Predicting Bitcoin Price Movement Utilizing Individual Stock Prices

Paola Rodriguez, Tommy Barron

University of San Diego

ADS-502-01: Applied Data Mining

An Tran

December 9, 2024

Abstract

The increasing volatility of cryptocurrency markets, particularly Bitcoin, has created a demand for predictive models that can provide insights into price fluctuations. This study investigates the potential of using individual stock prices as predictors for Bitcoin price movements, leveraging machine learning models to analyze patterns and relationships. The dataset includes daily stock prices from major companies such as Apple, Tesla, Microsoft, Google, Nvidia, Berkshire Hathaway, Netflix, Amazon, and Meta, alongside Bitcoin's historical price data. We conducted exploratory data analysis (EDA) to identify trends, correlations, and distribution characteristics within the data. Subsequently, we implemented and evaluated multiple predictive models, including Naive Bayes, random forests, and neural networks, to determine their effectiveness in forecasting Bitcoin price changes. Additionally, lagged variables were incorporated to capture temporal dependencies in Bitcoin's price movement. The results indicate varying levels of predictive accuracy across the models, highlighting the challenges and opportunities in predicting Bitcoin price movements using stock market data. This paper contributes to the understanding of cross-market influences and proposes a framework for leveraging machine learning in financial prediction. The findings have implications for traders, investors, and researchers aiming to enhance their strategies in cryptocurrency markets.

Keywords: cryptocurrency markets, individual stock prices, predictive models, lagged variables

Table of Contents

Predicting Bitcoin Price Movement Utilizing Individual Stock Prices

The volatility of Bitcoin and its rapid growth as a digital asset have made it a focal point for predictive modeling. Understanding and anticipating its price movements is of significant interest to traders, analysts, and researchers. While various economic indicators and cryptocurrency-specific metrics already exist and have been used to forecast Bitcoin prices, this study explores the predictive potential of stock market data.

This paper examines whether individual stock prices, including those of major companies such as Apple, Tesla, and Microsoft, can be leveraged to forecast Bitcoin's daily price movements. By applying machine learning techniques, including neural networks, linear regression, and Naive Bayes, this analysis evaluates the relationship between stock market trends and Bitcoin price changes. The findings aim to contribute to financial modeling strategies and expand the understanding of cross-market correlations.

Methodology

The data that was chosen for this project is a collection of US stock market and commodities data from 2020-2024. The data contains 37 columns of numerical data, a date column, and an index column. The dates were not formatted correctly and were standardized in the YYYY-MM-DD format. The data was then split into a training and test data set where 75% of the data is in the training set and the rest is in the test set. The data was split exactly at 2023-01-31. It was determined that a date split is preferable instead of random sampling because in the domain of the stock market, a data scientist cannot use future data to predict future prices. Since there are no missing variables, the test and train numerical data were then standardized to assist with the models. For this project, the models will be focusing on using individual stock prices to predict Bitcoin price movement. To achieve this goal, Bitcoin prices were lagged by a day so the current stock prices can predict the predictor variable in the models. A rolling seven-day average for Bitcoin prices was calculated for each instance and each instance was compared to the last day; therefore, if the price increases, the movement column will indicate an increase and vice versa for decreasing prices. This creates the binary predictor variable of Bitcoin price movement where increases are 1 and decreases are 0. Finally, the train and test data were inspected for balancing issues and was confirmed to be balanced at 412 decreases and 341 increases for the train set and 133 decreases and 113 increases for the test set.

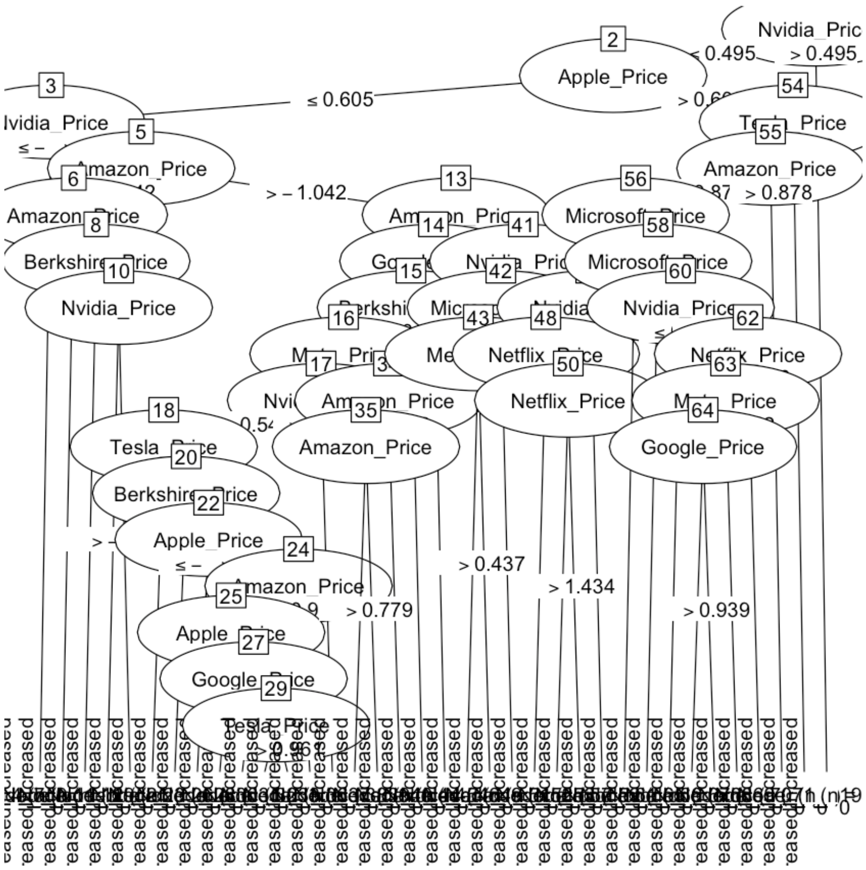
C5.0

The C5.0 decision tree highlights the relationship between individual stock prices and Bitcoin's price movement, identifying variables like Nvidia_Price, Amazon_Price, and Apple_Price as significant predictors. The tree's structure prioritizes Nvidia_Price as the initial decision node, emphasizing its importance in determining Bitcoin's direction. However, the model exhibits a bias toward predicting "Increase," as shown in the confusion matrix, where most "Decrease" instances are misclassified as "Increase." While the tree provides detailed decision boundaries based on stock price thresholds, its depth and complexity may lead to overfitting, reducing generalizability.

To improve the model, steps such as pruning the tree, balancing the training data, and performing cross-validation can address bias and enhance interpretability. Additionally, simplifying the tree and analyzing feature importance could help refine the predictors further. Overall, the model demonstrates potential in using individual stock prices to predict Bitcoin movements but requires optimization for better performance and reliability.

Figure 1

C5.0 Decision Tree Predicting Bitcoin Price

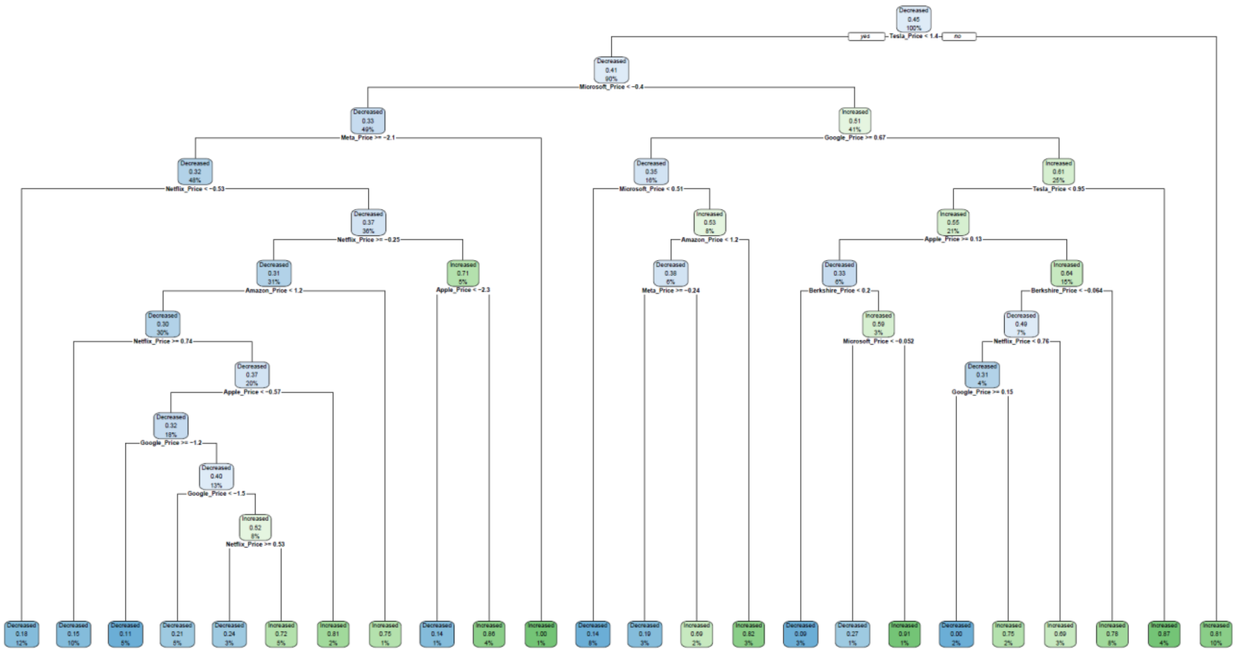


CART

The goal of the CART model is to find the best split of each input until a stopping rule is satisfied (Geeks for Geeks, 2024). In Figure 2, the CART decision tree has 24 leaf nodes and 23 decision nodes.

Figure 2

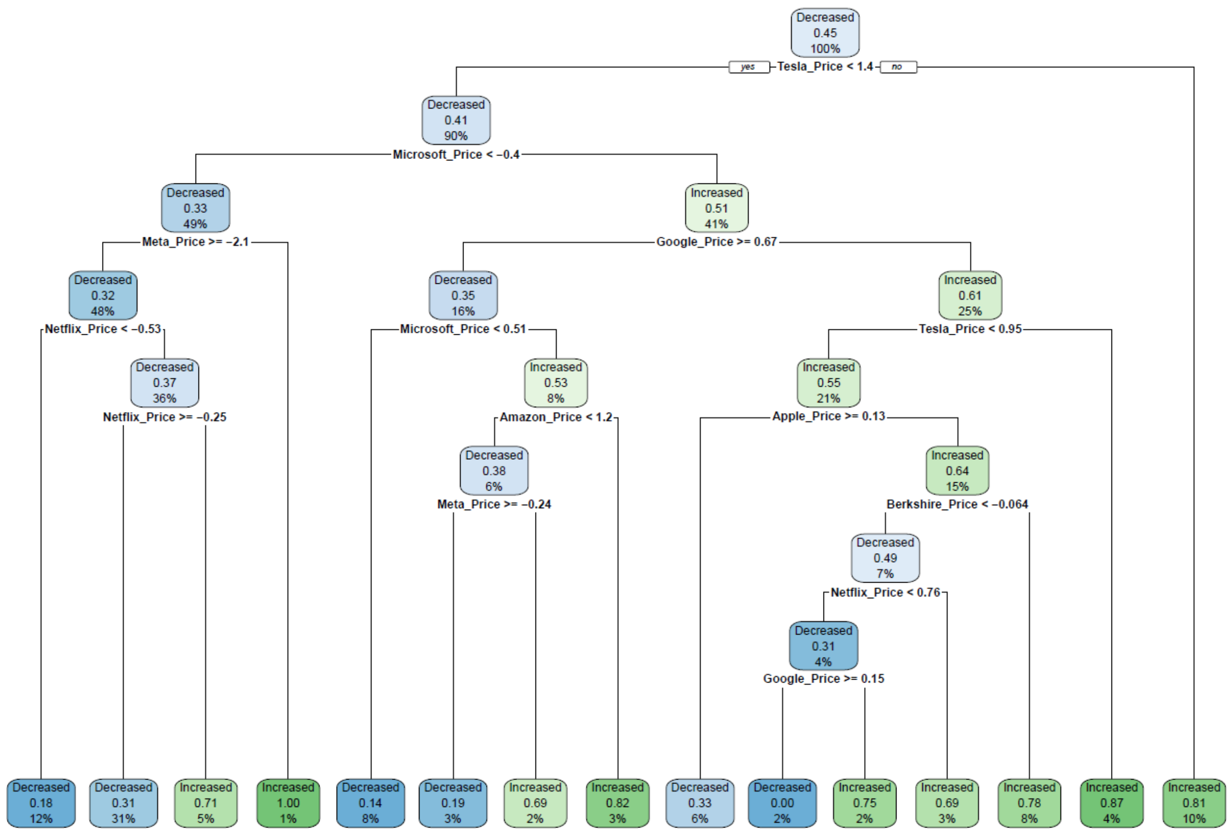
CART Decision Tree for Predicting Bitcoin Price Movement



The complex nature of the decision tree created indicates that there are some issues with overfitting due to high variance (Geeks for Geeks, 2024). Additional pruning and regularization of parameters will need to occur to limit these issues. Figure 3 is a second CART model that was developed using pruning tactics to understand the impact of certain variables. The model was pruned with a complexity parameter of 0.015 and has a classification threshold set at 0.3. The resulting decision tree has 15 leaves and 14 decision nodes with overall increased model evaluation metrics, most notably the F-scores.

Figure 3

Pruned CART Decision Tree for Predicting Bitcoin Price Movement



Logistic Regression

Between linear and logistic regression, logistic regression was chosen to best represent the data because of the binary target variable. The generalized logistic regression equation takes the form:

$$p(y) = \frac{\exp\left(\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p\right)}{1 + \exp\left(\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p\right)} + \varepsilon \tag{1}$$

Where β_p represents predictor variable coefficients and x_p represents the predictor variable value. Therefore, plugging in the values calculated from the logistic regression function yields:

$$\hat{p}(Movement) = \frac{\exp\,(0.3216 - 1.8567(Apple) + 0.8394(Tesla) + 1.6823(Microsoft)}{1 + \exp\,(0.3216 - 1.8567(Apple) + 0.8394(Tesla) + 1.6823(Microsoft)}$$

$$\frac{-1.4646(Google) + 0.8578(Nvidia) + 0.8860(Berkshire)}{-1.4646(Google) + 0.8578(Nvidia) + 0.8860(Berkshire)} \tag{2}$$

$$\frac{-0.6075(Netflix) + 0.2561(Amazon) + 0.4536(Meta))}{-0.6075(Netflix) + 0.2561(Amazon) + 0.4536(Meta))}$$

It is important to note that the logistic regression has calculated varying p-values for the predictor variables. The variables that showed the least significance at $\alpha > 0.05$ are Nvidia, Amazon, and Meta. Therefore, the variables that showed the greatest significance are Apple, Tesla, Microsoft, Google, Berkshire, and Netflix. When evaluating the regression model, the area under the receiver operating characteristic curve was calculated to measure the model's discrimination rate. The area under the curve came out to be 0.505 which indicates that this model is no better than random guessing.

A second logistic regression model was created after removing the statistically insignificant predictor variables (p-value > 0.05). While creating the model, the variables that were deemed unimpactful were Netflix, Amazon, Nvidia, and Meta. After these variables were removed the resulting area under the curve to be 0.5289. This indicates a "stronger" model but is still quite low and may need further development. The formula for the second model is then made to be:

$$\hat{p}(Movement) = \frac{\exp\,(0.3216 - 1.8567(Apple) + 0.8394(Tesla)}{1 + \exp\,(0.3216 - 1.8567(Apple) + 0.8394(Tesla)} \tag{3}$$

$$\frac{+1.6823(Microsoft) - 1.4646(Google) + 0.8860(Berkshire))}{+1.6823(Microsoft) - 1.4646(Google) + 0.8860(Berkshire))}$$

Random Forests

The goal of the random forest model is to combine multiple decision trees together into a single output (IBM, n.d.). With a relatively larger amount of predictor variables, the random forests model generated 500 trees. The model produced an accuracy of 54.07%, a sensitivity of 81.42%, a specificity of 30.83, a precision of 44.57%, and an AUC-ROC of 0.5612. During the development, it was speculated that adjusting the number of trees may improve the model. Two additional models were developed to test against the baseline. The first model had an “ntree” count of 1000 trees. When evaluating the model, only specificity went up from 0.3083 to 0.4812, but this comes at the cost of decreasing sensitivity from 0.8142 to 0.6460. The slight increase in specificity does not justify the drastic decrease in the overall metrics. The second model was developed using 100 trees. This model’s performance decreased with the three-evaluation metrics, sensitivity, specificity, and recall. Therefore, the baseline, 500 trees, is the preferred random forest model.

Naïve Bayes

The Naive Bayes model for predicting Bitcoin price movement based on individual stock prices demonstrates moderate accuracy at 50.42%. From the confusion matrix, it is evident that the model performs better at predicting "Increase" (85 correct predictions) compared to "Decrease" (35 correct predictions). However, the model struggles with misclassification, as 93 "Actual: Decrease" instances are predicted as "Increase," and 25 "Actual: Increase" instances are incorrectly classified as "Decrease."

Our basis for this approach hinges on Bayes Theorem:

$$p(X*) = \frac{p(Y = y*)p(Y = y*)}{p(X*)} \tag{4}$$

where x and y are the posterior probabilities (Larose, 2019, pp.113-114). This performance indicates that the Naive Bayes model has limited predictive power in this context, likely due to its assumption of feature independence, which may not hold true for stock prices that are often correlated. To improve accuracy, feature engineering, reducing noise in the data, or incorporating a different algorithm better suited for capturing complex relationships, such as Random Forest or Gradient Boosting, could enhance the predictive performance. Nonetheless, the model provides a baseline for understanding the relationship between stock prices and Bitcoin price movement.

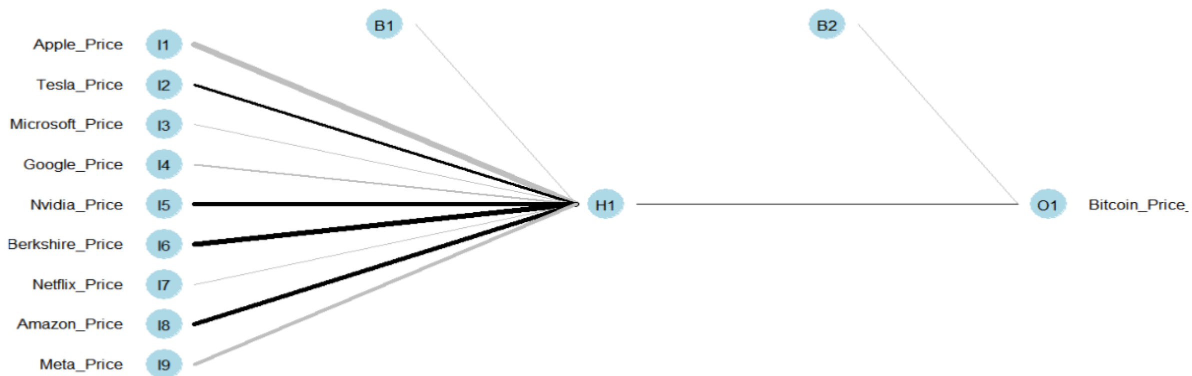
Neural Networks

The neural network model developed for this research analyzes the relationship between individual stock prices and Bitcoin price movement. The model consists of an input layer representing nine major stock prices (Apple, Tesla, Microsoft, Google, Nvidia, Berkshire Hathaway, Netflix, Amazon, and Meta), a hidden layer for processing these inputs, and an output layer predicting Bitcoin price movement. The connections between the input and hidden layers are weighted, reflecting the relative importance of each stock in influencing the model's predictions.

The results indicate that Nvidia and Berkshire Hathaway stock prices have the highest weights, suggesting they are the most influential in predicting Bitcoin price changes. This finding highlights potential stronger correlations between these specific stocks and Bitcoin's movement compared to others like Google or Meta, which have comparatively lower weights. By processing the non-linear relationships between these inputs, the model effectively captures complex dependencies, providing a nuanced understanding of how traditional equity markets interact with cryptocurrency prices. These insights could prove valuable for developing more robust predictive models in financial markets.

Figure 2

Neural Network Predicting Bitcoin Price



Results

The results shown in Table 1 display the model evaluation metrics shown for the models described in the methodology. To choose the best model, it is necessary for the model to perform well overall because predicting increases in Bitcoin movement is just as important as predicting deceases. The logistic regression model has a few well performing metrics of the F_1-score and the F_2-score at 61.92% and 75.53%, respectively. The relatively high F_1-score indicates the model has a good balance between precision and recall while the high F_2-score shows a particular strength with recall (Wood, n.d.). The C5.0 model has three strong metrics with the greatest sensitivity at 90.91%, specificity at 62.50%, and F_2-score at 75.76%. This implies that the C5.0 model performs well at classifying positive and negative records with a particular emphasis on recall rate. Although these metrics are relatively strong, the C5.0 performs the weakest with the highest error rate of 54.62% and one of the lowest precision scores of 45.45%. To choose a well-rounded model, it is important to have strong metrics, but also to limit the weakest metrics. Therefore, the best model to consider for this project is the CART decision tree. This model has the highest accuracy at 56.10%, precision at 52.03%, $F_{0.5}$-score at 52.89, and the greatest ROC-AUC at 56.14%. The high accuracy indicates that it has the highest proportion of correct to incorrect classifications. The $F_{0.5}$-score and, subsequently, precision demonstrates the model's ability to properly classify positives as true positives rather than false positives. Additionally, the greatest ROC-AUC indicates the highest proportion of actual positives and negatives identified from the data. Although the CART model has the lowest recall rate, it is important to note that the other models are not well-rounded and lack significantly in specificity and/or precision. Since both identifying increases and decreases in Bitcoin price movement is essential, the preferred method is the CART decision tree.

Table 1

Model Evaluation Table for Bitcoin Price Movement

Evaluation Measure	**C5.0**	**CART**	**Logistic Regression**	**Random Forest**	**Naïve Bayes**	**Neural Network**
Accuracy	0.4538	0.5610	0.5000	0.5407	0.5042	0.4664
Error Rate	0.5462	0.4390	0.5000	0.4593	0.5000	0.5340
Sensitivity	0.9091	0.5664	0.8850	0.8142	0.7727	0.7181
Specificity	0.6250	0.5564	0.1729	0.3083	0.2734	0.2500
Precision	0.4545	0.5203	0.4762	0.4457	0.4780	0.4514
F_1	0.5884	0.5424	0.6192	0.5761	0.5906	0.5420
F_2	0.7576	0.5565	0.7553	0.6987	0.6878	0.6422
$F_{0.5}$	0.5049	0.5289	0.5247	0.4901	0.5174	0.2194
AUC	0.5137	0.5614	0.5289	0.5612	0.4912	0.4513

Conclusion

From the models developed in this project, the recommended model for predicting Bitcoin price movement is the CART decision tree due to its best overall positive and negative identification rates. Although this was the best performing model, it is important to consider the limitations. Most notably, within the scope of this project predicting the stock market and cryptocurrency is almost an impossible task due to the complexity and deeper influences of product development and the economy at large. Politics, public relations, and other non-numeric factors move prices in ways that are more complicated than simply using individual stocks as predictor variables. Thus, more research must be done to find any useful implications to real-world applications. Future developments using current world affairs, politics, and other categorical variables may prove useful for domain expansion across the stock market.

References

Geeks for Geeks. (2024, Sep 19). Geeks for Geeks. CART (Classification And Regression Tree) in Machine Learning. Geeks for Geeks. https://www.geeksforgeeks.org/cart-classification-and-regression-tree-in-machine-learning/

IBM. (n.d.). What is Random Forest? IBM. https://www.ibm.com/topics/random-forest

Larose, C., & Larose, D. (2019). Data science using Python and R. John Wiley & Sons, Inc.

Tan, P. N., Steinbach, M., Karpatne, A., & Kumar, V. (2019). Introduction to data mining (2nd ed.). Pearson.

Appendix

Upload data and check for N/A values

```
df <- read.csv("us_stock_data.csv")
head(df)

##   X       Date Natural_Gas_Price Natural_Gas_Vol. Crude_oil_Price
## 1 0   2/2/2024             2.079               NA           72.28
## 2 1   1/2/2024             2.050           161340           73.82
## 3 2 31-01-2024             2.100           142860           75.85
## 4 3 30-01-2024             2.077           139750           77.82
## 5 4 29-01-2024             2.490             3590           76.78
## 6 5 26-01-2024             2.712            73020           78.01
##   Crude_oil_Vol. Copper_Price Copper_Vol. Bitcoin_Price Bitcoin_Vol.
## 1             NA       3.8215          NA     43,194.70        42650
## 2         577940       3.8535          NA     43,081.40        47690
## 3         344490       3.9060          NA     42,580.50        56480
## 4         347240       3.9110          NA     42,946.20        55130
## 5         331930       3.8790          NA     43,299.80        45230
## 6         365460       3.8520          NA     41,811.30        69470
##   Platinum_Price Platinum_Vol. Ethereum_Price Ethereum_Vol. S.P_500_Price
## 1          901.6            NA       2,309.28        246890      4,958.61
## 2          922.3            NA       2,304.28        323610      4,906.19
## 3          932.6            NA       2,283.14        408790      4,848.87
## 4          931.7            NA       2,343.11        387120      4,924.97
## 5          938.3            NA       2,317.79        318840      4,927.93
## 6          921.3            NA       2,267.55        377790      4,890.97
##   Nasdaq_100_Price Nasdaq_100_Vol. Apple_Price Apple_Vol. Tesla_Price
## 1        17,642.73       315620000      185.85  102550000      187.91
## 2        17,344.71       240640000      186.86   53490000      188.86
## 3        17,137.24       366450000      184.40   54830000      187.29
## 4        17,476.71       236210000      188.04   55270000      191.59
## 5        17,596.27       238750000      191.73   46890000      190.93
## 6        17,421.01       252940000      192.42   44590000      183.25
##   Tesla_Vol. Microsoft_Price Microsoft_Vol. Silver_Price Silver_Vol.
## 1  110610000          411.22       28260000       22.796          NA
## 2   90680000          403.78       29230000       23.236       85160
## 3  102270000          397.58       46780000       23.169       66910
## 4  105540000          408.59       29340000       23.225       53370
## 5  123600000          409.72       23290000       23.134         330
## 6  107340000          403.93       17800000       22.758         330
##   Google_Price Google_Vol. Nvidia_Price Nvidia_Vol. Berkshire_Price
## 1       142.38    62500000       661.60    47660000        5,89,498
## 2       141.16    37120000       630.27    36020000        5,81,600
## 3       140.10    71370000       615.27    45070000        5,78,020
## 4       151.46    33060000       627.74    39600000        5,84,680
## 5       153.51    27590000       624.65    33900000        5,78,800
```

```
## 6       152.18    26120000       610.31    39030000        5,82,300
##   Berkshire_Vol. Netflix_Price Netflix_Vol. Amazon_Price Amazon_Vol. Meta_
Price
## 1          10580        564.64      4030000       171.81   117220000     4
74.99
## 2           9780        567.51      3150000       159.28    66360000     3
94.78
## 3           9720        564.11      4830000       155.20    49690000     3
90.14
## 4           9750        562.85      6120000       159.00    42290000     4
00.06
## 5          13850        575.79      6880000       161.26    42840000     4
01.02
## 6          10040        570.42     12770000       159.12    51050000     3
94.14
##   Meta_Vol. Gold_Price Gold_Vol.
## 1  84710000   2,053.70        NA
## 2  25140000   2,071.10    260920
## 3  20010000   2,067.40    238370
## 4  18610000   2,050.90    214590
## 5  17790000   2,034.90      1780
## 6  13160000   2,026.60       410
```

```
colSums(is.na(df))
```

```
##                X              Date Natural_Gas_Price  Natural_Gas_Vol.
##                0                 0                 0                 4
##   Crude_oil_Price    Crude_oil_Vol.      Copper_Price       Copper_Vol.
##                0                23                 0                37
##     Bitcoin_Price      Bitcoin_Vol.    Platinum_Price     Platinum_Vol.
##                0                 0                 0               377
##    Ethereum_Price     Ethereum_Vol.     S.P_500_Price  Nasdaq_100_Price
##                0                 0                 0                 0
##   Nasdaq_100_Vol.       Apple_Price        Apple_Vol.       Tesla_Price
##                1                 0                 0                 0
##        Tesla_Vol.   Microsoft_Price     Microsoft_Vol.      Silver_Price
##                0                 0                 0                 0
##       Silver_Vol.      Google_Price       Google_Vol.      Nvidia_Price
##               46                 0                 0                 0
##       Nvidia_Vol.   Berkshire_Price    Berkshire_Vol.     Netflix_Price
##                0                 0                 0                 0
##      Netflix_Vol.      Amazon_Price       Amazon_Vol.        Meta_Price
##                0                 0                 0                 0
##         Meta_Vol.        Gold_Price         Gold_Vol.
##                0                 0                 2
```

Data Pre-processing

Reformat date using lubridate package

```
df$standardized_dates <- parse_date_time(df$Date, orders = c("dmy"))
df$standardized_dates <- as.Date(df$standardized_dates)
head(df)
```

```
##   X       Date Natural_Gas_Price Natural_Gas_Vol. Crude_oil_Price
## 1 0   2/2/2024             2.079               NA           72.28
## 2 1   1/2/2024             2.050           161340           73.82
## 3 2 31-01-2024             2.100           142860           75.85
## 4 3 30-01-2024             2.077           139750           77.82
## 5 4 29-01-2024             2.490             3590           76.78
## 6 5 26-01-2024             2.712            73020           78.01
##   Crude_oil_Vol. Copper_Price Copper_Vol. Bitcoin_Price Bitcoin_Vol.
## 1             NA       3.8215          NA     43,194.70        42650
## 2         577940       3.8535          NA     43,081.40        47690
## 3         344490       3.9060          NA     42,580.50        56480
## 4         347240       3.9110          NA     42,946.20        55130
## 5         331930       3.8790          NA     43,299.80        45230
## 6         365460       3.8520          NA     41,811.30        69470
##   Platinum_Price Platinum_Vol. Ethereum_Price Ethereum_Vol. S.P_500_Price
## 1          901.6            NA       2,309.28        246890      4,958.61
## 2          922.3            NA       2,304.28        323610      4,906.19
## 3          932.6            NA       2,283.14        408790      4,848.87
## 4          931.7            NA       2,343.11        387120      4,924.97
## 5          938.3            NA       2,317.79        318840      4,927.93
## 6          921.3            NA       2,267.55        377790      4,890.97
##   Nasdaq_100_Price Nasdaq_100_Vol. Apple_Price Apple_Vol. Tesla_Price
## 1        17,642.73       315620000      185.85  102550000      187.91
## 2        17,344.71       240640000      186.86   53490000      188.86
## 3        17,137.24       366450000      184.40   54830000      187.29
## 4        17,476.71       236210000      188.04   55270000      191.59
## 5        17,596.27       238750000      191.73   46890000      190.93
## 6        17,421.01       252940000      192.42   44590000      183.25
##   Tesla_Vol. Microsoft_Price Microsoft_Vol. Silver_Price Silver_Vol.
## 1  110610000          411.22       28260000       22.796          NA
## 2   90680000          403.78       29230000       23.236       85160
## 3  102270000          397.58       46780000       23.169       66910
## 4  105540000          408.59       29340000       23.225       53370
## 5  123600000          409.72       23290000       23.134         330
## 6  107340000          403.93       17800000       22.758         330
##   Google_Price Google_Vol. Nvidia_Price Nvidia_Vol. Berkshire_Price
## 1       142.38    62500000       661.60    47660000        5,89,498
## 2       141.16    37120000       630.27    36020000        5,81,600
## 3       140.10    71370000       615.27    45070000        5,78,020
## 4       151.46    33060000       627.74    39600000        5,84,680
## 5       153.51    27590000       624.65    33900000        5,78,800
## 6       152.18    26120000       610.31    39030000        5,82,300
```

```
##   Berkshire_Vol. Netflix_Price Netflix_Vol. Amazon_Price Amazon_Vol. Meta_
Price
## 1          10580        564.64      4030000       171.81   117220000     4
74.99
## 2           9780        567.51      3150000       159.28    66360000     3
94.78
## 3           9720        564.11      4830000       155.20    49690000     3
90.14
## 4           9750        562.85      6120000       159.00    42290000     4
00.06
## 5          13850        575.79      6880000       161.26    42840000     4
01.02
## 6          10040        570.42     12770000       159.12    51050000     3
94.14
##   Meta_Vol. Gold_Price Gold_Vol. standardized_dates
## 1  84710000   2,053.70        NA         2024-02-02
## 2  25140000   2,071.10    260920         2024-02-01
## 3  20010000   2,067.40    238370         2024-01-31
## 4  18610000   2,050.90    214590         2024-01-30
## 5  17790000   2,034.90      1780         2024-01-29
## 6  13160000   2,026.60       410         2024-01-26
```

We will be focusing on predicting Bitcoin Price Movement using individual stocks

Individual stocks: Apple, Tesla, Microsoft, Google, Nvidia, Berkshire Hathaway, Netflix, Amazon, and Meta

```
# Create filtered data frame with individual stocks and dates

filtered_df <- df[, c("standardized_dates", "Bitcoin_Price", "Apple_Price", "
Tesla_Price", "Microsoft_Price", "Google_Price", "Nvidia_Price", "Berkshire_P
rice", "Netflix_Price", "Amazon_Price", "Meta_Price")]

filtered_df$Bitcoin_Price <- gsub("[^0-9.-]", "", filtered_df$Bitcoin_Price)
filtered_df$Bitcoin_Price <- as.numeric(filtered_df$Bitcoin_Price)
filtered_df$Berkshire_Price <- gsub("[^0-9.-]", "", filtered_df$Berkshire_Pri
ce)
filtered_df$Berkshire_Price <- as.numeric(filtered_df$Berkshire_Price)
head(filtered_df)

##   standardized_dates Bitcoin_Price Apple_Price Tesla_Price Microsoft_Price
## 1         2024-02-02       43194.7      185.85      187.91          411.22
## 2         2024-02-01       43081.4      186.86      188.86          403.78
## 3         2024-01-31       42580.5      184.40      187.29          397.58
## 4         2024-01-30       42946.2      188.04      191.59          408.59
## 5         2024-01-29       43299.8      191.73      190.93          409.72
## 6         2024-01-26       41811.3      192.42      183.25          403.93
##   Google_Price Nvidia_Price Berkshire_Price Netflix_Price Amazon_Price
## 1       142.38       661.60          589498        564.64       171.81
## 2       141.16       630.27          581600        567.51       159.28
```

```
## 3       140.10       615.27          578020         564.11       155.20
## 4       151.46       627.74          584680         562.85       159.00
## 5       153.51       624.65          578800         575.79       161.26
## 6       152.18       610.31          582300         570.42       159.12
##   Meta_Price
## 1     474.99
## 2     394.78
## 3     390.14
## 4     400.06
## 5     401.02
## 6     394.14
```

```
# Sort the filtered data frame by date

sorted_filtered_df <- filtered_df[order(filtered_df$standardized_dates, decre
asing = TRUE),]
head(sorted_filtered_df)
```

```
##   standardized_dates Bitcoin_Price Apple_Price Tesla_Price Microsoft_Price
## 1         2024-02-02       43194.7      185.85      187.91          411.22
## 2         2024-02-01       43081.4      186.86      188.86          403.78
## 3         2024-01-31       42580.5      184.40      187.29          397.58
## 4         2024-01-30       42946.2      188.04      191.59          408.59
## 5         2024-01-29       43299.8      191.73      190.93          409.72
## 6         2024-01-26       41811.3      192.42      183.25          403.93
##   Google_Price Nvidia_Price Berkshire_Price Netflix_Price Amazon_Price
## 1       142.38       661.60          589498         564.64       171.81
## 2       141.16       630.27          581600         567.51       159.28
## 3       140.10       615.27          578020         564.11       155.20
## 4       151.46       627.74          584680         562.85       159.00
## 5       153.51       624.65          578800         575.79       161.26
## 6       152.18       610.31          582300         570.42       159.12
##   Meta_Price
## 1     474.99
## 2     394.78
## 3     390.14
## 4     400.06
## 5     401.02
## 6     394.14
```

Partition the Data into Test (25%) and Train (75%)

```
split_point <- floor(0.25 * nrow(sorted_filtered_df))

test_data <- sorted_filtered_df[1:split_point,]
train_data <- sorted_filtered_df[(split_point + 1):nrow(sorted_filtered_df),
]

n_test <- dim(test_data)[1]
test_data$Index <- c(1:n_test)
```

```
n_train <- dim(train_data)[1]
train_data$Index <- c(1:n_train)

# Uncomment to check the test and train data

## Train data information:
## 760 Data points from 2020-01-02 to 2023-01-31

## Test data information:
## 253 Data points from 2023-02-01 to 2024-02-02

head(train_data)

##     standardized_dates Bitcoin_Price Apple_Price Tesla_Price Microsoft_Pri
ce
## 254         2023-01-31       23125.1      144.29      173.22          247.
81
## 255         2023-01-30       22832.2      143.00      166.66          242.
71
## 256         2023-01-27       23074.6      145.93      177.90          248.
16
## 257         2023-01-26       23016.0      143.96      160.27          248.
00
## 258         2023-01-25       23055.1      141.86      144.43          240.
61
## 259         2023-01-24       22632.5      142.53      143.89          242.
04
##     Google_Price Nvidia_Price Berkshire_Price Netflix_Price Amazon_Price
## 254        98.84       195.37          473000        353.86       103.13
## 255        96.94       191.62          465040        353.11       100.55
## 256        99.37       203.65          470000        360.77       102.24
## 257        97.52       198.02          469960        364.87        99.22
## 258        95.22       193.23          471658        367.96        97.18
## 259        97.70       192.65          471000        363.83        96.32
##     Meta_Price Index
## 254     148.97     1
## 255     147.06     2
## 256     151.74     3
## 257     147.30     4
## 258     141.50     5
## 259     143.14     6

head(test_data)

##   standardized_dates Bitcoin_Price Apple_Price Tesla_Price Microsoft_Price
## 1         2024-02-02       43194.7      185.85      187.91          411.22
## 2         2024-02-01       43081.4      186.86      188.86          403.78
## 3         2024-01-31       42580.5      184.40      187.29          397.58
## 4         2024-01-30       42946.2      188.04      191.59          408.59
## 5         2024-01-29       43299.8      191.73      190.93          409.72
```

```
## 6         2024-01-26       41811.3      192.42      183.25          403.93
##   Google_Price Nvidia_Price Berkshire_Price Netflix_Price Amazon_Price
## 1       142.38       661.60          589498        564.64       171.81
## 2       141.16       630.27          581600        567.51       159.28
## 3       140.10       615.27          578020        564.11       155.20
## 4       151.46       627.74          584680        562.85       159.00
## 5       153.51       624.65          578800        575.79       161.26
## 6       152.18       610.31          582300        570.42       159.12
##   Meta_Price Index
## 1     474.99     1
## 2     394.78     2
## 3     390.14     3
## 4     400.06     4
## 5     401.02     5
## 6     394.14     6
```

Create Test & Train sets that have standardized variables

```
# Standardize the variables into a new data frame

std_sorted_filtered_df <- sorted_filtered_df
std_sorted_filtered_df[, 2:11] <- scale(sorted_filtered_df[, 2:11])

# Split the new standardized data frame into the train and test sets

std_split_point <- floor(0.25 * nrow(std_sorted_filtered_df))

std_test <- std_sorted_filtered_df[1:std_split_point,]
std_train <- std_sorted_filtered_df[(split_point + 1):nrow(std_sorted_filtere
d_df),]

# Create Index for the train and test sets

n_test <- dim(std_test)[1]
std_test$Index <- c(1:n_test)
n_train <- dim(std_train)[1]
std_train$Index <- c(1:n_train)

# For time series analysis lag the target variable by one day to use previous
 day predictor variables to predict the movement of next day target variable

std_train$Bitcoin_Lag <- lag(std_train$Bitcoin_Price, 1)
std_test$Bitcoin_Lag <- lag(std_test$Bitcoin_Price, 1)

## Create a seven day rolling mean for the target variable

std_train <- std_train %>%
  mutate(var1_7day_avg = rollmean(Bitcoin_Price, k=7, fill = NA, align = "rig
ht"))
```

```
std_test <- std_test %>%
  mutate(var2_7day_avg = rollmean(Bitcoin_Price, k=7, fill = NA, align = "rig
ht"))

## Use the seven day rolling mean to see if the predictor variable increases
or decreases

std_train <- std_train %>%
  mutate(Bitcoin_Price_Change = ifelse(var1_7day_avg > lag(var1_7day_avg), "I
ncreased",
                                       ifelse(var1_7day_avg < lag(var1_7day_a
vg), "Decreased", "No Change")))
std_test <- std_test %>%
  mutate(Bitcoin_Price_Change = ifelse(var2_7day_avg > lag(var2_7day_avg), "I
ncreased",
                                ifelse(var2_7day_avg < lag(var2_7day_avg), "D
ecreased", "No Change")))

std_train$Bitcoin_Price_Change <- factor(std_train$Bitcoin_Price_Change)
std_test$Bitcoin_Price_Change <- factor(std_test$Bitcoin_Price_Change)

std_train <- na.omit(std_train)
std_test <- na.omit(std_test)

head(std_train)

##     standardized_dates Bitcoin_Price Apple_Price Tesla_Price Microsoft_Pri
ce
## 261         2023-01-20    -0.4325585  -0.1212140  -0.8891951      -0.44559
46
## 262         2023-01-19    -0.5381446  -0.1981854  -0.9627959      -0.58978
44
## 263         2023-01-18    -0.5653086  -0.1999617  -0.9438364      -0.52229
87
## 264         2023-01-17    -0.5344465  -0.1783505  -0.9119230      -0.44333
35
## 265         2023-01-13    -0.6145027  -0.2132836  -1.0189681      -0.46281
39
## 266         2023-01-12    -0.6856674  -0.2532495  -1.0053078      -0.47533
70
##     Google_Price Nvidia_Price Berkshire_Price Netflix_Price Amazon_Price
## 261   -0.4182623   -0.3213529       0.5416961    -0.6640914    -1.470477
## 262   -0.6136369   -0.4048231       0.4485979    -0.8871933    -1.601141
## 263   -0.6895068   -0.3572591       0.5007425    -0.7991048    -1.535992
## 264   -0.6828239   -0.3320004       0.6185823    -0.8000232    -1.514397
## 265   -0.6501960   -0.3944088       0.6793290    -0.7449157    -1.438634
## 266   -0.6891137   -0.4245637       0.7015782    -0.7673762    -1.542946
##     Meta_Price Index Bitcoin_Lag var1_7day_avg Bitcoin_Price_Change
## 261  -1.546021     8   -0.4167934    -0.4187346            Decreased
```

```
## 262  -1.590161     9  -0.4325585    -0.4352833           Decreased
## 263  -1.633068    10  -0.5381446    -0.4580034           Decreased
## 264  -1.600990    11  -0.5653086    -0.4757607           Decreased
## 265  -1.578783    12  -0.5344465    -0.5053243           Decreased
## 266  -1.582484    13  -0.6145027    -0.5410602           Decreased
```

```
head(std_test)
```

```
##    standardized_dates Bitcoin_Price Apple_Price  Tesla_Price Microsoft_Pri
ce
## 8          2024-01-24     0.7191500    1.555282 -0.012933283         2.3780
22
## 9          2024-01-23     0.7061039    1.575413  0.002493447         2.3143
63
## 10         2024-01-22     0.6841134    1.537223 -0.001510437         2.2727
93
## 11         2024-01-19     0.8224867    1.468245  0.038410644         2.3103
62
## 12         2024-01-18     0.7989812    1.381504  0.034760044         2.2268
75
## 13         2024-01-17     0.8966285    1.205358  0.077978441         2.1503
45
##    Google_Price Nvidia_Price Berkshire_Price Netflix_Price Amazon_Price
## 8      1.574009     3.061210        1.767665     1.0256221    0.7116560
## 9      1.508753     2.945487        1.684862     0.5857639    0.6805454
## 10     1.467477     2.928466        1.614485     0.5316583    0.6351605
## 11     1.482808     2.915798        1.595055     0.5086134    0.6556569
## 12     1.368807     2.730516        1.507546     0.5283185    0.5883117
## 13     1.289792     2.648600        1.456406     0.4867373    0.5227965
##    Meta_Price Index Bitcoin_Lag var2_7day_avg Bitcoin_Price_Change
## 8    1.899251     8   0.7092066     0.8433251            Decreased
## 9    1.823856     9   0.7191500     0.8131520            Decreased
## 10   1.776975    10   0.7061039     0.7845714            Decreased
## 11   1.799867    11   0.6841134     0.7723021            Decreased
## 12   1.699523    12   0.8224867     0.7533331            Decreased
## 13   1.593148    13   0.7989812     0.7623815            Increased
```

```
table(std_train$Bitcoin_Price_Change)
```

```
## 
## Decreased Increased 
##       412       341
```

```
table(std_test$Bitcoin_Price_Change)
```

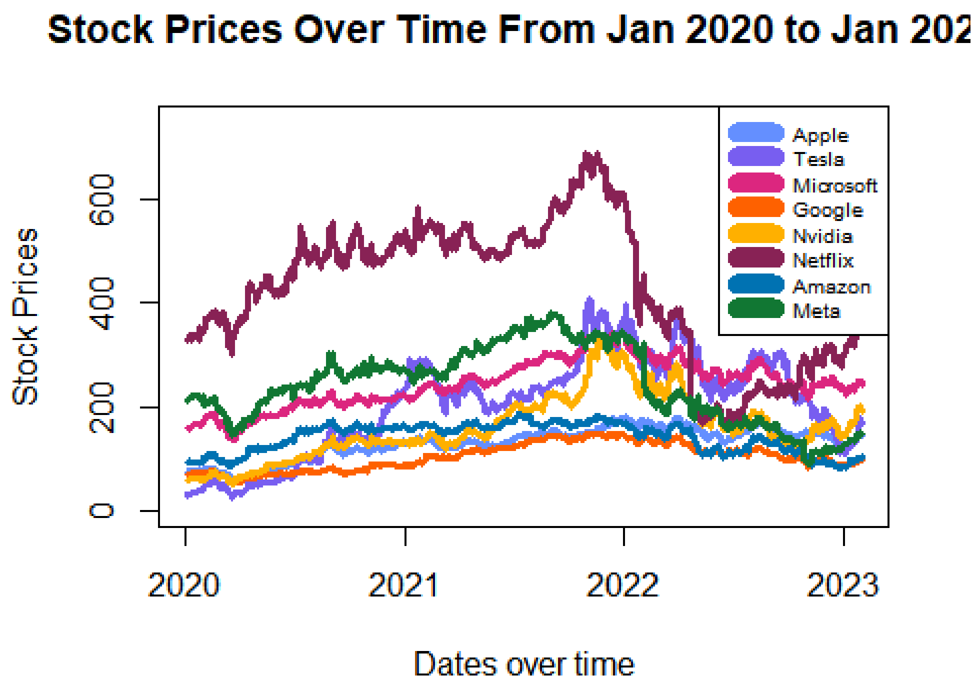
```
## 
## Decreased Increased 
##       133       113
```

Exploratory Data Analysis

Overlapping Line Graph with Stock Prices

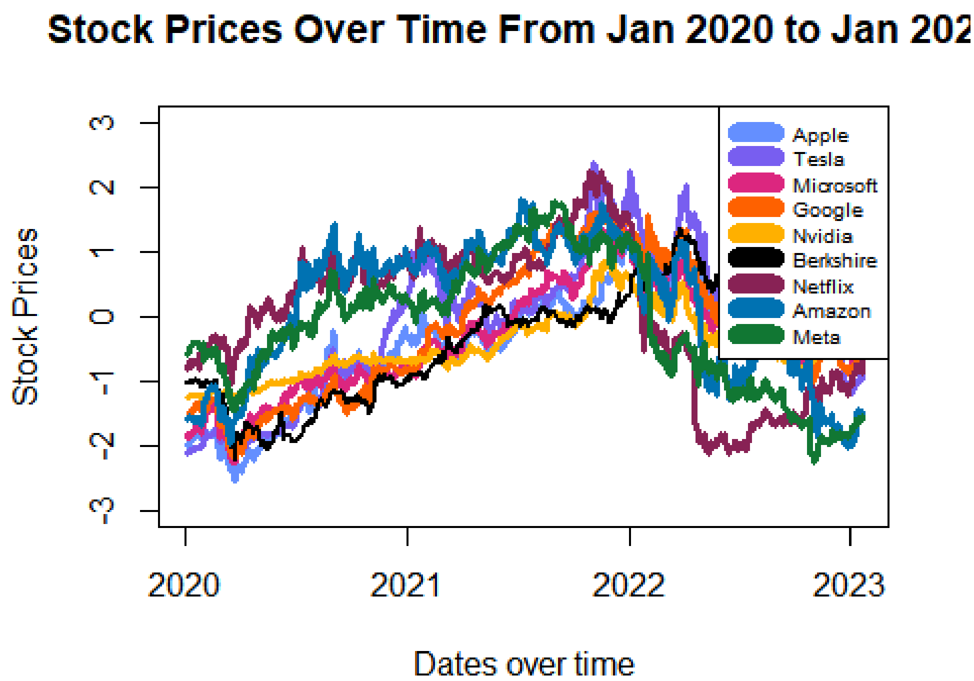
Note: Berkshire_Price was not graphed because it has extreme values

```
plot(train_data$standardized_dates, train_data$Apple_Price, type = "l", col =
 "#648fff", lwd = 3, xlab = "Dates over time", ylab = "Stock Prices", main =
"Stock Prices Over Time From Jan 2020 to Jan 2023", ylim = c(0, 750))
lines(train_data$standardized_dates, train_data$Tesla_Price, col = "#785ef0",
 lwd = 3)
lines(train_data$standardized_dates, train_data$Microsoft_Price, col = "#dc26
7f", lwd = 3)
lines(train_data$standardized_dates, train_data$Google_Price, col = "#fe6100"
, lwd = 3)
lines(train_data$standardized_dates, train_data$Nvidia_Price, col = "#ffb000"
, lwd = 3)
# lines(train_data$standardized_dates, train_data$Berkshire_Price, col = "#78
5ef0", lwd = 2)
lines(train_data$standardized_dates, train_data$Netflix_Price, col = "#882255
", lwd = 3)
lines(train_data$standardized_dates, train_data$Amazon_Price, col = "#0072b2"
, lwd = 3)
lines(train_data$standardized_dates, train_data$Meta_Price, col = "#117733",
lwd = 3)
legend("topright", legend = c("Apple", "Tesla", "Microsoft", "Google", "Nvidi
a", "Netflix", "Amazon", "Meta"),col = c("#648fff", "#785ef0", "#dc267f", "#f
e6100", "#ffb000", "#882255", "#0072b2", "#117733"), lwd = 10, cex = 0.65)
```



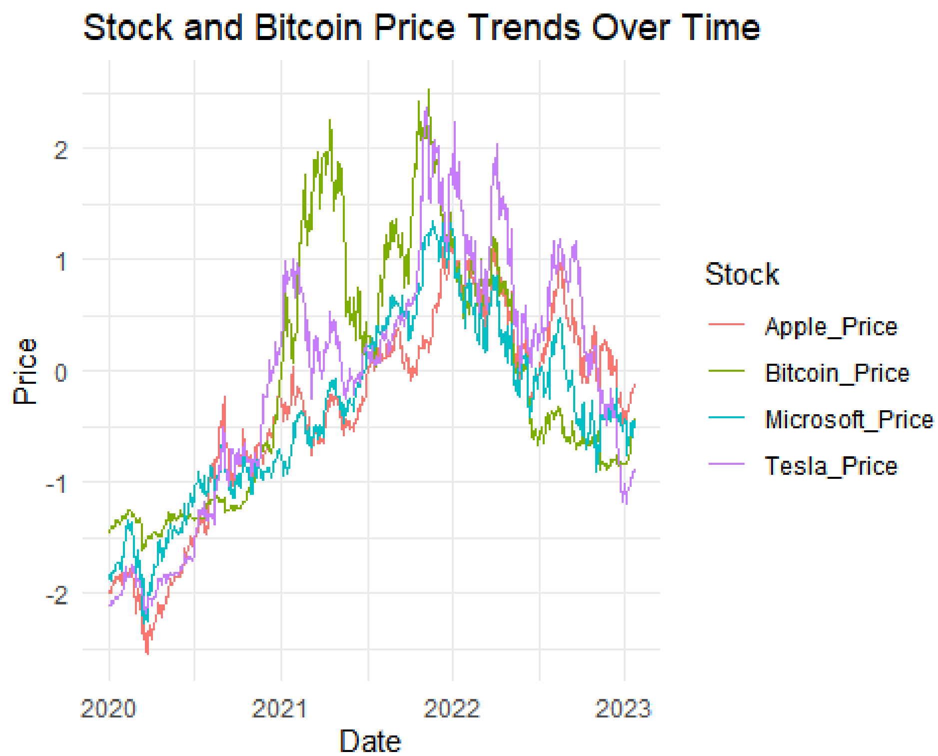
Overlapping Line Graph with Standardized Stock Prices (Berkshire Included)

```
plot(std_train$standardized_dates, std_train$Apple_Price, type = "l", col = "
#648fff", lwd = 3, xlab = "Dates over time", ylab = "Stock Prices", main = "S
tock Prices Over Time From Jan 2020 to Jan 2023", ylim = c(-3,3))
lines(std_train$standardized_dates, std_train$Tesla_Price, col = "#785ef0", l
wd = 3)
lines(std_train$standardized_dates, std_train$Microsoft_Price, col = "#dc267f
", lwd = 3)
lines(std_train$standardized_dates, std_train$Google_Price, col = "#fe6100",
lwd = 3)
lines(std_train$standardized_dates, std_train$Nvidia_Price, col = "#ffb000",
lwd = 3)
lines(std_train$standardized_dates, std_train$Berkshire_Price, col = "#000000
", lwd = 2)
lines(std_train$standardized_dates, std_train$Netflix_Price, col = "#882255",
 lwd = 3)
lines(std_train$standardized_dates, std_train$Amazon_Price, col = "#0072b2",
lwd = 3)
lines(std_train$standardized_dates, std_train$Meta_Price, col = "#117733", lw
d = 3)
legend("topright", legend = c("Apple", "Tesla", "Microsoft", "Google", "Nvidi
a", "Berkshire", "Netflix", "Amazon", "Meta"),col = c("#648fff", "#785ef0", "
#dc267f", "#fe6100", "#ffb000", "#000000", "#882255", "#0072b2", "#117733"),
lwd = 10, cex = 0.65)
```



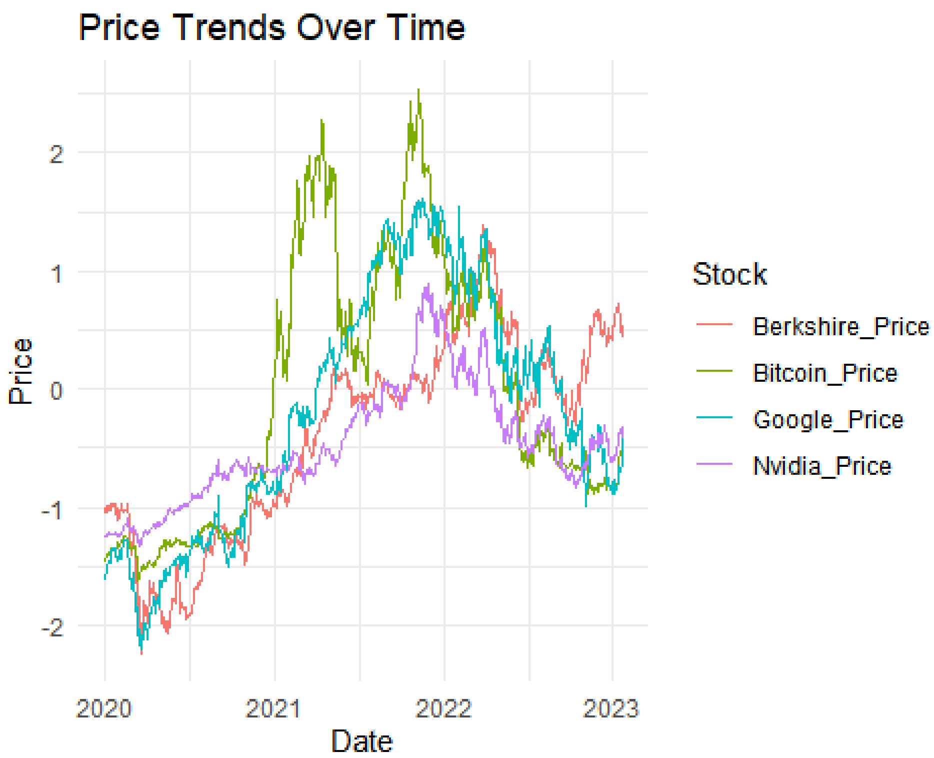
Overlapping Line Graph with Only Apple, Bitcoin, Microsoft, and Tesla

```
# Convert wide data to long format for visualization
std_train_long <- std_train %>%
  pivot_longer(
    cols = c(Bitcoin_Price, Apple_Price, Tesla_Price, Microsoft_Price,
    ),
    names_to = "Stock",
    values_to = "Price"
  )

# Create line plots for all stocks and Bitcoin price over time
ggplot(std_train_long, aes(x = standardized_dates, y = Price, color = Stock))
 +
  geom_line() +
  labs(title = "Stock and Bitcoin Price Trends Over Time",
       x = "Date",
       y = "Price",
       color = "Stock") +
  theme_minimal()
```



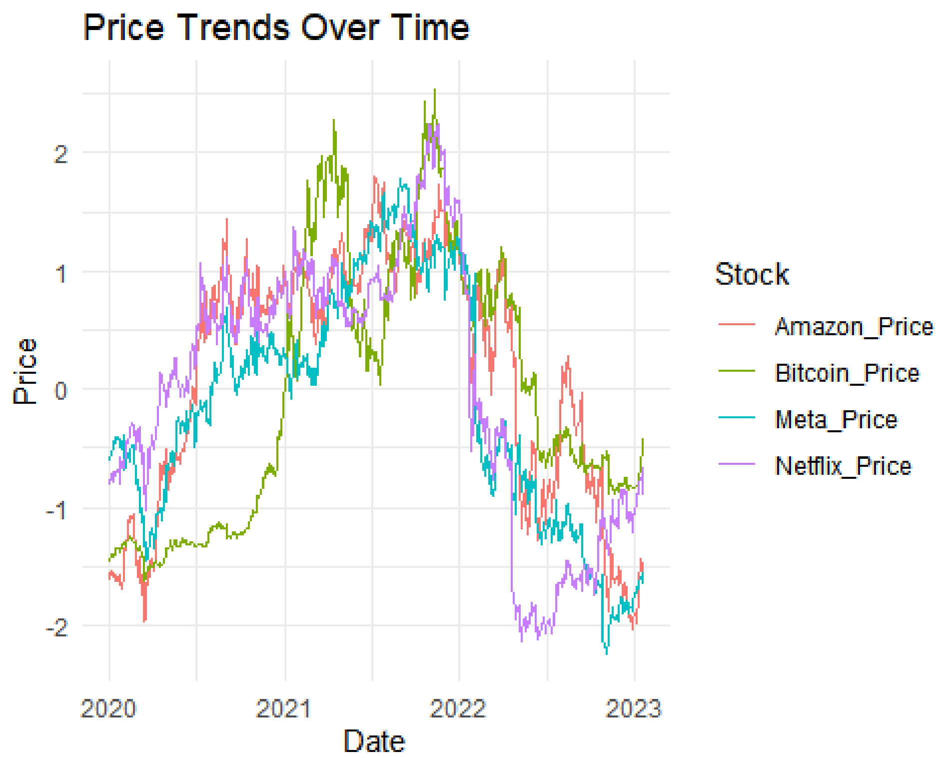
Overlapping Line Graph with Only Berkshire, Bitcoin, Google, and Nvidia

```
long_data <- std_train %>%
  pivot_longer(cols = c(Bitcoin_Price, Google_Price, Nvidia_Price, Berkshire_
Price),
               names_to = "Stock", values_to = "Price")

ggplot(long_data, aes(x = standardized_dates, y = Price, color = Stock)) +
  geom_line() +
  labs(title = "Price Trends Over Time", x = "Date", y = "Price") +
  theme_minimal()
```



Overlapping Line Graph with Only Amazon, Bitcoin, Meta, and Netflix

```
long_data <- std_train %>%
  pivot_longer(cols = c(Bitcoin_Price, Netflix_Price,
                        Amazon_Price, Meta_Price),
               names_to = "Stock", values_to = "Price")

ggplot(long_data, aes(x = standardized_dates, y = Price, color = Stock)) +
  geom_line() +
  labs(title = "Price Trends Over Time", x = "Date", y = "Price") +
  theme_minimal()
```



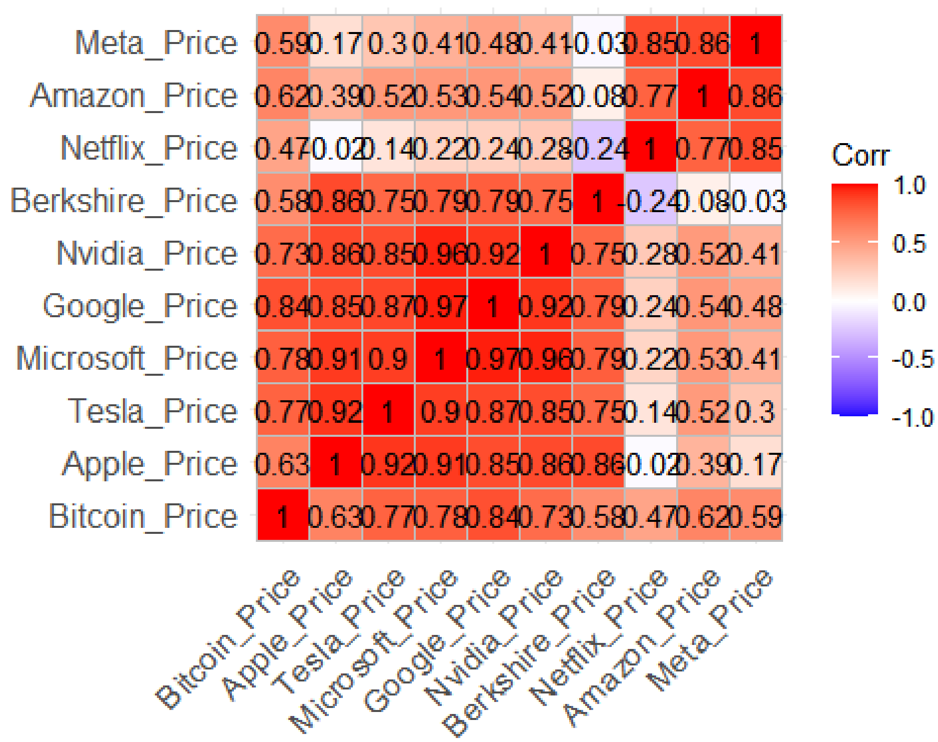
Correlation Heat Map of all Variables

```
#Correlation Analysis
#Explore the correlation between Bitcoin's price and the prices of individual
 stocks

correlation_matrix <- cor(train_data[, 2:11], use = "complete.obs")
print(correlation_matrix)
```

```
##                 Bitcoin_Price Apple_Price Tesla_Price Microsoft_Price
## Bitcoin_Price       1.0000000  0.62562027   0.7705045       0.7776901
## Apple_Price         0.6256203  1.00000000   0.9188914       0.9101306
## Tesla_Price         0.7705045  0.91889138   1.0000000       0.9016140
## Microsoft_Price     0.7776901  0.91013058   0.9016140       1.0000000
## Google_Price        0.8365944  0.85461788   0.8741953       0.9681007
## Nvidia_Price        0.7300287  0.85800207   0.8527103       0.9589585
## Berkshire_Price     0.5825099  0.86000432   0.7465021       0.7914660
## Netflix_Price       0.4747068 -0.02108043   0.1363454       0.2226306
## Amazon_Price        0.6216082  0.39447118   0.5185501       0.5322344
## Meta_Price          0.5926030  0.16594103   0.3012957       0.4122428
##                 Google_Price Nvidia_Price Berkshire_Price Netflix_Price
## Bitcoin_Price      0.8365944    0.7300287      0.58250986    0.47470678
## Apple_Price        0.8546179    0.8580021      0.86000432   -0.02108043
## Tesla_Price        0.8741953    0.8527103      0.74650206    0.13634543
## Microsoft_Price    0.9681007    0.9589585      0.79146605    0.22263056
## Google_Price       1.0000000    0.9188245      0.79049291    0.24303381
## Nvidia_Price       0.9188245    1.0000000      0.75083653    0.28381916
```

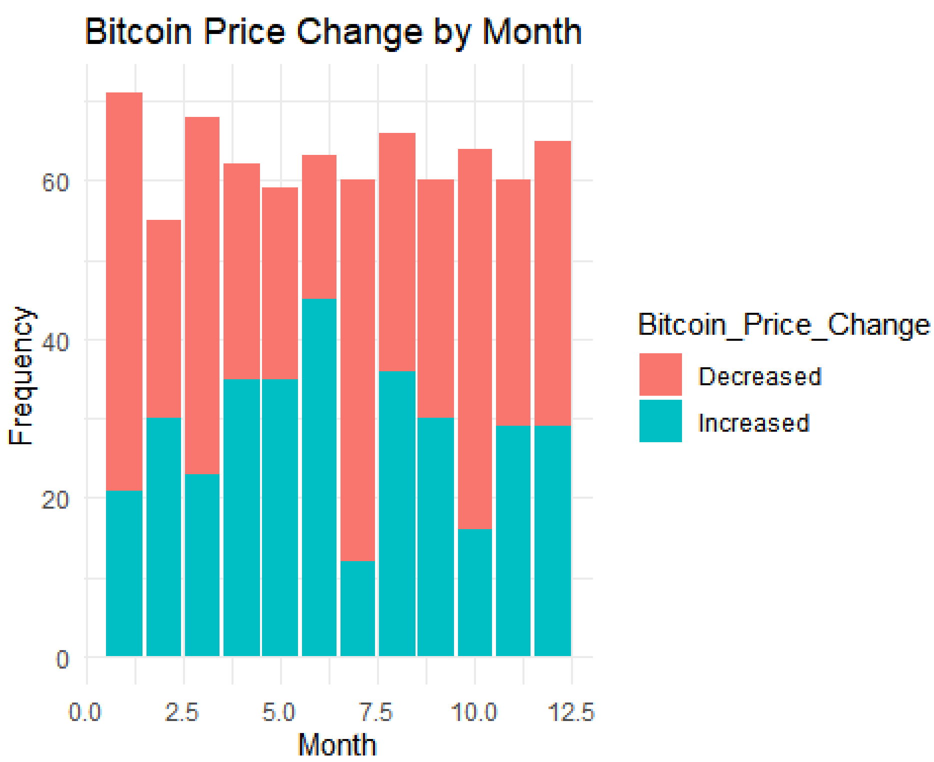
```
## Berkshire_Price    0.7904929    0.7508365     1.00000000    -0.23557635
## Netflix_Price      0.2430338    0.2838192    -0.23557635     1.00000000
## Amazon_Price       0.5436828    0.5222250     0.08161363     0.76643792
## Meta_Price         0.4832585    0.4146457    -0.02753685     0.84721147
##                 Amazon_Price  Meta_Price
## Bitcoin_Price     0.62160815  0.59260295
## Apple_Price       0.39447118  0.16594103
## Tesla_Price       0.51855006  0.30129573
## Microsoft_Price   0.53223444  0.41224280
## Google_Price      0.54368278  0.48325853
## Nvidia_Price      0.52222499  0.41464569
## Berkshire_Price   0.08161363 -0.02753685
## Netflix_Price     0.76643792  0.84721147
## Amazon_Price      1.00000000  0.86083900
## Meta_Price        0.86083900  1.00000000
```

```
# Heatmap of correlations
ggcorrplot(correlation_matrix, lab = TRUE)
```




```
#Seasonality or Temporal Trends
#Check if there are any temporal patterns in Bitcoin price changes

ggplot(std_train, aes(x = month(standardized_dates), fill = Bitcoin_Price_Cha
nge)) +
  geom_bar() +
  labs(title = "Bitcoin Price Change by Month", x = "Month", y = "Frequency")
```

```
+
  theme_minimal()
```



Model building

Logistic Regression Model Based on Stock Prices

Accuracy: (39 + 81)/246 = 0.4878 Error Rate: 1 - 0.4878 = 0.5122 Sensitivity: 81/113 = 0.7168 Specificity: 39/133 = 0.3451 Precision: 81/175 = 0.4629 F1: (2)(0.4629)(0.7168)/(0.4629 + 0.7168) = 0.5625 F2: (5)(0.4629)(0.7168)/((4)(0.4629) + 0.7168) = 0.6459 F0.5: (1.25)(0.4629)(0.7168)/((0.25)(0.4629) + 0.7168) = 0.4982

```
lrs01_model <- glm(formula = Bitcoin_Price_Change ~ Apple_Price + Tesla_Price
 + Microsoft_Price + Google_Price + Nvidia_Price + Berkshire_Price + Netflix_
Price + Amazon_Price + Meta_Price, data = std_train, family = binomial)
summary(lrs01_model)

##
## Call:
## glm(formula = Bitcoin_Price_Change ~ Apple_Price + Tesla_Price +
##     Microsoft_Price + Google_Price + Nvidia_Price + Berkshire_Price +
##     Netflix_Price + Amazon_Price + Meta_Price, family = binomial,
##     data = std_train)
##
## Coefficients:
```

```
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)         0.3216     0.2169   1.483 0.138133
## Apple_Price        -1.8567     0.4207  -4.413 1.02e-05 ***
## Tesla_Price         0.8394     0.2234   3.758 0.000172 ***
## Microsoft_Price     1.6823     0.7121   2.362 0.018156 *
## Google_Price       -1.4646     0.4653  -3.148 0.001646 **
## Nvidia_Price        0.8578     0.6194   1.385 0.166051
## Berkshire_Price     0.8860     0.2878   3.079 0.002080 **
## Netflix_Price      -0.6075     0.1752  -3.468 0.000524 ***
## Amazon_Price        0.2561     0.2197   1.166 0.243669
## Meta_Price          0.4536     0.2542   1.785 0.074317 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1037.18  on 752  degrees of freedom
## Residual deviance:  972.61  on 743  degrees of freedom
## AIC: 992.61
##
## Number of Fisher Scoring iterations: 4
```

```r
print("Predictor Variable Multicollinearity:")
```

```
## [1] "Predictor Variable Multicollinearity:"
```

```r
ypred <- predict(object = lrs01_model, newdata = std_test, type = "response")
predicted_classes <- ifelse(ypred > 0.7, 1, 0)

# Contingency Table

t1 <- table(std_test$Bitcoin_Price_Change, predicted_classes)
row.names(t1) <- c("Actual: Decrease", "Actual: Increase")
colnames(t1) <- c("Predicted: Decrease", "Predicted: Increase")
t1 <- addmargins(A = t1, FUN = list(Total = sum), quiet = TRUE)
t1
```

```
##                   predicted_classes
##                    Predicted: Decrease Predicted: Increase Total
##   Actual: Decrease                  39                  94   133
##   Actual: Increase                  32                  81   113
##   Total                             71                 175   246
```

```r
# Logistic Regression AUC score

lrs01_actual <- (std_test$Bitcoin_Price_Change == "Increased")
lrs01_roc <- roc(lrs01_actual, predicted_classes)
```

```
## Setting levels: control = FALSE, case = TRUE
```

```
## Setting direction: controls < cases
```

```
lrs01_auc <- auc(lrs01_roc)
lrs01_auc

## Area under the curve: 0.505
```

Logistic Regression Model with Removed Predictor Variables

Accuracy: (23 + 100)/246 = 0.5000 Error Rate: 1 - 0.5000 = 0.5000 Sensitivity: 100/113 = 0.8850 Specificity: 23/133 = 0.1729 Precision: 100/210 = 0.4762 F1: (2)(0.4762)(0.8850)/(0.4762 + 0.8850) = 0.6192 F2: (5)(0.4762)(0.8850)/((4)(0.4762) + 0.8850) = 0.7553 F0.5: (1.25)(0.4762)(0.8850)/((0.25)(0.4762) + 0.8850) = 0.5247

```
lrs02_model <- glm(formula = Bitcoin_Price_Change ~ Apple_Price + Tesla_Price
 + Microsoft_Price + Google_Price + Berkshire_Price, data = std_train, family
 = binomial)
summary(lrs02_model)

##
## Call:
## glm(formula = Bitcoin_Price_Change ~ Apple_Price + Tesla_Price +
##     Microsoft_Price + Google_Price + Berkshire_Price, family = binomial,
##     data = std_train)
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -0.0001739  0.1292146  -0.001 0.998926
## Apple_Price     -1.4123601  0.3714315  -3.802 0.000143 ***
## Tesla_Price      0.7366876  0.2107599   3.495 0.000473 ***
## Microsoft_Price  1.4869419  0.5271918   2.820 0.004795 **
## Google_Price    -0.8823492  0.3499998  -2.521 0.011702 *
## Berkshire_Price  0.7415763  0.2038872   3.637 0.000276 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1037.2  on 752  degrees of freedom
## Residual deviance:  985.9  on 747  degrees of freedom
## AIC: 997.9
##
## Number of Fisher Scoring iterations: 4

print("Predictor Variable Multicollinearity:")

## [1] "Predictor Variable Multicollinearity:"

vif(lrs02_model)

##     Apple_Price     Tesla_Price Microsoft_Price    Google_Price Berkshire_
Price
```

```
##       18.825248        9.230368       30.850198       21.932795        5.0
49894
```

```
ypred <- predict(object = lrs02_model, newdata = std_test, type = "response")
predicted_classes <- ifelse(ypred > 0.5, 1, 0)

# Contingency Table

t2 <- table(std_test$Bitcoin_Price_Change, predicted_classes)
row.names(t2) <- c("Actual: Decrease", "Actual: Increase")
colnames(t2) <- c("Predicted: Decrease", "Predicted: Increase")
t2 <- addmargins(A = t2, FUN = list(Total = sum), quiet = TRUE)
t2
```

```
##                   predicted_classes
##                    Predicted: Decrease Predicted: Increase Total
##   Actual: Decrease                  23                 110   133
##   Actual: Increase                  13                 100   113
##   Total                             36                 210   246
```

```
# Logistic Regression AUC score

lrs02_actual <- (std_test$Bitcoin_Price_Change == "Increased")
lrs02_roc <- roc(lrs02_actual, predicted_classes)
```

```
## Setting levels: control = FALSE, case = TRUE

## Setting direction: controls < cases
```

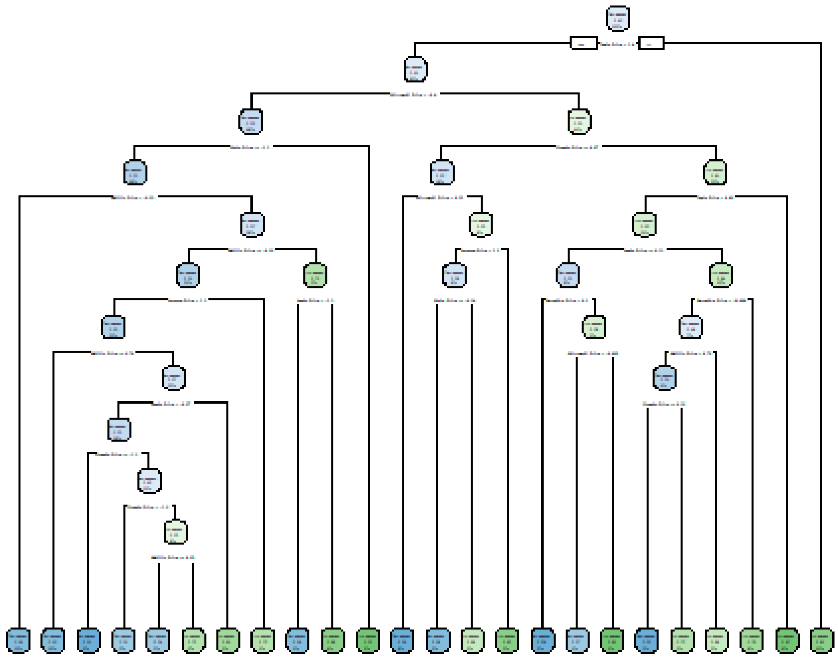
```
lrs02_auc <- auc(lrs02_roc)
lrs02_auc
```

```
## Area under the curve: 0.5289
```

CART

Accuracy: (83 + 55)/246 = 0.5610 Error Rate: 1 - 0.5610 = 0.4390 Sensitivity: 55/113 = 0.4867 Specificity: 83/133 = 0.6241 Precision: 55/105 = 0.5238 F1: (2)(0.5238)(0.4867)/(0.5238 + 0.4867) = 0.5046 F2: (5)(0.5238)(0.4867)/((4)(0.5238) + 0.4867) = 0.4937 F0.5: (1.25)(0.5238)(0.4867)/((0.25)(0.5238) + 0.4867) = 0.5159

```
cart01_model <- rpart(formula = Bitcoin_Price_Change ~ Apple_Price + Tesla_Pr
ice + Microsoft_Price + Google_Price + Nvidia_Price + Berkshire_Price + Netfl
ix_Price + Amazon_Price + Meta_Price, data = std_train, method = "class")

rpart.plot(cart01_model)
```



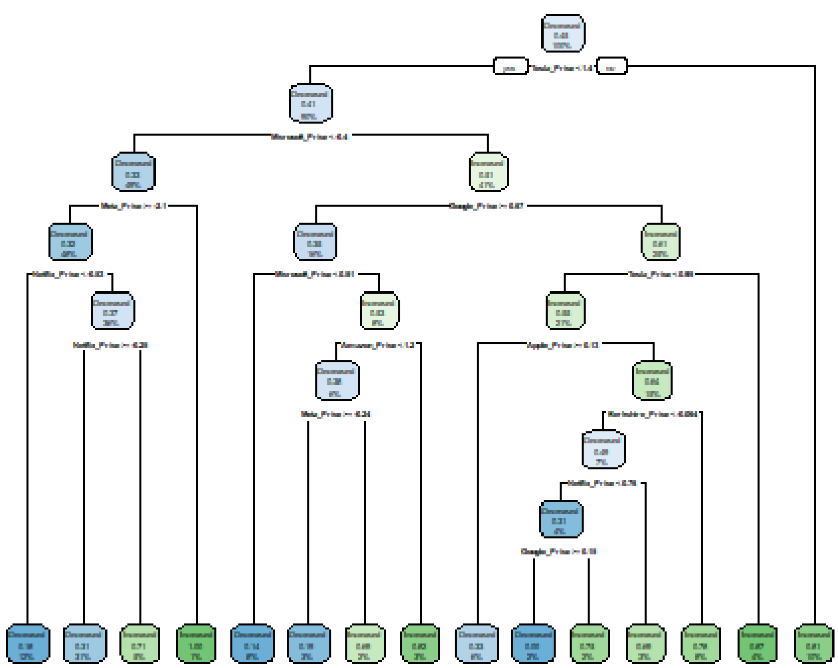
```
ypred <- predict(object = cart01_model, newdata = std_test, type = "class")
# predicted_class <- ifelse(ypred[, 2] > 0.5, 1, 0)

# Contingency Table

t1 <- table(std_test$Bitcoin_Price_Change, ypred)
row.names(t1) <- c("Actual: Decrease", "Actual: Increase")
colnames(t1) <- c("Predicted: Decrease", "Predicted: Increase")
t1 <- addmargins(A = t1, FUN = list(Total = sum), quiet = TRUE)
t1

##                    ypred
##                     Predicted: Decrease Predicted: Increase Total
##   Actual: Decrease                   83                  50   133
##   Actual: Increase                   58                  55   113
##   Total                             141                 105   246

# Node counts

total_nodes <- length(cart01_model$frame$var)
total_nodes

## [1] 47

leaf_nodes <- sum(cart01_model$frame$var == "<leaf>")
leaf_nodes
```

```
## [1] 24

decision_nodes <- total_nodes - leaf_nodes
decision_nodes

## [1] 23

# Information regarding the CART model CP score

# summary(cart01_model)$splits
```

Pruned CART Model

Accuracy: (74 + 64)/246 = 0.5610 Error Rate: 1 - 0.5610 = 0.4390 Sensitivity: 64/113 = 0.5664 Specificity: 74/133 = 0.5564 Precision: 64/123 = 0.5203 F1: (2)(0.5203)(0.5664)/(0.5203 + 0.5664) = 0.5424 F2: (5)(0.5203)(0.5664)/((4)(0.5203) + 0.5664) = 0.5565 F0.5: (1.25)(0.5203)(0.5664)/((0.25)(0.5203) + 0.5664) = 0.5289

```
# cart01_model$cptable
cart02_model <- prune(cart01_model, cp = 0.015)

rpart.plot(cart02_model)
```



```
ypred <- predict(object = cart02_model, newdata = std_test, type = "prob")
predicted_classes <- ifelse(ypred[, 2] > 0.3, 1, 0)

# Contingency Table
```

```
t2 <- table(std_test$Bitcoin_Price_Change, predicted_classes)
row.names(t2) <- c("Actual: Decrease", "Actual: Increase")
colnames(t2) <- c("Predicted: Decrease", "Predicted: Increase")
t2 <- addmargins(A = t2, FUN = list(Total = sum), quiet = TRUE)
t2

##                   predicted_classes
##                    Predicted: Decrease Predicted: Increase Total
##   Actual: Decrease                  74                  59   133
##   Actual: Increase                  49                  64   113
##   Total                            123                 123   246

# Node counts

total_nodes <- length(cart02_model$frame$var)
total_nodes

## [1] 29

leaf_nodes <- sum(cart02_model$frame$var == "<leaf>")
leaf_nodes

## [1] 15

decision_nodes <- total_nodes - leaf_nodes
decision_nodes

## [1] 14

# CART AUC score

cart02_actual <- (std_test$Bitcoin_Price_Change == "Increased")
cart02_roc <- roc(cart02_actual, predicted_classes)

## Setting levels: control = FALSE, case = TRUE

## Setting direction: controls < cases

cart02_auc <- auc(cart02_roc)
cart02_auc

## Area under the curve: 0.5614
```

Random Forests

Accuracy: (41 + 92)/246 = 0.5407 Error Rate: 1 - 0.4919 = 0.4593 Sensitivity: 92/113 = 0.8142 Specificity: 41/133 = 0.3083 Precision: 92/184 = 0.4457 F1: (2)(0.4457)(0.8142)/(0.4457 + 0.8142) = 0.5761 F2: (5)(0.4457)(0.8142)/((4)(0.4457) + 0.8142) = 0.6987 F0.5: (1.25)(0.4457)(0.8142)/((0.25)(0.4457) + 0.8142) = 0.4901

```
set.seed(35)
rf01_model <- randomForest(formula = Bitcoin_Price_Change ~ Apple_Price + Tes
la_Price + Microsoft_Price + Google_Price + Nvidia_Price + Berkshire_Price +
Netflix_Price + Amazon_Price + Meta_Price, data = std_train, method = "class"
)

ypred <- predict(object = rf01_model, newdata = std_test, type = "class")

# Contingency Table

t1 <- table(std_test$Bitcoin_Price_Change, ypred)
row.names(t1) <- c("Actual: Decrease", "Actual: Increase")
colnames(t1) <- c("Predicted: Decrease", "Predicted: Increase")
t1 <- addmargins(A = t1, FUN = list(Total = sum), quiet = TRUE)
t1

##                   ypred
##                    Predicted: Decrease Predicted: Increase Total
##   Actual: Decrease                  41                  92   133
##   Actual: Increase                  21                  92   113
##   Total                             62                 184   246

rf01_model$ntree

## [1] 500

# RF AUC score

rf01_roc <- roc(std_test$Bitcoin_Price_Change, as.numeric(ypred))

## Setting levels: control = Decreased, case = Increased

## Setting direction: controls < cases

rf01_auc <- auc(rf01_roc)
rf01_auc

## Area under the curve: 0.5612
```

Random Forests with 1000 ntree

```
set.seed(35)
rf02_model <- randomForest(formula = Bitcoin_Price_Change ~ Apple_Price + Tes
la_Price + Microsoft_Price + Google_Price + Nvidia_Price + Berkshire_Price +
Netflix_Price + Amazon_Price + Meta_Price, data = std_train, method = "class"
, ntree = 1000)

ypred <- predict(object = rf02_model, newdata = std_test, type = "class")

# Contingency Table
```

```
t2 <- table(std_test$Bitcoin_Price_Change, ypred)
row.names(t2) <- c("Actual: Decrease", "Actual: Increase")
colnames(t2) <- c("Predicted: Decrease", "Predicted: Increase")
t2 <- addmargins(A = t2, FUN = list(Total = sum), quiet = TRUE)
t2

##                    ypred
##                     Predicted: Decrease Predicted: Increase Total
##   Actual: Decrease                   64                  69   133
##   Actual: Increase                   40                  73   113
##   Total                             104                 142   246

rf02_model$ntree

## [1] 1000

# RF AUC score

rf02_roc <- roc(std_test$Bitcoin_Price_Change, as.numeric(ypred))

## Setting levels: control = Decreased, case = Increased

## Setting direction: controls < cases

rf02_auc <- auc(rf02_roc)
rf02_auc

## Area under the curve: 0.5636
```

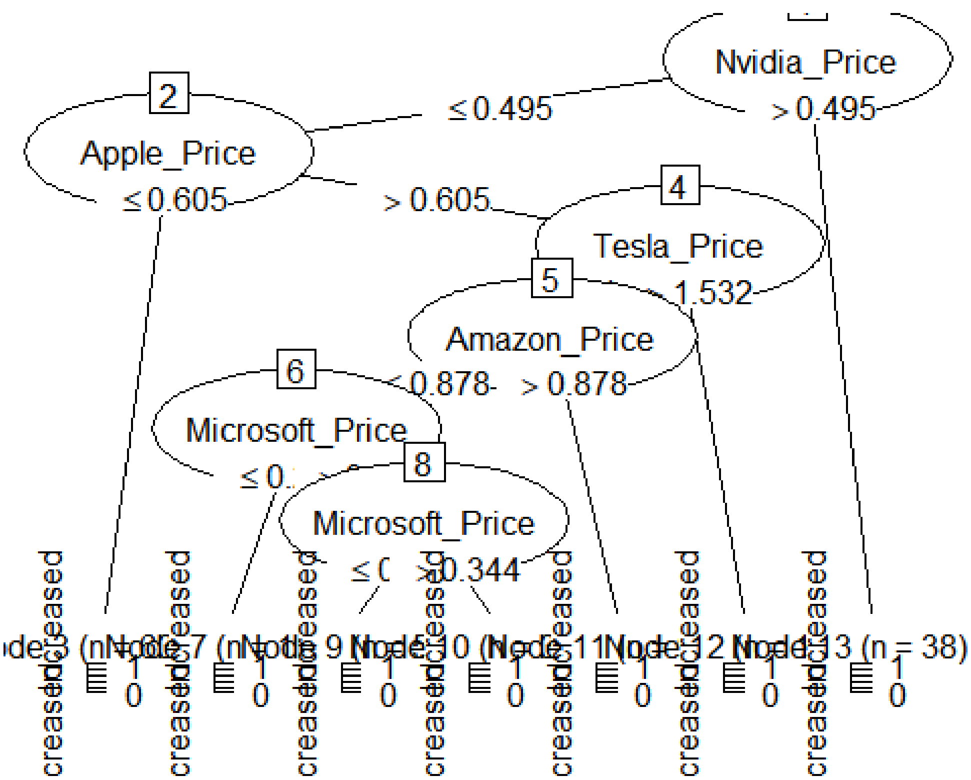
Random Forests with 100 ntree

```
set.seed(35)
rf03_model <- randomForest(formula = Bitcoin_Price_Change ~ Apple_Price + Tes
la_Price + Microsoft_Price + Google_Price + Nvidia_Price + Berkshire_Price +
Netflix_Price + Amazon_Price + Meta_Price, data = std_train, method = "class"
, ntree = 100)

ypred <- predict(object = rf03_model, newdata = std_test, type = "class")

# Contingency Table

t3 <- table(std_test$Bitcoin_Price_Change, ypred)
row.names(t3) <- c("Actual: Decrease", "Actual: Increase")
colnames(t3) <- c("Predicted: Decrease", "Predicted: Increase")
t3 <- addmargins(A = t3, FUN = list(Total = sum), quiet = TRUE)
t3

##                    ypred
##                     Predicted: Decrease Predicted: Increase Total
##   Actual: Decrease                   37                  96   133
##   Actual: Increase                   27                  86   113
##   Total                              64                 182   246
```

```
rf03_model$ntree

## [1] 100

# RF AUC score

rf03_roc <- roc(std_test$Bitcoin_Price_Change, as.numeric(ypred))

## Setting levels: control = Decreased, case = Increased

## Setting direction: controls < cases

rf03_auc <- auc(rf03_roc)
rf03_auc

## Area under the curve: 0.5196
```

C5.0

```
c5_model <- C5.0(formula = Bitcoin_Price_Change ~ Apple_Price + Tesla_Price +
 Microsoft_Price + Google_Price + Nvidia_Price + Berkshire_Price + Netflix_Pr
ice + Amazon_Price + Meta_Price, data = std_train, methond = "class")

plot(c5_model)
```

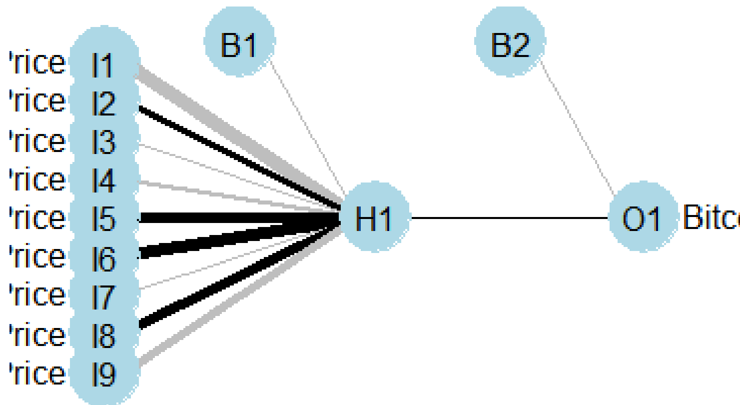



```
ypred <- predict(object = c5_model, newdata = std_test, type = "class")

t1 <- table(std_test$Bitcoin_Price_Change, ypred)
```

```
row.names(t1) <- c("Actual: Decrease", "Actual: Increase")
colnames(t1) <- c("Predicted: Decrease", "Predicted: Increase")
t1 <- addmargins(A = t1, FUN = list(Total = sum), quiet = TRUE)
t1

##                   ypred
##                    Predicted: Decrease Predicted: Increase Total
##   Actual: Decrease                  26                 107   133
##   Actual: Increase                  19                  94   113
##   Total                             45                 201   246
```

Neural Networks

```
set.seed(35)
nnet01_model <- nnet(formula = Bitcoin_Price_Change ~ Apple_Price + Tesla_Pri
ce +
                     Microsoft_Price + Google_Price + Nvidia_Price + Berkshir
e_Price +
                     Netflix_Price + Amazon_Price + Meta_Price, data = std_tr
ain, size = 1)

## # weights:  12
## initial  value 518.573865
## iter  10 value 486.752168
## iter  20 value 462.476803
## iter  30 value 458.052086
## iter  40 value 457.592883
## iter  50 value 456.919594
## iter  60 value 456.493738
## iter  70 value 456.328923
## iter  80 value 456.305374
## iter  90 value 456.275022
## iter 100 value 456.262995
## final  value 456.262995
## stopped after 100 iterations

plotnet(nnet01_model)
```




```
ypred <- predict(object = nnet01_model, newdata = std_test, type = "class")

t1 <- table(std_test$Bitcoin_Price_Change, ypred)
row.names(t1) <- c("Actual: Decrease", "Actual: Increase")
colnames(t1) <- c("Predicted: Decrease", "Predicted: Increase")
t1 <- addmargins(A = t1, FUN = list(Total = sum), quiet = TRUE)
t1

##                    ypred
##                     Predicted: Decrease Predicted: Increase Total
##   Actual: Decrease                   40                  93   133
##   Actual: Increase                   45                  68   113
##   Total                              85                 161   246
```

Naive Bayes

```
# Train the Naive Bayes model
nb_model <- naiveBayes(Bitcoin_Price_Change ~ Apple_Price + Tesla_Price + Mic
rosoft_Price + Google_Price + Nvidia_Price + Berkshire_Price + Netflix_Price
+ Amazon_Price + Meta_Price, data = std_train)

# Predict on the test set
nb_predictions_prob <- predict(nb_model, newdata = std_test, type = "raw")
nb_predictions <- ifelse(nb_predictions_prob[, "Increased"] >= 0.9, "Increase
d", "Decreased")

# Confusion matrix
```

```
conf_matrix <- table(std_test$Bitcoin_Price_Change, nb_predictions)

# Add proper row and column names
row.names(conf_matrix) <- c("Actual: Decrease", "Actual: Increase")
colnames(conf_matrix) <- c("Predicted: Decrease", "Predicted: Increase")

# Print the confusion matrix
print(conf_matrix)
```

```
##                   nb_predictions
##                    Predicted: Decrease Predicted: Increase
##   Actual: Decrease                  35                  98
##   Actual: Increase                  27                  86
```