**Task Report Cos30018 Option B**

**B.3: Data Processing 2**

**Name: Le Bao Nguyen**

**Student Id: 104169837**

1. Importing libraries (From the task B.2):

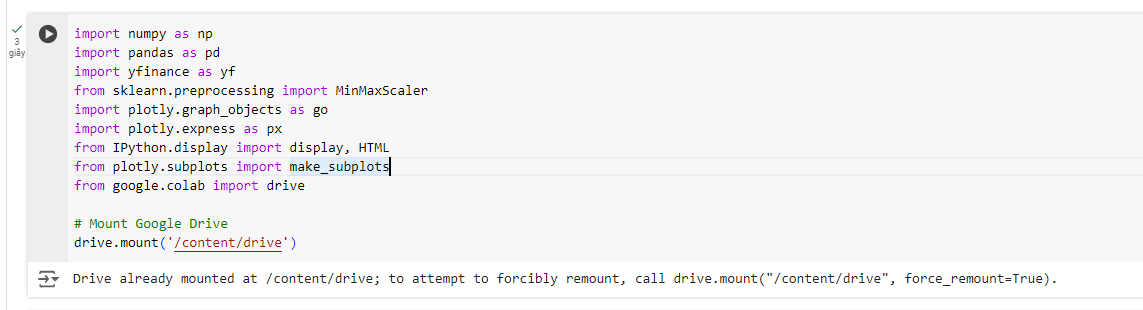


Figure 1: Importing libraries to run the code.

* The script import several libraries:
* "numpy": Supports numerical operations as well as arrays.
* "pandas": Utilized for analysis and data modification.
* "yfinance": Allows downloading historical market data from Yahoo Finance.
* "MinMaxScaler" from "sklearn.preprocessing": Applying a part of scikit-learn, data is scaled to a range (0, 1).
* "plotly.graph\_objects" and "plotly.express": Creating interactive plots
* "IPython.display": Displaying data frames as HTML in Jupyter notebooks.
* "make\_subplots" from "plotly.subplots": This function allows the creation of multi-panel figures with multiple subplots. It is particularly useful for arranging different types of plots, such as boxplots and volume charts, in a cohesive layout.
* "Drive" from "google.collab": Import drive from google collab to connect and save file to google drive.

1. Data loading and processing (From the task B.2):

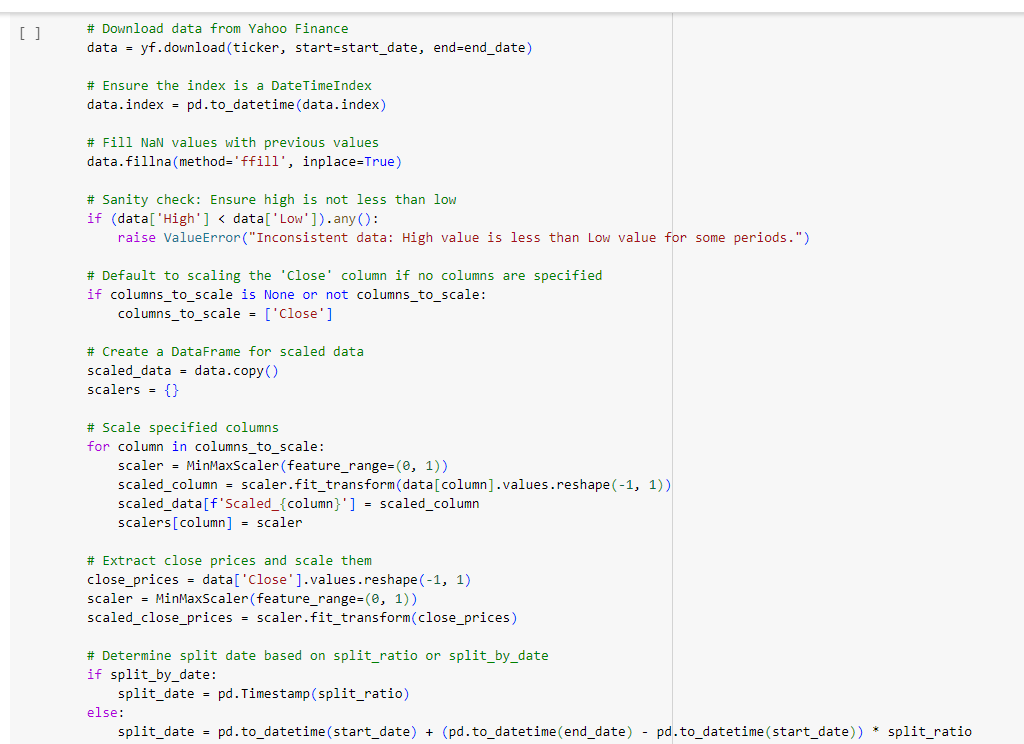


Figure 2: Loading and processing data (1).

