



## ▼ \*STOCK MARKET PRICE PREDICTION MODEL \*

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
```

```
df = pd.read_csv('INR=X.csv')
```

## ▼ DATA FEATURING AND PREPROCESSING

```
df.head()
```

|   | Date       | Open      | High      | Low       | Close     | Adj Close | Volume |  |  |
|---|------------|-----------|-----------|-----------|-----------|-----------|--------|---|---|
| 0 | 2020-01-27 | 71.320000 | 71.635002 | 71.320000 | 71.324997 | 71.324997 | 0      |   |   |
| 1 | 2020-01-28 | 71.654999 | 71.654999 | 71.178001 | 71.440002 | 71.440002 | 0      |   |   |
| 2 | 2020-01-29 | 71.230103 | 71.425003 | 71.168503 | 71.230400 | 71.230400 | 0      |   |   |
| 3 | 2020-01-30 | 71.300003 | 71.711998 | 71.300003 | 71.300003 | 71.300003 | 0      |   |   |
| 4 | 2020-01-31 | 71.639999 | 71.639999 | 71.277496 | 71.639999 | 71.639999 | 0      |   |   |

```
#converting date column in time frame series
df['Date'] = pd.to_datetime(df['Date'])
```

```
df.isnull().sum()
```

```
Date      0
Open      0
High      0
Low       0
Close     0
Adj Close 0
Volume    0
dtype: int64
```

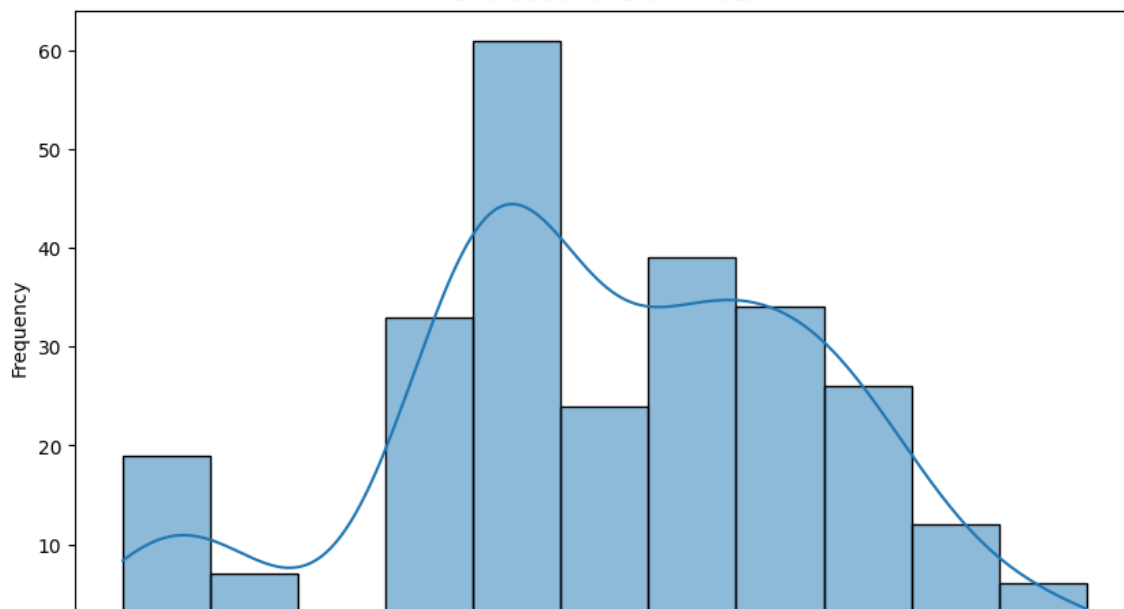
```
df.describe()
```

|       | Open       | High       | Low        | Close      | Adj Close  | Volume |  |  |
|-------|------------|------------|------------|------------|------------|--------|---|---|
| count | 262.000000 | 262.000000 | 262.000000 | 262.000000 | 262.000000 | 262.0  |   |   |
| mean  | 74.373533  | 74.631087  | 74.011048  | 74.358489  | 74.358489  | 0.0    |   |   |
| std   | 1.417620   | 1.494521   | 1.311330   | 1.426245   | 1.426245   | 0.0    |   |   |
| min   | 71.100403  | 71.279999  | 71.064003  | 71.099998  | 71.099998  | 0.0    |   |   |
| 25%   | 73.546175  | 73.706577  | 73.202003  | 73.531049  | 73.531049  | 0.0    |   |   |
| 50%   | 74.332001  | 74.531300  | 73.881748  | 74.275799  | 74.275799  | 0.0    |   |   |
| 75%   | 75.484551  | 75.737499  | 75.067053  | 75.489424  | 75.489424  | 0.0    |   |   |
| max   | 77.684998  | 77.754997  | 76.496300  | 77.570000  | 77.570000  | 0.0    |   |   |

## ▼ EXPLORATORY DATA ANALYSIS

```
# Visualize the distribution of the stock prices
plt.figure(figsize=(10, 6))
sns.histplot(df['Close'], kde=True)
plt.xlabel('Stock Price')
plt.ylabel('Frequency')
plt.title('Distribution of Stock Prices')
plt.show()
```

Distribution of Stock Prices



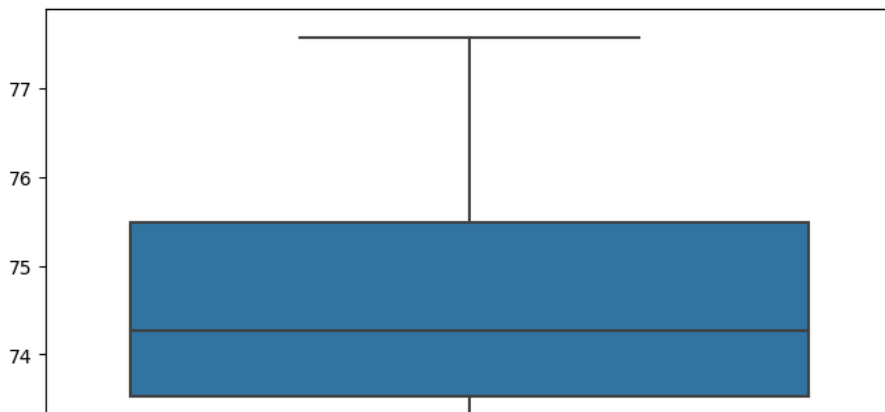
```
# Visualize the trend of stock prices over time
plt.figure(figsize=(12, 6))
plt.plot(df['Date'], df['Close'])
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.title('Stock Prices Over Time')
plt.xticks(rotation=45)
plt.show()
```

Stock Prices Over Time



```
# Check for outliers using box plots
plt.figure(figsize=(8, 6))
sns.boxplot(df['Close'])
plt.xlabel('Stock Price')
plt.title('Box Plot of Stock Prices')
plt.show()
```

Box Plot of Stock Prices

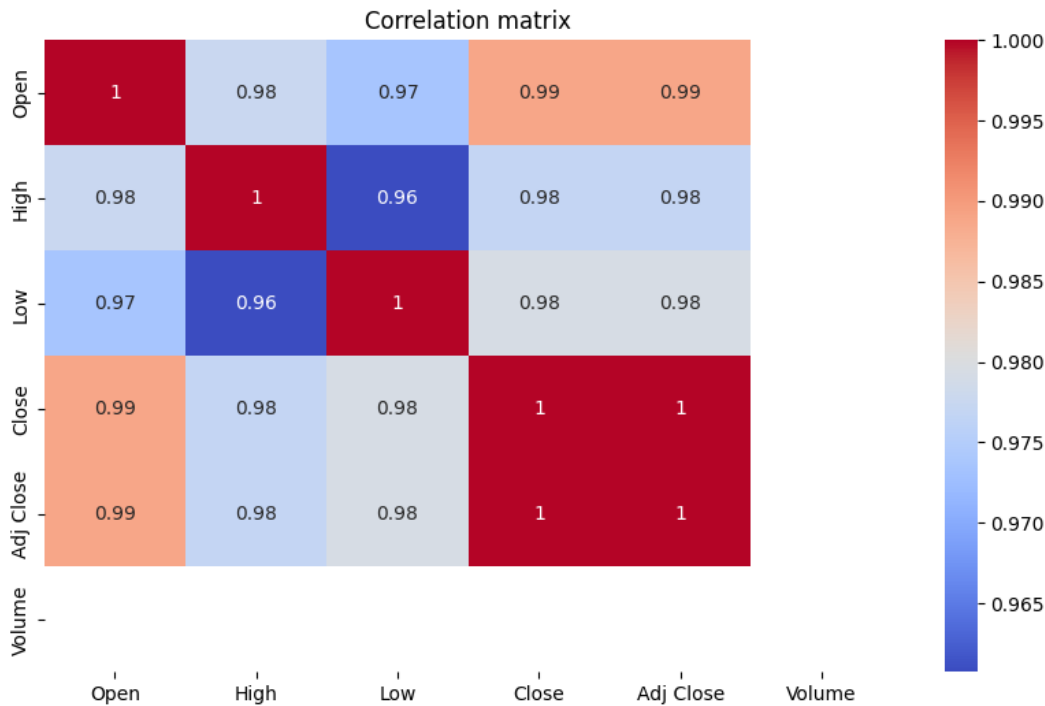


## FEATURE SELECTION

```
corr_matrix = df.corr()

plt.figure(figsize=(10,6))
sns.heatmap(corr_matrix,annot=True,cmap='coolwarm')
plt.title('Correlation matrix')
plt.show()
```

<ipython-input-39-4c8c71f27190>:1: FutureWarning: The default value of numeric\_only in DataFrame.corr is deprecated. In a f  
corr\_matrix = df.corr()



df.dtypes

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

```
df['Date'] = df['Date'].astype(np.int64)
```

```
target = 'Close'
```

```
X = df.drop(target, axis=1)
y = df[target]
model = RandomForestRegressor()
model.fit(X, y)
importances = model.feature_importances_
```

```
# Sorting features as per importance
```

```

indices = np.argsort(importances)[-10:]
sorted_features = X.columns[indices]

# Printing the features importance scores
for feature, importance in zip(sorted_features, importances[indices]):
    print(f'{feature}: {importance}')

# Selecting top k features based on importance
k = 5 # taking out the desired number of features
selected_features = sorted_features[:k]
print(f'Selected Features: {selected_features}')

Adj Close: 0.8228316332548671
Low: 0.057391179439155504
Date: 0.04930431766387224
Open: 0.0468803950981489
High: 0.023592474543956344
Volume: 0.0
Selected Features: Index(['Adj Close', 'Low', 'Date', 'Open', 'High'], dtype='object')

```

## ▼ SPLITTING THE DATA

```

scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(df['Close'].values.reshape(-1, 1))

# Splitting the data into training and testing sets
train_size = int(len(scaled_data) * 0.8)
train_data = scaled_data[:train_size]
test_data = scaled_data[train_size:]

# Creating sequences and labels for training
def create_sequences(data, sequence_length):
    sequences = []
    labels = []
    for i in range(len(data) - sequence_length):
        sequences.append(data[i:i+sequence_length])
        labels.append(data[i+sequence_length])
    return np.array(sequences), np.array(labels)

sequence_length = 30
train_sequences, train_labels = create_sequences(train_data, sequence_length)
test_sequences, test_labels = create_sequences(test_data, sequence_length)

# Reshaping the test_sequences to match the number of features
test_sequences = test_sequences.reshape(test_sequences.shape[0], test_sequences.shape[1])

```

## ▼ APPLYING LSTM

```

prices = df["Close"].values.reshape(-1, 1)

```

## ▼ TEST DATA PREDICTION VISUALISATION

```

# Building the LSTM model
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(sequence_length, 1)))
model.add(LSTM(50))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')

# model training
model.fit(train_sequences, train_labels, epochs=10, batch_size=32)

# model evaluation
mse = model.evaluate(test_sequences, test_labels)

# Making predictions
predicted_data = model.predict(test_sequences)
predicted_data = predicted_data.reshape(predicted_data.shape[0], 1)

# Inverse scaling for test data
predicted_prices = scaler.inverse_transform(predicted_data)
actual_prices = scaler.inverse_transform(test_labels)

```

```

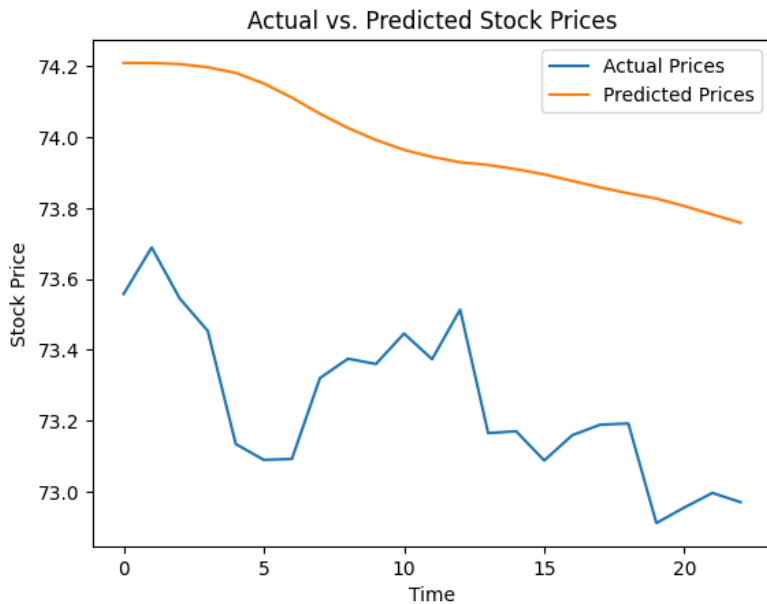
Epoch 1/10
6/6 [=====] - 7s 49ms/step - loss: 0.2148
Epoch 2/10
6/6 [=====] - 0s 46ms/step - loss: 0.0432
Epoch 3/10
6/6 [=====] - 0s 44ms/step - loss: 0.0168
Epoch 4/10
6/6 [=====] - 0s 42ms/step - loss: 0.0227
Epoch 5/10
6/6 [=====] - 0s 47ms/step - loss: 0.0136
Epoch 6/10
6/6 [=====] - 0s 49ms/step - loss: 0.0139
Epoch 7/10
6/6 [=====] - 0s 42ms/step - loss: 0.0116
Epoch 8/10
6/6 [=====] - 0s 48ms/step - loss: 0.0115
Epoch 9/10
6/6 [=====] - 0s 45ms/step - loss: 0.0103
Epoch 10/10
6/6 [=====] - 0s 37ms/step - loss: 0.0105
1/1 [=====] - 2s 2s/step - loss: 0.0136
1/1 [=====] - 1s 1s/step

```

```

plt.plot(actual_prices, label='Actual Prices')
plt.plot(predicted_prices, label='Predicted Prices')
plt.xlabel('Time')
plt.ylabel('Stock Price')
plt.title('Actual vs. Predicted Stock Prices')
plt.legend()
plt.show()

```



## ▼ FULL DATA PREDICTION VISUALISATION

```

import matplotlib.pyplot as plt

# Defining the sequence length
sequence_length = 50

# Create the sequences and labels
sequences, labels = create_sequences(scaled_data, sequence_length)

# Reshaping the input data
sequences = np.reshape(sequences, (sequences.shape[0], sequences.shape[1], 1))

# Defining the input shape for LSTM
input_shape = (sequences.shape[1], 1)

# Building the LSTM model
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=input_shape))
model.add(LSTM(units=50))
model.add(Dense(units=1))

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

# Training the model
model.fit(sequences, labels, epochs=10, batch_size=32)

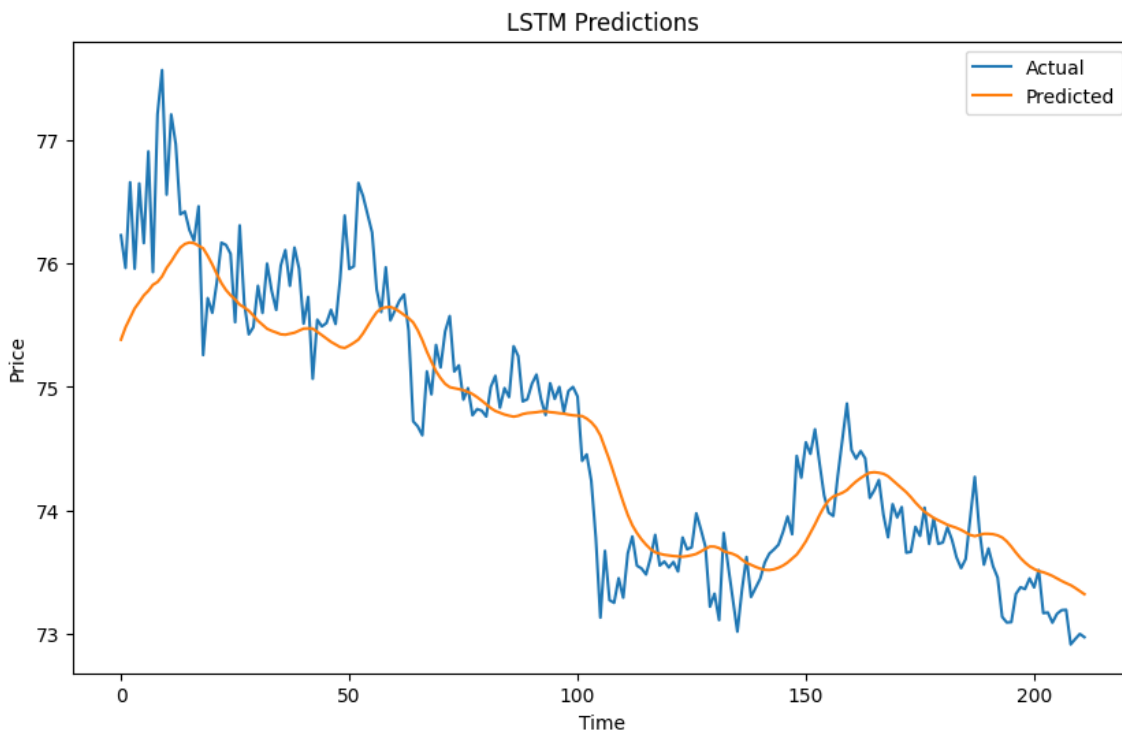
```

```
# Making predictions
predicted_fulldata = model.predict(sequences)
predicted_prices = scaler.inverse_transform(predicted_fulldata)
```

```
# Inverse scaling for the actual prices
actual_prices = scaler.inverse_transform(labels.reshape(-1, 1))
```

```
Epoch 1/10
7/7 [=====] - 7s 58ms/step - loss: 0.1984
Epoch 2/10
7/7 [=====] - 0s 52ms/step - loss: 0.0280
Epoch 3/10
7/7 [=====] - 0s 59ms/step - loss: 0.0114
Epoch 4/10
7/7 [=====] - 1s 104ms/step - loss: 0.0128
Epoch 5/10
7/7 [=====] - 1s 102ms/step - loss: 0.0075
Epoch 6/10
7/7 [=====] - 1s 105ms/step - loss: 0.0071
Epoch 7/10
7/7 [=====] - 1s 87ms/step - loss: 0.0065
Epoch 8/10
7/7 [=====] - 0s 68ms/step - loss: 0.0055
Epoch 9/10
7/7 [=====] - 0s 52ms/step - loss: 0.0054
Epoch 10/10
7/7 [=====] - 0s 52ms/step - loss: 0.0052
WARNING:tensorflow:5 out of the last 17 calls to <function Model.make_predict_function.<locals>.predict_function at 0x7d2b5a5877f0> triggered
7/7 [=====] - 1s 19ms/step
```

```
# Plotting the actual and predicted prices
plt.figure(figsize=(10, 6))
plt.plot(actual_prices, label='Actual')
plt.plot(predicted_prices, label='Predicted')
plt.xlabel('Time')
plt.ylabel('Price')
plt.title('LSTM Predictions')
plt.legend()
plt.show()
```



✓ 1s completed at 11:28 PM

