Importing Libraries :

**The libraries used in this notebook are the as follows:

1. **Pandas**: Used for data manipulation and analysis, enabling tasks like loading CSV data with `pd.read_csv()` and summarizing data with `df.head()` and `df.describe()`.
2. **NumPy**: Provides support for numerical operations and array manipulation, with functions such as `np.array()` for creating arrays and `np.mean()` for calculating the mean.
3. **Matplotlib & Seaborn**: These libraries are used for creating static visualizations, with `plt.plot()` and `sns.heatmap()` to generate plots and heatmaps for data analysis.
4. **Statsmodels**: Offers tools for time-series analysis and forecasting, including models like `ARIMA()` and `SARIMAX()` for stock price predictions.
5. **TensorFlow**: A deep learning framework used to build and train the LSTM (Long Short-Term Memory) model for stock price prediction, utilizing layers like `Sequential()` and `LSTM()`.
6. **Scikit-learn**: Provides a suite of model evaluation metrics, including `mean_absolute_error()`, `mean_squared_error()`, and `mean_absolute_percentage_error()` to assess the performance of the models.
7. **Cufflinks**: Enables the creation of interactive plots directly from Pandas DataFrames, enhancing data visualization with functions like `df.iplot()`.
8. **Plotly**: Used for creating interactive visualizations, allowing for dynamic and engaging charts through functions like `plotly.express` and `iplot()`.
9. **Warnings**: Helps suppress unnecessary warnings for cleaner outputs during code execution by using `warnings.filterwarnings('ignore')`.

```
from dateutil.parser import parse
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
plt.rcParams.update({'figure.figsize': (10, 7), 'figure.dpi': 120})
import pandas as pd
import pandas_datareader as web
import datetime

import matplotlib.pyplot as plt
import seaborn as sns

from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.arima_model import ARIMA
#relax the display limits on columns and rows
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)\a

import warnings
warnings.filterwarnings('ignore')

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dropout


from sklearn.metrics import mean_absolute_percentage_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error


%matplotlib inline
import cufflinks as cf
cf.go_offline()
```

```
from plotly import __version__
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot

import cufflinks as cf
```

```
init_notebook_mode(connected=True)
# For offline use
cf.go_offline()
```

```
df = pd.read_csv("NIFTY 50 - 3 minute_with_indicators_.csv")
```

EDA

```
df.head(5)
```

	date	close	high	low	open	volume	sma5	sma10	sma15	sma20	ema5	ema10	e
0	2015-01-09 12:24:00+05:30	8217.40	8226.55	8217.15	8226.05	0	8221.42	8225.845	8236.173333	8241.4200	8221.286878	8227.126723	8233.4
1	2015-01-09 12:27:00+05:30	8214.70	8217.40	8210.35	8217.35	0	8219.99	8221.405	8233.443333	8238.9175	8219.091252	8224.867319	8231.1
2	2015-01-09 12:30:00+05:30	8216.95	8219.05	8210.40	8214.85	0	8218.52	8217.785	8230.653333	8237.1500	8218.377501	8223.427807	8229.3
3	2015-01-09 12:33:00+05:30	8209.20	8219.50	8198.40	8217.20	0	8216.87	8217.330	8227.546667	8234.6175	8215.318334	8220.840933	8226.8
4	2015-01-09 12:36:00+05:30	8202.90	8212.05	8201.00	8209.65	0	8212.23	8215.480	8224.003333	8232.1275	8211.178890	8217.578945	8223.8

```
df.tail()
```

	date	close	high	low	open	volume	sma5	sma10	sma15	sma20	ema5	
230200	2022-10-24 19:00:00+05:30	17723.65	17733.85	17721.3	17733.10	0	17731.69	17733.345	17739.116667	17709.6575	17730.340412	17727.
230201	2022-10-24 19:03:00+05:30	17711.40	17728.95	17708.4	17723.00	0	17726.96	17730.685	17735.370000	17716.2850	17724.026941	17724.
230202	2022-10-24 19:06:00+05:30	17731.00	17732.10	17709.3	17709.30	0	17726.22	17731.120	17733.373333	17723.4125	17726.351294	17725.
230203	2022-10-24 19:09:00+05:30	17735.15	17736.10	17728.1	17732.70	0	17726.97	17730.660	17732.763333	17730.5700	17729.284196	17727.
230204	2022-10-24 19:12:00+05:30	17738.95	17740.80	17732.2	17734.55	0	17728.03	17731.065	17732.646667	17738.3300	17732.506131	17729.

```
df = df[['date', 'close', 'high','low','open']]
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 230205 entries, 0 to 230204
Data columns (total 5 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   date    230205 non-null  object 
 1   close   230205 non-null  float64
 2   high    230205 non-null  float64
 3   low     230205 non-null  float64
 4   open    230205 non-null  float64
dtypes: float64(4), object(1)
memory usage: 8.8+ MB
```

```
#I am renaming my headers for more clarity
headers = ["Date", "Close", "High", "Low", "Open"]
df.columns = headers
df.head()
```

	Date	Close	High	Low	Open
0	2015-01-09 12:24:00+05:30	8217.40	8226.55	8217.15	8226.05
1	2015-01-09 12:27:00+05:30	8214.70	8217.40	8210.35	8217.35
2	2015-01-09 12:30:00+05:30	8216.95	8219.05	8210.40	8214.85
3	2015-01-09 12:33:00+05:30	8209.20	8219.50	8198.40	8217.20
4	2015-01-09 12:36:00+05:30	8202.90	8212.05	8201.00	8209.65

```
df.tail()
```

	Date	Close	High	Low	Open
230200	2022-10-24 19:00:00+05:30	17723.65	17733.85	17721.3	17733.10
230201	2022-10-24 19:03:00+05:30	17711.40	17728.95	17708.4	17723.00
230202	2022-10-24 19:06:00+05:30	17731.00	17732.10	17709.3	17709.30
230203	2022-10-24 19:09:00+05:30	17735.15	17736.10	17728.1	17732.70
230204	2022-10-24 19:12:00+05:30	17738.95	17740.80	17732.2	17734.55

```
#I am using df.info in order to have better insight of the informations and variables of my dataset df
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 230205 entries, 0 to 230204
Data columns (total 5 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Date    230205 non-null  object 
 1   Close   230205 non-null  float64
 2   High    230205 non-null  float64
 3   Low     230205 non-null  float64
 4   Open    230205 non-null  float64
dtypes: float64(4), object(1)
memory usage: 8.8+ MB
```

```
df.dtypes
```

```
Date      object
Close    float64
High     float64
Low      float64
Open     float64
dtype: object
```

```
df['Date'] = pd.to_datetime(df['Date'])
df['Date'] = pd.to_datetime(df['Date'], format='%d/%m/%Y')
```

Step 1: Convert "Date" to `datetime` format

```
df['Date'] = pd.to_datetime(df['Date'])
```

- **Objective**: The goal of this step is to convert the "Date" column, which is currently a string (object) type, into a `datetime` type.

Step 2: Explicitly define the date format for parsing

```
df['Date'] = pd.to_datetime(df['Date'], format='%d/%m/%Y')
```

- **Objective**: This step explicitly defines the format in which the "Date" column is structured to ensure correct parsing.
- **Result**: After both steps, the "Date" column will be in `datetime64` format, making it easier to work with in time-series analysis or any date-based manipulations (e.g., filtering, resampling).

```
df.describe()
```

	Close	High	Low	Open
count	230205.000000	230205.000000	230205.000000	230205.000000
mean	11422.741621	11427.481308	11417.989490	11422.819581
std	3134.843360	3136.149966	3133.497664	3134.845819
min	6852.450000	6883.500000	6826.350000	6853.350000
25%	8741.200000	8744.800000	8737.450000	8741.200000
50%	10724.500000	10728.300000	10720.500000	10724.500000
75%	12640.850000	12648.900000	12632.150000	12641.150000
max	18592.150000	18602.350000	18582.250000	18602.350000

The data appears to show significant fluctuations in stock prices, as indicated by the wide range between the minimum and maximum values. The standard deviation is also relatively high, suggesting that the stock prices are volatile. The stock prices show general upward trends (since the median and 75% quartile are much higher than the minimum and 25% quartile), although there are some periods of sharp decline (as seen in the min value).

```
df.shape
```

```
(230205, 5)
```

```
# i am using this function in order to know my null values and see if they are more than 10% of my dataset
df.isnull().sum()
```

```
Date     0
Close    0
High     0
Low      0
Open     0
dtype: int64
```

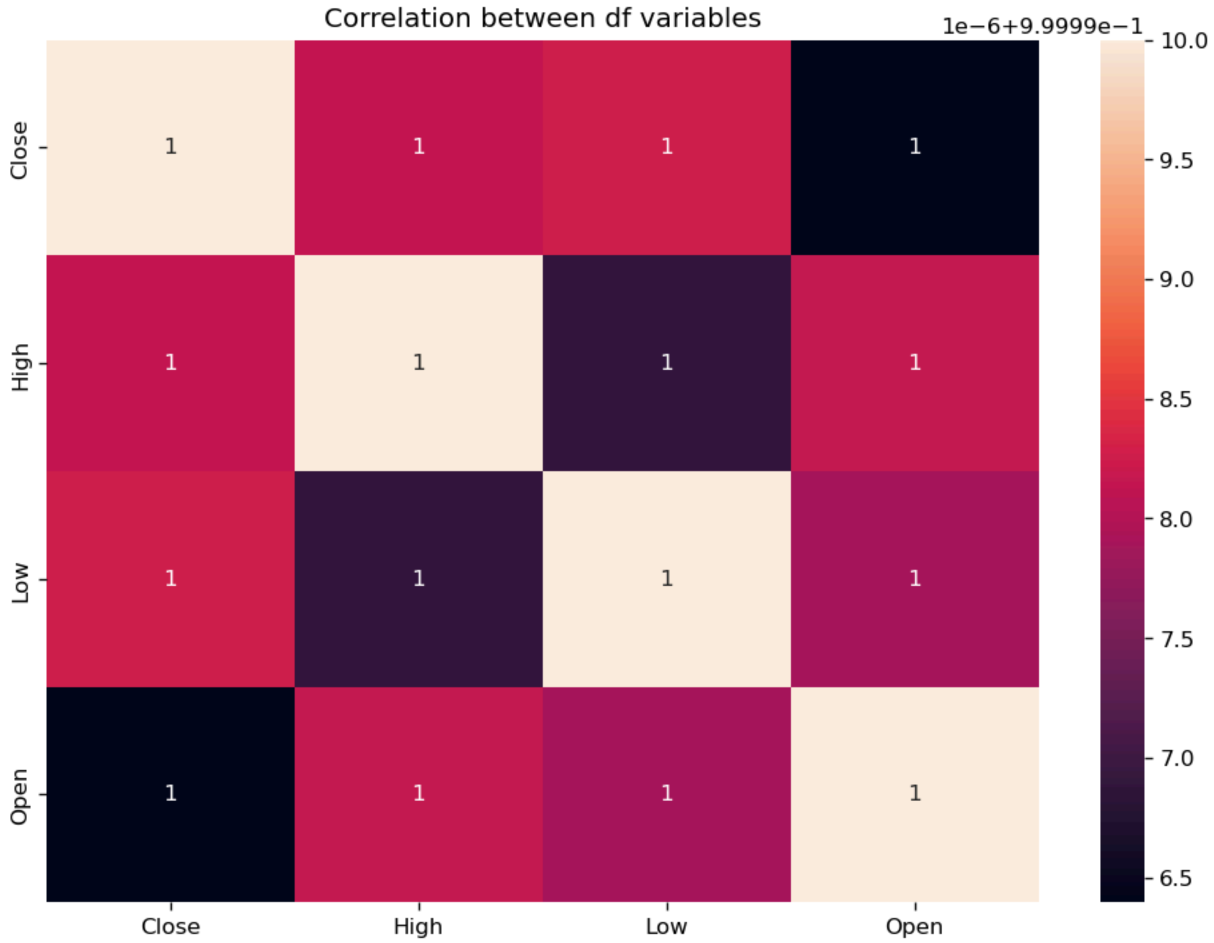
```
df.corr()
```

	Close	High	Low	Open
Close	1.000000	0.999998	0.999998	0.999996
High	0.999998	1.000000	0.999997	0.999998
Low	0.999998	0.999997	1.000000	0.999998
Open	0.999996	0.999998	0.999998	1.000000

The features (Close, High, Low, Open) in this dataset are extremely highly correlated with each other, with values very close to 1. This means the values of these features move in a very similar manner, and they likely convey the same underlying trends. This high correlation suggests that predicting one of these features could give insights into the others.

