In [1]:

```python
import yfinance as yf
# Remote data access for pandas
import pandas_datareader as webreader
# Mathematical functions
import math
# Fundamental package for scientific computing with Python
import numpy as np
# Additional functions for analysing and manipulating data
import pandas as pd
# Date Functions
from datetime import date, timedelta
# This function adds plotting functions for calender dates
from pandas.plotting import register_matplotlib_converters
# Important package for visualization - we use this to plot the market data
import matplotlib.pyplot as plt
# Formatting dates
import matplotlib.dates as mdates
# Packages for measuring model performance / errors
from sklearn.metrics import mean_absolute_error, mean_squared_error
# Tools for predictive data analysis. We will use the MinMaxScaler to normalize the price data
from sklearn.preprocessing import MinMaxScaler
# Deep learning library, used for neural networks
from keras.models import Sequential
# Deep learning classes for recurrent and regular densely-connected layers
from keras.layers import LSTM, Dense
import os
from bs4 import BeautifulSoup
from urllib.request import urlopen, Request
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import plotly.express as px
from IPython.display import display
import plotly.graph_objects as go
# Setting the timeframe for the data extraction
today = date.today()
date_today = today.strftime("%Y-%m-%d")
date_start = '2015-01-01'

# Getting Stock Details
stockname = input('Enter Stock name : ')
symbol = input('Enter Stock symbol : ')
days = int(input('Enter the number of days value you wish to predict [10  20  30] :'))
#look_back = 90
df = yf.Ticker(symbol)
df = df.history(start=date_start,  end=today)
df = pd.DataFrame(df)
df = df.drop(['Dividends', 'Stock Splits'], axis = 1)
# Taking a look at the shape of the dataset
#print(df.shape)
print('Latest 5 days values of',stockname)
display(df.tail(5))

def opening(df,days):
#Plotting close price to dates
    x = df.index
    y = df['Open']
    fig = px.line(df, x=x, y=y, title='Opening Price of '+stockname)
    fig.show()

    # Splitting the Data
    # Create a new dataframe with only the Close column and convert to numpy array
    data = df.filter(['Open'])
    npdataset = data.values

    # Get the number of rows to train the model on 75% of the data
    training_data_length = math.ceil(len(npdataset) * 0.75)

    # Transform features by scaling each feature to a range between 0 and 1
    mmscaler = MinMaxScaler(feature_range=(0, 1))
    scaled_data = mmscaler.fit_transform(npdataset)
    #print(scaled_data)
    train_data = scaled_data[0:training_data_length, :]
    #print(len(train_data))

    # Creating the input shape.
    # Create a scaled training data set
    train_data = scaled_data[0:training_data_length, :]

    # Split the data into x_train and y_train data sets
    x_train = []
    y_train = []
    trainingdatasize = len(train_data)
    for i in range(100, trainingdatasize):
        x_train.append(train_data[i-100: i, 0]) #contains 100 values 0-100
        y_train.append(train_data[i, 0]) #contains all other values

    # Convert the x_train and y_train to numpy arrays
    a = x_train
    b = y_train
    x_train = np.array(x_train)
    y_train = np.array(y_train)

    # Reshape the data
    x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
    #print(x_train.shape)
    #print(y_train.shape)

    # Configure the neural network model
    model = Sequential()

    # Model with 100 Neurons - inputshape = 100 Timestamps
    model.add(LSTM(100, return_sequences=True, input_shape=(x_train.shape[1], 1)))
    model.add(LSTM(100, return_sequences=False))
    model.add(Dense(100, activation='relu'))
    model.add(Dense(1))

    # Compile the model
    model.compile(optimizer='adam', loss='mean_squared_error')

    # Training the model
    #model.fit(x_train, y_train, epochs=60)
    lst=[]
    #model.fit(x_train, y_train, epochs=10)
    history = model.fit(x_train, y_train, epochs=100)
    lst=list(history.history['loss'])

    from statistics import mean
    all_acc_of_epochs = []
    for i in range(len(lst)):
        acc = 100-lst[i]
        #print("Total accuracy for %s epoch is %s" % (i+1, acc))
        all_acc_of_epochs.append(acc)
    #print(all_acc_of_epochs)
    avg_of_all_epochs = mean(all_acc_of_epochs)
    print('Total average accuracy of all epochs :',avg_of_all_epochs)

    # Create a new array containing scaled test values
    test_data = scaled_data[training_data_length - 100:, :]

    # Create the data sets x_test and y_test
    x_test = []
    y_test = npdataset[training_data_length:, :]
    for i in range(100, len(test_data)):
        x_test.append(test_data[i-100:i, 0])

    # Convert the data to a numpy array
    x_test = np.array(x_test)

    # Reshape the data, so that we get an array with multiple test datasets
    x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
    #print(len(x_test))

    #Get the predicted values and inverse the scaling
    predictions = model.predict(x_test)
    predictions = mmscaler.inverse_transform(predictions)
    #print(predictions)
    #print(len(predictions))

    # Calculate the mean absolute error (MAE)
    mae = mean_absolute_error(predictions, y_test)
    print('MAE: ' + str(round(mae, 1)))

    mse = mean_squared_error(predictions,y_test)
    print('MSE: ' + str(round(mse, 1)))

    # Calculate the root mean squarred error (RMSE)
    rmse = np.sqrt(mse)
    print('RMSE: ' + str(round(rmse, 1)))



    # The date from which on the date is displayed
    display_start_date = "2018-01-01"

    # Add the difference between the valid and predicted prices
    train = data[:training_data_length + 1]
    valid = data[training_data_length:]
    valid.insert(1, "Predictions", predictions, True)
    valid.insert(1, "Difference", valid["Predictions"] - valid["Open"], True)

    # Zoom in to a closer timeframe
    valid = valid[valid.index > display_start_date]
    train = train[train.index > display_start_date]

    xt = train.index
    yt = train[["Open"]]
    xv = valid.index
    yv = valid[["Open", "Predictions"]]


    fig = px.line(df,x=x, y=y, title = 'Predictied value v/s actual value of Opening Price '+stockname)
    fig.add_scatter(x=xv, y=yv['Predictions'],name = 'Predicted Values')
    fig.show()

    fut_pred = df
    open_data = fut_pred.tail(100)
    open_data = open_data['Open'].values
    open_data = open_data.reshape((-1))
    #print(open_data)
    if(days == 10):
        look_back = 50
    elif(days == 20):
        look_back = 70
    elif(days == 30):
        look_back = 90
    num_prediction = days
    forecast = predict(num_prediction, model,open_data,look_back)
    forecast_dates = predict_dates(num_prediction,fut_pred)
    f = pd.DataFrame(forecast)
    f_d = pd.DataFrame(forecast_dates)
    if(days == 10):
        f = f.head(11)
        f_d = f_d.head(11)
    elif(days == 20):
        f = f.head(21)
        f_d = f_d.head(21)
    elif(days == 30):
        f = f.head(31)
        f_d = f_d.head(31)


    df_1 = df.index

    # Printing the predicted values of next 10 days
    f.columns = ['Open']
    f_d.columns = ['Date']

    fp = pd.concat([f_d, f], axis=1)
    fp = pd.DataFrame(fp)

    fp = fp.set_index('Date')

    a = pd.DataFrame()
    b = pd.DataFrame()

    a = data.tail(10)
    b = fp

    finals = a.append(b)

    df_1 = df.index


    final = finals.values.tolist()

    #Analysis for future predicttions
    increase = 0
    decrease = 0
    neutral = 0
    for i, j in enumerate(final[:-1]):
        if j < final[i+1]:
            increase = increase + 1
        if j > final[i+1]:
            decrease = decrease + 1
        if j == final[i+1]:
            neutral = neutral + 1
    #print(increase)

    Faith_of_Open = -1
    if(increase>decrease and increase>neutral):
        print('Stock Faith is Positive on Opening Price.')
        Faith_of_Open = 1
    elif(decrease>increase and decrease>neutral):
        print('Stock Faith is Negative on Opening Price.')
        Faith_of_Open = -1
    else:
        print('Stock Faith is Neutral on Opening Price.')
        #Faith_of_Open = 0
    return finals,Faith_of_Open

def closing(df,days):
    x = df.index
    y = df['Close']
    fig = px.line(df, x=x, y=y, title='Closing Price of '+stockname)
    fig.show()

    # Splitting the Data
    # Create a new dataframe with only the Close column and convert to numpy array
    data = df.filter(['Close'])
    npdataset = data.values
```

```python
    # Get the number of rows to train the model on 75% of the data
    training_data_length = math.ceil(len(npdataset) * 0.75)

    # Transform features by scaling each feature to a range between 0 and 1
    mmscaler = MinMaxScaler(feature_range=(0, 1))
    scaled_data = mmscaler.fit_transform(npdataset)
    #print(scaled_data)
    train_data = scaled_data[0:training_data_length, :]
    #print(len(train_data))

    # Creating the input shape.
    # Create a scaled training data set
    train_data = scaled_data[0:training_data_length, :]

    # Split the data into x_train and y_train data sets
    x_train = []
    y_train = []
    trainingdatasize = len(train_data)
    for i in range(100, trainingdatasize):
        x_train.append(train_data[i-100: i, 0]) #contains 100 values 0-100
        y_train.append(train_data[i, 0]) #contains all other values

    # Convert the x_train and y_train to numpy arrays
    a = x_train
    b = y_train
    x_train = np.array(x_train)
    y_train = np.array(y_train)

    # Reshape the data
    x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
    #print(x_train.shape)
    #print(y_train.shape)

    # Configure the neural network model
    model = Sequential()

    # Model with 100 Neurons - inputshape = 100 Timestamps
    model.add(LSTM(100, return_sequences=True, input_shape=(x_train.shape[1], 1)))
    model.add(LSTM(100, return_sequences=False))
    model.add(Dense(100, activation='relu'))
    model.add(Dense(1))

    # Compile the model
    model.compile(optimizer='adam', loss='mean_squared_error')

    # Training the model
    #model.fit(x_train, y_train, epochs=60)
    lst=[]
    #model.fit(x_train, y_train, epochs=10)
    history = model.fit(x_train, y_train, epochs=100)
    lst=list(history.history['loss'])

    from statistics import mean
    all_acc_of_epochs = []
    for i in range(len(lst)):
        acc = 100-lst[i]
        #print("Total accuracy for %s epoch is %s" % (i+1, acc))
        all_acc_of_epochs.append(acc)
    #print(all_acc_of_epochs)
    avg_of_all_epochs = mean(all_acc_of_epochs)
    print('Total average accuracy of all epochs :',avg_of_all_epochs)

    # Create a new array containing scaled test values
    test_data = scaled_data[training_data_length - 100:, :]

    # Create the data sets x_test and y_test
    x_test = []
    y_test = npdataset[training_data_length:, :]
    for i in range(100, len(test_data)):
        x_test.append(test_data[i-100:i, 0])

    # Convert the data to a numpy array
    x_test = np.array(x_test)

    # Reshape the data, so that we get an array with multiple test datasets
    x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
    #print(len(x_test))

    #Get the predicted values and inverse the scaling
    predictions = model.predict(x_test)
    predictions = mmscaler.inverse_transform(predictions)
    #print(predictions)
    #print(len(predictions))

    # Calculate the mean absolute error (MAE)
    mae = mean_absolute_error(predictions, y_test)
    print('MAE: ' + str(round(mae, 1)))

    mse = mean_squared_error(predictions,y_test)
    print('MSE: ' + str(round(mse, 1)))

    # Calculate the root mean squarred error (RMSE)
    rmse = np.sqrt(mse)
    print('RMSE: ' + str(round(rmse, 1)))



    # The date from which on the date is displayed
    display_start_date = "2018-01-01"

    # Add the difference between the valid and predicted prices
    train = data[:training_data_length + 1]
    valid = data[training_data_length:]
    valid.insert(1, "Predictions", predictions, True)
    valid.insert(1, "Difference", valid["Predictions"] - valid["Close"], True)

    # Zoom in to a closer timeframe
    valid = valid[valid.index > display_start_date]
    train = train[train.index > display_start_date]

    # Visualize the data
   # fig, ax1 = plt.subplots(figsize=(22, 8), sharex=True)
    xt = train.index
    yt = train[["Close"]]
    xv = valid.index
    yv = valid[["Close", "Predictions"]]


    fig = px.line(df,x=x, y=y, title = 'Predictied value v/s actual value of Closing Price '+stockname)
    fig.add_scatter(x=xv, y=yv['Predictions'],name = 'Predicted Values')
    fig.show()

    fut_pred = df
    close_data = fut_pred.tail(100)
    close_data = close_data['Close'].values
    close_data = close_data.reshape((-1))
    #print(close_data)
    if(days == 10):
        look_back = 50
    elif(days == 20):
        look_back = 70
    elif(days == 30):
        look_back = 90
    num_prediction = days
    forecast = predict(num_prediction, model,close_data,look_back)
    forecast_dates = predict_dates(num_prediction,fut_pred)
    f = pd.DataFrame(forecast)
    f_d = pd.DataFrame(forecast_dates)
    if(days == 10):
        f = f.head(11)
        f_d = f_d.head(11)
    elif(days == 20):
        f = f.head(21)
        f_d = f_d.head(21)
    elif(days == 30):
        f = f.head(31)
        f_d = f_d.head(31)

    df_1 = df.index
    f.columns = ['Close']
    f_d.columns = ['Date']

    fp = pd.concat([f_d, f], axis=1)
    fp = pd.DataFrame(fp)

    fp = fp.set_index('Date')

    a = pd.DataFrame()
    b = pd.DataFrame()

    a = data.tail(10)
    b = fp

    finals = a.append(b)

    final = finals.values.tolist()

    #Analysis for future predicttions
    increase = 0
    decrease = 0
    neutral = 0
    for i, j in enumerate(final[:-1]):
        if j < final[i+1]:
            increase = increase + 1
        if j > final[i+1]:
            decrease = decrease + 1
        if j == final[i+1]:
            neutral = neutral + 1
    #print(increase)

    Faith_of_Close = -1
    if(increase>decrease and increase>neutral):
        print('Stock Faith is Positive on Closing Price.')
        Faith_of_Close = 1
    elif(decrease>increase and decrease>neutral):
        print('Stock Faith is Negative on Closing Price.')
        Faith_of_Close = -1
    else:
        print('Stock Faith is Neutral on Closing Price.')
        #Faith_of_Close = 0
    return finals,Faith_of_Close

def high(df,days):
#Plotting close price to dates
    x = df.index
    y = df['High']
    fig = px.line(df, x=x, y=y, title='Everyday High of '+stockname)
    fig.show()

    # Splitting the Data
    # Create a new dataframe with only the Close column and convert to numpy array
    data = df.filter(['High'])
    npdataset = data.values

    # Get the number of rows to train the model on 75% of the data
    training_data_length = math.ceil(len(npdataset) * 0.75)

    # Transform features by scaling each feature to a range between 0 and 1
    mmscaler = MinMaxScaler(feature_range=(0, 1))
    scaled_data = mmscaler.fit_transform(npdataset)
    #print(scaled_data)
    train_data = scaled_data[0:training_data_length, :]
    #print(len(train_data))

    # Creating the input shape.
    # Create a scaled training data set
    train_data = scaled_data[0:training_data_length, :]

    # Split the data into x_train and y_train data sets
    x_train = []
    y_train = []
    trainingdatasize = len(train_data)
    for i in range(100, trainingdatasize):
        x_train.append(train_data[i-100: i, 0]) #contains 100 values 0-100
        y_train.append(train_data[i, 0]) #contains all other values

    # Convert the x_train and y_train to numpy arrays
    a = x_train
    b = y_train
    x_train = np.array(x_train)
    y_train = np.array(y_train)

    # Reshape the data
    x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
    #print(x_train.shape)
    #print(y_train.shape)

    # Configure the neural network model
    model = Sequential()

    # Model with 100 Neurons - inputshape = 100 Timestamps
    model.add(LSTM(100, return_sequences=True, input_shape=(x_train.shape[1], 1)))
    model.add(LSTM(100, return_sequences=False))
    model.add(Dense(100, activation='relu'))
    model.add(Dense(1))

    # Compile the model
    model.compile(optimizer='adam', loss='mean_squared_error')

    # Training the model
    #model.fit(x_train, y_train, epochs=60)
    lst=[]
    #model.fit(x_train, y_train, epochs=10)
    history = model.fit(x_train, y_train, epochs=100)
    lst=list(history.history['loss'])

    from statistics import mean
    all_acc_of_epochs = []
    for i in range(len(lst)):
        acc = 100-lst[i]
        #print("Total accuracy for %s epoch is %s" % (i+1, acc))
        all_acc_of_epochs.append(acc)
    #print(all_acc_of_epochs)
    avg_of_all_epochs = mean(all_acc_of_epochs)
    print('Total average accuracy of all epochs :',avg_of_all_epochs)

    # Create a new array containing scaled test values
    test_data = scaled_data[training_data_length - 100:, :]
```

```python
    # Create the data sets x_test and y_test
    x_test = []
    y_test = npdataset[training_data_length:, :]
    for i in range(100, len(test_data)):
        x_test.append(test_data[i-100:i, 0])

    # Convert the data to a numpy array
    x_test = np.array(x_test)

    # Reshape the data, so that we get an array with multiple test datasets
    x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
    #print(len(x_test))

    #Get the predicted values and inverse the scaling
    predictions = model.predict(x_test)
    predictions = mmscaler.inverse_transform(predictions)
    #print(predictions)
    #print(len(predictions))

    # Calculate the mean absolute error (MAE)
    mae = mean_absolute_error(predictions, y_test)
    print('MAE: ' + str(round(mae, 1)))

    mse = mean_squared_error(predictions,y_test)
    print('MSE: ' + str(round(mse, 1)))

    # Calculate the root mean squarred error (RMSE)
    rmse = np.sqrt(mse)
    print('RMSE: ' + str(round(rmse, 1)))




    # The date from which on the date is displayed
    display_start_date = "2018-01-01"

    # Add the difference between the valid and predicted prices
    train = data[:training_data_length + 1]
    valid = data[training_data_length:]
    valid.insert(1, "Predictions", predictions, True)
    valid.insert(1, "Difference", valid["Predictions"] - valid["High"], True)

    # Zoom in to a closer timeframe
    valid = valid[valid.index > display_start_date]
    train = train[train.index > display_start_date]

    # Visualize the data
    #fig, ax1 = plt.subplots(figsize=(22, 8), sharex=True)
    xt = train.index
    yt = train[["High"]]
    xv = valid.index
    yv = valid[["High", "Predictions"]]
    #plt.title("Predictions vs Ground Truth", fontsize=20)
    #plt.ylabel(stockname, fontsize=18)
    #plt.plot(yt, color="red", linewidth=2.0)
    #plt.plot(yv["Predictions"], color="blue", linewidth=2.0)
    #plt.plot(yv["Open"], color="black", linewidth=2.0)
    #plt.legend(["Train", "Train Predictions", "Ground Truth"], loc="upper left")

    #plt.show()

    fig = px.line(df,x=x, y=y, title = 'Predictied value v/s actual value of everyday High '+stockname)
    fig.add_scatter(x=xv, y=yv['Predictions'],name = 'Predicted Values')
    fig.show()

    fut_pred = df
    high_data = fut_pred.tail(100)
    high_data = high_data['High'].values
    high_data = high_data.reshape((-1))
    #print(high_data)
    if(days == 10):
        look_back = 50
    elif(days == 20):
        look_back = 70
    elif(days == 30):
        look_back = 90
    num_prediction = days
    forecast = predict(num_prediction, model,high_data,look_back)
    forecast_dates = predict_dates(num_prediction,fut_pred)
    f = pd.DataFrame(forecast)
    f_d = pd.DataFrame(forecast_dates)
    if(days == 10):
        f = f.head(11)
        f_d = f_d.head(11)
    elif(days == 20):
        f = f.head(21)
        f_d = f_d.head(21)
    elif(days == 30):
        f = f.head(31)
        f_d = f_d.head(31)

    df_1 = df.index


    # Printing the predicted values of next 10 days
    f.columns = ['High']
    f_d.columns = ['Date']

    fp = pd.concat([f_d, f], axis=1)
    fp = pd.DataFrame(fp)

    fp = fp.set_index('Date')

    a = pd.DataFrame()
    b = pd.DataFrame()

    a = data.tail(10)
    b = fp

    finals = a.append(b)

    final = finals.values.tolist()


    #final

    #Analysis for future predicttions
    increase = 0
    decrease = 0
    neutral = 0
    for i, j in enumerate(final[:-1]):
        if j < final[i+1]:
            increase = increase + 1
        if j > final[i+1]:
            decrease = decrease + 1
        if j == final[i+1]:
            neutral = neutral + 1
    #print(increase)

    Faith_of_High = -1
    if(increase>decrease and increase>neutral):
        print('Stock Faith is Positive on everyday High Price.')
        Faith_of_High = 1
    elif(decrease>increase and decrease>neutral):
        print('Stock Faith is Negative on everyday High Price.')
        Faith_of_High = -1
    else:
        print('Stock Faith is Neutral on everyday High Price.')
        #Faith_of_Open = 0
    return finals,Faith_of_High

def low(df,days):
#Plotting close price to dates
    x = df.index
    y = df['Low']
    fig = px.line(df, x=x, y=y, title='Everyday Low of '+stockname)
    fig.show()

    # Splitting the Data
    # Create a new dataframe with only the Close column and convert to numpy array
    data = df.filter(['Low'])
    npdataset = data.values

    # Get the number of rows to train the model on 75% of the data
    training_data_length = math.ceil(len(npdataset) * 0.75)

    # Transform features by scaling each feature to a range between 0 and 1
    mmscaler = MinMaxScaler(feature_range=(0, 1))
    scaled_data = mmscaler.fit_transform(npdataset)
    #print(scaled_data)
    train_data = scaled_data[0:training_data_length, :]
    #print(len(train_data))

    # Creating the input shape.
    # Create a scaled training data set
    train_data = scaled_data[0:training_data_length, :]

    # Split the data into x_train and y_train data sets
    x_train = []
    y_train = []
    trainingdatasize = len(train_data)
    for i in range(100, trainingdatasize):
        x_train.append(train_data[i-100: i, 0]) #contains 100 values 0-100
        y_train.append(train_data[i, 0]) #contains all other values

    # Convert the x_train and y_train to numpy arrays
    a = x_train
    b = y_train
    x_train = np.array(x_train)
    y_train = np.array(y_train)

    # Reshape the data
    x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
    #print(x_train.shape)
    #print(y_train.shape)

    # Configure the neural network model
    model = Sequential()

    # Model with 100 Neurons - inputshape = 100 Timestamps
    model.add(LSTM(100, return_sequences=True, input_shape=(x_train.shape[1], 1)))
    model.add(LSTM(100, return_sequences=False))
    model.add(Dense(100, activation='relu'))
    model.add(Dense(1))

    # Compile the model
    model.compile(optimizer='adam', loss='mean_squared_error')

    # Training the model
    #model.fit(x_train, y_train, epochs=60)
    lst=[]
    #model.fit(x_train, y_train, epochs=10)
    history = model.fit(x_train, y_train, epochs=100)
    lst=list(history.history['loss'])

    from statistics import mean
    all_acc_of_epochs = []
    for i in range(len(lst)):
        acc = 100-lst[i]
        #print("Total accuracy for %s epoch is %s" % (i+1, acc))
        all_acc_of_epochs.append(acc)
    #print(all_acc_of_epochs)
    avg_of_all_epochs = mean(all_acc_of_epochs)
    print('Total average accuracy of all epochs :',avg_of_all_epochs)

    # Create a new array containing scaled test values
    test_data = scaled_data[training_data_length - 100:, :]

    # Create the data sets x_test and y_test
    x_test = []
    y_test = npdataset[training_data_length:, :]
    for i in range(100, len(test_data)):
        x_test.append(test_data[i-100:i, 0])

    # Convert the data to a numpy array
    x_test = np.array(x_test)

    # Reshape the data, so that we get an array with multiple test datasets
    x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
    #print(len(x_test))

    #Get the predicted values and inverse the scaling
    predictions = model.predict(x_test)
    predictions = mmscaler.inverse_transform(predictions)
    #print(predictions)
    #print(len(predictions))

    # Calculate the mean absolute error (MAE)
    mae = mean_absolute_error(predictions, y_test)
    print('MAE: ' + str(round(mae, 1)))

    mse = mean_squared_error(predictions,y_test)
    print('MSE: ' + str(round(mse, 1)))

    # Calculate the root mean squarred error (RMSE)
    rmse = np.sqrt(mse)
    print('RMSE: ' + str(round(rmse, 1)))




    # The date from which on the date is displayed
    display_start_date = "2018-01-01"

    # Add the difference between the valid and predicted prices
    train = data[:training_data_length + 1]
    valid = data[training_data_length:]
    valid.insert(1, "Predictions", predictions, True)
    valid.insert(1, "Difference", valid["Predictions"] - valid["Low"], True)

    # Zoom in to a closer timeframe
    valid = valid[valid.index > display_start_date]
    train = train[train.index > display_start_date]

    # Visualize the data
    #fig, ax1 = plt.subplots(figsize=(22, 8), sharex=True)
    xt = train.index
    yt = train[["Low"]]
    xv = valid.index
    yv = valid[["Low", "Predictions"]]
```

```python
    fig = px.line(df,x=x, y=y, title = 'Predictied value v/s actual value of everyday Low '+stockname)
    fig.add_scatter(x=xv, y=yv['Predictions'],name = 'Predicted Values')
    fig.show()

    fut_pred = df
    low_data = fut_pred.tail(100)
    low_data = low_data['Low'].values
    low_data = low_data.reshape((-1))
    #print(low_data)
    if(days == 10):
        look_back = 50
    elif(days == 20):
        look_back = 70
    elif(days == 30):
        look_back = 90
    num_prediction = days
    forecast = predict(num_prediction, model,low_data,look_back)
    forecast_dates = predict_dates(num_prediction,fut_pred)
    f = pd.DataFrame(forecast)
    f_d = pd.DataFrame(forecast_dates)
    if(days == 10):
        f = f.head(11)
        f_d = f_d.head(11)
    elif(days == 20):
        f = f.head(21)
        f_d = f_d.head(21)
    elif(days == 30):
        f = f.head(31)
        f_d = f_d.head(31)

    #Printing the graph of predicted price.

    df_1 = df.index

    # Printing the predicted values of next 10 days
    f.columns = ['Low']
    f_d.columns = ['Date']

    fp = pd.concat([f_d, f], axis=1)
    fp = pd.DataFrame(fp)

    fp = fp.set_index('Date')

    a = pd.DataFrame()
    b = pd.DataFrame()

    a = data.tail(10)
    b = fp

    finals = a.append(b)

    final = finals.values.tolist()


    #final

    #Analysis for future predicttions
    increase = 0
    decrease = 0
    neutral = 0
    for i, j in enumerate(final[:-1]):
        if j < final[i+1]:
            increase = increase + 1
        if j > final[i+1]:
            decrease = decrease + 1
        if j == final[i+1]:
            neutral = neutral + 1
    #print(increase)

    Faith_of_Low = -1
    if(increase>decrease and increase>neutral):
        print('Stock Faith is Positive on everyday Low Price.')
        Faith_of_Low = 1
    elif(decrease>increase and decrease>neutral):
        print('Stock Faith is Negative on everyday Low Price.')
        Faith_of_Low = -1
    else:
        print('Stock Faith is Neutral on everyday Low Price.')
        #Faith_of_Open = 0
    return finals,Faith_of_Low

def predict(num_prediction,model,data,look_back):
    prediction_list = data[-look_back:]
    #print(prediction_list)
    #print(len(prediction_list))
    for _ in range(num_prediction):
        x = prediction_list[-look_back:]
        #print(x)
        x = x.reshape((1, look_back, 1))
        out = model.predict(x)[0][0]
        #print(out)
        prediction_list = np.append(prediction_list, out)
        #print(prediction_list)
    prediction_list = prediction_list[-look_back:]
    #pediction_list = prediction_list.reshape((-1))
    return prediction_list

def predict_dates(num_prediction,fut_pred):
    last_date = fut_pred.index
    last_date = last_date.values[-1]
    prediction_dates = pd.date_range(date_today, periods=num_prediction+1).tolist()
    return prediction_dates


def senti(symbol):
    web_url = 'https://finviz.com/quote.ashx?t='

    news_tables = {}
    tickers = symbol
    url = web_url + tickers
    req = Request(url=url,headers={"User-Agent":"Chrome"})
    response = urlopen(req)
    html = BeautifulSoup(response,"html.parser")
    news_table = html.find(id='news-table')
    news_tables[tickers] = news_table

    #print(url)

    #print(news_tables)

    data = news_tables[tickers]
    data_tr = data.findAll('tr')

    for x, table_row in enumerate(data_tr):
        a_text = table_row.a.text
        td_text = table_row.td.text
        #print(a_text)
        #print(td_text)
        if x == 3:
            break

    news_list = []

    for file_name, news_table in news_tables.items():
        for i in news_table.findAll('tr'):

            text = i.a.get_text()

            date_scrape = i.td.text.split()

            if len(date_scrape) == 1:
                time = date_scrape[0]

            else:
                date = date_scrape[0]
                time = date_scrape[1]

            tick = file_name.split('_')[0]

            news_list.append([tick, date, time, text])


    #news_list

    vader = SentimentIntensityAnalyzer()

    columns = ['ticker', 'date', 'time', 'headline']

    news_df = pd.DataFrame(news_list, columns=columns)

    scores = news_df['headline'].apply(vader.polarity_scores).tolist()

    scores_df = pd.DataFrame(scores)

    news_df = news_df.join(scores_df, rsuffix='_right')

    news_df['date'] = pd.to_datetime(news_df.date).dt.date

    news_df = pd.DataFrame(news_df)

    #display(news_df)

    plt.rcParams['figure.figsize'] = [10, 6]

    mean_scores = news_df.groupby(['ticker','date']).mean()

    mean_scores = mean_scores.unstack()

    mean_scores = mean_scores.xs('compound', axis="columns").transpose()

    m = pd.DataFrame()
    m = news_df.groupby(['date']).mean()
    #m

    x = m.index
    y = m[['neg']]
    #fig = px.bar(m, x=m.index, y=["neg", "neu", "pos"], title="Sentiment Analysis on news of "+symbol)
    #fig.show()

    a = pd.DataFrame()
    a = m[['compound']]
    #print(a)

    a = a.values.tolist()

    final = a
    #print(final)
    increase = 0
    decrease = 0
    neutral = 0
    for i, j in enumerate(final[:-1]):
        if i > 0:
            increase = increase + 1
        if i < 0:
            decrease = decrease + 1
        if i == 0:
            neutral = neutral + 1
    #print(increase)

    Faith_of_Senti = -1
    if(increase>decrease and increase>neutral):
        print('\nStock Faith is Positive on analysis of news of the stock.')
        Faith_of_Senti = 1
    elif(decrease>increase and decrease>neutral):
        print('\nStock Faith is Negative on analysis of news of the stock.')
        Faith_of_Senti = -1
    else:
        print('\nStock Faith is Neutral on analysis of news of the stock.')
        #Faith_of_Senti = 0
    return m,news_df,Faith_of_Senti

Open_Val,O = opening(df,days)
Close_Val,C = closing(df,days)
High_Val,H = high(df,days)
Low_Val,L = low(df,days)
m,news,S = senti(symbol)


print('\n',stockname, 'predicted values with previous 10 days actual values :')

x = pd.concat([Open_Val, Close_Val, High_Val, Low_Val], axis=1, ignore_index=True)
x.columns = ['Open', 'Close', 'High','Low']
display(x)

import plotly.graph_objects as go

trace1 = go.Scatter(
    x = x.index,
    y = x['Open'],
    mode = 'lines',
    name = 'Open'
)
trace2 = go.Scatter(
    x = x.index,
    y = x['Close'],
    mode = 'lines',
    name = 'Close'
)
trace3 = go.Scatter(
    x = x.index,
    y = x['High'],
    mode = 'lines',
    name = 'High'
)
trace4 = go.Scatter(
    x = x.index,
    y = x['Low'],
    mode = 'lines',
    name = 'Low'
)
layout = go.Layout(
    title = stockname+' Predicted values with previous 10 days actual values.',
    xaxis = {'title' : "Date"},
    yaxis = {'title' : "Prices"})

fig = go.Figure(data=[trace1,trace2,trace3,trace4], layout=layout)
fig.show()

print('\nNews Analysis on ',stockname)
display(news.tail())
```

