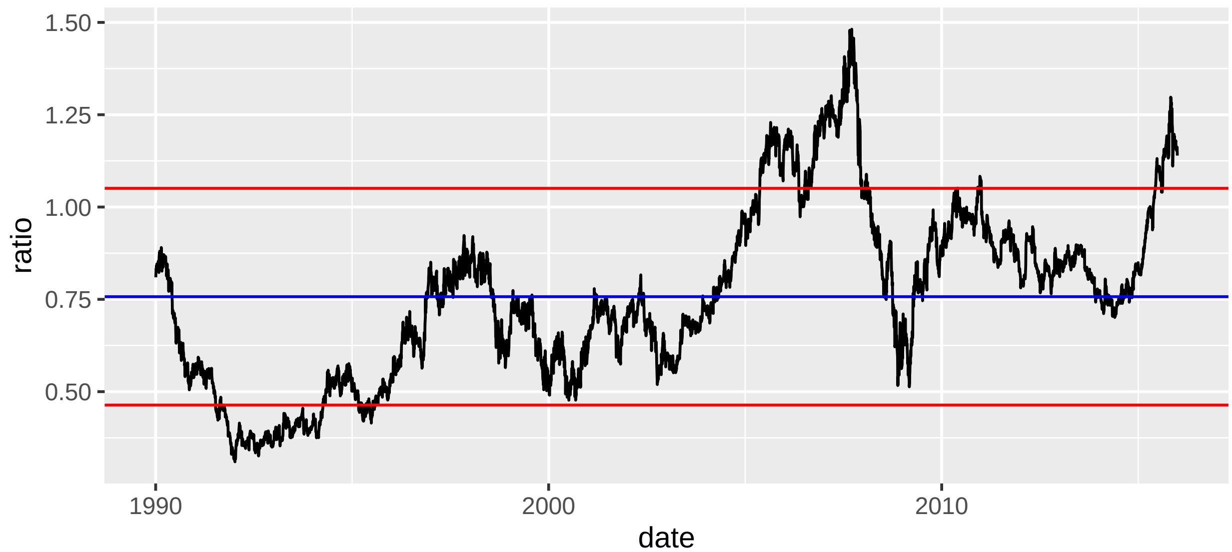
Adv R Closing Assignment

Mohamadreza Hoseinpour

November 4th, 2019

Prologue: Getting a sense of Data

In this section, I created the example data and the graph based on which I developed the required functions for the rest of the questions.



Question 1.a) Developing the Function for "Finding Next Position"

```r
#' Title: Find Next Position
#'        Which finds the opening and closing indices of the first new position
#'        on or after a given starting day.
#' @param ratio: The daily price ratio for the two stocks (a numeric vector).
#' @param starting_from: Index of the first day on which the next position could
#'                       be opened (default value = 1).
#' @param k: The value of "k" for calculating the decision boundaries (m - k * s)
#'           and (m + k * s) (default value = 1).
#' @param m: The estimated mean ratio m in the formula for the boundaries
#'           (default value = the mean of "ratio").
#' @param s: The standard deviation s of the price ratio, used in the formula for
#'           the boundaries (default value = the standard deviation of "ratio").
#' @return: An integer vector of length two containing the indices of "ratio"
#'          where the next position should be opened and closed. In case no
#'          position is found, it returns a length zero integer vector.
#' @export
#'
#' @examples
find_next_position <- function(ratio, starting_from = 1, k,
                               m = mean(ratio), s = sd(ratio)) {
  #' Turn the input into a dataframe and make sure that starting_from is
```

```r
  #'   stored as a number when I want the next positions in the next iterations.
  data <- as.data.frame(ratio)
  starting_from <- as.numeric(starting_from)
  #' A condition to check whether the "k" is too big or not. If it is, then an
  #'   empty integer vector.
  if ((m - k * s) <= min(data) & (m + k * s) >= max(data)) {
    return(integer())
  } else {
    #' The opening positions will be called and stored based on the criteria.
    open <- which(data[starting_from:nrow(data), ] <= (m - k * s) |
      data[starting_from:nrow(data), ] >= (m + k * s))[1] +
      (starting_from - 1)
    #' If the open is not found, then it will return an empty integer vector.
    if (is.na(open)) {
      return(integer())
    } else {
      #' The closing posiiton will be called based on the criterai that the open
      #'   was below or above the mean.
      close <- if (data[open, ] < m) {
        which(data[open:nrow(data), ] >= m)[1] + open - 1
      } else {
        which(data[open:nrow(data), ] <= m)[1] + open - 1
      }
      #' This is the condition for checking whether the function has reached the
      #'   end of the dataframe.
      if (is.na(close)) {
        close <- nrow(data)
      }
      c("open" = open, "close" = close)
    }
  }
}
```

Question 1.a) Testing the Function

To test the function, I used the retail_stocks mentioned in the assignment.

```
##  open close
##   387  1743
```

```
## # A tibble: 2 x 4
##   date         TGT   WMT ratio
##   <date>     <dbl> <dbl> <dbl>
## 1 1991-07-12  3.51  7.71 0.455
## 2 1996-11-19  6.36  8.38 0.759
```

```
##  open close
##  3878  4673
```

```
## # A tibble: 2 x 4
##   date         TGT   WMT ratio
##   <date>     <dbl> <dbl> <dbl>
## 1 2005-05-17  36.2  33.9 1.07
## 2 2008-07-15  32.3  42.8 0.756
```

```
##   open close
##   5144  6029
```

```
## # A tibble: 2 x 4
##   date         TGT   WMT ratio
##   <date>     <dbl> <dbl> <dbl>
## 1 2010-05-27  42.3  40.2 1.05
## 2 2013-12-02  51.8  69.7 0.742
```

```
##   open close
##   6411  6553
```

```
## # A tibble: 2 x 4
##   date         TGT   WMT ratio
##   <date>     <dbl> <dbl> <dbl>
## 1 2015-06-10  68.8  65.4  1.05
## 2 2015-12-31  63.6  55.8  1.14
```

```
## integer(0)
```

As we can see, if we put a large value of **"k"** in this function, such as:

```r
pos_5 <- find_next_position(ratio = retail_stocks$ratio, k = 3.25)
```

Then it results in finding no position. Hence, a length zero integer vector will return.

Question 1.b) Developing the Function for "Finding All Positions"

```r
#' Title: Find All Positons
#'        Which finds all the opening and closing indices.
#' @param ratio: The daily price ratio for the two stocks (a numeric vector).
#' @param k: The value of "k" for calculating the decision boundaries (m - k * s)
#'          and (m + k * s) (default value = 1).
#' @param m: The estimated mean ratio m in the formula for the boundaries
#'           (default value = the mean of "ratio").
#' @param s: The standard deviation s of the price ratio, used in the formula for
#'           the boundaries (default value = the standard deviation of "ratio").
#' @return: A list of the indices of all positions.
#' @export
#'
#' @examples
find_all_positions <- function(ratio, k, m = mean(ratio),
                               s = sd(ratio)) {
  #' Turn the input into a dataframe.
  data <- as.data.frame(ratio)
  #' A condition to check whether there are positions. If there is none, then it
  #'   returns a length-zero list.
  if ((m - k * s) < min(data) & (m + k * s) > max(data)) {
    return(list())
  } else {
    #' The initial values needed for the loop are assigned.
```

```r
    starting_from <- 1
    i <- 0
    positions <- vector("list", )
    while (starting_from < nrow(data)) {
      position <- find_next_position(ratio, starting_from, k)
      if (length(position) == 0) {
        break
      } else {
        i <- i + 1
        name <- paste("positions", i, sep = "_")
        positions[[name]] <- position
        #' The loop starts from the first closing position.
        starting_from <- position[2]
      }
    }
  }
  return(positions)
}
```

Question 1.b) Testing the Function

```
## $positions_1
##  open close
##   387  1743
##
## $positions_2
##  open close
##  3878  4673
##
## $positions_3
##  open close
##  5144  6029
##
## $positions_4
##  open close
##  6411  6553
```

Question 2.a) Developing the Function for "Profit for Positions"

```r
#' Title: Profit for a Position
#'
#' @param position: A given position including opening and closing indices.
#' @param stock_a: The daily price for the first stock.
#' @param stock_b: The daily price for the second stock.
#' @param m: The average of the ratio of the two stocks.
#' @param p: The proportion commission for a transaction which %p of a
#'           transaction.
#'
#' @return: A numeric vector including the profit for stock_a, profit for stock_b
#'          cost for the whole transaction, and the total profit.
#' @export
#'
```

```r
#' @examples
position_profit <- function(position, stock_a, stock_b,
                            m = mean(stock_a / stock_b), p) {
  #' Compute the ratio of stocks.
  ratio <- stock_a[position][1] / stock_b[position][1]
  #' Compute the mean of the ratio of the two stocks
  m <- mean(stock_a / stock_b)
  #' Compute the unit of stock_a to be bought or sold.
  open_unit <- as.numeric(1 / stock_a[position][1])
  #' Compute the unit of stock_b to be bought or sold.
  close_unit <- as.numeric(1 / stock_b[position][1])
  #' Compute the revenue from buying or selling stock_a.
  revenue_a <- open_unit * as.numeric(stock_a[position][2])
  #' Compute the revenue from buying or selling stock_b.
  revenue_b <- close_unit * as.numeric(stock_b[position][2])
  #' This is the condition check that determines the buying or selling of stock.
  #'   It is based on the opening position to be either below or above mean. if
  #'   it is below mean, then it computes the profit based on "buying stock_a
  #'   and selling stock_b". If it is above mean, then it computes the profit
  #'   based on "selling stock_a and buying stock_b".
  if (ratio <= m) {
    profit_stock_a <- round((revenue_a - 1), 5)
    profit_stock_b <- round((1 - revenue_b), 5)
  } else {
    profit_stock_b <- round((revenue_b - 1), 5)
    profit_stock_a <- round((1 - revenue_a), 5)
  }
  #' Compute the cost and profit, with rounding.
  cost <- round((2 * -p + (revenue_a + revenue_b) * -p), 5)
  profit <- round((profit_stock_a + profit_stock_b + cost), 5)
  #' The output of the function in terms of a numeric vector.
  return(c("stock_a" = profit_stock_a,
           "stock_b" = profit_stock_b,
           "cost" = cost,
           "profit" = profit))
}
```

Question 2.a) Testing the Function

```
##  stock_a  stock_b     cost   profit
##  0.81157 -0.08669 -0.04898  0.67590
```

```
##  stock_a  stock_b     cost   profit
##  0.10789  0.26040 -0.04153  0.32676
```

```
##  stock_a  stock_b     cost   profit
## -0.22525  0.73509 -0.04960  0.46024
```

```
##  stock_a  stock_b     cost   profit
##  0.07493 -0.14664 -0.03778 -0.10949
```

Question 2.b) Developing the function for "Profit for All Positions"

```r
#' Title: Profit of Strategy
#'        Which calculate and aggregate the profit for all positions resulting
#'        from a certain strategy.
#' @param k: The value of "k" for calculating the decision boundaries (m - k * s)
#'           and (m + k * s) (default value = 1).
#' @param stock_a: The daily price for the first stock.
#' @param stock_b: The daily price for the second stock.
#' @param m: The average of the ratio of the two stocks.
#' @param sd: The standard deviation of the ratio of the two stocks.
#' @param p: The proportion commission for a transaction which %p of a
#'           transaction.
#'
#' @return: A list including two dataframes. The first dataframe will be the
#'          aggregated profit calculated for all positions combined and the
#'          second dataframe will be the disaggregated profit calculated for each
#'          positions.
#' @export
#'
#' @examples
strategy_profit <- function(k, stock_a, stock_b,
                            m = mean(stock_a / stock_b),
                            sd = sd(stock_a / stock_b),
                            p) {
  #' Find all positions using the function from the previous part.
  positions <- find_all_positions(ratio = stock_a / stock_b, k)
  #' Producing the output format (in the shape of a dataframe as shown in the
  #'   assignment) to show the profit for all positions in a
  #'   disaggregate manner.
  disaggregate <- data.frame(matrix(ncol = 8, nrow = length(positions)))
  #' The loop for reading data from position_profit function and fill in the
  #'   disaggregate dataframe.
  for (i in seq_along(positions)) {
    profit_data <- position_profit(positions[[i]],
      stock_a,
      stock_b,
      p = 0.01
      )
    disaggregate[i, ] <- c(
      names(positions)[i],
      map(positions, 1)[[i]],
      map(positions, 2)[[i]],
      map(positions, 2)[[i]] - map(positions, 1)[[i]],
      profit_data[[1]],
      profit_data[[2]],
      profit_data[[3]],
      round(profit_data[[4]], digits = 4)
    )
  }
  #' Here, I make sure the output of this dataframe is stored as numbers and
  #'   not chracters to be used for further analysis.
  disaggregate[, 2:4] <- sapply(disaggregate[, 2:4], as.integer)
  disaggregate[, 5:8] <- sapply(disaggregate[, 5:8], as.double)
```

```r
  #' Set the column names for disaggregate.
  colnames(disaggregate) <- c(
    "position",
    "open",
    "close",
    "duration",
    "stock_a",
    "stock_b",
    "cost",
    "profit"
  )
  #' Producing the output format (in the shape of a dataframe as shown in the
  #'   assignment) to show the profit for all positions in an aggregate manner.
  aggregate <- data.frame(matrix(ncol = 6, nrow = 1))
  #' Read data from positions to fill in the aggregate dataframe.
  aggregate[1, ] <- c(
    length(positions),
    sum(disaggregate$duration),
    round(sum(disaggregate$stock_a), digits = 4),
    round(sum(disaggregate$stock_b), digits = 4),
    round(sum(disaggregate$cost), digits = 4),
    round(sum(disaggregate$profit), digits = 3)
  )
  #' Set the column names for aggregate.
  colnames(aggregate) <- c(
    "positions",
    "duration",
    "stock_a",
    "stock_b",
    "cost",
    "profit"
  )
  return(list("aggregate" = aggregate, "disaggregate" = disaggregate))
}
```

Question 2.b) Testing the Function

```
## $aggregate
##   positions duration stock_a stock_b    cost profit
## 1         4     3178  0.7691  0.7622 -0.1779  1.353
##
## $disaggregate
##      position open close duration  stock_a  stock_b     cost  profit
## 1 positions_1  387  1743     1356  0.81157 -0.08669 -0.04898  0.6759
## 2 positions_2 3878  4673      795  0.10789  0.26040 -0.04153  0.3268
## 3 positions_3 5144  6029      885 -0.22525  0.73509 -0.04960  0.4602
## 4 positions_4 6411  6553      142  0.07493 -0.14664 -0.03778 -0.1095
```

