CaseStudy 02 : Google Stock Data Analysis

NOTE

This notebook contains the practical implementations of the analysis planned in the article.

Here is Article 121 - Time Series Analysis on Google Stock Data

Importing Relevant Libraries

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

⚠ Data Warning

For the code ahead we are going to use the Google Stock Data which is in the Data folder or you can download it from the Kaggle.
Kaggle Data : Download

```
google_stock_df = pd.read_csv(
    filepath_or_buffer = '../Data/GoogleStockData2004To2024.csv'
)
```

Data Preparation

1. Converting `dtype` of Date from `object` to `np.datetime64`

```
google_stock_df.Date.dtype
```

```
dtype('O')
```

```
google_stock_df.Date = pd.to_datetime(
    arg = google_stock_df.Date
)

google_stock_df.Date.dtype
```

```
dtype('<M8[ns]')
```

2. Setting `Date` as Index of DataFrame

```
google_stock_df.set_index(
    keys = 'Date',
    inplace = True
)
```

Now, as we have set the `Date` column as Index, we can access data for any date by passing the Date using *loc*.

```
google_stock_df.loc['2024-06-04']
```

```
Open          1.732800e+02
High          1.738500e+02
Low           1.718900e+02
Close         1.737900e+02
Adj Close     1.731623e+02
Volume        2.687960e+07
Name: 2024-06-04 00:00:00, dtype: float64
```

Similarly, now we can use slicing to access data for a period.

```
google_stock_df.loc['2022-01-01' : '2022-01-31']
```

```
                  Open        High         Low       Close   Adj Close
\
Date

2022-01-03  145.054993  145.850998  143.712997  144.991501  144.467789

2022-01-04  145.395996  146.485001  143.716507  144.399506  143.877945

2022-01-05  144.419998  144.499496  137.688004  137.774994  137.277359

2022-01-06  136.998505  139.940002  136.558502  137.747498  137.249954

2022-01-07  138.145493  138.448502  135.766495  137.016998  136.522110

2022-01-10  135.078003  138.819504  133.164505  138.669495  138.168625

2022-01-11  138.007004  140.216003  136.692505  139.735992  139.231277

2022-01-12  141.149994  142.608002  140.694504  141.430496  140.919647

2022-01-13  141.539993  142.850006  138.408997  138.587006  138.086441

2022-01-14  137.078995  140.742004  136.998505  139.480499  138.976700

2022-01-18  136.175003  137.131500  135.438507  135.998001  135.506775

2022-01-19  136.523499  137.959503  135.015503  135.116501  134.628448

2022-01-20  136.250000  137.626007  132.964493  133.307495  132.825989

2022-01-21  132.593506  134.865494  130.086502  130.351501  129.880676

2022-01-24  125.977997  131.203003  124.500000  130.804001  130.331543
```

```
2022-01-25  128.740005  129.399002  126.500504  126.934998  126.476509

2022-01-26  131.119003  133.000000  127.141998  129.233002  128.766205

2022-01-27  131.304001  132.652496  128.932495  129.005005  128.539047

2022-01-28  129.658997  133.356506  128.485001  133.350998  132.869339

2022-01-31  134.162003  135.473007  132.209503  135.303497  134.814774
```

```
               Volume
Date
2022-01-03  28646000
2022-01-04  28400000
2022-01-05  54618000
2022-01-06  37348000
2022-01-07  29760000
2022-01-10  44408000
2022-01-11  28730000
2022-01-12  26108000
2022-01-13  31436000
2022-01-14  29662000
2022-01-18  34872000
2022-01-19  28648000
2022-01-20  29908000
2022-01-21  55652000
2022-01-24  76622000
2022-01-25  46960000
2022-01-26  49130000
2022-01-27  31950000
2022-01-28  34362000
2022-01-31  39986000
```

Similarly, we can now do partial indexing as we have converted the date into a DatetimeIndex object of Pandas.
Partial Indexing : Partial Indexing means passing partial dates to access data.
Let's see the examples.

 Example 01 : All records of the year 2022.

```
google_stock_df.loc['2022']
```

```
                  Open        High         Low       Close   Adj Close
\
Date

2022-01-03  145.054993  145.850998  143.712997  144.991501  144.467789

2022-01-04  145.395996  146.485001  143.716507  144.399506  143.877945
```

```
2022-01-05  144.419998  144.499496  137.688004  137.774994  137.277359
2022-01-06  136.998505  139.940002  136.558502  137.747498  137.249954
2022-01-07  138.145493  138.448502  135.766495  137.016998  136.522110
...                ...         ...         ...         ...         ...
2022-12-23   87.110001   89.550003   87.070000   89.230003   88.907707
2022-12-27   88.800003   88.940002   87.010002   87.389999   87.074348
2022-12-28   86.980003   88.040001   85.940002   86.019997   85.709297
2022-12-29   86.620003   88.849998   86.610001   88.449997   88.130516
2022-12-30   86.980003   88.300003   86.570000   88.230003   87.911316
```

```
               Volume
Date
2022-01-03  28646000
2022-01-04  28400000
2022-01-05  54618000
2022-01-06  37348000
2022-01-07  29760000
...              ...
2022-12-23  23003000
2022-12-27  20097300
2022-12-28  19523200
2022-12-29  23333500
2022-12-30  23986300

[251 rows x 6 columns]
```

3. Creating Month Name, Day Name, and Quarter columns

```
google_stock_df['month_name'] = google_stock_df.index.month_name()

google_stock_df['weekday_name'] = google_stock_df.index.day_name()

google_stock_df['Quarter'] = google_stock_df.index.quarter
```

4. Creating the Daily Stock Return Column

$$R_t = \frac{P_t - P_{t-1}}{P_{t-1}}$$

```
google_stock_df['DailyReturns'] = (google_stock_df.Close - \
                                   google_stock_df.Close.shift(1)) / \
                                   google_stock_df.Close.shift(1)
```

