Share Market Analysis

Project For Data Analytics And Visualisation

Jeet Mehta & Akash Gupta

This project is focused on analysing 3 stocks: Tesla, Ford and GM to take decisions based on the historical data and analysing their data. The focus is on analysis of stocks on various techinical indicators that stock market analysts use such as moving averages, volume traded, amount of stock traded and using this information to make their decisions.

In [35]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import yfinance as yf
from pandas_datareader import data as pdr
yf.pdr_override()
# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input directory

import os

# Any results you write to the current directory are saved as output.
```

In [36]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

In [37]:
```
# import pandas_datareader
import datetime
# import pandas_datareader.data as web
```

In [38]:
```
start = datetime.datetime(2012, 1, 1)
end = datetime.datetime(2023, 1, 1)
tesla = pdr.DataReader("TSLA", start, end)
ford = pdr.DataReader("F", start, end)
gm = pdr.DataReader("GM", start, end)
```

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

Question 1

Show the headers of the Dataframes

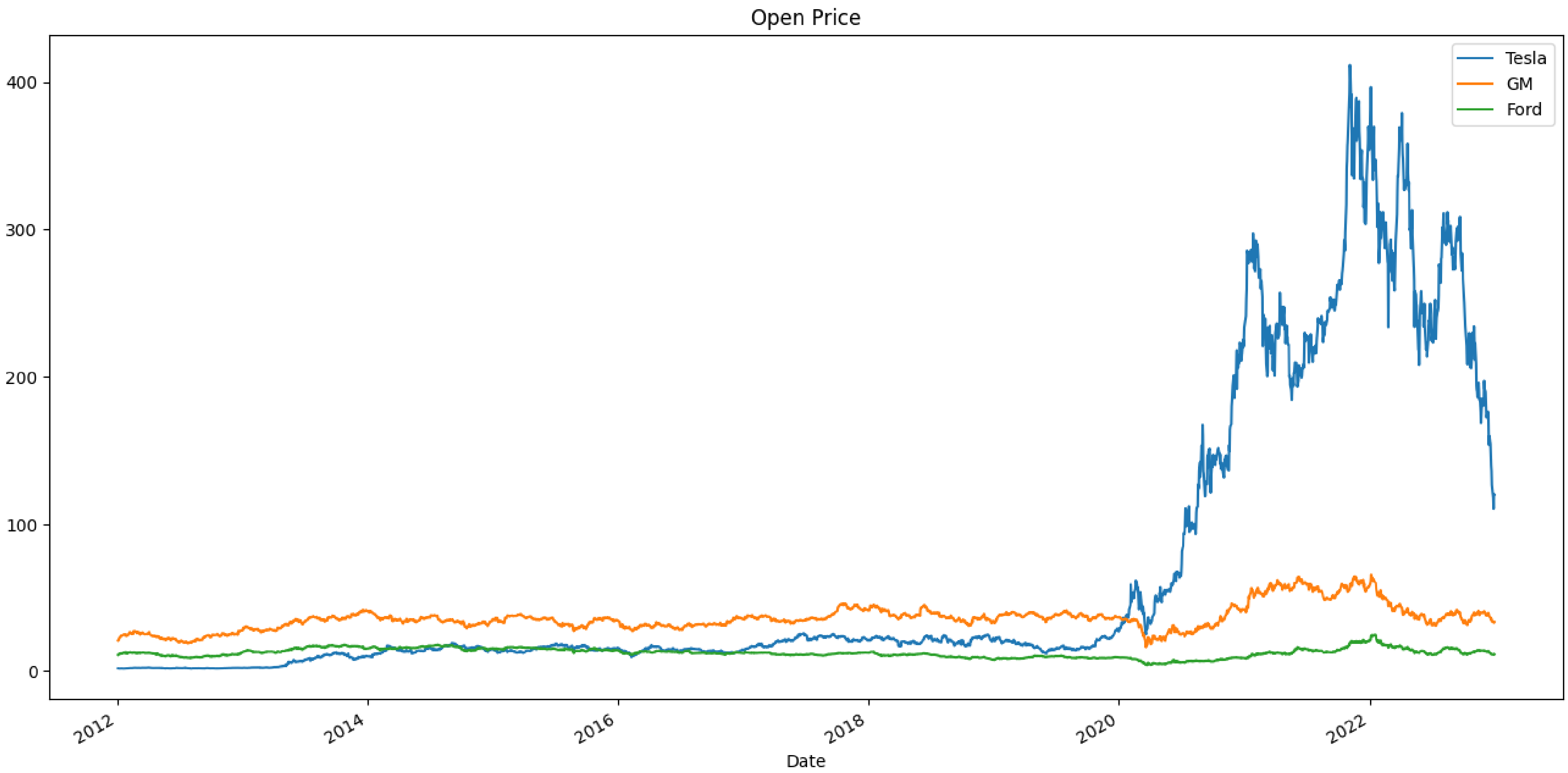
In [39]:
```
# Let's See the data top columns

print("TESLA head\n")
print(tesla.head())
print("\nGM head\n")
print(gm.head())
print("\nFORD head\n")
print(ford.head())
```

```
TESLA head

                Open      High       Low     Close  Adj Close    Volume
Date
2012-01-03  1.929333  1.966667  1.843333  1.872000   1.872000  13921500
2012-01-04  1.880667  1.911333  1.833333  1.847333   1.847333   9451500
2012-01-05  1.850667  1.862000  1.790000  1.808000   1.808000  15082500
2012-01-06  1.813333  1.852667  1.760667  1.794000   1.794000  14794500
2012-01-09  1.800000  1.832667  1.741333  1.816667   1.816667  13455000

GM head

                 Open       High        Low      Close  Adj Close    Volume
Date
2012-01-03  20.830000  21.180000  20.750000  21.049999  16.196335   9321300
2012-01-04  21.049999  21.370001  20.750000  21.150000  16.273277   7856700
2012-01-05  21.100000  22.290001  20.959999  22.170000  17.058088  17880600
2012-01-06  22.260000  23.030001  22.240000  22.920000  17.635155  18234500
2012-01-09  23.200001  23.430000  22.700001  22.840000  17.573601  12084500

FORD head

             Open   High    Low  Close  Adj Close    Volume
Date
2012-01-03  11.00  11.25  10.99  11.13   7.269921  45709900
2012-01-04  11.15  11.53  11.07  11.30   7.380964  79725200
2012-01-05  11.33  11.63  11.24  11.59   7.570387  67877500
2012-01-06  11.74  11.80  11.52  11.71   7.648771  59840700
2012-01-09  11.83  11.95  11.70  11.80   7.707555  53981500
```

Question 2

Plot the Opening Prices of the stocks

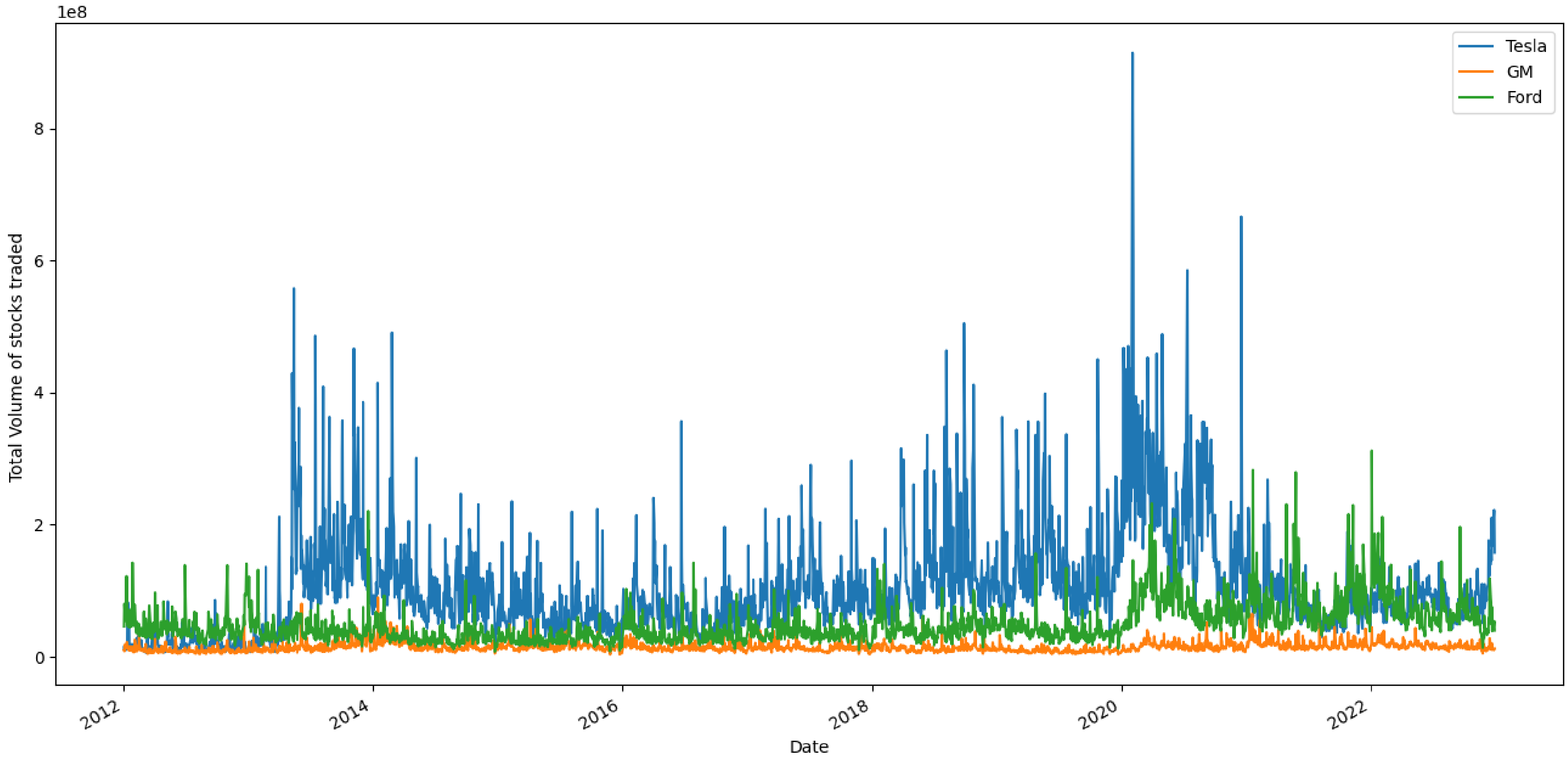
In [40]:
```
tesla['Open'].plot(label='Tesla',figsize=(16,8),title='Open Price')
gm['Open'].plot(label='GM')
ford['Open'].plot(label='Ford')
plt.legend()
plt.show()
```



Question 3

Plot the Volume of the stocks traded to analyse stock's volatility and it's day to day trading

In [41]:
```
tesla['Volume'].plot(label='Tesla',figsize=(16,8))
gm['Volume'].plot(label='GM',figsize=(16,8))
ford['Volume'].plot(label='Ford',figsize=(16,8))
plt.legend()
plt.ylabel('Total Volume of stocks traded ')
plt.show()
```



Question 4

Find out the date when the traded volume of the stocks was highest of all time.

In [42]:
```
max_vol_i=ford['Volume'].argmax()
print("Ford Highest Volume ")
```

```
print(ford.iloc[max_vol_i])
print("\nTesla Highest Volume ")
max_vol_i=tesla['Volume'].argmax()
print(tesla.iloc[max_vol_i])
print("\nGM Highest Volume ")
max_vol_i=gm['Volume'].argmax()
print(gm.iloc[max_vol_i])
```

```
Ford Highest Volume
Open         2.252000e+01
High         2.456000e+01
Low          2.242000e+01
Close        2.431000e+01
Adj Close    2.354246e+01
Volume       3.116452e+08
Name: 2022-01-04 00:00:00, dtype: float64

Tesla Highest Volume
Open         5.886400e+01
High         6.459933e+01
Low          5.559200e+01
Close        5.913733e+01
Adj Close    5.913733e+01
Volume       9.140820e+08
Name: 2020-02-04 00:00:00, dtype: float64

