In [64]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import yfinance as yf
import warnings
warnings.filterwarnings('ignore')
import datetime as dt
```

In [3]:
```
# Collecting data from yfinance website using API
df = yf.download(tickers="^NSEI",start="2023-01-01",end=dt.datetime.today(),inte
```

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

In [4]:
```
#df = pd.read_excel('NIFTY.xlsx')
```

In [5]:
```
df
```

Out[5]:

	Open	High	Low	Close	Adj Close	Volume
Date						
2023-01-02	18131.699219	18215.150391	18086.500000	18197.449219	18197.449219	256100
2023-01-03	18163.199219	18251.949219	18149.800781	18232.550781	18232.550781	208700
2023-01-04	18230.650391	18243.000000	18020.599609	18042.949219	18042.949219	235200
2023-01-05	18101.949219	18120.300781	17892.599609	17992.150391	17992.150391	269900
2023-01-06	18008.050781	18047.400391	17795.550781	17859.449219	17859.449219	238200
...	...	...	...	...	...	...
2024-08-13	24342.349609	24359.949219	24116.500000	24139.000000	24139.000000	239700
2024-08-14	24184.400391	24196.500000	24099.699219	24143.750000	24143.750000	303300
2024-08-16	24334.849609	24563.900391	24204.500000	24541.150391	24541.150391	271600
2024-08-19	24636.349609	24638.800781	24522.949219	24572.650391	24572.650391	243600
2024-08-20	24648.900391	24734.300781	24607.199219	24688.800781	24688.800781	0

400 rows × 6 columns

In [7]:
```
df.T # Transform data
```

Out[7]:

Date	2023-01-02	2023-01-03	2023-01-04	2023-01-05	2023-01-06
Open	18131.699219	18163.199219	18230.650391	18101.949219	18008.050781
High	18215.150391	18251.949219	18243.000000	18120.300781	18047.400391
Low	18086.500000	18149.800781	18020.599609	17892.599609	17795.550781
Close	18197.449219	18232.550781	18042.949219	17992.150391	17859.449219
Adj Close	18197.449219	18232.550781	18042.949219	17992.150391	17859.449219
Volume	256100.000000	208700.000000	235200.000000	269900.000000	238200.000000

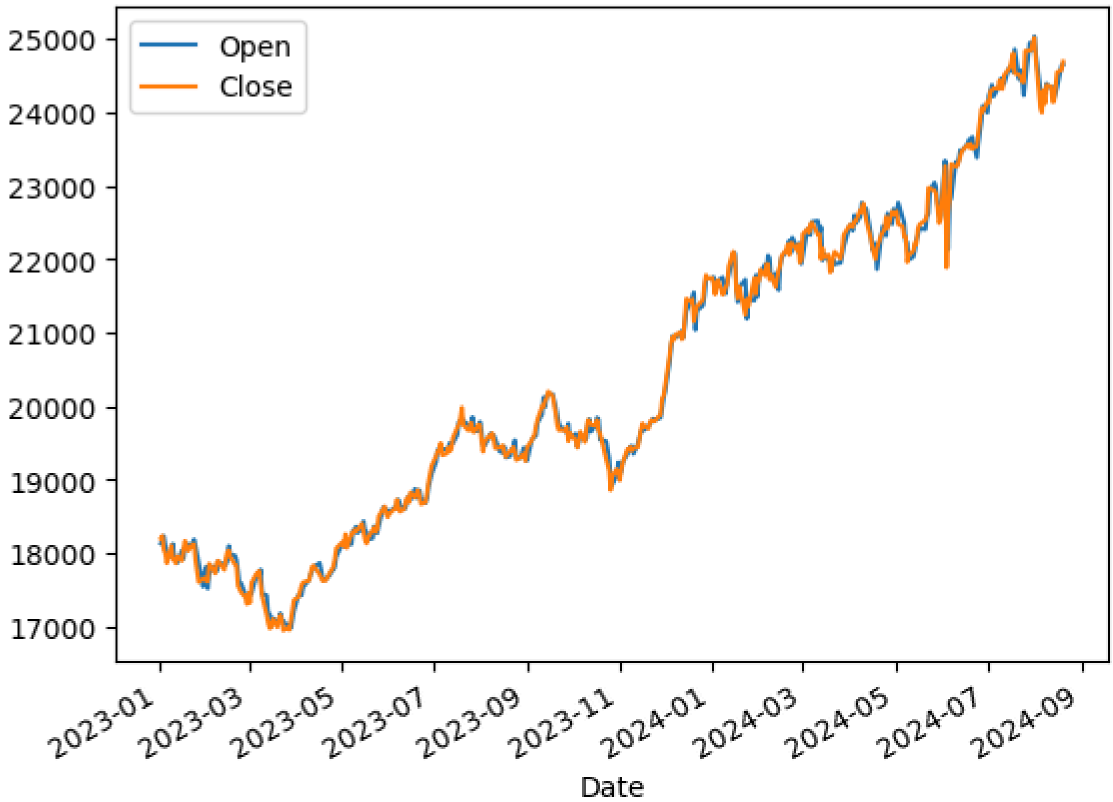
6 rows × 400 columns

In [8]:

```
plt.figure()
df[["Open","Close"]].plot()
#plt.legend(loc="best")
```

Out[8]: <Axes: xlabel='Date'>

<Figure size 640x480 with 0 Axes>



In [9]:

```
df.isnull().sum()
```

Out[9]:

```
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

In [10]:
```
df.shape
```
Out[10]:
```
(400, 6)
```

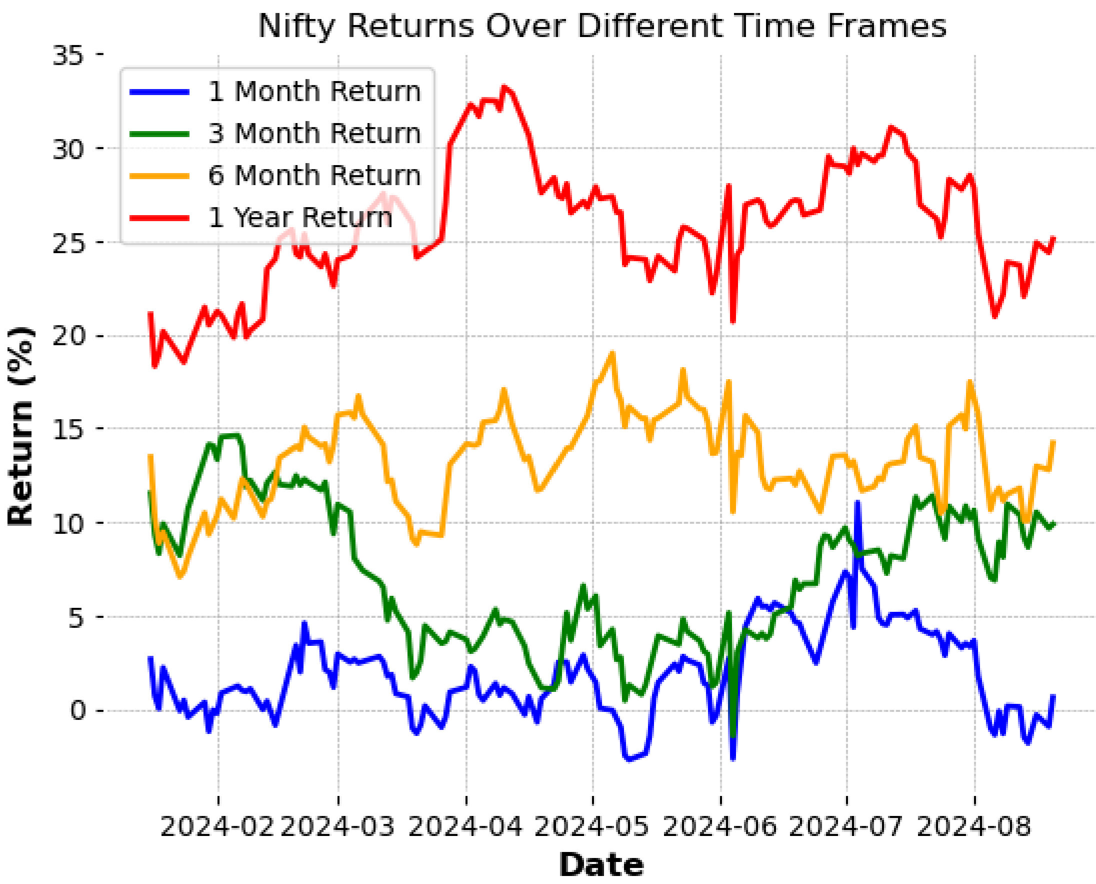
In [11]:
```
df.describe()
```
Out[11]:

	Open	High	Low	Close	Adj Close	Vol
count	400.000000	400.000000	400.000000	400.000000	400.000000	4.000000e
mean	20423.912236	20501.207485	20317.482725	20414.729014	20414.729014	2.881818e
std	2188.122149	2204.156938	2174.961959	2192.739565	2192.739565	1.008678e
min	16977.300781	17061.750000	16828.349609	16945.050781	16945.050781	0.000000e
25%	18590.499512	18598.000488	18482.487305	18534.325195	18534.325195	2.318000e
50%	19768.950195	19825.575195	19691.024414	19747.125000	19747.125000	2.683000e
75%	22164.999512	22276.913086	22046.324707	22160.162598	22160.162598	3.331500e
max	25030.949219	25078.300781	24956.400391	25010.900391	25010.900391	1.006100e

In [12]:
```
df.info()
```
```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 400 entries, 2023-01-02 to 2024-08-20
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Open       400 non-null    float64
 1   High       400 non-null    float64
 2   Low        400 non-null    float64
 3   Close      400 non-null    float64
 4   Adj Close  400 non-null    float64
 5   Volume     400 non-null    int64
dtypes: float64(5), int64(1)
memory usage: 21.9 KB
```

In [61]:
```
def calculate_return(df,period):
    df['Return_{}'.format(period)] = df['Close'].pct_change(periods=period)*100
    return df

# 1 Month return
nifty_return = calculate_return(df,21)

# 3 Month Return
nifty_return = calculate_return(df,63)

# 6 Month Return
nifty_return = calculate_return(df,126)

# 1 Year Return
nifty_return = calculate_return(df,256)

nifty_return = nifty_return.dropna()
```

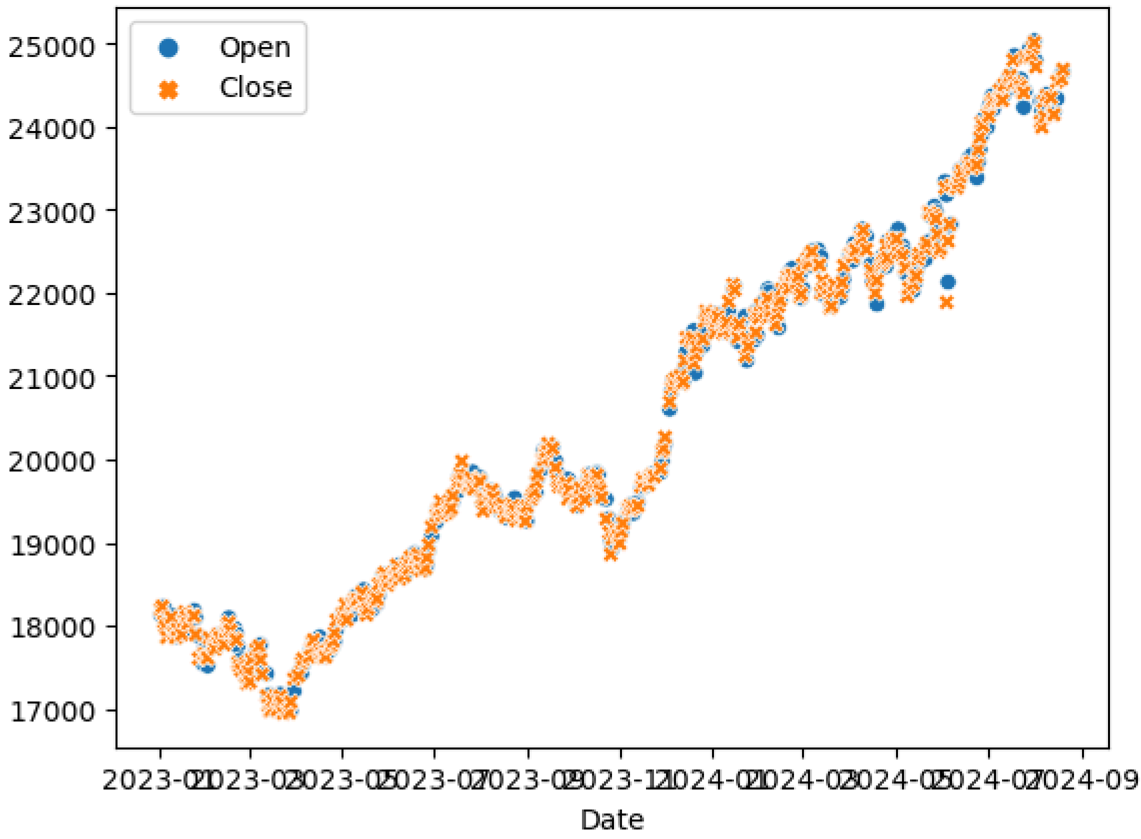
```
plt.plot(nifty_return.index,nifty_return['Return_21'],label = '1 Month Return',
plt.plot(nifty_return.index,nifty_return['Return_63'],label = '3 Month Return',
plt.plot(nifty_return.index,nifty_return['Return_126'],label = '6 Month Return',
plt.plot(nifty_return.index,nifty_return['Return_256'],label = '1 Year Return',

plt.xlabel('Date')
plt.ylabel('Return (%)')
plt.title('Nifty Returns Over Different Time Frames')
plt.legend()
plt.grid(True)
plt.show()
```