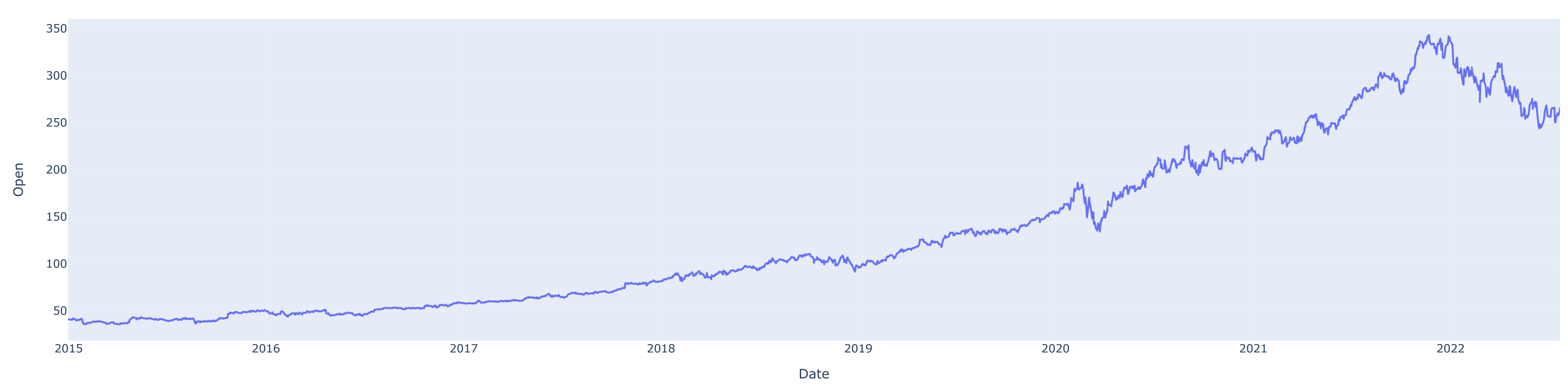
```python
fig = px.bar(m, x=m.index, y=["neg", "neu", "pos"], title="Sentiment Analysis on news of "+symbol)
fig.show()


print('\nConsidering the best possible attributes of the stock : \n\t')
print('Attributes : Opening Price, Closing Price, Everyday High, Everyday Low and News on Stock')
if(O == -1 and C == -1 and H == -1 and L == -1 and S == -1):
    print('Stock Faith is Negative, may not be a good option.')
elif(O == -1 and C == -1 and H == -1 and L == -1 and S == 1):
    print('Stock Faith is Low, news of the stock is positive.')
elif(O == -1 and C == -1 and H == -1 and L == 1 and S == -1):
    print('Stock Faith is Low, everyday Low price is good.')
elif(O == -1 and C == -1 and H == -1 and L == 1 and S == 1):
    print('Stock Faith is Low, everyday Low price and news is good.')
elif(O == -1 and C == -1 and H == 1 and L == -1 and S == -1):
    print('Stock Faith is Low, everyday High price is good.')
elif(O == -1 and C == -1 and H == 1 and L == -1 and S == 1):
    print('Stock Faith is low, everyday High price and news is good.')
elif(O == -1 and C == -1 and H == 1 and L == 1 and S == -1):
    print('Stock Faith is low, everyday High and Low price is good.')
elif(O == -1 and C == -1 and H == 1 and L == 1 and S == 1):
    print('Stock Faith is good, Opening and Closing price is low.')
elif(O == -1 and C == 1 and H == -1 and L == -1 and S == -1):
    print('Stock Faith is Low, only Closing price is good.')
elif(O == -1 and C == 1 and H == -1 and L == -1 and S == 1):
    print('Stock Faith is low, Closing price and news is good.')
elif(O == -1 and C == 1 and H == -1 and L == 1 and S == -1):
    print('Stock Faith is low, closing price and and everyday Low is good.')
elif(O == -1 and C == 1 and H == -1 and L == 1 and S == 1):
    print('Stock Faith is good, Opening price and everyday High is low.')
elif(O == -1 and C == 1 and H == 1 and L == -1 and S == -1):
    print('Stock Faith is low, Closing price and everyday High is good.')
elif(O == -1 and C == 1 and H == 1 and L == -1 and S == 1):
    print('Stock Faith is good, Opening price and everyday Low is low.')
elif(O == -1 and C == 1 and H == 1 and L == 1 and S == -1):
    print('Stock Faith is good, Opening price and news is low.')
elif(O == -1 and C == 1 and H == 1 and L == 1 and S == 1):
    print('Stock Faith is good, only Opening price is Low.')
elif(O == 1 and C == -1 and H == -1 and L == -1 and S == -1):
    print('Stock Faith is low, only Opening price is good.')
elif(O == 1 and C == -1 and H == -1 and L == -1 and S == 1):
    print('Stock Faith is low, Opening price and news is good.')
elif(O == 1 and C == -1 and H == -1 and L == 1 and S == -1):
    print('Stock Faith is low, Opening price and everyday Low is good.')
elif(O == 1 and C == -1 and H == -1 and L == 1 and S == 1):
    print('Stock Faith is good, Closing price and everyday High is low.')
elif(O == 1 and C == -1 and H == 1 and L == -1 and S == -1):
    print('Stock Faith is low, Opening price and everyday High is good.')
elif(O == 1 and C == -1 and H == 1 and L == -1 and S == 1):
    print('Stock Faith is good, Closing price and everyday Low is low.')
elif(O == 1 and C == -1 and H == 1 and L == 1 and S == -1):
    print('Stock Faith is good, Closing price and news is low.')
elif(O == 1 and C == -1 and H == 1 and L == 1 and S == 1):
    print('Stock Faith is good, only Closing price is low.')
elif(O == 1 and C == 1 and H == -1 and L == -1 and S == -1):
    print('Stock Faith is low, Opening Price and Closing price is good')
elif(O == 1 and C == 1 and H == -1 and L == -1 and S == 1):
    print('Stock Faith is good, everyday High and Low is low')
elif(O == 1 and C == 1 and H == -1 and L == 1 and S == -1):
    print('Stock Faith is good, everyday High and news is low.')
elif(O == 1 and C == 1 and H == -1 and L == 1 and S == -1):
    print('Stock Faith is good, only everyday High is low.')
elif(O == 1 and C == 1 and H == 1 and L == -1 and S == -1):
    print('Stock Faith is good, everyday Low and news is low.')
elif(O == 1 and C == 1 and H == 1 and L == -1 and S == 1):
    print('Stock Faith is good, only everyday Low is low.')
elif(O == 1 and C == 1 and H == 1 and L == 1 and S == -1):#
    print('Stock Faith is good, only news is low.')
elif(O == 1 and C == 1 and H == 1 and L == 1 and S == 1):
    print('Stock Faith is Positive, a good option.')
```

```
Enter Stock name : MICROSOFT
Enter Stock symbol : MSFT
Enter the number of days value you wish to predict [10  20  30] :20
Latest 5 days values of MICROSOFT
```