GM Highest Volume
Open         3.963000e+01
High         3.977000e+01
Low          3.896000e+01
Close        3.938000e+01
Adj Close    3.029984e+01
Volume       8.920760e+07
Name: 2014-01-15 00:00:00, dtype: float64
```

Tesla, GM and Ford observed highest trading at their highest volumes on 4th Feb 2020, 15th Jan 2014 and 4th Jan 2022 respectively.

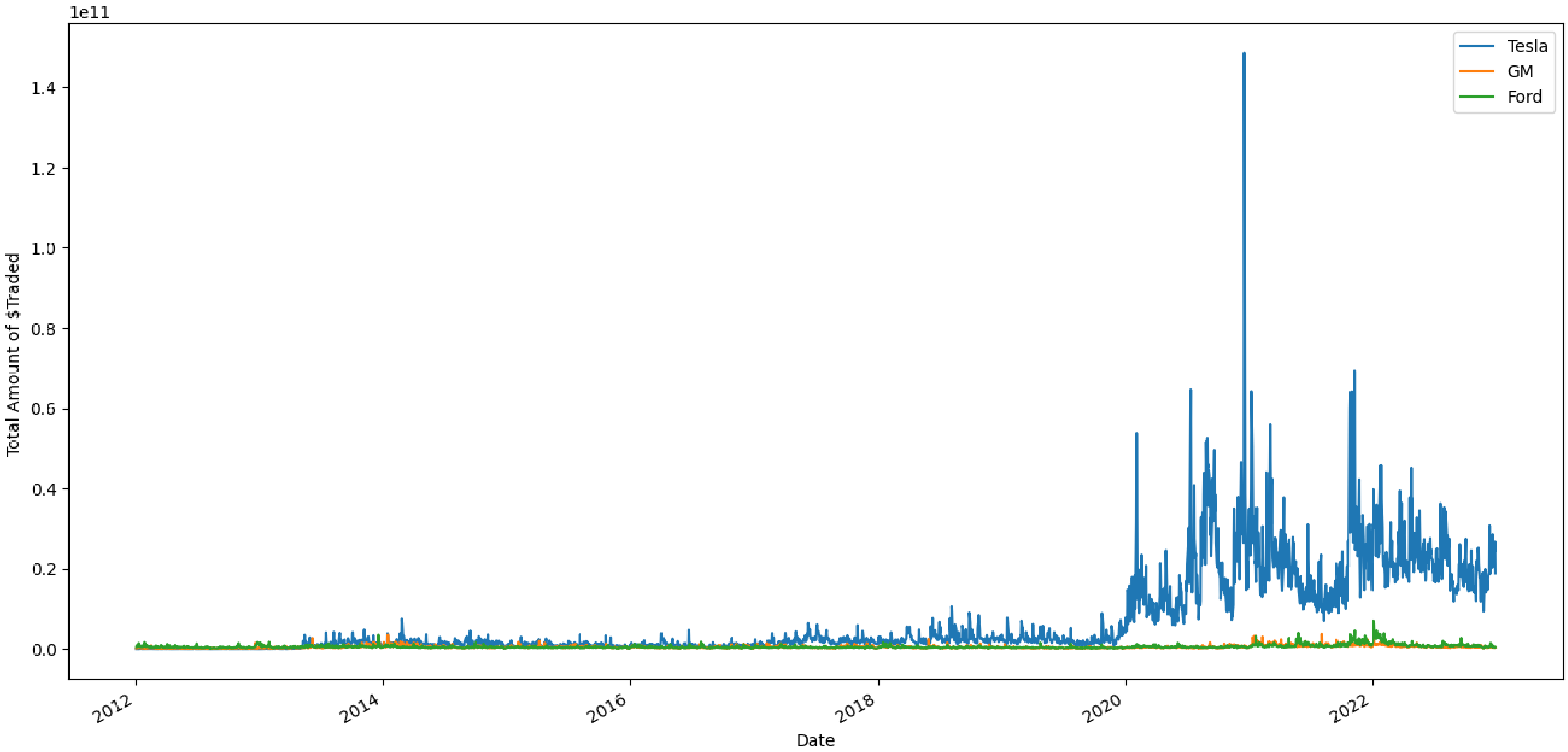
Question 5

Now Plot how much a stock traded on a particular day in terms of Dollars

In [43]:

```
tesla['Total Traded'] = tesla['Open']*tesla['Volume']
ford['Total Traded'] = ford['Open']*ford['Volume']
gm['Total Traded'] = gm['Open']*gm['Volume']
tesla['Total Traded'].plot(label='Tesla',figsize=(16,8))
gm['Total Traded'].plot(label='GM',figsize=(16,8))
ford['Total Traded'].plot(label='Ford',figsize=(16,8))
plt.legend()
```

```
plt.ylabel('Total Amount of $Traded ')
plt.show()
```



Question 6

Looks like there was huge amount of money traded for Tesla somewhere in early 2021 and recent years. What date was that?

In [44]:
```
#Interesting, looks like there was huge amount of money traded for Tesla somewhere in early 2021 and recent years. What date was that and what happened?
tesla_max_i=tesla['Total Traded'].argmax()
print(tesla.iloc[tesla_max_i])
```

```
Open            2.229667e+02
High            2.316667e+02
Low             2.095133e+02
Close           2.316667e+02
Adj Close       2.316667e+02
Volume          6.663786e+08
Total Traded    1.485802e+11
Name: 2020-12-18 00:00:00, dtype: float64
```

The most amount of money traded for Tesla was on 18th Decemeber 2020. The reason was Tesla inclusion in S&P 500 index which invests in 500 largest companies of the United States in terms of market cap, so this made the demand of the stock very high on that day.

Question 7

Draw a Conclusion using the plots above.

The plotting shows Trading of Tesla stock far exceeds the trading of Ford and GM stock after year 2020, also the plotting of Opening Prices of the stocks shows that Tesla began trading at much higher prices than Ford and GM after 2020, so the trading volume increase is justified

Question 8

Give a brief on the stock's data

In [45]:
```
print("A brief about FORD stock\n")
print(ford.describe())
print("\nA brief about TESLA stock\n")
print(tesla.describe())
print("\nA brief about GM stock\n")
print(gm.describe())
```

A brief about FORD stock

	Open	High	Low	Close	Adj Close
count	2768.000000	2768.000000	2768.000000	2768.000000	2768.000000
mean	12.498981	12.637691	12.343999	12.490972	10.119645
std	3.180500	3.216234	3.142602	3.181092	2.720162
min	4.270000	4.420000	3.960000	4.010000	3.863936
25%	10.270000	10.397500	10.150000	10.267500	8.537114
50%	12.415000	12.545000	12.280000	12.420000	9.858241
75%	14.900000	15.040000	14.712500	14.872500	11.285801
max	24.870001	25.870001	24.370001	25.190001	24.394676

	Volume	Total Traded
count	2.768000e+03	2.768000e+03
mean	4.801780e+07	5.939340e+08
std	2.913197e+07	4.604296e+08
min	7.128800e+06	1.095697e+08
25%	2.918260e+07	3.556582e+08
50%	4.005815e+07	4.715915e+08
75%	5.734472e+07	6.557393e+08
max	3.116452e+08	7.018250e+09

A brief about TESLA stock

	Open	High	Low	Close	Adj Close
count	2768.000000	2768.000000	2768.000000	2768.000000	2768.000000
mean	66.747593	68.239869	65.095015	66.687498	66.687498
std	99.502320	101.789751	96.912802	99.362267	99.362267
min	1.774667	1.790000	1.509333	1.519333	1.519333
25%	13.480667	13.731500	13.268167	13.505834	13.505834
50%	17.408334	17.640333	17.055667	17.396334	17.396334
75%	49.458668	51.806001	47.773333	49.939668	49.939668
max	411.470001	414.496674	405.666656	409.970001	409.970001

	Volume	Total Traded
count	2.768000e+03	2.768000e+03
mean	1.036379e+08	6.786365e+09
std	8.167655e+07	1.041519e+10
min	5.473500e+06	1.208549e+07
25%	5.557402e+07	8.884526e+08
50%	8.391190e+07	1.771661e+09
75%	1.247370e+08	9.596304e+09
max	9.140820e+08	1.485802e+11

A brief about GM stock

	Open	High	Low	Close	Adj Close
count	2768.000000	2768.000000	2768.000000	2768.000000	2768.000000
mean	36.097121	36.523584	35.629155	36.076517	32.333809
std	8.647418	8.749308	8.514919	8.631921	10.067612
min	16.340000	18.559999	14.330000	16.799999	14.465135
25%	31.330000	31.750000	30.950001	31.385000	26.207603
50%	35.344999	35.735001	34.959999	35.320000	30.204569
75%	38.712500	39.042500	38.237499	38.650002	36.638216
max	65.519997	67.209999	62.689999	65.739998	65.444710

```
             Volume  Total Traded
count  2.768000e+03  2.768000e+03
mean   1.427922e+07  5.299604e+08
std    7.684780e+06  3.570277e+08
min    2.899300e+06  8.002390e+07
25%    9.374025e+06  3.114644e+08
50%    1.251010e+07  4.318621e+08
75%    1.680605e+07  6.311067e+08
max    8.920760e+07  3.737259e+09
```

Question 9

Give information to summarise the dataframes used.

In [46]:
```
# information on all those stocks data
print("Ford info")
print(ford.info())

print("\nTesla info\n")
print(tesla.info())


print("\GM info\n")
print(gm.info())
```

```
Ford info
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2768 entries, 2012-01-03 to 2022-12-30
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Open          2768 non-null   float64
 1   High          2768 non-null   float64
 2   Low           2768 non-null   float64
 3   Close         2768 non-null   float64
 4   Adj Close     2768 non-null   float64
 5   Volume        2768 non-null   int64  
 6   Total Traded  2768 non-null   float64
dtypes: float64(6), int64(1)
memory usage: 173.0 KB
None

Tesla info

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2768 entries, 2012-01-03 to 2022-12-30
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Open          2768 non-null   float64
 1   High          2768 non-null   float64
 2   Low           2768 non-null   float64
 3   Close         2768 non-null   float64
 4   Adj Close     2768 non-null   float64
 5   Volume        2768 non-null   int64  
 6   Total Traded  2768 non-null   float64
dtypes: float64(6), int64(1)
memory usage: 173.0 KB
None
\GM info

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2768 entries, 2012-01-03 to 2022-12-30
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Open          2768 non-null   float64
 1   High          2768 non-null   float64
 2   Low           2768 non-null   float64
 3   Close         2768 non-null   float64
 4   Adj Close     2768 non-null   float64
 5   Volume        2768 non-null   int64  
 6   Total Traded  2768 non-null   float64
dtypes: float64(6), int64(1)
memory usage: 173.0 KB
None
```

Question 10

What Does a Moving Average Indicate?

A moving average is a statistic that captures the average change in a data series over time. In finance, moving averages are often used by technical analysts to keep track of price trends for specific securities. An upward trend in a moving average might signify an upswing in the price or momentum of a security, while a downward trend would be seen as a sign of decline

Question 11

Calculate 50 days Moving Averages.

In [47]:

```
# Let's Compare 50 days moving averages
gm['gm_MA50'] = gm['Open'].rolling(50).mean()
tesla['tesla_MA50'] = tesla['Open'].rolling(50).mean()
ford['ford_MA50'] = ford['Open'].rolling(50).mean()

print("Ford Date wise 50 days Moving Average")
print(ford[50:].head()['ford_MA50'])

print("\nTesla Date wise 50 days Moving Average\n")
print(ford[50:].head()['ford_MA50'])

print("\nGM Date wise 50 days Moving Average\n")
print(ford[50:].head()['ford_MA50'])
```

```
Ford Date wise 50 days Moving Average
Date
2012-03-15    12.3980
2012-03-16    12.4322
2012-03-19    12.4560
2012-03-20    12.4708
2012-03-21    12.4860
Name: ford_MA50, dtype: float64