5. Creating the Logarithmic Return Column

$$r_t = \ln\left(\frac{P_t}{P_{t-1}}\right) = \ln\left(P_t\right) - \ln\left(P_{t-1}\right)$$

```
google_stock_df['LogReturns'] = np.log(google_stock_df.Close) -
np.log(google_stock_df.Close.shift(1))
```

6. Creating Percentage Change Column

$$\%\,\text{Change}_t = \left(\frac{P_t - P_{t-1}}{P_{t-1}}\right) \times 100$$

```
google_stock_df['PercentageChanges'] = ((google_stock_df.Close - \

google_stock_df.Close.shift(1)) / \

google_stock_df.Close.shift(1)) * 100
```

7. Creating the Intraday Return Column

```
google_stock_df['IntraDayReturn'] = (google_stock_df['Close'] -
google_stock_df['Open']) / google_stock_df['Open']
```

8. Performing Data Vaidation Check

As we know that if the data is accurate, then it will follow the following conditions.

$$ High \geq Close \geq Low, \\ High \geq Open \geq Low, \\ Low \leq High $$

Lets Validate our data for these conditions.

```
((google_stock_df['High'] >= google_stock_df['Close']) & \
(google_stock_df['Close'] >= google_stock_df['Low'])).sum() -
google_stock_df.shape[0]

0

((google_stock_df['High'] >= google_stock_df['Open']) & \
(google_stock_df['Open'] >= google_stock_df['Low'])).sum() -
google_stock_df.shape[0]

0
```

```
(google_stock_df['Low'] <= google_stock_df['High']).sum() -
google_stock_df.shape[0]
```

```
0
```

Conclusion : The given Stock Data does not contain any ambiguity. All the Validation Check is passed.

Data Analysis

Let's try to fetch data of stock for a particular date for every year.
For Example : On the Indian Independence Day.

```
google_stock_df[
    google_stock_df.index.isin(
        values = pd.date_range(
            start = '2005-08-15',
            end = '2024-08-15',
            freq = pd.DateOffset(
                years = 1
            )
        )
    )
]
```

Date	Open	High	Low	Close	Adj Close
2005-08-15	7.252252	7.326577	7.101351	7.107107	7.081437
2006-08-15	9.362112	9.551301	9.324324	9.533784	9.499349
2007-08-15	12.737738	12.805055	12.430180	12.451201	12.406228
2008-08-15	12.687437	12.779279	12.650150	12.766517	12.720404
2011-08-15	13.849600	14.138889	13.664915	13.944695	13.894327
2012-08-15	16.773773	16.873123	16.619120	16.705206	16.644867
2013-08-15	21.632633	21.644646	21.471472	21.513014	21.435308
2014-08-15	29.375000	29.473499	29.038000	29.185499	29.080082
2016-08-15	40.360500	40.568001	40.201500	40.298000	40.152447
2017-08-15	47.051498	47.153500	46.832001	46.903999	46.734581
2018-08-15	62.214001	62.598999	61.312000	61.611000	61.388462

2019-08-15	58.421501	58.820999	58.150002	58.466000	58.254822
2022-08-15	121.129997	122.300003	120.610001	122.080002	121.639053
2023-08-15	131.100006	131.419998	129.279999	129.779999	129.311234
2024-08-15	160.500000	161.639999	159.610001	161.300003	160.901855

Date	Volume	month_name	weekday_name	Quarter	DailyReturns
2005-08-15	326661012	August	Monday	3	-0.019743
2006-08-15	267660072	August	Tuesday	3	0.031237
2007-08-15	216163620	August	Wednesday	3	-0.021726
2008-08-15	141686172	August	Friday	3	0.009219
2011-08-15	285510204	August	Monday	3	-0.011600
2012-08-15	96331572	August	Wednesday	3	-0.001675
2013-08-15	74693232	August	Thursday	3	-0.011669
2014-08-15	34468000	August	Friday	3	-0.001608
2016-08-15	18596000	August	Monday	3	-0.001351
2017-08-15	22130000	August	Tuesday	3	-0.000905
2018-08-15	37052000	August	Wednesday	3	-0.020602
2019-08-15	28506000	August	Thursday	3	0.004355
2022-08-15	19494800	August	Monday	3	0.003287
2023-08-15	19770700	August	Tuesday	3	-0.011802
2024-08-15	31524300	August	Thursday	3	0.005799

Date	LogReturns	PercentageChanges	IntraDayReturn
2005-08-15	-0.019941	-1.974314	-0.020014
2006-08-15	0.030759	3.123731	0.018337
2007-08-15	-0.021966	-2.172630	-0.022495
2008-08-15	0.009177	0.921875	0.006233
2011-08-15	-0.011668	-1.160042	0.006866

```
2012-08-15   -0.001676          -0.167491       -0.004088
2013-08-15   -0.011738          -1.166919       -0.005530
2014-08-15   -0.001609          -0.160783       -0.006451
2016-08-15   -0.001352          -0.135061       -0.001549
2017-08-15   -0.000906          -0.090528       -0.003135
2018-08-15   -0.020817          -2.060186       -0.009692
2019-08-15    0.004345           0.435470        0.000762
2022-08-15    0.003282           0.328732        0.007843
2023-08-15   -0.011873          -1.180235       -0.010069
2024-08-15    0.005782           0.579914        0.004984
```

Let's plot some visualizations to see the patterns in our stock data.

