**AI342 (Introduction to Data Science)**

**Project Title: ……………………………**

**Team Members**

|  |  |
| --- | --- |
| **Names** | **IDs** |
| **Mostafa Farag** | **200034359** |
| **Yousef Mohammed Ali** | **200033377** |
| **Yousef Mohamed Salama** | **200033523** |
| **Adham Jehad El Oraby** | **200029570** |
| **Shady Wafik Heshmat** | **200026518** |

**Supervised by:**

**Dr. Mohamed Marii  
TA/ Ahmed Tarek**

**Fall/2024-2025**

**Abstract**

* **This project focuses on analyzing and predicting Apple Inc.’s stock price trends over the past five years using historical stock data. The dataset includes essential attributes such as opening price, closing price, high, low, adjusted prices, and trading volume, spanning approximately 1822 business days. The primary objectives are to understand Apple’s stock price behavior, identify significant trends through Exploratory Data Analysis (EDA), and provide insights to aid financial decision-making.**
* **During the analysis, key patterns and milestones influencing stock performance were identified. Events such as strong financial results in February 2020, driven by Apple’s growing revenue from services (App Store, iCloud, and Apple Music) and wearables (AirPods, Apple Watch), were linked to notable stock price increases. Additionally, market sentiment and product innovation played critical roles in shaping price trends.**
* **The project highlights a clear relationship between external events and stock price fluctuations, supported by visualizations and statistical insights. Outlier analysis, missing value handling, and transformations were performed to ensure data integrity, which formed the basis for building a reliable predictive model.**
* **The results demonstrate the stock's volatility and responsiveness to key financial and market events. Although the methodology and model-building are ongoing, the project establishes a solid foundation for predicting stock prices, offering valuable insights for investors and analysts. Future enhancements could involve implementing advanced models like Random Forest to improve forecasting accuracy and capture long-term dependencies.**

**Introduction**

**Background :**

Stock price prediction is a crucial aspect of financial forecasting and investment strategies, providing investors and organizations with insights to make informed decisions. Over the years, advancements in data science, machine learning, and time-series analysis have enabled more accurate and reliable methods for stock price prediction. Apple Inc., a global leader in technology and innovation, has been a focal point for investors due to its significant market influence and strong financial performance.

By analyzing Apple’s historical stock data, it becomes possible to uncover key patterns, trends, and events that drive price fluctuations. This analysis is valuable not only for financial decision-making but also for understanding the relationship between external events (e.g., product launches, earnings reports) and stock performance.

### ****Problem Statement :****

The financial market is inherently volatile and influenced by numerous factors, making accurate stock price prediction a complex task. Investors often struggle to identify trends and make timely decisions due to the unpredictability of price movements. The project addresses the challenge of analyzing and predicting Apple Inc.'s stock price behavior using historical data over the past five years. By examining historical price data and identifying patterns, the goal is to provide insights that assist in predicting future stock price trends and understanding the impact of significant financial events.

### ****Objectives :****

The primary objectives of this project are as follows:

1. To conduct an in-depth **Exploratory Data Analysis (EDA)** of Apple Inc.’s stock price data to identify trends, patterns, and key events.
2. To analyze the impact of historical events, such as financial performance and market sentiment, on stock price behavior.
3. To preprocess the dataset by handling missing values, detecting outliers, and performing feature engineering for better data quality.
4. To lay the groundwork for building machine learning models that can accurately predict stock price trends.
5. To visualize the relationship between external factors and stock price movements to assist investors in making data-driven decisions.

**Methodology**

### ****Dataset :****

The dataset used in this project consists of **real, 100% authentic historical stock price data** for **Apple Inc.**, covering a period of **5 years (approximately 1822 business days)**.

* **Source**: Kaggle
* **Size**: The dataset contains several key features, including:
  + date: The trading date.
  + open, high, low, close: Prices for opening, highest, lowest, and closing of the day.
  + volume: Number of shares traded.
  + adjClose, adjHigh, adjLow, adjOpen: Adjusted stock prices to account for splits and dividends.

**Preprocessing Steps:**

1. Removal of irrelevant columns: Columns such as Unnamed: 0, symbol, and divCash were dropped.
2. Handling Missing Values:
   * Missing values in the splitFactor column were replaced with the mean value.
3. Duplicate Removal: Duplicate rows were removed to ensure data integrity.
4. Outlier Detection and Treatment:
   * Outliers in critical columns like close, high, low, and open were analyzed using box plots to identify any anomalies.
5. Feature Engineering:
   * The date column was converted to **datetime format**, and additional time-based features like Year and Year\_Quarter were extracted.

**Tools and Libraries:**

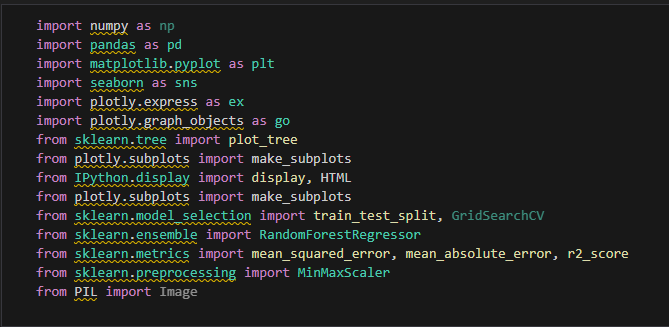
The following tools and libraries were used for data processing, visualization, and model building:

 **Programming Language**:

* Python

 **Libraries**:

* **Data Analysis**:
  + numpy
  + pandas
* **Visualization**:
  + matplotlib (pyplot)
  + seaborn
  + plotly (express, graph\_objects, subplots)
* **Machine Learning**:
  + sklearn.tree (e.g., plot\_tree)
  + sklearn.model\_selection (e.g., train\_test\_split, GridSearchCV)
  + sklearn.ensemble (e.g., RandomForestRegressor)
  + sklearn.metrics (e.g., mean\_squared\_error, mean\_absolute\_error, r2\_score)
  + sklearn.preprocessing (e.g., MinMaxScaler)
* **Display Utilities**:
  + IPython.display (e.g., display, HTML)
* **Image Processing**:
  + PIL (e.g., Image)



 **Development Environments**:

* **Jupyter Notebook**: For coding and documenting the project.
* **Visual Studio Code**: Used as an integrated development environment (IDE) for writing and debugging code.
* **Google Colab**: Utilized for running code in a cloud environment with access to GPU acceleration.

 **Platforms**:

* **Kaggle**: Accessed datasets and participated in discussions for additional insights.

 **AI Assistance**:

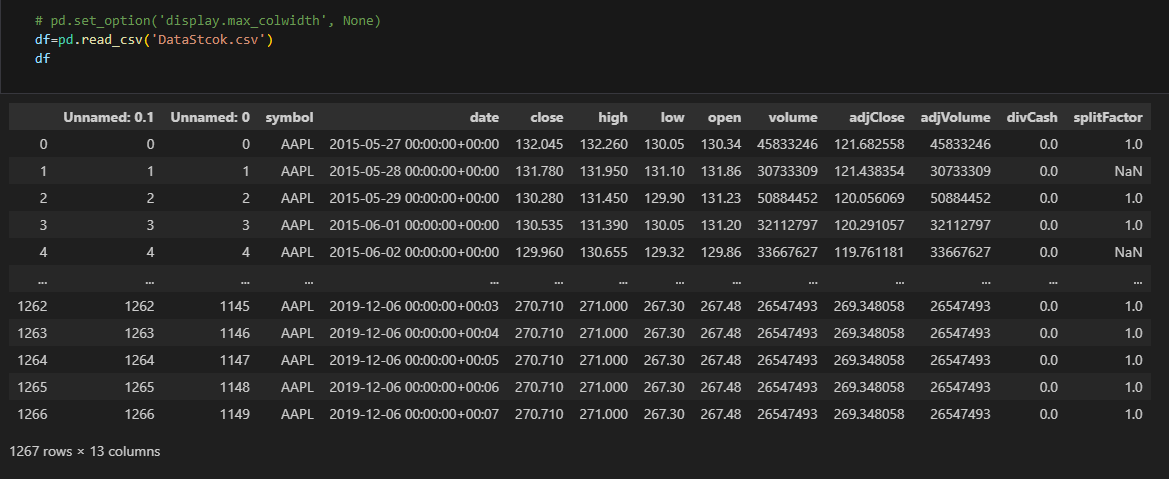
* **ChatGPT**: Employed for brainstorming, code suggestions, **and refining project documentation**.

 **Hardware**:

* Standard personal computer with sufficient RAM and processing power to handle the dataset and run models.

**Techniques:**

**Data Preprossesing :**



#### ****Code Explanation:****

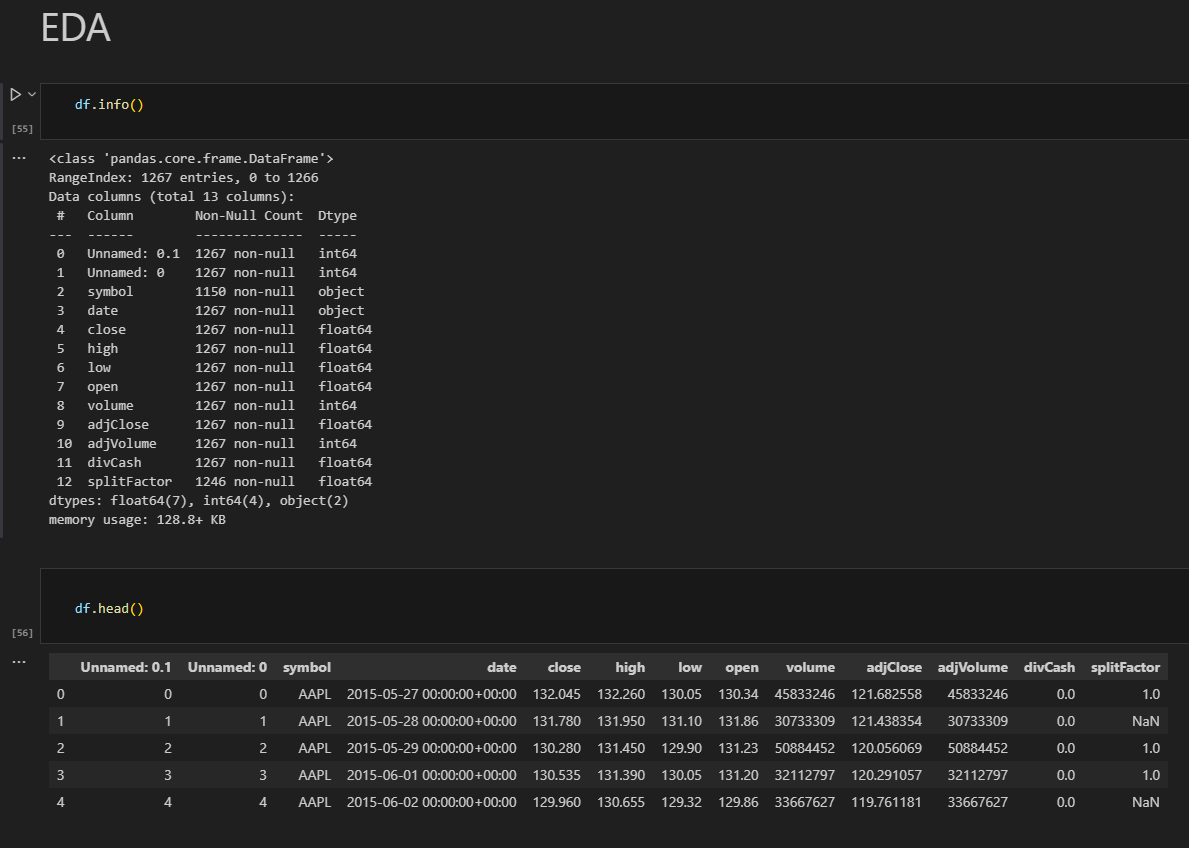
* **pd.read\_csv('DataStock.csv')**:
  + Loads the dataset from a CSV file into a DataFrame called df.
* **df**:
  + Displays the dataset, showing its structure (13 columns and 1267 rows).
  + Key columns include date, close, high, low, open, and volume.
  + There are redundant columns like Unnamed: 0.1, Unnamed: 0, and missing values in splitFactor.

#### ****Purpose:****

* Verify the data has been loaded correctly.
* Understand the dataset's structure and identify unnecessary columns and missing values.

#### ****Next Step:****

* Clean the data by:
  + Removing unnecessary columns.
  + Handling missing values.



#### ****Code Explanation:****

1. **df.info()**:
   * Used to check for missing values, column data types, and overall structure.
2. **df.head()**:
   * Displays the first five rows to quickly verify data integrity and formatting.

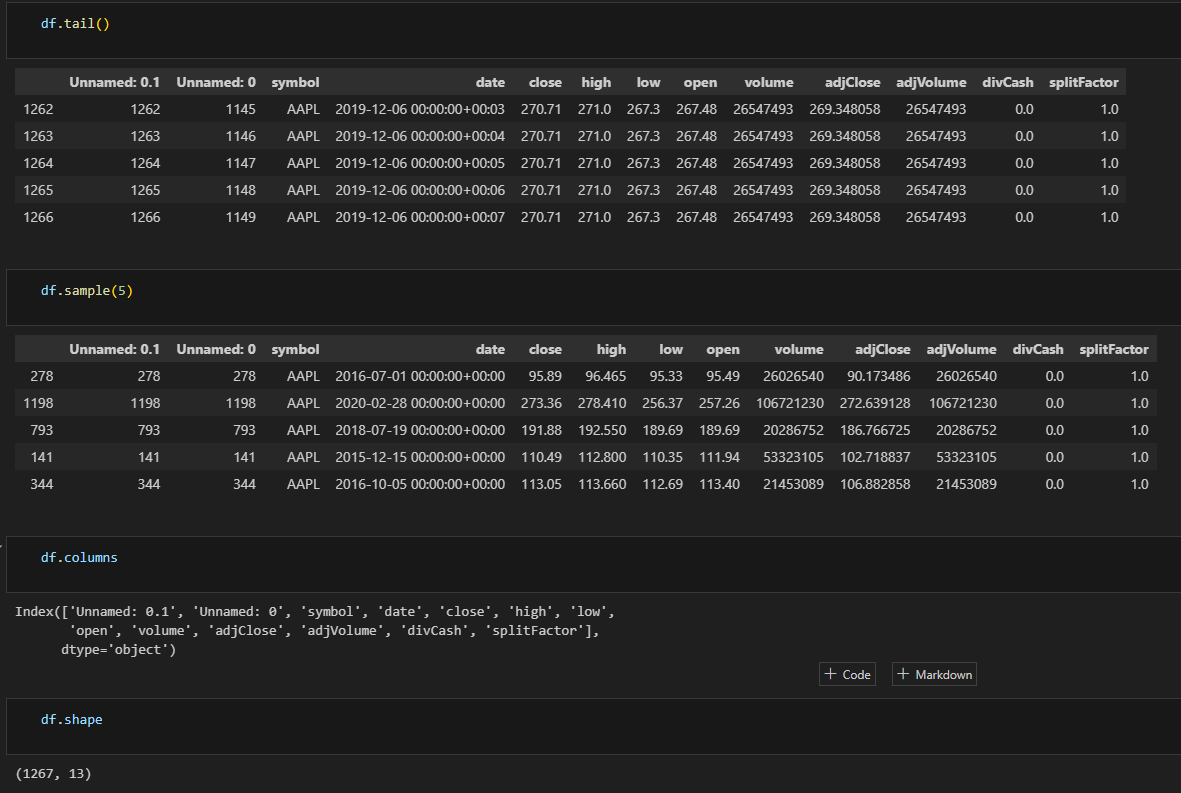
#### ****Techniques Used:****

* **Missing Value Identification**:
  + Identifies that the splitFactor column has missing values.
* **Unnecessary Columns**:
  + Highlights redundant columns like Unnamed: 0.1 and Unnamed: 0 that will need removal.

#### ****Purpose:****

This step lays the foundation for preprocessing by:

* Identifying columns with missing data for later treatment.
* Understanding the data types to facilitate cleaning and transformation



**Code Explanation:**

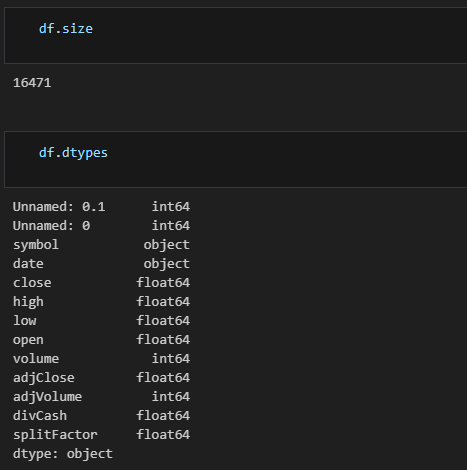
1. **df.tail()**: Displays the last 5 rows of the dataset to check the end of the data.
2. **df.sample(5)**: Displays 5 random rows to inspect the data for anomalies or inconsistencies.
3. **df.columns**: Lists all column names in the dataset.
4. **df.shape**: Outputs the shape of the dataset: 1267 rows and 13 columns.

**Techniques Used:**

* **Data Inspection**:
  + Verifying the dataset structure (number of rows, columns).
  + Checking random samples for inconsistencies.
* **Validation**:
  + Ensuring there are no unexpected missing data or abnormalities in the tail and random rows.

**Purpose:**

* To confirm that the dataset is properly loaded and structured.
* To identify redundant columns (Unnamed: 0.1, Unnamed: 0).
* To prepare for further cleaning and preprocessing steps.



**Code Explanation:**

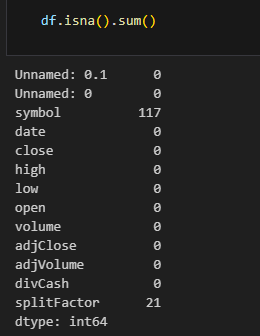
1. **df.size**: Returns the total number of elements in the DataFrame, which is **16,471**.
2. **df.dtypes**: Displays the data types of each column, including:
   * int64 for numerical identifiers (e.g., Unnamed: 0.1, volume).
   * float64 for numerical features like close, high, low.
   * object for non-numerical features such as symbol and date.

**Techniques Used:**

* **Data Type Inspection**:
  + Identifies data types to differentiate between numerical and non-numerical columns.
  + Ensures the correct data types are set for analysis and modeling.
* **Dataset Size Calculation**:
  + Confirms the total number of elements in the dataset for validation purposes.

**Purpose:**

* To verify the dataset structure and column types before preprocessing.
* To identify any columns requiring type conversion (e.g., converting date to datetime format).
* To ensure the data is ready for further cleaning and transformation.



**Code Explanation:**

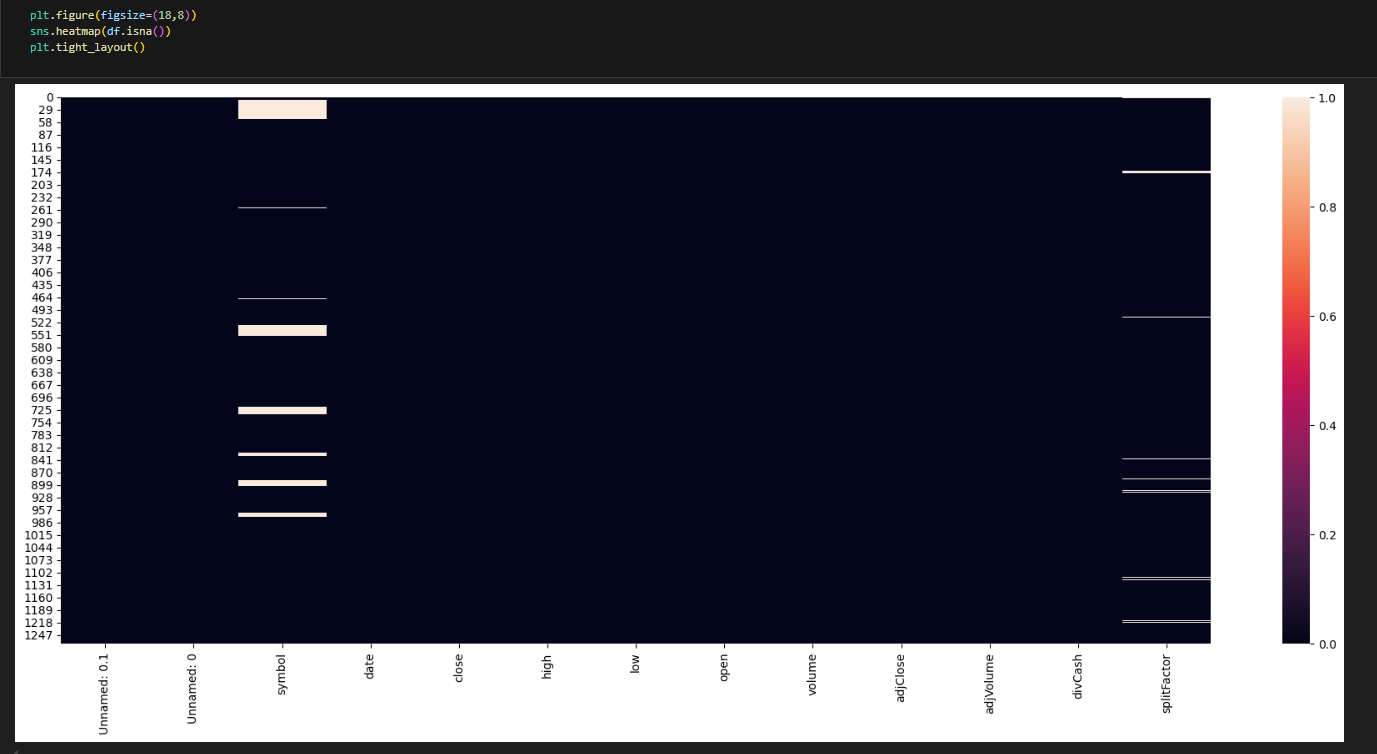
* **df.isna().sum()**:
  + Calculates the total number of missing values for each column in the dataset.

**Techniques Used:**

* **Missing Value Detection**:
  + Identifies that:
    - symbol has **117 missing values**.
    - splitFactor has **21 missing values**.
  + All other columns have no missing values.

**Purpose:**

* To detect missing values that require handling in the next steps of preprocessing.
* This helps ensure data quality and completeness for analysis or model building.



**Code Explanation:**

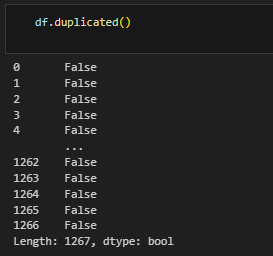
1. **plt.figure(figsize=(18,8))**: Sets the figure size for better visualization.
2. **sns.heatmap(df.isna())**:
   * Creates a heatmap to visualize missing values in the dataset.
   * Missing values are shown in lighter colors.
3. **plt.tight\_layout()**: Adjusts the layout to prevent overlaps.

**Techniques Used:**

* **Missing Value Visualization**:
  + A heatmap is used to identify where missing values exist in the dataset.
  + This helps quickly pinpoint the affected columns (symbol and splitFactor) and rows.

**Purpose:**

* To provide a visual understanding of missing data in the dataset.
* To identify patterns in missing values and determine the extent of data loss.
* This information guides the decision-making process for handling missing values (e.g., imputation or removal).



**Code Explanation:**

* **df.duplicated()**:
  + Checks for duplicate rows in the dataset.
  + Returns a **Boolean series** where True indicates a duplicate row and False indicates a unique row.

**Techniques Used:**

* **Duplicate Detection**:
  + Identifies whether there are any duplicate rows in the dataset that might need removal to ensure data quality.

**Purpose:**

* To confirm there are no duplicate rows in the dataset, as duplicates can distort analysis and modeling results.
* This step ensures the dataset remains clean and reliable for further preprocessing.



**Code Explanation:**

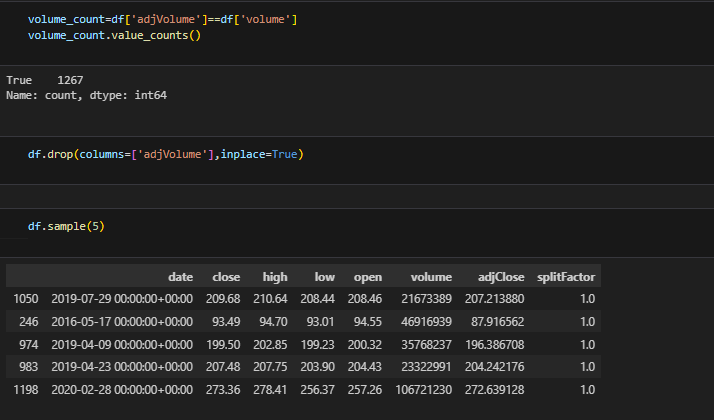
1. **df.drop(columns=['Unnamed: 0', 'symbol'], inplace=True)**
   * Drops the columns Unnamed: 0 and symbol.
2. **df.drop(columns=['Unnamed: 0.1'], inplace=True)**
   * Removes the redundant Unnamed: 0.1 column.
3. **df.drop(columns=['divCash'], inplace=True)**
   * Drops the column divCash, which is not needed for analysis.
4. **df['splitFactor'].describe()**
   * Provides statistical details for the splitFactor column (e.g., count, mean, min, max).
5. **mean = df['splitFactor'].mean()**
   * Calculates the mean of the splitFactor column to handle missing values.
6. **df['splitFactor'].fillna(mean, inplace=True)**
   * Fills the missing values in splitFactor with its mean.

**Techniques Used:**

* **Column Removal**:
  + Unnecessary and redundant columns were removed to clean the dataset (Unnamed, symbol, divCash).
* **Handling Missing Values**:
  + Missing values in the splitFactor column were replaced with the mean value to maintain data consistency.
* **Statistical Summarization**:
  + Used describe() to analyze and confirm that splitFactor has no variability (all values are 1).

**Purpose:**

* To clean the dataset by removing irrelevant columns and ensuring only meaningful data remains.
* To address missing values in splitFactor without introducing bias using mean imputation.
* To prepare the dataset for further analysis and modeling.



**Code Explanation:**

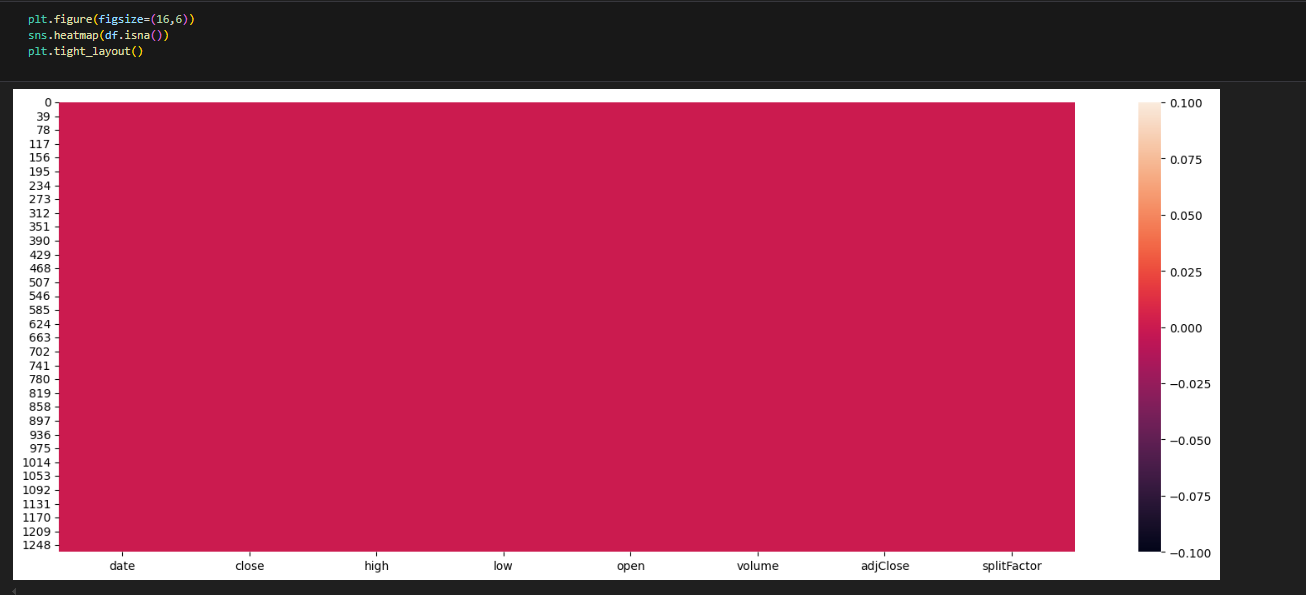
1. **volume\_count = df['adjVolume'] == df['volume']**
   * Checks if the values in adjVolume are equal to the values in volume.
2. **volume\_count.value\_counts()**
   * Counts how many rows have True or False values. In this case, all rows are True.
3. **df.drop(columns=['adjVolume'], inplace=True)**
   * Drops the adjVolume column because it is identical to the volume column.
4. **df.sample(5)**
   * Displays 5 random rows to confirm the dataset after dropping the column.

**Techniques Used:**

* **Column Redundancy Check**:
  + Compared adjVolume with volume to identify redundant columns.
* **Column Removal**:
  + Dropped adjVolume to avoid duplication and simplify the dataset.

**Purpose:**

* To identify and remove unnecessary columns (adjVolume) that duplicate information.
* To simplify the dataset and reduce computational complexity for further analysis.



**Code Explanation:**

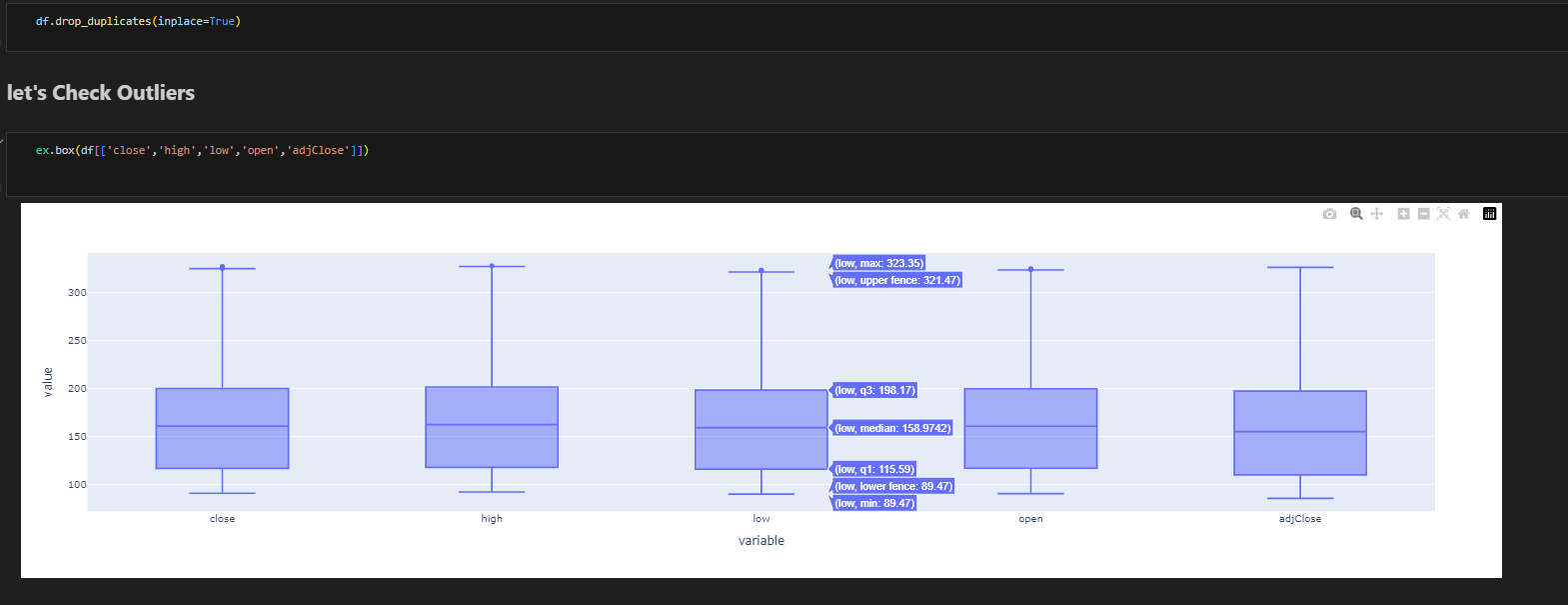
1. **plt.figure(figsize=(16,6))**
   * Sets the figure size to make the visualization clear.
2. **sns.heatmap(df.isna())**
   * Plots a heatmap to visualize missing values in the dataset.
   * The absence of lighter lines or gaps indicates there are **no missing values** in the dataset.
3. **plt.tight\_layout()**
   * Adjusts the layout to prevent overlaps and ensure the figure is properly displayed.

**Techniques Used:**

* **Missing Value Visualization**:
  + Confirms visually that all missing values have been handled, and the dataset is now clean.

**Purpose:**

* To verify that no missing values remain in the dataset after the preprocessing steps (column drops and missing value imputation).
* Ensures the data is ready for further analysis and modeling.



**Code Explanation:**

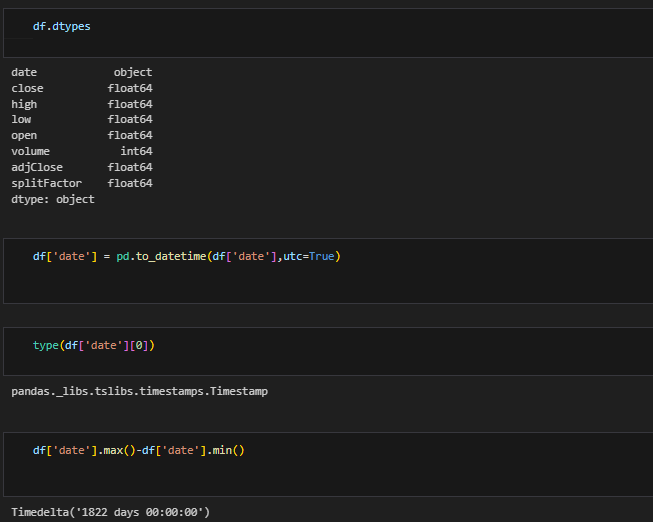
1. **df.drop\_duplicates(inplace=True)**:
   * Removes any duplicate rows in the dataset to ensure data integrity.
2. **ex.box(df[['close', 'high', 'low', 'open', 'adjClose']])**:
   * Generates boxplots for the columns close, high, low, open, and adjClose.
   * Boxplots visually display the data distribution, median, and potential outliers.

**Techniques Used:**

* **Duplicate Removal**:
  + Ensures the dataset contains only unique rows for accuracy.
* **Outlier Detection**:
  + Boxplots help identify outliers by visualizing extreme values outside the whiskers.
  + Outliers can be observed for all five columns, especially in higher values (e.g., close and high).

**Purpose:**

* To clean the dataset by eliminating duplicate records.
* To detect potential outliers in key numerical columns, which might need further treatment depending on their impact on the analysis or model performance.



**Code Explanation:**

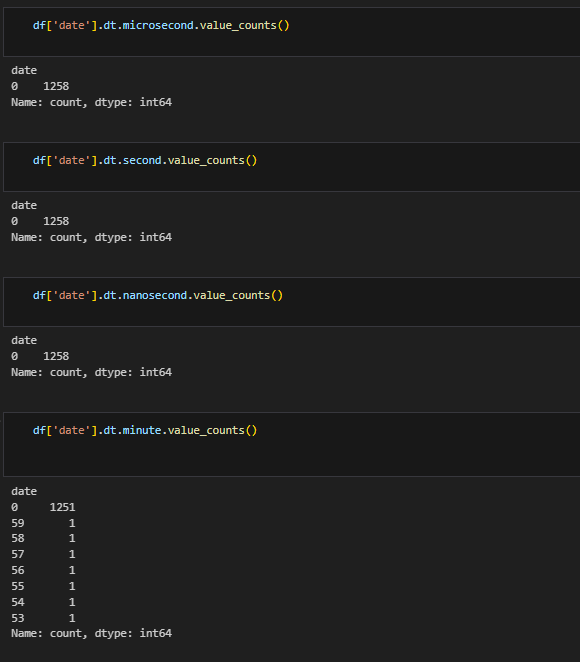
1. **df.dtypes**:
   * Displays the data types of all columns, showing that date is currently an **object** type.
2. **df['date'] = pd.to\_datetime(df['date'], utc=True)**:
   * Converts the date column from an object type to a **datetime** type with UTC timezone.
3. **type(df['date'][0])**:
   * Confirms that the conversion was successful by checking the type of the first value, which is now a **Timestamp**.
4. **df['date'].max() - df['date'].min()**:
   * Calculates the time range (duration) covered by the date column.
   * Result: **1822 days**.

**Techniques Used:**

* **Data Type Conversion**:
  + Converted the date column to a proper **datetime** format for easier time-based analysis.
* **Time Range Calculation**:
  + Evaluated the total time span of the dataset to verify its completeness.

**Purpose:**

* To prepare the date column for time-series analysis and visualization.
* To confirm the dataset spans **1822 days**, which matches the intended data range and ensures temporal consistency.



**Code Explanation:**

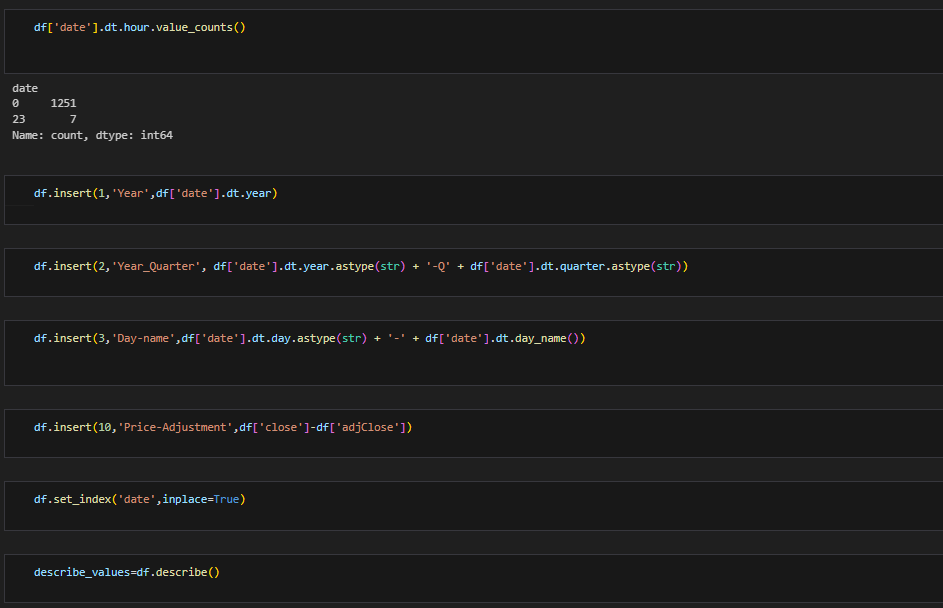
1. **df['date'].dt.microsecond.value\_counts()**:
   * Checks the distribution of **microsecond** values in the date column.
   * Result: All values are 0, indicating no microsecond information exists.
2. **df['date'].dt.second.value\_counts()**:
   * Checks the distribution of **second** values in the date column.
   * Result: All values are 0, meaning there are no second-level timestamps.
3. **df['date'].dt.nanosecond.value\_counts()**:
   * Verifies the distribution of **nanosecond** values.
   * Result: All values are 0, confirming no nanosecond-level data exists.
4. **df['date'].dt.minute.value\_counts()**:
   * Checks the distribution of **minute** values in the date column.
   * Most values are 0, but a few scattered entries exist with minute values like 59, 58, 57, etc.

**Techniques Used:**

* **Timestamp Granularity Check**:
  + Analyzed the date column to determine the granularity of the time data (microsecond, second, nanosecond, minute).

**Purpose:**

* To confirm that the date column has no **microsecond**, **second**, or **nanosecond** values, simplifying time-series analysis.
* To identify any inconsistencies or anomalies in minute-level data, which might require cleaning or investigation.



**Code Explanation:**

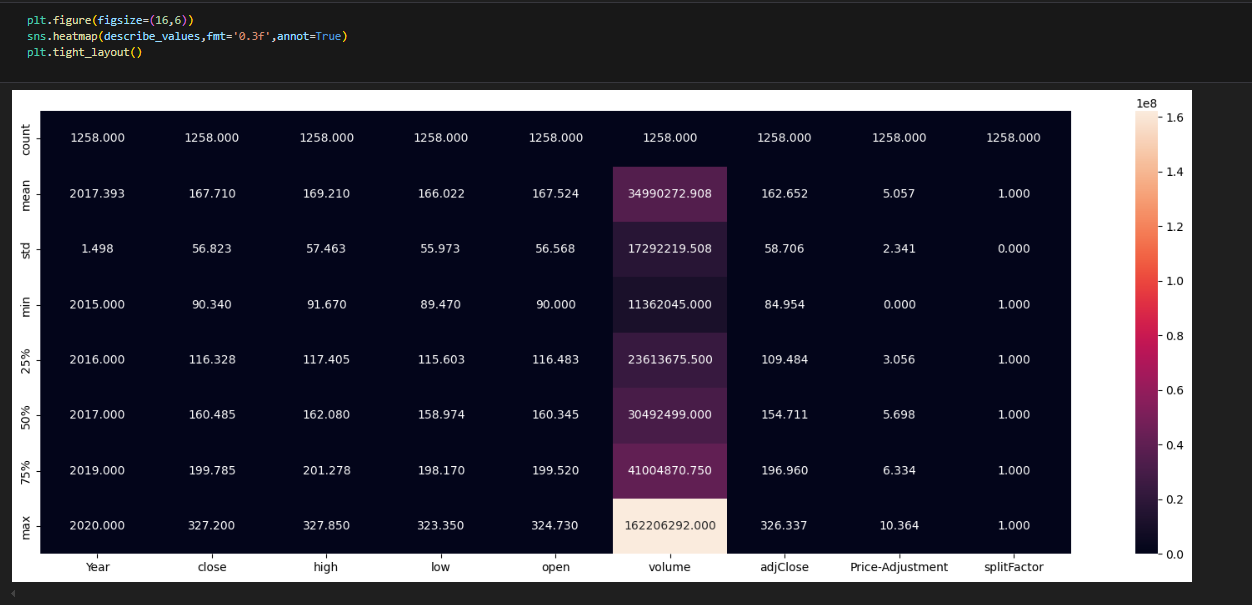
1. **df['date'].dt.hour.value\_counts()**:
   * Checks the distribution of hour values in the date column.
   * Most values are 0, confirming no significant hour-level data.
2. **df.insert(1, 'Year', df['date'].dt.year)**:
   * Adds a new column Year to extract the year from the date column.
3. **df.insert(2, 'Year\_Quarter', df['date'].dt.year.astype(str) + '-Q' + df['date'].dt.quarter.astype(str))**:
   * Creates a Year\_Quarter column, combining the year and quarter for easy grouping.
4. **df.insert(3, 'Day-name', df['date'].dt.day.astype(str) + '-' + df['date'].dt.day\_name())**:
   * Adds a Day-name column that combines the day number and day name from the date column.
5. **df.insert(10, 'Price-Adjustment', df['close'] - df['adjClose'])**:
   * Computes a new column Price-Adjustment, which is the difference between close and adjClose.
6. **df.set\_index('date', inplace=True)**:
   * Sets the date column as the DataFrame index for easier time-based analysis.
7. **describe\_values = df.describe()**:
   * Generates statistical summaries for all numerical columns, including count, mean, and standard deviation.

**Techniques Used:**

* **Feature Engineering**:
  + Extracted new features (Year, Year\_Quarter, Day-name) from the date column to improve temporal analysis.
  + Created the Price-Adjustment column to analyze price differences.
* **Indexing**:
  + Set date as the index to optimize time-series operations.
* **Descriptive Statistics**:
  + Summarized numerical data to understand the central tendencies and variability.

**Purpose:**

* To enhance the dataset with additional time-based features for better analysis.
* To compute derived metrics like Price-Adjustment for deeper financial insights.
* To ensure the dataset is properly indexed and ready for time-series analysis and visualization.



**Code Explanation:**

1. **plt.figure(figsize=(16,6))**:
   * Sets the figure size for the heatmap.
2. **sns.heatmap(describe\_values, fmt='0.3f', annot=True)**:
   * Creates a heatmap to visualize descriptive statistics (e.g., count, mean, std, min, max) for each numerical column.
   * **annot=True** ensures the values are displayed on the heatmap.
   * **fmt='0.3f'** formats the numbers to 3 decimal places.
3. **plt.tight\_layout()**:
   * Adjusts the layout to prevent overlaps and ensure clarity.

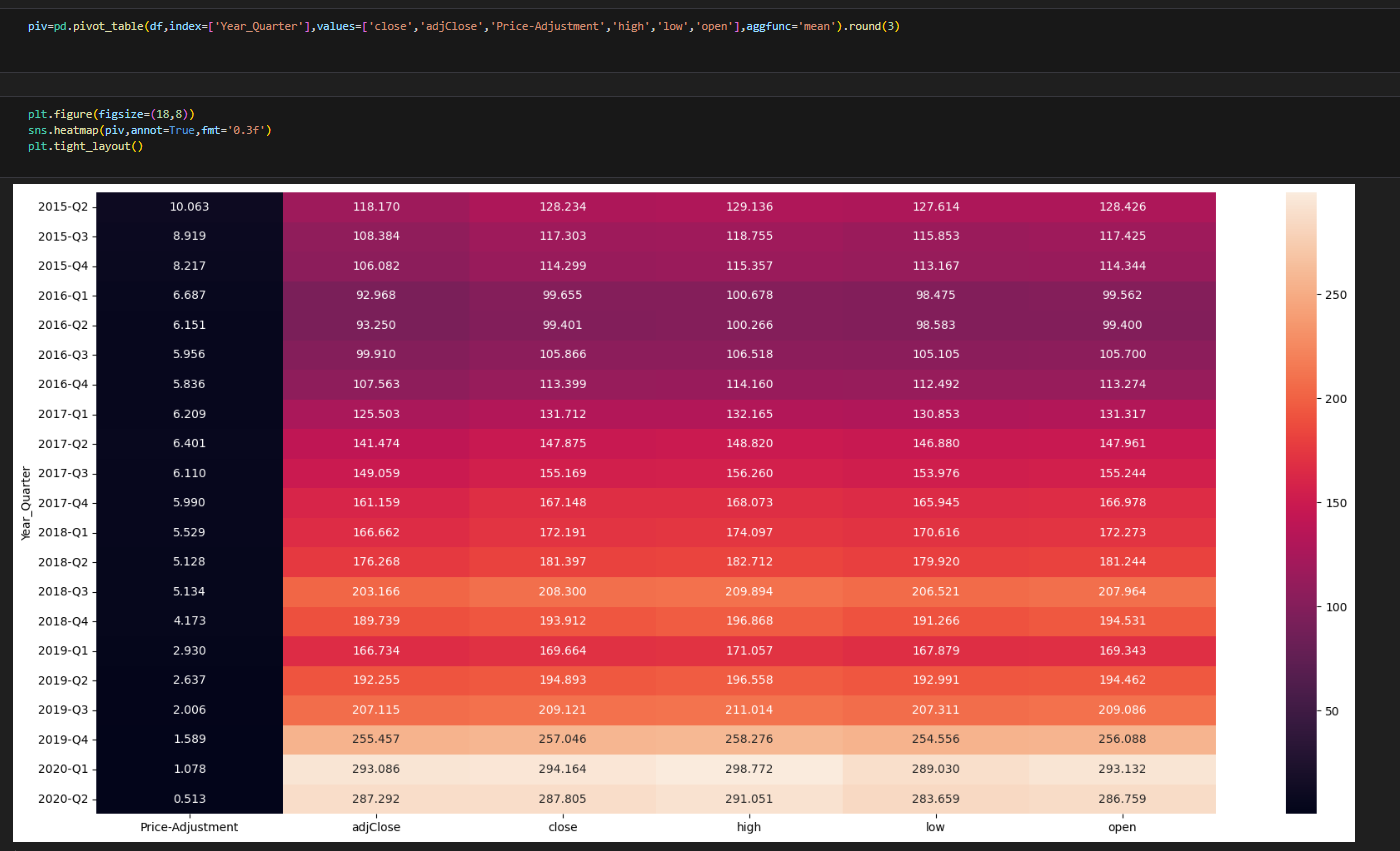
**Techniques Used:**

* **Descriptive Statistics Visualization**:
  + Summarizes numerical features (e.g., mean, standard deviation, min, max) to provide a clear overview of the dataset.
* **Heatmap Representation**:
  + Highlights the statistical summaries in an easy-to-read, visually appealing format.

**Purpose:**

* To analyze the central tendency (mean, median) and spread (standard deviation) of each numerical column.
* To identify potential anomalies (e.g., extreme max values in volume and close).
* To ensure a deep understanding of the data distribution before moving to further analysis or modeling.

**Visualizations :**



**Code Explanation:**

1. **piv = pd.pivot\_table(...)**:
   * Creates a pivot table with:
     + Year\_Quarter as the index.
     + Columns: 'close', 'adjClose', 'Price-Adjustment', 'high', 'low', and 'open'.
     + Aggregation function: **mean**.
     + Values rounded to **3 decimal places**.
2. **plt.figure(figsize=(18,8))**:
   * Adjusts the figure size for better readability.
3. **sns.heatmap(piv, annot=True, fmt='0.3f')**:
   * Visualizes the pivot table as a heatmap.
   * **annot=True**: Displays the values.
   * **fmt='0.3f'**: Formats values to 3 decimal places.
4. **plt.tight\_layout()**:
   * Ensures the plot layout is clean and prevents overlapping.

**Techniques Used:**

* **Pivot Table Creation**:
  + Aggregates key numerical columns (close, high, low, etc.) by quarter to analyze trends over time.
* **Heatmap Visualization**:
  + Highlights variations across columns (e.g., adjClose, high) and time (Year\_Quarter) using a color gradient.

**Purpose:**

* To identify quarterly trends in stock performance, including close, high, and low prices.
* To observe the gradual **increase** in adjusted close prices and other metrics over time.
* To provide an intuitive visual summary of stock behavior, aiding in further analysis and modeling.



**Code Explanation:**

1. **fig = ex.line(df, y=['close', 'adjClose'], height=500, width=1100)**:
   * Creates a line chart for close and adjClose prices using **plotly express**.
   * Sets the plot dimensions to a height of 500 and a width of 1100 for better visibility.
2. **fig.update\_layout(...)**:
   * Updates the layout with a dark theme using **template='plotly\_dark'**.
   * Sets the plot title to **"Stock Close/adjClose Trends"**.

**Techniques Used:**

* **Time-Series Visualization**:
  + Plots close and adjClose prices over time to observe trends and fluctuations.
* **Line Chart**:
  + Compares the regular closing price (close) and adjusted closing price (adjClose), highlighting any differences visually.

**Purpose:**

* To identify trends in stock performance over time.
* To observe the relationship between close and adjClose prices.
* To analyze how stock prices evolved and spot significant upward/downward movements.



**Code Explanation:**

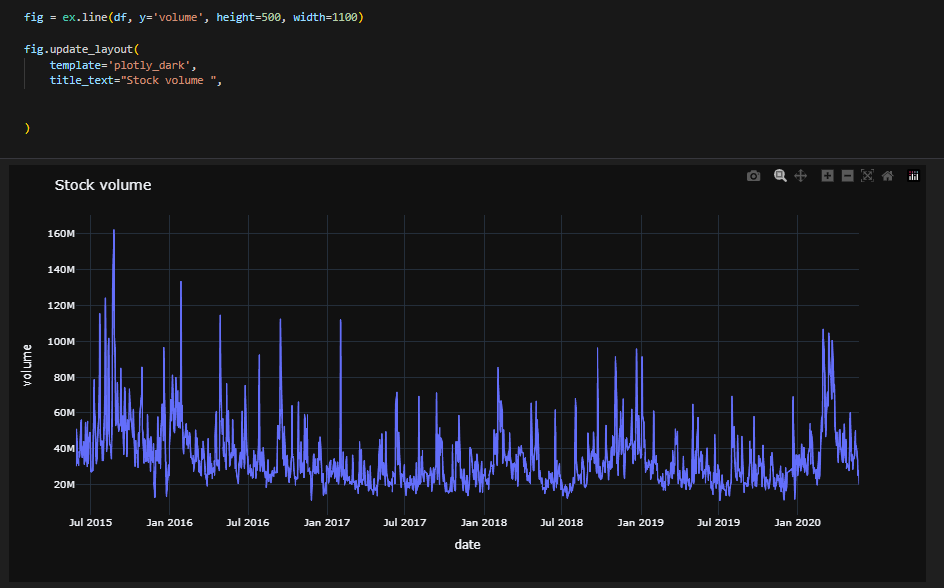
1. **fig = ex.line(df, y=['high', 'low'], title='Stock Low / High Price in Trends', height=500, width=1100)**:
   * Creates a line chart for the **high** and **low** stock prices over time.
   * Sets the title and adjusts the plot dimensions for clarity.
2. **fig.update\_layout(...)**:
   * Applies the **plotly\_dark** template for a dark-themed chart.
   * Updates the title text to **"Stock High/LOW Trends"** for clarity.

**Techniques Used:**

* **Time-Series Visualization**:
  + Displays high and low stock prices across the time range to observe variations.
* **Line Chart**:
  + Clearly compares the maximum (high) and minimum (low) stock prices over time, highlighting the price volatility.

**Purpose:**

* To analyze trends in the stock's **highest** and **lowest** prices over time.
* To observe significant fluctuations and identify periods of increased volatility.
* To understand the spread between high and low prices, which can provide insights into market behavior and risk.



**Code Explanation:**

1. **fig = ex.line(df, y=['high', 'low'], title='Stock Low / High Price in Trends', height=500, width=1100)**:
   * Creates a line chart for the **high** and **low** stock prices over time.
   * Sets the title and adjusts the plot dimensions for clarity.
2. **fig.update\_layout(...)**:
   * Applies the **plotly\_dark** template for a dark-themed chart.
   * Updates the title text to **"Stock High/LOW Trends"** for clarity.

**Techniques Used:**

* **Time-Series Visualization**:
  + Displays high and low stock prices across the time range to observe variations.
* **Line Chart**:
  + Clearly compares the maximum (high) and minimum (low) stock prices over time, highlighting the price volatility.

**Purpose:**

* To analyze trends in the stock's **highest** and **lowest** prices over time.
* To observe significant fluctuations and identify periods of increased volatility.
* To understand the spread between high and low prices, which can provide insights into market behavior and risk.