Tesla Date wise 50 days Moving Average

Date
2012-03-15    12.3980
2012-03-16    12.4322
2012-03-19    12.4560
2012-03-20    12.4708
2012-03-21    12.4860
Name: ford_MA50, dtype: float64

GM Date wise 50 days Moving Average

Date
2012-03-15    12.3980
2012-03-16    12.4322
2012-03-19    12.4560
2012-03-20    12.4708
2012-03-21    12.4860
Name: ford_MA50, dtype: float64
```

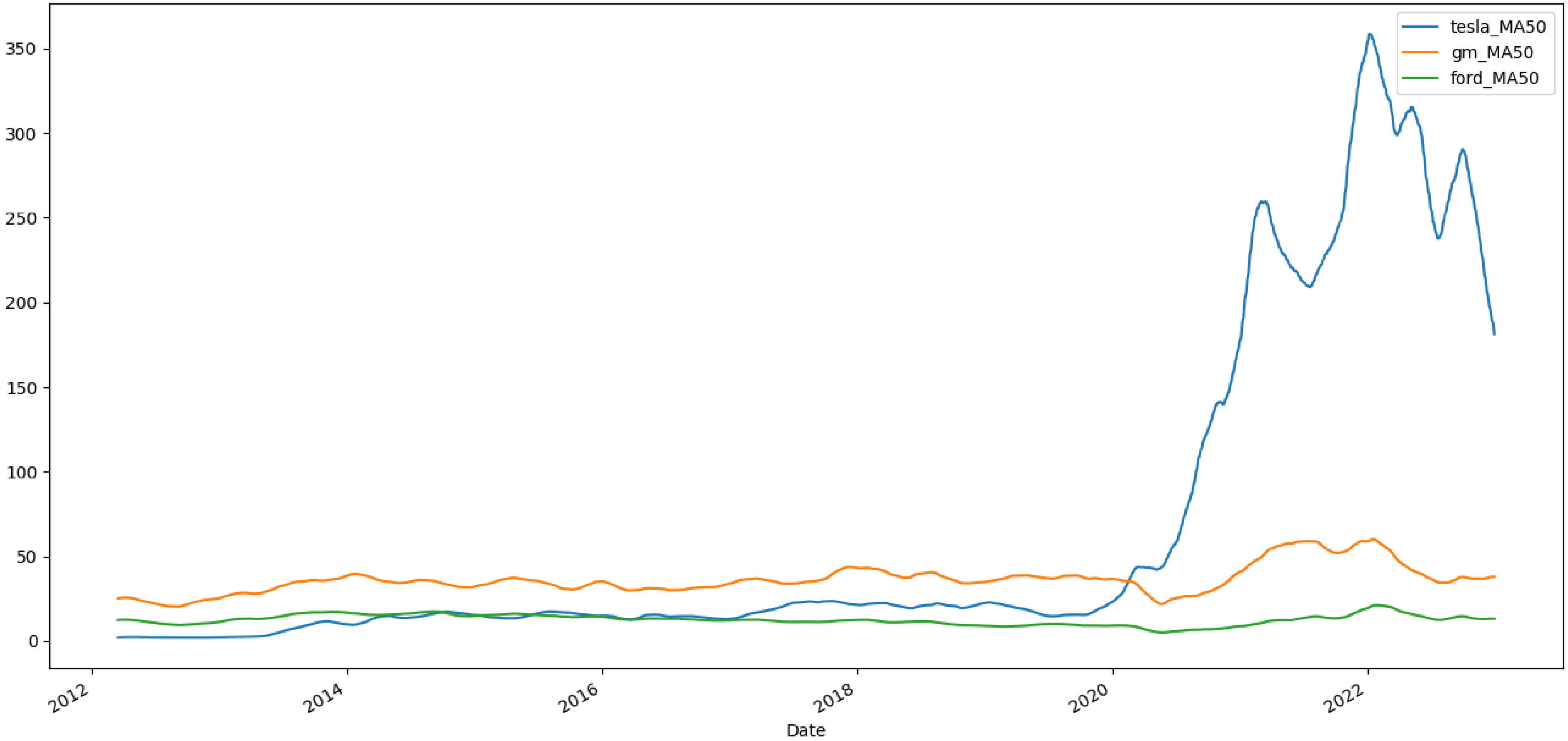
Computed 50 Days Moving Average of the Stocks.

Question 12

Now Plot 50 days moving average of stocks to analyse and compare them

In [48]:

```
MA50=gm.copy()
MA50.drop(['Open', 'Close','High','Low','Volume','Total Traded','Adj Close'], axis = 1, inplace = True)
MA50['tesla_MA50']=tesla['Open'].rolling(50).mean()
MA50['ford_MA50']=ford['Open'].rolling(50).mean()
MA50=MA50[50:]
MA50[['tesla_MA50','gm_MA50','ford_MA50']].plot(label='Moving Average',figsize=(16,8));
```



Question 13

Calculate 100 days Moving Averages.

In [49]:
```
gm['gm_MA100'] = gm['Open'].rolling(100).mean()
tesla['tesla_MA100'] = tesla['Open'].rolling(100).mean()
ford['ford_MA100'] = ford['Open'].rolling(100).mean()
```

In [50]:
```
MA100=gm.copy()
MA100.drop(['Open', 'Close','High','Low','Volume','Total Traded','Adj Close'], axis = 1, inplace = True)
MA100['tesla_MA100']=tesla['Open'].rolling(100).mean()
MA100['ford_MA100']=ford['Open'].rolling(100).mean()
MA100=MA100[100:]
```

```
print("Ford Date wise 50 days Moving Average")
print(ford[100:].head()['ford_MA100'])

print("\nTesla Date wise 50 days Moving Average\n")
print(ford[100:].head()['ford_MA100'])

print("\nGM Date wise 50 days Moving Average\n")
print(ford[100:].head()['ford_MA100'])
```

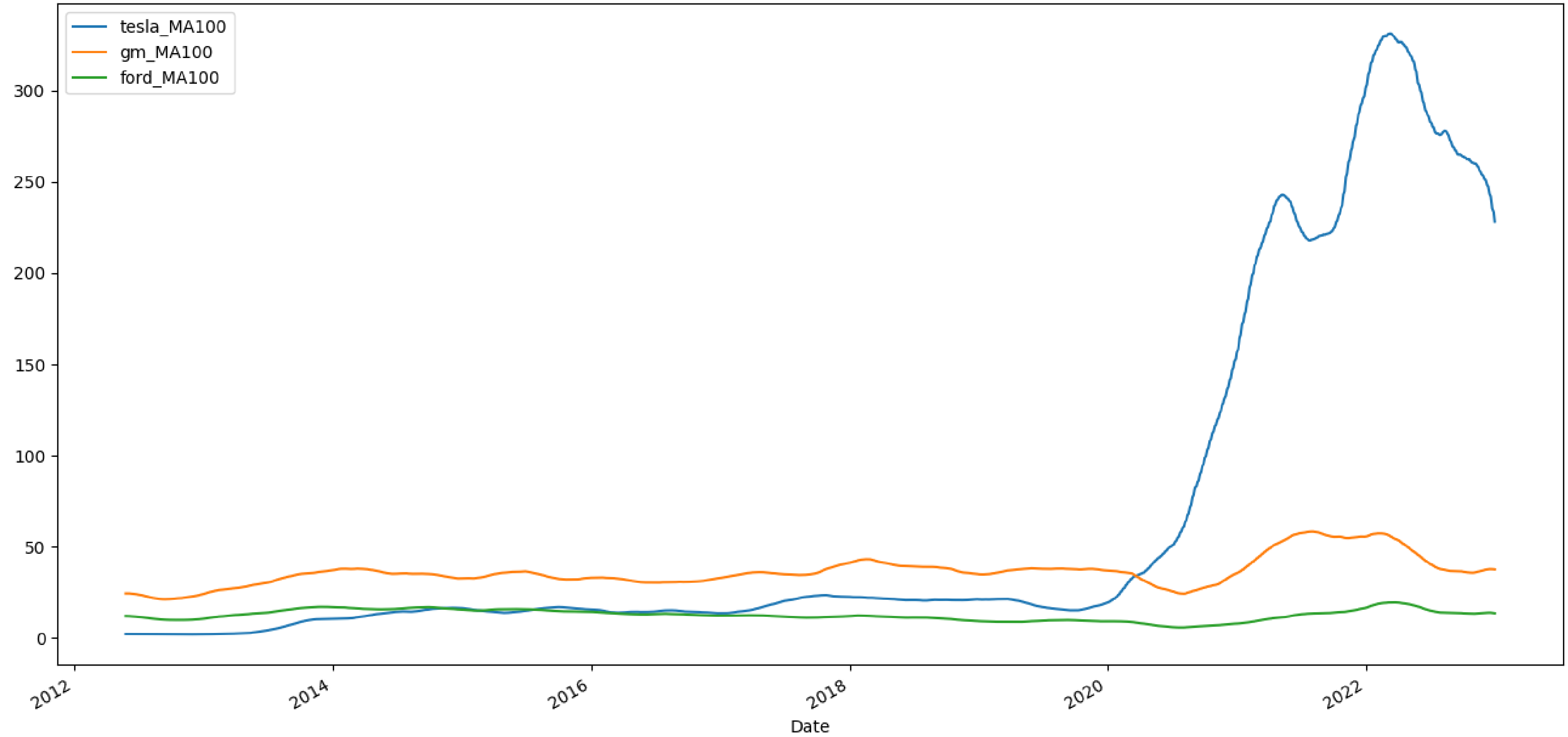
```
Ford Date wise 50 days Moving Average
Date
2012-05-25    11.9684
2012-05-29    11.9638
2012-05-30    11.9578
2012-05-31    11.9466
2012-06-01    11.9316
Name: ford_MA100, dtype: float64

Tesla Date wise 50 days Moving Average

Date
2012-05-25    11.9684
2012-05-29    11.9638
2012-05-30    11.9578
2012-05-31    11.9466
2012-06-01    11.9316
Name: ford_MA100, dtype: float64

GM Date wise 50 days Moving Average

Date
2012-05-25    11.9684
2012-05-29    11.9638
2012-05-30    11.9578
2012-05-31    11.9466
2012-06-01    11.9316
Name: ford_MA100, dtype: float64
```

Computed 100 Days Moving Average of the Stocks.

Question 14

Now Plot 100 days moving average of stocks to analyse and compare them

In [51]:
```
MA100[['tesla_MA100','gm_MA100','ford_MA100']].plot(label='Moving Average',figsize=(16,8));
```



Question 15

Calculate 200 days Moving Averages.

In [52]:
```
gm['gm_MA200'] = gm['Open'].rolling(200).mean()
tesla['tesla_MA200'] = tesla['Open'].rolling(200).mean()
ford['ford_MA200'] = ford['Open'].rolling(200).mean()
```

In [53]:
```
MA200=gm.copy()
MA200.drop(['Open', 'Close','High','Low','Volume','Total Traded','Adj Close'], axis = 1, inplace = True)
MA200['tesla_MA200']=tesla['Open'].rolling(200).mean()
MA200['ford_MA200']=ford['Open'].rolling(200).mean()
MA200=MA200[200:]
```

```
print("Ford Date wise 200 days Moving Average")
print(ford[200:].head()['ford_MA100'])

print("\nTesla Date wise 200 days Moving Average\n")
print(ford[200:].head()['ford_MA100'])

print("\nGM Date wise 200 days Moving Average\n")
print(ford[200:].head()['ford_MA100'])
```

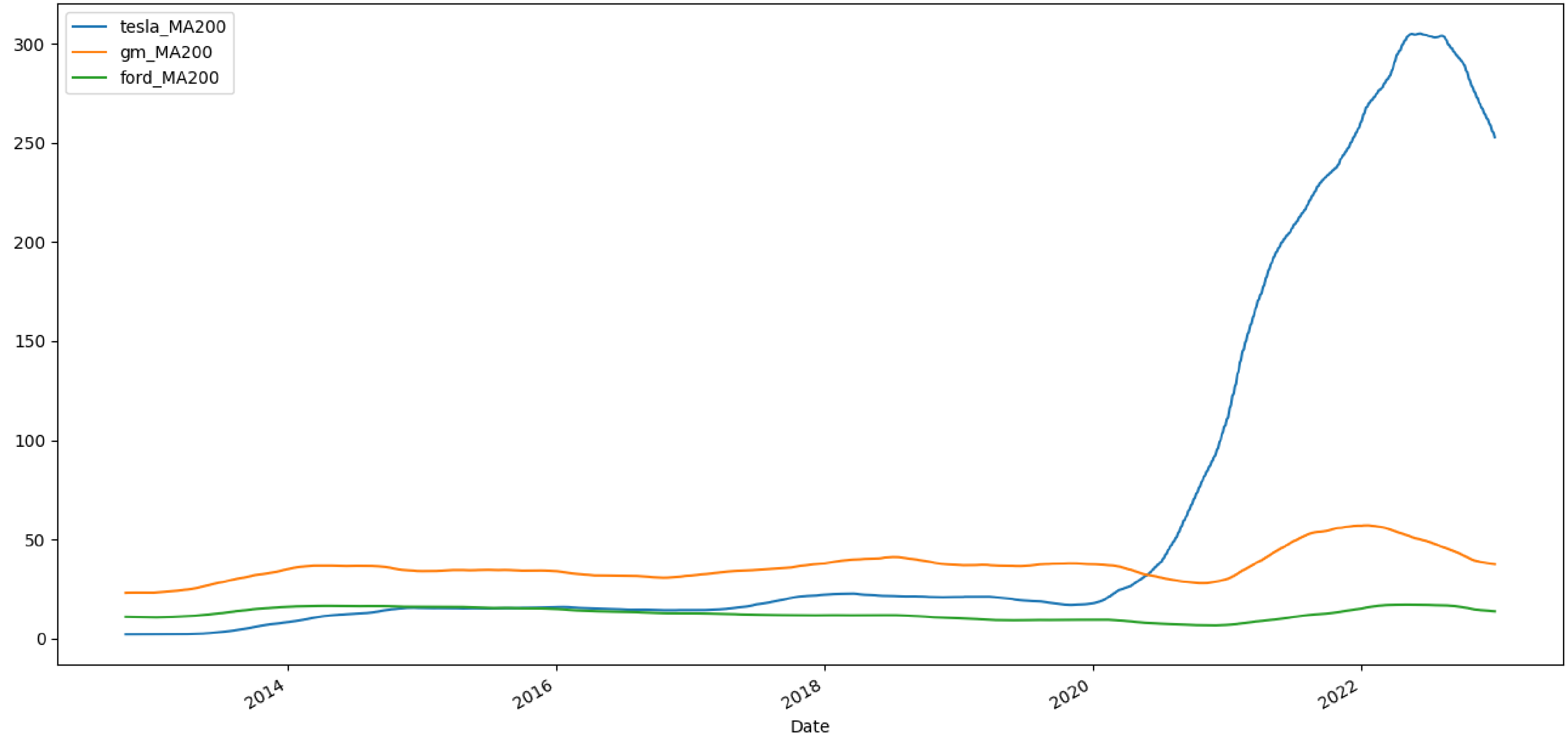
```
Ford Date wise 200 days Moving Average
Date
2012-10-17    9.8282
2012-10-18    9.8249
2012-10-19    9.8218
2012-10-22    9.8170
2012-10-23    9.8139
Name: ford_MA100, dtype: float64

Tesla Date wise 200 days Moving Average

Date
2012-10-17    9.8282
2012-10-18    9.8249
2012-10-19    9.8218
2012-10-22    9.8170
2012-10-23    9.8139
Name: ford_MA100, dtype: float64

GM Date wise 200 days Moving Average

Date
2012-10-17    9.8282
2012-10-18    9.8249
2012-10-19    9.8218
2012-10-22    9.8170
2012-10-23    9.8139
Name: ford_MA100, dtype: float64
```

Computed 200 Days Moving Average of the Stocks.

Question 16

Now Plot 200 days moving average of stocks to analyse and compare them

In [54]:
```
MA200[['tesla_MA200','gm_MA200','ford_MA200']].plot(label='Moving Average',figsize=(16,8));
```

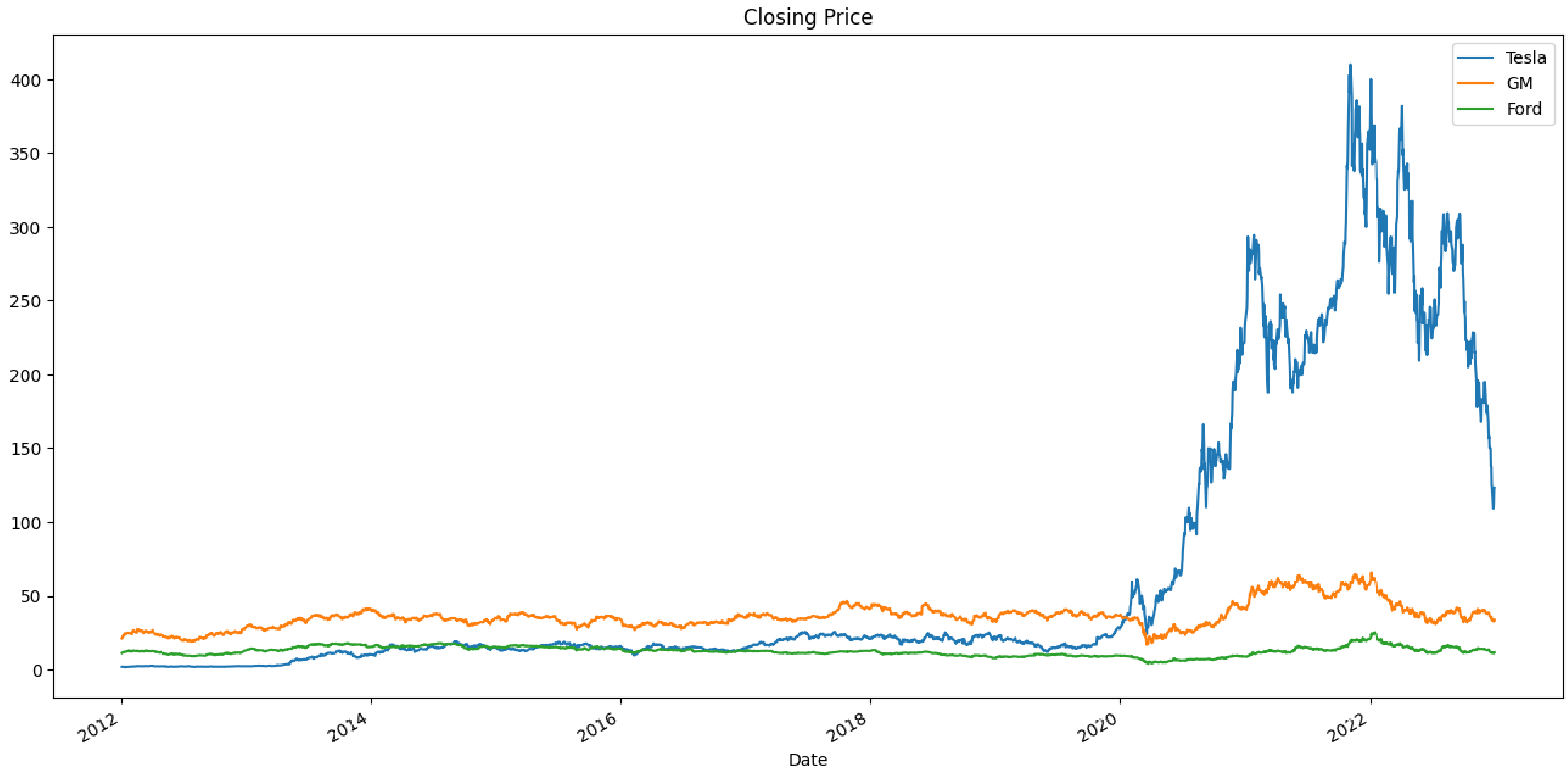


Question 17

Now Plot the Closing prices of stocks to analyse and compare them

In [55]:

```
#Lets see the closing prices to analyse them
tesla['Close'].plot(label='Tesla',figsize=(16,8),title='Closing Price')
gm['Close'].plot(label='GM')
ford['Close'].plot(label='Ford')
plt.legend()
plt.show()
```



Question 18

How a Stock's Moving Average is used for technical analysis?

When a short-term moving average (eg: 50MA) crosses over a major long-term moving average(eg: 200MA) to the upside then it is interpreted by analysts and traders as signaling a definitive upward turn in a market.

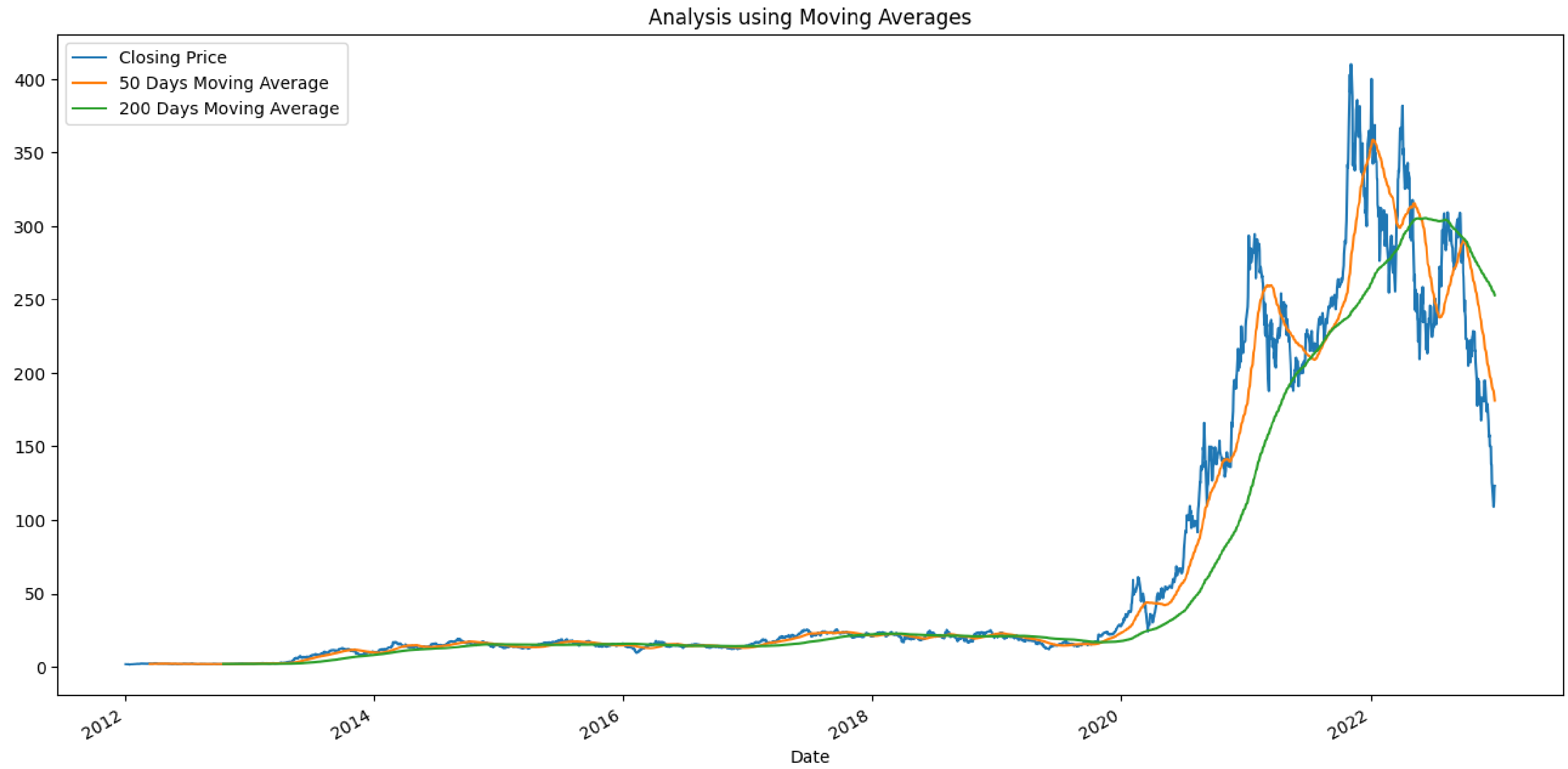
Similarly when a short-term moving average (eg: 50MA) crosses over a major long-term moving average(eg: 200MA) to the downside then it is interpreted by analysts and traders as signaling a definitive downward turn in a market.

* Here 50MA is 50 days Moving Average and 200MA is 200 Days Moving Average

Question 19

Compare Tesla Stock and it's 50 and 200 days moving average and Draw a conclusion.

In [56]:
```
tesla['Close'].plot(label='Closing Price',figsize=(16,8),title='Analysis using Moving Averages')
tesla['tesla_MA50'].plot(label='50 Days Moving Average')
tesla['tesla_MA200'].plot(label='200 Days Moving Average')
plt.legend()
plt.show()
```



Here as written in Answer of Q18, When a short term Moving Average crosses a long term Moving Average:

In upward direction then the stock will most probably increase, signifying upward trend.

In downward direction then the stock will most probably decrease, signifying downward trend.

Here It's seen that 50MA crosses 200 MA in upward direction in 2020, and at that period Tesla Stock starts increasing as seen above.

While when 50MA crosses 200MA in downward direction in 2022, the stock declines.

So I conclude that using moving averages as a technical indicator has been successful to predict a downward or upward trend of a Stock.

In [ ]:

```

```

In [19]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pandas_datareader as data
```

In [20]:
```
pip install pandas_datareader
```

```
Requirement already satisfied: pandas_datareader in c:\users\jeet\anaconda3\envs\ai\lib\site-packages (0.10.0)Note: you may need to restart the kernel
to use updated packages.

Requirement already satisfied: requests>=2.19.0 in c:\users\jeet\anaconda3\envs\ai\lib\site-packages (from pandas_datareader) (2.28.1)
Requirement already satisfied: lxml in c:\users\jeet\anaconda3\envs\ai\lib\site-packages (from pandas_datareader) (4.9.1)
Requirement already satisfied: pandas>=0.23 in c:\users\jeet\anaconda3\envs\ai\lib\site-packages (from pandas_datareader) (1.5.2)
Requirement already satisfied: numpy>=1.21.0 in c:\users\jeet\anaconda3\envs\ai\lib\site-packages (from pandas>=0.23->pandas_datareader) (1.23.5)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\jeet\anaconda3\envs\ai\lib\site-packages (from pandas>=0.23->pandas_datareader) (2.
8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\jeet\anaconda3\envs\ai\lib\site-packages (from pandas>=0.23->pandas_datareader) (2022.6)
Requirement already satisfied: idna<4,>=2.5 in c:\users\jeet\anaconda3\envs\ai\lib\site-packages (from requests>=2.19.0->pandas_datareader) (3.4)
Requirement already satisfied: charset-normalizer<3,>=2 in c:\users\jeet\anaconda3\envs\ai\lib\site-packages (from requests>=2.19.0->pandas_datareade
r) (2.1.1)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\jeet\anaconda3\envs\ai\lib\site-packages (from requests>=2.19.0->pandas_datareader) (202
2.9.24)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\jeet\anaconda3\envs\ai\lib\site-packages (from requests>=2.19.0->pandas_datareader)
(1.26.13)
Requirement already satisfied: six>=1.5 in c:\users\jeet\anaconda3\envs\ai\lib\site-packages (from python-dateutil>=2.8.1->pandas>=0.23->pandas_datare
ader) (1.16.0)
```

In [ ]:

In [21]:
```
from pandas_datareader import data as pdr
yf.pdr_override()
start='2010-01-01'
end='2019-12-31'
```

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

In [22]:
```
df.head()
```

Out[22]:

	Open	High	Low	Close	Adj Close	Volume
Date						
2010-06-29	1.266667	1.666667	1.169333	1.592667	1.592667	281494500
2010-06-30	1.719333	2.028000	1.553333	1.588667	1.588667	257806500
2010-07-01	1.666667	1.728000	1.351333	1.464000	1.464000	123282000
2010-07-02	1.533333	1.540000	1.247333	1.280000	1.280000	77097000
2010-07-06	1.333333	1.333333	1.055333	1.074000	1.074000	103003500

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

Out[23]:

Date	Open	High	Low	Close	Adj Close	Volume
2019-12-23	27.452000	28.134001	27.333332	27.948000	27.948000	199794000
2019-12-24	27.890667	28.364668	27.512667	28.350000	28.350000	120820500
2019-12-26	28.527332	28.898666	28.423332	28.729334	28.729334	159508500
2019-12-27	29.000000	29.020666	28.407333	28.691999	28.691999	149185500
2019-12-30	28.586000	28.600000	27.284000	27.646667	27.646667	188796000

In [24]:
```
df=df.reset_index()
```

Out[24]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2010-06-29	1.266667	1.666667	1.169333	1.592667	1.592667	281494500
1	2010-06-30	1.719333	2.028000	1.553333	1.588667	1.588667	257806500
2	2010-07-01	1.666667	1.728000	1.351333	1.464000	1.464000	123282000
3	2010-07-02	1.533333	1.540000	1.247333	1.280000	1.280000	77097000
4	2010-07-06	1.333333	1.333333	1.055333	1.074000	1.074000	103003500
...	...	...	...	...	...	...	...
2388	2019-12-23	27.452000	28.134001	27.333332	27.948000	27.948000	199794000
2389	2019-12-24	27.890667	28.364668	27.512667	28.350000	28.350000	120820500
2390	2019-12-26	28.527332	28.898666	28.423332	28.729334	28.729334	159508500
2391	2019-12-27	29.000000	29.020666	28.407333	28.691999	28.691999	149185500
2392	2019-12-30	28.586000	28.600000	27.284000	27.646667	27.646667	188796000

2393 rows × 7 columns

In [25]:
```
df=df.drop(['Date','Adj Close'],axis=1)
```

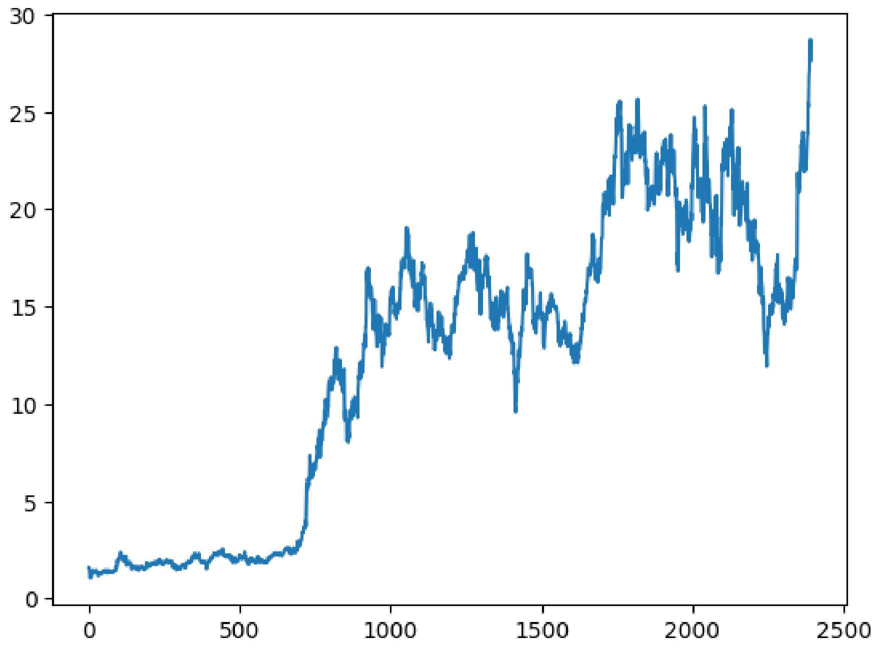
Out[25]:

	Open	High	Low	Close	Volume
0	1.266667	1.666667	1.169333	1.592667	281494500
1	1.719333	2.028000	1.553333	1.588667	257806500
2	1.666667	1.728000	1.351333	1.464000	123282000
3	1.533333	1.540000	1.247333	1.280000	77097000
4	1.333333	1.333333	1.055333	1.074000	103003500
...	...	...	...	...	...
2388	27.452000	28.134001	27.333332	27.948000	199794000
2389	27.890667	28.364668	27.512667	28.350000	120820500
2390	28.527332	28.898666	28.423332	28.729334	159508500
2391	29.000000	29.020666	28.407333	28.691999	149185500
2392	28.586000	28.600000	27.284000	27.646667	188796000

2393 rows × 5 columns

In [26]:
```
plt.plot(df.Close)
```

Out[26]: [<matplotlib.lines.Line2D at 0x20221c7cc10>]



In [27]:
```
df
```

Out[27]:

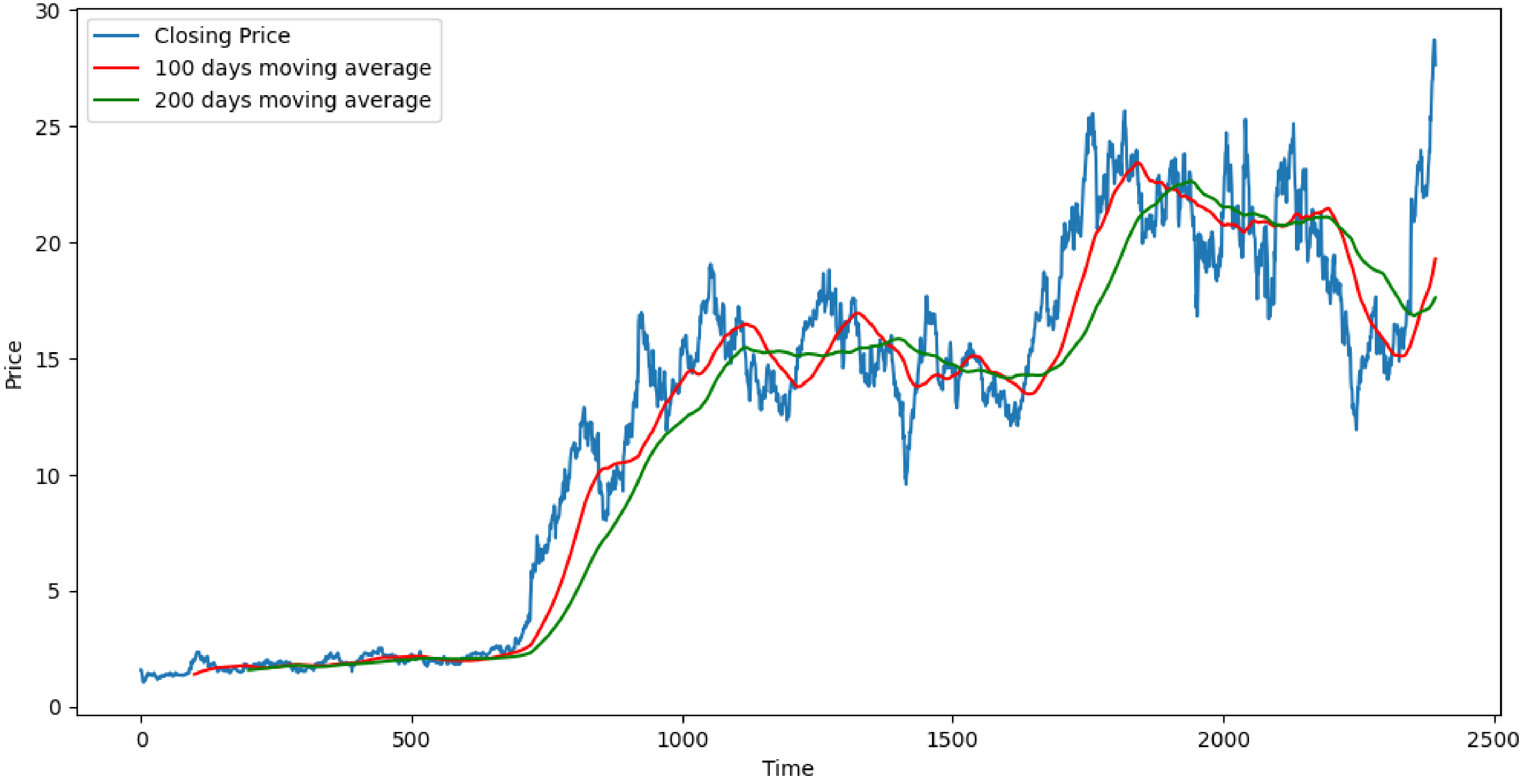
	Open	High	Low	Close	Volume
0	1.266667	1.666667	1.169333	1.592667	281494500
1	1.719333	2.028000	1.553333	1.588667	257806500
2	1.666667	1.728000	1.351333	1.464000	123282000
3	1.533333	1.540000	1.247333	1.280000	77097000
4	1.333333	1.333333	1.055333	1.074000	103003500
...	...	...	...	...	...
2388	27.452000	28.134001	27.333332	27.948000	199794000
2389	27.890667	28.364668	27.512667	28.350000	120820500
2390	28.527332	28.898666	28.423332	28.729334	159508500
2391	29.000000	29.020666	28.407333	28.691999	149185500
2392	28.586000	28.600000	27.284000	27.646667	188796000