Date	Open	High	Low	Close	Volume
2022-07-18	259.750000	260.839996	253.300003	254.250000	20975000
2022-07-19	257.579987	259.720001	253.679993	259.529999	25012600
2022-07-20	259.899994	264.869995	258.910004	262.269989	22788300
2022-07-21	259.790009	264.890015	257.029999	264.839996	22404700
2022-07-22	265.239990	265.329987	259.070007	260.359985	21871000

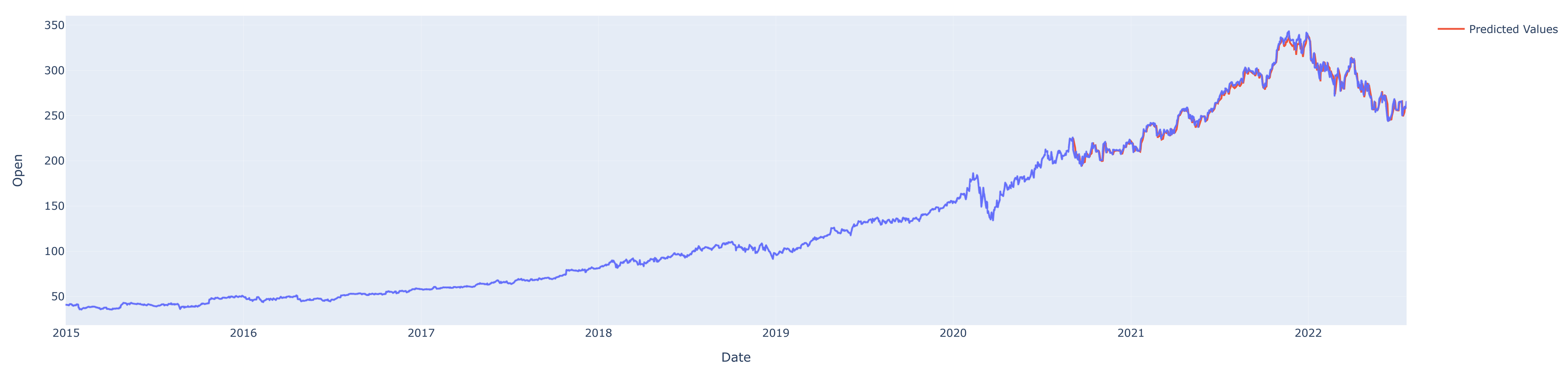
Opening Price of MICROSOFT




```
Epoch 1/100
42/42 [==============================] - 12s 189ms/step - loss: 0.0056
Epoch 2/100
42/42 [==============================] - 8s 191ms/step - loss: 1.6916e-04
Epoch 3/100
42/42 [==============================] - 8s 202ms/step - loss: 1.6457e-04
Epoch 4/100
42/42 [==============================] - 9s 206ms/step - loss: 1.6356e-04
Epoch 5/100
42/42 [==============================] - 8s 200ms/step - loss: 1.5898e-04
Epoch 6/100
42/42 [==============================] - 8s 198ms/step - loss: 1.5776e-04
Epoch 7/100
42/42 [==============================] - 8s 180ms/step - loss: 1.4841e-04
Epoch 8/100
42/42 [==============================] - 8s 186ms/step - loss: 1.3672e-04
Epoch 9/100
42/42 [==============================] - 8s 189ms/step - loss: 1.9845e-04
Epoch 10/100
42/42 [==============================] - 8s 189ms/step - loss: 1.4530e-04
Epoch 11/100
42/42 [==============================] - 8s 185ms/step - loss: 1.2822e-04
Epoch 12/100
42/42 [==============================] - 8s 192ms/step - loss: 1.4147e-04
Epoch 13/100
42/42 [==============================] - 8s 186ms/step - loss: 1.3932e-04
Epoch 14/100
42/42 [==============================] - 8s 197ms/step - loss: 1.4514e-04
Epoch 15/100
42/42 [==============================] - 9s 205ms/step - loss: 1.1562e-04
Epoch 16/100
42/42 [==============================] - 9s 207ms/step - loss: 1.1785e-04
Epoch 17/100
42/42 [==============================] - 8s 192ms/step - loss: 1.1332e-04
Epoch 18/100
42/42 [==============================] - 8s 201ms/step - loss: 9.8150e-05
Epoch 19/100
42/42 [==============================] - 8s 193ms/step - loss: 1.0916e-04
Epoch 20/100
42/42 [==============================] - 8s 189ms/step - loss: 1.3763e-04
Epoch 21/100
42/42 [==============================] - 8s 183ms/step - loss: 1.2000e-04
Epoch 22/100
42/42 [==============================] - 9s 203ms/step - loss: 1.4328e-04
Epoch 23/100
42/42 [==============================] - 8s 187ms/step - loss: 1.4589e-04
Epoch 24/100
42/42 [==============================] - 8s 191ms/step - loss: 1.8074e-04
Epoch 25/100
42/42 [==============================] - 8s 187ms/step - loss: 1.0830e-04
Epoch 26/100
42/42 [==============================] - 9s 207ms/step - loss: 8.2247e-05
Epoch 27/100
42/42 [==============================] - 9s 211ms/step - loss: 9.3229e-05
Epoch 28/100
42/42 [==============================] - 8s 186ms/step - loss: 9.1218e-05
Epoch 29/100
42/42 [==============================] - 8s 188ms/step - loss: 8.1025e-05
Epoch 30/100
42/42 [==============================] - 9s 203ms/step - loss: 8.4808e-05
Epoch 31/100
42/42 [==============================] - 8s 190ms/step - loss: 7.9464e-05
Epoch 32/100
42/42 [==============================] - 8s 190ms/step - loss: 1.1148e-04
Epoch 33/100
42/42 [==============================] - 8s 191ms/step - loss: 8.6605e-05
Epoch 34/100
42/42 [==============================] - 8s 185ms/step - loss: 7.5054e-05
Epoch 35/100
42/42 [==============================] - 7s 179ms/step - loss: 9.5086e-05
Epoch 36/100
42/42 [==============================] - 8s 193ms/step - loss: 1.4474e-04
Epoch 37/100
42/42 [==============================] - 8s 193ms/step - loss: 8.3217e-05
Epoch 38/100
42/42 [==============================] - 8s 188ms/step - loss: 7.8248e-05
Epoch 39/100
42/42 [==============================] - 8s 188ms/step - loss: 7.7240e-05
Epoch 40/100
42/42 [==============================] - 8s 193ms/step - loss: 7.5469e-05
Epoch 41/100
42/42 [==============================] - 8s 189ms/step - loss: 7.3476e-05
Epoch 42/100
42/42 [==============================] - 8s 190ms/step - loss: 9.3066e-05
Epoch 43/100
42/42 [==============================] - 8s 182ms/step - loss: 7.1840e-05
Epoch 44/100
42/42 [==============================] - 8s 183ms/step - loss: 9.7113e-05
Epoch 45/100
42/42 [==============================] - 8s 187ms/step - loss: 6.9138e-05
Epoch 46/100
42/42 [==============================] - 8s 185ms/step - loss: 7.3724e-05
Epoch 47/100
42/42 [==============================] - 8s 186ms/step - loss: 6.8410e-05
Epoch 48/100
42/42 [==============================] - 7s 173ms/step - loss: 8.0023e-05
Epoch 49/100
42/42 [==============================] - 8s 183ms/step - loss: 6.8690e-05
Epoch 50/100
42/42 [==============================] - 8s 187ms/step - loss: 6.3980e-05
Epoch 51/100
42/42 [==============================] - 8s 190ms/step - loss: 6.4426e-05
Epoch 52/100
42/42 [==============================] - 8s 184ms/step - loss: 7.1678e-05
Epoch 53/100
42/42 [==============================] - 8s 185ms/step - loss: 6.6604e-05
Epoch 54/100
42/42 [==============================] - 8s 186ms/step - loss: 6.3342e-05
Epoch 55/100
42/42 [==============================] - 8s 187ms/step - loss: 7.6216e-05
Epoch 56/100
42/42 [==============================] - 8s 188ms/step - loss: 6.9449e-05
Epoch 57/100
42/42 [==============================] - 8s 186ms/step - loss: 9.0089e-05
Epoch 58/100
42/42 [==============================] - 8s 187ms/step - loss: 6.8440e-05
Epoch 59/100
42/42 [==============================] - 8s 185ms/step - loss: 6.1494e-05
Epoch 60/100
42/42 [==============================] - 8s 190ms/step - loss: 7.4358e-05
Epoch 61/100
42/42 [==============================] - 8s 184ms/step - loss: 5.8204e-05
Epoch 62/100
42/42 [==============================] - 8s 187ms/step - loss: 7.0395e-05
Epoch 63/100
42/42 [==============================] - 8s 189ms/step - loss: 6.3114e-05
Epoch 64/100
42/42 [==============================] - 8s 189ms/step - loss: 5.4958e-05
Epoch 65/100
42/42 [==============================] - 8s 188ms/step - loss: 7.5296e-05
Epoch 66/100
42/42 [==============================] - 8s 193ms/step - loss: 6.1306e-05
Epoch 67/100
42/42 [==============================] - 8s 186ms/step - loss: 5.9203e-05
Epoch 68/100
42/42 [==============================] - 8s 188ms/step - loss: 5.7880e-05
Epoch 69/100
42/42 [==============================] - 8s 185ms/step - loss: 5.3033e-05
Epoch 70/100
42/42 [==============================] - 8s 185ms/step - loss: 5.9463e-05
Epoch 71/100
42/42 [==============================] - 8s 188ms/step - loss: 5.8429e-05
Epoch 72/100
42/42 [==============================] - 8s 189ms/step - loss: 5.1111e-05
Epoch 73/100
42/42 [==============================] - 8s 189ms/step - loss: 4.9875e-05
Epoch 74/100
42/42 [==============================] - 8s 190ms/step - loss: 5.4085e-05
Epoch 75/100
42/42 [==============================] - 8s 192ms/step - loss: 5.3318e-05
```