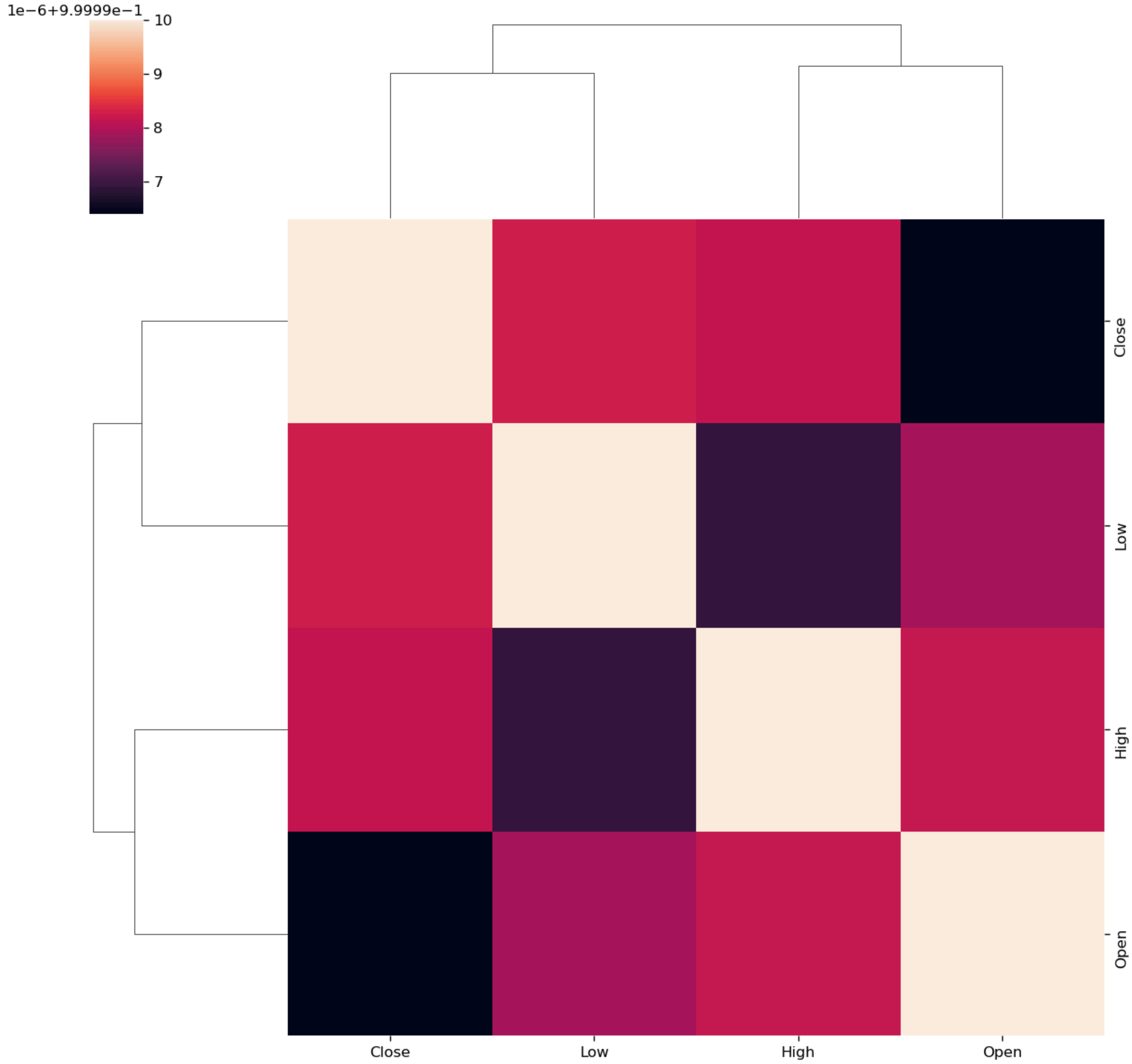
```
sns.heatmap(df.corr(), annot=True)
plt.title('Correlation between df variables')
```

```
Text(0.5, 1.0, 'Correlation between df variables')
```



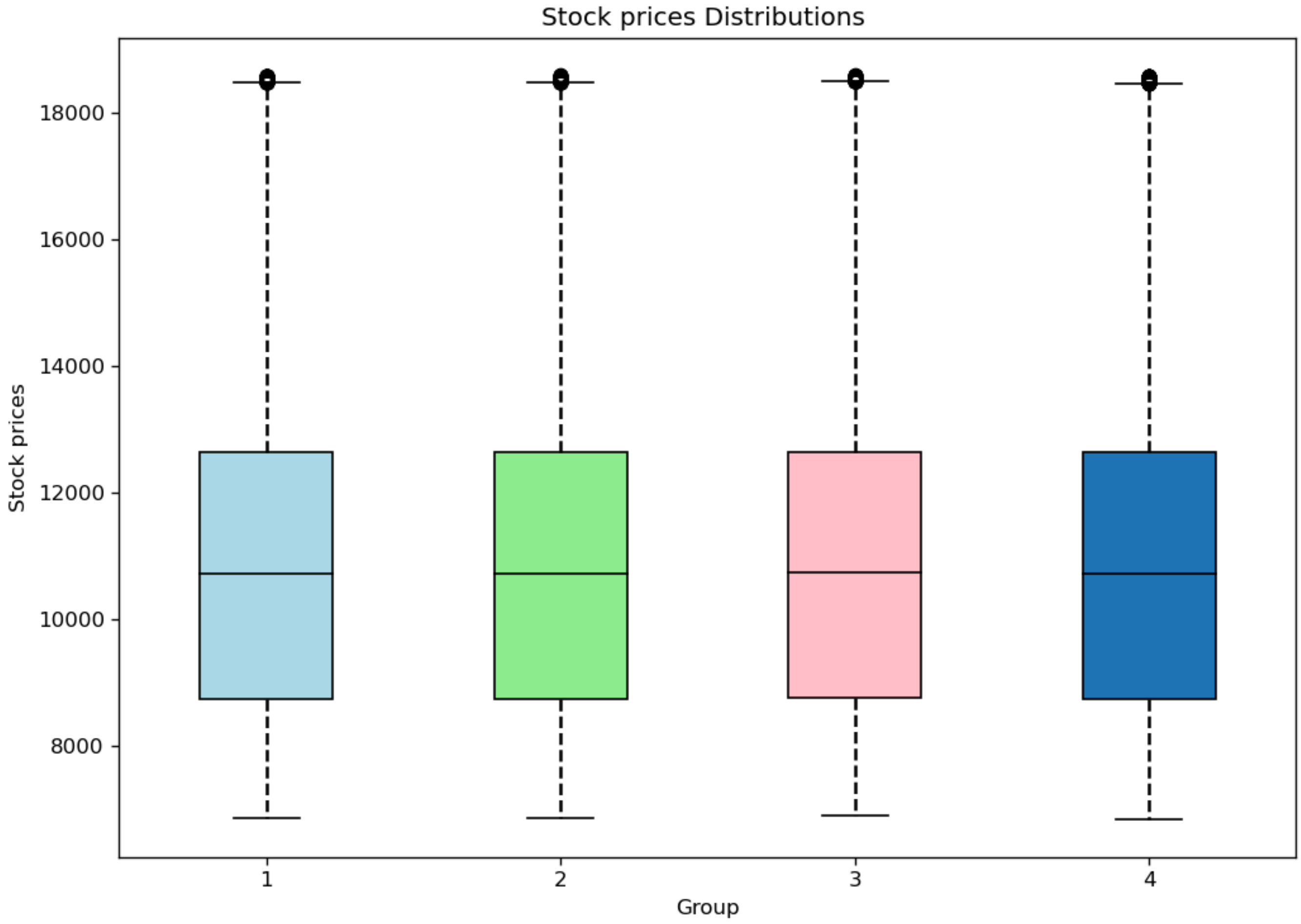

```
sns.clustermap(df.corr())
```

<seaborn.matrix.ClusterGrid at 0x1aa809b43d0>




```
fig, ax = plt.subplots()
bp = ax.boxplot([df.Close, df.Open, df.High, df.Low], patch_artist=True,
                medianprops=dict(color='black'))
colors = ['lightblue', 'lightgreen', 'pink']
for box, color in zip(bp['boxes'], colors):
    box.set_facecolor(color)
for whisker in bp['whiskers']:
    whisker.set(color='black', linestyle='--', linewidth=1.5)
ax.set_title('Stock prices Distributions ')
ax.set_xlabel('Group')
ax.set_ylabel('Stock prices')
```

```
Text(0, 0.5, 'Stock prices')
```




```
df.dtypes
```

```
Date     datetime64[ns, pytz.FixedOffset(330)]
Close                                  float64
High                                   float64
Low                                    float64
Open                                   float64
dtype: object
```

We create a second dataset for visulalisation

```
df2 = df
```

```
df2['Date']= pd.to_datetime(df2['Date'])
df2['Date'] = pd.to_datetime(df2['Date'], format='%d-%m-%y')
```

```
df2 = df2.set_index('Date')
```

```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 230205 entries, 2015-01-09 12:24:00+05:30 to 2022-10-24 19:12:00+05:30
Data columns (total 4 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Close   230205 non-null  float64
 1   High    230205 non-null  float64
 2   Low     230205 non-null  float64
 3   Open    230205 non-null  float64
dtypes: float64(4)
memory usage: 8.8 MB
```

```
df2.head()
```

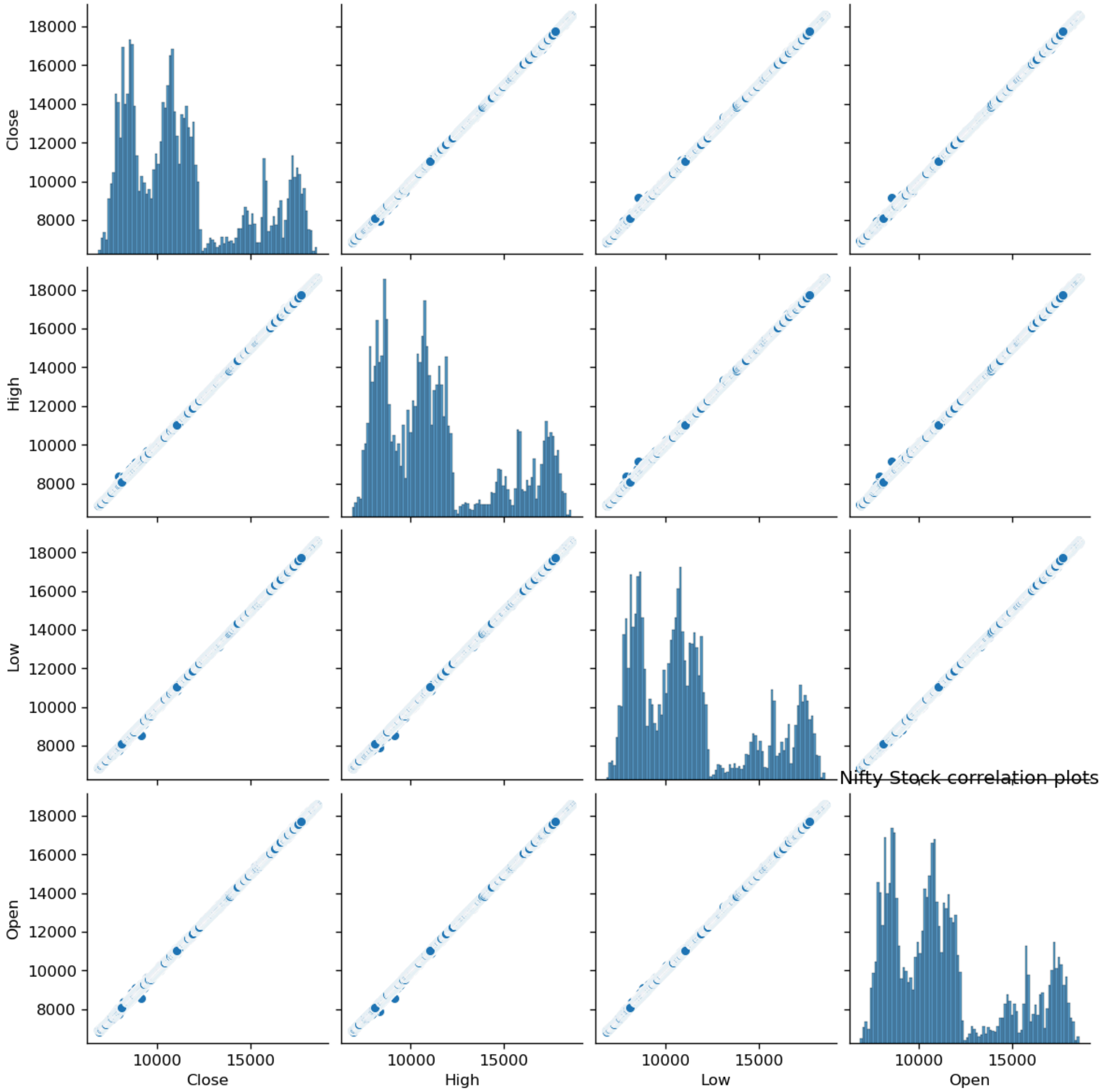
	Close	High	Low	Open
Date				
2015-01-09 12:24:00+05:30	8217.40	8226.55	8217.15	8226.05
2015-01-09 12:27:00+05:30	8214.70	8217.40	8210.35	8217.35
2015-01-09 12:30:00+05:30	8216.95	8219.05	8210.40	8214.85
2015-01-09 12:33:00+05:30	8209.20	8219.50	8198.40	8217.20
2015-01-09 12:36:00+05:30	8202.90	8212.05	8201.00	8209.65

Visualisation

```
df2.iplot(kind='box',title = 'Stock prices Distributions' )
%matplotlib inline
```

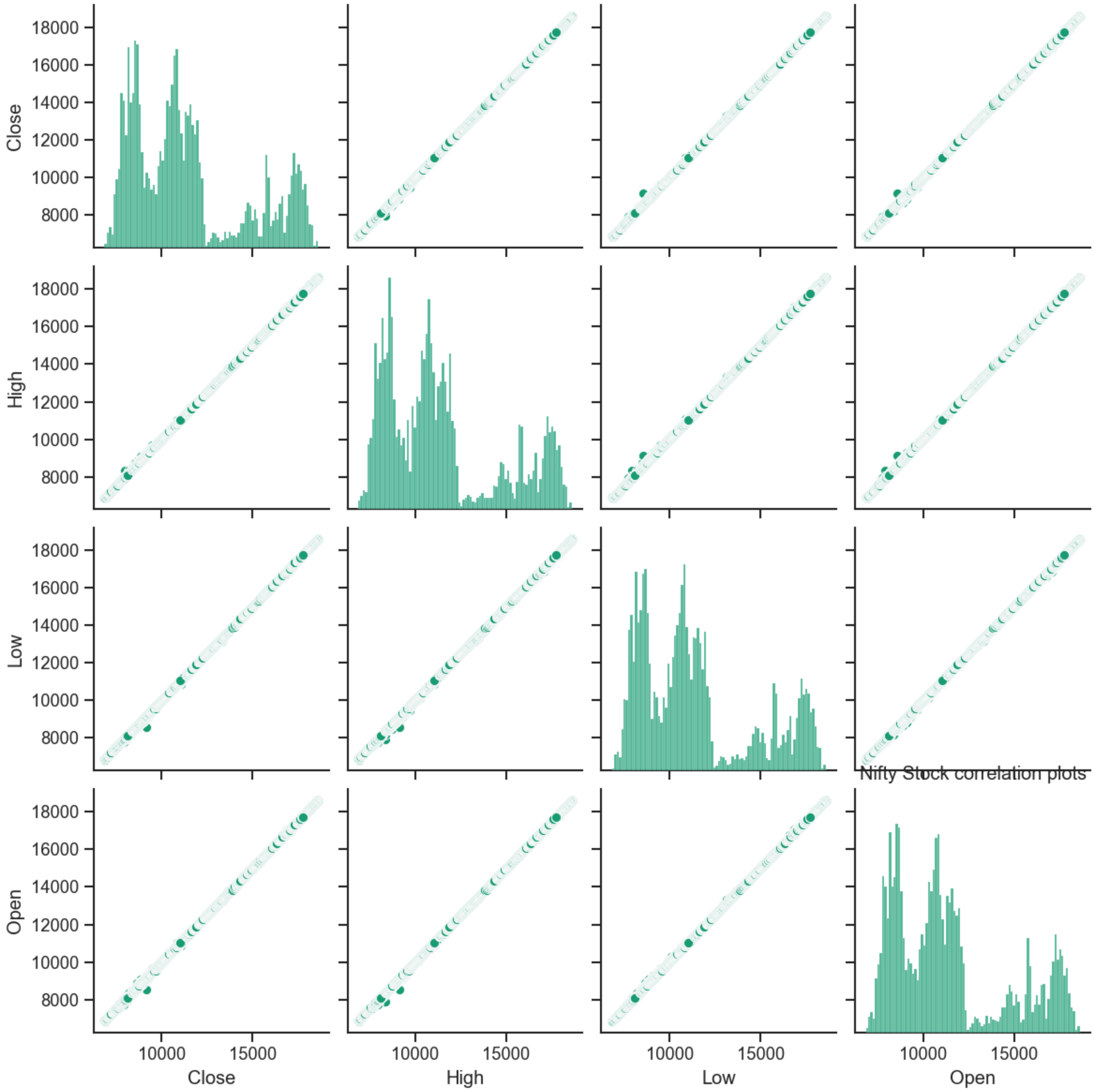
```
sns.pairplot(df2,palette='rainbow')
plt.title('Nifty Stock correlation plots')
```

```
Text(0.5, 1.0, 'Nifty Stock correlation plots')
```



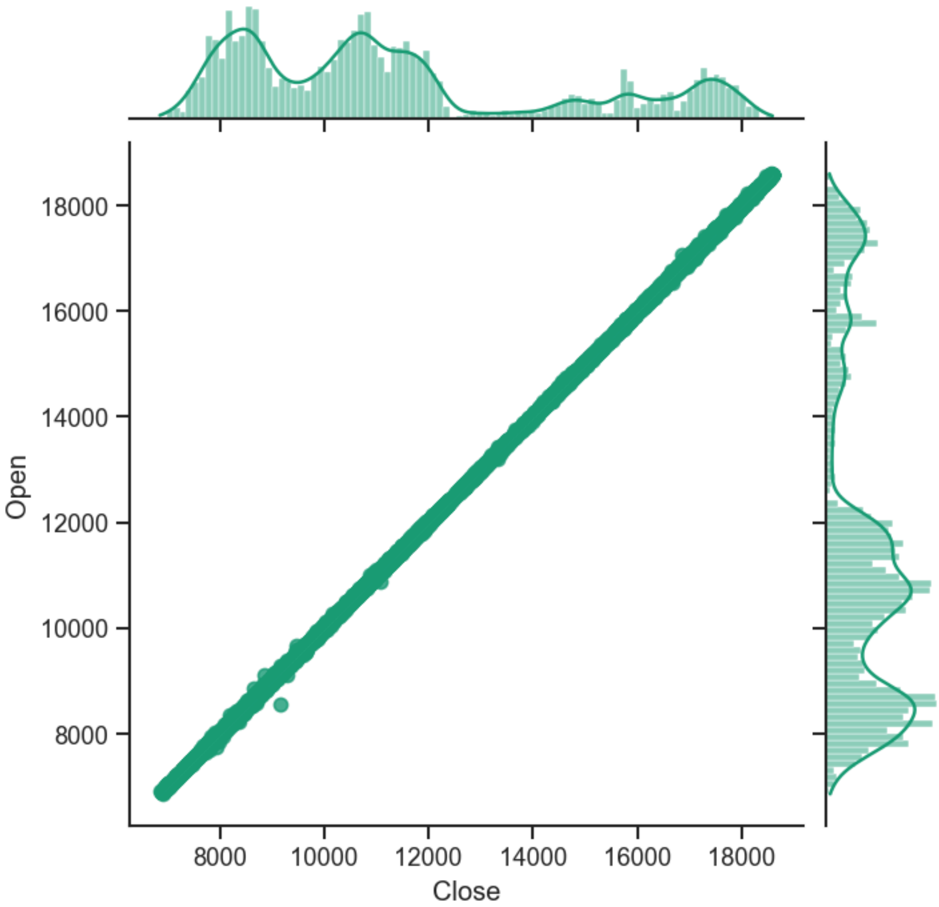

```
sns.set(style='ticks')
sns.set_palette('Dark2')
sns.pairplot(df2)
plt.title('Nifty Stock correlation plots')
```

```
Text(0.5, 1.0, 'Nifty Stock correlation plots')
```



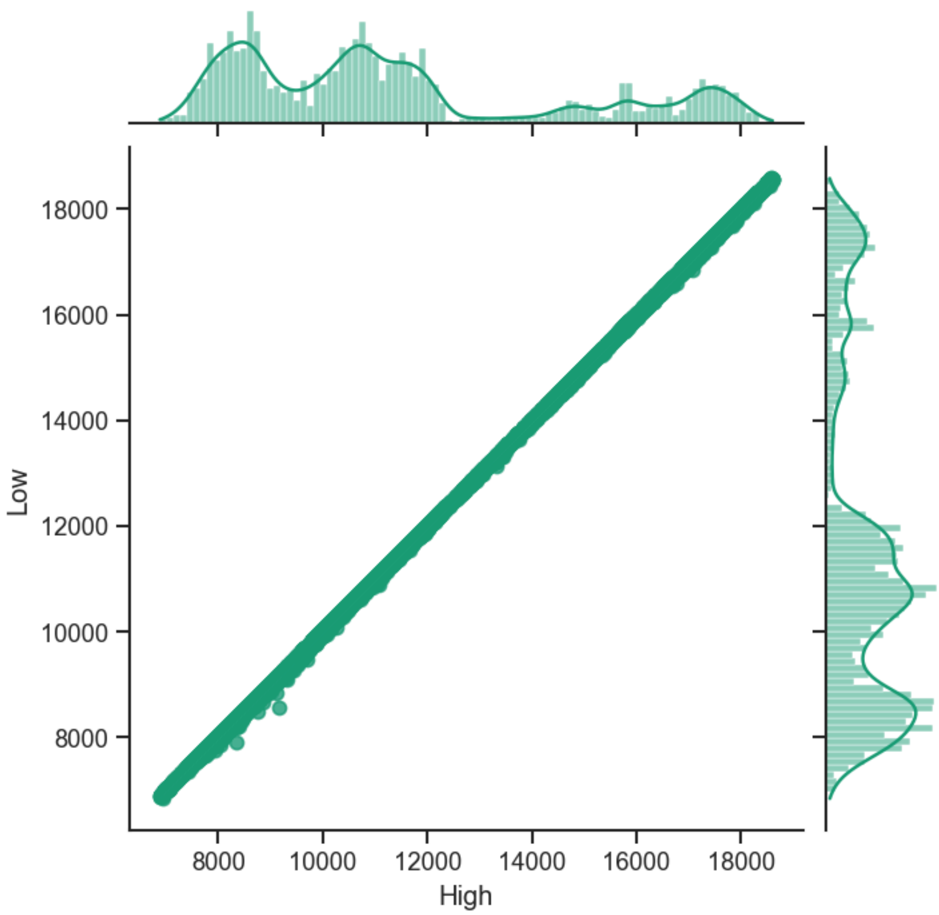

```
sns.jointplot(x='Close',y='Open',data=df2,kind='reg')
```

<seaborn.axisgrid.JointGrid at 0x1aa8095be90>

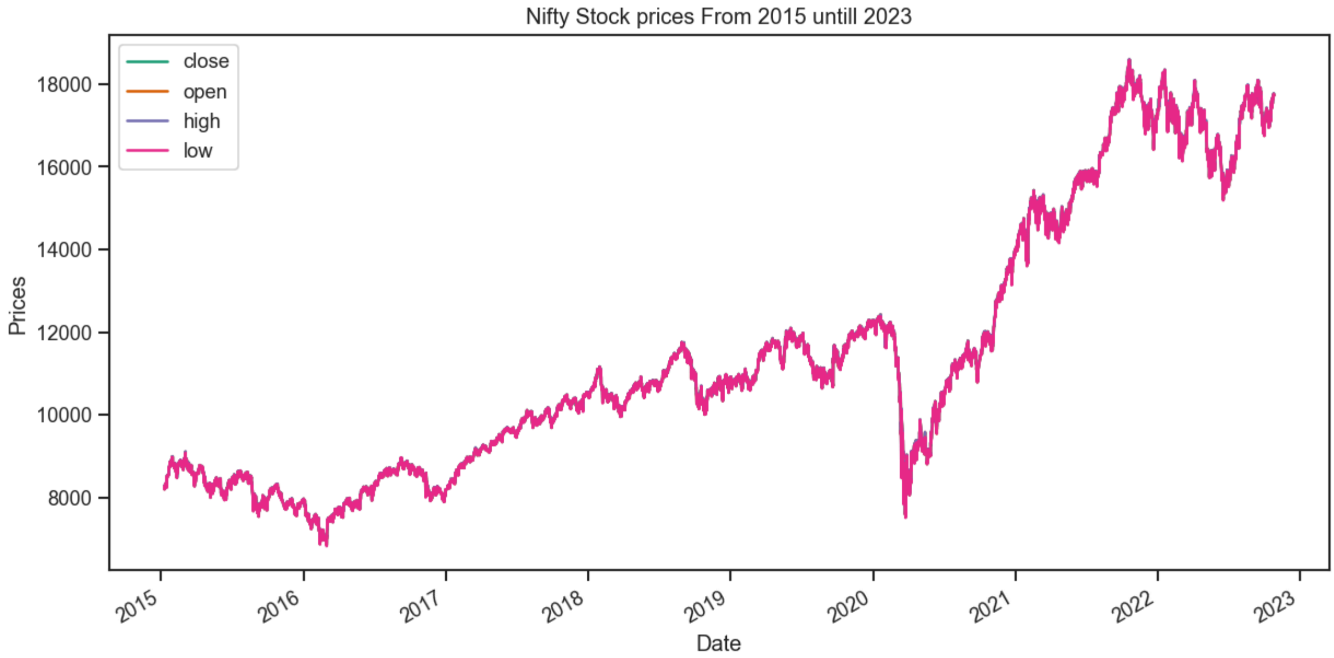



```
sns.jointplot(x='High',y='Low',data=df2,kind='reg')
```

<seaborn.axisgrid.JointGrid at 0x1aa854b7710>



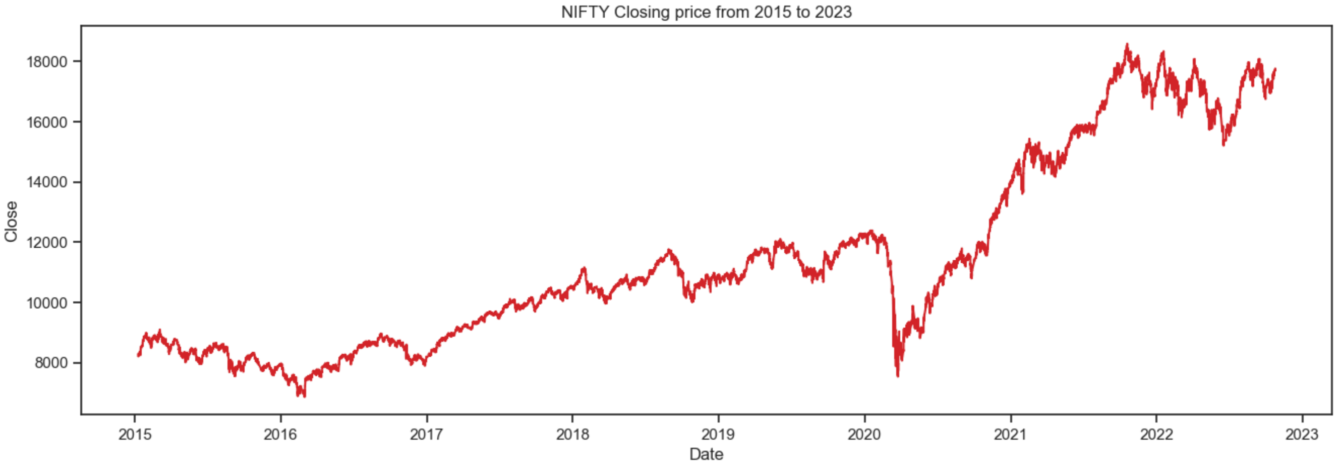
```
#Multi Line plot

plt.subplots(figsize=(12,6))
plt.xlabel('Date')
plt.ylabel('Prices')
df2['Close'].plot(label='close')
df2['Open'].plot(label='open')
df2['High'].plot(label='high')
df2['Low'].plot(label='low')
plt.legend()
plt.title('Nifty Stock prices From 2015 untill 2023')
plt.show()
```



From the above graphs, we can make the gfollowing observation :

```
There is a drastic drop in stock prices in 2015-2016 period.This can be attributed to a Recession that happened during this period.

There is a drop in stock prices in the year 2016. This can be attributed to Demonitisation drive by the central government.

There is a drastic drop in stock prices in 2020. This is due to the global breakdown amid coronavirus pandemic induced lockdown in India.

By the end of 2020 untill 2023, the stock price started rising.This can be attributed to the lifting of lockdown in the country and across the wor
```

```
# Draw Plot
def plot_df(df2, x, y, title="", xlabel='Date', ylabel='Close', dpi=100):
    plt.figure(figsize=(16,5), dpi=dpi)
    plt.plot(x, y, color='tab:red')
    plt.gca().set(title=title, xlabel=xlabel, ylabel=ylabel)
    plt.show()

plot_df(df2, x=df2.index, y=df2.Close, title='NIFTY Closing price from 2015 to 2023')
```




```
# Import necessary libraries
import plotly.graph_objects as go
```

