```
import sys
print(sys.version)
```

```
3.10.12 (main, Jun 11 2023, 05:26:28) [GCC 11.4.0]
```

```
import numpy as np
np.random.seed(1)

import pandas as pd
import matplotlib.pyplot as plt
from keras.models import Sequential, load_model
from keras.layers import Dense
from keras.layers import LSTM
from keras import optimizers
from keras.callbacks import EarlyStopping
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, r2_score
from math import sqrt
import datetime as dt
import time
plt.style.use('ggplot')
```

```
df = pd.read_csv('sp500.csv',parse_dates = True,index_col=0)
df[['Open', 'High', 'Low', 'Close', 'Volume', 'Adj Close']] = df[['Open', 'High', 'Low', 'Close*', 'Volume', 'Adj Close**']].replace({',': ''
df = df.drop(['Close*', 'Adj Close**'], axis='columns')
df.tail()
```

Date	Open	High	Low	Volume	Return	Close	Adj Close
2018-01-08	2742.67	2748.51	2737.60	3.246160e+09	4.56	2747.71	2747.71
2018-01-05	2731.33	2743.45	2727.92	3.239280e+09	19.16	2743.15	2743.15
2018-01-04	2719.31	2729.29	2719.07	3.697340e+09	10.93	2723.99	2723.99
2018-01-03	2697.85	2714.37	2697.77	3.544030e+09	17.25	2713.06	2713.06
2018-01-02	2683.73	2695.89	2682.36	3.397430e+09	22.20	2695.81	2695.81

```
data_to_train = df[:1000]
data_to_test = df[1000:]
```

```
df= df.iloc[: , 5:6]
df.head()
```

Date	Close
2023-01-03	3824.14
2022-12-30	3839.50
2022-12-29	3849.28
2022-12-28	3783.22
2022-12-27	3829.25

```
trainig_set= df.iloc[:1000,:].values
```

```
test_set= df.iloc[1000:,:].values
```

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

```
# Create a data structure with 60 timesteps and 1 output
X_train=[] #Independent variables
y_train= [] # Dependent variables
# I am going to append past 60 days data
for i in range(60,1000):
    X_train.append(trainig_set_scaled[i-60:i,0]) # Appending prevois 60 days data not including 60
    y_train.append(trainig_set_scaled[i,0])

X_train, y_train= np.array(X_train), np.array(y_train)
```

```
# lETS CHECK THE SHAPE OF X_train and y_train
X_train.shape, y_train.shape
```

```
((940, 60), (940,))
```

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

```
(940, 60, 1)
```

```
# Importing the Keras libraries and packages
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM
```

```
model = Sequential()
model.add(LSTM(100, return_sequences=True, input_shape= (X_train.shape[1], 1)))
model.add(LSTM(50, return_sequences=False))
model.add(Dense(25))
model.add(Dense(1))

#  Initialising the RNN
# model= Sequential()

# # Adding first LSTM layer and some dropout Dropout regularisation
# model.add(LSTM(units=100,return_sequences=True, input_shape=(X_train.shape[1],1)))
# model.add(Dropout(rate=0.2))

# # Adding second LSTM layer and some dropout Dropout regularisation
# model.add(LSTM(units=100,return_sequences=True))
# model.add(Dropout(rate=0.2))

# # Adding third LSTM layer and some dropout Dropout regularisation
# model.add(LSTM(units=100,return_sequences=True))
# model.add(Dropout(rate=0.2))

# # Adding fourth LSTM layer and some dropout Dropout regularisation
# model.add(LSTM(units=100,return_sequences=True))
# model.add(Dropout(rate=0.2))

# # Adding fifth LSTM layer and some dropout Dropout regularisation
# model.add(LSTM(units=100))
# model.add(Dropout(rate=0.2))

# # Adding the Output Layer
# model.add(Dense(units=1))

# Compiling the Model
# Because we're doing regression hence mean_squared_error
model.compile(loss='mean_squared_error', optimizer='adam')

model.summary()
```

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_42 (LSTM)              (None, 60, 100)           40800

 lstm_43 (LSTM)              (None, 50)                30200

 dense_10 (Dense)            (None, 25)                1275

 dense_11 (Dense)            (None, 1)                 26
```

```
=================================================================
Total params: 72,301
Trainable params: 72,301
Non-trainable params: 0
_________________________________________________________________
```