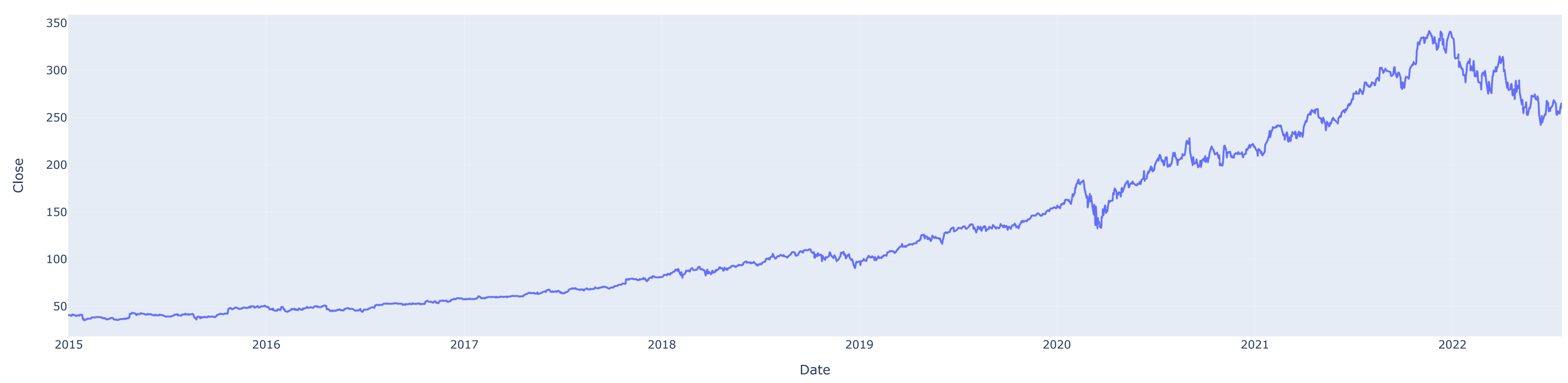
```
Epoch 76/100
42/42 [==============================] - 8s 186ms/step - loss: 6.0774e-05
Epoch 77/100
42/42 [==============================] - 8s 189ms/step - loss: 7.1290e-05
Epoch 78/100
42/42 [==============================] - 8s 190ms/step - loss: 5.4539e-05
Epoch 79/100
42/42 [==============================] - 8s 188ms/step - loss: 5.3538e-05
Epoch 80/100
42/42 [==============================] - 8s 186ms/step - loss: 5.2031e-05
Epoch 81/100
42/42 [==============================] - 8s 187ms/step - loss: 5.3947e-05
Epoch 82/100
42/42 [==============================] - 8s 188ms/step - loss: 5.1040e-05
Epoch 83/100
42/42 [==============================] - 8s 190ms/step - loss: 5.3509e-05
Epoch 84/100
42/42 [==============================] - 8s 186ms/step - loss: 6.7658e-05
Epoch 85/100
42/42 [==============================] - 8s 186ms/step - loss: 6.2591e-05
Epoch 86/100
42/42 [==============================] - 8s 191ms/step - loss: 4.5332e-05
Epoch 87/100
42/42 [==============================] - 8s 187ms/step - loss: 5.2169e-05
Epoch 88/100
42/42 [==============================] - 8s 187ms/step - loss: 5.1445e-05
Epoch 89/100
42/42 [==============================] - 8s 186ms/step - loss: 5.8752e-05
Epoch 90/100
42/42 [==============================] - 8s 187ms/step - loss: 7.0113e-05
Epoch 91/100
42/42 [==============================] - 8s 185ms/step - loss: 5.4965e-05
Epoch 92/100
42/42 [==============================] - 8s 186ms/step - loss: 5.4181e-05
Epoch 93/100
42/42 [==============================] - 8s 185ms/step - loss: 5.4092e-05
Epoch 94/100
42/42 [==============================] - 8s 188ms/step - loss: 5.8843e-05
Epoch 95/100
42/42 [==============================] - 8s 187ms/step - loss: 4.8192e-05
Epoch 96/100
42/42 [==============================] - 8s 190ms/step - loss: 5.5913e-05
Epoch 97/100
42/42 [==============================] - 8s 197ms/step - loss: 4.9275e-05
Epoch 98/100
42/42 [==============================] - 8s 192ms/step - loss: 6.5357e-05
Epoch 99/100
42/42 [==============================] - 8s 194ms/step - loss: 5.0390e-05
Epoch 100/100
42/42 [==============================] - 8s 189ms/step - loss: 5.3788e-05
Total average accuracy of all epochs : 99.99985902857745
15/15 [==============================] - 2s 73ms/step
MAE: 3.7
MSE: 23.9
RMSE: 4.9
```

Predictied value v/s actual value of Opening Price MICROSOFT




```
1/1 [==============================] - 1s 804ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 46ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 42ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 48ms/step
1/1 [==============================] - 0s 44ms/step
1/1 [==============================] - 0s 50ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 36ms/step
Stock Faith is Positive on Opening Price.
```

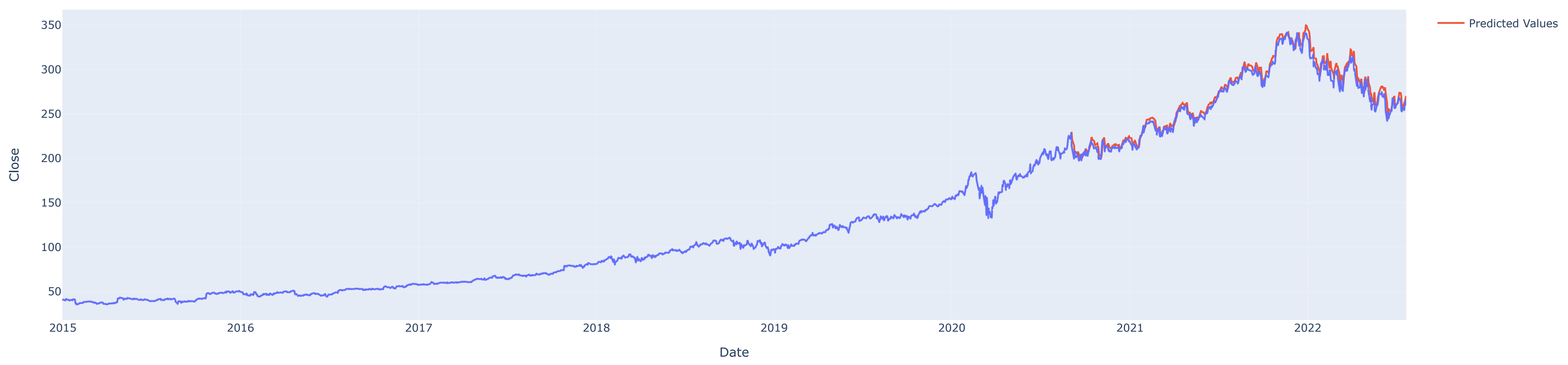
Closing Price of MICROSOFT




```
Epoch 1/100
42/42 [==============================] - 14s 242ms/step - loss: 0.0044
Epoch 2/100
42/42 [==============================] - 10s 228ms/step - loss: 2.1323e-04
Epoch 3/100
42/42 [==============================] - 9s 225ms/step - loss: 1.9823e-04
Epoch 4/100
42/42 [==============================] - 9s 219ms/step - loss: 1.9448e-04
Epoch 5/100
42/42 [==============================] - 9s 219ms/step - loss: 1.3898e-04
Epoch 6/100
42/42 [==============================] - 9s 222ms/step - loss: 1.4470e-04
Epoch 7/100
42/42 [==============================] - 9s 220ms/step - loss: 1.7471e-04
Epoch 8/100
42/42 [==============================] - 10s 228ms/step - loss: 1.2516e-04
Epoch 9/100
42/42 [==============================] - 9s 217ms/step - loss: 1.4443e-04
Epoch 10/100
42/42 [==============================] - 9s 224ms/step - loss: 1.3205e-04
Epoch 11/100
42/42 [==============================] - 9s 224ms/step - loss: 1.1892e-04
Epoch 12/100
42/42 [==============================] - 9s 225ms/step - loss: 1.5600e-04
Epoch 13/100
42/42 [==============================] - 9s 224ms/step - loss: 1.1594e-04
Epoch 14/100
42/42 [==============================] - 10s 241ms/step - loss: 1.6576e-04
Epoch 15/100
42/42 [==============================] - 9s 225ms/step - loss: 1.2722e-04
Epoch 16/100
42/42 [==============================] - 10s 231ms/step - loss: 1.1067e-04
Epoch 17/100
42/42 [==============================] - 9s 225ms/step - loss: 1.0399e-04
Epoch 18/100
42/42 [==============================] - 9s 223ms/step - loss: 1.2170e-04
Epoch 19/100
42/42 [==============================] - 9s 222ms/step - loss: 1.1656e-04
Epoch 20/100
42/42 [==============================] - 10s 228ms/step - loss: 9.7466e-05
Epoch 21/100
42/42 [==============================] - 10s 230ms/step - loss: 9.7603e-05
Epoch 22/100
42/42 [==============================] - 10s 229ms/step - loss: 9.5477e-05
Epoch 23/100
42/42 [==============================] - 10s 230ms/step - loss: 8.7531e-05
Epoch 24/100
42/42 [==============================] - 10s 244ms/step - loss: 9.5578e-05
Epoch 25/100
42/42 [==============================] - 10s 246ms/step - loss: 1.0272e-04
Epoch 26/100
42/42 [==============================] - 10s 232ms/step - loss: 1.0553e-04
Epoch 27/100
42/42 [==============================] - 10s 233ms/step - loss: 8.7664e-05
Epoch 28/100
42/42 [==============================] - 10s 231ms/step - loss: 9.3449e-05
Epoch 29/100
42/42 [==============================] - 9s 223ms/step - loss: 7.8674e-05
Epoch 30/100
42/42 [==============================] - 10s 227ms/step - loss: 1.0216e-04
Epoch 31/100
42/42 [==============================] - 9s 208ms/step - loss: 8.7176e-05
Epoch 32/100
42/42 [==============================] - 9s 220ms/step - loss: 1.0984e-04
Epoch 33/100
42/42 [==============================] - 9s 224ms/step - loss: 7.9352e-05
Epoch 34/100
42/42 [==============================] - 10s 228ms/step - loss: 6.9403e-05
Epoch 35/100
42/42 [==============================] - 9s 225ms/step - loss: 6.9852e-05
Epoch 36/100
42/42 [==============================] - 10s 232ms/step - loss: 9.5727e-05
Epoch 37/100
42/42 [==============================] - 10s 239ms/step - loss: 8.9082e-05
Epoch 38/100
42/42 [==============================] - 9s 219ms/step - loss: 7.3115e-05
Epoch 39/100
42/42 [==============================] - 10s 240ms/step - loss: 1.0271e-04
Epoch 40/100
42/42 [==============================] - 10s 231ms/step - loss: 8.5266e-05
Epoch 41/100
42/42 [==============================] - 9s 224ms/step - loss: 8.1637e-05
Epoch 42/100
42/42 [==============================] - 10s 231ms/step - loss: 7.7571e-05
Epoch 43/100
42/42 [==============================] - 9s 224ms/step - loss: 7.0327e-05
Epoch 44/100
42/42 [==============================] - 10s 234ms/step - loss: 6.4874e-05
Epoch 45/100
42/42 [==============================] - 10s 236ms/step - loss: 6.5432e-05
Epoch 46/100
42/42 [==============================] - 10s 234ms/step - loss: 9.7023e-05
Epoch 47/100
42/42 [==============================] - 10s 228ms/step - loss: 8.6336e-05
Epoch 48/100
42/42 [==============================] - 9s 225ms/step - loss: 7.3090e-05
Epoch 49/100
42/42 [==============================] - 9s 218ms/step - loss: 6.2355e-05
Epoch 50/100
42/42 [==============================] - 9s 213ms/step - loss: 6.9030e-05
Epoch 51/100
42/42 [==============================] - 9s 220ms/step - loss: 6.4164e-05
Epoch 52/100
42/42 [==============================] - 9s 225ms/step - loss: 7.6684e-05
Epoch 53/100
42/42 [==============================] - 9s 218ms/step - loss: 5.8918e-05
Epoch 54/100
42/42 [==============================] - 9s 212ms/step - loss: 6.3078e-05
Epoch 55/100
42/42 [==============================] - 9s 212ms/step - loss: 7.0986e-05
Epoch 56/100
42/42 [==============================] - 8s 201ms/step - loss: 6.0326e-05
Epoch 57/100
42/42 [==============================] - 9s 216ms/step - loss: 5.4988e-05
Epoch 58/100
42/42 [==============================] - 9s 211ms/step - loss: 5.9327e-05
Epoch 59/100
42/42 [==============================] - 9s 222ms/step - loss: 8.3830e-05
Epoch 60/100
42/42 [==============================] - 9s 218ms/step - loss: 8.3947e-05
Epoch 61/100
42/42 [==============================] - 10s 235ms/step - loss: 8.0311e-05
Epoch 62/100
42/42 [==============================] - 9s 219ms/step - loss: 6.8427e-05
Epoch 63/100
42/42 [==============================] - 9s 221ms/step - loss: 9.5728e-05
Epoch 64/100
42/42 [==============================] - 9s 223ms/step - loss: 6.7186e-05
Epoch 65/100
42/42 [==============================] - 9s 225ms/step - loss: 6.0569e-05
```

```
Epoch 66/100
42/42 [==============================] - 10s 227ms/step - loss: 7.6672e-05
Epoch 67/100
42/42 [==============================] - 9s 218ms/step - loss: 5.7099e-05
Epoch 68/100
42/42 [==============================] - 9s 223ms/step - loss: 6.8346e-05
Epoch 69/100
42/42 [==============================] - 9s 224ms/step - loss: 6.5592e-05
Epoch 70/100
42/42 [==============================] - 9s 221ms/step - loss: 5.7242e-05
Epoch 71/100
42/42 [==============================] - 10s 228ms/step - loss: 6.1154e-05
Epoch 72/100
42/42 [==============================] - 10s 233ms/step - loss: 7.7438e-05
Epoch 73/100
42/42 [==============================] - 9s 215ms/step - loss: 5.3443e-05
Epoch 74/100
42/42 [==============================] - 9s 221ms/step - loss: 5.4953e-05
Epoch 75/100
42/42 [==============================] - 9s 223ms/step - loss: 6.2939e-05
Epoch 76/100
42/42 [==============================] - 9s 220ms/step - loss: 5.9863e-05
Epoch 77/100
42/42 [==============================] - 9s 222ms/step - loss: 6.0874e-05
Epoch 78/100
42/42 [==============================] - 9s 224ms/step - loss: 4.9506e-05
Epoch 79/100
42/42 [==============================] - 9s 223ms/step - loss: 5.7186e-05
Epoch 80/100
42/42 [==============================] - 9s 219ms/step - loss: 5.6378e-05
Epoch 81/100
42/42 [==============================] - 9s 223ms/step - loss: 5.7834e-05
Epoch 82/100
42/42 [==============================] - 10s 227ms/step - loss: 6.2544e-05
Epoch 83/100
42/42 [==============================] - 9s 219ms/step - loss: 4.9105e-05
Epoch 84/100
42/42 [==============================] - 9s 221ms/step - loss: 5.4465e-05
Epoch 85/100
42/42 [==============================] - 10s 229ms/step - loss: 4.8595e-05
Epoch 86/100
42/42 [==============================] - 9s 226ms/step - loss: 4.7242e-05
Epoch 87/100
42/42 [==============================] - 10s 227ms/step - loss: 5.5808e-05
Epoch 88/100
42/42 [==============================] - 9s 222ms/step - loss: 5.7137e-05
Epoch 89/100
42/42 [==============================] - 10s 231ms/step - loss: 5.4828e-05
Epoch 90/100
42/42 [==============================] - 9s 221ms/step - loss: 6.0418e-05
Epoch 91/100
42/42 [==============================] - 9s 222ms/step - loss: 4.5601e-05
Epoch 92/100
42/42 [==============================] - 9s 225ms/step - loss: 4.9377e-05
Epoch 93/100
42/42 [==============================] - 10s 228ms/step - loss: 4.8796e-05
Epoch 94/100
42/42 [==============================] - 10s 231ms/step - loss: 5.9415e-05
Epoch 95/100
42/42 [==============================] - 10s 227ms/step - loss: 5.5140e-05
Epoch 96/100
42/42 [==============================] - 10s 238ms/step - loss: 5.5920e-05
Epoch 97/100
42/42 [==============================] - 10s 243ms/step - loss: 5.2548e-05
Epoch 98/100
42/42 [==============================] - 9s 222ms/step - loss: 5.6115e-05
Epoch 99/100
42/42 [==============================] - 9s 224ms/step - loss: 5.0527e-05
Epoch 100/100
42/42 [==============================] - 9s 220ms/step - loss: 4.9428e-05
Total average accuracy of all epochs : 99.99987350445008
15/15 [==============================] - 2s 73ms/step
MAE: 5.3
MSE: 45.9
RMSE: 6.8
```

