ADRIAN COLLEGE

STOCK MARKET PREDICTION USING MACHINE LEARNING

BY: BRANDEN GULA

CS491 – COMPUTER SCIENCE PRACTICUM

SUPERVISER: DR. YASSER ALGINAHI

DATE: 4/22/2024

A practicum report submitted to the Department of Computer Science in partial fulfillment of the requirements for the Degree of Arts in Computer Science at Adrian College.

Adrian, Michigan, USA

Abstract:

This project delves into stock price prediction. The stock price history data will be extracted from Yahoo Finance API for three different companies\index fund (Apple, Palantir, Nasdaq). The stock price will be predicted by going through the following data science steps: data collection, exploratory data analysis (EDA), model building, and visualizations. To enhance the accuracy of the stock price prediction, technical indicators will be applied to the machine learning model. Some technical indicators will be calculated from Yahoo stock data are Relative Strength Index (RSI), Moving Average Convergence Divergence (MACD), Stochastic Oscillator (%K), Williams Percentage Range (W%R), Price Rate of Change (PROC), On Balance Volume (OBV), Simple Moving Average (SMA), Exponential Moving Average (EMA). These indicators can provide insight into market sentiment, investor behavior, momentum, and whether the stock is overbought or oversold. ML algorithms will be utilized to predict the stock prices for these companies. The final result will include the three companies and their predicted stock prices compared to their actual stock price utilizing Python programming. The end goal is to offer a structured guide for stock price prediction providing a foundation for both technical and non-technical investors.

.

Table of Content

1. Motivation

I have been actively following and investing in stocks as a hobby for quite some time now. This capstone project serves as a perfect recipe to combine my educational background in Computer Science and my passion for trading. The ever-changing dynamic of the stock market, combined with volatility, uncertainty, and complexity, presents a challenge and an opportunity to explore potential returns that may surpass those found in other markets. This project sheds light on the need for an approach to stock price prediction, that combines conventional statistical techniques with powerful machine learning models. Through a detailed methodology the capstone project aims to provide technical and non-technical stakeholders with a solid foundation for making well-informed investment decisions. This condensed research offers a structured guide for as accurate as possible stock price predictions.

2. Introduction

Drawing from my own experiences as an active stock market participant, this project has been sculpted from my background in Computer Science and a genuine interest in trading. Motivated by a desire to contribute to the ongoing talk regarding predictive analytics in finance, this research aligns with a growing recognition in the stock market. The capstone project seeks to tackle the complexities of the financial market, where uncertainty, volatility, and complexity prevail. Through a step-by-step approach covering data collection, exploratory data analysis, model building, and visualizations, the capstone aims to provide a user-friendly guide for making accurate stock price predictions. This journey progresses from diving into essential data processes to crafting sophisticated models, reaching its peak with the creation of visualizations.

3. Related work

This capstone project began by identifying the most effective machine learning model for stock price prediction. The study referenced here, titled “Comparison of Machine Learning Models for Market Predictions with Different Time Horizons," investigates and compares various machine learning models applied to stock market predictions. The focus is on individual stock performance using technical indicators from Nasdaq and NYSE data. The study focuses on three models: Support Vector Machine (SVM), Long Short-Term Memory (LSTM), and

Random Forest (RF). These models are evaluated for prediction accuracy across different time frames. The study found that none of the models outperformed the basic strategy that assumes stock prices will go up. This suggests that even complex machine learning models struggle to outperform simple approaches. The authors also emphasize the significance of considering external factors and refining model training methods [1]. Overall, this article provided valuable insights into which machine learning models to dive into for further investigation.

The search to find the most effective machine learning model for stock price prediction continues. After stumbling across a few articles, this one stuck out to me the most: “Forecasting Stock Market Trends Using Machine Learning Algorithms with Technical Indicators.” The study aims to present models for predicting stock market trends using machine learning algorithms and eleven technical indicators. These models were trained and tested on data obtained from the Dhaka Stock Exchange (DSE). Here are the accuracy results reported for their results: The results include the following: 'today', 'tomorrow', and 'day_after_tomorrow, respectively:

- Random Forest: 86.67%, 61.59%, 66.03%
- AdaBoost: 86.67%, 64.13%, 69.21%
- KNeighbors: 82.22%, 56.19%, 64.13%
- Logistic Regression: 86.67%, 64.13%, 69.21%
- MLP: 86.67%, 63.17%, 69.84%
- SVM: 86.67%, 64.13%, 69.21%

Their system utilized technical indicators such as Exponential Moving Average (EMA), Moving Average Convergence Divergence (MACD), Volatility Stop (VTS), and more, showing a correlation between these indicators and stock price movements. Based on their final results, the study displayed, “When comparing the selected machine learning approaches for six ML algorithms, KNeighbors showed slightly worse performance than RandomForest and SVC (SVM).” [2]. These studies informed the selection of machine learning models and explored the connection between technical indicators and prediction accuracy. As the capstone progresses, the focus will be allocated towards identifying the optimal model. This selection will consider the significance of incorporating technical indicators into the chosen model designs.

Another key element displayed in the article is that technical indicators played a key role regarding the prediction model's accuracy. The article focuses on RF; however, it is important to note that incorporating technical indicators, such as Exponential Moving Average (EMA), Relative Strength Index (RSI), Moving Average Convergence Divergence (MACD), and more can significantly enhance the model's performance. The study also sheds light on something called Out of Bag (OOB) error. Essentially, this is similar to a self-check mechanism for the model. It helps solidify that the predictions are accurate, even on new data, making the model trustworthy [3]. All in all, this article not only recommends utilizing RF, but also breaks down how it can be implemented and applied. It highlighted the challenge of understanding decision rules, explained how having more decision trees improves predictions, and introduced the concept of OOB error for model reliability. It also highlighted the significance of the correlation between these technical indicators and stock price movements and the importance of developing a stock price machine learning model. Moving forward, the capstone will shift on a deeper exploration of the random forest decision tree process and the integration of these technical indicators to optimize the chosen model's effectiveness.

3.2 Random Forest

The Random Forest machine learning model is a powerful model for making predictions in data analysis. They work by combining many decision tree models to reduce errors and enhance accuracy. The basic idea is to create a bunch of diverse models and then average their predictions. Here is a small breakdown of how random forests work. First, create multiple decision trees using different subsets of the data. Second, each tree is built by randomly selecting a subset of features at each step. Lastly, predictions are made by averaging the results from all the trees (for regression) or using majority voting for classification. The randomness introduced during the tree-building process helps reduce the correlation between the trees, making the random forest more accurate. A reason why people find random forests to be useful is that they perform well without needing a lot of adjustments. Random Forest has proven to be effective in various applications one being stock price prediction, showcasing reliability and consistency in making predictions [4]

3.2.1 What are Decision Trees?

Decision trees are a form of supervised machine learning designed for handling classification and regression through "if-then" conditions. They go through datasets, selecting features as decision points based on criteria such as a detective that keeps choosing the best clue (feature) that helps solve the case more neatly once they've solved it or further clues that don't really affect the solution (Gini Impurity). [4] This process repeats until optimal accuracy is achieved, or splitting the data adds no value. Decision trees are optimal for predicting or classifying new data, but they run the risk of overfitting. This is where random forest comes into play. This method contains multiple decision trees, each created from subsets of features and data rows. By combining these predictions, random forests provide a more optimal solution. Branching away from the pitfalls of individual decision trees.

3.2.2 Decision Tree Terminology

To construct a decision tree, here are the several key components:

- Root Node: The node is located at the top of the tree and represents the entire dataset.
- Splitting: Divides the node into branches.
- Branches: Sections of the trees that represent the outcome of the dataset.
- Leaf/Terminal Node: Sub node that does not split.
- Internal/Decision Nodes: Sub-node branching into additional sub-nodes.
- Pruning: The process of removing sub-nodes of an internal or decision node.
- Parent Node: A node divided into sub-nodes.
- Child Node: The sub-nodes are child nodes.

3.2.3 Figure 9: Random Forest Example

Here is a breakdown of the following figure 1: The random forest algorithm takes into account the results from several decision trees, each created randomly, to produce the final result. The process of acquiring the result is known as ensemble learning, which combines the results of various individual models known as weak learners [16].

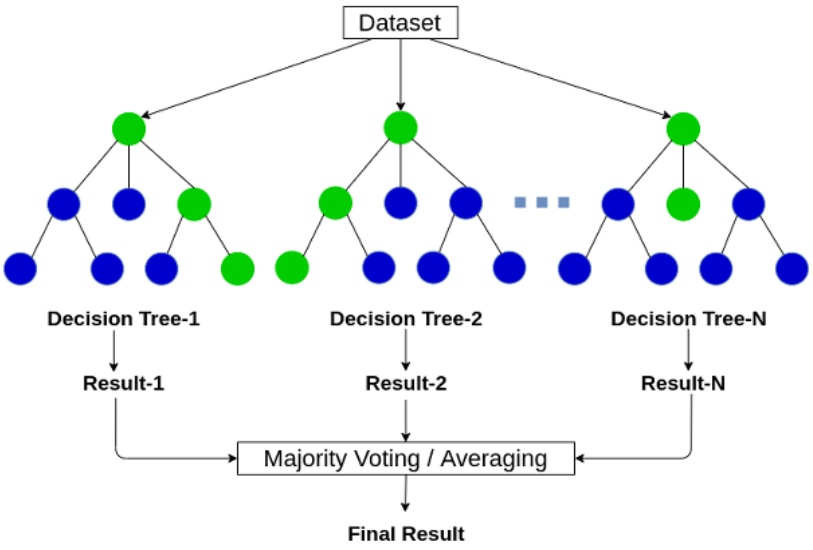

Figure 1: Random Forest Visualization [16]

3.2. Random Forest Regressor and Random Forest Classifier

The main differences between the models are: For random forest classifier tasks, the target variable (what the model is trying to predict) is categorical. The goal is to predict the category that an input instance belongs to. Regressor tasks, the target variable is continuous, the goal is to predict a continuous numerical value for each input instance. For classification tasks produce several measures of variable importance. These measures estimate the importance of a variable by looking at how much prediction error increases when data for that variable is altered while all others are left unchanged. Regression tasks produce a single measure of variable importance, which estimates the importance of a variable in predicting the target variable [6].

3.3. XGBoost Classification Model

Extreme Gradient Boosting (XGboost) is a powerful model that has gained some traction over the past few years. It falls under the ensemble learning umbrella that leverages the gradient boosting framework to achieve high accuracy. XGBoost is known for its ability to handle large datasets efficiently, address missing values effectively, and deliver quality performance regarding regression, classification, and ranking tasks. An application of XGBoost showcased in the research paper titled “Stock-Price Forecasting Based on XGBoost and LSTM." This paper delves into the use of XGBoost for predicting future stock price movements based on historical

data. Their research paper demonstrates that XGBoost, as an ensemble of decision trees, can be a valuable tool in stock price prediction. By combining multiple decision trees, XGBoost offers enhanced prediction capabilities compared to a single tree. Their study compared the performance of XGBoost with LightGBM, another gradient boosting model. Their findings suggest that an ensemble model combining XGBoost and LightGBM achieved the most promising results for stock price prediction. This highlights the potential benefits of combining different algorithms to leverage their strengths. However, due to time, XGBoost classification, Random Forest classification, and Random Forest Regressor will be utilized in this paper.

3.4. Technical Indicator 1: Relative Strength Index (RSI)

The Relative Strength Index (RSI) is a crucial momentum indicator used to measure the magnitude of recent price changes and identify potential overbought or oversold market conditions. RSI (14) is a 14-period calculation to calculate its values that range from 0-100. This indicator is utilized by traders who are identifying the market's strengths and potential reversal points. If a stock is in a strong uptrend, the RSI can often exceed the value of 70 for extended periods of time. On the other hand, if a stock is in a strong downtrend, the RSI can often go below the value of 30 for extended periods of time. A stock is considered overbought if the RSI technical indicator is above 70 and oversold if it is below 30. It is important to note that some traders use more extreme levels, such as 80/20, to reduce the occurrence of false signals. Thus, enhancing the accuracy of their analyses [8]. Equation (1) is the formula to calculate RSI.

$$RSI = 100 - \frac{100}{1+RS} \tag{1}$$

where,

$$RS = \frac{Average\ gain\ over\ the\ past\ 14\ days}{Average\ loss\ over\ the\ past\ 14\ days}$$

3.4.1. Technical Indicator 2: Stochastic Oscillator

The stochastic oscillator, created by George C. Lane in the late 1950s, serves as a momentum indicator, displaying the speed and momentum behind price movements. Lane clearly emphasized that its distinctive trait does not track price or volume but instead focuses on

the momentum of price shifts. As a rule, momentum changes direction before price. The oscillator also recognizes both bull and bear setups, providing an advantage in anticipating market reversals. Operating within a range-bound framework, the stochastic oscillator flourishes at pinpoint overbought and oversold conditions, enhancing the ability for traders to identify overbought and oversold levels based on extreme stochastic oscillator values. [9] Equation (2) is the formula to calculate the Stochastic Oscillator.

$$\%K = 100 * \frac{C - L14}{H14 - L14} \tag{2}$$

$Where,$

$C = Current\ Closing\ Price_{ys}$
$L14 = Lowest\ Low\ over\ the\ past\ 14\ days$
$H14 = Highest\ High\ over\ past\ 14\ days$

3.4.2. Technical Indicator 3: Moving Average Convergence Divergence (MACD)

Moving Average Convergence Divergence (MACD) is a pivotal momentum indicator in technical analysis, introduced by Gerald Appel in the late 1970s. This technical indicator operates by computing the inequalities between two time period intervals, commonly represented by historical closing prices. The indicator's foundation is to help identify momentum and the strength of its direction. Utilizing moving averages of two distinct time intervals, often time intervals using exponential moving averages (EMAs), the MACD calculates a momentum oscillator line through the subtraction of these moving averages, referred to as 'divergence'. The key idea when choosing these moving averages is to make sure one has a faster period than the other, emphasizing its role in evaluating changes in momentum and market trends [10]. Equation (3) is the formula to calculate MACD. When MACD lies above the signal sign, it flashes a buy signal; when MACD lies below the signal sign, it flashes a sell signal.

$$MACD = EMA_{12}(C) - EMA_{26}(C)$$
$$SignalLine = EMA_{9}(MACD) \tag{3}$$

$Where,$
$C = Closing\ Price\ Series$
EMA_n = n day Exponential Moving Average

3.4.3. Technical Indicator 4:Price Rate of Change (PROC)

The Price Rate of Change (PROC) technical indicator is described as an indicator that measures the percentage difference between the current price and the price from several periods ago. It serves as an oscillator to confirm trend movements as well as a trend following indicator. Understanding the signals is as follows: a growing indicator curve suggests that the asset's current price is increasing compared to previous values, indicating bullish trader sentiment and an upward trend. On the other hand, a decreasing ROC value signifies that sellers are in control [11]. Equation (4) calculates the PROC.

$$PROC(t) = \frac{C(t)-C(t-n)}{C(t-n)} \tag{4}$$

$Where,$
$PROC(t) = Price\ Rate\ of\ Change\ at\ time\ t$
$C(t) = Closing\ price\ at\ time\ t$

3.4.4. Technical Indicator 4: On Balance Volume OBV

On Balance Volume (OBV) is a tool that keeps track of the cumulative volume flowing in or out of a stock. If the stock closes higher than the previous day, the day's volume is considered up-volume, and if it closes lower, it is considered down-volume. A rising OBV is seen as a signal of smart money entering the stock, and as more people follow, both the stock and OBC show an increase. Non confirmations, where price movement does not align with OBV movement, often occur at market tops or bottoms. When the stock closes up, the day's volume is added to the cumulative total, when it closes down, the day's volume is subtracted [12]. Equation (5) calculates the OBV.

$$OBV(t) = \begin{cases} OBV(t-1) + Vol(t) & if\ C(t) > C(t-1) \\ OBV(t-1) - Vol(t) & if\ C(t) < C(t-1) \\ OBV(t-1) & if\ C(t) = C(t-1) \end{cases} \tag{5}$$

$Where,$
$OBV(t) = On\ Balance\ Volume\ at\ time\ t$
Vol(t) = On Balance Volume at time t
$C(t) = Closing\ Price\ at\ time\ t$

3.4.5. Technical Indicator 6: Williams % R

Williams Percent Range (W%R), a momentum indicator created by Larry Williams, aims to identify overbought or oversold conditions in the market. Operating on a scale from 0 to -100, it evaluates a stock's closing price relative to its high-low range over a set period, commonly 14 days. Similar to the stochastic oscillator, it provides insights into entry and exit points for traders. The indicator helps gauge the potential for a price reversal based on recent price movements [13].

Equation (6) is the formula to Calculate Williams Percent Range

$$\%R = \frac{H14 - C}{H14 - L14} * -100 \quad (6)$$

$where,$
$C = Current\ Closing\ Price$
$L14 = Lowest\ Low\ over\ the\ past\ 14\ days$
$H14 = Highest\ High\ over\ the\ past\ 14\ days$

3.4.6. Technical Indicator 7: Bollinger Bands

Bollinger Bands is a technical analysis tool developed by John Bollinger that has the purpose of defining high and low levels to assess whether an asset's price is relatively high or low compared to historical data. These bands consist of a moving average (middle band), an upper band, and a lower band. The middle band reflects the intermediate-term trend, combining trend information with relative price levels. Bollinger Bands adapt to market changes utilizing standard deviation, which offers insights on volatility. The width of the bands, known as bandwidth, represents volatility in relation to the middle band. %b, another indicator that shows where the price stands relative to the bands [14]. Equation (7) is the formula that calculates the bollinger bands.

$$BOLU = MA(TP, n) + m * \sigma[TP, n]$$
$$BOLD = MA(TP, n) - m * \sigma[TP, n] \quad (7)$$

$where,$
$BOLU = Upper\ Bollinger\ Band$
$BOLD = Lower\ Bollinger\ Band$
$TP\ (typical\ price) = (High + Low + Close) \div 3$
$n = Number\ of\ days\ in\ smoothing\ period\ (tyipically\ 20)$
$m = Number\ of\ standard\ deviations\ (typically\ 2)$
$\sigma[TP, n] = Standard\ Deviation\ over\ last\ n\ period\ of\ TP$

3.4.7 Technical Indicator 8: Average Directional Index (ADX)

The Average Directional Index (ADX) is a trend oscillator used to determine the direction and strength of a market trend. It features a solid ADX line and two dashed lines, +DI and -DI, representing directional components. The ADX assists traders in identifying whether the market is in a sideways or trending phase. It ranges from 0 to 100%, with values between 0 and 20% indicating a weak trend and 40% or more indicating an extremely strong trend. The ADX is viable for various trading assets and is effective on timeframes of 1 hour and higher, with the 1-day interval recommended by its creator. It provides accurate signals after range consolidation, helping traders discern the beginning of a trending market and potential trend reversals. The ADX shines when determining trend strength and avoiding false signals during flat market conditions [15]. Equation (8) is the formula to calculate ADX [16].

$$+DI = \left(\frac{Smoothed\ +DM}{ATR}\right) \times 100$$
$$-DI = \left(\frac{Smoothed\ -DM}{ATR}\right) \times 100$$
$$DX = \left(\frac{|+DI - -DI|}{|+DI \pm DI|}\right) \times 100$$
$$ADX = \frac{(PRIOR\ ADX \times 13) + Current\ ADX}{14} \quad (8)$$

where,
$+DM\ (Directional\ Movement) = Current\ High - PH$
$PH = Previous\ High$
$-DM = Previous\ Low - Current\ Low$
$CMD = Current\ DM$
$ATR = Average\ True\ Range$

3.4.8 Technical Indicator 9: Simple Moving Average

Simple moving average calculates the average of a set of stock prices over a specific time period (24) by summing up the stock prices and dividing the number of periods (24). Each stock price in the time series (24) contributes equally to the average, regardless of when it occurred. How can SMA be applied to the stock market? It helps traders identify existing trends, upcoming trends, and spot overextended trends that are on the verge of reversing. One way to spot an existing trend, when the SMA line is above the stock price, it suggests that the average price over

the specified period (24) is higher than the current price. This could indicate a bullish trend or potential bullish momentum [17]. Equation (9) calculates the simple moving average.

$$SMA_n = \frac{(C_1+C_2+C_3 \ldots C_n}{n}) \times 100 \quad (9)$$

where,
C_i *is Closing Price on day* i
n *is Moving Average period*

3.4.9 Technical Indicator 9: Exponential Moving Average

Exponential moving average is a type of moving average that places more weight on recent stock prices, making it more responsive to recent price changes. The EMA is calculated using a smoothing factor that gives more weight to the most recent data points. This exponentially decreasing weighting scheme means that older prices have a diminishing impact on the EMA [18]. Equation (10) calculates the exponential moving average.

$$EMA_n = \sum_{i=0}^{n-1} w_i C_{t-i} \quad (10)$$

where,
$\sum_{i=0}^{n-1} w_i C_{t-i}$ *and* n *is the input window length*

4. Methodology

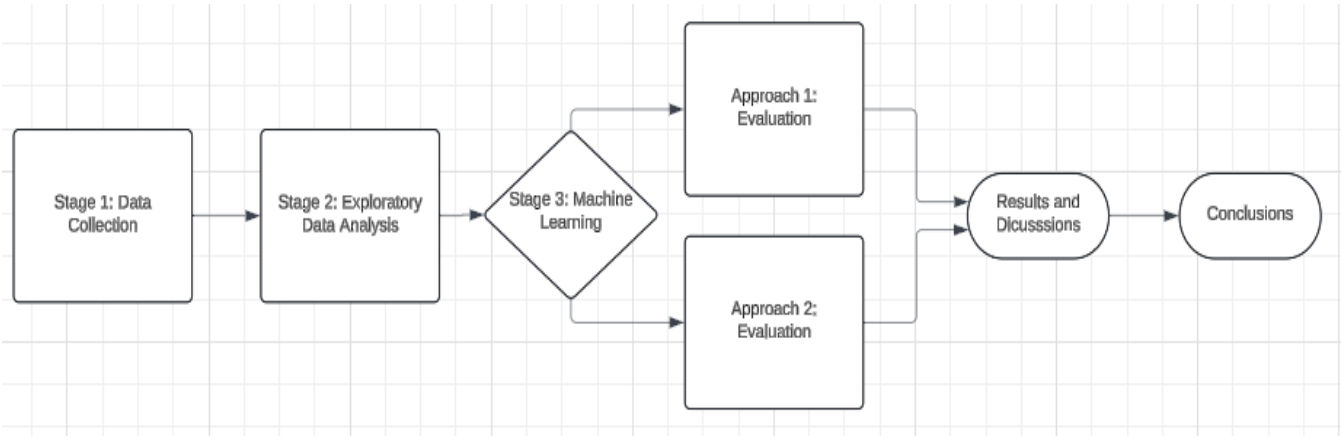

Figure 2: Data Steps Pipeline

As shown in figure 2: Data collection will contain obtaining historical stock prices utilizing Yahoo Finance API for up-to-date Apple and Palantir financial data. Exploratory Data Analysis (EDA) will consist of using Matplotlib for visualizations to understand the significance between technical indicators and the companies. Analyze key statistics (mean, median, and

standard deviation (STD) for data summary. Check for trends/seasonality in time series analysis to identify unique fluctuations in price. Machine learning portion will consist of data cleaning, splitting my data into train/test sets (80% training, 20% testing). This portion will also consist of a failed approach and steps to take to avoid it and refine future approaches. The refined approach 1 will include Random Forest Regressor predicting tomorrow's closing stock prices. Refined approach 2 will consist of Random Forest classification and XGBoost classification to predict whether tomorrow is an up day (1) or down day (0) based on today's closing price. Finally, tie results, discussions and conclusions based on the refined approach evaluations.

4.1 Data Collection

The initial step involved extracting stock market data. The idea is to extract stock market data for one company at a time, allowing for the application of learning to the remaining 2 companies (Palantir and Nasdaq). The focus was on publicly available data. A yahoo Finance overview article introduced the idea of 'yfinance' a Python API for accessing Yahoo Finance data. This article also provided a helpful example for retrieving Apple stock data, one of the three companies that will be utilized in this capstone. The following section presents the stock market data extraction code (table 1) along with an explanation of its implementation within this capstone project.

Table 1: Initialize Apple Stock DataFrame

```
data = yf.download('AAPL',
                   start='2022-01-01',
                   end='2024-01-01'
)
```
Source: [19]

Explanation of table 1: The code initializes a data frame, and the content is stored into apple data, by downloading the historical stock data for Apple (AAPL) using 'yfinance'. The stock market data spans from 2022-01-01 end 2024-01-01. Here is a table that includes the first five rows regarding my Apple DataFrame.

Table 2: Apple Stock DataFrame

	Open	High	Low	Close	Adj Close	Volume
Date						
2022-01-03	177.830002	182.880005	177.710007	182.009995	179.724533	104487900
2022-01-04	182.630005	182.940002	179.119995	179.699997	177.443558	99310400
2022-01-05	179.610001	180.169998	174.639999	174.919998	172.723587	94537600
2022-01-06	172.699997	175.300003	171.639999	172	169.84024	96904000
2022-01-07	172.889999	174.139999	171.029999	172.169998	170.008133	86709100

The values in table 2 provide necessary data to calculate all relevant technical indicators. This process of data collection will be explained in detail in the next section on EDA.

4.1.1 Current Apple Stock DataFrame

The current DataFrame contains seven times as many columns as it did before calculate the technical indicators on various timeframes. The new columns include RSI, %K, MACD, signal line, W%R, OBV, PROC, SMA, and EMA. Keep in mind that any values that are not a number (NaN) are because those indicators required previous values to perform calculations. Without any previous values to perform calculations, the output will be NaN until there is enough data required to perform said calculations.

4.2 Apple Stock Exploratory Data Analysis (EDA)

The following figure 3 showcases the RSI values alongside the corresponding Apple stock closing prices. The idea is to identify extremely oversold or overbought values and paint the picture to illustrate the potential RSI can bring to the table.

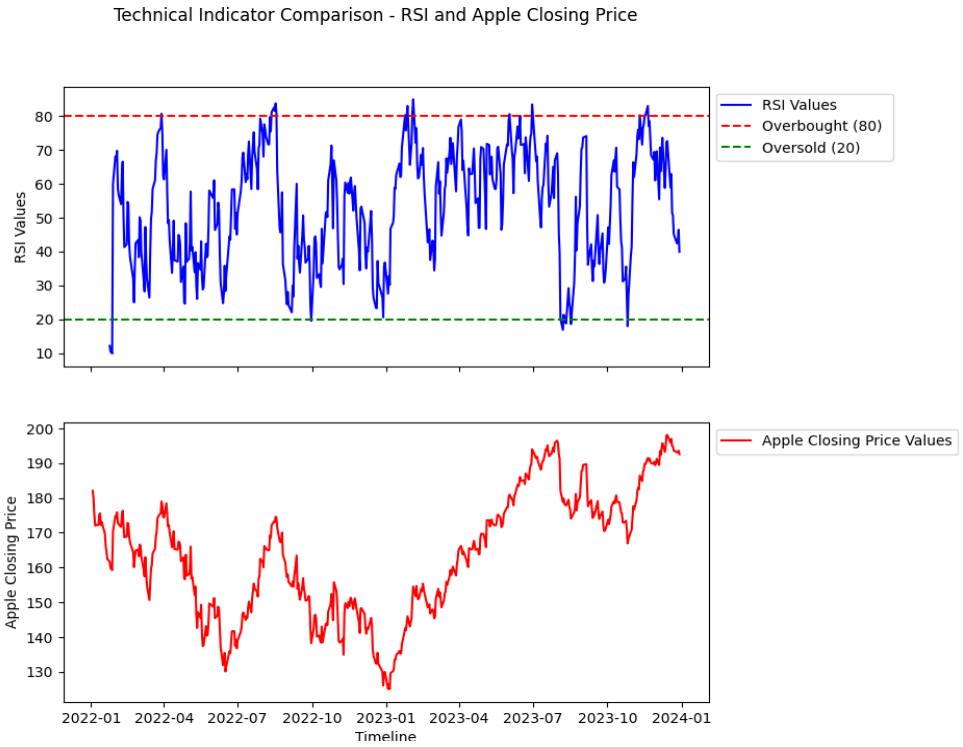

Figure 3: Technical Indicator Comparison - RSI and Apple Closing Price

Figure 3 showcases, oversold and overbought values, there is a trend developing. If a trader were to buy at oversold conditions and sell at oversold conditions according to this graph, that trader would be in the green. For example, if a trader were to purchase Apple stock around 2023-11, the RSI would be around 18 for around 170 per share, and if they sold around 2024-12, the RSI would peak between 80 and 83, around 190 per share. This trader would have generated roughly 11% in profit; this chart showcases the potential RSI displays.

4.2.1 RSI Descriptive Statistics for Apple Stock (2022-2024)

The descriptive statistics analysis regarding RSI for Apple stock reveals the following:

- RSI count: 487
- Mean RSI: 52.88

- Standard deviation: 16.76
- Minimum RSI: 9.86
- Quartile 1 (Q1): 39.80
- Quartile 2 (Q2): 55.57
- Quartile 3 (Q3): 67.33
- Maximum RSI: 84.84

4.2.2 RSI Descriptive Statistics Interpretation

The range between the min and max RSI values is quite substantial, indicating that apple stock has been quite volatile during the timeframe of 2022–2024. The mean is closer to the 75% quartile range, indicating the distribution of RSI values favors overbought territory. Based on the STD (16.76) and taking the mean (52.88), this indicates that there is a reasonable amount of fluctuation regarding the RSI values.

4.2.3 Apple Stock W%R EDA Example

Figure 4 presents Williams Percent Range (W%R) values alongside corresponding Apple stock closing price values. The analysis focuses on identifying periods of extreme overbought/oversold levels (-20,-80) respectively, to paint the picture that showcases the potential corresponding to W%R. Looking at the oversold and overbought values, we notice a trend playing out. If a trader were to buy at the extreme oversold conditions, and sell at the extreme overbought conditions, according to this graph, this trader would be in the green. For example, if a trader were to purchase Apple stock around 2023–08 with a W%R value of roughly -97 for around 175 per share and sell it around 2023–09 with a W%R value of roughly -5 for 188, this trader would have generated roughly 7% profit, indicating that this chart showcases the potential regarding W%R.

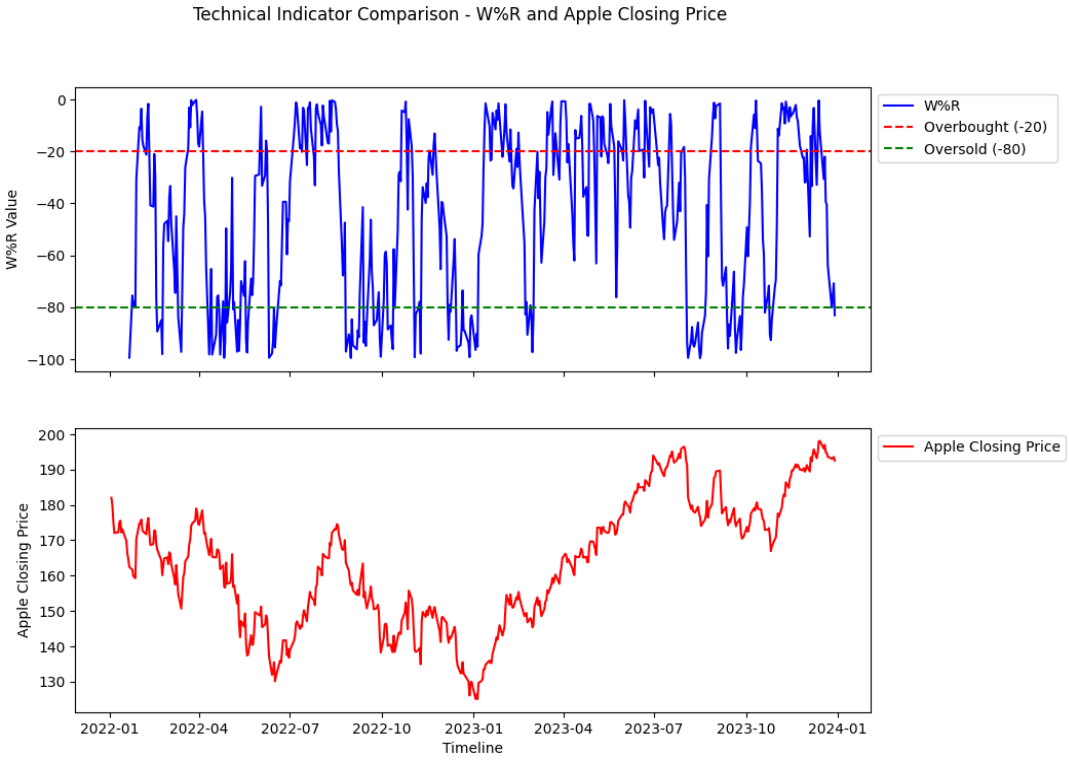

Figure 4: Technical Indicator Comparison - W%R and Apple Closing Price

4.2.4 W%R Descriptive Statistics for Apple Stock

The descriptive statistics analysis regarding Williams Percent Range for Apple stock showcase the following:

- W%R count: 488
- Mean W%R: -44.37
- Standard deviation: 32.57
- Minimum W%R: -99.65
- Quartile 1 (Q1): 75.53
- Quartile 2 (Q2): 39.30
- Quartile 3 (Q3): -14.16
- Maximum W%R: -.17

4.2.5 W%R Descriptive Statistics Interpretation

The range from the minimum to maximum values covers a broad range of numbers, indicating significant variation in W%R ranges over the timeframe (2022–2024). The mean being closer to the 25th percentile indicates a potential skewness towards the lower end of the distribution regarding the W%R values. Based on the STD (32.57) and taking account of the mean (52.88), this indicates that there is a moderate level of volatility regarding the W%R values.

4.2.6 Apple Stock MACD/Signal Line EDA Example

Information to understand figure 5 when the MACD (blue line) is above the signal line (red line), this indicates a bullish signal, and when the MACD is below the signal line (red line), this indicates a bearish signal. From this plot, we can see that when the MACD crosses from bullish to bearish, a downtrend occurs. When the MACD is far above the signal line, this results in a sharp downswing seen in 2022-04, 2022-08, 2022-11, 2023-09, and 2023-10. Similar to the other side, when the MACD is far below the signal line, this results in an uptrend, as seen in 2022-04, 2022-07, 2023-01, and 2022-09. These few examples paint the picture, showcasing the potential of MACD.

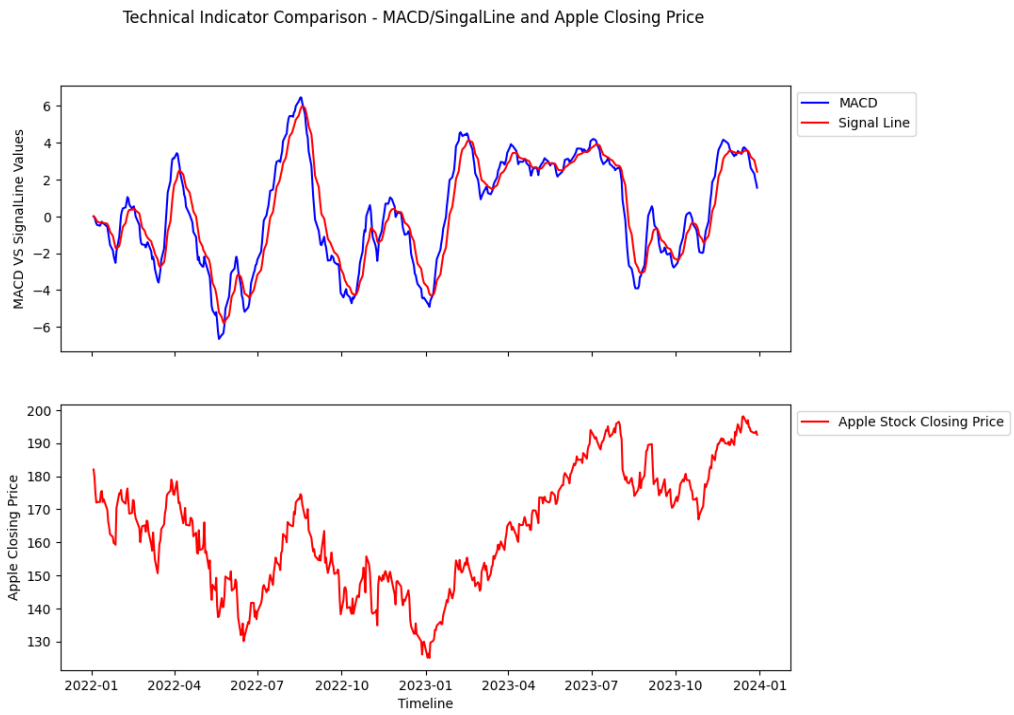

Figure 5: Technical Indicator Comparison - MACD/Signal Line and Apple Closing Price

4.2.7 MACD Descriptive Statistics for Apple Stock

The descriptive statistics analysis regarding MACD for Apple stock reveals the following:

- MACD count: 501
- Mean MACD: .27
- Standard deviation: 2.90
- Minimum MACD: -6.66
- Quartile 1 (Q1): -2.05
- Quartile 2 (Q2): -0.18
- Quartile 3 (Q3): 2.92
- Maximum MACD: 6.46

4.2.8 MACD Descriptive Statistics Interpretation

The MACD being substantially larger than the mean indicates that values are scattered across the distribution, hinting at volatility. The difference between the minimum and maximum

values hints at volatility. The mean favors the median, signaling the distribution around the mean, suggesting a balanced spread of values.

4.2.9 Signal Line Descriptive Statistics for Apple Stock

The descriptive statistics analysis regarding signal line for Apple stock displays the following values:

- Signal Line Count: 501
- Mean Signal Line: .25
- Standard deviation: 2.7
- Minimum Signal Line: -5.81
- Quartile 1 (Q1): -1.75
- Quartile 2 (Q2): -0.10
- Quartile 3 (Q3): 2.80
- Maximum Signal Line: 5.98

4.2.10 Signal Line Descriptive Statistics Interpretation

Similar to the MACD, the signa line being substantially larger than the mean indicates that values are scattered across the distribution, hinting at volatility. The difference between the min and max values hints at volatility as well. The fact that the mean is relatively close to the median suggests a somewhat symmetrical distribution around the mean, suggesting a balanced spread of values.

4.3 Apple Stock EDA Example: RSI and MACD

The following figure 6 will generate a subplot of buy and sell signals for Apple stock using RSI and MACD technical indicators. The blue line will display Apple stock's closing price over the time period of late 2020 through early 2024. Buy signals are if the RSI falls below the oversold threshold of (30) or a bullish MACD crossover occurs (MACD crosses above the signal line), a green triangle will be displayed at the corresponding date on the price chart, indicating a potential buying opportunity. If the RSI goes above the overbought threshold (75) or a bearish crossover occurs (MACD crosses below the signal line), a red inverted triangle will be printed at

the corresponding date for apple stock closing price, suggesting a potential selling opportunity [20].

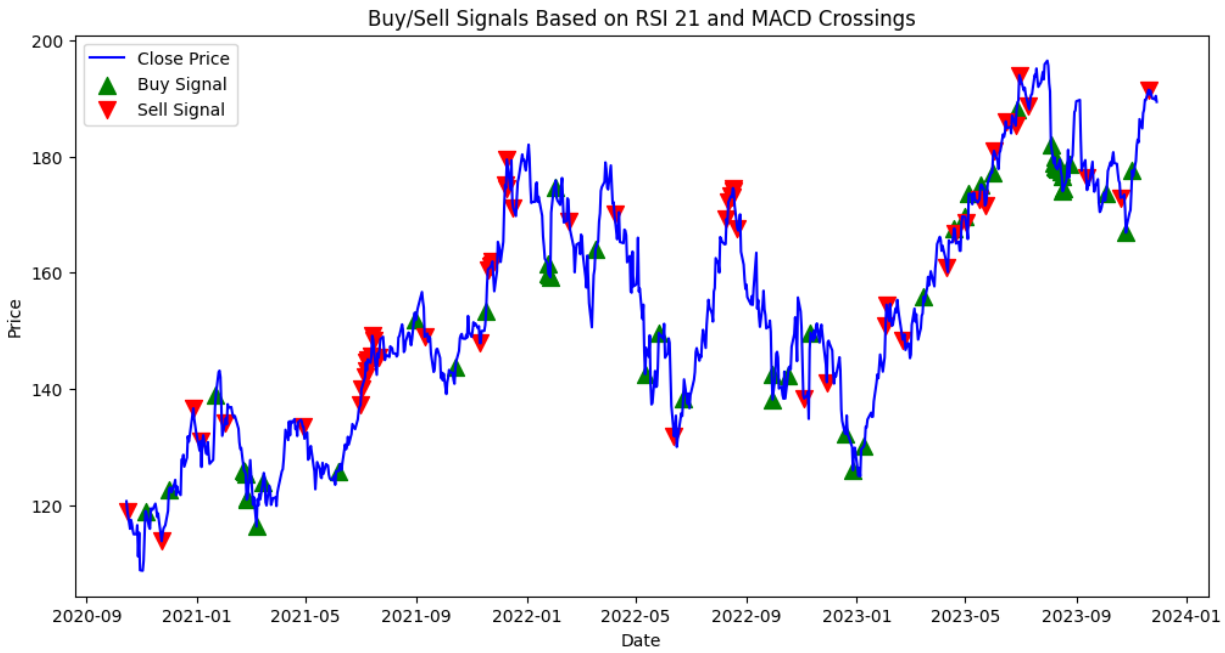

Figure 6: Apple Closing Price Buy/Sell Signals Based on RSI 21 and MACD Crossings

As seen in figure 6, the buy signals are more effective than the sell signals. From this insight acquired, maybe the model's will be superior at predicting up days compared to down days.

4.4 Time Series Analysis

The time series analysis focuses on the average quarterly closing stock prices of Nasdaq, Apple and Palantir from Q1 2020 to Q4 2023 figure 7. The primary objective is to identify potential discrepancies in stock price behavior that could significantly impact the performance of the machine learning models utilized for stock price prediction.

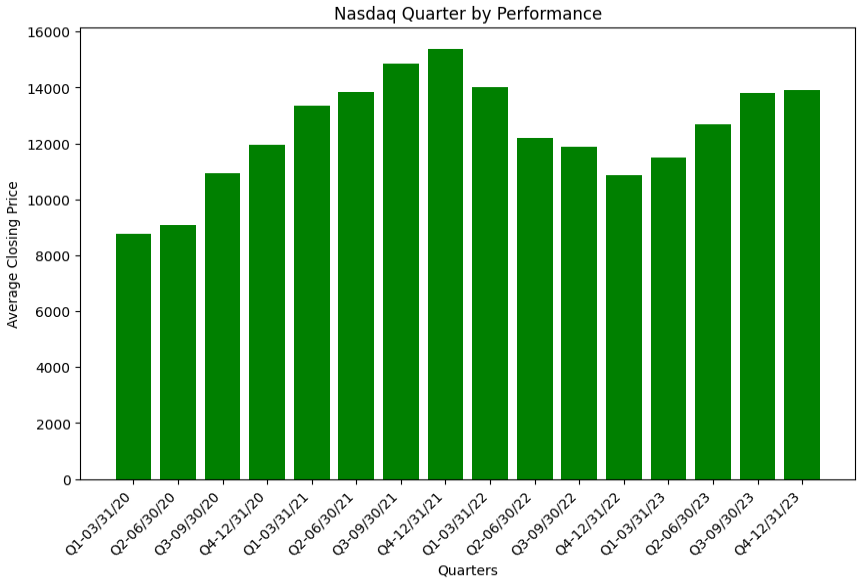

Figure 7: Nasdaq Quarter by Performance

As shown in figure 7, across the various timeframes, Nasdaq generally produces its strongest performance in quarter 4. These metrics are significant in identifying any discrepancies in terms of price behavior in the model's predictions. Following visualizations will include Apple and Palantir performances.

Examining figure 8, similar to Nasdaq's subplot, this graph showcases the average closing prices of Apple for each quarter within the same timeframe. Apple demonstrates its strongest performance in Q4 as well.

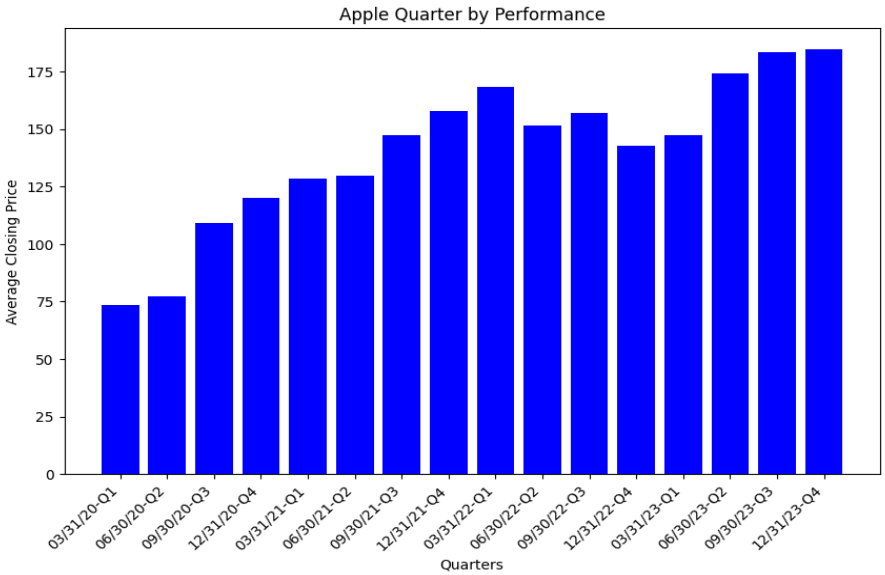

Figure 8: Apple Quarter by Performance

The data presented in figure 9, in contrast to Nasdaq and Apple, Palantir's performance diverts a bit. Regardless of the identical timeframe, Palantir tends to perform best in Q3 based on the timeframes used.

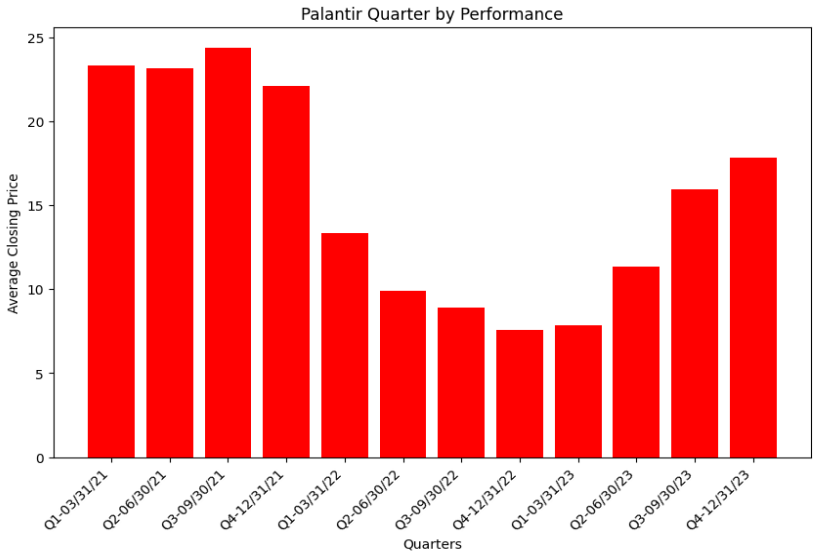

Figure 9: Palantir Quarter by Performance

4.4.1 Time Series Conclusion

The analysis reveals a seasonal trend across Nasdaq, Apple, and Palantir, with the strongest performance generally occurring in Q4. This information can be crucial for the models, as it helps account for potential biases related to specific time periods. The other position of the time series analysis is to identify any discrepancies in prices, as this information can significantly impact the performance of the models. Major news events, such as the surge in Palantir's stock price by 35.5% in November 2023 [21], can disrupt stock prices, impacting the accuracy of the models. These companies perform the strongest in the quarter (Q4), based on the time series analysis. Such explosive movements, like the one observed in Palantir's stock price, can potentially decrease the effectiveness of the models. The models may struggle to accurately capture and adjust to rapid changes that could potentially challenge the models to maintain accuracy in their predictions. Overall, the time series analysis provides insights regarding seasonal trends and potential challenges for stock price prediction associated with the models.

4.5 Machine Learning: A Learning Experience

This portion delves into a learning experience journey of optimizing Random Forest Regressor (RFR) model for stock price prediction. Here, we kickstart the journey of exploration and discovery, of blindly optimizing Random Forest Regressor, forgetting proper machine learning principles and real-world relevance. The journey begins with evaluating RFR model's performance using various metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared score, showcasing the balance between model complexity and the ability to predict tomorrow's closing price. Next up, the focus is on feature selection, whose sole purpose is to identify the most influential indicators for stock price prediction. We'll witness removing features that have low significance model performance and analyze how performance metrics increasingly become better. The journey will conclude with encountering a crucial learning experience (data leakage and no real-world application). This learning experience reveals the pitfalls of utilizing random split for training and testing the model, and shed light on minimal real-world application.

4.5.1 Machine Learning: First Model Breakdown

This breakdown will consist of a visualization (figure 10), a breakdown of the model's performance metrics and significant features associated with the model's performance. Let's take a look at figure 10, the first model created, how the model was improved and the learning experiences associated with them.

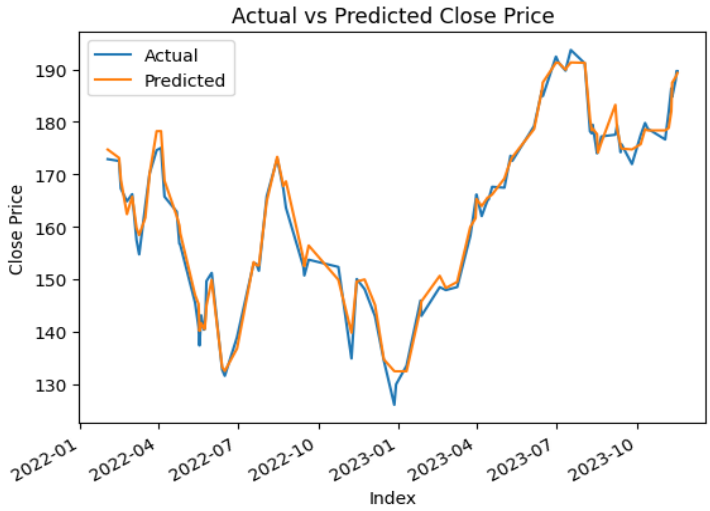

Figure 10: First Model

Figure 10 performance metrics breakdown:

- Training Mean Absolute Error: 1.458
- Testing Mean Absolute Error: 1.703
- Mean Squared Error: 4.842
- R-squared: 0.983

Here are the common ways to determine the performance of the RFR model, starting with training mean absolute error. This metric measures the absolute difference between the actual and predicted values during the training phase. In this case, the training mean absolute error is around 1.458. Meaning, on average my model's predictions during training are off by around $1.458 above or below the next day Apple's closing stock price. Similar to training mean absolute error the test is approximately 1.703. On average, my model's predictions on the test data are off by around $1.703 above or below the next day's closing stock price. [22]

Next, let's discuss overfitting and underfitting to further break down the MAE values. Overfitting occurs when the model performs well on the training data but poorly on unseen data. This occurs when my model memorizes the training data rather than learning from it. Low training data and high predicted values can be a potential sign for overfitting. On the other hand, underfitting is when the scenario occurs when my model is unable to capture the important insights from my data, this results in the model performing poorly on both the training and unseen data. [22]

Mean squared error measures the average squared difference between the actual and predicted values. Mean squared error also gives higher weight to large errors due to squaring. The average squared difference between my model's predictions and next day's closing price is $4.842. Lastly, R2 also known as confidence of determination, provides information regarding how well my model fits a specific dataset. Essentially, it tells us how closely the predicted values from the model align with the actual data values. R2 ranges from 0 to 1, where 0 indicates that my model does not fit the data at all, and 1 indicates a perfect fit where my model fits perfectly for the dataset. Overall, my R2 score of .983 indicates a very strong fit of my model to the dataset, suggesting that the chosen features incorporated are effective in explaining the fluctuations for the next day's closing stock prices. [22]

The first step towards optimizing the RFR model is to find which features performed the best and cut out all the ones that have no significance. Here is the information on the top 7 indicators that performed the best:

- SMA_1: 0.873847
- SMA_%: 0.116799
- PROC 1: 0.008831
- SignalLine: 0.00009
- MACD: 0.000061
- PROC 5: 0.000048
- PROC 21: 0.000047

Out of the top 7 indicators one stands out from the rest and 2 others display a sliver of contribution. The next step that was taken to optimize the RFR model is import a library called

GridSearchCV. This library takes in hyperparameters (n_estimators, min_samples_split, max_depth, and more) based on these parameters GridSearchCV helps find the optimal result in finding the lowest mean absolute error. This leads to the RFR that performs well in terms of minimizing the absolute differences between predicted and actual values on unseen data. [23]

4.5.2 First optimized RFR Model (All Indicators)

The optimized RFR hyperparameters were utilized in the creation of the figure 11 model. Figure 11 will display the optimized model visualization and results using every technical indicator.

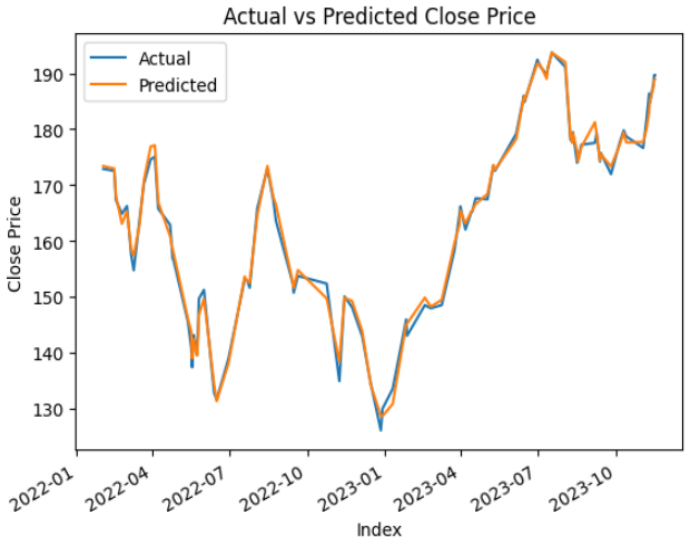

Figure 11: Optimized Model Using Every Indicator

The following bullet points correspond to figure 11 metrics:

- Training Mean Absolute Error: 0.437
- Testing Mean Absolute Error: 1.07
- Mean Squared Error: 1.893
- R-squared score: 0.994

Every performance metric has beat the previous model's performance, showcasing the changes made proved to be effective. The next step finetunes the model further.

4.5.3 Second optimized RFR Model Top 7 Indicators

As previously seen, the majority of the technical indicators are considered as noise meaning the indicators barely contribute to the RFR performance. Therefore, curiosity came into the picture. What if the technical indicators that were considered as noise were stripped from the model's performance? The idea here is to cut out the noise and take a look at the top seven technical indicators that displayed the highest contribution toward the model's performance.

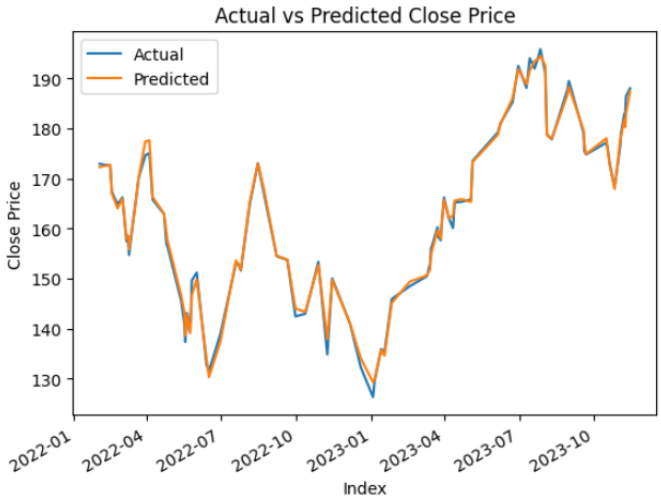

Figure 12: Optimized Model Using Top 7 Indicators

- Training Mean Absolute Error: 0.381
- Testing Mean Absolute Error: 0.905
- Mean Squared Error: 1.45
- R-squared score: 0.995

As shown in figure 11 it displays the best performance on optimizing the model. This model's performance beat every previous performance metric. Therefore, cutting out the noise is proving to be effective.

4.5.4 Second Optimized RFR Model Top 3 Indicators

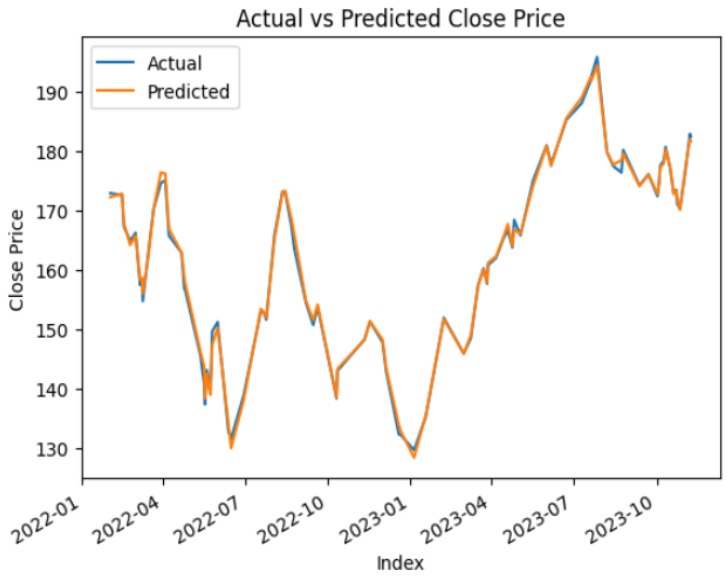

Figure 13: Optimized Model Using Top 3 Indicators

- Training Mean Absolute Error: 0.352
- Testing Mean Absolute Error: 0.676
- Mean Squared Error: 0.815
- R-squared score: 0.997

Figure 11 results were quite impressive. Again, the optimization process has led to another model that beat every previous performance metric. From this moment, the trend emerged, cutting the noise of insignificant technical indicators the model has proved to be superior every time. Therefore, the next step is to cut down the model to the top two best performing indicators. Will the trend of cutting down the noise continue to optimize the model's performance?

4.5.5 RFR Model Top 2 Indicators

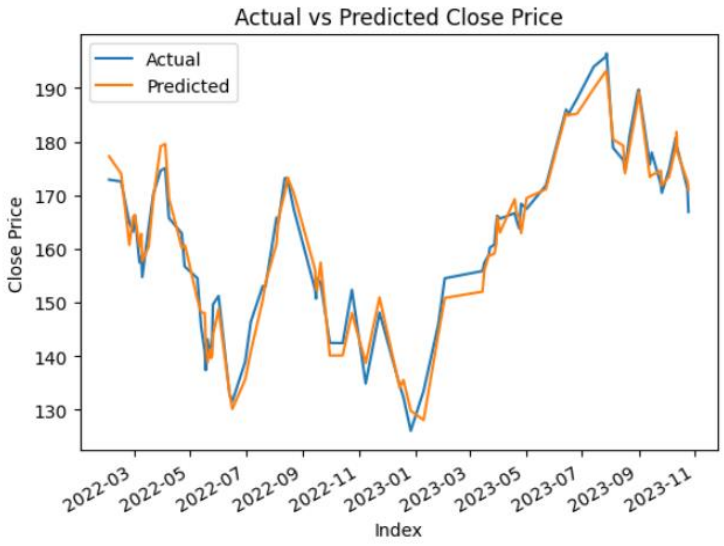

Figure 14: Model Using Top 2 Indicators

- Training Mean Absolute Error: 1.019
- Testing Mean Absolute Error: 2.664
- Mean Squared Error: 9.517
- R-squared score: 0.964

The initial theory of removing insignificant technical indicators would consistently improve the model's performance was correct. However, utilizing the top two best performing indicators resulted in the model's worst performance. This unexpected outcome is further emphasised by the following figure 15. Despite having low next to 0 significance, PROC 1 significantly impacts the model's performance.

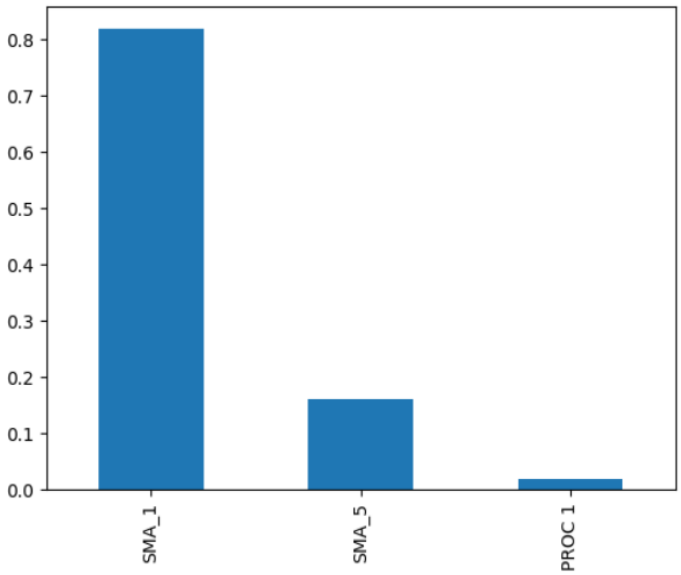

Figure 15: Technical Indicator Impact

As shown in figure 15, a hypothesis can be formed. Feature selection and optimizing the model are crucial, removing all insignificant indicators can be detrimental, as insignificant indicators may hold hidden importance.

4.5.6 Machine Learning: Learning Experience Evaluation

The same methods were applied to Palantir and Nasdaq however, after further examination these results proved to be insignificant. The model was prone to data leakage thus displaying false results of stock price prediction. The stock market data was randomly divided throughout the dataset causing the model to use future values to predict past values and so on. The following refined approaches take account of the learning experiences and apply the knowledge learned.

4.6 Refined Approach 1

This refined approach takes a different route. In this approach the random forest regressor is trained on the length of 2022-2024 data. 80% of this dataset will be utilized for training, while

the remaining 20% is utilized for testing. The predictions will start on 2023-09-25 (start of testing data). The target variable will remain the same, predicting tomorrow's closing stock price.

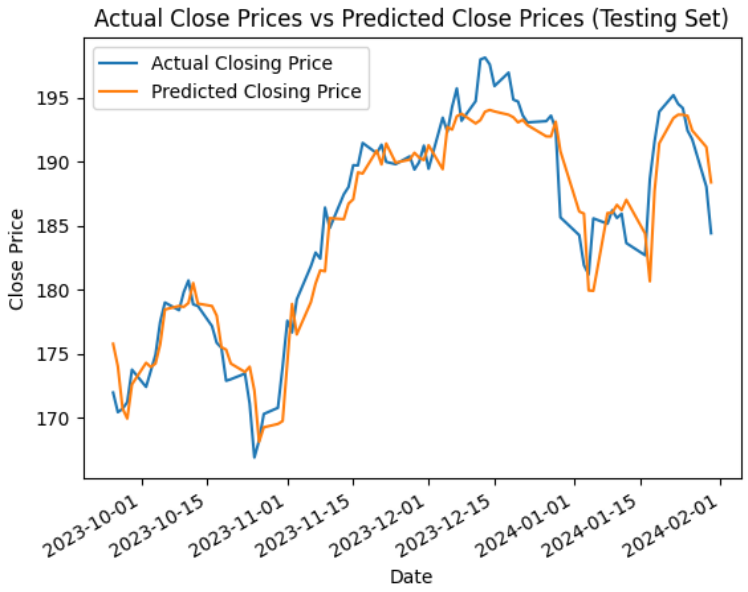

Figure 16: Apple Next Day Closing Price Result

Figure 16: Utilizing the following indicators ('RSI 14', 'PROC 5', 'MACD', 'OBV', 'EMA_9', '%K 14', 'SMA_5', 'Signal Line', and 'W%R 14'] and yahoo finance columns ('High', 'Low', 'Adj Close', 'Open', and 'Close'). Contributing to a training mean absolute error of around 1.63, testing mean absolute error 1.89, mean squared error: 5.96, and R2 score of 0.92. Yahoo finance dominated the model's performance while the technical indicators slightly contributed to the model's performance.

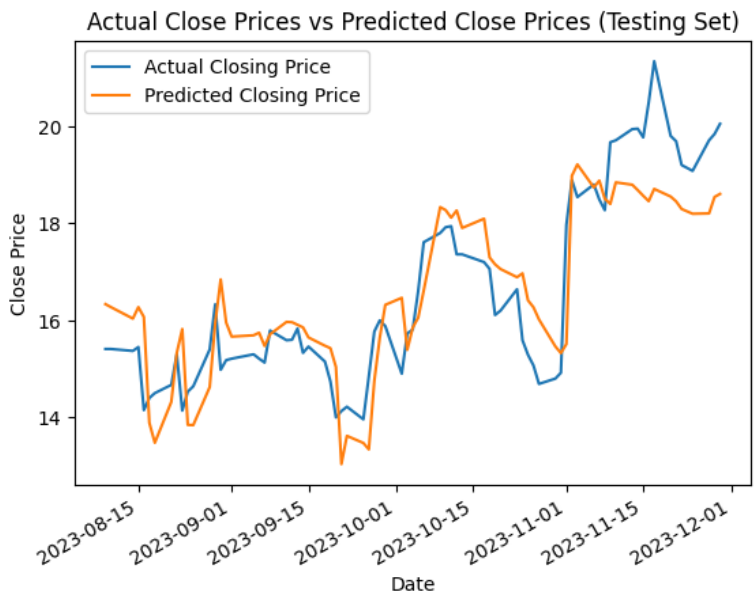

Figure 17: Palantir Next Day Closing Price Result

Figure 17: Utilizing the following indicators ('RSI 14', 'PROC 5', 'MACD', 'OBV', 'EMA_9', '%K 14', 'SMA_5', 'Signal Line', and 'W%R 14') and yahoo finance columns ('High', 'Low', 'Adj Close' 'Open', and 'Close'). Contributing to a training mean absolute error of around 0.16, testing mean absolute error 0.82, mean squared error: 0.98, and R2 score 0.75. Concerns regarding overfitting and relatively low R2 however, nonetheless the model still performs. Regarding the feature contributors goes to yahoo finance columns and low significance of the technical indicators that contributed to the model's performance. The subplot displays a discrepancy in price action, a contributing factor to potentially the strong Q4 earnings (November 2023).

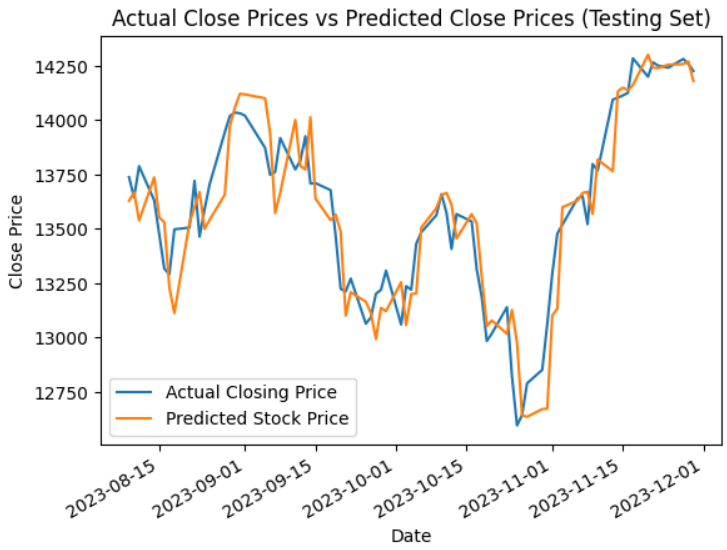

Figure 18: Nasdaq Next Day Closing Price Result

Figure 18: Combination of features that were used are the following, ('RSI 5', 'MACD', 'SMA_5', '%K 5', 'W%R 5', 'OBV', 'Open', 'High', 'Low', 'Close', and 'Volume'). Training MAE and test MAE 111.5, 129.33 respectively. Mean Squared Error: 27195.16 and R2 score: 0.84. Major contributors to the machine learning model are yahoo finance API columns with low relevance regarding the technical indicators.

4.6.1 Refined Approach 1: Evaluation

All in all, RFR models showcased potential for predicting next day closing prices for the chosen stocks. Yahoo finance features were superior compared to the technical indicators. Based on the complexity of this approach, predicting the next day's closing stock price is too complex. Therefore, instead of predicting the exact closing price, the next goal is to predict whether or not the next day’s closing stock price will be higher or lower than the current day’s closing price.

4.7 Refined Approach 2

The stock market is notorious for its unpredictability therefore instead of predicting the next day's closing stock price of Apple, Palantir and Nasdaq. These steps will include creating a binary target variable called (target) indicating whether the next day’s closing price is expected to be higher (1) or lower (0) than the current day’s closing price [24]. Random Forest Classifier

and XGBoost classifier will continue using the same stock dataset timeframes (2022-2024). Next step is, delving into the realm of classification model performance metrics.

4.7.1 Classification Model Performance Metrics

Accuracy reflects the overall success rate of the model in predicting whether tomorrow's price will be higher (1) or lower (0) than today's closing price. Recall measures how well the model identifies days where the price actually goes up (true positive) or where the price actually goes down (true negative). Precision is the quality of the models (1) and (0) predictions. F1-score takes precision and recall, combines the two into a single score utilizing the harmonic mean between the two metrics. Support is all the instances between (1) and (0) predictions [3].

$$Accuracy = \frac{tp+tn}{tp+tn+fp+fn} \tag{11}$$

$$Precision = \frac{tp}{tp+fp} \tag{12}$$

$$Recall = \frac{tp}{tn+fp} \tag{13}$$

$$F_1\ score = 2 \times \frac{precision \times recall}{precision+recall} \tag{13}$$

$where,$
$tp\ = Number\ of\ true\ positive\ values$
$tn\ = Number\ of\ true\ negative\ values$
$fp\ = Number\ of\ false\ positive\ values$
$fn\ = Number\ of\ false\ negative\ values$

4.7.2 Classification Model's Results

These results display the performance of two classification models, Random Forest Classifier (RFC) and XGBoost classifier, to predict Apple, Nasdaq, and Palantir stock prices. The models were trained to predict whether a stock's closing price on a given day will be higher (1) or lower (0) than the current day's closing price. The evaluation metrics utilized are the classification performance metrics. The results will compare the performance of RFC and XGB for the three companies chosen.

Table 6: Apple Stock Classification Results

RFC Apple				
Metric	Precision	Recall	F1-score	Support
0	0.7	0.59	0.64	44
1	0.65	0.75	0.69	44
XGB Apple			0.67	88
Metric	Precision	Recall	F1-score	Support
0	0.69	0.61	0.65	44
1	0.65	0.73	0.69	44
Accuracy			0.67	88

As shown in table 6 Apple stock model results demonstrated similar performances. The precision, recall, F1-score, and accuracy were comparable between the two models, indicating that neither classification model significantly outperformed the other in terms of predicting the target variable.

Table 7: Nasdaq Classification Results

RFC Nasdaq				
Metrics	Precision	Recall	F1-Score	Support
0	0.5	0.89	0.64	35
1	0.78	0.31	0.44	45
Accuracy			0.56	80
XGBoost Nasdaq				
0	0.62	0.6	0.61	35
1	0.7	0.71	0.7	45
Accuracy			0.66	80

As seen in table 7, when predicting Nasdaq, the XGBoost showcased superior results over the RFC model. XGBoost displayed higher precision, recall, F1-score and accuracy compared to RFC, suggesting that XGBoost is more effective for Nasdaq stock prediction.

Table 8: Palantir Classification Results

RFC Palantir				
	Precision	Recall	F1-Score	Support
0	0.54	0.88	0.67	43
1	0.67	0.23	0.34	43
Accuracy			0.56	86
XGB Palantir				
0	0.56	0.86	0.68	43
1	0.7	0.33	0.44	43
Accuracy			0.59	86

The results shown in table 8 are similar to Nasdaq results. The XGBoost model outperformed the RFC model. XGBoost demonstrated higher precision, recall, F1 score, and accuracy, indicating its effectiveness in predicting Palantir higher or lower than the current day's closing price.

4.7.3 Refined Approach 2: Evaluation

The provided tables (6, 7, and 8) showcase the performance results of both models. Both models (RFC and XGBoost) displayed similar performance across all performance metrics. This suggests that neither model significantly outperformed the other for predicting Apple stock target variables. XGBoost displayed superior performance compared to RFC across all metrics. This indicates that XGBoost is a more effective model for predicting the trend of Nasdaq's closing price. Similar to Nasdaq, XGBoost outperformed RFC in all metrics. This suggests that XGBoost is also a better model for predicting the trend of Palantir's closing price.

5. Approach 1 and 2 Results

Random forest regressor model was utilized to predict tomorrow's closing price for the selected companies. The model utilized historical data from 2022-2024, 80% used for training and the remaining 20% for testing. The model's performance was evaluated using standard regression performance metrics: Testing and training mean absolute error, mean squared error and r-squared. Recap: Apple's model achieved a high r-squared (.92), indicating a good fit

between predicted and actual closing prices. However, the testing MAE (1.89) signals some room for improvement in prediction accuracy. Palantir's model displayed a lower r-squared (.75) compared to Apple and Nasdaq, suggesting a weaker fit. The training MAE (0.16) is substantially lower than the testing MAE (0.82), these metrics raise concerns about overfitting. Nasdaq's model displayed a solid r-squared (0.84) with relatively high MAE values for both training and testing. The feature importance is significant; Yahoo finance columns remained dominant over the technical indicators.

The random forest regressor models demonstrated some solid potential for predicting next-day closing prices, particularly for Apple and Nasdaq. However, the limitations of this approach are present. The complexity of the stock market makes predicting tomorrow's closing prices a sophisticated challenge. Additionally, concerns regarding overfitting (Palantir) and news events shed light on the need for further refinement. Nonetheless, the visualizations showcased promising results.

Random Forest Classifier and XGBoost classification were utilized to predict whether tomorrow's closing price is an up (1) or down (0) day based on today's closing price. The model's performance was evaluated using well known classification performance metrics: Precision, recall, accuracy, f1-score, and support. Apple stock model's performance tied with an accuracy score of 67%. Nasdaq XGBoost proved to be the superior model beating RFC accuracy by 10%. Similar to Nasdaq, Palantir's XGBoost model beat RFC by 3% in terms of accuracy. All technical indicators proved to contribute towards the model's performance. No Yahoo Finance columns were used towards the model's predictions. All in all, the choice of which classification model can significantly impact the performance of stock price prediction models. While both RFC and XGBoost classifiers may perform similarly for certain stocks like Apple, XGBoost tends to outperform RFC for stocks like Nasdaq and Palantir. Thus, it is crucial to select the appropriate classification model based on the specific characteristics and behaviors of the chosen companies.

6. Discussion

The random forest regression model achieved some success in predicting next-day closing prices, mainly for Apple and Nasdaq (r-squared of 0.92 and 0.84, respectively). This indicates a solid correlation between the predicted and actual closing prices for these companies. However, it is important to acknowledge limitations, especially for Palantir. The high MAE for Palantir (0.16) compared to the testing MAE (0.82) suggests overfitting, meaning the model may be too focused on the training data and struggle with unseen stock price data. Also, the complexity of the stock market makes predicting future stock prices extremely challenging.

The XGBoost classifier outperformed the Random Forest Classifier for both Nasdaq and Palantir in predicting up/down price movement. This indicates that XGboost may be a more situatable model for Apple, Palantir and Nasdaq. The finding that all technical indicators contributed to the model's performance while no Yahoo Finance data was utilized, hint at the importance of technical analysis for predicting price direction.

The time series analysis potentially showcased how major news events can have an impact on stock price prediction. Due to an unexpected earnings beat in November, 2023 displayed a discrepancy in Palantir's stock price prediction. Therefore, it can be wise to choose historical stock data time frames where volatility consistently remains low for effective stock price prediction. These random surges in stock price, highlights the randomness and complexity of predicting future stock prices.

This capstone highlights the potential of machine learning models for stock price prediction, particularly XGBoost for classification. However, limitations are present. The models were trained on historical data, and future market events can significantly impact stock prices. Also, the study focused on a limited number of companies and timeframes. Further research is needed to explore the effectiveness of these models across a broader range of companies, market conditions, and timeframes. Additionally, incorporating news sentiment analysis could potentially improve prediction accuracy. As well as, incorporating more impactful indicators while minimizing less impactful indicators could potentially improve the prediction accuracy.

Lastly, selecting the optimal machine learning model that fits an individual's parameters may increase prediction accuracy.

7. Conclusion

This capstone project delved into the potential of machine learning models for predicting stock prices. The research analyzed led to a significant contribution to understanding the complex realm of stock price prediction. First off, the importance of feature selection in machine learning models for stock price prediction was highlighted. It was assumed that eliminating insignificant technical indicators would consistently improve the model's performance. However, that was not the case, even when statistically speaking the indicator held no significance, it can hold hidden importance that contributes to the model's predictions. This finding sheds light on the importance of proper feature selection rather than throwing every feature together in finance. Second, the project showcased that machine learning models can achieve some success in predicting next-day closing prices. This is highlighted by the models achieving r-squared values of 0.92 and 0.84, respectively. These results suggest that machine learning has potential to be a valuable tool for stock price prediction. However, keep in mind the complexity of the stock market and the influence of random news events present challenges to achieving optimal accuracy. Third, the compared performance of the two models (Random Forest Classifier and XGBoost Classifier). The findings revealed that XGBoost outperformed RFC in predicting up/down price movements for Nasdaq and Palantir stocks. These findings can suggest that XGBoost may be a more suitable model for predicting stock prices within certain timeframes, and specific companies. Fifth, this capstone project shed light on the impact of major news events on stock prices and its ability to affect the model's predictions. The discrepancy in Palantir's stock price prediction due to an unexpected earnings beat in November 2023. This highlights the significance of news events when constructing model predictions and selecting historical data for training and testing. Historical data with consistent low volatility could potentially be ideal for improving prediction accuracy.

All in all, this capstone project has showcased the potential of machine learning models, particularly for XGBoost. However, the complexity of the stock market and the influence of random news events need further research. Future studies should explore these models across a

broader range of companies, market conditions, and timeframes. Additionally, incorporating factors like news sentiment analysis, optimizing technical indicators, and selecting a low volatility stock could further improve the accuracy of stock price predictions. By further dwelling into those sections, an individual can potentially enhance the effectiveness of their machine learning models in the complex realm of stock market analysis.

8. Acknowledgements

I would like to express my gratitude to my family, friends and Dr. Yasser Alginahi for their endless support throughout my journey.

.

9. References

[1] Gerholm and A. Lindberg, “Comparison of machine learning models for market predictions with different time horizons.” Available: https://www.diva-portal.org/smash/get/diva2:1597259/FULLTEXT01.pdf

[2] P. P. Dey, N. Nahar, and B. M. M. Hossain, “Forecasting Stock Market Trend using Machine Learning Algorithms with Technical Indicators,” *International Journal of Information Technology and Computer Science*, vol. 12, no. 3, pp. 32–38, Jun. 2020, doi: https://doi.org/10.5815/ijitcs.2020.03.05.

[3] L. Khaidem, S. Saha, and S. Dey, “Predicting the direction of stock market prices using random forest,” *Applied Mathematical Finance*, vol. 0, no. 0, pp. 1–20, 2016, Available: https://arxiv.org/pdf/1605.00003.pdf

[4] M. Mcgill, “Random Forests,” *Math Mcgill* , Mar. 28, 2018. https://www.math.mcgill.ca/yyang/resources/doc/randomforest.pdf (accessed Feb. 18, 2024).

[5] A. Sharma, “Decision Tree vs. Random Forest - Which Algorithm Should you Use?,” *Analytics Vidhya*, May 11, 2020. https://www.analyticsvidhya.com/blog/2020/05/decision-tree-vs-random-forest-algorithm/

[6] A. Liaw and M. Wiener, “Classification and Regression by randomForest,” *Classification and Regression by randomForest*, vol. 2, no. 3, 2002, Available: https://journal.r-project.org/articles/RN-2002-022/RN-2002-022.pdf

[7] P. Hoang Vuong, T. Tan Dat, T. Khoi Mai, P. Hoang Uyen, and P. The Bao, “Stock Price Forecasting Based on XGBoost and LSTM,” *Computer Systems Science and Engineering*, vol. 40, no. 1, pp. 237–246, 2022, doi: https://doi.org/10.32604/csse.2022.017685.

[8] T. E. Times, “What is Relative Strength Index? Definition of Relative Strength Index, Relative Strength Index Meaning,” *The Economic Times*, Feb. 18, 2024. https://economictimes.indiatimes.com/definition/relative-strength-index (accessed Feb. 18, 2024).

[9] S. Charts, “Stochastic Oscillator [ChartSchool],” *school.stockcharts.com*, Mar. 16, 2022. https://school.stockcharts.com/doku.php?id=technical_indicators:stochastic_oscillator_fast_slow_and_full (accessed Feb. 18, 2024).

[10] Groww, “What is MACD? Meaning and How to Read MACD?,” *Groww*, Jun. 29, 2021. https://groww.in/p/what-is-macd

[11] O. Tkachenko and J. Kane, “Price Rate Of Change Indicator - Formula and ROC Trading Strategies,” *LiteFinance*, Feb. 10, 2022. https://www.litefinance.org/blog/for-beginners/best-technical-indicators/price-rate-of-change/ (accessed Feb. 19, 2024).

[12] T. Ameritrade, “Learning Center - OnBalanceVolume,” *tlc.thinkorswim.com*, Apr. 07, 2021. https://tlc.thinkorswim.com/center/reference/Tech-Indicators/studies-library/O-Q/OnBalanceVolume (accessed Feb. 19, 2024).

[13] C. MITCHELL, S. SILBERSTEIN, and K. MUNICHIELLO, “Williams %R Definition and Uses,” *Investopedia*, Sep. 21, 2021. https://www.investopedia.com/terms/w/williamsr.asp (accessed Feb. 18, 2024).

[14] “A complete explanation of Bollinger Bands,” *bollingerbands*, May 21, 2021. https://www.bollingerbands.com/bollinger-bands (accessed Feb. 18, 2024).

[15] O. Tkachenko and J. Kane, “Average Directional Index Explained. What Is the ADX Indicator? ,” *LiteFinance*, Feb. 07, 2024. https://www.litefinance.org/blog/for-beginners/best-technical-indicators/adx-indicator-average-directional-index/ (accessed Feb. 18, 2024).

[16] C. MITCHELL, A. GANTI, and A. COURAGE, “Wilder’s DMI (ADX) Indicator: Definition and Calculation Formula,” *Investopedia*, Sep. 18, 2022. https://www.investopedia.com/terms/w/wilders-dmi-adx.asp (accessed Feb. 19, 2024).

[17] Q. Hu, S. Qin, and S. Zhang, “Comparison of Stock Price Prediction Based on Different Machine Learning Approaches,” *Atlantis Highlights in Intelligent Systems*, pp. 215–231, Dec. 2022, doi: https://doi.org/10.2991/978-94-6463-030-5_24.

[18] Y. Shynkevich, T. M. McGinnity, S. A. Coleman, A. Belatreche, and Y. Li, “Forecasting price movements using technical indicators: Investigating the impact of varying input window length,” *Neurocomputing*, vol. 264, pp. 71–88, Nov. 2017, doi: https://doi.org/10.1016/j.neucom.2016.11.095.

[19] R. Sodhi, “YFinance ignoring start date when end date is today · Issue #1272 · ranaroussi/yfinance,” *GitHub*, Jan. 01, 2023. https://github.com/ranaroussi/yfinance/issues/1272

[20] G. C. Zotin, “GZotin/RSI_MACD_strategy,” *GitHub*, Apr. 07, 2024. https://github.com/GZotin/RSI_MACD_strategy (accessed Apr. 22, 2024).

[21] stat muse, “Palantir Technologies Inc Stocks In Nov 2023 | StatMuse Money,” *StatMuse*, Nov. 29, 2023. https://www.statmuse.com/money/ask/palantir-technologies-inc-stocks-in-nov-2023

[22] S. Tonight, “What is Mean Squared Error, Mean Absolute Error, Root Mean Squared Error and R Squared? - Studytonight,” *www.studytonight.com*, Mar. 29, 2023. https://www.studytonight.com/post/what-is-mean-squared-error-mean-absolute-error-root-mean-squared-error-and-r-squared#:~:text=MSE%20and%20MAE%20report%20the

[23] S. Learn, “3.2. Tuning the hyper-parameters of an estimator — scikit-learn 0.22 documentation,” *Scikit-learn.org*, 2012. https://scikit-learn.org/stable/modules/grid_search.html

[24] Dataquest, “Predict The Stock Market With Machine Learning And Python,” *YouTube*. May 23, 2022. Available: https://www.youtube.com/watch?v=1O_BenficgE

.

10. Appendix: Apple Stock

.

```
# using yahooo finance api to extract apple stock stock market data
# Small account, I do not want to day trade. I am using a small account to capture the overall trend and see some nice returns
# Swing trading holding positions for several days to weeks. Therefore, to capture these trends. I will be using
# the daily interval.
data = yf.download('AAPL',
                   start='2022-01-01',
                   end='2024-01-01',
                   interval='1d'
)
data.head(5)
```

```
# Calculating simple moving averages for Apple's closing price
data['SMA_1'] = data['Close'].rolling(window=1).mean() # short timeframe
data['SMA_5'] = data['Close'].rolling(window=5).mean() # short timeframe
data['SMA_12'] = data['Close'].rolling(window=12).mean() # short timeframe
data['SMA_24'] = data['Close'].rolling(window=24).mean() # short/medium timeframe

fig, (ax1,ax2) = plt.subplots(2,1, figsize= (10,8), sharex=True) # Creating a subplot with 2 axes one for technical indicators and the other is Apple's closing stock price for comparison

ax1.plot(data['SMA_12'], label = 'SMA 12',color='blue') # Plot SMA values for the first axis
ax1.plot(data['SMA_24'], label = 'SMA 24', color='red')
ax1.set_ylabel('SMA and Apple Closing Price', color = 'Black')

ax1.legend(loc='upper left',bbox_to_anchor=(1, 1)) # Display legend for the first axis

ax2.plot(data['Close'], label = 'Apple Stock Closing Price', color = 'Red')  #Plot Apple Closing price values on the second axis
ax2.set_ylabel('Apple Closing Price')

ax2.legend(loc='upper left',bbox_to_anchor=(1, 1)) # Display legend for the second axis

# Plotting finishing touches
fig.suptitle('Technical Indicator Comparison - MACD/SingalLine and Apple Closing Price')
plt.xlabel('Timeline')
plt.show()
```

```
data['EMA_9'] = data['Close'].ewm(9).mean() # Common way to calcluate EMA, common timeframe = 9 for short term trading. Usually combined with SMA for bullish/bearish crossovers

fig, (ax1,ax2) = plt.subplots(2,1, figsize= (10,8), sharex=True) # Creating a subplot with 2 axes one for technical indicators and the other is Apple's closing stock price for comparison

ax1.plot(data['EMA_9'], label = 'EMA_9',color='blue') # Plot EMA values for the first axis
ax1.set_ylabel('EMA and Apple Closing Price', color = 'Black')

ax1.legend(loc='upper left',bbox_to_anchor=(1, 1)) # Display legend for the first plot as well

ax2.plot(data['Close'], label = 'Apple Stock Closing Price', color = 'Red')  # Plot Apple Closing price values on the second axis
ax2.set_ylabel('Apple Closing Price')

ax2.legend(loc='upper left',bbox_to_anchor=(1, 1)) # Display legend for the second axis

# Plotting finishing touches
fig.suptitle('Technical Indicator Comparison - MACD/SingalLine and Apple Closing Price')
plt.xlabel('Timeline')
plt.show()
```

.

```
# taking a small group of my dataset this case 14. and performing a calculation on that dataset
# my calc is .min(). As I am going through the data set I am taking the minimum value within that dataset
# .rolling().max() takes the maximum value within the rolling window of 14 tuples, this will continue in every window
# until .rolling reaches the end of the dataset

L14 = data['Low'].rolling(window=1).min()
H14 = data['High'].rolling(window=1).max() # what this does is create a new column called h14. Selecting the data['high' column using the rolling function that takes the windows of 14 tuples and runs the .max() function to extract the max value in every14 tuple window
data['%K 1'] = 100 * ((data['Close']- L14) / (H14-L14))


L14 = data['Low'].rolling(window=5).min()
H14 = data['High'].rolling(window=5).max() # what this does is create a new column called h14. Selecting the data['high' column using the rolling function that takes the windows of 14 tuples and runs the .max() function to extract the max value in every14 tuple window
data['%K 5'] = 100 * ((data['Close']- L14) / (H14-L14))

L14 = data['Low'].rolling(window=14).min()
H14 = data['High'].rolling(window=14).max() # what this does is create a new column called h14. Selecting the data['high' column using the rolling function that takes the windows of 14 tuples and runs the .max() function to extract the max value in every14 tuple window
data['%K 14'] = 100 * ((data['Close']- L14) / (H14-L14))



L14 = data['Low'].rolling(window=21).min()
H14 = data['High'].rolling(window=21).max() # what this does is create a new column called h14. Selecting the data['high' column using the rolling function that takes the windows of 14 tuples and runs the .max() function to extract the max value in every14 tuple window
data['%K 21'] = 100 * ((data['Close']- L14) / (H14-L14))


# fig is the entire figure, ax1 ax2 are the subplots, use .plt.subplots to create the subplots stacked vertically (2,1) 2 rows 1 column
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8), sharex=True)

ax1.plot(data['%K 14'], label='%K 14', color='blue') # Plotting %K on the first subplot
ax1.set_ylabel('%K 14 Value', color='black')
ax1.tick_params('y', colors='blue')

# this article helped me with plotting overbought and oversold references for the first subplot (SO)
# https://rbdundas.medium.com/calculate-stochastic-oscillator-in-python-and-pandas-and-chart-with-matplotlib-aafde26b4a1f
ax1.axhline(y=80, color='r', linestyle='--', label='Overbought (80)')
ax1.axhline(y=20, color='g', linestyle='--', label='Oversold (20)')

ax1.legend(loc='upper left') # Display legend for the first axis


ax2.plot(data['Close'], label='Apple Closing Price', alpha=0.5, color='green') # Plotting Apple's closing price on the second subplot
ax2.set_ylabel('Apple Closing Price', color='green')
ax2.tick_params('y', colors='green')

ax2.legend(loc='upper left') # Display legend for the second axis

# Plotting finishing touches
fig.suptitle('Stochastic Oscillator vs Apple Closing Price')
plt.xlabel('Timeline')
plt.show()
```

```
# Source: https://stackoverflow.com/questions/20526414/relative-strength-index-in-python-pandas
# Set the value of n to 14. Used to calculate RSI (14)
n = 1

# Calculate the change in price by subtracting Apple stock's previous closing price from the current closing day price
# For example: 2021-01-05 closing price and takes the difference of 131.009995 ( 2021-01-05) - 129.410004 ( 2021-01-04) = 1.599991
data['Change_in_price'] = data['Close'].diff()

# Create two variables that contain the change in price regarding Apple stock that separates positive and negative changes
up_day = data['Change_in_price'].copy()
down_day = data['Change_in_price'].copy()

# sets all the values that are less than 0 to 0.
# Eliminating all negative values and only containing only up days
up_day[up_day < 0] = 0

# sets all values that are greater than 0 to 0
# Eliminating all positive values and only containing only down days
down_day[down_day > 0] = 0

# take the absolute value of the variable down_day, making all values positive
down_day = down_day.abs()

# Calculate the Exponential Weighted Moving Average (EMWA) for up days using the span of n
up_emwa = up_day.ewm(span=n, min_periods=n).mean()

# Calculate the Exponential Weighted Moving Average (EMWA) for up days using the span of n
down_emwa = down_day.ewm(span=n, min_periods=n).mean()

# Calculate relative strength
relative_strength = up_emwa / down_emwa

# Calculate Relative Strength Index (RSI)
rsi = 100.0 - (100.0 / (1.0 + relative_strength))

# Add RSI 1 values to the dataframe
data['RSI 1'] = rsi

fig, (ax1,ax2) = plt.subplots(2,1, figsize= (8,8), sharex=True) # Create a subplot, with two axes stacked on eachother (2 rows, 1 column)

ax1.plot(data['RSI 14'], label='RSI Values', color='blue') # Plot RSI values for the first axis
ax1.set_ylabel('RSI Values')

# Add horizontal dotted lines to indicate extremly overbought and oversold levels
# Source: Assisted me with the inclusion of the horizontal lines https://rbdundas.medium.com/calculate-stochastic-oscillator-in-python-and-pandas-and-chart-with-matplotlib-aafde26b4a1f
ax1.axhline(y=80, color = 'red', linestyle='--',label = 'Extremely Overbought (80) ')
ax1.axhline(y=20, color='green', linestyle='--', label = 'Extremly Oversold (20) ')

ax1.legend(loc='upper left',bbox_to_anchor=(1, 1)) # Display legend for the first axis

ax2.plot(data['Close'],label='Apple Closing Price Values', color='red') # Plot Apple Closing price values on the second axis
ax2.set_ylabel('Apple Closing Price')

ax2.legend(loc='upper left', bbox_to_anchor=(1, 1)) # Display legend for the second axis

# Plotting finishing touches
fig.suptitle('Technical Indicator Comparison - RSI and Apple Closing Price')
plt.xlabel('Timeline')
plt.show()
```

.

```
# Moving Average Convergence Divergence MACD
# Source:  https://tcoil.info/compute-macd-indicator-for-stocks-with-python/
# Most common way to calculate MACD 9,12,26
ema_26 = data['Close'].ewm(span=26).mean()
ema_12 = data['Close'].ewm(span=12).mean()

data['MACD'] = ema_12 - ema_26
data['SignalLine'] = data['MACD'].ewm(span = 9).mean()

# Same principle for the figure
fig, (ax1,ax2) = plt.subplots(2,1, figsize= (10,8), sharex=True)

ax1.plot(data['MACD'], label = 'MACD',color='blue')
ax1.plot(data['SignalLine'], label = 'Signal Line', color='red')
ax1.set_ylabel('MACD VS SignalLine Values', color = 'Black')

ax1.legend(loc='upper left',bbox_to_anchor=(1, 1))


ax2.plot(data['Close'], label = 'Apple Stock Closing Price', color = 'Red')
ax2.set_ylabel('Apple Closing Price')


ax2.legend(loc='upper left',bbox_to_anchor=(1, 1))


fig.suptitle('Technical Indicator Comparison - MACD/SingalLine and Apple Closing Price')
plt.xlabel('Timeline')

plt.show()


# essentially if MACD is > the Signalline column than this indicates a buy signal, vice versa this indicates a sell signal
# if the MACD (Blue line) is above the SignalLine (red line) than that is indicating a bullish signal
# if the MACD is below the signalline than that is indicating a bearish signal
```

.

```
# Williams Percentage (W%R)
# Calculate the highest high over the past 1 day and store it into the 'H14' column
H14 = data['High'].rolling(window=1).max()

# Calculate the lowest low over the past 1 day and store it into the 'L14' column
L14 = data['Low'].rolling(window=1).min()

# Calculate the Williams Percent Range and store the values in the column "W%R"
data['W%R 1'] = (H14 - data['Close']) / (H14 - L14) * -100


H14 = data['High'].rolling(window=3).max()
L14 = data['Low'].rolling(window=3).min()
data['W%R 5'] = (H14 - data['Close']) / (H14 - L14) * -100


H14 = data['High'].rolling(window=14).max()
L14 = data['Low'].rolling(window=14).min()
data['W%R 14'] = (H14 - data['Close']) / (H14 - L14) * -100


H14 = data['High'].rolling(window=21).max()
L14 = data['Low'].rolling(window=21).min()
data['W%R 21'] = (H14 - data['Close']) / (H14 - L14) * -100


fig, (ax1,ax2) = plt.subplots(2,1, figsize= (10,8), sharex=True)

ax1.plot(data['W%R 14'], label = 'W%R 14', color='blue')
ax1.set_ylabel('W%R Value')


# overbought/oversold conditions -80, -20 respectively
ax1.axhline(y=-20,color='r', linestyle='--', label = 'Overbought (-20) ')
ax1.axhline(y=-80,color='g', linestyle='--', label='Oversold (-80) ')

ax1.legend(loc='upper left', bbox_to_anchor=(1, 1))

ax2.plot(data['Close'], label = 'Apple Closing Price', color='red')
ax2.set_ylabel('Apple Closing Price')


ax2.legend(loc='upper left', bbox_to_anchor=(1, 1))

fig.suptitle('Technical Indicator Comparison - W%R and Apple Closing Price')
plt.xlabel('Timeline')
plt.show()
```

.

```python
def obv(group):
  # Take the volume and the difference in close price for Apple
    volume = data['Volume']
    change = data['Close'].diff()

    # Initialize the previous OBV value
    prev_obv = 0
    # empty list to store calcluated OBV values
    obv_values = []

    # looping through the pairs of change and volume values
    for i, j in zip(change, volume):
        if i > 0:  # checks if the price change is positive (i > 0), negative (i<0), or 0
            current_obv = prev_obv + j # If the price is positive add previous obv and current volume to curernt_obv
        elif i < 0:
            current_obv = prev_obv - j # If the price is negative subtract the previous obv and the current volume store this information in current_obv
        else:
            current_obv = prev_obv # for 0 price change, keep obv the same

        prev_obv = current_obv # Updating previous obv value
        obv_values.append(current_obv) # append current OBV value to the list
    # Creating a Pandas Series with OBV values, maintain apple's index, and name 'On Balance Volumne'
    return pd.Series(obv_values, index=data.index, name='On Balance Volumne')

# Calling the obv function with the Apple stock's DataFrame and storing the result in the variable obv_values
obv_values = obv(data)

data['OBV'] = obv_values # creating a new column called OBV, that contains the calculated OBV values

fig, (ax1,ax2) = plt.subplots(2,1, figsize= (10,8), sharex=True)

ax1.plot(data['OBV'], label='On Balance Volumne',color='red')
ax1.set_ylabel('On Balance Volume Values')

ax1.legend(loc='upper left')

ax2.plot(data['Close'], label = 'Apple Closing Price', color='blue')
ax2.set_ylabel('Apple Closing Price', color='black')
ax2.tick_params('y', colors='black')


ax2.legend(loc='upper left')


fig.suptitle('Technical Indicator Comparison - OBV and Apple Closing Price')
plt.xlabel('Timeline')

# note volume is equal to 10^9 in the billions. .25 (1/4) of a billion is 250m volume
# -1.5 is -1.5b volumne
# Positive OBV is an indicator that the cumulative volume is tilted towards buyng. Flahses that more volume has
# traded on days when the price closed higher
# Negative cumulative volume is tilited towards selling. Suggests that more volume has traded on days when the price closed lower
```

```python
n = 1 # window size of 3
data['PROC 1'] = (data['Close'] - data['Close'].shift(-n)) / data['Close'].shift(-n) # calculating PROC using the same formula given in my paper
# further breakdown essentially you are taking todays closing price subtracting the closing price of 30 days ago then dividing it by the closing price of 1 day ago. The output is the % gain or % loss


n = 5
data['PROC 5'] = (data['Close'] - data['Close'].shift(-n)) / data['Close'].shift(-n)


n = 14
data['PROC 14'] = (data['Close'] - data['Close'].shift(-n)) / data['Close'].shift(-n)


n = 21
data['PROC 21'] = (data['Close'] - data['Close'].shift(-n)) / data['Close'].shift(-n)


fig, (ax1,ax2) = plt.subplots(2,1, figsize= (10,8), sharex=True)

ax1.plot(data['PROC 14'], label = 'Price Rate of Change 14', color = 'red')
ax1.set_ylabel('PROC 14 Values')

ax1.legend(loc='upper left')

ax2.plot(data['Close'], label='Apple Closing Price', color='blue')
ax2.set_ylabel('Apple Closing Price')

ax2.legend(loc='upper left')


fig.suptitle('Technical Indicator Comparison - PROC 14 and Apple Closing Price')
```

.

```
# Srouce: https://github.com/GZotin/RSI_MACD_strategy

oversold_threshold = {'RSI 21': 30} # Thresholds to identify over_sold and over_bought levels
overbought_threshold = {'RSI 21': 75}

buy_signals = [] # Initialize lists to store buy and sell signals
sell_signals = []


for i in range(len(data)): # Iterate through the data and generate buy/sell signals
  signal = '' # Initialize an empty string to store signals for the current loop iteration

  for indicator, threshold in oversold_threshold.items(): # Checking for oversold/overbought conditions
    if data[indicator].iloc[i] < threshold:
      signal += 'Oversold '
  for indicator, threshold in overbought_threshold.items():
    if data[indicator].iloc[i] > threshold:
      signal += 'Overbought '


  if data['MACD'].iloc[i] > data['SignalLine'].iloc[i] and data['MACD'].iloc[i-1] <= data['SignalLine'].iloc[i-1]: # Checking for bullish crossing
    signal += 'Bullish Cross '
  elif data['MACD'].iloc[i] < data['SignalLine'].iloc[i] and data['MACD'].iloc[i-1] >= data['SignalLine'].iloc[i-1]:
    signal += 'Bearish Cross '


  # Buy signal only when both RSI and MACD conditions are met
  if 'Oversold' in signal and 'Bullish Cross' in signal:
    buy_signals.append((data.index[i], data['Close'].iloc[i]))

  # Sell signal based on similar logic (modify as needed)
  if 'Overbought' in signal and 'Bearish Cross' in signal:
    sell_signals.append((data.index[i], data['Close'].iloc[i]))


# Plotting section (same as before)
plt.figure(figsize=(12, 6))
plt.plot(data.index, data['Close'], label='Close Price', color='blue')

if buy_signals:
  buy_dates, buy_prices = zip(*buy_signals)
  plt.scatter(buy_dates, buy_prices, color='green', label='Buy Signal', marker='^', s=100)

if sell_signals:
  sell_dates, sell_prices = zip(*sell_signals)
  plt.scatter(sell_dates, sell_prices, color='red', label='Sell Signal', marker='v', s=100)

# Plotting the finishing touches (same as before)
plt.xlabel('Date')
plt.ylabel('Price')
plt.title('Buy/Sell Signals Based on Both RSI and MACD Crossings')
plt.legend()
plt.show()
```

```
# Source that helped me with this. https://jakevdp.github.io/PythonDataScienceHandbook/03.11-working-with-time-series.html
new_aapl = yf.download('AAPL', start = '2020-01-01', end = '2024-01-01')

aapl_quart_performance = new_aapl.resample('Q').mean() # Resample data to quarterly fashion and calculate mean for each quarter

avg_closing_prices = new_aapl.groupby(new_aapl.index.quarter)['Close'].mean() # Calculate the average closing price for each quarter

best_quarter = avg_closing_prices.idxmax() # Identify the quarter with the maximum average closing price
best_quarter = (best_quarter)

aapl_quart_performance['Date'] = aapl_quart_performance.index.to_period('Q').strftime('%D-Q%q') # Updating the date index column to represent the start of each quarter with the formatted string (year then quarter)

plt.figure(figsize=(10,6)) # Plot Apple's quarter by performance
plt.bar(aapl_quart_performance['Date'], aapl_quart_performance['Close'],color='blue')
plt.title('Apple Quarter by Performance')
plt.xlabel('Quarters')
plt.ylabel('Average Closing Price')
plt.xticks(rotation = 45, ha='right')
plt.show()

print('On average Apple generally performs the best in Quarter:', best_quarter)
```

.

```
# includes all technical indicator correlations with Apple stock closing price
RSI_corr = data['RSI 14'].corr(data['Close'])
print('RSI correlation with Apple Closing Price:', RSI_corr, '\n')

so_corr = data['%K 14'].corr(data['Close'])
print('Stochastic Oscillator correlation with Apple Closing Price:', so_corr, '\n')

williams_corr = data['W%R 14'].corr(data['Close'])
print('W%R correlation with Apple Closing Price:', williams_corr, '\n')

macd_corr = data['MACD'].corr(data['Close'])
print('MACD correlation with Apple Closing Price:', macd_corr, '\n')

signal_line_corr = data['SignalLine'].corr(data['Close'])
print('SignalLine correlation with Apple Closing Price:', signal_line_corr, '\n')

macd_signalLine_corr = data['MACD'].corr(data['SignalLine'])
print('MACD correlation with SignalLine ', macd_signalLine_corr, '\n')

obv_corr = data['OBV'].corr(data['Close'])
print('OBV correlation with Apple Closing Price:', obv_corr, '\n')

obv_descriptive = data['OBV'].describe() # All descriptive stats for technical indicators regarding Apple stock
print('Descriptive Statistics for OBV', obv_descriptive)
rsi_descriptive = data['RSI 14'].describe()
print('Descriptive Statistics for RSI:', rsi_descriptive)
WR_descriptive = data['W%R 14'].describe()
print('Descriptive Statistics for W%R', WR_descriptive)
macd_descriptive = data['MACD'].describe()
print('Descriptive Statistics for MACD: ', macd_descriptive)
sigLine_stats = data['SignalLine'].describe()
print('Descriptive Statistics for SignalLine', sigLine_stats)
```

.

```
data['Next_Close'] = data['Close'].shift(-1) # Creating a target variable called Next_Close. The idea here is to use todays data to predict tomorrows closing price

data = data.dropna() # Random Forest is not happy with NaN values, therefore I drop all NaN values in Apple stock's data frame.

predictors = ['SMA_1','SMA_5','PROC 1'] # Using my technical indicators as my predictors. To assist my training data to predict my testing data.

target_variable = 'Next_Close'

x = data[predictors] # Splitting my data into predictors x and target variable y
y = data[target_variable]

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42) # Splitting the data into 80% training and 20% testing

model = RandomForestRegressor(n_estimators=350, min_samples_split=3, max_depth=5) # Initialize RandomForestRegressor


model.fit(x_train, y_train) # Train the model

preds = model.predict(x_test) # Make predictions based on the trained model

combined = pd.DataFrame({'Actual': y_test, 'Predicted': preds})
combined.plot() # Plot the actual and predicted values
plt.xlabel('Index')
plt.ylabel('Close Price')
plt.title('Actual vs Predicted Close Price')
plt.legend()
plt.show()

# Performance Metrics Random Forest Regressor Model:
# Source: https://www.youtube.com/watch?v=YUsx5ZNlYWc
mae_train = mean_absolute_error(y_train, model.predict(x_train))
mae_test = mean_absolute_error(y_test, preds)
mse = mean_squared_error(preds, y_test)
r2 = r2_score(preds, y_test)

print("Training Mean Aboslute Error:", mae_train)
print("Test Mean Aboslute Error:", mae_test)
print('Mean Squared Error', mse)
print('R2 Score', r2)
```

```
total_samples = len(data)
train_samples = int(total_samples * 0.8)  # 80% of Apple stock data will be used for training

split_index = train_samples  # Variable used to split Apple stock data

# Split the Apple stock data into training and testing splits
train_data = data[:split_index]  # All Apple stock data up to split_index for training
test_data = data[split_index:]  # All Apple stock data from split_index and so on for testing

train_data['Target'] = train_data['Close'].shift(-1) # Creating the target variable
test_data['Target'] = test_data['Close'].shift(-1)

train_data = train_data.dropna() # Random Forest Regressor is not happy with NaN values
test_data = test_data.dropna()

predictors = ['RSI 14', 'PROC 5', 'MACD', 'OBV', 'EMA_9','%K 14','SMA_12','SignalLine','W%R 14','High', 'Low', 'Adj Close', 'Open', 'Close']

x_train = train_data[predictors] # Splitting the data into predictors (x) and target variable (y)
y_train = train_data['Target']  # Target variable is the close price for the next day
x_test = test_data[predictors]
y_test = test_data['Target']

model = RandomForestRegressor(n_estimators=500, min_samples_split=5, max_depth=5) # Initialize Random Forest Regressor model with optimized hyperparameters

model.fit(x_train, y_train) # Train the model

preds = model.predict(x_test) # Use the trained model to make predictions

combined_test_results = pd.DataFrame({'Actual Closing Price': y_test, 'Predicted Closing Price': preds}, index=y_test.index)

combined_test_results.plot() # Plot the actual and predicted values for the testing set
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.title('Apple Actual Close Prices vs Predicted Close Prices (Testing Set)')
plt.legend()
plt.show()

mae_train = mean_absolute_error(y_train, model.predict(x_train)) # Evaluate the model's performance based on common performance metrics
mae_test = mean_absolute_error(y_test, preds)
mse = mean_squared_error(y_test, preds)
r2 = r2_score(y_test, preds)

print("Training Mean Absolute Error:", mae_train)
print("Test Mean Absolute Error:", mae_test)
print('Mean Squared Error:', mse)
print('R2 Score:', r2)
```

.

```
preds_list = list(x_train.columns) # Creating a list of predictor names from the training set
preds_importance = pd.Series(model.feature_importances_, index=preds_list).sort_values(ascending=False) # Creating a panda series of predictor importances, order by DESC

importances = preds_importance.nlargest(15).plot(kind='bar') # Plotting all the predictors
plt.xlabel('Predictors')
plt.ylabel('Importance')
plt.title('Top 8 Predictor Importances')
plt.show()
```

```
total_samples = len(data)
train_samples = int(total_samples * 0.8)  # 80% of the Apple stock data will be used for training

split_index = train_samples  # Variable used to split Apple stock data

# Split the Apple stock data into training and testing splits
train_data = data[:split_index]  # All Apple stock data up to split_index for training
test_data = data[split_index:]   # All Apple stock data from split_index and so on for testing

train_data['Next_Close'] = train_data['Close'].shift(-1) # Creating the target variable
train_data['Target'] = (train_data['Next_Close'] > train_data['Close']).astype(int)

test_data['Next_Close'] = test_data['Close'].shift(-1)
test_data['Target'] = (test_data['Next_Close'] > test_data['Close']).astype(int)

train_data = train_data.dropna() # XGBoost Classifier is not happy with NaN values
test_data = test_data.dropna()

predictors = ['RSI 14', 'PROC 5', 'MACD', 'OBV', 'EMA_9', '%K 14', 'SMA_5', 'SignalLine', 'W%R 14']


x_train = train_data[predictors] # Splitting the data into predictors (x) and target variable (y)
y_train = train_data['Target'] # Target variable is the close price for the next day
x_test = test_data[predictors]
y_test = test_data['Target']

model = xgb.XGBClassifier(n_estimators=300, max_depth=1, learning_rate=0.1, min_child_weight=1) # Initialize XGBoost Classifier

model.fit(x_train, y_train) # Train the model

preds = model.predict(x_test) # Used the trained model to make predictions

# Calculating Classification Performance Metrics
accuracy = accuracy_score(y_test, preds)
precision = precision_score(y_test, preds)
recall = recall_score(y_test, preds)
f1 = f1_score(y_test, preds)

print("Accuracy:", accuracy) # Printing Classifcation Performance Metrics
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)

report = classification_report(y_test, preds) # Creating a classification report
print("Classification Report: ")
print(report)
```

.

```
total_samples = len(data)
train_samples = int(total_samples * 0.8)  # 80% of the Apple stock data will be used for training

split_index = train_samples  # Variable used to split Apple stock data

# Splitting Apple stock data into training and testing splits
train_data = data[:split_index]  # All Apple stock data up to split_index for training
test_data = data[split_index:]   # All Apple stock data from split_index and so on for testing

train_data['Next_Close'] = train_data['Close'].shift(-1) # Creating target variables
train_data['Target'] = (train_data['Next_Close'] > train_data['Close']).astype(int)

test_data['Next_Close'] = test_data['Close'].shift(-1)
test_data['Target'] = (test_data['Next_Close'] > test_data['Close']).astype(int)

train_data = train_data.dropna() # RFC is not happy with NaN values
test_data = test_data.dropna()

features = ['RSI 5', 'PROC 5', 'MACD', 'OBV', 'EMA_9','%K 5','SMA_5','SignalLine','W%R 5']


x_train = train_data[features]  # Splitting the data into features (x) and target variable (y)
y_train = train_data['Target']  # Target variable is the close price for the next day
x_test = test_data[features]
y_test = test_data['Target']


model = RandomForestClassifier(n_estimators=200, max_depth=1, min_samples_leaf=3, min_samples_split=4) # Initialize optimized Random Forest Classifier

model.fit(x_train, y_train) # Training the model

preds = model.predict(x_test) # Use the trained model to make predictions


accuracy = accuracy_score(y_test, preds)
precision = precision_score(y_test, preds)
recall = recall_score(y_test, preds)
f1 = f1_score(y_test, preds)

print("Accuracy:", accuracy) # Printing Classifcation Performance Metrics
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)

report = classification_report(y_test, preds) # Creating a classification report
print("Classification Report: ")
print(report)
```

```
# Defining optiized hyperparameter grid
param_grid = {
    'n_estimators': [100, 150, 200, 250, 300],
    'max_depth': [1, 3, 5, 10],
    'min_samples_leaf': [1, 2, 3],
    'min_samples_split': [2, 3, 4]
}


grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring='accuracy') # Performed grid search the different models
grid_search.fit(x_train, y_train)


best_params = grid_search.best_params_ # Retrieves and print best hyperparameters I did this for every model
print("Best Parameters:", best_params)


best_model = grid_search.best_estimator_ # Use the best model for predictions
best_preds = best_model.predict(x_test)
```

.