In [13]:
```
#sns.heatmap(df)
```

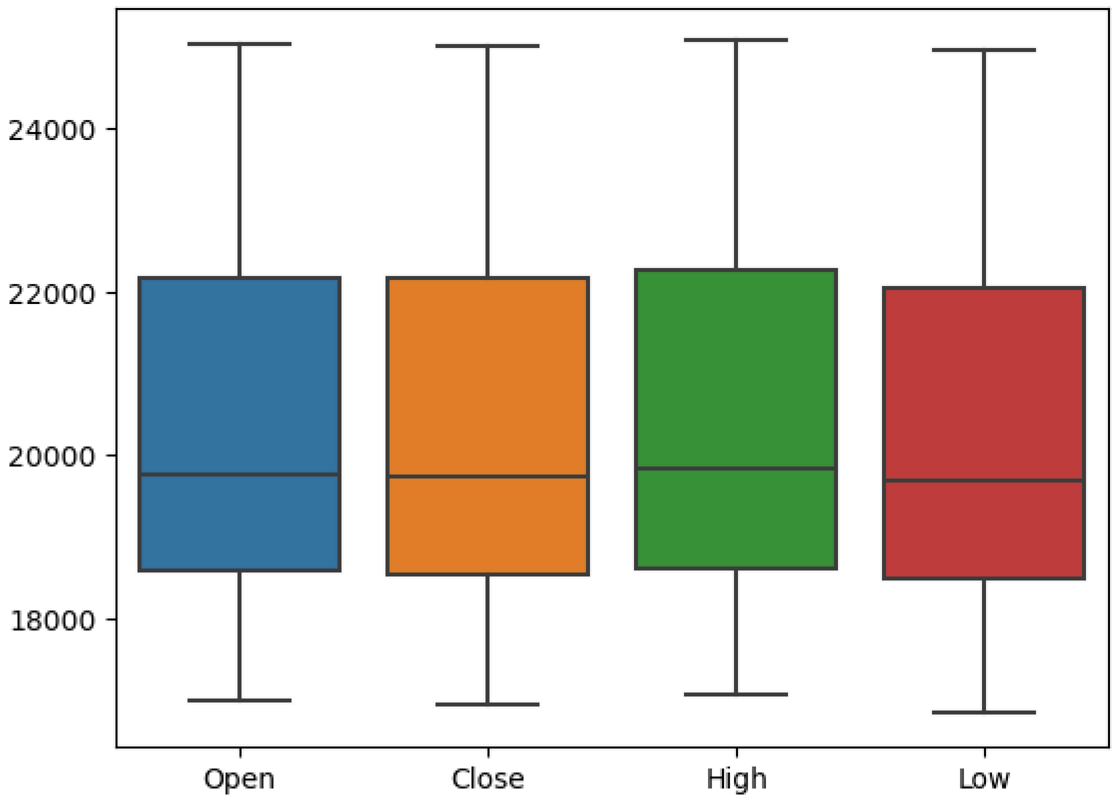
In [14]:
```
#sns.scatterplot(df["Open"])
sns.scatterplot(df[["Open","Close"]])
```