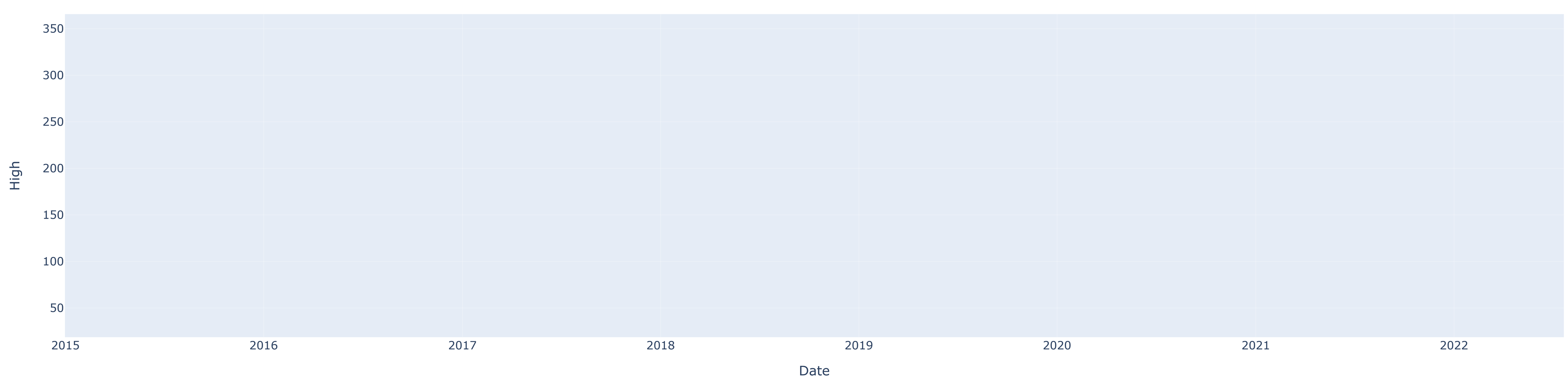
Predictied value v/s actual value of Closing Price MICROSOFT




```
1/1 [==============================] - 1s 626ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 27ms/step
Stock Faith is Neutral on Closing Price.
```

☹

Everyday High of MICROSOFT

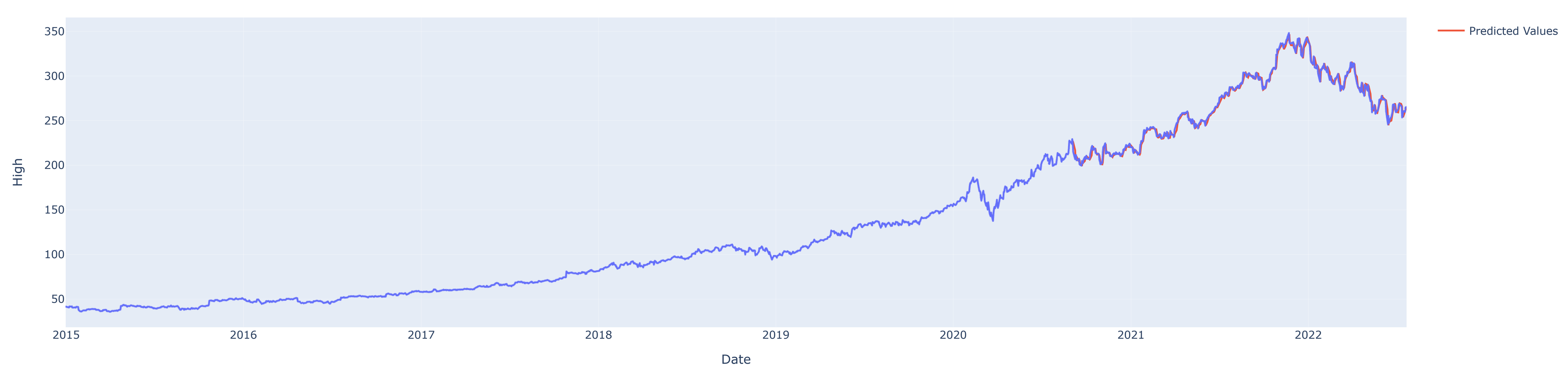



```
Epoch 1/100
42/42 [==============================] - 10s 182ms/step - loss: 0.0047
Epoch 2/100
42/42 [==============================] - 8s 184ms/step - loss: 1.6821e-04
Epoch 3/100
42/42 [==============================] - 8s 187ms/step - loss: 1.5965e-04
Epoch 4/100
42/42 [==============================] - 8s 182ms/step - loss: 1.3184e-04
Epoch 5/100
42/42 [==============================] - 8s 183ms/step - loss: 1.7270e-04
Epoch 6/100
42/42 [==============================] - 8s 183ms/step - loss: 1.2391e-04
Epoch 7/100
42/42 [==============================] - 8s 188ms/step - loss: 1.5579e-04
Epoch 8/100
42/42 [==============================] - 8s 181ms/step - loss: 1.1545e-04
Epoch 9/100
42/42 [==============================] - 8s 185ms/step - loss: 1.2185e-04
Epoch 10/100
42/42 [==============================] - 8s 193ms/step - loss: 1.1952e-04
Epoch 11/100
42/42 [==============================] - 8s 184ms/step - loss: 1.0767e-04
Epoch 12/100
42/42 [==============================] - 8s 179ms/step - loss: 1.2106e-04
Epoch 13/100
42/42 [==============================] - 9s 211ms/step - loss: 1.0807e-04
Epoch 14/100
42/42 [==============================] - 8s 192ms/step - loss: 1.0516e-04
Epoch 15/100
42/42 [==============================] - 7s 178ms/step - loss: 1.1637e-04
Epoch 16/100
42/42 [==============================] - 7s 179ms/step - loss: 8.9083e-05
Epoch 17/100
42/42 [==============================] - 7s 178ms/step - loss: 8.7709e-05
Epoch 18/100
42/42 [==============================] - 8s 186ms/step - loss: 8.7018e-05
Epoch 19/100
42/42 [==============================] - 8s 182ms/step - loss: 1.2130e-04
Epoch 20/100
42/42 [==============================] - 8s 181ms/step - loss: 9.1274e-05
Epoch 21/100
42/42 [==============================] - 8s 180ms/step - loss: 9.3297e-05
Epoch 22/100
42/42 [==============================] - 8s 182ms/step - loss: 8.9645e-05
Epoch 23/100
42/42 [==============================] - 8s 188ms/step - loss: 8.8296e-05
Epoch 24/100
42/42 [==============================] - 8s 182ms/step - loss: 1.1259e-04
Epoch 25/100
42/42 [==============================] - 8s 185ms/step - loss: 1.0515e-04
Epoch 26/100
42/42 [==============================] - 7s 175ms/step - loss: 9.6319e-05
Epoch 27/100
42/42 [==============================] - 8s 179ms/step - loss: 7.5387e-05
Epoch 28/100
42/42 [==============================] - 7s 178ms/step - loss: 8.7157e-05
Epoch 29/100
42/42 [==============================] - 7s 175ms/step - loss: 7.6326e-05
Epoch 30/100
42/42 [==============================] - 8s 180ms/step - loss: 7.5621e-05
Epoch 31/100
42/42 [==============================] - 8s 184ms/step - loss: 7.9927e-05
Epoch 32/100
42/42 [==============================] - 8s 185ms/step - loss: 7.1664e-05
Epoch 33/100
42/42 [==============================] - 8s 180ms/step - loss: 7.0830e-05
Epoch 34/100
42/42 [==============================] - 7s 177ms/step - loss: 6.6657e-05
Epoch 35/100
42/42 [==============================] - 7s 179ms/step - loss: 7.0817e-05
Epoch 36/100
42/42 [==============================] - 8s 180ms/step - loss: 8.0448e-05
Epoch 37/100
42/42 [==============================] - 7s 175ms/step - loss: 6.8620e-05
Epoch 38/100
42/42 [==============================] - 8s 181ms/step - loss: 7.5954e-05
Epoch 39/100
42/42 [==============================] - 7s 176ms/step - loss: 8.7745e-05
Epoch 40/100
42/42 [==============================] - 8s 184ms/step - loss: 6.9548e-05
Epoch 41/100
42/42 [==============================] - 8s 187ms/step - loss: 7.4258e-05
Epoch 42/100
42/42 [==============================] - 8s 180ms/step - loss: 5.7444e-05
Epoch 43/100
42/42 [==============================] - 9s 226ms/step - loss: 5.8826e-05
Epoch 44/100
42/42 [==============================] - 8s 182ms/step - loss: 6.6904e-05
Epoch 45/100
42/42 [==============================] - 8s 179ms/step - loss: 5.9498e-05
Epoch 46/100
42/42 [==============================] - 7s 178ms/step - loss: 6.4926e-05
Epoch 47/100
42/42 [==============================] - 8s 181ms/step - loss: 6.0583e-05
Epoch 48/100
42/42 [==============================] - 8s 182ms/step - loss: 5.1864e-05
Epoch 49/100
42/42 [==============================] - 8s 182ms/step - loss: 6.0581e-05
Epoch 50/100
42/42 [==============================] - 8s 180ms/step - loss: 6.4569e-05
Epoch 51/100
42/42 [==============================] - 8s 181ms/step - loss: 6.1015e-05
Epoch 52/100
42/42 [==============================] - 8s 182ms/step - loss: 6.4517e-05
Epoch 53/100
42/42 [==============================] - 7s 179ms/step - loss: 6.3167e-05
Epoch 54/100
42/42 [==============================] - 8s 183ms/step - loss: 1.2179e-04
Epoch 55/100
42/42 [==============================] - 8s 185ms/step - loss: 6.7633e-05
```

```
Epoch 56/100
42/42 [==============================] - 8s 183ms/step - loss: 5.4754e-05
Epoch 57/100
42/42 [==============================] - 8s 180ms/step - loss: 5.7463e-05
Epoch 58/100
42/42 [==============================] - 8s 182ms/step - loss: 5.0164e-05
Epoch 59/100
42/42 [==============================] - 8s 189ms/step - loss: 5.6710e-05
Epoch 60/100
42/42 [==============================] - 8s 184ms/step - loss: 7.5131e-05
Epoch 61/100
42/42 [==============================] - 8s 180ms/step - loss: 5.8695e-05
Epoch 62/100
42/42 [==============================] - 8s 181ms/step - loss: 6.1735e-05
Epoch 63/100
42/42 [==============================] - 8s 184ms/step - loss: 6.6826e-05
Epoch 64/100
42/42 [==============================] - 8s 187ms/step - loss: 6.3385e-05
Epoch 65/100
42/42 [==============================] - 8s 183ms/step - loss: 4.6227e-05
Epoch 66/100
42/42 [==============================] - 8s 184ms/step - loss: 4.4542e-05
Epoch 67/100
42/42 [==============================] - 8s 186ms/step - loss: 4.9374e-05
Epoch 68/100
42/42 [==============================] - 8s 184ms/step - loss: 4.8290e-05
Epoch 69/100
42/42 [==============================] - 8s 186ms/step - loss: 4.4288e-05
Epoch 70/100
42/42 [==============================] - 8s 184ms/step - loss: 4.7905e-05
Epoch 71/100
42/42 [==============================] - 8s 181ms/step - loss: 4.9043e-05
Epoch 72/100
42/42 [==============================] - 8s 184ms/step - loss: 4.1448e-05
Epoch 73/100
42/42 [==============================] - 8s 185ms/step - loss: 5.8779e-05
Epoch 74/100
42/42 [==============================] - 8s 183ms/step - loss: 4.3504e-05
Epoch 75/100
42/42 [==============================] - 8s 184ms/step - loss: 3.9257e-05
Epoch 76/100
42/42 [==============================] - 8s 182ms/step - loss: 5.1049e-05
Epoch 77/100
42/42 [==============================] - 8s 184ms/step - loss: 4.2007e-05
Epoch 78/100
42/42 [==============================] - 8s 184ms/step - loss: 5.1788e-05
Epoch 79/100
42/42 [==============================] - 8s 182ms/step - loss: 5.7510e-05
Epoch 80/100
42/42 [==============================] - 8s 185ms/step - loss: 7.4056e-05
Epoch 81/100
42/42 [==============================] - 8s 182ms/step - loss: 4.0933e-05
Epoch 82/100
42/42 [==============================] - 8s 183ms/step - loss: 5.7909e-05
Epoch 83/100
42/42 [==============================] - 8s 183ms/step - loss: 5.1292e-05
Epoch 84/100
42/42 [==============================] - 8s 184ms/step - loss: 4.0822e-05
Epoch 85/100
42/42 [==============================] - 8s 185ms/step - loss: 4.4556e-05
Epoch 86/100
42/42 [==============================] - 8s 183ms/step - loss: 4.6514e-05
Epoch 87/100
42/42 [==============================] - 8s 186ms/step - loss: 4.9887e-05
Epoch 88/100
42/42 [==============================] - 8s 183ms/step - loss: 3.5797e-05
Epoch 89/100
42/42 [==============================] - 8s 188ms/step - loss: 3.6901e-05
Epoch 90/100
42/42 [==============================] - 8s 187ms/step - loss: 3.8087e-05
Epoch 91/100
42/42 [==============================] - 8s 184ms/step - loss: 4.3099e-05
Epoch 92/100
42/42 [==============================] - 8s 185ms/step - loss: 3.5118e-05
Epoch 93/100
42/42 [==============================] - 8s 183ms/step - loss: 4.0996e-05
Epoch 94/100
42/42 [==============================] - 8s 186ms/step - loss: 3.9468e-05
Epoch 95/100
42/42 [==============================] - 8s 184ms/step - loss: 3.5159e-05
Epoch 96/100
42/42 [==============================] - 8s 184ms/step - loss: 4.2573e-05
Epoch 97/100
42/42 [==============================] - 8s 186ms/step - loss: 3.9683e-05
Epoch 98/100
42/42 [==============================] - 8s 184ms/step - loss: 4.2743e-05
Epoch 99/100
42/42 [==============================] - 8s 186ms/step - loss: 5.5936e-05
Epoch 100/100
42/42 [==============================] - 8s 180ms/step - loss: 5.1298e-05
Total average accuracy of all epochs : 99.99988106486303
15/15 [==============================] - 2s 71ms/step
MAE: 2.9
MSE: 15.1
RMSE: 3.9
```

