Stock Market Direction Prediction

Data Analytics Project

Aleksandar Gradev and Aleksandar Milosavljevic

Contents

Project Overview

In this project, we develop an algorithmic trading strategy based on ML decisions designed to predict daily stock market direction with the goal of outperforming the market. The project combines the two types of statistical learning methods learned in the Data Analytics course with financial data analysis knowledge from our Finance specialization to simulate a trading strategy based on predicted stock movements.

Traditional investors often struggle to consistently outperform the market. By leveraging machine learning models, we aim to capture hidden patterns in historical data and design a systematic trading approach that is data-driven, adaptive and potentially more profitable than passive investment.

In our project we use the K-Means Clustering method to build a diversified trading portfolio. Feature engineering is used to generate lagged returns and momentum-based signals. After deriving our predictor variables, we apply three supervised machine learning models - logistic regression, decision tree and random forest to predict if the stock is going to close at higher price than today. The predictions of those models are used in combination with a precision-weighted investment function that dynamically allocates capital depending on the agreement and confidence of the models. With this information, we backtest the strategy by simulating daily trading and we compare the results of it to the S&P 500 on key performance metrics.

Through this project, we expect to demonstrate that a data-driven, model-based trading strategy can outperform passive investment in the market index over a multiyear period. Specifically, we anticipate that our predictive models will be able to capture meaningful patterns in the stock data, which would suggest that our models are not just overfitting noise but are learning useful information. We will check if the strategy handles different market conditions more adaptively, if it shows a superior Sharpe ratio than the benchmark, indicating that the returns we generate are not just high, but also adjusted for the risk we are taking and finally if trading each company in the portfolio is better than a passive investment in them.

Data Setup and Pre-processing

The data used by us in the project is from Kaggle.com and can be found here. The data set contains a comprehensive historical record of stock prices for one of the world's most famous publicly traded companies and it is updated on a daily basis. We thank Lahiru Nelgiriyewithana for making this data set public.

Environment Setup

We begin by setting a random seed for reproducibility, loading the required libraries and importing the data set.

```
set.seed(123)

library(quantmod)
library(ggcorrplot)
library(dplyr)
library(TTR)
library(listviewer)
library(rpart)
library(partykit)
library(ranger)
library(plotly)
library(tidyverse)

raw_data <- read.csv("World-Stock-Prices-Dataset.csv")
```

Exploratory Data Analysis

Companies in the Data Set

The companies in our data set are:

```
##  [1] "peloton"                   "amd"
##  [3] "adidas"                    "american express"
##  [5] "puma"                      "visa"
##  [7] "adobe"                     "unilever"
##  [9] "cisco"                     "jpmorgan chase & co"
## [11] "lvmh"                      "airbnb"
## [13] "marriott"                  "ubisoft"
## [15] "zoominfo"                  "toyota"
## [17] "hilton"                    "mcdonald's"
## [19] "the home depot"            "mastercard"
## [21] "johnson & johnson"         "uber"
## [23] "procter & gamble"          "coinbase"
## [25] "fedex"                     "3m"
## [27] "nordstrom"                 "philips"
## [29] "netflix"                   "the coca-cola company"
## [31] "foot locker"               "crocs"
## [33] "southwest airlines"        "shopify"
## [35] "amazon"                    "apple"
## [37] "nike"                      "target"
## [39] "google"                    "spotify"
## [41] "zoom video communications" "the walt disney company"
## [43] "roblox"                    "nintendo"
## [45] "delta air lines"           "microsoft"
## [47] "costco"                    "american eagle outfitters"
## [49] "hershey company"           "tesla"
## [51] "pinterest"                 "bmw group"
## [53] "chipotle"                  "porsche"
## [55] "logitech"                  "colgate palmolive"
## [57] "salesforce / slack"        "nvidia"
## [59] "starbucks"                 "honda"
## [61] "block"
```

We see that some of the company names in the data set include a variety of special characters such as spaces, slashes (/), apostrophes ('), ampersands (&) and others. These characters often appear due to the way company names are written in the real world. While these characters are meaningful for human readability, they can cause issues during data processing or analysis, such as inconsistent matching or parsing errors. To make our work smoother, we will change them in the following loop.

```
for (name in unique(raw_data$Brand_Name)) {
  characters <- strsplit(name, "")[[1]]
  new_characters <- c()

  for (character in characters) {
    new_character <- as.character("a")

    bad_characters <- c("-","&","/", " ")
```

```
    if (character %in% bad_characters) {
      new_character <- "_"
    } else if (character == "'") {
      new_character <- ""
    } else {
      new_character <- character
    }

    new_characters <- c(new_characters, new_character)
  }
  new_name <- paste0(new_characters, collapse = "")

  raw_data$Brand_Name[raw_data$Brand_Name == name] <- new_name
}

print(unique(raw_data$Brand_Name))
```

```
##  [1] "peloton"                   "amd"
##  [3] "adidas"                    "american_express"
##  [5] "puma"                      "visa"
##  [7] "adobe"                     "unilever"
##  [9] "cisco"                     "jpmorgan_chase___co"
## [11] "lvmh"                      "airbnb"
## [13] "marriott"                  "ubisoft"
## [15] "zoominfo"                  "toyota"
## [17] "hilton"                    "mcdonalds"
## [19] "the_home_depot"            "mastercard"
## [21] "johnson___johnson"         "uber"
## [23] "procter___gamble"          "coinbase"
## [25] "fedex"                     "3m"
## [27] "nordstrom"                 "philips"
## [29] "netflix"                   "the_coca_cola_company"
## [31] "foot_locker"               "crocs"
## [33] "southwest_airlines"        "shopify"
## [35] "amazon"                    "apple"
## [37] "nike"                      "target"
## [39] "google"                    "spotify"
## [41] "zoom_video_communications" "the_walt_disney_company"
## [43] "roblox"                    "nintendo"
## [45] "delta_air_lines"           "microsoft"
## [47] "costco"                    "american_eagle_outfitters"
## [49] "hershey_company"           "tesla"
## [51] "pinterest"                 "bmw_group"
## [53] "chipotle"                  "porsche"
## [55] "logitech"                  "colgate_palmolive"
## [57] "salesforce___slack"        "nvidia"
## [59] "starbucks"                 "honda"
## [61] "block"
```

Structure of the Data Set

Now, let's inspect the structure of the data set.

```r
# Inspect the structure of the data
str(raw_data)
```

```
## 'data.frame':    306715 obs. of  13 variables:
##  $ Date         : chr  "2025-04-21 00:00:00-04:00" "2025-04-21 00:00:00-04:00" "2025-04-21 00:00:00-0
##  $ Open         : num  5.28 86.02 115.93 248.85 22.2 ...
##  $ High         : num  5.37 86.14 115.93 250.26 22.2 ...
##  $ Low          : num  5.17 83.75 110.84 239.27 22.2 ...
##  $ Close        : num  5.32 85.56 112.44 242.51 22.2 ...
##  $ Volume       : num  8299600 33774700 56200 4302500 100 ...
##  $ Dividends    : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Stock.Splits : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Brand_Name   : chr  "peloton" "amd" "adidas" "american_express" ...
##  $ Ticker       : chr  "PTON" "AMD" "ADDYY" "AXP" ...
##  $ Industry_Tag : chr  "fitness" "technology" "apparel" "finance" ...
##  $ Country      : chr  "usa" "usa" "germany" "usa" ...
##  $ Capital.Gains: num  NA NA NA NA NA NA NA NA NA NA ...
```

The data set contains 306,715 observations and 13 variables, including stock price data (*Open*, *High*, *Low*, *Close*), *trading volume*, *dividend* and split info, as well as company metadata such as *brand name*, *ticker*, *industry* and *country*. The *Date* variable is currently stored as a character and requires conversion to Date format. Moreover, the variable *Capital.Gains* contains mostly missing values, so we will inspect it manually.

```r
# change the class of the date variable
raw_data$Date <- as.Date(raw_data$Date, format="%Y-%m-%d")

# Check how many non-NA values are there
length(is.na(raw_data$Capital.Gains)) - sum(is.na(raw_data$Capital.Gains))
```

```
## [1] 2
```

```r
# Check the non-NA values
unique(raw_data$Capital.Gains)
```

```
## [1] NA  0
```

```r
# Remove the variable
raw_data$Capital.Gains <- NULL
```

We see that there are only two non-NA entries. Upon inspection, those two values are both zero, indicating that the variable carries no meaningful information for the analysis. Therefore, it is safe and appropriate to remove it from the data set to simplify further processing.

Data Cleaning

Split Data by Company

As we know from the data set information there are multiple observations per date, so we need to split the data set by company for further analysis. We decided to work with a list, where each element corresponds to the information for one company. For each company we extract its metadata (name, ticker, industry,

and country) into a separate *Company_Info* data frame and store its time series data (e.g. prices, volume, dividends) in a *Historical_Data* data frame. These are combined into a nested list structure for more efficient access.

```
# Split the data by companies
split_data <- split(raw_data, raw_data$Brand_Name)

# Create an empty list to store the data
transformed_data <- list()

# Fill the list with the data for each company
for (company in names(split_data)) {

  # Extract the original dataframe
  df <- split_data[[company]]

  # Create the company info dataframe
  company_info <- data.frame(
    Brand_Name = unique(df$Brand_Name),
    Ticker = unique(df$Ticker),
    Industry_Tag = unique(df$Industry_Tag),
    Country = unique(df$Country)
  )

  # Remove company info columns from the historical data
  historical_data <- df[, !(names(df) %in% c("Brand_Name", "Ticker", "Industry_Tag","Country"))]

  # Nest the company info and historical data into a list
  transformed_data[[company]] <- list(
    Company_Info = company_info,
    Historical_Data = historical_data
  )
}
```

Remove Companies with Insufficient Data

We know that some companies in the data set were listed after the year 2000 and therefore do not have complete historical data for the full analysis period. To ensure consistency across time, we will continue working only with companies that were traded during the whole period. To identify which ones to leave and which ones to remove, we create a binary flag (*binary_2000*) indicating whether the company has data for the whole period.

```
# Create the first_day_trade data frame
first_day_trade <- data.frame(
  first_day = rep(NA, length(transformed_data)),
  name = rep(NA, length(transformed_data))
)

# Extract first trading day and company name
for (i in 1:length(transformed_data)) {
  # Get the last row date (earliest trading date)
  first_day_trade$first_day[i] <- min(transformed_data[[i]][["Historical_Data"]]$Date, na.rm = TRUE)
  first_day_trade$name[i] <- transformed_data[[i]][["Company_Info"]]$Brand_Name
```

```
}

# Convert the date column into date format
first_day_trade$first_day <- as.Date(first_day_trade$first_day)

# Create the binary flag variable
first_day_trade$binary_2000 <- ifelse(first_day_trade$first_day <= as.Date("2000-01-03"), 1, 0)

# Print the count of companies that meet the criteria
cat(sum(first_day_trade$binary_2000), "companies were traded for the whole period")
```

```
## 34 companies were traded for the whole period
```

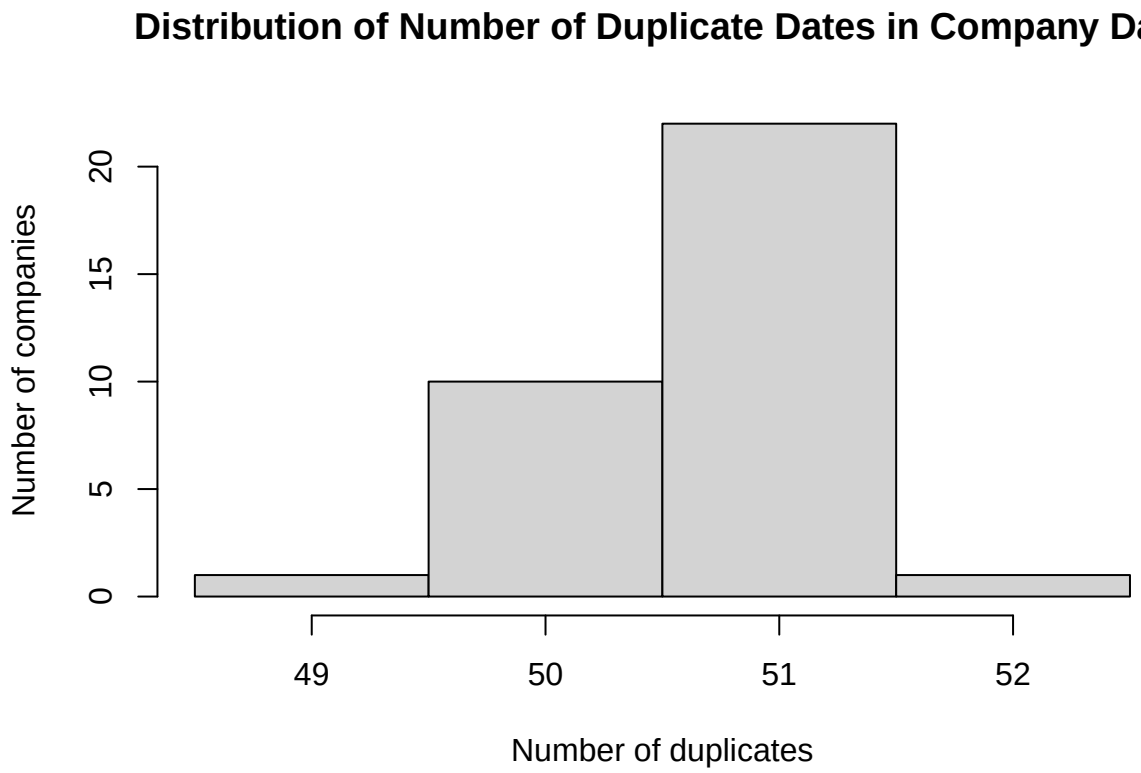
```
# Subset to keep only companies that have data from the beginning
first_day_trade <- subset(first_day_trade, binary_2000 == 1)

# Extract the names of companies to remove
companies_to_remove <- setdiff(names(transformed_data), first_day_trade$name)

# Remove companies that do not meet the criteria
final_data <- transformed_data[!names(transformed_data) %in% companies_to_remove]

# Reorder data in chronological order
for (i in 1:length(final_data)){
  final_data[[i]][["Historical_Data"]] <- final_data[[i]][["Historical_Data"]][order(final_data[[i]][["
}
```

Check for Duplicate Trading Days

To ensure data quality, we check for duplicate trading days within each company's historical data. For each company, we count how many duplicate *Date* entries exist and store the results in a summary table. Duplicate rows can arise from data entry errors or overlapping records and can distort time series analysis.

```
# Create data frame to store number of duplicates for each company
duplicates <- data.frame(
  company = rep(NA, length(final_data)),
  duplicates = rep(NA, length(final_data)),
  country = rep(NA, length(final_data))
)

# Count the duplicates for each company
for (i in 1:length(final_data)) {

  duplicates$company[i] <- final_data[[i]][["Company_Info"]]$Brand_Name[1]
  duplicates$duplicates[i] <- sum(duplicated(final_data[[i]][["Historical_Data"]]$Date))
  duplicates$country[i] <- final_data[[i]][["Company_Info"]]$Country[1]
}

# Visualize the number of duplicates
hist(duplicates$duplicates,
     breaks = seq(48.5, 52.5, 1),
     main = "Distribution of Number of Duplicate Dates in Company Data",
```

```
     xlab = "Number of duplicates",
     ylab = "Number of companies",
     col = "lightgray",
     border = "black")
```



The histogram shows how many companies have a specific number of duplicate trading dates in their historical data. Since companies come from different countries and stock exchanges, these inconsistencies likely result from variations in local trading calendars (e.g., holidays or weekends) or data collection errors. We remove all duplicated dates, keeping only the first occurrence to maintain a clean and consistent data set.

```
# Remove duplicates
for (i in 1:length(final_data)) {

  final_data[[i]][["Historical_Data"]] <- final_data[[i]][["Historical_Data"]][!duplicated(final_data[[
}
```

Price Data Anomalies Inspection

To detect unusually large price movements, we begin by calculating daily returns - defined as the percentage change in the closing price from one trading day to the next. These returns serve as the basis for identifying price anomalies across all companies.

```
daily_returns_fun <- function(df) {
  df$Return <- (df$Close / dplyr::lag(df$Close, 1)) - 1
  return(df)
```

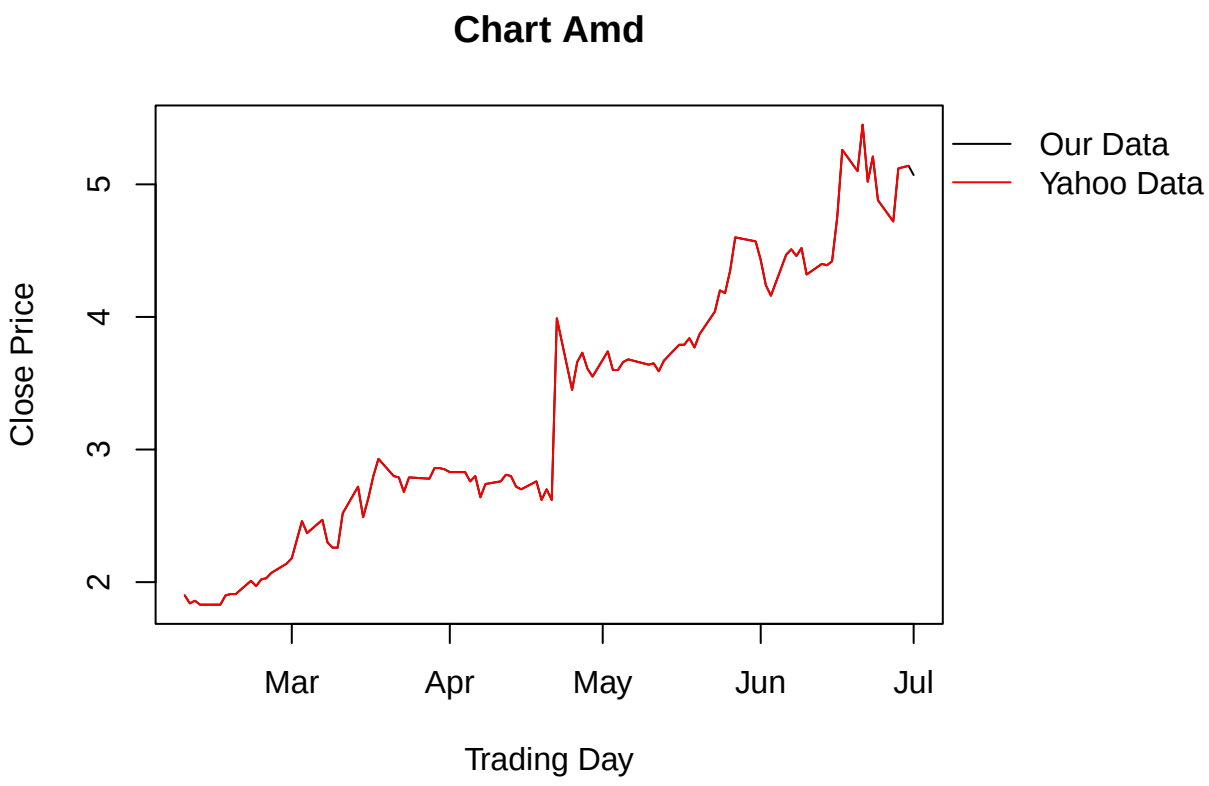
```
}

# Apply the function to each company
for (i in 1:length(final_data)) {
  final_data[[i]][["Historical_Data"]] <- daily_returns_fun(final_data[[i]][["Historical_Data"]])
}
```

We define a price anomaly as any single-day price change greater than $+50\%$ or -50%, which could be considered unrealistic for large companies. For each such anomaly, the algorithm extracts a window of 100 trading days (50 before and 50 after the anomaly) and plot it. To validate whether the anomaly is a real market event or a data error, the official stock price data from Yahoo Finance is downloaded and ploted over our data's graph, comparing the two sources visually.