Out[14]: <Axes: xlabel='Date'>



In [15]:
```
sns.boxplot(df[["Open","Close","High","Low"]])
```

Out[15]: <Axes: >



In [16]:
```
#deleting unneccesarry columns from data
df.columns
```

Out[16]: Index(['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'], dtype='object')

In [17]:
```
df.drop(["Adj Close","Volume"],axis=1,inplace=True)
```

In [18]:
```
df
```

Out[18]:

Date	Open	High	Low	Close
2023-01-02	18131.699219	18215.150391	18086.500000	18197.449219
2023-01-03	18163.199219	18251.949219	18149.800781	18232.550781
2023-01-04	18230.650391	18243.000000	18020.599609	18042.949219
2023-01-05	18101.949219	18120.300781	17892.599609	17992.150391
2023-01-06	18008.050781	18047.400391	17795.550781	17859.449219
...	...	...	...	...
2024-08-13	24342.349609	24359.949219	24116.500000	24139.000000
2024-08-14	24184.400391	24196.500000	24099.699219	24143.750000
2024-08-16	24334.849609	24563.900391	24204.500000	24541.150391
2024-08-19	24636.349609	24638.800781	24522.949219	24572.650391
2024-08-20	24648.900391	24734.300781	24607.199219	24688.800781

400 rows × 4 columns

In [19]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 400 entries, 2023-01-02 to 2024-08-20
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Open    400 non-null    float64
 1   High    400 non-null    float64
 2   Low     400 non-null    float64
 3   Close   400 non-null    float64
dtypes: float64(4)
memory usage: 15.6 KB
```

In [20]:
```
# ML Model Building Process
```

In [21]:
```
# Seperating x and y
# Where x contains all the features
# y contains output/target column values.
```

In [22]:
```
x = df.iloc[:,0:3]
y = df["Close"]
```

In [23]:
```
# Linear Regression Algorithm
from sklearn.model_selection import train_test_split

xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size=0.3,random_state=1)
```

In [24]:
```
from sklearn.linear_model import LinearRegression