**Code Explanation:**

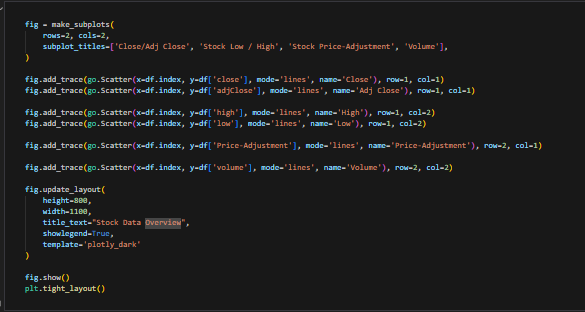
1. **fig = ex.line(df, y=['high', 'low'], title='Stock Low / High Price in Trends', height=500, width=1100)**:
   * Creates a line chart for the **high** and **low** stock prices over time.
   * Sets the title and adjusts the plot dimensions for clarity.
2. **fig.update\_layout(...)**:
   * Applies the **plotly\_dark** template for a dark-themed chart.
   * Updates the title text to **"Stock High/LOW Trends"** for clarity.

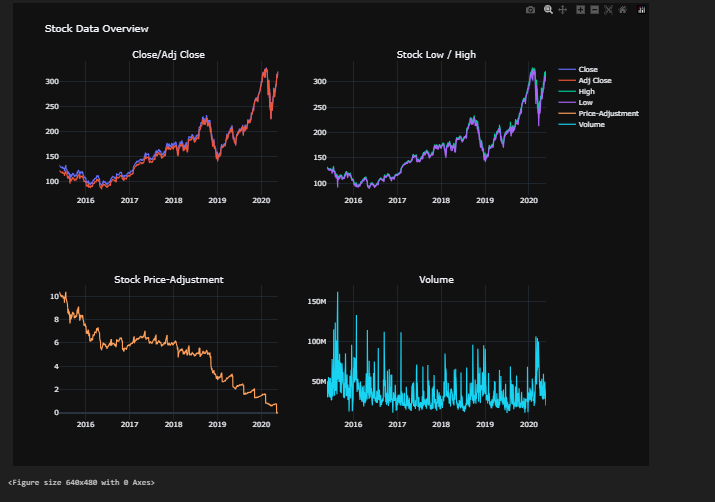
**Techniques Used:**

* **Time-Series Visualization**:
  + Displays high and low stock prices across the time range to observe variations.
* **Line Chart**:
  + Clearly compares the maximum (high) and minimum (low) stock prices over time, highlighting the price volatility.

**Purpose:**

* To analyze trends in the stock's **highest** and **lowest** prices over time.
* To observe significant fluctuations and identify periods of increased volatility.
* To understand the spread between high and low prices, which can provide insights into market behavior and risk.





**Code Explanation:**

1. **make\_subplots(rows=2, cols=2, subplot\_titles=[...])**:
   * Creates a 2x2 grid of subplots with four titles:
     + Close/Adj Close
     + Stock Low / High
     + Stock Price-Adjustment
     + Volume.
2. **fig.add\_trace(go.Scatter(...))**:
   * Adds line charts to each subplot:
     + **Top-Left**: close and adjClose prices.
     + **Top-Right**: high and low prices.
     + **Bottom-Left**: Price-Adjustment column.
     + **Bottom-Right**: volume column.
3. **fig.update\_layout(...)**:
   * Sets the figure layout:
     + Dimensions: height=800, width=1100.
     + Title: "Stock Data Overview".
     + Theme: plotly\_dark for dark visualization.
4. **fig.show()**: Displays the final figure with all subplots.

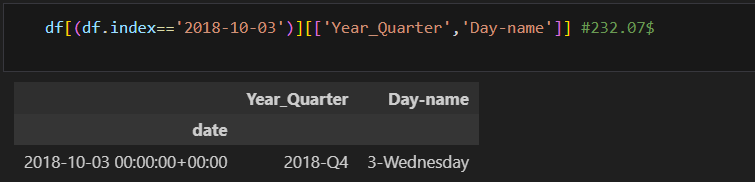
**Techniques Used:**

* **Multi-Subplot Visualization**:
  + Combines four related stock metrics into a single figure for better comparison.
* **Time-Series Plotting**:
  + Analyzes trends across close, high, low, price-adjustment, and volume data.

**Purpose:**

* **Close/Adj Close**: Tracks the general stock price movement and compares adjusted closing prices.
* **Stock Low/High**: Highlights the range of stock prices, showing volatility over time.
* **Price Adjustment**: Displays the adjustment difference between close and adjClose.
* **Volume**: Analyzes trading activity and identifies spikes or drops in volume.

**Data Preprocessing after Visualizations**



**Code Explanation:**

1. **df[(df.index == '2018-10-03')][['Year\_Quarter', 'Day-name']]**:
   * Filters the DataFrame for the specific date 2018-10-03.
   * Selects the columns Year\_Quarter and Day-name for display.
2. **#232.07$**:
   * A comment indicating the stock price on this specific date, likely for reference.

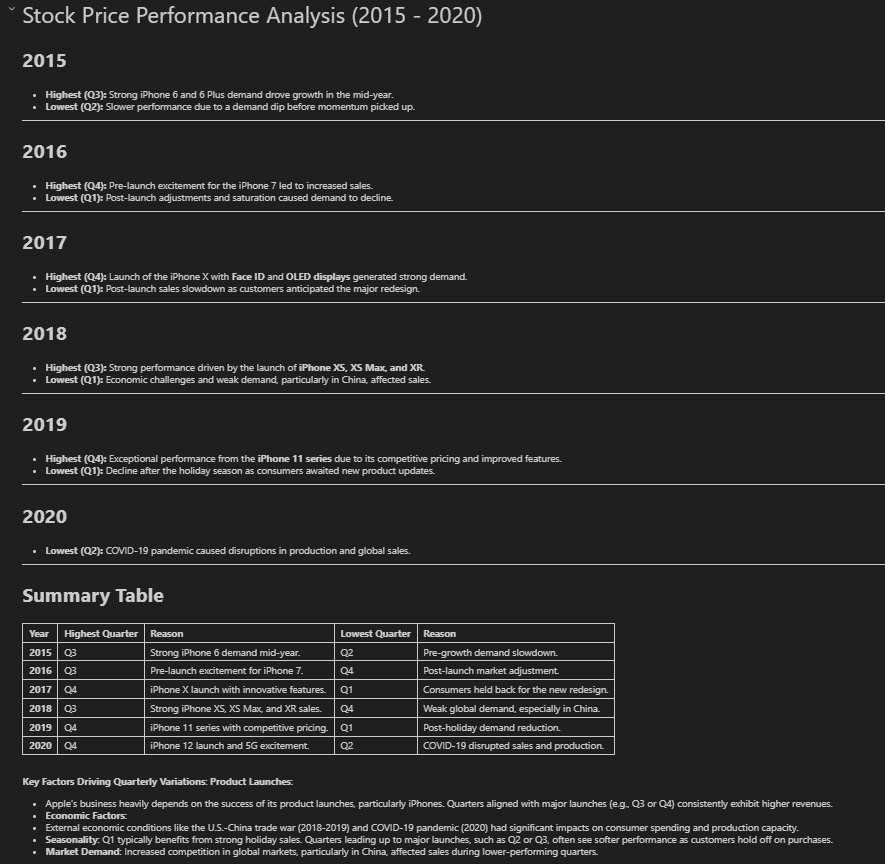
**Techniques Used:**

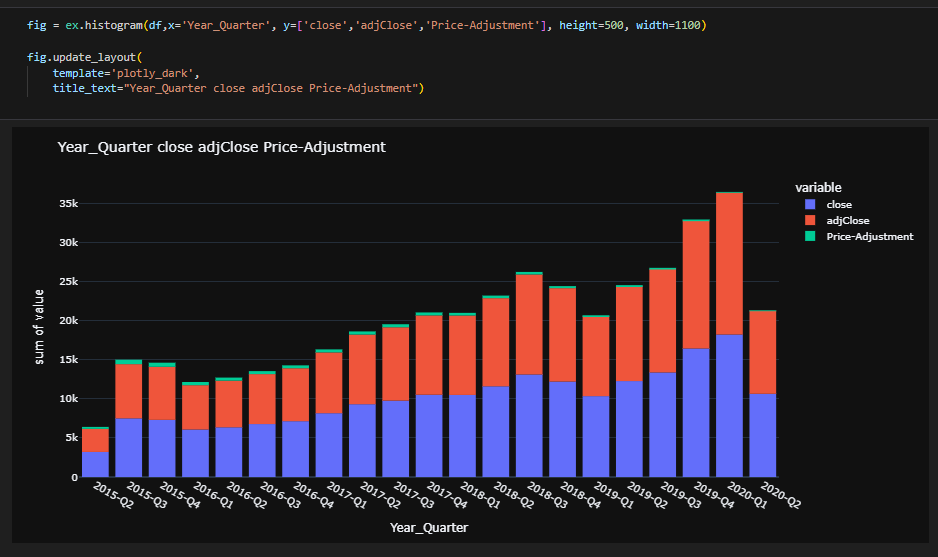
* **Date-Based Filtering**:
  + Filters the DataFrame using the index (date) to isolate records for a specific date.
* **Feature Extraction**:
  + Retrieves relevant preprocessed features (Year\_Quarter and Day-name) for the selected date.

**Purpose:**

* To extract detailed information (quarter and day name) for a specific date (2018-10-03).
* Provides insights into the stock’s performance on a particular day, facilitating event analysis or historical review.

**Discussion**





**Code Explanation:**

1. **ex.histogram(...)**:
   * Plots a stacked bar chart for close, adjClose, and Price-Adjustment values.
   * Groups data by Year\_Quarter.
   * Adjusts figure height to 500 and width to 1100 for better visibility.
2. **fig.update\_layout(...)**:
   * Applies a **dark theme** using plotly\_dark.
   * Adds the title **"Year\_Quarter close adjClose Price-Adjustment"** for clarity.

**Techniques Used:**

* **Stacked Bar Chart**:
  + Visualizes the sum of close, adjClose, and Price-Adjustment values across each quarter.
* **Quarterly Grouping**:
  + Aggregates the values by Year\_Quarter to show trends over time.

**Purpose:**

* To analyze the quarterly performance of key metrics:
  + **close**: The actual closing price.
  + **adjClose**: Adjusted closing price accounting for splits/dividends.
  + **Price-Adjustment**: The difference between close and adjClose.
* Highlights the upward trend in stock prices over time, especially in **2019-Q3** and **2019-Q4**, reflecting strong performance.
* Helps identify dips or declines (e.g., **2020-Q2**) caused by external factors like the **COVID-19 pandemic**.

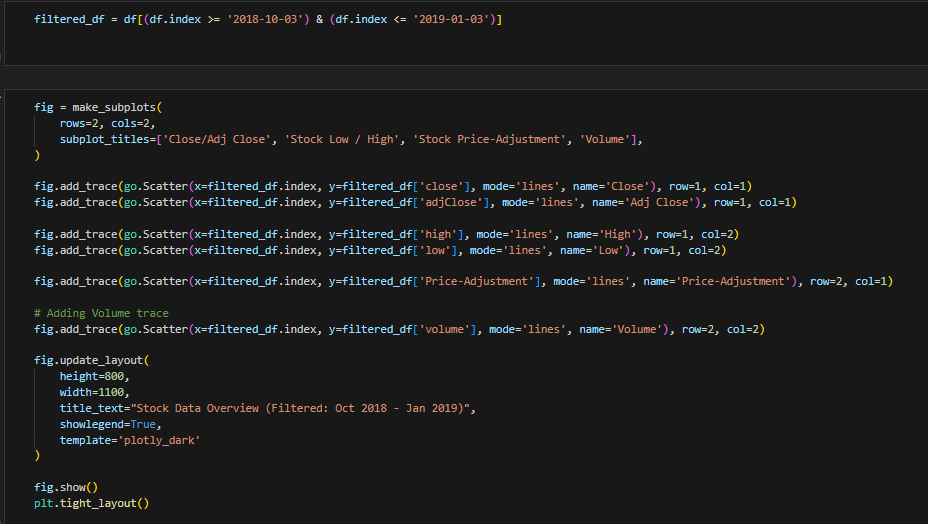
### Let’s Take Turning Point Filtered: Oct 2018 - Jan 2019

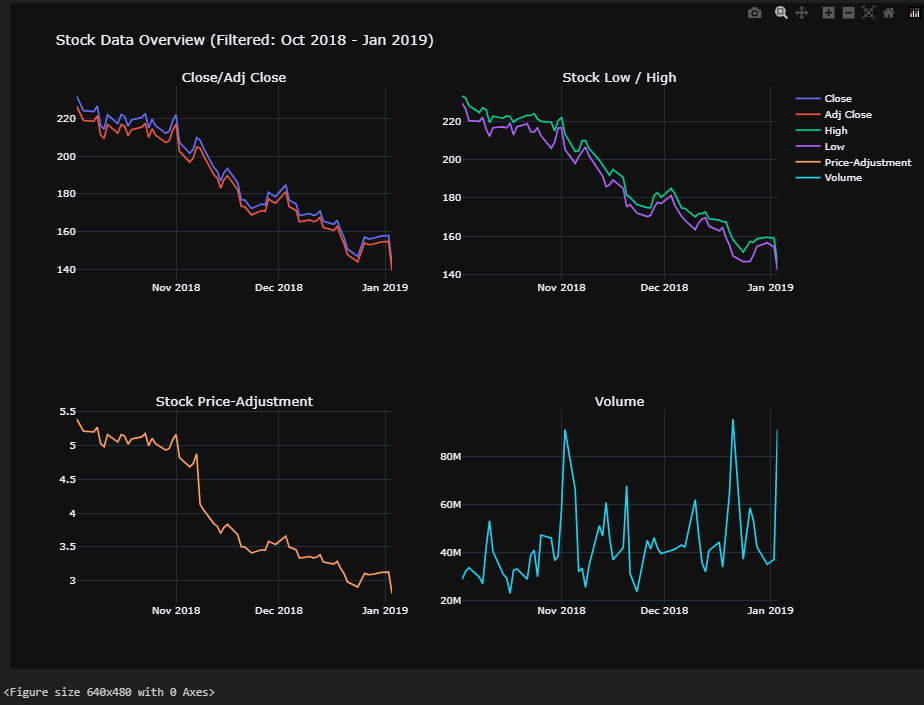
1. **3/Oct/2018 - $232.07 (High)**  
   **Reason: Strong Performance in the Market, Anticipation of New Products**

* Q3 2018 Earnings Report: In October 2018, Apple reported its fiscal Q3 earnings, which showed impressive revenue growth driven by higher-than-expected iPhone sales, strong performance from wearables, and a surge in services like the App Store and iCloud.
* Product Launches and Innovation: Apple had recently launched the iPhone XS, XS Max, and XR models in September 2018, which were well-received by the market. The anticipation of new products and innovations like Apple’s evolving ecosystem often generates investor optimism.
* Market Sentiment: At the time, Apple was viewed as a stable, high-performing tech stock, and investors were willing to pay a premium for its shares, pushing up the stock price.