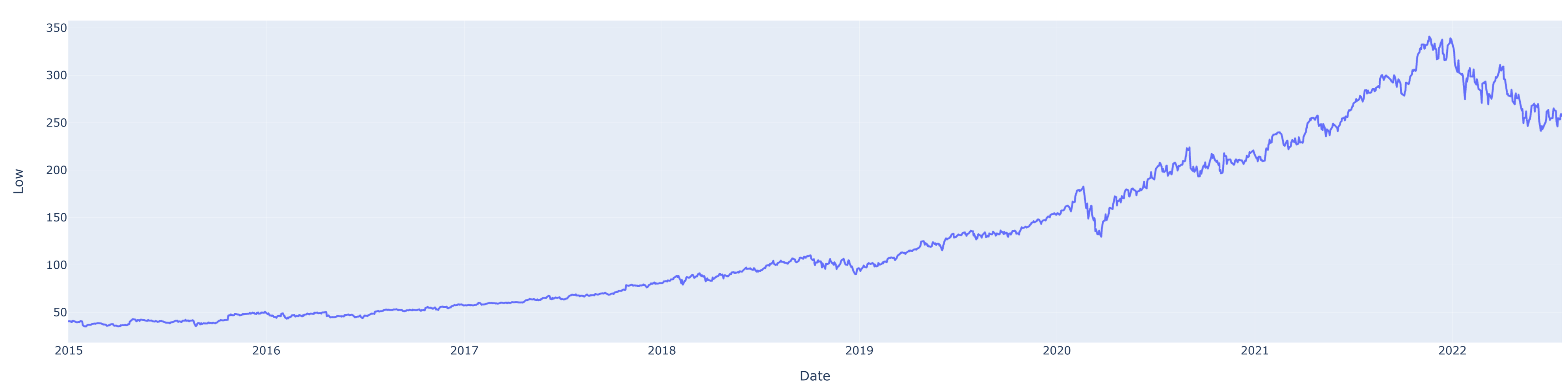
☹

Predictied value v/s actual value of everyday High MICROSOFT




```
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 32ms/step
Stock Faith is Positive on everyday High Price.
```

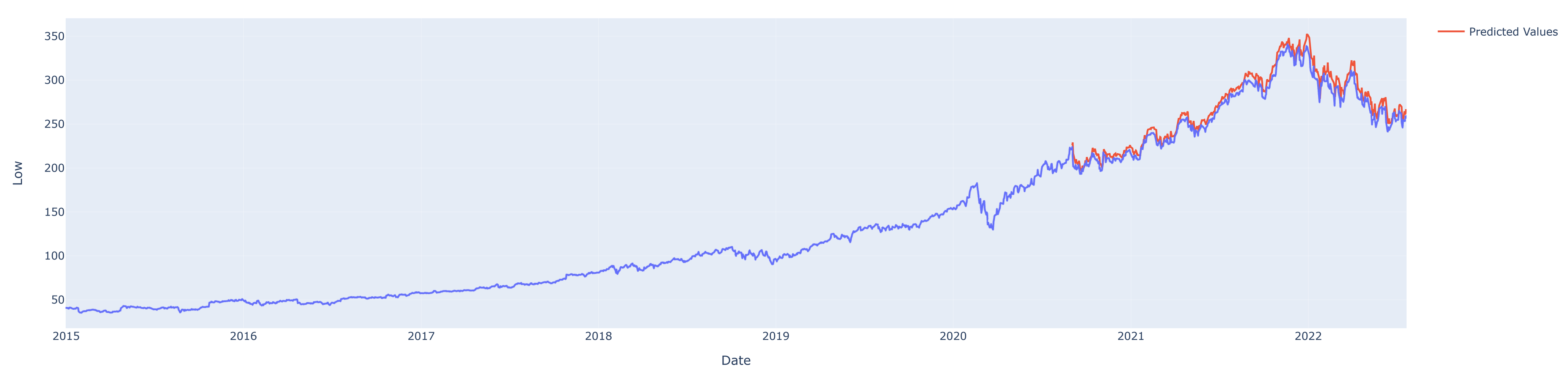
Everyday Low of MICROSOFT




```
Epoch 1/100
42/42 [==============================] - 12s 200ms/step - loss: 0.0040
Epoch 2/100
42/42 [==============================] - 8s 192ms/step - loss: 1.7535e-04
Epoch 3/100
42/42 [==============================] - 8s 193ms/step - loss: 1.6187e-04
Epoch 4/100
42/42 [==============================] - 8s 198ms/step - loss: 1.6648e-04
Epoch 5/100
42/42 [==============================] - 8s 197ms/step - loss: 1.6127e-04
Epoch 6/100
42/42 [==============================] - 8s 198ms/step - loss: 1.8220e-04
Epoch 7/100
42/42 [==============================] - 8s 198ms/step - loss: 1.9455e-04
Epoch 8/100
42/42 [==============================] - 8s 190ms/step - loss: 1.8017e-04
Epoch 9/100
42/42 [==============================] - 8s 193ms/step - loss: 1.3342e-04
Epoch 10/100
42/42 [==============================] - 8s 196ms/step - loss: 1.3216e-04
Epoch 11/100
42/42 [==============================] - 8s 193ms/step - loss: 1.3784e-04
Epoch 12/100
42/42 [==============================] - 8s 195ms/step - loss: 1.2597e-04
Epoch 13/100
42/42 [==============================] - 8s 195ms/step - loss: 1.2896e-04
Epoch 14/100
42/42 [==============================] - 8s 190ms/step - loss: 1.7351e-04
Epoch 15/100
42/42 [==============================] - 8s 188ms/step - loss: 1.3475e-04
Epoch 16/100
42/42 [==============================] - 8s 190ms/step - loss: 1.7370e-04
Epoch 17/100
42/42 [==============================] - 8s 190ms/step - loss: 1.1925e-04
Epoch 18/100
42/42 [==============================] - 8s 195ms/step - loss: 1.0831e-04
Epoch 19/100
42/42 [==============================] - 8s 197ms/step - loss: 1.3750e-04
Epoch 20/100
42/42 [==============================] - 8s 191ms/step - loss: 9.7591e-05
Epoch 21/100
42/42 [==============================] - 8s 192ms/step - loss: 9.8535e-05
Epoch 22/100
42/42 [==============================] - 8s 194ms/step - loss: 1.0154e-04
Epoch 23/100
42/42 [==============================] - 8s 191ms/step - loss: 1.6119e-04
Epoch 24/100
42/42 [==============================] - 8s 192ms/step - loss: 1.0256e-04
Epoch 25/100
42/42 [==============================] - 8s 196ms/step - loss: 8.9296e-05
Epoch 26/100
42/42 [==============================] - 8s 194ms/step - loss: 9.3124e-05
Epoch 27/100
42/42 [==============================] - 8s 193ms/step - loss: 8.4415e-05
Epoch 28/100
42/42 [==============================] - 8s 194ms/step - loss: 7.7689e-05
Epoch 29/100
42/42 [==============================] - 8s 195ms/step - loss: 7.8531e-05
Epoch 30/100
42/42 [==============================] - 8s 195ms/step - loss: 1.0971e-04
Epoch 31/100
42/42 [==============================] - 8s 192ms/step - loss: 8.1480e-05
Epoch 32/100
42/42 [==============================] - 8s 194ms/step - loss: 7.2786e-05
Epoch 33/100
42/42 [==============================] - 8s 193ms/step - loss: 7.1947e-05
Epoch 34/100
42/42 [==============================] - 8s 189ms/step - loss: 7.8270e-05
Epoch 35/100
42/42 [==============================] - 8s 192ms/step - loss: 7.3593e-05
Epoch 36/100
42/42 [==============================] - 8s 194ms/step - loss: 8.1619e-05
Epoch 37/100
42/42 [==============================] - 8s 194ms/step - loss: 8.1758e-05
Epoch 38/100
42/42 [==============================] - 8s 195ms/step - loss: 7.7941e-05
Epoch 39/100
42/42 [==============================] - 8s 191ms/step - loss: 7.3045e-05
Epoch 40/100
42/42 [==============================] - 8s 194ms/step - loss: 9.2011e-05
Epoch 41/100
42/42 [==============================] - 8s 193ms/step - loss: 9.2546e-05
Epoch 42/100
42/42 [==============================] - 8s 199ms/step - loss: 9.9104e-05
Epoch 43/100
42/42 [==============================] - 8s 200ms/step - loss: 7.2493e-05
Epoch 44/100
42/42 [==============================] - 8s 201ms/step - loss: 7.0933e-05
Epoch 45/100
42/42 [==============================] - 9s 206ms/step - loss: 6.8925e-05
```

```
Epoch 46/100
42/42 [==============================] - 9s 205ms/step - loss: 6.6934e-05
Epoch 47/100
42/42 [==============================] - 8s 202ms/step - loss: 6.4580e-05
Epoch 48/100
42/42 [==============================] - 9s 206ms/step - loss: 5.8754e-05
Epoch 49/100
42/42 [==============================] - 8s 200ms/step - loss: 5.5327e-05
Epoch 50/100
42/42 [==============================] - 9s 205ms/step - loss: 6.6704e-05
Epoch 51/100
42/42 [==============================] - 8s 200ms/step - loss: 5.8447e-05
Epoch 52/100
42/42 [==============================] - 8s 200ms/step - loss: 4.9812e-05
Epoch 53/100
42/42 [==============================] - 8s 200ms/step - loss: 5.8660e-05
Epoch 54/100
42/42 [==============================] - 9s 203ms/step - loss: 7.2761e-05
Epoch 55/100
42/42 [==============================] - 9s 221ms/step - loss: 6.7056e-05
Epoch 56/100
42/42 [==============================] - 9s 218ms/step - loss: 7.2314e-05
Epoch 57/100
42/42 [==============================] - 9s 213ms/step - loss: 5.5174e-05
Epoch 58/100
42/42 [==============================] - 10s 227ms/step - loss: 5.1655e-05
Epoch 59/100
42/42 [==============================] - 9s 216ms/step - loss: 7.4183e-05
Epoch 60/100
42/42 [==============================] - 9s 203ms/step - loss: 7.0213e-05
Epoch 61/100
42/42 [==============================] - 8s 202ms/step - loss: 5.2423e-05
Epoch 62/100
42/42 [==============================] - 9s 203ms/step - loss: 4.6163e-05
Epoch 63/100
42/42 [==============================] - 9s 208ms/step - loss: 4.5342e-05
Epoch 64/100
42/42 [==============================] - 8s 193ms/step - loss: 5.9091e-05
Epoch 65/100
42/42 [==============================] - 8s 192ms/step - loss: 4.3523e-05
Epoch 66/100
42/42 [==============================] - 9s 202ms/step - loss: 4.4475e-05
Epoch 67/100
42/42 [==============================] - 8s 197ms/step - loss: 4.1323e-05
Epoch 68/100
42/42 [==============================] - 8s 191ms/step - loss: 4.4698e-05
Epoch 69/100
42/42 [==============================] - 8s 188ms/step - loss: 4.1283e-05
Epoch 70/100
42/42 [==============================] - 8s 193ms/step - loss: 4.3266e-05
Epoch 71/100
42/42 [==============================] - 8s 192ms/step - loss: 4.5097e-05
Epoch 72/100
42/42 [==============================] - 8s 190ms/step - loss: 4.5874e-05
Epoch 73/100
42/42 [==============================] - 8s 189ms/step - loss: 1.0419e-04
Epoch 74/100
42/42 [==============================] - 8s 194ms/step - loss: 4.3756e-05
Epoch 75/100
42/42 [==============================] - 8s 191ms/step - loss: 4.2079e-05
Epoch 76/100
42/42 [==============================] - 8s 193ms/step - loss: 4.2263e-05
Epoch 77/100
42/42 [==============================] - 8s 189ms/step - loss: 4.3902e-05
Epoch 78/100
42/42 [==============================] - 8s 191ms/step - loss: 3.9015e-05
Epoch 79/100
42/42 [==============================] - 8s 194ms/step - loss: 4.0593e-05
Epoch 80/100
42/42 [==============================] - 8s 197ms/step - loss: 4.2939e-05
Epoch 81/100
42/42 [==============================] - 8s 193ms/step - loss: 4.1068e-05
Epoch 82/100
42/42 [==============================] - 8s 191ms/step - loss: 3.9171e-05
Epoch 83/100
42/42 [==============================] - 8s 195ms/step - loss: 4.3510e-05
Epoch 84/100
42/42 [==============================] - 8s 193ms/step - loss: 4.5120e-05
Epoch 85/100
42/42 [==============================] - 8s 191ms/step - loss: 4.5724e-05
Epoch 86/100
42/42 [==============================] - 8s 190ms/step - loss: 3.7270e-05
Epoch 87/100
42/42 [==============================] - 8s 192ms/step - loss: 5.1366e-05
Epoch 88/100
42/42 [==============================] - 8s 191ms/step - loss: 4.9532e-05
Epoch 89/100
42/42 [==============================] - 8s 197ms/step - loss: 5.4117e-05
Epoch 90/100
42/42 [==============================] - 8s 196ms/step - loss: 4.6834e-05
Epoch 91/100
42/42 [==============================] - 8s 194ms/step - loss: 4.7012e-05
Epoch 92/100
42/42 [==============================] - 8s 194ms/step - loss: 3.5028e-05
Epoch 93/100
42/42 [==============================] - 8s 191ms/step - loss: 5.7959e-05
Epoch 94/100
42/42 [==============================] - 8s 190ms/step - loss: 4.1982e-05
Epoch 95/100
42/42 [==============================] - 8s 194ms/step - loss: 4.5705e-05
Epoch 96/100
42/42 [==============================] - 8s 192ms/step - loss: 3.9993e-05
Epoch 97/100
42/42 [==============================] - 8s 192ms/step - loss: 4.1961e-05
Epoch 98/100
42/42 [==============================] - 8s 193ms/step - loss: 3.7307e-05
Epoch 99/100
42/42 [==============================] - 8s 193ms/step - loss: 3.4112e-05
Epoch 100/100
42/42 [==============================] - 8s 192ms/step - loss: 3.6281e-05
Total average accuracy of all epochs : 99.99988235544977
15/15 [==============================] - 2s 75ms/step
MAE: 7.6
MSE: 78.7
RMSE: 8.9
```

