DATA 606 Group project

2023-06-10

Group Members

Jelilat ADEBAYO Victor OLATOPE

Olayinka MOGAJI Ritah NABAWEESI

STOCK PRICE CLASSIFICATION: PREDICTING BUY, HOLD, OR SELL DECISIONS USING PRICE DATA FOR THE S&P 5OO Index

INTRODUCTION

When an investor decides to participate in the stock market, they are usually being driven by the need to either grow their wealth or store the value of their wealth. To achieve the investment goal, the investor will have to monitor the performance of their investment portfolio, either passively or aggressively to maximize growth or limit losses. However, for a first-time investor, they might not have the relevant trading expertise to make the best decisions on when to buy, hold or sell the stock.

The decision to buy, hold or sell a given stock is influenced by several factors such as one's risk tolerance (level of risk an investor is willing to take), investment time horizon (long-term, short-term, or medium-term) and financial goals. The process of deciding when to sell, hold or buy a particular stock is termed an investment strategy. This strategy would vary from one individual to another, and there are no one-size fits all. For example, "Common Stocks and Uncommon Profits" by Philip Fisher (1958) recommended identifying high-quality companies and emphasizes the importance of long-term investment horizons. Another literature, "The Little Book That Beats the Market" by Joel Greenblatt (2005) presents a simple and effective investment strategy called the "Magic Formula" that combines value and quality metrics to identify undervalued stocks.

One of the strategies widely adopted in the industry is the use of various technical indicators, either in isolation or in combination with other fundamental indicators. While using the technical indicators, the investor or investment advisor monitors the movements of the stock price in relation to the chosen technical indicators and the interaction of a combination of technical indicators. It is imperative that an investor makes an informed timely investment decision on their portfolio, in line with the set investment strategy and not act based on emotions.

OBJECTIVE

Our goal is to build a classification model based on prevailing stock market prices and designed investment strategy that informs the investor on when to buy, hold or sell the S&P 500 Index. This will ultimately ease the investment decision for the investor and enable real time decision making.

DATA SET

The data set with the daily stock prices for the S&P 500 Index, ticker symbol ^GSPC, was downloaded from Yahoo Finance through the use of the quantmod::getSymbols R package for the duration 01-01-2000 to 08-06-2023.

The default data set is as below:

```
library(quantmod)
```

```
#Defining the stock symbol and data range

stock_symbol <- "^GSPC" #S&P500 Index
start_date <- "2000-01-01"
end_date <- "2023-06-08"

#download the stock data
getSymbols(stock_symbol, from = start_date, to = end_date)
```

```
## [1] "GSPC"
```

```
#convert to dataframe

stock = as.data.frame(GSPC)
head(stock)
```

```
##            GSPC.Open GSPC.High GSPC.Low GSPC.Close GSPC.Volume GSPC.Adjusted
## 2000-01-03   1469.25   1478.00  1438.36    1455.22   931800000       1455.22
## 2000-01-04   1455.22   1455.22  1397.43    1399.42  1009000000       1399.42
## 2000-01-05   1399.42   1413.27  1377.68    1402.11  1085500000       1402.11
## 2000-01-06   1402.11   1411.90  1392.10    1403.45  1092300000       1403.45
## 2000-01-07   1403.45   1441.47  1400.73    1441.47  1225200000       1441.47
## 2000-01-10   1441.47   1464.36  1441.47    1457.60  1064800000       1457.60
```

The variables in the data frame above are explained:

• GSPC.Open: the price at which the stock index begins to trade at that date.

• GSPC.High: The highest price at which a stock index trades on a given day

• GSPC.Low: the lowest price at which a stock index trades during a given period

• GSPC.Close: the final price at which the stock index trades

• GSPC.Volume: the number of shares traded on that day

• GSPC.Adjusted: the stock price adjusted for dividends and splits and/or capital gain distributions.

METHODOLOGY

The success of the study is hinged on the right classification of the investment decision, to either buy, hold or sell the stock. In order to classify the information as either buy, hold or sell, we set out to design an investment strategy for this project on which to base the buy/sell signals.

A buy signal as an event or condition selected by a trader or investor as an alert for entering a purchase order for an investment. It follows that the sell signal is an event or condition selected by a trader or investor as an alert for entering a sell order for an investment.

The alert signals are defined to suit the investors risk profile and investment horizon. For this study, the alert signals were built on **four** key indicators; relative strength index, Stochastics and the price of the stock relative to the 50-day and 200-day exponential moving average (EMA) and the Volume confirmation.

The **relative strength index (RSI)** is a momentum indicator used in technical analysis. RSI measures the speed and magnitude of a security's recent price changes to evaluate overvalued or undervalued conditions in the price of that security (Source: https://www.investopedia.com/terms/r/rsi.asp (https://www.investopedia.com/terms/r/rsi.asp)). It compares the security's strength on days when prices go up to it's strength on days when prices go down.

Stochastics measures the current price of a stock relative to it's price range over a specific period of time. It has two components: %K and %D. - %K represents the current closing price of the stock relative to the highest and lowest prices over a defined period and it ranges from 0 to 100. When %K is near 0, it suggests the stock is trading near the lower end of its price range, indicating an oversold condition. When %K is near 100, it suggests the stock is trading near the upper end of its price range, indicating an overbought condition. - %D is the smoothed version of %K and is often represented as moving average of %K. (Source: https://www.investopedia.com/terms/s/stochasticoscillator.asp (https://www.investopedia.com/terms/s/stochasticoscillator.asp))

The commonly used thresholds by stock traders are %K above 80 suggests overbought conditions, indicating a potential selling opportunity while %K below 20 suggests oversold conditions, indicating a potential buying opportunity. This approach was adopted as part of the investment strategy.

In order to align with commonly used industry practice, the 50-day exponential moving average and 200-day moving average is adopted in deriving the trade signals for the study. The 50-day EMA is used to assess the short term patterns while the 200-day EMA is long term. The EMAs are used as support and resistance price levels. If the price is above the EMA, the EMA can serve as support level, i.e if the stock declines, the price might have a more difficult time falling below the EMA.Conversely, if the price is below the EMA, the EMA can serve as a strong resistance level in the sense that if the stock were to increase, the price might struggle to rise above the EMA.

For the investment strategy adopted,if the price falls below the support level that is interpreted as a sell signal while if the stock rises above the resistance level, that is a buy signal. A comparison of the short term moving averages with the long term moving averages, when the short term moving average crosses above the long term moving average it indicates a potential uptrend triggering a buy signal.

The volume confirmation that was adopted as part of the Investment strategy was an increase in trading volume as the price rises indicates a stronger buying interest hence a buy signal should be triggered.

Summary of Investment strategy

Signal	RSI	Stochastic..K	Price.EMA	Volume.EMA
Buy			ema50>=ema200	Vol > vol.ema200
Sell	>70	>80	Close < ema50 or close < ema200	Vol < vol.ema200
Hold	None of above satisfied			

Once the investment strategy was defined, the classes/signals (buy, sell, hold) were assigned based on the investment strategy.

```
#compute the technical indicators
rsi <- RSI(Cl(GSPC))

#compute the stochastics
stochastics <- stoch(GSPC[, c("GSPC.High", "GSPC.Low", "GSPC.Close")])

#compute moving average of volume
mvolume = EMA(Vo(GSPC), n = 200)

ma_50 <- EMA(Cl(GSPC), n = 50) #50-day moving average of the closing price
ma_200 <- EMA(Cl(GSPC), n = 200) #200-day moving average moving average of the closing price
```

```
# Generate buy/sell signals based on the moving averages RSI and stochastics
buy_signal.2 <- ifelse(ma_50 >= ma_200 | Vo(GSPC) > mvolume, "Buy", "")

sell_signal.2 <- ifelse(rsi > 70 && stochastics$fastK > 80  | Cl(GSPC) < ma_50 | Cl(GSPC) < ma_200, "Sell" , "")
hold_signal.2 <- ifelse(buy_signal.2 == "" & sell_signal.2 == "", "Hold", "")
```

```
##            GSPC.Open GSPC.High GSPC.Low GSPC.Close GSPC.Volume GSPC.Adjusted
## 2000-01-03   1469.25   1478.00  1438.36    1455.22   931800000       1455.22
## 2000-01-04   1455.22   1455.22  1397.43    1399.42  1009000000       1399.42
## 2000-01-05   1399.42   1413.27  1377.68    1402.11  1085500000       1402.11
## 2000-01-06   1402.11   1411.90  1392.10    1403.45  1092300000       1403.45
## 2000-01-07   1403.45   1441.47  1400.73    1441.47  1225200000       1441.47
##            Signals
## 2000-01-03    Hold
## 2000-01-04    Hold
## 2000-01-05    Hold
## 2000-01-06    Hold
## 2000-01-07    Hold
```

To the default data set, a number of derived variables were added to the data set as these were critical in the prediction of the response varible, Signals. The derived variables include:

- Lag returns : Lag 1, 10,15,50
- Exponential moving averages of the daily returns: ema_10 and ema_50

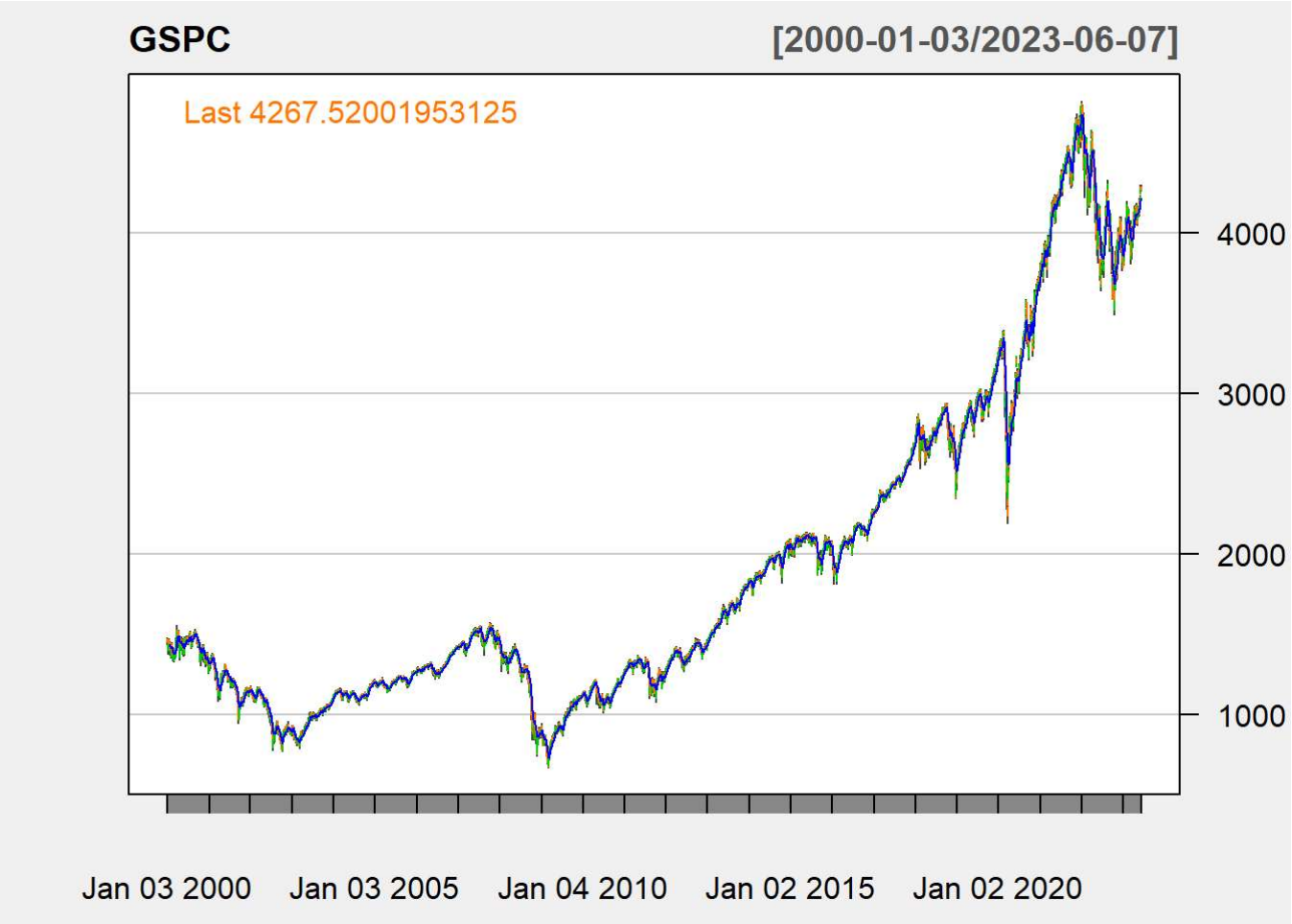
The choice of EMA is based on the fact that it is a weighted moving average which assigns more weight to the most recent market data.

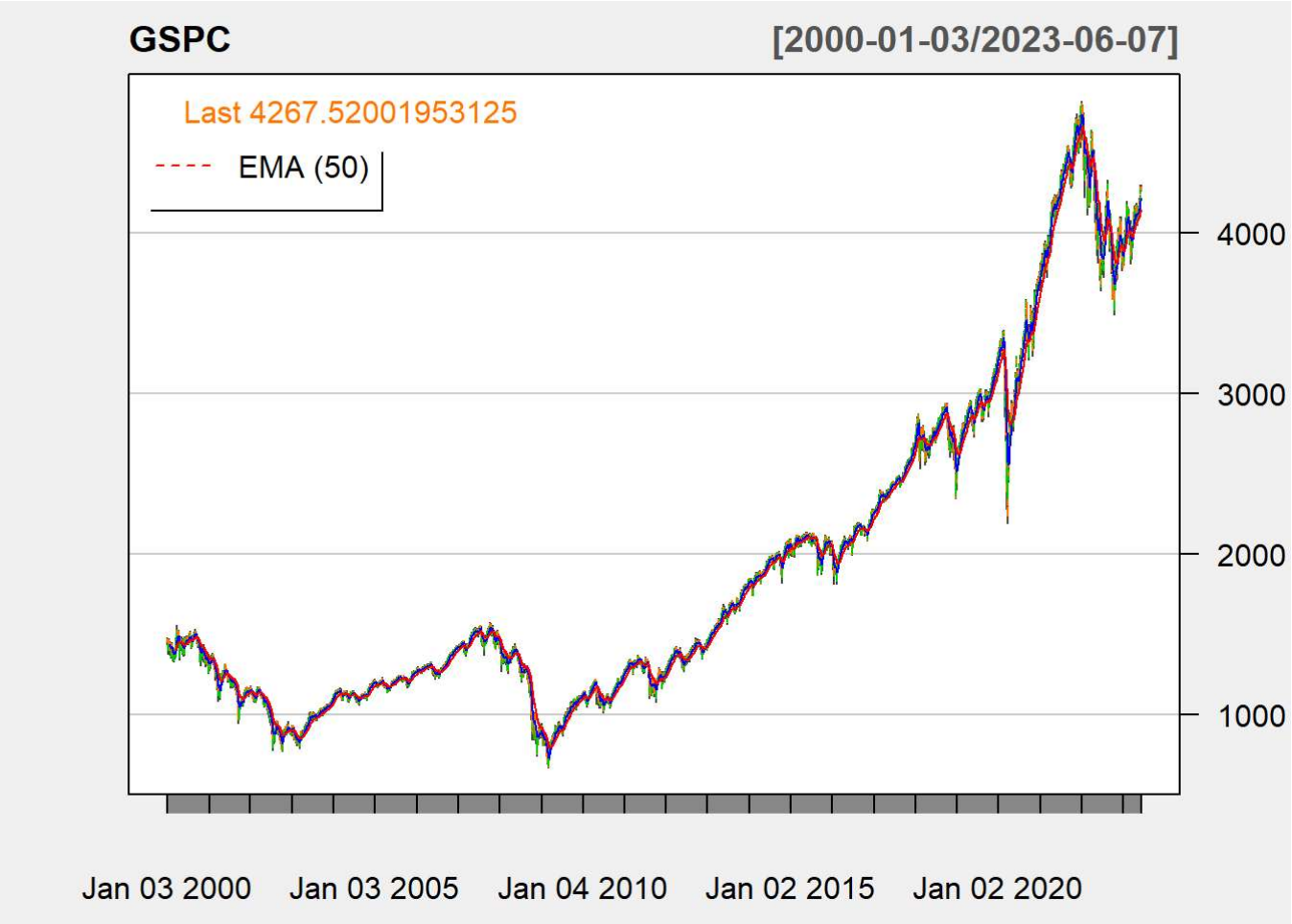
```
library(dplyr)

#Lagged returns
stock <- stock %>% mutate(Lag1 = (GSPC.Close - lag(GSPC.Close, n = 1))/lag(GSPC.Close, n = 1)*100)
stock <- stock %>% mutate(Lag10 = (GSPC.Close - lag(GSPC.Close, n = 10))/lag(GSPC.Close, n = 10)*100)
stock <- stock %>% mutate(Lag15 = (GSPC.Close - lag(GSPC.Close, n = 15))/lag(GSPC.Close, n = 15)*100)
stock <- stock %>% mutate(Lag50 = (GSPC.Close - lag(GSPC.Close, n = 50))/lag(GSPC.Close, n = 50)*100)
#stock <- stock %>% mutate(Lag200 = (GSPC.Close - lag(GSPC.Close, n = 200))/lag(GSPC.Close, n = 200)*100)

#compute the returns and exponential moving average of returns, based on the closing price of the stock. The time frames can
be adjusted, it's not cast in stone.
returns <- dailyReturn(Cl(GSPC))
ema_50 <- EMA(returns, n = 50)
ema_10 <- EMA(returns, n = 10)

#Add the columns to the dataframe
stock$ema_10 <- ema_10
stock$ema_50 <- ema_50
```

The exploratory analysis of the variables was performed and the model building exercise followed. Based on the consolidated data set, the response variable is Signals while the predictors are Lag 1 through to 50 and ema_10 to ema_50.

EXPLORATORY ANALYSIS OF THE DATA SET

Trend Analysis: S&P500 Daily prices with Exponential Moving Averages indicators


GSPC
[2000-01-03/2023-06-07]
Last 4267.52001953125
4000
3000
2000
1000
Jan 03 2000
Jan 03 2005
Jan 04 2010
Jan 02 2015
Jan 02 2020



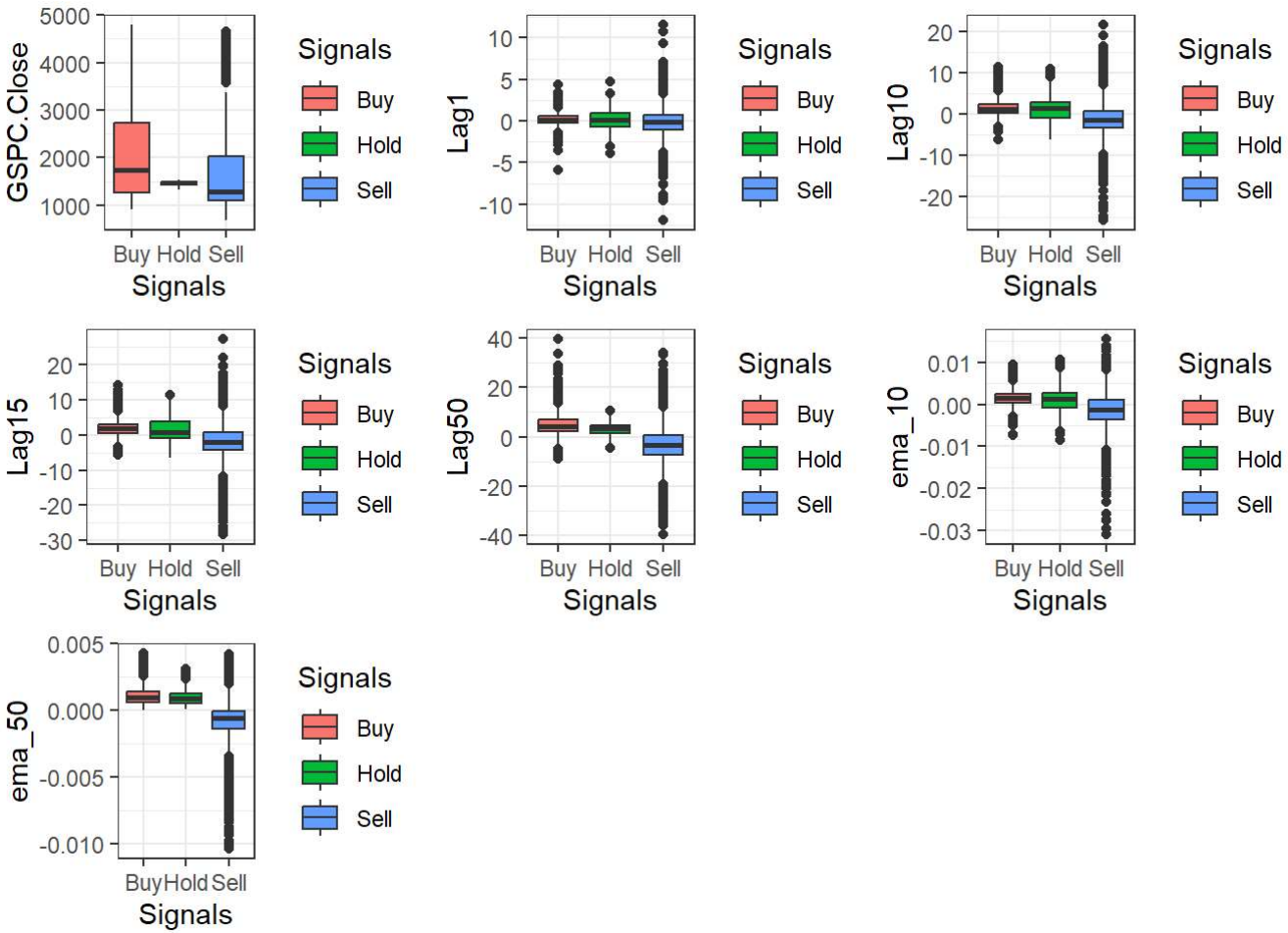
Figures 1 and 2 show the S&P500 Daily prices with Exponential Moving Averages indicators. From the time series plot, the closing price of the S&P 500 Index has been on an upward trend from 2000 to June 08, 2023. In 2008, there was a drop in stock prices – an impact of the US economic crisis at the time. However, beyond this point, the stock price has risen from ~$1,000 to ~$4,000 in 2023. This suggest that an investor that bought shares between the 2008 – 2010 and held unto them would be 300% time richer in 2023.

Another step taken was to assess if there is any relationship between the daily returns on the shares and the volumes being traded. Figure 3 shows there is no relationship between the range of the returns and entire spread of the volume of shares.

The next step was to evaluate the relationship between the Signal classifications and the investment strategy parameters, i.e., Lag1 to 50 and ema50 and 200.

Box plot analysis: Distributions of the Signals across the predictor variables

```r
library(ggplot2)

library(gridExtra)

plot1 <- ggplot(stock, aes(x = Signals, y = GSPC.Close, fill = Signals)) +
  geom_boxplot() +
  theme_bw()

plot2 <- ggplot(stock, aes(x = Signals, y = Lag1, fill = Signals)) +
  geom_boxplot() +
  theme_bw()

plot3 <- ggplot(stock, aes(x = Signals, y = Lag10, fill = Signals)) +
  geom_boxplot() +
  theme_bw()

plot4 <- ggplot(stock, aes(x = Signals, y = Lag15, fill = Signals)) +
  geom_boxplot() +
  theme_bw()

plot5 <- ggplot(stock, aes(x = Signals, y = Lag50, fill = Signals)) +
  geom_boxplot() +
  theme_bw()

plot6 <- ggplot(stock, aes(x = Signals, y = ema_10, fill = Signals)) +
  geom_boxplot() +
  theme_bw()

plot7 <- ggplot(stock, aes(x = Signals, y = ema_50, fill = Signals)) +
  geom_boxplot() +
  theme_bw()

# Arrange the plots as subplots
grid.arrange(plot1, plot2, plot3, plot4, plot5, plot6, plot7, ncol = 3)
```




```
#plot6
#plot7
```

-Lag1:

The median between the groups is not significantly different

Hence may aid the classification

-Lag10:

The median between buy and hold group is not significantly, however relative to Sell, there is a step change.

These would be relevant for classification

-Lag15: same observation as Lag10

-Lag50: same observation as Lag10

-GPSC.Close: The median between the groups not significantly different. Hence may aid the classification.

MODEL BUILDING

In this section, with the Signals as the response variable, we focus on four models: -Linear Discriminant Analysis (LDA) -Quadratic Discriminant Analysis (QDA) -Classification Tree -Multinomial Logistic regression

Using the stratified sampling method, the data set was split into 75% training and 25% test set.

```
# Remove the NA values
stock.1 <- na.omit(stock)

#Checking order of the classes
unique(stock.1$Signals)
```

```
## [1] "Sell" "Hold" "Buy"
```

```
#checking number of observations in each class
table(stock.1$Signals)
```

```
##
##  Buy Hold Sell
## 3337   89 2419
```

Split the data set into 75% training and 25% as test set

```
library(sampling)
```

```
## Warning: package 'sampling' was built under R version 4.2.3
```

```
library(survey)
set.seed(2023)

idx=sampling:::strata(stock.1, stratanames=c("Signals"), size=c(1814,67,2503), method="srswor")
train_data =stock.1[idx$ID_unit,] #specifies the rows to pick from the dataset
test_data = stock.1[-idx$ID_unit,]
```

Linear Discriminant Analysis (LDA)

The LDA model is hinged on two assumptions: -The multivariate normality -Equality of variances

We test first assumption of the model

The multivariate normality test

This energy test checks whether several variables are normally distributed as a group. The hypothesis at test is:

Ho: The variables follow a multivariate normal distribution

H1: The variables do not follow a multivariate normal distribution

```
library(energy)
variables <- stock.1[, c("Lag1", "Lag10", "Lag15", "Lag50", "ema_10","ema_50")]
mvnorm.etest(variables, R = 100)
```

```
##
##  Energy test of multivariate normality: estimated parameters
##
## data:  x, sample size 5845, dimension 6, replicates 100
## E-statistic = 283.97, p-value < 2.2e-16
```

Since the p-value < 2.2e-16 is less than 0.05, we reject the null hypothesis. There is sufficient evidence to support the alternative hypothesis that the variables do not follow a multivariate normal distribution. We however choose to keep the variables for the model.

Training the model on the train data set

```
library(MASS)
lda_stock <- lda(Signals ~ Lag1 + Lag10 + Lag15 + Lag50 + ema_10 + ema_50, data = train_data)

lda_stock
```

```
## Call:
## lda(Signals ~ Lag1 + Lag10 + Lag15 + Lag50 + ema_10 + ema_50,
##     data = train_data)
##
## Prior probabilities of groups:
##        Buy       Hold       Sell
## 0.57093978 0.01528285 0.41377737
##
## Group means:
##            Lag1     Lag10     Lag15     Lag50        ema_10        ema_50
## Buy   0.1496736  1.352502  1.943070  4.767626  4.283311e-04  4.465177e-04
## Hold  0.2879856  1.697971  2.495395  3.065633  4.521306e-04  6.579119e-04
## Sell -0.1803765 -1.458917 -1.888613 -3.698834 -5.397563e-05 -1.961629e-05
##
## Coefficients of linear discriminants:
##                   LD1           LD2
## Lag1   -2.698426e-03 -1.524702e-01
## Lag10  -5.350860e-02  4.169729e-03
## Lag15  -7.036799e-02 -1.838842e-01
## Lag50  -1.369495e-01  1.120126e-01
## ema_10  1.886801e+01  1.618416e+02
## ema_50 -1.741827e+02 -5.882135e+02
##
## Proportion of trace:
##    LD1    LD2
## 0.9958 0.0042
```

The prior probabilities for each group indicate that:

Buy: suggests that approximately 57.1% of the observations in the data set or population are classified as “Buy” based on the available information.

Hold: indicates that around 1.5% of the observations are categorized as “Hold” based on the given data.

Sell: about 41.4% of the observations fall into the “Sell” category according to the available information.

The model is fitted to the test data set and the confusion matrix is derived

```
invest_pred <- predict(lda_stock, test_data)
ldamatrix = table(invest_pred$class, test_data$Signals)
ldamatrix
```

```
##
##        Buy Hold Sell
##   Buy  820   21  164
##   Hold   0    0    0
##   Sell  14    1  441
```

The LDA model doesn't return any prediction for the Hold class. From the confusion matrix, 200 of the 1,461 observations were wrongly classified.

```
misclasslda = 1 - sum(diag(ldamatrix))/sum(ldamatrix)
misclasslda
```

```
## [1] 0.1368925
```

The misclassification rate is 0.137 implying 86.3% of the time the model will correctly predict the signals based on the chosen response variables of Lag 1 through to 50 and ema_10 - ema_50.

Quadratic Discriminant Analysis (QDA)

Using the same test and train data sets that were established earlier, we train the QDA model.

```
qda.stock<-qda(Signals ~ Lag1 + Lag10 + Lag15 + Lag50 + ema_10 + ema_50, data = train_data)
qda.stock
```

```
## Call:
## qda(Signals ~ Lag1 + Lag10 + Lag15 + Lag50 + ema_10 + ema_50,
##     data = train_data)
##
## Prior probabilities of groups:
##        Buy       Hold       Sell
## 0.57093978 0.01528285 0.41377737
##
## Group means:
##            Lag1     Lag10     Lag15     Lag50        ema_10        ema_50
## Buy   0.1496736  1.352502  1.943070  4.767626  4.283311e-04  4.465177e-04
## Hold  0.2879856  1.697971  2.495395  3.065633  4.521306e-04  6.579119e-04
## Sell -0.1803765 -1.458917 -1.888613 -3.698834 -5.397563e-05 -1.961629e-05
```

The interpretation of the prior probabilities is similar to that of the LDA model. The QDA model was fitted on the test data set and the confusion matrix derived

```
qda.class<-predict(qda.stock, test_data)$class
matrixqda = table(qda.class, test_data$Signals)
matrixqda
```

```
##
## qda.class Buy Hold Sell
##      Buy  791   12  174
##      Hold   2    0    0
##      Sell  41   10  431
```

Unlike the LDA model, the QDA manages to predict the Hold class unfortunately the prediction for this class is wrong.From the confusion matrix, 239 of the 1,461 observations were wrongly classified. As a result, the misclassification rate of the model is 16.36%.Hence the accuracy rate is 83.64% based on the test data.

The accuracy rate of the QDA model is less than that of the LDA.

```
#compute the misclassification
misclassqda = 1 - sum(diag(matrixqda))/sum(matrixqda)
misclassqda
```

```
## [1] 0.1635866
```

The Classification Tree

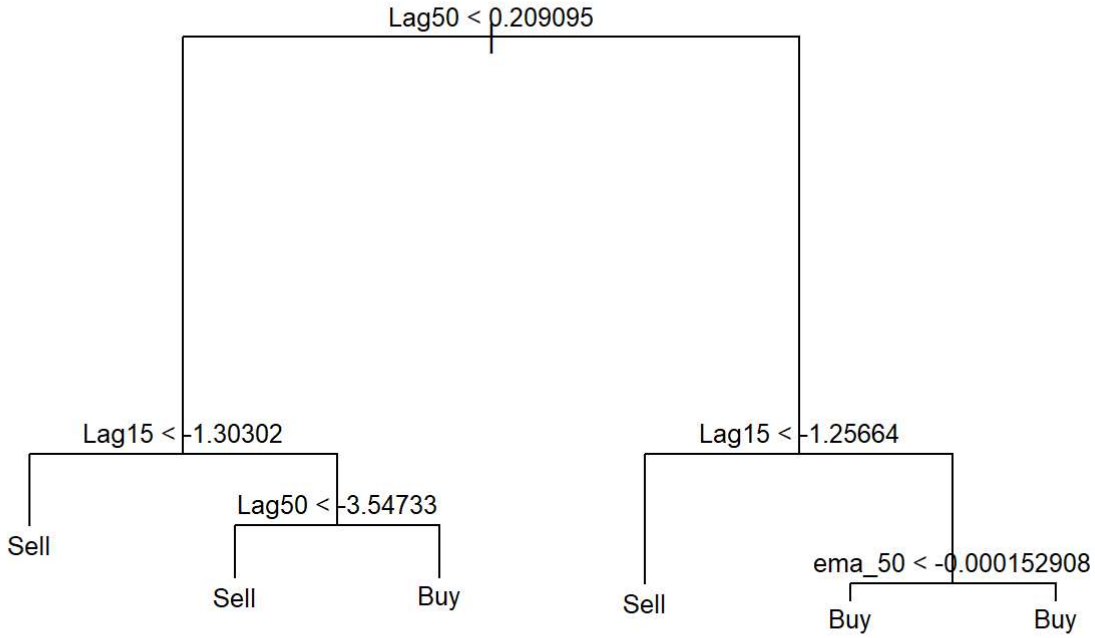
```
library(tree)
tree.stock <- tree(factor(Signals) ~ Lag1 + Lag10 + Lag15 + Lag50 + ema_10 + ema_50, train_data)
summary(tree.stock)
```

```
##
## Classification tree:
## tree(formula = factor(Signals) ~ Lag1 + Lag10 + Lag15 + Lag50 +
##     ema_10 + ema_50, data = train_data)
## Variables actually used in tree construction:
## [1] "Lag50"  "Lag15"  "ema_50"
## Number of terminal nodes:  6
## Residual mean deviance:  0.7349 = 3218 / 4378
## Misclassification error rate: 0.1341 = 588 / 4384
```

From the defined predictor variables, the classification tree is actually constructed from 3 variables: Lag50, Lag15 and ema_50.

The classification tree has a misclassification error rate of 13.41%. Hence the accuracy rate is 86.59% based on the test data. This is the highest accuracy rate amongst the 3 models developed to this stage.

The classification tree is plotted below:



The classification has 6 terminal nodes, of which the two terminal nodes from ema_50 < -0.0001522908 branch have the same predicted value of Buy. The split on this node leads to improved node purity.

Based on our judgement, the 6 nodes of the tree are not complex hence the tree pruning is not performed at this point.

```
stock.test <- predict(tree.stock, test_data, type="class")
matrixtree = table(stock.test, test_data$Signals )
matrixtree
```

```
##
## stock.test Buy Hold Sell
##       Buy  793   20  151
##       Hold   0    0    0
##       Sell  41    2  454
```

```
#compute the misclassification
misclasstree = 1 - sum(diag(matrixtree))/sum(matrixtree)
misclasstree
```

```
## [1] 0.146475
```

```
1-misclasstree
```

```
## [1] 0.853525
```

Multinomial regression using logit model

```
library(VGAM)
mlogit2=vglm(Signals ~ Lag1 + Lag10 + Lag15 + Lag50 + ema_10 + ema_50,family=multinomial,data=train_data)
summary(mlogit2)
```

```
##
## Call:
## vglm(formula = Signals ~ Lag1 + Lag10 + Lag15 + Lag50 + ema_10 +
##     ema_50, family = multinomial, data = train_data)
##
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept):1   -0.26571    0.04876  -5.449 5.06e-08 ***
## (Intercept):2   -3.86431    0.20737 -18.635  < 2e-16 ***
## Lag1:1           0.09185    0.04612   1.992 0.046420 *
## Lag1:2           0.18054    0.13985   1.291 0.196716
## Lag10:1          0.16468    0.02540   6.482 9.02e-11 ***
## Lag10:2          0.15486    0.06790   2.281 0.022561 *
## Lag15:1          0.17058    0.02101   8.117 4.76e-16 ***
## Lag15:2          0.22490    0.05287   4.254 2.10e-05 ***
## Lag50:1          0.32507    0.01152  28.210  < 2e-16 ***
## Lag50:2          0.24664    0.02779   8.875  < 2e-16 ***
## ema_10:1       -26.92769   18.81081  -1.432 0.152287
## ema_10:2      -141.00521   69.67262  -2.024 0.042988 *
## ema_50:1       252.12107   46.38424   5.435 5.46e-08 ***
## ema_50:2       637.15298  180.11222   3.538 0.000404 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Names of linear predictors: log(mu[,1]/mu[,3]), log(mu[,2]/mu[,3])
##
## Residual deviance: 3929.881 on 8754 degrees of freedom
##
## Log-likelihood: -1964.94 on 8754 degrees of freedom
##
## Number of Fisher scoring iterations: 8
##
## Warning: Hauck-Donner effect detected in the following estimate(s):
## '(Intercept):2'
##
##
## Reference group is level  3  of the response
```

The reference group level 3 is the “Buy” group. Level 1 and Level 2 refer to the two categories of the response variable, namely “Sell” and “Hold,” respectively. The coefficients for Level 1 and Level 2 represent the differences in log-odds between those categories and the reference category (Buy).

```
coef(mlogit2)
```

```
## (Intercept):1 (Intercept):2        Lag1:1        Lag1:2       Lag10:1
##   -0.26571242   -3.86431085    0.09185494    0.18054128    0.16467685
##       Lag10:2       Lag15:1       Lag15:2       Lag50:1       Lag50:2
##    0.15485969    0.17057807    0.22489853    0.32507489    0.24663585
##      ema_10:1      ema_10:2      ema_50:1      ema_50:2
##  -26.92768594 -141.00521313  252.12107177  637.15297535
```

> A check of the goodness of fit for the model.

The test of hypothesis is:

Ho: The multinomial logistic model fits the data well

H1: The multinomial logistic model does not fit the data well

```
1-pchisq(deviance(mlogit2),df.residual(mlogit2))
```

```
## [1] 1
```

The output is 1, it means that the p-value is equal to 1, indicating that there is no evidence to reject the null hypothesis. In other words, the model fits the data well, and there is no significant lack of fit.

The model has an accuracy rate of 85.93% based on the test data.

```
library(Rfast)
prob.fit<-fitted(mlogit2)
fitted.result<-colnames(prob.fit)[rowMaxs(prob.fit)]
misClasificError <- mean(fitted.result != train_data$Signals)
print(paste('MAccuracy',1-misClasificError))
```

```
## [1] "MAccuracy 0.85926094890511"
```

```
confusion_matrix <- table(fitted.result, train_data$Signals)
print(confusion_matrix)
```

```
##
## fitted.result  Buy Hold Sell
##          Buy  2312   64  359
##          Sell  191    3 1455
```

CROSS VALIDATION OF THE MODELS

The four models that have been designed to this point have provided varying accuracy rates. In order to select the best model, we performed a cross validation of the models to assess the appropriateness of the methodology that we chose. The cross validation is performed on the entire data set to eliminate dependency on the partition that was created when we split the data into train and test parts.

Using the k-fold cross validation approach, the k was set to 10-folds for each of the models.

Cross validation of the LDA model

```
#Cross validation for LDA
set.seed(2023)
library(caret)
model_fit1<-train(Signals ~ Lag1 + Lag10 + Lag15 + Lag50 + ema_10 + ema_50, data = stock.1, trControl = trainControl(method
= "cv", number=10), method='lda')
model_fit1
```

```
## Linear Discriminant Analysis
##
## 5845 samples
##    6 predictor
##    3 classes: 'Buy', 'Hold', 'Sell'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 5260, 5260, 5260, 5261, 5260, 5261, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.8579982  0.7018376
```

The model accuracy rate is 85.80% based on the entire data set.

Cross validation of the QDA model

```
set.seed(2023)
model_fit2<-train(Signals ~ Lag1 + Lag10 + Lag15 + Lag50 + ema_10 + ema_50, data = stock.1, trControl = trainControl(method
= "cv", number=10), method='qda')
model_fit2
```

```
## Quadratic Discriminant Analysis
##
## 5845 samples
##    6 predictor
##    3 classes: 'Buy', 'Hold', 'Sell'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 5260, 5260, 5260, 5261, 5260, 5261, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.8528624  0.6977157
```

The accuracy of the QDA model is 85.27% based on the entire data set.

Cross validation of the classification tree

```
library(rpart)

set.seed(2023)


folds <- createFolds(factor(stock.1$Signals), k=10) # stratified k-fold



# assessing one of the fold of the stratified folds

fold1<-stock.1[folds$Fold1,]

table(fold1$Signals)
```

```
##
##  Buy Hold Sell
##  333    9  242
```

```
misclass_class_tree <- function(idx){

  train_class_tree <- stock[-idx,]

  testclass_tree <- stock[idx,]


  fit_class_tree <-rpart(Signals ~ Lag1 + Lag10 + Lag15 + Lag50 + ema_10 + ema_50, data = stock.1)

  pred_class_tree <- predict(fit_class_tree,testclass_tree, type = "class")


  return(1- mean(pred_class_tree == testclass_tree$Signals))

}


mis_rate_class_tree = lapply(folds, misclass_class_tree)

1 - mean(as.numeric(mis_rate_class_tree))
```

```
## [1] 0.9113751
```

The accuracy rate of the classification tree is 91.13% based on the entire data set.

Cross validation of the multinomial logistic model

```
set.seed(2023)
model_fit3 <- train(Signals ~ Lag1 + Lag10 + Lag15 + Lag50 + ema_10 + ema_50, data = stock.1, trControl = trainControl(metho
d = "cv", number=10), method='multinom')
```

```
## # weights:  24 (14 variable)
## initial  value 5778.700638
## iter  10 value 2481.860232
## iter  20 value 2330.268054
## iter  30 value 2275.209200
## iter  40 value 2273.936998
## iter  50 value 2229.686251
## iter  60 value 2167.125618
## iter  70 value 2165.102298
## iter  80 value 2149.196200
## iter  90 value 2143.712630
## iter 100 value 2143.536319
## final  value 2143.536319
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 5778.700638
## iter  10 value 2482.889075
## iter  20 value 2332.358632
## final  value 2332.322506
## converged
## # weights:  24 (14 variable)
## initial  value 5778.700638
## iter  10 value 2481.861261
## iter  20 value 2330.272627
## iter  30 value 2291.150452
## iter  40 value 2290.736101
## iter  50 value 2276.785849
## iter  60 value 2270.976611
## iter  70 value 2270.893985
## iter  80 value 2270.123537
## iter  90 value 2269.971321
## final  value 2269.970387
## converged
## # weights:  24 (14 variable)
## initial  value 5778.700638
## iter  10 value 2745.728274
## iter  20 value 2414.133670
## iter  30 value 2352.459749
## iter  40 value 2349.056509
```

```
## iter  50 value 2299.542904
## iter  60 value 2262.031462
## iter  70 value 2256.168154
## iter  80 value 2216.340002
## iter  90 value 2195.206725
## iter 100 value 2194.885618
## final  value 2194.885618
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 5778.700638
## iter  10 value 2746.799190
## iter  20 value 2416.352755
## final  value 2416.316877
## converged
## # weights:  24 (14 variable)
## initial  value 5778.700638
## iter  10 value 2745.729347
## iter  20 value 2414.138990
## iter  30 value 2369.776815
## iter  40 value 2368.164517
## iter  50 value 2351.728215
## iter  60 value 2345.807764
## iter  70 value 2345.586513
## iter  80 value 2343.759486
## iter  90 value 2342.991386
## final  value 2342.989606
## converged
## # weights:  24 (14 variable)
## initial  value 5778.700638
## iter  10 value 2665.305136
## iter  20 value 2380.700819
## iter  30 value 2318.955286
## iter  40 value 2318.248078
## iter  50 value 2275.996473
## iter  60 value 2253.611135
## iter  70 value 2251.030483
## iter  80 value 2171.334579
## iter  90 value 2151.731099
## final  value 2151.730929
```

```
## converged
## # weights:  24 (14 variable)
## initial  value 5778.700638
## iter  10 value 2666.306489
## iter  20 value 2383.044886
## final  value 2383.011242
## converged
## # weights:  24 (14 variable)
## initial  value 5778.700638
## iter  10 value 2665.306139
## iter  20 value 2380.707170
## iter  30 value 2335.963780
## iter  40 value 2335.852198
## iter  50 value 2320.962101
## iter  60 value 2315.994594
## iter  70 value 2315.645573
## iter  80 value 2313.756058
## iter  90 value 2313.128093
## iter 100 value 2313.120979
## final  value 2313.120979
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 5779.799251
## iter  10 value 2640.308650
## iter  20 value 2392.767433
## iter  30 value 2331.029537
## iter  40 value 2330.100372
## iter  50 value 2288.737125
## iter  60 value 2266.043244
## iter  70 value 2264.920454
## iter  80 value 2201.567495
## iter  90 value 2159.034168
## final  value 2159.033794
## converged
## # weights:  24 (14 variable)
## initial  value 5779.799251
## iter  10 value 2640.615632
## iter  20 value 2395.143826
## final  value 2395.124135
```

```
## converged
## # weights:  24 (14 variable)
## initial  value 5779.799251
## iter  10 value 2640.308956
## iter  20 value 2392.773934
## iter  30 value 2347.860142
## iter  40 value 2347.687828
## iter  50 value 2332.407838
## iter  60 value 2327.156381
## iter  70 value 2326.286639
## iter  80 value 2324.592202
## final  value 2324.348149
## converged
## # weights:  24 (14 variable)
## initial  value 5778.700638
## iter  10 value 2497.156928
## iter  20 value 2331.234161
## iter  30 value 2275.095103
## iter  40 value 2274.420465
## iter  50 value 2232.081440
## iter  60 value 2202.944860
## iter  70 value 2196.362159
## iter  80 value 2154.670955
## iter  90 value 2145.031276
## iter 100 value 2144.742028
## final  value 2144.742028
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 5778.700638
## iter  10 value 2497.412526
## iter  20 value 2333.383460
## final  value 2333.352634
## converged
## # weights:  24 (14 variable)
## initial  value 5778.700638
## iter  10 value 2497.157181
## iter  20 value 2331.239629
## iter  30 value 2292.329364
## iter  40 value 2292.058598
```

```
## iter  50 value 2278.716141
## iter  60 value 2274.121704
## iter  70 value 2273.652940
## iter  80 value 2272.141268
## iter  90 value 2271.961027
## iter  90 value 2271.961022
## final  value 2271.960866
## converged
## # weights:  24 (14 variable)
## initial  value 5779.799251
## iter  10 value 2680.960838
## iter  20 value 2369.169760
## iter  30 value 2308.260691
## iter  40 value 2307.215626
## iter  50 value 2264.724870
## iter  60 value 2243.318557
## iter  70 value 2240.269828
## iter  80 value 2168.876882
## iter  90 value 2143.708147
## iter 100 value 2143.302491
## final  value 2143.302491
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 5779.799251
## iter  10 value 2680.819678
## iter  20 value 2371.432157
## final  value 2371.415578
## converged
## # weights:  24 (14 variable)
## initial  value 5779.799251
## iter  10 value 2680.960695
## iter  20 value 2369.176053
## iter  30 value 2325.388879
## iter  40 value 2324.946883
## iter  50 value 2310.465253
## iter  60 value 2306.031671
## iter  70 value 2305.886274
## iter  80 value 2302.997412
## iter  90 value 2302.410034
```

```
## iter 100 value 2302.392720
## final  value 2302.392720
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 5779.799251
## iter  10 value 2642.628719
## iter  20 value 2383.188209
## iter  30 value 2323.828410
## iter  40 value 2322.335439
## iter  50 value 2277.008606
## iter  60 value 2233.711682
## iter  70 value 2231.262127
## iter  80 value 2209.285504
## iter  90 value 2194.231954
## iter 100 value 2193.054528
## final  value 2193.054528
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 5779.799251
## iter  10 value 2643.191825
## iter  20 value 2385.505695
## final  value 2385.480691
## converged
## # weights:  24 (14 variable)
## initial  value 5779.799251
## iter  10 value 2642.629282
## iter  20 value 2383.194533
## iter  30 value 2340.383421
## iter  40 value 2340.231422
## iter  50 value 2324.886012
## final  value 2316.908819
## converged
## # weights:  24 (14 variable)
## initial  value 5779.799251
## iter  10 value 2680.660837
## iter  20 value 2396.194714
## iter  30 value 2337.645330
## iter  40 value 2336.569039
## iter  50 value 2295.343471
```

```
## iter  60 value 2271.020814
## iter  70 value 2267.238365
## iter  80 value 2178.528138
## iter  90 value 2175.043686
## iter 100 value 2174.940336
## final  value 2174.940336
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 5779.799251
## iter  10 value 2681.767910
## iter  20 value 2398.409369
## final  value 2398.377867
## converged
## # weights:  24 (14 variable)
## initial  value 5779.799251
## iter  10 value 2680.661945
## iter  20 value 2396.199940
## iter  30 value 2354.001637
## iter  40 value 2353.671319
## iter  50 value 2338.763242
## iter  60 value 2332.687952
## iter  70 value 2332.438229
## iter  80 value 2331.458339
## iter  90 value 2331.064480
## iter 100 value 2331.060356
## final  value 2331.060356
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 5779.799251
## iter  10 value 2686.373887
## iter  20 value 2383.866421
## iter  30 value 2325.994537
## iter  40 value 2323.009436
## iter  50 value 2292.296604
## iter  60 value 2272.277483
## iter  70 value 2270.879561
## iter  80 value 2240.458970
## iter  90 value 2230.057757
## iter 100 value 2229.873542
```

```
## final  value 2229.873542
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 5779.799251
## iter  10 value 2687.404634
## iter  20 value 2386.099569
## final  value 2386.068842
## converged
## # weights:  24 (14 variable)
## initial  value 5779.799251
## iter  10 value 2686.374919
## iter  20 value 2383.872061
## iter  30 value 2341.429241
## iter  40 value 2340.755731
## iter  50 value 2325.878693
## iter  60 value 2319.306078
## iter  70 value 2318.995499
## iter  80 value 2318.538081
## iter  90 value 2318.457717
## final  value 2318.457447
## converged
## # weights:  24 (14 variable)
## initial  value 5778.700638
## iter  10 value 2729.505128
## iter  20 value 2413.542747
## iter  30 value 2355.362734
## iter  40 value 2354.034895
## iter  50 value 2310.363615
## iter  60 value 2281.703286
## iter  70 value 2277.140533
## iter  80 value 2228.865519
## iter  90 value 2218.182798
## iter 100 value 2218.041045
## final  value 2218.041045
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 5778.700638
## iter  10 value 2730.567853
## iter  20 value 2415.783432
```

```
## final  value 2415.757078
## converged
## # weights:  24 (14 variable)
## initial  value 5778.700638
## iter  10 value 2729.506192
## iter  20 value 2413.548740
## iter  30 value 2371.236284
## iter  40 value 2370.998142
## iter  50 value 2356.258694
## iter  60 value 2351.721961
## iter  70 value 2351.677497
## iter  80 value 2349.039644
## iter  90 value 2348.206880
## iter 100 value 2348.195134
## final  value 2348.195134
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 6421.388827
## iter  10 value 3032.528076
## iter  20 value 2644.517407
## iter  30 value 2578.164278
## iter  40 value 2577.279219
## iter  50 value 2528.285039
## iter  60 value 2485.777281
## iter  70 value 2481.172758
## iter  80 value 2454.725612
## iter  90 value 2438.811972
## iter 100 value 2435.825085
## final  value 2435.825085
## stopped after 100 iterations
```

```
model_fit3
```

```
## Penalized Multinomial Regression
##
## 5845 samples
##    6 predictor
##    3 classes: 'Buy', 'Hold', 'Sell'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 5260, 5260, 5260, 5261, 5260, 5261, ...
## Resampling results across tuning parameters:
##
##   decay  Accuracy   Kappa
##   0e+00  0.8798990  0.7539527
##   1e-04  0.8686061  0.7307316
##   1e-01  0.8610792  0.7154482
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was decay = 0.
```

The logit model applied to the entire data set has an accuracy rate of 87.99%

CONCLUSION

Based on the adopted investment strategy and the set of predictors, the accuracy rates of the four models are summarized below:

Summary of model accuracy rates

Model	Accuracy.rate
LDA	85.80%
QDA	85.27%
Classification tree	91.14%
Multinomial logistic model	87.77%

The classification tree with the highest accuracy rate of 91.14% offers a better prediction of the buy, hold and sell decisions of the S&P500 Index based on the adopted investment strategy.

References

1. Buy, Sell, and Hold Decisions: What to Consider - Ticker Tape. (n.d.). The Ticker Tape. https://tickertape.tdameritrade.com/investing/buy-sell-hold-strategies-new-investors-17466# (https://tickertape.tdameritrade.com/investing/buy-sell-hold-strategies-new-investors-17466#):~:text=Technical%20analysis%20is%20often%20used,%2C%20sell%2C%20and%20hold%20decisions
2. Relative Strength Index (RSI) Indicator Explained With Formula. (2023, March 31). Investopedia. https://www.investopedia.com/terms/r/rsi.asp (https://www.investopedia.com/terms/r/rsi.asp)
3. Stochastic Oscillator: What It Is, How It Works, How To Calculate. (2021, June 25). Investopedia. https://www.investopedia.com/terms/s/stochasticoscillator.asp (https://www.investopedia.com/terms/s/stochasticoscillator.asp)
4. Philip Fisher (1958): Common Stocks and Uncommon Profits
5. Joel Greenblatt (2005): The Little Book That Beats the Market