1. **3/Jan/2019 - $142.19 (Low)**

* China’s Economic Slowdown: Economic slowdown in China, which is a major market for Apple, and the ongoing trade tensions between the U.S. and China likely affected Apple’s sales and investor sentiment. The drop in stock price reflects a loss of confidence in Apple’s growth prospects.
* General Market Volatility: The broader market was experiencing increased volatility during this period, partly due to concerns about the global economy, trade wars, and uncertainty over interest rates, all of which contributed to a broad sell-off in tech stocks.





### ****Code Explanation:****

1. **Filtering Data (First Code Cell):**

filtered\_df = df[(df.index >= '2018-10-03') & (df.index <= '2019-01-03')]

* + Filters the original dataset to only include data between **October 3, 2018** and **January 3, 2019**.

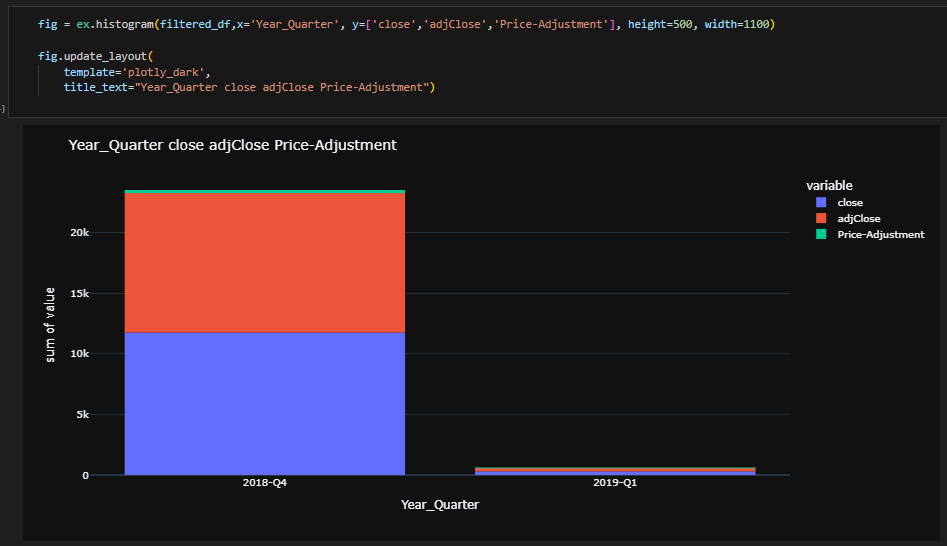
1. **Subplots Setup (Second Code Cell):**
   * **make\_subplots(rows=2, cols=2, subplot\_titles=[...])**: Creates a 2x2 subplot grid with the following visualizations:
     + **Top Left**: Close and Adj Close trends.
     + **Top Right**: High and Low stock prices.
     + **Bottom Left**: Price-Adjustment changes.
     + **Bottom Right**: Volume trends.
   * **fig.add\_trace(go.Scatter(...))**: Adds line charts for each metric.
   * **fig.update\_layout(...)**: Sets layout properties including figure size, title, and theme.
2. **Final Output (Third Image - Visualization):**
   * **Close/Adj Close**:
     + Declining trend observed from **October 2018** to **January 2019**, signaling a consistent drop in stock price.
   * **Stock Low/High**:
     + The low and high prices follow a similar downward trend, indicating price volatility during this period.
   * **Stock Price-Adjustment**:
     + Gradual decline in the difference between close and adjClose, which may suggest lower market adjustments.
   * **Volume**:
     + Noticeable spikes in trading volume, particularly in late **November** and early **January**, signaling significant market activity.

### ****Insights:****

1. **Market Behavior:**
   * Stock prices experienced a downward trend during this period, likely due to external factors (e.g., economic slowdown in China, trade tensions).
2. **Volume Activity:**
   * High trading volumes suggest increased investor activity, possibly reflecting panic selling or speculative trades during volatility.
3. **Price Adjustment:**
   * The narrowing difference between close and adjClose prices indicates reduced adjustments in stock value, reflecting stabilized market adjustments.

### ****Purpose:****

* To highlight the turning point from **October 2018** to **January 2019**, analyzing stock performance and investor activity during this critical period.
* Visualizing multiple metrics (prices, volume, and adjustments) provides a comprehensive understanding of market behavior.



**Code Explanation:**

1. **filtered\_df Histogram Plot:**
   * **X-axis:** Year\_Quarter
   * **Y-axis:** Aggregated values for close, adjClose, and Price-Adjustment.
   * Filters the data to the range of 2018-Q4 and 2019-Q1.
2. **fig.update\_layout(...):**
   * Applies a **dark theme** for better visibility using plotly\_dark.
   * Adds the title **"Year\_Quarter close adjClose Price-Adjustment"**.

**Techniques Used:**

* **Stacked Histogram:**
  + Visualizes the sum of close, adjClose, and Price-Adjustment across the filtered quarters:
    - **2018-Q4**: Significant values, with a noticeable contribution from close and adjClose.
    - **2019-Q1**: Drastic drop in stock prices, highlighting a significant decline in performance.

**Purpose:**

* **Key Insights:**
  + In **2018-Q4**, Apple experienced strong stock performance with high closing and adjusted closing values.
  + By **2019-Q1**, there was a sharp decline across all metrics, which aligns with market concerns like **China’s economic slowdown** and **global market volatility**.
* **Visualization Benefit:**
  + Makes it clear how the market performance changed over the two quarters, emphasizing the dramatic drop during early **2019-Q1**.

**3. 12/Feb/2020 - $327.2 (High)**

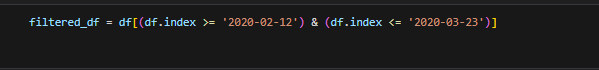
**Reason: Strong Performance, Services Growth, and Market Optimism**

* *Strong Financial Results*: By February 2020, Apple was riding high on solid Q1 2020 earnings. The company reported significant growth in services revenue (App Store, iCloud, Apple Music) and wearables (AirPods, Apple Watch), which helped offset slowing iPhone sales.
* *Market Sentiment*: Apple’s stock was benefiting from strong investor confidence, not only in its hardware but also in its growing services business, which provided more consistent, recurring revenue streams. The stock was also benefiting from broader optimism in the stock market.
* *Innovation and Product Development*: The launch of products like the Apple Watch and AirPods were gaining traction in the market. In addition, Apple was continuing to focus on increasing its services revenue, which helped diversify its income streams beyond just hardware.

**4. 23/Mar/2020 - $224.37 (Low):**

* **COVID-19 pandemic caused global market panic.**
* **Supply chain disruptions and uncertainty about future sales.**

**In short**, highs were driven by strong financials and optimism, while lows were due to external factors like economic concerns and the pandemic.



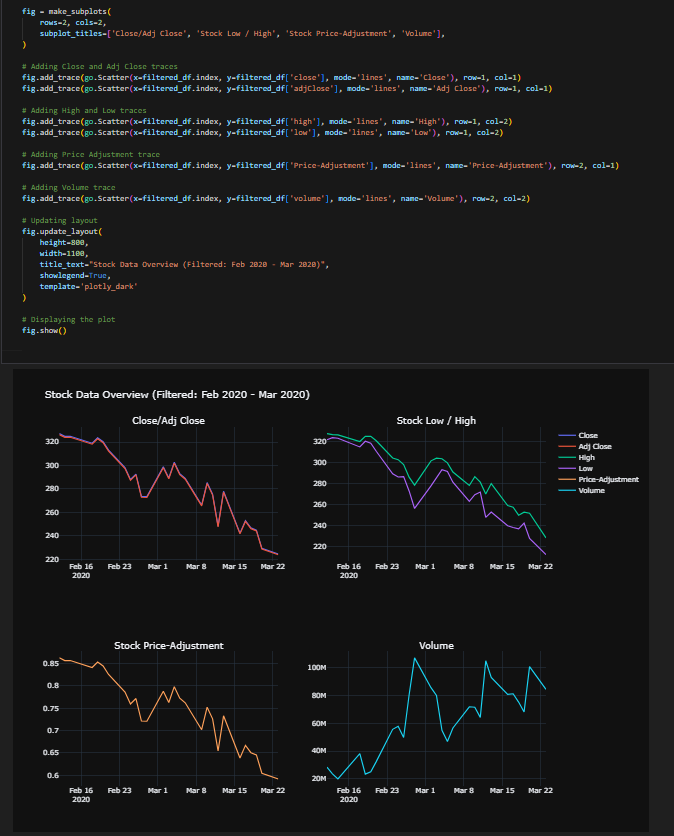
**Code Explanation**:  
The code filters the dataset to only include rows between **February 12, 2020**, and **March 23, 2020**.

filtered\_df = df[(df.index >= '2020-02-12') & (df.index <= '2020-03-23')]

* **df.index >= '2020-02-12'**: Includes dates on or after February 12, 2020.
* **df.index <= '2020-03-23'**: Includes dates on or before March 23, 2020.
* **&**: Ensures both conditions are applied simultaneously.

**Purpose**:  
To isolate data for a **critical time period** when the stock hit a high (February 12, 2020) and later dropped significantly (March 23, 2020) due to the COVID-19 pandemic and related market disruptions.

**Result**:  
A filtered dataset, stored in filtered\_df, containing only the relevant dates for further analysis.



### ****Code Explanation****

This code generates a **2x2 grid of line charts** to analyze stock metrics during the filtered period **(Feb 2020 - Mar 2020):**

1. **Subplots Setup**
   * **make\_subplots(rows=2, cols=2, subplot\_titles=[...])**:  
     Creates a **2x2 grid** with 4 separate plots for detailed analysis.
   * Titles for each subplot:
     + "Close/Adj Close"
     + "Stock Low / High"
     + "Stock Price-Adjustment"
     + "Volume"
2. **Adding Traces (Line Charts)**
   * **Close/Adj Close** (Top Left):
     + go.Scatter: Plots the **closing price** (close) and **adjusted closing price** (adjClose).
     + **Purpose**: Highlights the drop in stock values during this period.
   * **High/Low** (Top Right):
     + Plots the **highest** (high) and **lowest** (low) prices for each day.
     + **Purpose**: Shows volatility and price ranges during the stock decline.
   * **Price-Adjustment** (Bottom Left):
     + Displays the **adjustment difference** (Price-Adjustment) between close and adjClose.
     + **Purpose**: Tracks any consistent change or correction in stock values.
   * **Volume** (Bottom Right):
     + Visualizes the **trading volume** (volume).
     + **Purpose**: Peaks in volume reflect increased investor activity, likely due to market panic.
3. **Figure Layout**
   * **fig.update\_layout**:
     + Sets the figure dimensions (height=800, width=1100).
     + Adds a unified title: **"Stock Data Overview (Filtered: Feb 2020 - Mar 2020)"**.
     + Applies a clean, professional dark theme (template='plotly\_dark').
4. **Displaying the Plot**
   * **fig.show()**: Outputs the final visualization.

### ****Purpose****

* To analyze stock performance during the market's significant decline caused by the **COVID-19 pandemic**:
  + **Close/Adj Close**: Clear downward trend in stock price.
  + **High/Low**: Highlights volatility as the market adjusted to uncertainty.
  + **Price-Adjustment**: Tracks how adjusted closing prices aligned with closing prices.
  + **Volume**: Significant peaks in trading activity indicate panic selling or investor reactions.



### ****Code Explanation:****

This code creates a **stacked histogram** to display the aggregated values of close, adjClose, and Price-Adjustment for the filtered dataset based on Year\_Quarter.

**Code Details:**

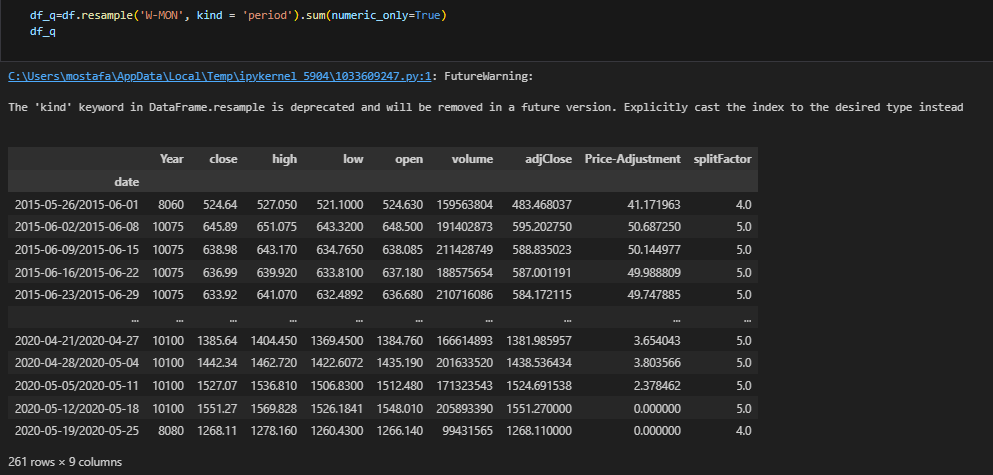
* **filtered\_df**: The filtered data is used.
* **X-axis**: Year\_Quarter – shows the quarters in the data.
* **Y-axis**: The sum of values for close, adjClose, and Price-Adjustment.
* **Stacked View**: The close (blue), adjClose (red), and Price-Adjustment (green) values are stacked vertically for comparison.
* **Layout**:
  + A dark theme (plotly\_dark) is applied.
  + Figure dimensions are customized to **height=500** and **width=1100**.
  + The title is set to **"Year\_Quarter close adjClose Price-Adjustment"**.