```
history=model.fit(X_train,y_train,epochs=100,batch_size=20)
```

```
Epoch 60/100
47/47 [==============================] - 3s 72ms/step - loss: 3.6673e-04
Epoch 61/100
47/47 [==============================] - 4s 80ms/step - loss: 3.8204e-04
Epoch 62/100
47/47 [==============================] - 4s 91ms/step - loss: 4.7341e-04
Epoch 63/100
47/47 [==============================] - 3s 67ms/step - loss: 3.8527e-04
Epoch 64/100
47/47 [==============================] - 3s 69ms/step - loss: 3.7659e-04
Epoch 65/100
47/47 [==============================] - 4s 82ms/step - loss: 3.6099e-04
Epoch 66/100
47/47 [==============================] - 4s 76ms/step - loss: 3.6702e-04
Epoch 67/100
47/47 [==============================] - 3s 70ms/step - loss: 4.1394e-04
Epoch 68/100
47/47 [==============================] - 3s 70ms/step - loss: 3.9597e-04
Epoch 69/100
47/47 [==============================] - 4s 91ms/step - loss: 3.4978e-04
Epoch 70/100
47/47 [==============================] - 3s 68ms/step - loss: 4.1599e-04
Epoch 71/100
47/47 [==============================] - 3s 70ms/step - loss: 3.6394e-04
Epoch 72/100
47/47 [==============================] - 3s 69ms/step - loss: 3.9613e-04
Epoch 73/100
47/47 [==============================] - 4s 90ms/step - loss: 3.6675e-04
Epoch 74/100
47/47 [==============================] - 3s 68ms/step - loss: 4.8270e-04
Epoch 75/100
47/47 [==============================] - 3s 68ms/step - loss: 4.2554e-04
Epoch 76/100
47/47 [==============================] - 4s 80ms/step - loss: 3.9963e-04
Epoch 77/100
47/47 [==============================] - 4s 81ms/step - loss: 3.7054e-04
Epoch 78/100
47/47 [==============================] - 3s 71ms/step - loss: 4.0292e-04
Epoch 79/100
47/47 [==============================] - 3s 70ms/step - loss: 4.0501e-04
Epoch 80/100
47/47 [==============================] - 4s 90ms/step - loss: 4.0773e-04
Epoch 81/100
47/47 [==============================] - 3s 69ms/step - loss: 4.0279e-04
Epoch 82/100
47/47 [==============================] - 3s 69ms/step - loss: 4.9918e-04
Epoch 83/100
47/47 [==============================] - 3s 69ms/step - loss: 3.6013e-04
Epoch 84/100
47/47 [==============================] - 4s 91ms/step - loss: 3.5858e-04
Epoch 85/100
47/47 [==============================] - 3s 69ms/step - loss: 3.6052e-04
Epoch 86/100
47/47 [==============================] - 3s 71ms/step - loss: 3.5254e-04
Epoch 87/100
47/47 [==============================] - 4s 77ms/step - loss: 4.2790e-04
Epoch 88/100
47/47 [==============================] - 4s 82ms/step - loss: 4.1132e-04
Epoch 89/100
```

```
data_to_train.to_csv('train_data.csv')
data_to_test.to_csv('test_data.csv')
```

```
#GEtting ready both train and est data set
train_data= pd.read_csv('train_data.csv')
test_data= pd.read_csv('test_data.csv')
```

```
real_stock_price = test_data.iloc[:, 6:7].values
```

```
train_data
```

	Date	Open	High	Low	Volume	Return	Close	Adj Close
0	2023-01-03	3853.29	3878.46	3794.33	3.959140e+09	-15.36	3824.14	3824.14
1	2022-12-30	3829.06	3839.85	3800.34	2.979870e+09	-9.78	3839.50	3839.50
2	2022-12-29	3805.45	3858.19	3805.45	3.003680e+09	66.06	3849.28	3849.28
3	2022-12-28	3829.56	3848.32	3780.78	3.083520e+09	-46.03	3783.22	3783.22
4	2022-12-27	3843.34	3846.65	3813.22	3.030300e+09	-15.57	3829.25	3829.25
...	...	...	...	...	...	...	...	...
995	2019-01-22	2657.88	2657.88	2617.27	3.923950e+09	-37.81	2632.90	2632.90
996	2019-01-18	2651.27	2675.47	2647.58	4.009010e+09	34.75	2670.71	2670.71
997	2019-01-17	2609.28	2645.06	2606.36	3.802410e+09	19.86	2635.96	2635.96
998	2019-01-16	2614.75	2625.76	2612.68	3.882180e+09	5.80	2616.10	2616.10
999	2019-01-15	2585.10	2613.08	2585.10	3.601180e+09	27.69	2610.30	2610.30

1000 rows × 8 columns

```
real_stock_price.shape
```

```
(260, 1)
```

```
test_set.shape
```

```
(260, 1)
```

```
data_total= pd.concat([train_data['Close'], test_data['Close']],  axis=0)
inputs= data_total[len(data_total)-len(test_data)-60:].values
inputs = inputs.reshape(-1,1)
inputs = sc.transform(inputs)

X_test = []
for i in range(60, 320):
    X_test.append(inputs[i-60:i, 0])

X_test = np.array(X_test)
# 3D format
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
```