# Object for Linear Regression class
lr = LinearRegression()

# fitting trainig data
lr.fit(xtrain,ytrain)

# prediction on testing data
ypred = lr.predict(xtest)
ypred
```

Out[24]:
```
array([24542.42884054, 19486.73368014, 22307.44963381, 22928.94399682,
       19966.51925513, 22900.51007753, 19641.55878714, 22458.28720204,
       17778.24696443, 22193.09669802, 20097.25890749, 24224.85821378,
       19641.46762444, 22685.54789895, 22583.61731352, 20782.26347985,
       24406.47880421, 18112.04693168, 18029.71671973, 20072.77576352,
       19322.14218696, 21650.29477853, 24381.80681853, 19632.31850846,
       17924.73471523, 18123.66222894, 24100.02281414, 18903.18415762,
       23715.7428435 , 19245.12570739, 18239.44649444, 17893.37040203,
       19435.50187442, 21806.12729401, 17939.33917354, 18061.22441476,
       17035.07134113, 18515.6873069 , 24441.61871548, 19799.00947428,
       21815.14078776, 19104.53019697, 19974.0787367 , 17344.38342847,
       24916.25016294, 18249.81748649, 24048.71001901, 22471.15394933,
       22353.06036291, 22111.06689114, 17659.77937229, 20268.02126731,
       20873.33263823, 17615.7736341 , 22012.39356744, 21987.90185493,
       21719.88869085, 18683.28577581, 21430.68450345, 23508.32439212,
       18599.35295054, 17787.0660763 , 21167.90946908, 21608.22877658,
       23377.6893536 , 19872.17490191, 18692.04434253, 22655.74082127,
       21912.50600348, 22725.32310877, 19804.53585421, 18357.9789815 ,
       23249.98583002, 25004.85153514, 18032.02369521, 21863.10097661,
       17445.92104524, 19668.93287322, 19681.38358131, 20082.07070729,
       18305.0025366 , 17761.01223332, 21820.40745655, 18108.42536323,
       18175.20896045, 17362.15356639, 24387.86173984, 19435.90429612,
       24245.0809661 , 19374.83440161, 19458.60849163, 23382.90438524,
       22121.75529685, 22547.42758231, 19320.72115562, 22028.2292452 ,
       22016.78873144, 22329.76854184, 21375.97838127, 20049.12456537,
       18161.98839487, 24232.15787221, 19726.11332956, 17836.81416658,
       18310.58544813, 22648.51717776, 17597.74058399, 20827.44248991,
       21857.76651783, 21575.36034296, 18793.20559945, 24773.16815695,
       18606.78465942, 21713.56938899, 19507.19208855, 18165.03595368,
       22444.8180554 , 21639.43632782, 19371.23795188, 24186.08679625])
```

In [25]:
```
# Comparision of Actual Data and predicted values
pd.DataFrame(data={"Actual":ytest,"Pedicted":ypred})
```

Out[25]:

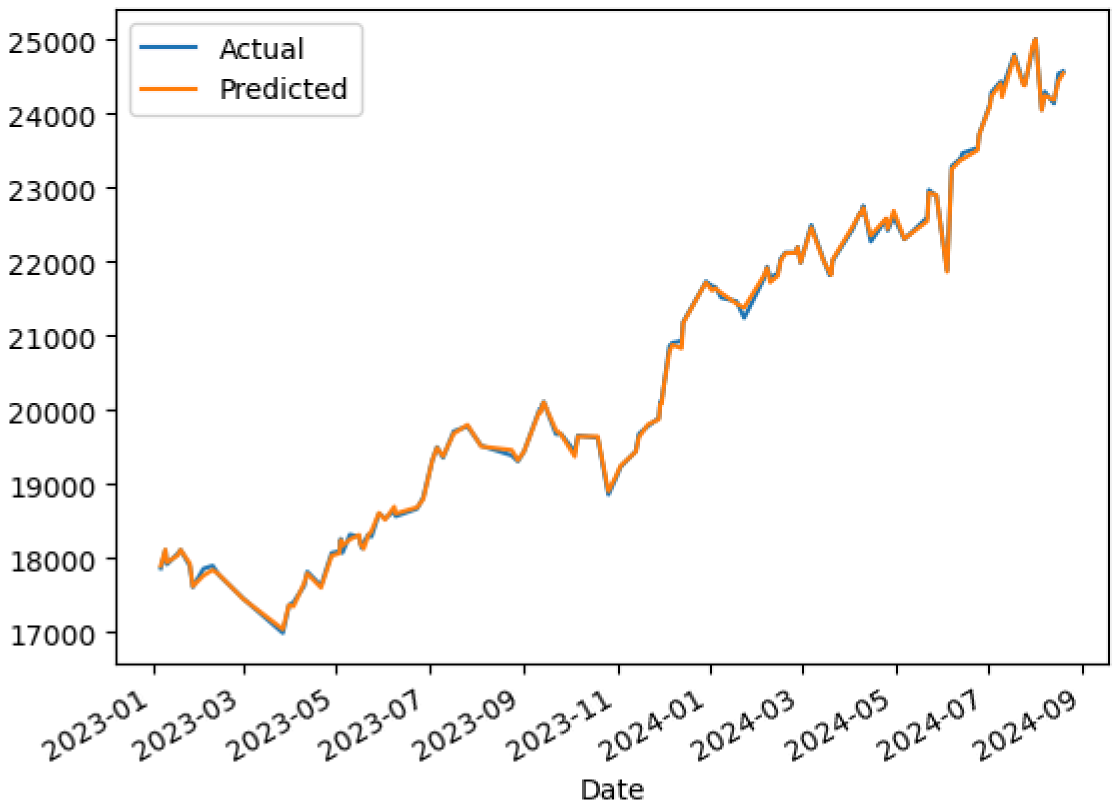
Date	Actual	Pedicted
2024-08-19	24572.650391	24542.428841
2023-07-06	19497.300781	19486.733680
2024-05-07	22302.500000	22307.449634
2024-05-23	22967.650391	22928.943997
2023-09-12	19993.199219	19966.519255
...	...	...
2023-05-18	18129.949219	18165.035954
2024-04-26	22419.949219	22444.818055
2024-01-04	21658.599609	21639.436328
2023-10-04	19436.099609	19371.237952
2024-08-13	24139.000000	24186.086796

120 rows × 2 columns

In [26]:

```
data = {"Actual":ytest,"Predicted":ypred}
data = pd.DataFrame(data)
data.plot(kind="line")
# diff between actual and predicted values
```

Out[26]: <Axes: xlabel='Date'>



In [27]:
```
# checking accuracy of the model

from sklearn.metrics import r2_score
print("Accuracy of the linear Regression Model",r2_score(ytest,ypred))
```

```
Accuracy of the linear Regression Model 0.999654602777578
```

In [28]:
```
#actual values testing
open = float(input("Enter Open Price:")) #24742
high = float(input("Enter High Price:"))#24752
low = float(input("Enter Low Price:"))#24390
```

In [29]:
```
lr.predict([[open,high,low]])
```

Out[29]:
```
array([24007.62576909])
```

In [30]:
```
!pip install mplfinance
```

```
Requirement already satisfied: mplfinance in c:\users\rohan-rd\anaconda3\lib\site
-packages (0.12.10b0)
Requirement already satisfied: matplotlib in c:\users\rohan-rd\anaconda3\lib\site
-packages (from mplfinance) (3.8.0)
Requirement already satisfied: pandas in c:\users\rohan-rd\anaconda3\lib\site-pac
kages (from mplfinance) (2.1.4)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\rohan-rd\anaconda3\li
b\site-packages (from matplotlib->mplfinance) (1.2.0)
Requirement already satisfied: cycler>=0.10 in c:\users\rohan-rd\anaconda3\lib\si
te-packages (from matplotlib->mplfinance) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\rohan-rd\anaconda3\l
ib\site-packages (from matplotlib->mplfinance) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\rohan-rd\anaconda3\l
ib\site-packages (from matplotlib->mplfinance) (1.4.4)
Requirement already satisfied: numpy<2,>=1.21 in c:\users\rohan-rd\anaconda3\lib
\site-packages (from matplotlib->mplfinance) (1.26.4)
Requirement already satisfied: packaging>=20.0 in c:\users\rohan-rd\anaconda3\lib
\site-packages (from matplotlib->mplfinance) (23.1)
Requirement already satisfied: pillow>=6.2.0 in c:\users\rohan-rd\anaconda3\lib\s
ite-packages (from matplotlib->mplfinance) (10.2.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\rohan-rd\anaconda3\li
b\site-packages (from matplotlib->mplfinance) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\rohan-rd\anaconda
3\lib\site-packages (from matplotlib->mplfinance) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\rohan-rd\anaconda3\lib\si
te-packages (from pandas->mplfinance) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in c:\users\rohan-rd\anaconda3\lib
\site-packages (from pandas->mplfinance) (2023.3)
Requirement already satisfied: six>=1.5 in c:\users\rohan-rd\anaconda3\lib\site-p
ackages (from python-dateutil>=2.7->matplotlib->mplfinance) (1.16.0)
```

In [31]:
```
# mplfinance is ploting library can be used for ploting candle charts in python
import mplfinance as mpf
```

In [32]:
```
df2 = pd.read_excel('NIFTY.xlsx')
```

In [33]:
```
df2
```

Out[33]:

	Date	Open	High	Low	Close	Adj Close	Volu
0	2023-07-17	19612.150391	19731.849609	19562.949219	19711.449219	19711.449219	268
1	2023-07-18	19787.500000	19819.449219	19690.199219	19749.250000	19749.250000	286
2	2023-07-19	19802.949219	19851.699219	19727.449219	19833.150391	19833.150391	259
3	2023-07-20	19831.699219	19991.849609	19758.400391	19979.150391	19979.150391	274
4	2023-07-21	19800.449219	19887.400391	19700.000000	19745.000000	19745.000000	312
...	...	...	...	...	...	...	
240	2024-07-10	24459.849609	24461.050781	24141.800781	24324.449219	24324.449219	292
241	2024-07-11	24396.550781	24402.650391	24193.750000	24315.949219	24315.949219	306
242	2024-07-12	24387.949219	24592.199219	24331.150391	24502.150391	24502.150391	325
243	2024-07-15	24587.599609	24635.050781	24522.750000	24586.699219	24586.699219	305
244	2024-07-16	24615.900391	24661.250000	24587.650391	24613.000000	24613.000000	

245 rows × 7 columns

In [34]:

```
# Nifty Price in Candlestick Data
df2["Date"] = pd.to_datetime(df2["Date"])
df2.set_index('Date',inplace=True)
mpf.plot(df2, type='candle', style='charles', title='Stock Prices', ylabel='Pric
```

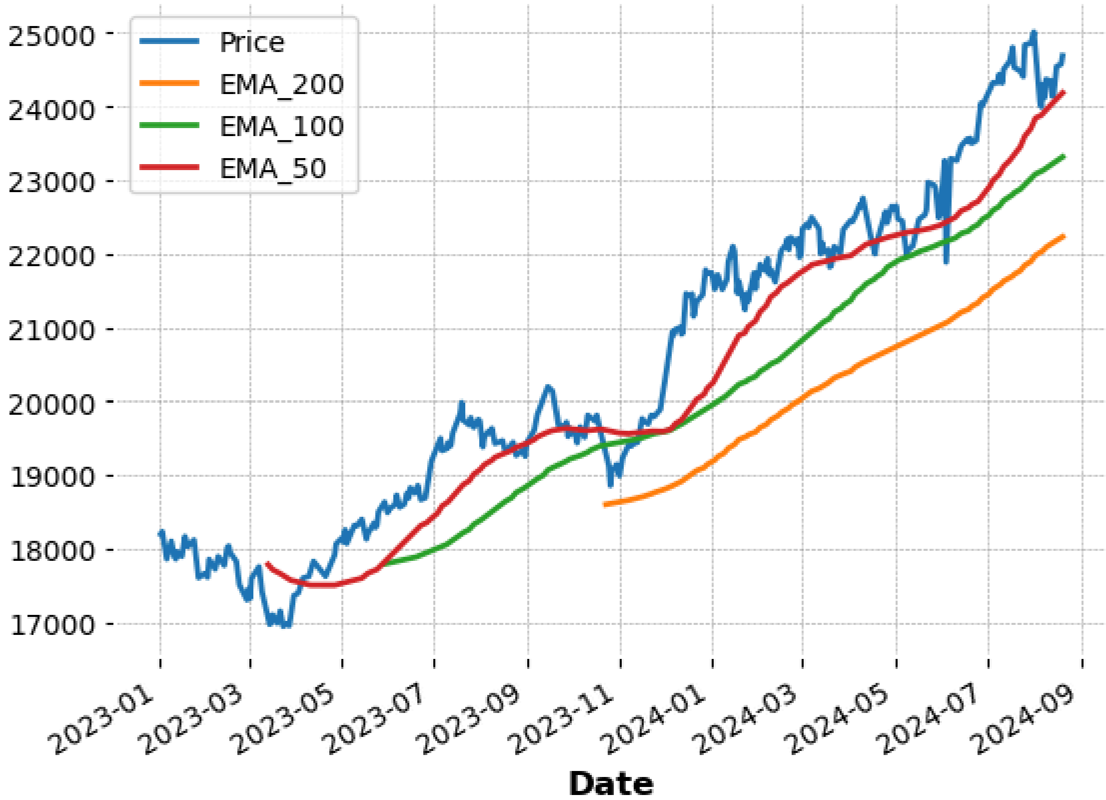
In [35]:
```
df2.columns
```

Out[35]: Index(['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'], dtype='object')

In [62]:
```
df["200_EMA"] = df["Close"].rolling(window=200).mean()
df["100_EMA"] = df["Close"].rolling(window=100).mean()
df["50_EMA"] = df["Close"].rolling(window=50).mean()
```

In [63]:
```
ema = {"Price":df["Close"],"EMA_200":df["200_EMA"],"EMA_100":df["100_EMA"],"EMA_
ema = pd.DataFrame(ema)
ema.plot(kind="line")
# Closing price with 200 EMA
```

Out[63]: <Axes: xlabel='Date'>


Price
EMA_200
EMA_100
EMA_50
25000
24000
23000
22000
21000
20000
19000
18000
17000
2023-01
2023-03
2023-05
2023-07
2023-09
2023-11
2024-01
2024-03
2024-05
2024-07
2024-09
Date

In [ ]: