Step 1: Load All Datasets

Purpose: In this step, we load the four datasets into separate Pandas DataFrames. This allows us to independently analyze and manipulate each dataset. By displaying the first few rows, we verify that the data has been loaded correctly and that we can view the structure of each dataset.

```
import yfinance as yf
import pandas as pd


dataset_1=yf.Ticker("^GSPC")
dataset_1=dataset_1.history(period="max")
dataset_2=pd.read_csv("/content/drive/MyDrive/dataset1.csv")
dataset_3=pd.read_csv("/content/drive/MyDrive/ADANIPORTS.csv")
dataset_4=pd.read_csv("/content/drive/MyDrive/RELIANCE.csv")
dataset_5=pd.read_csv("/content/drive/MyDrive/NIFTY50_all.csv")

# Displaying first few rows of each dataset
print("Dataset 1:")
print(dataset_1.head())

print("\nDataset 2:")
print(dataset_2.head())

print("\nDataset 3:")
print(dataset_3.head())

print("\nDataset 4:")
print(dataset_4.head())

print("\nDataset 5:")
print(dataset_5.head())
```

```
Dataset 1:
                                Open       High        Low      Close  Vo
Date
1927-12-30 00:00:00-05:00  17.660000  17.660000  17.660000  17.660000
1928-01-03 00:00:00-05:00  17.760000  17.760000  17.760000  17.760000
1928-01-04 00:00:00-05:00  17.719999  17.719999  17.719999  17.719999
1928-01-05 00:00:00-05:00  17.549999  17.549999  17.549999  17.549999
1928-01-06 00:00:00-05:00  17.660000  17.660000  17.660000  17.660000

                           Dividends  Stock Splits
Date
1927-12-30 00:00:00-05:00        0.0           0.0
1928-01-03 00:00:00-05:00        0.0           0.0
```

```
1928-01-04 00:00:00-05:00       0.0           0.0
1928-01-05 00:00:00-05:00       0.0           0.0
1928-01-06 00:00:00-05:00       0.0           0.0

Dataset 2:
         Date       Open       High        Low      Close  Adj Close    V
0  1999-11-18  32.546494  35.765381  28.612303  31.473534  27.068665  625
1  1999-11-19  30.713520  30.758226  28.478184  28.880543  24.838577  152
2  1999-11-22  29.551144  31.473534  28.657009  31.473534  27.068665   65
3  1999-11-23  30.400572  31.205294  28.612303  28.612303  24.607880   59
4  1999-11-24  28.701717  29.998211  28.612303  29.372318  25.261524   48

Dataset 3:
         Date      Symbol Series  Prev Close    Open     High    Low   La
0  2007-11-27  MUNDRAPORT     EQ      440.00  770.00  1050.00  770.0  959
1  2007-11-28  MUNDRAPORT     EQ      962.90  984.00   990.00  874.0  885
2  2007-11-29  MUNDRAPORT     EQ      893.90  909.00   914.75  841.0  887
3  2007-11-30  MUNDRAPORT     EQ      884.20  890.00   958.00  890.0  929
4  2007-12-03  MUNDRAPORT     EQ      921.55  939.75   995.00  922.0  980

    Close    VWAP    Volume      Turnover  Trades  Deliverable Volume  \
0  962.90  984.72  27294366  2.687719e+15     NaN             9859619
1  893.90  941.38   4581338  4.312765e+14     NaN             1453278
2  884.20  888.09   5124121  4.550658e+14     NaN             1069678
3  921.55  929.17   4609762  4.283257e+14     NaN             1260913
4  969.30  965.65   2977470  2.875200e+14     NaN              816123

   %Deliverble
0       0.3612
1       0.3172
2       0.2088
3       0.2735
4       0.2741

Dataset 4:
         Date    Symbol Series  Prev Close    Open    High     Low    Las
0  2000-01-03  RELIANCE     EQ      233.05  237.50  251.70  237.50  251.7
1  2000-01-04  RELIANCE     EQ      251.70  258.40  271.85  251.30  271.8
2  2000-01-05  RELIANCE     EQ      271.85  256.65  287.90  256.65  286.7
3  2000-01-06  RELIANCE     EQ      282.50  289.00  300.70  289.00  293.5
4  2000-01-07  RELIANCE     EQ      294.35  295.00  317.90  293.00  314.5

    Close    VWAP    Volume      Turnover  Trades  Deliverable Volume  \
0  251.70  249.37   4456424  1.111319e+14     NaN                 NaN
1  271.85  263.52   9487878  2.500222e+14     NaN                 NaN
2  282.50  274.79  26833684  7.373697e+14     NaN                 NaN
```

Step 2: Check Data Quality for Each Dataset

Purpose : We check the structure of each dataset using info(). This step gives us an understanding of the number of columns, data types, and any missing values in each dataset. It's essential to know what data we're working with and whether any columns need special attention (e.g., missing or improperly formatted values).

```
# Check dataset information for each dataset
print("Dataset 1 Information:")
dataset_1.info()

print("\nDataset 2 Information:")
dataset_2.info()

print("\nDataset 3 Information:")
dataset_3.info()

print("\nDataset 4 Information:")
dataset_4.info()


print("\nDataset 5 Information:")
dataset_5.info()
```

```
Dataset 1 Information:
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 24292 entries, 1927-12-30 00:00:00-05:00 to 2024-09-13 00:
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Open          24292 non-null  float64
 1   High          24292 non-null  float64
 2   Low           24292 non-null  float64
 3   Close         24292 non-null  float64
 4   Volume        24292 non-null  int64
 5   Dividends     24292 non-null  float64
 6   Stock Splits  24292 non-null  float64
dtypes: float64(6), int64(1)
memory usage: 1.5 MB

Dataset 2 Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5124 entries, 0 to 5123
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       5124 non-null   object
 1   Open       5124 non-null   float64
```

```
 2   High       5124 non-null   float64
 3   Low        5124 non-null   float64
 4   Close      5124 non-null   float64
 5   Adj Close  5124 non-null   float64
 6   Volume     5124 non-null   int64
dtypes: float64(5), int64(1), object(1)
memory usage: 280.3+ KB

Dataset 3 Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3322 entries, 0 to 3321
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Date                3322 non-null   object
 1   Symbol              3322 non-null   object
 2   Series              3322 non-null   object
 3   Prev Close          3322 non-null   float64
 4   Open                3322 non-null   float64
 5   High                3322 non-null   float64
 6   Low                 3322 non-null   float64
 7   Last                3322 non-null   float64
 8   Close               3322 non-null   float64
 9   VWAP                3322 non-null   float64
 10  Volume              3322 non-null   int64
 11  Turnover            3322 non-null   float64
 12  Trades              2456 non-null   float64
 13  Deliverable Volume  3322 non-null   int64
 14  %Deliverble         3322 non-null   float64
dtypes: float64(10), int64(2), object(3)
memory usage: 389.4+ KB

Dataset 4 Information:
<class 'pandas.core.frame.DataFrame'>
```

Step 3: Check for Missing Values

Purpose : This step identifies any missing values in the datasets. Missing values can lead to erroneous results, so it's important to determine how many missing entries are present in each column of each dataset before proceeding with further analysis.

```
# Check for missing values in each dataset
print("Missing values in Dataset 1:")
print(dataset_1.isnull().sum())

print("\nMissing values in Dataset 2:")
print(dataset_2.isnull().sum())

print("\nMissing values in Dataset 3:")
```

```
print(dataset_3.isnull().sum())

print("\nMissing values in Dataset 4:")
print(dataset_4.isnull().sum())

print("\nMissing values in Dataset 5:")
print(dataset_5.isnull().sum())
```

```
Missing values in Dataset 1:
Open            0
High            0
Low             0
Close           0
Volume          0
Dividends       0
Stock Splits    0
dtype: int64

Missing values in Dataset 2:
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64

Missing values in Dataset 3:
Date                    0
Symbol                  0
Series                  0
Prev Close              0
Open                    0
High                    0
Low                     0
Last                    0
Close                   0
VWAP                    0
Volume                  0
Turnover                0
Trades                866
Deliverable Volume      0
%Deliverble             0
dtype: int64

Missing values in Dataset 4:
Date                     0
Symbol                   0
Series                   0
Prev Close               0
Open                     0
High                     0
```

```
Low                  0
Last                 0
Close                0
VWAP                 0
Volume               0
Turnover             0
Trades            2850
Deliverable Volume 514
%Deliverble        514
dtype: int64

Missing values in Dataset 5:
Date                   0
Symbol                 0
```

Step 4: Handle Missing Values

Purpose: After identifying missing values, we handle them by filling in the missing values with the mean of the respective column. This is a standard approach to dealing with missing data, ensuring that we don't lose valuable data rows while avoiding introducing bias.

```
# Fill missing values with the mean for each dataset

dataset_2['Date'] = pd.to_datetime(dataset_2['Date'], errors='coerce')
dataset_3['Date'] = pd.to_datetime(dataset_3['Date'], errors='coerce')
del dataset_3['Symbol']
del dataset_3['Series']
dataset_4['Date'] = pd.to_datetime(dataset_4['Date'], errors='coerce')
del dataset_4['Symbol']
del dataset_4['Series']
dataset_5['Date'] = pd.to_datetime(dataset_5['Date'], errors='coerce')
del dataset_5['Symbol']
del dataset_5['Series']

dataset_1.fillna(dataset_1.mean(), inplace=True)
dataset_2.fillna(dataset_2.mean(), inplace=True)
dataset_3.fillna(dataset_3.mean(), inplace=True)
dataset_4.fillna(dataset_4.mean(), inplace=True)
dataset_5.fillna(dataset_5.mean(), inplace=True)

# Verify missing values have been filled
print("Missing values after handling in Dataset 1:")
print(dataset_1.isnull().sum())
print("Missing values after handling in Dataset 2:")
print(dataset_2.isnull().sum())
print("Missing values after handling in Dataset 3:")
print(dataset_3.isnull().sum())
print("Missing values after handling in Dataset 4:")
```

```
print(dataset_4.isnull().sum())
print("Missing values after handling in Dataset 5:")
print(dataset_5.isnull().sum())
```

```
Missing values after handling in Dataset 1:
Open            0
High            0
Low             0
Close           0
Volume          0
Dividends       0
Stock Splits    0
dtype: int64
Missing values after handling in Dataset 2:
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
Missing values after handling in Dataset 3:
Date                  0
Prev Close            0
Open                  0
High                  0
Low                   0
Last                  0
Close                 0
VWAP                  0
Volume                0
Turnover              0
Trades                0
Deliverable Volume    0
%Deliverble           0
dtype: int64
Missing values after handling in Dataset 4:
Date                  0
Prev Close            0
Open                  0
High                  0
Low                   0
Last                  0
Close                 0
VWAP                  0
Volume                0
Turnover              0
Trades                0
Deliverable Volume    0
%Deliverble           0
dtype: int64
Missing values after handling in Dataset 5:
```

```
Date            0
Prev Close      0
Open            0
High            0
Low             0
Last            0
Close           0
VWAP            0
Volume          0
Turnover        0
```

```
del dataset_1['Dividends']
del dataset_1['Stock Splits']
del dataset_2['Adj Close']
del dataset_3['Prev Close']
del dataset_3['Last']
del dataset_3['VWAP']
del dataset_3['Turnover']
del dataset_3['Trades']
del dataset_3['Deliverable Volume']
del dataset_4['Prev Close']
del dataset_4['Last']
del dataset_4['VWAP']
del dataset_4['Turnover']
del dataset_4['Trades']
del dataset_4['Deliverable Volume']

del dataset_5['Prev Close']
del dataset_5['Last']
del dataset_5['VWAP']
del dataset_5['Turnover']
del dataset_5['Trades']
del dataset_5['Deliverable Volume']
```

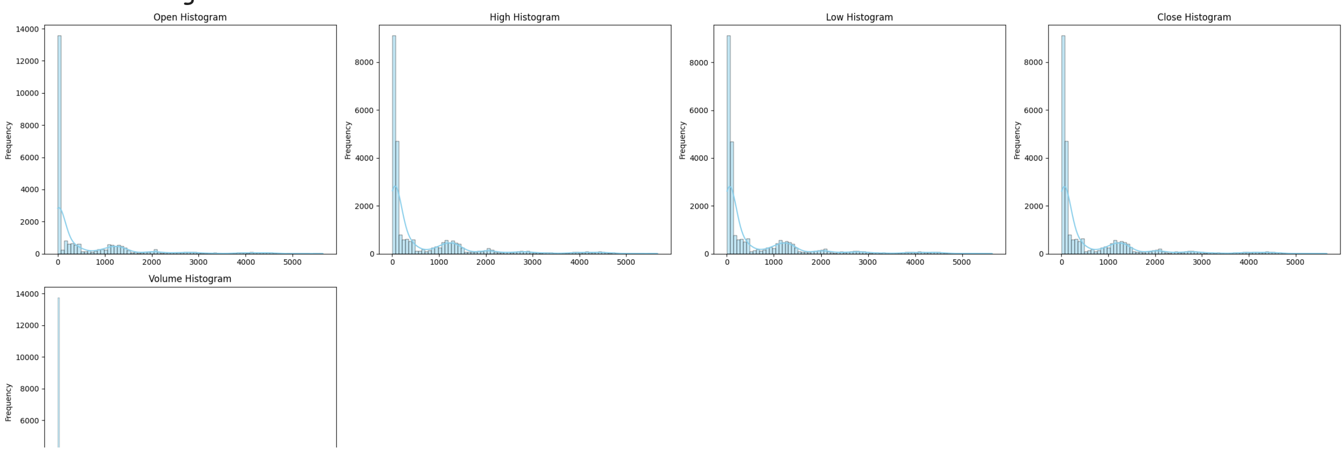
```
del dataset_4['%Deliverble']
del dataset_3['%Deliverble']
del dataset_5['%Deliverble']
```

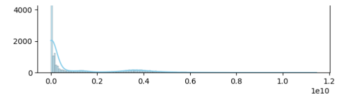
Step 5: Check for Dataset Imbalance

Purpose: In this step, we check the distribution of the target variable in each dataset to see if there's an imbalance. An imbalanced dataset (e.g., where one class is much larger than others) can affect model performance, so it's important to know if our datasets are balanced or not.

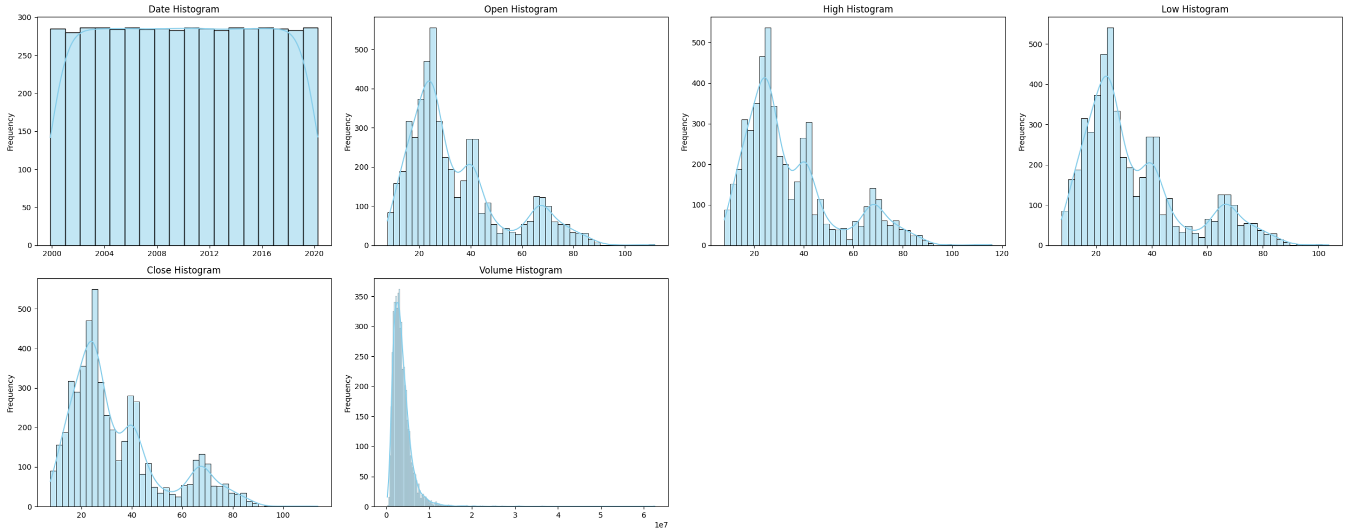
```
import seaborn as sns
import matplotlib.pyplot as plt

# Function to check and visualize the distribution of all attributes in the dat
def check_and_visualize_all_attributes(df, dataset_num, axarr):
    num_cols = len(df.columns)
    num_rows = (num_cols + 3) // 4

    for i, column in enumerate(df.columns):
        ax = axarr[i // 4, i % 4]
        if df[column].dtype == 'object' or df[column].nunique() < 10:
            sns.countplot(x=column, data=df, ax=ax)
            ax.set_title(f'{column} Count Distribution')
            ax.set_xlabel('')
            ax.set_ylabel('Count')
        else:
            sns.histplot(df[column], kde=True, color='skyblue', ax=ax)
            ax.set_title(f'{column} Histogram')
            ax.set_xlabel('')
            ax.set_ylabel('Frequency')

    for j in range(num_cols, axarr.size):
        axarr[j // 4, j % 4].axis('off')

# Check and visualize all attributes for each dataset
for i, df in enumerate([dataset_1, dataset_2, dataset_3, dataset_4,dataset_5],
    print(f"\nVisualizing attributes in Dataset {i}")

    num_cols = len(df.columns)
    num_rows = (num_cols + 3) // 4
    fig, axarr = plt.subplots(num_rows, 4, figsize=(25, 5 * num_rows), squeeze=

    check_and_visualize_all_attributes(df, i, axarr)
    plt.tight_layout()
    plt.show()
```

```
Visualizing attributes in Dataset 1
```

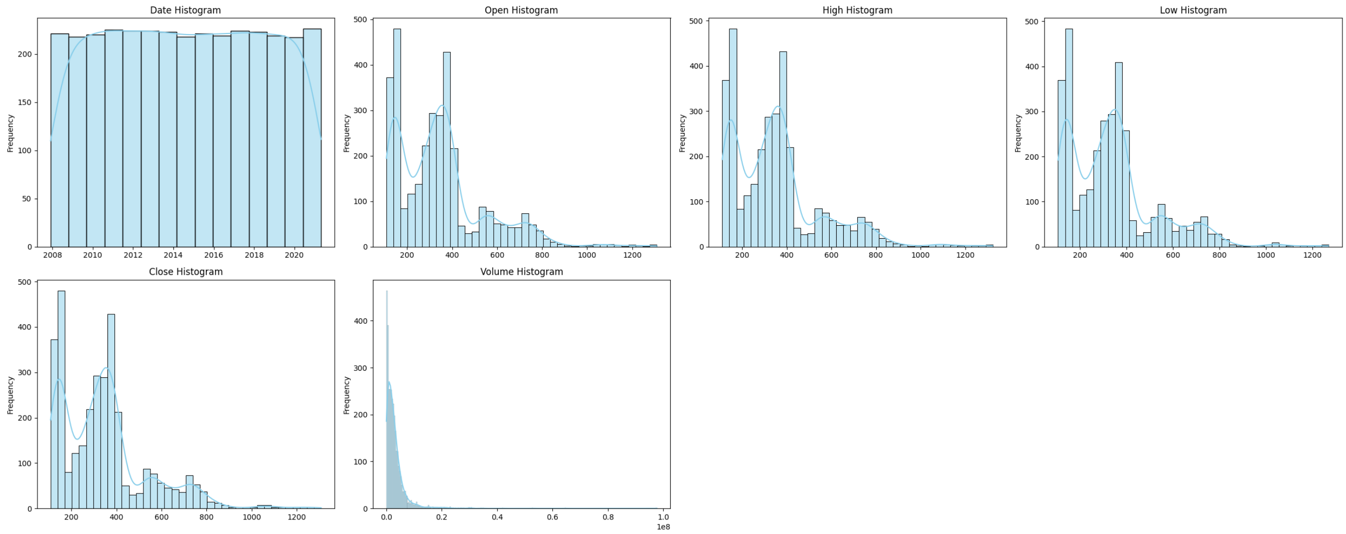




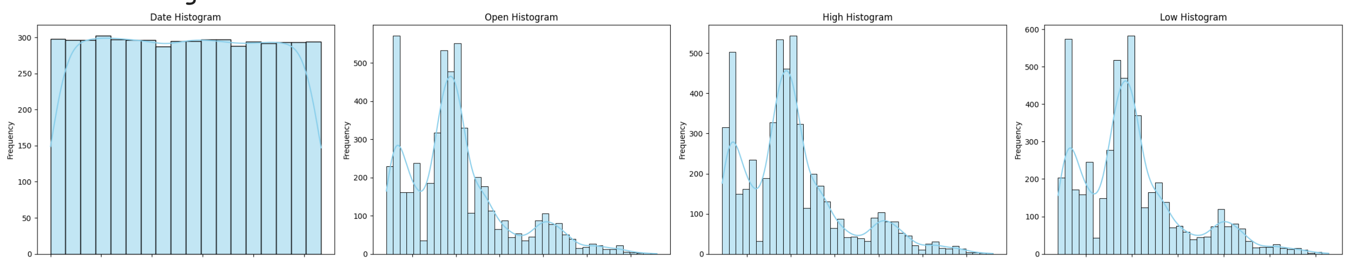
Visualizing attributes in Dataset 2



Visualizing attributes in Dataset 3



Visualizing attributes in Dataset 4



Step 6: Generate Descriptive Statistics for Each Dataset

Purpose: Generating descriptive statistics helps us understand the basic statistical properties of the dataset, such as the mean, median, standard deviation, and percentiles. This gives us insights into the central tendencies and the spread of the data.

1e7

```
# Generate descriptive statistics for each dataset
print("\nDescriptive Statistics for Dataset 1:")
print(dataset_1.describe())

print("\nDescriptive Statistics for Dataset 2:")
print(dataset_2.describe())

print("\nDescriptive Statistics for Dataset 3:")
print(dataset_3.describe())

print("\nDescriptive Statistics for Dataset 4:")
print(dataset_4.describe())

print("\nDescriptive Statistics for Dataset 5:")
print(dataset_5.describe())
```

```
Descriptive Statistics for Dataset 1:
               Open          High           Low         Close          Volu
count  24292.000000  24292.000000  24292.000000  24292.000000  2.429200e+
mean     622.359384    645.900625    638.164824    642.279419  9.097825e+
std     1058.350289   1053.136039   1041.265893   1047.581029  1.620352e+
min        0.000000      4.400000      4.400000      4.400000  0.000000e+
25%        9.700000     24.697501     24.697501     24.697501  1.530000e+
50%       42.924999    103.160004    101.500000    102.309998  2.043500e+
75%     1033.772522   1040.939941   1026.654999   1033.872528  9.937250e+
max     5644.089844   5669.669922   5639.020020   5667.200195  1.145623e+

Descriptive Statistics for Dataset 2:
                                Date         Open         High            L
count                           5124  5124.000000  5124.000000  5124.0000
mean   2010-01-26 03:25:25.995315968    34.090255    34.560553    33.6294
min              1999-11-18 00:00:00     7.653791     7.961373     7.5107
25%              2004-12-26 00:00:00    21.101574    21.452074    20.7850
50%              2010-01-27 12:00:00    27.328326    27.703863    27.0100
75%              2015-03-02 06:00:00    41.500000    41.860001    41.1300
max              2020-04-01 00:00:00   111.587982   115.879829   103.7195
std                              NaN    18.608831    18.834528    18.3817

             Close        Volume
count  5124.000000  5.124000e+03
mean     34.106245  3.693250e+06
min       7.761087  2.719000e+05
25%      21.130186  2.206475e+06
50%      27.396280  3.174050e+06
75%      41.525204  4.508075e+06
max     113.733902  6.254630e+07
std      18.611595  2.481855e+06

Descriptive Statistics for Dataset 3:
                                Date         Open         High            L
count                           3322  3322.000000  3322.000000  3322.0000
```