```
predicted_stock_price = model.predict(X_test)
```

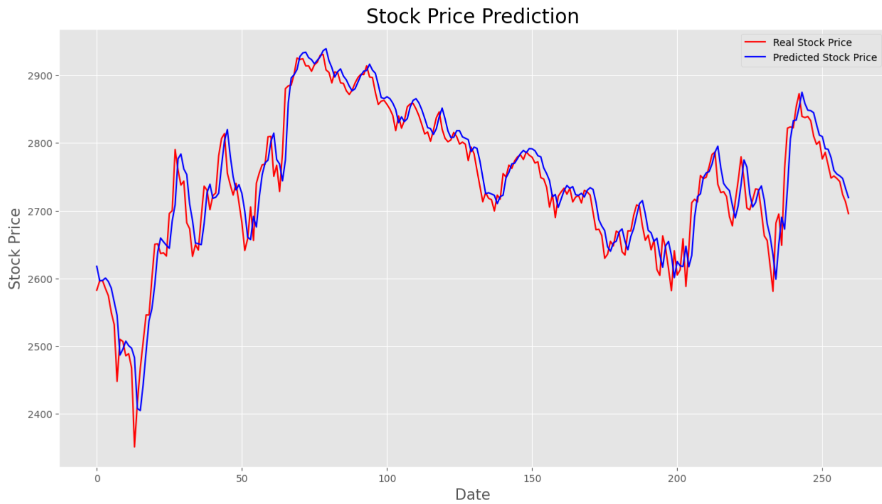
```
9/9 [==============================] - 0s 28ms/step
```

```
# Inverse the scaling
predicted_stock_price = sc.inverse_transform(predicted_stock_price)
```

```
real_stock_price.shape
```

```
(260, 1)
```

```
# Visualising the results
plt.figure(figsize=(15,8))
plt.plot(real_stock_price, color='Red', label='Real Stock Price')
plt.plot(predicted_stock_price, color='Blue', label='Predicted Stock Price')
plt.title('Stock Price Prediction',fontsize=20)
plt.xlabel('Date', fontsize=15)
plt.ylabel('Stock Price',fontsize=15)
plt.legend()
plt.show()
```




```
from sklearn.metrics import mean_squared_error, mean_absolute_error
mse = mean_squared_error(real_stock_price, predicted_stock_price)
mae = mean_absolute_error(real_stock_price, predicted_stock_price)

print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
```

```
Mean Squared Error (MSE): 973.49
Mean Absolute Error (MAE): 23.27
```

```
real_stock_price = real_stock_price.flatten()
predicted_stock_price = predicted_stock_price.flatten()
```

```
quick_test = pd.DataFrame({'Actual': real_stock_price, 'Predicted' : predicted_stock_price})
quick_test.head(20)
```

	Actual	Predicted
0	2582.61	2617.868408
1	2596.26	2596.710938

```
quick_test.dropna(inplace=True)
quick_test.to_csv('form1.csv',index=False)
# pd.read_csv('form1.csv').info()
set = pd.read_csv('form1.csv')
```

```
set.head(10)
```

	Actual	Predicted
0	2582.61	2617.8684
1	2596.26	2596.7110
2	2596.64	2596.9546
3	2584.96	2600.5825
4	2574.41	2595.3936
5	2549.69	2585.6000
6	2531.94	2565.6277
7	2447.89	2545.5754
8	2510.03	2486.7786
9	2506.85	2495.6592

```
set['real_stock_price_Return'] = set.Actual.diff()
set['predicted_stock_price_Return'] = set.Predicted.diff()
```

```
real_stock_price_Return = set.Actual.diff()
# real_stock_price_Return.dropna(inplace=True)
real_stock_price_Return.fillna(method='bfill', inplace=True)
real_stock_price_Return
```

```
0      13.65
1      13.65
2       0.38
3     -11.68
4     -10.55
        ...
255    -3.58
256    -4.56
257   -19.16
258   -10.93
259   -17.25
Name: Actual, Length: 260, dtype: float64
```

```
predicted_stock_price_Return = set.Predicted.diff()
predicted_stock_price_Return.fillna(method='bfill', inplace=True)
```

```
real_stock_price_Return = real_stock_price_Return.iloc[1:]
# predicted_stock_price_Return = predicted_stock_price_Return.iloc[1:]
```

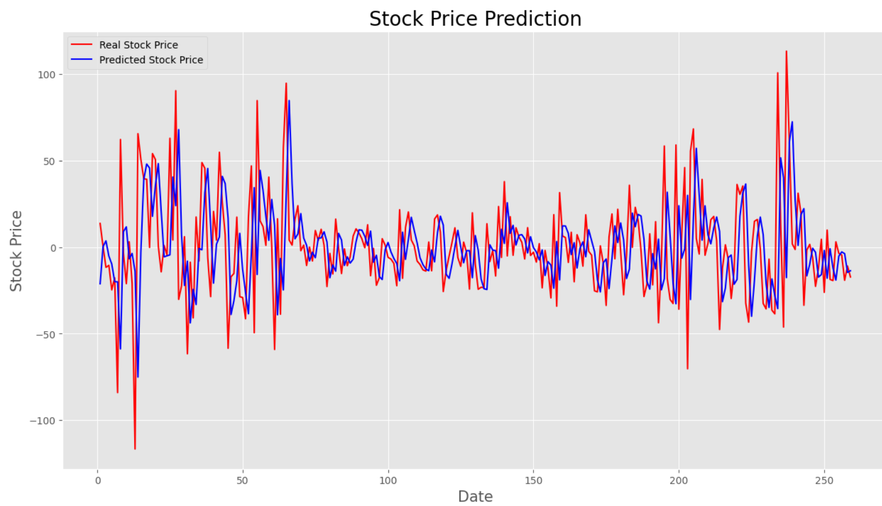
```
predicted_stock_price_Return
```

