SC4001 Neural Networks and Deep Learning

Gupta Tushar Sandeep

College of Computing and Data Science

Link to Repository:
https://github.com/29tushar/SC4001_NeuralNetworks/tree/main

Table of Contents:

Objectives

Welcome to the report of our neural networks assignment. The aim of this report is to showcase the methodology, experiments, results and evaluation of the iterative process taken to perform exploratory data analysis and creating an LSTM model based on the Apple Stock data for the period of January 1, 2014 to August 1, 2024 to predict the next-day closing price. Furthermore, we will also analyze the risk of a stock, based on its previous performance history.

Background

Nowadays, time series data are important objects to study and stock analysis is necessary to dive into a market. People usually need to extract patterns of the data to make decisions on weather forecasting, disease prevention and investment. However, time series data, especially stock, are not easy to extract due to unstable volatility caused by national and social policies. A feasible way to avoid potential risks and gain benefits is that some models can be trained based on historical data of stock and make predictions in a specific time interval. As machine learning is rapidly developing, neural network derives many transformation models which are proved to have better performance on prediction problems as they can discover complex and hidden patterns in data more efficiently compared to physical and empirical models

What is Time Series data?

Time Series data is a series of data points indexed in time order. Time series data is everywhere, so manipulating them is important for any data analyst or data scientist.

What is LSTM?

A Long Short-Term Memory network (LSTM) is a type of deep neural network that is designed to capture historical information of time series data and is suitable for predicting long-term nonlinear series. It consists of gates, including forget gate, input gate, and output gate, which can capture both long-term and short-term memory along the time steps and avoid gradient exploding and/or vanishing in standard RNNs. The LSTM unit also includes a cell that stores the accumulated information over arbitrary time intervals and operation of an LSTM unit can be summarized as a series of equations that involve activation functions, weight parameters, and bias. LSTM networks are well-suited for preprocessing, classification, and predictions based on time-series datasets.

Why is LSTM preferred over other prediction models?

It is preferred since it can capture historical information of time series, store knowledge of previous states, and mitigate problems of recurrent networks such as vanishing gradient and exploding gradient. LSTM can extract the nonlinear and dynamic characteristics of process data to achieve satisfactory prediction performance,

How does LSTM differ from other machine learning models?

LSTM differs from other machine learning models in its ability to store information over long periods of time and overcome the vanishing and exploding gradient problems in traditional RNN units.

Part 1: Exploratory Data Analysis

Question 1. a) Why is Apple Stock chosen?

Answer 1. a)
Apple has a long history of trading, and high-quality data and it is a globally recognized brand, leading to high public interest and sentiment that is regularly reflected in stock price movements. This makes it suitable for sentiment analysis-based models, where news and social media data impact the stock price.

Question 1. b) Why is there a 10 year period for the financial data?

Answer 1. b) A 10-year dataset includes various economic cycles. From 2014 to 2024, events like the 2018 market correction, the COVID-19 pandemic crash in 2020, and subsequent recovery are captured. A 5-year dataset captures fewer market cycles and might overemphasize specific events, leading to overfitting or bias. A 15-year dataset reflects outdated market structures and trading practices which could be irrelevant or misleading for current predictions.

Question 1. c) What was the change in price of the stock over time?

Answer 1. c) The closing price is the last price at which the stock is traded during the regular trading day. A stock's closing price is the standard benchmark used by investors to track its performance over time. We have created a plot to showcase the variation in closing price of the Apple Stock with time. There have been periods of high volatility, especially around 2018 and 2020, where the stock price fluctuated more rapidly. The stock reached its highest point in 2024. Low Points and experienced a low point around 2016. Refer to Figure 1 in the Appendix.

Question 1. d) What was the volume of sales over time?

Answer 1. d) Volume is the amount of an asset or security that changes hands over some period of time, often over the course of a day. For instance, the stock trading volume would refer to the number of shares of security traded between its daily open and close. Trading volume, and changes to volume over the course of time, are important inputs for technical traders. The volume of Apple stock trading has decreased over the period shown. We can see a significant drop from the high volumes in 2016 to a more stable, lower volume in recent years. These spikes might correspond to news events, product launches, or other factors that increased investor interest and trading activity. Refer to Figure 2 in the Appendix.

Question 1. e) What was the daily return of the stock on average?

Answer: 1. e) We are now going to analyze the risk of the stock. In order to do so we'll need to take a closer look at the daily changes of the stock, and not just its absolute value. The chart shows a significant amount of volatility in Apple's daily returns. The returns fluctuate both positively and negatively, indicating periods of both gains and losses for the stock. Refer to Figure 3 in the Appendix.

Question 1. f) What was the moving average of the various stocks?

Answer: 1. f) The moving average (MA) is a simple technical analysis tool that smooths out price data by creating a constantly updated average price. The average is taken over a specific period of time, like 10 days, 20 minutes, 30 weeks, or any time period the trader chooses. The moving averages smooth out the price fluctuations and provide insights into the stock's trend and momentum. Refer to Figure 4 in the Appendix.

Question 1. g) How much value do we put at risk by investing in a particular stock?

Answer 1. g) There are ways we can quantify risk, one of the most basic ways using the information we've gathered on daily percentage returns is by comparing the expected return with the standard deviation of the daily returns. Apple's position on the chart suggests that it offers a relatively moderate level of risk compared to its expected return. Refer to Figure 5 in the Appendix.

There are more exploratory data analysis plots plotted for understanding the data. Please refer to Figures 5 - 12 in the Appendix.

Part 2: Stationary Tests

Before implementing an LSTM model for stock price prediction, we perform these statistical tests to understand the underlying characteristics of the time series data. This is crucial because:

- **a)** LSTMs can handle non-stationary data better than traditional time series models. However, stationary data often leads to more stable training and better predictions and it helps in deciding preprocessing steps
- **b)** Test results influence how we transform our data.

The stationarity tests are extremely essential when preparing a time series dataset for training an LSTM (Long Short-Term Memory) model, particularly when predicting stock prices like Apple's closing price. LSTM models, which are a type of recurrent neural network (RNN), are highly sensitive to patterns in data. They are designed to capture dependencies and trends over time, but they assume the input data is stationary. Stationarity means that the statistical properties of the series, such as the mean, variance, and autocorrelation, do not change over time.

1. Augmented Dickey-Fuller (ADF) Test:
The ADF test is used to check for the presence of a unit root in the time series data, which would indicate that the series is non-stationary. Non-stationary data has properties (like mean and variance) that change over time, making it difficult to predict future values accurately. Stationary data, on the other hand, has constant properties over time and is more predictable. If the series is non-stationary, it can be differenced or transformed to achieve stationarity. The ADF test is crucial for determining whether we need to preprocess the data (e.g., by differencing) before feeding it into an LSTM model.

2. Kwiatkowski–Phillips–Schmidt–Shin (KPSS) Test:
The KPSS test is another method for testing stationarity but with a different hypothesis. It tests the null hypothesis that the data is stationary. If the p-value is greater than 0.05, we can conclude that the series is stationary. If it's less than 0.05, the series is likely non-stationary. The KPSS test is useful for confirming the findings from the ADF test.

3. Ljung-Box Test:
The Ljung-Box test is performed to check for autocorrelation in the residuals (errors) of the time series. Autocorrelation refers to the correlation of a time series with its past values, which could indicate that the data contains patterns that have not yet been captured. The Ljung-Box test helps us assess whether the residuals from the time series are independent (random) or not. If the residuals are not independent (i.e., if the p-value is small), it suggests that the model may not have captured all the temporal patterns and that further preprocessing or model adjustments may be required.

Original Series Test

1. Augmented Dickey-Fuller (ADF) Test:

a) **Interpretation**: The high p-value (0.9828 >> 0.05) which is a comparatively strong value. The null hypothesis (that the series has a unit root and is non-stationary) cannot be rejected, which means the data is non-stationary.
b) **Implication**: The original Apple stock prices follow a random walk, which is typical for financial time series.

2. Kwiatkowski–Phillips–Schmidt–Shin (KPSS) Test:

a) **Interpretation:** Very low p-value (0.01 < 0.05) confirms that null hypothesis (that the series is stationary) has to be rejected and it is non stationary.
b) **Implication:** The series shows significant trend and possibly seasonal components.

3. Ljung-Box Test:

a) **Interpretation**: Extremely low p-value indicates strong serial correlation.
b) **Implication**: Strong temporal dependencies exist in the data.
c) **Impact on LSTM**: LSTM should be used since it can capture these dependencies.

Differenced Series Test

1. **Augmented Dickey-Fuller (ADF) Test**:
   a) **Interpretation**: Very low p-value indicates stationarity.
   b) **Implication**: First-order differencing successfully removed the trend.

2. **Kwiatkowski–Phillips–Schmidt–Shin (KPSS) Test**:
   a) **Interpretation**: Higher p-value (0.1 > 0.05) confirms stationarity.
   b) **Implication**: No significant trend remains in differenced data.

3. **Ljung-Box Test**:
   a) **Interpretation**: Still shows temporal dependencies.
   b) **Implication**: Serial correlation persists even in differenced data.

Part 3: Data Pre-Processing

Data Collection

The financial data is retrieved using the yfinance library, specifically for Apple Inc. (AAPL) stock:

a) **Time period:** January 1, 2014 to August 1, 2024
b) **Interval:** Daily data points except the weekends when the market is closed.
c) **Retrieved features:** Open, High, Low, Close, Adjusted Close, and Volume

Column Name	Column Description
Open:	Opening price for the trading day.
High:	Highest price during the trading day.
Low:	Lowest price during the trading day.
Close:	Closing price for the trading day.
Adj Close:	Adjusted closing price (corrected for stock splits and dividends).
Volume:	Number of shares traded.

Table 1: *Column descriptions in retrieved dataset*

Data Preparation

The prepare_dataframe_for_lstm function implements the following steps:

1. *Deep Copy Creation:* Creates an isolated copy of the dataframe to prevent data leakage in order to reserve original data integrity
2. *Feature Scaling:* Using MinMaxScaler, the Close prices are scaled to the range [−1,1]. Scaling ensures that the data is normalized, which is crucial for gradient-based learning and avoiding issues with varying magnitudes of values. The scaler object (scaler) is stored for later use to reverse the scaling during prediction evaluation.
3. *Time Series Feature Engineering:* Creates a sliding window of size n_steps (lookback period = 7 days) and generates lagged features: Close(t-1) to Close(t-7). Each of these feature represents the closing price from previous time steps

Data Structuring for LSTM Model Input

1. The processed DataFrame is converted to a NumPy array for efficient numerical computations and the The array is split into features X and the target y where:
    a) X: Lagged Close values (Close(t-1), ..., Close(t-n))
    b) y: Current Close value (Close(t))

2. Input features “X” are reshaped to (samples, time_steps, features) and the target values “y” are reshaped to (samples, 1)

Dataset Splitting

1. The dataset is split into:
    a) Training Set (85%): Used to optimize model weights.
    b) Validation Set (10%): Used for hyperparameter tuning and overfitting checks.
    c) Test Set (5%): Used for final evaluation of the model.
2. TensorDataset objects are created for training, validation, and test sets. This facilitates efficient batch processing and ensures the temporal order of data is maintained.

Inverse Scaling of Predictions

1. The inverse_transform_predictions function:
    a) Converts scaled predictions back to original price scale
    b) Reshapes predictions if necessary
    c) And, applies inverse transformation using the same scaler used for initial scaling

Part 4: Model Training & Evaluation - LSTM

Model Definition:

1. The LSTMStockPredictor class is defined, which inherits from nn.Module (PyTorch's base class for neural network models).
2. The constructor __init__ method sets up the LSTM and a fully connected (FC) layer
3. The LSTM layer takes in the input size (1, as we have a single feature), hidden size and number of LSTM layers
4. The dropout parameter adds dropout regularization to the LSTM layers to prevent overfitting.
5. The FC layer takes the output from the last LSTM layer and produces a single output value (the predicted stock price). This layer learns to map the final hidden state of the LSTM to the target stock price.
6. The forward method defines the forward pass of the model, where the LSTM layer processes the input sequence, and the FC layer produces the final output. The last hidden state (corresponding to the final time step) is taken to pass through the FC layer.

Evaluation Function:

1. The evaluate function is responsible for evaluating the model's performance on a given dataset (validation or test set).
2. It iterates through the data loader, computes the loss, and collects the predictions and ground truth targets.
3. The predictions and targets are then transformed back to the original scale using the inverse_transform_predictions function, which applies the inverse transformation using

the same scaler that was used for data preprocessing. This is applied since the model was trained on scaled data, and the metrics on the original stock price scale are to be displayed.

4. Various evaluation metrics are computed, such as RMSE, MAE, Explained Variance, and R-Squared, and the results are returned as a dictionary.
5. The function returns a dictionary containing the overall loss and the rest of the metrics.

Training Function:

1. The train_epoch function defines the training loop for a single epoch.
2. It iterates through the training data loader, computes the loss, and backpropagates the gradients, and updates the model parameters using the Adam optimizer
3. Gradient clipping is applied to prevent exploding gradients.
4. The function returns the average loss for the current epoch being calculated.

Hyperparameter Search and Training Epoch Loop

1. This part of the LSTM model does a grid search to find the best hyperparameters for the model.
2. The hyperparameters being searched are:
    a) Batch size amongst [16, 32, 64]
    b) Number of LSTM layers (depth) amongst [2, 3, 4]
    c) LSTM hidden size (width) amongst [32, 64, 128, 256]
3. Dropout=0.2 randomly drops neurons during training and makes the model more robust.
4. The training loop iterates through the different hyperparameter configurations, training the model for a fixed number of epochs (100).
5. During training, the validation loss, RMSE, and R-Squared are tracked, and the best model is saved based on these metrics.
6. Early stopping is implemented to prevent overfitting, with a patience of 5 epochs. This is required in order to avoid overfitting and find the optimal number of training epochs.

Final Evaluation

1. After the training loop, the best model (based on the validation performance) is evaluated on the test set, and the test metrics are printed.
2. Moreover, the best hyperparameter configurations are printed to display the optimal model architecture for this specific task and dataset.

Our best model has the following metrics:

a) Dropout: 0.1, Learning Rate: 0.001, Batch Size: 64
b) Depth (Number of Layers): 4, Width (Hidden Size): 256
c) Validation Loss: 0.0010, Test Loss: 0.0057
d) RMSE: 8.204, MAE: 6.037, Explained Variance: 0.8939, R-Squared: 0.823

Please refer to Table 2 in the Appendix for the outputs of various hyperparameter combinations.

Part 5: Predictions

predict_next_n_days(model, input_data, n_steps) function:

1. This function is used to generate multi-step ahead predictions using the trained LSTM model and takes three parameters:
    a) model: The trained LSTM model to be used for making predictions.
    b) input_data: The input data (features) for which the predictions need to be made.
    c) n_steps: The number of future time steps to predict.
2. Furthermore:
    a) It sets the model to evaluation mode using model.eval(), which disables dropout and batch normalization layers during inference.
    b) It initializes an empty list called predictions to store the predicted values.
    c) It creates a copy of the input data using current_data = input_data.clone() to avoid modifying the original input.
    d) It enters a loop that iterates n_steps times:
    (i) It appends this prediction to the predictions list.
    (ii) Then, it updates the input data by shifting the sequence one step to the left and adding the latest prediction to the end. This allows the model to use its own previous predictions as input for the next step.
3. The predicted value is converted to a NumPy array and appended to the predictions list.
4. The input data is then updated by shifting the sequence one step to the left and appending the latest prediction to the end.
5. Finally, the function returns the predictions array, which contains the multi-step ahead predictions.

avg_error_percentage(y_actual, y_predicted)

This function calculates the average percentage error between the actual stock prices and the prices predicted by the model.

a) Firstly, it calculates the absolute difference between the actual and predicted prices for each data point.
b) Secondly, it converts these absolute differences into percentage errors by dividing them by the actual prices and multiplying by 100.
c) Lastly, it takes the average of all these percentage errors and rounds the result to 2 decimal places. The average percentage error is a useful metric because it gives us a sense of how close the model's predictions are to the true stock prices, on average. A lower percentage error indicates the model is making more accurate predictions.

time_series_analysis(x_datas, y_datas, dates, model, scaler, forecast_steps=1)
This function is responsible for visualizing the model's performance on the training, validation, and test datasets.

a) It creates a figure with 3 subplots, one for each of the training, validation, and test sets.
b) It uses the trained model to make predictions, either one step ahead (if forecast_steps=1) or multiple steps ahead (if forecast_steps>1).
c) It plots the actual and predicted prices on the subplot, using different colors to distinguish them for the various train, val and test dataset.

Output:
Our average error percentages are mentioned below and Figure 15 displays the respective plots:

a) Training dataset: **2.25%**
b) Validation dataset: **1.83%**
c) Test dataset: **2.06%**

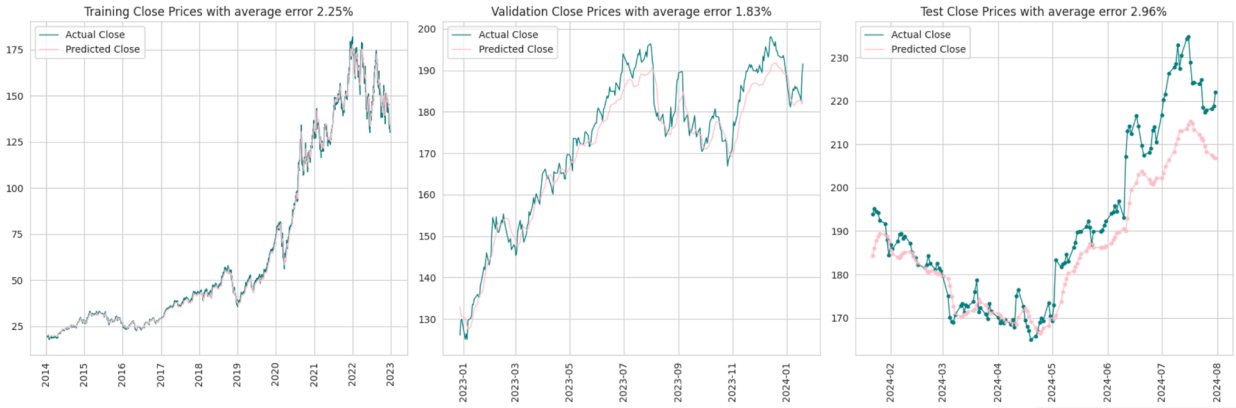


Figure 15. *Average Percentage Error Plots*

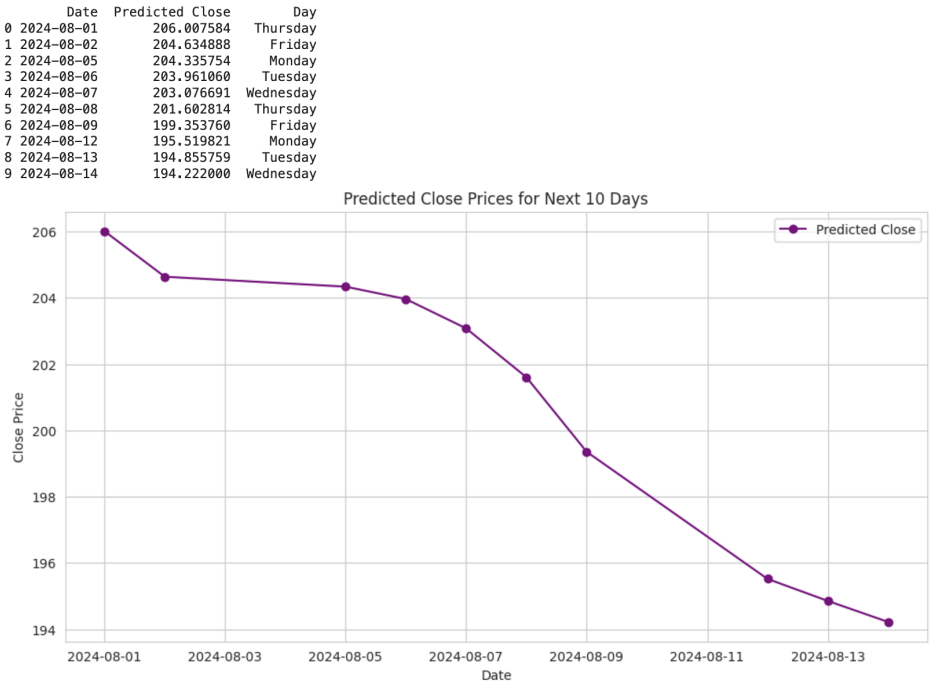
.predict_future(y_data, dates, model, scaler, num_future_days)

This function generates predictions for a number of future days, beyond the original test dataset.:

a) Firstly, it sets the model to evaluation mode, just like the predict_next_n_days function.
b) Then, it initializes an empty list called future_preds to store the predicted future close prices.
c) It takes the last 7 close prices from the test set data and uses these as the starting point for the future predictions.
d) It then enters a loop that runs for the desired number of future days:
   (i) Inside the loop, it uses the model to make a single prediction for the next day's close price, based on the last 7 days of data (including any previous predictions).
   (ii) It applies the inverse transformation to the predicted value to get the actual close price and appends this predicted close price to the future_preds list.
   (iii) It updates the last 7 days of data by shifting the sequence one step to the left and adding the latest prediction.

e) The 'B' frequency stands for business days in pandas. This ensures that the generated dates only include working days, typically Monday through Friday, excluding weekends since the stock markets operate on weekdays only.

We have plotted the chart to display the predicted close prices of the next 10 business days that is from August 1 - August 14.



	Date	Predicted Close	Day
0	2024-08-01	206.007584	Thursday
1	2024-08-02	204.634888	Friday
2	2024-08-05	204.335754	Monday
3	2024-08-06	203.961060	Tuesday
4	2024-08-07	203.076691	Wednesday
5	2024-08-08	201.602814	Thursday
6	2024-08-09	199.353760	Friday
7	2024-08-12	195.519821	Monday
8	2024-08-13	194.855759	Tuesday
9	2024-08-14	194.222000	Wednesday

Figure 16. *Predicted Close Prices for next 10 days*

References:

1. Zijie Zhong, Difei Wu, Wenxuan Mai. Stock Prediction Based on ARIMA Model and GRU Model. Academic Journal of Computing & Information Science (2023), Vol. 6, Issue 7: 114-123. https://doi.org/10.25236/AJCIS.2023.060715.
2. Cerqueira, V., & Roque, L. (2021). *Deep Learning for Time Series Cookbook: Use PyTorch and Python Recipes for Forecasting, Classification, and Anomaly Detection.* Packt Publishing.
3. https://www.sciencedirect.com/topics/computer-science/long-short-term-memory-network#:~:text=A%20Long%20Short%2DTerm%20Memory,that%20occur%20in%20standard%20RNNs.
4. https://www.machinelearningplus.com/time-series/augmented-dickey-fuller-test/
5. https://www.machinelearningplus.com/time-series/kpss-test-for-stationarity/

Appendix:

Why Test Loss is Calculated Instead of Test Accuracy

In the context of a regression problem like stock price prediction, the model is tasked with predicting a continuous numerical value (the stock price) rather than a discrete class label. As such, the performance metric that is most suitable is the loss function, which quantifies the difference between the predicted and actual values.

The most common loss function for regression problems is Mean Squared Error (MSE) or Root Mean Squared Error (RMSE). These metrics capture the overall magnitude of the errors, which is more informative than simply counting the number of predictions that fall within a certain range. Accuracy is more appropriate for classification tasks where the goal is to predict the correct class label. In a regression setting, accuracy does note provide a meaningful evaluation of the model's performance, as it depends quite a lot on how the "correct" predictions are defined (e.g., rounding to the nearest integer, using a fixed tolerance, etc.).

Tables:

Batch Size	Depth	Width	Test Loss	RMSE	MAE	Explained Variance	R-Squared
16	2	32	0.0152	13.371800422668500	11.180999755859400	0.854	0.5298
16	2	64	0.0146	13.094099998474100	10.618399620056200	0.839	0.5491
16	2	128	0.0037	6.6269001960754400	5.084499835968020	0.8881	0.8845
16	2	256	0.0034	6.282700061798100	4.993199825286870	0.8974	0.8962
16	3	32	0.0139	12.805399894714400	9.272000312805180	0.7614	0.5688
16	3	64	0.0159	13.681300163269000	11.087400436401400	0.8263	0.5078
16	3	128	0.0074	9.304200172424320	6.922299861907960	0.8716	0.7724
16	3	256	0.015	13.300399780273400	11.174300193786600	0.8595	0.5348
16	4	32	0.0305	18.931699752807600	15.47920036315920	0.6875	0.0575
16	4	64	0.0139	12.79419994354250	9.095999717712400	0.7437	0.5695
16	4	128	0.0257	17.37779998779300	14.604499816894500	0.7667	0.2059
16	4	256	0.0127	12.240799903869600	10.204899787902800	0.8765	0.606
32	2	32	0.015	13.282899856567400	9.797300338745120	0.7614	0.536
32	2	64	0.0121	11.936100006103500	9.767200469970700	0.8686	0.6253
32	2	128	0.0064	8.653800010681150	6.407199859619140	0.8754	0.8031
32	2	256	0.0035	6.402299880981450	4.958199977874760	0.8937	0.8922
32	3	32	0.0167	14.009400367736800	11.357999801635700	0.8179	0.4839
32	3	64	0.0081	9.782899856567380	7.11929988861084	0.8534	0.7483
32	3	128	0.0118	11.794400215148900	9.600000381469730	0.8702	0.6342
32	3	256	0.009	10.294899940490700	7.958499908447270	0.8702	0.7213

32	4	32	0.0216	15.962400436401400	11.735899925231900	0.671	0.33
32	4	64	0.0144	13.03689956665040	9.688199996948240	0.7794	0.5531
32	4	128	0.0106	11.170299530029300	8.905099868774410	0.8728	0.6719
32	4	256	0.0104	11.08810043334960	8.064000129699710	0.7403	0.6767
64	2	32	0.0114	11.595199584960900	9.032600402832030	0.8451	0.6464
64	2	64	0.0129	12.337699890136700	9.795700073242190	0.8394	0.5997
64	2	128	0.0085	10.02560043334960	7.7220001220703100	0.8721	0.7357
64	2	256	0.0076	9.442099571228030	7.28980016708374	0.8857	0.7656
64	3	32	0.0145	13.070199966430700	10.19379997253420	0.813	0.5508
64	3	64	0.0088	10.202199935913100	7.6890997886657700	0.8614	0.7263
64	3	128	0.0127	12.214699745178200	10.003899574279800	0.8642	0.6077
64	3	256	0.0066	8.782600402832030	6.618199825286870	0.8893	0.7972
64	4	32	0.0169	14.110699653625500	10.352899551391600	0.7344	0.4764
64	4	64	0.0121	11.94789981842040	8.656200408935550	0.794	0.6246
64	4	128	0.0287	18.364999771118200	16.304500579834000	0.8121	0.1131
64	**4**	**256**	**0.0057**	**8.203900337219240**	**6.0370001792907700**	**0.8939**	**0.823**

Table 2: *Outputs of Different Combinations*

Figures

Figure 1: *Closing Price of Apple Stock*

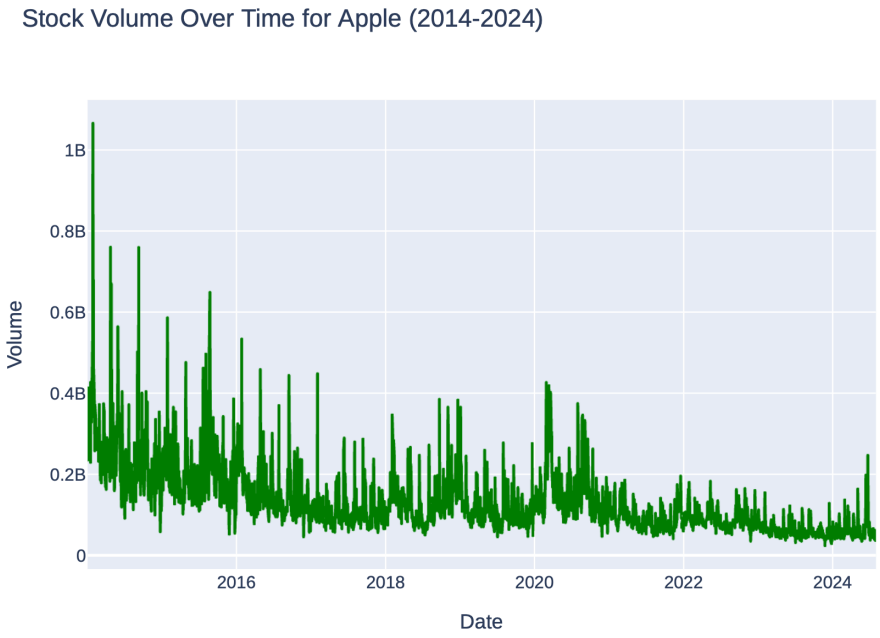


Figure 2: *Apple Stock Volume over time*

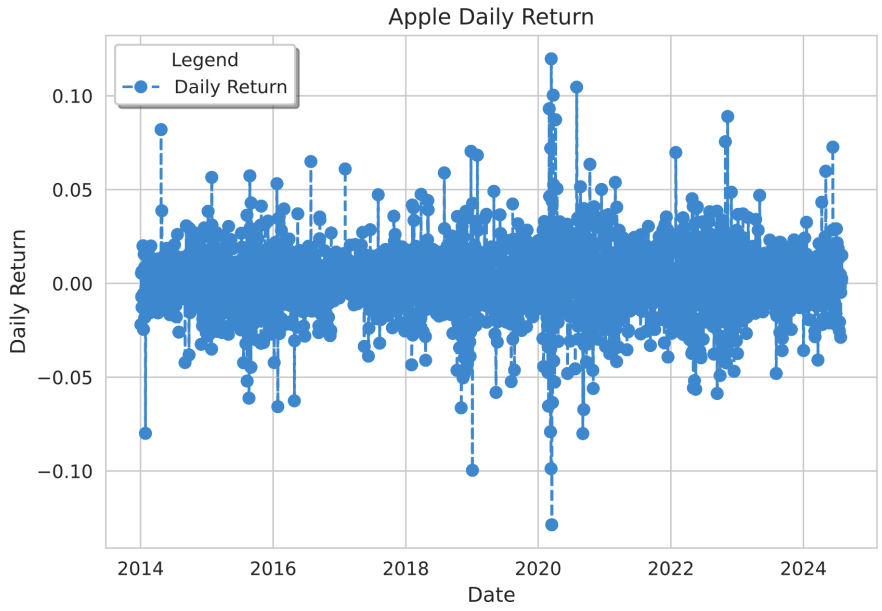


Figure 3: *Apple Daily Return*

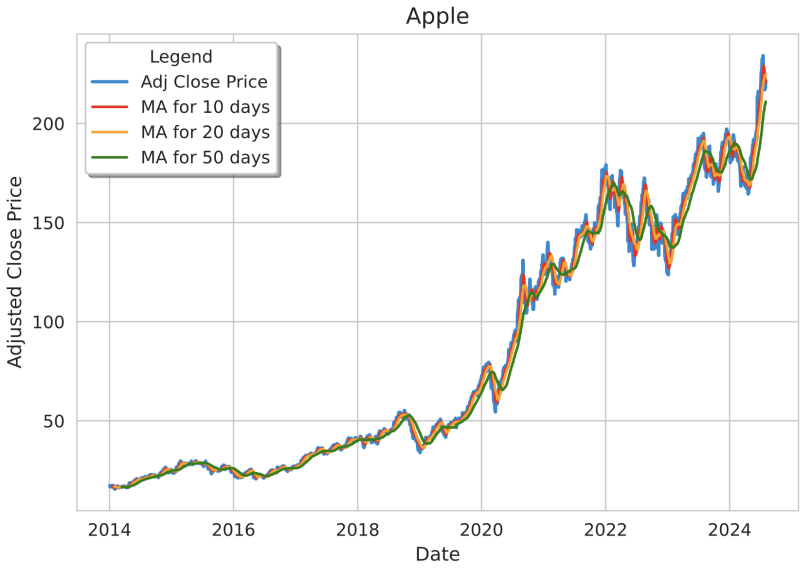


Figure 4: *Moving Average of Apple Stock*

Figure 5: *Risk vs Expected Return*

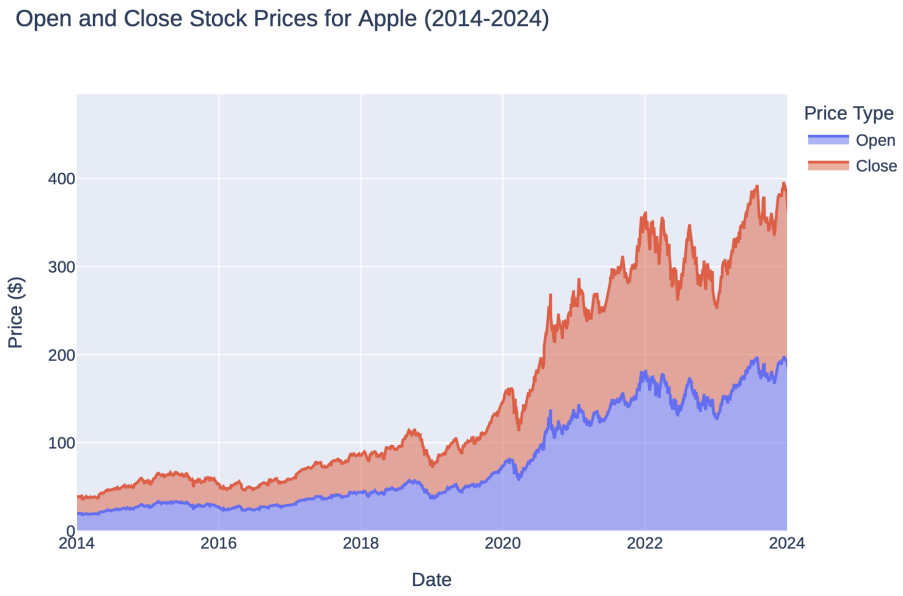


Figure 6: *Open and Close Apple Stock Prices*

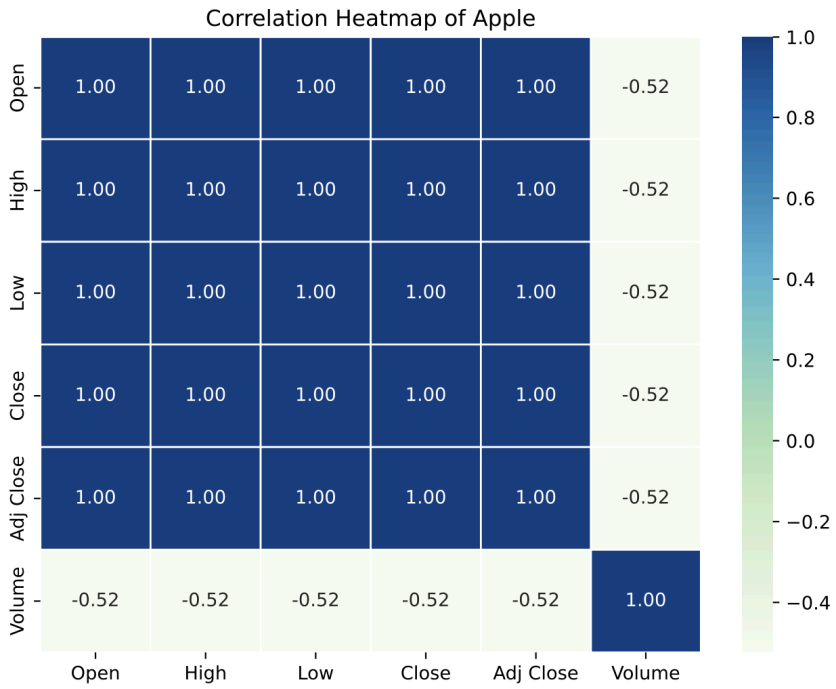


Figure 7: *Correlation Heatmap of Apple Stock*

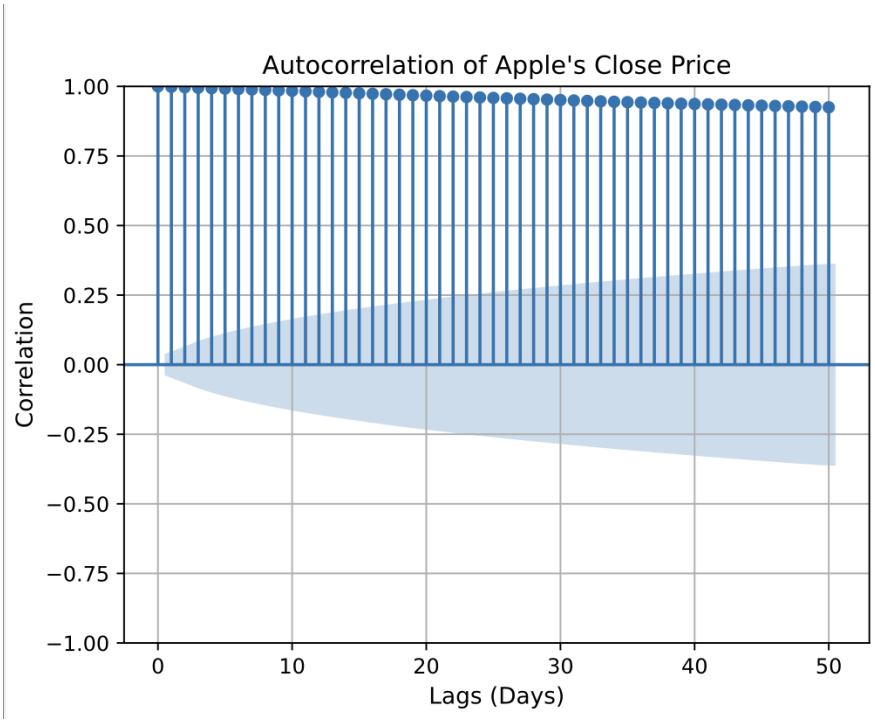


Figure 8: *Autocorrelation of Apple Stock Close Price*

Figure 9: *Apple Stock Distribution over Different Time Windows*

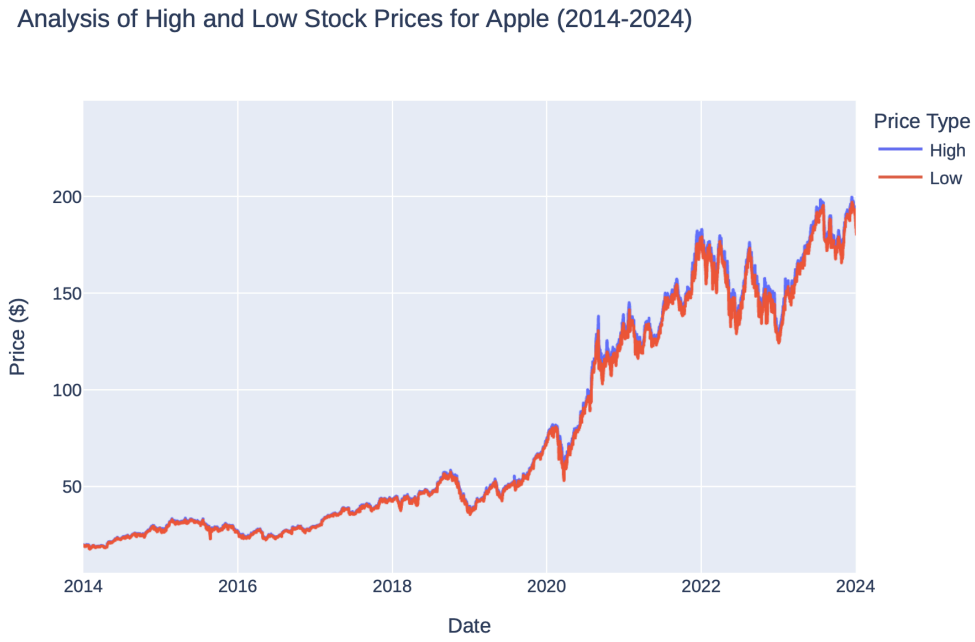


Figure 10: *Analysis of High and Low Apple Stock Prices*

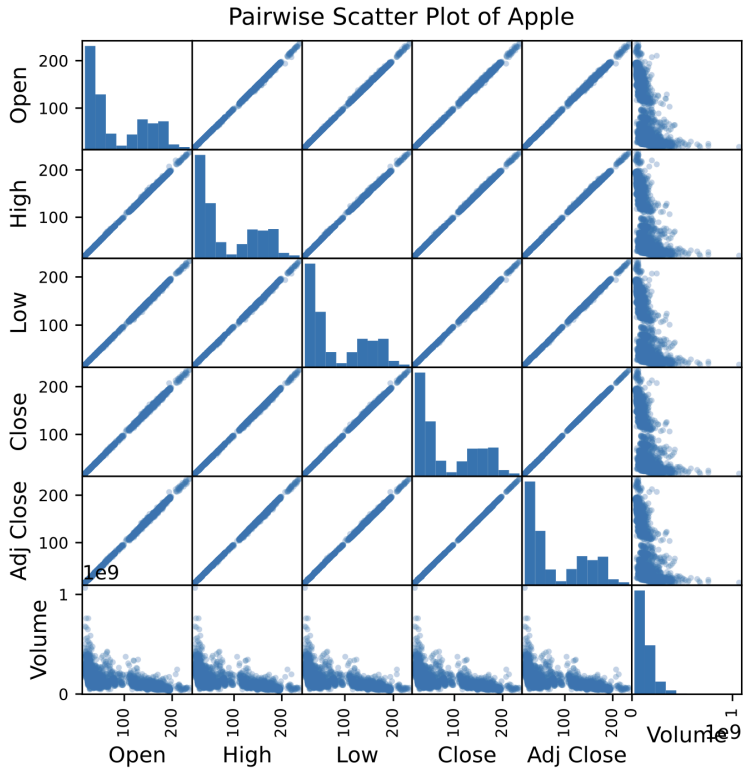


Figure 11: *Pairwise Scatter Plot of Apple*

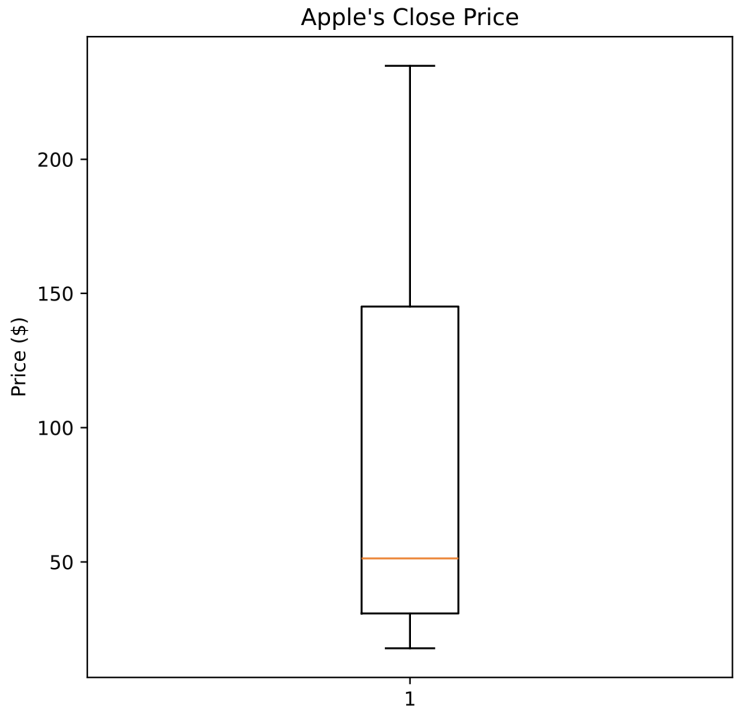
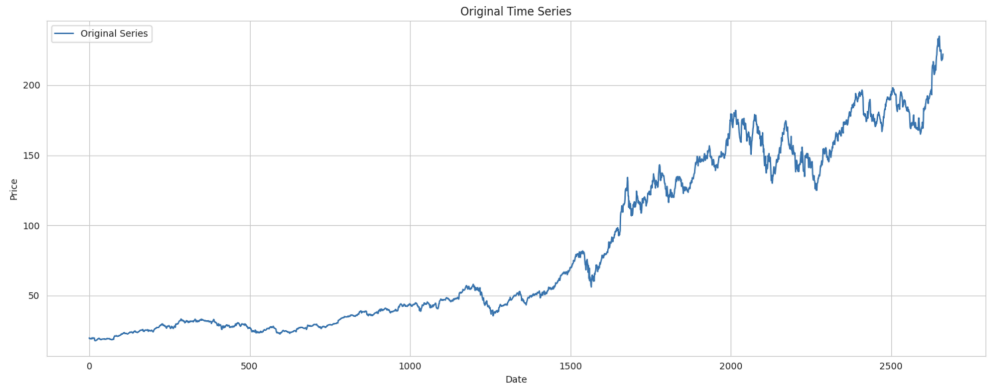


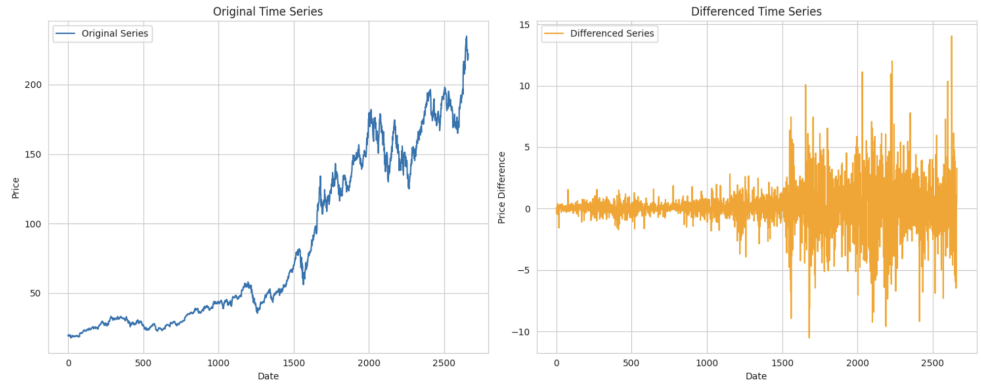
Figure 12: *Box Plot of Closing Price of Apple Stock*




```
=== Testing Original Series ===

ADF Test Results:
ADF Statistic: 0.4362
p-value: 0.9828
Critical values:
        1%: -3.4328
        5%: -2.8626
        10%: -2.5674

KPSS Test Results:
KPSS Statistic: 7.9957
p-value: 0.0100
Critical values:
        10%: 0.3470
        5%: 0.4630
        2.5%: 0.5740
        1%: 0.7390

Ljung-Box Test Results:
Test Statistic: 100340.1704
p-value: 0.0000
```

Figure 13: *Results of Stationary Tests on Original Time Series*

```
=== Testing Differenced Series ===
```




```
ADF Test Results:
ADF Statistic: -10.9112
p-value: 0.0000
Critical values:
        1%: -3.4328
        5%: -2.8626
        10%: -2.5674

KPSS Test Results:
KPSS Statistic: 0.2109
p-value: 0.1000
Critical values:
        10%: 0.3470
        5%: 0.4630
        2.5%: 0.5740
        1%: 0.7390

Ljung-Box Test Results:
Test Statistic: 84.1550
p-value: 0.0001
```

Figure 14: *Results of Stationary Tests on DifferencedTime Series*