1. Open Column Visualization

```
google_stock_df['Open'].plot(
    kind = 'line'
)
plt.title('Opening Price')
plt.show()
```

2. Close Column Visualization

```
google_stock_df['Close'].plot(
    kind = 'line'
```

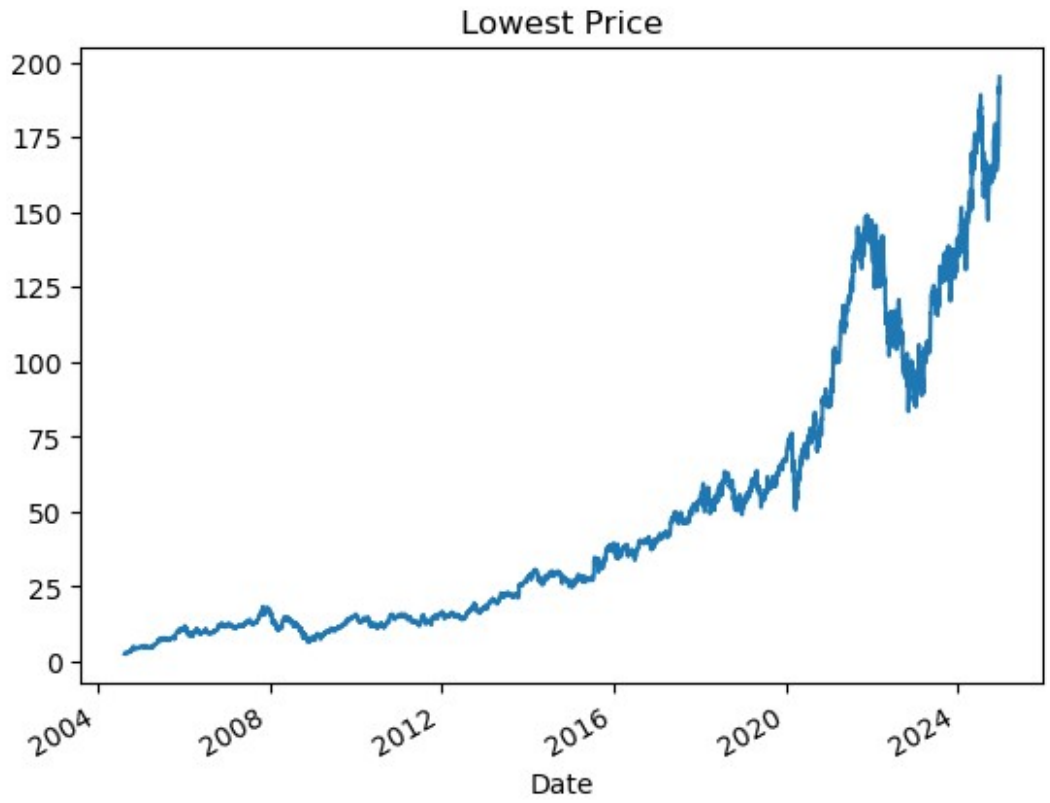
```
)
plt.title('Closing Price')
plt.show()
```

3. High Column Visualization

```
google_stock_df['High'].plot(
    kind = 'line'
)
plt.title('Highest Price')
plt.show()
```

4. Low Column Visualization

```
google_stock_df['Low'].plot(
    kind = 'line'
)
plt.title('Lowest Price')
plt.show()
```



5. Open, Close, Low & High Column Visualization

```
columns_to_plot = ['Open', 'Close', 'Low', 'High']
ax = google_stock_df[columns_to_plot].plot(
    subplots = True,
    layout = (2, 2),
    sharex = False,
    sharey = False,
    linewidth = 0.7,
    fontsize = 10,
    legend = False,
    figsize = (15, 10),
    title = ['Opening Price', 'Closing Price', 'Lowest Price',
'Highest Price']
)
```

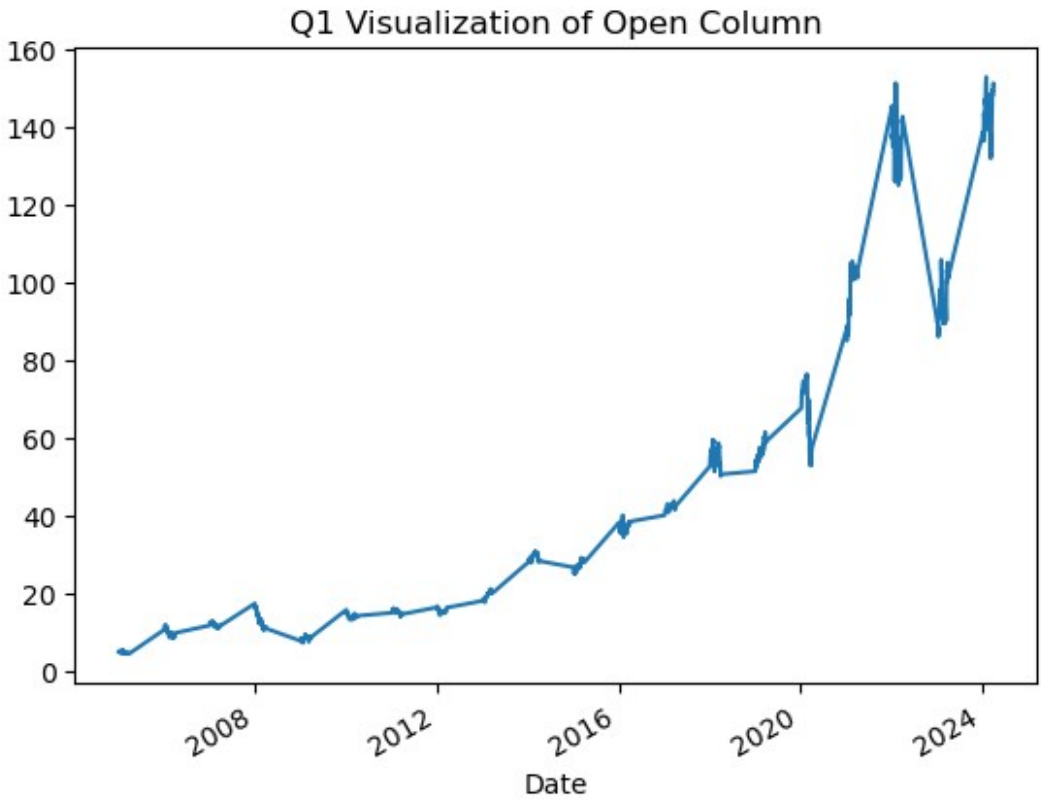
6. Dynamic Quarter Visualization of Any Column

Function : Defining a function to ease the process and a little bit dynamic.

