Files

main

Go to file t

static
templates
README.md
Stock Price Prediction .ipynb
app.py
powergrid.csv
stock_dl_model.h5

stock_price_prediction / Stock Price Prediction .ipynb

Chando0185 Add files via upload 90a0334 · 9 months ago History

Preview Code Blame 23778 lines (23778 loc) · 1.78 MB Raw

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pandas_datareader as data

plt.style.use('fivethirtyeight')
%matplotlib inline
```

```
d:\python38\lib\site-packages\numpy\_distributor_init.py:30: UserWarning: loaded more than 1 DLL from .libs:
d:\python38\lib\site-packages\numpy\.libs\libopenblas.4SP5SUA7CBGXUEOC35YP2ASOICYYEQZZ.gfortran-win_amd64.dll
d:\python38\lib\site-packages\numpy\.libs\libopenblas.FB5AE2TYXYH2IJRDKGDGQ3XBKLKTF43H.gfortran-win_amd64.dll
d:\python38\lib\site-packages\numpy\.libs\libopenblas.WCDJNK7YVMPZQ2ME2ZZHJJRJ3JIKNDB7.gfortran-win_amd64.dll
d:\python38\lib\site-packages\numpy\.libs\libopenblas64__v0.3.21-gcc_10_3_0.dll
  warnings.warn("loaded more than 1 DLL from .libs:"
```

In [2]:
```
import yfinance as yf
import datetime as dt

stock = "POWERGRID.NS"
start = dt.datetime(2000, 1, 1)
end = dt.datetime(2024, 11, 1)

df = yf.download(stock, start, end)
```

```
[*********************100%**********************]  1 of 1 completed
```

In [6]:
```
df.head()
```

Out[6]:

Price	Adj Close	Close	High	Low	Open	Volume
Ticker	POWERGRID.NS	POWERGRID.NS	POWERGRID.NS	POWERGRID.NS	POWERGRID.NS	POWERGRID.NS
Date						
2007-10-05 00:00:00+00:00	32.926037	56.587513	61.593765	46.771885	50.512512	855215656
2007-10-08 00:00:00+00:00	31.240450	53.690639	58.500015	53.353138	58.500015	126671715
2007-10-09 00:00:00+00:00	33.367889	57.346889	57.853138	50.821888	53.718761	116725709
2007-10-10 00:00:00+00:00	33.678829	57.881264	59.062515	57.375015	58.837513	67931378
2007-10-11 00:00:00+00:00	37.442734	64.350014	67.500015	57.375015	67.500015	106320954

In [7]:
```
df.shape
```

Out[7]: (4208, 6)

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 4208 entries, 2007-10-05 00:00:00+00:00 to 2024-10-31 00:00:00+00:00
Data columns (total 6 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   (Adj Close, POWERGRID.NS)  4208 non-null   float64
 1   (Close, POWERGRID.NS)      4208 non-null   float64
 2   (High, POWERGRID.NS)       4208 non-null   float64
 3   (Low, POWERGRID.NS)        4208 non-null   float64
 4   (Open, POWERGRID.NS)       4208 non-null   float64
 5   (Volume, POWERGRID.NS)     4208 non-null   int64
dtypes: float64(5), int64(1)
memory usage: 230.1 KB
```

In [9]:
```
df.isnull().sum()
```

Out[9]:
```
Price      Ticker
Adj Close  POWERGRID.NS    0
Close      POWERGRID.NS    0
High       POWERGRID.NS    0
Low        POWERGRID.NS    0
Open       POWERGRID.NS    0
Volume     POWERGRID.NS    0
dtype: int64
```

In [10]:
```
df.describe()
```

Out[10]:

Price	Adj Close	Close	High	Low	Open	Volume
Ticker	POWERGRID.NS	POWERGRID.NS	POWERGRID.NS	POWERGRID.NS	POWERGRID.NS	POWERGRID.NS
count	4208.000000	4208.000000	4208.000000	4208.000000	4208.000000	4.208000e+03
mean	80.413097	104.165058	105.540691	102.811225	104.222802	1.188963e+07
std	64.314025	60.546190	61.222254	59.830531	60.527683	2.061999e+07
min	19.223955	32.625008	34.875008	29.250008	34.706257	0.000000e+00
25%	36.956319	60.546108	61.284389	59.737514	60.609390	4.799653e+06
50%	56.154064	84.937523	86.779709	83.657833	85.289085	8.344398e+06
75%	84.084982	117.091434	118.307840	115.889091	117.112526	1.362032e+07
max	360.278564	365.450012	366.250000	357.200012	364.049988	8.552157e+08

In [11]:
```
df = df.reset_index()
```

In [14]:
```
df.columns
```

Out[14]:
```
MultiIndex([(     'Date',            ''),
            ('Adj Close', 'POWERGRID.NS'),
            (    'Close', 'POWERGRID.NS'),
            (     'High', 'POWERGRID.NS'),
            (      'Low', 'POWERGRID.NS'),
            (     'Open', 'POWERGRID.NS'),
            (   'Volume', 'POWERGRID.NS')],
           names=['Price', 'Ticker'])
```

In [18]:
```
df.to_csv("powergrid.csv")
```

In [19]:
```
data01 = pd.read_csv("powergrid.csv")
```

In [20]:
```
data01.head()
```

Out[20]:

	Price	Date	Adj Close	Close	High	Low	Open	V
0	Ticker	NaN	POWERGRID.NS	POWERGRID.NS	POWERGRID.NS	POWERGRID.NS	POWERGRID.NS	POWERG
1	0	2007-10-05 00:00:00+00:00	32.9260368347168	56.5875129699707	61.59376525878906	46.77188491821289	50.51251220703125	855

2	1	2007-10-08 00:00:00+00:00	31.240449905395508	53.69063949584961	58.50001525878906	53.3531379699707	58.50001525878906	126
3	2	2007-10-09 00:00:00+00:00	33.367889404296875	57.34688949584961	57.8531379699707	50.8218879699707	53.7187614440918	116
4	3	2007-10-10 00:00:00+00:00	33.678829193115234	57.881263732910156	59.06251525878906	57.37501525878906	58.8375129699707	67

In [21]:
```
# Candlesticks
import plotly.graph_objects as go

fig = go.Figure(data=[go.Candlestick(x = data01['Date'], open = data01['Open'],
                                     high = data01['High'],
                                     low = data01['Low'],
                                     close = data01['Close'])])
fig.update_layout(xaxis_rangeslider_visible=False)
fig.show()
```

In [22]:
```
df = df.drop(['Date', 'Adj Close'], axis = 1)
```

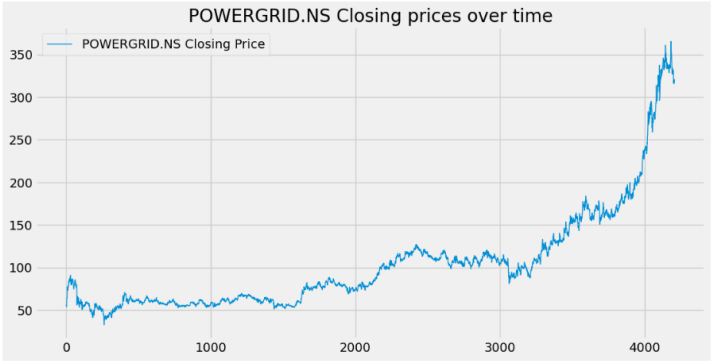
```
<ipython-input-22-8e9e58ec18af>:1: PerformanceWarning:

dropping on a non-lexsorted multi-index without a level parameter may impact performance.
```

In [23]:
```
df.head()
```

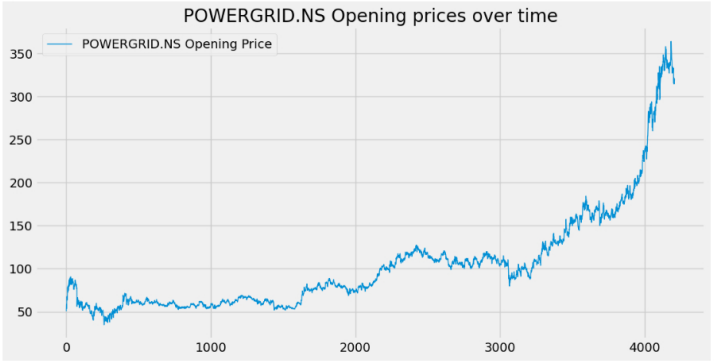
Out[23]:

Price	Close	High	Low	Open	Volume
Ticker	POWERGRID.NS	POWERGRID.NS	POWERGRID.NS	POWERGRID.NS	POWERGRID.NS
0	56.587513	61.593765	46.771885	50.512512	855215656
1	53.690639	58.500015	53.353138	58.500015	126671715
2	57.346889	57.853138	50.821888	53.718761	116725709
3	57.881264	59.062515	57.375015	58.837513	67931378
4	64.350014	67.500015	57.375015	67.500015	106320954

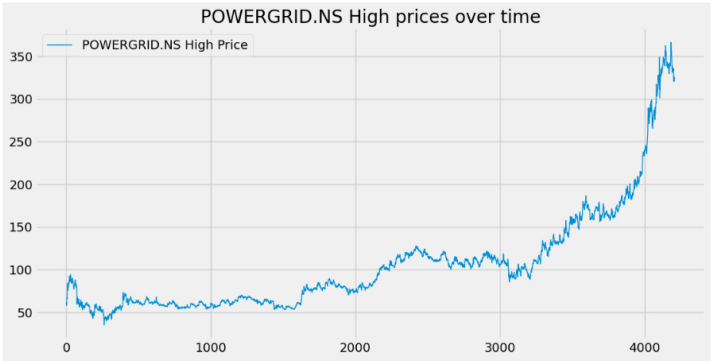
In [29]:
```
plt.figure(figsize=(12, 6))
plt.plot(df['Close'], label = f'{stock} Closing Price', linewidth = 1)
plt.title(f'{stock} Closing prices over time')
plt.legend()
plt.show()
```



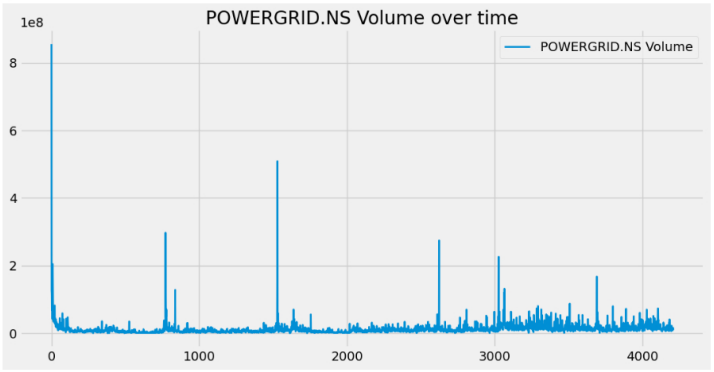
In [30]:
```
plt.figure(figsize=(12, 6))
plt.plot(df['Open'], label = f'{stock} Opening Price', linewidth = 1)
plt.title(f'{stock} Opening prices over time')
plt.legend()
plt.show()
```



In [31]:
```
plt.figure(figsize=(12, 6))
plt.plot(df['High'], label = f'{stock} High Price', linewidth = 1)
plt.title(f'{stock} High prices over time')
plt.legend()
plt.show()
```



In [33]:
```
plt.figure(figsize=(12, 6))
plt.plot(df['Volume'], label = f'{stock} Volume', linewidth = 2)
plt.title(f'{stock} Volume over time')
plt.legend()
plt.show()
```



In [39]:
```
# Moving Average
# [10, 20, 30, 40, 50, 60, 70, 80, 90]
# moving average for last 5 days -> null null null null 30.0 40.0 50.0

temp_data = [10, 20, 30, 40, 50, 60, 70, 80, 90]
print(sum(temp_data[2:7])/5)
```

```
50.0
```

In [41]:
```
import pandas as pd
df01 = pd.DataFrame(temp_data)
```

In [43]:
```
df01.rolling(5).mean()
```

Out[43]:

	0
0	NaN
1	NaN
2	NaN
3	NaN
4	30.0
5	40.0
6	50.0
7	60.0
8	70.0

In [ ]:
```
```

In [45]:
```
ma100 = df.Close.rolling(100).mean()
```

In [46]:
```
ma100
```

Out[46]:

Ticker	POWERGRID.NS
0	NaN
1	NaN
2	NaN
3	NaN
4	NaN
...	...
4203	334.890499
4204	335.113999
4205	335.331499
4206	335.511499
4207	335.625999

4208 rows × 1 columns

In [47]:
```
ma200 = df.Close.rolling(200).mean()
```

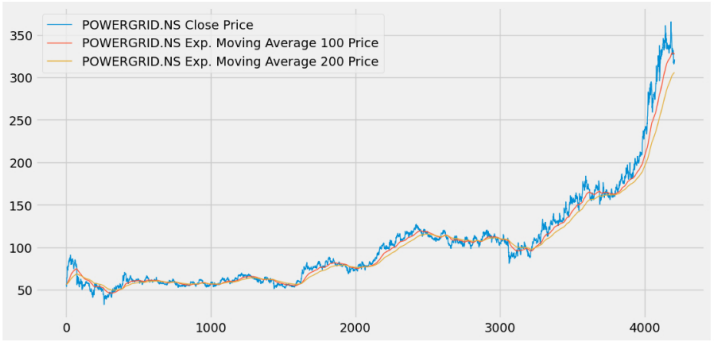
In [50]:
```
plt.figure(figsize=(12, 6))
plt.plot(df.Close, label = f'{stock} Close Price', linewidth = 1)
plt.plot(ma100, label = f'{stock} Moving Average 100 Price', linewidth = 1)
plt.plot(ma200, label = f'{stock} Moving Average 200 Price', linewidth = 1)
plt.legend()
plt.show()
```

In [51]:
```
ema100 = df.Close.ewm(span=100, adjust = False).mean()
```

In [54]:
```
ema200 = df['Close'].ewm(span=200, adjust = False).mean()
```

In [55]:

```
plt.figure(figsize=(12, 6))
plt.plot(df.Close, label = f'{stock} Close Price', linewidth = 1)
plt.plot(ema100, label = f'{stock} Exp. Moving Average 100 Price', linewidth = 1)
plt.plot(ema200, label = f'{stock} Exp. Moving Average 200 Price', linewidth = 1)
plt.legend()
plt.show()
```



In [56]:
```
# Training & Testing

data_training = pd.DataFrame(df['Close'][0:int(len(df)*0.70)])
data_testing = pd.DataFrame(df['Close'][int(len(df)*0.70): int(len(df))])
```

In [57]:
```
data_training.shape
```

Out[57]: (2945, 1)

In [58]:
```
data_testing.shape
```

Out[58]: (1263, 1)

In [59]:
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range = (0, 1))
```

In [60]:
```
data_training_array = scaler.fit_transform(data_training)
```

In [62]:
```
data_training_array
```

Out[62]:
```
array([[0.25364691],
       [0.22298304],
       [0.26168503],
       ...,
       [0.82286392],
       [0.77582614],
       [0.78862755]])
```

In [64]:
```
data_training_array.shape[0]
```

Out[64]: 2945

In [65]:
```
x_train = []
y_train = []

for i in range(100, data_training_array.shape[0]):
    x_train.append(data_training_array[i-100:i])
    y_train.append(data_training_array[i, 0])

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

In [66]:
```
x_train.shape
```

Out[66]: (2845, 100, 1)

In [68]:
```
# Model Building
from keras.layers import Dense, Dropout, LSTM
from keras.models import Sequential
```

LSTM Input -> 3D Array (batch_size, time_steps, seq_len) LSTM 2D OR 3D 2D -> (batch_sizem units) 3D -> (batch_size, time_steps, units)

In [69]:
```
model = Sequential()

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

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

model.add(LSTM(units = 80, 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))
```

In [70]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 100, 50)           10400

 dropout (Dropout)           (None, 100, 50)           0

 lstm_1 (LSTM)               (None, 100, 60)           26640

 dropout_1 (Dropout)         (None, 100, 60)           0

 lstm_2 (LSTM)               (None, 100, 80)           45120

 dropout_2 (Dropout)         (None, 100, 80)           0

 lstm_3 (LSTM)               (None, 120)               96480

 dropout_3 (Dropout)         (None, 120)               0

 dense (Dense)               (None, 1)                 121

=================================================================
Total params: 178,761
Trainable params: 178,761
Non-trainable params: 0
_________________________________________________________________
```

In [71]:
```
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
model.fit(x_train, y_train, epochs = 50)
```

```
Epoch 1/50
89/89 [==============================] - 39s 386ms/step - loss: 0.0336
Epoch 2/50
```

```
89/89 [==============================] - 31s 350ms/step - loss: 0.0090
Epoch 3/50
89/89 [==============================] - 23s 260ms/step - loss: 0.0080
Epoch 4/50
89/89 [==============================] - 24s 269ms/step - loss: 0.0070
Epoch 5/50
89/89 [==============================] - 24s 268ms/step - loss: 0.0064
Epoch 6/50
89/89 [==============================] - 24s 267ms/step - loss: 0.0061
Epoch 7/50
89/89 [==============================] - 24s 271ms/step - loss: 0.0058
Epoch 8/50
89/89 [==============================] - 24s 271ms/step - loss: 0.0050
Epoch 9/50
89/89 [==============================] - 24s 272ms/step - loss: 0.0050
Epoch 10/50
89/89 [==============================] - 24s 273ms/step - loss: 0.0050
Epoch 11/50
89/89 [==============================] - 24s 270ms/step - loss: 0.0045
Epoch 12/50
89/89 [==============================] - 24s 269ms/step - loss: 0.0047
Epoch 13/50
89/89 [==============================] - 25s 276ms/step - loss: 0.0041
Epoch 14/50
89/89 [==============================] - 25s 276ms/step - loss: 0.0038
Epoch 15/50
89/89 [==============================] - 25s 278ms/step - loss: 0.0040
Epoch 16/50
89/89 [==============================] - 24s 275ms/step - loss: 0.0039
Epoch 17/50
89/89 [==============================] - 24s 275ms/step - loss: 0.0031
Epoch 18/50
89/89 [==============================] - 24s 272ms/step - loss: 0.0031
Epoch 19/50
89/89 [==============================] - 25s 276ms/step - loss: 0.0033
Epoch 20/50
89/89 [==============================] - 24s 273ms/step - loss: 0.0033
Epoch 21/50
89/89 [==============================] - 28s 311ms/step - loss: 0.0031
Epoch 22/50
89/89 [==============================] - 29s 321ms/step - loss: 0.0030
Epoch 23/50
89/89 [==============================] - 27s 308ms/step - loss: 0.0029
Epoch 24/50
89/89 [==============================] - 26s 289ms/step - loss: 0.0029
Epoch 25/50
89/89 [==============================] - 25s 279ms/step - loss: 0.0027
Epoch 26/50
89/89 [==============================] - 25s 286ms/step - loss: 0.0027
Epoch 27/50
89/89 [==============================] - 24s 273ms/step - loss: 0.0025
Epoch 28/50
89/89 [==============================] - 24s 274ms/step - loss: 0.0025
Epoch 29/50
89/89 [==============================] - 24s 274ms/step - loss: 0.0026
Epoch 30/50
89/89 [==============================] - 24s 273ms/step - loss: 0.0024
Epoch 31/50
89/89 [==============================] - 24s 274ms/step - loss: 0.0024
Epoch 32/50
89/89 [==============================] - 24s 275ms/step - loss: 0.0025
Epoch 33/50
89/89 [==============================] - 25s 277ms/step - loss: 0.0023
Epoch 34/50
89/89 [==============================] - 24s 275ms/step - loss: 0.0024
Epoch 35/50
89/89 [==============================] - 25s 276ms/step - loss: 0.0022
Epoch 36/50
89/89 [==============================] - 25s 282ms/step - loss: 0.0022
Epoch 37/50
89/89 [==============================] - 25s 276ms/step - loss: 0.0022
Epoch 38/50
89/89 [==============================] - 24s 273ms/step - loss: 0.0022
Epoch 39/50
89/89 [==============================] - 24s 273ms/step - loss: 0.0021
Epoch 40/50
89/89 [==============================] - 24s 272ms/step - loss: 0.0023
Epoch 41/50
89/89 [==============================] - 24s 272ms/step - loss: 0.0024
Epoch 42/50
89/89 [==============================] - 24s 273ms/step - loss: 0.0023
Epoch 43/50
89/89 [==============================] - 25s 277ms/step - loss: 0.0023
Epoch 44/50
89/89 [==============================] - 25s 275ms/step - loss: 0.0022
Epoch 45/50
89/89 [==============================] - 27s 299ms/step - loss: 0.0022
Epoch 46/50
89/89 [==============================] - 25s 284ms/step - loss: 0.0021
Epoch 47/50
89/89 [==============================] - 25s 276ms/step - loss: 0.0022
Epoch 48/50
89/89 [==============================] - 24s 275ms/step - loss: 0.0022
Epoch 49/50
89/89 [==============================] - 24s 273ms/step - loss: 0.0022
Epoch 50/50
89/89 [==============================] - 24s 272ms/step - loss: 0.0021
```

Out[71]:
```
<keras.callbacks.History at 0x1f289ec0d90>
```

In [72]:
```
past_100_days = data_training.tail(100)
```

In [73]:
```
final_df = past_100_days.append(data_testing, ignore_index = True)
```

```
<ipython-input-73-1e7e7f089a7f>:1: FutureWarning:

The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
```

In [75]:
```
final_df.head()
```

Out[75]:

Ticker	POWERGRID.NS
0	104.850029
1	106.987526
2	107.353149
3	107.128151
4	107.746902

In [74]:
```
input_data = scaler.fit_transform(final_df)
```

In [76]:
```
x_test = []
y_test = []

for i in range(100, input_data.shape[0]):
    x_test.append(input_data[i-100:i])
    y_test.append(input_data[i, 0])

x_test, y_test  = np.array(x_test), np.array(y_test)
```

In [77]:
```
x_test.shape
```

Out[77]:
```
(1263, 100, 1)
```

In [78]:
```
y_predicted = model.predict(x_test)
```

```
40/40 [==============================] - 5s 112ms/step
```

In [79]:
```
y_predicted.shape
```

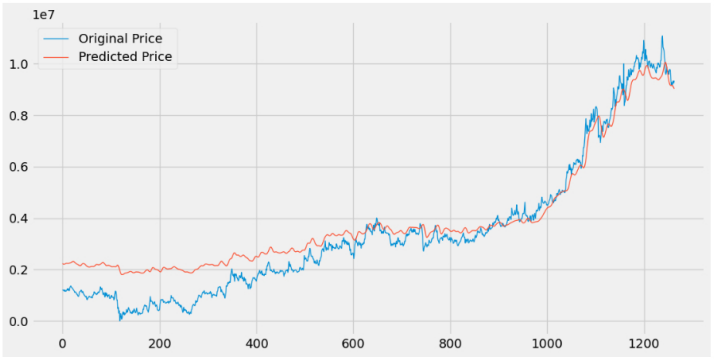
Out[79]:
```
(1263, 1)
```

In [80]:
```
scaler.scale_
```

Out[80]:
```
array([0.0035166])
```

In [86]:
```
scaler_factor = 1 / 0.0035166
y_predicted = y_predicted * scaler_factor
y_test = y_test * scaler_factor
```

In [89]:
```
plt.figure(figsize=(12, 6))
plt.plot(y_test, label = 'Original Price', linewidth = 1)
plt.plot(y_predicted, label = 'Predicted Price', linewidth = 1)
plt.legend()
plt.show()
```



In [88]:
```
model.save('stock_dl_model.h5')
```

In [ ]:
```

```