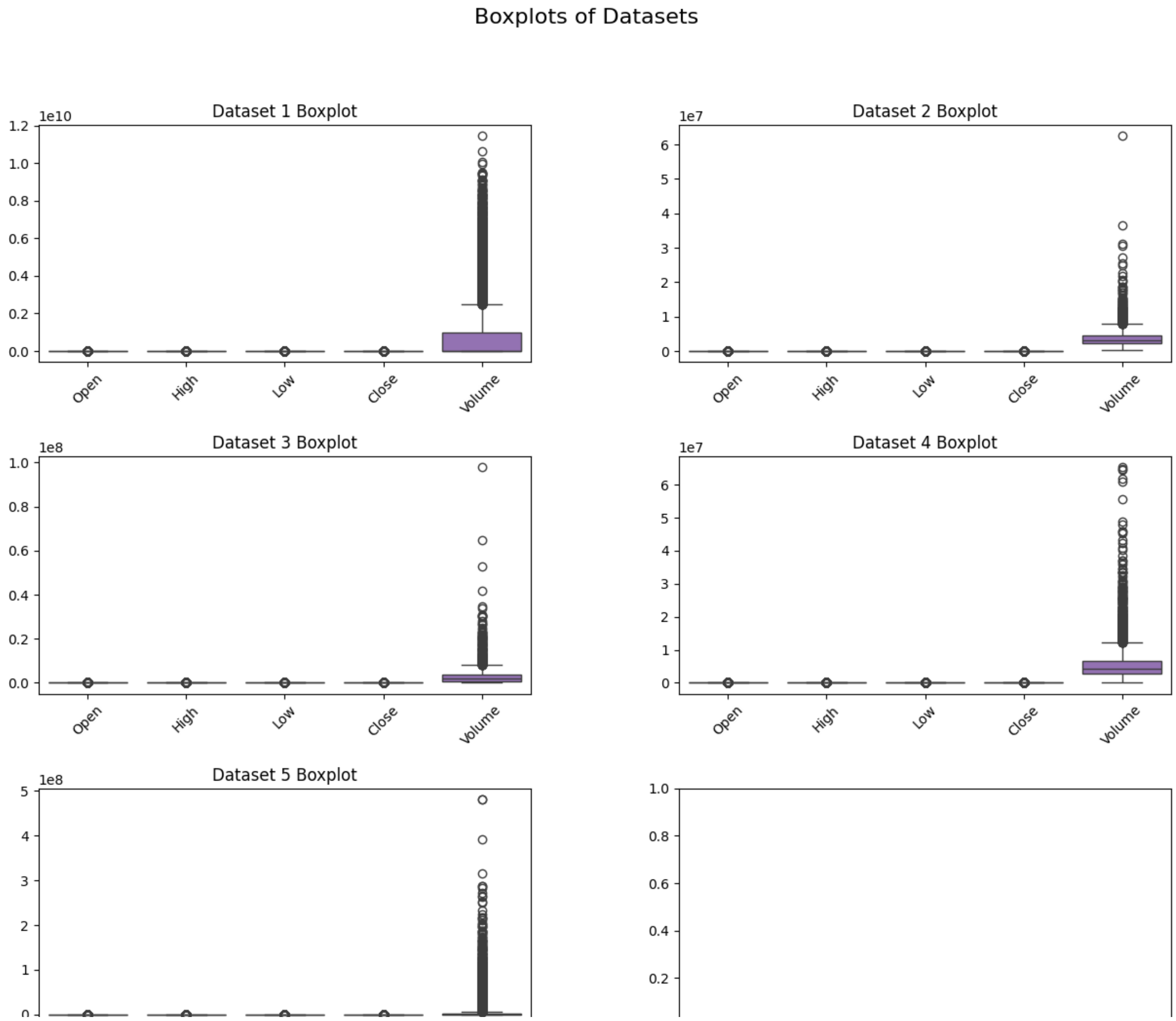
```
mean   2014-08-14 03:47:08.416616448   344.763019   351.608007   337.5319
min              2007-11-27 00:00:00   108.000000   110.450000   105.6500
25%              2011-04-07 06:00:00   164.850000   168.000000   161.6000
50%              2014-08-06 12:00:00   325.750000   331.275000   319.8500
75%              2017-12-18 18:00:00   401.000000   407.187500   395.0000
max              2021-04-30 00:00:00  1310.250000  1324.000000  1270.0000
std                              NaN   193.619992   198.617808   188.6766

             Close        Volume
count  3322.000000  3.322000e+03
mean    344.201626  2.954564e+06
min     108.000000  1.236600e+04
25%     164.312500  7.493682e+05
50%     324.700000  2.007292e+06
75%     400.912500  3.636883e+06
max    1307.450000  9.771788e+07
std     193.045886  4.104227e+06

Descriptive Statistics for Dataset 4:
                                Date         Open         High          L
count                           5306  5306.000000  5306.000000  5306.0000
mean   2010-08-18 21:26:56.132679936  1012.602375  1026.823803   996.8869
```

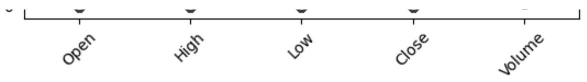
Step 7: Visualize Outliers Using Boxplots

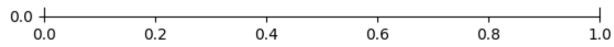
Purpose: Boxplots are useful for visually identifying outliers. Outliers are extreme values that can distort statistical analyses, so it’s crucial to detect and manage them. This step creates boxplots for each dataset, allowing us to see any values that are significantly higher or lower than the rest of the data.

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

# Create boxplots for all datasets
fig, axs = plt.subplots(3, 2, figsize=(15, 12))
fig.suptitle('Boxplots of Datasets', fontsize=16)

# Plot boxplot for Dataset 1
sns.boxplot(data=dataset_1, ax=axs[0, 0])
axs[0, 0].set_title('Dataset 1 Boxplot')
axs[0, 0].tick_params(axis='x', rotation=45)

# Plot boxplot for Dataset 2
sns.boxplot(data=dataset_2, ax=axs[0, 1])
axs[0, 1].set_title('Dataset 2 Boxplot')
axs[0, 1].tick_params(axis='x', rotation=45)
```

```
# Plot boxplot for Dataset 3
sns.boxplot(data=dataset_3, ax=axs[1, 0])
axs[1, 0].set_title('Dataset 3 Boxplot')
axs[1, 0].tick_params(axis='x', rotation=45)

# Plot boxplot for Dataset 4
sns.boxplot(data=dataset_4, ax=axs[1, 1])
axs[1, 1].set_title('Dataset 4 Boxplot')
axs[1, 1].tick_params(axis='x', rotation=45)

# Plot boxplot for Dataset 5
sns.boxplot(data=dataset_5, ax=axs[2, 0])
axs[2, 0].set_title('Dataset 5 Boxplot')
axs[2, 0].tick_params(axis='x', rotation=45)

# Adjust spacing between subplots
plt.subplots_adjust(hspace=0.4, wspace=0.3)

plt.show()
```





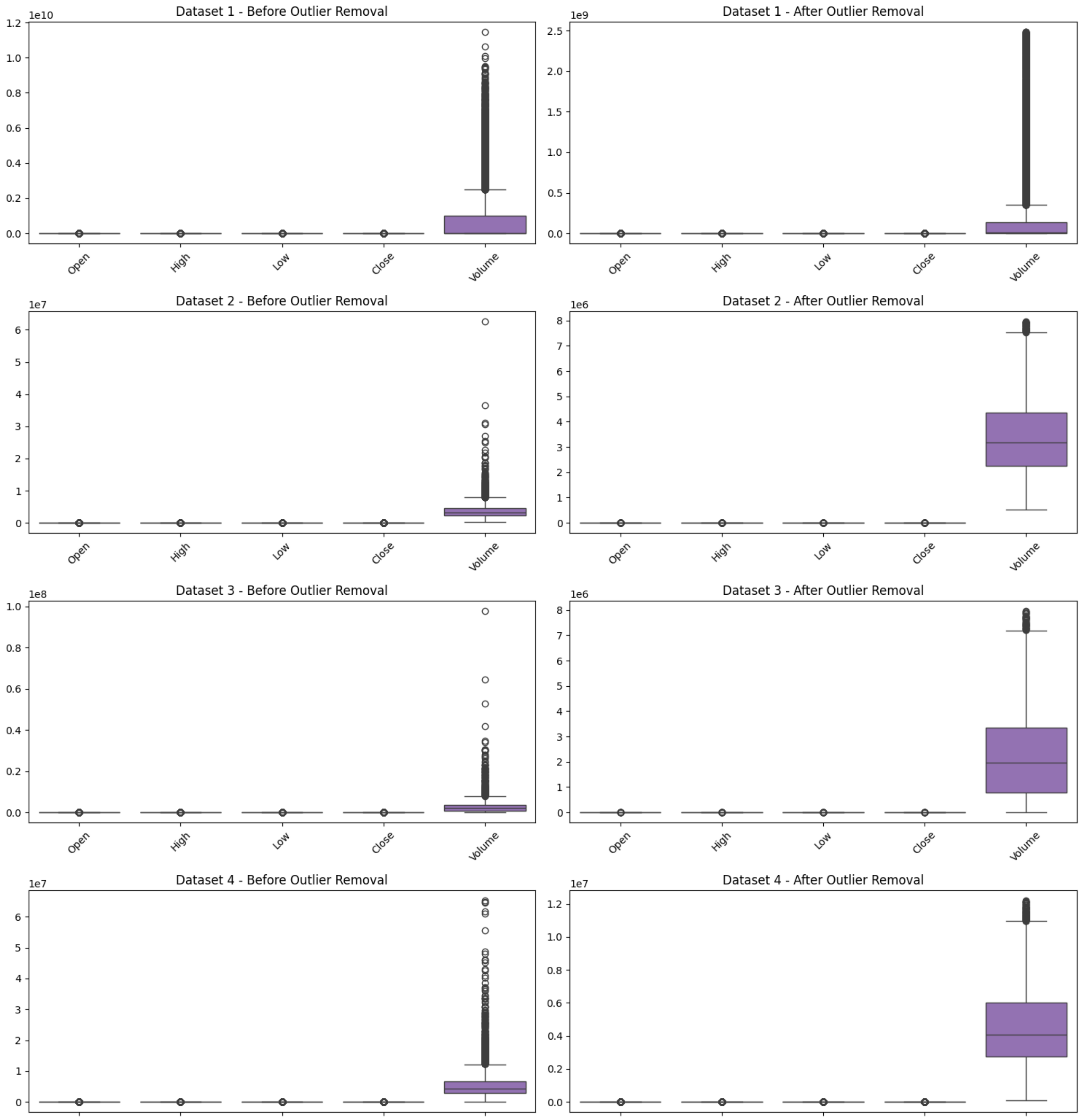


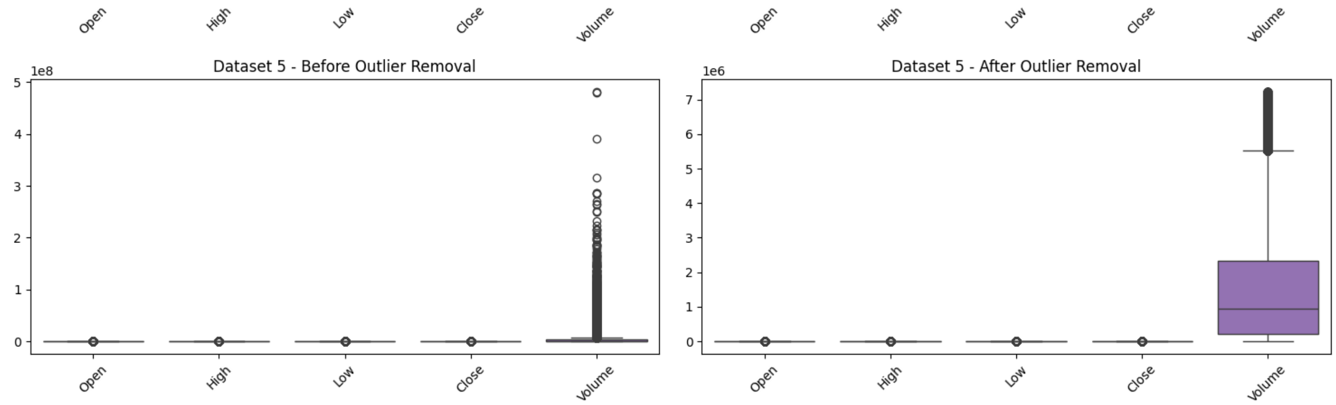
Step 8: Remove Outliers Using the IQR Method

Purpose: Outliers are removed using the Interquartile Range (IQR) method. This method identifies and excludes values that lie outside the range of typical data points (beyond 1.5 times the IQR). Removing outliers can improve the accuracy of our models and prevent skewed analyses.

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

# Prepare subplots for comparison
fig, axs = plt.subplots(5, 2, figsize=(15, 20))  # 5 rows, 2 columns

# Iterate over datasets to remove outliers and plot results
for i, df in enumerate([dataset_1, dataset_2, dataset_3, dataset_4,dataset_5],
    # Before removing outliers
    sns.boxplot(data=df, ax=axs[i-1, 0])
    axs[i-1, 0].set_title(f'Dataset {i} - Before Outlier Removal')
    axs[i-1, 0].tick_params(axis='x', rotation=45)  # Rotate x-axis labels

    # Calculate IQR and remove outliers
```

```
    Q1 = df.quantile(0.25)
    Q3 = df.quantile(0.75)
    IQR = Q3 - Q1
    df_cleaned = df[~((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).any(a›

    # After removing outliers
    sns.boxplot(data=df_cleaned, ax=axs[i-1, 1])
    axs[i-1, 1].set_title(f'Dataset {i} - After Outlier Removal')
    axs[i-1, 1].tick_params(axis='x', rotation=45)

# Adjust spacing between subplots
plt.tight_layout()
plt.show()
```






```
dataset_1["Tomorrow"]=dataset_1["Close"].shift(-1)
dataset_2["Tomorrow"]=dataset_2["Close"].shift(-1)
dataset_3["Tomorrow"]=dataset_3["Close"].shift(-1)
dataset_4["Tomorrow"]=dataset_4["Close"].shift(-1)
dataset_5["Tomorrow"]=dataset_5["Close"].shift(-1)
```

```
dataset_1["Target"]=(dataset_1["Tomorrow"]>dataset_1["Close"]).astype(int)
dataset_2["Target"]=(dataset_2["Tomorrow"]>dataset_2["Close"]).astype(int)
dataset_3["Target"]=(dataset_3["Tomorrow"]>dataset_3["Close"]).astype(int)
dataset_4["Target"]=(dataset_4["Tomorrow"]>dataset_4["Close"]).astype(int)
dataset_5["Target"]=(dataset_5["Tomorrow"]>dataset_5["Close"]).astype(int)
```

```
del dataset_2['Date']
del dataset_3['Date']
del dataset_4['Date']
del dataset_5['Date']
```

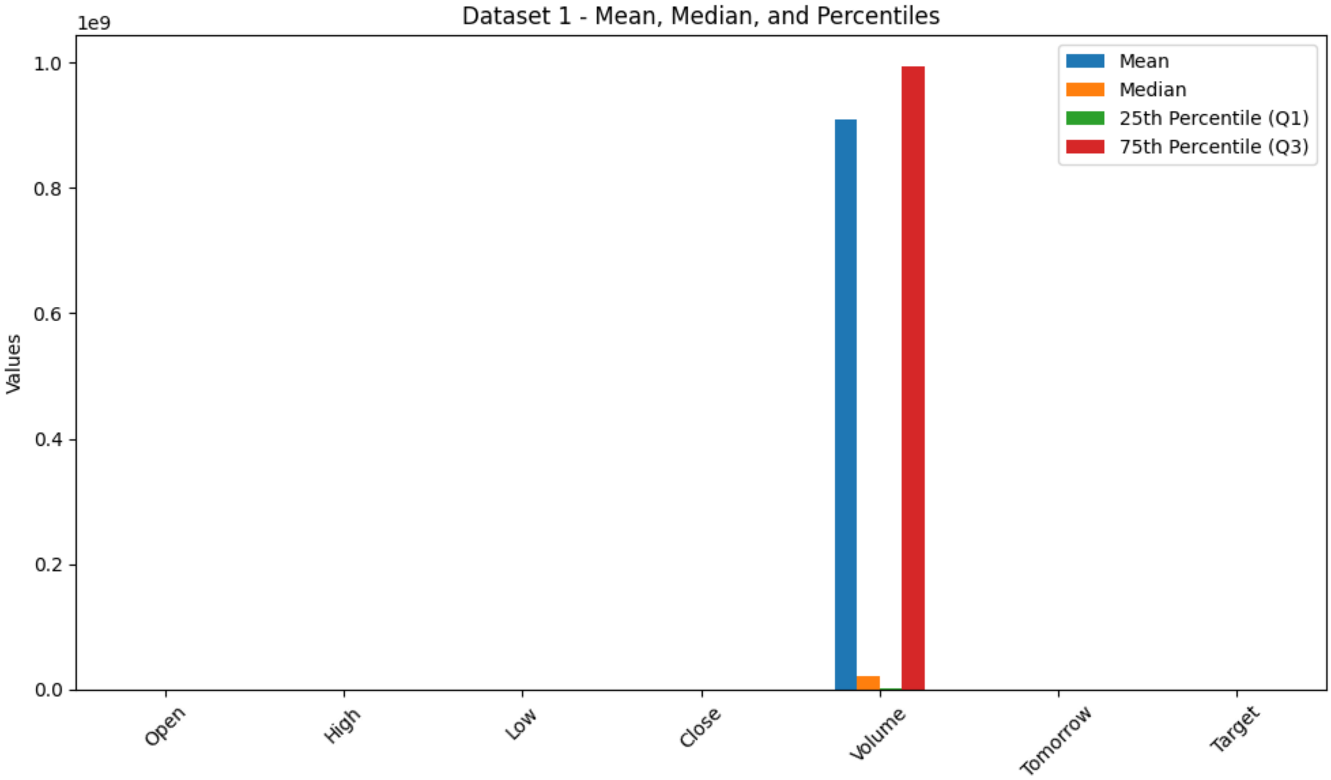
Step 9: Identify the Distribution Pattern of Data

Purpose: In this step, we calculate the mean, median, and percentiles (Q1 and Q3) for each dataset. This helps us understand the distribution pattern of the data, such as whether it's normally distributed, skewed, or has a high level of variance. Understanding the data distribution is crucial for model selection and performance.

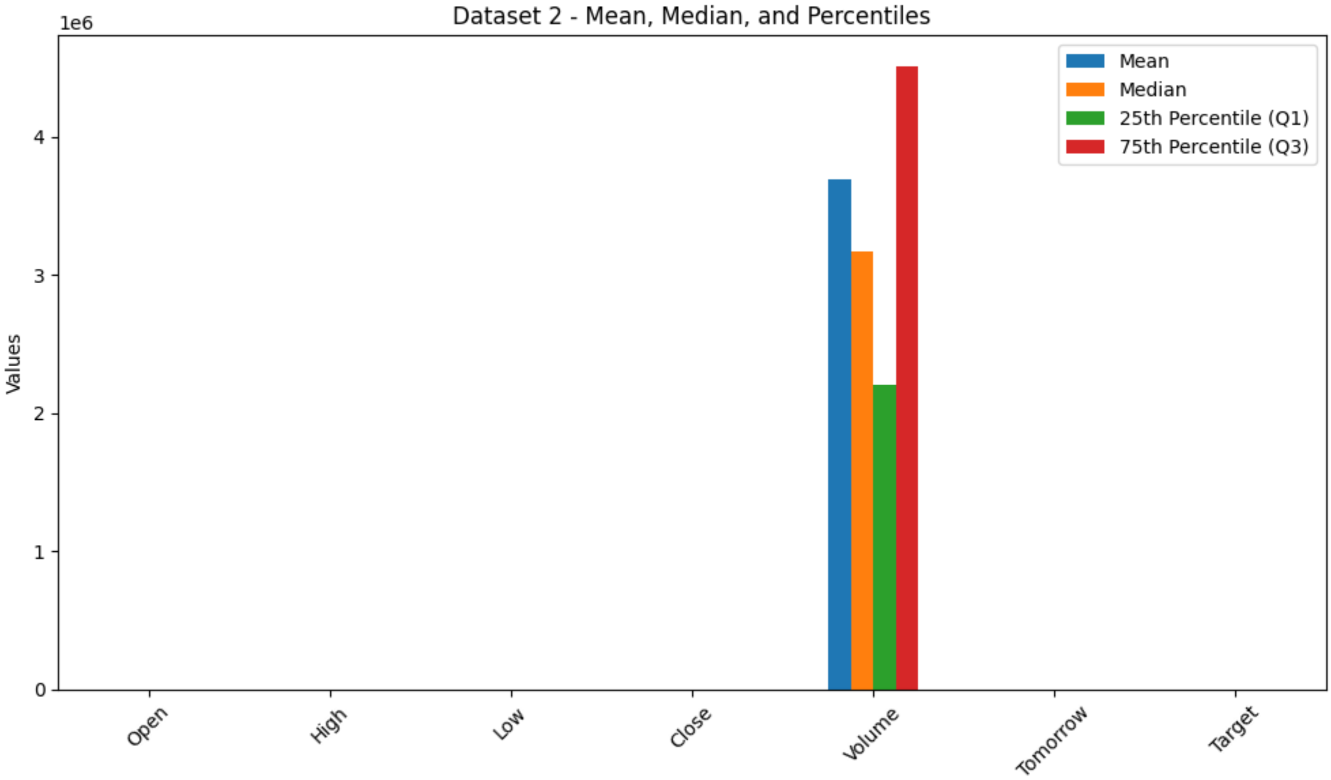
```
import seaborn as sns
```

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

# Iterate over datasets to calculate and plot mean, median, and percentiles
for i, df in enumerate([dataset_1, dataset_2, dataset_3, dataset_4,dataset_5],
    mean = df.mean()
    median = df.median()
    percentile_25 = df.quantile(0.25)
    percentile_75 = df.quantile(0.75)

    # Create a DataFrame for the statistics to plot
    stats_df = pd.DataFrame({
        'Mean': mean,
        'Median': median,
        '25th Percentile (Q1)': percentile_25,
        '75th Percentile (Q3)': percentile_75
    })

    # Plot the statistics for the current dataset
    plt.figure(figsize=(10, 6))
    stats_df.plot(kind='bar', figsize=(10, 6))
    plt.title(f'Dataset {i} - Mean, Median, and Percentiles')
    plt.ylabel('Values')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```

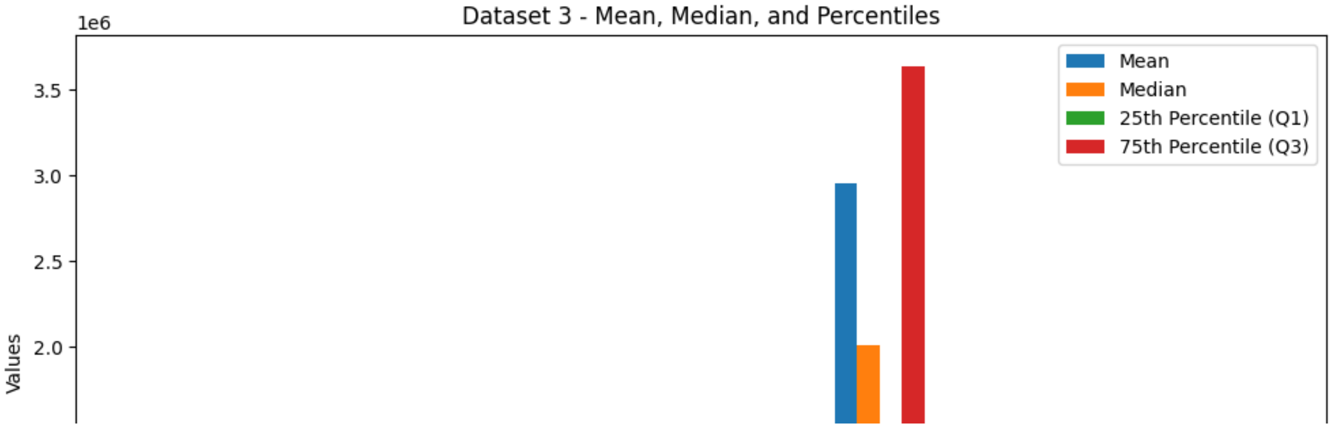
```
<Figure size 1000x600 with 0 Axes>
```



```
<Figure size 1000x600 with 0 Axes>
```




```
<Figure size 1000x600 with 0 Axes>
```



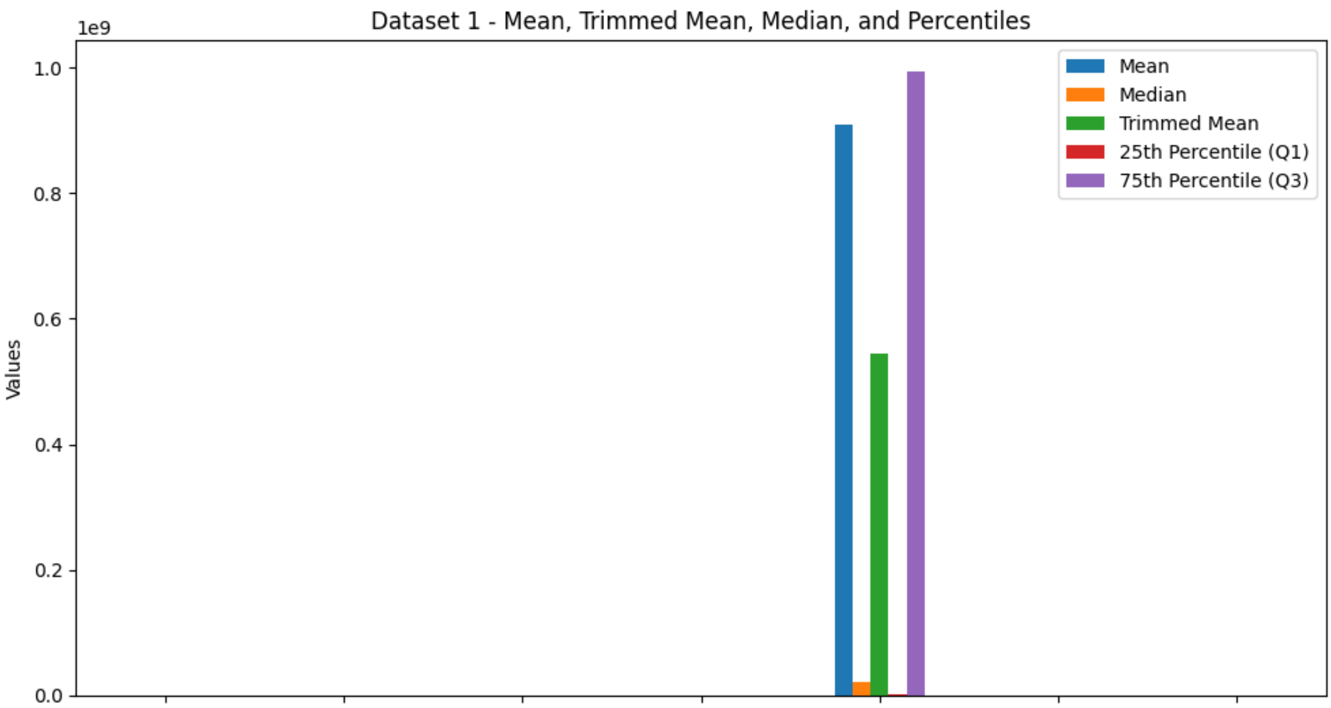
Step 10: Calculate Trimmed Mean

Purpose: The trimmed mean is calculated by excluding a certain percentage of extreme values from both ends of the data. This is useful when we want a measure of central tendency that is not affected by outliers. The trimmed mean provides a more robust estimate of the true center of the data.

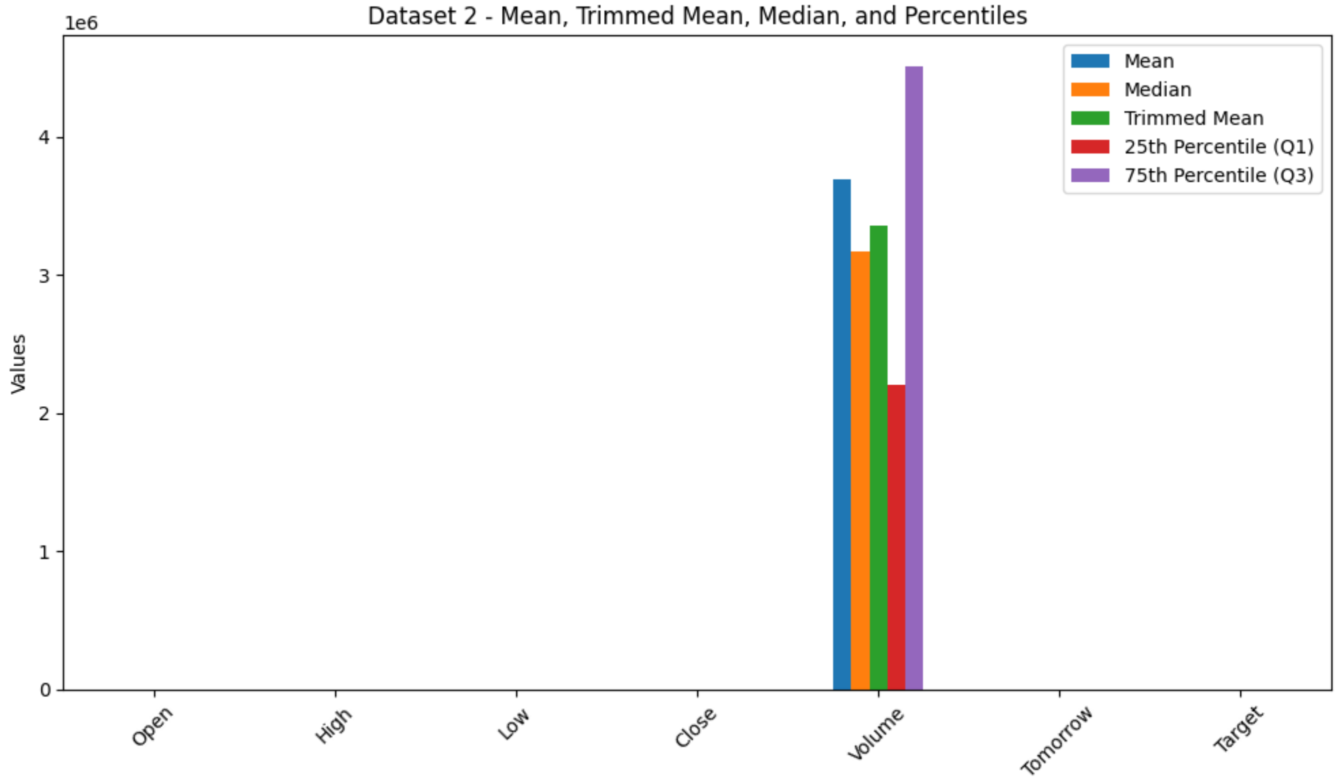
```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
from scipy.stats import trim_mean

# Trimming fraction (10%)
trim_fraction = 0.1
```

```
# Iterate over datasets to calculate trimmed mean and other statistics, and plo
for i, df in enumerate([dataset_1, dataset_2, dataset_3, dataset_4,dataset_5],
    # Calculate statistics
    mean = df.mean()
    median = df.median()
    percentile_25 = df.quantile(0.25)
    percentile_75 = df.quantile(0.75)

    # Calculate trimmed mean
    trimmed_mean = trim_mean(df, proportiontocut=trim_fraction)

    # Create a DataFrame for the statistics to plot
    stats_df = pd.DataFrame({
        'Mean': mean,
        'Median': median,
        'Trimmed Mean': trimmed_mean,
        '25th Percentile (Q1)': percentile_25,
        '75th Percentile (Q3)': percentile_75
    })

    # Plot the statistics for the current dataset
    plt.figure(figsize=(10, 6))
    stats_df.plot(kind='bar', figsize=(10, 6))
    plt.title(f'Dataset {i} - Mean, Trimmed Mean, Median, and Percentiles')
    plt.ylabel('Values')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```

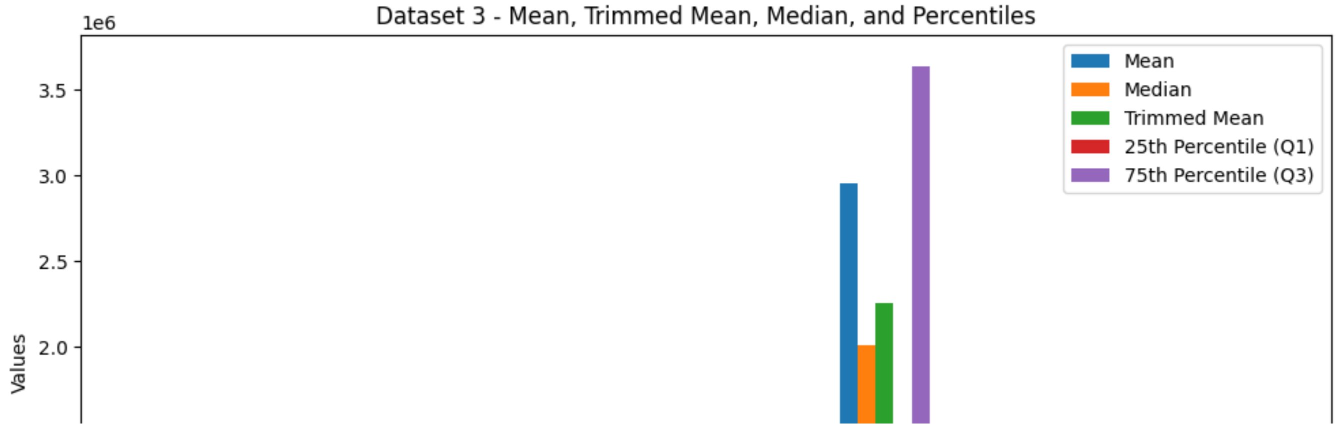
```
<Figure size 1000x600 with 0 Axes>
```



<Figure size 1000x600 with 0 Axes>



<Figure size 1000x600 with 0 Axes>

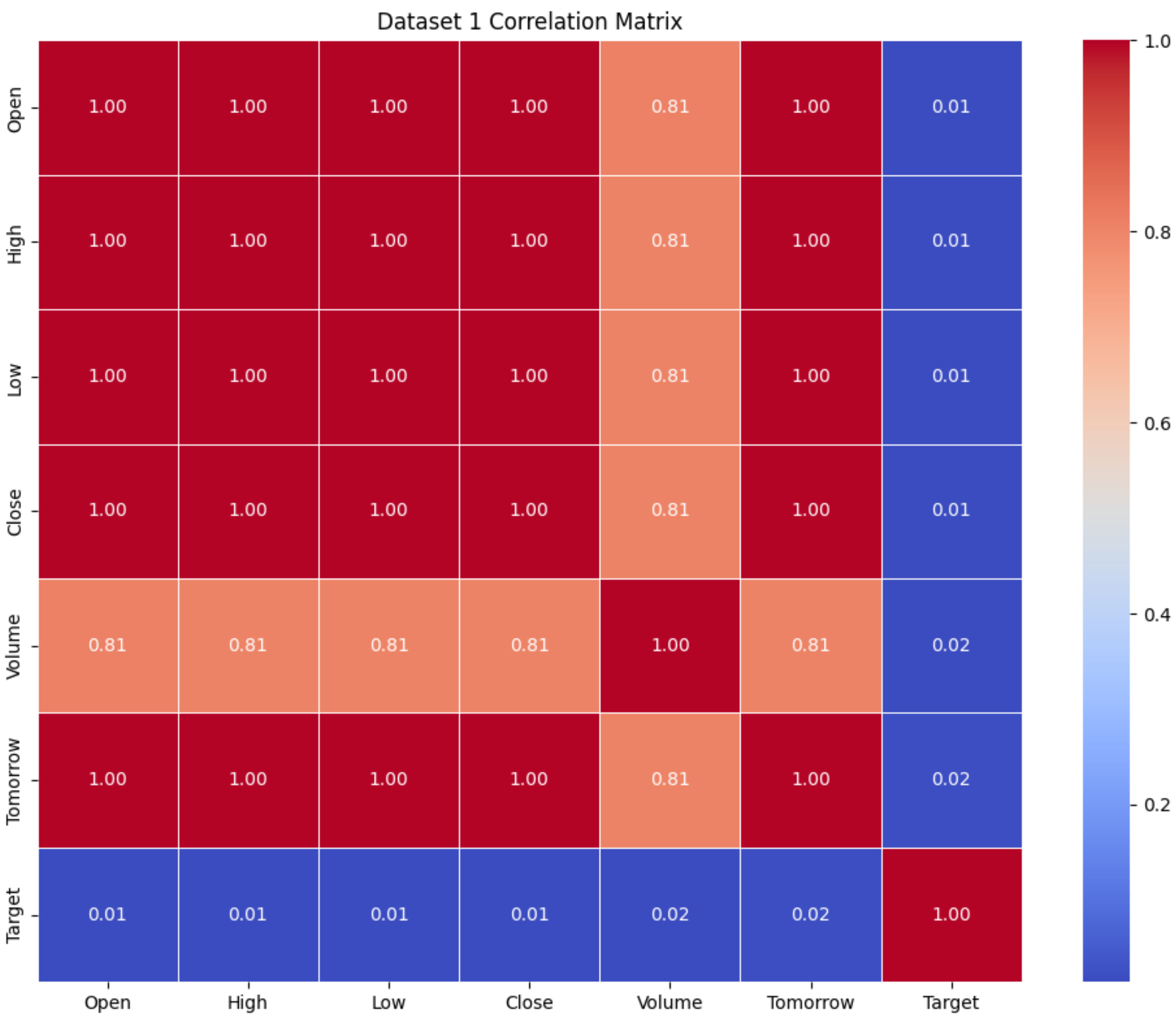


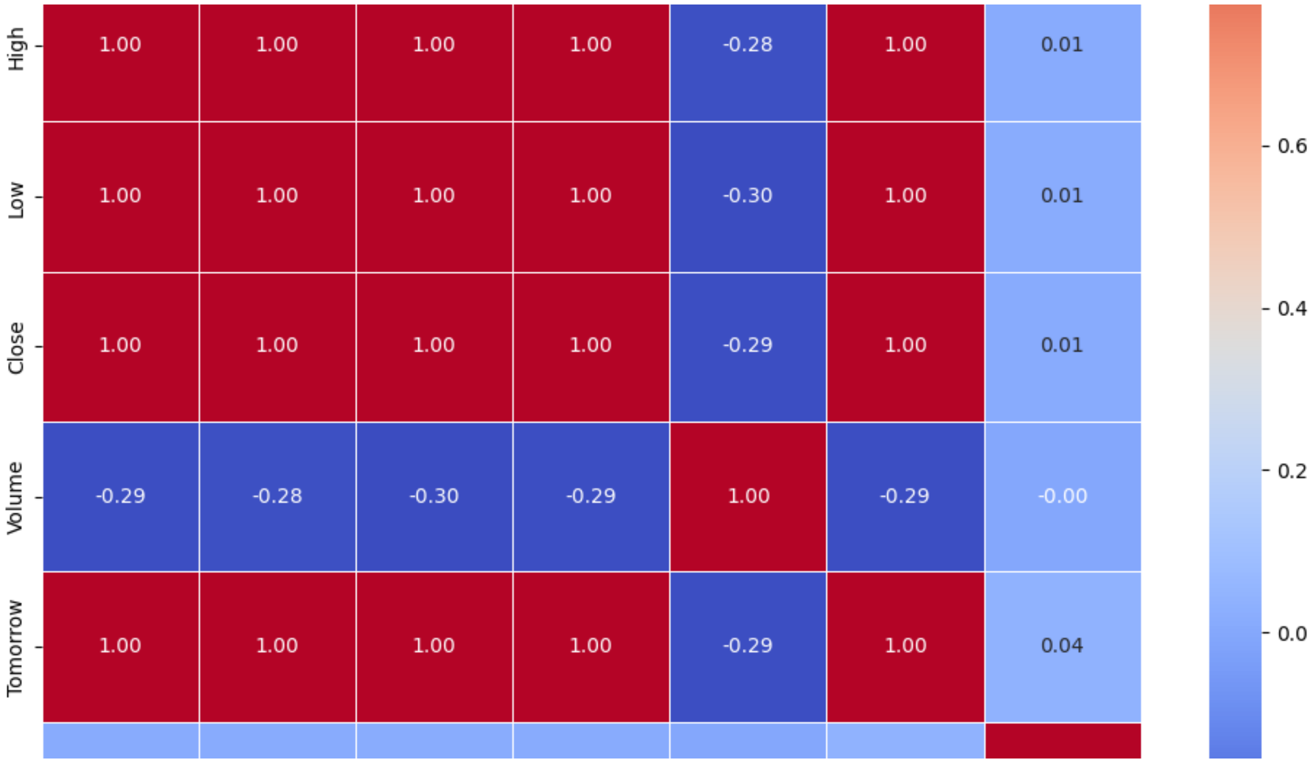
Step 11: Correlation Analysis

Purpose: In this step, we perform correlation analysis to identify relationships between variables. A correlation matrix is created to show how strongly different variables are related to each other. This helps in understanding which features are most important and which ones are redundant.

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

# Create a correlation matrix heatmap for each dataset
```

```
for i, df in enumerate([dataset_1, dataset_2, dataset_3, dataset_4,dataset_5],
    plt.figure(figsize=(10, 8))

    # Generate a correlation matrix
    corr_matrix = df.corr()

    # Plot heatmap
    sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths
    plt.title(f'Dataset {i} Correlation Matrix')

    # Show the heatmap
    plt.tight_layout()
    plt.show()
```





SUMMARY STATISTICS FOR EACH COLUMN

```
#indivisual cols statistics
# Summary statistics for specific columns
specific_columns_stats1 = dataset_1[['Open', 'High', 'Low', 'Close', 'Volume']]
specific_columns_stats2 = dataset_2[['Open', 'High', 'Low', 'Close', 'Volume']]
specific_columns_stats3 = dataset_3[['Open', 'High', 'Low', 'Close', 'Volume']]
specific_columns_stats4 = dataset_4[['Open', 'High', 'Low', 'Close', 'Volume']]
specific_columns_stats5 = dataset_5[['Open', 'High', 'Low', 'Close', 'Volume']]
specific_columns_stats1
specific_columns_stats2
specific_columns_stats3
specific_columns_stats4
specific_columns_stats5
```

	Open	High	Low	Close	Volume
count	235192.000000	235192.000000	235192.000000	235192.000000	2.351920e+05
mean	1267.759708	1286.581440	1247.488465	1266.554351	3.045903e+06
std	2585.259609	2619.649216	2546.621396	2582.140942	7.333981e+06
min	8.500000	9.750000	8.500000	9.150000	3.000000e+00
25%	275.000000	279.500000	269.600000	274.350000	2.190095e+05
50%	567.025000	576.900000	556.500000	566.700000	1.010938e+06
75%	1243.312500	1263.000000	1221.650000	1242.400000	3.019851e+06
max	33399.950000	33480.000000	32468.100000	32861.950000	4.810589e+08

SKEWNESS AND KURTOSIS

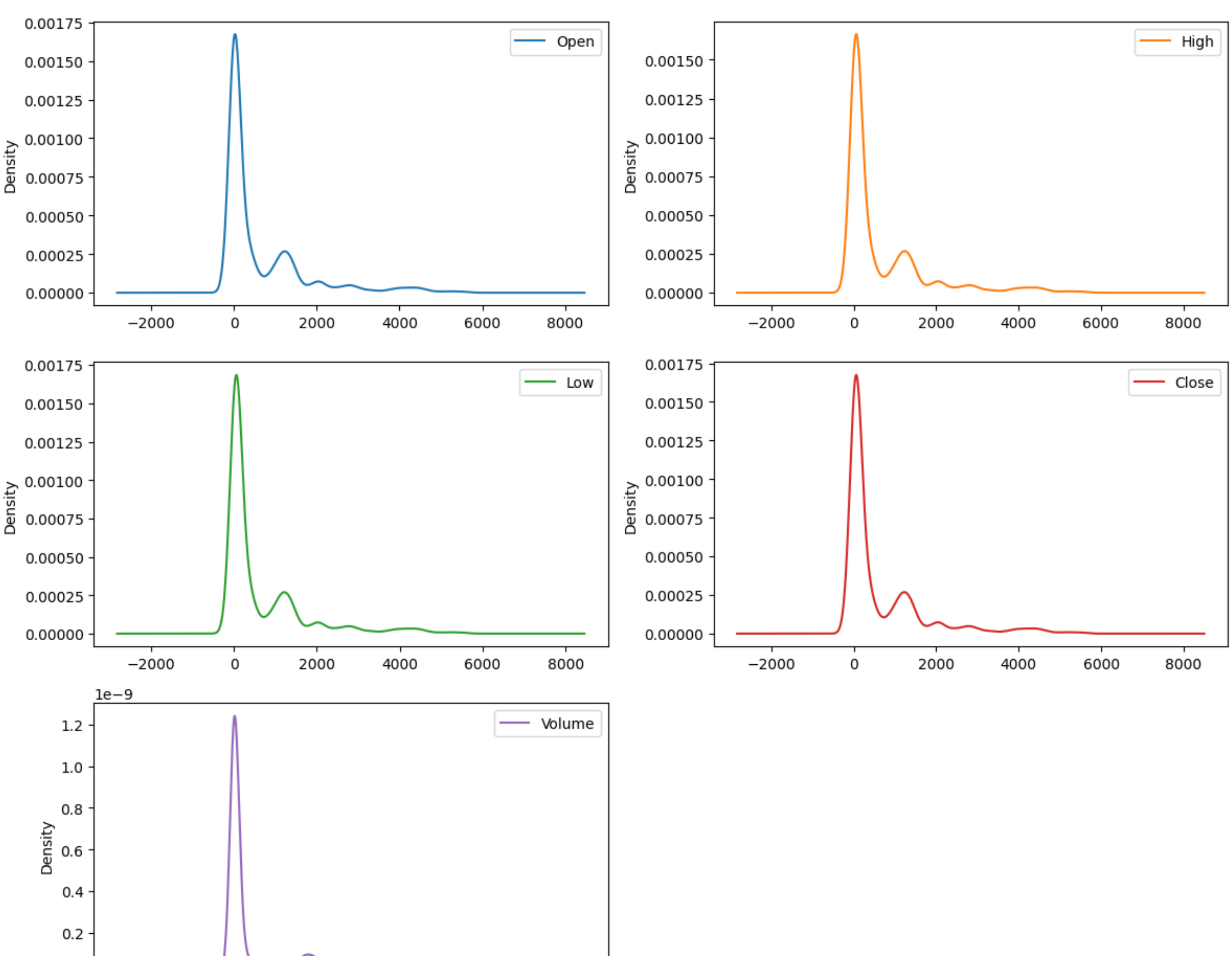

```
#skewness and kurtosis
# Skewness
skewness1 = dataset_1[['Open', 'High', 'Low', 'Close', 'Volume']].skew()
skewness2 = dataset_2[['Open', 'High', 'Low', 'Close', 'Volume']].skew()
skewness3 = dataset_3[['Open', 'High', 'Low', 'Close', 'Volume']].skew()
skewness4 = dataset_4[['Open', 'High', 'Low', 'Close', 'Volume']].skew()
skewness5 = dataset_5[['Open', 'High', 'Low', 'Close', 'Volume']].skew()
print("Skewness_1:\n", skewness1)
print("Skewness_2:\n", skewness2)
print("Skewness_3:\n", skewness3)
print("Skewness_4:\n", skewness4)
print("Skewness_5:\n", skewness5)

# Kurtosis
```

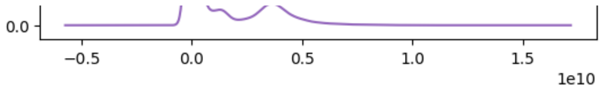
```
kurtosis1 = dataset_1[['Open', 'High', 'Low', 'Close', 'Volume']].kurt()
kurtosis2 = dataset_2[['Open', 'High', 'Low', 'Close', 'Volume']].kurt()
kurtosis3 = dataset_3[['Open', 'High', 'Low', 'Close', 'Volume']].kurt()
kurtosis4 = dataset_4[['Open', 'High', 'Low', 'Close', 'Volume']].kurt()
kurtosis5 = dataset_5[['Open', 'High', 'Low', 'Close', 'Volume']].kurt()
print("Kurtosis_1:\n", kurtosis1)
print("Kurtosis_2:\n", kurtosis2)
print("Kurtosis_3:\n", kurtosis3)
print("Kurtosis_4:\n", kurtosis4)
print("Kurtosis_5:\n", kurtosis5)
```

```
Skewness_1:
 Open      2.257427
High      2.287889
Low       2.291378
Close     2.289435
Volume    1.831796
dtype: float64
Skewness_2:
 Open      1.079005
High      1.094200
Low       1.067634
Close     1.082845
Volume    5.488032
dtype: float64
Skewness_3:
 Open      1.283868
High      1.309294
Low       1.255761
Close     1.274576
Volume    7.600000
dtype: float64
Skewness_4:
 Open      1.011280
High      1.024458
Low       0.984558
Close     1.002786
Volume    4.219159
dtype: float64
Skewness_5:
 Open       6.280255
High       6.270576
Low        6.284039
Close      6.276493
Volume    12.461644
dtype: float64
Kurtosis_1:
 Open      5.065715
High      5.211980
Low       5.232640
Close     5.221190
Volume    2.576180
```

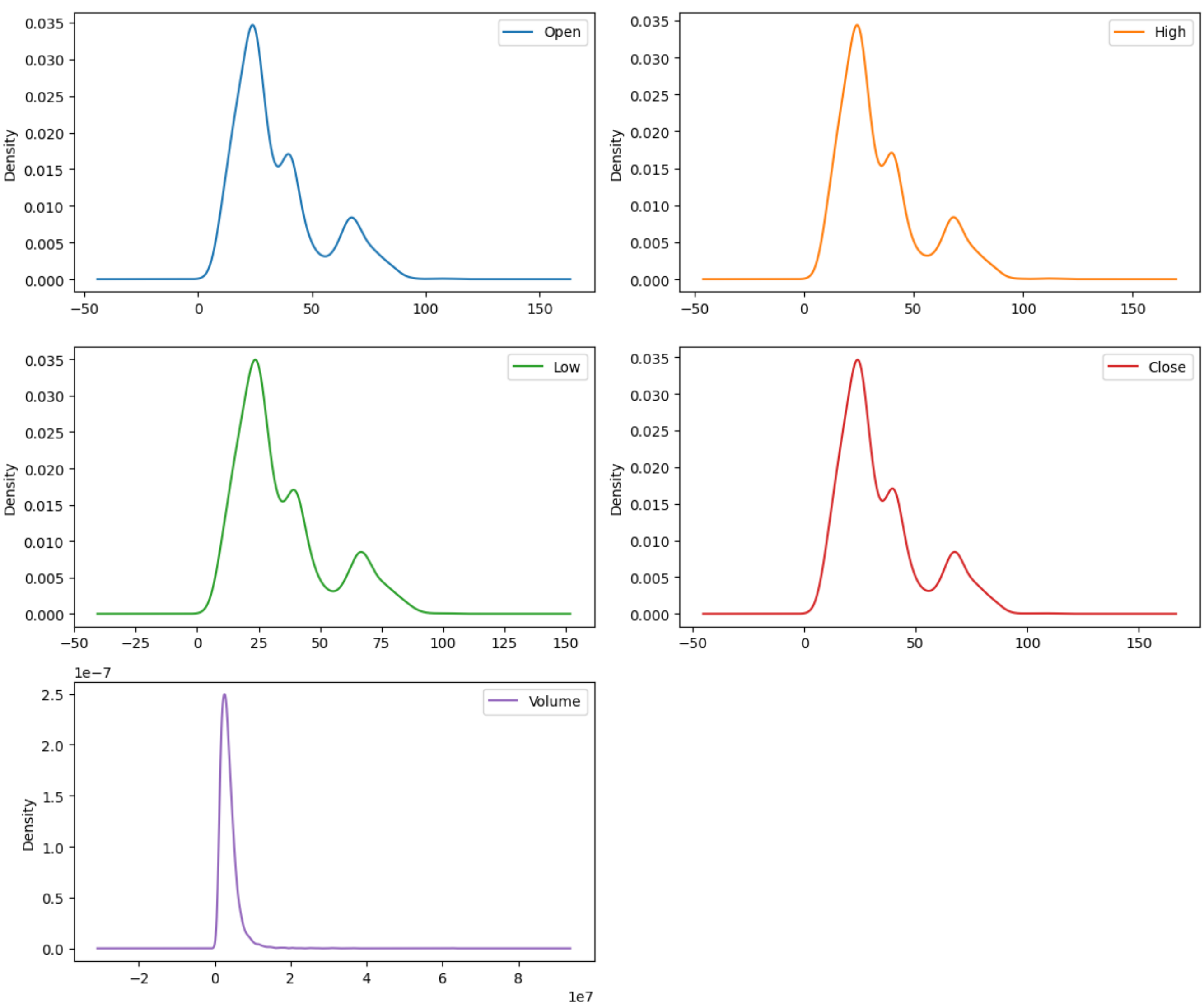
```
dtype: float64
Kurtosis_2:
 Open        0.352871
High        0.423413
Low         0.300561
Close       0.371435
Volume     81.463251
dtype: float64
Kurtosis_3:
 Open         2.171044
High         2.238984
Low          2.056181
Close        2.114807
Volume     117.867919
dtype: float64
Kurtosis_4:
 Open         0.868539
High        0.893625
```

```
# Density plots for numerical columns
dataset_1[['Open', 'High', 'Low', 'Close', 'Volume']].plot(kind='density', subp
plt.tight_layout()
plt.show()
```

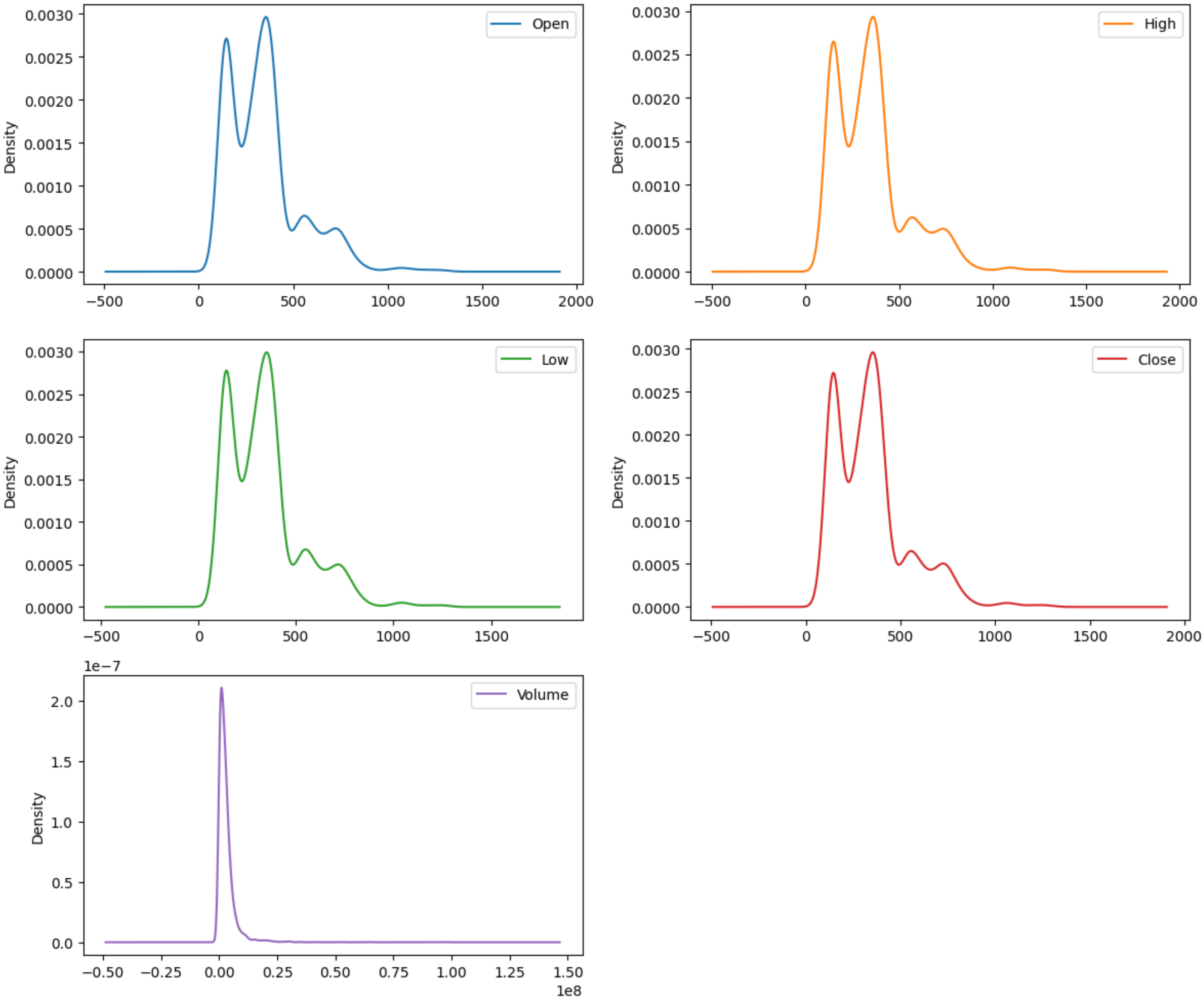



0.0
−0.5
0.0
0.5
1.0
1.5
1e10

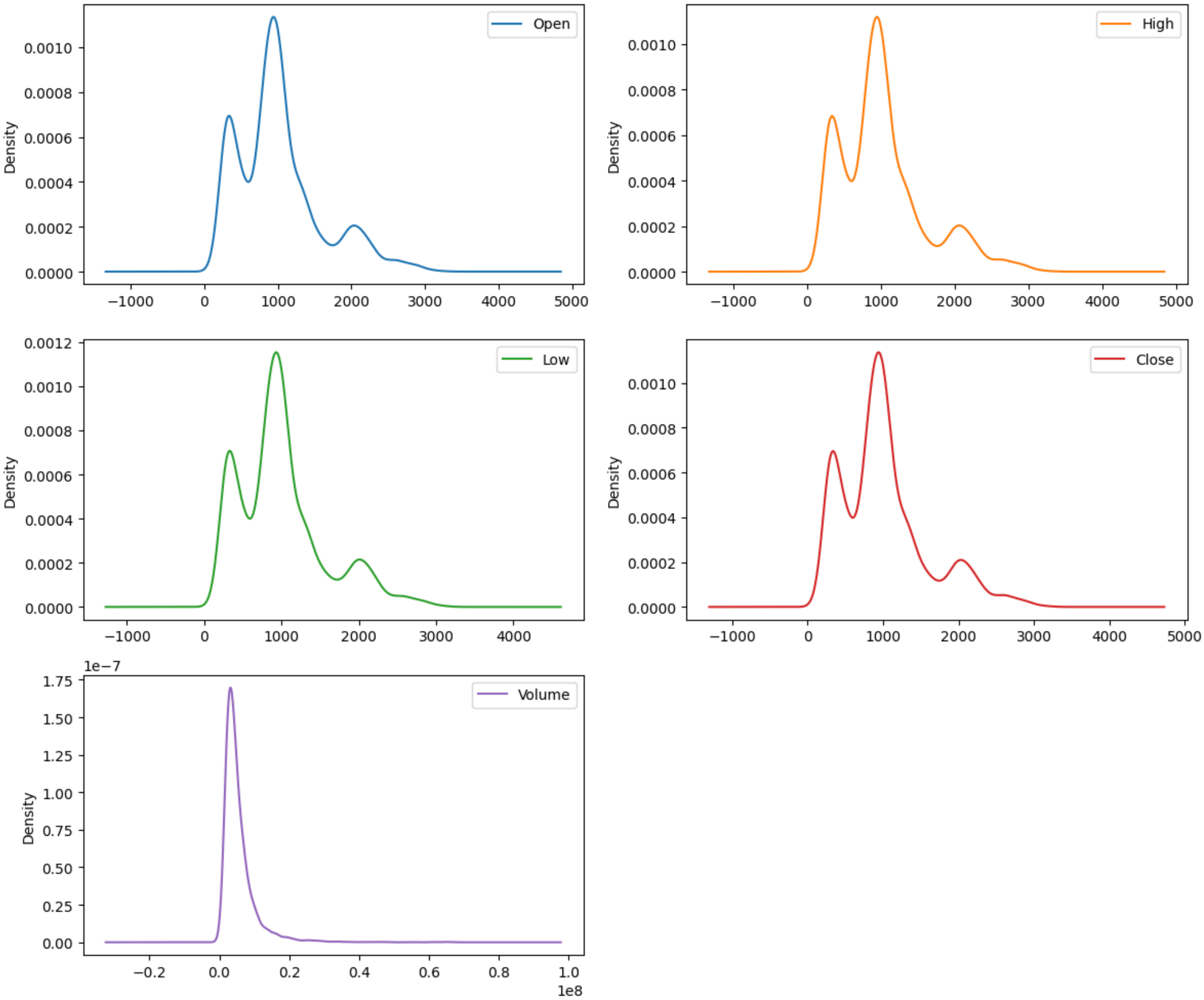
```
dataset_2[['Open', 'High', 'Low', 'Close', 'Volume']].plot(kind='density', subp
plt.tight_layout()
plt.show()
```



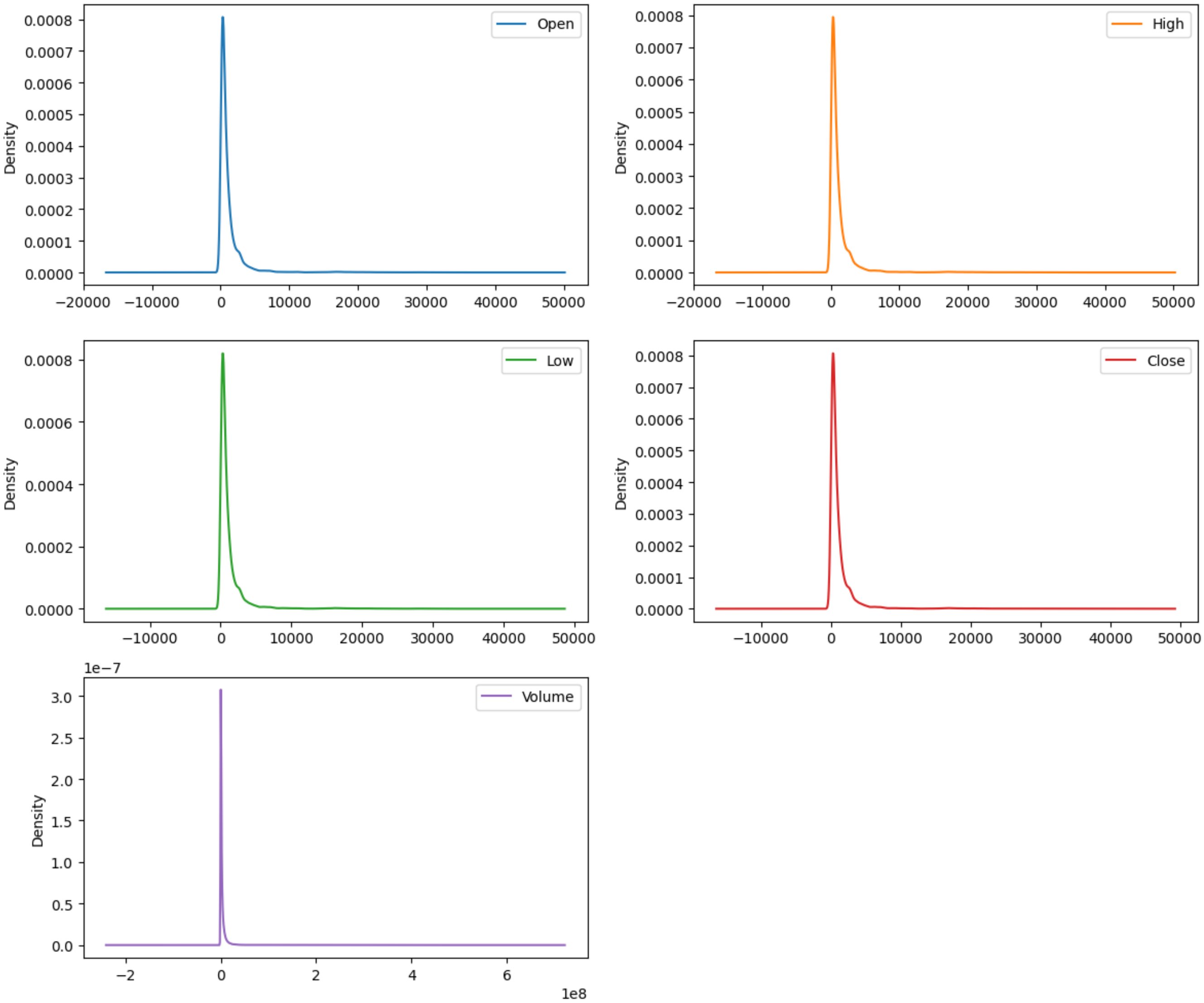
```
dataset_3[['Open', 'High', 'Low', 'Close', 'Volume']].plot(kind='density', subp
plt.tight_layout()
plt.show()
```



```
dataset_4[['Open', 'High', 'Low', 'Close', 'Volume']].plot(kind='density', subp
plt.tight_layout()
plt.show()
```



```
dataset_5[['Open', 'High', 'Low', 'Close', 'Volume']].plot(kind='density', subp
plt.tight_layout()
plt.show()
```



TRIMMED STANDARD DEVIATION

```
import numpy as np

# Function to calculate trimmed standard deviation
def trimmed_std(data, trim_percent):
    trim_count = int(trim_percent * len(data))
    sorted_data = np.sort(data)
    trimmed_data = sorted_data[trim_count:-trim_count]  # Trim from both ends
    return np.std(trimmed_data, ddof=1)  # Use ddof=1 for sample standard devia

# Calculate the trimmed standard deviation (trimming 10% from both ends)
trimmed_std_open = trimmed_std(dataset_1['Open'], 0.1)
trimmed_std_high = trimmed_std(dataset_1['High'], 0.1)
trimmed_std_low = trimmed_std(dataset_1['Low'], 0.1)
trimmed_std_close = trimmed_std(dataset_1['Close'], 0.1)
trimmed_std_volume = trimmed_std(dataset_1['Volume'], 0.1)

print("Trimmed Standard Deviation for dataset_1 (10% trim):")
print(f"Open: {trimmed_std_open}")
print(f"High: {trimmed_std_high}")
print(f"Low: {trimmed_std_low}")
print(f"Close: {trimmed_std_close}")
print(f"Volume: {trimmed_std_volume}")
```

```
Trimmed Standard Deviation for dataset_1 (10% trim):
Open: 523.5435746788896
High: 511.85517234029174
Low: 505.1650524280973
Close: 508.73703657430013
Volume: 1038315741.8393118
```

```
# Calculate the trimmed standard deviation (trimming 10% from both ends)
trimmed_std_open = trimmed_std(dataset_2['Open'], 0.1)
trimmed_std_high = trimmed_std(dataset_2['High'], 0.1)
trimmed_std_low = trimmed_std(dataset_2['Low'], 0.1)
trimmed_std_close = trimmed_std(dataset_2['Close'], 0.1)
trimmed_std_volume = trimmed_std(dataset_2['Volume'], 0.1)

print("Trimmed Standard Deviation for dataset_2 (10% trim):")
print(f"Open: {trimmed_std_open}")
print(f"High: {trimmed_std_high}")
print(f"Low: {trimmed_std_low}")
print(f"Close: {trimmed_std_close}")
print(f"Volume: {trimmed_std_volume}")
```

```
Trimmed Standard Deviation for dataset_2 (10% trim):
Open: 12.57932086975378
High: 12.715704897456499
Low: 12.425579003320584
Close: 12.56878590062317
Volume: 1153412.5335452817
```

```
# Calculate the trimmed standard deviation (trimming 10% from both ends)
trimmed_std_open = trimmed_std(dataset_3['Open'], 0.1)
trimmed_std_high = trimmed_std(dataset_3['High'], 0.1)
trimmed_std_low = trimmed_std(dataset_3['Low'], 0.1)
trimmed_std_close = trimmed_std(dataset_3['Close'], 0.1)
trimmed_std_volume = trimmed_std(dataset_3['Volume'], 0.1)

print("Trimmed Standard Deviation for dataset_3 (10% trim):")
print(f"Open: {trimmed_std_open}")
print(f"High: {trimmed_std_high}")
print(f"Low: {trimmed_std_low}")
print(f"Close: {trimmed_std_close}")
print(f"Volume: {trimmed_std_volume}")
```

```
Trimmed Standard Deviation for dataset_3 (10% trim):
Open: 122.04048060458287
High: 124.78167702311183
Low: 119.2993734896381
Close: 121.73123864903987
Volume: 1460565.165079545
```

```python
# Calculate the trimmed standard deviation (trimming 10% from both ends)
trimmed_std_open = trimmed_std(dataset_4['Open'], 0.1)
trimmed_std_high = trimmed_std(dataset_4['High'], 0.1)
trimmed_std_low = trimmed_std(dataset_4['Low'], 0.1)
trimmed_std_close = trimmed_std(dataset_4['Close'], 0.1)
trimmed_std_volume = trimmed_std(dataset_4['Volume'], 0.1)

print("Trimmed Standard Deviation for dataset_4 (10% trim):")
print(f"Open: {trimmed_std_open}")
print(f"High: {trimmed_std_high}")
print(f"Low: {trimmed_std_low}")
print(f"Close: {trimmed_std_close}")
print(f"Volume: {trimmed_std_volume}")
```

```
Trimmed Standard Deviation for dataset_4 (10% trim):
Open: 359.83856986250066
High: 364.7563318584844
Low: 354.5117262094316
Close: 359.8150954437327
Volume: 2000694.348557587
```

```python
# Calculate the trimmed standard deviation (trimming 10% from both ends)
trimmed_std_open = trimmed_std(dataset_5['Open'], 0.1)
trimmed_std_high = trimmed_std(dataset_5['High'], 0.1)
trimmed_std_low = trimmed_std(dataset_5['Low'], 0.1)
trimmed_std_close = trimmed_std(dataset_5['Close'], 0.1)
trimmed_std_volume = trimmed_std(dataset_5['Volume'], 0.1)

print("Trimmed Standard Deviation for dataset_5 (10% trim):")
print(f"Open: {trimmed_std_open}")
print(f"High: {trimmed_std_high}")
print(f"Low: {trimmed_std_low}")
print(f"Close: {trimmed_std_close}")
print(f"Volume: {trimmed_std_volume}")
```

```
Trimmed Standard Deviation for dataset_5 (10% trim):
Open: 584.3713996429924
High: 592.5630702302993
Low: 575.7092924636017
Close: 584.0093265605535
Volume: 1725081.4874577571
```