```
1     -21.1574
2       0.2436
3       3.6279
4      -5.1889
5      -9.7936
         ...
255    -5.2710
256    -2.7808
257    -3.7597
258   -14.6175
259   -13.6679
Name: Predicted, Length: 259, dtype: float64
```

```
real_stock_price_Return
```

```
1       13.65
2        0.38
3      -11.68
4      -10.55
5      -24.72
        ...
255     -3.58
256     -4.56
257    -19.16
258    -10.93
259    -17.25
Name: Actual, Length: 259, dtype: float64
```

```
# Visualising the results
plt.figure(figsize=(15,8))
plt.plot(real_stock_price_Return, color='Red', label='Real Stock Price')
plt.plot(predicted_stock_price_Return, color='Blue', label='Predicted Stock Price')
plt.title('Stock Price Prediction',fontsize=20)
plt.xlabel('Date', fontsize=15)
plt.ylabel('Stock Price',fontsize=15)
plt.legend()
plt.show()
```




```
from sklearn.metrics import mean_squared_error, mean_absolute_error
mse = mean_squared_error(real_stock_price_Return, predicted_stock_price_Return)
mae = mean_absolute_error(real_stock_price_Return, predicted_stock_price_Return)

print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
```

```
Mean Squared Error (MSE): 1320.91
Mean Absolute Error (MAE): 25.98
```

```
df = pd.read_csv('fin.csv',parse_dates = True,index_col=0)
df = df.drop(['Price_change'], axis='columns')
df.rename(columns={'Change': 'Return'}, inplace=True)
df
```

Date	Return	ma20	S&P500	Close	Positive	Negative	Neutral	Volume
2018-01-02	22.20	2788.1035	2695.81	2695.81	0.255685	0.267352	0.476963	3397430000
2018-01-03	17.25	2794.5035	2713.06	2713.06	0.203532	0.295476	0.500992	3544030000
2018-01-04	10.93	2799.9495	2723.99	2723.99	0.201664	0.272806	0.525530	3697340000
2018-01-05	19.16	2801.8565	2743.15	2743.15	0.134174	0.150020	0.715806	3239280000
2018-01-08	4.56	2797.1460	2747.71	2747.71	0.281969	0.176601	0.541430	3246160000
...	...	...	...	...	...	...	...	...
2020-05-08	48.61	2984.3185	2929.80	2929.80	0.314460	0.095021	0.590519	4876030000
2020-05-11	0.39	2999.4480	2930.32	2930.19	0.184708	0.111766	0.703526	4819730000
2020-05-12	-60.07	3013.2975	2870.12	2870.12	0.160295	0.077055	0.762650	5119630000
2020-05-13	-50.12	3029.2985	2820.00	2820.00	0.217852	0.200098	0.582050	6151650000
2020-05-14	32.50	3038.4035	2852.50	2852.50	0.202053	0.104108	0.693839	5651130000

596 rows × 8 columns

```
data_to_train = df[:500]
data_to_test = df[500:]
```

```
df= df.iloc[: , 0:1]
df.head()
```

Date	Return
2018-01-02	22.20
2018-01-03	17.25
2018-01-04	10.93
2018-01-05	19.16
2018-01-08	4.56

```
trainig_set= df.iloc[:500,:].values
test_set= df.iloc[500:,:].values
```

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

```
# Create a data structure with 60 timesteps and 1 output
X_train=[] #Independent variables
y_train= [] # Dependent variables
# I am going to append past 60 days data
for i in range(60,500):
    X_train.append(trainig_set_scaled[i-60:i,0]) # Appending prevois 60 days data not including 60
    y_train.append(trainig_set_scaled[i,0])

X_train, y_train= np.array(X_train), np.array(y_train)
```

```
# lETS CHECK THE SHAPE OF X_train and y_train
X_train.shape, y_train.shape
```

```
((440, 60), (440,))
```

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

```
(440, 60, 1)
```

```
# Importing the Keras libraries and packages
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM

#  Initialising the RNN
model= Sequential()

# Adding first LSTM layer and some dropout Dropout regularisation
model.add(LSTM(units=100,return_sequences=True, input_shape=(X_train.shape[1],1)))
model.add(Dropout(rate=0.2))

# Adding second LSTM layer and some dropout Dropout regularisation
model.add(LSTM(units=100,return_sequences=True))
model.add(Dropout(rate=0.2))

# Adding third LSTM layer and some dropout Dropout regularisation
model.add(LSTM(units=100,return_sequences=True))
model.add(Dropout(rate=0.2))

# Adding fourth LSTM layer and some dropout Dropout regularisation
model.add(LSTM(units=100,return_sequences=True))
model.add(Dropout(rate=0.2))

# Adding fifth LSTM layer and some dropout Dropout regularisation
model.add(LSTM(units=100))
model.add(Dropout(rate=0.2))

# Adding the Output Layer
model.add(Dense(units=1))

# Compiling the Model
# Because we're doing regression hence mean_squared_error
model.compile(loss='mean_squared_error', optimizer='adam')

model.summary()

