Task B3:
Data Processing 2

Tutor: Dr Ru Jia

Name: Min Khant Aung
ID: 103833225

Report: Task B3 - Data Processing 2

Summary

In this task, I implemented two key data visualization techniques to represent stock market financial data using candlestick and boxplot charts. The objective was to enhance the visual representation of stock data to enable better insights and support prediction techniques.

Candlestick Chart Function

Code Explanation

The plot_candlestick_chart function was developed to visualize stock market data using candlestick charts. Candlestick charts are widely used in financial analysis as they provide comprehensive insights into the price movements within a specific time frame. Here's how the function is structured:

```
def candlestick_chart(data, company, n_days=1):
    """
    Plot a candlestick chart for the given stock market data.
    Each candlestick represents 'n_days' of trading data.

    Arguments:
    - data: DataFrame containing stock market data with columns like 'Open',
'High', 'Low', 'Close', and 'Volume'.
    - company: A string representing the stock ticker symbol (e.g., 'AAPL' for
Apple Inc.).
    - n_days: Integer specifying the number of trading days each candlestick
should represent (default is 1).
    """

    # If n_days is greater than 1, resample the data to group it into larger
time intervals
    if n_days > 1:
        # Resampling the data to aggregate based on 'n_days'
        data_resampled = data.resample(f'{n_days}D').agg({
            'Open': 'first',  # The first opening price in the period
            'High': 'max',    # The highest price in the period
            'Low': 'min',     # The lowest price in the period
            'Close': 'last',  # The last closing price in the period
            'Volume': 'sum'   # The total volume of stocks traded in the
period
        }).dropna()  # Remove any rows with NaN values that might have been
created during resampling
```

```
    else:
        # If n_days is 1, use the original data without resampling
        data_resampled = data

    # Plotting the candlestick chart using the resampled data
    mpf.plot(data_resampled, type='candle', title=f'{company} Candlestick
Chart', style='charles', volume=True)
```

Key Arguments and Parameters:

- **data**: The DataFrame containing stock market data.
- **company**: A string representing the stock ticker symbol.
- **n_days**: The number of trading days each candlestick should represent. Default is 1, meaning each candlestick represents a single day.

Challenges and Research:

The main challenge was to resample the data to allow each candlestick to represent multiple trading days (n_days). Research was conducted on how to aggregate the data using pandas' resample method and how to apply the mplfinance library to plot candlestick charts. This was particularly challenging as it required aggregating Open, High, Low, Close, and Volume data correctly to accurately represent the stock's behavior over the specified period.

Boxplot Chart Function

Code Explanation

The plot_boxplot function was created to visualize stock market data using boxplots. Boxplots are particularly useful for displaying the distribution of data, especially when analyzing data over a moving window of n consecutive trading days.

Here's how the function is structured (with appropriate comments):

```
def boxplot(data, company, column='Close', n_days=1):
    """
    Plot multiple boxplot charts for the given stock market data.
    Each boxplot shows the distribution of data over a moving window of
'n_days'.

    Arguments:
    - data: DataFrame containing stock market data.
    - company: A string representing the stock ticker symbol.
    - column: The column of the data to be visualized (default is 'Close'
price).
    - n_days: Integer specifying the number of consecutive trading days in the
moving window (default is 1).
    """

    # Calculate the rolling mean over a window of 'n_days' and drop any
resulting NaN values
    rolling_data = data[column].rolling(window=n_days).mean().dropna()

    # Create a list of rolling data segments to use for each boxplot
    boxplot_data = [rolling_data[i:i + n_days] for i in range(0,
len(rolling_data), n_days)]

    # Create a figure for the boxplot with a specific size
    plt.figure(figsize=(12, 6))

    # Set the title of the boxplot chart
    plt.title(f'{company} Boxplot Chart')

    # Generate the boxplot with various customization options
    plt.boxplot(boxplot_data, patch_artist=True, showmeans=True)

    # Label the x-axis with the rolling periods and y-axis with 'Closing
Price'
    plt.xlabel(f'Rolling {n_days}-Day Period')
    plt.ylabel('Closing Price')
```

```
    # Customize the x-axis labels to reflect the date ranges for each boxplot
    plt.xticks(ticks=range(1, len(boxplot_data) + 1), labels=[f'{i*n_days+1}-
{(i+1)*n_days}' for i in range(len(boxplot_data))])

    # Add a grid to the chart for better readability
    plt.grid(True)
```

Key Arguments and Parameters:

- **data**: The DataFrame containing stock market data.
- **company**: A string representing the stock ticker symbol.
- **column**: The column to be analyzed, defaulting to 'Close'.
- **n_days**: The number of consecutive trading days to consider in the moving window. Default is 1.

Challenges and Research:

The primary challenge involved managing the rolling window to accurately capture the distribution of closing prices over n_days. Research was conducted on how to use pandas' rolling and boxplot functions to visualize this distribution effectively. Another challenge was ensuring that the data was segmented correctly into intervals for the boxplot representation.

Visualization of Results

After developing these functions, the task involved visualizing the results using these methods in a coherent and meaningful way. The line chart was plotted to compare actual versus predicted stock prices. At the same time, the candlestick and boxplot charts were displayed for a comprehensive view of the stock's behavior over the specified periods.

Here's how the function is structured (with appropriate comments) :

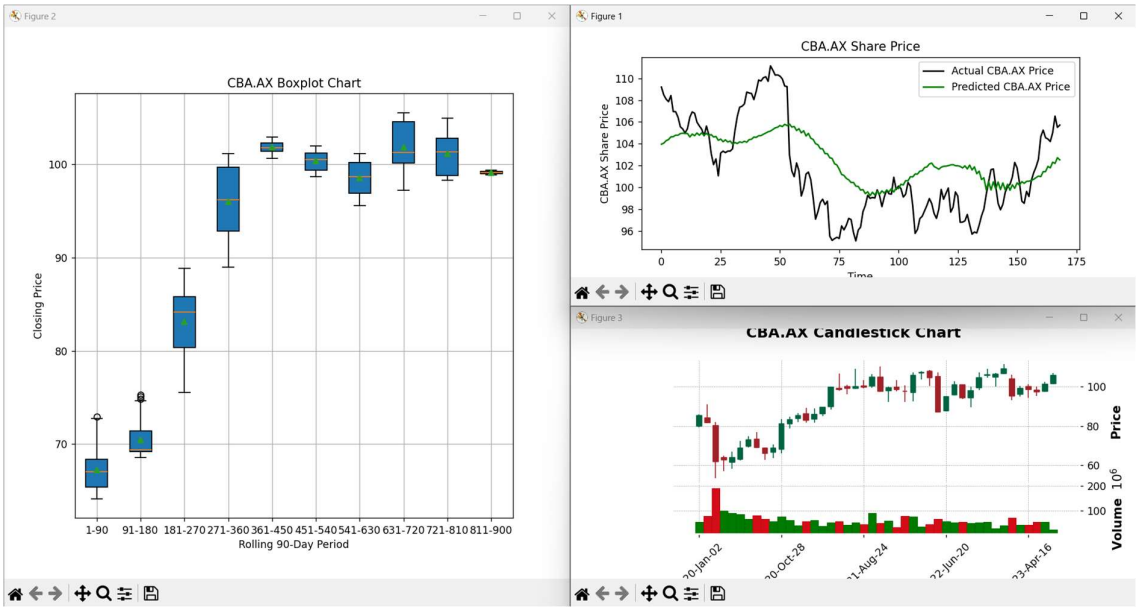
```
# Plot results: line chart
plt.plot(y_test_unscaled, color="black", label=f"Actual {COMPANY} Price")
plt.plot(predicted_prices, color="green", label=f"Predicted {COMPANY} Price")
plt.title(f"{COMPANY} Share Price")
plt.xlabel("Time")
plt.ylabel(f"{COMPANY} Share Price")
plt.legend()


# Visualization: boxplot charts
Boxplot_chart(data, company=COMPANY, n_days=90)
plt.show(block=False) #to continuously show candlestick chart
```

```
# Visualization: candlestick chart
candlestick_chart(data, company=COMPANY, n_days=30)
plt.show()
```

This code successfully integrates the three visualization techniques at the same moment.

The Result screenshot:



Conclusion

This task improved my ability to visualize and analyze stock market data using Python. By implementing candlestick and boxplot charts, I have added valuable tools to my data analysis toolkit, allowing for more nuanced interpretations of stock price movements over time. The process of coding these functions and integrating them with the existing project required significant research and problem-solving, particularly in managing data aggregation and visualization techniques.

References:

- mplfinance documentation

  https://github.com/matplotlib/mplfinance

- Pandas resample method

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.resample.html

- Boxplot visualization techniques

  https://www.dundas.com/Support/learning/documentation/data-visualizations/how-to/creating-a-box-plot