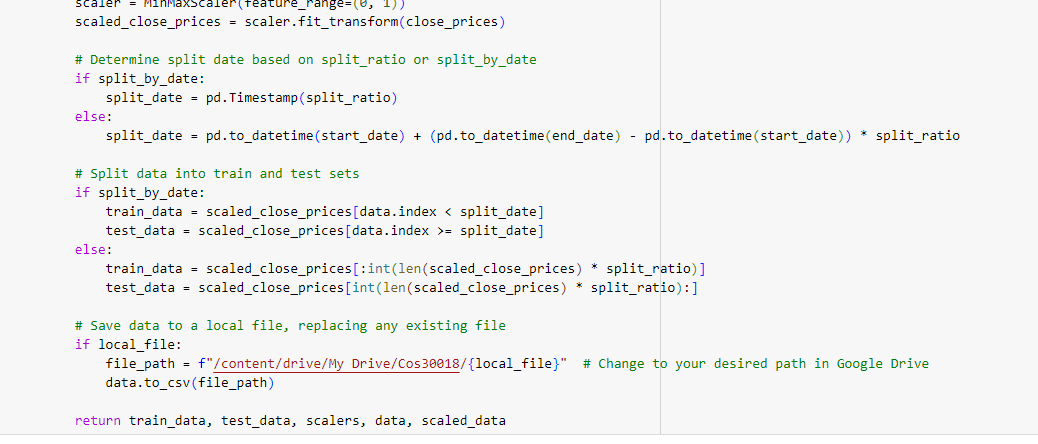


Figure 3: Loading and processing data (2).

* Using yfinance, the "load\_and\_processdata" function retrieves stock data for a specified ticker symbol and date range. The 'Close' prices are extracted and scaled using MinMaxScaler to a range between 0 and 1, after any missing values in the data are filled in using forward fill.
* Additionally, the function divides the data by a given ratio or date to create training and testing sets. The function saves the downloaded data as a CSV file if a local file name is given. The location that the file saved is in the gg drive.

1. Displaying data in a custom table (From the task B.2):

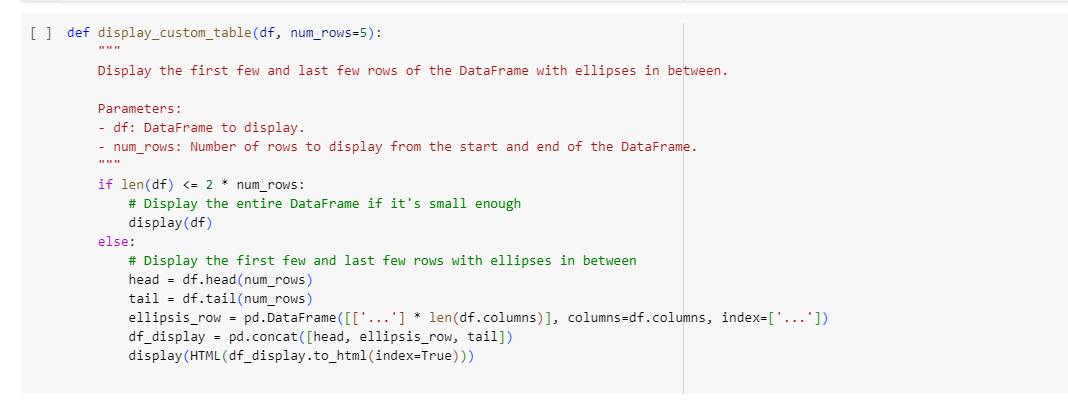


Figure 4: Displaying the data from csv file.

* If a data frame is too big to display completely, the "display\_custom\_table" method shows it in a truncated format. The data frame's beginning and last few rows are displayed, with an ellipsis row between them to represent the middle rows that were left out.
* This method guarantees that users won't be able to overflow the display when viewing a representative sample of huge data frame. The number of rows to display at the top and bottom of the data frame, as well as the data frame to be displayed, are the two arguments passed to the function.

