Task 3: Data Processing 2

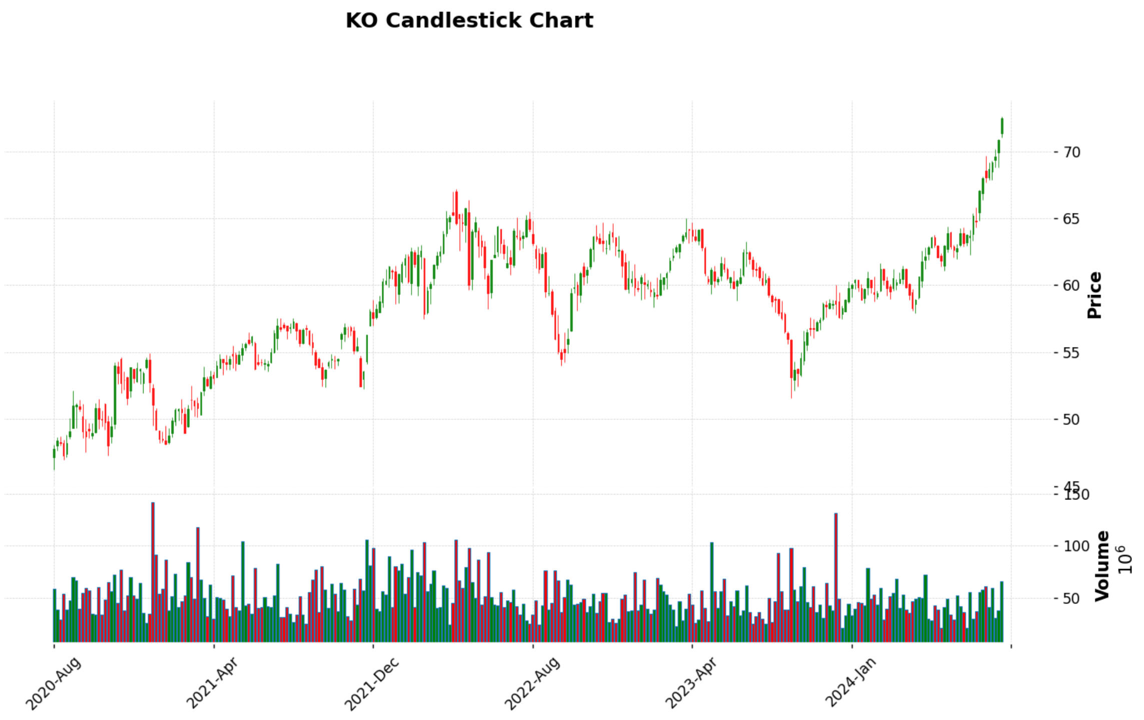
Muhammad Ali, 103960437, COS30018, 23/08/2024.

Requirements / Deliverables –

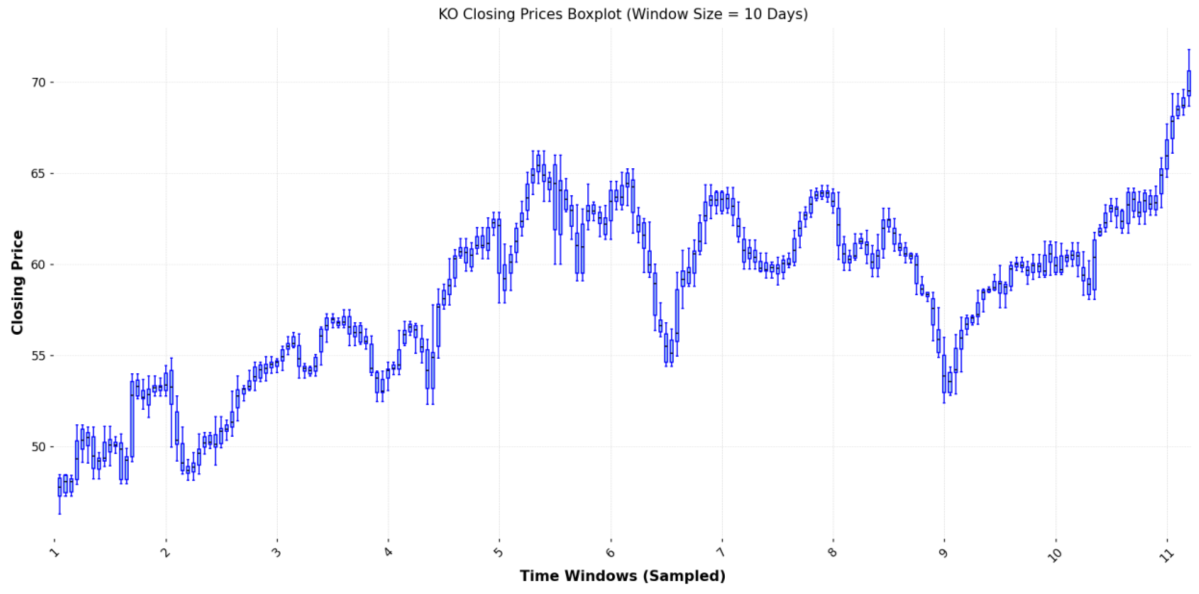
- ➢ Write a function to display stock market financial data using a candlestick chart. Include an option in the function to allow each candle stick to express the data of n trading days (n > 1).
- ➢ Write a function to display stock market financial data using boxplot chart.

Result –

Candlestick Chart –



Boxplot –



Candlestick Chart Source Code (*Also included in Repo*) –

```python
def plot_candlestick(data, n_days=1, ticker='Stock', save_plot=False):
    """
    Plots a candlestick chart using mplfinance with the option to aggregate
data over multiple trading days.

    The function enhances chart readability by adjusting styles, date
formatting, and gridlines.
    Interactive zoom is enabled by default when displayed.

    Parameters:
    - data (pd.DataFrame): DataFrame containing stock data with columns
['Open', 'High', 'Low', 'Close', 'Volume'].
    - n_days (int): Number of trading days each candlestick should
represent (e.g., 1 for daily, 5 for weekly).
    - ticker (str): The stock ticker symbol, used for labeling the plot.
    - save_plot (bool): If True, saves the plot as an image file;
otherwise, it displays the plot.

    Returns:
    - None
    """
    # Enable interactive mode
    plt.ioff() # TODO: Currently not working, make it display

    # Ensure data has the required columns
    required_columns = ['Open', 'High', 'Low', 'Close', 'Volume']
    if not all(column in data.columns for column in required_columns):
        raise ValueError(f"Data must contain columns: {required_columns}")

    # Resample data if n_days > 1 to aggregate data over the specified
number of days
    if n_days > 1:
        data = data.resample(f'{n_days}D').agg({
            'Open': 'first',
            'High': 'max',
            'Low': 'min',
            'Close': 'last',
            'Volume': 'sum'
        }).dropna()

    # Define custom style for improved readability
    style = mpf.make_mpf_style(
        base_mpf_style='charles',  # Clean and modern style
        marketcolors=mpf.make_marketcolors(
            up='green', down='red',  # Use green for up days and red for
down days
            wick='inherit',  # Use the same colors for wicks
            edge='inherit',  # Use the same colors for edges
            volume='in',  # Match volume color to price movement
        ),
        gridcolor='lightgray',  # Light gray gridlines to minimize
distraction
        gridstyle='--',  # Dashed gridlines
        facecolor='white',  # White background for simplicity
    )

    # Plotting the candlestick chart using mplfinance
    fig, axlist = mpf.plot(
        data,
        type='candle',  # Candlestick type plot
        style=style,  # Apply the custom style
        title=f'{ticker} Candlestick Chart',  # Title of the plot
        ylabel='Price',  # Label for the y-axis
        volume=True,  # Include volume in the plot
        ylabel_lower='Volume',  # Label for the volume axis
        datetime_format='%Y-%b',  # Format date as Year-Month (e.g., 2020-
Aug)
        xrotation=45,  # Rotate x-axis labels for better readability
        tight_layout=False,  # Turn off tight layout to prevent clipping
        figsize=(14, 8),  # Increase figure size for more space
        show_nontrading=False,  # Exclude non-trading days
        returnfig=True  # Return the figure and axes to adjust further if
needed
    )

    # Adjust the layout to avoid clipping
    fig.subplots_adjust(right=0.95, left=0.1, top=0.9, bottom=0.15)  #
Adjust margins to prevent clipping

    # Save the plot as an image file if save_plot is True
    if save_plot:
        fig.savefig(f'{ticker}_candlestick.png', bbox_inches='tight')  #
Ensure no clipping when saving
    else:
        plt.show()  # Display the plot with interactive mode enabled
```

The candlestick chart includes functionality to resample data over the specified 'n' trading days, it includes the relevant labels displaying the Price and Volume on the y axis and figures on the x axis. The plot can also be saved if specified in the function parameters by specifying 'save_plot' as True. The result is this:

Boxplot Chart Source Code (*Also included in Repo*) –

```
def plot_boxplot(data, window_size=20, step=5, ticker='Stock'):
    """
    Plots a boxplot chart for the closing prices of a stock using a moving
window of n consecutive trading days.
    Improves readability by adjusting the window size, step, and boxplot
appearance.

    Parameters:
    - data (pd.DataFrame): DataFrame containing stock data with at least a
'Close' column.
    - window_size (int): The size of the moving window in trading days.
    - step (int): Step size to reduce the number of boxplots shown.
    - ticker (str): The stock ticker symbol, used for labeling the plot.

    Returns:
    - None
    """
    # Ensure data has the required 'Close' column
    if 'Close' not in data.columns:
        raise ValueError("Data must contain 'Close' column for boxplot.")

    # Generate moving windows of closing prices
    windowed_data = [
```

```python
        data['Close'].iloc[i:i + window_size].values
        for i in range(0, len(data) - window_size + 1, step)
    ]

    # Plotting the boxplot
    plt.figure(figsize=(14, 7))
    plt.boxplot(
        windowed_data,
        patch_artist=True,  # Fill the boxes with color
        showfliers=False,  # Hide outliers to reduce clutter
        boxprops=dict(facecolor='lightblue', color='blue'),  # Box color
        whiskerprops=dict(color='blue'),  # Whisker color
        capprops=dict(color='blue'),  # Cap color
        medianprops=dict(color='black')  # Median line color
    )

    # Set plot title and labels
    plt.title(f'{ticker} Closing Prices Boxplot (Window Size =
{window_size} Days)')
    plt.xlabel('Time Windows (Sampled)')
    plt.ylabel('Closing Price')
    plt.grid(axis='y', linestyle='--', alpha=0.7)  # Light grid lines on y-
axis for reference

    # Reduce the number of x-ticks and rotate them for better readability
    plt.xticks(ticks=range(0, len(windowed_data), max(1, len(windowed_data)
// 10)), rotation=45)

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