Q_Learning_Presentation

Davide

2025-03-12

Reinforcement Learning for Algorithmic Trading: A Q-Learning Approach to Strategy Optimization

Github Link: https://github.com/DavideDevetak24/Reinforcement-Learning-for-Algorithmic-Trading-A-Q-Learning-Approach-to-Strategy-Optimization

1. Introduction

In financial markets, traders are constantly seeking strategies to maximize returns while managing risk. Traditional approaches rely on historical price patterns, economic indicators, and company performance metrics to make trading decisions. However, financial markets are highly complex and achieving consistently above-average-returns is challenging.

Especially in the last decade, many investment firms switched to more mathematical approaches to generate higher returns, especially leveraging the breakthroughs in Machine Learning or Artificial Intelligence in general.

Reinforcement Learning (RL), a sub-field of AI, offers a data-driven approach to developing trading strategies capable of adapting in the market environment. Unlike traditional methods, RL does not rely on predefined rules but instead learns optimal decision-making policies through interaction with the market environment. By continuously updating its strategy based on rewards and penalties, the algorithm is capable of identifying patterns and improve its trading decisions over time.

In this study, I create a Q-Learning-based trading strategy using Apple Inc. (AAPL) time series, leveraging the widely used technical indicators Bollinger Bands and MACD (Moving Average Convergence/Divergence). The goal of the study is to train an RL agent to autonomously generate buy and sell signals, optimizing trading strategy performance by maximizing cumulative returns. The results then will be tested against a traditional "Buy and Hold" approach and a purely technical trading strategy.

2. Libraries and Data

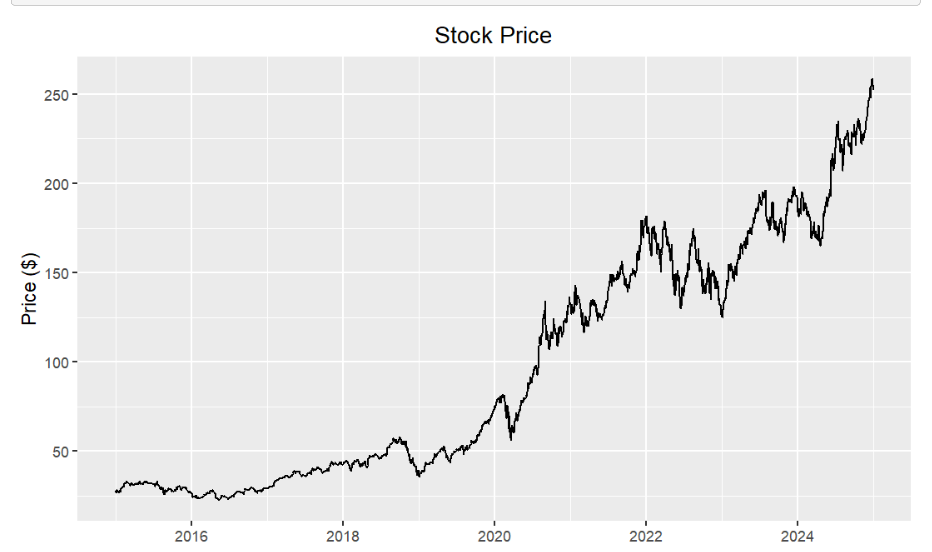
To perform the proposed problem, I start by calling the libraries needed for the computations. Tidyverse, data.table and tidyquant are used to perform operations on datasets, data frames and downloading market data. TTR is a popular library used to compute technical indicators, which I used to create Bollinger Bands and MACD Indicator. Foreach, doParallel and parallel are used for parallel computations in the train part.

```r
set.seed(42)
options(warn=-1)
library(tidyverse, warn.conflicts = FALSE)
library(tidyquant, warn.conflicts = FALSE)
library(TTR, warn.conflicts = FALSE)
library(data.table)
library(foreach)
library(doParallel)
library(parallel)
```

As for the data, I used 10 year worth of Adjusted Close of Apple Inc. stock, ranging from 2014-12-31 to 2024-12-31. The dataset I used comes from Yahoo Finance, for which I used the library tidyquant.