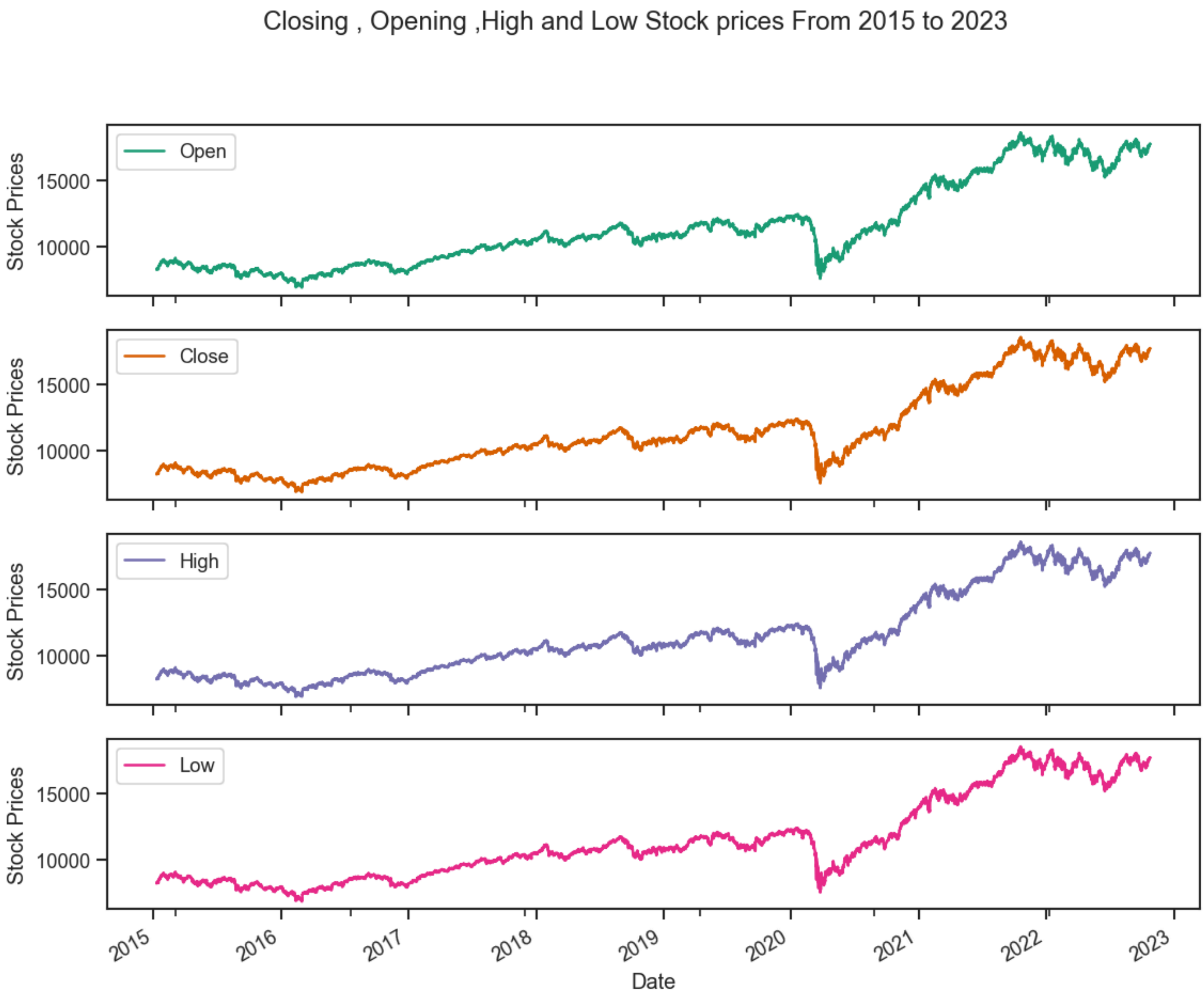
```
fig = go.Figure([go.Scatter(x=df2.index, y=df2['Close'])])
fig.update_layout(
    autosize=False,
    width=1000,
    height=500,
    title='Closing Price Price from 2015 to 2023',
    template="simple_white",
)
fig.update_xaxes(title="Close")
fig.update_yaxes(title="Date")
fig.show()
%matplotlib inline
```

```
df2[['Close','Open']].iplot(kind='spread', title='Closing and Opening Price from 2015 to 2023')
%matplotlib inline
```

```
df2[['High','Low']].iplot(kind='spread' ,title='High and Low Prices from 2015 to 2023')
%matplotlib inline
```

```
fig = go.Figure([go.Scatter(x=df2.index, y=df2['Close'])])
fig.update_layout(
    autosize=False,
    width=1000,
    height=500,
    template='simple_white',
    title='Closing Price from 2015 to 2023'
)
fig.update_xaxes(title="Date")
fig.update_yaxes(title="Close")
fig.show()
%matplotlib inline
```

```
cols_plot = ['Open', 'Close', 'High','Low']
axes = df2[cols_plot].plot(figsize=(11, 9), subplots=True, title='Closing , Opening ,High and Low Stock prices From 2015 to 2023')
for ax in axes:
    ax.set_ylabel('Stock Prices')
```



**now I am going to save the dataset as a csv file to use in a testing data set to test the model's prediction

```
df.to_csv('NIFTY50_VM.csv', index=False)
```

```
df3 = df
```

```
df3['Date'] = pd.to_datetime(df3['Date'])
```

```
df3['Date'] = pd.to_datetime(df3['Date'], format='%d/%m/%Y')
```

```
df_test = df3[df3.Date >= "2022"]
```

```
df_test.head()
```

	Date	Close	High	Low	Open
206185	2022-01-03 09:15:00+05:30	17457.70	17459.55	17387.15	17387.15
206186	2022-01-03 09:18:00+05:30	17472.25	17478.10	17455.30	17455.95
206187	2022-01-03 09:21:00+05:30	17455.50	17472.55	17452.20	17471.85
206188	2022-01-03 09:24:00+05:30	17441.50	17455.35	17436.35	17454.55
206189	2022-01-03 09:27:00+05:30	17462.35	17463.80	17441.20	17441.25

```
df_test = df_test.query("Date.dt.month >= 9")
```

```
df_test.head()
```

	Date	Close	High	Low	Open
225935	2022-09-01 09:15:00+05:30	17538.80	17567.35	17485.70	17485.70
225936	2022-09-01 09:18:00+05:30	17551.10	17557.65	17524.45	17538.30
225937	2022-09-01 09:21:00+05:30	17563.40	17573.95	17548.55	17548.55
225938	2022-09-01 09:24:00+05:30	17555.30	17564.00	17533.75	17564.00
225939	2022-09-01 09:27:00+05:30	17556.45	17558.35	17538.35	17555.15

```
df_test.to_csv('NIFTY50_test_data.csv', index=False)
```

After running the previous codes, df_test contains the subset of data from September 2022 and later, which I will use as my testing dataset.

Building and Training an RNN Model

RNN

```
df.head(1)
```

	Date	Close	High	Low	Open
0	2015-01-09 12:24:00+05:30	8217.4	8226.55	8217.15	8226.05

```
df.iloc[:, 1].values
```

```
array([ 8217.4 ,  8214.7 ,  8216.95, ..., 17731.  , 17735.15, 17738.95])
```

```
training_data = df.iloc[:, 1].values
```

```
training_data
```

```
array([ 8217.4 ,  8214.7 ,  8216.95, ..., 17731.  , 17735.15, 17738.95])
```

```
type(training_data)
```

```
numpy.ndarray
```

```
from sklearn.preprocessing import MinMaxScaler
```

```
scaler = MinMaxScaler()
```

```
training_data = scaler.fit_transform(training_data.reshape(-1, 1))
```

```
x_training_data = []

y_training_data =[]
```

```
for i in range(50, len(training_data)):
    x_training_data.append(training_data[i-50:i, 0])
    y_training_data.append(training_data[i, 0])
```

```
x_training_data = np.array(x_training_data)

y_training_data = np.array(y_training_data)
```

```
print(x_training_data.shape)

print(y_training_data.shape)
```

```
(230155, 50)
(230155,)
```

```
x_training_data = np.reshape(x_training_data, (x_training_data.shape[0],

                                               x_training_data.shape[1],

                                               1))
```

```
print(x_training_data.shape)
```

```
(230155, 50, 1)
```

```
rnn = Sequential()
```

```
rnn.add(LSTM(units = 45, return_sequences = True, input_shape = (x_training_data.shape[1], 1)))
```

```
rnn.add(Dropout(0.2))
```

```
rnn.add(LSTM(units = 45, return_sequences = True))

rnn.add(Dropout(0.2))

rnn.add(LSTM(units = 45, return_sequences = True))

rnn.add(Dropout(0.2))

rnn.add(LSTM(units = 45))

rnn.add(Dropout(0.2))
```

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

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

- **`optimizer = 'adam'`**:
    - **Adam (Adaptive Moment Estimation)** is a popular optimization algorithm that computes adaptive learning rates for each parameter. It's often used for training deep learning models, especially LSTM networks, as it adapts the learning rate during training, making it faster and more efficient.
- **`loss = 'mean_squared_error'`**:
    - The **Mean Squared Error (MSE)** is the loss function used for regression tasks. Since you're predicting stock prices (a continuous variable), MSE is a good choice because it penalizes large prediction errors more heavily, which encourages the model to make accurate predictions.
    - **MSE formula**: [MSE = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2] where (y_i) is the actual value and (\hat{y}_i) is the predicted value for the (i)-th data point. Minimizing this error is the goal during training.

```
rnn.fit(x_training_data, y_training_data, epochs = 5, batch_size = 64)
```

```
Epoch 1/5
3597/3597 ━━━━━━━━━━━━━━━━━━━━ 459s 124ms/step - loss: 0.0034
Epoch 2/5
3597/3597 ━━━━━━━━━━━━━━━━━━━━ 477s 133ms/step - loss: 5.8742e-04
Epoch 3/5
3597/3597 ━━━━━━━━━━━━━━━━━━━━ 476s 132ms/step - loss: 5.4500e-04
Epoch 4/5
3597/3597 ━━━━━━━━━━━━━━━━━━━━ 472s 131ms/step - loss: 5.1218e-04
Epoch 5/5
3597/3597 ━━━━━━━━━━━━━━━━━━━━ 380s 106ms/step - loss: 5.0756e-04
<keras.src.callbacks.history.History at 0x1aaa26f2b10>
```

```
rnn.summary()
```

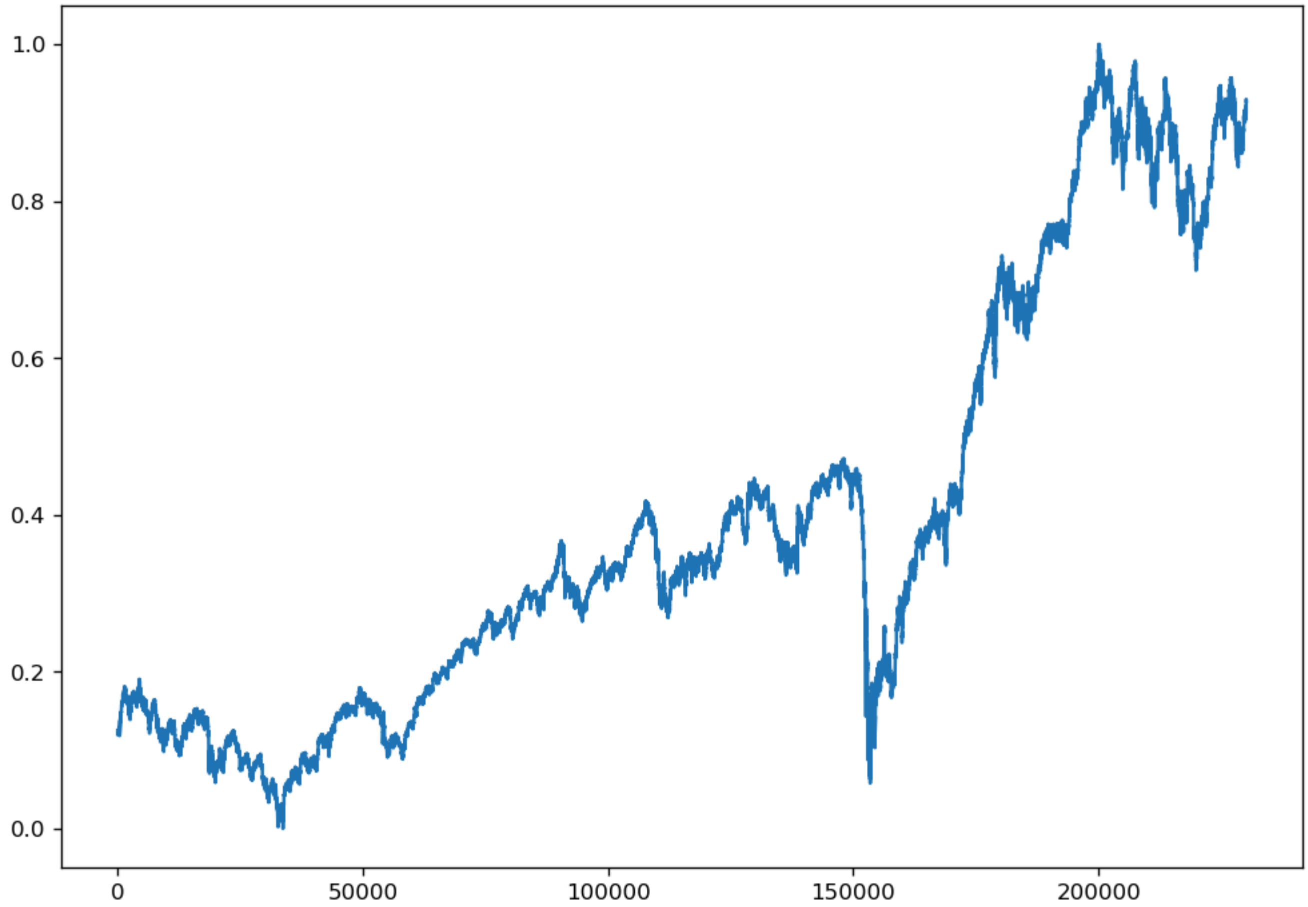
```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
lstm_4 (LSTM)	(None, 50, 45)	8,460
dropout_4 (Dropout)	(None, 50, 45)	0
lstm_5 (LSTM)	(None, 50, 45)	16,380
dropout_5 (Dropout)	(None, 50, 45)	0
lstm_6 (LSTM)	(None, 50, 45)	16,380
dropout_6 (Dropout)	(None, 50, 45)	0
lstm_7 (LSTM)	(None, 45)	16,380
dropout_7 (Dropout)	(None, 45)	0
dense_1 (Dense)	(None, 1)	46

```
Total params: 172,940 (675.55 KB)
Trainable params: 57,646 (225.18 KB)
Non-trainable params: 0 (0.00 B)
```

```
plt.plot(y_training_data)
```

```
[<matplotlib.lines.Line2D at 0x1f3fa818190>]
```




```
predictions = rnn.predict(x_training_data)
```

```
7193/7193 ━━━━━━━━━━━━━━━━━━━━ 115s 16ms/step
```

```
predictions
```

```
array([[0.11938658],
       [0.11965877],
       [0.11989936],
       ...,
       [0.91208017],
       [0.91243684],
       [0.9128008 ]], dtype=float32)
```

```
predictions.shape
```

```
(230155, 1)
```

Making prediction

```
test_data = pd.read_csv('NIFTY50_test_data.csv')
```

```
test_data.head()
```

	Date	Close	High	Low	Open
0	2022-09-01 09:15:00+05:30	17538.80	17567.35	17485.70	17485.70
1	2022-09-01 09:18:00+05:30	17551.10	17557.65	17524.45	17538.30
2	2022-09-01 09:21:00+05:30	17563.40	17573.95	17548.55	17548.55
3	2022-09-01 09:24:00+05:30	17555.30	17564.00	17533.75	17564.00
4	2022-09-01 09:27:00+05:30	17556.45	17558.35	17538.35	17555.15

```
test_data.dtypes
```

```
Date      object
Close    float64
High     float64
Low      float64
Open     float64
dtype: object
```

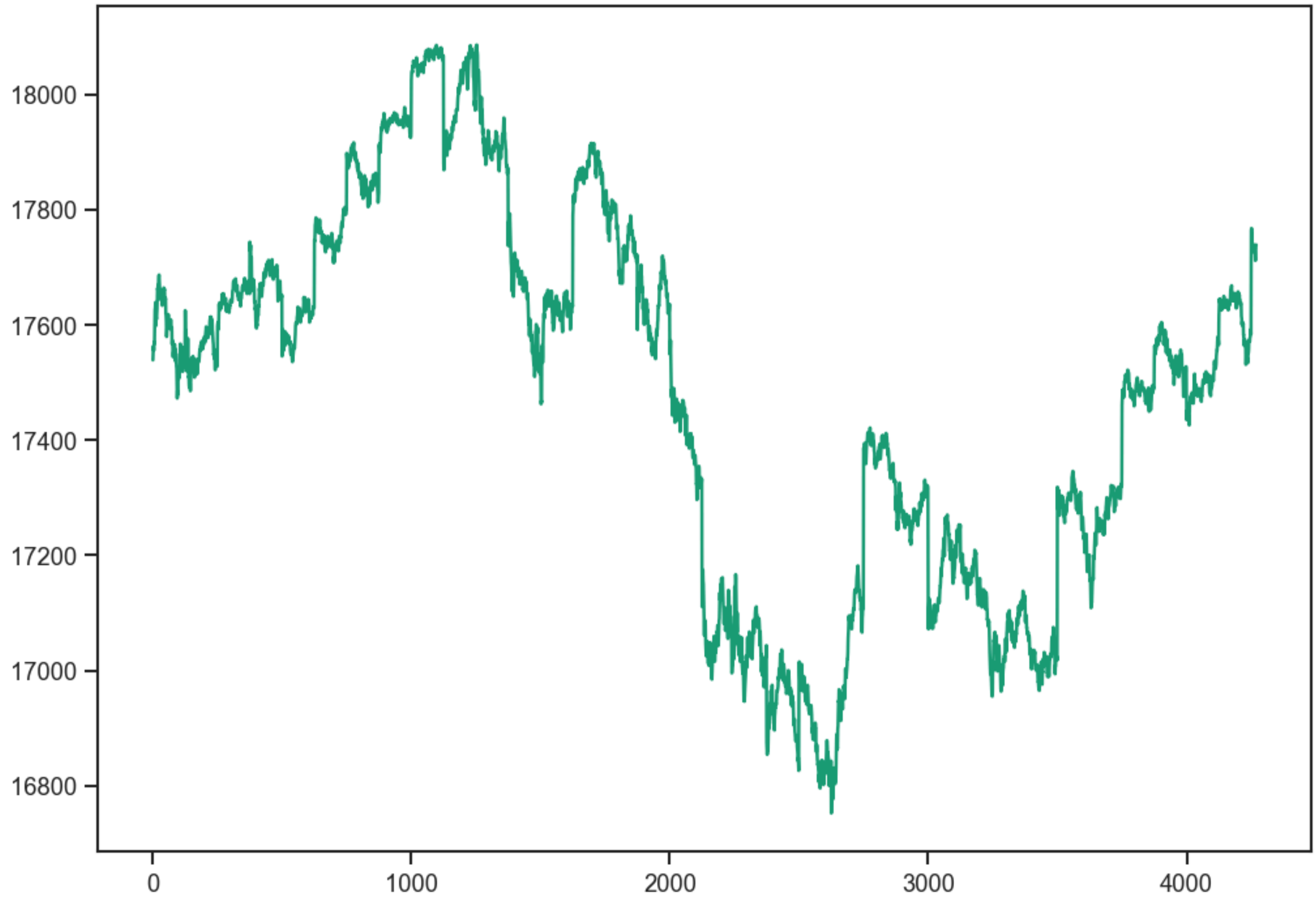
```
test_data = test_data.iloc[:, 1].values
```

```
print(test_data.shape)
```

```
(4270,)
```

```
plt.plot(test_data)
```

```
[<matplotlib.lines.Line2D at 0x1aa9ea83c50>]
```



```
unscaled_training_data = pd.read_csv('NIFTY50_VM.csv')

unscaled_test_data = pd.read_csv('NIFTY50_test_data.csv')

unscaled_training_data.tail(10)
```

	Date	Close	High	Low	Open
230195	2022-10-24 18:45:00+05:30	17735.70	17738.20	17729.70	17735.35
230196	2022-10-24 18:48:00+05:30	17735.05	17737.05	17732.90	17735.70
230197	2022-10-24 18:51:00+05:30	17734.70	17735.90	17731.35	17734.50
230198	2022-10-24 18:54:00+05:30	17731.40	17736.15	17728.90	17733.10
230199	2022-10-24 18:57:00+05:30	17733.65	17734.20	17730.15	17731.05
230200	2022-10-24 19:00:00+05:30	17723.65	17733.85	17721.30	17733.10
230201	2022-10-24 19:03:00+05:30	17711.40	17728.95	17708.40	17723.00
230202	2022-10-24 19:06:00+05:30	17731.00	17732.10	17709.30	17709.30
230203	2022-10-24 19:09:00+05:30	17735.15	17736.10	17728.10	17732.70
230204	2022-10-24 19:12:00+05:30	17738.95	17740.80	17732.20	17734.55

```
all_data=pd.concat((unscaled_training_data['Close'],unscaled_test_data['Close']), axis = 0)

x_test_data = all_data[len(all_data) - len(test_data) - 50:].values

len(x_test_data)
```

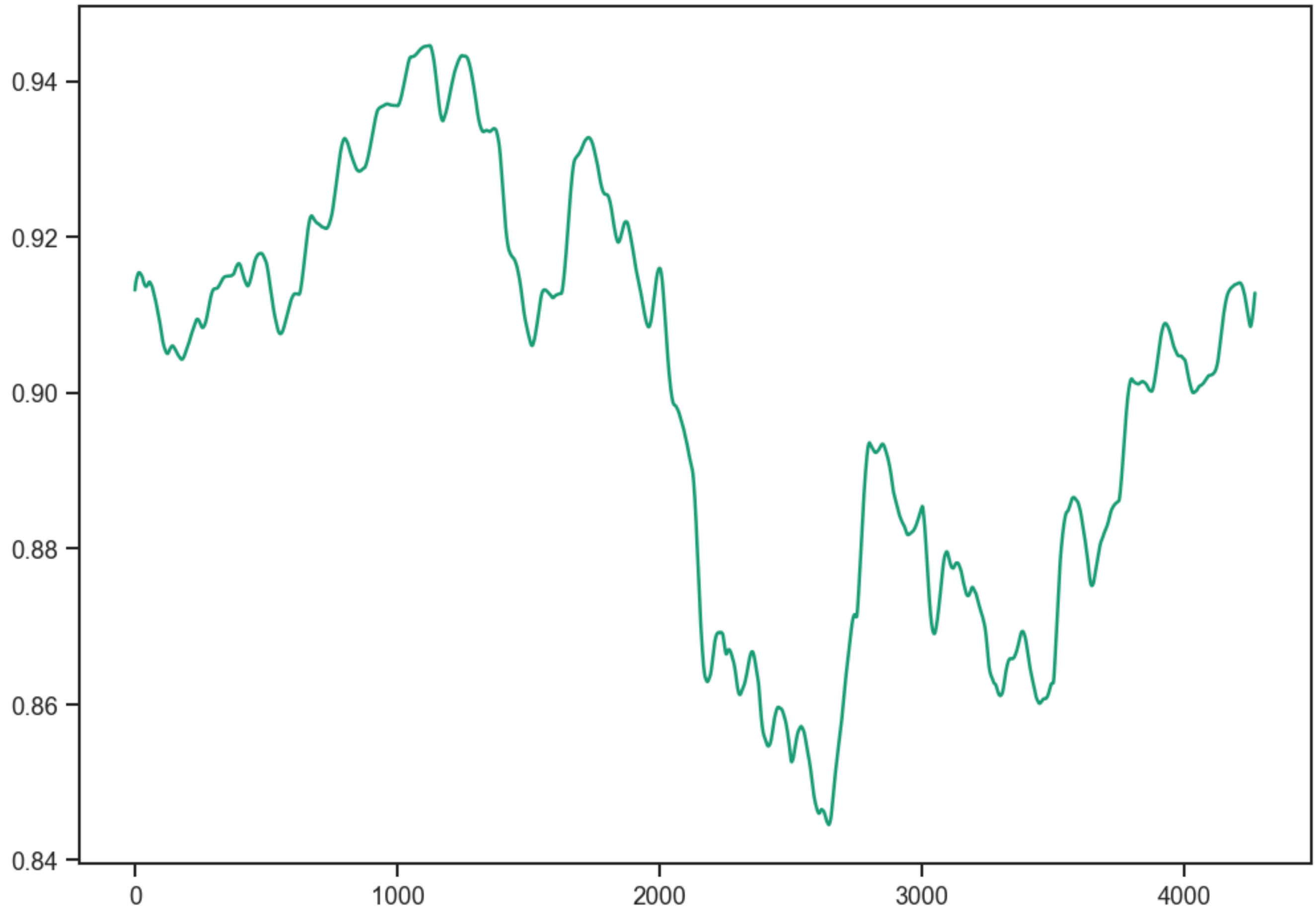
```
4320
```

```
x_test_data = np.reshape(x_test_data, (-1, 1))

x_test_data = scaler.transform(x_test_data)

final_x_test_data = []
for i in range(50, len(x_test_data)):
    final_x_test_data.append(x_test_data[i-50:i, 0])
final_x_test_data = np.array(final_x_test_data)

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

predictions = rnn.predict(final_x_test_data)
```

```
134/134 ━━━━━━━━━━━━━━━━━━━━ 3s 19ms/step
```

```
plt.plot(predictions)
```

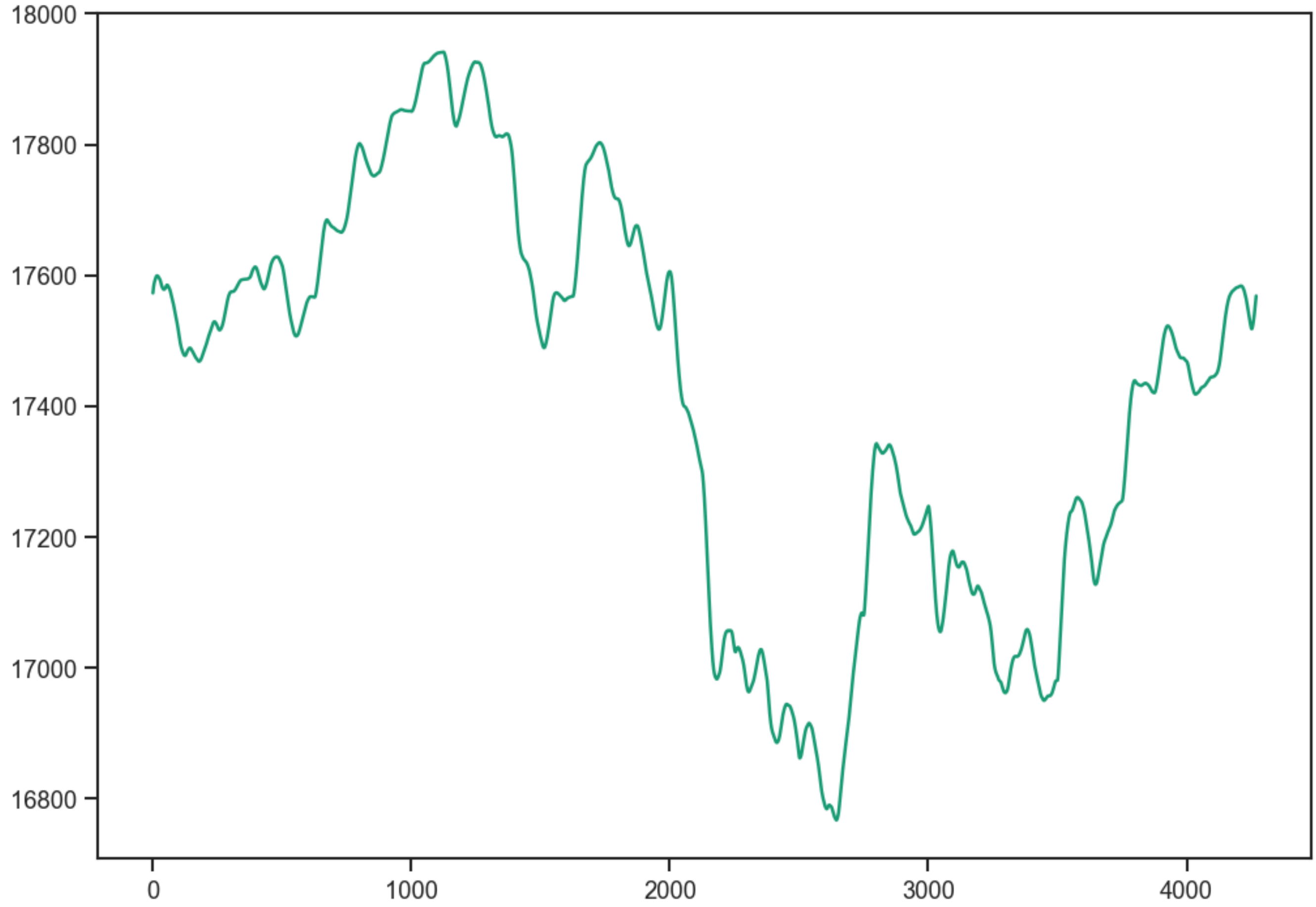
```
[<matplotlib.lines.Line2D at 0x1aa9f1cf210>]
```




```
unscaled_predictions = scaler.inverse_transform(predictions)
```

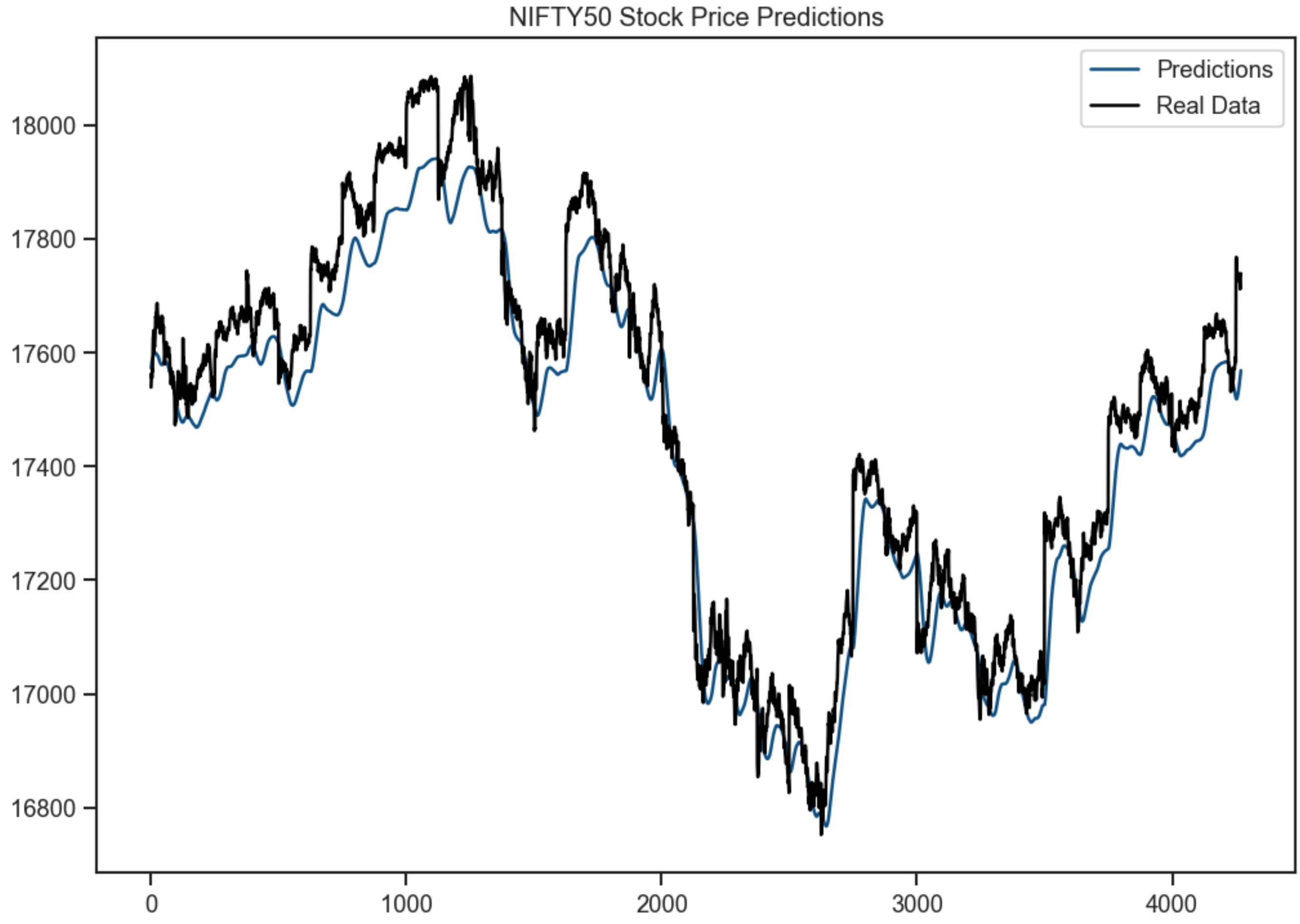
```
plt.plot(unscaled_predictions)
```

```
[<matplotlib.lines.Line2D at 0x1aa9f199190>]
```




```
plt.plot(unscaled_predictions, color = '#135485', label = "Predictions")

plt.plot(test_data, color = 'black', label = "Real Data")

plt.title('NIFTY50 Stock Price Predictions')

plt.legend()
```

```
<matplotlib.legend.Legend at 0x1aaa0a4bd50>
```




```
unscaled_test_data.tail(20)
```

	Date	Close	High	Low	Open
4250	2022-10-24 18:15:00+05:30	17763.35	17776.50	17733.45	17736.35
4251	2022-10-24 18:18:00+05:30	17767.60	17772.10	17760.55	17762.85
4252	2022-10-24 18:21:00+05:30	17760.95	17769.90	17760.95	17767.65
4253	2022-10-24 18:24:00+05:30	17744.30	17761.65	17744.10	17761.65
4254	2022-10-24 18:27:00+05:30	17740.70	17747.90	17735.45	17744.15
4255	2022-10-24 18:30:00+05:30	17739.75	17744.35	17738.15	17739.55
4256	2022-10-24 18:33:00+05:30	17738.00	17743.65	17730.50	17741.20
4257	2022-10-24 18:36:00+05:30	17726.65	17739.00	17726.65	17739.00
4258	2022-10-24 18:39:00+05:30	17739.75	17740.25	17725.40	17727.00
4259	2022-10-24 18:42:00+05:30	17734.90	17739.35	17732.90	17739.35
4260	2022-10-24 18:45:00+05:30	17735.70	17738.20	17729.70	17735.35
4261	2022-10-24 18:48:00+05:30	17735.05	17737.05	17732.90	17735.70
4262	2022-10-24 18:51:00+05:30	17734.70	17735.90	17731.35	17734.50
4263	2022-10-24 18:54:00+05:30	17731.40	17736.15	17728.90	17733.10
4264	2022-10-24 18:57:00+05:30	17733.65	17734.20	17730.15	17731.05
4265	2022-10-24 19:00:00+05:30	17723.65	17733.85	17721.30	17733.10
4266	2022-10-24 19:03:00+05:30	17711.40	17728.95	17708.40	17723.00
4267	2022-10-24 19:06:00+05:30	17731.00	17732.10	17709.30	17709.30
4268	2022-10-24 19:09:00+05:30	17735.15	17736.10	17728.10	17732.70
4269	2022-10-24 19:12:00+05:30	17738.95	17740.80	17732.20	17734.55

```
print("The Prprint("The Prediction 2022-09-01 10:09:00+05:30 is: ",unscaled_predictions[18])
print(ediction for 2022-09-01 09:15:00+05:30 is: ",unscaled_predictions[0])
print("The Real Prediction for 2022-09-01 09:15:00+05:30  is: ",test_data[0])
```

```
The Prediction for 2022-09-01 09:15:00+05:30 is:  [17572.809]
The Real Prediction for 2022-09-01 09:15:00+05:30  is:  17538.8
```

```
print("The Prediction 2022-09-01 10:09:00+05:30 is: ",unscaled_predictions[18])
print("The The Real Prediction 2022-09-01 10:09:00+05:30 is: ",test_data[18])
```

```
The Prediction 2022-09-01 10:09:00+05:30 is:  [17599.068]
The The Real Prediction 2022-09-01 10:09:00+05:30 is:  17663.35
```

```
print("The Prediction for  2022-10-24 19:09:00+05:30 is: ",unscaled_predictions[4268])
print("The Real Prediction for  2022-10-24 19:09:00+05:30 is: ",test_data[4268])
```

```
The Prediction for  2022-10-24 19:09:00+05:30 is:  [17564.184]
The Real Prediction for  2022-10-24 19:09:00+05:30 is:  17735.15
```

```
print("The Prediction for 2022-10-24 18:39:00+05:30  is: ",unscaled_predictions[4258])
print("The Real Prediction for 2022-10-24 18:39:00+05:30  is: ",test_data[4258])
```

```
The Prediction for 2022-10-24 18:39:00+05:30  is:  [17527.934]
The Real Prediction for 2022-10-24 18:39:00+05:30  is:  17739.75
```

```
print("The Prediction for 2022-10-24 18:33:00+05:30 is: ",unscaled_predictions[4256])
print("The Real Prediction for 2022-10-24 18:33:00+05:30  is: ",test_data[4256])
```

```
The Prediction for 2022-10-24 18:33:00+05:30 is:  [17523.143]
The Real Prediction for 2022-10-24 18:33:00+05:30  is:  17738.0
```

```
unscaled_predictions
```

```
array([[17572.809],
       [17576.936],
       [17580.451],
       ...,
       [17559.998],
       [17564.184],
       [17568.457]], dtype=float32)
```

```
pd_prediction = pd.DataFrame(unscaled_predictions,index = unscaled_test_data.index,columns=['LSTMpredictions'])
```

```
unscaled_test_data["prediction"] = np.reshape(unscaled_predictions,(unscaled_predictions.shape[0]))
```

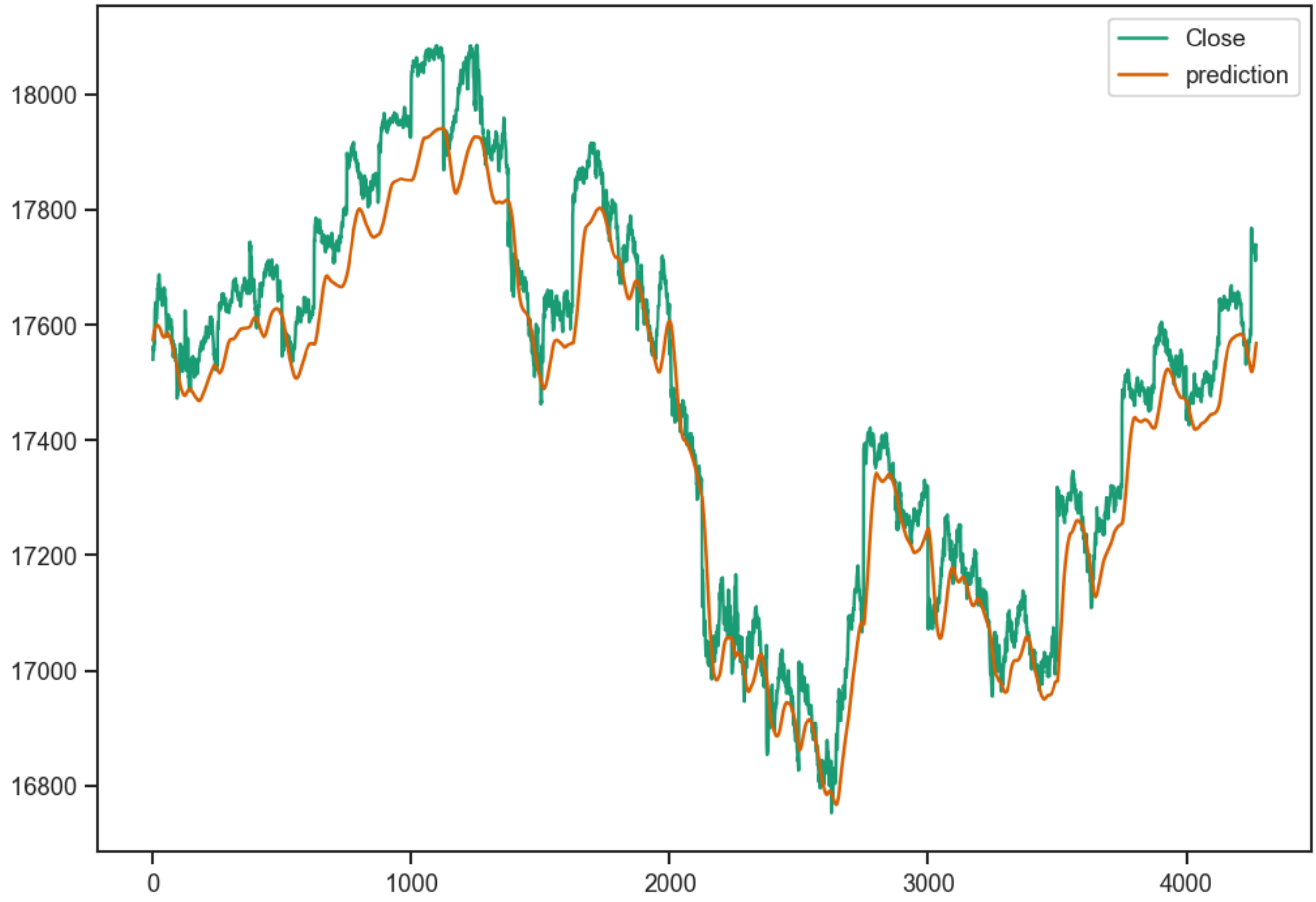
```
unscaled_test_data.head()
```

	Date	Close	High	Low	Open	prediction
0	2022-09-01 09:15:00+05:30	17538.80	17567.35	17485.70	17485.70	17572.808594
1	2022-09-01 09:18:00+05:30	17551.10	17557.65	17524.45	17538.30	17576.935547
2	2022-09-01 09:21:00+05:30	17563.40	17573.95	17548.55	17548.55	17580.451172
3	2022-09-01 09:24:00+05:30	17555.30	17564.00	17533.75	17564.00	17583.427734
4	2022-09-01 09:27:00+05:30	17556.45	17558.35	17538.35	17555.15	17585.902344

```
df["prediction"] = unscaled_test_data["prediction"]
```

```
unscaled_test_data[['Close','prediction']].plot()
```

```
<Axes: >
```




```
# Import evaluation metrics
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.metrics import mean_absolute_percentage_error
```