Predictied value v/s actual value of everyday Low MICROSOFT



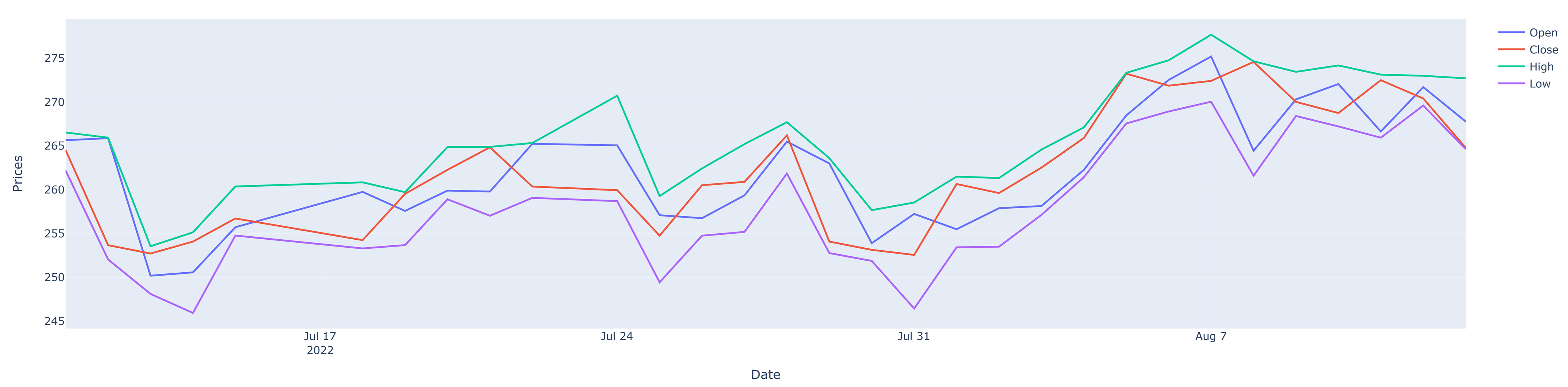

```
1/1 [==============================] - 1s 700ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 50ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 52ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 29ms/step
Stock Faith is Positive on everyday Low Price.

Stock Faith is Positive on analysis of news of the stock.

 MICROSOFT predicted values with previous 10 days actual values :
```

Date	Open	Close	High	Low
2022-07-11	265.649994	264.510010	266.529999	262.179993
2022-07-12	265.880005	253.669998	265.940002	252.039993
2022-07-13	250.190002	252.720001	253.550003	248.110001
2022-07-14	250.570007	254.080002	255.139999	245.940002
2022-07-15	255.720001	256.720001	260.369995	254.770004
2022-07-18	259.750000	254.250000	260.839996	253.300003
2022-07-19	257.579987	259.529999	259.720001	253.679993
2022-07-20	259.899994	262.269989	264.869995	258.910004
2022-07-21	259.790009	264.839996	264.890015	257.029999
2022-07-22	265.239990	260.359985	265.329987	259.070007
2022-07-24	265.062634	259.944550	270.729428	258.697454
2022-07-25	257.091212	254.756653	259.276125	249.439036
2022-07-26	256.752016	260.513245	262.428797	254.756663
2022-07-27	259.355934	260.892365	265.202334	255.185655
2022-07-28	265.491640	266.200012	267.706483	261.850127
2022-07-29	263.000000	254.080002	263.600006	252.770004
2022-07-30	253.899994	253.139999	257.670013	251.880005
2022-07-31	257.239990	252.559998	258.540009	246.440002
2022-08-01	255.490005	260.649994	261.500000	253.429993
2022-08-02	257.890015	259.619995	261.329987	253.500000
2022-08-03	258.140015	262.519989	264.579987	257.130005
2022-08-04	262.269989	265.899994	267.109985	261.429993
2022-08-05	268.480011	273.239990	273.339996	267.559998
2022-08-06	272.529999	271.869995	274.769989	268.929993
2022-08-07	275.200012	272.420013	277.690002	270.040009
2022-08-08	264.450012	274.579987	274.649994	261.600006
2022-08-09	270.309998	270.019989	273.450012	268.410004
2022-08-10	272.059998	268.750000	274.179993	267.220001
2022-08-11	266.640015	272.500000	273.130005	265.940002
2022-08-12	271.709991	270.410004	273.000000	269.609985
2022-08-13	267.779999	264.790009	272.709991	264.630005

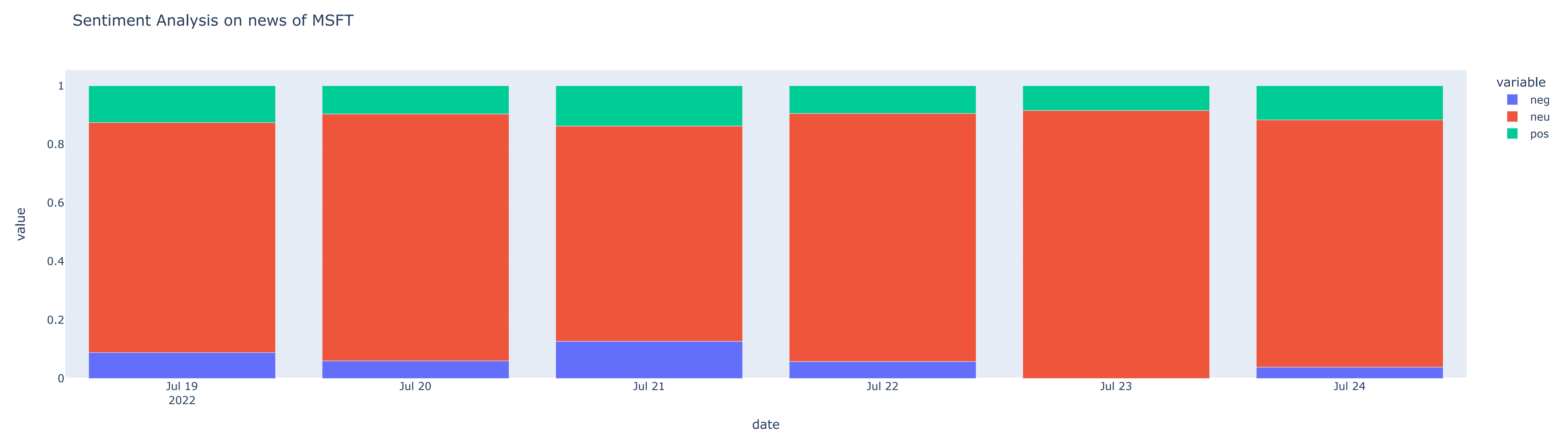
MICROSOFT Predicted values with previous 10 days actual values.




```
News Analysis on  MICROSOFT
```

	ticker	date	time	headline	neg	neu	pos	compound
95	MSFT	2022-07-19	09:12AM	Microsoft Is a Good Place to Hide in a Recessi...	0.322	0.537	0.141	-0.4019
96	MSFT	2022-07-19	09:09AM	Jefferies Recommends These 6 Defensive Stocks ...	0.237	0.508	0.254	-0.2023
97	MSFT	2022-07-19	08:44AM	Hosepipes on Roofs Are Keeping UKs Data Center...	0.000	0.777	0.223	0.3182

	ticker	date	time	headline	neg	neu	pos	compound
98	MSFT	2022-07-19	08:11AM	Tech Stocks Rise After Apples Hiring Slowdown	0.000	1.000	0.000	0.0000
99	MSFT	2022-07-19	02:17AM	Japanese Game Maker Nippon Ichi Is Overlooked ...	0.253	0.506	0.241	0.2023




```
Considering the best possible attributes of the stock :

Attributes : Opening Price, Closing Price, Everyday High, Everyday Low and News on Stock
Stock Faith is good, only Closing price is low.
```

In [ ]:

```
print(O,C,L,H,S)
type(S)
```

In [ ]: