BHARAT INTERN TASK 1 - STOCK PRICE PREDICTION USING LSTM

DESCRIPTION OF THE DATASET

Date - date

Open - the opening price of bitcoin at which the stock began trading.

High - maximum prices in a given time period.

Low - minimum prices in a given time period.

Close - the prices at which a stock ended trading in the same period.

Adj Close - the closing price after dividend payouts, stock splits, or the issue of additional shares have been taken into account.

Volume - volume is the amount of an asset that changes hands over some period of time.

IMPORTING NECESSARY LIBRARIES

In [2]:
```
# data processing
import pandas as pd

# data analysis
import numpy as np

# provides mathematical functions & constants for performing various mathematical operations.
import math

# for working with time-sensitive data
import datetime as dt

# data visualization
import matplotlib.pyplot as plt
import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots
```

In [3]:
```
# ignore warnings
import warnings
warnings.filterwarnings("ignore")
```

LOADING THE DATASET

In [74]:
```
df = pd.read_csv('D:/MDA 4th Sem/ML/BTC-USD.csv')
df.head(5)
```

Out[74]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2014-09-17	465.864014	468.174011	452.421997	457.334015	457.334015	21056800
1	2014-09-18	456.859985	456.859985	413.104004	424.440002	424.440002	34483200
2	2014-09-19	424.102997	427.834991	384.532013	394.795990	394.795990	37919700
3	2014-09-20	394.673004	423.295990	389.882996	408.903992	408.903992	36863600
4	2014-09-21	408.084991	412.425995	393.181000	398.821014	398.821014	26580100

In [5]:
```
df.shape
```

Out[5]:
```
(3248, 7)
```

There are 3248 rows and 7 columns in the dataframe.

In [6]:
```
print("Total number of days present in the dataset:" , df.shape[0])
```

```
Total number of days present in the dataset: 3248
```

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3248 entries, 0 to 3247
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Date       3248 non-null   object 
 1   Open       3248 non-null   float64
 2   High       3248 non-null   float64
 3   Low        3248 non-null   float64
 4   Close      3248 non-null   float64
 5   Adj Close  3248 non-null   float64
 6   Volume     3248 non-null   int64  
dtypes: float64(5), int64(1), object(1)
memory usage: 177.8+ KB
```

We have 6 numerical variables and 1 categorical variable. Additionally, presence of null values are not detected.

In [8]:
```
df.describe()
```

Out[8]:

	Open	High	Low	Close	Adj Close	Volume
count	3248.000000	3248.000000	3248.000000	3248.000000	3248.000000	3.248000e+03
mean	13761.612459	14093.852698	13398.213209	13769.104085	13769.104085	1.653628e+10
std	16016.005064	16413.031493	15563.010184	16013.423702	16013.423702	1.943633e+10
min	176.897003	211.731003	171.509995	178.102997	178.102997	5.914570e+06
25%	770.015976	775.216507	762.011261	771.068771	771.068771	1.319662e+08
50%	7783.005127	8015.491944	7567.979981	7801.329834	7801.329834	1.037589e+10
75%	20629.445801	21144.436035	20235.452637	20648.897949	20648.897949	2.731271e+10
max	67549.734375	68789.625000	66382.062500	67566.828125	67566.828125	3.509679e+11

It gives the count, mean, median, standerd deviation, minimum value, maximum value, 1st quartile and 3rd quartile values of each numerical variable.

CHECKING FOR NULL VALUES

In [9]:
```
df.isnull().sum()
```

Out[9]:
```
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

It does not have any null values.

EXPLORATORY DATA ANALYSIS

In [10]:
```
# retrieving the value located in the first row and first column of the DataFrame
sd = df.iloc[0][0]
print("Starting date of the bitcoin stock prices:" , sd)
```

```
Starting date of the bitcoin stock prices: 2014-09-17
```

In [11]:
```
# retrieving the value located in the last row and first column of the dataframe
ed = df.iloc[-1][0]
print("Ending date of the bitcoin stock prices:" , ed)
```

```
Ending date of the bitcoin stock prices: 2023-08-08
```

We have the bitcoin stock prices from 2014 to 2023.

In [75]:
```
df['Date'] = pd.to_datetime(df['Date'], format='%Y-%m-%d')
df.dtypes
```

Out[75]:
```
Date         datetime64[ns]
Open                float64
High                float64
Low                 float64
Close               float64
Adj Close           float64
Volume                int64
dtype: object
```

Here the column 'Date' was in the object type datatype so we converted it into the datetime format.

ANALYSING THE YEAR 2014

In [13]:
```
# extracting the stock prices in 2014
Year_2014 = df.loc[(df['Date'] >= '2014-09-17') & (df['Date'] <= '2014-12-31')]
Year_2014
```

Out[13]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2014-09-17	465.864014	468.174011	452.421997	457.334015	457.334015	21056800
1	2014-09-18	456.859985	456.859985	413.104004	424.440002	424.440002	34483200
2	2014-09-19	424.102997	427.834991	384.532013	394.795990	394.795990	37919700
3	2014-09-20	394.673004	423.295990	389.882996	408.903992	408.903992	36863600
4	2014-09-21	408.084991	412.425995	393.181000	398.821014	398.821014	26580100
...	...	...	...	...	...	...	...
101	2014-12-27	327.583008	328.911011	312.630005	315.863007	315.863007	15185200
102	2014-12-28	316.160004	320.028015	311.078003	317.239014	317.239014	11676600
103	2014-12-29	317.700989	320.266998	312.307007	312.670013	312.670013	12302500
104	2014-12-30	312.718994	314.808990	309.372986	310.737000	310.737000	12528300
105	2014-12-31	310.914001	320.192993	310.210999	320.192993	320.192993	13942900

106 rows × 7 columns

In [14]:
```
# dropping the columns 'Adj Close' and 'Volume'
Year_2014.drop(Year_2014[['Adj Close','Volume']], axis=1)
```

Out[14]:

	Date	Open	High	Low	Close
0	2014-09-17	465.864014	468.174011	452.421997	457.334015
1	2014-09-18	456.859985	456.859985	413.104004	424.440002
2	2014-09-19	424.102997	427.834991	384.532013	394.795990
3	2014-09-20	394.673004	423.295990	389.882996	408.903992
4	2014-09-21	408.084991	412.425995	393.181000	398.821014
...	...	...	...	...	...
101	2014-12-27	327.583008	328.911011	312.630005	315.863007
102	2014-12-28	316.160004	320.028015	311.078003	317.239014
103	2014-12-29	317.700989	320.266998	312.307007	312.670013
104	2014-12-30	312.718994	314.808990	309.372986	310.737000
105	2014-12-31	310.914001	320.192993	310.210999	320.192993

