In [3]:
```
import math
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM
import matplotlib.pyplot as plt
```

In [7]:
```
#dataframe= https://github.com/Amritpal-001/Stock-price-predicition/blob/master/csv%20files/individual_stocks_5yr/IBM
```

In [8]:
```
df=pd.read_csv("IBM_data.csv")
```

In [9]:
```
df.head()
```

Out[9]:

	date	open	high	low	close	volume	Name
0	2013-02-08	199.97	202.090	199.68	201.68	2893254	IBM
1	2013-02-11	200.98	201.950	199.75	200.16	2944651	IBM
2	2013-02-12	200.01	200.735	199.02	200.04	2461779	IBM
3	2013-02-13	200.65	200.950	199.57	200.09	2169757	IBM
4	2013-02-14	199.73	200.320	199.26	199.65	3294126	IBM

In [10]:
```
df.tail()
```

Out[10]:

	date	open	high	low	close	volume	Name
1254	2018-02-01	163.19	164.13	161.9000	162.40	4434242	IBM
1255	2018-02-02	161.70	162.00	158.8663	159.03	5251938	IBM
1256	2018-02-05	157.89	158.50	150.0000	152.53	8746599	IBM
1257	2018-02-06	150.29	155.49	149.1100	155.34	9867678	IBM
1258	2018-02-07	154.17	155.34	153.2800	153.85	6149207	IBM

In [11]:
```
df.describe()
```

Out[11]:

	open	high	low	close	volume
count	1259.000000	1259.000000	1259.000000	1259.000000	1.259000e+03
mean	167.230871	168.362928	166.156247	167.261926	4.352535e+06
std	20.184908	20.257137	20.157675	20.207108	2.346671e+06
min	118.460000	119.660000	116.901000	117.850000	1.193025e+06
25%	152.400000	153.318950	151.594600	152.435000	3.067106e+06
50%	162.650000	163.905000	161.750000	162.670000	3.804943e+06
75%	184.555000	185.730000	183.535000	184.365000	4.828166e+06
max	215.380000	215.900000	214.300000	215.800000	3.049019e+07

In [13]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1259 entries, 0 to 1258
Data columns (total 7 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   date    1259 non-null   object 
 1   open    1259 non-null   float64
 2   high    1259 non-null   float64
 3   low     1259 non-null   float64
 4   close   1259 non-null   float64
 5   volume  1259 non-null   int64  
 6   Name    1259 non-null   object 
dtypes: float64(4), int64(1), object(2)
memory usage: 69.0+ KB
```

In [14]:
```
pd.isnull(df)
```

Out[14]:

	date	open	high	low	close	volume	Name
0	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...
1254	False	False	False	False	False	False	False
1255	False	False	False	False	False	False	False
1256	False	False	False	False	False	False	False
1257	False	False	False	False	False	False	False
1258	False	False	False	False	False	False	False

1259 rows × 7 columns

In [16]:
```
pd.isnull(df).sum()
```

Out[16]:
```
date      0
open      0
high      0
low       0
close     0
volume    0
Name      0
dtype: int64
```

In [18]:
```
df.dropna(inplace=True)
```

In [19]:
```
df.shape
```

Out[19]:
```
(1259, 7)
```

In [20]:
```
df.columns
```

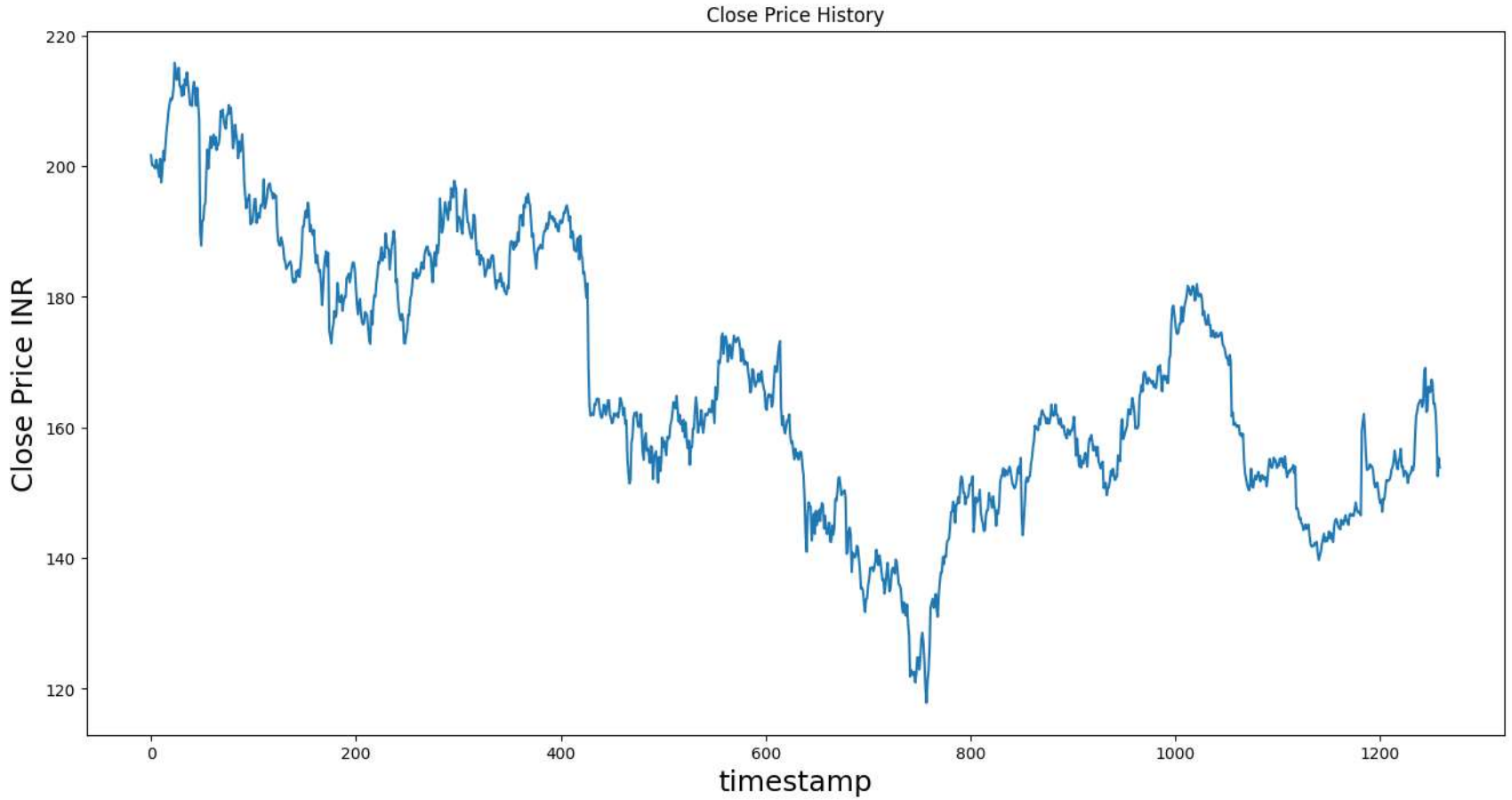
Out[20]:
```
Index(['date', 'open', 'high', 'low', 'close', 'volume', 'Name'], dtype='object')
```

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

Out[22]:

	open	high	low	close	volume
count	1259.000000	1259.000000	1259.000000	1259.000000	1.259000e+03
mean	167.230871	168.362928	166.156247	167.261926	4.352535e+06
std	20.184908	20.257137	20.157675	20.207108	2.346671e+06
min	118.460000	119.660000	116.901000	117.850000	1.193025e+06
25%	152.400000	153.318950	151.594600	152.435000	3.067106e+06
50%	162.650000	163.905000	161.750000	162.670000	3.804943e+06
75%	184.555000	185.730000	183.535000	184.365000	4.828166e+06
max	215.380000	215.900000	214.300000	215.800000	3.049019e+07

In [23]:
```
import seaborn as sns
plt.figure(figsize=(16,8))
plt.title('Close Price History')
plt.plot(df['close'])
#ax=sns.lineplot(data=df, x='timestamp',y='close', color="blue");
plt.xlabel('timestamp',fontsize=18)
plt.ylabel('Close Price INR',fontsize=18)
plt.show()
```



In [24]:
```
data = df.filter(['close'])
dataset = data.values
training_data_len = math.ceil( len(dataset) *.8)
```

In [25]:
```
dataset
```

Out[25]:
```
array([[201.68],
       [200.16],
       [200.04],
       ...,
       [152.53],
       [155.34],
       [153.85]])
```

In [26]:
```
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(dataset)
```

In [30]:
```
train_data = scaled_data[0:training_data_len  , : ]
x_train=[]
y_train = []
for i in range(60,len(train_data)):
    x_train.append(train_data[i-60:i,0])
    y_train.append(train_data[i,0])
```

In [32]:
```
x_train, y_train = np.array(x_train), np.array(y_train)
x_train = np.reshape(x_train, (x_train.shape[0],x_train.shape[1],1))
```

In [53]:
```
#Building a LSTM Model for Stock Market Prediction
model = Sequential()
model.add(LSTM(units=50, return_sequences=True,input_shape=(x_train.shape[1],1)))
model.add(LSTM(units=50, return_sequences=False))
model.add(Dense(units=25))
model.add(Dense(units=1))
```

In [54]:
```
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_10 (LSTM)              (None, 60, 50)            10400

 lstm_11 (LSTM)              (None, 50)                20200

 dense_10 (Dense)            (None, 25)                1275

 dense_11 (Dense)            (None, 1)                 26

=================================================================
Total params: 31901 (124.61 KB)
Trainable params: 31901 (124.61 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [55]:
```
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(x_train, y_train, batch_size=64, epochs=1)
```

```
15/15 [==============================] - 11s 125ms/step - loss: 0.0839
```

Out[55]:
```
<keras.src.callbacks.History at 0x2667308ddd0>
```

In [62]:
```
 test_data = scaled_data[training_data_len - 60: , : ]#Create the x_test and y_test data sets
x_test = []
y_test =  dataset[training_data_len : , : ] #Get all of the rows from index 1603 to the rest and all of the columns (
for i in range(60,len(test_data)):
    x_test.append(test_data[i-60:i,0])
```

In [63]:
```
x_test = np.array(x_test)
```

In [64]:
```
x_test = np.reshape(x_test, (x_test.shape[0],x_test.shape[1],1))
predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)
```

```
8/8 [==============================] - 2s 16ms/step
```

In [65]:
```
#Finding the root mean squared error
rmse=np.sqrt(np.mean(((predictions- y_test)**2)))
rmse
```

Out[65]:
```
6.095391869365111
```

In [66]:
```
train = data[:training_data_len]
display = data[training_data_len:]
display['Predictions'] = predictions#Visualize the data
plt.figure(figsize=(16,8))
plt.title('Model')
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price INR', fontsize=18)
plt.plot(train['close'])
plt.plot(display['close'])
plt.plot(display['Predictions'])
plt.legend(['Train', 'Val', 'Predictions'], loc='upper right')
plt.show()
```

```
C:\Users\Kavisha\AppData\Local\Temp\ipykernel_138972\2702570860.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  display['Predictions'] = predictions#Visualize the data
```