Week 2 Attempt Summary

Introduction

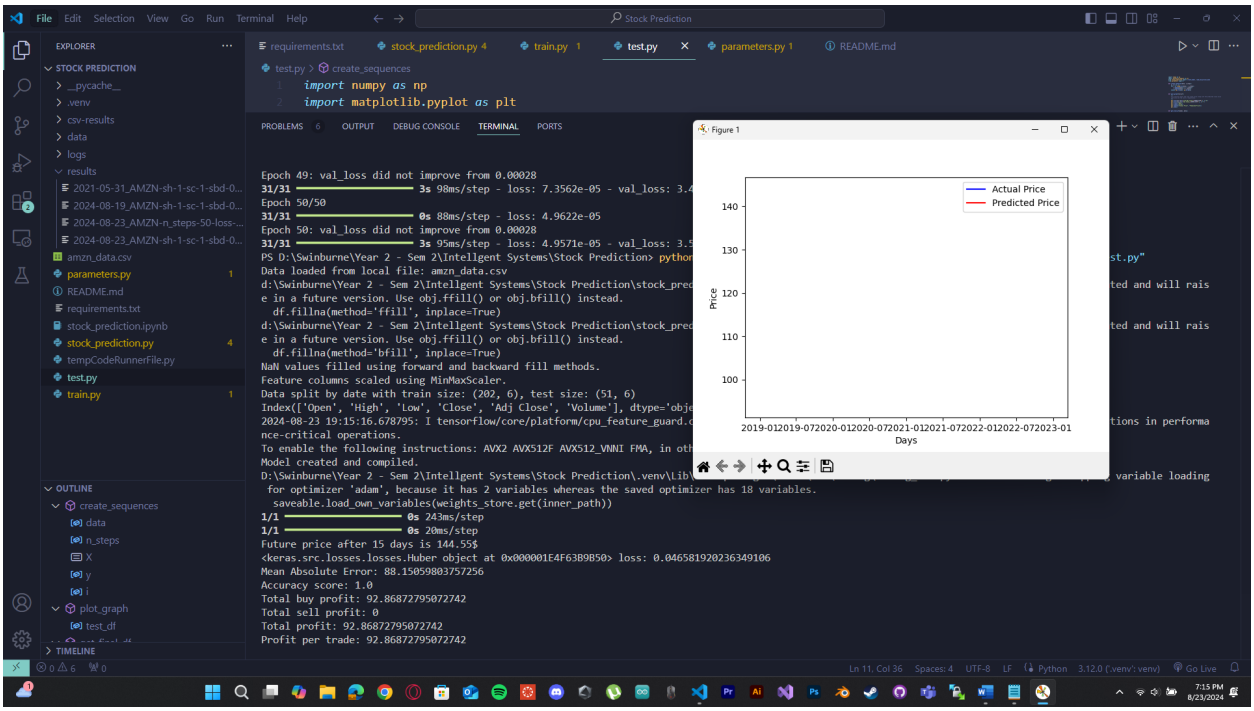
In this week we were advised to create a load and process data function to a dataset with multiple features, with the following functional requirements

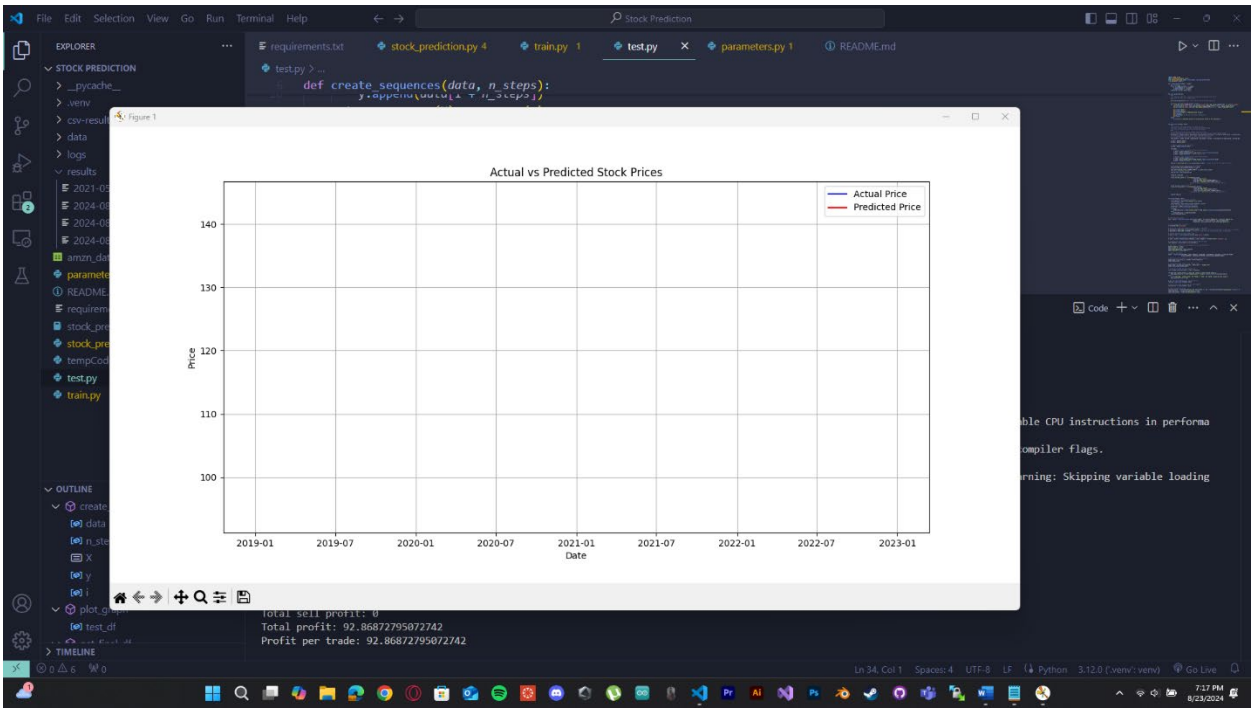
- Should be able to specify and start and end date for the whole dataset
- Deal with Nan values in the dataset
- Split train/test data according to a specified ratio (Date)
- Should be able to download the data to the local machine and store them to future use
- Scaling in the data structure

Load_and_process_data Function

```python
def load_and_process_data(ticker, start_date, end_date, handle_nan='fill', split_ratio=0.8, split_by_date=True, scale=True, save_local=True, local_file='data.csv'):
    # Load the data from Yahoo Finance or a local file
    if save_local and local_file and os.path.exists(local_file):
        df = pd.read_csv(local_file, index_col='Date', parse_dates=True)
        print(f"Data loaded from local file: {local_file}")
    else:
        df = yf.download(ticker, start=start_date, end=end_date)
        if save_local:
            df.to_csv(local_file)
            print(f"Data downloaded for ticker: {ticker}")
            print(f"Data saved locally at: {local_file}")

    # Handle NaN values
    if handle_nan == 'drop':
        df.dropna(inplace=True)
        print("NaN values dropped from dataset.")
    elif handle_nan == 'fill':
        df.fillna(method='ffill', inplace=True)
        df.fillna(method='bfill', inplace=True)
        print("NaN values filled using forward and backward fill methods.")

    # Scale the features
    if scale:
        scaler = MinMaxScaler()
        df[df.columns] = scaler.fit_transform(df[df.columns])
        column_scaler = {col: scaler for col in df.columns}
        print("Feature columns scaled using MinMaxScaler.")
    else:
        column_scaler = {}

    # Split data into train/test sets
    if split_by_date:
        train_df = df[:int(len(df) * split_ratio)]
        test_df = df[int(len(df) * split_ratio):]
    else:
        train_df = df.sample(frac=split_ratio, random_state=42)
        test_df = df.drop(train_df.index)

    print(f"Data split by {'date' if split_by_date else 'random sampling'} with train size: {train_df.shape}, test size: {test_df.shape}")

    return {
```

Even If I managed to get the code working to get an output in the terminal, I failed to get the data visualized in the plot





How the load and process data function work

This function is designed to load historical data, process it by handling Nan values, scaling and then splitting the data into training and testing datasets and then save the processed data locally for future usage.

Parameters

Ticker – The Ticker Symbol is used identify a specific stock market, in our case its amazon. So, AMZN

Start and End dates

Operations for Nan values – "drop" or "fill" – filling out Nan values from backwards or forward fill methods

Split ratio – the ratio of splitting the training and testing data, 0.8 refers to 80% training and 20% testing data

Boolean (Split by date) = Defaulted to True

Scale – preprocessed by minmaxscaler

While researching data splitting, I stumbled against some data splitting techniques

- **Simple Train-Test Split**: The most straightforward method, where the dataset is randomly divided into a training set and a test set, typically in an 80:20 or 70:30 ratio. This method is simple but can sometimes lead to variability in the performance of the model depending on how the data is split and this is the splitting method that I have used in this variation of the code base with the option to split data by date (80:20 ratio)

- **Stratified Split**: Particularly useful for classification tasks, stratified splitting ensures that the distribution of classes is consistent across both the training and test sets. This is crucial when dealing with

imbalanced datasets to ensure that the model learns from all classes proportionally.

- **K-Fold Cross-Validation**: In k-fold cross-validation, the dataset is divided into k subsets (folds). The model is trained on k-1 of these folds and validated on the remaining fold. This process is repeated k times with each fold serving as the validation set once. The results are then averaged to provide a more reliable estimate of model performance.

- **Holdout Validation**: Similar to the train-test split, but with an additional validation set. A part of the data is held out for validation purposes, which helps in tuning the model before the final evaluation on the test set.