106 rows × 5 columns

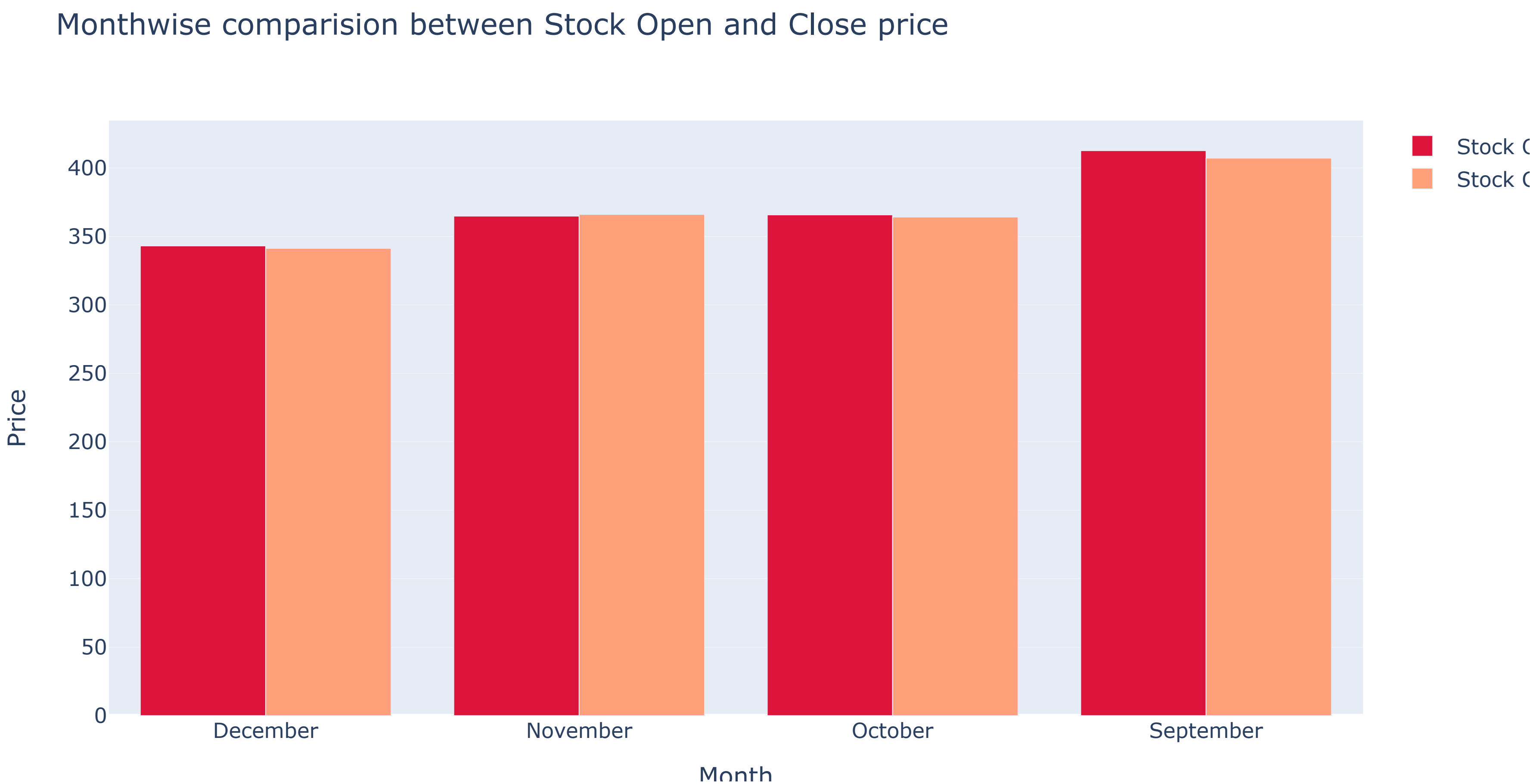
In [15]:
```
# Extracting monthwise
# strftime() method in pandas is used to format datetime objects into strings
Year_2014['Month'] = Year_2014['Date'].dt.strftime('%B')
monthwise = Year_2014.groupby('Month')[['Open', 'Close']].mean()
monthwise
```

Out[15]:

	Open	Close
Month		
December	343.074836	341.267871
November	364.850235	366.099799
October	365.748000	364.148873
September	412.654003	407.182428

OPEN vs CLOSE PRICES

In [16]:
```
fig = go.Figure()
fig.add_trace(go.Bar(x = monthwise.index, y = monthwise['Open'], name='Stock Open Price',marker_color='crimson'))
fig.add_trace(go.Bar(x = monthwise.index, y = monthwise['Close'], name='Stock Close Price', marker_color='lightsalmon'
fig.update_layout(
    title='Monthwise comparision between Stock Open and Close price',
    xaxis_title='Month',
    yaxis_title='Price',
    barmode='group')
```



HIGH vs LOW PRICES

In [17]:
```
monthwise_high = Year_2014.groupby('Month')[['High']].max()
monthwise_low = Year_2014.groupby('Month')[['Low']].min()
```

In [18]:
```
fig = go.Figure()
fig.add_trace(go.Bar(x = monthwise_high.index, y = monthwise_high['High'], name='Stock High Price'))
fig.add_trace(go.Bar(x = monthwise_low.index, y = monthwise_low['Low'], name='Stock Low Price'))
fig.update_layout(
    title='Monthwise comparision between Stock High and Low price',
    xaxis_title='Month',
    yaxis_title='Price',
    barmode='group')
```

OVERALL ANALYSIS

In [19]:
```
Year_all = df.loc[(df['Date'] >= '2014-09-17') & (df['Date'] <= '2023-08-08')]
Year_all
```

Out[19]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2014-09-17	465.864014	468.174011	452.421997	457.334015	457.334015	21056800
1	2014-09-18	456.859985	456.859985	413.104004	424.440002	424.440002	34483200
2	2014-09-19	424.102997	427.834991	384.532013	394.795990	394.795990	37919700
3	2014-09-20	394.673004	423.295990	389.882996	408.903992	408.903992	36863600
4	2014-09-21	408.084991	412.425995	393.181000	398.821014	398.821014	26580100
...	...	...	...	...	...	...	...
3243	2023-08-04	29174.382813	29302.078125	28885.335938	29074.091797	29074.091797	12036639988
3244	2023-08-05	29075.388672	29102.464844	28957.796875	29042.126953	29042.126953	6598366353
3245	2023-08-06	29043.701172	29160.822266	28963.833984	29041.855469	29041.855469	7269806994
3246	2023-08-07	29038.513672	29244.281250	28724.140625	29180.578125	29180.578125	13618163710
3247	2023-08-08	29185.019531	29258.433594	29114.607422	29222.226563	29222.226563	13701349376

3248 rows × 7 columns

In [20]:
```
# dropping the columns 'Adj Close' and 'Volume'
Year_all.drop(Year_all[['Adj Close','Volume']], axis=1)
```

Out[20]:

	Date	Open	High	Low	Close
0	2014-09-17	465.864014	468.174011	452.421997	457.334015
1	2014-09-18	456.859985	456.859985	413.104004	424.440002
2	2014-09-19	424.102997	427.834991	384.532013	394.795990
3	2014-09-20	394.673004	423.295990	389.882996	408.903992
4	2014-09-21	408.084991	412.425995	393.181000	398.821014
...	...	...	...	...	...
3243	2023-08-04	29174.382813	29302.078125	28885.335938	29074.091797
3244	2023-08-05	29075.388672	29102.464844	28957.796875	29042.126953
3245	2023-08-06	29043.701172	29160.822266	28963.833984	29041.855469
3246	2023-08-07	29038.513672	29244.281250	28724.140625	29180.578125
3247	2023-08-08	29185.019531	29258.433594	29114.607422	29222.226563