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

```
Epoch 75/100
14/14 [==============================] - 4s 308ms/step - loss: 0.0120
Epoch 76/100
14/14 [==============================] - 5s 374ms/step - loss: 0.0126
Epoch 77/100
14/14 [==============================] - 4s 303ms/step - loss: 0.0125
Epoch 78/100
14/14 [==============================] - 5s 356ms/step - loss: 0.0128
Epoch 79/100
14/14 [==============================] - 5s 323ms/step - loss: 0.0122
Epoch 80/100
14/14 [==============================] - 4s 308ms/step - loss: 0.0119
Epoch 81/100
14/14 [==============================] - 5s 399ms/step - loss: 0.0125
Epoch 82/100
14/14 [==============================] - 4s 307ms/step - loss: 0.0127
Epoch 83/100
14/14 [==============================] - 4s 305ms/step - loss: 0.0124
Epoch 84/100
14/14 [==============================] - 5s 394ms/step - loss: 0.0122
Epoch 85/100
14/14 [==============================] - 4s 303ms/step - loss: 0.0121
Epoch 86/100
```

```
data_to_train.to_csv('train_data.csv')
data_to_test.to_csv('test_data.csv')
```

```
train_data= pd.read_csv('train_data.csv')
test_data= pd.read_csv('test_data.csv')
```

```
test_data
```

	Date	Return	ma20	S&P500	Close	Positive	Negative	Neutral	Volume
0	2019-12-27	0.11	3273.7765	3240.02	3240.02	0.274682	0.302670	0.422647	2429150000
1	2019-12-30	-18.73	3275.5875	3221.29	3221.29	0.273029	0.284337	0.442633	3021720000
2	2019-12-31	9.49	3278.1930	3230.78	3230.78	0.335628	0.111374	0.552998	2894760000
3	2020-01-02	27.07	3280.8370	3257.85	3257.85	0.419117	0.172851	0.408032	3459930000
4	2020-01-03	-23.00	3279.2205	3234.85	3234.85	0.323531	0.188997	0.487472	3484700000
...	...	...	...	...	...	...	...	...	...
91	2020-05-08	48.61	2984.3185	2929.80	2929.80	0.314460	0.095021	0.590519	4876030000
92	2020-05-11	0.39	2999.4480	2930.32	2930.19	0.184708	0.111766	0.703526	4819730000

```
real_stock_price = test_data.iloc[:, 1:2].values
```

```
real_stock_price.shape
```

```
(96, 1)
```

```
test_set.shape
```

```
(96, 1)
```

```
data_total= pd.concat([train_data['Return'], test_data['Return']],  axis=0)
inputs= data_total[len(data_total)-len(test_data)-60:].values
inputs = inputs.reshape(-1,1)
inputs = sc.transform(inputs)

X_test = []
for i in range(60, 156):
    X_test.append(inputs[i-60:i, 0])
```

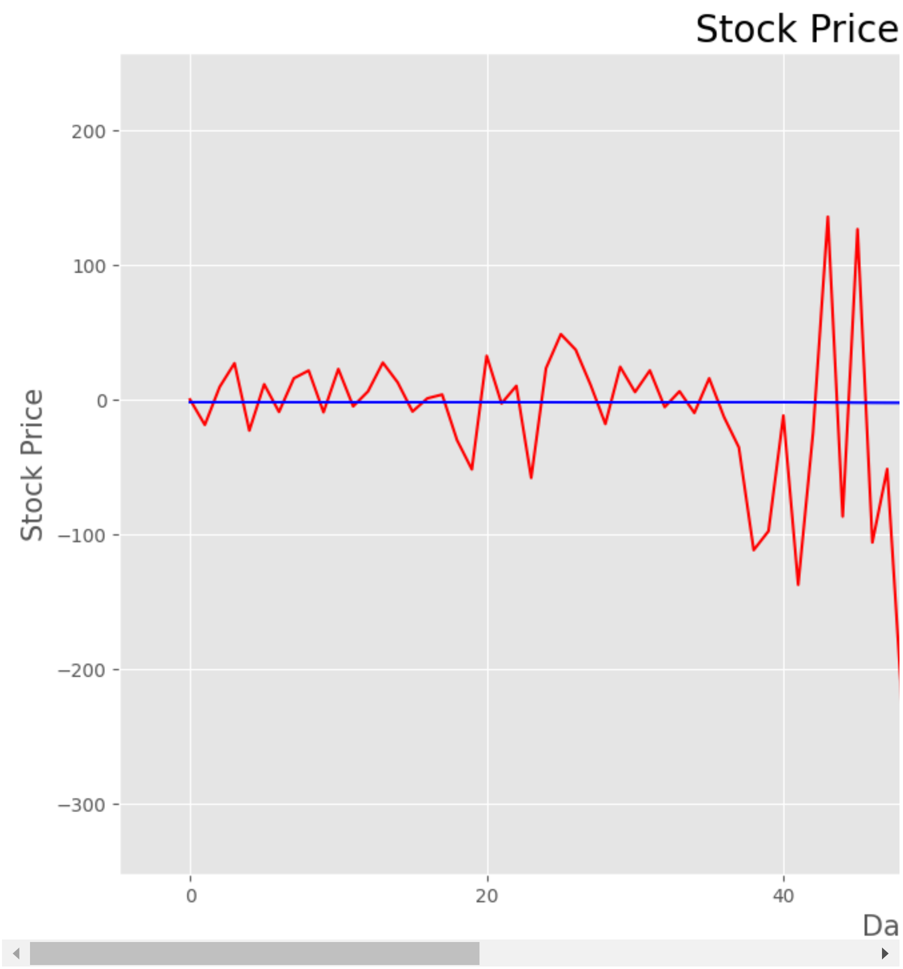
```
X_test = np.array(X_test)
# 3D format
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
```

```
predicted_stock_price = model.predict(X_test)
```

```
3/3 [==============================] - 2s 93ms/step
```

```
# Inverse the scaling
predicted_stock_price = sc.inverse_transform(predicted_stock_price)
```

```
# Visualising the results
plt.figure(figsize=(15,8))
plt.plot(real_stock_price, color='Red', label='Real Stock Price')
plt.plot(predicted_stock_price, color='Blue', label='Predicted Stock Price')
plt.title('Stock Price Prediction',fontsize=20)
plt.xlabel('Date', fontsize=15)
plt.ylabel('Stock Price',fontsize=15)
plt.legend()
plt.show()
```



sfbsnsb

```
df.drop(df[df['Volume']==0].index, inplace = True)
```

```
# Setting up an early stop
earlystop = EarlyStopping(monitor='val_loss', min_delta=0.0001, patience=80,  verbose=1, mode='min')
callbacks_list = [earlystop]
```

```
#Build and train the model
def fit_model(train,val,timesteps,hl,lr,batch,epochs):
    X_train = []
```

```
    Y_train = []
    X_val = []
    Y_val = []

    # Loop for training data
    for i in range(timesteps,train.shape[0]):
        X_train.append(train[i-timesteps:i])
        Y_train.append(train[i][0])
    X_train,Y_train = np.array(X_train),np.array(Y_train)

    # Loop for val data
    for i in range(timesteps,val.shape[0]):
        X_val.append(val[i-timesteps:i])
        Y_val.append(val[i][0])
    X_val,Y_val = np.array(X_val),np.array(Y_val)

    # Adding Layers to the model
    model = Sequential()
    model.add(LSTM(X_train.shape[2],input_shape = (X_train.shape[1],X_train.shape[2]),return_sequences = True,
                   activation = 'relu'))
    for i in range(len(hl)-1):
        model.add(LSTM(hl[i],activation = 'relu',return_sequences = True))
    model.add(LSTM(hl[-1],activation = 'relu'))
    model.add(Dense(1))
    model.compile(optimizer = optimizers.Adam(lr = lr), loss = 'mean_squared_error')
    #print(model.summary())

    # Training the data
    history = model.fit(X_train,Y_train,epochs = epochs,batch_size = batch,validation_data = (X_val, Y_val),verbose = 0,
                        shuffle = False, callbacks=callbacks_list)
    model.reset_states()
    return model, history.history['loss'], history.history['val_loss']

    def evaluate_model(model,test,timesteps):
      X_test = []
      Y_test = []

    # Loop for testing data
    for i in range(timesteps,test.shape[0]):
        X_test.append(test[i-timesteps:i])
        Y_test.append(test[i][0])
    X_test,Y_test = np.array(X_test),np.array(Y_test)
    #print(X_test.shape,Y_test.shape)

    # Prediction Time !!!!
    Y_hat = model.predict(X_test)
    mse = mean_squared_error(Y_test,Y_hat)
    rmse = sqrt(mse)
    r = r2_score(Y_test,Y_hat)
    return mse, rmse, r, Y_test, Y_hat

    def plot_data(Y_test,Y_hat):
      plt.plot(Y_test,c = 'r')
      plt.plot(Y_hat,c = 'y')
      plt.xlabel('Day')
      plt.ylabel('Price')
      plt.title('Stock Prediction Graph using Multivariate-LSTM model')
      plt.legend(['Actual','Predicted'],loc = 'lower right')
      plt.show()


# Plotting the training errors
def plot_error(train_loss,val_loss):
    plt.plot(train_loss,c = 'r')
    plt.plot(val_loss,c = 'b')
    plt.ylabel('Loss')
    plt.legend(['train','val'],loc = 'upper right')
    plt.show()

series = df[['Close','High','Low']] # Picking the series with high correlation
print(series.shape)
print(series.tail())
```

```
(1260, 3)
              Close     High      Low
Date
2018-01-08  2747.71  2748.51  2737.60
```

```
2018-01-05  2743.15  2743.45  2727.92
2018-01-04  2723.99  2729.29  2719.07
2018-01-03  2713.06  2714.37  2697.77
2018-01-02  2695.81  2695.89  2682.36
```

```
train_start = dt.date(2018, 1, 2)
train_end = dt.date(2019, 10, 31)

val_start = dt.date(2019, 11, 1)
val_end = dt.date(2019, 11, 21)

test_start = dt.date(2019, 11, 22)
test_end = dt.date(2019, 12, 26)

# Convert date objects to datetime objects
train_start_dt = pd.to_datetime(train_start)
train_end_dt = pd.to_datetime(train_end)

val_start_dt = pd.to_datetime(val_start)
val_end_dt = pd.to_datetime(val_end)

test_start_dt = pd.to_datetime(test_start)
test_end_dt = pd.to_datetime(test_end)

# Split the data using boolean indexing
train_data = series[(series.index >= train_start_dt) & (series.index <= train_end_dt)]
val_data = series[(series.index >= val_start_dt) & (series.index <= val_end_dt)]
test_data = series[(series.index >= test_start_dt) & (series.index <= test_end_dt)]

print(train_data.shape, val_data.shape, test_data.shape)
```

```
(462, 3) (15, 3) (23, 3)
```

```
sc = MinMaxScaler()
train = sc.fit_transform(train_data)
val = sc.transform(val_data)
test = sc.transform(test_data)
print(train.shape,val.shape,test.shape)
```

```
(462, 3) (15, 3) (23, 3)
```

```
timesteps = 50
hl = [40,35]
lr = 1e-3
batch_size = 64
num_epochs = 250

model,train_error,val_error = fit_model(train,val,timesteps,hl,lr,batch_size,num_epochs)
plot_error(train_error,val_error)
```

```
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not avai
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-424-0437dec82bcd> in <cell line: 7>()
      5 num_epochs = 250
      6 
----> 7 model,train_error,val_error =
fit_model(train,val,timesteps,hl,lr,batch_size,num_epochs)
      8 plot_error(train_error,val_error)
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from keras.models import Sequential, load_model
from keras.layers.core import Dense
from keras.layers import LSTM
from keras import optimizers
from keras.callbacks import EarlyStopping
```

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, r2_score
from math import sqrt
import datetime as dt
import time
plt.style.use('ggplot')
```

```
data_training = pd.read_csv('new_train.csv')
data_training = data_training.drop(['Date'], axis='columns')
```

```
scaler = MinMaxScaler()
data_training_scaled = scaler.fit_transform(data_training)
print(data_training_scaled.shape)
```

```
(500, 12)
```

```
data_training_scaled
```

```
array([[0.56451543, 1.        , 1.        , ..., 1.        , 1.        ,
        0.83333333],
       [0.48983855, 0.99523028, 0.9814021 , ..., 1.        , 1.        ,
        0.76666667],
       [0.5047217 , 0.98851272, 0.98211091, ..., 1.        , 1.        ,
        0.73333333],
       ...,
       [0.54014535, 0.37324659, 0.41953848, ..., 0.        , 0.        ,
        0.1       ],
       [0.56764872, 0.36604008, 0.40724114, ..., 0.        , 0.        ,
        0.06666667],
       [0.58919013, 0.35757117, 0.38783317, ..., 0.        , 0.        ,
        0.03333333]])
```

```
X_train = []
y_train = []

for i in range(60, data_training.shape[0]):
    X_train.append(data_training_scaled[i-60: i])
    y_train.append(data_training_scaled[i, 0])
```

```
X_train, y_train = np.array(X_train), np.array(y_train)
X_train.shape, y_train.shape
```

```
((440, 60, 12), (440,))
```

```
regressor = Sequential()

regressor.add(LSTM(units = 50, activation = 'relu', return_sequences = True, input_shape = (X_train.shape[1], 12)))
regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 60, activation = 'relu', return_sequences = True))
regressor.add(Dropout(0.3))

regressor.add(LSTM(units = 80, activation = 'relu', return_sequences = True))
regressor.add(Dropout(0.4))

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

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

```
regressor.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_1 (LSTM)               (None, 60, 50)            12600

 dropout (Dropout)           (None, 60, 50)            0

 lstm_2 (LSTM)               (None, 60, 60)            26640

 dropout_1 (Dropout)         (None, 60, 60)            0

 lstm_3 (LSTM)               (None, 60, 80)            45120
```

```
 dropout_2 (Dropout)         (None, 60, 80)            0

 lstm_4 (LSTM)               (None, 120)               96480

 dropout_3 (Dropout)         (None, 120)               0

 dense_1 (Dense)             (None, 1)                 121

=================================================================
Total params: 180,961
Trainable params: 180,961
Non-trainable params: 0
_________________________________________________________________
```

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

```
regressor.fit(X_train, y_train, epochs=50, batch_size = 32)
```

```
Epoch 1/50
14/14 [==============================] - 67s 304ms/step - loss: 0.0940
Epoch 2/50
14/14 [==============================] - 3s 206ms/step - loss: 0.0273
Epoch 3/50
14/14 [==============================] - 3s 209ms/step - loss: 0.0216
Epoch 4/50
14/14 [==============================] - 3s 209ms/step - loss: 0.0211
Epoch 5/50
14/14 [==============================] - 4s 314ms/step - loss: 0.0212
Epoch 6/50
14/14 [==============================] - 3s 207ms/step - loss: 0.0206
Epoch 7/50
14/14 [==============================] - 3s 208ms/step - loss: 0.0182
Epoch 8/50
14/14 [==============================] - 3s 208ms/step - loss: 0.0189
Epoch 9/50
14/14 [==============================] - 4s 313ms/step - loss: 0.0187
Epoch 10/50
14/14 [==============================] - 3s 207ms/step - loss: 0.0182
Epoch 11/50
14/14 [==============================] - 3s 207ms/step - loss: 0.0192
Epoch 12/50
14/14 [==============================] - 3s 208ms/step - loss: 0.0187
Epoch 13/50
14/14 [==============================] - 4s 318ms/step - loss: 0.0177
Epoch 14/50
14/14 [==============================] - 3s 208ms/step - loss: 0.0183
Epoch 15/50
14/14 [==============================] - 4s 315ms/step - loss: 0.0184
Epoch 16/50
14/14 [==============================] - 3s 235ms/step - loss: 0.0172
Epoch 17/50
14/14 [==============================] - 4s 272ms/step - loss: 0.0183
Epoch 18/50
14/14 [==============================] - 3s 208ms/step - loss: 0.0178
Epoch 19/50
14/14 [==============================] - 3s 208ms/step - loss: 0.0173
Epoch 20/50
14/14 [==============================] - 3s 238ms/step - loss: 0.0173
Epoch 21/50
14/14 [==============================] - 4s 269ms/step - loss: 0.0177
Epoch 22/50
14/14 [==============================] - 3s 210ms/step - loss: 0.0171
Epoch 23/50
14/14 [==============================] - 3s 207ms/step - loss: 0.0183
Epoch 24/50
14/14 [==============================] - 3s 251ms/step - loss: 0.0169
Epoch 25/50
14/14 [==============================] - 4s 260ms/step - loss: 0.0176
Epoch 26/50
14/14 [==============================] - 3s 208ms/step - loss: 0.0175
Epoch 27/50
14/14 [==============================] - 3s 209ms/step - loss: 0.0159
Epoch 28/50
14/14 [==============================] - 4s 254ms/step - loss: 0.0166
Epoch 29/50
14/14 [==============================] - 4s 260ms/step - loss: 0.0171
```

```
past_60 = data_training.tail(60)
data_testing = pd.read_csv('new_test.csv')
data_testing = data_testing.drop(['Date'], axis='columns')
dt = past_60.append(data_testing, ignore_index = True)
```

```
dt
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-fad2275e92fe> in <cell line: 1>()
----> 1 past_60 = data_training.tail(60)
      2 data_testing = pd.read_csv('new_test.csv')
      3 data_testing = data_testing.drop(['Date'], axis='columns')
      4 dt = past_60.append(data_testing, ignore_index = True)
      5 dt

NameError: name 'data_training' is not defined
```

SEARCH STACK OVERFLOW

```
inputs = scaler.fit_transform(dt)
print(inputs.shape)
inputs
```

```
(156, 12)
array([[0.57137969, 0.16557567, 0.32      , ..., 0.18181818, 0.        ,
        0.9       ],
       [0.50238623, 0.16391752, 0.3266333 , ..., 0.18181818, 0.        ,
        0.86666667],
       [0.71168981, 0.16540292, 0.36661589, ..., 0.18181818, 0.        ,
        0.83333333],
       ...,
       [0.60219353, 0.94430121, 0.86474864, ..., 1.        , 0.5       ,
        1.        ],
       [0.5513714 , 0.94107467, 0.85648749, ..., 1.        , 0.5       ,
        0.96666667],
       [0.58530084, 0.93883201, 0.87279217, ..., 1.        , 0.5       ,
        0.86666667]])
```

```
X_test = []
y_test = []

for i in range(60, inputs.shape[0]):
    X_test.append(inputs[i-60:i])
    y_test.append(inputs[i, 0])

X_test, y_test = np.array(X_test), np.array(y_test)
X_test.shape, y_test.shape
```

```
((96, 60, 12), (96,))
```

```
y_pred = regressor.predict(X_test)
```

```
3/3 [==============================] - 1s 69ms/step
```

```
scale = 1/scaler.scale_[0]
```

```
y_pred = y_pred*scale
y_test = y_test*scale
```

```
y_test
```

```
array([357.39, 274.77, 264.82, 325.28, 373.5 , 357.66, 304.87, 350.59,
       336.92, 243.17, 297.81, 401.01, 309.8 , 366.63, 363.83, 323.38,
       387.64, 238.29, 273.49, 399.9 , 341.08, 262.19, 409.32, 296.7 ,
       364.73, 415.46, 320.62, 499.92, 286.64, 381.29, 210.8 , 282.83,
       410.07, 236.29, 479.4 , 353.12, 534.82, 257.37, 220.42, 336.18,
       193.8 , 467.95,   0.  , 555.27,  64.15, 184.04, 460.56,  99.08,
       273.32, 218.71, 451.64, 238.03, 460.9 , 300.35, 187.26, 313.07,
       227.21, 213.03, 289.41, 311.97, 340.75, 315.02, 331.11, 319.38,
       346.59, 330.55, 349.27, 306.82, 335.98, 361.99, 373.56, 348.29,
       266.75, 335.15, 322.05, 357.5 , 273.05, 294.82, 328.68, 325.85,
       316.06, 337.7 , 352.41, 331.03, 319.91, 347.67, 315.54, 346.54,
       340.76, 315.79, 336.32, 301.89, 351.96, 334.38, 306.16, 325.  ])
```

```
y_pred
```

```
array([[281.74115],
       [280.89377],
       [279.95258],
```