An Attempt to Predict Stock Market Prices

Data Description:

In this mission, we'll apply ARIMA on TCS stock market prices and LSTM on S&P500 time series da

a) TCS

The TCS dataset is taken from Official NSE website. Timeline of Data recording : 1-1-2015 to 31-12

Column Descriptors: Date: date on which data is recorded Symbol: NSE symbol of the stock Series

b) S&P 500

The S&P500 dataset is stored in sphist.csv. Each row in the file contains a **daily record of the price**

The columns of the dataset are:

```
Date -- The date of the record.
Open -- The opening price of the day (when trading starts).
High -- The highest trade price during the day.
Low -- The lowest trade price during the day.
Close -- The closing price for the day (when trading is finished).
Volume -- The number of shares traded.
Adj Close -- The daily closing price, adjusted for corporate actions.
```

We'll be using both datasets to develop a predictive model. For S&P 500, we'll train the model wit predictions from 2013-2015. For TCS, we will have a train and test data split.

```
import tensorflow as tf
tf.test.gpu_device_name()
```

The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you upgrade now or ensure your notebook will continue to use TensorFlow 1.x via the `%tensorflow_version 1.x` magic: more info.

```
'/device:GPU:0'
```

Loading TCS Data

```
# Importing necessary modules
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Importing the statistics module
from statistics import mean
from statistics import median

# To load the input data
```

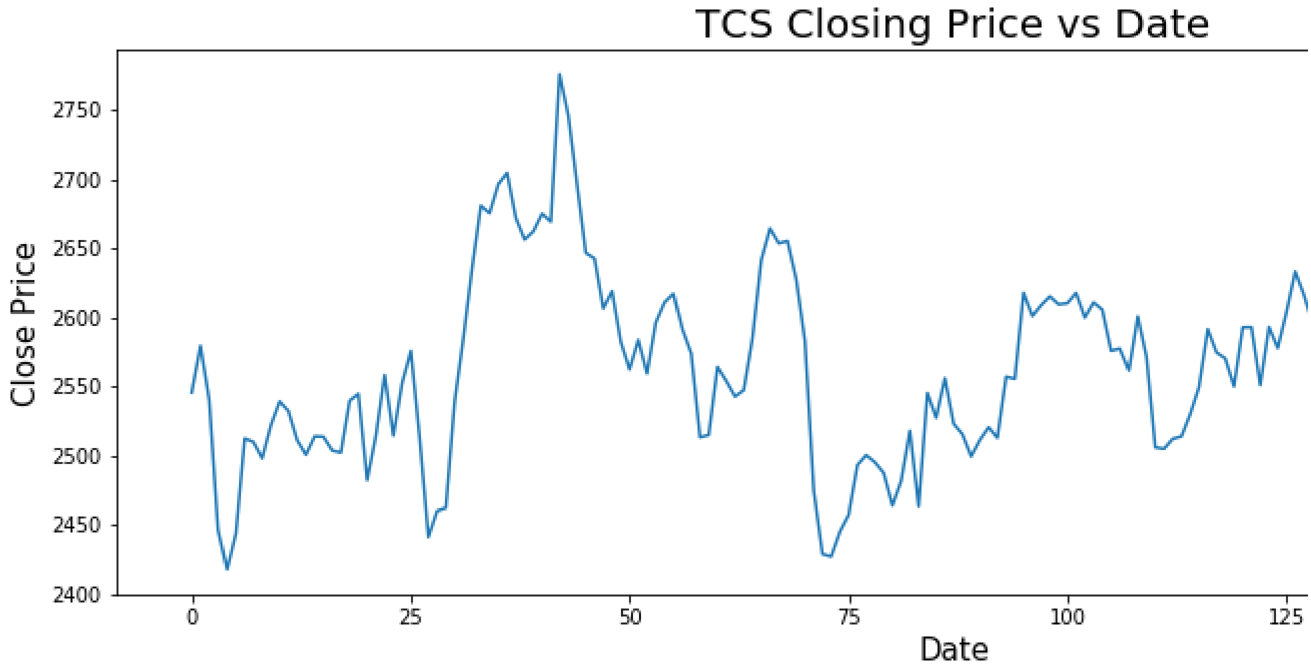
```
# To load the input data
import pandas as pd
import numpy as np
from datetime import datetime
from matplotlib import pyplot

# used to format headings
bold = '\033[1m'
end = '\033[0m'

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

# Read the s&p 500 input data set and sorting based on date.
sp500 = pd.read_csv("gdrive/My Drive/tseries/IT/tcs_stock.csv", index_col=False)

sp500["Date"] = pd.to_datetime(sp500["Date"])
sp_sorted = sp500.sort_values("Date")

print(sp_sorted.head(3))
# print(sp_sorted.tail(3))
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_

Enter your authorization code:
··········
Mounted at /content/gdrive
        Date Symbol Series  ...  Trades  Deliverable Volume  %Deliverble
0 2015-01-01    TCS     EQ  ...    8002               52870       0.2883
1 2015-01-02    TCS     EQ  ...   27585              309350       0.6683
2 2015-01-05    TCS     EQ  ...   43234              456728       0.5207

[3 rows x 15 columns]
```

From the sorted data, we can see that data since Jan 2015 is there in the input dataset.

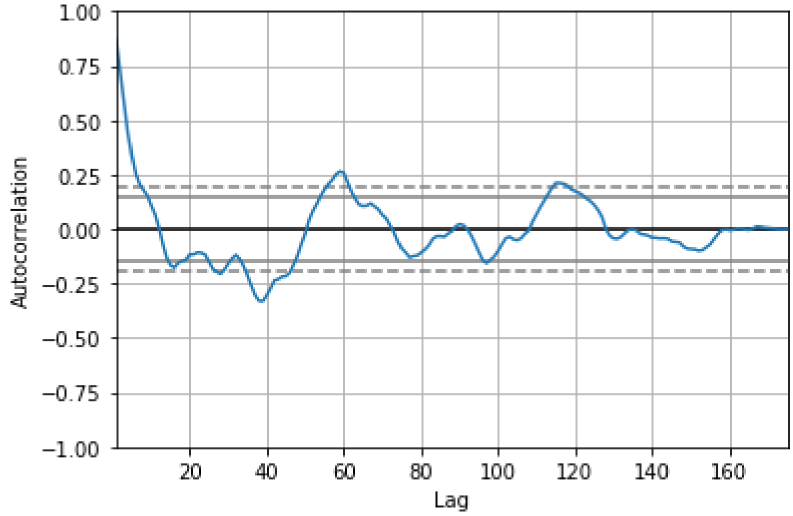
```
# Visualization Module

close = sp_sorted['Close'][:175]
x = np.arange(0, len(close))
print(len(close))

plt.figure(figsize=(15,5))
plt.plot(x, close, label="Close Price")

plt.legend()
plt.title("TCS Closing Price vs Date", fontsize=20)
plt.ylabel("Close Price", fontsize=15)
plt.xlabel("Date", fontsize=15)
plt.show()
```

175



To estimate the lag lets do a auto correlation plot.

```
from pandas.plotting import autocorrelation_plot
from matplotlib import pyplot

# close = np.real(filtered_sig)

autocorrelation_plot(close)
pyplot.show()
```

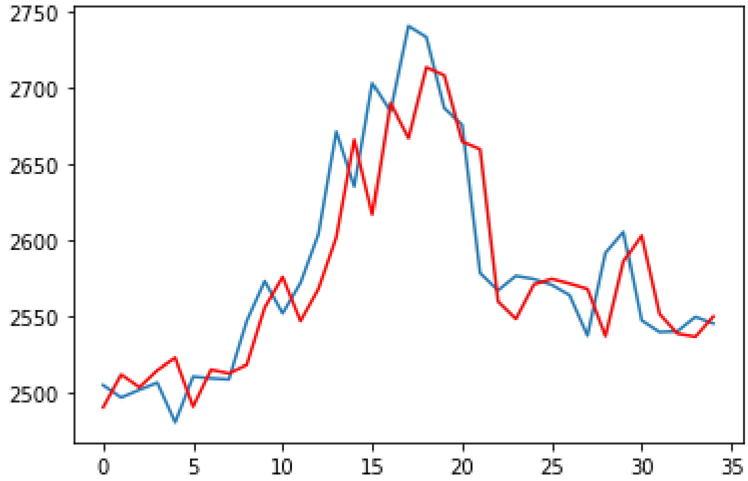


There is a positive correlation with the first 500 lags. Hence, a reasonable value for autocorrelatior

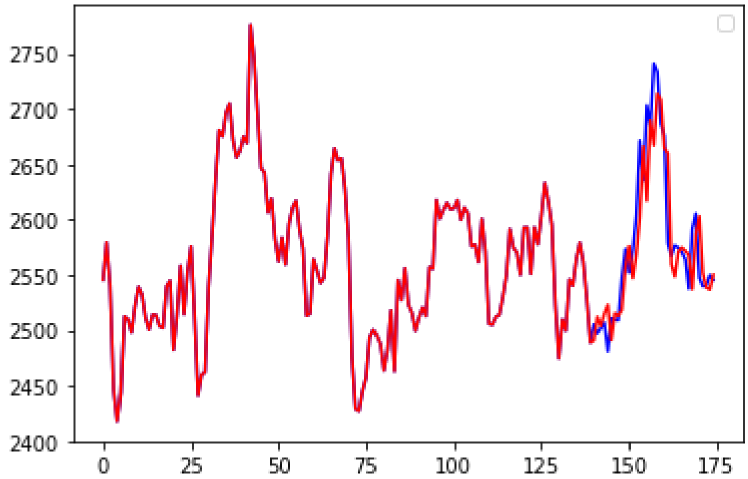
```
# https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/
import warnings
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error
```

```
X = close.values
size = int(len(X) * 0.8)
train, test = X[0:size], X[size:len(X)]
history = [x for x in train]
predictions = list()
print(len(test))

warnings.filterwarnings("ignore")
for t in range(len(test)):
  model = ARIMA(history, order=(8,0,2))
  model_fit = model.fit(disp=0)
  output = model_fit.forecast()
  yhat = output[0]
  predictions.append(yhat)
  obs = test[t]
  history.append(obs)
  # print('predicted=%f, expected=%f' % (yhat, obs))
error = mean_squared_error(test, predictions)
print('Test MSE: %.3f' % error)
# plot
pyplot.plot(test)
pyplot.plot(predictions, color='red')
pyplot.show()

train_data = train.tolist()
for i in range(len(predictions)):
  train_data.append(predictions[i][0])
```

```
35
Test MSE: 1180.102
```

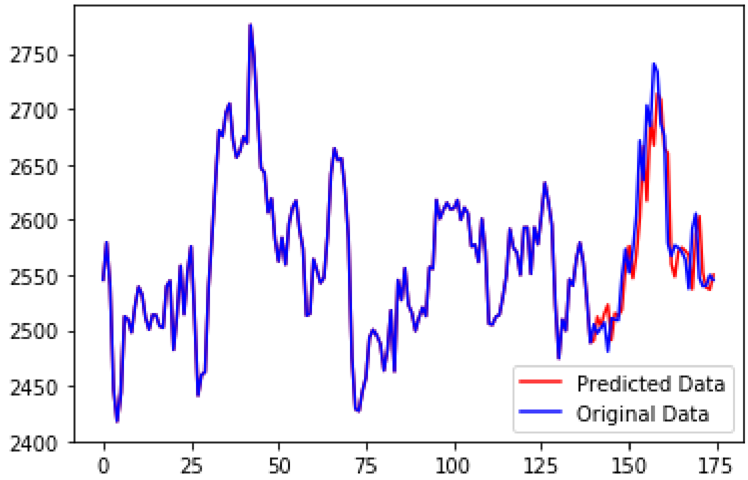



```
No handles with labels found to put in legend.
```




```
# Visualization with History + Prediction

print(predictions)
train_data = train.tolist()

for i in range(len(predictions)):
  train_data.append(predictions[i][0])
# print(train.tolist().append(predictions))
x_axis = np.arange(0, len(history))
pyplot.plot(train_data, color='red', label='Predicted Data')
pyplot.plot(close, color='blue', label='Original Data')
pyplot.legend()
pyplot.show()
```

```
[array([2490.52443477]), array([2512.00045077]), array([2503.73096076]), array
```



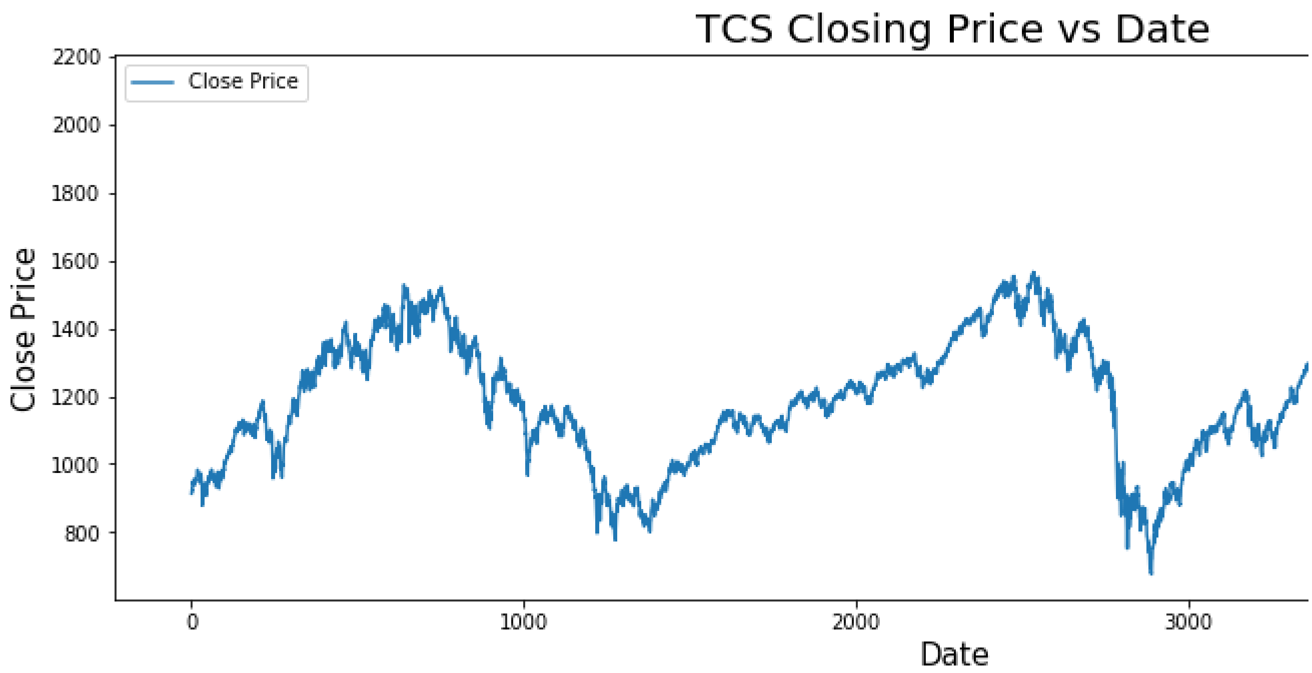
LSTM on S&P500 Time Series Data

```
# Read the s&p 500 input data set and sorting based on date.
sp500 = pd.read_csv("gdrive/My Drive/tseries/sphist.csv", index_col=False)

sp500["Date"] = pd.to_datetime(sp500["Date"])
sp_sorted = sp500.sort_values("Date")

print(sp_sorted.head(3))
```

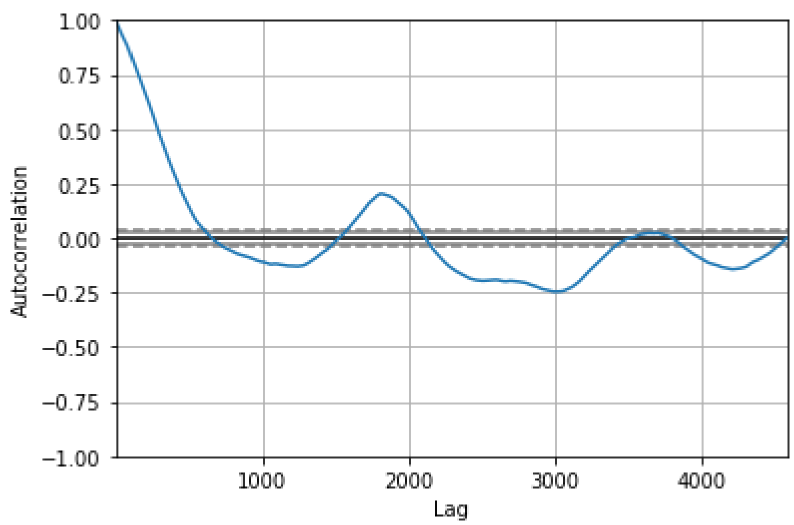
```
             Date   Open   High    Low  Close     Volume  Adj Close
16589 1950-01-03  16.66  16.66  16.66  16.66  1260000.0      16.66
16588 1950-01-04  16.85  16.85  16.85  16.85  1890000.0      16.85
16587 1950-01-05  16.93  16.93  16.93  16.93  2550000.0      16.93
```

```
# Visualization Module

close = sp_sorted['Close'][12000:]
x = np.arange(0, len(close))
print(len(close))

plt.figure(figsize=(15,5))
plt.plot(x, close, label="Close Price")

plt.legend()
plt.title("TCS Closing Price vs Date", fontsize=20)
plt.ylabel("Close Price", fontsize=15)
plt.xlabel("Date", fontsize=15)
plt.show()
```

```
4590
```




```
autocorrelation_plot(close)
pyplot.show()
```



From the above graph, we estimate the lag to be around 100.

```
# univariate LSTM example
from numpy import array
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
import warnings
from sklearn.metrics import mean_squared_error

# split a univariate sequence into samples
def split_sequence(sequence, n_steps):
  X, y = list(), list()
  for i in range(len(sequence)):
```

```
    # find the end of this pattern
    end_ix = i + n_steps
    # check if we are beyond the sequence
    if end_ix > len(sequence)-1:
      break
    # gather input and output parts of the pattern
    seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
    X.append(seq_x)
    y.append(seq_y)
  return array(X), array(y)

# define input sequence
raw_seq = list(close)
print(raw_seq)


size = int(len(raw_seq) * 0.9)
train, test = raw_seq[0:size], raw_seq[size:len(raw_seq)]
history = [x for x in train]
predictions = list()


# choose a number of time steps
n_steps = 100
print('before split sequence')
# split into samples
X, y = split_sequence(raw_seq, n_steps)

print('after split sequence')
# reshape from [samples, timesteps] into [samples, timesteps, features]
n_features = 1
X = X.reshape((X.shape[0], X.shape[1], n_features))
# define model
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(n_steps, n_features)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# fit model
model.fit(X, y, epochs=200, verbose=0)
print('model fitting over')

count = 0
warnings.filterwarnings("ignore")
# demonstrate prediction
for t in range(len(test)):
  x_input = array(raw_seq[-n_steps:])
  x_input = x_input.reshape((1, n_steps, n_features))
  yhat = model.predict(x_input, verbose=0)
  print(yhat[0][0])
  predictions.append(yhat[0][0])
  # as we predict the difference
  raw_seq.append(yhat[0][0])


error = mean_squared_error(test, predictions)
```

```
print('Test MSE: %.3f' % error)
pyplot.plot(test)
pyplot.plot(predictions, color='red')
pyplot.legend()
pyplot.show()

x_axis = np.arange(0, len(raw_seq))
pyplot.plot(x_axis, raw_seq, color='red')
pyplot.legend()
pyplot.show()
```

```
[912.590027, 923.909973, 919.7700199999999, 945.6400150000001, 943.0, 947.2899
before split sequence
after split sequence
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/t

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/t

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/t

Using TensorFlow backend.
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimizer

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/t

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/t

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/t

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/t

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/t

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/t

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/t

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/t

model fitting over
2009.6586
1999.673
1996.5353
1995.6016
1995.3267
1995.2456
1995.1567
1995.0874
1995.1855
1994.8098
1994.9089
1994.4658
1992.6172
1991.1746
1990.6826
1990.5731
1990.5492
1990.5406
1990.538
1990.5374
1990.5369
1990.537
1990.537
1990.537
1990.537
1990.537
1990.537
1990.537
1990.5369
```

```
1990.537
1990.537
1990.5371
1990.5372
1990.5375
1990.5377
1990.5382
1990.5386
1990.5391
1990.539
1990.5388
1990.5388
1990.5385
1990.5391
1990.538
1990.5399
1990.5381
1990.5416
1990.5415
1990.5432
1990.5493
1990.5549
1990.5573
1990.5631
1990.5664
1990.574
1990.571
1990.5935
1990.575
1990.5574
1990.6094
1990.5636
1990.1365
1990.418
1990.0643
1990.7942
1988.3472
1987.8984
1986.3228
1985.6857
1985.4597
1985.4049
1985.3812
1985.3799
1985.3767
1985.3744
1985.3745
1985.3715
1985.3969
1985.3641
1985.385
1985.2742
1985.2903
1984.8201
1981.6353
1983.1556
1972.917
1971.3711
1969.58
1961.4019
1961.8921
1951.626
```

```
1913.4485
1898.5732
1894.972
1893.7566
1893.3958
1893.0713
1892.8129
1892.4883
1891.363
1890.064
1889.583
1889.4773
1889.4453
1889.4359
1889.4331
1889.4323
1889.432
1889.432
1889.432
1889.432
1889.432
1889.432
1889.432
1889.432
1889.432
1889.432
1889.432
1889.432
1889.432
1889.432
1889.432
1889.432
1889.432
1889.432
1889.432
1889.432
1889.432
1889.432
1889.432
1889.4321
1889.432
1889.4319
1889.4319
1889.4315
1889.4318
1889.4309
1889.4312
1889.4299
1889.4271
1889.4266
1889.4248
1889.4241
1889.4229
1889.4158
1889.4136
1889.3975
1889.384
1889.1948
1889.2465
1888.4417
1888.5308
```