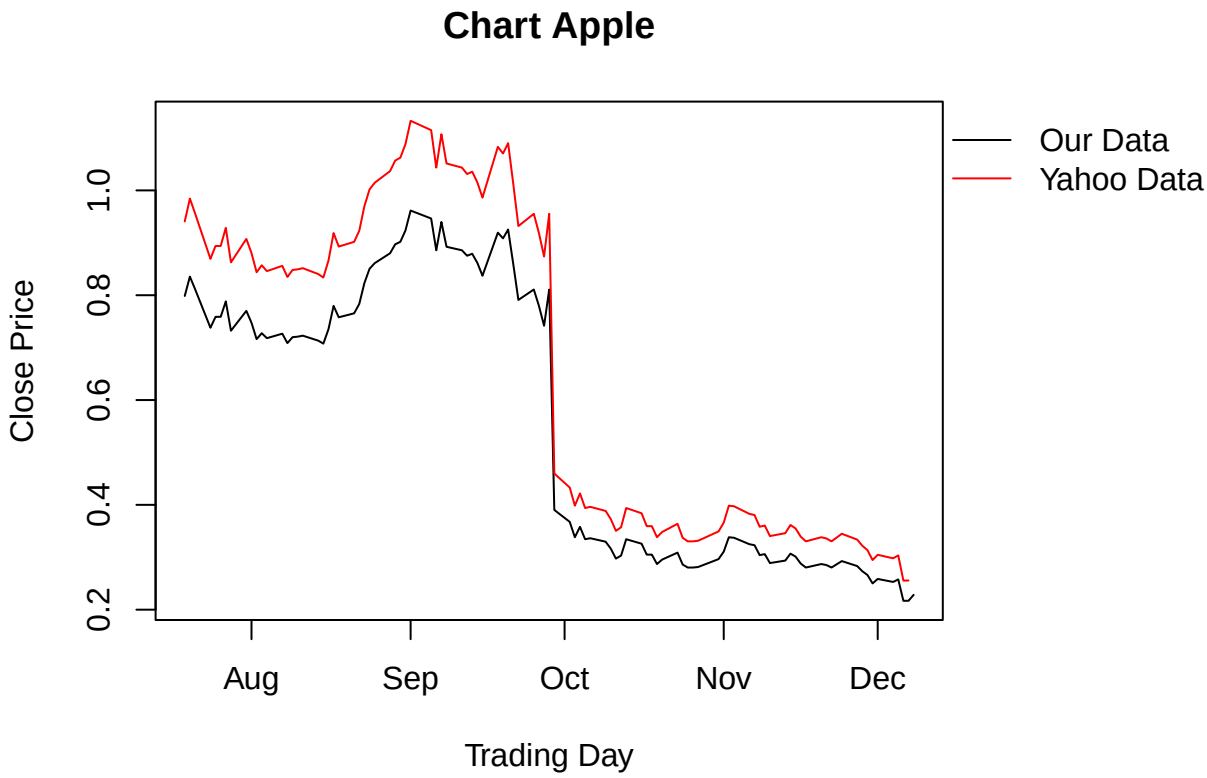
```
# Iterate over each company to find anomalies
for (company in names(final_data)) {

  hist_data <- final_data[[company]][["Historical_Data"]]
  n <- nrow(hist_data)

  # Iterate over each day
  for (i in 1:(n - 1)) {
    # Check if it matches the criteria for price anomaly
    if (abs(hist_data$Return[i + 1]) > 0.5) {
      # Print the day of the price anomaly
      cat(" ========================================\n",
      tools::toTitleCase(company),
      "\n Anomaly on", format(hist_data$Date[i], "%Y-%m-%d"), ":\n")

      # Plot our data vs yahoo data to check if the problem is with the data
      start_idx <- max(1, i - 50)
      end_idx <- min(n, i + 50)
      start_date <-   final_data[[company]][["Historical_Data"]]$Date[start_idx]
      end_date <- final_data[[company]][["Historical_Data"]]$Date[end_idx]

      ticker <- final_data[[company]][["Company_Info"]]$Ticker

      company_yahoo <- getSymbols(ticker, auto.assign = FALSE,
                                  src = "yahoo", from = start_date,
                                  to = end_date)
      company_yahoo_sorted <- company_yahoo[order(index(company_yahoo)), ]

      par(mar = c(5, 4, 4, 8), xpd = TRUE)

      # Get values for ylim to fix the limits of the y-axis
      y1 <- hist_data$Close[start_idx:end_idx]
      y2 <- as.numeric(company_yahoo_sorted[, 4])
      combined_range <- range(c(y1, y2), na.rm = TRUE)

      # Plot our data
      plot(hist_data$Date[start_idx:end_idx],
           y1,
           main = paste("Chart", tools::toTitleCase(company)),
           xlab = "Trading Day", ylab = "Close Price",
```

```
            type = "l", col = 'black',
            ylim = combined_range)

      # Add yahoo data as a red line
      lines(x = index(company_yahoo_sorted),
            y = y2,
            col = "red", type = "l")

      legend("topright", inset = c(-0.35, 0), legend = c("Our Data", "Yahoo Data"),
       col = c("black", "red"), lty = 1, bty = "n")

    }
  }
}
```

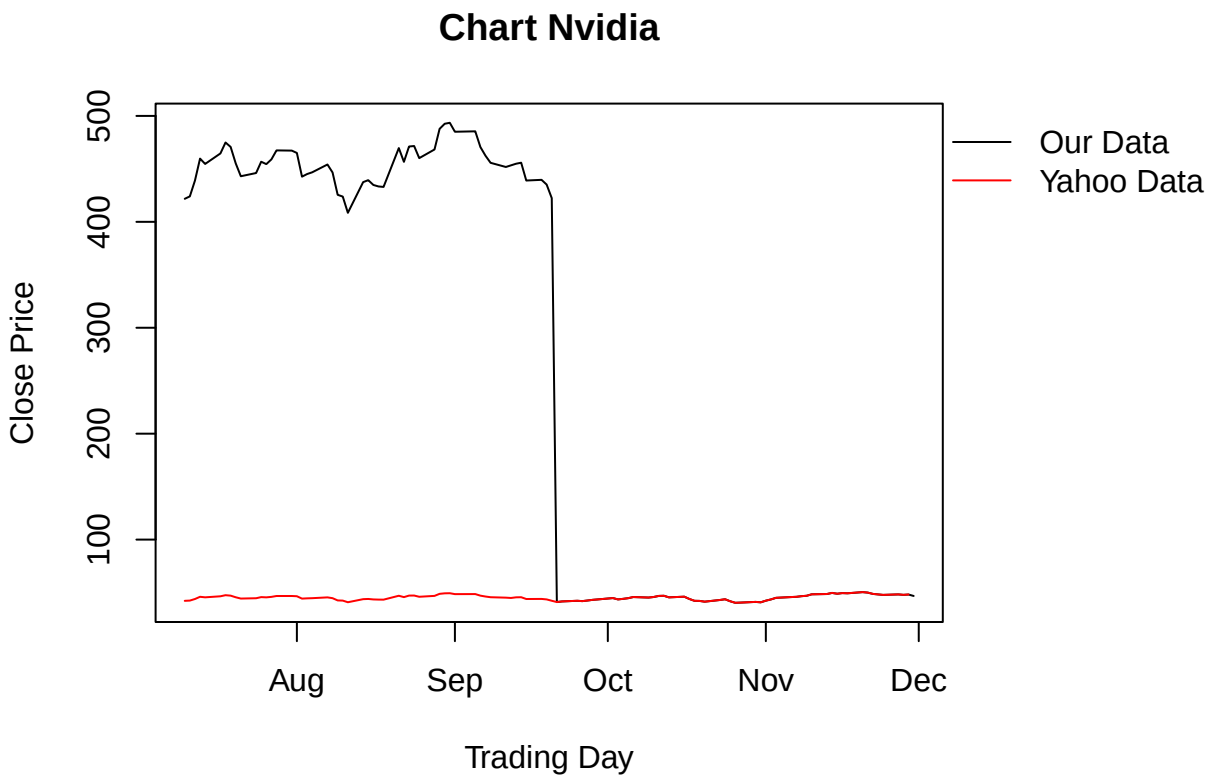
```
##  ====================================
##  Amd
##  Anomaly on 2016-04-21 :
```




```
##  ====================================
##  Apple
##  Anomaly on 2000-09-28 :
```




```
##  ====================================
##  Nvidia
##  Anomaly on 2023-09-20 :
```

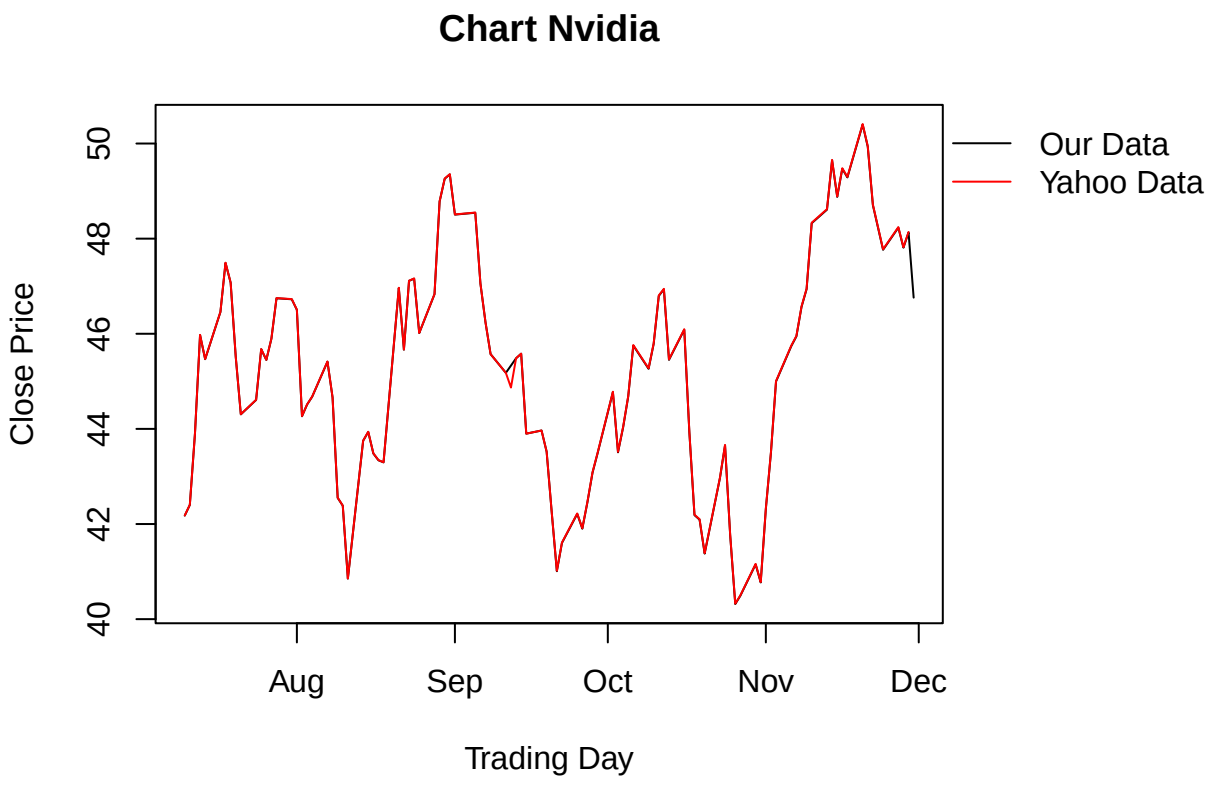


As shown, the loop detected three price anomalies across three different companies:

- AMD - the Yahoo Finance data aligns perfectly with our data set, indicating that the spike in share price is genuine and not due to a data issue.
- Apple - our data closely follows the trend of the Yahoo data but is slightly lower. Despite this difference, both sources reflect a significant drop in price, confirming the anomaly as valid. For consistency and practicality, we will proceed with our own data set.
- Nvidia - there is a noticeable discrepancy between our data and Yahoo's. After a manual review, we identified that our data source did not adjust for the 1:10 stock split. Correcting for this by dividing pre-split prices by 10 resolves the anomaly.

```
# Manually fix our data for nvidia
final_data[["nvidia"]][["Historical_Data"]]$Close[final_data[["nvidia"]][["Historical_Data"]]$Date <= "

# Plot results after fixing
start_idx <- 5916
end_idx <- 6016
start_date <-   final_data[["nvidia"]][["Historical_Data"]]$Date[start_idx]
end_date <- final_data[["nvidia"]][["Historical_Data"]]$Date[end_idx]

nvidia_yahoo <- getSymbols("NVDA", auto.assign = FALSE,
                                  src = "yahoo", from = start_date,
                                  to = end_date)
nvidia_yahoo_sorted <- company_yahoo[order(index(nvidia_yahoo)), ]
```

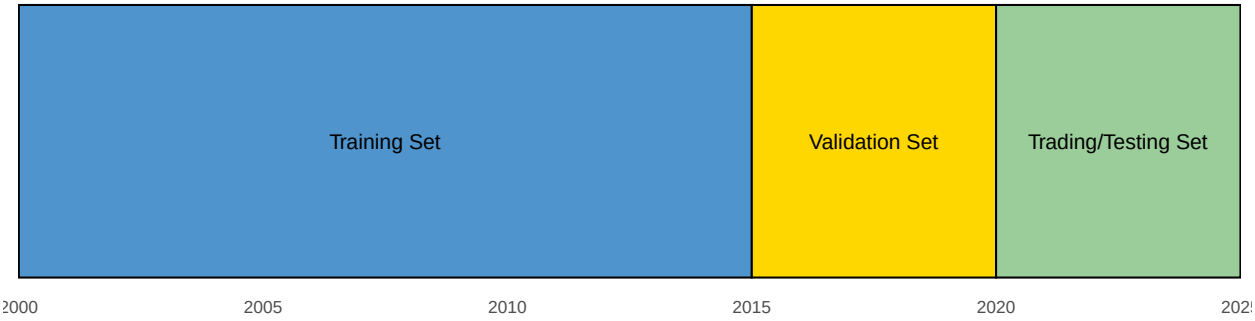
```r
par(mar = c(5, 4, 4, 8), xpd = TRUE)

# Get values for ylim to fix the limits of the y-axis
y1 <- final_data[["nvidia"]][["Historical_Data"]]$Close[start_idx:end_idx]
y2 <- as.numeric(nvidia_yahoo_sorted[, 4])
combined_range <- range(c(y1, y2), na.rm = TRUE)

# Plot our data
plot(final_data[["nvidia"]][["Historical_Data"]]$Date[start_idx:end_idx],
           y1,
           main = "Chart Nvidia",
           xlab = "Trading Day", ylab = "Close Price",
           type = "l", col = 'black',
           ylim = combined_range)

# Add yahoo data as a red line
lines(x = index(nvidia_yahoo_sorted),
            y = y2,
            col = "red", type = "l")

legend("topright", inset = c(-0.35, 0), legend = c("Our Data", "Yahoo Data"),
       col = c("black", "red"), lty = 1, bty = "n")
```



The corrected Nvidia data now aligns almost perfectly with the Yahoo Finance data, so we will proceed with our adjusted data set.

Data Splitting

This code splits the historical stock data for each selected portfolio company into three time-based subsets: training, validation, and trading/testing sets. Specifically, it extracts data from 2000 to the end of 2015 as the training set, data from 2016 to the end of 2020 as the validation set and data from 2021 to present day as the trading/testing set. An additional clustering data set is created, combining the training and the validation sets.



The idea behind using the **Validation Set approach** is primarily to avoid bias. Since we don't have the ability to continuously gather new real-time data, we simulate the scenario as if **today is the 1st of January 2021**, and we want to run a trading experiment starting from that point.

The **Training set** is used to fit our models, while the validation set serves to evaluate their performance and select the ones that achieve the highest precision. Once we understand how well the three models (Logit, Decision Tree and Random Forest) perform by testing them on the **Validation set**, we assign "reliability weights" to each of their predictions. These "weights" are then used as a risk management tool to determine the degree of confidence we place in each model's signal. More about this can be found in **Investment function** part and the appendix.

Armed with this information, we simulate a day-by-day trading strategy on the **Trading/Testing set**, predicting whether the price of each stock in our portfolio will rise, and placing trades based on these predictions.

The combined **Clustering set** is essential for identifying the best-performing companies to include in our portfolio. To make these selections, we rely on the well-known risk/reward ratio, which is explained in more detail in the appendix.

```
# Split data into Training set, Validation set and Testing/Trading set
for (company in names(final_data)) {

  # Store the df as variable for shorter code
  fin_data <- final_data[[company]][["Historical_Data"]]

  # Create new data frames as elements in each company's list
  final_data[[company]][["Train_Data"]] <-
    fin_data[fin_data$Date < as.Date("2015-12-31"),]

  final_data[[company]][["Validation_Data"]] <-
    fin_data[fin_data$Date > as.Date("2015-12-31") &
               fin_data$Date < as.Date("2020-12-31"),]

  final_data[[company]][["Trade_Data"]] <-
    fin_data[fin_data$Date > as.Date("2020-12-31"),]
```

```
  final_data[[company]][["Clustering_Data"]] <-
    fin_data[fin_data$Date < as.Date("2020-12-31"),]

  # Order the data
  final_data[[company]][["Train_Data"]] <-
    final_data[[company]][["Train_Data"]][order(final_data[[company]][["Train_Data"]]$Date), ]

  final_data[[company]][["Validation_Data"]] <-
    final_data[[company]][["Validation_Data"]][order(final_data[[company]][["Validation_Data"]]$Date), ]

  final_data[[company]][["Trade_Data"]] <-
    final_data[[company]][["Trade_Data"]][order(final_data[[company]][["Trade_Data"]]$Date), ]

  final_data[[company]][["Clustering_Data"]] <-
    final_data[[company]][["Clustering_Data"]][order(final_data[[company]][["Clustering_Data"]]$Date), ]
}
```

Primary Feature Engineering

In this section, we focus on deriving key financial metrics from each company's historical stock data that we will need for the portfolio selection. Specifically, we calculate the **annualized historical return** and **annualized volatility** for every stock in the data set for the 2000-2020 period and store them into each company's **return_volatility_df** data frame. **Annualized return** is the daily average rate of return scaled to yearly basis, while **annualized volatility** is the standard deviation of returns scaled to a yearly basis, measuring the investment's risk or variability. These two measures (return and risk) form the foundation of quantitative portfolio analysis. In part 5 we will visualize the relationship between them to understand the risk to reward ratio for each stock. The results will serve as inputs for the stocks clustering and portfolio construction.

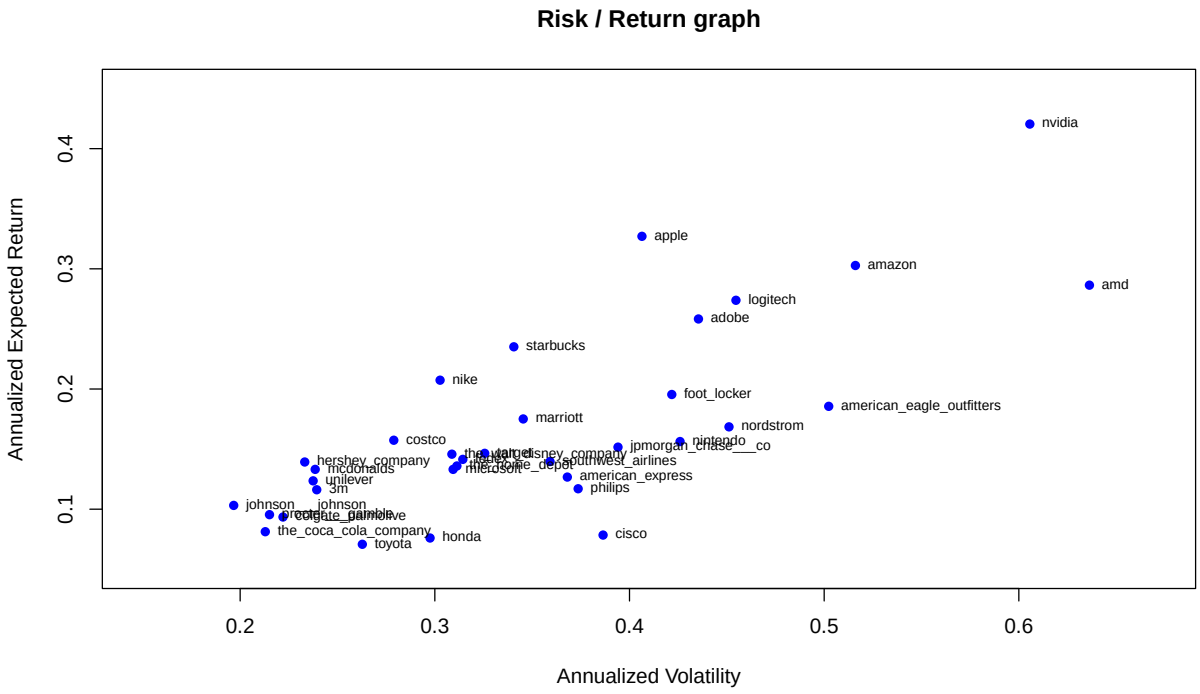
```
# Define function to calculate annualized exp return
annualized_hist_return_fun <- function(df) {
  mean_daily_return <- mean(df$Return, na.rm = T)

  annual_hist_return <- 252*mean_daily_return
  return(annual_hist_return)
}

# Define function to calculate volatility (standard deviation)
annualized_volatility_fun <- function(df) {
  daily_volatility <- sd(df$Return, na.rm = TRUE)
  annualized_volatility <- daily_volatility * sqrt(252)
  return(annualized_volatility)
}

# Run both functions for each company (for each time period/data set)
for (company in names(final_data)) {
  data_sets <- c("Train_Data", "Validation_Data",
                 "Trade_Data", "Clustering_Data")

  return_volatility_df <- data.frame(
    Ann_Exp_Return = rep(NA, length(data_sets)),
    Ann_Volatility = rep(NA, length(data_sets))
```

```
  )

  rownames(return_volatility_df) <- data_sets

  final_data[[company]][["return_volatility_df"]] <- return_volatility_df

  for (i in 1:length(data_sets)) {
    data_set <- data_sets[i]
    data <- final_data[[company]][[data_set]]

    annual_hist_return <- annualized_hist_return_fun(data)
    annual_volatility <- annualized_volatility_fun(data)

  final_data[[company]][["return_volatility_df"]]$Ann_Volatility[i] <- annual_volatility
  final_data[[company]][["return_volatility_df"]]$Ann_Exp_Return[i] <- annual_hist_return
  }
}
```

Portfolio Selection Using K-Means Clustering

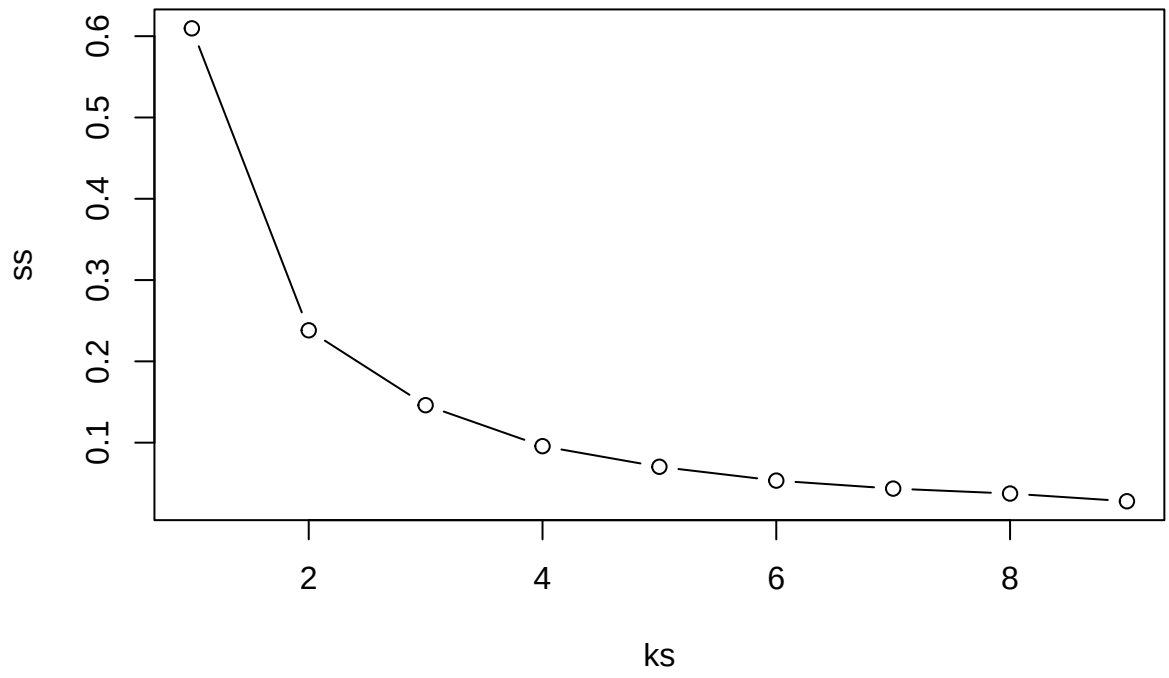
After calculating the **annualized return** and **annualized volatility** of each stock on the Clustering data set, we are plotting those two variables against each other. Together, they form the well known **risk/return graph**.

```
# Plot the results on a risk/reward graph
plot_data <- data.frame(
  Brand_Name = rep(NA, length(final_data)),
  Ann_Exp_Return = rep(NA, length(final_data)),
  Ann_Volatility = rep(NA, length(final_data))
)

for (i in 1:length(final_data)) {
  company <- final_data[[i]][["Company_Info"]]$Brand_Name

  plot_data$Brand_Name[i] <- company

  plot_data$Ann_Exp_Return[i] <- final_data[[i]][["return_volatility_df"]]$Ann_Exp_Return[4]

  plot_data$Ann_Volatility[i] <- final_data[[i]][["return_volatility_df"]]$Ann_Volatility[4]
}

plot(plot_data$Ann_Volatility, plot_data$Ann_Exp_Return,
     xlab = "Annualized Volatility",
     ylab = "Annualized Expected Return",
     main = "Risk / Return graph",
     pch = 16, col = "blue",
     xlim = c(0.15, 0.67),
     ylim = c(0.05, 0.45)
)

text(plot_data$Ann_Volatility, plot_data$Ann_Exp_Return,
     labels = plot_data$Brand_Name, pos = 4, cex = 0.7, col = "black")
```



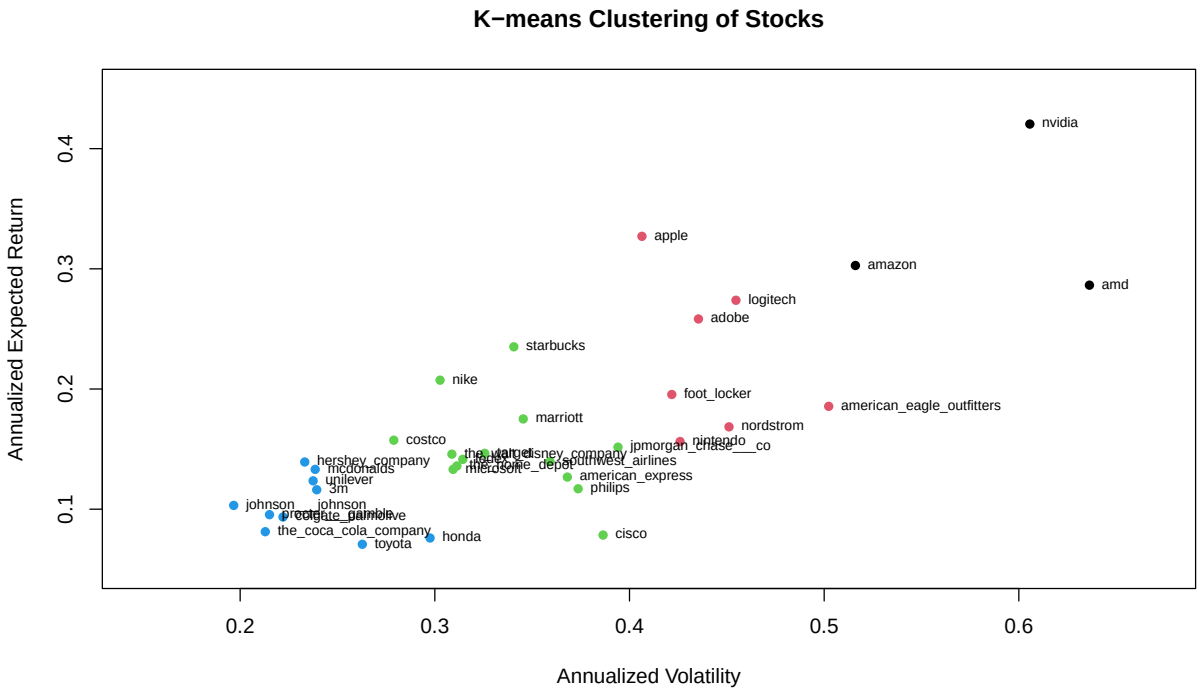
We perform k-means clustering on the stocks based on their risk-return profiles to help select a diversified portfolio. We start by preparing a data set containing each stock's annualized expected return and volatility. In order not to be biased, we use the data from the **Clustering set**. After clustering, the stocks are plotted by risk (volatility) and return, color-coded by cluster, with the stock names labeled and cluster centers marked. We then calculate a return-to-volatility ratio for each stock to assess performance efficiency.

```
# Create a data frame storing only the return and volatility
numeric_plot_data <- plot_data[, c("Ann_Exp_Return", "Ann_Volatility")]

# Determine optimal level of k for k-means clustering
kmcs <- lapply(ks <- 1:9, \(k) kmeans(numeric_plot_data, nstart = 10,
                                      centers = k))

# Store the total withing-cluster sum of squares for each cluster
ss <- unlist(lapply(kmcs, \(obj) obj$tot.withinss))

# Plot the elbow method plot
plot(ks, ss, type = "b")
```



By using the elbow method, we determine that 4 clusters are optimal. Those 4 clusters represent 4 types of stocks - Ultra High Risk but Ultra High Return, High Risk but High Return, Medium Risk with Medium Return and Low Risk with Low Return.

```
# Perform k-means clustering
stock_kmc <- kmeans(numeric_plot_data, centers = 4, nstart = 10)

# Plot the clustered data
plot(plot_data$Ann_Volatility, plot_data$Ann_Exp_Return,
     col = stock_kmc$cluster, main = "K-means Clustering of Stocks",
     xlab = "Annualized Volatility", ylab = "Annualized Expected Return",
     pch = 16,
     xlim = c(0.15, 0.67),
     ylim = c(0.05, 0.45))

# Add company names
text(plot_data$Ann_Volatility,
     plot_data$Ann_Exp_Return,
     labels = plot_data$Brand_Name, pos = 4, cex = 0.7)
```



We will now set up our diversified portfolio. It will consist of 4 stocks — one from each group. The idea behind this is to capture different risk-return profiles across clusters, reduce unsystematic risk and ensure that the portfolio is not overly exposed to a single market behavior or sector dynamic.

```
# Calculate exp return / volatility ratio
plot_data$ratio <- plot_data$Ann_Exp_Return / plot_data$Ann_Volatility

# Show which group each stock got into
plot_data$cluster <- stock_kmc$cluster

plot_data$Risk_Profile <- ifelse(plot_data$cluster == 1, "Ultra High",
                                 ifelse(plot_data$cluster == 2, "High",
                                        ifelse(plot_data$cluster == 3, "Medium",
                                               "Low")))

plot_data <- plot_data[order(-plot_data$ratio), ]

plot_split_data <- split(plot_data, plot_data$Risk_Profile)

# Find the row with the maximum ratio for each cluster
plot_split_result <- do.call(rbind, lapply(plot_split_data,
                                           function(df) df[which.max(df$ratio), ]))

# Display the result
print(plot_split_result, row.names = FALSE)
```

```
##       Brand_Name Ann_Exp_Return Ann_Volatility     ratio cluster Risk_Profile
##            apple      0.3270466      0.4063978 0.8047449       2         High
##  hershey_company      0.1392652      0.2331743 0.5972579       4          Low
##        starbucks      0.2351219      0.3405869 0.6903432       3       Medium
```

```
##           nvidia      0.4204449      0.6056456 0.6942094       1   Ultra High
```

```r
# Store the names of the stocks for the portfolio
portfolio_stock_names <- unique(plot_split_result$Brand_Name)

# Create a list storing data only for stocks in the portfolio
portfolio_data <- final_data[names(final_data) %in% portfolio_stock_names]
```

The best-performing stock from each cluster (based on the highest ratio) is selected to form a diversified portfolio. We filter the data set to retain only those selected stocks, which are Apple, The Hershey Company, Starbucks and Nvidia. We create a new list called **portfolio_data**, that is storing only the necessary information for our 4 portfolio stocks.

Secondary Feature Engineering

This section focuses on creating the predictor variables needed for our classification models. All of them are derived from the existing price data. The idea is to build features that capture momentum, trend signals and lagged information to help predict the direction of stock price movement (up or not up). To avoid look-ahead bias, all features are based on past and lagged.

We begin by ensuring that the historical data for each stock is properly sorted in chronological order. This is essential because all lagging operations (like lagged returns or SMAs) depend on the correct temporal sequence of observations.

```r
# Reorder data in chronological order
for (i in 1:length(portfolio_data)) {
  portfolio_data[[i]][["Historical_Data"]] <- portfolio_data[[i]][[
    "Historical_Data"]][order(portfolio_data[[i]][["Historical_Data"]]$Date),]
}
```

For each stock, we create a new data frame called Predictor_Data that contains only the relevant variables for modeling: the date, the closing price and the daily return. This isolates the modeling features from the rest of the historical data and ensures consistency across all companies.

```r
# Create a new data frame storing only the inf  ormation needed for training
# and testing; Store it as a element in each company element

# Copy the Date, Close price and Return from the Historical_Data dataframe

for (i in 1:length(portfolio_data)) {
  company <- names(portfolio_data)[i]
  portfolio_data[[company]][["Predictor_Data"]] <- data.frame(
    Date = portfolio_data[[company]][["Historical_Data"]]$Date,
    Close = portfolio_data[[company]][["Historical_Data"]]$Close,
    Return = portfolio_data[[company]][["Historical_Data"]]$Return
  )
}
```

We define the classification target for each observation as a binary variable:

- 1 if the return for the day is positive (stock went up),

- 0 if the return is zero or negative (stock stayed the same or dropped).

This will be the dependent variable our models aim to predict.

```
# Create a binary variable target for each company
# if the stock went up is encoded as 1, else 0
for (i in 1:length(portfolio_data)) {
  company <- names(portfolio_data)[i]
  portfolio_data[[company]][["Predictor_Data"]]$Target <- ifelse(portfolio_data[[
    company]][["Predictor_Data"]]$Return > 0, 1, 0)
}
```

We compute several Simple Moving Averages (SMAs) over different time horizons (e.g., 4-day, 6-day, 135-day). These SMAs capture short-term, medium-term and long-term trends. However, instead of using them directly, we lag them by one day to ensure they do not leak future information into the model. Only lagged SMAs are stored in Predictor_Data. More information on what a Simple Moving Average is can be found in the appendix.

We also generate a binary indicator for whether a dividend was paid on a given day (1 if yes, 0 if no). Dividends can influence price movements, so this information is potentially valuable for prediction. Like the SMAs, this indicator is lagged by one day before being used in the model.

```
# Calculate SMAs for each stock
periods <- c(4,6,8,65,80,135)
sma_column_names <- c("sma_4","sma_6","sma_8","sma_65","sma_80","sma_135")
lag_sma_column_names <- c("sma_4_lag","sma_6_lag","sma_8_lag","sma_65_lag",
                          "sma_80_lag","sma_135_lag")

for (company in names(portfolio_data)) {

  for (j in 1:length(periods)) {
    sma_column_name <- sma_column_names[j]
    lag_sma_column_name <- lag_sma_column_names[j]

    # Calculate the SMA
    portfolio_data[[company]][["Historical_Data"]][[sma_column_name]] <-
      TTR::SMA(portfolio_data[[company]][["Historical_Data"]]$Close, n = periods[j])

    # Lag the calculated SMA
    portfolio_data[[company]][["Predictor_Data"]][[lag_sma_column_name]] <-
      dplyr::lag(portfolio_data[[company]][["Historical_Data"]][[sma_column_name]], 1)
  }
}

# Create a binary variable showing 1 if dividend was paid on this day
for (company in names(portfolio_data)) {
  portfolio_data[[company]][["Historical_Data"]]$Dividend_Indicator <-
    ifelse(portfolio_data[[company]][["Historical_Data"]]$Dividends > 0, 1, 0)
}

# Lag the dividend indicator in the predictor data
for (company in names(portfolio_data)) {
  portfolio_data[[company]][["Predictor_Data"]]$Dividend_Indicator_lag <-
    dplyr::lag(portfolio_data[[company]][["Historical_Data"]]$Dividend_Indicator,1)
}
```

We define particular buy signals based on SMA crossover strategies. A buy signal is generated when a short-term SMA crosses above a long-term SMA. A lagged binary variable will store 1 if a buy signal was generated after the short-term SMA crossed the long-term one. More information on the buy signals can be found in the appendix.

```
# Add the Buy Signals for each company
for (company in names(portfolio_data)) {

  # Buy signal if the 6-day SMA crossed over the 135-day SMA
  portfolio_data[[company]][["Predictor_Data"]]$Buy_Signal_6_135 <-
    ifelse(lag(portfolio_data[[company]][["Predictor_Data"]]$sma_6_lag) <
             lag(portfolio_data[[company]][["Predictor_Data"]]$sma_135_lag) &
             portfolio_data[[company]][["Predictor_Data"]]$sma_6_lag >=
             portfolio_data[[company]][["Predictor_Data"]]$sma_135_lag, 1, 0)

  # Buy signal if the 8-day SMA crossed over the 65-day SMA
  portfolio_data[[company]][["Predictor_Data"]]$Buy_Signal_8_65 <-
    ifelse(lag(portfolio_data[[company]][["Predictor_Data"]]$sma_8_lag) <
             lag(portfolio_data[[company]][["Predictor_Data"]]$sma_65_lag) &
             portfolio_data[[company]][["Predictor_Data"]]$sma_8_lag >=
             portfolio_data[[company]][["Predictor_Data"]]$sma_65_lag, 1, 0)

  # Buy signal if the 4-day SMA crossed over the 80-day SMA
  portfolio_data[[company]][["Predictor_Data"]]$Buy_Signal_4_80 <-
    ifelse(lag(portfolio_data[[company]][["Predictor_Data"]]$sma_4_lag) <
             lag(portfolio_data[[company]][["Predictor_Data"]]$sma_80_lag) &
             portfolio_data[[company]][["Predictor_Data"]]$sma_4_lag >=
             portfolio_data[[company]][["Predictor_Data"]]$sma_80_lag, 1, 0)
}
```

To capture short-term momentum, we add five lagged daily returns as predictors. These features help the model detect return autocorrelation patterns (e.g. mean reversion or momentum) that might exist in the data.

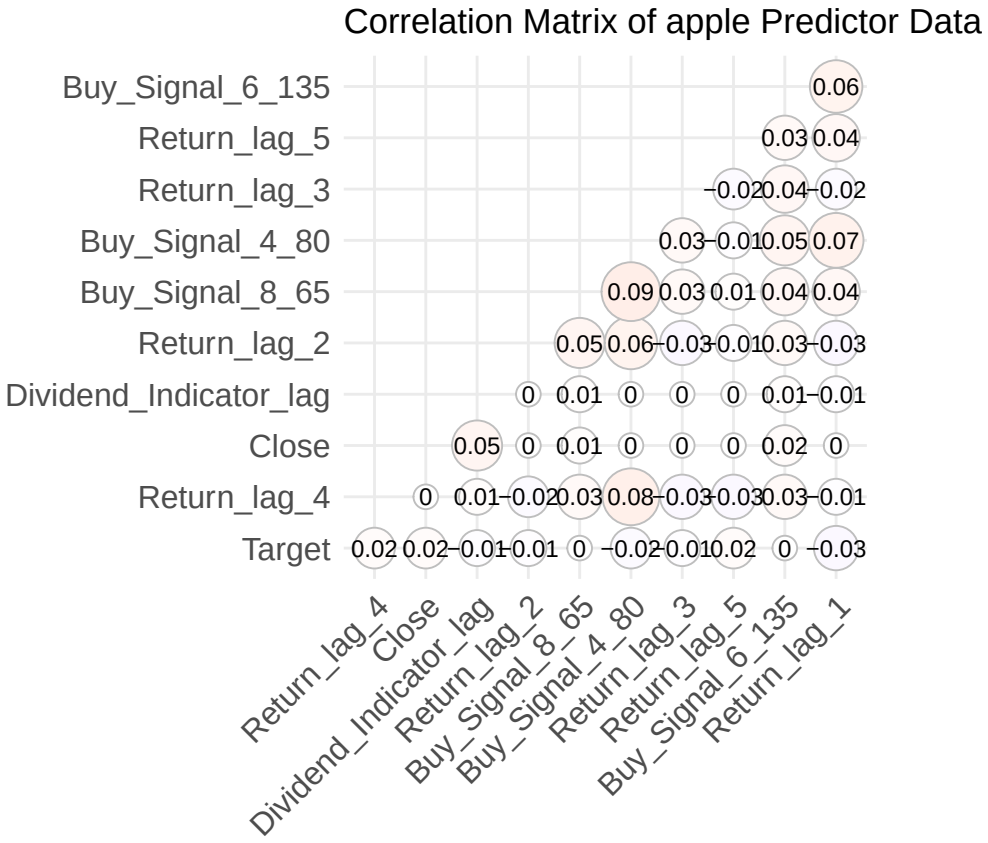
```
# Add lagged returns
for (company in names(portfolio_data)) {
  portfolio_data[[company]][["Predictor_Data"]]$Return_lag_1 <-
    dplyr::lag(portfolio_data[[company]][["Historical_Data"]]$Return, 1)

  portfolio_data[[company]][["Predictor_Data"]]$Return_lag_2 <-
    dplyr::lag(portfolio_data[[company]][["Historical_Data"]]$Return, 2)

  portfolio_data[[company]][["Predictor_Data"]]$Return_lag_3 <-
    dplyr::lag(portfolio_data[[company]][["Historical_Data"]]$Return, 3)

  portfolio_data[[company]][["Predictor_Data"]]$Return_lag_4 <-
    dplyr::lag(portfolio_data[[company]][["Historical_Data"]]$Return, 4)

  portfolio_data[[company]][["Predictor_Data"]]$Return_lag_5 <-
    dplyr::lag(portfolio_data[[company]][["Historical_Data"]]$Return, 5)
}
```

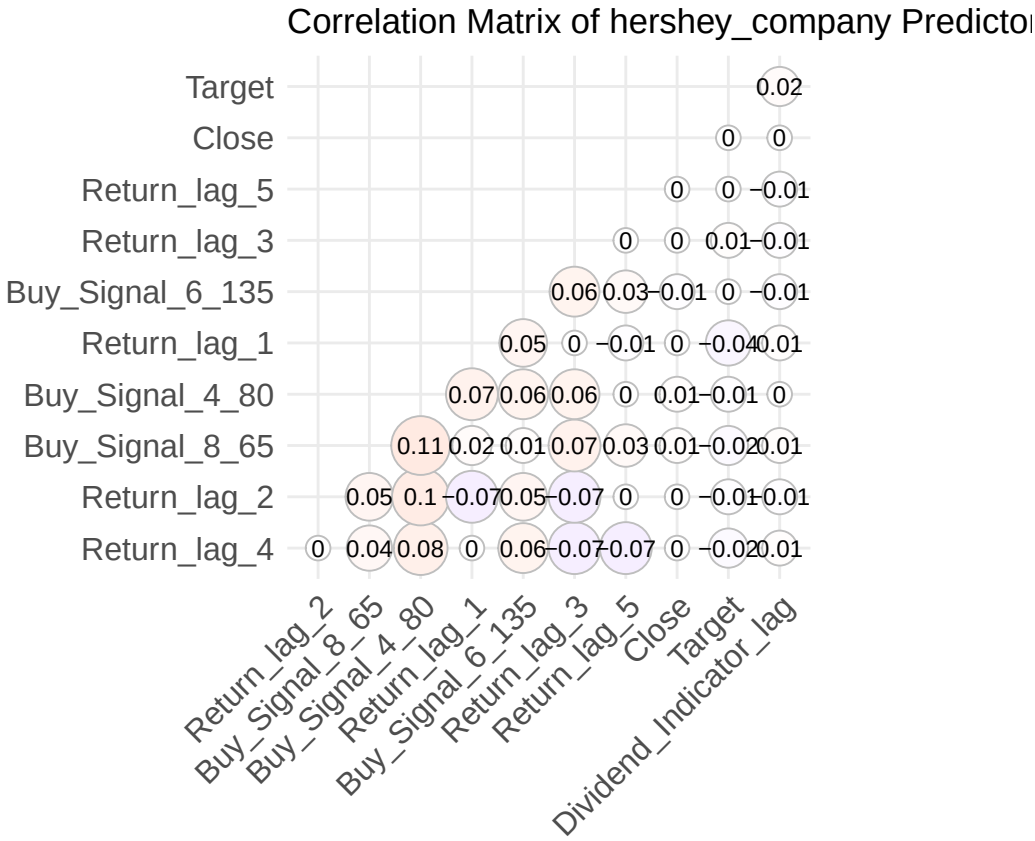
After generating the features, we remove any rows containing NA values created by lagging. We also drop the raw SMA lag columns and the original Return variable from Predictor_Data, as they are either highly correlated or directly used to create the target. This avoids multicollinearity and improves model stability.

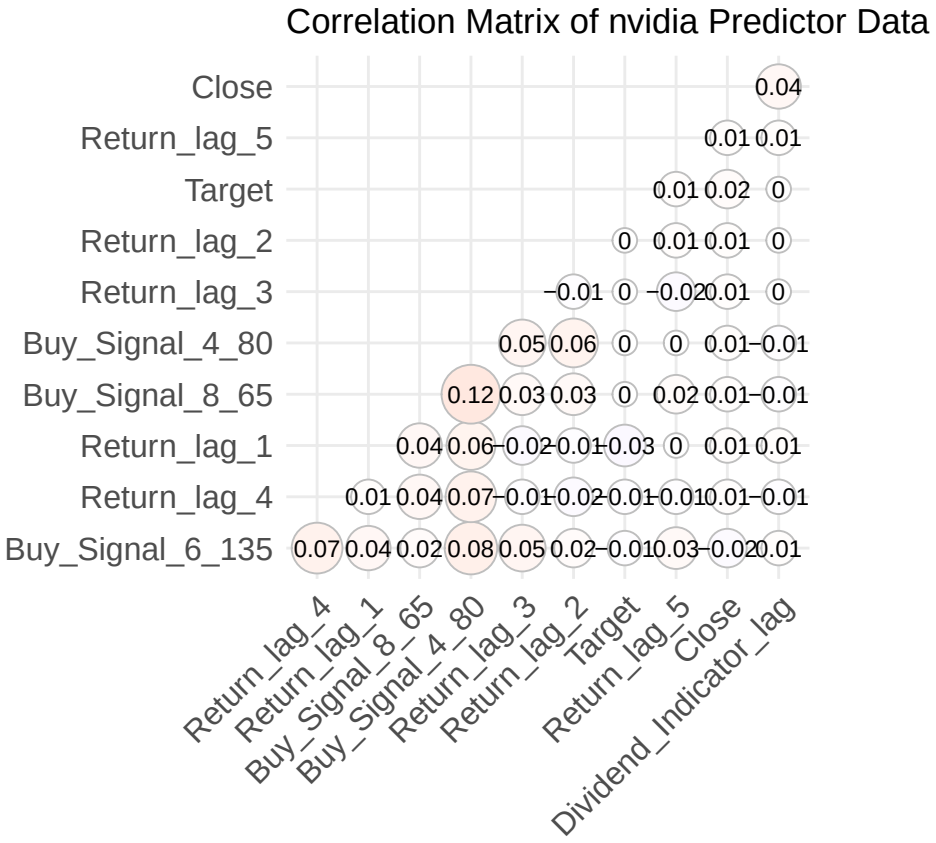
```r
# Remove NAs and indicators that we dont need
# (remove SMAs because they are correlated)
for (company in names(portfolio_data)) {
  portfolio_data[[company]][["Predictor_Data"]] <-
    na.omit( portfolio_data[[company]][["Predictor_Data"]])

  portfolio_data[[company]][["Predictor_Data"]]$sma_4_lag <- NULL
  portfolio_data[[company]][["Predictor_Data"]]$sma_6_lag <- NULL
  portfolio_data[[company]][["Predictor_Data"]]$sma_8_lag <- NULL
  portfolio_data[[company]][["Predictor_Data"]]$sma_65_lag <- NULL
  portfolio_data[[company]][["Predictor_Data"]]$sma_80_lag <- NULL
  portfolio_data[[company]][["Predictor_Data"]]$sma_135_lag <- NULL
  portfolio_data[[company]][["Predictor_Data"]]$Return <- NULL

}
```

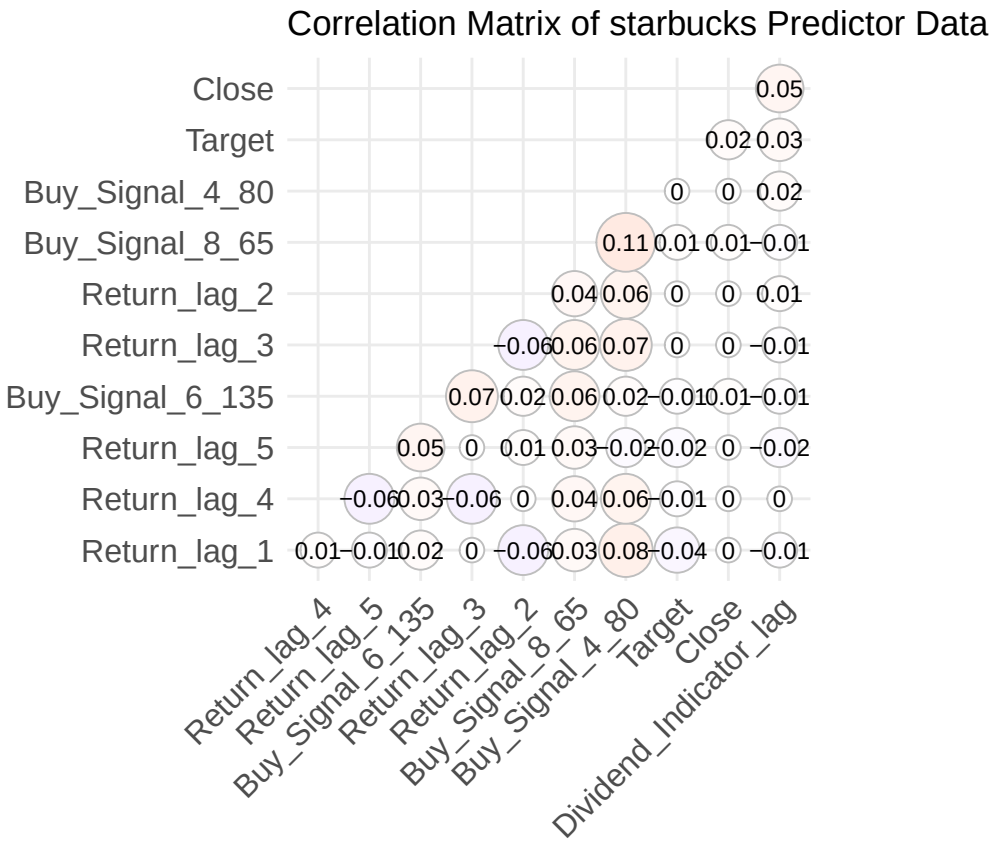
We generate correlation matrices for each company's Predictor_Data to inspect how strongly the features relate to one another. This helps diagnose multicollinearity and understand the overall feature landscape.

```r
# Check final correlation between predictor variables
for (company in names(portfolio_data)) {
  # Extract the numeric data from the Predictor_Data of the current company
  numeric_data <- portfolio_data[[company]][["Predictor_Data"]]
  numeric_data <- numeric_data[, sapply(numeric_data, is.numeric)]

  # Calculate the correlation matrix
  cor_matrix <- cor(numeric_data, use = "complete.obs")

  # Plot the correlation matrix
  print(
    ggcorrplot(cor_matrix,
               hc.order = TRUE,
               type = "lower",
               lab = TRUE,
               lab_size = 3,
               method = "circle",
               colors = c("blue", "white", "red"),
               title = paste("Correlation Matrix of", company, "Predictor Data"),
               ggtheme = ggplot2::theme_minimal()) +
      theme(legend.position = "none")
  )
}
```


Correlation Matrix of apple Predictor Data
Buy_Signal_6_135
Return_lag_5
Return_lag_3
Buy_Signal_4_80
Buy_Signal_8_65
Return_lag_2
Dividend_Indicator_lag
Close
Return_lag_4
Target
Return_lag_4
Close
Dividend_Indicator_lag
Return_lag_2
Buy_Signal_8_65
Buy_Signal_4_80
Return_lag_3
Return_lag_5
Buy_Signal_6_135
Return_lag_1


Correlation Matrix of hershey_company Predicto
Target
Close
Return_lag_5
Return_lag_3
Buy_Signal_6_135
Return_lag_1
Buy_Signal_4_80
Buy_Signal_8_65
Return_lag_2
Return_lag_4
Return_lag_2
Buy_Signal_8_65
Buy_Signal_4_80
Return_lag_1
Buy_Signal_6_135
Return_lag_3
Return_lag_5
Close
Target
Dividend_Indicator_lag


Correlation Matrix of nvidia Predictor Data
Close
Return_lag_5
Target
Return_lag_2
Return_lag_3
Buy_Signal_4_80
Buy_Signal_8_65
Return_lag_1
Return_lag_4
Buy_Signal_6_135
Return_lag_4
Return_lag_1
Buy_Signal_8_65
Buy_Signal_4_80
Return_lag_3
Return_lag_2
Target
Return_lag_5
Close
Dividend_Indicator_lag



The correlation matrices indicates that there is no strong multicollinearity among our predictor variables, which allows us to proceed confidently with model training and further analysis.

As a final step before initiating the model fitting process, we split and overwrite the three datasets using the contents of **Predictor_Data**, keeping only the relevant variables. This streamlines the workflow and improves computational efficiency by reducing data size and eliminating unnecessary columns.

```
for (company in names(portfolio_data)) {
  # store the df as variable for shorter code
  pred_data <- portfolio_data[[company]][["Predictor_Data"]]

  # create new data frames as elements in each company's list
  portfolio_data[[company]][["Train_Data"]] <-
    pred_data[pred_data$Date < as.Date("2015-12-31"),]

  portfolio_data[[company]][["Validation_Data"]] <-
    pred_data[pred_data$Date > as.Date("2015-12-31") &
                pred_data$Date < as.Date("2020-12-31"),]

  portfolio_data[[company]][["Clustering_Data"]] <-
    pred_data[pred_data$Date < as.Date("2020-12-31"),]

  portfolio_data[[company]][["Trade_Data"]] <-
    pred_data[pred_data$Date > as.Date("2020-12-31"),]

  # order the data
  portfolio_data[[company]][["Train_Data"]] <-
```

```
    portfolio_data[[company]][["Train_Data"]][order(portfolio_data[[company]][["Train_Data"]]$Date), ]

  portfolio_data[[company]][["Validation_Data"]] <-
    portfolio_data[[company]][["Validation_Data"]][order(portfolio_data[[company]][["Validation_Data"]]$

  portfolio_data[[company]][["Clustering_Data"]] <-
    portfolio_data[[company]][["Clustering_Data"]][order(portfolio_data[[company]][["Clustering_Data"]]$

  portfolio_data[[company]][["Trade_Data"]] <-
    portfolio_data[[company]][["Trade_Data"]][order(portfolio_data[[company]][["Trade_Data"]]$Date), ]

}
```

Model Training

This part estimates three machine learning models - logistic regression (logit), decision tree and random forest for each stock in the portfolio, using their training data sets. The logistic regression model uses lagged returns, dividend indicators and our technical SMA crossover buy signals as predictors and applies backward step wise selection to retain only the most relevant variables. The decision tree model is trained using the same predictors to capture non-linear relationships. The random forest model, built with 100 trees, provides probabilistic classification and calculates variable importance through permutation.

All trained models are stored within each company's data structure under appropriate keys (logit_model, tree_model, rf_model) for later use in validation or trading.

```
# Fit the logit model
for (company in names(portfolio_data)) {

  logit_model  <- glm(Target ~ Return_lag_1
                      + Return_lag_2 + Return_lag_3 + Return_lag_4
                      + Return_lag_5
                      + Dividend_Indicator_lag + Buy_Signal_6_135
                      + Buy_Signal_8_65 + Buy_Signal_4_80,
                      data = portfolio_data[[company]][["Train_Data"]],
                      family = binomial)

  best_logit <- step(logit_model, direction = "backward")

  portfolio_data[[company]][["logit_model"]] <- best_logit
}
```

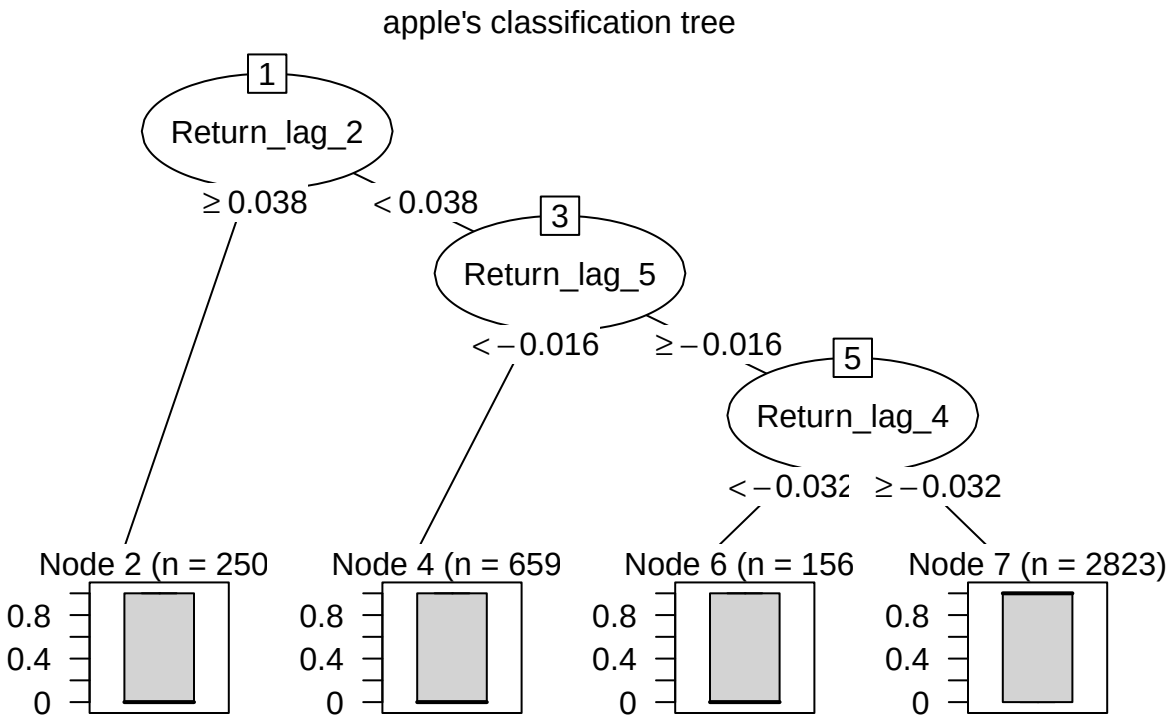
```
for (company in names(portfolio_data)) {
  cat(company, "- logistic regression formula:\n", paste(deparse(portfolio_data[[company]][["logit_model
}
```

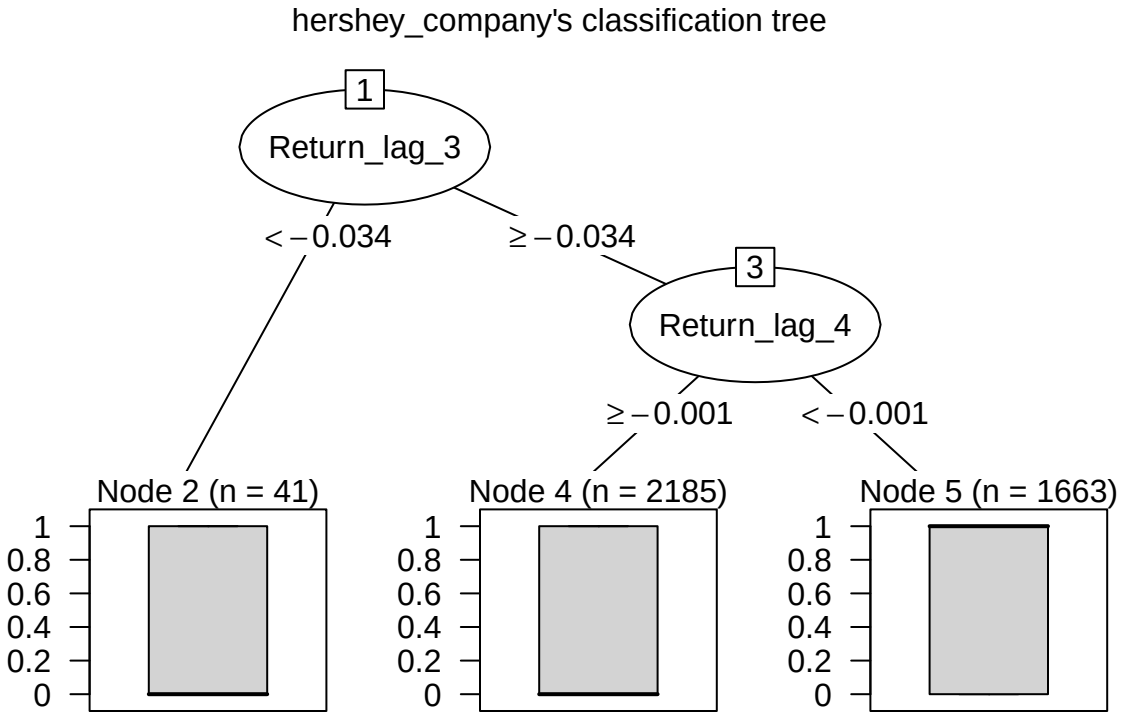
```
## apple - logistic regression formula:
##  Target ~ Return_lag_1 + Return_lag_4 + Return_lag_5 + Buy_Signal_4_80
##
##
## hershey_company - logistic regression formula:
##  Target ~ Return_lag_1 + Dividend_Indicator_lag + Buy_Signal_8_65
##
```

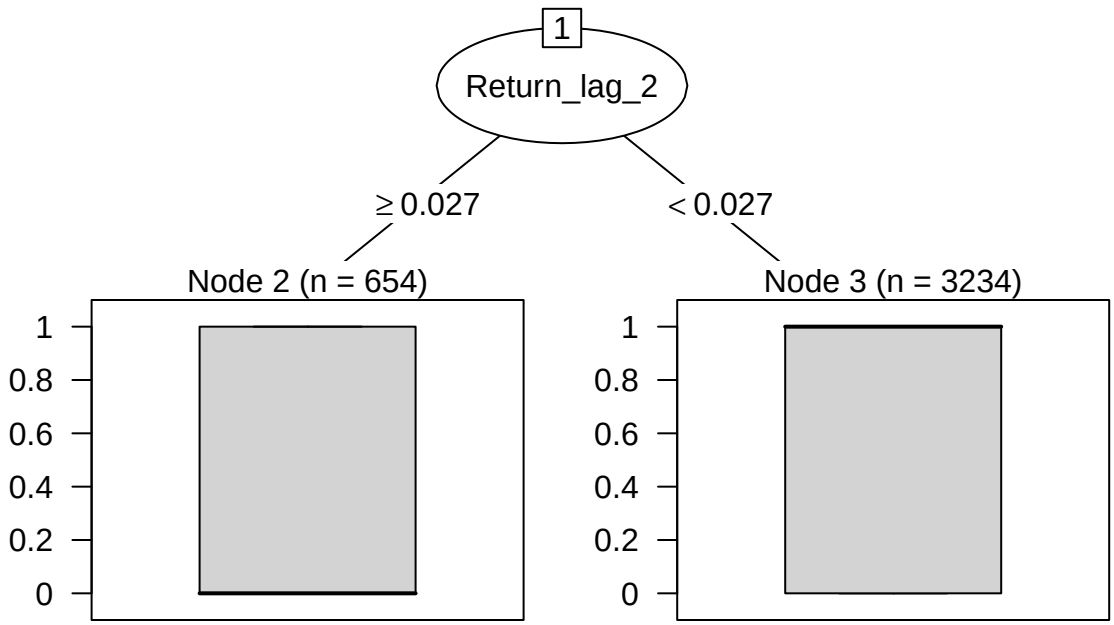
```
##
## nvidia - logistic regression formula:
##  Target ~ Return_lag_5 + Dividend_Indicator_lag
##
##
## starbucks - logistic regression formula:
##  Target ~ Return_lag_1 + Return_lag_2 + Return_lag_5 + Buy_Signal_8_65
```

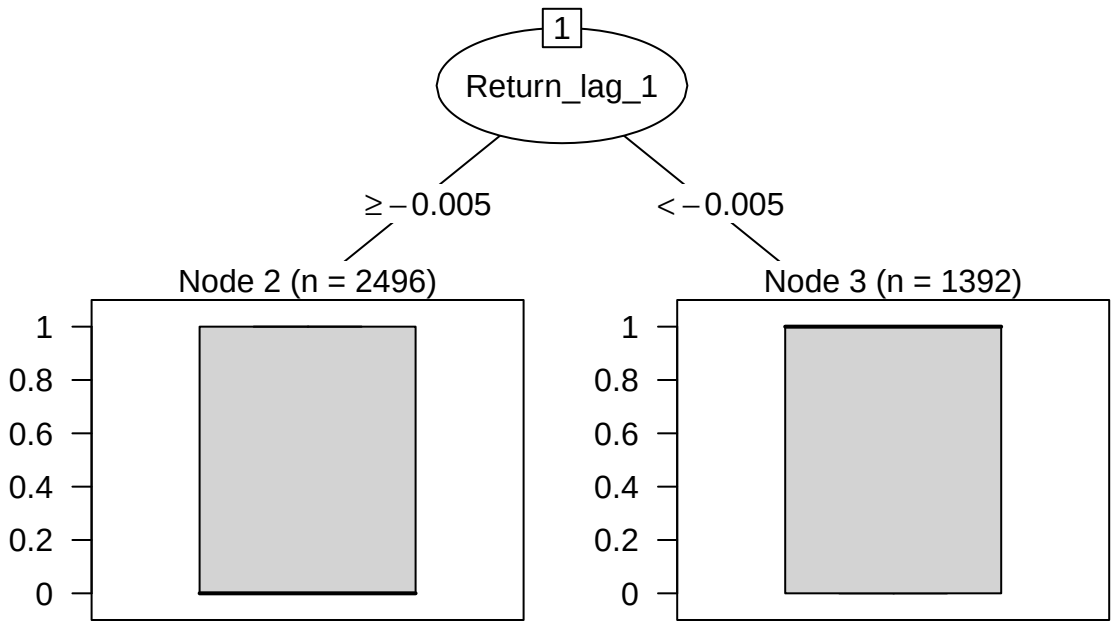
After fitting and optimizing the models, we take a look at the final selected formula for each company. For Apple we can see that prediction is driven mostly by short-term returns and a specific buy signal. For the Hershey Company the formula indicates dividend signals and buy momentum play a larger role. With Nvidia we have a simpler model that focuses on longer-term lag and dividend behavior. From the Starbucks' formula we can see that more return lags retained, suggesting price patterns over several days are predictive.

```
# Fit the decision tree
for (company in names(portfolio_data)) {

  tree_model <- rpart(Target ~ Return_lag_1
                      + Return_lag_2 + Return_lag_3 + Return_lag_4
                      + Return_lag_5
                      + Dividend_Indicator_lag + Buy_Signal_6_135
                      + Buy_Signal_8_65 + Buy_Signal_4_80,
                      data = portfolio_data[[company]][["Train_Data"]],
                      method = "class")
  portfolio_data[[company]][["tree_model"]] <- tree_model

  plot(as.party(portfolio_data[[company]][["tree_model"]]),
       main = paste0(company, "'s classification tree"))
}
```


apple's classification tree
1
Return_lag_2
≥ 0.038
< 0.038
3
Return_lag_5
< −0.016
≥ −0.016
5
Return_lag_4
< −0.032
≥ −0.032
Node 2 (n = 250
Node 4 (n = 659
Node 6 (n = 156
Node 7 (n = 2823)
0.8
0.4
0
0.8
0.4
0
0.8
0.4
0
0.8
0.4
0


hershey_company's classification tree
1
Return_lag_3
< −0.034
≥ −0.034
3
Return_lag_4
≥ −0.001
< −0.001
Node 2 (n = 41)
Node 4 (n = 2185)
Node 5 (n = 1663)
1
0.8
0.6
0.4
0.2
0


nvidia's classification tree
1
Return_lag_2
≥ 0.027
< 0.027
Node 2 (n = 654)
Node 3 (n = 3234)
1
0.8
0.6
0.4
0.2
0
1
0.8
0.6
0.4
0.2
0



Apple's tree is the most complex and it suggests that recent performance over the past few days significantly influences the prediction. In contrast, for Hershey the tree is simpler and indicates the model is relying on moderately recent historical returns to classify future price direction. Nvidia's tree is very shallow, with only a single decision split, which may lack flexibility to capture more nuanced behavior. Similarly, Starbucks has also a very simple tree with a single split that suggests that immediate past return is the most informative feature for this stock.

```
# Fit the RF
for (company in names(portfolio_data)) {

  rf_model <- ranger(Target ~ Return_lag_1
                       + Return_lag_2 + Return_lag_3 + Return_lag_4
                       + Return_lag_5
                       + Dividend_Indicator_lag + Buy_Signal_6_135
                       + Buy_Signal_8_65 + Buy_Signal_4_80,
                     data = portfolio_data[[company]][["Train_Data"]],
                     probability = TRUE, num.trees = 100,
                     importance = "permutation", classification = TRUE)

  portfolio_data[[company]][["rf_model"]] <- rf_model
}
```

Model Validation and Evaluation

In this part we evaluate the performance of three predictive models on the validation dataset for each company in the portfolio. First, we retrieve the trained models and make predictions on the validation data. For the logistic model, the code computes predicted probabilities and applies a threshold of 0.5 to classify outcomes as 0 or 1. The decision tree directly predicts class labels and the random forest model extracts the probability of the positive class and similarly applies a 0.5 threshold.

```
# Predictions on Validation data
for (company in names(portfolio_data)) {
  logit_model <- portfolio_data[[company]]$logit_model
  tree_model <- portfolio_data[[company]]$tree_model
  rf_model <- portfolio_data[[company]]$rf_model
  Validation_Data <- portfolio_data[[company]][["Validation_Data"]]

  # Logit predictions
  logit_pred <- predict(logit_model, newdata = Validation_Data, type = "response")
  portfolio_data[[company]][["Validation_Data"]]$logit_pred <-
    ifelse(logit_pred > 0.50, 1, 0)

  # Tree predictions
  tree_pred <- predict(tree_model, newdata = Validation_Data, type = "class")
  portfolio_data[[company]][["Validation_Data"]]$tree_pred <-
    as.numeric(as.character(tree_pred))

  # RF predictions
  rf_pred <- predict(rf_model, data = Validation_Data,
                     type = "response")$predictions[,2]
  portfolio_data[[company]][["Validation_Data"]]$rf_pred <-
    ifelse(rf_pred > 0.5, 1, 0)
}
```

After generating predictions, we construct confusion matrices for each model to compare predicted versus actual target values. We then calculate precision and accuracy for each model. We store the results into the *portfolio_data* list under a dedicated *Validation_Results* section for each company.

```
# Check the confusion matrices, precision and accuracy
for (company in names(portfolio_data)) {
  cat("\n====================\n")
  cat("Company:", toupper(company), "\n")
  cat("====================\n\n")

  Validation_Data <- portfolio_data[[company]][["Validation_Data"]]

  logit_table <- table(Predicted = Validation_Data$logit_pred, Actual = Validation_Data$Target)
  tree_table  <- table(Predicted = Validation_Data$tree_pred, Actual = Validation_Data$Target)
  rf_table    <- table(Predicted = Validation_Data$rf_pred, Actual = Validation_Data$Target)

  logit_prec <- logit_table[2,2] / (logit_table[2,1] + logit_table[2,2])
  tree_prec  <- tree_table[2,2]  / (tree_table[2,1] + tree_table[2,2])
  rf_prec    <- rf_table[2,2]    / (rf_table[2,1] + rf_table[2,2])

  logit_acc <- mean(Validation_Data$logit_pred == Validation_Data$Target)
```

```
  tree_acc  <- mean(Validation_Data$tree_pred == Validation_Data$Target)
  rf_acc    <- mean(Validation_Data$rf_pred == Validation_Data$Target)

  # Display confusion matrices
  cat("Confusion Matrix - Logit:\n")
  print(logit_table)
  cat("\nConfusion Matrix - Decision Tree:\n")
  print(tree_table)
  cat("\nConfusion Matrix - Random Forest:\n")
  print(rf_table)

  # Display precision
  cat("\nPrecision:\n")
  cat(sprintf("  Logit:         %.3f\n", logit_prec))
  cat(sprintf("  Decision Tree: %.3f\n", tree_prec))
  cat(sprintf("  Random Forest: %.3f\n", rf_prec))

  # Display accuracy
  cat("\nAccuracy:\n")
  cat(sprintf("  Logit:         %.3f\n", logit_acc))
  cat(sprintf("  Decision Tree: %.3f\n", tree_acc))
  cat(sprintf("  Random Forest: %.3f\n", rf_acc))

  cat("\n======================\n\n")

  # Store accuracy in the portfolio_data list
  portfolio_data[[company
  ]][["Validation_Results"]][["Model_Precision"]] <- data.frame(
    logit = logit_prec,
    tree  = tree_prec,
    rf    = rf_prec
  )

  # Store precision in the portfolio_data list
  portfolio_data[[company
  ]][["Validation_Results"]][["Model_Accuracy"]] <- data.frame(
    logit = logit_acc,
    tree  = tree_acc,
    rf    = rf_acc
  )

  # Store confusion matrix in the portfolio_data list
  portfolio_data[[company]][["Validation_Results"]][["Logit_Table"]] <- table(
    Predicted = Validation_Data$logit_pred, Actual = Validation_Data$Target)
  portfolio_data[[company]][["Validation_Results"]][["Tree_Table"]] <- table(
    Predicted = Validation_Data$tree_pred, Actual = Validation_Data$Target)
  portfolio_data[[company]][["Validation_Results"]][["RF_Table"]] <- table(
    Predicted = Validation_Data$rf_pred, Actual = Validation_Data$Target)
}
```

```
##
## ======================
## Company: APPLE
```

```
## ====================
##
## Confusion Matrix - Logit:
##           Actual
## Predicted   0   1
##         0  81  65
##         1 493 619
##
## Confusion Matrix - Decision Tree:
##           Actual
## Predicted   0   1
##         0  89  97
##         1 485 587
##
## Confusion Matrix - Random Forest:
##           Actual
## Predicted   0   1
##         0 230 287
##         1 344 397
##
## Precision:
##   Logit:         0.557
##   Decision Tree: 0.548
##   Random Forest: 0.536
##
## Accuracy:
##   Logit:         0.556
##   Decision Tree: 0.537
##   Random Forest: 0.498
##
## ====================
##
##
## ====================
## Company: HERSHEY_COMPANY
## ====================
##
## Confusion Matrix - Logit:
##           Actual
## Predicted   0   1
##         0  88 103
##         1 490 577
##
## Confusion Matrix - Decision Tree:
##           Actual
## Predicted   0   1
##         0 346 398
##         1 232 282
##
## Confusion Matrix - Random Forest:
##           Actual
## Predicted   0   1
##         0 261 313
##         1 317 367
```

```
##
## Precision:
##   Logit:        0.541
##   Decision Tree: 0.549
##   Random Forest: 0.537
##
## Accuracy:
##   Logit:        0.529
##   Decision Tree: 0.499
##   Random Forest: 0.499
##
## ====================
##
##
## ====================
## Company: NVIDIA
## ====================
##
## Confusion Matrix - Logit:
##          Actual
## Predicted   0   1
##         0 181 242
##         1 378 457
##
## Confusion Matrix - Decision Tree:
##          Actual
## Predicted   0   1
##         0  66 110
##         1 493 589
##
## Confusion Matrix - Random Forest:
##          Actual
## Predicted   0   1
##         0 286 358
##         1 273 341
##
## Precision:
##   Logit:        0.547
##   Decision Tree: 0.544
##   Random Forest: 0.555
##
## Accuracy:
##   Logit:        0.507
##   Decision Tree: 0.521
##   Random Forest: 0.498
##
## ====================
##
##
## ====================
## Company: STARBUCKS
## ====================
##
## Confusion Matrix - Logit:
```

```
##          Actual
## Predicted   0   1
##         0 299 340
##         1 308 311
##
## Confusion Matrix - Decision Tree:
##          Actual
## Predicted   0   1
##         0 428 470
##         1 179 181
##
## Confusion Matrix - Random Forest:
##          Actual
## Predicted   0   1
##         0 309 340
##         1 298 311
##
## Precision:
##   Logit:        0.502
##   Decision Tree: 0.503
##   Random Forest: 0.511
##
## Accuracy:
##   Logit:        0.485
##   Decision Tree: 0.484
##   Random Forest: 0.493
##
## ======================
```

We chose to compare the models based on their Precision score, as a False Positive prediction is more costly for us. When trading, a FP prediction would make the algorithm buy, when the price actually closes at lower level tomorrow and this will make the portfolio lose money. On the other hand, a FN prediction would make the algorithm not buy, when the price actually rises. In especially risky setups like in our project, we follow the idea that capital preservation is more important than chasing profit. A high precision would mean that when the model predicts a "buy", it is more likely to be correct, thus reducing unprofitable trades. As the great investor Warren Buffett has said:

"*The first rule of trading is: don't lose money. The second rule is: don't forget the first rule*".

The most suitable model for each stock according to precision is as follows:

Company	Best Model	Precision
Apple	Logit	0.557
Hershey Company	Decision Tree	0.549
Nvidia	Random Forest	0.555
Starbucks	Random Forest	0.511

Investment Function Design

To make our trading strategy more intelligent and risk-aware, we designed an investment function that determines how much of the company's capital to invest on any given trading day for each stock. This function takes into account the predictions of three models -logistic regression, decision tree, and random forest - and weighs them by their precision scores.

Each prediction model provides a binary prediction: 1 if it predicts the stock will go up, and 0 otherwise. However, since not all models are equally reliable, we first evaluate their precision scores on validation data and assign them a rank - 1 for most precise, 2 for second most precise, and 4 for least precise. These ranks reflect how much trust we place in each model's predictions.

You can find a more detailed explanation and a clarifying example in the appendix.

```
# Define investment function
investment_function <- function(company) {

  # Load the precision results
  a <- as.data.frame(t(portfolio_data[[company
  ]][["Validation_Results"]][["Model_Precision"]]))
  a$rank <- rank(-a$V1, ties.method = "first")
  # Adjust the score from 3 to 4
  a$rank <- ifelse(a$rank == 3, 4, a$rank)

  # Create matrix b storing the combinations
  b <- data.frame(
    logit = c(0,1,0,0,1,1,0,1),
    tree = c(0,0,1,0,1,0,1,1),
    rf = c(0,0,0,1,0,1,1,1)
  )

  # Transpose the data frame 'a' to match the dimensions
  # for matrix multiplication
  a_t <- t(as.matrix(a[,2, drop = FALSE]))

  # Perform matrix multiplication
  result <- a_t %*% t(as.matrix(b))

  # Convert the result to a data frame and add to 'b'
  result <- as.data.frame(t(result))
  colnames(result) <- "result"
  result_df <- cbind(b, result)

  # Percentage for score
  c <- data.frame(
    score = c(0,1,2,4,3,5,6,7),
    percentage = c(0,0.4,0.37,0.35,0.5,0.47,0.45,0.6)
  )

  # Merge the result with the percentage data
  merged_df <- merge(result_df, c, by.x = "result",
                     by.y = "score", all.x = TRUE)
  merged_df <- merged_df[match(result_df$result, merged_df$result), ]
  return(merged_df)
}
```

Trading Strategy and Simulation

In this part of our group project, we simulate how our trading strategy would have performed in real market conditions between 2021 and 2025. The idea is to test the predictive power of our models (logistic

regression, decision tree and random forest) and evaluate how well they guide investment decisions based on their predictions and respective precision.

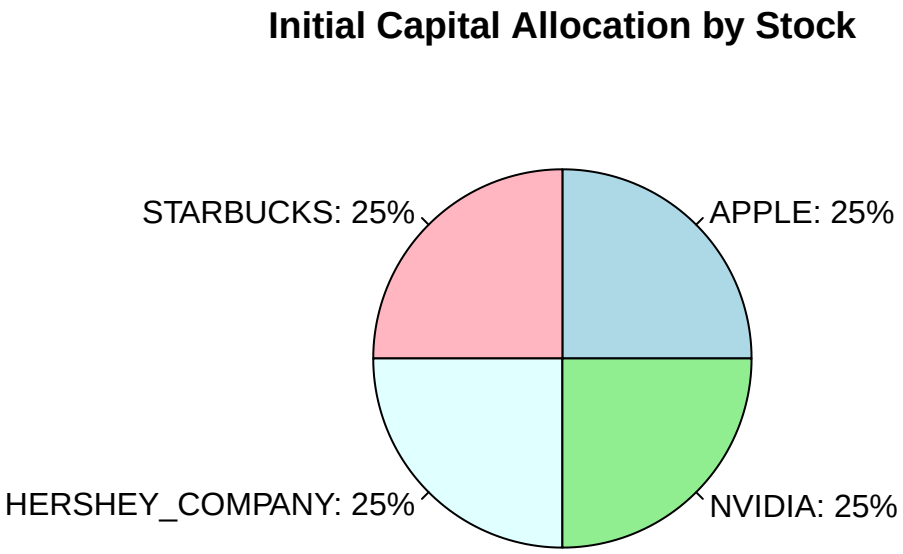
We start by initializing our simulation with a total capital of $10,000, which is evenly divided across the four selected stocks: Apple, Hershey Company, Nvidia and Starbucks. For each company, we prepare a data frame to register daily trades, including the buy price, sell price and resulting profit. Additionally, we create a *Portfolio* data frame that tracks the collective performance of all four stocks, including daily profit and the total portfolio value.

```
# Set the trading balance
initial_balance <- 10000
balance <- initial_balance
trades <- setNames(vector("list", length(portfolio_data)), names(portfolio_data))

# Initialize trades data frames within the list
for (company in names(trades)) {
  trades[[company]] <- data.frame(
    Date = portfolio_data[[company]][["Trade_Data"]]$Date,
    Buy = rep(NA, length(portfolio_data[[company]][["Trade_Data"]]$Date)),
    Sell = rep(NA, length(portfolio_data[[company]][["Trade_Data"]]$Date)),
    Profit = rep(NA, length(portfolio_data[[company]][["Trade_Data"]]$Date)),
    stringsAsFactors = FALSE)
}

trades[["Portfolio"]] <- data.frame(
  Date = portfolio_data[["apple"]][["Trade_Data"]]$Date,
  apple = rep(NA, length(portfolio_data[["apple"]][["Trade_Data"]]$Date)),
  starbucks = rep(NA, length(portfolio_data[["starbucks"]][["Trade_Data"]]$Date)),
  hershey_company = rep(NA, length(portfolio_data[["hershey_company"]][["Trade_Data"]]$Date)),
  nvidia = rep(NA, length(portfolio_data[["nvidia"]][["Trade_Data"]]$Date)),
  Total_Profit = rep(NA, length(portfolio_data[["apple"]][["Trade_Data"]]$Date)),
  stringsAsFactors = FALSE
)

# Set initial balances for each stock (25% of total balance)
balance_df <- data.frame(
  apple = balance * 0.25,
  starbucks = balance * 0.25,
  hershey_company = balance * 0.25,
  nvidia = balance * 0.25,
  Portfolio = balance
)
```

Let's observe how the capital is allocated initially in our portfolio.

```
# Initial capital allocation
balance_values <- as.numeric(balance_df[1, 1:4])
balance_names <- names(balance_df)[1:4]

# Calculate percentages
balance_percentages <- round(100 * balance_values / sum(balance_values), 1)
balance_labels <- paste0(toupper(balance_names), ": ", balance_percentages, "%")

# Create pie chart with percentage labels
```

```r
pie(balance_values, labels = balance_labels, col = c("lightblue", "lightpink", "lightcyan", "lightgreen
    main = "Initial Capital Allocation by Stock")
```



The core of the simulation runs through each day in the trading period (2021–2025). For each day, and for each company, we generate predictions using all three models based on the day's available data. These predictions are converted into binary buy/not buy signals (1 or 0). We then use our previously defined investment function, which takes into account the combination of predictions and the precision of each model, to determine the percentage of capital to invest for that day.

If the models suggest investing, we simulate a buy at today's closing price and a sell at the next day's closing price. The profit from this trade is calculated and added back to the company's capital allocation. This process is repeated for each company on each trading day.

At the end of each day, we record whether each company made a profit, loss or had no trade and compute the total profit for the day across the entire portfolio. We also update the portfolio's cumulative value, which reflects how the total capital grows or shrinks over time based on the trades made.

```r
# Set up progress bar (Use the first company's trade data for length)
trade_data_example <- portfolio_data[[1]][["Trade_Data"]]
pb <- txtProgressBar(min = 0, max = nrow(trade_data_example) - 1, style = 3)

# Run the Simulation
for (i in 1:(nrow(trade_data_example) - 1)) {
  setTxtProgressBar(pb, i)

  for (company in names(portfolio_data)) {
```

```
  # Assign variables for easier use
  logit_model <- portfolio_data[[company]]$logit_model
  tree_model <- portfolio_data[[company]]$tree_model
  rf_model <- portfolio_data[[company]]$rf_model
  trade_data <- portfolio_data[[company]][["Trade_Data"]]

  # Predict tomorrow's move using today's info
  logit_pred <- predict(logit_model, newdata = trade_data[i, , drop = FALSE],
                        type = "response")
  tree_pred <- predict(tree_model, newdata = trade_data[i, , drop = FALSE],
                       type = "class")
  rf_pred <- predict(rf_model, data = trade_data[i, ,
                                                 drop = FALSE])$predictions[, 2]

  # Turn predictions into binary variables
  logit_pred_binary <- ifelse(logit_pred > 0.5, 1, 0)
  tree_pred_binary <- as.numeric(as.character(tree_pred))
  rf_pred_binary <- ifelse(rf_pred > 0.5, 1, 0)

  # Add the predictions to the portfolio_data
  portfolio_data[[company]][["Trade_Data"]]$logit_pred[i] <- logit_pred_binary
  portfolio_data[[company]][["Trade_Data"]]$tree_pred[i] <- tree_pred_binary
  portfolio_data[[company]][["Trade_Data"]]$rf_pred[i] <- rf_pred_binary

  # Combine predictions into a data frame
  prediction_df <- data.frame(logit = logit_pred_binary,
                              tree = tree_pred_binary, rf = rf_pred_binary)

  # Choose how much to trade based on the prediction
  trading_rule <- investment_function(company)
  matched_row <- which(
    apply(trading_rule[, c("logit", "tree", "rf")], 1,
          function(x) all(x == unlist(prediction_df)))
  )
  trading_percentage <- trading_rule$percentage[matched_row]

  # Simulate the trade
  buy_price <- trade_data$Close[i]
  sell_price <- trade_data$Close[i + 1]
  investment <- balance_df[1, company] * trading_percentage
  shares <- investment / buy_price
  profit <- (sell_price - buy_price) * shares
  balance_df[1, company] <- balance_df[1, company] + profit

  # Assign the results to the trades company data frame
  trades[[company]]$Buy[i] <- buy_price
  trades[[company]]$Sell[i] <- sell_price
  trades[[company]]$Profit[i] <- profit

}

# Check which stocks profited, which lost and which were not traded
apple_win <- ifelse(trades[["apple"]]$Profit[i] > 0, 1,
```

```
                      ifelse(trades[["apple"]]$Profit[i] == 0, 0, -1))
  starbucks_win <- ifelse(trades[["starbucks"]]$Profit[i] > 0, 1,
                       ifelse(trades[["starbucks"]]$Profit[i] == 0, 0, -1))
  hershey_company_win <- ifelse(trades[["hershey_company"]]$Profit[i] > 0, 1,
                          ifelse(trades[["hershey_company"]]$Profit[i] == 0, 0, -1))
  nvidia_win <- ifelse(trades[["nvidia"]]$Profit[i] > 0, 1,
                       ifelse(trades[["nvidia"]]$Profit[i] == 0, 0, -1))

  balance_df$Portfolio <- sum(balance_df[1,
                                         c("apple", "starbucks", "hershey_company", "nvidia")])

  # Add the total results of the portfolio for the day
  trades[["Portfolio"]]$apple[i] <- apple_win
  trades[["Portfolio"]]$starbucks[i] <- starbucks_win
  trades[["Portfolio"]]$hershey_company[i] <- hershey_company_win
  trades[["Portfolio"]]$nvidia[i] <- nvidia_win
  trades[["Portfolio"]]$Total_Profit[i] <- sum(trades[["apple"]]$Profit[i],
                                               trades[["starbucks"]]$Profit[i],
                                               trades[["hershey_company"]]$Profit[i],
                                               trades[["nvidia"]]$Profit[i])
  trades[["Portfolio"]]$portfolio_balance[i] <- balance_df$Portfolio

}
```

Model Evaluation on the Trading data set

After generating predictions on the Trading data set, we want to first evaluate how the models performed on predicting the direction of the market. To do so, we will take a look at the Confusion Matrices, the Accuracy and Precision scores of each of the three ML models for the four companies in our portfolio.

```
# Check the confusion matrices, precision and accuracy
for (company in names(portfolio_data)) {
  cat("============================\n")
  cat(toupper(company), "\n")
  cat("============================\n")

  Trade_Data <- portfolio_data[[company]][["Trade_Data"]]

  logit_table <- table(Predicted = Trade_Data$logit_pred, Actual = Trade_Data$Target)
  tree_table  <- table(Predicted = Trade_Data$tree_pred, Actual = Trade_Data$Target)
  rf_table    <- table(Predicted = Trade_Data$rf_pred, Actual = Trade_Data$Target)

  logit_prec <- logit_table[2,2] / (logit_table[2,1] + logit_table[2,2])
  tree_prec  <- tree_table[2,2]  / (tree_table[2,1] + tree_table[2,2])
  rf_prec    <- rf_table[2,2]    / (rf_table[2,1] + rf_table[2,2])

  logit_acc <- mean(Trade_Data$logit_pred == Trade_Data$Target)
  tree_acc  <- mean(Trade_Data$tree_pred == Trade_Data$Target)
  rf_acc    <- mean(Trade_Data$rf_pred == Trade_Data$Target)

  # Display confusion matrices
  cat("Confusion Matrix - Logit:\n")
```

```
  print(logit_table)
  cat("\nConfusion Matrix - Decision Tree:\n")
  print(tree_table)
  cat("\nConfusion Matrix - Random Forest:\n")
  print(rf_table)

  # Display precision
  cat("\nPrecision:\n")
  cat(sprintf("  Logit:         %.3f\n", logit_prec))
  cat(sprintf("  Decision Tree: %.3f\n", tree_prec))
  cat(sprintf("  Random Forest: %.3f\n", rf_prec))

  # Display accuracy
  cat("\nAccuracy:\n")
  cat(sprintf("  Logit:         %.3f\n", logit_acc))
  cat(sprintf("  Decision Tree: %.3f\n", tree_acc))
  cat(sprintf("  Random Forest: %.3f\n", rf_acc))

  cat("\n")

  # Store accuracy in the portfolio_data list
  portfolio_data[[company
  ]][["Test_Results"]][["Model_Precision"]] <- data.frame(
    logit = logit_prec,
    tree  = tree_prec,
    rf    = rf_prec
  )

  # Store precision in the portfolio_data list
  portfolio_data[[company
  ]][["Test_Results"]][["Model_Accuracy"]] <- data.frame(
    logit = logit_acc,
    tree  = tree_acc,
    rf    = rf_acc
  )

  # Store confusion matrix in the portfolio_data list
  portfolio_data[[company]][["Test_Results"]][["Logit_Table"]] <- table(
    Predicted = Trade_Data$logit_pred, Actual = Trade_Data$Target)
  portfolio_data[[company]][["Test_Results"]][["Tree_Table"]] <- table(
    Predicted = Trade_Data$tree_pred, Actual = Trade_Data$Target)
  portfolio_data[[company]][["Test_Results"]][["RF_Table"]] <- table(
    Predicted = Trade_Data$rf_pred, Actual = Trade_Data$Target)
}
```

```
## ============================
## APPLE
## ============================
## Confusion Matrix - Logit:
##          Actual
## Predicted   0   1
##         0  65  78
##         1 444 490
```

```
##
## Confusion Matrix - Decision Tree:
##          Actual
## Predicted   0   1
##         0  95  95
##         1 414 473
##
## Confusion Matrix - Random Forest:
##          Actual
## Predicted   0   1
##         0 216 222
##         1 293 346
##
## Precision:
##   Logit:        0.525
##   Decision Tree: 0.533
##   Random Forest: 0.541
##
## Accuracy:
##   Logit:        0.515
##   Decision Tree: 0.527
##   Random Forest: 0.522
##
## ============================
## HERSHEY_COMPANY
## ============================
## Confusion Matrix - Logit:
##          Actual
## Predicted   0   1
##         0 104  99
##         1 429 445
##
## Confusion Matrix - Decision Tree:
##          Actual
## Predicted   0   1
##         0 300 307
##         1 233 237
##
## Confusion Matrix - Random Forest:
##          Actual
## Predicted   0   1
##         0 234 265
##         1 299 279
##
## Precision:
##   Logit:        0.509
##   Decision Tree: 0.504
##   Random Forest: 0.483
##
## Accuracy:
##   Logit:        0.510
##   Decision Tree: 0.499
##   Random Forest: 0.476
##
```

```
## ============================
## NVIDIA
## ============================
## Confusion Matrix - Logit:
##          Actual
## Predicted   0   1
##         0 208 226
##         1 294 349
##
## Confusion Matrix - Decision Tree:
##          Actual
## Predicted   0   1
##         0  96 114
##         1 406 461
##
## Confusion Matrix - Random Forest:
##          Actual
## Predicted   0   1
##         0 264 320
##         1 238 255
##
## Precision:
##   Logit:        0.543
##   Decision Tree: 0.532
##   Random Forest: 0.517
##
## Accuracy:
##   Logit:        0.517
##   Decision Tree: 0.517
##   Random Forest: 0.482
##
## ============================
## STARBUCKS
## ============================
## Confusion Matrix - Logit:
##          Actual
## Predicted   0   1
##         0 263 253
##         1 280 281
##
## Confusion Matrix - Decision Tree:
##          Actual
## Predicted   0   1
##         0 353 342
##         1 190 192
##
## Confusion Matrix - Random Forest:
##          Actual
## Predicted   0   1
##         0 261 242
##         1 282 292
##
## Precision:
##   Logit:        0.501
```

```
##   Decision Tree: 0.503
##   Random Forest: 0.509
##
## Accuracy:
##   Logit:         0.505
##   Decision Tree: 0.506
##   Random Forest: 0.513
```

When looking at the results, we can see that the model performance is relatively week with precision scores just above 50%, indicating that the ML models predict just slightly better than random guessing. Random Forests tend to perform slightly better for Apple and Starbucks, but not consistently across all companies in the portfolio. For Nvidia and the Hershey Company the model with the highest precision score is the Logistic Regression.

The model that performed the best on the Trading data set for each stock according to precision is as follows:

Company	Best Model	Precision
Apple	Random Forest	0.541
Hershey Company	Logit	0.509
Nvidia	Logit	0.543
Starbucks	Random Forest	0.509

Trading Strategy Performance

After we evaluated the predictions of the models, it is time to evaluate the trading strategy in finance terms. We will calculate a couple of metrics of our portfolio that will tell us how well the strategy using the model performed.

```
cat(" Start balance: $", initial_balance, "\n",
    "End balance: $", round(balance_df$Portfolio, 2), "\n",
    "Total profit: $", round(balance_df$Portfolio - initial_balance, 2), "\n")
```

```
##  Start balance: $ 10000
##  End balance: $ 19003.73
##  Total profit: $ 9003.73
```

The initial evaluation of our trading strategy shows a strong overall performance - **the strategy made money** - which is a promising first signal. However, simply observing that the algorithm made money is not sufficient. To truly assess its effectiveness, we need to dig deeper by examining what drove this profit and whether the strategy offered a more profitable and risk-efficient alternative to passive buy-and-hold investing, either in the S&P 500 or in the individual stocks themselves.

We start by comparing our portfolio to the benchmark we chose - the S&P500.

```
# Calculate cumulative profit
trades[["Portfolio"]]$Cumulative_Profit <- cumsum(
  trades[["Portfolio"]]$Total_Profit)

# Get SPY  data for the trading period
GSPC <- getSymbols("^GSPC", src = "yahoo",
           from = trades[["Portfolio"]]$Date[1],
```

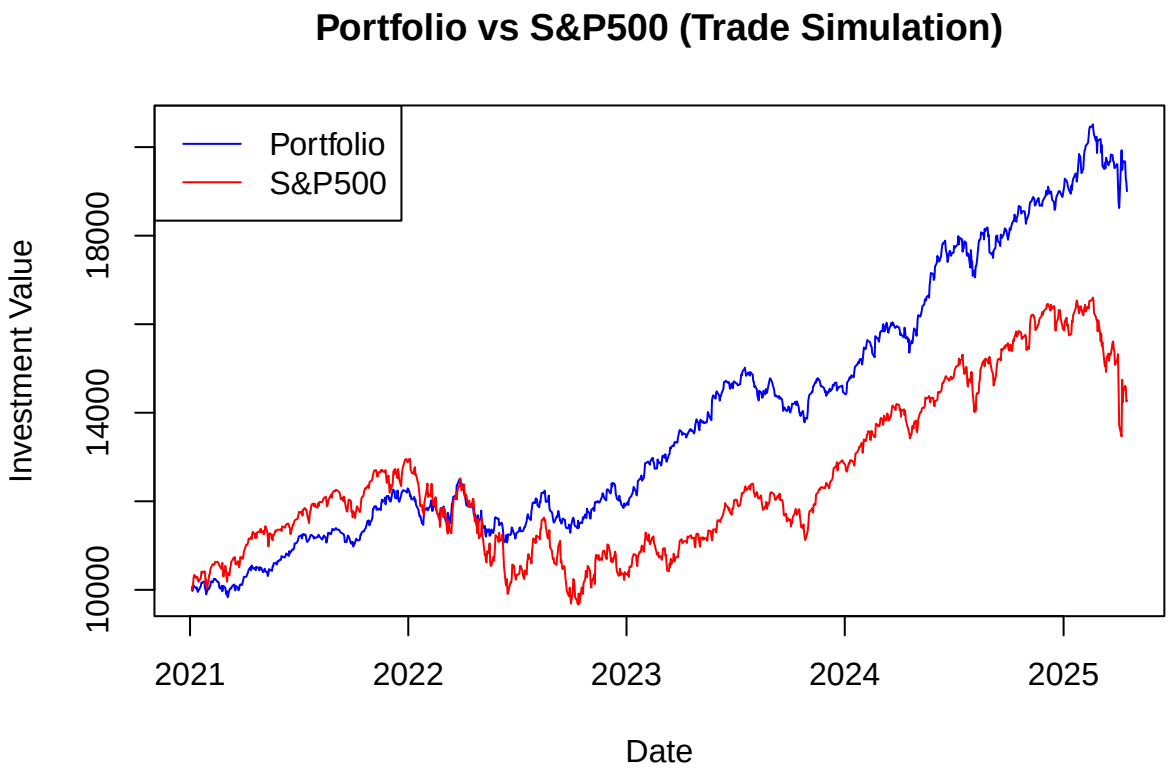
```
          to = trades[["Portfolio"]]$Date[nrow(trades[["Portfolio"]])],
          auto.assign = F)

# Prepare sp500 data
GSPC_df <- data.frame(
  Date = index(GSPC),
  Close = as.numeric(Cl(GSPC))  # Closing price
)

# Merge portfolio and sp500 by Date
trades[["Portfolio"]]$Date <- as.Date(trades[["Portfolio"]]$Date)
merged_data <- merge(trades[["Portfolio"]], GSPC_df, by = "Date", all = FALSE)

colnames(merged_data)[9] <- "close_sp500"

number_of_shares_sp500 <- initial_balance / merged_data$close_sp500[1]

merged_data$sp_value <- merged_data$close_sp500 * number_of_shares_sp500

# Normalize sp500  performance to start from 0
merged_data$sp_cum_profit <- merged_data$sp_value - merged_data$sp_value[1]

# Portfolio return over the whole period (inception-to-date)
portfolio_return_ITD <- ((merged_data$portfolio_balance[nrow(merged_data)]-
                            merged_data$portfolio_balance[1])  /
                           merged_data$portfolio_balance[1])

sp500_return_ITD <- ((merged_data$sp_value[nrow(merged_data)]-
                        merged_data$sp_value[1])  /
                       merged_data$sp_value[1])
cat("Cummulative Returns:\n",
    "Portfolio:",round(portfolio_return_ITD*100,2),"%",
    "\n S&P500:", round(sp500_return_ITD*100,2),"%")
```

```
## Cummulative Returns:
##  Portfolio: 89.01 %
##  S&P500: 42.75 %
```

When comparing cumulative returns, the strategy shows clearly that it outperformed the benchmark. Over the full investment period, our model-driven portfolio achieved a return of **89.01%**, more than doubling the **42.75%** return of the S&P 500 index. This result suggests that the strategy was not only profitable, but also successful in generating excess returns relative to the broader market.

```
# Plot portfolio cumulative profit
plot(merged_data$Date, merged_data$Cumulative_Profit + initial_balance,
     type = "l", col = "blue",
     xlab = "Date", ylab = "Investment Value",
     main = "Portfolio vs S&P500 (Trade Simulation)")

# Add sp500 line
lines(merged_data$Date, merged_data$sp_cum_profit + initial_balance, col = "red")

# Add legend
```

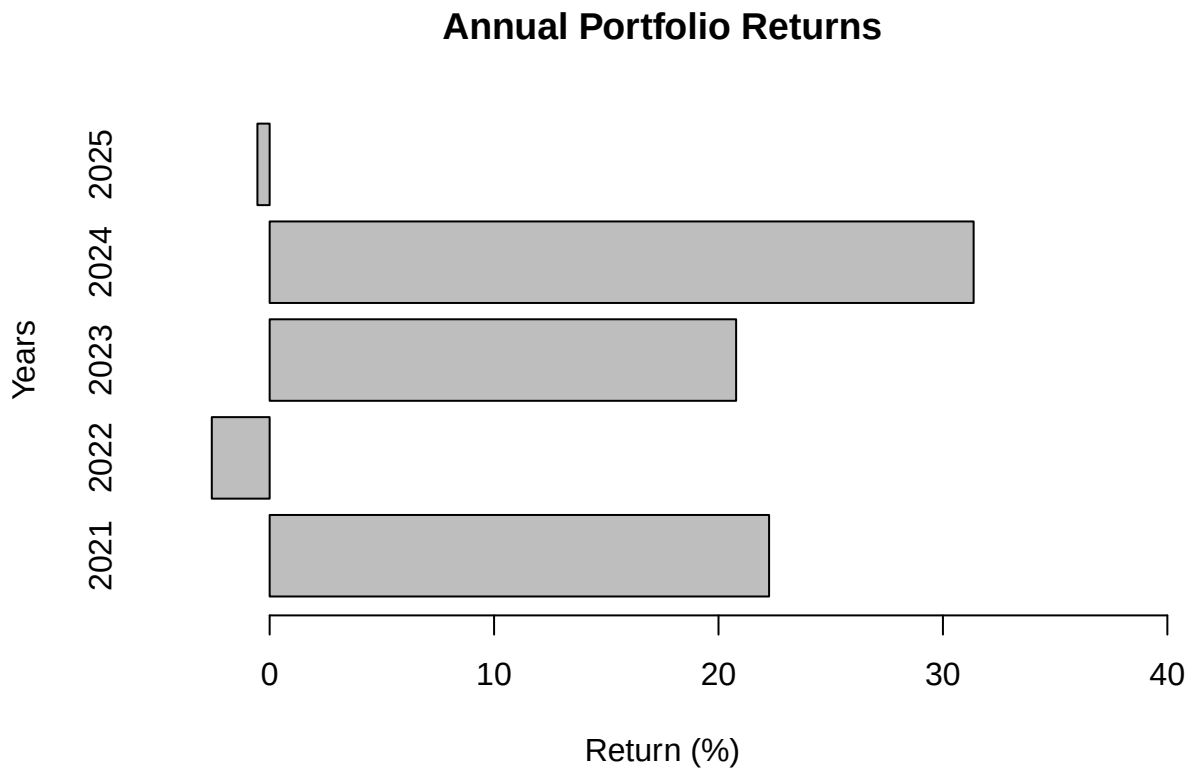
```
legend("topleft", legend = c("Portfolio", "S&P500"),
       col = c("blue", "red"), lty = 1)
```



The cumulative profit chart provides a clear visual confirmation of our strategy's outperformance. While both the portfolio and the S&P 500 followed similar trends in the early stages, the index rose faster than our portfolio. Around the beginning of the 2022 our portfolio proved to be more resilient and both got relatively same value. From late 2022 onward, the portfolio consistently outpaced the market, capturing stronger upward movements and recovering more quickly from downturns. In the early 2025 after the Trump tariffs news our portfolio proved again its resilience compared to the S&P500. The whole movement suggests that the strategy was able to both exploit market opportunities and manage downside risk more effectively than a passive market investment.

```
years <- c("2021","2022","2023","2024","2025")

annual_portfolio_returns <- data.frame(
  Year = years,
  Annual_Return = rep(NA, 5)
)

trades[["Portfolio"]]$Year <- format(trades[["Portfolio"]]$Date, "%Y")

portfolio_by_year <- split(trades[["Portfolio"]], trades[["Portfolio"]]$Year)

for (i in seq(years)) {
  year <- years[i]
  data <- portfolio_by_year[[year]]
```

```
  data <- na.omit(data)

  year_return <- ((data$portfolio_balance[nrow(data)] - data$portfolio_balance[1])
                  / data$portfolio_balance[1])

  annual_portfolio_returns$Annual_Return[i] <- year_return
}

barplot(annual_portfolio_returns$Annual_Return*100,
        names.arg = annual_portfolio_returns$Year,
        xlab = "Return (%)", ylab = "Years",
        main = "Annual Portfolio Returns",
        horiz = T, xlim = c(-5,40))
```

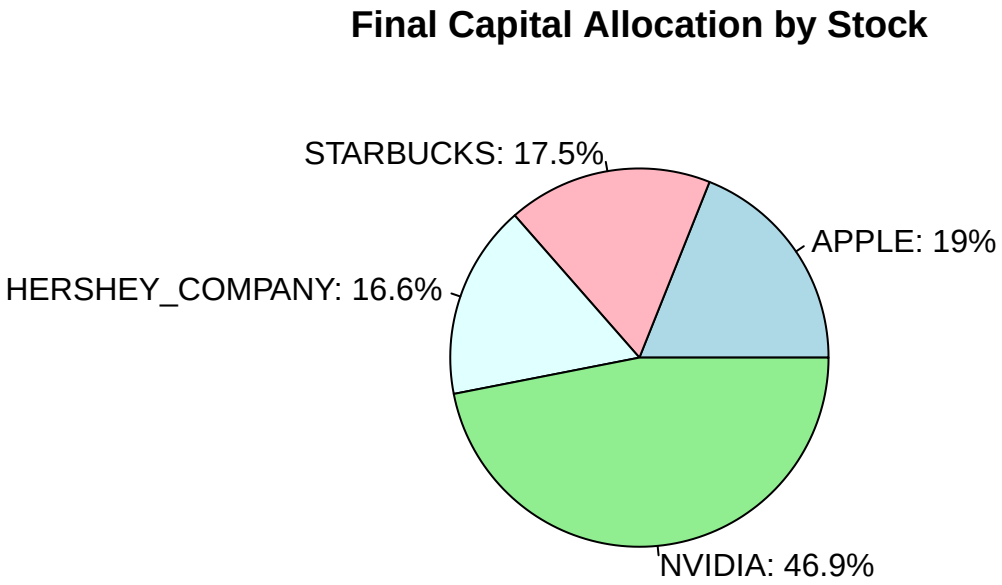


The annual return breakdown reveals how the portfolio's performance evolved over time. While in 2022 the portfolio ended at a little lower level than the beginning of the year and the current value is lower than the beginning of 2025, the strategy delivered exceptionally strong returns in 2021, 2023 and especially 2024, with the latter approaching a **40%** annual return. This pattern highlights the ability of the strategy to capture upward momentum during bullish periods while remaining relatively resilient during less favorable conditions. The consistency of high returns across multiple years strengthens the case for the model's effectiveness beyond random chance or market timing.

Let's take a closer look at our portfolio by analyzing performance at the individual stock level.

```
# Extract the balances (exclude the total portfolio column)
balance_values <- as.numeric(balance_df[1, 1:4])
```

```r
balance_names <- names(balance_df)[1:4]

# Calculate percentages
balance_percentages <- round(100 * balance_values / sum(balance_values), 1)
balance_labels <- paste0(toupper(balance_names), ": ", balance_percentages, "%")

# Create pie chart with percentage labels
pie(balance_values, labels = balance_labels, col = c("lightblue", "lightpink", "lightcyan", "lightgreen
    main = "Final Capital Allocation by Stock")
```



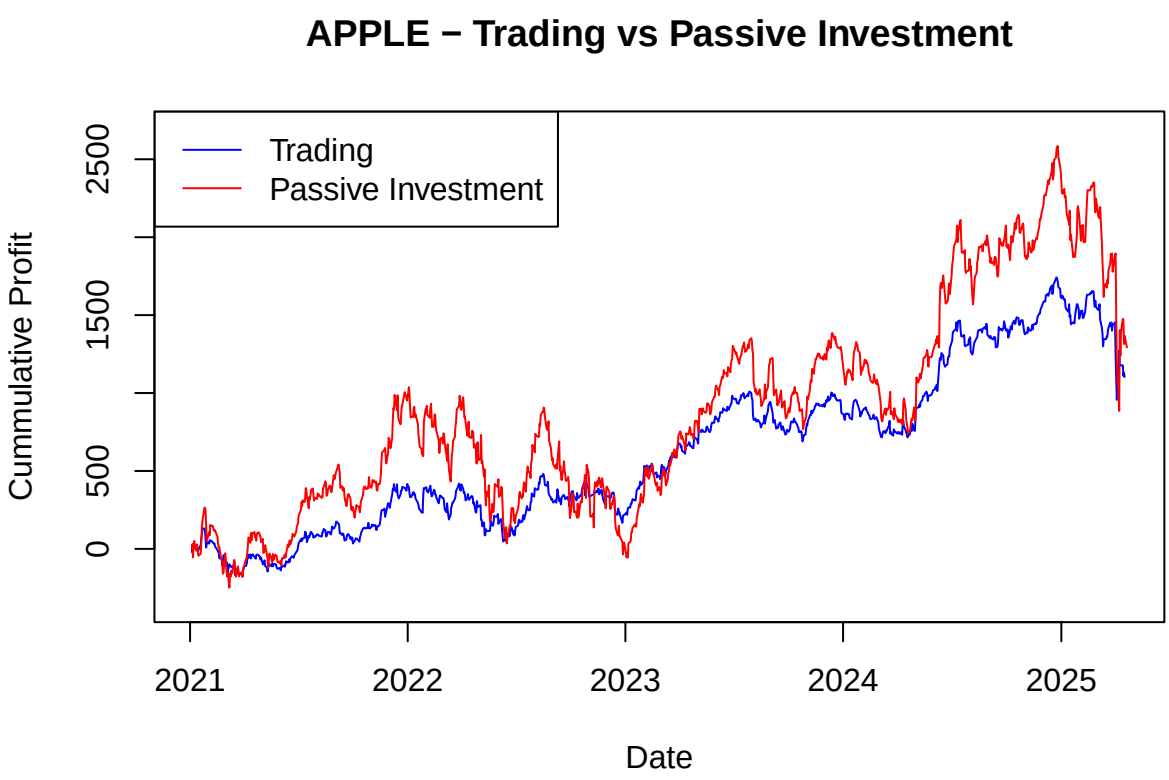
At the beginning of the trading period, our portfolio was equally allocated across the four selected stocks and as the strategy progressed, capital was dynamically reallocated based on the trading signals and returns generated by each stock, leading to the final distribution shown in the pie chart. The domination of Nvidia in our final portfolio suggests it delivered the strongest returns based on our model's predictions. In contrast, Hershey had the weakest performance, ending with just **16.6%**.

Let's now look at how each company performed individually. We will compare the results of our trading strategy to a passive buy-and-hold investment in each stock.

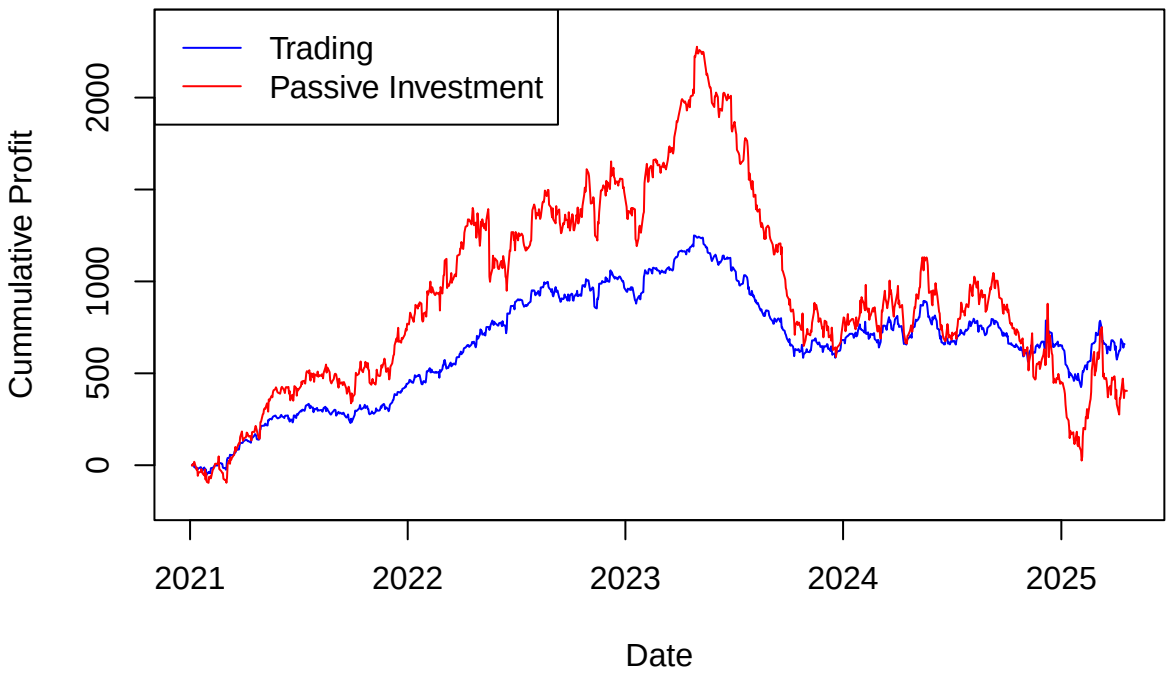
```r
for (company in names(portfolio_data)) {

  # Calculate cumsum of our trades of the company
  trades[[company]]$Cumulative_Trade_Profit <- cumsum(
  trades[[company]]$Profit)

  # Store the ticker as variable
```

```r
ticker <- portfolio_data[[company]][["Company_Info"]]$Ticker

# Store historical data as variable
data <- portfolio_data[[company]][["Historical_Data"]]

# Get start date and end date
start_date <- trades[[company]]$Date[1]
start_date_idx <- which(data$Date == start_date)
end_date <- trades[[company]]$Date[nrow(trades[[company]])]
end_date_idx <- which(data$Date == end_date)

# Get company stock price data
price_data <- data[data$Date >= start_date & data$Date <= end_date, ]

# Prepare the data
df <- data.frame(
Date = price_data$Date,
Close = as.numeric(price_data$Close)
)

# Merge our trades and company stock price data by Date
trades[[company]]$Date <- as.Date(trades[[company]]$Date)

merged_data <- merge(trades[[company]], df, by = "Date", all = FALSE)

number_of_shares <- (initial_balance/4) / merged_data$Close[1]

merged_data$passive_value <- merged_data$Close * number_of_shares

# Normalize passive investment performance to start from 0
merged_data$passive_cum_profit <- merged_data$passive_value - merged_data$passive_value[1]

# Get y limits
y_lower <- min(merged_data$Cumulative_Trade_Profit,
               merged_data$passive_cum_profit, na.rm = T) - 100


y_upper <- max(merged_data$Cumulative_Trade_Profit,
               merged_data$passive_cum_profit, na.rm = T) + 100

# Plot our trades cumulative profit
plot(merged_data$Date, merged_data$Cumulative_Trade_Profit, type = "l", col = "blue",
     xlab = "Date", ylab = "Cummulative Profit",
     main = paste0(toupper(company),' - Trading vs Passive Investment'),
     ylim = c(y_lower,y_upper)
)

# Add passive investment line
lines(merged_data$Date, merged_data$passive_cum_profit, col = "red")

# Add legend
legend("topleft", legend = c("Trading", "Passive Investment"),
       col = c("blue", "red"), lty = 1)
```

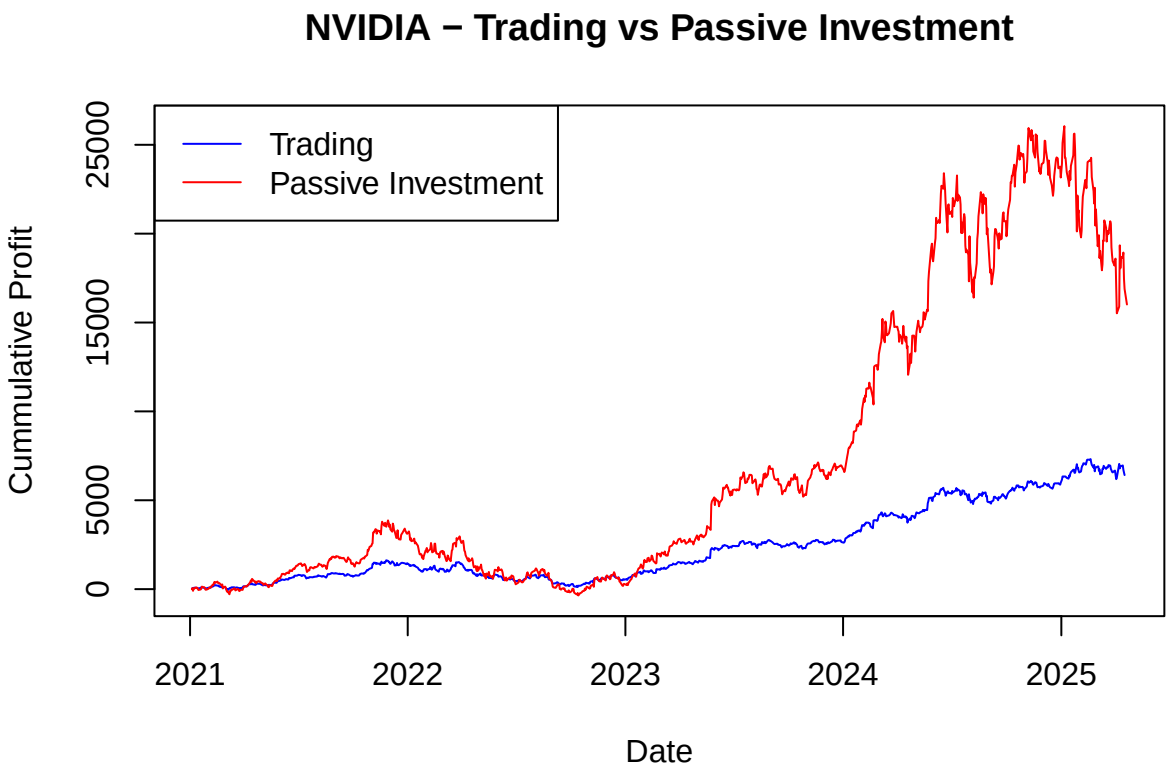
```
  # Trades annual return
  total_trading_days <- nrow(merged_data)
  trading_days_in_years <- total_trading_days / 252

  merged_data$trading_returns <- TTR::ROC(merged_data$Cumulative_Trade_Profit+2500,
                                      type = "discrete")

  mean_daily_trading_return <- mean(merged_data$trading_returns, na.rm = T)

  annualized_trading_return <- mean_daily_trading_return*252

  # Passive investment annual return
  merged_data$investment_daily_returns <- TTR::ROC(merged_data$passive_value,
                                                   type = "discrete")
  mean_daily_investment_return <- mean(merged_data$investment_daily_returns,
                                       na.rm = T)

  annualized_investment_return <- mean_daily_investment_return*252

  # Volatility
  trading_volatility <- sd(merged_data$trading_returns, na.rm = T)
  annualized_trading_volatility <- trading_volatility*sqrt(252)

  investment_volatility <- sd(merged_data$investment_daily_returns, na.rm = T)
  annualized_investment_volatility <- investment_volatility*sqrt(252)

  # Print results
  cat(toupper(company), ":\n",
      "Annualized Return:\n",
      "  Trading:",round(annualized_trading_return*100,2),"%\n",
      "  Passive Investment:", round(annualized_investment_return*100,2),"%\n\n",
      "Annualized Volatility:\n",
      "  Trading:",round(annualized_trading_volatility*100,2),"%\n",
      "  Passive Investment:",
      round(annualized_investment_volatility*100,2),"%\n\n"
     )
}
```



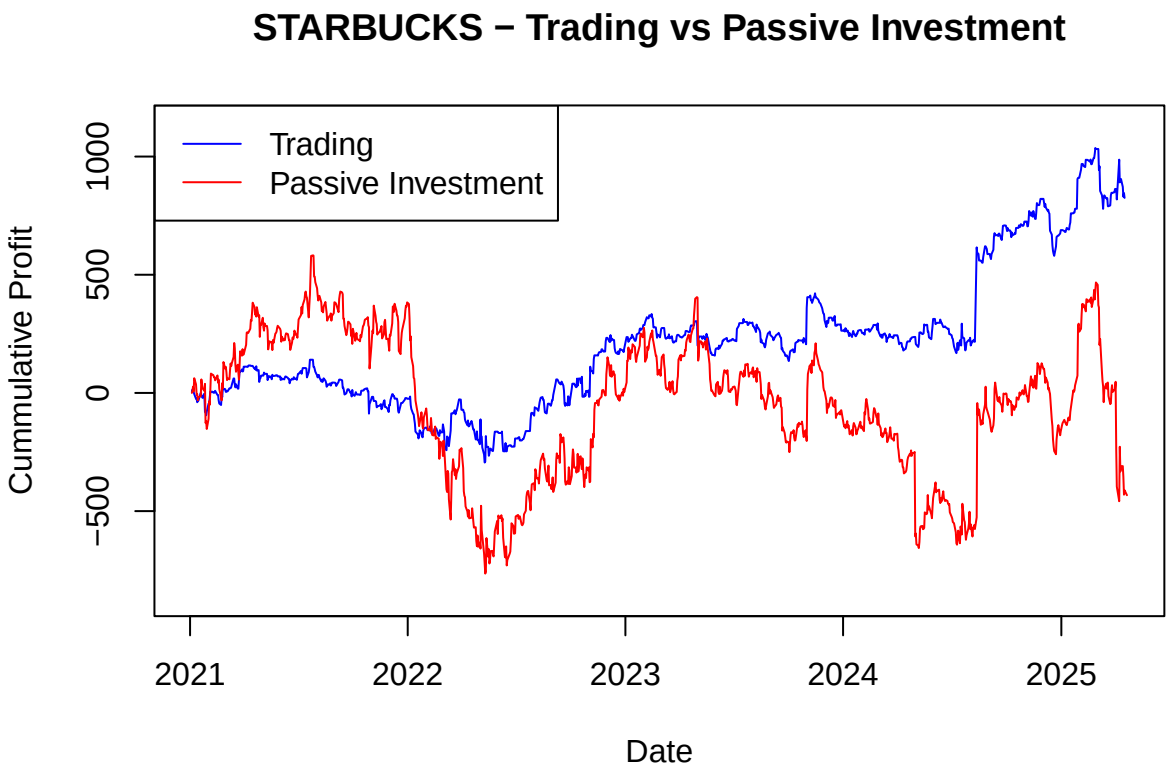

```
## APPLE :
##  Annualized Return:
##    Trading: 9.5 %
##    Passive Investment: 13.88 %
##
##  Annualized Volatility:
##    Trading: 14.9 %
##    Passive Investment: 28.76 %
```




```
## HERSHEY_COMPANY :
##  Annualized Return:
##    Trading: 5.92 %
##    Passive Investment: 5.67 %
##
##  Annualized Volatility:
##    Trading: 9.27 %
##    Passive Investment: 20.81 %
```




```
## NVIDIA :
##  Annualized Return:
##    Trading: 32.51 %
##    Passive Investment: 61.75 %
##
##  Annualized Volatility:
##    Trading: 24.56 %
##    Passive Investment: 54.74 %
```




```
## STARBUCKS :
##  Annualized Return:
##    Trading: 7.63 %
##    Passive Investment: 0.45 %
##
##  Annualized Volatility:
##    Trading: 14.17 %
##    Passive Investment: 31.53 %
```

- **Apple** - while the passive investment yielded a higher annualized return (**13.88%**) than our trading strategy (**9.5%**), it also came with significantly higher volatility - **28.76%** compared to **14.9%**. The trading strategy offered a smoother and less risky growth path. This suggests that for risk-averse investors, the model provided a more stable alternative, even if it underperformed the passive investment in absolute terms.
- **Hershey Company** - In the case of Hershey, the trading strategy slightly outperformed passive investment in terms of annualized return (**5.92%** vs **5.67%**) while cutting volatility by more than half (**9.27%** vs **20.81%**). This is a strong result for the model, as it achieved comparable profits with significantly lower risk.
- **Nvidia** - the passive investment strategy significantly outperformed in terms of raw return, achieving an impressive **61.75%** annualized return compared to **32.51%** from our trading model. While the model didn't capture the full upside of Nvidia's explosive growth, it offered a much smoother ride with a volatility of **24.56%** - way lower than the passive investment's volatility of **54.74%**. Our model sacrificed some profit potential in exchange for far greater stability, highlighting its conservative nature.

- **Starbucks** - Here the trading strategy was superior. It delivered an annualized return of **7.63%**, compared to just **0.45%** for passive investment. At the same time, it also reduced volatility drastically (**14.17%** vs **31.53%**). The strategy consistently avoided the major drawdowns seen in the passive approach, while still catching the upward movements.

Final Words and Next Steps

This project demonstrates the potential of using machine learning to in stock trading and that it is possible to outperform traditional passive investment strategies.By combining predictive machine learning models with our dynamic investment function as a risk management tool, we were able to build and trade a portfolio that not only delivered strong cumulative returns but also managed risk effectively, especially in volatile market conditions.

While not every individual stock outperformed its passive investment alternative in terms of return, the overall portfolio showed consistent value creation and lower drawdowns compared to the market benchmark. The results from our project suggest that algorithmic trading strategies based on machine learning models can be a good complement or even an alternative to passive investing.

However, it's important to acknowledge that a significant portion of the portfolio's gains may not have come solely from the machine learning models themselves, but rather from favorable market conditions. The period under analysis was largely bullish, especially for high-growth stocks like Nvidia, which benefited from strong market momentum. In such environments, even moderately effective strategies can appear successful. Still, classifying the companies when building the portfolio and selecting Nvidia as one of them was because of the unsupervised learning method used. Our model still proved good in making returns and especially in reducing downside risk.

As next steps, several improvements could help strengthen the strategy's effectiveness. First, including additional predictive variables such as macroeconomic indicators, sector-specific trends, sentiment indicators and more technical oscillators and signals could provide the models with more context and improve their ability to detect meaningful patterns. Feature selection techniques and dimensionality reduction may also help eliminate noise and focus the models on the most relevant information. We could also explore more sophisticated machine learning algorithms, such as the XGBoost, which could lead to better predictive precision. Another important step would be to integrate trading costs and slippage for more realistic net return estimates and evaluate strategy performance across different market environments.

Appendix

Risk-reward ratio

The risk-reward ratio is a financial metric used to evaluate the risk-adjusted return of an investment. It shows how much excess return an investor earns for each unit of risk taken. Essentially it helps determine whether the returns of a portfolio are due to careful investment decisions or simply taking on more risk.

To calculate the risk-reward ratio we simply divide the expected return by the volatility of a stock.

Simple Moving Averages (SMAs)

A simple moving average is the average price (usually closing price, as in this project) of an asset for a specific period of time.

$$\text{SMA}_t = \frac{1}{n} \sum_{i=0}^{n-1} P_{t-i}$$

Where:

SMA_t = Simple Moving Average at time t
n = number of periods (e.g., 4-day, 6-day, 135-day)
P_{t-i} = price at time $t - i$

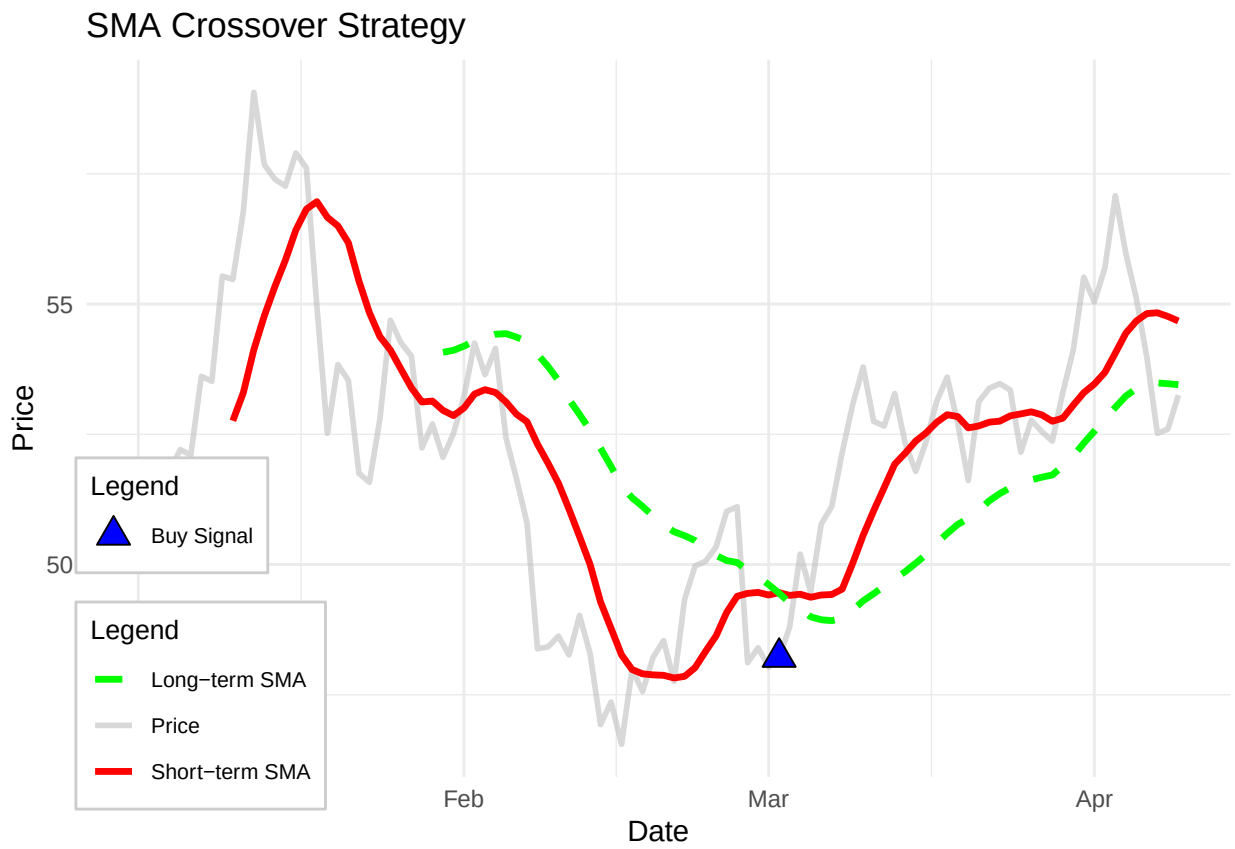
For example, one could add the closing price of a security for 5 consecutive days and then divide this total by that same number of days - this is a 5-day SMA. Short-term averages respond quickly to changes in the price of the underlying security, while long-term averages are slower to react.

However, we use SMAs that are lagged by one day to avoid bias. Lagging an SMA means shifting it backward in time so that its value at any point reflects only past information. In this context, lagging the SMAs by one day ensures that today's prediction uses only data available up to yesterday, preventing the model from "seeing the future." This avoids data leakage and maintains the integrity of the predictive analysis.

Buy signal (SMA crossover)

We implement buy signals using a well-established method in technical analysis known as the Simple Moving Average (SMA) crossover strategy. This strategy is based on comparing two SMAs of the same asset but calculated over different time horizons - typically, one short-term and one long-term. The key idea is that changes in the relative position of these two averages can indicate shifts in market momentum.

Specifically, a buy signal is generated when the short-term SMA (which reacts more quickly to recent price changes) crosses above the long-term SMA (which smooths out longer-term trends). This crossover suggests that recent price movement is stronger than the longer-term trend, which hints toward potential upward price movement.



To incorporate this logic into our model without introducing future information, we store the signal in a lagged binary variable. This variable takes a value of 1 (a buy signal) if the crossover occurred on the previous day , and 0 otherwise. Again, lagging ensures that the model only uses information that would have been known at the time of prediction.

Investment function

To make our trading strategy more intelligent and risk-aware, we designed an investment function that determines how much of the company's capital to invest on any given trading day for each stock. This function takes into account the predictions of three models—logistic regression, decision tree, and random forest—and weighs them by their precision scores.

Each prediction model provides a binary prediction: 1 if it predicts the stock will go up, and 0 otherwise. However, since not all models are equally reliable, we first evaluate their precision scores on validation data and assign them a rank - 1 for most precise, 2 for second most precise, and 4 for least precise. These ranks reflect how much trust we place in each model's predictions. We will clarify the choice of using 4 instead of 3 for the rank of the least precise model [1] after providing an example of how the function works.

Let us assume that for a given stock, each model (logit, decision tree and random forest) has generated predictions and we have calculated the precision scores:

Best Model	Precision
Logit	0.520
Decision Tree	0.550
Random Forest	0.517

We assign a rank for these precision scores as explained above:

Best Model	**Precision**	**Rank**
Logit	0.550	1
Decision Tree	0.520	2
Random Forest	0.517	4

We then take all possible combinations of 1 and 0 for each predictor model, as shown below:

Combinations

Logit	Decision Tree	Random Forest
0	0	0
1	0	0
0	1	0
0	0	1
1	1	0
1	0	1
0	1	1
1	1	1

For instance, *1 1 0* means that the Logit and Decision Tree models have predicted that the stock price will increase on the next day, while the Random Forest has predicted it will not increase.

Combinations matrix

We store these combinations in a *8x3* matrix.

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Rankings vector

We also store the rankings that we assigned to the models based on precision as a 3x1 vector.

$$\begin{bmatrix} 1 \\ 2 \\ 4 \end{bmatrix}$$

Now we will perform a typical matrix multiplication in order to find the score of each combination's strength, using the transpose of the *rankings vector*:

$$\begin{bmatrix}1 & 2 & 4\end{bmatrix} \times \begin{bmatrix}0 & 0 & 0\\ 1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\\ 1 & 1 & 0\\ 1 & 0 & 1\\ 0 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} = \begin{bmatrix}0 & 1 & 2 & 4 & 3 & 5 & 6 & 7\end{bmatrix}$$

We take the transpose of this resulting vector:

$$\begin{bmatrix}0\\ 1\\ 2\\ 4\\ 3\\ 5\\ 6\\ 7\end{bmatrix}$$

This way we can see how each combination scores by its precision. 0 is the lowest rank - no model predicts Buy , 7 the highest - all models predict Buy.

$$\begin{bmatrix}0 & 0 & 0\\ 1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\\ 1 & 1 & 0\\ 1 & 0 & 1\\ 0 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} \quad \begin{bmatrix}0\\ 1\\ 2\\ 4\\ 3\\ 5\\ 6\\ 7\end{bmatrix}$$

For example, 3 identifies the combination 1 1 0 and 4 identifies the combination 0 0 1. As 1 1 0 indicates a stronger buy signal, 3 is considered stronger score than 4

For each rank we assign a percentage - the percentage of the the company's balance that will be used for the particular trade. For example, if the models predict 0 1 1 for Apple with current balance of $2750 (rank 6), 45% of the current stock's balance ($1237,5) will be used to buy the stock on the current day and sell it on following day.

Rank	**Percentage**
0	0%
1	40%
2	37%
4	35%
3	50%
5	47%
6	45%
7	60%

We now have the complete framework for determining the investment amount for each case in the example. We can examine how each prediction combination is weighted accordingly and ensure that the combination-percentage matching works properly.

Logit	Decision Tree	Random Forest	Rank	Percentage
0	0	0	0	0%
1	0	0	1	40%
0	1	0	2	37%
1	1	0	3	50%
0	0	1	4	35%
1	0	1	5	47%
0	1	1	6	45%
1	1	1	7	60%

We remind that Logit, Decision Tree and Random Forest are ranked in that precise order based on their precision. For instance, we can see that the algorithm invests more if the combination of predictions is 1 0 0 (40% of balance) and less if the combination is 0 1 0 (37%), because the Logit model has higher precision than the Decision Tree model.

Analogically, if the predictions are 110, more is invested (50%) than if the predictions were 101 (47%).

Note: a different ranking of the precision scores (e.g. 4 for Logit, 1 for Decision Tree and 2 for Random forest) would yield a different order of the rankings within the 8x1 vector. However, the percentage scores are always fixed to the same ranking (e.g. 40% for 1, 37% for 2, etc.)

[1] We use the number 4 instead of 3 to identify the model with the least precision because of the rules of matrix multiplication.

If we use ranking order

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

and we multiply the rankings vector by the 4th row in the combinations matrix

$$\begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = 3$$

we get as a result the scalar 3.

However, if we multiply the rankings vector by the 5th row

0 0 1 and 1 1 0

$$\begin{bmatrix} 1 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = 3$$

We get the scalar 3 again. This will create a problem because this way we will not be able to rank the combinations of 0 and 1 correctly. Both the combinations 0 0 1 and 1 1 0 will yield the same rank, however the latter is a stronger one because it means that two models have predicted Buy.

To avoid this problem, use use 4 to identify the lowest ranking model by precision. Using 4 will not create identical rankings for different combinations of 0 and 1.