BHARAT INTERN TASK 1 - STOCK PRICE PREDICTION USING LSTM

SUBMITTED BY : ROHITH RAHUL

TESLA STOCK PRICE PREDICTION

OVERVIEW

The Tesla stock price dataset refers to a collection of data points that represent the historical or current price of the stock for Tesla Inc., a multinational corporation specializing in electric vehicles, energy storage and solar panel manufacturing based in Palo Alto, California. The data points typically include information such as the date, time, and the opening, closing, highest and lowest prices of the stock during a specific period of time. The Tesla stock price dataset can be used for a variety of purposes, such as studying market trends, conducting technical analysis, or training machine learning models for stock price prediction. The data can be obtained from various financial sources such as stock market exchanges, financial websites, or by directly accessing APIs provided by financial data providers.

The following are the common features found in a Tesla stock price dataset:

Date: The date on which the stock price data was recorded.

Open: This refers to the price of the stock at the beginning of the trading day.

Close: This refers to the price of the stock at the end of the trading day.

Adj. Close: The adjusted close price accounts for any corporate actions such as stock splits, dividends, etc. that occurred on that day.

High : The highest price of the stock during the trading day.

Low: The lowest price of the stock during the trading day.

These features can provide valuable information about the stock performance, trends and volatility over a certain period of time, and can be used in financial analysis, prediction, and decision making.

TABLE OF CONTENTS

1. Importing Essential Libraries
2. Importing Data

3. EDA and Feature Engineering

   A. Check for Null values

   B. Plots

   C. Moving Averages

4. Splitting the Time-series Data
5. Scaling Data using Min-Max scaler
6. Model Building
7. Prediction

IMPORTING ESSENTIAL LIBRARIES

In [1]:
```
import pandas as pd  #for data manipulation operations
import numpy as np   #for linear algebra

#Libraries for visualisation
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots

import datetime as dt

from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.preprocessing import MinMaxScaler

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.layers import LSTM

from itertools import cycle
```

IMPORTING DATA

In [2]:
```
#Loading the required data
df=pd.read_csv("C:/Users/ASUS/Downloads/Tesla Stock Price (2010 to 2023).csv")
df.set_index('Date',inplace=True)
df.head()
```

Out[2]:

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

In [3]:
```
print('Number of days present in the dataset: ',df.shape[0])
print('Number of fields present in the dataset: ',df.shape[1])
```

```
Number of days present in the dataset:  3162
Number of fields present in the dataset:  6
```

