```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Read data from CSV file into DataFrame
df = pd.read_csv("/content/TCS__EQ__NSE__NSE__MINUTE.csv")


# Display basic information about the dataset
print("Basic Information:")
df.info()
```

```
Basic Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 370546 entries, 0 to 370545
Data columns (total 6 columns):
 #   Column     Non-Null Count   Dtype
---  ------     --------------   -----
 0   timestamp  370546 non-null  object
 1   open       370405 non-null  float64
 2   high       370405 non-null  float64
 3   low        370405 non-null  float64
 4   close      370405 non-null  float64
 5   volume     370405 non-null  float64
dtypes: float64(5), object(1)
memory usage: 17.0+ MB
```

```
import pandas as pd

# Read the CSV file
df = pd.read_csv('/content/TCS__EQ__NSE__NSE__MINUTE.csv')

# Convert the 'timestamp' column to datetime
df['timestamp'] = pd.to_datetime(df['timestamp'])

# Save the changes back to the same CSV file (optional)
df.to_csv('/content/TCS__EQ__NSE__NSE__MINUTE.csv', index=False)

print(df['timestamp'].head())  # Just to confirm the format
```

```
0   2017-01-02 09:15:00+05:30
1   2017-01-02 09:16:00+05:30
2   2017-01-02 09:17:00+05:30
3   2017-01-02 09:18:00+05:30
4   2017-01-02 09:19:00+05:30
Name: timestamp, dtype: datetime64[ns, UTC+05:30]
```

```
# Display basic information about the dataset
print("Basic Information:")
df.info()
```

```
Basic Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 370546 entries, 0 to 370545
```

```
Data columns (total 6 columns):
 #   Column     Non-Null Count   Dtype
---  ------     --------------   -----
 0   timestamp  370546 non-null  datetime64[ns, UTC+05:30]
 1   open       370405 non-null  float64
 2   high       370405 non-null  float64
 3   low        370405 non-null  float64
 4   close      370405 non-null  float64
 5   volume     370405 non-null  float64
dtypes: datetime64[ns, UTC+05:30](1), float64(5)
memory usage: 17.0 MB
```

```
# Check for missing values
print("\nMissing Values:")
print(df.isnull().sum())
```

```
Missing Values:
timestamp      0
open         141
high         141
low          141
close        141
volume       141
dtype: int64
```

```
# Handling missing values
df.dropna(inplace=True)

# Drop duplicate rows if any
df.drop_duplicates(inplace=True)


print("\nMissing Values:")
print(df.isnull().sum())
```

```
Missing Values:
timestamp    0
open         0
high         0
low          0
close        0
volume       0
dtype: int64
```

eda

```
# Summary statistics for numerical columns
print("\nSummary Statistics:")
print(df.describe())
```

```
Summary Statistics:
```

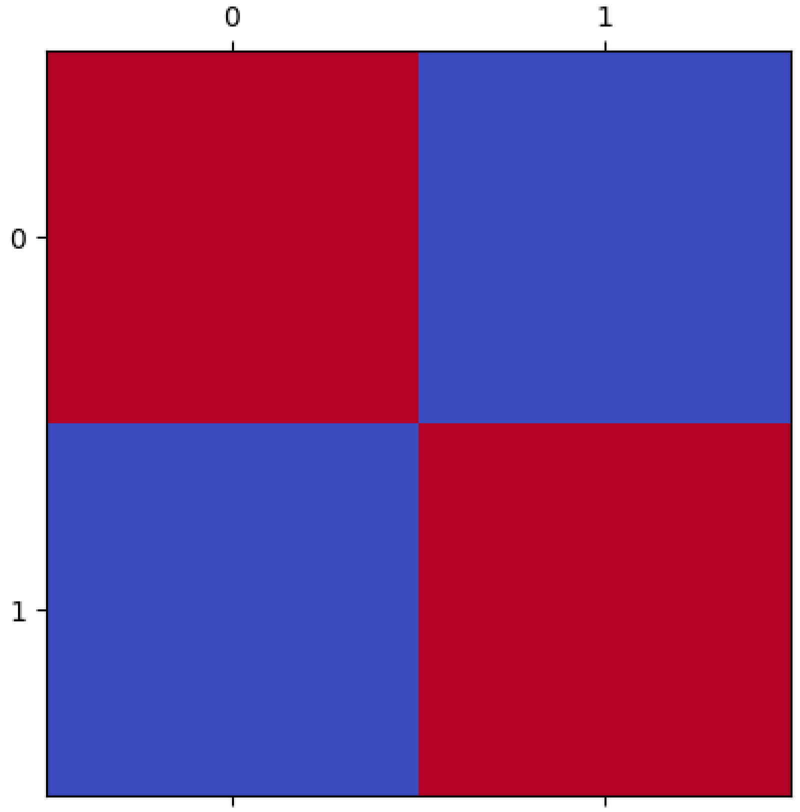
```
               open           high            low          close  \
count  370405.000000  370405.000000  370405.000000  370405.000000
mean     1848.705882    1849.528074    1847.836017    1848.698045
std       445.711672     445.956437     445.450803     445.709579
min      1076.500000    1078.030000    1076.500000    1076.750000
25%      1357.730000    1358.230000    1357.500000    1357.680000
50%      1967.250000    1968.000000    1966.350000    1967.250000
75%      2155.850000    2156.650000    2155.000000    2155.850000
max      2950.050000    2952.000000    2949.500000    2950.050000

             volume
count  3.704050e+05
mean   8.461558e+03
std    1.003912e+05
min    0.000000e+00
25%    2.300000e+03
50%    4.394000e+03
75%    8.858000e+03
max    5.957057e+07
```

```
# Check the correlation between numerical features
print("\nCorrelation Matrix:")
print(df.corr())
```

```
Correlation Matrix:
           timestamp      open      high       low     close    volume
timestamp   1.000000  0.905541  0.905710  0.905368  0.905542  0.013106
open        0.905541  1.000000  0.999996  0.999996  0.999994  0.010235
high        0.905710  0.999996  1.000000  0.999993  0.999996  0.010381
low         0.905368  0.999996  0.999993  1.000000  0.999997  0.010002
close       0.905542  0.999994  0.999996  0.999997  1.000000  0.010170
volume      0.013106  0.010235  0.010381  0.010002  0.010170  1.000000
```

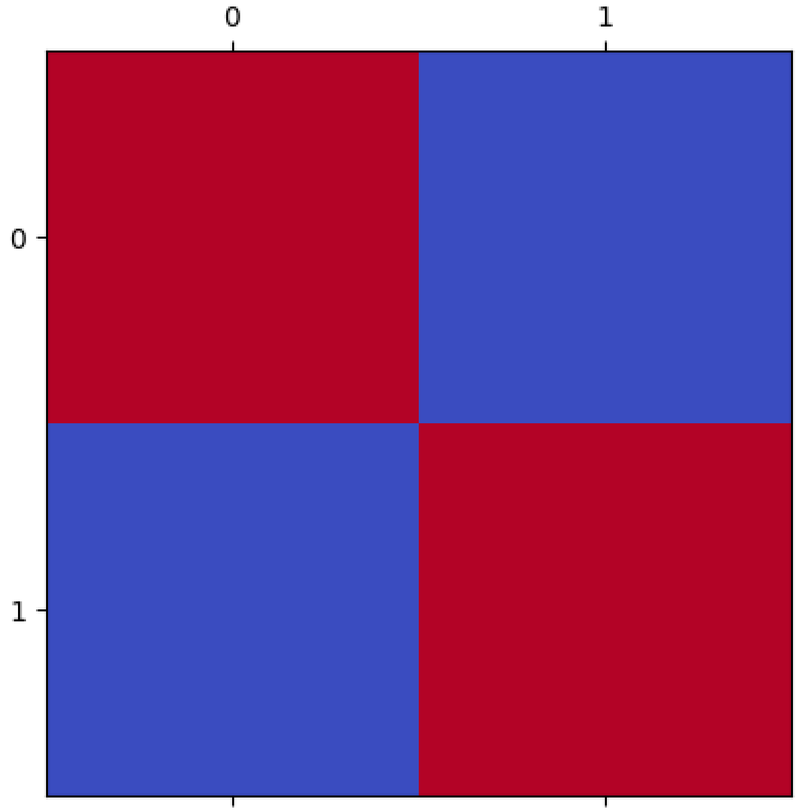
```
import pandas as pd
import matplotlib.pyplot as plt
# Heatmap for Open and Close
print("Heatmap - Open vs Close Correlation")
plt.matshow(df[["open", "close"]].corr(), cmap="coolwarm")
```

```
Heatmap - Open vs Close Correlation
<matplotlib.image.AxesImage at 0x7803b3500ca0>
```

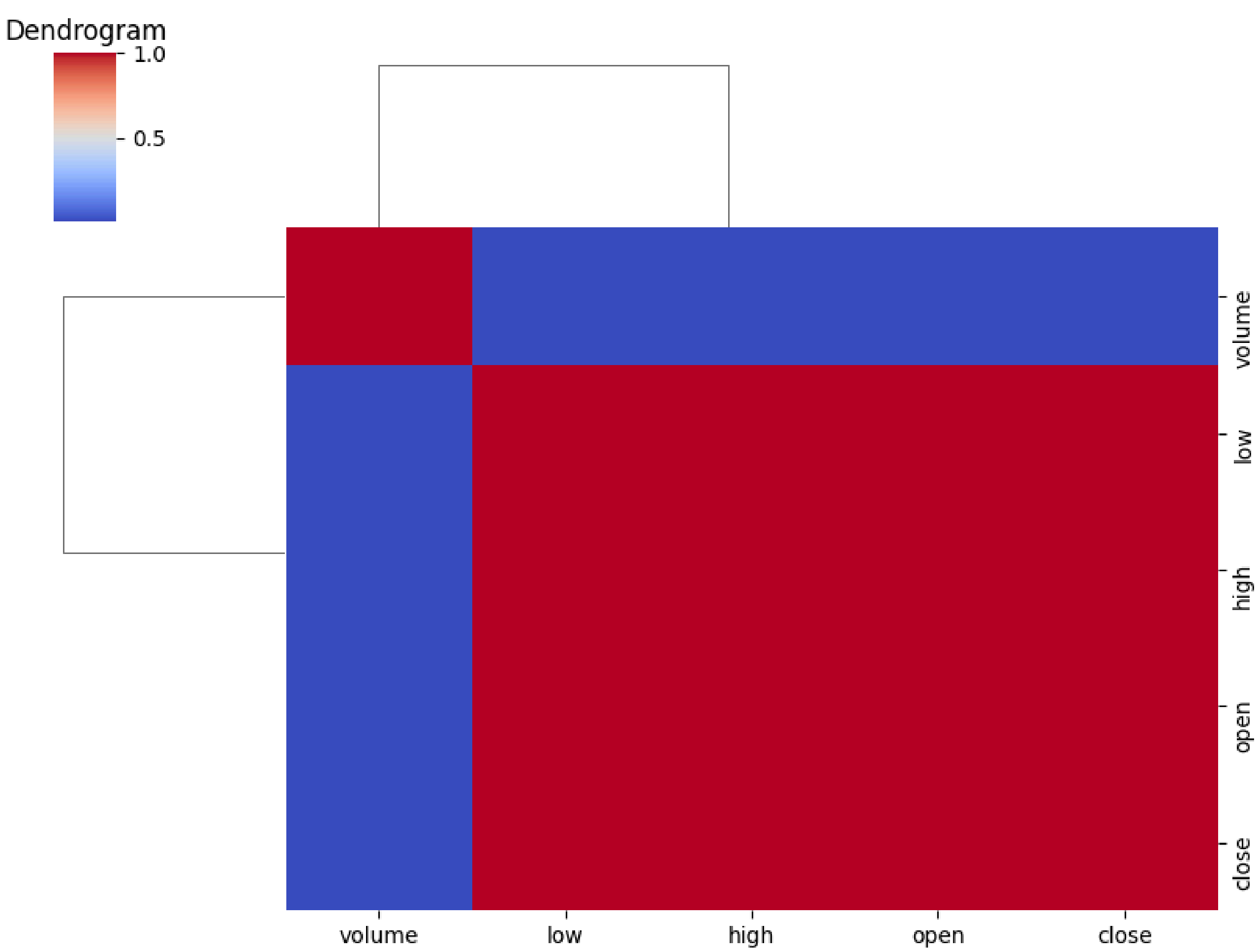



```
# Heatmap for High and Low
print("Heatmap - high vs low Correlation")
plt.matshow(df[["high", "low"]].corr(), cmap="coolwarm")
```

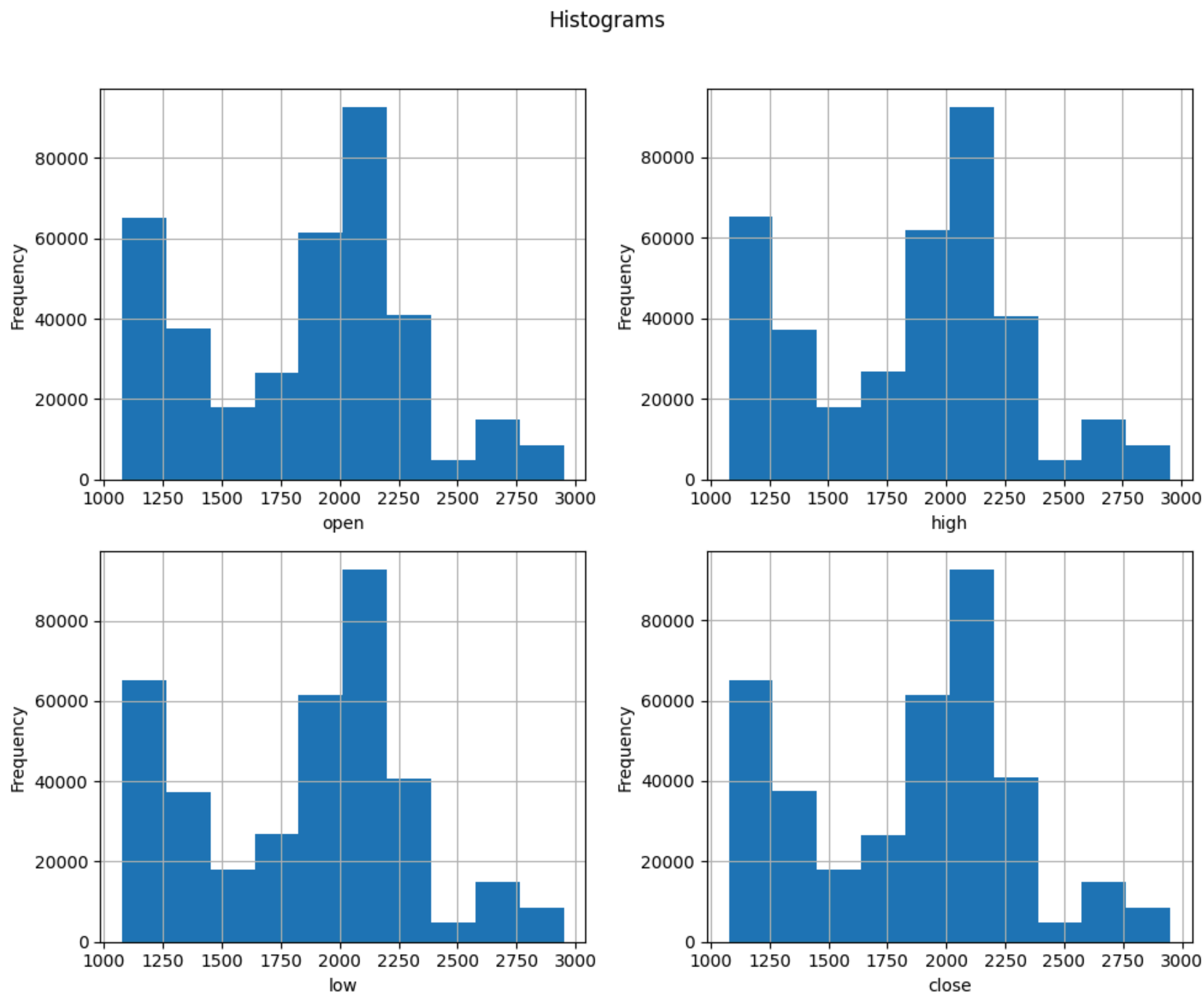
```
Heatmap - high vs low Correlation
<matplotlib.image.AxesImage at 0x7803b104fdf0>
```



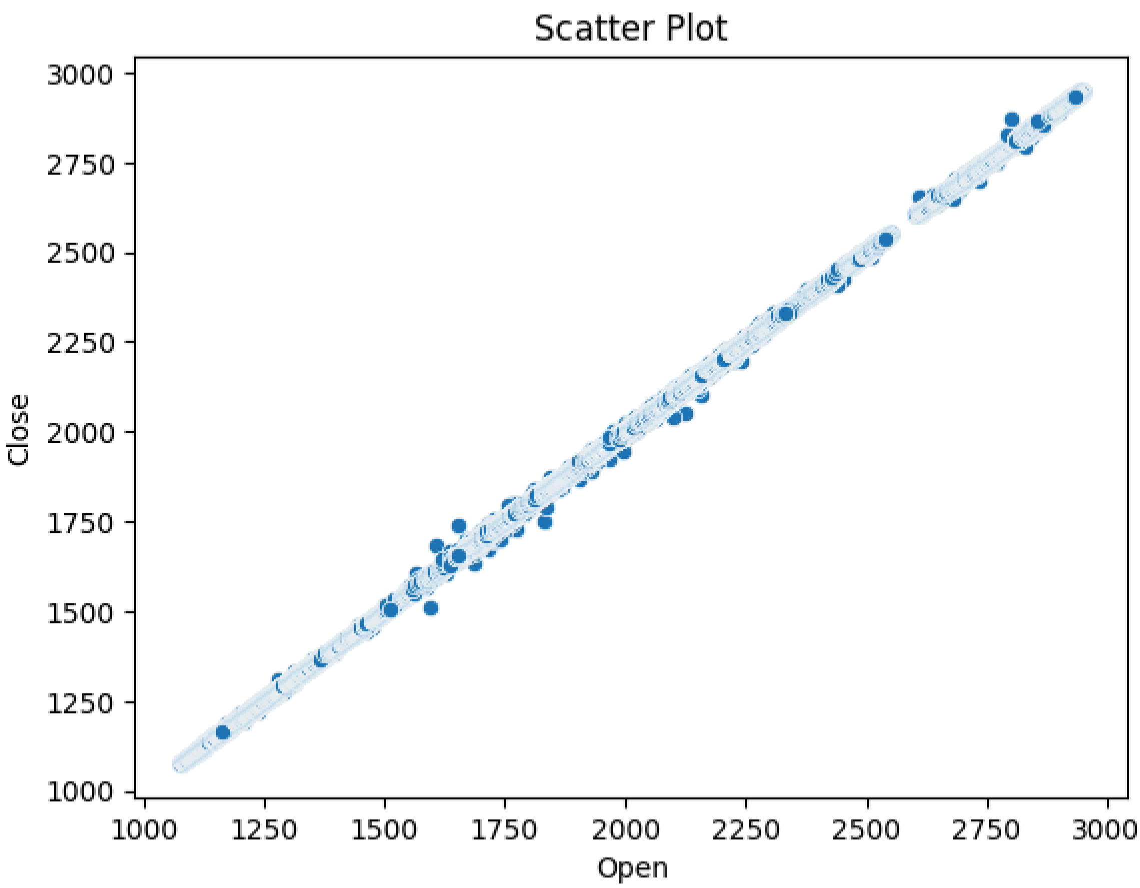

```
# Dendrogram (for hierarchical clustering)
correlation_matrix = df[['open', 'high', 'low', 'close', 'volume']].corr()
sns.clustermap(correlation_matrix, cmap='coolwarm', figsize=(8, 6))
plt.title('Dendrogram')
plt.show()
```



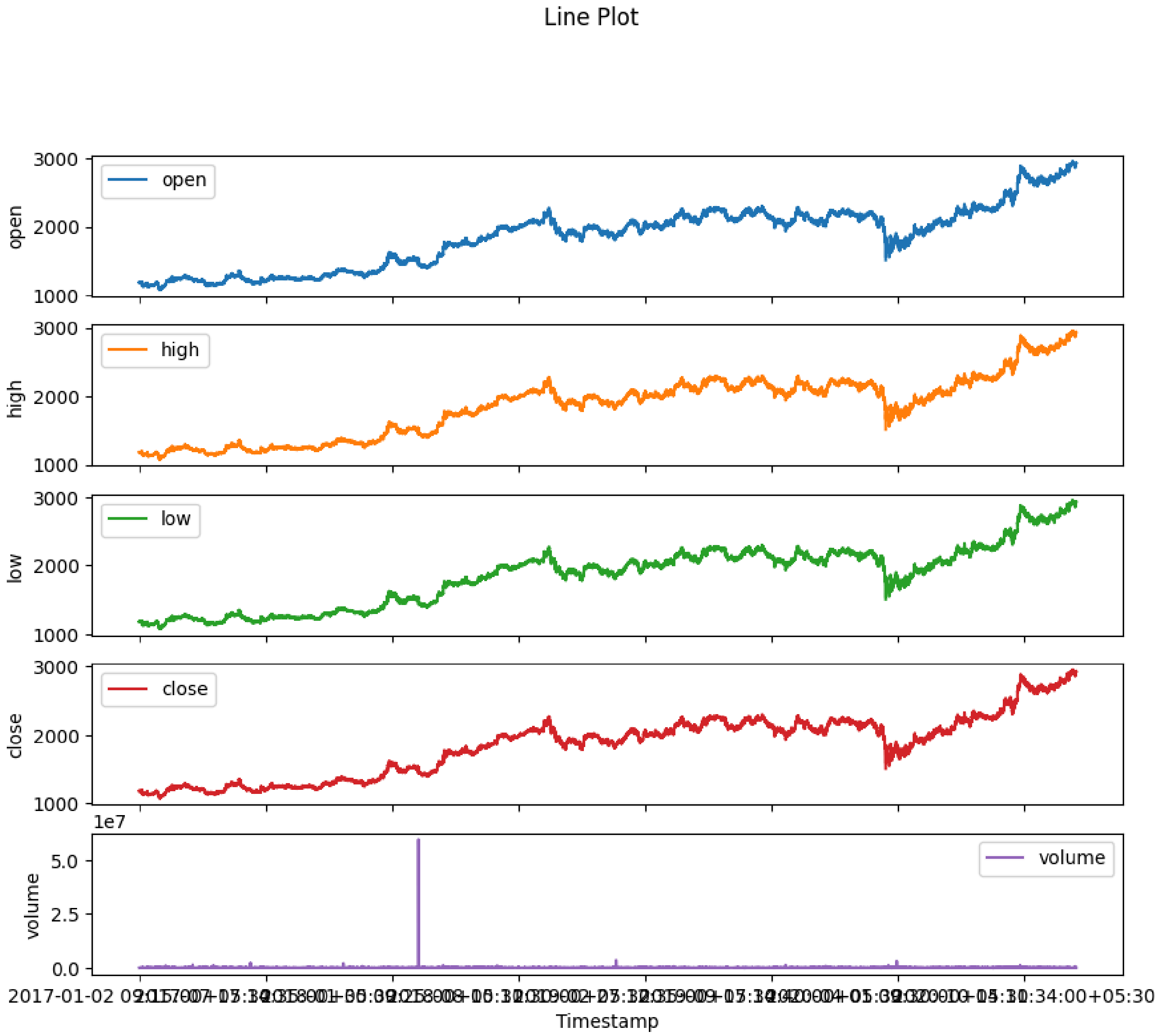

```
import pandas as pd
import matplotlib.pyplot as plt

# Read data from CSV file into DataFrame
df = pd.read_csv("/content/TCS__EQ__NSE__NSE__MINUTE.csv")

# Plot histograms for each numerical column
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10, 8))

for ax, column in zip(axes.flatten(), df.select_dtypes(include='number').columns):
    df[column].hist(ax=ax)
    ax.set_xlabel(column)  # Set x-axis label
    ax.set_ylabel('Frequency')  # Set y-axis label

plt.suptitle('Histograms', x=0.5, y=1.02)
plt.tight_layout()
plt.show()
```




```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Read data from CSV file into DataFrame
df = pd.read_csv("/content/TCS__EQ__NSE__NSE__MINUTE.csv")

# Scatter Plot
sns.scatterplot(data=df, x='open', y='close')
plt.title('Scatter Plot')
plt.xlabel('Open')  # Set x-axis label
plt.ylabel('Close')  # Set y-axis label
plt.show()
```

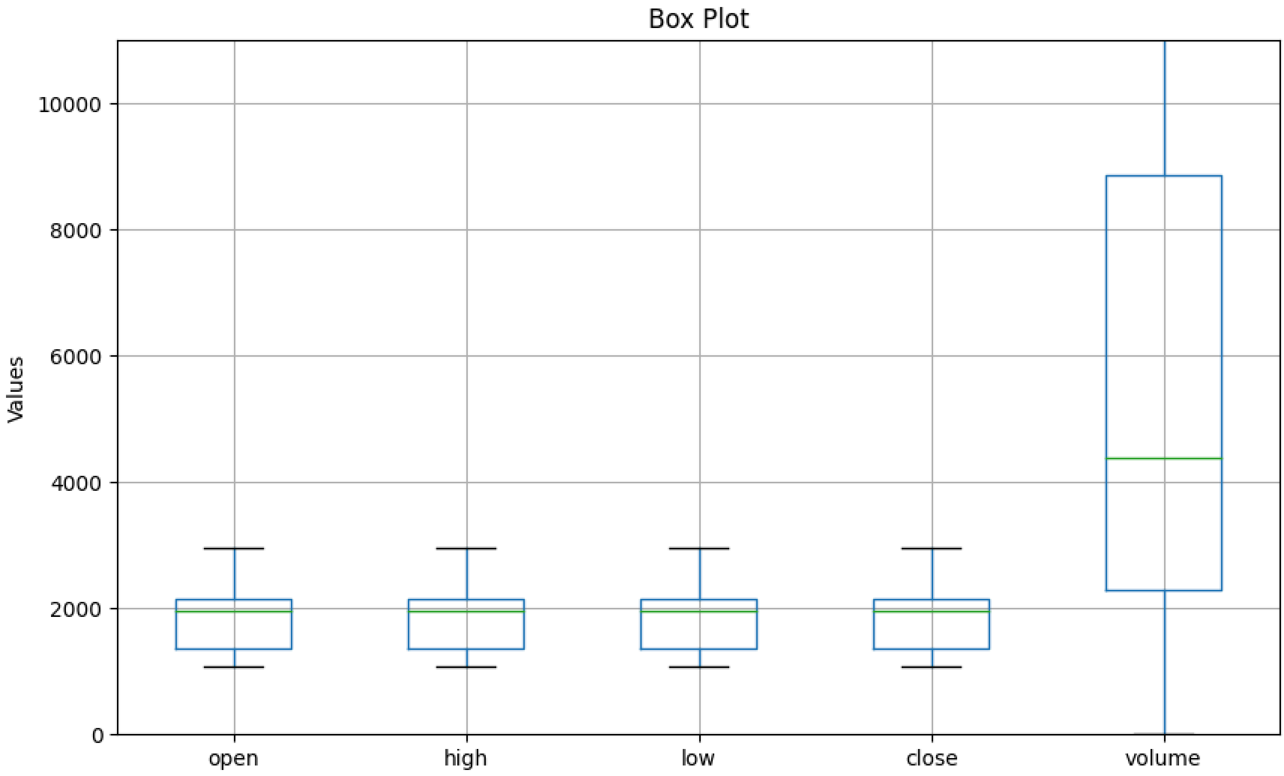



```
import pandas as pd
import matplotlib.pyplot as plt

# Read data from CSV file into DataFrame
df = pd.read_csv("/content/TCS__EQ__NSE__NSE__MINUTE.csv")

# Line Plot
axes = df.plot(x='timestamp', y=['open', 'high', 'low', 'close', 'volume'], subplots=True

# Set axis labels for each subplot
for ax, column in zip(axes, df[['open', 'high', 'low', 'close', 'volume']]):
    ax.set_xlabel('Timestamp')  # Set x-axis label
    ax.set_ylabel(column)  # Set y-axis label

plt.suptitle('Line Plot', x=0.5, y=1.02)
plt.show()
```



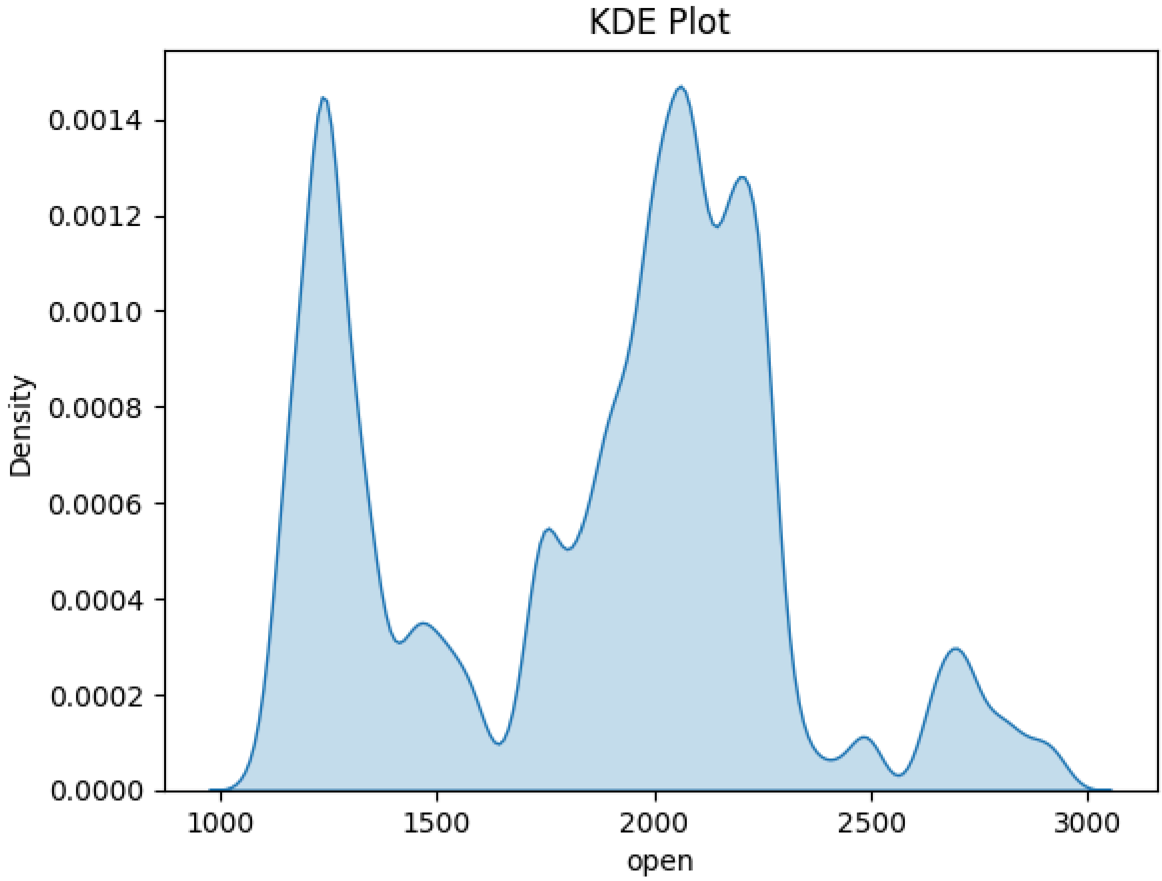

```
import pandas as pd
import matplotlib.pyplot as plt

# Read data from CSV file into DataFrame
df = pd.read_csv("/content/TCS__EQ__NSE__NSE__MINUTE.csv")

# Create a smaller figure
fig, ax = plt.subplots(figsize=(10, 6))

# Box Plot
df.boxplot(ax=ax)
ax.set_ylim(0, 11000) # Specify the axis to plot on
```

```
plt.title('Box Plot')

# Specify y-axis label
plt.ylabel('Values')  # Add y-axis label

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

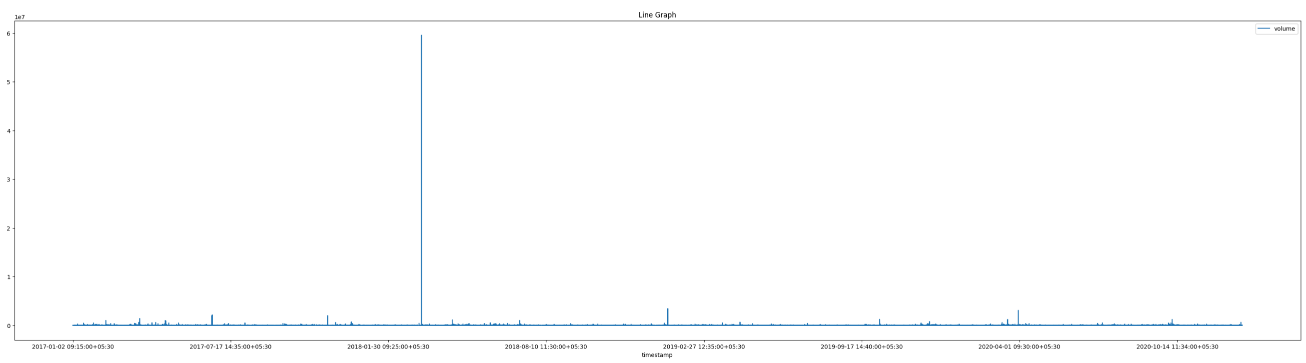



```
# KDE Plot
sns.kdeplot(df['open'], shade=True)
plt.title('KDE Plot')
plt.show()
```

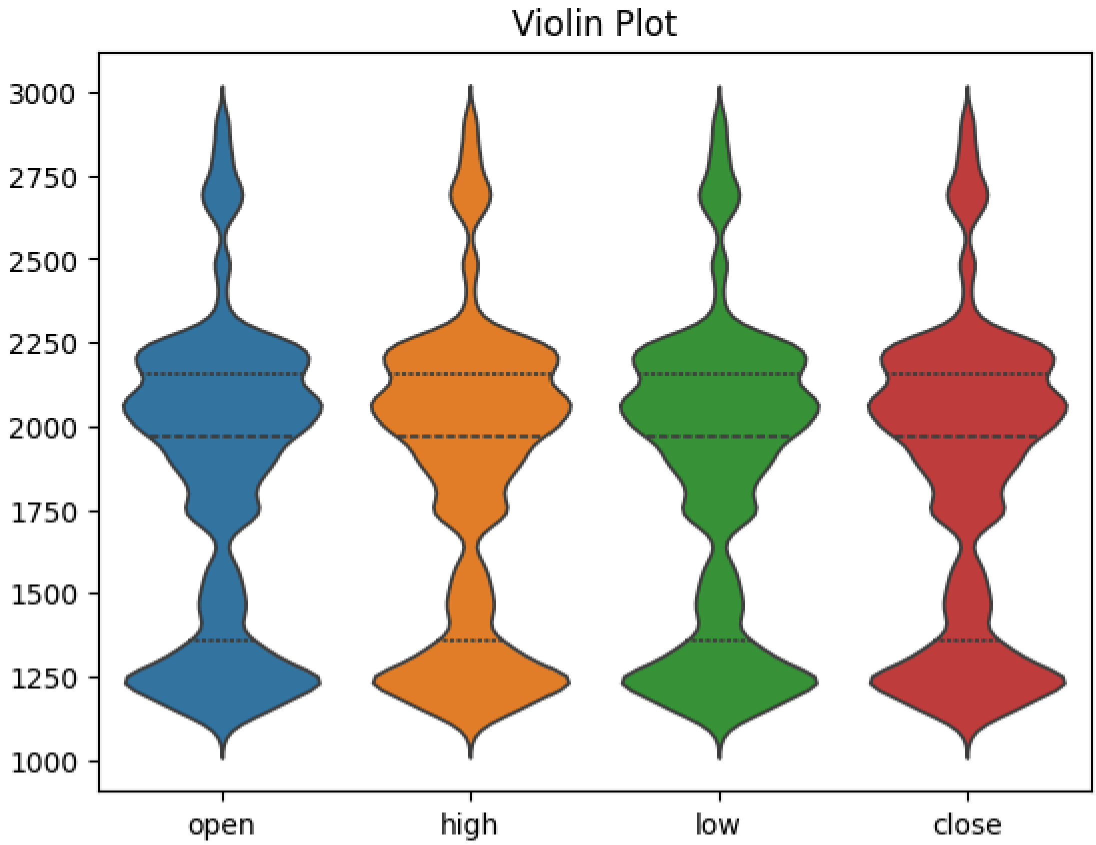
```
<ipython-input-17-cd92547e0e66>:2: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(df['open'], shade=True)
```



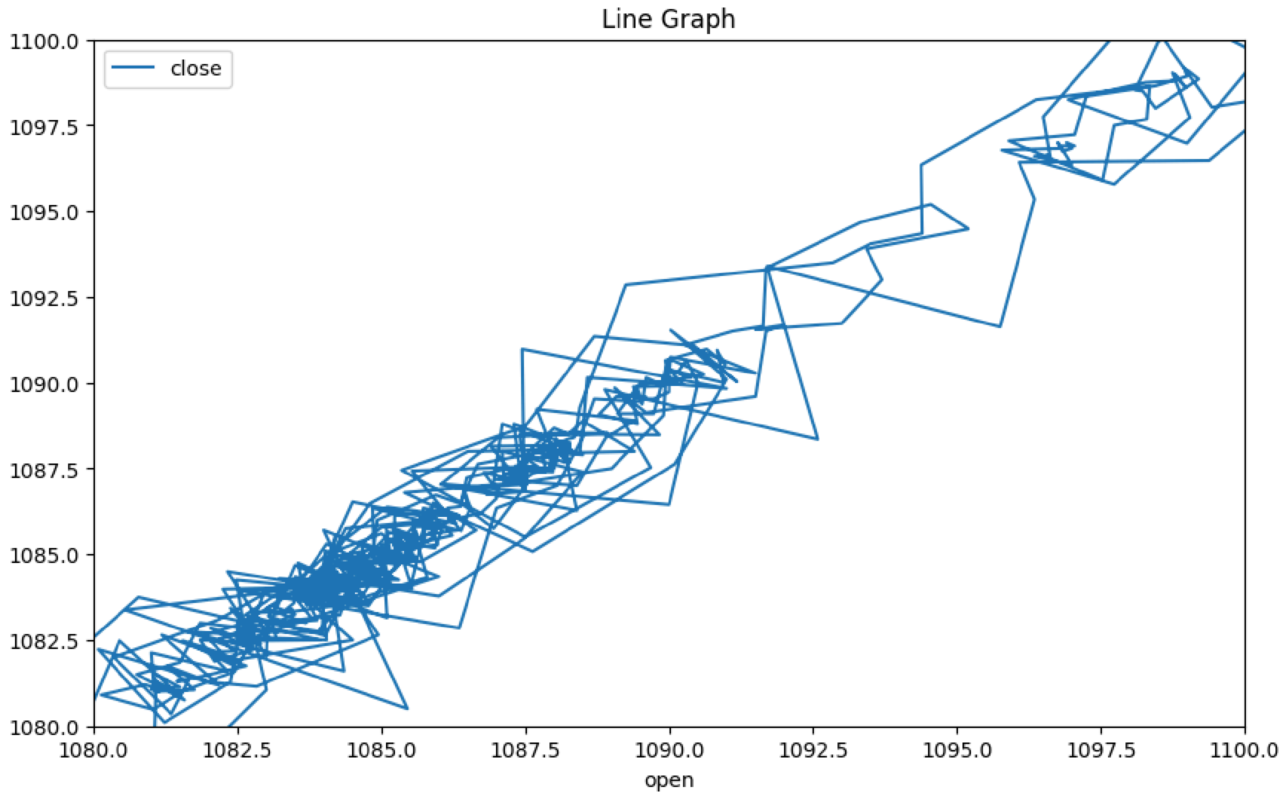

```
# Line Graph
df.plot(x='timestamp', y='volume', figsize=(40, 10))
plt.title('Line Graph')
plt.show()
```




```
# Violin Plot
sns.violinplot(data=df[['open', 'high', 'low', 'close']], inner='quartile')
plt.title('Violin Plot')
plt.show()
```

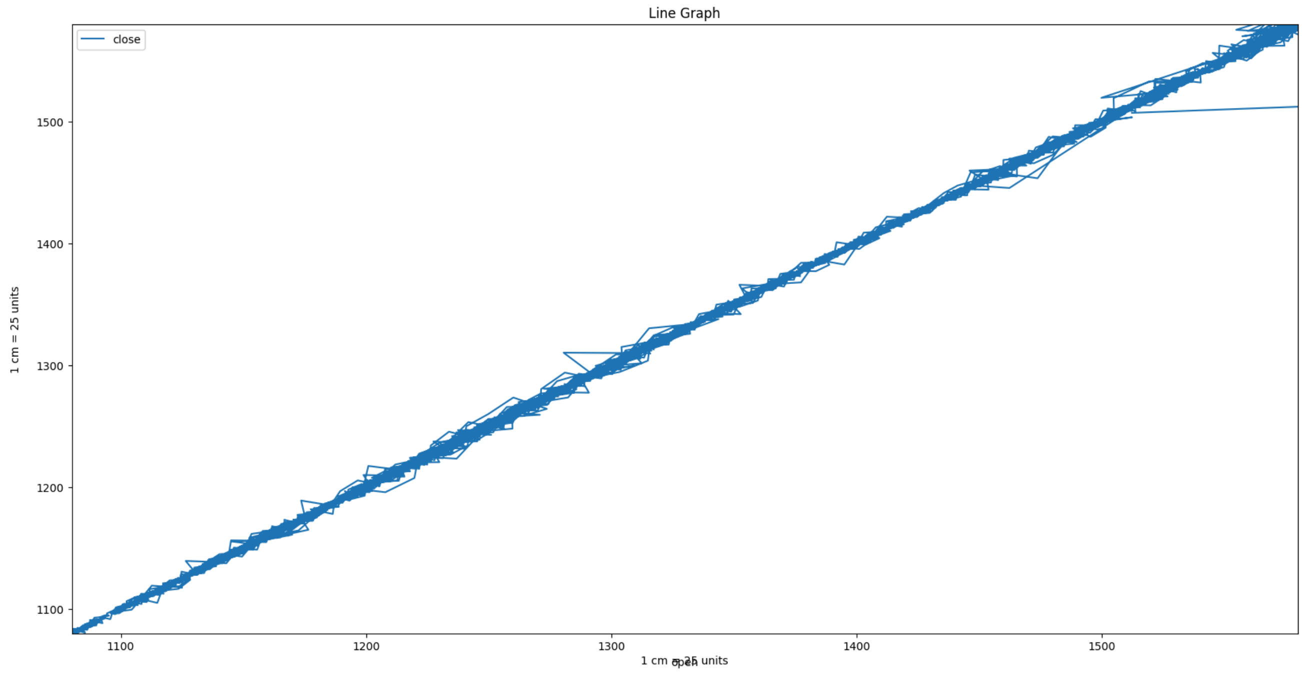



```
import matplotlib.pyplot as plt

# Assuming df is your DataFrame

# Plotting the data
ax = df.plot(x='open', y='close', figsize=(10, 6))

# Setting the x and y limits
ax.set_xlim(1080, 1100)
ax.set_ylim(1080, 1100)

# Adding title
plt.title('Line Graph')

# Display the plot
plt.show()
```

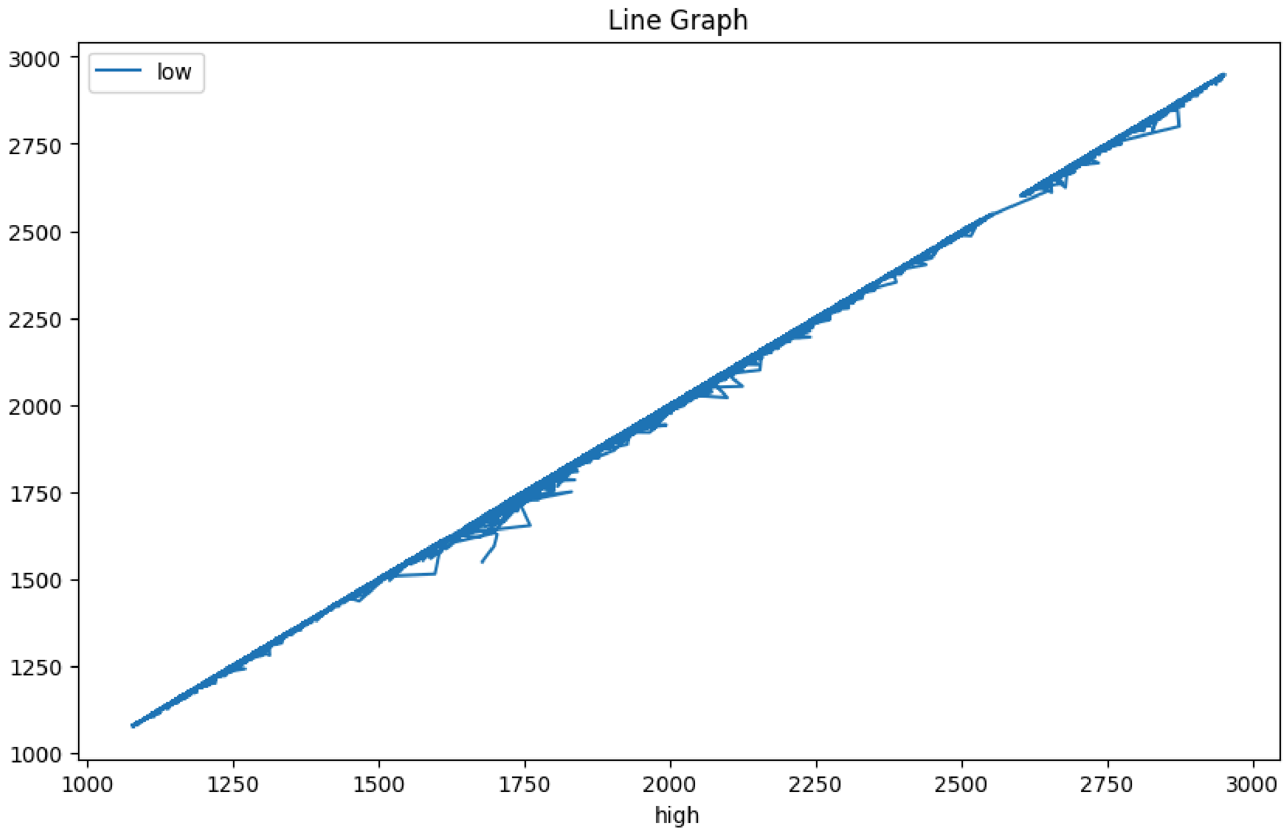



```
import matplotlib.pyplot as plt

# Assuming df is your DataFrame

# Plotting the data
ax = df.plot(x='open', y='close', figsize=(20, 10))

# Setting the x and y limits
ax.set_xlim(1080, 1100)
ax.set_ylim(1080, 1100)
# For example, if you want 1 cm to represent 10 units on both axes
# Calculate the range of units covered by 1 cm
units_per_cm = 25

# Calculate the range in data coordinates (assuming 1 cm = 10 units)
x_range = units_per_cm * (ax.get_xlim()[1] - ax.get_xlim()[0])
y_range = units_per_cm * (ax.get_ylim()[1] - ax.get_ylim()[0])

# Set the limits accordingly
ax.set_xlim(ax.get_xlim()[0], ax.get_xlim()[0] + x_range)
ax.set_ylim(ax.get_ylim()[0], ax.get_ylim()[0] + y_range)

# Adding title
plt.title('Line Graph')
```

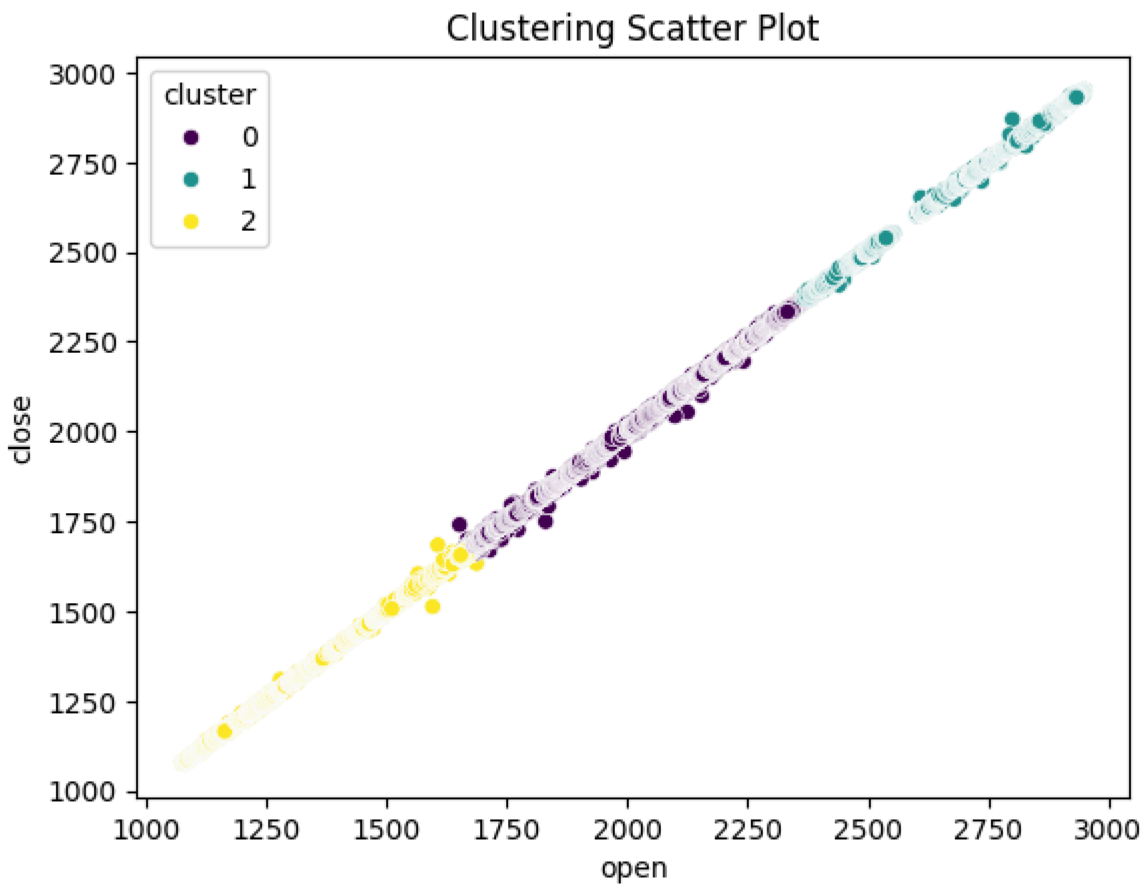
```
# Annotate x-axis scale
x_scale_annotation = f'1 cm = {units_per_cm} units'
ax.annotate(x_scale_annotation, xy=(0.5, -0.05), xycoords='axes fraction', ha='center')

# Annotate y-axis scale
y_scale_annotation = f'1 cm = {units_per_cm} units'
ax.annotate(y_scale_annotation, xy=(-0.05, 0.5), xycoords='axes fraction', va='center', r

# Display the plot
plt.show()
```



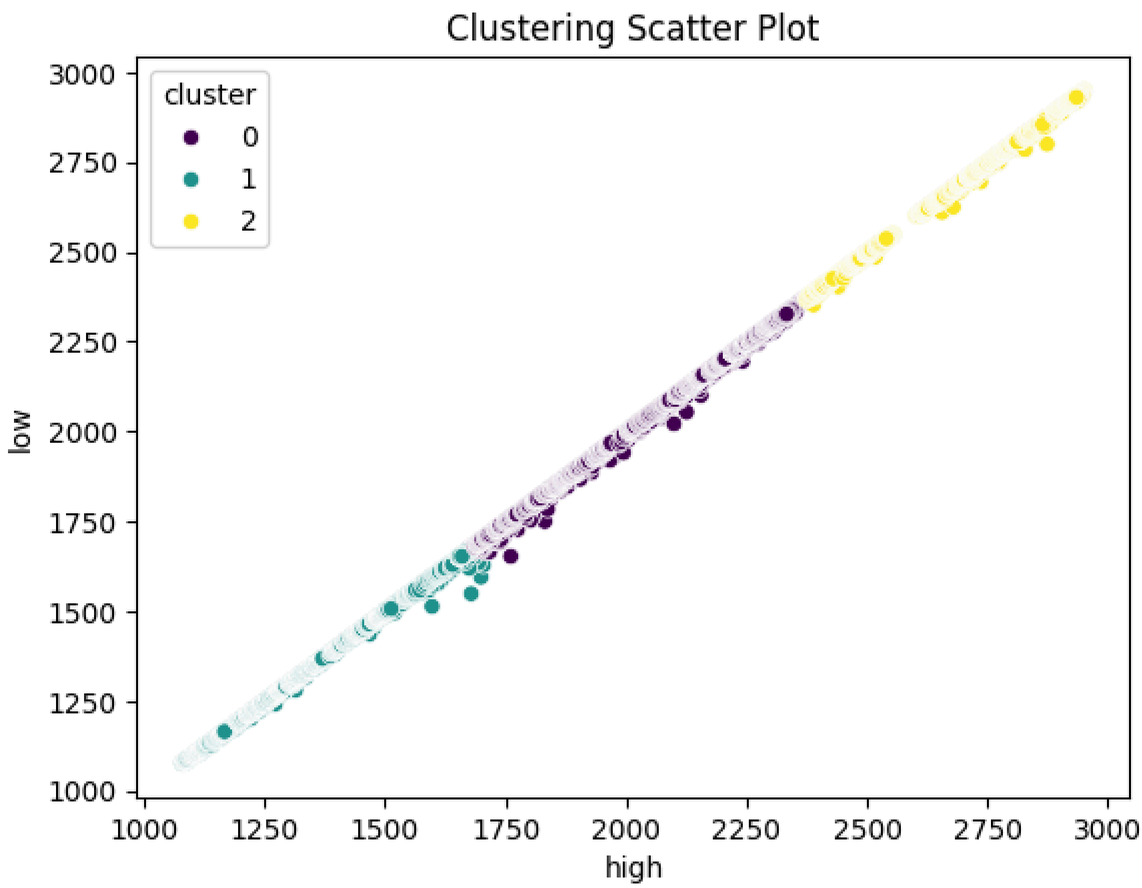

```
df.plot(x='high', y='low', figsize=(10, 6))
plt.title('Line Graph')
plt.show()
```




```
from sklearn.cluster import KMeans
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming df is your DataFrame with missing values
# Let's fill missing values with 0
df_filled = df.fillna(0)

# Fit KMeans clustering
kmeans = KMeans(n_clusters=3)
df['cluster'] = kmeans.fit_predict(df_filled[['open', 'close']])

# Plot clustering scatter plot
sns.scatterplot(data=df, x='open', y='close', hue='cluster', palette='viridis')
plt.title('Clustering Scatter Plot')
plt.show()
```




```
from sklearn.cluster import KMeans
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming df is your DataFrame with missing values
# Let's fill missing values with 0
df_filled = df.fillna(0)

# Fit KMeans clustering
kmeans = KMeans(n_clusters=3)
df['cluster'] = kmeans.fit_predict(df_filled[['high', 'low']])

# Plot clustering scatter plot
sns.scatterplot(data=df, x='high', y='low', hue='cluster', palette='viridis')
plt.title('Clustering Scatter Plot')
plt.show()
```




```
!pip install mplfinance
```

```
Collecting mplfinance
  Downloading mplfinance-0.12.10b0-py3-none-any.whl.metadata (19 kB)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (frc
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (f
Downloading mplfinance-0.12.10b0-py3-none-any.whl (75 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 75.0/75.0 kB 3.3 MB/s eta 0:00:00
Installing collected packages: mplfinance
Successfully installed mplfinance-0.12.10b0
```

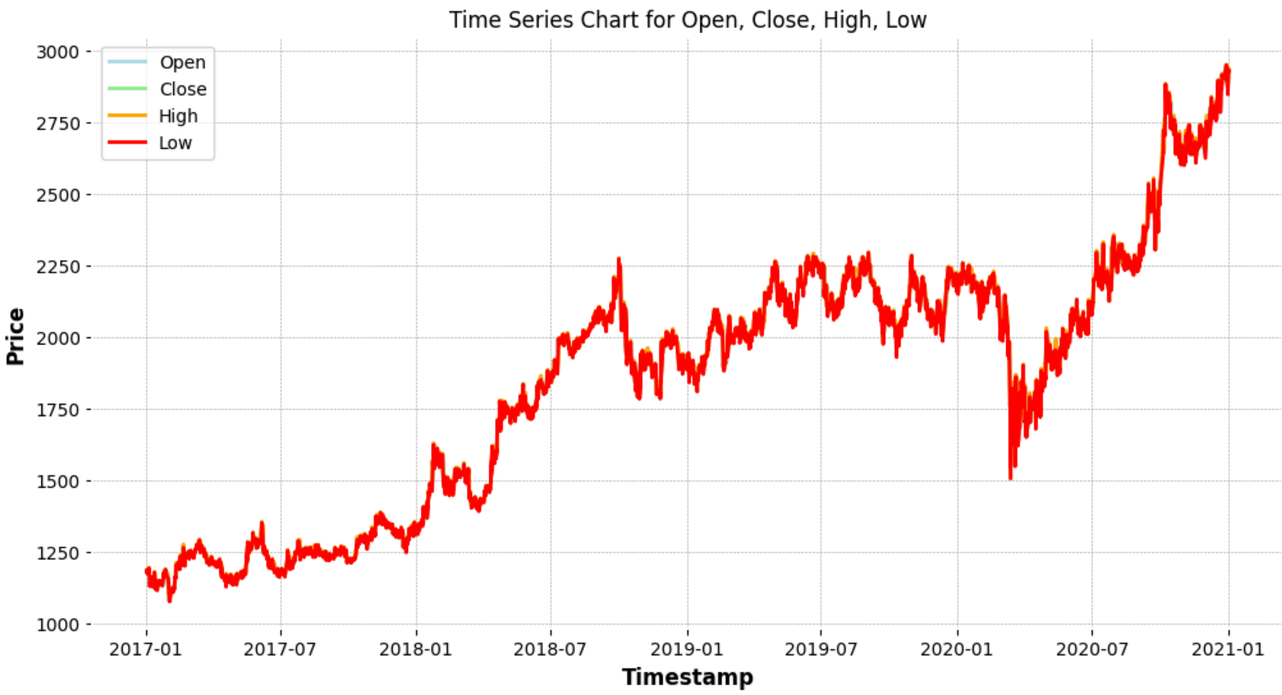
```
import pandas as pd
import mplfinance as mpf

# Assuming df is your DataFrame with the 'timestamp' column
# Convert 'timestamp' column to datetime format
```

```
df['timestamp'] = pd.to_datetime(df['timestamp'])

# Set 'timestamp' column as the index of the DataFrame
df.set_index('timestamp', inplace=True)

# Plot the candlestick chart
mpf.plot(df, type='candle', style='charles')
```

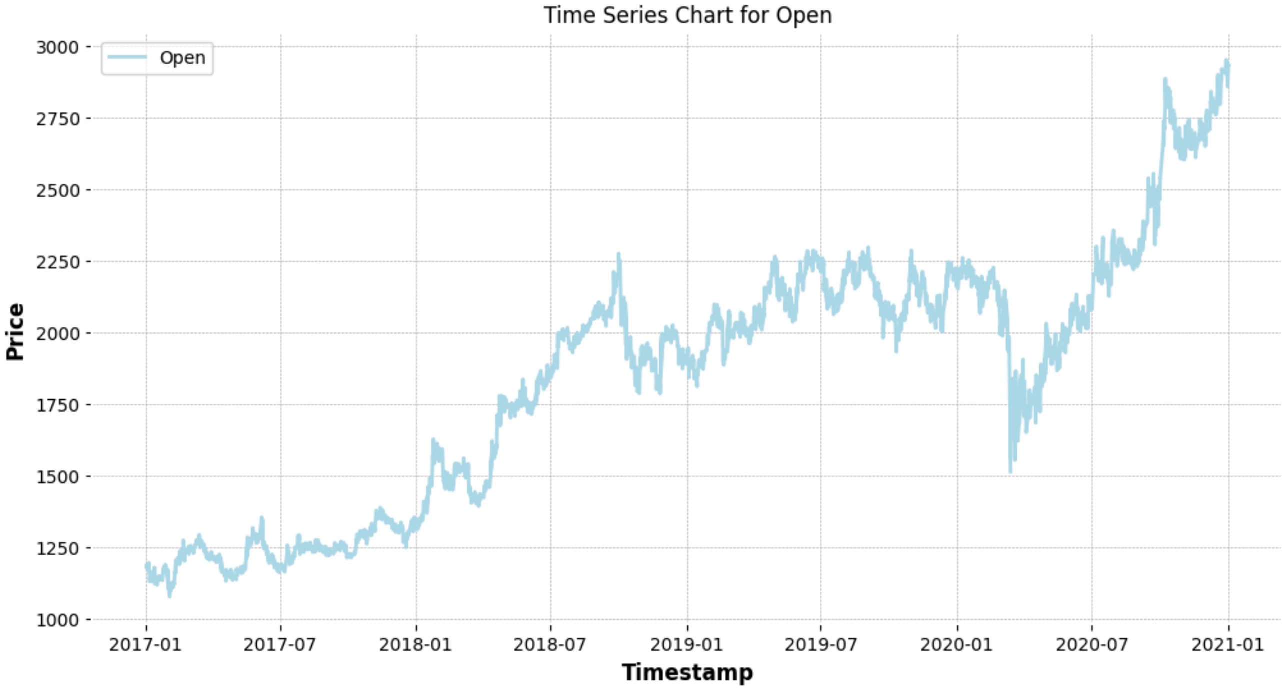
```
/usr/local/lib/python3.10/dist-packages/mplfinance/_arg_validators.py:84: UserWarning

 ==================================================================

  WARNING: YOU ARE PLOTTING SO MUCH DATA THAT IT MAY NOT BE
           POSSIBLE TO SEE DETAILS (Candles, Ohlc-Bars, Etc.)
  For more information see:
  - https://github.com/matplotlib/mplfinance/wiki/Plotting-Too-Much-Data

  TO SILENCE THIS WARNING, set `type='line'` in `mpf.plot()`
  OR set kwarg `warn_too_much_data=N` where N is an integer
  LARGER than the number of data points you want to plot.

 ================================================================
  warnings.warn('\n\n ==================================================================
```


```
import matplotlib.pyplot as plt

# Plotting Time Series Chart for 'open', 'close', 'high', 'low' columns
```

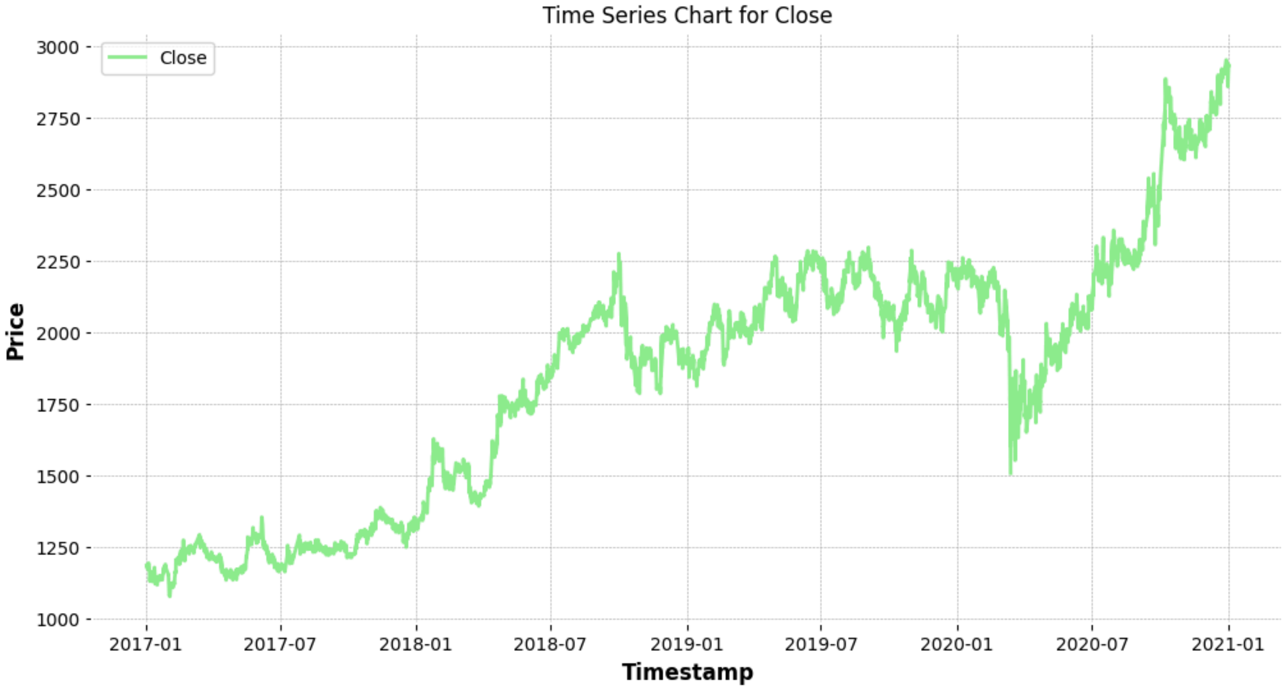
```
plt.figure(figsize=(12, 6))

plt.plot(df.index, df['open'], label='Open', color='lightblue')
plt.plot(df.index, df['close'], label='Close', color='lightgreen')
plt.plot(df.index, df['high'], label='High', color='orange')
plt.plot(df.index, df['low'], label='Low', color='red')

plt.xlabel('Timestamp')
plt.ylabel('Price')
plt.title('Time Series Chart for Open, Close, High, Low')
plt.legend()
plt.show()
```



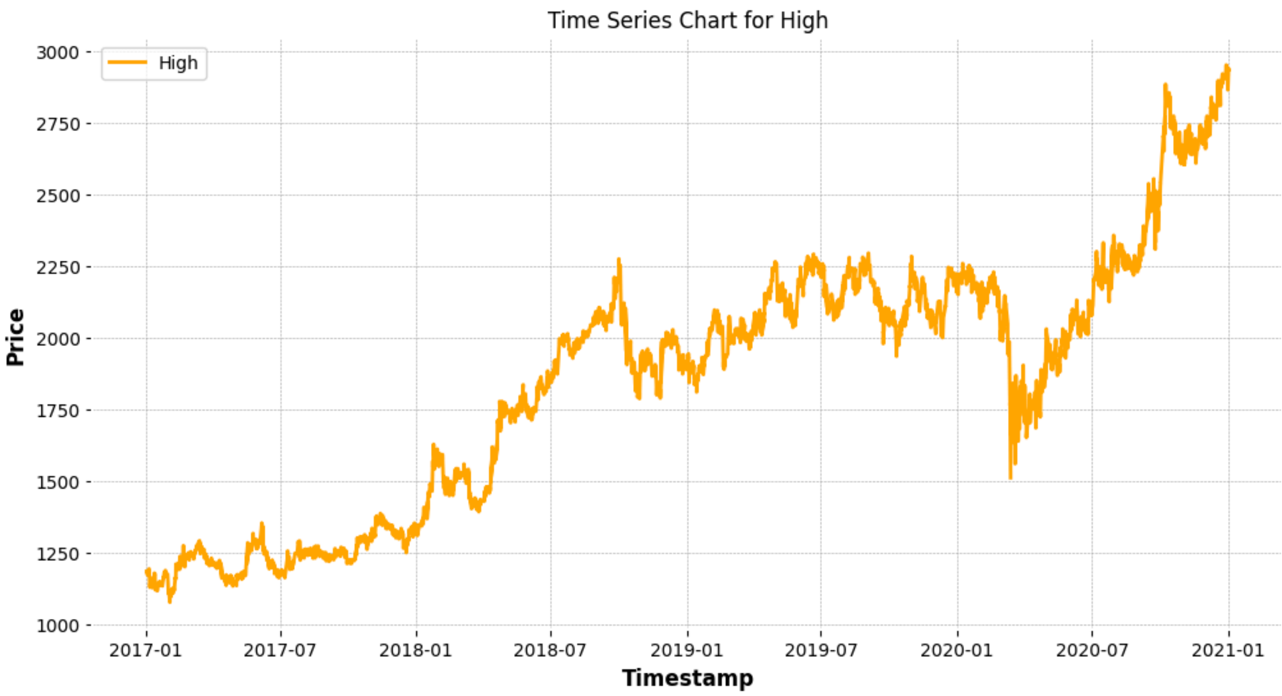

```
import matplotlib.pyplot as plt
# Plotting Time Series Chart for 'open' column
plt.figure(figsize=(12, 6))

plt.plot(df.index, df['open'], label='Open', color='lightblue')

plt.xlabel('Timestamp')
plt.ylabel('Price')
plt.title('Time Series Chart for Open')
plt.legend()
plt.show()
```



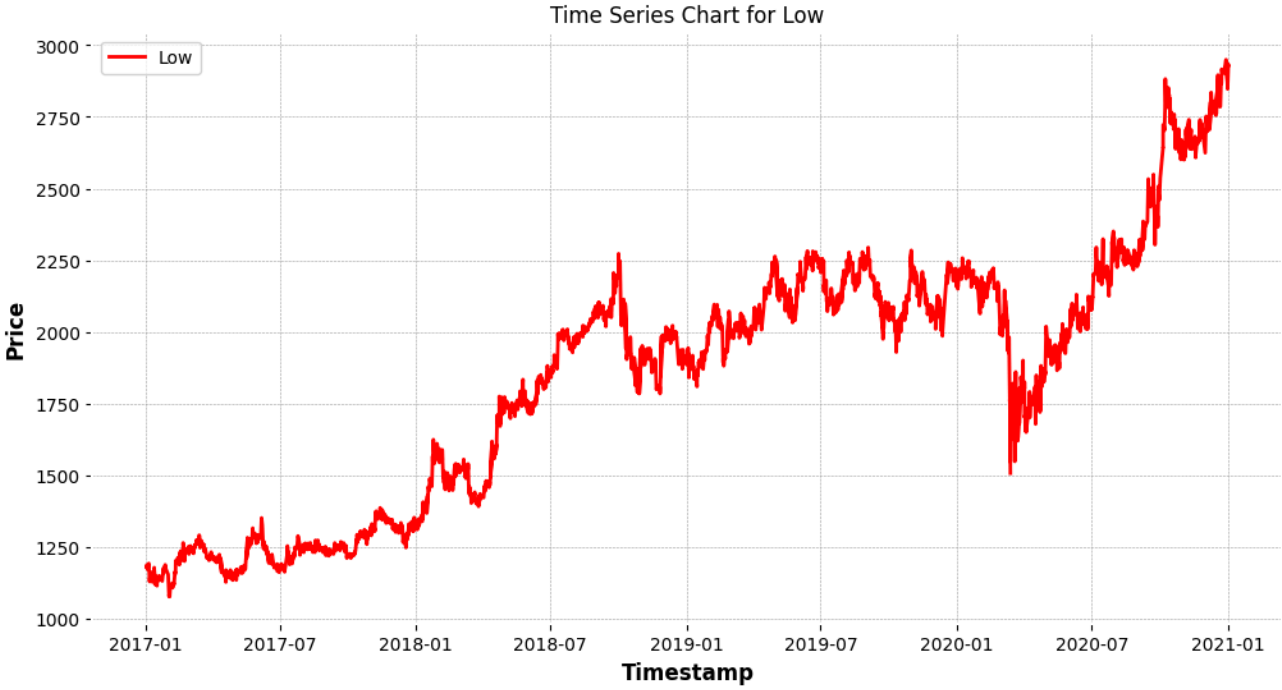

```python
import matplotlib.pyplot as plt
# Plotting Time Series Chart for 'close' column
plt.figure(figsize=(12, 6))

plt.plot(df.index, df['close'], label='Close', color='lightgreen')

plt.xlabel('Timestamp')
plt.ylabel('Price')
plt.title('Time Series Chart for Close')
plt.legend()
plt.show()
```




```
import matplotlib.pyplot as plt
# Plotting Time Series Chart for 'high' column
plt.figure(figsize=(12, 6))

plt.plot(df.index, df['high'], label='High', color='orange')

plt.xlabel('Timestamp')
plt.ylabel('Price')
plt.title('Time Series Chart for High')
plt.legend()
plt.show()
```




```
import matplotlib.pyplot as plt
# Plotting Time Series Chart for 'low' column
plt.figure(figsize=(12, 6))

plt.plot(df.index, df['low'], label='Low', color='red')

plt.xlabel('Timestamp')
plt.ylabel('Price')
plt.title('Time Series Chart for Low')
plt.legend()
plt.show()
```



Time series forcasting

Here we will do live stoc market forcasting

Importing and downloading library requires

```
!pip install yfinance plotly pandas statsmodels
```

```
Requirement already satisfied: yfinance in /usr/local/lib/python3.10/dist-packages (0
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages (5.2
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (2.2
Requirement already satisfied: statsmodels in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: numpy>=1.16.5 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: requests>=2.31 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: lxml>=4.9.1 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: platformdirs>=2.0.0 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: pytz>=2022.5 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: frozendict>=2.3.4 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: peewee>=3.16.2 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: beautifulsoup4>=4.11.1 in /usr/local/lib/python3.10/di
Requirement already satisfied: html5lib>=1.1 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/di
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: scipy!=1.9.2,>=1.8 in /usr/local/lib/python3.10/dist-p
```

```
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: six>=1.9 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-p
```

Genrating data online from yfinance

```
import yfinance as yf

# Fetch live stock data for TCS (NSE)
ticker = 'TCS.NS'
data = yf.download(ticker, period='1d', interval='1m')

# Preview the data
print(data.head())
```

```
[*********************100%***********************]  1 of 1 completed
Datetime
2024-10-07 09:15:00+05:30  4273.899902  4276.950195  4262.799805  4276.200195
2024-10-07 09:16:00+05:30  4276.250000  4279.450195  4274.000000  4276.649902
2024-10-07 09:17:00+05:30  4277.000000  4278.600098  4270.149902  4272.000000
2024-10-07 09:18:00+05:30  4271.950195  4278.200195  4268.899902  4273.799805
2024-10-07 09:19:00+05:30  4272.100098  4275.350098  4267.149902  4271.000000

                             Adj Close  Volume
Datetime
2024-10-07 09:15:00+05:30  4276.200195       0
2024-10-07 09:16:00+05:30  4276.649902    8536
2024-10-07 09:17:00+05:30  4272.000000    7912
2024-10-07 09:18:00+05:30  4273.799805    8070
2024-10-07 09:19:00+05:30  4271.000000    4237
```

decplaying the data and downloading the data in csv format

```
import pandas as pd

# Display the data in a table format
print(data.to_string())

# Create a CSV file and download it
from google.colab import files
data.to_csv('tcs_stock_data.csv')
files.download('tcs_stock_data.csv')
```

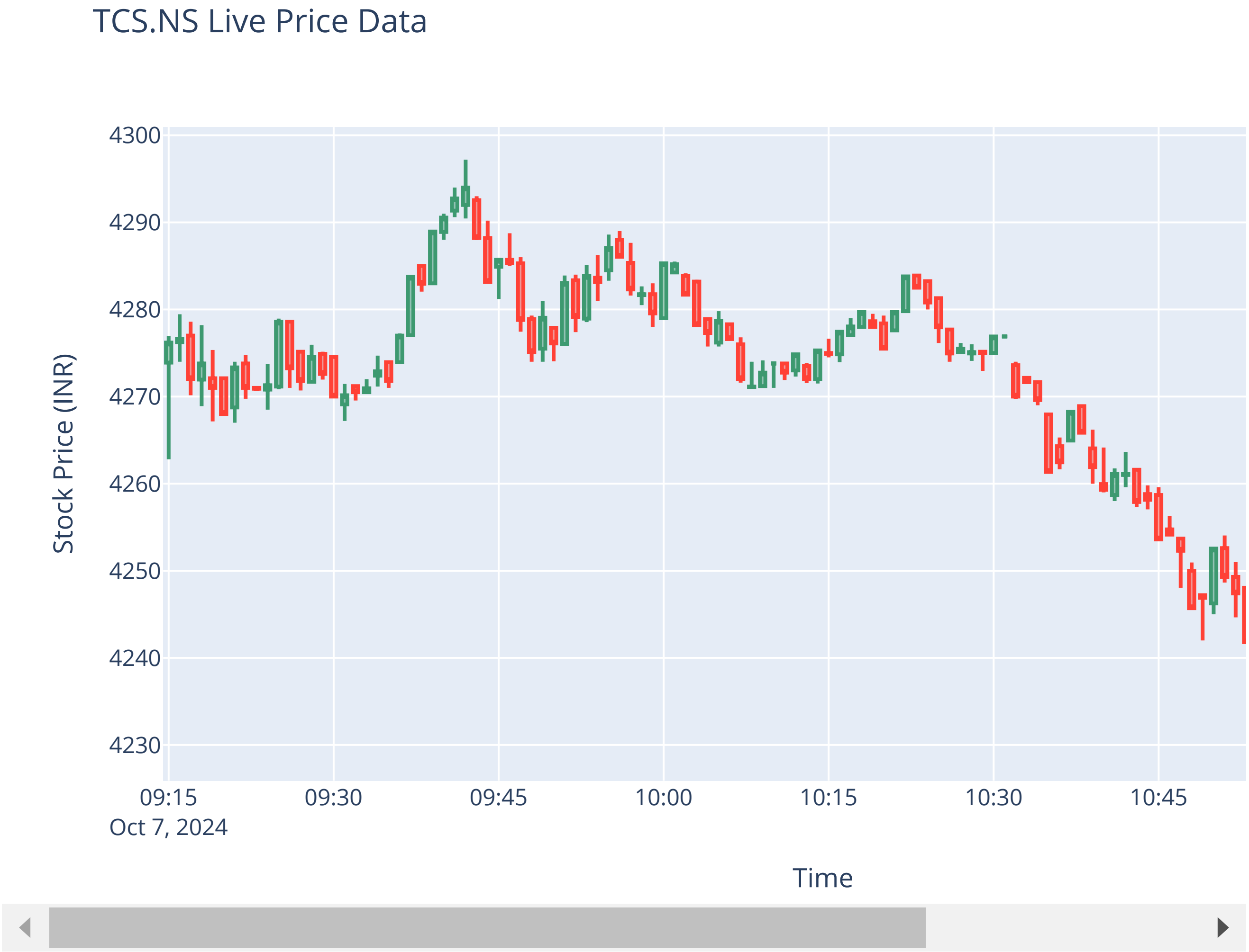
Datetime	Open	High	Low	Close	Adj
2024-10-07 09:15:00+05:30	4273.899902	4276.950195	4262.799805	4276.200195	4276.2
2024-10-07 09:16:00+05:30	4276.250000	4279.450195	4274.000000	4276.649902	4276.6
2024-10-07 09:17:00+05:30	4277.000000	4278.600098	4270.149902	4272.000000	4272.0
2024-10-07 09:18:00+05:30	4271.950195	4278.200195	4268.899902	4273.799805	4273.7
2024-10-07 09:19:00+05:30	4272.100098	4275.350098	4267.149902	4271.000000	4271.0
2024-10-07 09:20:00+05:30	4272.100098	4272.149902	4268.000000	4268.000000	4268.0
2024-10-07 09:21:00+05:30	4268.750000	4274.000000	4267.000000	4273.350098	4273.3
2024-10-07 09:22:00+05:30	4273.799805	4274.799805	4269.750000	4271.000000	4271.0
2024-10-07 09:23:00+05:30	4271.000000	4271.000000	4270.899902	4270.899902	4270.8
2024-10-07 09:24:00+05:30	4270.799805	4273.750000	4268.500000	4271.299805	4271.2
2024-10-07 09:25:00+05:30	4271.149902	4278.950195	4270.850098	4278.600098	4278.6
2024-10-07 09:26:00+05:30	4278.600098	4278.600098	4271.000000	4273.250000	4273.2
2024-10-07 09:27:00+05:30	4275.149902	4275.149902	4270.700195	4271.799805	4271.7
2024-10-07 09:28:00+05:30	4271.700195	4275.950195	4271.700195	4274.549805	4274.5
2024-10-07 09:29:00+05:30	4274.899902	4275.149902	4271.950195	4272.600098	4272.6
2024-10-07 09:30:00+05:30	4274.549805	4274.549805	4270.000000	4270.000000	4270.0
2024-10-07 09:31:00+05:30	4269.100098	4271.450195	4267.200195	4270.149902	4270.1
2024-10-07 09:32:00+05:30	4271.200195	4271.299805	4269.549805	4270.500000	4270.5
2024-10-07 09:33:00+05:30	4270.500000	4272.000000	4270.399902	4270.950195	4270.9
2024-10-07 09:34:00+05:30	4272.399902	4274.700195	4271.149902	4272.899902	4272.8
2024-10-07 09:35:00+05:30	4273.950195	4273.950195	4271.000000	4271.799805	4271.7
2024-10-07 09:36:00+05:30	4273.950195	4277.250000	4273.950195	4277.000000	4277.0
2024-10-07 09:37:00+05:30	4277.049805	4283.750000	4277.049805	4283.750000	4283.7
2024-10-07 09:38:00+05:30	4285.000000	4285.000000	4282.049805	4282.950195	4282.9
2024-10-07 09:39:00+05:30	4283.000000	4288.950195	4283.000000	4288.950195	4288.9
2024-10-07 09:40:00+05:30	4288.899902	4291.000000	4288.000000	4290.549805	4290.5
2024-10-07 09:41:00+05:30	4291.350098	4294.000000	4290.600098	4292.750000	4292.7
2024-10-07 09:42:00+05:30	4292.000000	4297.200195	4290.450195	4294.000000	4294.0
2024-10-07 09:43:00+05:30	4292.549805	4293.000000	4288.000000	4288.200195	4288.2
2024-10-07 09:44:00+05:30	4288.200195	4290.200195	4283.149902	4283.149902	4283.1
2024-10-07 09:45:00+05:30	4284.899902	4285.899902	4281.200195	4285.700195	4285.7
2024-10-07 09:46:00+05:30	4285.700195	4288.750000	4285.000000	4285.350098	4285.3
2024-10-07 09:47:00+05:30	4285.299805	4286.000000	4277.450195	4278.799805	4278.7
2024-10-07 09:48:00+05:30	4278.899902	4279.299805	4274.000000	4275.100098	4275.1
2024-10-07 09:49:00+05:30	4275.500000	4281.000000	4274.000000	4279.049805	4279.0
2024-10-07 09:50:00+05:30	4277.899902	4278.100098	4274.049805	4276.100098	4276.1
2024-10-07 09:51:00+05:30	4276.049805	4283.899902	4276.049805	4283.049805	4283.0
2024-10-07 09:52:00+05:30	4283.350098	4284.000000	4277.399902	4279.149902	4279.1
2024-10-07 09:53:00+05:30	4278.950195	4285.100098	4278.549805	4283.850098	4283.8
2024-10-07 09:54:00+05:30	4283.649902	4286.250000	4280.950195	4283.149902	4283.1
2024-10-07 09:55:00+05:30	4284.500000	4288.600098	4283.299805	4287.049805	4287.0
2024-10-07 09:56:00+05:30	4288.000000	4289.000000	4286.000000	4286.049805	4286.0
2024-10-07 09:57:00+05:30	4285.350098	4287.649902	4281.600098	4282.299805	4282.2
2024-10-07 09:58:00+05:30	4281.649902	4282.649902	4280.500000	4281.750000	4281.7
2024-10-07 09:59:00+05:30	4281.750000	4283.000000	4278.000000	4279.549805	4279.5
2024-10-07 10:00:00+05:30	4279.000000	4285.299805	4279.000000	4285.299805	4285.2
2024-10-07 10:01:00+05:30	4284.250000	4285.500000	4284.250000	4285.200195	4285.2
2024-10-07 10:02:00+05:30	4283.850098	4284.149902	4281.700195	4281.700195	4281.7
2024-10-07 10:03:00+05:30	4283.200195	4283.399902	4278.200195	4278.200195	4278.2
2024-10-07 10:04:00+05:30	4278.899902	4279.000000	4275.750000	4277.299805	4277.2
2024-10-07 10:05:00+05:30	4276.200195	4279.799805	4275.750000	4278.700195	4278.7
2024-10-07 10:06:00+05:30	4278.299805	4278.299805	4276.600098	4276.600098	4276.6
2024-10-07 10:07:00+05:30	4276.000000	4276.799805	4271.600098	4272.000000	4272.0
2024-10-07 10:08:00+05:30	4271.100098	4274.000000	4271.100098	4271.149902	4271.1
2024-10-07 10:09:00+05:30	4271.200195	4274.149902	4271.000000	4272.750000	4272.7
2024-10-07 10:10:00+05:30	4273.799805	4273.799805	4271.000000	4273.799805	4273.7
2024-10-07 10:11:00+05:30	4273.799805	4273.799805	4271.899902	4272.750000	4272.7
2024-10-07 10:12:00+05:30	4273.000000	4274.850098	4272.299805	4274.850098	4274.8

2024-10-07 10:13:00+05:30	4273.500000	4273.850098	4271.549805	4272.000000	4272.0
2024-10-07 10:14:00+05:30	4272.000000	4275.299805	4271.500000	4275.299805	4275.2
2024-10-07 10:15:00+05:30	4275.000000	4276.649902	4274.500000	4274.899902	4274.8
2024-10-07 10:16:00+05:30	4274.850098	4277.450195	4273.950195	4277.450195	4277.4
2024-10-07 10:17:00+05:30	4277.049805	4279.000000	4277.049805	4278.049805	4278.0
2024-10-07 10:18:00+05:30	4278.000000	4279.950195	4278.000000	4279.700195	4279.7
2024-10-07 10:19:00+05:30	4278.649902	4279.500000	4278.000000	4278.000000	4278.0
2024-10-07 10:20:00+05:30	4278.399902	4279.299805	4275.500000	4275.500000	4275.5
2024-10-07 10:21:00+05:30	4277.600098	4279.750000	4277.549805	4279.750000	4279.7
2024-10-07 10:22:00+05:30	4279.799805	4283.899902	4279.799805	4283.799805	4283.7
2024-10-07 10:23:00+05:30	4283.899902	4284.100098	4282.500000	4282.500000	4282.5
2024-10-07 10:24:00+05:30	4283.250000	4283.250000	4280.000000	4280.799805	4280.7
2024-10-07 10:25:00+05:30	4281.299805	4281.299805	4276.149902	4277.950195	4277.9
2024-10-07 10:26:00+05:30	4277.700195	4277.799805	4274.000000	4274.950195	4274.9
2024-10-07 10:27:00+05:30	4275.100098	4276.149902	4275.100098	4275.500000	4275.5
2024-10-07 10:28:00+05:30	4274.950195	4276.000000	4274.100098	4275.100098	4275.1
2024-10-07 10:29:00+05:30	4275.149902	4275.299805	4272.950195	4275.000000	4275.0
2024-10-07 10:30:00+05:30	4275.000000	4276.899902	4275.000000	4276.899902	4276.8
2024-10-07 10:31:00+05:30	4276.899902	4276.899902	4276.899902	4276.899902	4276.8
2024-10-07 10:32:00+05:30	4273.600098	4274.000000	4269.750000	4270.000000	4270.0
2024-10-07 10:33:00+05:30	4272.149902	4272.200195	4271.649902	4271.649902	4271.6
2024-10-07 10:34:00+05:30	4271.649902	4271.649902	4269.000000	4269.600098	4269.6
2024-10-07 10:35:00+05:30	4267.950195	4268.149902	4261.350098	4261.350098	4261.3
2024-10-07 10:36:00+05:30	4264.299805	4265.299805	4261.649902	4262.450195	4262.4
2024-10-07 10:37:00+05:30	4264.950195	4268.250000	4264.950195	4268.250000	4268.2
2024-10-07 10:38:00+05:30	4268.899902	4268.899902	4265.850098	4265.850098	4265.8
2024-10-07 10:39:00+05:30	4264.049805	4266.200195	4260.000000	4261.899902	4261.8
2024-10-07 10:40:00+05:30	4259.950195	4264.149902	4259.000000	4259.299805	4259.2
2024-10-07 10:41:00+05:30	4258.700195	4261.750000	4258.000000	4261.100098	4261.1
2024-10-07 10:42:00+05:30	4261.000000	4263.649902	4259.600098	4261.149902	4261.1
2024-10-07 10:43:00+05:30	4261.600098	4261.600098	4257.299805	4257.899902	4257.8
2024-10-07 10:44:00+05:30	4258.799805	4259.799805	4257.049805	4258.149902	4258.1
2024-10-07 10:45:00+05:30	4258.700195	4259.600098	4253.600098	4253.600098	4253.6
2024-10-07 10:46:00+05:30	4254.750000	4256.299805	4254.000000	4254.149902	4254.1
2024-10-07 10:47:00+05:30	4253.700195	4253.700195	4248.049805	4252.299805	4252.2
2024-10-07 10:48:00+05:30	4250.000000	4250.950195	4245.700195	4245.700195	4245.7
2024-10-07 10:49:00+05:30	4247.200195	4247.200195	4242.000000	4246.899902	4246.8
2024-10-07 10:50:00+05:30	4246.250000	4252.549805	4245.000000	4252.549805	4252.5
2024-10-07 10:51:00+05:30	4252.600098	4254.049805	4248.649902	4249.299805	4249.2
2024-10-07 10:52:00+05:30	4249.299805	4251.000000	4244.649902	4247.399902	4247.3
2024-10-07 10:53:00+05:30	4248.049805	4248.250000	4241.600098	4241.799805	4241.7
2024-10-07 10:54:00+05:30	4241.799805	4245.000000	4241.250000	4243.450195	4243.4
2024-10-07 10:55:00+05:30	4244.799805	4247.799805	4243.200195	4246.700195	4246.7
2024-10-07 10:56:00+05:30	4246.700195	4249.549805	4244.399902	4249.200195	4249.2
2024-10-07 10:57:00+05:30	4249.600098	4254.149902	4246.700195	4252.950195	4252.9
2024-10-07 10:58:00+05:30	4251.450195	4253.350098	4245.049805	4245.649902	4245.6
2024-10-07 10:59:00+05:30	4245.649902	4249.549805	4244.299805	4246.850098	4246.8
2024-10-07 11:00:00+05:30	4247.000000	4247.350098	4244.450195	4247.350098	4247.3
2024-10-07 11:01:00+05:30	4247.049805	4247.299805	4242.000000	4242.549805	4242.5
2024-10-07 11:02:00+05:30	4239.149902	4239.899902	4235.700195	4235.700195	4235.7
2024-10-07 11:03:00+05:30	4237.299805	4238.850098	4237.250000	4238.850098	4238.8
2024-10-07 11:04:00+05:30	4237.750000	4237.750000	4233.750000	4234.500000	4234.5
2024-10-07 11:05:00+05:30	4234.450195	4235.399902	4232.399902	4232.399902	4232.3
2024-10-07 11:06:00+05:30	4230.000000	4233.799805	4229.649902	4233.700195	4233.7
2024-10-07 11:07:00+05:30	4233.700195	4234.850098	4232.500000	4234.049805	4234.0
2024-10-07 11:08:00+05:30	4234.000000	4238.399902	4234.000000	4236.950195	4236.9
2024-10-07 11:09:00+05:30	4238.100098	4241.000000	4237.600098	4238.049805	4238.0
2024-10-07 11:10:00+05:30	4236.000000	4236.700195	4235.399902	4236.100098	4236.1
2024-10-07 11:11:00+05:30	4236.000000	4238.200195	4235.600098	4237.200195	4237.2
2024-10-07 11:12:00+05:30	4237.200195	4237.200195	4233.149902	4233.149902	4233.1

```
2024-10-07 11:13:00+05:30  4232.799805  4234.649902  4231.950195  4234.299805  4234.2
```

Visulizing the data

```
import plotly.graph_objects as go
import pandas as pd

# Real-time plot
fig = go.Figure()

# Candlestick chart for open, high, low, close
fig.add_trace(go.Candlestick(x=data.index,
                open=data['Open'],
                high=data['High'],
                low=data['Low'],
                close=data['Close'],
                name='Market Data'))

# Add titles and labels
fig.update_layout(
    title=f'{ticker} Live Price Data',
    xaxis_title='Time',
    yaxis_title='Stock Price (INR)',
    xaxis_rangeslider_visible=False)

fig.show()
```



Forcasting using ARIMA(auto-regressive integrated moving avrage)

```
import pandas as pd
from statsmodels.tsa.arima.model import ARIMA

# Forecast using ARIMA on the closing price
close_prices = data['Close'].dropna()

# Fit ARIMA model
model = ARIMA(close_prices, order=(5,1,0))
model_fit = model.fit()

# Forecast next 10 minutes
forecast = model_fit.forecast(steps=10)
print(forecast)
```

```
2024-10-07 11:15:00+05:30    4240.068031
2024-10-07 11:16:00+05:30    4239.201877
2024-10-07 11:17:00+05:30    4239.704185
2024-10-07 11:18:00+05:30    4240.241432
2024-10-07 11:19:00+05:30    4240.263906
2024-10-07 11:20:00+05:30    4240.132447
2024-10-07 11:21:00+05:30    4240.177726
2024-10-07 11:22:00+05:30    4240.247877
```

```
2024-10-07 11:23:00+05:30    4240.250389
2024-10-07 11:24:00+05:30    4240.235989
Freq: min, Name: predicted_mean, dtype: float64
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueW

No frequency information was provided, so inferred frequency min will be used.

/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueW

No frequency information was provided, so inferred frequency min will be used.

/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueW

No frequency information was provided, so inferred frequency min will be used.
```

visulizing forcasting

```
import numpy as np

# Combine live data and forecasts
forecast_index = pd.date_range(data.index[-1], periods=11, freq='T')[1:]
forecast_series = pd.Series(forecast, index=forecast_index)

# Plot the forecast
fig.add_trace(go.Scatter(x=forecast_series.index, y=forecast_series, mode='lines', name='

# Update the figure to include forecasts
fig.show()
```

```
<ipython-input-37-5239c802d693>:4: FutureWarning:

'T' is deprecated and will be removed in a future version, please use 'min' instead.
```

Recommendation using random forest

```
import pandas as pd
import numpy as np

# Add moving averages as features
data['SMA_5'] = data['Close'].rolling(window=5).mean()
data['SMA_10'] = data['Close'].rolling(window=10).mean()

# Add percentage change feature
data['Pct_Change'] = data['Close'].pct_change()

# Add volatility (standard deviation of the closing price)
data['Volatility'] = data['Close'].rolling(window=10).std()

# Drop any rows with NaN values after adding features
data.dropna(inplace=True)
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Label the target: 1 for hold, 0 for sell
# You can define the logic for labeling (e.g., if price increases 5% in next n minutes)
data['Target'] = np.where(data['Close'].shift(-1) > data['Close'], 1, 0)

# Features and target variable
features = ['SMA_5', 'SMA_10', 'Pct_Change', 'Volatility']
X = data[features]
y = data['Target']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train a RandomForest Classifier
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Predict on test data and evaluate
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy*100:.2f}%")
```

```
Model Accuracy: 41.18%
```

```
import yfinance as yf
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Fetch historical stock data for TCS (NSE)
ticker = 'TCS.NS'
data = yf.download(ticker, period='1y', interval='1d')

# Feature Engineering: Creating moving averages and technical indicators
data['SMA_10'] = data['Close'].rolling(window=10).mean()
data['SMA_20'] = data['Close'].rolling(window=20).mean()

# Lagging the 'Close' column to predict future price movements
data['Price_Change'] = data['Close'].shift(-1) - data['Close']
data['Target'] = np.where(data['Price_Change'] > 0, 0, 1)  # 0 = Hold, 1 = Sell

# Drop missing values caused by rolling window calculations
data.dropna(inplace=True)

# Features (X) and Target (y)
X = data[['SMA_10', 'SMA_20', 'Close']]  # Features: Moving averages and close price
y = data['Target']  # Target: Hold (0) or Sell (1)
```