3248 rows × 5 columns

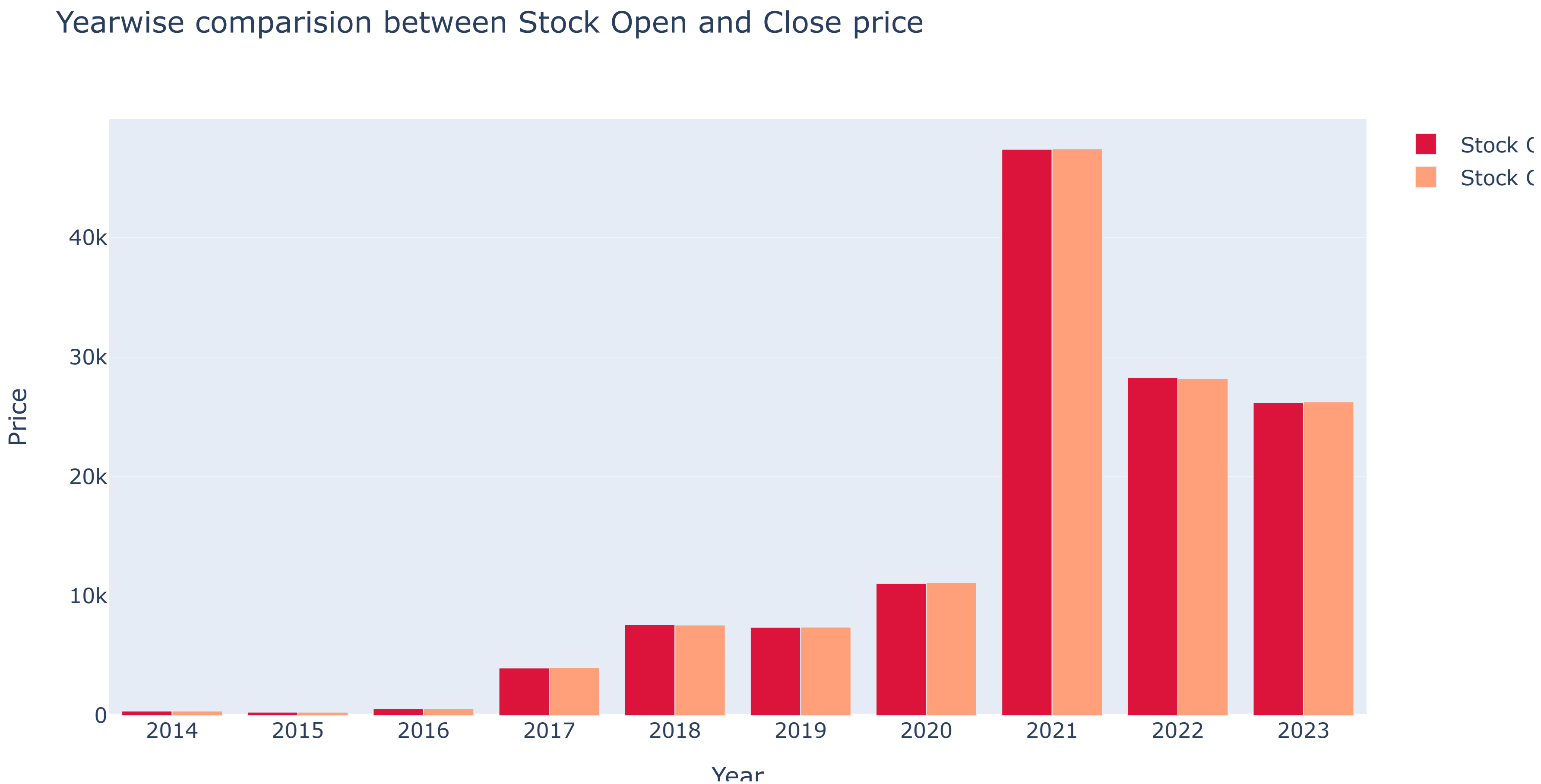
In [21]:
```
Year_all['Year'] = Year_all['Date'].dt.strftime('%Y')
yearwise = Year_all.groupby('Year')[['Open', 'Close']].mean()
yearwise
```

Out[21]:

Year	Open	Close
2014	365.058217	363.693085
2015	272.149011	272.453381
2016	567.141429	568.492407
2017	3970.644848	4006.033629
2018	7601.018680	7572.298947
2019	7385.218456	7395.246282
2020	11056.787201	11116.378092
2021	47402.115663	47436.932021
2022	28278.690293	28197.754099
2023	26193.512411	26251.699077

OPEN vs CLOSE PRICES

In [22]:
```
fig = go.Figure()
fig.add_trace(go.Bar(x = yearwise.index, y = yearwise['Open'], name='Stock Open Price',marker_color='crimson'))
fig.add_trace(go.Bar(x = yearwise.index, y = yearwise['Close'], name='Stock Close Price', marker_color='lightsalmon'))
fig.update_layout(
    title='Yearwise comparision between Stock Open and Close price',
    xaxis_title='Year',
    yaxis_title='Price',
    barmode='group')
```



HIGH vs LOW PRICES

In [23]:
```
yearwise_high = Year_all.groupby('Year')[['High']].max()
yearwise_low = Year_all.groupby('Year')[['Low']].min()
```

In [24]:
```
fig = go.Figure()
fig.add_trace(go.Bar(x = yearwise_high.index, y = yearwise_high['High'], name='Stock High Price'))
fig.add_trace(go.Bar(x = yearwise_low.index, y = yearwise_low['Low'], name='Stock Low Price'))
fig.update_layout(
    title='Yearwise comparision between Stock High and Low price',
    xaxis_title='Year',
    yaxis_title='Price',
    barmode='group')
```

In [35]:
```
closedf = df[['Date','Close']]
```

We want to predict Close Price of the Bitcoin so we are just considering the columns 'Close' and 'Date'.

In [36]:
```
closedf = closedf[closedf['Date'] > '2022-08-08']
closedf
```

Out[36]:

	Date	Close
2883	2022-08-09	23164.318359
2884	2022-08-10	23947.642578
2885	2022-08-11	23957.529297
2886	2022-08-12	24402.818359
2887	2022-08-13	24424.068359
...	...	...
3243	2023-08-04	29074.091797
3244	2023-08-05	29042.126953
3245	2023-08-06	29041.855469
3246	2023-08-07	29180.578125
3247	2023-08-08	29222.226563

365 rows × 2 columns

Cryptocurrency markets like Bitcoin can be highly volatile. Such markets are influenced by various factors, including news events, economic indicators, and market sentiment. These factors can change over time, and the patterns that emerge within a specific year might not hold true for a longer period. So we we will just consider 1 Year to avoid this type of flucation in the data.

In [37]:
```
close_stock = closedf.copy()
```

Normalizing Data

In [38]:
```
from sklearn.preprocessing import MinMaxScaler
del closedf['Date']
scaler=MinMaxScaler(feature_range=(0,1))
closedf=scaler.fit_transform(np.array(closedf).reshape(-1,1))
closedf
```

Out[38]:
```
array([[0.47021128],
       [0.52014028],
       [0.52077046],
       [0.54915313],
       [0.5505076 ],
       [0.54383184],
       [0.53220815],
       [0.51603852],
       [0.48115413],
       [0.47329756],
       [0.32445311],
       [0.34284257],
       [0.3663027 ],
       [0.35768425],
       [0.36591815],
       [0.35743639],
       [0.37055946],
       [0.28509162],
       [0.2711784 ],
```

By using Min-Max Normalization we are transforming features so that they have values between a specified minimum and maximum value, often between 0 and 1. The scaled value is calculated by:

scaled value = (value - minimum value) / (maximum value - minimum value)

In [39]:
```
closedf.shape
```

Out[39]:
```
(365, 1)
```

There are 365 values in closedf.

SPLITTING YOUR DATSET INTO TRAINING AND TEST SETS

In [40]:
```
training_size=int(len(closedf)*0.70)
test_size=len(closedf)-training_size
train_data,test_data=closedf[0:training_size,:],closedf[training_size:len(closedf),:1]
print("train_data: ", train_data.shape)
print("test_data: ", test_data.shape)
```

```
train_data:  (255, 1)
test_data:  (110, 1)
```

Here we allocate 70% of the data for training the machine learning model and the remaining 30% of the data for testing the model's performance on unseen data. Our trai data contains 255 datapoints and test data contains 110 datapoints.

VISUALIZING THE TRAINING AND TEST DATA

In [41]:
```
fig = go.Figure()
# Adding the train data line
fig.add_trace(go.Scatter(x=close_stock['Date'][:255], y=close_stock['Close'][:255],
                         mode='lines',
                         name='Train Data'))
# Adding the test data line
fig.add_trace(go.Scatter(x=close_stock['Date'][255:], y=close_stock['Close'][255:],
                         mode='lines',
                         name='Test Data'))
# Customize the layout
fig.update_layout(
    title='Train & Test Data',
    xaxis_title='Date',
    yaxis_title='Close')
```

CREATING INPUT-OUTPUT PAIRS FOR TIME SERIES FORECASTING

In [42]:
```
# convert an array of values into a dataset matrix

def create_dataset(dataset, time_step=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-time_step-1):
        a = dataset[i:(i+time_step), 0]   ###i=0, 0,1,2,3-----99   100
        dataX.append(a)
        dataY.append(dataset[i + time_step, 0])
    return np.array(dataX), np.array(dataY)
```

dataX will contain sequences of past values that the model will use to predict the corresponding target values in dataY

In [43]:
```
time_step = 15
X_train, y_train = create_dataset(train_data, time_step)
X_test, y_test = create_dataset(test_data, time_step)

print("X_train: ", X_train.shape)
print("y_train: ", y_train.shape)
print("X_test: ", X_test.shape)
print("y_test", y_test.shape)
```

```
X_train:  (239, 15)
y_train:  (239,)
X_test:  (94, 15)
y_test (94,)
```

RESHAPING THE INPUT DATA ARRAYS X_train AND X_test

In [44]:
```
X_train =X_train.reshape(X_train.shape[0],X_train.shape[1] , 1)
X_test = X_test.reshape(X_test.shape[0],X_test.shape[1] , 1)
```

Reshaping the input data arrays X_train and X_test to match the expected input shape for sequence-based models like Long Short-Term Memory (LSTM). The reason for reshaping the data is that many sequence-based neural network models (such as LSTMs) expect input data in a specific format that includes the number of samples, the length of sequences, and the number of features at each time step. By adding the extra dimension of size 1, you're indicating that you have only one feature (past stock price) at each time step of the sequence.

MODEL BUILDING

In [45]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.layers import LSTM
```

In [46]:
```
model=Sequential()

model.add(LSTM(10,input_shape=(None,1),activation="relu"))

model.add(Dense(1))

model.compile(loss="mean_squared_error",optimizer="adam")
```

In [47]:
```
history = model.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=200,batch_size=32,verbose=1)
```

```
Epoch 1/200
8/8 [==============================] - 9s 174ms/step - loss: 0.1737 - val_loss: 0.6604
Epoch 2/200
8/8 [==============================] - 0s 13ms/step - loss: 0.1630 - val_loss: 0.6286
Epoch 3/200
8/8 [==============================] - 0s 13ms/step - loss: 0.1530 - val_loss: 0.5969
Epoch 4/200
8/8 [==============================] - 0s 13ms/step - loss: 0.1421 - val_loss: 0.5623
Epoch 5/200
8/8 [==============================] - 0s 11ms/step - loss: 0.1290 - val_loss: 0.5166
Epoch 6/200
8/8 [==============================] - 0s 11ms/step - loss: 0.1150 - val_loss: 0.4644
Epoch 7/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0998 - val_loss: 0.4041
Epoch 8/200
8/8 [==============================] - 0s 12ms/step - loss: 0.0836 - val_loss: 0.3324
Epoch 9/200
8/8 [==============================] - 0s 12ms/step - loss: 0.0661 - val_loss: 0.2413
Epoch 10/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0457 - val_loss: 0.1251
Epoch 11/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0281 - val_loss: 0.0280
Epoch 12/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0209 - val_loss: 0.0070
Epoch 13/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0168 - val_loss: 0.0072
Epoch 14/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0130 - val_loss: 0.0079
Epoch 15/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0112 - val_loss: 0.0184
Epoch 16/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0099 - val_loss: 0.0338
Epoch 17/200
8/8 [==============================] - 0s 10ms/step - loss: 0.0093 - val_loss: 0.0212
Epoch 18/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0091 - val_loss: 0.0326
Epoch 19/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0088 - val_loss: 0.0297
Epoch 20/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0085 - val_loss: 0.0269
Epoch 21/200
```

```
8/8 [==============================] - 0s 13ms/step - loss: 0.0083 - val_loss: 0.0252
Epoch 22/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0081 - val_loss: 0.0239
Epoch 23/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0079 - val_loss: 0.0182
Epoch 24/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0077 - val_loss: 0.0212
Epoch 25/200
8/8 [==============================] - 0s 16ms/step - loss: 0.0077 - val_loss: 0.0253
Epoch 26/200
8/8 [==============================] - 0s 16ms/step - loss: 0.0074 - val_loss: 0.0145
Epoch 27/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0072 - val_loss: 0.0171
Epoch 28/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0070 - val_loss: 0.0193
Epoch 29/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0070 - val_loss: 0.0141
Epoch 30/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0068 - val_loss: 0.0179
Epoch 31/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0067 - val_loss: 0.0177
Epoch 32/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0065 - val_loss: 0.0130
Epoch 33/200
8/8 [==============================] - 0s 16ms/step - loss: 0.0064 - val_loss: 0.0133
Epoch 34/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0063 - val_loss: 0.0154
Epoch 35/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0063 - val_loss: 0.0162
Epoch 36/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0061 - val_loss: 0.0101
Epoch 37/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0061 - val_loss: 0.0123
Epoch 38/200
8/8 [==============================] - 0s 16ms/step - loss: 0.0059 - val_loss: 0.0119
Epoch 39/200
8/8 [==============================] - 0s 20ms/step - loss: 0.0059 - val_loss: 0.0125
Epoch 40/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0058 - val_loss: 0.0117
Epoch 41/200
8/8 [==============================] - 0s 16ms/step - loss: 0.0057 - val_loss: 0.0107
Epoch 42/200
```

```
8/8 [==============================] - 0s 16ms/step - loss: 0.0058 - val_loss: 0.0102
Epoch 43/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0056 - val_loss: 0.0127
Epoch 44/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0056 - val_loss: 0.0099
Epoch 45/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0055 - val_loss: 0.0080
Epoch 46/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0054 - val_loss: 0.0123
Epoch 47/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0054 - val_loss: 0.0071
Epoch 48/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0052 - val_loss: 0.0082
Epoch 49/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0051 - val_loss: 0.0092
Epoch 50/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0050 - val_loss: 0.0075
Epoch 51/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0050 - val_loss: 0.0068
Epoch 52/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0049 - val_loss: 0.0069
Epoch 53/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0049 - val_loss: 0.0062
Epoch 54/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0048 - val_loss: 0.0064
Epoch 55/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0047 - val_loss: 0.0066
Epoch 56/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0047 - val_loss: 0.0062
Epoch 57/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0046 - val_loss: 0.0045
Epoch 58/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0045 - val_loss: 0.0057
Epoch 59/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0044 - val_loss: 0.0059
Epoch 60/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0044 - val_loss: 0.0042
Epoch 61/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0043 - val_loss: 0.0048
Epoch 62/200
8/8 [==============================] - 0s 16ms/step - loss: 0.0042 - val_loss: 0.0036
Epoch 63/200
```

```
8/8 [==============================] - 0s 13ms/step - loss: 0.0042 - val_loss: 0.0050
Epoch 64/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0041 - val_loss: 0.0038
Epoch 65/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0041 - val_loss: 0.0031
Epoch 66/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0040 - val_loss: 0.0042
Epoch 67/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0039 - val_loss: 0.0034
Epoch 68/200
8/8 [==============================] - 0s 16ms/step - loss: 0.0039 - val_loss: 0.0033
Epoch 69/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0038 - val_loss: 0.0030
Epoch 70/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0038 - val_loss: 0.0028
Epoch 71/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0037 - val_loss: 0.0030
Epoch 72/200
8/8 [==============================] - 0s 16ms/step - loss: 0.0037 - val_loss: 0.0023
Epoch 73/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0036 - val_loss: 0.0023
Epoch 74/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0036 - val_loss: 0.0026
Epoch 75/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0035 - val_loss: 0.0021
Epoch 76/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0035 - val_loss: 0.0021
Epoch 77/200
8/8 [==============================] - 0s 16ms/step - loss: 0.0035 - val_loss: 0.0021
Epoch 78/200
8/8 [==============================] - 0s 16ms/step - loss: 0.0034 - val_loss: 0.0022
Epoch 79/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0034 - val_loss: 0.0020
Epoch 80/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0034 - val_loss: 0.0020
Epoch 81/200
8/8 [==============================] - 0s 16ms/step - loss: 0.0034 - val_loss: 0.0019
Epoch 82/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0034 - val_loss: 0.0021
Epoch 83/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0033 - val_loss: 0.0019
```

```
Epoch 84/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0033 - val_loss: 0.0021
Epoch 85/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0034 - val_loss: 0.0021
Epoch 86/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0033 - val_loss: 0.0019
Epoch 87/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0032 - val_loss: 0.0022
Epoch 88/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0032 - val_loss: 0.0019
Epoch 89/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0032 - val_loss: 0.0023
Epoch 90/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0031 - val_loss: 0.0019
Epoch 91/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0031 - val_loss: 0.0019
Epoch 92/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0031 - val_loss: 0.0027
Epoch 93/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0031 - val_loss: 0.0018
Epoch 94/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0030 - val_loss: 0.0022
Epoch 95/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0030 - val_loss: 0.0023
Epoch 96/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0030 - val_loss: 0.0018
Epoch 97/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0030 - val_loss: 0.0025
Epoch 98/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0030 - val_loss: 0.0019
Epoch 99/200
8/8 [==============================] - 0s 16ms/step - loss: 0.0030 - val_loss: 0.0019
Epoch 100/200
8/8 [==============================] - 0s 16ms/step - loss: 0.0030 - val_loss: 0.0022
Epoch 101/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0029 - val_loss: 0.0019
Epoch 102/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0029 - val_loss: 0.0020
Epoch 103/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0029 - val_loss: 0.0020
Epoch 104/200
```

```
8/8 [==============================] - 0s 11ms/step - loss: 0.0028 - val_loss: 0.0018
Epoch 105/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0028 - val_loss: 0.0020
Epoch 106/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0028 - val_loss: 0.0021
Epoch 107/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0028 - val_loss: 0.0018
Epoch 108/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0028 - val_loss: 0.0021
Epoch 109/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0029 - val_loss: 0.0018
Epoch 110/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0028 - val_loss: 0.0021
Epoch 111/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0027 - val_loss: 0.0018
Epoch 112/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0027 - val_loss: 0.0017
Epoch 113/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0028 - val_loss: 0.0020
Epoch 114/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0027 - val_loss: 0.0016
Epoch 115/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0028 - val_loss: 0.0021
Epoch 116/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0026 - val_loss: 0.0015
Epoch 117/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0026 - val_loss: 0.0018
Epoch 118/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0026 - val_loss: 0.0019
Epoch 119/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0026 - val_loss: 0.0017
Epoch 120/200
8/8 [==============================] - 0s 20ms/step - loss: 0.0026 - val_loss: 0.0019
Epoch 121/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0025 - val_loss: 0.0017
Epoch 122/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0026 - val_loss: 0.0017
Epoch 123/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0025 - val_loss: 0.0016
Epoch 124/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0025 - val_loss: 0.0017
Epoch 125/200
```

```
8/8 [==============================] - 0s 11ms/step - loss: 0.0025 - val_loss: 0.0016
Epoch 126/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0025 - val_loss: 0.0016
Epoch 127/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0025 - val_loss: 0.0016
Epoch 128/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0024 - val_loss: 0.0014
Epoch 129/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0025 - val_loss: 0.0017
Epoch 130/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0024 - val_loss: 0.0015
Epoch 131/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0024 - val_loss: 0.0016
Epoch 132/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0024 - val_loss: 0.0015
Epoch 133/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0024 - val_loss: 0.0014
Epoch 134/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0024 - val_loss: 0.0015
Epoch 135/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0024 - val_loss: 0.0016
Epoch 136/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0024 - val_loss: 0.0014
Epoch 137/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0023 - val_loss: 0.0016
Epoch 138/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0024 - val_loss: 0.0014
Epoch 139/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0023 - val_loss: 0.0016
Epoch 140/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0023 - val_loss: 0.0013
Epoch 141/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0022 - val_loss: 0.0016
Epoch 142/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0023 - val_loss: 0.0013
Epoch 143/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0022 - val_loss: 0.0014
Epoch 144/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0022 - val_loss: 0.0013
Epoch 145/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0022 - val_loss: 0.0013
Epoch 146/200
```

```
8/8 [==============================] - 0s 13ms/step - loss: 0.0022 - val_loss: 0.0015
Epoch 147/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0022 - val_loss: 0.0013
Epoch 148/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0022 - val_loss: 0.0016
Epoch 149/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0022 - val_loss: 0.0012
Epoch 150/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0022 - val_loss: 0.0013
Epoch 151/200
8/8 [==============================] - 0s 16ms/step - loss: 0.0022 - val_loss: 0.0012
Epoch 152/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0022 - val_loss: 0.0014
Epoch 153/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0021 - val_loss: 0.0012
Epoch 154/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0021 - val_loss: 0.0013
Epoch 155/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0021 - val_loss: 0.0012
Epoch 156/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0021 - val_loss: 0.0012
Epoch 157/200
8/8 [==============================] - 0s 12ms/step - loss: 0.0021 - val_loss: 0.0013
Epoch 158/200
8/8 [==============================] - 0s 15ms/step - loss: 0.0021 - val_loss: 0.0012
Epoch 159/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0020 - val_loss: 0.0013
Epoch 160/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0020 - val_loss: 0.0012
Epoch 161/200
8/8 [==============================] - 0s 20ms/step - loss: 0.0020 - val_loss: 0.0012
Epoch 162/200
8/8 [==============================] - 0s 22ms/step - loss: 0.0020 - val_loss: 0.0012
Epoch 163/200
8/8 [==============================] - 0s 14ms/step - loss: 0.0020 - val_loss: 0.0012
Epoch 164/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0020 - val_loss: 0.0014
Epoch 165/200
8/8 [==============================] - 0s 10ms/step - loss: 0.0020 - val_loss: 0.0012
Epoch 166/200
```

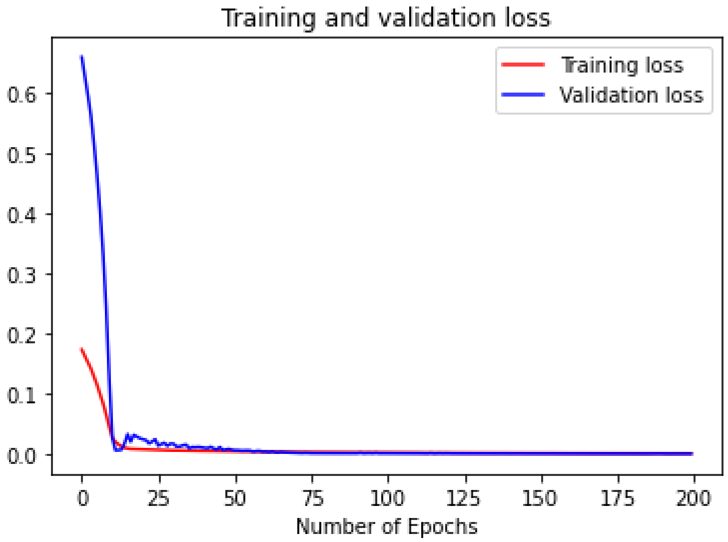
```
8/8 [==============================] - 0s 11ms/step - loss: 0.0020 - val_loss: 0.0012
Epoch 167/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0019 - val_loss: 0.0011
Epoch 168/200
8/8 [==============================] - 0s 12ms/step - loss: 0.0020 - val_loss: 0.0011
Epoch 169/200
8/8 [==============================] - 0s 12ms/step - loss: 0.0019 - val_loss: 0.0012
Epoch 170/200
8/8 [==============================] - 0s 13ms/step - loss: 0.0019 - val_loss: 0.0011
Epoch 171/200
8/8 [==============================] - 0s 12ms/step - loss: 0.0019 - val_loss: 0.0012
Epoch 172/200
8/8 [==============================] - 0s 10ms/step - loss: 0.0019 - val_loss: 0.0012
Epoch 173/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0019 - val_loss: 0.0011
Epoch 174/200
8/8 [==============================] - 0s 12ms/step - loss: 0.0019 - val_loss: 0.0011
Epoch 175/200
8/8 [==============================] - 0s 12ms/step - loss: 0.0019 - val_loss: 0.0011
Epoch 176/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0019 - val_loss: 0.0011
Epoch 177/200
8/8 [==============================] - 0s 17ms/step - loss: 0.0019 - val_loss: 0.0011
Epoch 178/200
8/8 [==============================] - 0s 16ms/step - loss: 0.0019 - val_loss: 0.0011
Epoch 179/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0019 - val_loss: 0.0011
Epoch 180/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0018 - val_loss: 0.0012
Epoch 181/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0018 - val_loss: 0.0011
Epoch 182/200
8/8 [==============================] - 0s 12ms/step - loss: 0.0018 - val_loss: 0.0011
Epoch 183/200
8/8 [==============================] - 0s 15ms/step - loss: 0.0018 - val_loss: 0.0011
Epoch 184/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0018 - val_loss: 0.0011
Epoch 185/200
8/8 [==============================] - 0s 12ms/step - loss: 0.0018 - val_loss: 0.0011
Epoch 186/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0018 - val_loss: 0.0012
```

```
Epoch 187/200
8/8 [==============================] - 0s 12ms/step - loss: 0.0018 - val_loss: 0.0011
Epoch 188/200
8/8 [==============================] - 0s 12ms/step - loss: 0.0018 - val_loss: 0.0011
Epoch 189/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0018 - val_loss: 0.0011
Epoch 190/200
8/8 [==============================] - 0s 14ms/step - loss: 0.0017 - val_loss: 0.0014
Epoch 191/200
8/8 [==============================] - 0s 17ms/step - loss: 0.0018 - val_loss: 0.0011
Epoch 192/200
8/8 [==============================] - 0s 14ms/step - loss: 0.0018 - val_loss: 0.0011
Epoch 193/200
8/8 [==============================] - 0s 12ms/step - loss: 0.0017 - val_loss: 0.0011
Epoch 194/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0017 - val_loss: 0.0011
Epoch 195/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0018 - val_loss: 0.0011
Epoch 196/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0018 - val_loss: 0.0011
Epoch 197/200
8/8 [==============================] - 0s 12ms/step - loss: 0.0018 - val_loss: 0.0013
Epoch 198/200
8/8 [==============================] - 0s 12ms/step - loss: 0.0017 - val_loss: 0.0013
Epoch 199/200
8/8 [==============================] - 0s 11ms/step - loss: 0.0017 - val_loss: 0.0011
Epoch 200/200
8/8 [==============================] - 0s 12ms/step - loss: 0.0017 - val_loss: 0.0011
```

PLOTTING TRAINING LOSS AND VALIDATION LOSS

In [50]:
```
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(loss))

plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Number of Epochs')
plt.legend(loc=0)
plt.figure()
```

Out[50]:
```
<Figure size 432x288 with 0 Axes>
```




```
<Figure size 432x288 with 0 Axes>
```

TRAINING LOSS

Training loss (or training error) is a measure of how well the model is performing on the training data during the training process. It quantifies the difference between the predicted values generated by the model and the actual target values in the training dataset.

High training loss indicates that the model is struggling to fit the training data. This could be due to underfitting, where the model's capacity is not sufficient to capture the complexities of the data.

VALIDATION LOSS

Validation loss (or validation error) is a measure of how well the model generalizes to unseen data. During training, a separate validation dataset (distinct from the training dataset) is used to evaluate the model's performance. The validation loss is computed by applying the model to the validation data and comparing its predictions to the actual target values.

Low training loss doesn't guarantee a well-performing model. If the model becomes too complex, it might memorize the training data and perform poorly on new, unseen data.

Here the two curves are moving closer together, it suggests that the model is generalizing well. Both training loss and validation loss decreasing during training.

MAKING PREDICTIONS ON BOTH TRAINING AND TEST DATA

In [51]:
```
train_predict=model.predict(X_train)
test_predict=model.predict(X_test)
train_predict.shape, test_predict.shape
```

```
8/8 [==============================] - 31s 27ms/step
3/3 [==============================] - 0s 10ms/step
```

Out[51]:
```
((239, 1), (94, 1))
```

MODEL EVALUATION

In [52]:
```
# applying inverse transformations so you have the predicted and original target values in the same scale as the origi
train_predict = scaler.inverse_transform(train_predict)
test_predict = scaler.inverse_transform(test_predict)
original_ytrain = scaler.inverse_transform(y_train.reshape(-1,1))
original_ytest = scaler.inverse_transform(y_test.reshape(-1,1))
```

In [53]:
```
from sklearn.metrics import mean_squared_error, mean_absolute_error, explained_variance_score, r2_score
from sklearn.metrics import mean_poisson_deviance, mean_gamma_deviance, accuracy_score
```

In [62]:
```
# Calculating Root mean squared error
print("Train data RMSE: ", math.sqrt(mean_squared_error(original_ytrain,train_predict)))
print("Test data RMSE: ", math.sqrt(mean_squared_error(original_ytest,test_predict)))
```

```
Train data RMSE:  642.6693852971731
Test data RMSE:  530.4692393229878
```

The test RMSE (530.47) is lower than the training RMSE (642.67). This suggests that your model's predictions on the test data are, on average, closer to the actual target values compared to the predictions on the training data. This is a positive sign indicating that your model is generalizing well to new data and is not overly fitting the training data.

In [59]:
```
# Calculating Mean squared error
print("Train data MSE: ", mean_squared_error(original_ytrain,train_predict))
print("Test data MSE: ", mean_squared_error(original_ytest,test_predict))
```

```
Train data MSE:  413023.9387982464
Test data MSE:  281397.6138679093
```

The test MSE (281397.61) is lower than the training MSE (413023.94). This suggests that your model's predictions on the test data have, on average, smaller squared differences from the actual target values compared to the predictions on the training data. This is a positive sign indicating that your model is generalizing well to new data.

In [60]:
```
# calculating Mean absolute error
print("Train data MAE: ", mean_absolute_error(original_ytrain,train_predict))
print("Test data MAE: ", mean_absolute_error(original_ytest,test_predict))
```

```
Train data MAE:  421.3229970237971
Test data MAE:  379.1718750039892
```

The test MAE (379.17) is lower than the training MAE (421.32). This suggests that your model's predictions on the test data have, on average, smaller absolute differences from the actual target values compared to the predictions on the training data. This indicates that your model is generalizing well to new data and is making predictions that are closer to the actual values.

EXPLAINED VARIANCE REGRESSION SCORE

In [64]:
```
print("Train data explained variance regression score:",
      explained_variance_score(original_ytrain, train_predict))
print("Test data explained variance regression score:",
      explained_variance_score(original_ytest, test_predict))
```

```
Train data explained variance regression score: 0.9721007198642504
Test data explained variance regression score: 0.901863308396677
```

An explained variance score close to 1 indicates that the model's predictions are closely aligned with the actual values, explaining a high proportion of the variance and the score closer to 0 suggests that the model's predictions do not explain much of the variance and might not be capturing the underlying patterns well. Higher variance indicate the strong relationship between the model's predictions and the actual data.

R-SQUARED VALUE

In [65]:
```
print("Train data R2 score:", r2_score(original_ytrain, train_predict))
print("Test data R2 score:", r2_score(original_ytest, test_predict))
```

```
Train data R2 score: 0.9720501209453751
Test data R2 score: 0.9013773624544079
```

The R-squared (R2) value, also known as the coefficient of determination, is a statistical measure used to assess how well a model fits the observed data. The training and test R2 scores are high, indicating that your model is performing well on both datasets. The higher training score (0.9721) suggests that the model's predictions match the actual values closely in the training data. The slightly lower test score (0.9014) indicates that the model's performance on the test data is still strong and is capturing a significant portion of the variance.

Comparision of original stock close price and predicted close price

In [68]:
```
from itertools import cycle
```

In [69]:
```
# shift train predictions for plotting

look_back=time_step
trainPredictPlot = np.empty_like(closedf)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back, :] = train_predict
print("Train predicted data: ", trainPredictPlot.shape)

# shift test predictions for plotting
testPredictPlot = np.empty_like(closedf)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(closedf)-1, :] = test_predict
print("Test predicted data: ", testPredictPlot.shape)

names = cycle(['Original close price','Train predicted close price','Test predicted close price'])


plotdf = pd.DataFrame({'date': close_stock['Date'],
                       'original_close': close_stock['Close'],
                      'train_predicted_close': trainPredictPlot.reshape(1,-1)[0].tolist(),
                      'test_predicted_close': testPredictPlot.reshape(1,-1)[0].tolist()})

fig = px.line(plotdf,x=plotdf['date'], y=[plotdf['original_close'],plotdf['train_predicted_close'],
                                          plotdf['test_predicted_close']],
              labels={'value':'Stock price','date': 'Date'})
fig.update_layout(title_text='Comparision between original close price vs predicted close price',
                  plot_bgcolor='white', font_size=15, font_color='black', legend_title_text='Close Price')
fig.for_each_trace(lambda t:  t.update(name = next(names)))

fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
```

```
Train predicted data:  (365, 1)
Test predicted data:  (365, 1)
```

Predicting next 30 days

In [70]:
```
x_input=test_data[len(test_data)-time_step:].reshape(1,-1)
temp_input=list(x_input)
temp_input=temp_input[0].tolist()

from numpy import array

lst_output=[]
n_steps=time_step
i=0
pred_days = 30
while(i<pred_days):

    if(len(temp_input)>time_step):

        x_input=np.array(temp_input[1:])
        #print("{} day input {}".format(i,x_input))
        x_input = x_input.reshape(1,-1)
        x_input = x_input.reshape((1, n_steps, 1))
        yhat = model.predict(x_input, verbose=0)
        #print("{} day output {}".format(i,yhat))
        temp_input.extend(yhat[0].tolist())
        temp_input=temp_input[1:]
        #print(temp_input)

        lst_output.extend(yhat.tolist())
        i=i+1

    else:

        x_input = x_input.reshape((1, n_steps,1))
        yhat = model.predict(x_input, verbose=0)
        temp_input.extend(yhat[0].tolist())

        lst_output.extend(yhat.tolist())
        i=i+1

print("Output of predicted next days: ", len(lst_output))
```

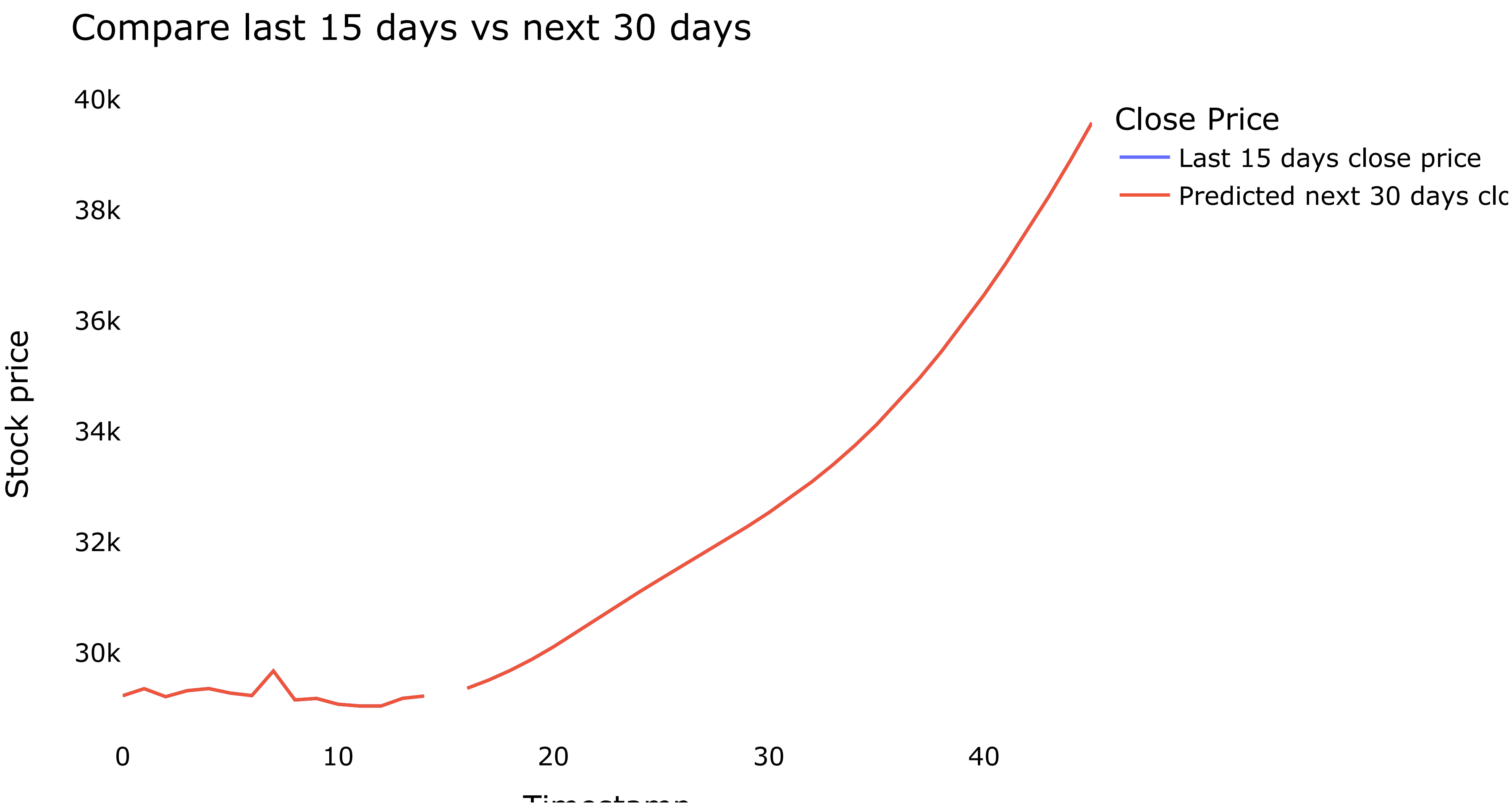
```
Output of predicted next days:  30
```

Plotting last 15 days of dataset and next predicted 30 days

In [71]:
```
last_days=np.arange(1,time_step+1)
day_pred=np.arange(time_step+1,time_step+pred_days+1)
print(last_days)
print(day_pred)
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15]
[16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
 40 41 42 43 44 45]
```

In [72]:
```
temp_mat = np.empty((len(last_days)+pred_days+1,1))
temp_mat[:] = np.nan
temp_mat = temp_mat.reshape(1,-1).tolist()[0]

last_original_days_value = temp_mat
next_predicted_days_value = temp_mat

last_original_days_value[0:time_step+1] = scaler.inverse_transform(closedf[len(closedf)-time_step:]).reshape(1,-1).tol
next_predicted_days_value[time_step+1:] = scaler.inverse_transform(np.array(lst_output).reshape(-1,1)).reshape(1,-1).t

new_pred_plot = pd.DataFrame({
    'last_original_days_value':last_original_days_value,
    'next_predicted_days_value':next_predicted_days_value
})

names = cycle(['Last 15 days close price','Predicted next 30 days close price'])

fig = px.line(new_pred_plot,x=new_pred_plot.index, y=[new_pred_plot['last_original_days_value'],
                                                      new_pred_plot['next_predicted_days_value']],
              labels={'value': 'Stock price','index': 'Timestamp'})
fig.update_layout(title_text='Compare last 15 days vs next 30 days',
                  plot_bgcolor='white', font_size=15, font_color='black',legend_title_text='Close Price')

fig.for_each_trace(lambda t:  t.update(name = next(names)))
fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
```



Plotting entire Closing Stock Price with next 30 days period of prediction

In [73]:
```
lstmdf=closedf.tolist()
lstmdf.extend((np.array(lst_output).reshape(-1,1)).tolist())
lstmdf=scaler.inverse_transform(lstmdf).reshape(1,-1).tolist()[0]

names = cycle(['Close price'])

fig = px.line(lstmdf,labels={'value': 'Stock price','index': 'Timestamp'})
fig.update_layout(title_text='Plotting whole closing stock price with prediction',
                  plot_bgcolor='white', font_size=15, font_color='black',legend_title_text='Stock')

fig.for_each_trace(lambda t:  t.update(name = next(names)))

fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
```


Plotting whole closing stock price with prediction
Stock
Clc
Stock price
40k
35k
30k
25k
20k
15k
0
50
100
150
200
250
300
350

In [ ]:

In [ ]: