1. Import Libraries

In [ ]:
```
import warnings
warnings.filterwarnings('ignore')

import math
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout, Dense, Activation

import nltk
from nltk.classify import NaiveBayesClassifier
from nltk.corpus import subjectivity
from nltk.sentiment import SentimentAnalyzer
from nltk.sentiment.util import *

from sklearn import preprocessing, metrics
from sklearn.preprocessing import MinMaxScaler
```

2. Upload Datasets For Stock Data And News Headlines

In [ ]:
```
stock_price = pd.read_csv('/content/drive/MyDrive/task7_stock_market_predictior
stock_headlines = pd.read_csv('/content/drive/MyDrive/task7_stock_market_predic
```

3. Data Cleaning

In [ ]:
```
stock_price.head()
```

Out[21]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2001-01-02	3953.219971	4028.570068	3929.370117	4018.879883	4018.879883	0.0
1	2001-01-03	3977.580078	4067.659912	3977.580078	4060.020020	4060.020020	0.0
2	2001-01-04	4180.970215	4180.970215	4109.549805	4115.370117	4115.370117	0.0
3	2001-01-05	4116.339844	4195.009766	4115.350098	4183.729980	4183.729980	0.0
4	2001-01-08	4164.759766	4206.720215	4101.529785	4120.430176	4120.430176	0.0

In [ ]:
```
stock_headlines.head()
```

Out[22]:

	publish_date	headline_category	headline_text
0	20010102	unknown	Status quo will not be disturbed at Ayodhya; s...
1	20010102	unknown	Fissures in Hurriyat over Pak visit
2	20010102	unknown	America's unwanted heading for India?
3	20010102	unknown	For bigwigs; it is destination Goa
4	20010102	unknown	Extra buses to clear tourist traffic

In [ ]:
```
# displaying number of records in both stock_price and stock_headlines datasets
len(stock_price), len(stock_headlines)
```

Out[23]:
```
(5043, 3424067)
```

In [ ]:
```
# checking for null values in both the datasets
stock_price.isna().any(), stock_headlines.isna().any()
```

Out[24]:
```
(Date         False
 Open          True
 High          True
 Low           True
 Close         True
 Adj Close     True
 Volume        True
 dtype: bool, publish_date         False
 headline_category    False
 headline_text        False
 dtype: bool)
```

3.1. Numerical Stock Data

In [ ]:
```
# dropping duplicates
stock_price = stock_price.drop_duplicates()

# coverting the datatype of column 'Date' from type object to type 'datetime'
stock_price['Date'] = pd.to_datetime(stock_price['Date']).dt.normalize()

# filtering the important columns required
stock_price = stock_price.filter(['Date', 'Close', 'Open', 'High', 'Low', 'Volu

# setting column 'Date' as the index column
stock_price.set_index('Date', inplace= True)

# sorting the data according to the index i.e 'Date'
stock_price = stock_price.sort_index(ascending=True, axis=0)
stock_price
```

Out[26]:

Date	Close	Open	High	Low	Volume
2001-01-02	4018.879883	3953.219971	4028.570068	3929.370117	0.0
2001-01-03	4060.020020	3977.580078	4067.659912	3977.580078	0.0
2001-01-04	4115.370117	4180.970215	4180.970215	4109.549805	0.0
2001-01-05	4183.729980	4116.339844	4195.009766	4115.350098	0.0
2001-01-08	4120.430176	4164.759766	4206.720215	4101.529785	0.0
...	...	...	...	...	...
2021-03-01	49849.839844	49747.710938	50058.421875	49440.460938	18400.0
2021-03-02	50296.890625	50258.089844	50439.820313	49807.121094	17500.0
2021-03-03	51444.648438	50738.210938	51539.890625	50512.839844	15800.0
2021-03-04	50846.078125	50812.140625	51256.550781	50539.921875	21800.0
2021-03-05	50405.320313	50517.359375	50886.191406	50160.539063	19200.0

5043 rows × 5 columns

3.2. Textual News Headlines Data

In [ ]:
```
# dropping duplicates
stock_headlines = stock_headlines.drop_duplicates()

# coverting the datatype of column 'Date' from type string to type 'datetime'
stock_headlines['publish_date'] = stock_headlines['publish_date'].astype(str)
stock_headlines['publish_date'] = stock_headlines['publish_date'].apply(lambda
stock_headlines['publish_date'] = pd.to_datetime(stock_headlines['publish_date'

# filtering the important columns required
stock_headlines = stock_headlines.filter(['publish_date', 'headline_text'])

# grouping the news headlines according to 'Date'
stock_headlines = stock_headlines.groupby(['publish_date'])['headline_text'].ap

# setting column 'Date' as the index column
stock_headlines.set_index('publish_date', inplace= True)

# sorting the data according to the index i.e 'Date'
stock_headlines = stock_headlines.sort_index(ascending=True, axis=0)
stock_headlines
```

Out[29]:

	headline_text
publish_date	
2001-01-02	Status quo will not be disturbed at Ayodhya; s...
2001-01-03	Powerless north India gropes in the dark,Think...
2001-01-04	The string that pulled Stephen Hawking to Indi...
2001-01-05	Light combat craft takes India into club class...
2001-01-06	Light combat craft takes India into club class...
...	...
2020-12-27	#BigInterview! Dhritiman Chatterjee: Nobody da...
2020-12-28	Horoscope Today; 28 December 2020: Check astro...
2020-12-29	Man recovers charred remains of 'thief' from h...
2020-12-30	Numerology Readings 30 December 2020: Predicti...
2020-12-31	Horoscope Today; 31 December 2020: Check astro...

7262 rows × 1 columns

4. Combine Stock Data

In [ ]:
```python
# concatenating the datasets stock_price and stock_headlines
stock_data = pd.concat([stock_price, stock_headlines], axis=1)

# dropping the null values if any
stock_data.dropna(axis=0, inplace=True)

# displaying the combined stock_data
stock_data
```

Out[31]:

	Close	Open	High	Low	Volume	headline_text
2001-01-02	4018.879883	3953.219971	4028.570068	3929.370117	0.0	Status quo will not be disturbed at Ayodhya; s...
2001-01-03	4060.020020	3977.580078	4067.659912	3977.580078	0.0	Powerless north India gropes in the dark,Think...
2001-01-04	4115.370117	4180.970215	4180.970215	4109.549805	0.0	The string that pulled Stephen Hawking to Indi...
2001-01-05	4183.729980	4116.339844	4195.009766	4115.350098	0.0	Light combat craft takes India into club class...
2001-01-08	4120.430176	4164.759766	4206.720215	4101.529785	0.0	Sangh Parivar; Babri panel up the ante,Frontru...
...	...	...	...	...	...	...
2020-12-24	46973.539063	46743.488281	47053.398438	46539.019531	13700.0	How to set the mood for sex during cold winter...
2020-12-28	47353.750000	47153.589844	47406.718750	47148.238281	9600.0	Horoscope Today; 28 December 2020: Check astro...
2020-12-29	47613.078125	47466.621094	47714.550781	47361.898438	12800.0	Man recovers charred remains of 'thief' from h...
2020-12-30	47746.218750	47789.031250	47807.851563	47358.359375	15600.0	Numerology Readings 30 December 2020: Predicti...
2020-12-31	47751.328125	47753.109375	47896.968750	47602.121094	13900.0	Horoscope Today; 31 December 2020: Check astro...

4893 rows × 6 columns

In [ ]:
```
#alternate way is to use merge funtion and inner join operation
pd.merge(stock_price, stock_headlines, left_index=True, right_index=True, how='
```

Out[34]:

	Close	Open	High	Low	Volume	headline_text
2001-01-02	4018.879883	3953.219971	4028.570068	3929.370117	0.0	Status quo will not be disturbed at Ayodhya; s...
2001-01-03	4060.020020	3977.580078	4067.659912	3977.580078	0.0	Powerless north India gropes in the dark,Think...
2001-01-04	4115.370117	4180.970215	4180.970215	4109.549805	0.0	The string that pulled Stephen Hawking to Indi...
2001-01-05	4183.729980	4116.339844	4195.009766	4115.350098	0.0	Light combat craft takes India into club class...
2001-01-08	4120.430176	4164.759766	4206.720215	4101.529785	0.0	Sangh Parivar; Babri panel up the ante,Frontru...
...	...	...	...	...	...	...
2020-12-24	46973.539063	46743.488281	47053.398438	46539.019531	13700.0	How to set the mood for sex during cold winter...
2020-12-28	47353.750000	47153.589844	47406.718750	47148.238281	9600.0	Horoscope Today; 28 December 2020: Check astro...
2020-12-29	47613.078125	47466.621094	47714.550781	47361.898438	12800.0	Man recovers charred remains of 'thief' from h...
2020-12-30	47746.218750	47789.031250	47807.851563	47358.359375	15600.0	Numerology Readings 30 December 2020: Predicti...
2020-12-31	47751.328125	47753.109375	47896.968750	47602.121094	13900.0	Horoscope Today; 31 December 2020: Check astro...

4968 rows × 6 columns

5. Sentiment Analysis

In [ ]:
```
# adding empty sentiment columns to stock_data for later calculation
stock_data['compound'] = ''
stock_data['negative'] = ''
stock_data['neutral'] = ''
stock_data['positive'] = ''
stock_data.head()
```

Out[35]:

	Close	Open	High	Low	Volume	headline_text	compound	ne
2001-01-02	4018.879883	3953.219971	4028.570068	3929.370117	0.0	Status quo will not be disturbed at Ayodhya; s...		
2001-01-03	4060.020020	3977.580078	4067.659912	3977.580078	0.0	Powerless north India gropes in the dark,Think...		
2001-01-04	4115.370117	4180.970215	4180.970215	4109.549805	0.0	The string that pulled Stephen Hawking to Indi...		
2001-01-05	4183.729980	4116.339844	4195.009766	4115.350098	0.0	Light combat craft takes India into club class...		
2001-01-08	4120.430176	4164.759766	4206.720215	4101.529785	0.0	Sangh Parivar; Babri panel up the ante,Frontru...		

In [ ]:
```
import nltk
nltk.download('vader_lexicon')
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
```

Out[41]: True

In [ ]:
```
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import unicodedata

# instantiating the Sentiment Analyzer
sid = SentimentIntensityAnalyzer()

# calculating sentiment scores
stock_data['compound'] = stock_data['headline_text'].apply(lambda x: sid.polari
stock_data['negative'] = stock_data['headline_text'].apply(lambda x: sid.polari
stock_data['neutral'] = stock_data['headline_text'].apply(lambda x: sid.polarit
stock_data['positive'] = stock_data['headline_text'].apply(lambda x: sid.polari

# displaying the stock data
stock_data.head()
```

Out[50]:

	Close	Open	High	Low	Volume	headline_text	compound	ne
2001-01-02	4018.879883	3953.219971	4028.570068	3929.370117	0.0	Status quo will not be disturbed at Ayodhya; s...	-0.9621	
2001-01-03	4060.020020	3977.580078	4067.659912	3977.580078	0.0	Powerless north India gropes in the dark,Think...	0.6322	
2001-01-04	4115.370117	4180.970215	4180.970215	4109.549805	0.0	The string that pulled Stephen Hawking to Indi...	0.6648	
2001-01-05	4183.729980	4116.339844	4195.009766	4115.350098	0.0	Light combat craft takes India into club class...	0.9032	
2001-01-08	4120.430176	4164.759766	4206.720215	4101.529785	0.0	Sangh Parivar; Babri panel up the ante,Frontru...	-0.9638	

In [ ]:
```
# dropping the 'headline_text' which is unwanted now
stock_data.drop(['headline_text'], inplace=True, axis=1)

# rearranging the columns of the whole stock_data
stock_data = stock_data[['Close', 'compound', 'negative', 'neutral', 'positive'

# set the index name
stock_data.index.name = 'Date'

# displaying the final stock_data
stock_data.head()
```

Out[55]:

Date	Close	compound	negative	neutral	positive	Open	High	Low
2001-01-02	4018.879883	-0.9621	0.119	0.817	0.064	3953.219971	4028.570068	3929.37011
2001-01-03	4060.020020	0.6322	0.084	0.817	0.098	3977.580078	4067.659912	3977.58007
2001-01-04	4115.370117	0.6648	0.077	0.843	0.080	4180.970215	4180.970215	4109.54980
2001-01-05	4183.729980	0.9032	0.105	0.746	0.149	4116.339844	4195.009766	4115.35009
2001-01-08	4120.430176	-0.9638	0.119	0.855	0.026	4164.759766	4206.720215	4101.52978

In [ ]:
```
# writing the prepared stock_data to disk
stock_data.to_csv('stock_data.csv')
```

6. Exploratory Data Analysis

In [ ]:
```
# displaying the shape i.e. number of rows and columns of stock_data
stock_data.shape
```

Out[57]:
```
(4893, 9)
```

In [ ]:
```
# checking for null values
stock_data.isna().any()
```

Out[58]:
```
Close       False
compound    False
negative    False
neutral     False
positive    False
Open        False
High        False
Low         False
Volume      False
dtype: bool
```

In [ ]:
```python
# displaying stock_data statistics
stock_data.describe(include='all')
```

Out[59]:

	Close	compound	negative	neutral	positive	Open	
count	4893.000000	4893.000000	4893.000000	4893.000000	4893.000000	4893.000000	4893.00
mean	18685.761055	-0.877818	0.125464	0.789046	0.085496	18706.141903	18818.84
std	11233.725489	0.440666	0.024224	0.033163	0.020759	11250.819220	11290.04
min	2600.120117	-1.000000	0.000000	0.000000	0.000000	2621.889893	2682.59
25%	8929.440430	-0.999800	0.112000	0.769000	0.075000	8939.379883	9033.99
50%	17618.349609	-0.999100	0.127000	0.786000	0.085000	17650.820313	17769.25
75%	27288.169922	-0.994600	0.141000	0.807000	0.096000	27316.429688	27445.24
max	47751.328125	1.000000	0.444000	1.000000	0.608000	47789.031250	47896.96

In [ ]:
```python
# displaying stock_data information
stock_data.info()
```

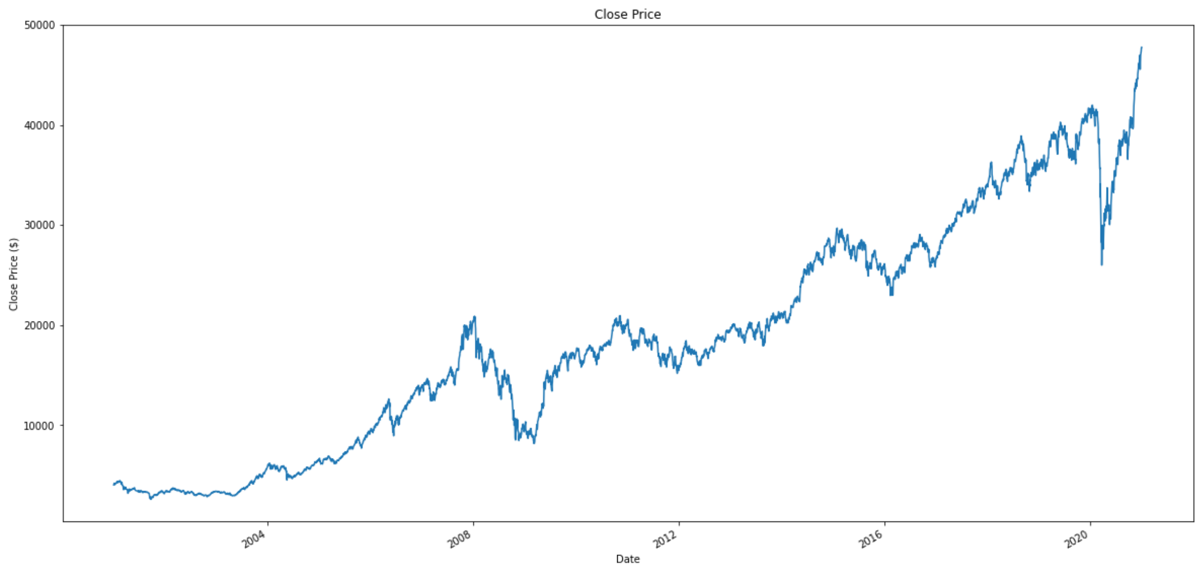
```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 4893 entries, 2001-01-02 to 2020-12-31
Data columns (total 9 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Close     4893 non-null   float64
 1   compound  4893 non-null   float64
 2   negative  4893 non-null   float64
 3   neutral   4893 non-null   float64
 4   positive  4893 non-null   float64
 5   Open      4893 non-null   float64
 6   High      4893 non-null   float64
 7   Low       4893 non-null   float64
 8   Volume    4893 non-null   float64
dtypes: float64(9)
memory usage: 542.3 KB
```

In [ ]:
```
# setting figure size
plt.figure(figsize=(20,10))

# plotting close price
stock_data['Close'].plot()

# setting plot title, x and y labels
plt.title("Close Price")
plt.xlabel('Date')
plt.ylabel('Close Price ($)')
```

Out[64]:
```
Text(0, 0.5, 'Close Price ($)')
```



In [ ]:

```
# calculating 7 day rolling mean
stock_data.rolling(7).mean().head(20)
```

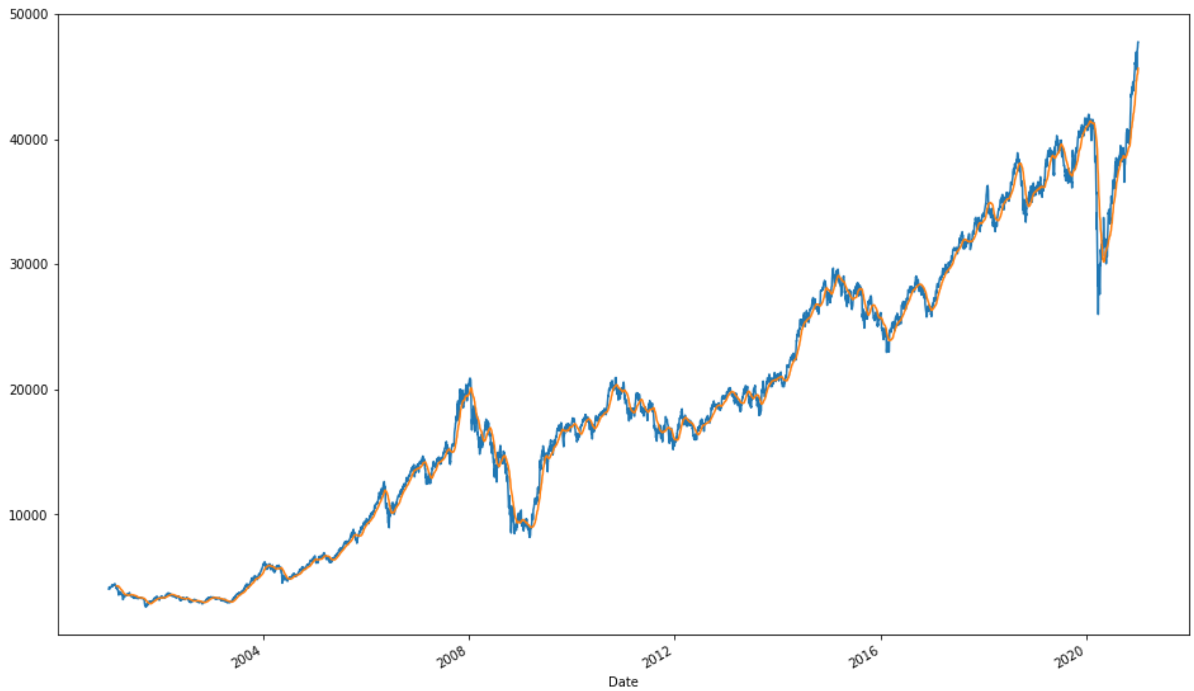
Out[65]:

Date	Close	compound	negative	neutral	positive	Open	High	L
2001-01-02	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
2001-01-03	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
2001-01-04	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
2001-01-05	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
2001-01-08	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
2001-01-09	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
2001-01-10	4095.911447	-0.179071	0.121714	0.810429	0.067571	4094.170027	4143.089983	4052.832
2001-01-23	4135.598598	0.091157	0.109857	0.818571	0.071286	4140.542899	4184.972831	4101.904
2001-01-24	4173.655727	-0.128286	0.111571	0.825286	0.063000	4189.532854	4223.794294	4147.351
2001-01-25	4204.348598	-0.364029	0.120143	0.818000	0.061714	4210.514230	4246.702846	4170.801
2001-01-29	4211.611433	-0.354529	0.111000	0.829857	0.058857	4216.588518	4256.381417	4177.017
2001-01-30	4247.555699	-0.339414	0.110714	0.823143	0.065857	4231.538574	4280.167132	4199.192
2001-01-31	4276.328578	-0.282729	0.107143	0.826143	0.066571	4267.658552	4314.808594	4230.860
2001-02-01	4310.395717	-0.319629	0.089714	0.834143	0.076000	4289.308524	4343.312919	4258.678
2001-02-02	4318.334263	-0.313943	0.094286	0.811286	0.094286	4289.755650	4350.691476	4259.834
2001-02-05	4324.627162	-0.109571	0.097429	0.793571	0.108857	4292.185686	4358.945731	4265.807
2001-02-06	4331.065709	0.140486	0.087000	0.799286	0.113571	4301.385742	4367.994280	4279.554
2001-02-07	4342.260045	0.121786	0.089714	0.797000	0.113286	4332.537179	4384.631487	4300.185
2001-02-08	4343.567174	0.371843	0.079571	0.799571	0.120857	4336.031459	4386.721470	4305.059
2001-02-09	4353.654297	0.589529	0.075714	0.790143	0.134143	4340.011440	4390.600028	4314.389

In [ ]:
```
# setting figure size
plt.figure(figsize=(16,10))

# plotting the close price and a 30-day rolling mean of close price
stock_data['Close'].plot()
stock_data.rolling(window=30).mean()['Close'].plot()
```

Out[70]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2fd75fc3d0>



7. Data Preparation

In [ ]:
```
# calculating data_to_use
percentage_of_data = 1.0
data_to_use = int(percentage_of_data*(len(stock_data)-1))

# using 80% of data for training
train_end = int(data_to_use*0.8)
total_data = len(stock_data)
start = total_data - data_to_use

# printing number of records in the training and test datasets
print("Number of records in Training Data:", train_end)
print("Number of records in Test Data:", total_data - train_end)
```

```
Number of records in Training Data: 3913
Number of records in Test Data: 980
```

In [ ]:
```
# predicting one step ahead
steps_to_predict = 1

# capturing data to be used for each column
close_price = stock_data.iloc[start:total_data,0] #close
compound = stock_data.iloc[start:total_data,1] #compound
negative = stock_data.iloc[start:total_data,2] #neg
neutral = stock_data.iloc[start:total_data,3] #neu
positive = stock_data.iloc[start:total_data,4] #pos
open_price = stock_data.iloc[start:total_data,5] #open
high = stock_data.iloc[start:total_data,6] #high
low = stock_data.iloc[start:total_data,7] #low
volume = stock_data.iloc[start:total_data,8] #volume

# printing close price
print("Close Price:")
close_price
```

```
Close Price:
```

Out[72]:
```
Date
2001-01-03     4060.020020
2001-01-04     4115.370117
2001-01-05     4183.729980
2001-01-08     4120.430176
2001-01-09     4125.310059
                  ...
2020-12-24    46973.539063
2020-12-28    47353.750000
2020-12-29    47613.078125
2020-12-30    47746.218750
2020-12-31    47751.328125
Name: Close, Length: 4892, dtype: float64
```

In [ ]:
```
# shifting next day close
close_price_shifted = close_price.shift(-1)

# shifting next day compound
compound_shifted = compound.shift(-1)

# concatenating the captured training data into a dataframe
data = pd.concat([close_price, close_price_shifted, compound, compound_shifted,

# setting column names of the revised stock data
data.columns = ['close_price', 'close_price_shifted', 'compound', 'compound_shi

# dropping nulls
data = data.dropna()
data.head(10)
```

Out[73]:

Date	close_price	close_price_shifted	compound	compound_shifted	volume	open_price	
2001-01-03	4060.020020	4115.370117	0.6322	0.6648	0.0	3977.580078	406
2001-01-04	4115.370117	4183.729980	0.6648	0.9032	0.0	4180.970215	418
2001-01-05	4183.729980	4120.430176	0.9032	-0.9638	0.0	4116.339844	419
2001-01-08	4120.430176	4125.310059	-0.9638	-0.9559	0.0	4164.759766	420
2001-01-09	4125.310059	4047.639893	-0.9559	-0.5719	0.0	4114.740234	416
2001-01-10	4047.639893	4296.689941	-0.5719	0.9295	0.0	4151.580078	415
2001-01-23	4296.689941	4326.419922	0.9295	-0.9039	0.0	4277.830078	432
2001-01-24	4326.419922	4330.220215	-0.9039	-0.9854	0.0	4320.509766	433
2001-01-25	4330.220215	4234.569824	-0.9854	0.9697	0.0	4327.839844	434
2001-01-29	4234.569824	4372.040039	0.9697	-0.8580	0.0	4158.859863	426

7.1. Setting Target Variable And Feature Dataset

In [ ]:
```
# setting the target variable as the shifted close_price
y = data['close_price_shifted']
y
```

Out[74]:
```
Date
2001-01-03     4115.370117
2001-01-04     4183.729980
2001-01-05     4120.430176
2001-01-08     4125.310059
2001-01-09     4047.639893
                  ...
2020-12-23    46973.539063
2020-12-24    47353.750000
2020-12-28    47613.078125
2020-12-29    47746.218750
2020-12-30    47751.328125
Name: close_price_shifted, Length: 4891, dtype: float64
```

In [ ]:
```
# setting the features dataset for prediction
cols = ['close_price', 'compound', 'compound_shifted', 'volume', 'open_price',
x = data[cols]
x
```

Out[75]:

Date	close_price	compound	compound_shifted	volume	open_price	high	
2001-01-03	4060.020020	0.6322	0.6648	0.0	3977.580078	4067.659912	3977.5
2001-01-04	4115.370117	0.6648	0.9032	0.0	4180.970215	4180.970215	4109.5
2001-01-05	4183.729980	0.9032	-0.9638	0.0	4116.339844	4195.009766	4115.3
2001-01-08	4120.430176	-0.9638	-0.9559	0.0	4164.759766	4206.720215	4101.5
2001-01-09	4125.310059	-0.9559	-0.5719	0.0	4114.740234	4166.839844	4101.0
...	...	...	...	...	...	...	
2020-12-23	46444.179688	-0.9995	-0.9966	10500.0	46072.300781	46513.320313	45899.1
2020-12-24	46973.539063	-0.9966	-0.9997	13700.0	46743.488281	47053.398438	46539.0
2020-12-28	47353.750000	-0.9997	-0.9997	9600.0	47153.589844	47406.718750	47148.2
2020-12-29	47613.078125	-0.9997	-0.9997	12800.0	47466.621094	47714.550781	47361.8
2020-12-30	47746.218750	-0.9997	-0.9996	15600.0	47789.031250	47807.851563	47358.3

4891 rows × 7 columns

7.3. Scaling the Target Variable and the Feature Dataset

Since we are using LSTM to predict stock prices, which is a time series data, it is important to understand that LSTM can be very sensitive to the scale of the data. Right now, if the data is observed, it is present in different scales. Therefore, it is important to re-scale the data so that the range of the dataset is same, for almost all records. Here a feature range of (-1,1) is used.

In [ ]:
```
# scaling the feature dataset
scaler_x = preprocessing.MinMaxScaler (feature_range=(-1, 1))
x = np.array(x).reshape((len(x) ,len(cols)))
x = scaler_x.fit_transform(x)

# scaling the target variable
scaler_y = preprocessing.MinMaxScaler (feature_range=(-1, 1))
y = np.array (y).reshape ((len( y), 1))
y = scaler_y.fit_transform (y)

# displaying the scaled feature dataset and the target variable
x, y
```

Out[76]:
```
(array([[-0.93532553,  0.6322    ,  0.6648    , ..., -0.93997007,
         -0.93861222, -0.93822641],
        [-0.93287349,  0.6648    ,  0.9032    , ..., -0.93096396,
         -0.93359019, -0.93233057],
        [-0.92984511,  0.9032    , -0.9638    , ..., -0.93382579,
         -0.93296794, -0.93207144],
        ...,
        [ 0.98261339, -0.9997    , -0.9997    , ...,  0.97186267,
          0.98222136,  0.99045457],
        [ 0.99410179, -0.9997    , -0.9997    , ...,  0.98572369,
          0.99586481,  1.        ],
        [ 1.        , -0.9997    , -0.9996    , ...,  1.        ,
          1.        ,  0.99984189]]), array([[-0.93288109],
        [-0.92985305],
        [-0.93265695],
        ...,
        [ 0.99387613],
        [ 0.99977368],
        [ 1.        ]]))
```

7.4. Dividing the dataset into Training and Test

Normally for any other dataset train_test_split from sklearn package is used, but for time series data like stock prices which is dependent on date, the dataset is divided into train and test dataset in a different way as shown below. In timeseries data, an observation for a particular date is always dependent on the previous date records.

In [ ]:
```
# preparing training and test dataset
X_train = x[0 : train_end,]
X_test = x[train_end+1 : len(x),]
y_train = y[0 : train_end]
y_test = y[train_end+1 : len(y)]

# printing the shape of the training and the test datasets
print('Number of rows and columns in the Training set X:', X_train.shape, 'and
print('Number of rows and columns in the Test set X:', X_test.shape, 'and y:',
```

```
Number of rows and columns in the Training set X: (3913, 7) and y: (3913, 1)
Number of rows and columns in the Test set X: (977, 7) and y: (977, 1)
```

In [ ]:
```
# reshaping the feature dataset for feeding into the model
X_train = X_train.reshape (X_train.shape + (1,))
X_test = X_test.reshape(X_test.shape + (1,))

# printing the re-shaped feature dataset
print('Shape of Training set X:', X_train.shape)
print('Shape of Test set X:', X_test.shape)
```

```
Shape of Training set X: (3913, 7, 1)
Shape of Test set X: (977, 7, 1)
```

9. Stock Data Modelling

In [ ]:
```
# setting the seed to achieve consistent and less random predictions at each ex
np.random.seed(2016)

# setting the model architecture
model=Sequential()
model.add(LSTM(100,return_sequences=True,activation='tanh',input_shape=(len(col
model.add(Dropout(0.1))
model.add(LSTM(100,return_sequences=True,activation='tanh'))
model.add(Dropout(0.1))
model.add(LSTM(100,activation='tanh'))
model.add(Dropout(0.1))
model.add(Dense(1))

# printing the model summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
lstm (LSTM)                 (None, 7, 100)            40800

dropout (Dropout)           (None, 7, 100)            0

lstm_1 (LSTM)               (None, 7, 100)            80400

dropout_1 (Dropout)         (None, 7, 100)            0

lstm_2 (LSTM)               (None, 100)               80400

dropout_2 (Dropout)         (None, 100)               0

dense (Dense)               (None, 1)                 101

=================================================================
Total params: 201,701
Trainable params: 201,701
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
# compiling the model
model.compile(loss='mse' , optimizer='adam')

# fitting the model using the training dataset
model.fit(X_train, y_train, validation_split=0.2, epochs=10, batch_size=8, verb
```

```
Epoch 1/10
392/392 [==============================] - 14s 21ms/step - loss: 0.0716 - val
_loss: 0.0224
Epoch 2/10
392/392 [==============================] - 7s 17ms/step - loss: 0.0029 - val_
loss: 0.0051
Epoch 3/10
392/392 [==============================] - 7s 17ms/step - loss: 0.0018 - val_
loss: 9.7626e-04
Epoch 4/10
392/392 [==============================] - 7s 18ms/step - loss: 0.0014 - val_
loss: 1.9364e-04
Epoch 5/10
392/392 [==============================] - 7s 18ms/step - loss: 0.0016 - val_
loss: 2.9687e-04
Epoch 6/10
392/392 [==============================] - 7s 18ms/step - loss: 0.0013 - val_
loss: 6.3891e-04
Epoch 7/10
392/392 [==============================] - 7s 18ms/step - loss: 0.0011 - val_
loss: 5.3691e-04
Epoch 8/10
392/392 [==============================] - 7s 17ms/step - loss: 9.8818e-04 -
val_loss: 6.9205e-04
Epoch 9/10
392/392 [==============================] - 7s 18ms/step - loss: 0.0010 - val_
loss: 2.5354e-04
Epoch 10/10
392/392 [==============================] - 7s 17ms/step - loss: 9.6923e-04 -
val_loss: 4.1875e-04
```

Out[80]:
```
<tensorflow.python.keras.callbacks.History at 0x7f2fd3218d50>
```

9.1. Saving the Model to disk

In [ ]:
```
# saving the model as a json file
model_json = model.to_json()
with open('model.json', 'w') as json_file:
    json_file.write(model_json)

# serialize weights to HDF5
model.save_weights('model.h5')
print('Model is saved to the disk')
```

```
Model is saved to the disk
```

10. Model Predictions

In [ ]:
```
# performing predictions
predictions = model.predict(X_test)

# unscaling the predictions
predictions = scaler_y.inverse_transform(np.array(predictions).reshape((len(pre

# printing the predictions
print('Predictions:')
predictions[0:5]
```

```
Predictions:
```

Out[82]:
```
array([[27186.035],
       [27393.72 ],
       [27548.578],
       [27664.785],
       [27590.85 ]], dtype=float32)
```

11. Model Evaluation

In [ ]:
```
# calculating the training mean-squared-error
train_loss = model.evaluate(X_train, y_train, batch_size = 1)

# calculating the test mean-squared-error
test_loss = model.evaluate(X_test, y_test, batch_size = 1)

# printing the training and the test mean-squared-errors
print('Train Loss =', round(train_loss,4))
print('Test Loss =', round(test_loss,4))
```

```
3913/3913 [==============================] - 12s 3ms/step - loss: 3.7877e-04
977/977 [==============================] - 3s 3ms/step - loss: 9.4026e-04
Train Loss = 0.0004
Test Loss = 0.0009
```

In [ ]:
```
# calculating root mean squared error
root_mean_square_error = np.sqrt(np.mean(np.power((y_test - predictions),2)))
print('Root Mean Square Error =', round(root_mean_square_error,4))
```

```
Root Mean Square Error = 36257.4782
```

In [ ]:
```
# calculating root mean squared error using sklearn.metrics package
rmse = metrics.mean_squared_error(y_test, predictions)
print('Root Mean Square Error (sklearn.metrics) =', round(np.sqrt(rmse),4))
```

```
Root Mean Square Error (sklearn.metrics) = 36257.4782
```

12. Plotting the Predictions against unseen data

In [ ]:
```
# unscaling the test feature dataset, x_test
X_test = scaler_x.inverse_transform(np.array(X_test).reshape((len(X_test), len(

# unscaling the test y dataset, y_test
y_train = scaler_y.inverse_transform(np.array(y_train).reshape((len(y_train), 1
y_test = scaler_y.inverse_transform(np.array(y_test).reshape((len(y_test), 1)))
```

In [ ]:
```
# plotting
plt.figure(figsize=(16,10))

# plt.plot([row[0] for row in y_train], label="Training Close Price")
plt.plot(predictions, label="Predicted Close Price")
plt.plot([row[0] for row in y_test], label="Testing Close Price")
plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.05), fancybox=True, shad
plt.show()
```