Vinamilk (VNM) STOCK PREDICTION

1. | Import Necessary Library

In [ ]:
```
from vnstock3 import Vnstock
import mplfinance as mpf
import statsmodels.api as sm
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_s
import tensorflow as tf
from tensorflow.keras.layers import Dense, Dropout, Embedding, Bidirectio
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Sequential
import warnings
warnings.filterwarnings('ignore')
import plotly.io as pio
pio.renderers.default = "notebook_connected"
```

2. | Exploratory Data Analysis

In [ ]:
```
company = Vnstock().stock(symbol='VNM', source='TCBS').company
company.overview()
```

Out[ ]:

	exchange	industry	company_type	no_shareholders	foreign_percent	outstand
0	HOSE	Thực phẩm và đồ uống	CT	0	0.517	

In [ ]:
```
stock = Vnstock().stock(symbol='VNM', source='VCI')
df = stock.quote.history(start='2019-01-01', end='2024-11-22')
df.info()
df.head()
```

```
2024-12-04 13:36:35 - vnstock3.common.data.data_explorer - WARNING - Thông
tin niêm yết & giao dịch sẽ được truy xuất từ TCBS
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1473 entries, 0 to 1472
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   time    1473 non-null   datetime64[ns]
 1   open    1473 non-null   float64
 2   high    1473 non-null   float64
 3   low     1473 non-null   float64
 4   close   1473 non-null   float64
 5   volume  1473 non-null   int64
dtypes: datetime64[ns](1), float64(4), int64(1)
memory usage: 69.2 KB
```

Out[ ]:

	time	open	high	low	close	volume
0	2019-01-02	78.08	79.69	77.76	79.69	403570
1	2019-01-03	79.69	79.69	78.15	78.98	449730
2	2019-01-04	78.72	80.33	77.76	80.33	498470
3	2019-01-07	82.07	84.19	80.65	84.19	897430
4	2019-01-08	84.44	84.44	82.71	83.54	448350

In [ ]:

```python
print(df.isnull().sum())
```

```
time      0
open      0
high      0
low       0
close     0
volume    0
dtype: int64
```

In [ ]:

```python
df.describe(include='all')
```

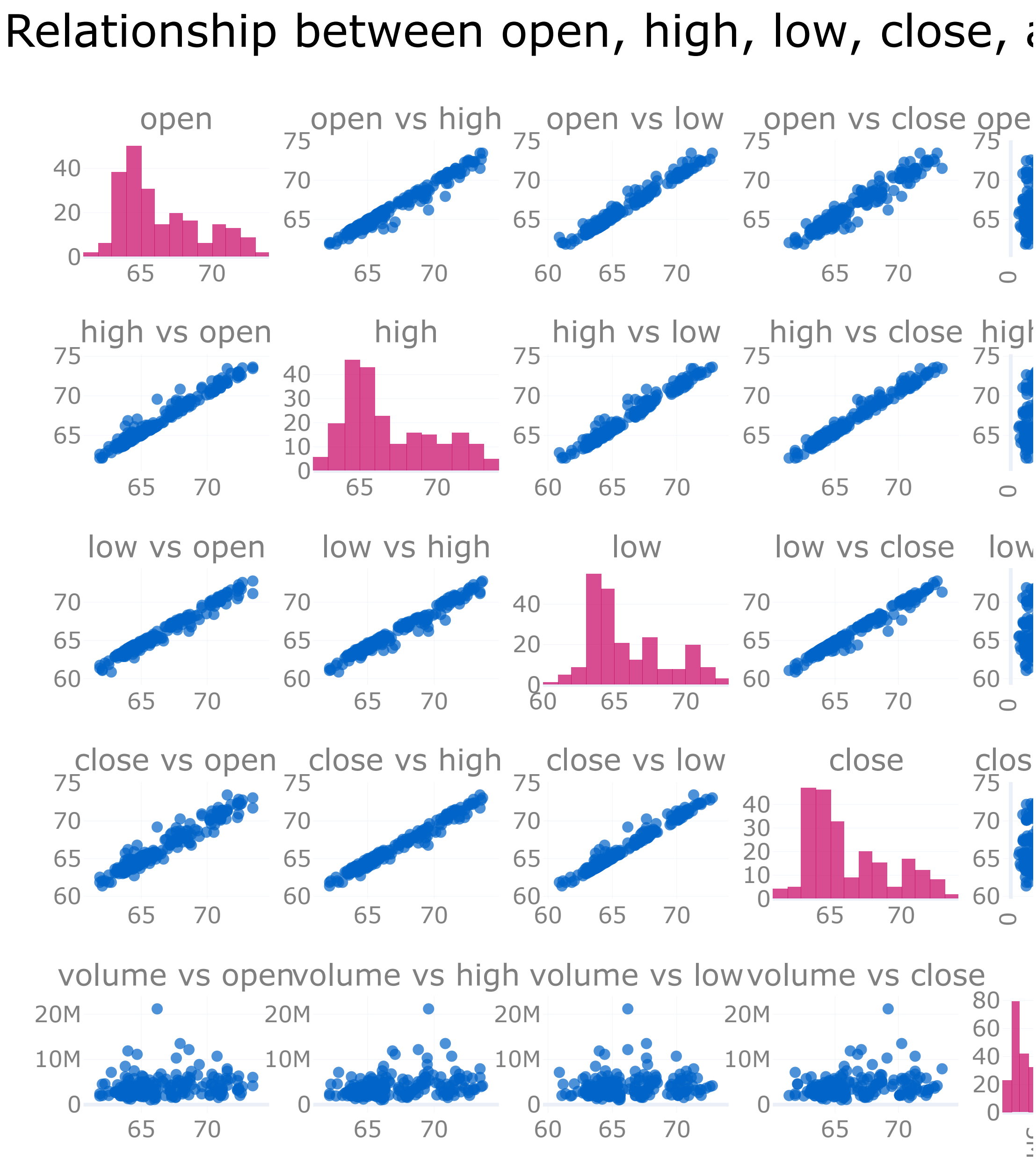
Out[ ]:

	time	open	high	low	close	
count	1473	1473.000000	1473.000000	1473.000000	1473.000000	1
mean	2021-12-14 14:21:15.763747328	73.870754	74.521704	73.213530	73.813327	2
min	2019-01-02 00:00:00	55.810000	57.870000	55.810000	55.810000	2
25%	2020-06-26 00:00:00	66.410000	67.000000	65.810000	66.290000	1
50%	2021-12-10 00:00:00	72.440000	73.200000	71.790000	72.360000	2
75%	2023-06-07 00:00:00	79.980000	80.630000	79.600000	80.050000	3
max	2024-11-22 00:00:00	96.810000	98.070000	95.900000	97.430000	
std	NaN	8.699546	8.705796	8.639016	8.693214	1

In [ ]:
```
#Filter VNM stock historical data from 01/2024 - 11/2024
#Convert time column
df_analyze = df.copy()
df_analyze['time'] = pd.to_datetime(df_analyze['time'])

#Set time column as index
df_analyze.set_index('time', inplace=True)

#Filter to Plotting chart
filtered_df = df_analyze.loc['2024-01-01':'2024-11-22']

#Plotting the candlestick chart for historical data from 01/01/2024 - 22/
mpf.plot(filtered_df, type='candle', volume=True, style='yahoo', title='V
         ylabel='Price', ylabel_lower='Volume', figsize=(14, 7))
```

In [ ]:
```
# Plot the closing price over time
fig = px.line(x= df_analyze.index, y= df_analyze['close'], title = 'Vinam
fig.update_layout(height=500, width=750, template="plotly_white", font_cc
fig.show()
```

In [ ]:
```
from plotly.subplots import make_subplots
# Dimensions for the scatterplot matrix
dimensions = ["open", "high", "low", "close", "volume"]

# Create a 5x5 subplot layout
fig = make_subplots(
    rows=len(dimensions), cols=len(dimensions),
    specs=[[{'type': 'scatter'}]*len(dimensions) for _ in range(len(dimer
    subplot_titles=[f"{y} vs {x}" if x != y else f"{x}" for y in dimensic
)

# Loop through dimensions to populate the matrix
for i, x in enumerate(dimensions):  # Iterate over columns
    for j, y in enumerate(dimensions):  # Iterate over rows
        if i != j:
            # Off-diagonal: scatter plot
            fig.add_trace(go.Scatter(x=filtered_df[x], y=filtered_df[y],
        else:
            # Diagonal: histogram
            fig.add_trace(go.Histogram(x=filtered_df[x], marker=dict(colc

# Update layout settings
fig.update_layout(height=700, width=750, template="plotly_white", title='
    font_color="grey", font_size=12, title_font_color="black", title_font

# Show the plot
fig.show()
```



3. | Modeling (Bidirectional LSTM)

3.1 Preprocessing

In [ ]:
```
df_data = df[['time','open','close']]
df_data.set_index('time',drop=True,inplace=True)
df_data.reset_index(inplace=True)
df_data.info()
df_data.head(10)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1473 entries, 0 to 1472
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   time    1473 non-null   datetime64[ns]
 1   open    1473 non-null   float64
 2   close   1473 non-null   float64
dtypes: datetime64[ns](1), float64(2)
memory usage: 34.7 KB
```

Out[ ]:

	time	open	close
0	2019-01-02	78.08	79.69
1	2019-01-03	79.69	78.98
2	2019-01-04	78.72	80.33
3	2019-01-07	82.07	84.19
4	2019-01-08	84.44	83.54
5	2019-01-09	83.54	84.77
6	2019-01-10	84.77	84.64
7	2019-01-11	84.64	86.11
8	2019-01-14	85.47	86.76
9	2019-01-15	86.76	86.69

In [ ]:

```
# Normalization Data
scaler = MinMaxScaler()
numerical_columns = ['open', 'close']
df_data[numerical_columns] = scaler.fit_transform(df_data[numerical_colum
```

In [ ]:

```
# Split Train - Test data
training_size = round(len(df_data) * 0.80)
train_data = df_data[:training_size]
test_data  = df_data[training_size:]

train_data.set_index('time',inplace=True)
test_data.set_index('time',inplace=True)

display(train_data.head())
display(test_data.tail())
display(test_data.shape, train_data.shape)
```

	open	close
time		
2019-01-02	0.543171	0.573763
2019-01-03	0.582439	0.556704
2019-01-04	0.558780	0.589140
2019-01-07	0.640488	0.681884
2019-01-08	0.698293	0.666266

	open	close
time		
2024-11-18	0.192439	0.177559
2024-11-19	0.182683	0.172753
2024-11-20	0.175366	0.187170
2024-11-21	0.192439	0.194378
2024-11-22	0.197317	0.203988

```
(295, 2)
(1178, 2)
```

In [ ]:
```
def create_sequence(dataset):
  sequences = []
  labels = []
  start_idx = 0
  for stop_idx in range(50,len(dataset)):
    sequences.append(dataset.iloc[start_idx:stop_idx])
    labels.append(dataset.iloc[stop_idx])
    start_idx += 1
  return (np.array(sequences),np.array(labels))

train_seq, train_label = create_sequence(train_data)
test_seq, test_label = create_sequence(test_data)

display(train_seq.shape, test_seq.shape)
```

```
(1128, 50, 2)
(245, 50, 2)
```

3.2 Define Bidirectional LSTM

In [ ]:
```
def build_BiLSTM_model():
    input = tf.keras.layers.Input(
        shape=(train_seq.shape[1], train_seq.shape[2]), name ="input"
    )
    x = tf.keras.layers.Bidirectional(LSTM(512, return_sequences=True))(i

    x = tf.keras.layers.Bidirectional(LSTM(316, return_sequences=False, d

    x = tf.keras.layers.Dense(128 , activation="relu", name ="dense_1")(x

    output = tf.keras.layers.Dense(2, name="last_dense")(x)

    model = tf.keras.Model(inputs=input, outputs=output)
    model.compile(loss='mean_squared_error', optimizer='adam', metrics=['

    return model

LSTM_model = build_BiLSTM_model()
LSTM_model.summary()
```

Model: "functional"

Layer (type)	Output Shape	I
input (InputLayer)	(None, 50, 2)	
bidirectional (Bidirectional)	(None, 50, 1024)	2,:
bidirectional_1 (Bidirectional)	(None, 632)	3,:
dense_1 (Dense)	(None, 128)	
last_dense (Dense)	(None, 2)	

Total params: 5,580,770 (21.29 MB)
Trainable params: 5,580,770 (21.29 MB)
Non-trainable params: 0 (0.00 B)

3.3 Training Data

In [ ]:
```
batch_size = 20
early_stopping_patience = 17
LSTM_model.fit(train_seq, train_label, epochs=80,validation_data=(test_se
```

```
Epoch 1/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 66s 2s/step - loss: 0.0847 - mean_absolute_erro
r: 0.1927 - val_loss: 0.0016 - val_mean_absolute_error: 0.0310
Epoch 2/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 50s 1s/step - loss: 0.0038 - mean_absolute_erro
r: 0.0472 - val_loss: 8.6478e-04 - val_mean_absolute_error: 0.0230
Epoch 3/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 45s 1s/step - loss: 0.0024 - mean_absolute_erro
r: 0.0376 - val_loss: 0.0014 - val_mean_absolute_error: 0.0329
Epoch 4/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 55s 2s/step - loss: 0.0028 - mean_absolute_erro
r: 0.0409 - val_loss: 6.1850e-04 - val_mean_absolute_error: 0.0180
Epoch 5/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 52s 1s/step - loss: 0.0020 - mean_absolute_erro
r: 0.0337 - val_loss: 5.6969e-04 - val_mean_absolute_error: 0.0177
Epoch 6/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 54s 2s/step - loss: 0.0021 - mean_absolute_erro
r: 0.0341 - val_loss: 6.1665e-04 - val_mean_absolute_error: 0.0195
Epoch 7/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 51s 1s/step - loss: 0.0018 - mean_absolute_erro
r: 0.0327 - val_loss: 6.3715e-04 - val_mean_absolute_error: 0.0204
Epoch 8/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 50s 1s/step - loss: 0.0015 - mean_absolute_erro
r: 0.0292 - val_loss: 5.3048e-04 - val_mean_absolute_error: 0.0168
Epoch 9/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 53s 1s/step - loss: 0.0016 - mean_absolute_erro
r: 0.0306 - val_loss: 5.0405e-04 - val_mean_absolute_error: 0.0164
Epoch 10/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 52s 1s/step - loss: 0.0016 - mean_absolute_erro
r: 0.0302 - val_loss: 4.1338e-04 - val_mean_absolute_error: 0.0149
Epoch 11/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 53s 1s/step - loss: 0.0016 - mean_absolute_erro
r: 0.0309 - val_loss: 4.6872e-04 - val_mean_absolute_error: 0.0158
Epoch 12/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 48s 1s/step - loss: 0.0017 - mean_absolute_erro
r: 0.0314 - val_loss: 3.6026e-04 - val_mean_absolute_error: 0.0138
Epoch 13/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 44s 1s/step - loss: 0.0015 - mean_absolute_erro
r: 0.0291 - val_loss: 3.8496e-04 - val_mean_absolute_error: 0.0142
Epoch 14/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 54s 2s/step - loss: 0.0011 - mean_absolute_erro
r: 0.0249 - val_loss: 3.0202e-04 - val_mean_absolute_error: 0.0124
Epoch 15/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 42s 1s/step - loss: 0.0011 - mean_absolute_erro
r: 0.0250 - val_loss: 3.7302e-04 - val_mean_absolute_error: 0.0145
Epoch 16/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 42s 1s/step - loss: 0.0014 - mean_absolute_erro
r: 0.0285 - val_loss: 3.4260e-04 - val_mean_absolute_error: 0.0139
Epoch 17/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 42s 1s/step - loss: 0.0011 - mean_absolute_erro
r: 0.0252 - val_loss: 3.3263e-04 - val_mean_absolute_error: 0.0136
Epoch 18/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 42s 1s/step - loss: 0.0011 - mean_absolute_erro
r: 0.0242 - val_loss: 4.0172e-04 - val_mean_absolute_error: 0.0158
Epoch 19/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 42s 1s/step - loss: 9.6122e-04 - mean_absolute_
error: 0.0231 - val_loss: 2.4665e-04 - val_mean_absolute_error: 0.0114
Epoch 20/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 42s 1s/step - loss: 0.0011 - mean_absolute_erro
r: 0.0247 - val_loss: 2.9030e-04 - val_mean_absolute_error: 0.0121
```

```
Epoch 21/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 42s 1s/step - loss: 0.0011 - mean_absolute_erro
r: 0.0243 - val_loss: 2.4649e-04 - val_mean_absolute_error: 0.0114
Epoch 22/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - loss: 8.9314e-04 - mean_absolute_
error: 0.0221 - val_loss: 2.3326e-04 - val_mean_absolute_error: 0.0111
Epoch 23/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - loss: 8.5615e-04 - mean_absolute_
error: 0.0215 - val_loss: 3.3937e-04 - val_mean_absolute_error: 0.0140
Epoch 24/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - loss: 0.0011 - mean_absolute_erro
r: 0.0243 - val_loss: 2.1832e-04 - val_mean_absolute_error: 0.0102
Epoch 25/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - loss: 8.6919e-04 - mean_absolute_
error: 0.0218 - val_loss: 2.2005e-04 - val_mean_absolute_error: 0.0102
Epoch 26/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 148s 4s/step - loss: 8.0523e-04 - mean_absolute
_error: 0.0208 - val_loss: 5.1243e-04 - val_mean_absolute_error: 0.0198
Epoch 27/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 42s 1s/step - loss: 8.9428e-04 - mean_absolute_
error: 0.0225 - val_loss: 2.2217e-04 - val_mean_absolute_error: 0.0101
Epoch 28/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 42s 1s/step - loss: 7.8200e-04 - mean_absolute_
error: 0.0198 - val_loss: 2.6589e-04 - val_mean_absolute_error: 0.0115
Epoch 29/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 47s 1s/step - loss: 8.5575e-04 - mean_absolute_
error: 0.0213 - val_loss: 4.0100e-04 - val_mean_absolute_error: 0.0163
Epoch 30/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - loss: 8.4289e-04 - mean_absolute_
error: 0.0214 - val_loss: 2.2104e-04 - val_mean_absolute_error: 0.0102
Epoch 31/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 42s 1s/step - loss: 7.8386e-04 - mean_absolute_
error: 0.0207 - val_loss: 2.1134e-04 - val_mean_absolute_error: 0.0105
Epoch 32/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 42s 1s/step - loss: 7.8216e-04 - mean_absolute_
error: 0.0204 - val_loss: 3.9510e-04 - val_mean_absolute_error: 0.0164
Epoch 33/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 42s 1s/step - loss: 7.0554e-04 - mean_absolute_
error: 0.0198 - val_loss: 2.0725e-04 - val_mean_absolute_error: 0.0099
Epoch 34/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 42s 1s/step - loss: 7.0309e-04 - mean_absolute_
error: 0.0191 - val_loss: 2.2576e-04 - val_mean_absolute_error: 0.0104
Epoch 35/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 45s 1s/step - loss: 8.3370e-04 - mean_absolute_
error: 0.0211 - val_loss: 2.2169e-04 - val_mean_absolute_error: 0.0105
Epoch 36/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 41s 1s/step - loss: 7.9471e-04 - mean_absolute_
error: 0.0195 - val_loss: 1.9735e-04 - val_mean_absolute_error: 0.0096
Epoch 37/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - loss: 7.9491e-04 - mean_absolute_
error: 0.0209 - val_loss: 1.9928e-04 - val_mean_absolute_error: 0.0097
Epoch 38/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - loss: 6.6708e-04 - mean_absolute_
error: 0.0188 - val_loss: 2.9345e-04 - val_mean_absolute_error: 0.0134
Epoch 39/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 44s 1s/step - loss: 7.6442e-04 - mean_absolute_
error: 0.0203 - val_loss: 2.0824e-04 - val_mean_absolute_error: 0.0103
Epoch 40/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 58s 2s/step - loss: 6.0804e-04 - mean_absolute_
error: 0.0181 - val_loss: 1.8088e-04 - val_mean_absolute_error: 0.0087
```

```
Epoch 41/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - loss: 6.5093e-04 - mean_absolute_
error: 0.0185 - val_loss: 2.4430e-04 - val_mean_absolute_error: 0.0114
Epoch 42/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - loss: 9.2134e-04 - mean_absolute_
error: 0.0224 - val_loss: 2.6503e-04 - val_mean_absolute_error: 0.0129
Epoch 43/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - loss: 7.1785e-04 - mean_absolute_
error: 0.0196 - val_loss: 2.7791e-04 - val_mean_absolute_error: 0.0129
Epoch 44/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - loss: 7.0877e-04 - mean_absolute_
error: 0.0199 - val_loss: 2.7300e-04 - val_mean_absolute_error: 0.0119
Epoch 45/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - loss: 7.6886e-04 - mean_absolute_
error: 0.0198 - val_loss: 2.3211e-04 - val_mean_absolute_error: 0.0104
Epoch 46/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 44s 1s/step - loss: 6.9791e-04 - mean_absolute_
error: 0.0186 - val_loss: 2.3554e-04 - val_mean_absolute_error: 0.0110
Epoch 47/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - loss: 6.6093e-04 - mean_absolute_
error: 0.0188 - val_loss: 2.3564e-04 - val_mean_absolute_error: 0.0120
Epoch 48/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - loss: 7.8752e-04 - mean_absolute_
error: 0.0211 - val_loss: 1.8731e-04 - val_mean_absolute_error: 0.0090
Epoch 49/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - loss: 6.2780e-04 - mean_absolute_
error: 0.0179 - val_loss: 3.9285e-04 - val_mean_absolute_error: 0.0164
Epoch 50/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - loss: 6.0595e-04 - mean_absolute_
error: 0.0180 - val_loss: 1.8783e-04 - val_mean_absolute_error: 0.0090
Epoch 51/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - loss: 7.0762e-04 - mean_absolute_
error: 0.0193 - val_loss: 2.0312e-04 - val_mean_absolute_error: 0.0092
Epoch 52/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 46s 1s/step - loss: 8.2688e-04 - mean_absolute_
error: 0.0218 - val_loss: 2.1467e-04 - val_mean_absolute_error: 0.0109
Epoch 53/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 52s 1s/step - loss: 8.4950e-04 - mean_absolute_
error: 0.0215 - val_loss: 2.1890e-04 - val_mean_absolute_error: 0.0103
Epoch 54/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 42s 1s/step - loss: 6.5526e-04 - mean_absolute_
error: 0.0188 - val_loss: 1.9742e-04 - val_mean_absolute_error: 0.0091
Epoch 55/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - loss: 6.6678e-04 - mean_absolute_
error: 0.0180 - val_loss: 2.5499e-04 - val_mean_absolute_error: 0.0121
Epoch 56/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - loss: 7.2529e-04 - mean_absolute_
error: 0.0197 - val_loss: 2.7022e-04 - val_mean_absolute_error: 0.0123
Epoch 57/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - loss: 7.3742e-04 - mean_absolute_
error: 0.0198 - val_loss: 2.1971e-04 - val_mean_absolute_error: 0.0100
Epoch 58/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - loss: 6.2083e-04 - mean_absolute_
error: 0.0179 - val_loss: 2.2117e-04 - val_mean_absolute_error: 0.0102
Epoch 59/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - loss: 5.9521e-04 - mean_absolute_
error: 0.0177 - val_loss: 2.3059e-04 - val_mean_absolute_error: 0.0104
Epoch 60/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 83s 1s/step - loss: 7.1195e-04 - mean_absolute_
error: 0.0194 - val_loss: 3.1769e-04 - val_mean_absolute_error: 0.0145
```

```
Epoch 61/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 47s 1s/step - loss: 6.4220e-04 - mean_absolute_
error: 0.0186 - val_loss: 2.5760e-04 - val_mean_absolute_error: 0.0121
Epoch 62/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 44s 1s/step - loss: 6.8278e-04 - mean_absolute_
error: 0.0187 - val_loss: 2.6474e-04 - val_mean_absolute_error: 0.0118
Epoch 63/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 44s 1s/step - loss: 8.4559e-04 - mean_absolute_
error: 0.0217 - val_loss: 1.7884e-04 - val_mean_absolute_error: 0.0084
Epoch 64/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - loss: 5.3990e-04 - mean_absolute_
error: 0.0163 - val_loss: 2.1717e-04 - val_mean_absolute_error: 0.0101
Epoch 65/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - loss: 6.4823e-04 - mean_absolute_
error: 0.0181 - val_loss: 2.9531e-04 - val_mean_absolute_error: 0.0135
Epoch 66/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 1149s 33s/step - loss: 6.6917e-04 - mean_absol
ute_error: 0.0190 - val_loss: 2.0730e-04 - val_mean_absolute_error: 0.0099
Epoch 67/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 47s 1s/step - loss: 6.6803e-04 - mean_absolute_
error: 0.0185 - val_loss: 2.0730e-04 - val_mean_absolute_error: 0.0092
Epoch 68/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - loss: 6.8554e-04 - mean_absolute_
error: 0.0181 - val_loss: 1.9693e-04 - val_mean_absolute_error: 0.0098
Epoch 69/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - loss: 6.3015e-04 - mean_absolute_
error: 0.0179 - val_loss: 2.1984e-04 - val_mean_absolute_error: 0.0101
Epoch 70/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - loss: 6.2842e-04 - mean_absolute_
error: 0.0185 - val_loss: 4.0819e-04 - val_mean_absolute_error: 0.0167
Epoch 71/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - loss: 6.6370e-04 - mean_absolute_
error: 0.0189 - val_loss: 1.9972e-04 - val_mean_absolute_error: 0.0096
Epoch 72/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - loss: 7.8786e-04 - mean_absolute_
error: 0.0205 - val_loss: 4.0659e-04 - val_mean_absolute_error: 0.0169
Epoch 73/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 44s 1s/step - loss: 7.1966e-04 - mean_absolute_
error: 0.0194 - val_loss: 3.5757e-04 - val_mean_absolute_error: 0.0147
Epoch 74/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - loss: 7.4310e-04 - mean_absolute_
error: 0.0201 - val_loss: 1.8155e-04 - val_mean_absolute_error: 0.0085
Epoch 75/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - loss: 5.6573e-04 - mean_absolute_
error: 0.0166 - val_loss: 1.8583e-04 - val_mean_absolute_error: 0.0088
Epoch 76/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - loss: 5.7012e-04 - mean_absolute_
error: 0.0173 - val_loss: 2.0073e-04 - val_mean_absolute_error: 0.0097
Epoch 77/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 44s 1s/step - loss: 6.1334e-04 - mean_absolute_
error: 0.0173 - val_loss: 2.6859e-04 - val_mean_absolute_error: 0.0126
Epoch 78/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 47s 1s/step - loss: 5.9799e-04 - mean_absolute_
error: 0.0175 - val_loss: 2.3385e-04 - val_mean_absolute_error: 0.0110
Epoch 79/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - loss: 5.7978e-04 - mean_absolute_
error: 0.0170 - val_loss: 2.2740e-04 - val_mean_absolute_error: 0.0112
Epoch 80/80
36/36 ━━━━━━━━━━━━━━━━━━━━ 44s 1s/step - loss: 6.0592e-04 - mean_absolute_
error: 0.0179 - val_loss: 1.7903e-04 - val_mean_absolute_error: 0.0086
```

Out[ ]:
```
<keras.src.callbacks.history.History at 0x30f11fcd0>
```

3.4 Evaluate Model

In [ ]:
```
test_predicted = LSTM_model.predict(test_seq)
test_inverse_predicted = scaler.inverse_transform(test_predicted)

test_inverse_predicted[:5]
```

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 5s 360ms/step
```

Out[ ]:
```
array([[64.873665, 64.69259 ],
       [64.00423 , 63.799187],
       [64.67144 , 64.51376 ],
       [65.164474, 65.034805],
       [64.67219 , 64.50622 ]], dtype=float32)
```

In [ ]:
```
df_slice_data = pd.concat([df_data .iloc[-245:].copy(),pd.DataFrame(test_
df_slice_data[['open','close']] = scaler.inverse_transform(df_slice_data[
display(df_slice_data.head())
df_slice_data.info()
```

	time	open	close	open_predicted	close_predicted
1228	2023-11-30	64.77	63.83	64.873665	64.692589
1229	2023-12-01	64.39	64.77	64.004227	63.799187
1230	2023-12-04	64.87	65.15	64.671440	64.513763
1231	2023-12-05	65.15	64.49	65.164474	65.034805
1232	2023-12-06	64.49	64.87	64.672188	64.506218

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 245 entries, 1228 to 1472
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   time             245 non-null    datetime64[ns]
 1   open             245 non-null    float64
 2   close            245 non-null    float64
 3   open_predicted   245 non-null    float32
 4   close_predicted  245 non-null    float32
dtypes: datetime64[ns](1), float32(2), float64(2)
memory usage: 7.8 KB
```

In [ ]:
```
df_slice_data['time'] = pd.to_datetime(df_slice_data['time'])

#Set time column as index
df_slice_data.set_index('time', inplace=True)
df_slice_data
```

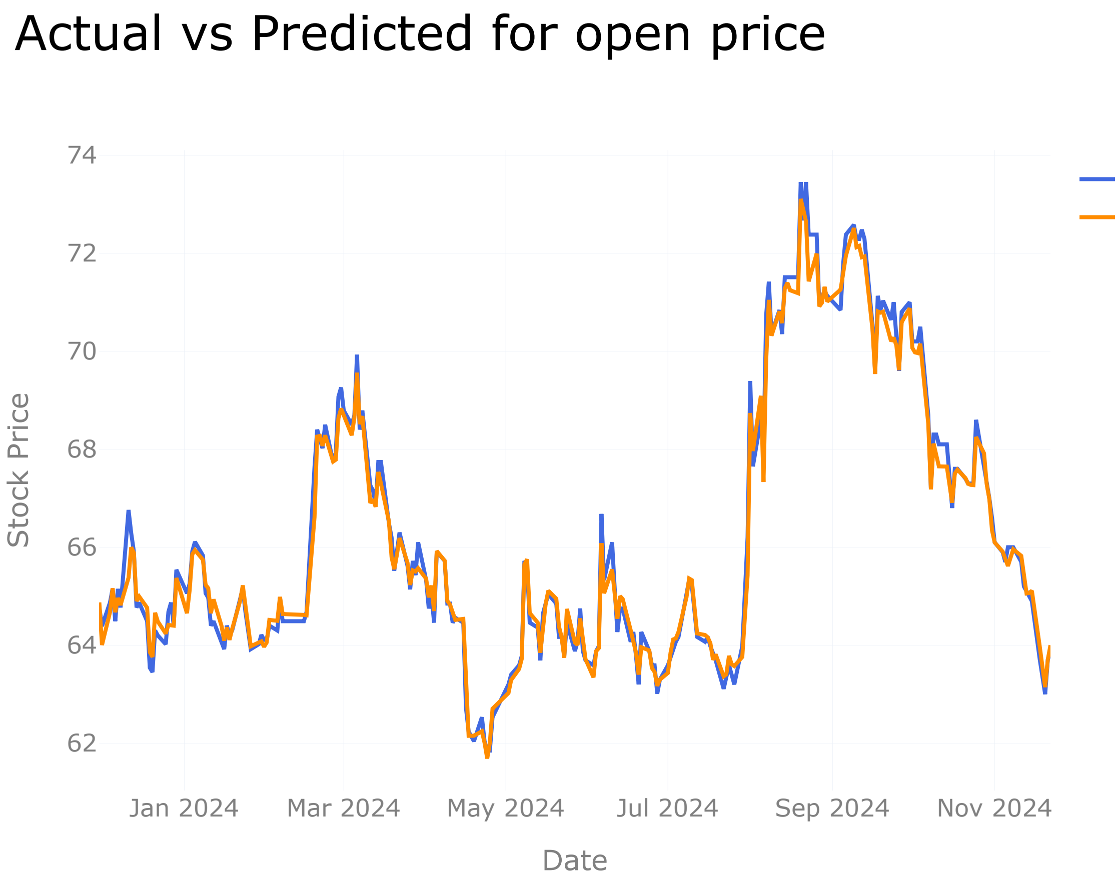
Out[ ]:

time	open	close	open_predicted	close_predicted
2023-11-30	64.77	63.83	64.873665	64.692589
2023-12-01	64.39	64.77	64.004227	63.799187
2023-12-04	64.87	65.15	64.671440	64.513763
2023-12-05	65.15	64.49	65.164474	65.034805
2023-12-06	64.49	64.87	64.672188	64.506218
...	...	...	...	...
2024-11-18	63.70	63.20	63.959431	63.757103
2024-11-19	63.30	63.00	63.396782	63.193104
2024-11-20	63.00	63.60	63.143879	62.945084
2024-11-21	63.70	63.90	63.707897	63.527714
2024-11-22	63.90	64.30	63.999714	63.835651

245 rows × 4 columns

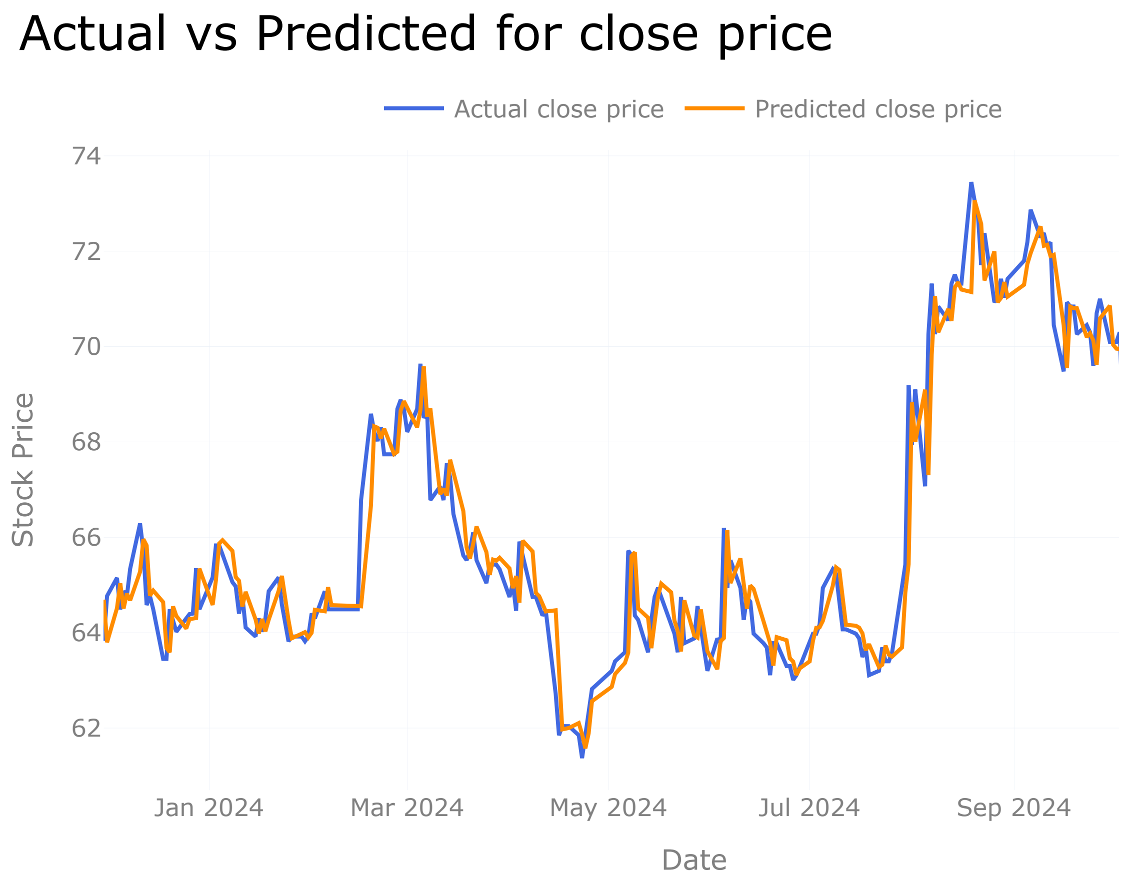
In [ ]:

```
# Plot Actual vs Predicted for open price
fig = go.Figure(data=go.Scatter(x= df_slice_data.index, y= df_slice_data[
fig.add_trace(go.Scatter(x=df_slice_data.index, y=df_slice_data['open_pre

fig.update_xaxes(title_text ="Date")
fig.update_yaxes(title_text ="Stock Price")
fig.update_layout(height=500, width=750, template="plotly_white", title =
        font_color="grey", font_size =12,
        title_font_color="black", title_font_size =24)
fig.show()
```



In [ ]:

```
# Plot Actual vs Predicted for close price
fig = go.Figure(data=go.Scatter(x= df_slice_data.index, y= df_slice_data[
fig.add_trace(go.Scatter(x=df_slice_data.index, y=df_slice_data['close_pr

fig.update_xaxes(title_text ="Date")
fig.update_yaxes(title_text ="Stock Price")
fig.update_layout(height=500, width=750, template="plotly_white", title =
        font_color="grey", font_size =12,
        title_font_color="black", title_font_size =24,
        legend_title=None, legend=dict(orientation='h', yanchor='bottom',
fig.show()
```



4. | Further Prediction (using Bidirectional LSTM)

In [ ]:
```
# Convert NumPy array to DataFrame
df_data = pd.DataFrame(df_data, columns=['open', 'close'])

# Keep rows from index 0 to 1472
df_data = df_data.iloc[:1473]

# Reset index
df_data.reset_index(drop=True, inplace=True)
# Create a date range for the next 180 business days
last_date = df['time'].iloc[-1]
business_days = pd.date_range(start=last_date + pd.Timedelta(days=1), per

predictions = []

# Loop through the business days
for next_date in business_days:
    # Get the last 50 rows of 'open' and 'close' prices for prediction
    last_50_data = df_data[-50:]  # Get the last 50 normalized rows

    # Reshape the data for LSTM input
    x_next = np.array([last_50_data])  # Shape will be (1, 50, 2)
    x_next = np.reshape(x_next, (x_next.shape[0], x_next.shape[1], 2))  #

    # Make predictions using the trained LSTM model
    y_next_predict = LSTM_model.predict(x_next)
```

```
    # Inverse transform the predicted values to get actual prices
    y_next_predict_inverse = scaler.inverse_transform(y_next_predict)

    # Prepare the DataFrame for the next day's prediction
    df_next = pd.DataFrame({
        'time': [next_date],
        'open_predicted': [y_next_predict_inverse[0][0]],
        'close_predicted': [y_next_predict_inverse[0][1]]
    })

    # Append the prediction to the predictions list
    predictions.append(df_next)

    # Normalize the predicted values before appending to df_data
    new_row_normalized = scaler.transform(np.array([[y_next_predict_inver

    # Append the new normalized predicted row to the normalized DataFrame
    new_row = pd.DataFrame(new_row_normalized, columns=['open', 'close'])
    df_data = pd.concat([df_data, new_row], ignore_index=True)  # Append

# Combine all predictions into a single DataFrame
df_predictions = pd.concat(predictions, ignore_index=True)

df_slice_col=df_slice_data.reset_index()
final_df = pd.concat([df_slice_col, df_predictions], ignore_index=True)
print(final_df)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 173ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 271ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 186ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 159ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 105ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 127ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 99ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 85ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 100ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 118ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 85ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 80ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 80ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 67ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step
           time   open  close  open_predicted  close_predicted
0    2023-11-30  64.77  63.83       64.873665        64.692589
1    2023-12-01  64.39  64.77       64.004227        63.799187
2    2023-12-04  64.87  65.15       64.671440        64.513763
3    2023-12-05  65.15  64.49       65.164474        65.034805
4    2023-12-06  64.49  64.87       64.672188        64.506218
..          ...    ...    ...             ...              ...
420  2025-07-28    NaN    NaN       66.621880        66.604301
421  2025-07-29    NaN    NaN       66.626602        66.609459
422  2025-07-30    NaN    NaN       66.631470        66.614761
423  2025-07-31    NaN    NaN       66.636467        66.620178
424  2025-08-01    NaN    NaN       66.641571        66.625702

[425 rows x 5 columns]
```

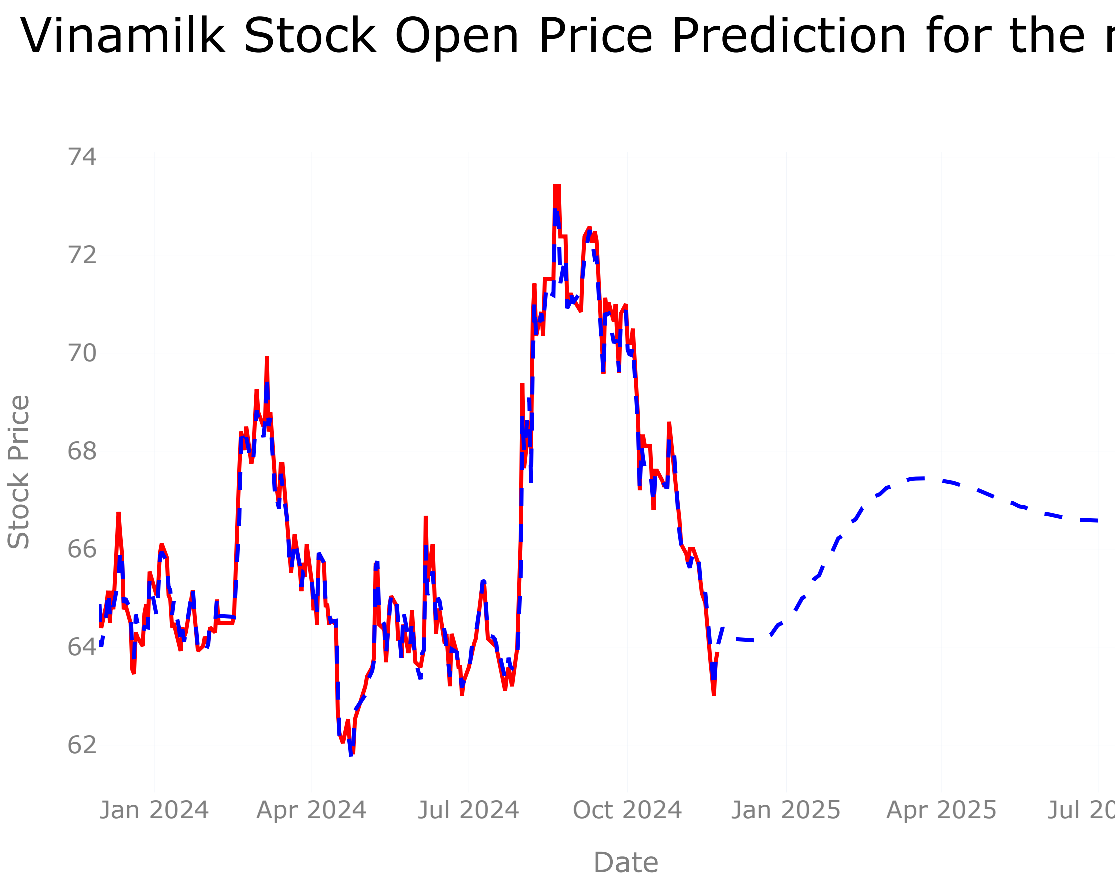
In [ ]:
```
file_name = 'VNM 6 months prediction.xlsx'

# saving the excel
final_df.to_excel(file_name)
```

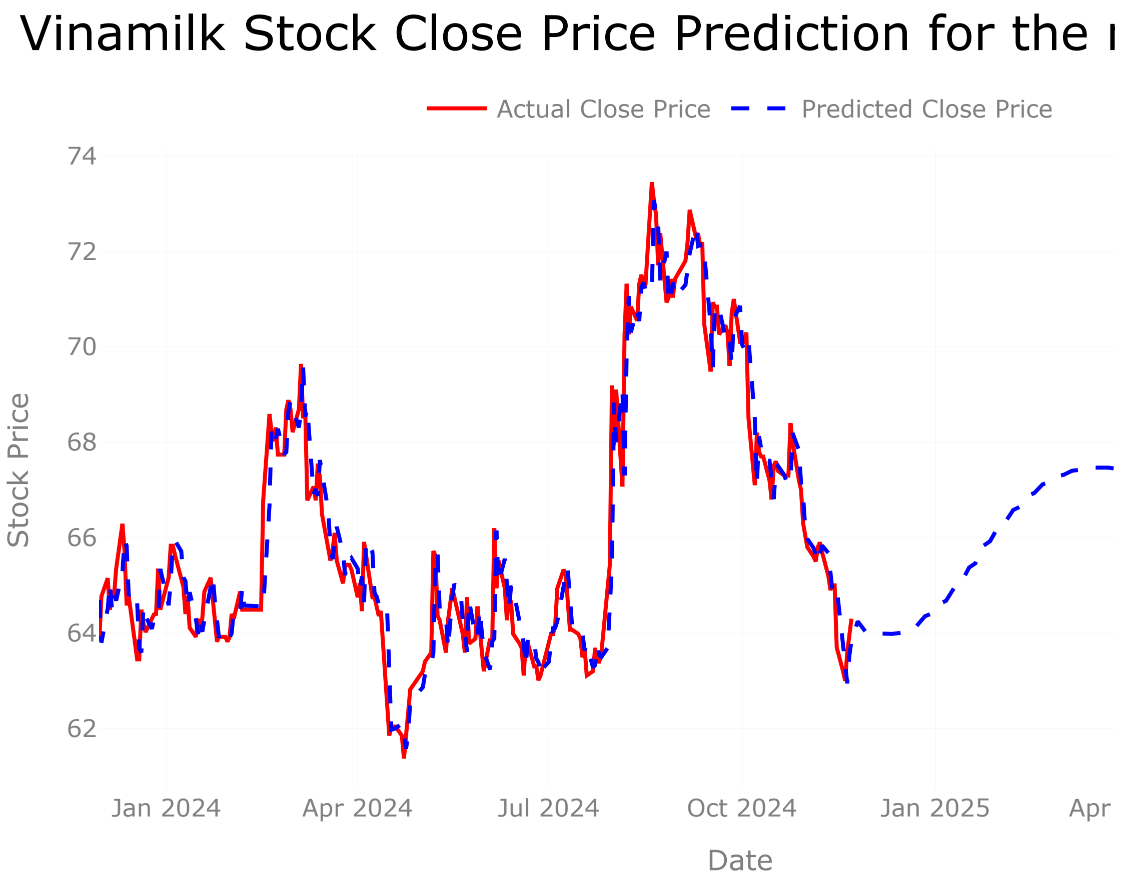
In [ ]:
```
print(df_predictions.head(15))
```

```
          time  open_predicted  close_predicted
0   2024-11-25       64.396118        64.245888
1   2024-11-26       64.347000        64.185394
2   2024-11-27       64.280724        64.114395
3   2024-11-28       64.221153        64.053589
4   2024-11-29       64.181702        64.013023
5   2024-12-02       64.163872        63.993649
6   2024-12-03       64.160667        63.988586
7   2024-12-04       64.162994        63.989716
8   2024-12-05       64.165321        63.992329
9   2024-12-06       64.162216        63.992214
10  2024-12-09       64.153114        63.988899
11  2024-12-10       64.140511        63.984440
12  2024-12-11       64.126968        63.980957
13  2024-12-12       64.116875        63.981350
14  2024-12-13       64.113014        63.985970
```

In [ ]:
```
# Plotting the predicted open data until 7/2025
fig1 = go.Figure(data=go.Scatter(x= final_df['time'], y= final_df['open']
fig1.add_trace(go.Scatter(x=final_df['time'], y=final_df['open_predicted'
                         marker_color ="blue", name='Predicted Open Price

fig1.update_xaxes(title_text ="Date")
fig1.update_yaxes(title_text ="Stock Price")
fig1.update_layout(height=500,width=800, template="plotly_white", title =
        font_color="grey", font_size =12,
        title_font_color="black", title_font_size =24)
fig1.show()

# Plotting the predicted close data until 7/2025
fig2 = go.Figure(data=go.Scatter(x= final_df['time'], y= final_df['close'
fig2.add_trace(go.Scatter(x=final_df['time'], y=final_df['close_predicted
                         marker_color ="blue", name='Predicted Close Pric

fig2.update_xaxes(title_text ="Date")
fig2.update_yaxes(title_text ="Stock Price")
fig2.update_layout(height=500, width=800, template="plotly_white", title
```

```
        font_color="grey", font_size =12,
        title_font_color="black", title_font_size =24,
        legend_title=None, legend=dict(orientation='h', yanchor='bottom',
fig2.show()
```




Vinamilk Stock Close Price Prediction for the ı
Actual Close Price
Predicted Close Price
Stock Price
74
72
70
68
66
64
62
Jan 2024
Apr 2024
Jul 2024
Oct 2024
Jan 2025
Apr
Date