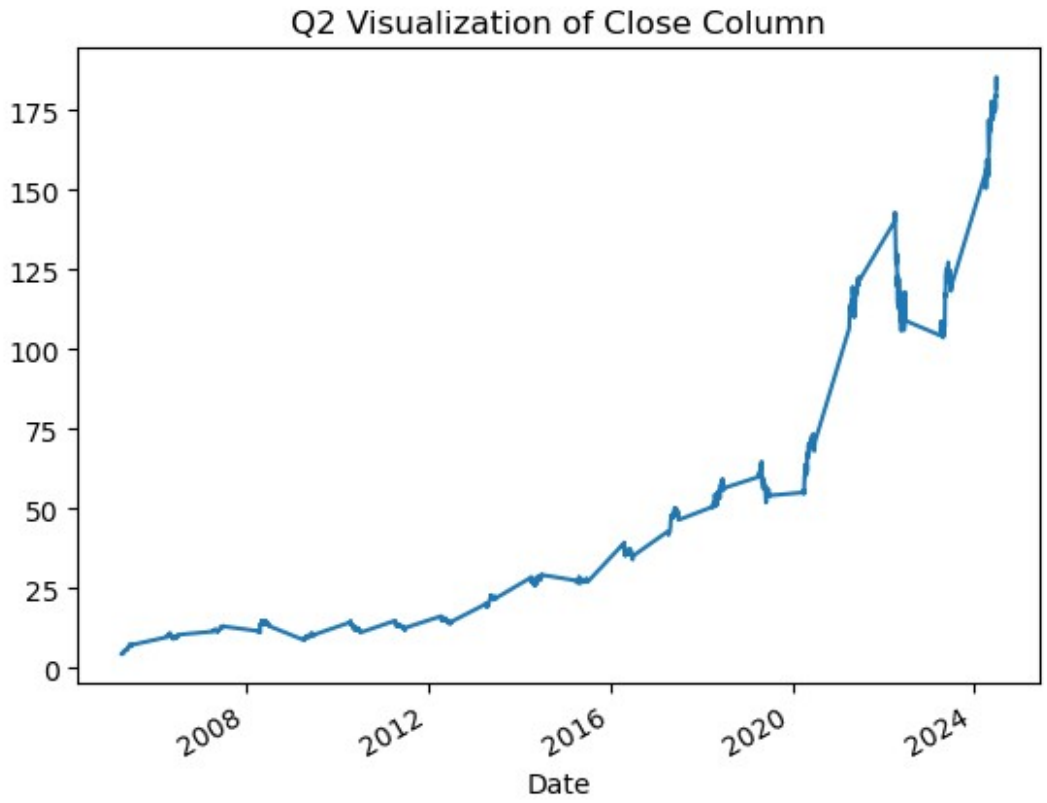
```
def visualize_quarter_values(
    q: int,
    column: str
):
    desired_df = google_stock_df[column][google_stock_df['Quarter'] ==
q]
    ax = desired_df.plot(
        kind = 'line',
        title = f'Q{q} Visualization of {column} Column'
    )
    plt.show()
    return
```

Sample Function Calls

```
visualize_quarter_values(
    q = 1,
    column = 'Open'
)
```




```
visualize_quarter_values(
    q = 2,
    column = 'Close'
)
```



7. Dynamic Visualization of Any Column for a Quarter of a particular Year.

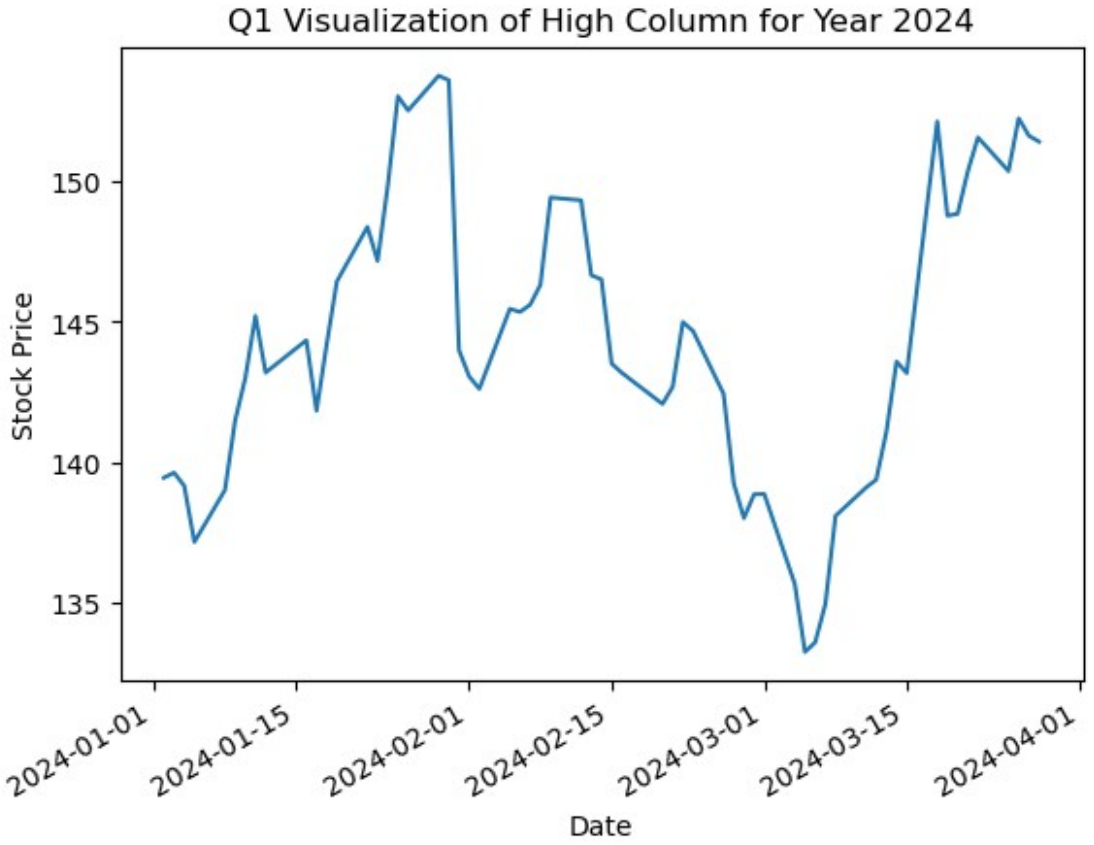
Function : Defining a function to ease the process and a little bit dynamic.

```
def visualize_year_quarter_values(
    q: int,
    year: int,
    column: str
):
    desired_df = google_stock_df[column].loc[str(year)]
[google_stock_df['Quarter'] == q]
    ax = desired_df.plot(
        kind = 'line',
        title = f'Q{q} Visualization of {column} Column for Year
{year}',
        ylabel = 'Stock Price'
    )
    plt.show()
    return
```

Sample Function Calls

```
visualize_year_quarter_values(
    q = 1,
```

```
    year = 2024,
    column = 'High'
)
```



8. Summary Statistics of Return Columns

```
def display_return_stats(
    df: pd.DataFrame
):
    '''
    Displays the Summary Statistics for all the Return Columns.
    '''

    return_columns = ['DailyReturns', 'LogReturns',
'PercentageChanges', 'IntraDayReturn']

    return_summary_df = pd.DataFrame(
        columns = return_columns
    )

    for col in return_columns:
        summary_stats_dict = {
            'Mean' : df[col].mean(),
```

```
            'Std. Dev' : df[col].std(),
            'Min' : df[col].min(),
            'Max' : df[col].max(),
            'Skewness' : df[col].skew(),
            'Kurtosis' : df[col].kurtosis(),
        }
        return_summary_df[col] = pd.Series(summary_stats_dict)

    return return_summary_df

display_return_stats(
    df = google_stock_df
)
```

	DailyReturns	LogReturns	PercentageChanges	IntraDayReturn
Mean	0.001036	0.000851	0.103576	0.000089
Std. Dev	0.019274	0.019176	1.927362	0.014925
Min	-0.116341	-0.123685	-11.634149	-0.091797
Max	0.199915	0.182251	19.991543	0.087598
Skewness	0.617945	0.325015	0.617945	-0.150294
Kurtosis	9.159034	7.940379	9.159034	3.340937

9. Visualizing Return Columns

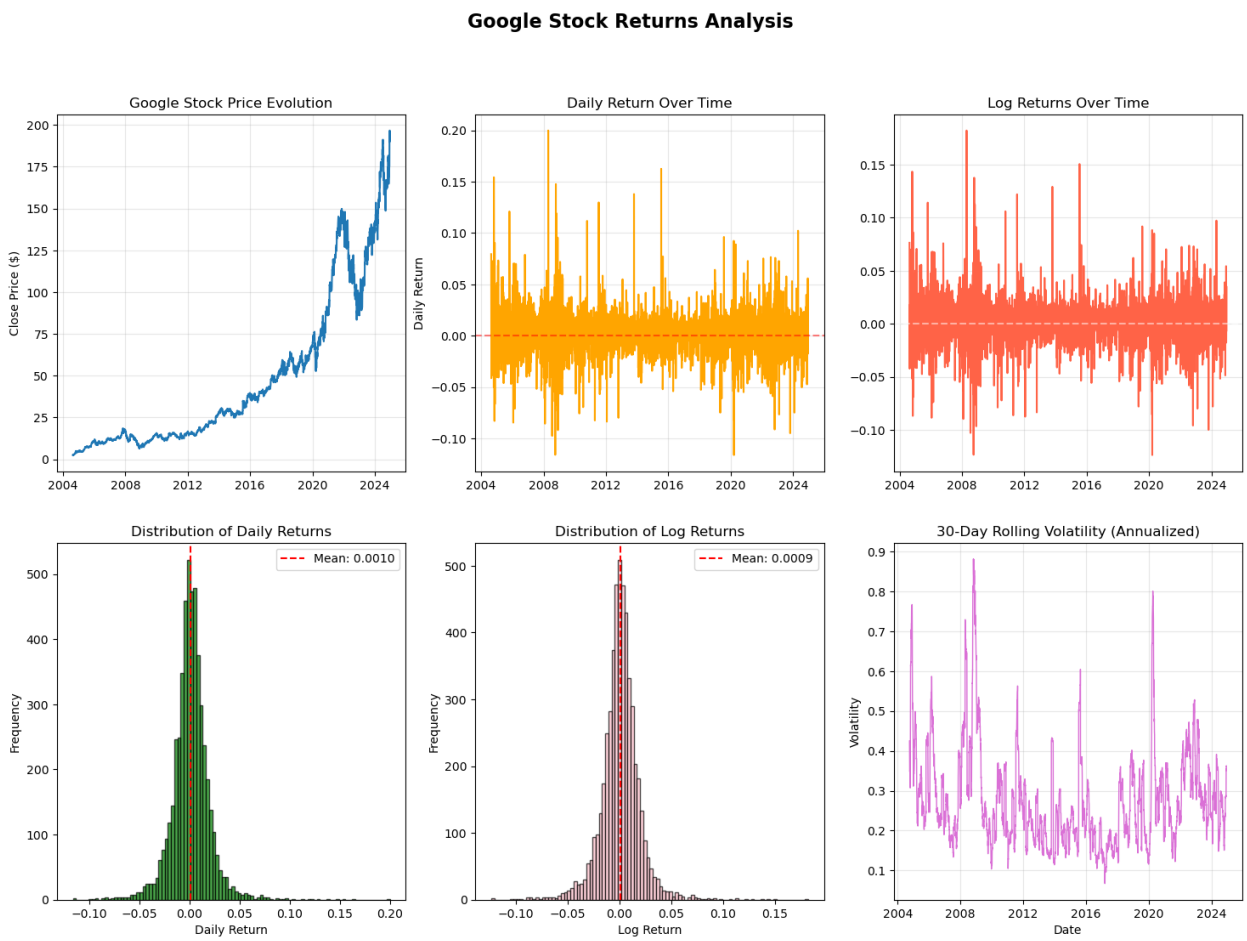
```
def plot_return_analysis(
    df : pd.DataFrame
):
    '''
    Displays Comprehensive Visualizations of Return Columns Analysis.
    '''

    fig, axes = plt.subplots(
        nrows = 2,
        ncols = 3,
        sharex = False,
        sharey = False,
        figsize = (18, 12)
    )
    fig.suptitle(
        t = 'Google Stock Returns Analysis',
        fontsize=16,
        fontweight='bold'
    )

    # 1. Google Stock Price Evolution
    axes[0,0].set_title('Google Stock Price Evolution')
    axes[0,0].plot(
        df.index,
        df['Close']
    )
```

```
    axes[0,0].set_ylabel('Close Price ($)')
    axes[0,0].grid(True, alpha = 0.3)

    # 2. Daily Returns Evolution
    axes[0,1].set_title('Daily Return Over Time')
    axes[0,1].plot(
        df.index,
        df['DailyReturns'],
        color = 'orange'
    )
    axes[0,1].set_ylabel('Daily Return')
    axes[0,1].axhline(y = 0, color = 'red', linestyle = '--', alpha =
0.5)
    axes[0,1].grid(True, alpha = 0.3)

    # 3. Log Returns Evolution
    axes[0,2].set_title('Log Returns Over Time')
    axes[0,2].plot(
        df.index,
        df['LogReturns'],
        color = 'tomato'
    )
    axes[0,2].axhline(y = 0, color = 'white', linestyle = '--', alpha
= 0.5)
    axes[0,2].grid(True, alpha = 0.3)

    # 4. Distribution of Daily Returns
    axes[1,0].hist(
        df['DailyReturns'],
        bins = 100,
        alpha = 0.7,
        color = 'green',
        edgecolor = 'black'
    )
    axes[1,0].set_title('Distribution of Daily Returns')
    axes[1,0].set_xlabel('Daily Return')
    axes[1,0].set_ylabel('Frequency')
    axes[1,0].axvline(
        x = df['DailyReturns'].mean(),
        color = 'red',
        linestyle = '--',
        label = f'Mean: {df["DailyReturns"].mean():.4f}')
    axes[1,0].legend()

    # 5. Distribution of Log Returns
    axes[1,1].hist(
        df['LogReturns'],
        bins = 100,
        alpha = 0.7,
```

```
        color = 'pink',
        edgecolor = 'black'
    )
    axes[1,1].set_title('Distribution of Log Returns')
    axes[1,1].set_xlabel('Log Return')
    axes[1,1].set_ylabel('Frequency')
    axes[1,1].axvline(
        x = df['LogReturns'].mean(),
        color = 'red',
        linestyle = '--',
        label = f'Mean: {df["LogReturns"].mean():.4f}')
    axes[1,1].legend()

    # 6. Rolling Volatility (30 Days)
    df['RollingVolatility'] =
df['DailyReturns'].rolling(window=30).std() * np.sqrt(252)
    axes[1,2].plot(
        df.index,
        df['RollingVolatility'],
        linewidth = 1,
        color = 'orchid'
    )
    axes[1,2].set_title('30-Day Rolling Volatility (Annualized)')
    axes[1,2].set_xlabel('Date')
    axes[1,2].set_ylabel('Volatility')
    axes[1,2].grid(True, alpha = 0.3)
```

Sample Function Call

```
plot_return_analysis(
    df = google_stock_df
)
```



Resampling

Resampling involves changing the frequency of your time series observations.
Two types of resampling are:

- **Upsampling :** Where you increase the frequency of the samples, such as from minutes to seconds.

- **Downsampling :** Where you decrease the frequency of the samples, such as from days to months.

The Pandas method used to do resampling is `pd.DataFrame.resample()` or `pd.Series.resample()` method.

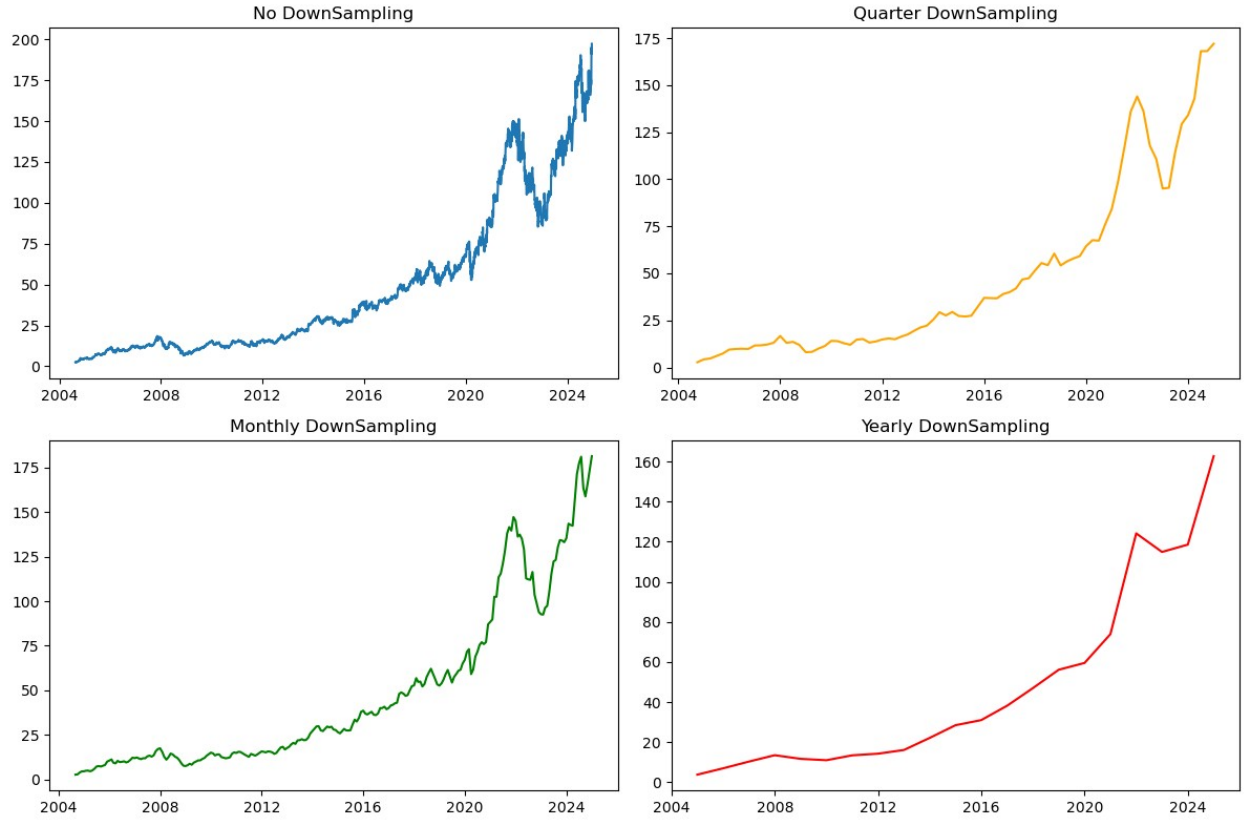
Downsampling

Keep in mind that when downsampling is done there is need of specifying two things.

1. freq : It uses the Offset Aliases to determine the frequency.
2. aggregation function : It is used so that the insight of the data do not get lost.

```
fig, axes = plt.subplots(
    nrows = 2,
    ncols = 2,
    sharex = False,
    sharey = False,
    figsize = (12, 8)
)

axes = axes.flatten()

axes[0].plot(
    google_stock_df.index,
    google_stock_df['Open']
)
axes[0].set_title('No DownSampling')

axes[1].plot(
    google_stock_df['Open'].resample(
        rule = 'QE',
    ).mean().index,
    google_stock_df['Open'].resample(
        rule = 'QE',
    ).mean(),
    color = 'orange'
)
axes[1].set_title('Quarter DownSampling')

axes[2].plot(
    google_stock_df['Open'].resample(
        rule = 'ME',
    ).mean().index,
    google_stock_df['Open'].resample(
        rule = 'ME',
    ).mean(),
    color = 'green'
)
axes[2].set_title('Monthly DownSampling')

axes[3].plot(
    google_stock_df['Open'].resample(
        rule = 'YE',
    ).mean().index,
    google_stock_df['Open'].resample(
        rule = 'YE',
    ).mean(),
    color = 'red'
)
axes[3].set_title('Yearly DownSampling')
```

```
plt.tight_layout()
plt.show()
```



Upsampling

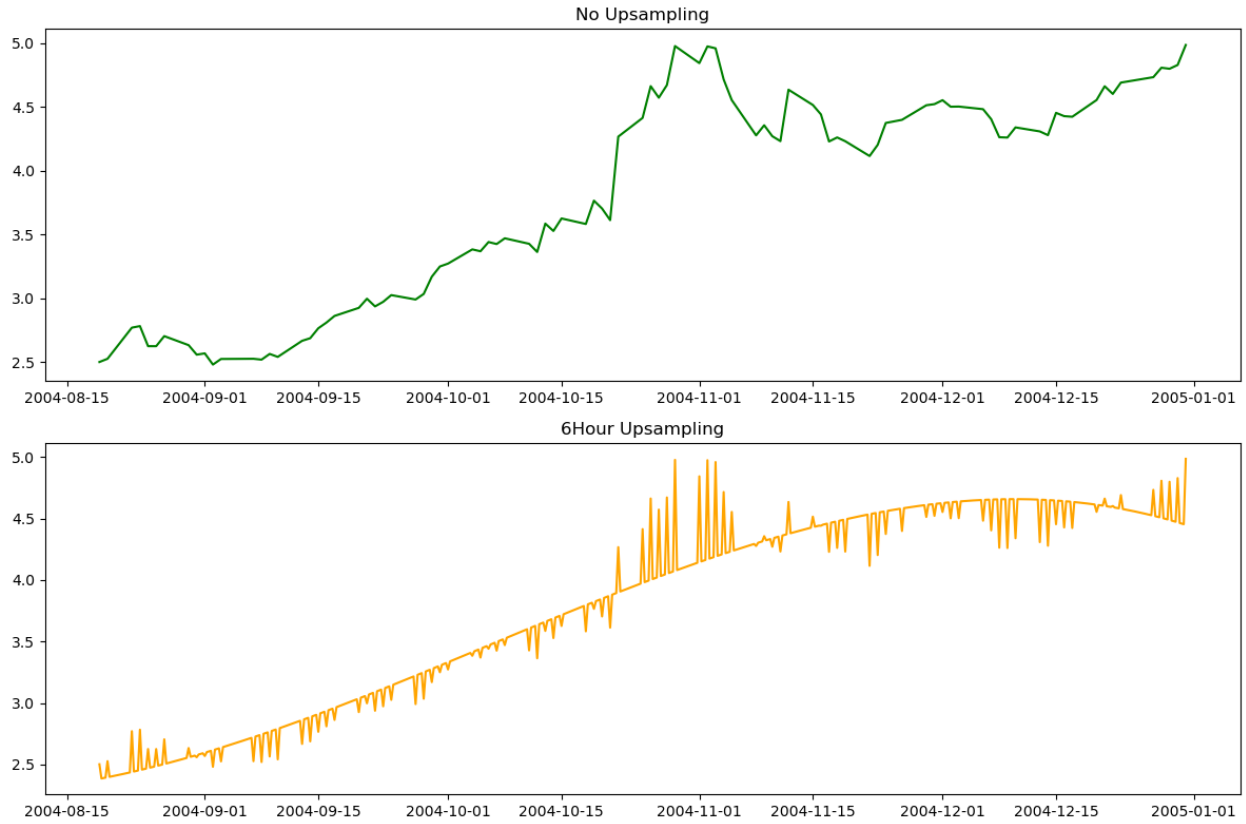
While performing Upsampling, we need to to keep this in mind.