1. Displaying the candlestick chart:

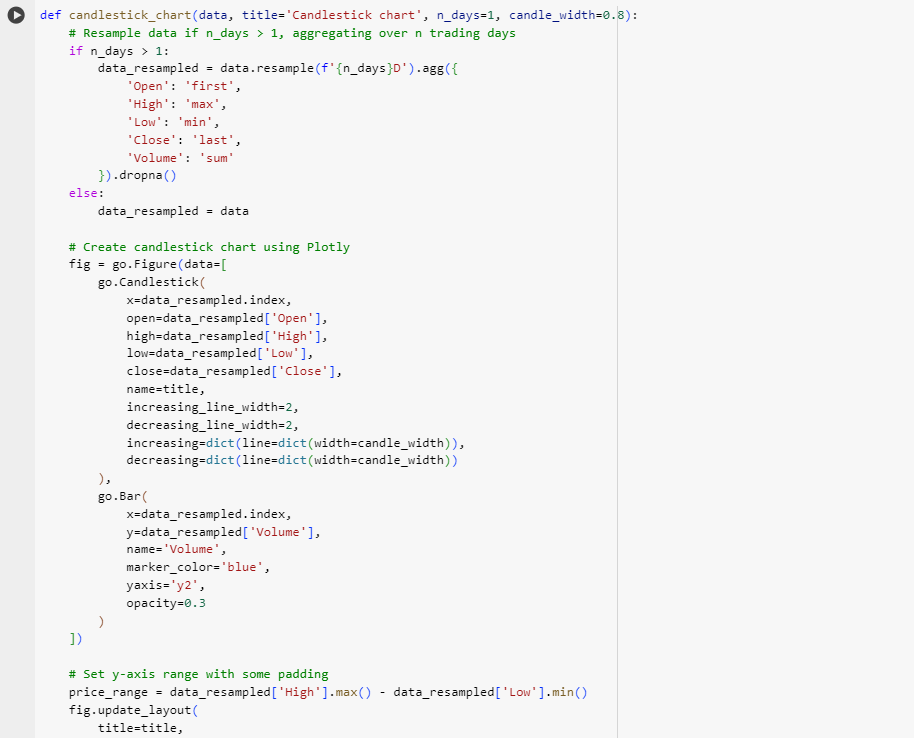


Figure 5: Code to print the candle stick chart (1).



Figure 6: Code to print the candle stick chart (2).

* Using Plotly, the "candlestick\_chart" function generates a candlestick chart using stock data. Resampling the data to aggregate over several trading days is an optional feature. The 'Open', 'High', 'Low', and 'Close' prices as well as the 'Volume' for the designated time are calculated.
* After that, the function creates the candlestick chart, which has selector buttons for various time periods and a range slider that allows you to choose the candle width. The function arguments consist of the width of the candlestick lines, the number of trading days each candlestick represents, the chart title, and the stock data DataFrame.
* "Data": Data frame containing stock data.
* "title": Title of the candlestick chart.
* "n\_days": Number of trading days each candlestick represents.
* "candle\_width": Width of the candlestick lines.