2393 rows × 5 columns

In [28]:
```
#past 100/200 days moving average
ma100=df.Close.rolling(100).mean()
```

Type *Markdown* and LaTeX: α^2

In [59]:
```
plt.figure(figsize=(12,6))
plt.plot(df.Close,label='Closing Price')
plt.plot(ma100,'r',label='100 days moving average')
plt.plot(ma200,'g',label='200 days moving average')
plt.xlabel('Time')
plt.ylabel('Price')
plt.legend()
```



In [30]:
```
df.shape
```

Out[30]:
```
(2393, 5)
```

In [31]:
```
#splitting the data into Training and Testing
data_training=(df['Close'][0:int(len(df)*0.70)])
data_testing=(df['Close'][int(len(df)*0.70):int(len(df))])
data_training=data_training.to_frame()
data_testing=data_testing.to_frame()
```

```
(1675, 1)
(718, 1)
```

In [32]:
```
data_testing.head()
```

Out[32]:

	Close
1675	17.066000
1676	17.133333
1677	16.415333
1678	16.666000
1679	16.667999

In [33]:
```
data_training.head()
```

Out[33]:

	Close
0	1.592667
1	1.588667
2	1.464000
3	1.280000
4	1.074000

In [34]:

In [35]:
```
data_training_array=scaler.fit_transform(data_training)
```

Out[35]:
```
(1675, 1)
```

In [36]:
```
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])
```

In [37]:
```
x_train.shape
```

Out[37]:
```
(1575, 100, 1)
```

In [38]:
```
from keras.layers import Dense,Dropout,LSTM
```

In [39]:
```
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=50,activation='relu'))
model.add(Dropout(0.5))
```

In [40]:
```
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, 50)                26200

 dropout_3 (Dropout)         (None, 50)                0

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 108,411
Trainable params: 108,411
Non-trainable params: 0
_________________________________________________________________
```

In [41]:
```
model.compile(optimizer='adam',loss='mean_squared_error')
```

```
Epoch 1/50
50/50 [==============================] - 19s 205ms/step - loss: 0.0810
Epoch 2/50
50/50 [==============================] - 10s 196ms/step - loss: 0.0332
Epoch 3/50
50/50 [==============================] - 10s 202ms/step - loss: 0.0203
Epoch 4/50
50/50 [==============================] - 9s 189ms/step - loss: 0.0173
Epoch 5/50
50/50 [==============================] - 11s 215ms/step - loss: 0.0156
Epoch 6/50
50/50 [==============================] - 11s 213ms/step - loss: 0.0146
Epoch 7/50
50/50 [==============================] - 10s 207ms/step - loss: 0.0141
Epoch 8/50
50/50 [==============================] - 9s 185ms/step - loss: 0.0128
Epoch 9/50
50/50 [==============================] - 9s 184ms/step - loss: 0.0131
Epoch 10/50
50/50 [==============================] - 11s 214ms/step - loss: 0.0121
Epoch 11/50
50/50 [==============================] - 11s 209ms/step - loss: 0.0124
Epoch 12/50
50/50 [==============================] - 11s 222ms/step - loss: 0.0112
Epoch 13/50
50/50 [==============================] - 9s 188ms/step - loss: 0.0116
Epoch 14/50
50/50 [==============================] - 9s 181ms/step - loss: 0.0113
Epoch 15/50
50/50 [==============================] - 10s 205ms/step - loss: 0.0110
Epoch 16/50
50/50 [==============================] - 11s 210ms/step - loss: 0.0109
Epoch 17/50
50/50 [==============================] - 11s 211ms/step - loss: 0.0093
Epoch 18/50
50/50 [==============================] - 10s 208ms/step - loss: 0.0114
Epoch 19/50
50/50 [==============================] - 9s 177ms/step - loss: 0.0092
Epoch 20/50
50/50 [==============================] - 10s 195ms/step - loss: 0.0095
Epoch 21/50
50/50 [==============================] - 10s 199ms/step - loss: 0.0100
Epoch 22/50
50/50 [==============================] - 10s 196ms/step - loss: 0.0091
Epoch 23/50
50/50 [==============================] - 10s 192ms/step - loss: 0.0092
Epoch 24/50
50/50 [==============================] - 9s 183ms/step - loss: 0.0087
Epoch 25/50
50/50 [==============================] - 10s 206ms/step - loss: 0.0086
Epoch 26/50
50/50 [==============================] - 9s 184ms/step - loss: 0.0085
```

```
Epoch 27/50
50/50 [==============================] - 9s 176ms/step - loss: 0.0080
Epoch 28/50
50/50 [==============================] - 9s 185ms/step - loss: 0.0080
Epoch 29/50
50/50 [==============================] - 9s 177ms/step - loss: 0.0075
Epoch 30/50
50/50 [==============================] - 10s 192ms/step - loss: 0.0079
Epoch 31/50
50/50 [==============================] - 9s 181ms/step - loss: 0.0081
Epoch 32/50
50/50 [==============================] - 9s 180ms/step - loss: 0.0073
Epoch 33/50
50/50 [==============================] - 9s 189ms/step - loss: 0.0083
Epoch 34/50
50/50 [==============================] - 10s 193ms/step - loss: 0.0076
Epoch 35/50
50/50 [==============================] - 9s 184ms/step - loss: 0.0077
Epoch 36/50
50/50 [==============================] - 9s 177ms/step - loss: 0.0073
Epoch 37/50
50/50 [==============================] - 10s 194ms/step - loss: 0.0075
Epoch 38/50
50/50 [==============================] - 10s 191ms/step - loss: 0.0078
Epoch 39/50
50/50 [==============================] - 11s 213ms/step - loss: 0.0075
Epoch 40/50
50/50 [==============================] - 13s 270ms/step - loss: 0.0073
Epoch 41/50
50/50 [==============================] - 13s 256ms/step - loss: 0.0075
Epoch 42/50
50/50 [==============================] - 13s 256ms/step - loss: 0.0072
Epoch 43/50
50/50 [==============================] - 15s 307ms/step - loss: 0.0068
Epoch 44/50
50/50 [==============================] - 12s 245ms/step - loss: 0.0075
Epoch 45/50
50/50 [==============================] - 12s 242ms/step - loss: 0.0076
Epoch 46/50
50/50 [==============================] - 11s 223ms/step - loss: 0.0072
Epoch 47/50
50/50 [==============================] - 12s 233ms/step - loss: 0.0080
Epoch 48/50
50/50 [==============================] - 12s 232ms/step - loss: 0.0068
Epoch 49/50
50/50 [==============================] - 13s 262ms/step - loss: 0.0073
Epoch 50/50
50/50 [==============================] - 12s 245ms/step - loss: 0.0076
```

Out[41]:
```
<keras.callbacks.History at 0x2023a7070d0>
```

In [42]:
```
model.save('keras_model_2.h5')
```

In [43]:
```
data_testing.head()
```

Out[43]:

	Close
1675	17.066000
1676	17.133333
1677	16.415333
1678	16.666000
1679	16.667999

In [44]:
```
past_100_days=data_training.tail(100)
```

In [45]:
```
final_df=past_100_days.append(data_testing,ignore_index=True)
```

```
C:\Users\Jeet\AppData\Local\Temp\ipykernel_16292\3595571042.py:1: FutureWarning: The frame.append method is deprecated and will be removed from pandas
in a future version. Use pandas.concat instead.
  final_df=past_100_days.append(data_testing,ignore_index=True)
```

In [46]:
```
    final_df.head()
```

Out[46]:

	Close
0	13.380000
1	13.602000
2	14.246667
3	14.094000
4	13.897333

In [47]:
```
input_data=scaler.fit_transform(final_df)
```

Out[47]:
```
array([[0.08624047],
       [0.09945634],
       [0.13783392],
       [0.12874551],
       [0.11703777],
       [0.08743106],
       [0.07000836],
       [0.08723264],
       [0.08385924],
       [0.08945513],
       [0.08441482],
       [0.06961152],
       [0.05949126],
       [0.07989046],
       [0.09759106],
       [0.07989046],
       [0.0838195 ],
       [0.09441602],
       [0.09274916],
```

In [48]:
```
input_data.shape
```

Out[48]:
```
(818, 1)
```

In [49]:
```
x_test=[]
y_test=[]

for i in range(100,input_data.shape[0]):
```

In [50]:
```
x_test,y_test = np.array(x_test), np.array(y_test)
```

```
(718, 100, 1)
(718,)
```

In [51]:
```
#making predictions
```

```
23/23 [==============================] - 3s 72ms/step
```

In [52]:
```
y_predicted.shape
```

Out[52]:
```
(718, 1)
```

In [53]:
```
y_test
```

Out[53]:
```
array([0.30567133, 0.30967974, 0.26693653, 0.281859  , 0.28197799,
       0.2838037 , 0.28812958, 0.28670083, 0.27630273, 0.26947656,
       0.26165822, 0.25685602, 0.26669843, 0.31364852, 0.30463954,
       0.32972177, 0.32753896, 0.32920582, 0.28459739, 0.30178203,
       0.30086912, 0.33412713, 0.36214635, 0.39084026, 0.39056242,
       0.39270549, 0.39421361, 0.47446122, 0.49501933, 0.46049141,
       0.47517566, 0.49041548, 0.5295075 , 0.51490262, 0.46779379,
       0.49620986, 0.48604997, 0.48132713, 0.50224234, 0.49029649,
       0.50255981, 0.51220386, 0.53506365, 0.52069688, 0.51458503,
       0.53617493, 0.57094095, 0.55530416, 0.52407034, 0.46231689,
       0.51347387, 0.50887014, 0.5647101 , 0.58042625, 0.5720126 ,
       0.57879902, 0.54335831, 0.54784305, 0.50458389, 0.53216652,
       0.52331628, 0.52141132, 0.49565428, 0.52089536, 0.54712861,
       0.58010878, 0.61963726, 0.64309238, 0.64055247, 0.63848877,
       0.6681351 , 0.6900821 , 0.71706953, 0.75814587, 0.70782233,
       0.7145295 , 0.78175973, 0.80045238, 0.77933881, 0.76370203,
       0.75735207, 0.76703574, 0.78354558, 0.80819146, 0.81152517,
       0.78787158, 0.72786447, 0.76306696, 0.72143504, 0.72484812,
       0.6891693 , 0.58784774, 0.51537883, 0.53280147, 0.544033  ,
```

In [54]:
```
y_predicted
```

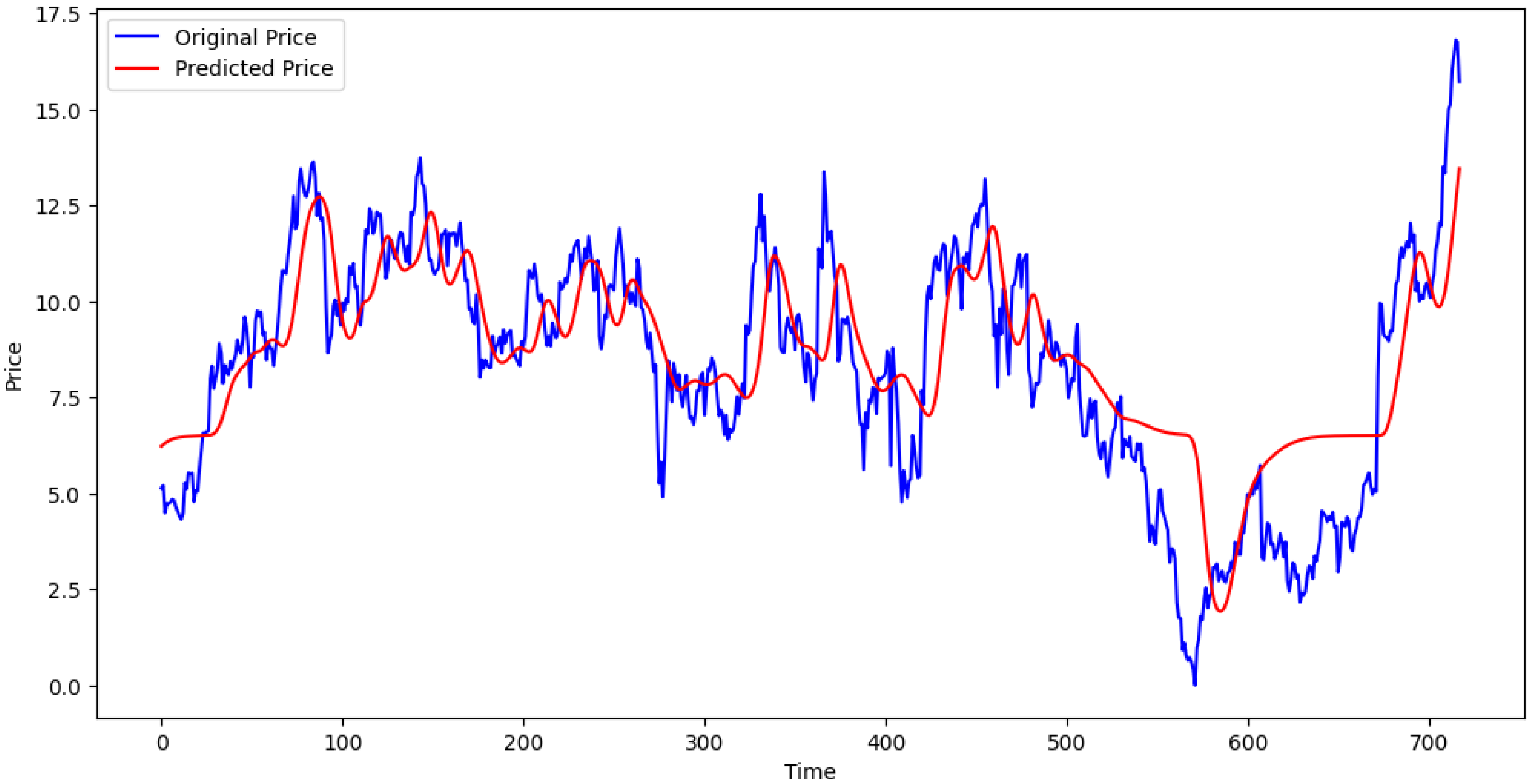
Out[54]:
```
array([[0.37072742],
       [0.37317175],
       [0.3753258 ],
       [0.37723893],
       [0.37889126],
       [0.3802936 ],
       [0.3814705 ],
       [0.38244164],
       [0.3832356 ],
       [0.38388285],
       [0.38440973],
       [0.38483948],
       [0.3851918 ],
       [0.38548136],
       [0.38571966],
       [0.3859168 ],
       [0.38608307],
       [0.3862231 ],
       [0.38634154],
```

In [55]:
```
scaler.scale_
```

Out[55]:
```
array([0.05953089])
```

In [56]:
```
scale_factor=1/0.05953089
```

In [57]:
```
plt.figure(figsize=(12,6))
plt.plot(y_test,'b',label='Original Price')
plt.plot(y_predicted,'r',label='Predicted Price')
plt.xlabel('Time')
plt.ylabel('Price')
```



In [ ]:

Code for Application for Predicting Prices

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pandas_datareader as data
from keras.models import load_model
from sklearn.preprocessing import MinMaxScaler
import streamlit as st
import yfinance as yf
from pandas_datareader import data as pdr
yf.pdr_override()
start='2010-01-01'
end='2023-01-01'

user_input=st.text_input('Enter Stock Picker','TSLA')

df=pdr.DataReader(user_input, start, end)

#describing the data
st.subheader('Data from 2010-2022')
st.write(df.describe())


#visualisations
st.subheader('Closing Price vs Time Chart')
fig=plt.figure(figsize=(12,6))
plt.plot(df.Close)
st.pyplot(fig)


st.subheader('Closing Price vs Time Chart with 100 moving average')
ma100=df.Close.rolling(100).mean()
fig=plt.figure(figsize=(12,6))
plt.plot(df.Close)
plt.plot(ma100)
st.pyplot(fig)

st.subheader('Closing Price vs Time Chart with 100 and 200 moving average')
ma100=df.Close.rolling(100).mean()
ma200=df.Close.rolling(200).mean()
fig=plt.figure(figsize=(12,6))
plt.plot(df.Close,'b',label='Original Price')
plt.plot(ma100,'r',label='Moving Avg. of past 100 days')
plt.plot(ma200,'g',label='Moving Avg. of last 200 days')
plt.xlabel('Time')
```

```
plt.ylabel('Price')
plt.legend()
st.pyplot(fig)

#splitting the data into Training and Testing
data_training=(df['Close'][0:int(len(df)*0.70)])
data_testing=(df['Close'][int(len(df)*0.70):int(len(df))])
data_training=data_training.to_frame()
data_testing=data_testing.to_frame()

scaler=MinMaxScaler(feature_range=(0,1))
data_training_array=scaler.fit_transform(data_training)

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)

#Loading my model
model=load_model('keras_model.h5')

#Testing Part
past_100_days=data_training.tail(100)
final_df=past_100_days.append(data_testing,ignore_index=True)
input_data=scaler.fit_transform(final_df)

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)

#predictions
y_predicted=model.predict(x_test)
scaler=scaler.scale_

scale_factor=1/scaler[0]
y_predicted=y_predicted*scale_factor
y_test=y_test*scale_factor


#Final graph
```

```
st.subheader('Predictions vs Original')
fig2=plt.figure(figsize=(12,6))
plt.plot(y_test,'b',label='Original Price')
plt.plot(y_predicted,'r',label='Predicted Price')
plt.xlabel('Time')
plt.ylabel('Price')
plt.legend()
st.pyplot(fig2)
```

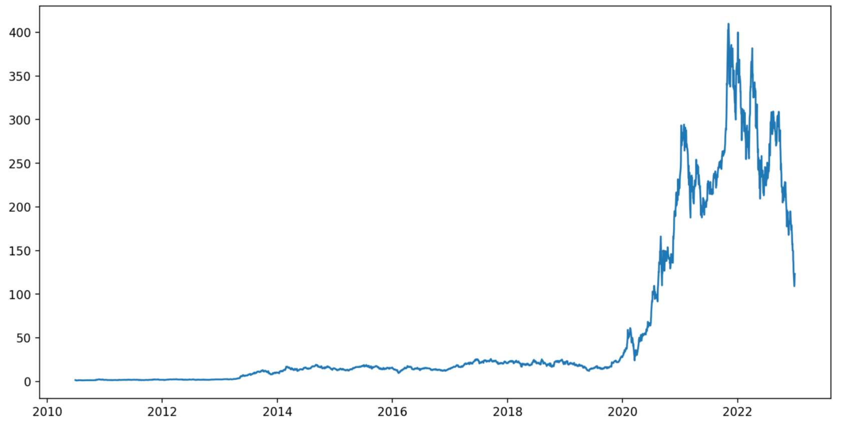
Enter Stock Picker

TSLA

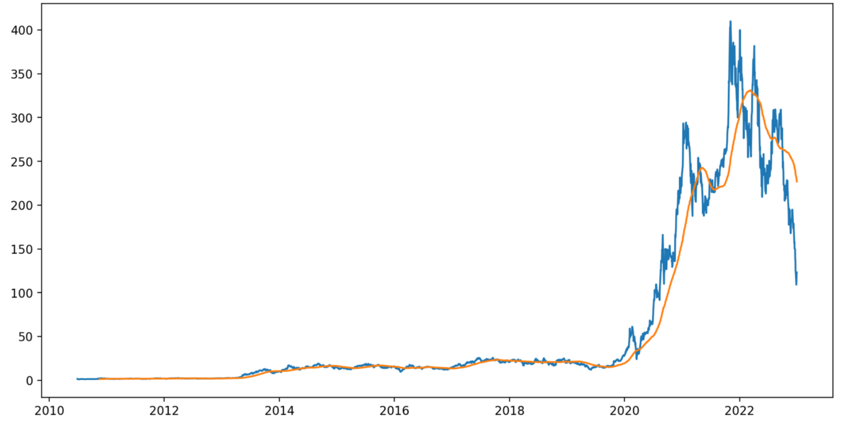
Data from 2010-2022

	Open	High	Low	Close	Adj Close	Volume
count	3,150.0000	3,150.0000	3,150.0000	3,150.0000	3,150.0000	3,150.0000
mean	58.8606	60.1767	57.4030	58.8075	58.8075	93,596,391.4286
std	95.6586	97.8546	93.1753	95.5264	95.5264	81,698,443.8503
min	1.0760	1.1087	0.9987	1.0533	1.0533	1,777,500.0000
25%	8.9762	9.1175	8.7657	8.9577	8.9577	42,346,575.0000
50%	16.2290	16.4910	15.9450	16.2223	16.2223	75,966,000.0000
75%	24.6225	25.0867	24.1587	24.4480	24.4480	117,297,750.0000
max	411.4700	414.4967	405.6667	409.9700	409.9700	914,082,000.0000

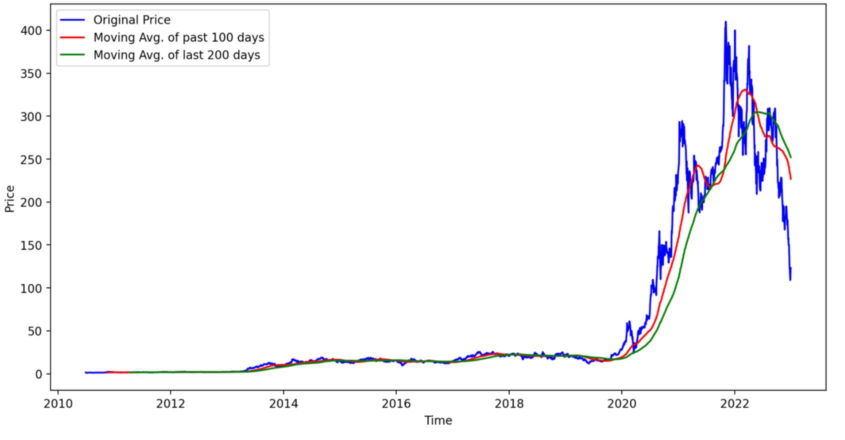
Closing Price vs Time Chart



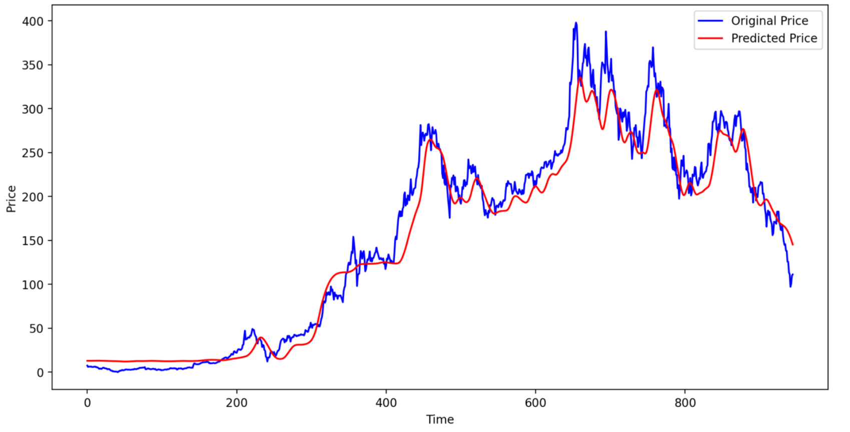
Closing Price vs Time Chart with 100 moving average



Closing Price vs Time Chart with 100 and 200 moving average



Predictions vs Original


Original Price
Predicted Price
Price
Time
0
50
100
150
200
250
300
350
400
0
200
400
600
800

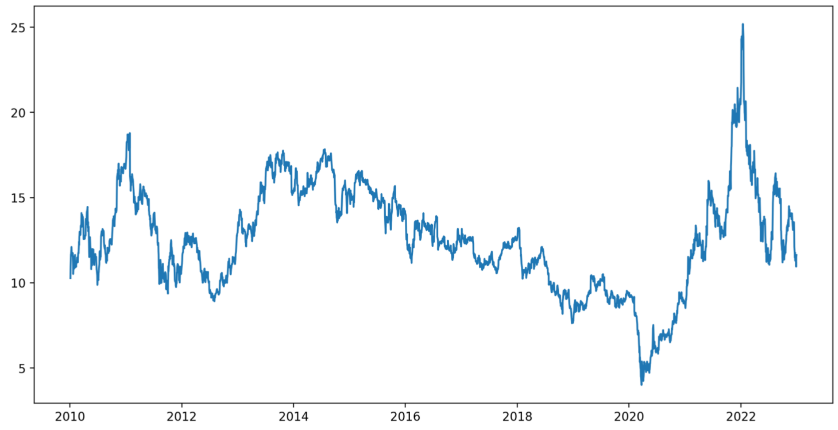
Enter Stock Picker

F

Data from 2010-2022

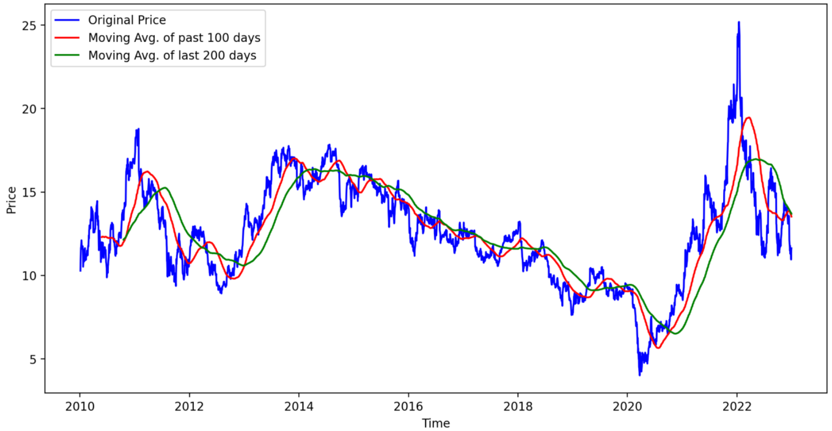
	Open	High	Low	Close	Adj Close	Volume
count	3,272.0000	3,272.0000	3,272.0000	3,272.0000	3,272.0000	3,272.0000
mean	12.5894	12.7332	12.4245	12.5792	9.8752	53,342,136.2775
std	3.0533	3.0848	3.0183	3.0530	2.6256	35,610,919.0839
min	4.2700	4.4200	3.9600	4.0100	3.8639	7,128,800.0000
25%	10.6175	10.7775	10.4475	10.6100	8.2713	30,741,675.0000
50%	12.4650	12.5800	12.3200	12.4600	9.5625	43,763,700.0000
75%	14.8300	15.0000	14.6525	14.8425	11.0063	63,943,575.0000
max	24.8700	25.8700	24.3700	25.1900	24.3947	480,879,500.0000

Closing Price vs Time Chart

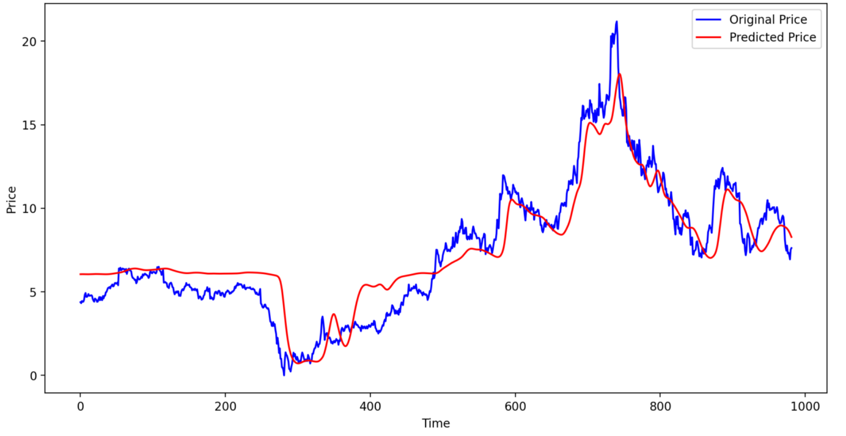


Closing Price vs Time Chart with 100 moving average

Closing Price vs Time Chart with 100 and 200 moving average



Predictions vs Original


Original Price
Predicted Price
Price
Time
0
5
10
15
20
0
200
400
600
800
1000

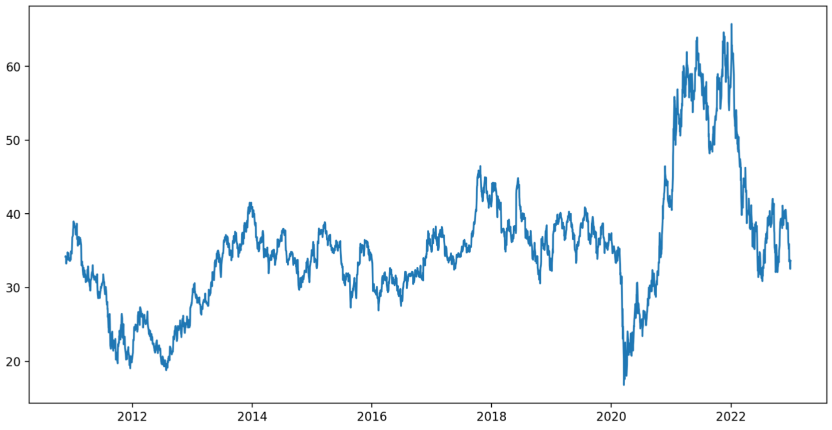
Enter Stock Picker

GM

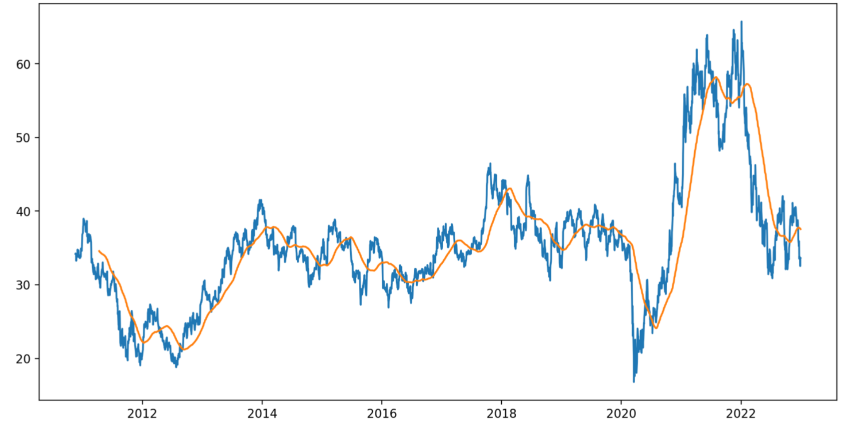
Data from 2010-2022

	Open	High	Low	Close	Adj Close	Volume
count	3,050.0000	3,050.0000	3,050.0000	3,050.0000	3,050.0000	3,050.0000
mean	35.4375	35.8583	34.9686	35.4136	31.4007	14,522,368.9836
std	8.6608	8.7565	8.5413	8.6488	10.1109	11,263,430.5644
min	16.3400	18.5600	14.3300	16.8000	14.4651	2,757,600.0000
25%	30.7900	31.1350	30.3825	30.7500	25.2942	9,435,225.0000
50%	35.0000	35.3150	34.5500	34.9150	29.3339	12,576,300.0000
75%	38.2500	38.6000	37.7575	38.2300	35.8995	16,827,525.0000
max	65.5200	67.2100	62.6900	65.7400	65.4447	457,044,300.0000

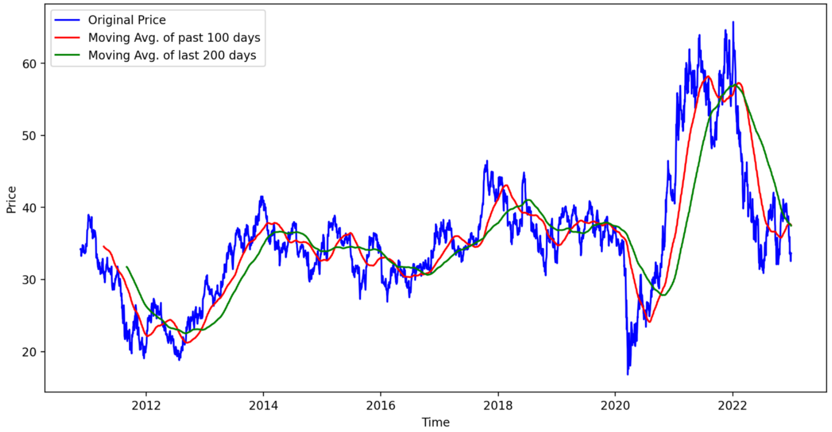
Closing Price vs Time Chart



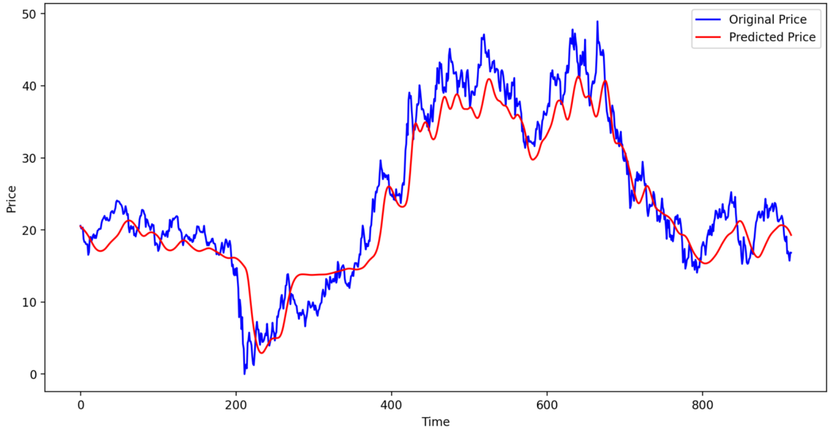
Closing Price vs Time Chart with 100 moving average



Closing Price vs Time Chart with 100 and 200 moving average



Predictions vs Original


Original Price
Predicted Price
Price
Time
0
200
400
600
800
0
10
20
30
40
50