====] - 0s 11ms/step - loss: 2.9971e-04 -
Epoch 9/20
38/38 [=====] - 0s 11ms/step - loss: 2.6865e-04 -
Epoch 10/20
38/38 [=====] - 0s 11ms/step - loss: 2.7031e-04 -
Epoch 11/20
38/38 [=====] - 0s 12ms/step - loss: 2.7786e-04 -
Epoch 12/20
38/38 [=====] - 0s 11ms/step - loss: 2.3303e-04 -
Epoch 13/20
38/38 [=====] - 0s 13ms/step - loss: 2.7933e-04 -
Epoch 14/20
38/38 [=====] - 0s 11ms/step - loss: 2.3730e-04 -
Epoch 15/20
38/38 [=====] - 0s 11ms/step - loss: 2.3949e-04 -
Epoch 16/20
38/38 [=====] - 0s 11ms/step - loss: 2.2529e-04 -
Epoch 17/20
38/38 [=====] - 0s 11ms/step - loss: 2.1532e-04 -
Epoch 18/20
38/38 [=====] - 0s 12ms/step - loss: 2.2061e-04 -
Epoch 19/20
38/38 [=====] - 0s 11ms/step - loss: 2.1161e-04 -
Epoch 20/20
38/38 [=====] - 0s 11ms/step - loss: 2.3217e-04 -
```

```
# Make prediction
predicted = model.predict(X)

test_predicted = []

for i in predicted:
    test_predicted.append(i[0])

df_predicted = price_volume[1:][['Date']]

df_predicted['Predicted'] = test_predicted

df_predicted
```

	Date	Predicted	
1	2012-01-13	0.013066	
2	2012-01-17	0.010017	
3	2012-01-18	0.012196	
4	2012-01-19	0.019032	
5	2012-01-20	0.022106	
...	...	...	
2154	2020-08-05	0.937698	
2155	2020-08-06	0.946719	
2156	2020-08-07	0.955771	
2157	2020-08-10	0.956667	
2158	2020-08-11	0.960548	

2158 rows x 2 columns

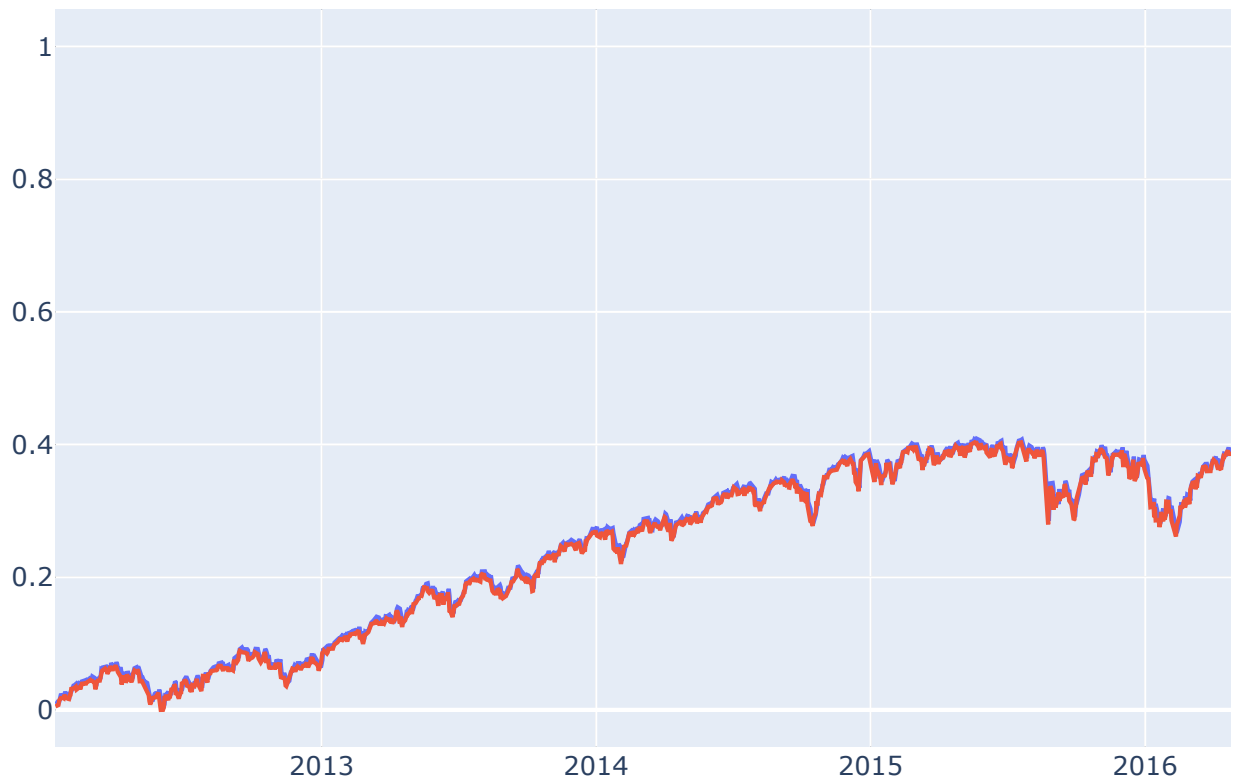
```
close = []
for i in training_set_scaled:
    close.append(i[0])

df_predicted['Close'] = close[1:]
```



```
interactive_plot(df_predicted, "Original Vs Predicted")
```

## Original Vs Predicted



---

✓ 0s completed at 1:55 PM