1. Displaying the box plot chart:

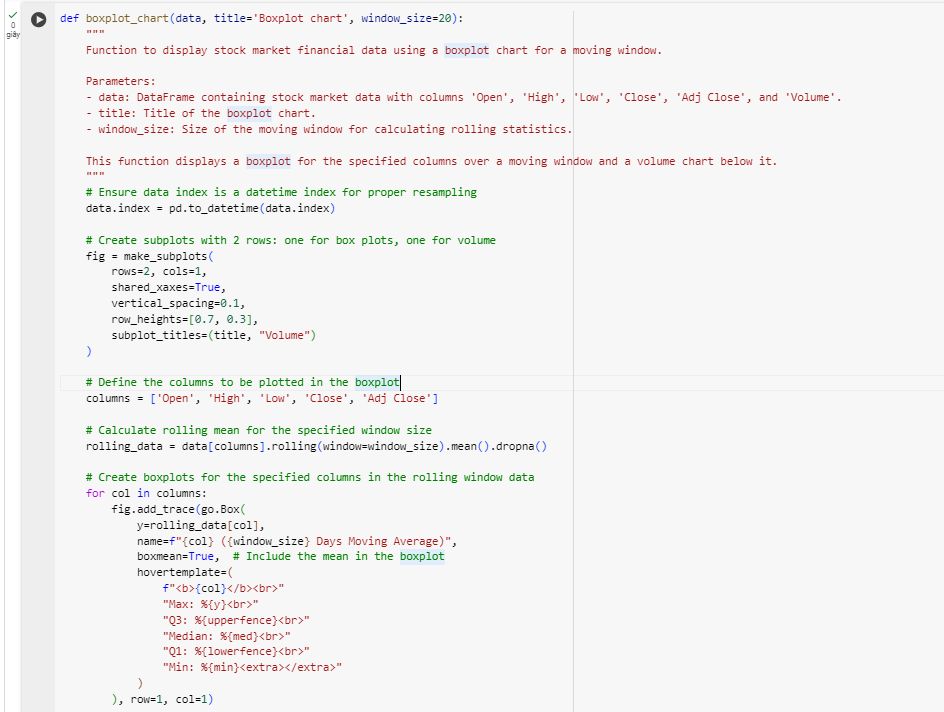


Figure 7: Code to print the box plot chart (1).

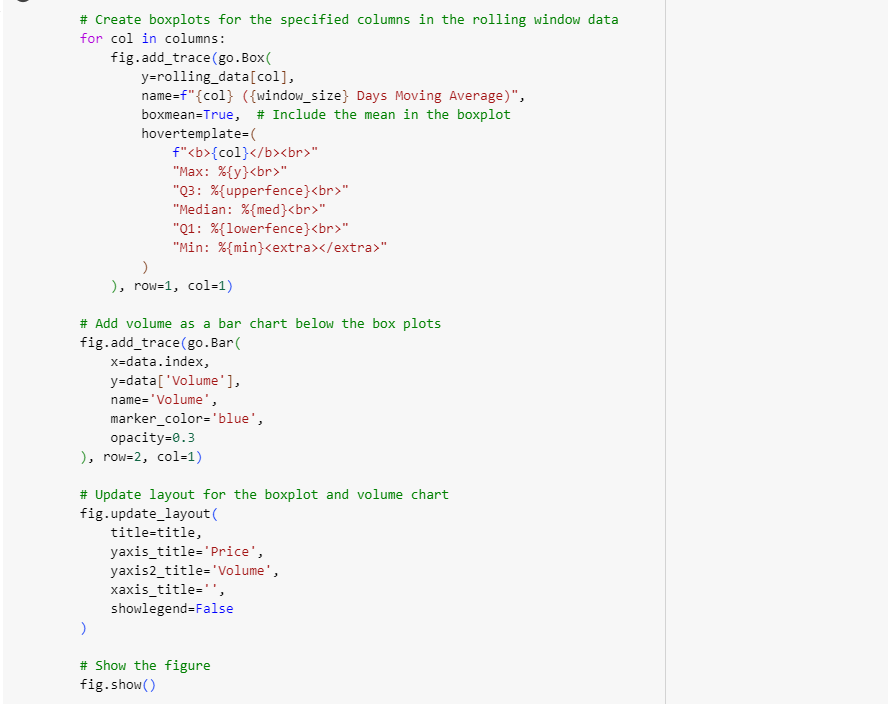


Figure 8: Code to print the box plot chart (2).

* In order to visualize the distribution of stock prices throughout the rolling periods, the "boxplot\_chart" function creates boxplot charts for specified rolling windows of stock data.
* Plotting these values as boxplots, it calculates the rolling mean for each price column ('Open,' High,' Low,' Close,' Adj Close') over the specified window widths.
* This function uses Plotly to create interactive visualizations and accepts the stock data DataFrame, the chart title, and a rolling window size as arguments.
* "data": DataFrame containing stock data.
* "title": Title of the boxplot chart.
* "window\_size": Sizes of the rolling windows for the boxplots.
* A bar chart is added to show the trade volume for each day below the boxplots. Given that changes in volume might reveal changes in investor views and possible price volatility, the volume data gives the price movements context.

1. Main script run:



Figure 9: The script to run the code and change the date.

* The defined functions are used as an example in the main block. It loads and analyzes Amazon.inc (AMZN) stock data, divides it based on dates and ratios, creates a personalized table, and generates boxplot and candlestick charts.

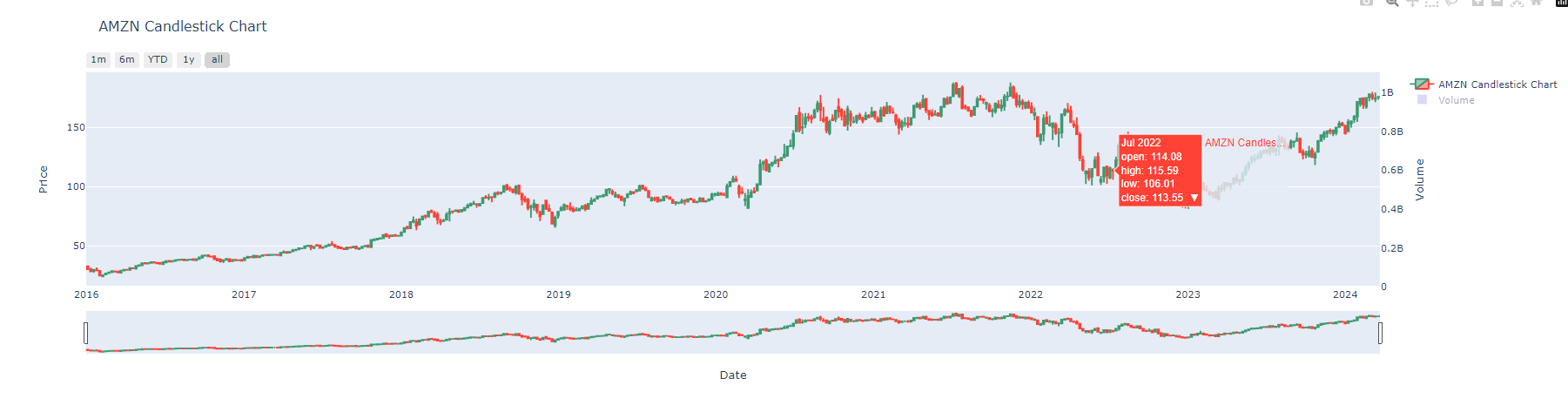


Figure 10: The AMZN candle stick chart.

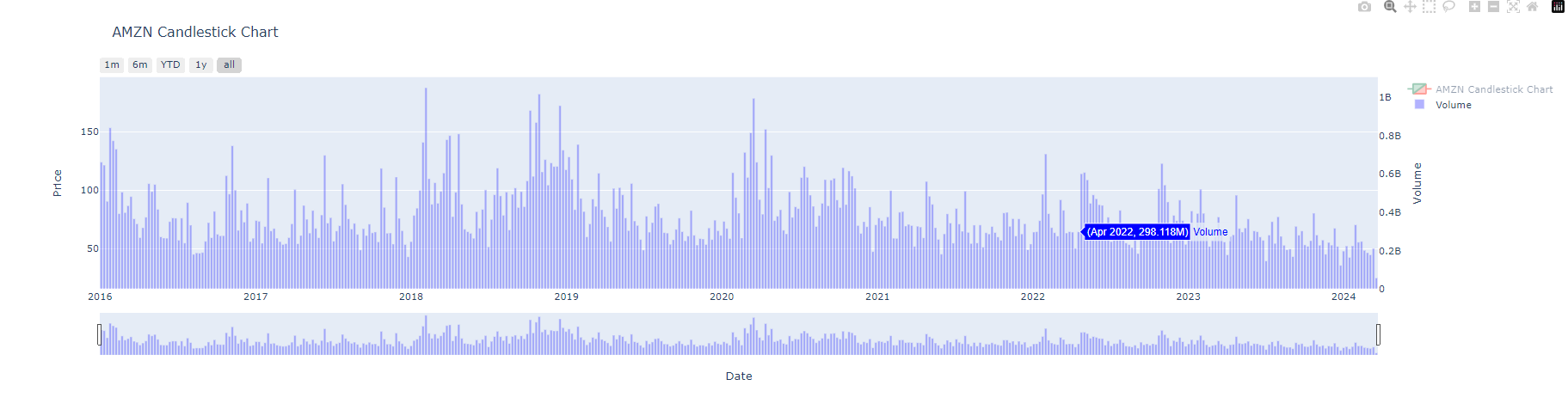


Figure 11: The AMZN volume chart.

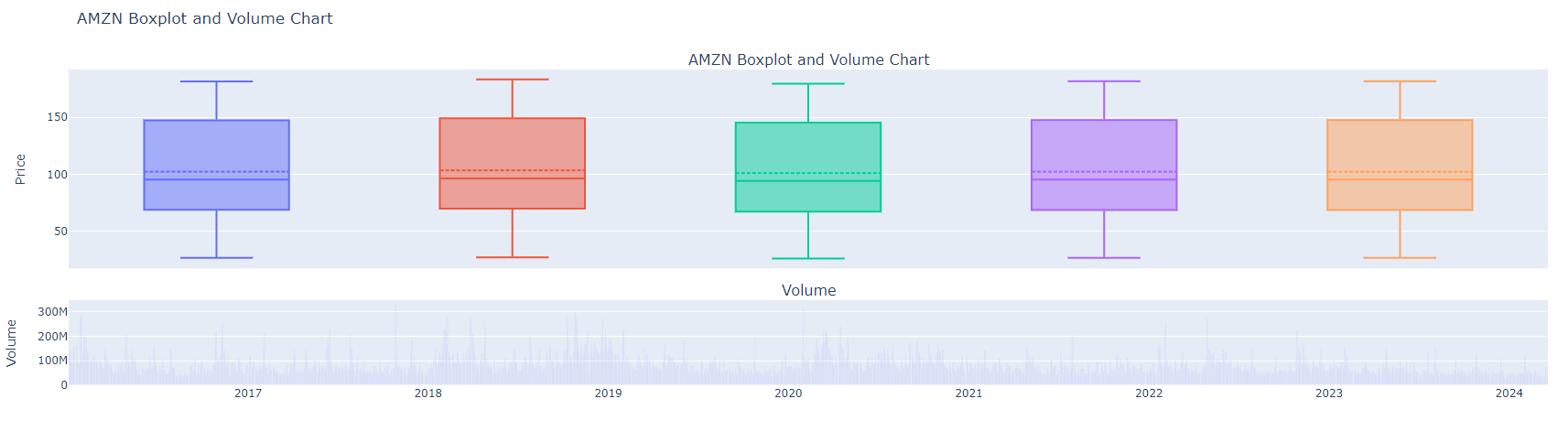


Figure 12: The AMZN box plot chart.

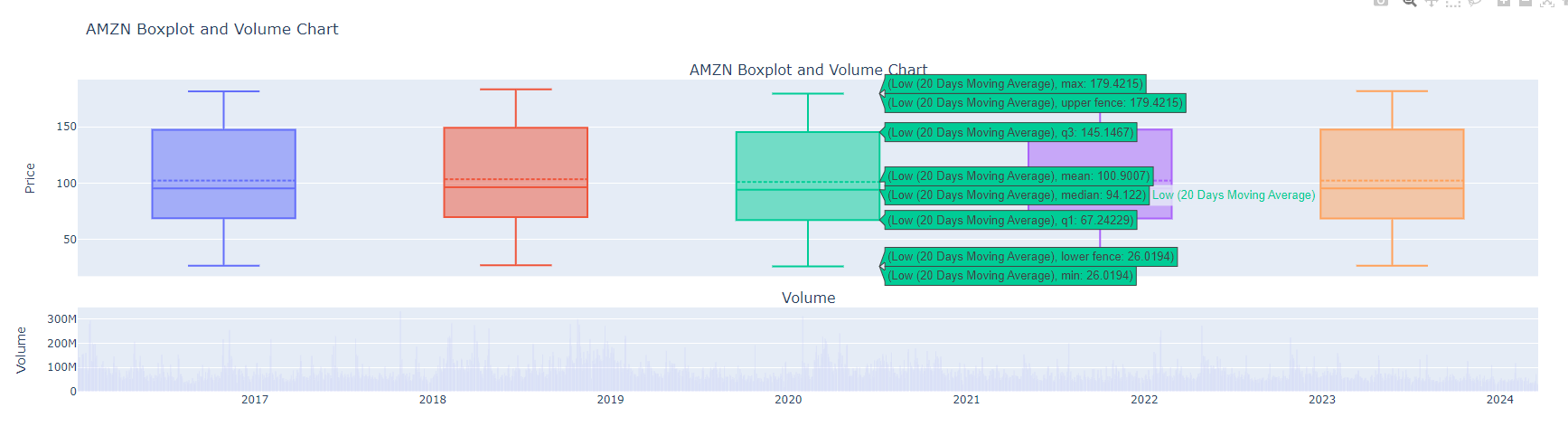


Figure 13: The AMZN box plot chart when hovering.



Figure 14: The AMZN volume bar of box plot chart when hovering.

1. Challenge faced:

* One of the main difficulties was making sure that the data was resampled accurately for the candlestick chart, which included knowing how to use the pandas aggregation algorithms appropriately. Additionally, a thorough understanding of pandas' rolling operations was required in order to set up the rolling windows for the boxplot display. Thorough investigation and testing were also necessary to properly configure Plotly's display parameters, such as range sliders and selector buttons. Also, the hover at the box plot still being duplicated at the end of median information.