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 3162 entries, 29/06/2010 to 19/01/2023
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Open       3162 non-null   float64
 1   High       3162 non-null   float64
 2   Low        3162 non-null   float64
 3   Close      3162 non-null   float64
 4   Adj Close  3162 non-null   float64
 5   Volume     3162 non-null   int64  
dtypes: float64(5), int64(1)
memory usage: 172.9+ KB
```

In [5]:
```
df.describe()
```

Out[5]:

	Open	High	Low	Close	Adj Close	Volume
count	3162.000000	3162.000000	3162.000000	3162.000000	3162.000000	3.162000e+03
mean	59.090024	60.415403	57.622371	59.039845	59.039845	9.394769e+07
std	95.550672	97.746213	93.067484	95.420232	95.420232	8.175154e+07
min	1.076000	1.108667	0.998667	1.053333	1.053333	1.777500e+06
25%	9.037333	9.252500	8.828500	9.066833	9.066833	4.243012e+07
50%	16.294334	16.514666	16.016334	16.295666	16.295666	7.609725e+07
75%	24.965833	25.212667	24.438666	24.986833	24.986833	1.179720e+08
max	411.470001	414.496674	405.666656	409.970001	409.970001	9.140820e+08

EDA AND FEATURE ENGINEERING

In [6]:
```
from plotly.offline import init_notebook_mode
init_notebook_mode(connected=True)
```

Check for Null values

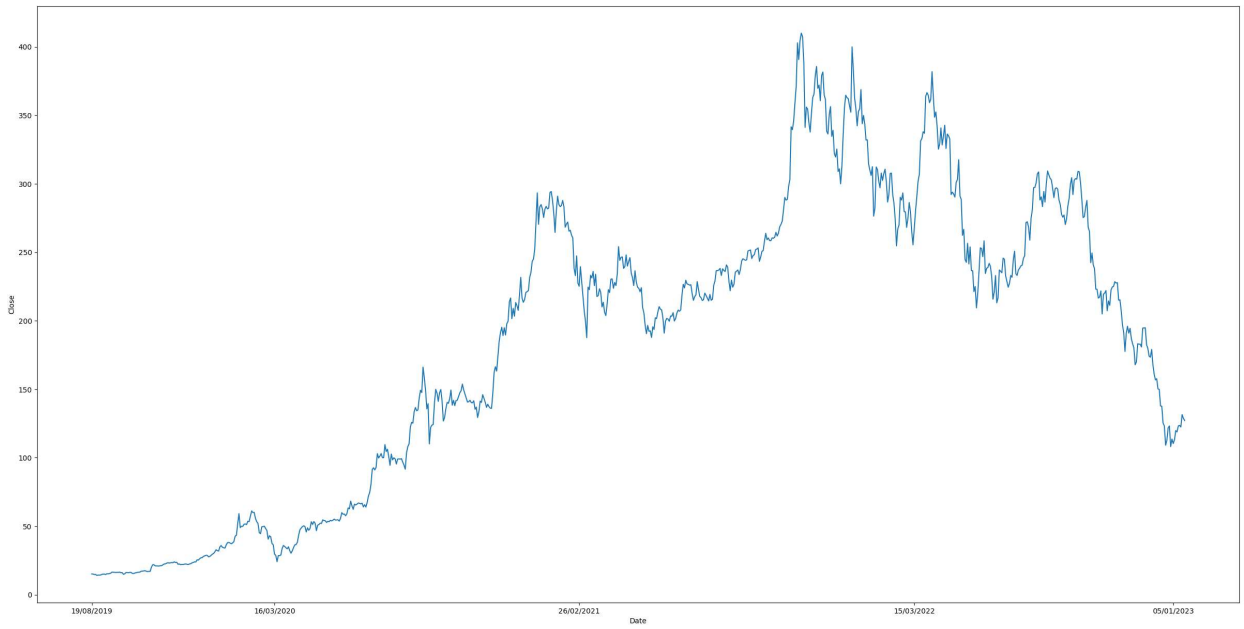
In [7]:
```
df.isnull().sum()
```

Out[7]:
```
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

Plots

The stock prices are almost constant until 2019 and thus I'll take the split of the data to only work on the data that shows methodical/abrupt changes.

In [23]:
```
data=df.iloc[2300:].copy()

plt.figure(figsize=(30,15))
ax=sns.lineplot(x=data.index,y=data['Close'])
plt.xticks(['19/08/2019','16/03/2020','26/02/2021','15/03/2022','05/01/2023'])
plt.show()
```



Moving Averages

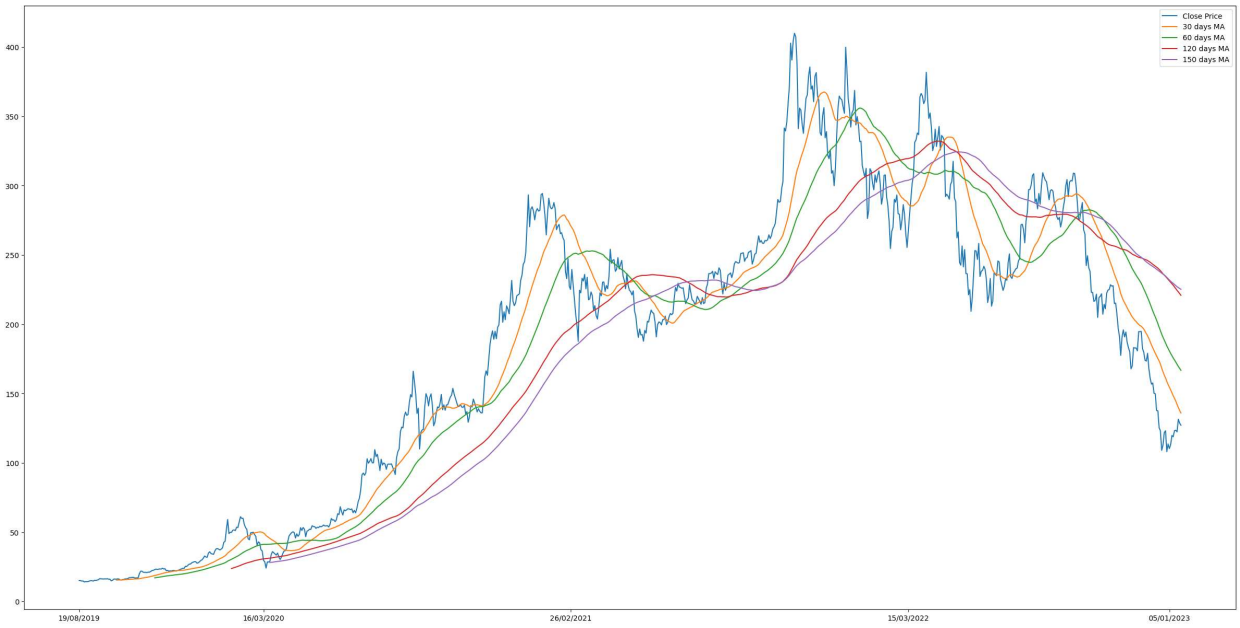
Moving Averages (MA) are a type of time series analysis method used to smooth out fluctuations in data by calculating the average of a set of data over a certain period of time. This average is then shifted forward in time to provide a smoothed representation of the data that can help to identify underlying patterns or trends. There are two main types of moving averages: simple moving averages (SMA) and weighted moving averages (WMA). A simple

moving average is calculated by taking the average of a set of data over a fixed period of time, while a weighted moving average gives more importance to the most recent data. Moving averages are widely used in finance, economics, and engineering to help forecast future trends and to identify buy/sell signals.

I'll take moving average for window sizes of 30,60,120 and 150 days.

In [9]:
```
ma_day = [30, 60, 120,150]

for ma in ma_day:
        column_name = f"MA for {ma} days"
        data[column_name] = data['Close'].rolling(ma).mean()
```

In [10]:
```
plt.figure(figsize=(30,15))
plt.plot(data['Close'],label='Close Price')
plt.plot(data['MA for 30 days'],label='30 days MA')
plt.plot(data['MA for 60 days'],label='60 days MA')
plt.plot(data['MA for 120 days'],label='120 days MA')
plt.plot(data['MA for 150 days'],label='150 days MA')
plt.xticks(['19/08/2019','16/03/2020','26/02/2021','15/03/2022','05/01/2023'])
plt.legend()
plt.show()
```



SPLITTING THE TIME-SERIES DATA

In [11]:
```
#Creating a new dataframe with only 'Close'
new_df = data['Close']
new_df.index = data.index

final_df=new_df.values

train_data=final_df[0:646,]
test_data=final_df[646:,]

train_df = pd.DataFrame()
test_df = pd.DataFrame()
```

```
train_df['Close'] = train_data
train_df.index = new_df[0:646].index
test_df['Close'] = test_data
test_df.index = new_df[646:].index
```

In [12]:
```
print("train_data: ", train_df.shape)
print("test_data: ", test_df.shape)
```

```
train_data:  (646, 1)
test_data:  (216, 1)
```

SCALING DATA USING MIN-MAX SCALER

Min-Max Scaler is a pre-processing technique used in machine learning for rescaling a feature or a set of features to a specific range, typically between 0 and 1. The method works by transforming the values of the feature to a new scale, while preserving the relative proportions between the values. The rescaling is done by subtracting the minimum value in the feature from each data point and dividing the result by the range (the difference between the maximum and minimum value). This ensures that all the values in the feature are now in the specified range, with 0 being the minimum and 1 being the maximum. Min-Max scaling is particularly useful when working with algorithms that make assumptions about the scale of the input features, such as some distance-based algorithms or algorithms sensitive to the scale of the input features.

In [13]:
```
# Using Min-Max scaler to scale data
scaler=MinMaxScaler(feature_range=(0,1))
scaled_data=scaler.fit_transform(final_df.reshape(-1,1))

X_train_data,y_train_data=[],[]

for i in range(60,len(train_df)):
    X_train_data.append(scaled_data[i-60:i,0])
    y_train_data.append(scaled_data[i,0])

X_train_data,y_train_data=np.array(X_train_data),np.array(y_train_data)

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

MODEL BUILDING

LSTMs are commonly used for modeling time series data as they are able to capture the long-term dependencies between inputs, while also being able to handle the noise and volatility that is often present in time series data. This makes LSTMs suitable for prediction tasks such as stock prices, weather forecasts, and energy demand. In a time series context, LSTMs take in previous time steps as inputs, and use their memory cells, gates, and state updates to process and make predictions on future time steps.

In [14]:
```
# Initializing the LSTM model
model = Sequential()
```

```
model.add(LSTM(units = 50, return_sequences = True, input_shape = (X_train_data.shape[
model.add(Dropout(0.2))
model.add(LSTM(units = 50, return_sequences = True))
model.add(Dropout(0.2))
model.add(LSTM(units = 50, return_sequences = True))
model.add(Dropout(0.2))
model.add(LSTM(units = 50))
model.add(Dropout(0.2))
model.add(Dense(units = 1))
```

In [15]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 60, 50)            10400

 dropout (Dropout)           (None, 60, 50)            0

 lstm_1 (LSTM)               (None, 60, 50)            20200

 dropout_1 (Dropout)         (None, 60, 50)            0

 lstm_2 (LSTM)               (None, 60, 50)            20200

 dropout_2 (Dropout)         (None, 60, 50)            0

 lstm_3 (LSTM)               (None, 50)                20200

 dropout_3 (Dropout)         (None, 50)                0

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 71051 (277.54 KB)
Trainable params: 71051 (277.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [16]:
```
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
model.fit(X_train_data, y_train_data, epochs = 150, batch_size = 32);
```

```
Epoch 1/150
19/19 [==============================] - 5s 40ms/step - loss: 0.0527
Epoch 2/150
19/19 [==============================] - 1s 39ms/step - loss: 0.0091
Epoch 3/150
19/19 [==============================] - 1s 40ms/step - loss: 0.0073
Epoch 4/150
19/19 [==============================] - 1s 40ms/step - loss: 0.0067
Epoch 5/150
19/19 [==============================] - 1s 41ms/step - loss: 0.0064
Epoch 6/150
19/19 [==============================] - 1s 41ms/step - loss: 0.0056
Epoch 7/150
19/19 [==============================] - 1s 41ms/step - loss: 0.0066
Epoch 8/150
19/19 [==============================] - 1s 41ms/step - loss: 0.0059
Epoch 9/150
19/19 [==============================] - 1s 40ms/step - loss: 0.0056
Epoch 10/150
19/19 [==============================] - 1s 41ms/step - loss: 0.0069
Epoch 11/150
19/19 [==============================] - 1s 40ms/step - loss: 0.0044
Epoch 12/150
19/19 [==============================] - 1s 40ms/step - loss: 0.0051
Epoch 13/150
19/19 [==============================] - 1s 40ms/step - loss: 0.0060
Epoch 14/150
19/19 [==============================] - 1s 43ms/step - loss: 0.0069
Epoch 15/150
19/19 [==============================] - 1s 42ms/step - loss: 0.0052
Epoch 16/150
19/19 [==============================] - 1s 42ms/step - loss: 0.0048
Epoch 17/150
19/19 [==============================] - 1s 41ms/step - loss: 0.0056
Epoch 18/150
19/19 [==============================] - 1s 43ms/step - loss: 0.0045
Epoch 19/150
19/19 [==============================] - 1s 41ms/step - loss: 0.0046
Epoch 20/150
19/19 [==============================] - 1s 44ms/step - loss: 0.0055
Epoch 21/150
19/19 [==============================] - 1s 43ms/step - loss: 0.0051
Epoch 22/150
19/19 [==============================] - 1s 42ms/step - loss: 0.0042
Epoch 23/150
19/19 [==============================] - 1s 42ms/step - loss: 0.0046
Epoch 24/150
19/19 [==============================] - 1s 42ms/step - loss: 0.0043
Epoch 25/150
19/19 [==============================] - 1s 44ms/step - loss: 0.0047
Epoch 26/150
19/19 [==============================] - 1s 42ms/step - loss: 0.0047
Epoch 27/150
19/19 [==============================] - 1s 43ms/step - loss: 0.0044
Epoch 28/150
19/19 [==============================] - 1s 41ms/step - loss: 0.0041
Epoch 29/150
19/19 [==============================] - 1s 42ms/step - loss: 0.0037
Epoch 30/150
19/19 [==============================] - 1s 43ms/step - loss: 0.0037
```

```
Epoch 31/150
19/19 [==============================] - 1s 44ms/step - loss: 0.0041
Epoch 32/150
19/19 [==============================] - 1s 43ms/step - loss: 0.0039
Epoch 33/150
19/19 [==============================] - 1s 43ms/step - loss: 0.0047
Epoch 34/150
19/19 [==============================] - 1s 46ms/step - loss: 0.0047
Epoch 35/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0043
Epoch 36/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0039
Epoch 37/150
19/19 [==============================] - 1s 43ms/step - loss: 0.0036
Epoch 38/150
19/19 [==============================] - 1s 44ms/step - loss: 0.0035
Epoch 39/150
19/19 [==============================] - 1s 44ms/step - loss: 0.0035
Epoch 40/150
19/19 [==============================] - 1s 44ms/step - loss: 0.0033
Epoch 41/150
19/19 [==============================] - 1s 44ms/step - loss: 0.0033
Epoch 42/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0035
Epoch 43/150
19/19 [==============================] - 1s 43ms/step - loss: 0.0034
Epoch 44/150
19/19 [==============================] - 1s 44ms/step - loss: 0.0039
Epoch 45/150
19/19 [==============================] - 1s 43ms/step - loss: 0.0028
Epoch 46/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0032
Epoch 47/150
19/19 [==============================] - 1s 44ms/step - loss: 0.0035
Epoch 48/150
19/19 [==============================] - 1s 43ms/step - loss: 0.0032
Epoch 49/150
19/19 [==============================] - 1s 43ms/step - loss: 0.0031
Epoch 50/150
19/19 [==============================] - 1s 44ms/step - loss: 0.0037
Epoch 51/150
19/19 [==============================] - 1s 44ms/step - loss: 0.0033
Epoch 52/150
19/19 [==============================] - 1s 43ms/step - loss: 0.0037
Epoch 53/150
19/19 [==============================] - 1s 47ms/step - loss: 0.0030
Epoch 54/150
19/19 [==============================] - 1s 46ms/step - loss: 0.0028
Epoch 55/150
19/19 [==============================] - 1s 46ms/step - loss: 0.0040
Epoch 56/150
19/19 [==============================] - 1s 43ms/step - loss: 0.0033
Epoch 57/150
19/19 [==============================] - 1s 44ms/step - loss: 0.0035
Epoch 58/150
19/19 [==============================] - 1s 46ms/step - loss: 0.0035
Epoch 59/150
19/19 [==============================] - 1s 44ms/step - loss: 0.0028
Epoch 60/150
19/19 [==============================] - 1s 44ms/step - loss: 0.0026
```

```
Epoch 61/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0030
Epoch 62/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0038
Epoch 63/150
19/19 [==============================] - 1s 44ms/step - loss: 0.0028
Epoch 64/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0027
Epoch 65/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0027
Epoch 66/150
19/19 [==============================] - 1s 43ms/step - loss: 0.0030
Epoch 67/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0036
Epoch 68/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0030
Epoch 69/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0030
Epoch 70/150
19/19 [==============================] - 1s 44ms/step - loss: 0.0024
Epoch 71/150
19/19 [==============================] - 1s 47ms/step - loss: 0.0028
Epoch 72/150
19/19 [==============================] - 1s 49ms/step - loss: 0.0024
Epoch 73/150
19/19 [==============================] - 1s 48ms/step - loss: 0.0022
Epoch 74/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0026
Epoch 75/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0028
Epoch 76/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0022
Epoch 77/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0024
Epoch 78/150
19/19 [==============================] - 1s 44ms/step - loss: 0.0026
Epoch 79/150
19/19 [==============================] - 1s 46ms/step - loss: 0.0026
Epoch 80/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0025
Epoch 81/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0022
Epoch 82/150
19/19 [==============================] - 1s 44ms/step - loss: 0.0021
Epoch 83/150
19/19 [==============================] - 1s 44ms/step - loss: 0.0026
Epoch 84/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0024
Epoch 85/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0024
Epoch 86/150
19/19 [==============================] - 1s 44ms/step - loss: 0.0024
Epoch 87/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0023
Epoch 88/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0021
Epoch 89/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0024
Epoch 90/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0029
```

```
Epoch 91/150
19/19 [==============================] - 1s 47ms/step - loss: 0.0024
Epoch 92/150
19/19 [==============================] - 1s 44ms/step - loss: 0.0025
Epoch 93/150
19/19 [==============================] - 1s 47ms/step - loss: 0.0020
Epoch 94/150
19/19 [==============================] - 1s 47ms/step - loss: 0.0022
Epoch 95/150
19/19 [==============================] - 1s 47ms/step - loss: 0.0022
Epoch 96/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0025
Epoch 97/150
19/19 [==============================] - 1s 46ms/step - loss: 0.0019
Epoch 98/150
19/19 [==============================] - 1s 44ms/step - loss: 0.0022
Epoch 99/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0022
Epoch 100/150
19/19 [==============================] - 1s 47ms/step - loss: 0.0021
Epoch 101/150
19/19 [==============================] - 1s 48ms/step - loss: 0.0018
Epoch 102/150
19/19 [==============================] - 1s 47ms/step - loss: 0.0026
Epoch 103/150
19/19 [==============================] - 1s 47ms/step - loss: 0.0029
Epoch 104/150
19/19 [==============================] - 1s 46ms/step - loss: 0.0021
Epoch 105/150
19/19 [==============================] - 1s 46ms/step - loss: 0.0019
Epoch 106/150
19/19 [==============================] - 1s 44ms/step - loss: 0.0020
Epoch 107/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0021
Epoch 108/150
19/19 [==============================] - 1s 52ms/step - loss: 0.0022
Epoch 109/150
19/19 [==============================] - 1s 48ms/step - loss: 0.0020
Epoch 110/150
19/19 [==============================] - 1s 46ms/step - loss: 0.0019
Epoch 111/150
19/19 [==============================] - 1s 47ms/step - loss: 0.0021
Epoch 112/150
19/19 [==============================] - 1s 44ms/step - loss: 0.0025
Epoch 113/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0019
Epoch 114/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0022
Epoch 115/150
19/19 [==============================] - 1s 46ms/step - loss: 0.0019
Epoch 116/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0020
Epoch 117/150
19/19 [==============================] - 1s 44ms/step - loss: 0.0017
Epoch 118/150
19/19 [==============================] - 1s 43ms/step - loss: 0.0021
Epoch 119/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0020
Epoch 120/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0018
```

```
Epoch 121/150
19/19 [==============================] - 1s 44ms/step - loss: 0.0017
Epoch 122/150
19/19 [==============================] - 1s 44ms/step - loss: 0.0020
Epoch 123/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0018
Epoch 124/150
19/19 [==============================] - 1s 44ms/step - loss: 0.0017
Epoch 125/150
19/19 [==============================] - 1s 44ms/step - loss: 0.0019
Epoch 126/150
19/19 [==============================] - 1s 44ms/step - loss: 0.0020
Epoch 127/150
19/19 [==============================] - 1s 48ms/step - loss: 0.0020
Epoch 128/150
19/19 [==============================] - 1s 46ms/step - loss: 0.0019
Epoch 129/150
19/19 [==============================] - 1s 48ms/step - loss: 0.0019
Epoch 130/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0022
Epoch 131/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0018
Epoch 132/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0017
Epoch 133/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0022
Epoch 134/150
19/19 [==============================] - 1s 44ms/step - loss: 0.0019
Epoch 135/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0017
Epoch 136/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0019
Epoch 137/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0017
Epoch 138/150
19/19 [==============================] - 1s 44ms/step - loss: 0.0018
Epoch 139/150
19/19 [==============================] - 1s 44ms/step - loss: 0.0020
Epoch 140/150
19/19 [==============================] - 1s 44ms/step - loss: 0.0016
Epoch 141/150
19/19 [==============================] - 1s 44ms/step - loss: 0.0018
Epoch 142/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0022
Epoch 143/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0018
Epoch 144/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0024
Epoch 145/150
19/19 [==============================] - 1s 46ms/step - loss: 0.0019
Epoch 146/150
19/19 [==============================] - 1s 47ms/step - loss: 0.0017
Epoch 147/150
19/19 [==============================] - 1s 46ms/step - loss: 0.0019
Epoch 148/150
19/19 [==============================] - 1s 47ms/step - loss: 0.0017
Epoch 149/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0016
Epoch 150/150
19/19 [==============================] - 1s 45ms/step - loss: 0.0018
```

PREDICTIONS

In [17]:
```
input_data=new_df[len(new_df)-len(test_df)-60:].values
input_data=input_data.reshape(-1,1)
input_data=scaler.transform(input_data)
```

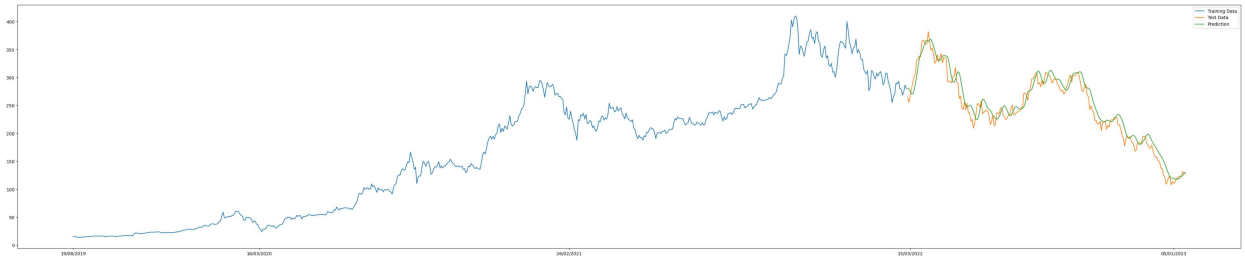
In [18]:
```
X_test=[]
for i in range(60,input_data.shape[0]):
    X_test.append(input_data[i-60:i,0])
X_test=np.array(X_test)

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

In [19]:
```
predicted=model.predict(X_test)
predicted=scaler.inverse_transform(predicted)
```

```
7/7 [==============================] - 1s 15ms/step
```

In [20]:
```
test_df['Predictions']=predicted
```

In [21]:
```
plt.figure(figsize=(50,10))
plt.plot(train_df['Close'],label='Training Data')
plt.plot(test_df['Close'],label='Test Data')
plt.plot(test_df['Predictions'],label='Prediction')
plt.xticks(['19/08/2019','16/03/2020','26/02/2021','15/03/2022','05/01/2023'])
plt.legend()
plt.show()
```



Root Mean Square Error (RMSE), Mean Square Error (MSE) and Mean absolute Error (MAE) are a standard way to measure the error of a model in predicting quantitative data.

In [22]:
```
print('The Mean Squared Error is',mean_squared_error(test_df['Close'].values,test_df['
print('The Mean Absolute Error is',mean_absolute_error(test_df['Close'].values,test_df
print('The Root Mean Squared Error is',np.sqrt(mean_squared_error(test_df['Close'].val
```

```
The Mean Squared Error is 240.19939782019537
The Mean Absolute Error is 12.484249478615087
The Root Mean Squared Error is 15.49836758565867
```