### ****Techniques Used:****

* **Stacked Histogram**:
  + Allows visual comparison of multiple variables (close, adjClose, Price-Adjustment) within a single quarter.

### ****Purpose:****

* To show how the **close** and **adjClose** values contributed to the overall stock behavior during **Q1 2020**.
* To highlight that the Price-Adjustment had a very small impact relative to close and adjClose values.



### ****Code Explanation:****

The code resamples the dataset **weekly** and calculates the **sum** of numeric columns.

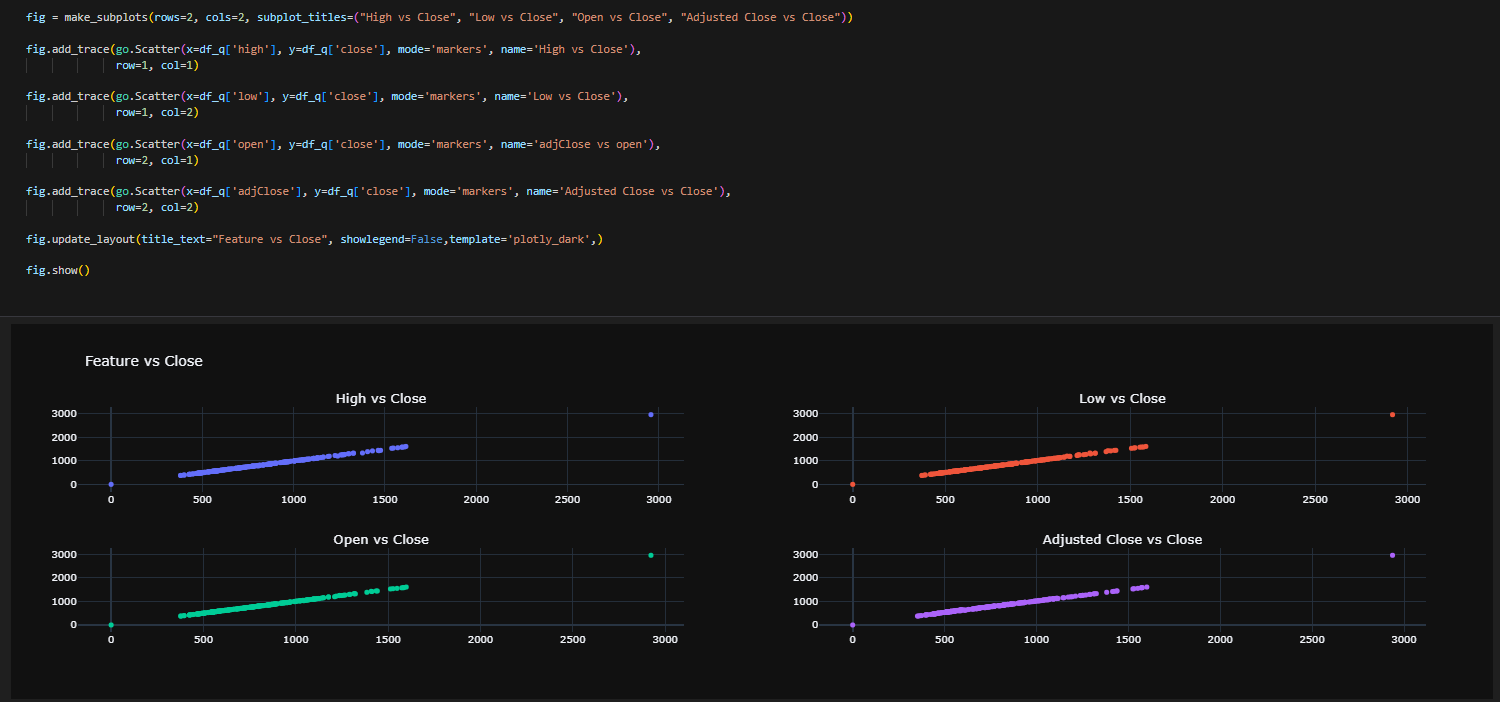
1. **df\_q = df.resample('W-MON', kind='period').sum(numeric\_only=True)**
   * **resample('W-MON')**: Resamples the data to group it by weeks ending on **Mondays**.
   * **kind='period'**: Indicates the grouping uses **periods** instead of timestamps.
   * **sum(numeric\_only=True)**: Sums all the **numeric columns** for each weekly group.
2. **Output:**
   * The result is stored in **df\_q**.
   * The table displays aggregated weekly sums for columns like close, high, low, open, volume, adjClose, Price-Adjustment, and splitFactor.

### ****Techniques Used:****

* **Resampling**: Aggregates the data based on a specific time frequency (weekly in this case).
* **Summation**: Computes the sum of numeric values for each week.

### ****Purpose:****

* To simplify and aggregate the dataset into **weekly data** for better analysis and visualization.
* Weekly data reduces noise from daily fluctuations and helps observe broader trends in close, high, low, volume, and other metrics.



### ****Code Explanation:****

This code generates a **2x2 grid of scatter plots** to visualize the relationship between various stock features and the **Close price**.

**Code Details:**

1. **Subplot Grid Setup**
   * **make\_subplots(rows=2, cols=2, subplot\_titles=[...])**
     + Creates a grid with **2 rows and 2 columns**.
     + Titles for each subplot:
       - "High vs Close"
       - "Low vs Close"
       - "Open vs Close"
       - "Adjusted Close vs Close"
2. **Adding Scatter Plots**
   * **Scatter 1** (Top Left):
     + **X-axis**: high (highest price).
     + **Y-axis**: close (closing price).
     + **Purpose**: Shows the relationship between the highest price and the close price.
   * **Scatter 2** (Top Right):
     + **X-axis**: low (lowest price).
     + **Y-axis**: close.
     + **Purpose**: Examines how the lowest price correlates with the close price.
   * **Scatter 3** (Bottom Left):
     + **X-axis**: open (opening price).
     + **Y-axis**: close.
     + **Purpose**: Displays how the opening price influences the close price.
   * **Scatter 4** (Bottom Right):
     + **X-axis**: adjClose (adjusted close price).
     + **Y-axis**: close.
     + **Purpose**: Highlights the relationship between adjusted close and close prices.
3. **Layout Customization**
   * **fig.update\_layout**:
     + Adds a main title: **"Feature vs Close"**.
     + Applies a **dark theme** (plotly\_dark) for styling.
     + Turns off the legend (showlegend=False) to avoid redundancy.
4. **Displaying the Plot**
   * **fig.show()**: Outputs the visualization.

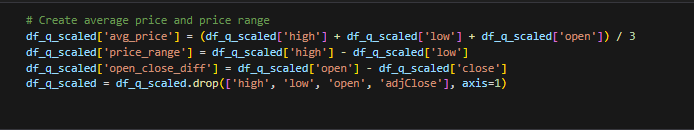
### ****Techniques Used:****

* **Scatter Plots**: Used to visualize pairwise relationships between features and the close price.
* **Subplots**: Combines multiple visualizations into a single, cohesive grid.

### ****Purpose:****

* To explore the **correlation** between key stock features (high, low, open, adjClose) and the **Close price**:
  + Helps identify patterns or trends between these variables.
  + Insights from these relationships can aid in predictive modeling or feature selection.

**Feature Engineering**



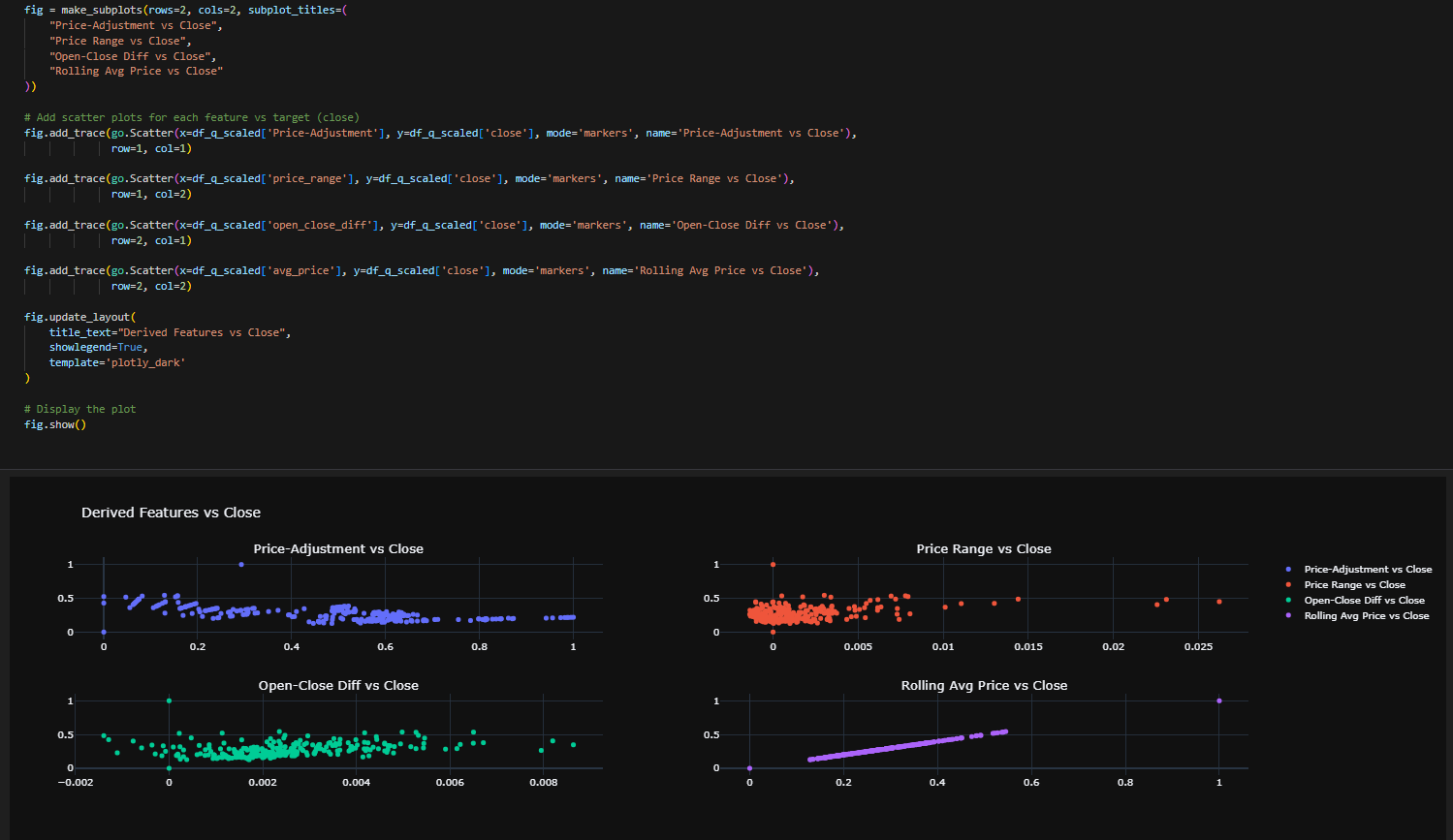
### ****Code Explanation****

The following code performs **feature engineering** by creating new meaningful columns and dropping redundant ones to optimize the dataset for analysis:

1. **Average Price Calculation**:  
   Combines high, low, and open prices, dividing by 3 to find the average price of the stock.
2. **Price Range**:  
   Calculates the difference between high and low prices to measure stock price volatility.
3. **Open-Close Difference**:  
   Computes the difference between open and close prices to indicate price movement during the session.
4. **Column Dropping**:  
   Removes the high, low, open, and adjClose columns to reduce redundancy.

### ****Purpose****

* Create new insights (average price, price range, and price difference).
* Simplify the dataset by keeping only the necessary features, ensuring a cleaner and more optimized structure for further analysis.



### ****Code Explanation****

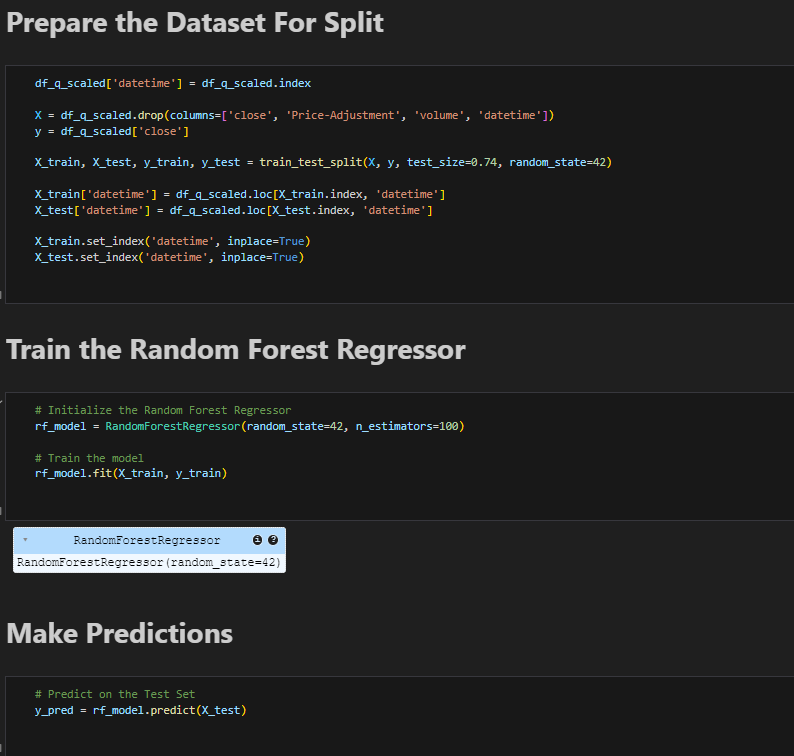
The following code generates scatter plots to visualize the correlation between newly created features (from feature engineering) and the target variable close:

1. **Price-Adjustment vs Close**:  
   Displays how the **Price Adjustment** feature relates to the close price.
2. **Price Range vs Close**:  
   Visualizes the correlation between the **Price Range** (difference between high and low prices) and the close price.
3. **Open-Close Diff vs Close**:  
   Shows the relationship between the **Open-Close Difference** and the close price.
4. **Rolling Avg Price vs Close**:  
   Plots the **Average Price** feature against the close price.

### ****Purpose****

* Analyze the **correlation** between derived features and the target variable close.
* Identify which features have stronger relationships with the close price for further model optimization.

**Implementation Details :**



### Code Explanation:

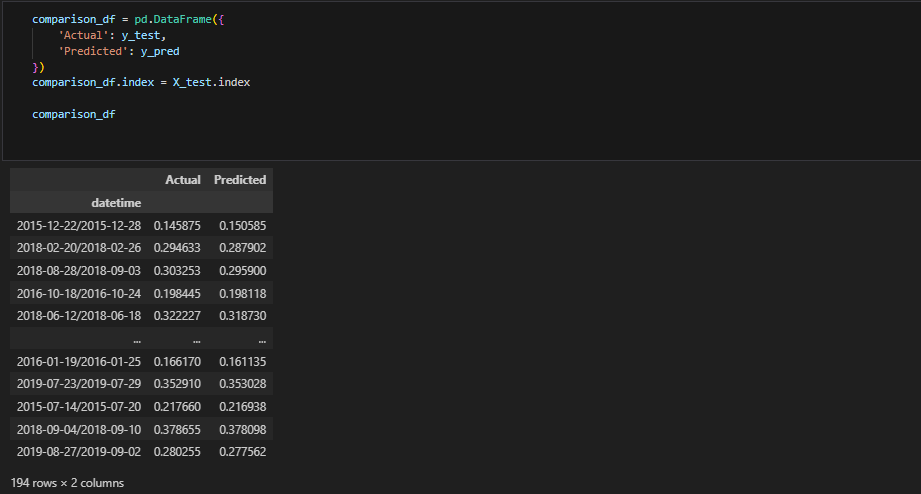
1. **Dataset Preparation for Splitting:**
   * df\_q\_scaled['datetime'] = df\_q\_scaled.index:  
     Adds the datetime column from the DataFrame index for easier manipulation.
   * X = df\_q\_scaled.drop(columns=['close', 'Price-Adjustment', 'volume', 'datetime']):  
     Defines the feature set X by dropping irrelevant columns like close (target) and datetime for splitting.
   * y = df\_q\_scaled['close']:  
     Sets the target variable y as the close column.
   * X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.74, random\_state=42):  
     Splits the dataset into training and test sets with a **74% test size** to train the model on a small subset.
   * X\_train['datetime'] = df\_q\_scaled.loc[X\_train.index, 'datetime'] and  
     X\_test['datetime'] = df\_q\_scaled.loc[X\_test.index, 'datetime']:  
     Adds the datetime column back to the training and test sets for better traceability.
   * X\_train.set\_index('datetime', inplace=True) and  
     X\_test.set\_index('datetime', inplace=True):  
     Sets datetime as the index for both training and test sets.
2. **Random Forest Regressor Training:**
   * rf\_model = RandomForestRegressor(random\_state=42, n\_estimators=100):  
     Initializes the Random Forest Regressor model with 100 decision trees (n\_estimators) and a fixed random state for reproducibility.
   * rf\_model.fit(X\_train, y\_train):  
     Trains the model using the training data (X\_train and y\_train).
3. **Prediction on Test Data:**
   * y\_pred = rf\_model.predict(X\_test):  
     Generates predictions for the test dataset using the trained Random Forest model.

### Techniques Used:

* **Data Splitting:**  
  Used train\_test\_split to separate the dataset into training and test sets to evaluate model performance effectively.
* **Feature Selection:**  
  Removed unnecessary features (Price-Adjustment, volume) to focus only on the relevant input for the model.
* **Datetime Handling:**  
  Incorporated the datetime index into the features for easier tracking and analysis.
* **Model Training:**  
  Trained a **Random Forest Regressor** with 100 estimators, a robust ensemble method that reduces overfitting.

### Purpose:

* Prepare the data for supervised machine learning (train-test split).
* Train a Random Forest model to predict stock closing prices.
* Generate predictions for unseen test data.



### Code Explanation:

1. **Creating a Comparison DataFrame:**
   * comparison\_df = pd.DataFrame({'Actual': y\_test, 'Predicted': y\_pred}):  
     A new DataFrame is created to compare the **actual values** (y\_test) with the **predicted values** (y\_pred).
2. **Setting the Index:**
   * comparison\_df.index = X\_test.index:  
     Sets the index of comparison\_df to the **datetime index** from the test data (X\_test.index) to keep the comparison aligned by date.
3. **Displaying the Results:**
   * comparison\_df:  
     Displays the DataFrame containing the actual and predicted values for the test data.

### Techniques Used:

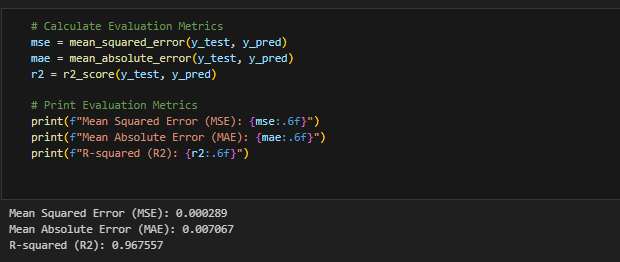
* **Result Comparison:**  
  A DataFrame is used to neatly display the actual vs. predicted values for easy analysis.
* **Datetime Alignment:**  
  The datetime index ensures proper alignment of results for a time-series comparison.

### Purpose:

* The table provides a direct comparison between the actual and predicted stock closing prices.
* Facilitates performance evaluation of the Random Forest model.

This belongs to the **Performance Metrics** section.

**Results !**



### Code Explanation:

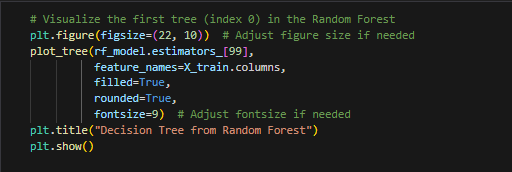
1. **Calculate Evaluation Metrics:**
   * mse = mean\_squared\_error(y\_test, y\_pred):  
     Calculates the **Mean Squared Error** (MSE), which measures the average squared difference between actual and predicted values.
   * mae = mean\_absolute\_error(y\_test, y\_pred):  
     Computes the **Mean Absolute Error** (MAE), representing the average of absolute differences between actual and predicted values.
   * r2 = r2\_score(y\_test, y\_pred):  
     Determines the **R-squared** (R²) value, which indicates how well the model explains the variability in the target variable.
2. **Print Evaluation Metrics:**
   * print(f"Mean Squared Error (MSE): {mse:.6f}"): Prints the MSE with six decimal places.
   * print(f"Mean Absolute Error (MAE): {mae:.6f}"): Outputs the MAE in a similar format.
   * print(f"R-squared (R2): {r2:.6f}"): Displays the R² score to evaluate the model's performance.
3. **Results:**
   * MSE: **0.000289** (low error indicates accurate predictions).
   * MAE: **0.007067** (small absolute differences between predicted and actual values).
   * R²: **0.967557** (close to 1, indicating excellent model performance).

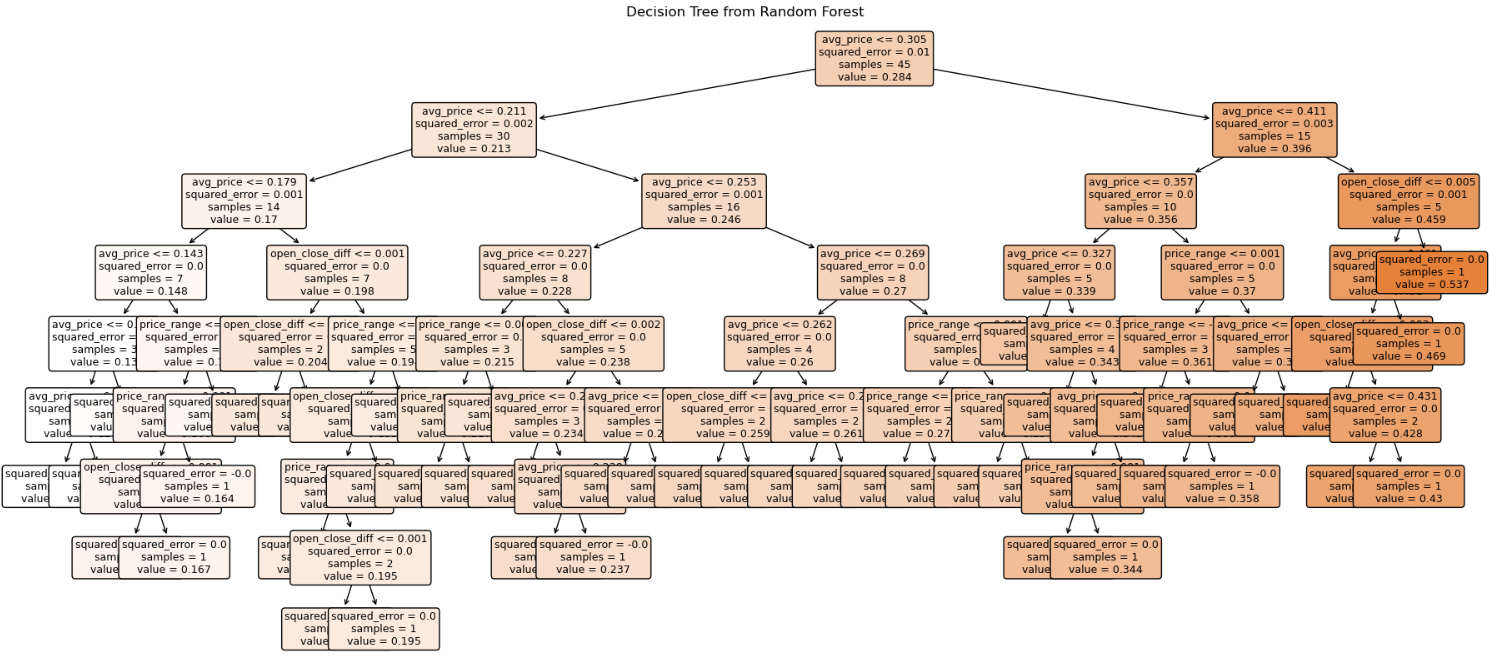
### Techniques Used:

* **Evaluation Metrics for Regression Models:**
  + MSE, MAE, and R² are standard performance metrics used to evaluate the accuracy and reliability of regression models.

### Purpose:

* These metrics provide a **quantitative assessment** of the model’s performance, helping determine how closely the predictions match the actual values.





### ****Code Explanation****

1. **Visualizing a Decision Tree in the Random Forest:**
   * plt.figure(figsize=(22, 10)): Sets the figure size for better visibility.
   * plot\_tree(rf\_model.estimators\_[99], ...):
     + Visualizes the **99th tree** from the trained Random Forest model.
     + feature\_names=X\_train.columns: Provides feature names for tree nodes.
     + filled=True: Colors nodes based on predicted values.
     + rounded=True: Rounds the edges of the tree nodes for readability.
     + fontsize=9: Adjusts the text size.
   * plt.title("Decision Tree from Random Forest"): Adds a title to the plot.
   * plt.show(): Displays the decision tree.
2. **Generated Decision Tree:**
   * The **tree structure** represents decision splits based on features (e.g., avg\_price, price\_range, open\_close\_diff).
   * **Nodes** display:
     + **Feature condition** (e.g., avg\_price <= 0.305)
     + **Squared Error**: Measures how well the splits reduce variance.
     + **Samples**: Number of data points in each node.
     + **Value**: Predicted value for the node.

### ****Purpose****

* This visualization highlights the **individual decision-making process** of a single tree within the Random Forest.
* It explains how specific features influence predictions and provides **insight into feature importance**.



### ****Code Explanation****

1. **Creating the Figure**
   * fig = go.Figure(): Initializes a Plotly figure object.
2. **Adding the Perfect Prediction Line**
   * fig.add\_trace(go.Scatter(...)):
     + Draws a **dashed red line** representing the **ideal prediction** where actual values = predicted values.
     + x=[min(y\_test), max(y\_test)]: x-axis range is the minimum and maximum of y\_test.
     + y=[min(y\_test), max(y\_test)]: y-axis range matches the x-axis for the perfect diagonal.
3. **Adding Predicted Values**
   * fig.add\_trace(go.Scatter(...)):
     + Plots **actual vs. predicted values** as scatter points.
     + x=y\_test, y=y\_pred: Maps actual values on the x-axis and predicted values on the y-axis.
     + marker=dict(color='blue', size=8, opacity=0.7): Customizes scatter points (blue color, size 8, semi-transparent).
4. **Updating Layout**
   * fig.update\_layout(...):
     + Adds titles and adjusts appearance:
       - title: "Actual vs Predicted Stock Prices"
       - xaxis\_title and yaxis\_title: Label axes for clarity.
       - template='plotly\_dark': Applies a dark theme.
       - width and height: Adjusts figure dimensions.
5. **Displaying the Plot**
   * fig.show(): Renders the final visualization.
   * plt.tight\_layout(): Ensures all components fit well in the layout.

### ****Purpose****

* **Visualization** of the model's performance by comparing **Actual vs. Predicted Stock Prices**.
* The **red dashed line** indicates the perfect prediction (ideal model), while the **blue scatter points** show the actual predictions.
* Helps identify how closely the model predictions align with the ground truth values.

**THANK YOU**