- When Upsampling is done, a lot of new Timestamps are created.
  So, we have to fill those with the help of interpolate function.

- While using interpolate function for upsampling, we need to choose the suitable method among the available ones.

```
fig, axes = plt.subplots(
    nrows = 2,
    ncols = 1,
    sharex = False,
    sharey = False,
    figsize = (12, 8)
)

axes = axes.flatten()

axes[0].plot(
    google_stock_df.loc['2004'].index,
```

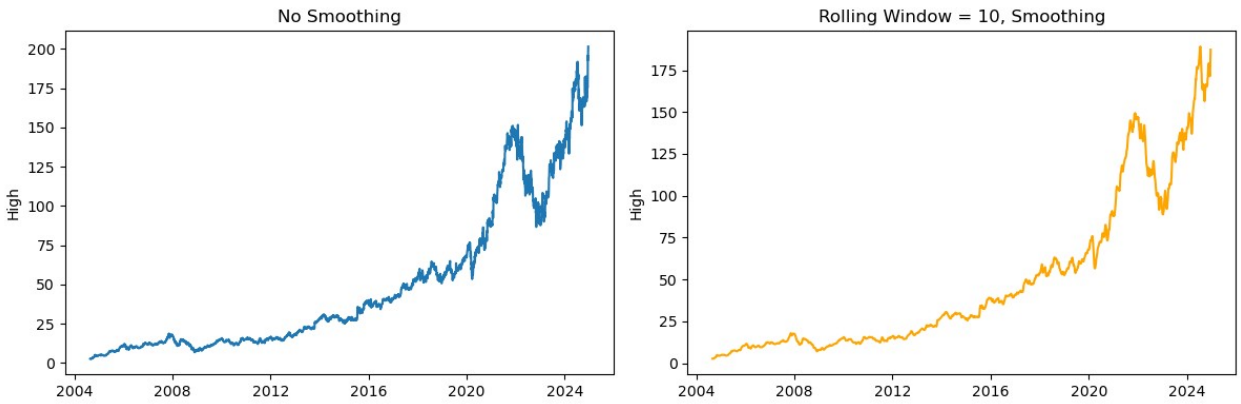
```
    google_stock_df['Open'].loc['2004'],
    color = 'green'
)
axes[0].set_title('No Upsampling')

axes[1].plot(
    google_stock_df['Open'].loc['2004'].resample(
        rule = '6h'
    ).interpolate(
        method = 'spline',
        order = 3
    ).index,
    google_stock_df['Open'].loc['2004'].resample(
        rule = '6h'
    ).interpolate(
        method = 'spline',
        order = 3
    ),
    color = 'orange'
)
axes[1].set_title('6Hour Upsampling')

plt.tight_layout()
plt.show()
```



Rolling Window (Smoothing)

Time series data in original format can be quite volatile, especially on smaller aggregation levels. The concept of rolling, or moving averages is a useful technique for smoothing time series data.

```
fig, axis = plt.subplots(
    nrows = 1,
    ncols = 2,
    sharex = False,
    sharey = False,
    figsize = (12, 4)
)

axis = axis.flatten()

axis[0].plot(
    google_stock_df['High'].index,
    google_stock_df['High']
)
axis[0].set_title('No Smoothing')
axis[0].set_ylabel('High')

axis[1].plot(
    google_stock_df['High'].rolling(
        window = 10
    ).mean().index,
    google_stock_df['High'].rolling(
        window = 10
    ).mean(),
    color = 'orange'
)
axis[1].set_title('Rolling Window = 10, Smoothing')
axis[1].set_ylabel('High')

plt.tight_layout()
plt.show()
```



Shifting

The `shift( )` function is Pandas is used to shift the entire series up or down by the desired number of periods.

See Documentation : Link

```
google_stock_df['Open'].shift(
    periods = 3
)
```

```
Date
2004-08-19           NaN
2004-08-20           NaN
2004-08-23           NaN
2004-08-24      2.502503
2004-08-25      2.527778
                  ...
2024-12-11    172.029999
2024-12-12    173.960007
2024-12-13    182.850006
2024-12-16    185.309998
2024-12-17    195.000000
Name: Open, Length: 5118, dtype: float64
```