```
from sklearn.metrics import mean_absolute_error, mean_squared_error
print("RMSE of RNN:", np.sqrt(mean_squared_error(unscaled_test_data.Close, unscaled_test_data.prediction)))
print("\nMAE of RNN:", mean_absolute_error(unscaled_test_data.Close, unscaled_test_data.prediction))
```

```
RMSE of RNN: 89.33396424186066

MAE of RNN: 72.80401337456088
```

```
# Calculating error for Auto-Arima without exgenous features
print("mean absoulte percentage error:", round(mean_absolute_percentage_error(unscaled_test_data.Close, unscaled_test_data.Close),2),"%'
```

```
mean absoulte percentage error: 0.0 %
```

```
#accuracy = 100 * (1 - (MAE / np.mean(unscaled_test_data.Close)))
```

```
accuracy = 100 * (1 - (71.09657302034545 / np.mean(unscaled_test_data.Close)))
```

```
accuracy
```

```
99.59324093035255
```

Build and train the LSTM model

**second model

```
training_data = pd.read_csv('NIFTY50_VM.csv')
```

```
training_data.head()
```

	Date	Close	High	Low	Open
0	2015-01-09 12:24:00+05:30	8217.40	8226.55	8217.15	8226.05
1	2015-01-09 12:27:00+05:30	8214.70	8217.40	8210.35	8217.35
2	2015-01-09 12:30:00+05:30	8216.95	8219.05	8210.40	8214.85
3	2015-01-09 12:33:00+05:30	8209.20	8219.50	8198.40	8217.20
4	2015-01-09 12:36:00+05:30	8202.90	8212.05	8201.00	8209.65

```
training_data = df.iloc[:, 1].values
```

```
type(training_data)
```

```
numpy.ndarray
```

```
max(training_data)
```

```
18592.15
```

```
scaler = MinMaxScaler()
```

```
training_data = scaler.fit_transform(training_data.reshape(-1, 1))
```

```
training_data = scaler.fit_transform(training_data.reshape(-1, 1))
x_training_data = []

y_training_data=[]
```

```
for i in range(50, len(training_data)):
    x_training_data.append(training_data[i-50:i, 0])
    y_training_data.append(training_data[i, 0])
```

```
x_training_data = np.array(x_training_data)

y_training_data = np.array(y_training_data)
```

```
print(x_training_data.shape)

print(y_training_data.shape)
```

```
(230155, 50)
(230155,)
```

```
x_training_data = np.reshape(x_training_data, (x_training_data.shape[0],

                                               x_training_data.shape[1],

                                               1))
```

```
lstm_model=Sequential()
```

```
lstm_model.add(LSTM(units = 100, return_sequences = True, input_shape = (x_training_data.shape[1], 1)))
```

```
lstm_model.add(LSTM(units = 45, return_sequences = True))

lstm_model.add(Dropout(0.2))

lstm_model.add(LSTM(units = 45, return_sequences = True))

lstm_model.add(Dropout(0.2))

lstm_model.add(LSTM(units = 45))

lstm_model.add(Dropout(0.2))
```

```
lstm_model.add(Dense(units = 1))
lstm_model.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

```
lstm_model.fit(x_training_data, y_training_data, epochs = 10, batch_size = 64)
```

```
Epoch 1/10
3597/3597 ━━━━━━━━━━━━━━━━━━━━ 272s 74ms/step - loss: 0.0041
Epoch 2/10
3597/3597 ━━━━━━━━━━━━━━━━━━━━ 258s 72ms/step - loss: 6.0615e-04
Epoch 3/10
3597/3597 ━━━━━━━━━━━━━━━━━━━━ 258s 72ms/step - loss: 5.1772e-04
Epoch 4/10
3597/3597 ━━━━━━━━━━━━━━━━━━━━ 258s 72ms/step - loss: 5.0120e-04
Epoch 5/10
3597/3597 ━━━━━━━━━━━━━━━━━━━━ 260s 72ms/step - loss: 5.0500e-04
Epoch 6/10
3597/3597 ━━━━━━━━━━━━━━━━━━━━ 261s 73ms/step - loss: 4.9159e-04
Epoch 7/10
3597/3597 ━━━━━━━━━━━━━━━━━━━━ 261s 73ms/step - loss: 4.7799e-04
Epoch 8/10
3597/3597 ━━━━━━━━━━━━━━━━━━━━ 260s 72ms/step - loss: 4.8221e-04
Epoch 9/10
3597/3597 ━━━━━━━━━━━━━━━━━━━━ 261s 73ms/step - loss: 4.7430e-04
Epoch 10/10
3597/3597 ━━━━━━━━━━━━━━━━━━━━ 263s 73ms/step - loss: 4.7314e-04
<keras.src.callbacks.history.History at 0x1aa9f17a450>
```

```
lstm_model.summary()
```

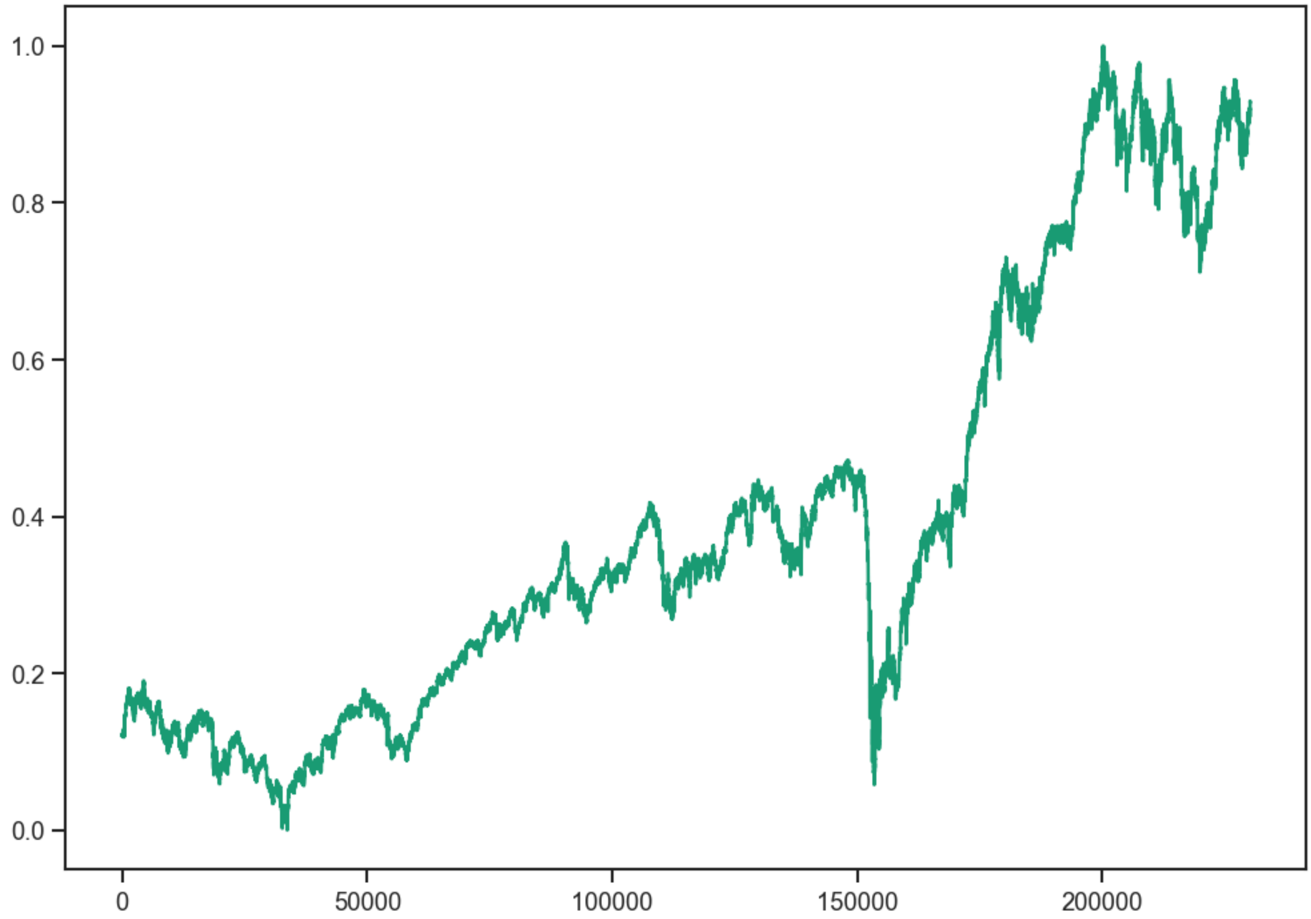
Model: "sequential_2"

Layer (type)	Output Shape	Param #
lstm_8 (LSTM)	(None, 50, 100)	40,800
lstm_9 (LSTM)	(None, 50, 45)	26,280
dropout_8 (Dropout)	(None, 50, 45)	0
lstm_10 (LSTM)	(None, 50, 45)	16,380
dropout_9 (Dropout)	(None, 50, 45)	0
lstm_11 (LSTM)	(None, 45)	16,380
dropout_10 (Dropout)	(None, 45)	0
dense_2 (Dense)	(None, 1)	46

Total params: 299,660 (1.14 MB)
Trainable params: 99,886 (390.18 KB)
Non-trainable params: 0 (0.00 B)

```
plt.plot(y_training_data)
```

```
[<matplotlib.lines.Line2D at 0x1aa9a34dcd0>]
```



**Third model

```
training_data = pd.read_csv('NIFTY50_VM.csv')

training_data = training_data.iloc[:, 1].values

scaler = MinMaxScaler()

training_data = scaler.fit_transform(training_data.reshape(-1, 1))


x_training_data = []

y_training_data=[]

for i in range(50, len(training_data)):
    x_training_data.append(training_data[i-50:i, 0])
    y_training_data.append(training_data[i, 0])

x_training_data = np.array(x_training_data)

y_training_data = np.array(y_training_data)

x_training_data = np.reshape(x_training_data, (x_training_data.shape[0],

                                               x_training_data.shape[1],

                                               1))

lstm_model=Sequential()

lstm_model.add(LSTM(units = 45, return_sequences = True, input_shape = (x_training_data.shape[1], 1)))

lstm_model.add(Dropout(0.2))
```

```
lstm_model.add(LSTM(units = 45, return_sequences = True))

lstm_model.add(Dropout(0.2))

lstm_model.add(LSTM(units = 45, return_sequences = True))

lstm_model.add(Dropout(0.2))

lstm_model.add(LSTM(units = 45))

lstm_model.add(Dropout(0.2))


lstm_model.add(Dense(units = 1))
lstm_model.compile(optimizer = 'adam', loss = 'mean_squared_error')


lstm_model.fit(x_training_data, y_training_data, epochs = 10, batch_size = 64)
```

```
Epoch 1/10
3597/3597 ━━━━━━━━━━━━━━━━━━━━ 223s 61ms/step - loss: 0.0035
Epoch 2/10
3597/3597 ━━━━━━━━━━━━━━━━━━━━ 219s 61ms/step - loss: 5.7741e-04
Epoch 3/10
3597/3597 ━━━━━━━━━━━━━━━━━━━━ 217s 60ms/step - loss: 5.2680e-04
Epoch 4/10
3597/3597 ━━━━━━━━━━━━━━━━━━━━ 216s 60ms/step - loss: 5.1887e-04
Epoch 5/10
3597/3597 ━━━━━━━━━━━━━━━━━━━━ 218s 60ms/step - loss: 5.0407e-04
Epoch 6/10
3597/3597 ━━━━━━━━━━━━━━━━━━━━ 221s 61ms/step - loss: 4.9668e-04
Epoch 7/10
3597/3597 ━━━━━━━━━━━━━━━━━━━━ 329s 92ms/step - loss: 4.8984e-04
Epoch 8/10
3597/3597 ━━━━━━━━━━━━━━━━━━━━ 410s 114ms/step - loss: 4.8677e-04
Epoch 9/10
3597/3597 ━━━━━━━━━━━━━━━━━━━━ 478s 133ms/step - loss: 4.8263e-04
Epoch 10/10
3597/3597 ━━━━━━━━━━━━━━━━━━━━ 476s 132ms/step - loss: 4.8301e-04
<keras.src.callbacks.history.History at 0x1aaf3ca6510>
```

```
lstm_model.summary()
```

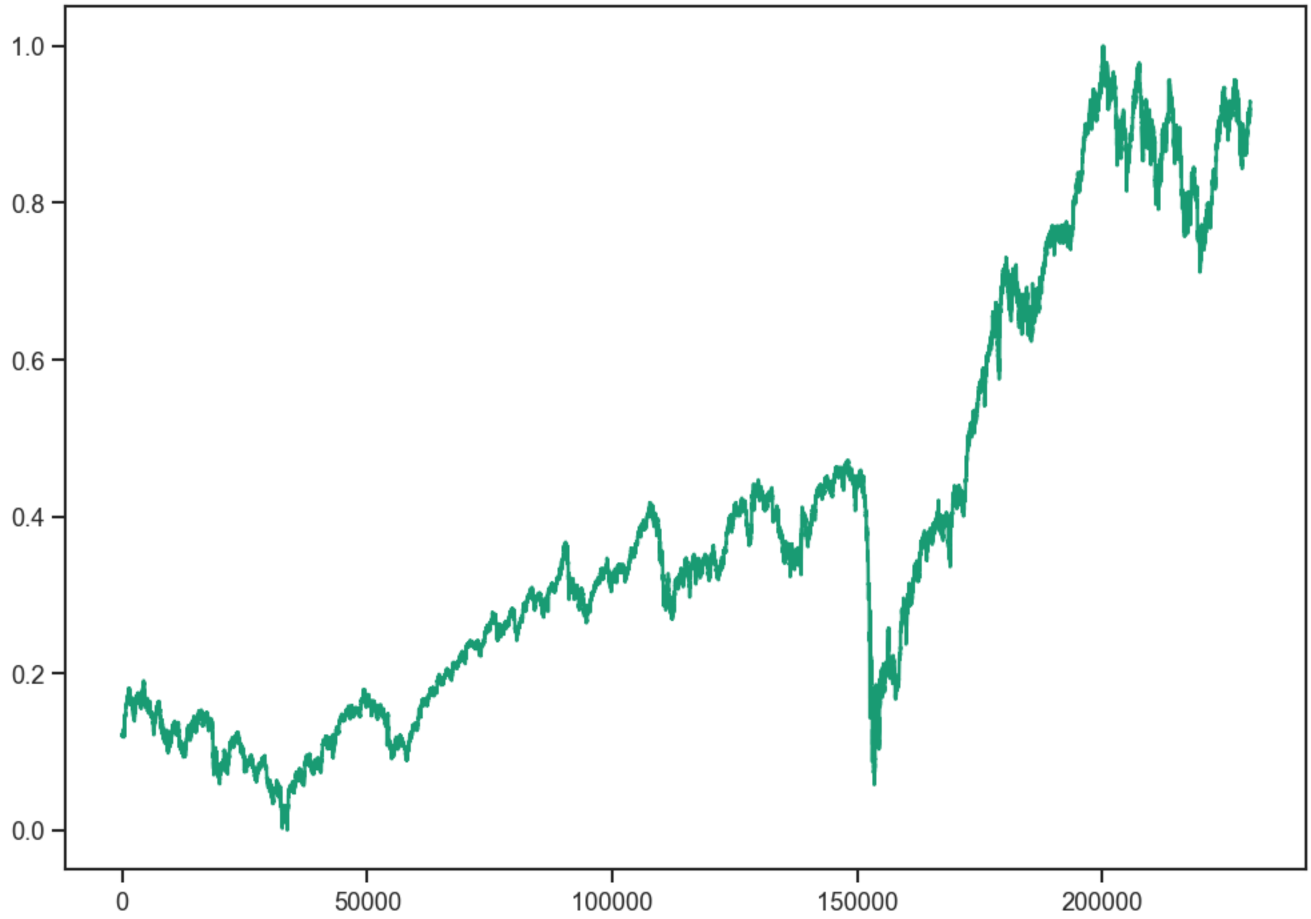
Model: "sequential_4"

