Task C.3 – Data Processing 2

COS30018 – Intelligent Systems

Rahul Raju

Student ID: 105143065

1. Introduction

The original code did a great job of downloading data, training a neural network, and plotting a basic line graph comparing actual vs. predicted prices. But as I learned more, I realized that to truly understand stock market behaviour, I needed more than just a line chart. I needed visualizations that traders and analysts actually use — namely, candlestick charts and boxplots.

This report outlines the key enhancements I made to the original code, focusing on the implementation of candlestick and boxplot charts. I'll explain what I changed, why I changed it, and what I learned along the way — including the inevitable debugging struggles.

2. Key Changes Implemented

2.1 Added Candlestick Chart Functionality

The original v0.1 code only plotted a simple line graph. While useful, it doesn't show the intra-day price movements (Open, High, Low, Close) that are crucial for understanding market sentiment.

What I Added:

- Created a new function called plot_candlestick_chart().
- Integrated the mplfinance library, which is specifically designed for financial charts.
- Added the ability to group data into n-day candles (e.g., 1-day, 3-day, 5-day periods).

How It Works:

Instead of plotting just the closing price, each "candle" now visually represents four data points:

- Open: Price at the start of the period.
- High: Highest price during the period.
- Low: Lowest price during the period.
- Close: Price at the end of the period.

The real challenge was handling n-day candles. Since mplfinance expects daily data by default, I had to write logic to group every n rows together, then calculate the Open (first), High (max), Low (min), and Close (last) for each group. This involved using numpy.arange() to create numeric groupings since you can't do math directly on date indexes.

```
# Create a numeric index for grouping (0, 1, 2, 3, ...)
numeric_index = np.arange(len(data))

# Create a new column to group every n_days rows using the numeric index
data['group'] = numeric_index // n_days  # Integer division to create groups
```

2.2 Implemented Boxplot Charts for Moving Windows

The second major addition was the plot_boxplot_chart() function. While candlesticks show price action, boxplots reveal the distribution of prices over a moving window — perfect for spotting volatility or consolidation periods.

What I Added:

- Created a function to generate boxplots for n-day moving windows.
- Each boxplot shows the median, quartiles, and outliers for the selected column (usually 'Close') within each window.
- Added color-coding for visual appeal and better readability.

Why It's Useful:

For example, a 5-day boxplot window lets you see how much the stock price varied each week. Tight boxes mean low volatility; tall boxes with long whiskers mean high volatility. This is something a simple line chart can't convey.

```
# Create a column to identify which window each row belongs to
# We'll create groups of 'window_size' consecutive days using numeric index
data_copy['window_group'] = numeric_index // window_size

# Create a list to store data for each window
window_data = []
window_labels = []

# Group the data by window_group and create boxplot data for each window
grouped = data_copy.groupby('window_group')

for name, group in grouped:
    # Extract the values for the specified column
    values = group[column].values
    window_data.append(values)
```

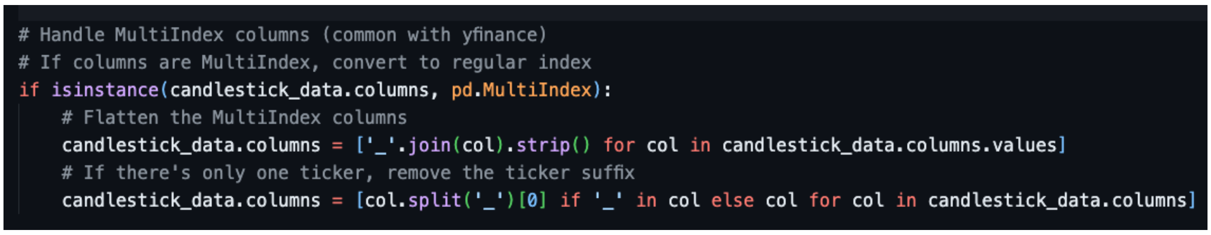
2.3 Data Cleaning for Robustness

One of the biggest headaches came from data type errors. The yfinance library sometimes returns data with MultiIndex columns or non-numeric values, which breaks mplfinance.

What I Fixed:

- Added code to flatten MultiIndex columns (common when downloading single stocks).
- Used pd.to_numeric(..., errors='coerce') to force columns into numeric types.
- Filled any remaining NaNs in the 'Volume' column with zeros.

This section taught me that real-world data is messy. You can't just assume your DataFrame is perfectly formatted — you need defensive programming.



```python
# Handle MultiIndex columns (common with yfinance)
# If columns are MultiIndex, convert to regular index
if isinstance(candlestick_data.columns, pd.MultiIndex):
    # Flatten the MultiIndex columns
    candlestick_data.columns = ['_'.join(col).strip() for col in candlestick_data.columns.values]
    # If there's only one ticker, remove the ticker suffix
    candlestick_data.columns = [col.split('_')[0] if '_' in col else col for col in candlestick_data.columns]
```

3. Challenges Faced & Lessons Learned

- Let's be honest — this wasn't smooth sailing. Here's what tripped me up and what I learned:
    - "TypeError: cannot perform floordiv with this index type: DatetimeArray"
    - The Problem: I tried to do data.index // n_days, thinking I could divide dates. Spoiler: you can't.
    - The Fix: Used np.arange(len(data)) to create a simple integer index [0, 1, 2, 3...] for grouping.
    - Lesson Learned: Not everything needs to be date-based. Sometimes a simple row count is more practical.
- "ValueError: Data for column 'Open' must be ALL float or int"
    - The Problem: mplfinance is strict — no NaNs or strings allowed.
    - The Fix: Added robust data cleaning: forward-fill, back-fill, convert to numeric, fill remaining NaNs.
    - Lesson Learned: Always clean your data before passing it to visualization libraries. Assume nothing.
- "KeyError: 'feature_scalers'"
    - The Problem: I had old saved .pkl files from a previous version of the code with a different dictionary structure.

    - The Fix: Deleted the old .pkl file in ./stock_data/ and let the program regenerate it.
    - Lesson Learned: When you change your data structure, old saved files become incompatible. Version control isn't just for code — it's for data too!
- Understanding LSTM return_sequences
    - Reading the provided TensorFlow and Machine Learning Mastery documentation helped me understand why we set return_sequences=True for stacked LSTMs. It's not just a random flag — it ensures the full sequence of hidden states is passed to the next layer, not just the final state. This was crucial for the model's ability to learn temporal patterns.

4. Reflection

This project was more than just adding a couple of charts. It was a deep dive into real-world data science challenges. I learned that:

- Visualization is storytelling. A candlestick chart tells a richer story than a line graph. It shows battles between buyers and sellers — something I now appreciate when looking at stock charts.
- Data cleaning is 50% of the job. I spent more time fixing data types and handling edge cases than writing the actual charting code. That's normal — and it's a skill I'll use in every future project.
- Debugging is a superpower. Every error message felt frustrating at first, but each one taught me something new about how pandas, numpy, and mplfinance work under the hood.
- Start simple, then extend. I could have gotten lost in complex features. Instead, I focused on making one candlestick chart work, then extended it to n-day candles. Small wins build confidence.

Most importantly, I learned that machine learning isn't just about the model. It's about the entire pipeline — from data acquisition and cleaning to training, evaluation, and visualization. A model is only as good as your ability to understand and communicate its results.

5. Conclusion

By enhancing the original stock_prediction.py script with candlestick and boxplot charts, I've transformed it from a basic prediction tool into a more comprehensive market analysis script. These visualizations don't just look professional — they provide real insights into price action and volatility that the original line chart couldn’t offer.

The journey was challenging, but incredibly rewarding. I now feel more confident working with financial data, debugging data type issues, and creating meaningful visualizations. This isn’t the end — it’s a foundation I can build on, whether I’m adding more technical indicators, experimenting with different models, or even building a simple trading dashboard.