Below the chart with the time series used.

```r
knitr::include_graphics("Images/Stock_price.png")
```



3. Technical Strategy

The Technical Trading Strategy combines two widely-used indicators: Bollinger Bands and MACD. The key idea is to generate trading signals using their ability to catch price volatility and momentum. By combining the two indicators the strategy aims to capture possible points of trend reversal.

Bollinger Bands (BB)

I calculate BBs using a 10-day moving average as the middle band. The upper and lower bands are at 2 standard deviations away from this MA.

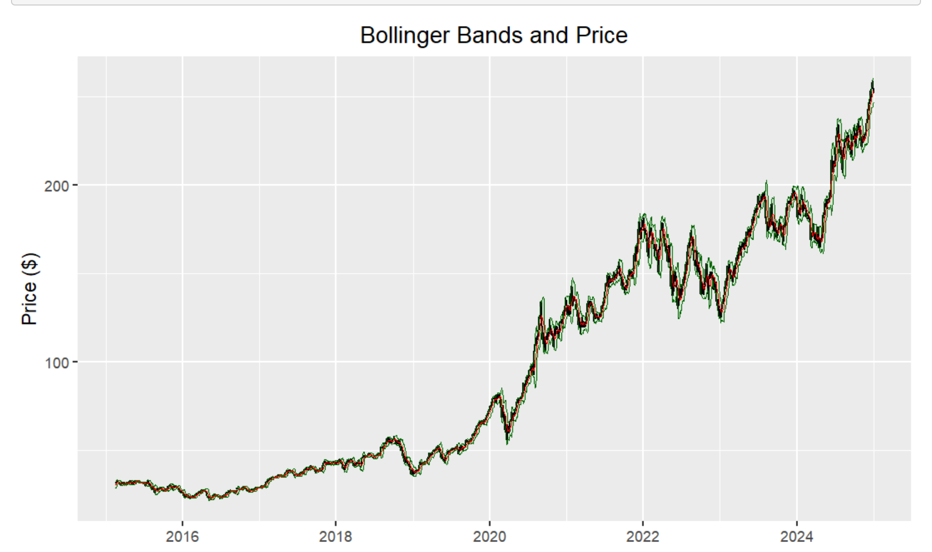
The interpretation is that when the Adj. Close price falls below the lower band, the asset may be considered oversold, thus signaling a potential buying opportunity. Conversely, when the price exceeds the upper band, it may be overbought, suggesting selling opportunity.

I decided to set the bands at 2 std away from the MA to generate signals that are distinct from noise while ensuring that the constraints are not too strict, allowing for a reasonable number of signals.

Below the code used to create the BB (TTR library) and the graphical representation of the BB.

```r
# Creation of Bollinger Bands
aapl_bbands <- BBands(df$close, n = 10,sd = 2)
```

```r
knitr::include_graphics("Images/BB_Price.png")
```



MACD Indicator

I construct the MACD (Moving Average Convergence/Divergence) by subtracting a 24-day slow exponential moving average from a 12-day fast exponential moving average. The 9-day signal line is computed to smooth out the MACD values. wilder=FALSE, indicates that I don't use Wilder's smoothing.

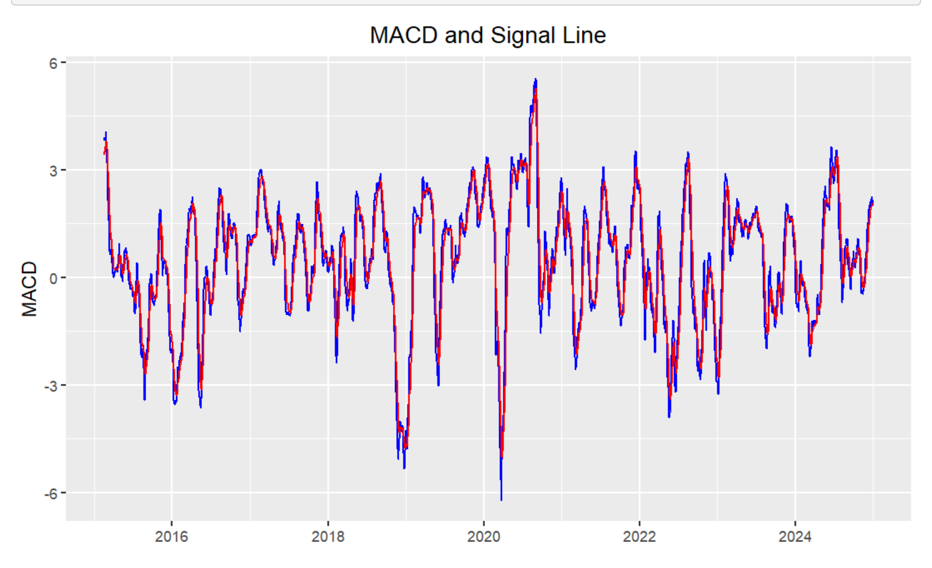
MACD = EMA_24 - EMA_12 Signal Line = EMA_9(MACD)

The MACD is useful to identify momentum changes. A MACD value that crosses above the signal line suggests bullish momentum, while a crossing below indicates bearish momentum.

Below the code used to compute MACD (TTR library) as well as its graphical representation.

```r
# Creation of MACD indicator
aapl_macd <- MACD(df$close, nFast = 12, nSlow = 24, nSig = 9, wilder=FALSE)
```

```r
knitr::include_graphics("Images/Macd_Signal.png")
```



Combining Bollinger Bands and MACD

To generate the trading signals I use the Bollinger Bands to identify overbought or oversold conditions and then combine them with MACD's momentum signals. The combination of teh two helps to filter out false signals.

I created two binary columns, one for the buy signals, the other for sell signals. If the stock appears to be oversold (price below BB) and MACD is above its signal line (bullish momentum) the algorithm triggers a buy signal, as this combination indicates a possible trend reversal. If the stock appears to be overbought (price above BB) and MACD is below its signal line (bearish momentum) the algorithm triggers a sell signal, as this combination indicates a bearish trend reversal.

Below the code used to generate the signals.

```r
# Creation of Trading Signals based on the behavior of Bollinger Bands and MACD Indicator
aapl_data$Buy_Signal <- ifelse((aapl_data$close <= aapl_data$bb_lower) &
                                 (aapl_data$macd > aapl_data$signal), 1, 0)
aapl_data$Sell_Signal <- ifelse((aapl_data$close >= aapl_data$bb_upper) &
                                  (aapl_data$macd < aapl_data$signal), 1, 0)
```

Technical Strategy

In order to use the trading signals generated I created a trading strategy. When the algorithm receives a trading signal it opens a long (1) or short (-1) position the day after and holds the position for ten days. I decided to make the algorithm to hold the position for more days to exploit the momentum effect given by the trend reversal. If an opposite signal is generated the algorithm immediately closes the open position and if there's no signal in the last ten days the position remains flat (0).

Below the algorithm for deciding the trading position and the strategy results.

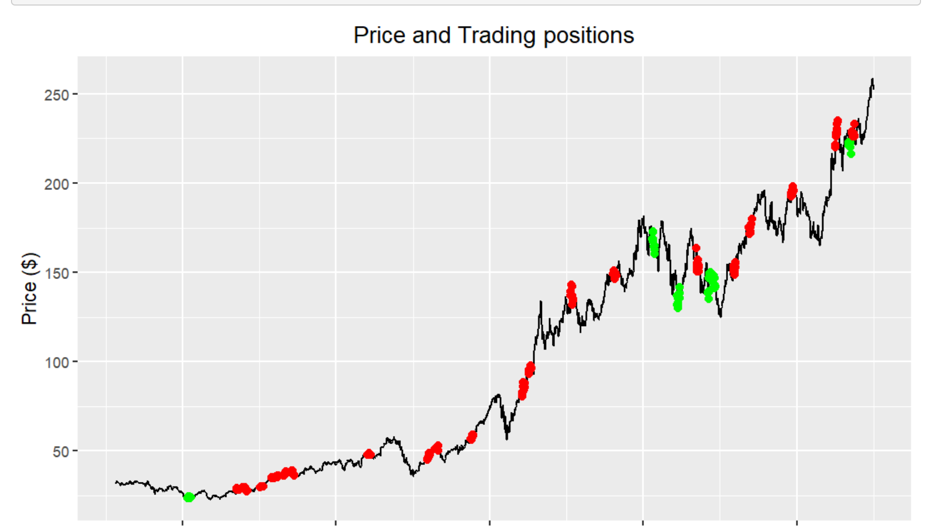
```r
# Trading Strategy setup
aapl_data$Position <- 0 #I set the initial position to zero (in other words "Neutral" position)
max_hold_days <- 10 #The strategy will hold the position for 10 days

current_position <- 0 #Needed to understand if the strategy is long (1), short (-1) or neutral (0)
days_in_position <- 0 #Counter for how long the strategy holds a position

#Algorithm for the creation of the trading position
#The idea is to maintain the current position unless there's a different signal or threshold that indicates other
wise
for(i in 1:nrow(aapl_data)){
  #Increment days_in_position if I'm holding a long or short position
  if(current_position != 0) {
    days_in_position <- days_in_position + 1
  }
  #Based on the trading signals I have 3 options: open a new position, maintain the position, close the position
  #If the current position is zero and I have a buy or sell signal, the algorithm takes a long or a short positio
n
  if(current_position == 0) {
    if(aapl_data$Buy_Signal[i] == 1) {
      current_position <- 1
      days_in_position <- 0 #The counter days_in_position is set to zero if a directional position is taken
    } else if(aapl_data$Sell_Signal[i] == 1) {
      current_position <- -1
      days_in_position <- 0
    }
    #If the current position is different from zero, I have two choices: maintain the position or close the posit
ion. I close the position if I have a contrary signal or I surpass max_hold_days
  } else {
    if(current_position == 1 && (aapl_data$Sell_Signal[i] == 1 || days_in_position >= max_hold_days)) {
      current_position <- 0
      days_in_position <- 0
    }

    if(current_position == -1 && (aapl_data$Buy_Signal[i] == 1 || days_in_position >= max_hold_days)) {
      current_position <- 0
      days_in_position <- 0
    }
  }
  #With each iteration I am creating a vector in which I synthesize the positions in the strategy
  aapl_data$Position[i] <- current_position
}
```

```r
knitr::include_graphics("Images/Trading_positions.png")
```




```r
knitr::include_graphics("Images/Tech_Strat_Return.png")
```

4. Q-Learning Intro

Q-Learning is a model-free Reinforcement Learning (RL) algorithm designed to enable an agent to learn an optimal policy through interaction with its environment. Q-Learning is different from standard Supervised Learning as it does not rely on labeled input-output pairs. Instead, the agent learns by trial and error, optimizing its decision policy to maximize the cumulative reward over time.

A key difference between Q-Learning and other RL approaches is that its model-free nature does not require a predefined model of the environment's dynamics, like the Markov Decision Process (MDP). Instead, Q-Learning relies on Temporal Difference (TD) learning, which allows the agent to update its value estimates based on new observations without needing to know the full transition probabilities of the system.

At the core of Q-Learning is the Q-value function, which estimates the expected future rewards for taking a given Action in a particular State. This function is updated iteratively using the Bellman Equation, which defines the relationship between current rewards and future expected rewards.

(Sources in part 8.)

```r
knitr::include_graphics("Images/Bellman_eq.png")
```

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]. \quad (6.6)$$

```r
#Reinforcement Learning: An Introduction, Richard S. Sutton and Andrew G. Barto, ch.6 eq.6.6
```

5. Q-Learning Trading Strategy

In the first part I create the states by combining the conditions obtained above with the creation of the trading signals and technical indicators. The rationale behind it is to let the agent take an autonomous trading actions based on the market states.

Then I divided the dataset into a training and a test one. The two periods I used are 2014-12-31/2022-12-31 and 2023-01-01/2024-12-31 respectively.

```r
#I create the State column which represents the environment of the trading strategy, then I create the possible a
ction of the trading strategy
#State combines Bollinger Bands, MACD, trading signals and trading positions into a single categorical variable
aapl_data$State <- paste0(
  ifelse(aapl_data$close < aapl_data$bb_lower, "Below_BB",
         ifelse(aapl_data$close > aapl_data$bb_upper, "Above_BB", "Middle_BB")),
  "_",
  ifelse(aapl_data$macd > aapl_data$signal, "MACD_Bullish", "MACD_Bearish"),
  "_",
  ifelse(aapl_data$Buy_Signal == 1, "Buy",
         ifelse(aapl_data$Sell_Signal == 1, "Sell", "No_Signal")),
  "_",
  ifelse(aapl_data$Position == 1, "Long",
         ifelse(aapl_data$Position == -1, "Short", "Flat"))
)

actions <- c("Long", "Short", "Flat")

#the Q-learning model will choose the best action based on the state
```

Then I created the Q-Table, a matrix defined by m states * n actions. The Q-table is initialized with all values equal to zero: during the training phase the values will be updated in order to make the agent take the best trading actions.

In the second part of the chunk, the hyperparameters used in the training phase and a brief description.

```r
#Creation of the Q-table where rows: states, columns: actions. Initialization with all values equal to zero
Q_table <- data.table(State = unique(df_train$State))
for (a in actions) {
  Q_table[[a]] <- 0
}

#Optimize research in the q-table by setting State as key
setkey(Q_table, State)

#Hyperparameters
alpha <- 0.1   #learning rate: how much info overrides the Q-Table
gamma <- 0.5   #discount factor: long term vs short term rewards
epsilon <- 1 #casual exploration
epsilon_decay <- 0.999 #decay of epsilon as time passes
min_epsilon <- 0.01 #minimum level of epsilon
```

The reward function is designed to give the agent a reward based on its trading actions. In general, a good action should be rewarded, while a bad one should be penalized. To capture both the correctness and the magnitude of the action, I decided to use the daily return at t+1 as reward, based on the action taken.

If the agent goes Long and the daily return is negative, this will be its penalty, and vice versa. Similarly, if the agent goes Short and the daily return is positive, the penalty will be -(daily return).

In the first versions of the model, the agent tended to avoid taking a trading position to minimize losses, making the performance relatively flat compared to the benchmark. To avoid this problem, I decided to heavily penalize being Flat and not taking a position. The performances drastically increased. However, it is important to note that an increase in trading actions, also also leads to higher trading fees. Therefore, this aspect should be considered in real investment decision, especially when operating with limited capital.

```r
#Create the reward function
reward_function <- function(df_row, action) {
  if (action == "Long") {
    return(df_row$daily_return)
  } else if (action == "Short") {
    return(-df_row$daily_return)
  } else {
    return(-1) #Agent penalized for inactivity
  }
}
```

The following chunk of code represents the core of the entire project, where I build the function that integrates the agent, the state, and the actions to create the Q-Tables needed for generating trading decisions.

The function is divided into an inner and an outer part. The outer part is necessary for the parallel computation explained below. Here, I define the seed, local copies of the Q-Table, epsilon, and the training dataframe. This procedure ensures that the variables are not overwritten each time the cycle restarts.

The inner function is a for loop. For each row in the dataset (except the last one), I save the current state and the next state. Then, based on the results of the action chosen by the Exploration/Exploitation algorithm and the consequent reward, I combine all this information into the Bellman Equation. This equation updates the weights of the Q-Table, refining the decision-making capabilities of the agent.

Finally, the output consists of the final Q-Table in data.table format generated by the process.

```r
#Training of the Q-Learning algorithm
training <- function(j, Q_table, df_train, actions, alpha, gamma, epsilon, min_epsilon, epsilon_decay) {

  set.seed(42 + j)
  #Copies of the df, q_table and epsilon
  Q_table_local <- copy(Q_table)
  df_train_local <- copy(df_train)
  local_epsilon <- epsilon

  for (i in 1:(nrow(df_train_local) - 1)) {
    #Definition of the states
    current_state <- df_train_local$State[i]
    next_state <- df_train_local$State[i + 1]

    #Exploration vs Exploitation: the agent explores a random action with probability epsilon, otherwise the agen
t selects the best action based on the current state. As time goes by, epsilon decays. This choice was made to ma
ke the model explore more at the beginning
    if (runif(1) < local_epsilon) { #runif(1) creates a random number between 0 and 1
      action <- sample(actions, 1) #sample chooses a single random action
    } else {
      action <- actions[which.max(Q_table_local[J(current_state), ..actions])] #if runif(1)>epsilon, the agent ch
ooses the best action
    } #action returns on of the actions chosen

    #Function defined above that takes as reward the return of i+1 based on the agent's action
    reward <- reward_function(df_train_local[i+1, ], action)
    #Select best future action
    best_future_q <- max(Q_table_local[J(next_state), ..actions], na.rm = TRUE)

    # Bellman Equation
    Q_table_local[J(current_state), (action) := (1 - alpha) * get(action) + alpha * (reward + gamma * best_future
_q)] #get(action) returns the value on the Q_table that is linked to that action given a state

    #Update of epsilon
    local_epsilon <- max(min_epsilon, local_epsilon * epsilon_decay)
  }

  #The function when iterated will return a list with the Q-tables
  return(Q_table_local = Q_table_local)
}
```

After defining the function, I iterated it 100 times using parallel computing to reduce execution time. I followed this procedure to ensure a fair representation of the results, as a single training phase could yield inaccurate outcomes. All the Q-Tables are saved into the results variable and then used to compute the final Q-Table considering the average value for each combination of State-Action.

Below is the code used to compute the function in parallel and generate the resulting Q-Table.

```r
#Number of iterations
iter <- 100

#Initialization for parallel computation
cores <- detectCores()
cl <- makeCluster(cores - 1)
registerDoParallel(cl)

#Time counter
start_time <- Sys.time()

#Results stored in a list. Added .packages because the function training uses package data.table
results <- foreach(j = 1:iter, .packages = c("data.table")) %dopar% {
  training(j, Q_table, df_train, actions, alpha, gamma, epsilon, min_epsilon, epsilon_decay)
}

stopCluster(cl)

end_time <- Sys.time()
print(end_time - start_time)
```

```r
knitr::include_graphics("Images/Q_Table.png")
```

Description: dt [14 x 4]

State <chr>	Long <dbl>	Short <dbl>	Flat <dbl>
Above_BB_MACD_Bearish_Sell_Flat	0.0002733898	4.239607e-05	-0.01894456
Above_BB_MACD_Bearish_Sell_Short	-0.0036088584	2.809833e-04	-0.26048511
Above_BB_MACD_Bullish_No_Signal_Flat	0.0053842728	-2.059403e-03	-0.64282971
Above_BB_MACD_Bullish_No_Signal_Short	-0.0010027795	-2.505374e-03	-0.17230236
Below_BB_MACD_Bearish_No_Signal_Flat	-0.0026357774	-9.144812e-03	-0.70952238
Below_BB_MACD_Bearish_No_Signal_Long	0.0066628449	-2.682721e-04	-0.01587625
Below_BB_MACD_Bearish_No_Signal_Short	-0.0009885599	1.485454e-03	-0.09764133
Below_BB_MACD_Bullish_Buy_Long	-0.0030778260	2.208058e-03	-0.07088258
Middle_BB_MACD_Bearish_No_Signal_Flat	-0.0088921450	3.172053e-03	-0.99994872
Middle_BB_MACD_Bearish_No_Signal_Long	0.0008617262	-5.567038e-03	-0.16972366

1-10 of 14 rows   Previous 1 2 Next

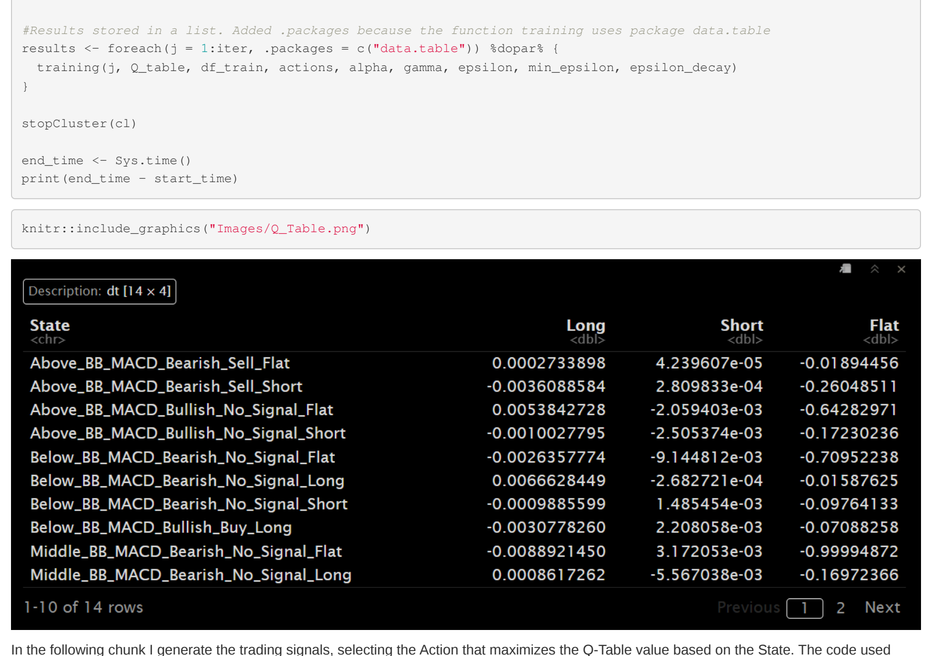
In the following chunk I generate the trading signals, selecting the Action that maximizes the Q-Table value based on the State. The code used refers to the train data, but the same chunk structure is used also for test data.

In the images below the results of the RL strategy, compared to the "Buy and Hold" and the Technical Strategy, for both the train and test period.

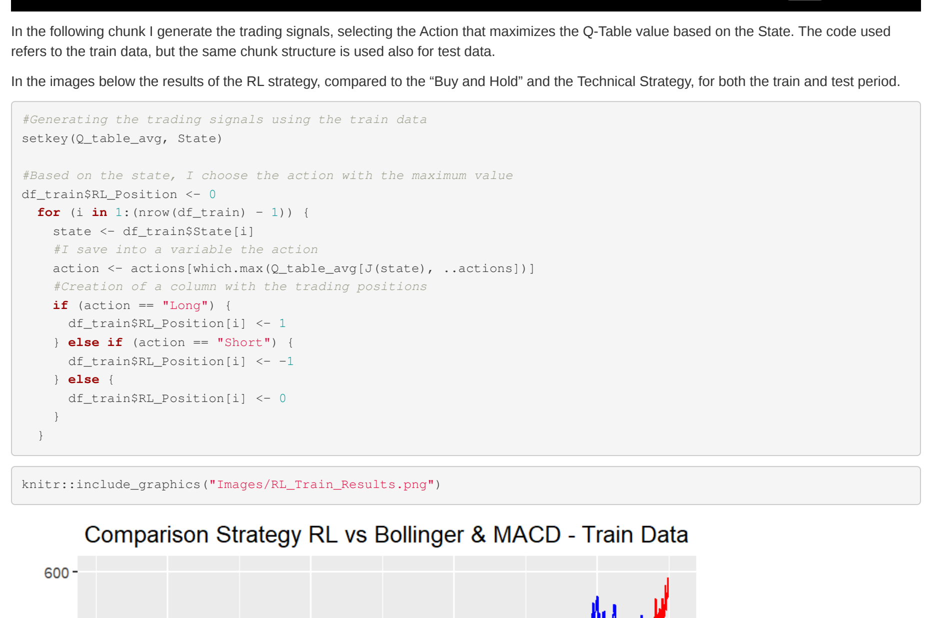
```r
#Generating the trading signals using the train data
setkey(Q_table_avg, State)

#Based on the state, I choose the action with the maximum value
df_train$RL_Position <- 0
  for (i in 1:(nrow(df_train) - 1)) {
    state <- df_train$State[i]
    #I save into a variable the action
    action <- actions[which.max(Q_table_avg[J(state), ..actions])]
    #Creation of a column with the trading positions
    if (action == "Long") {
      df_train$RL_Position[i] <- 1
    } else if (action == "Short") {
      df_train$RL_Position[i] <- -1
    } else {
      df_train$RL_Position[i] <- 0
    }
  }
```

```r
knitr::include_graphics("Images/RL_Train_Results.png")
```




```r
knitr::include_graphics("Images/RL_Test_Results.png")
```



6. Results

I decided to evaluate the strategies using four different criteria and risk measures: overall cumulative return, daily standard deviation, alpha, and Sharpe ratio.

For completeness, I define alpha as the difference between the daily average strategy return and the daily average benchmark (in this case Apple) return. The Sharpe ratio, on the other hand, is the difference between the average daily strategy return and the risk-free rate, divided by the standard deviation of the strategy. In simple terms, it compares the return of an investment with its risk.

Alpha = Strategy Return - Benchmark Return

Sharpe Ratio = (Strategy Return - Risk Free Rate)/Std Strategy

I used as risk free rate the average 10y bond yield in the selected period.

The results are presented for Train_Data and Test_Data separately.

```r
knitr::include_graphics("Images/Performance_Results.png")
```

Description: df [10 x 3]

Metric <chr>	Train_Data <dbl>	Test_Data <dbl>
Cumul AAPL Return	4.0657119867	1.941045264
Cumul Tech Strat Return	0.9285374625	0.875394327
Cumul RL Return	5.7470411935	1.140866414
AAPL Std	0.0188548052	0.013557149
Tech Strat Std	0.0055254885	0.004133433
RL Std	0.0188451734	0.013523370
Alpha Tech Strat	-0.0009075956	-0.001673183
Alpha RL	0.0001740548	-0.001061590
Sharpe Tech Strat	-0.0164417155	-0.078784602
Sharpe RL	0.0525758985	0.021144262

1-10 of 10 rows

As can be seen from the results, the "Buy and Hold" strategy consistently outperforms the Technical Trading Strategy, which, in both analyzed periods, severely underperforms even the Q-Learning Strategy. This result is also evident in the other measures: the Technical Strategy has a negative Alpha and a negative Sharpe Ratio in both cases. However, it appears to exhibit lower volatility (low standard deviation).

On the other hand, the Q-Learning Strategy performed very well compared to the "Buy and Hold" strategy when evaluated on the Train Data (+474.70% vs +306.57%). However, it underperformed in the Test period: the return is one-sixth of the "Buy and Hold" strategy, and the Sharpe Ratio is nearly half of that in the previously analyzed period (0.0526 vs 0.0211).

7. Conclusion and Further Research

The study overall showed good results. The Reinforcement Learning Strategy appears to yield returns consistently above the ones of the Technical Strategy. As for the "Buy and Hold" of the stock, the results are mixed but promising. The RL, despite beating the "Buy and Hold" in the train data, it failed to return higher performances with the test data. However, if we consider only the first few months the RL returned more or less the same performances of the Buy and Hold. This could suggest that the Q-Table maintains its edge only for a limited period, but in the long run could underperform. A strength of the RL is that we can continue to update the model (in this project the Q-Table) using past data. By creating n subsets in the test data, we could evaluate the performance for a meaningful and limited time and then update the Q-Table. In our case registering the performances of the RL strategy every two months and then re-updating the Q-Table, could be a winning implementation, as it could reflect the most recent market conditions and dynamics rather then distant ones.

Another idea to improve the strategy could be removing the outiler Q-Tables generated in the training cycle. In fact, some of them appeared to provide an exponential result when applied again to the train data: this could indicate a problem of overfitting. The same can be said with the problem of underfitting, as some outlier Q-Tables yielded a total loss of the capital when used on training data. These extreme result might affect the final Q-Table, so a deepen analysis on this topic could be carried out.

Finally, with adequate time and computational power, an analysis on the best reward function could be addressed. With this approach we can understand if penalizing more or less an action could yield a winning performance.

It can be said that the study showed good and promising results and thanks to these the topic could be further explored and analyzed.

8. Bibliography

1. Tsitsiklis, J.N. Asynchronous Stochastic Approximation and Q-Learning. Machine Learning 16, 185–202 (1994). https://doi.org/10.1023/A:1022689125041
2. Watkins, C.J., Dayan, P. Technical Note: Q-Learning. Machine Learning 8, 279–292 (1992). https://doi.org/10.1023/A:1022676722315
3. Jagdish Bhagwan Chakole, Mugdha S. Kolhe, Grishma D. Mahapurush, Anushka Yadav, Manish P. Kurhekar, A Q-learning agent for automated trading in equity stock markets, Expert Systems with Applications, Volume 163, 2021, 113761, ISSN 0957-4174, https://doi.org/10.1016/j.eswa.2020.113761.
4. Watkins, Christopher. (1989). Learning From Delayed Rewards.
5. Reinforcement Learning: An Introduction, Richard S. Sutton and Andrew G. Barto, 2014, 2015, The MIT Press
6. Machine Learning for Algorithmic Trading, Stefan Jansen, Edition 2, Publisher Packt Publishing, 2020, ISBN 1839217715, 978183921771