Layer (type)	Output Shape	Param #
lstm_16 (LSTM)	(None, 50, 45)	8,460
dropout_15 (Dropout)	(None, 50, 45)	0
lstm_17 (LSTM)	(None, 50, 45)	16,380
dropout_16 (Dropout)	(None, 50, 45)	0
lstm_18 (LSTM)	(None, 50, 45)	16,380
dropout_17 (Dropout)	(None, 50, 45)	0
lstm_19 (LSTM)	(None, 45)	16,380
dropout_18 (Dropout)	(None, 45)	0
dense_4 (Dense)	(None, 1)	46

Total params: 172,940 (675.55 KB)
Trainable params: 57,646 (225.18 KB)
Non-trainable params: 0 (0.00 B)

```
plt.plot(y_training_data)
```

```
[<matplotlib.lines.Line2D at 0x1aaa42d5cd0>]
```



Make predictions

```
Predict = lstm_model.predict(x_training_data)
```

```
7193/7193 ━━━━━━━━━━━━━━━━━━━━ 128s 18ms/step
```

```
Predict
```

```
array([[0.10847098],
       [0.10883221],
       [0.10908124],
       ...,
       [0.9417343 ],
       [0.9421647 ],
       [0.9425963 ]], dtype=float32)
```

```
Predict.shape
```

```
(230155, 1)
```

```
test_data = pd.read_csv('NIFTY50_test_data.csv')
```

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

```
unscaled_training_data = pd.read_csv('NIFTY50_VM.csv')
unscaled_test_data = pd.read_csv('NIFTY50_test_data.csv')
```

```
all_data=pd.concat((unscaled_training_data['Close'],unscaled_test_data['Close']), axis = 0)
```

```
x_test_data = all_data[len(all_data) - len(test_data) - 50:].values
```

```
x_test_data = np.reshape(x_test_data, (-1, 1))
```

```
x_test_data = scaler.transform(x_test_data)


final_x_test_data = []

for i in range(50, len(x_test_data)):

    final_x_test_data.append(x_test_data[i-50:i, 0])

final_x_test_data = np.array(final_x_test_data)


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


Predict = lstm_model.predict(final_x_test_data)
```

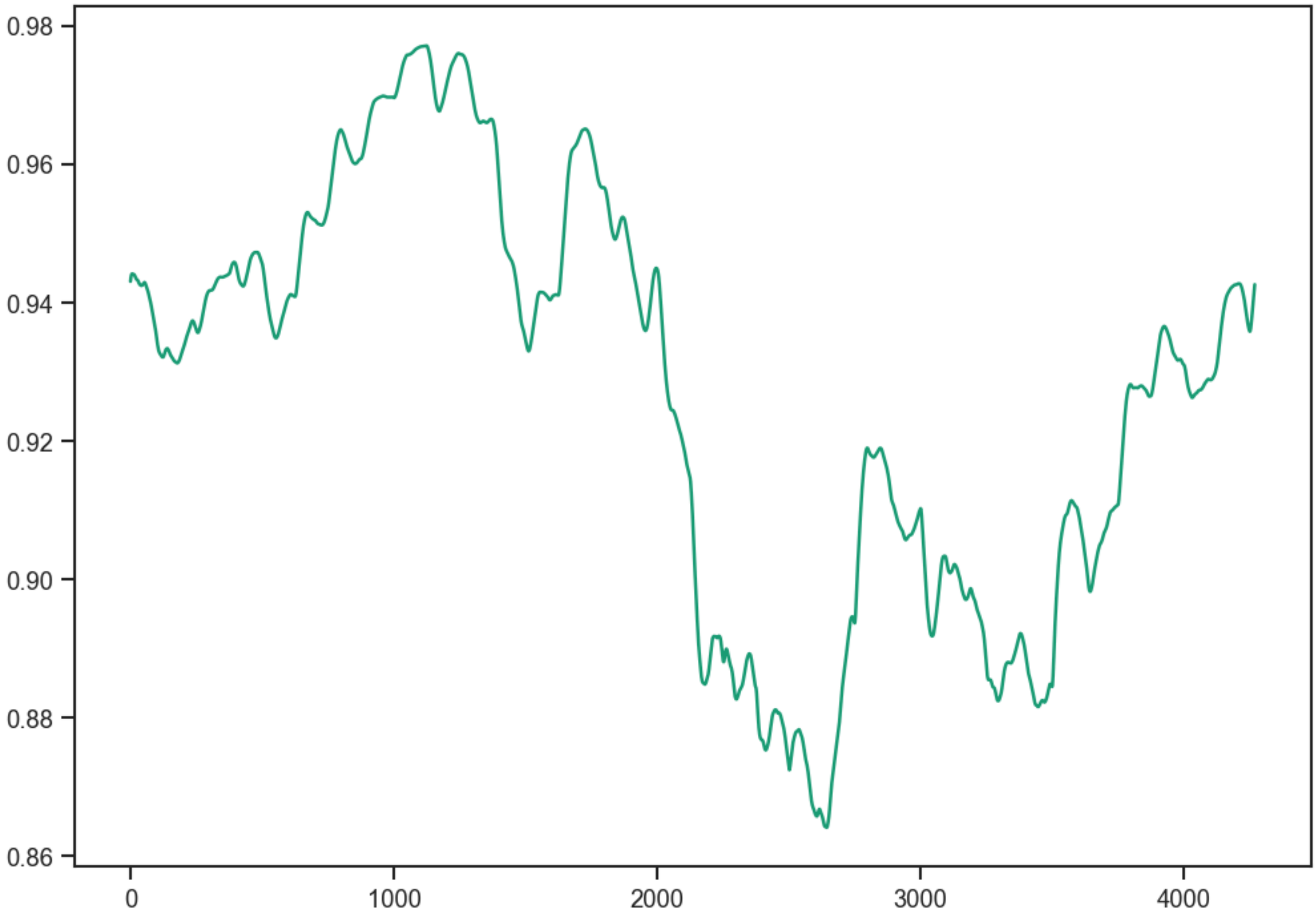
```
134/134 ━━━━━━━━━━━━━━━━━━━━ 2s 16ms/step
```

```
predictions
```

```
array([[0.9131715 ],
       [0.91352296],
       [0.91382253],
       ...,
       [0.91208017],
       [0.91243684],
       [0.9128008 ]], dtype=float32)
```

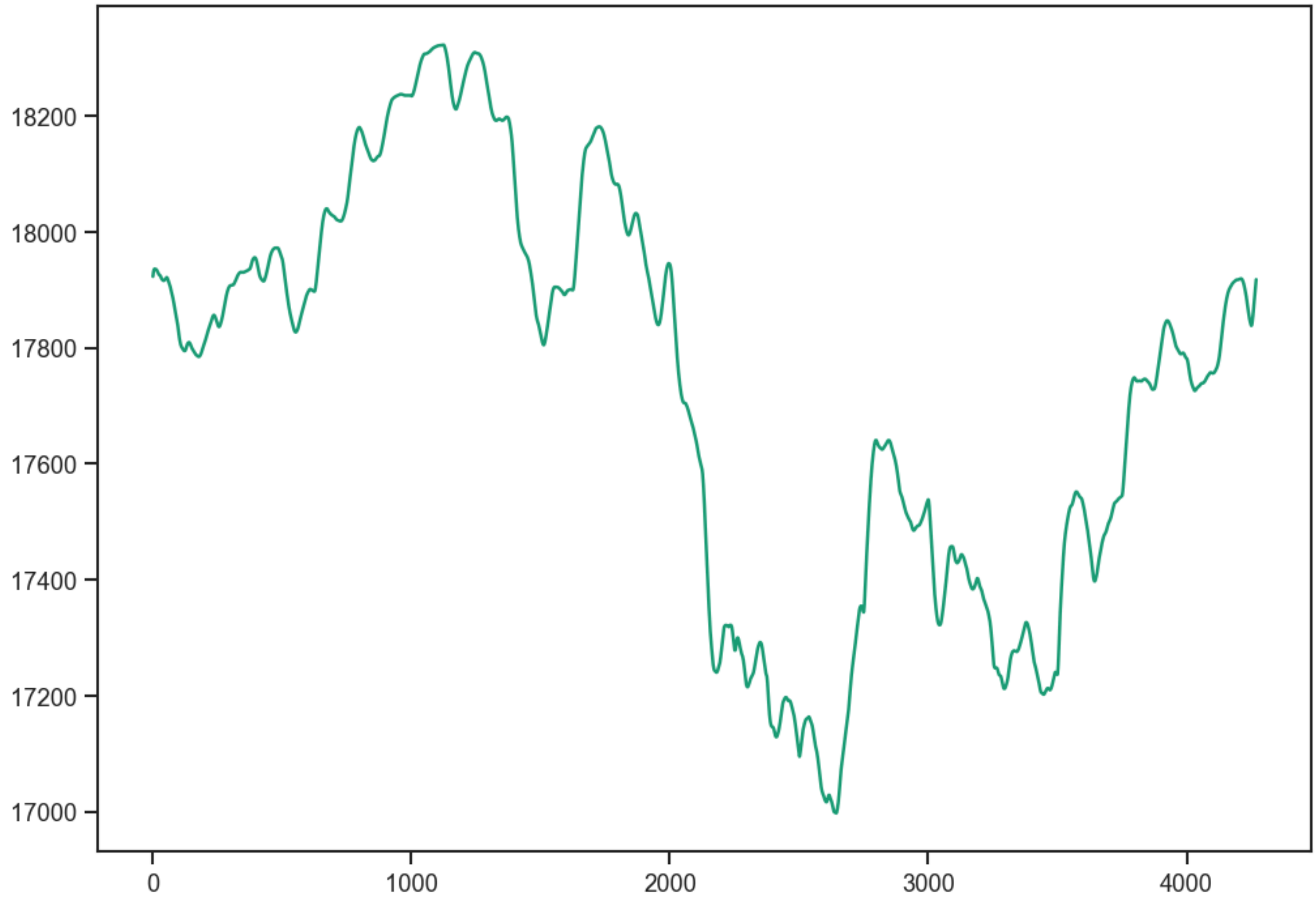
```
unscaled_predictions = scaler.inverse_transform(Predict)


plt.plot(Predict)
```

```
[<matplotlib.lines.Line2D at 0x1aaf880dcd0>]
```




```
plt.plot(unscaled_predictions)
```

```
[<matplotlib.lines.Line2D at 0x1aaff480110>]
```




```
plt.plot(unscaled_predictions, color = '#135485', label = "Predict")

plt.plot(test_data, color = 'black', label = "Real Data")

plt.title('NIFTY50 Stock Price Predictions')

plt.legend()
```

```
<matplotlib.legend.Legend at 0x1aaef0bdf90>
```


```
unscaled_test_data.tail(20)
```

	Date	Close	High	Low	Open
4250	2022-10-24 18:15:00+05:30	17763.35	17776.50	17733.45	17736.35
4251	2022-10-24 18:18:00+05:30	17767.60	17772.10	17760.55	17762.85
4252	2022-10-24 18:21:00+05:30	17760.95	17769.90	17760.95	17767.65
4253	2022-10-24 18:24:00+05:30	17744.30	17761.65	17744.10	17761.65
4254	2022-10-24 18:27:00+05:30	17740.70	17747.90	17735.45	17744.15
4255	2022-10-24 18:30:00+05:30	17739.75	17744.35	17738.15	17739.55
4256	2022-10-24 18:33:00+05:30	17738.00	17743.65	17730.50	17741.20
4257	2022-10-24 18:36:00+05:30	17726.65	17739.00	17726.65	17739.00
4258	2022-10-24 18:39:00+05:30	17739.75	17740.25	17725.40	17727.00
4259	2022-10-24 18:42:00+05:30	17734.90	17739.35	17732.90	17739.35
4260	2022-10-24 18:45:00+05:30	17735.70	17738.20	17729.70	17735.35
4261	2022-10-24 18:48:00+05:30	17735.05	17737.05	17732.90	17735.70
4262	2022-10-24 18:51:00+05:30	17734.70	17735.90	17731.35	17734.50
4263	2022-10-24 18:54:00+05:30	17731.40	17736.15	17728.90	17733.10
4264	2022-10-24 18:57:00+05:30	17733.65	17734.20	17730.15	17731.05
4265	2022-10-24 19:00:00+05:30	17723.65	17733.85	17721.30	17733.10
4266	2022-10-24 19:03:00+05:30	17711.40	17728.95	17708.40	17723.00
4267	2022-10-24 19:06:00+05:30	17731.00	17732.10	17709.30	17709.30
4268	2022-10-24 19:09:00+05:30	17735.15	17736.10	17728.10	17732.70
4269	2022-10-24 19:12:00+05:30	17738.95	17740.80	17732.20	17734.55