Question 2.c) Developing the function for "Strategy Assessment"

```r
#' Title: Assessment of Strategy
#'
#' @param start_k: Starting value of k.
#' @param end_k: Ending value of k.
#' @param step_k: Intervals of k that will be considered as a strategy.
#' @param stock_a: The daily price for the first stock.
#' @param stock_b: The daily price for the second stock.
#' @param m: The average of the ratio of the two stocks.
#' @param sd: The standard deviation of the ratio of the two stocks.
#' @param p: The proportion commission for a transaction which %p of a
#'           transaction.
#'
#' @return: A table of
#' @export
#'
#' @examples
assess_strategy <- function(start_k,
                            end_k,
                            step_k,
                            stock_a,
                            stock_b,
                            m = mean(stock_a / stock_b),
                            sd = sd(stock_a / stock_b),
                            p) {
  #' Set the interval for k.
  interval <- seq(start_k, end_k, step_k)
  #' Producing the output format (in the shape of a dataframe as shown in the
  #'   assignment) to show the profit for all k.
  assessment <- data.frame(matrix(ncol = 7, nrow = length(interval)))
  #' The loop for reading data from strategy_profit function and fill in the
  #'   assessment dataframe.
  i <- 1
  for (i in seq_along(interval)) {
    profit_data <- strategy_profit(interval[i],
      stock_a,
      stock_b,
      p = 0.01
    )
    assessment[i, ] <- c(
      interval[i],
      map(profit_data, 1)[[1]],
      map(profit_data, 2)[[1]],
      round(map(profit_data, 3)[[1]], digits = 2),
      round(map(profit_data, 4)[[1]], digits = 2),
      round(map(profit_data, 5)[[1]], digits = 2),
      round(map(profit_data, 6)[[1]], digits = 2)
    )
  }
  #' Here, I make sure the output of this dataframe is stored as numbers and
  #'   not chracters to be used for further analysis.
  assessment[, -2] <- sapply(assessment[, -2], as.double)
  assessment[, 2] <- sapply(assessment[, 2], as.integer)
  #' Set the column names for assessment.
```

```r
  colnames(assessment) <- c(
    "k",
    "position",
    "duration",
    "stock_a",
    "stock_b",
    "cost",
    "total"
  )
  return(assessment)
}
```

Question 2.c) Testing the Function

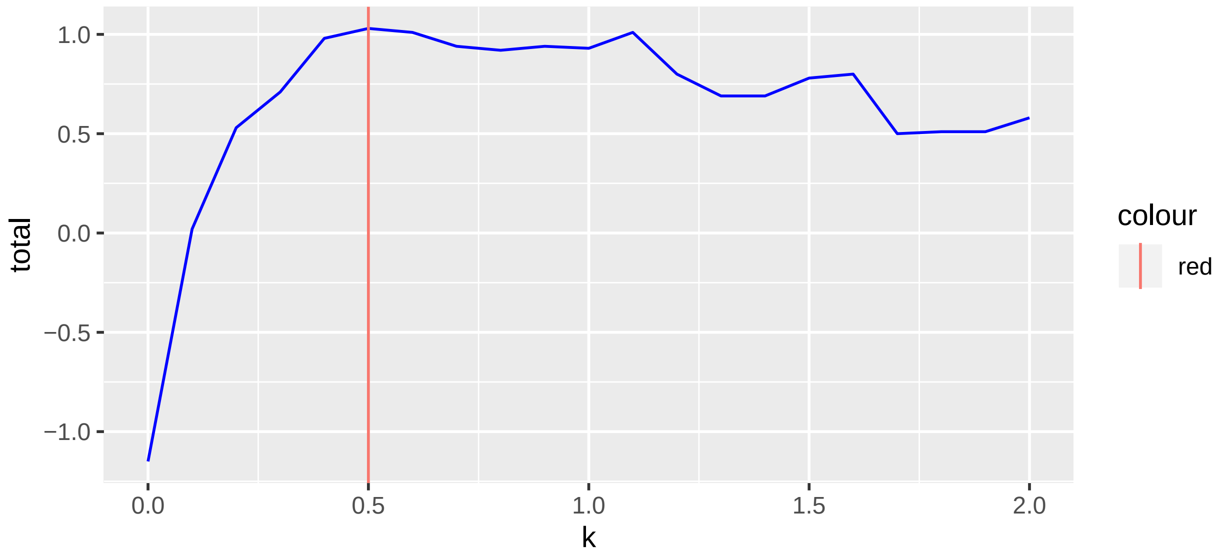
```
##      k position duration stock_a stock_b  cost total
## 1 0.50       12     5220    1.92    0.63 -0.56  1.99
## 2 0.75        9     4687    3.44   -0.63 -0.42  2.39
## 3 1.00        7     4318    2.82    0.01 -0.32  2.51
## 4 1.25        4     3178    0.77    0.76 -0.18  1.35
## 5 1.50        3     2198    1.36    0.01 -0.13  1.25
## 6 1.75        3     2078    1.52    0.16 -0.13  1.55
## 7 2.00        2      774    0.24    0.35 -0.08  0.52
```

Question 3) Applying Pairs Trading

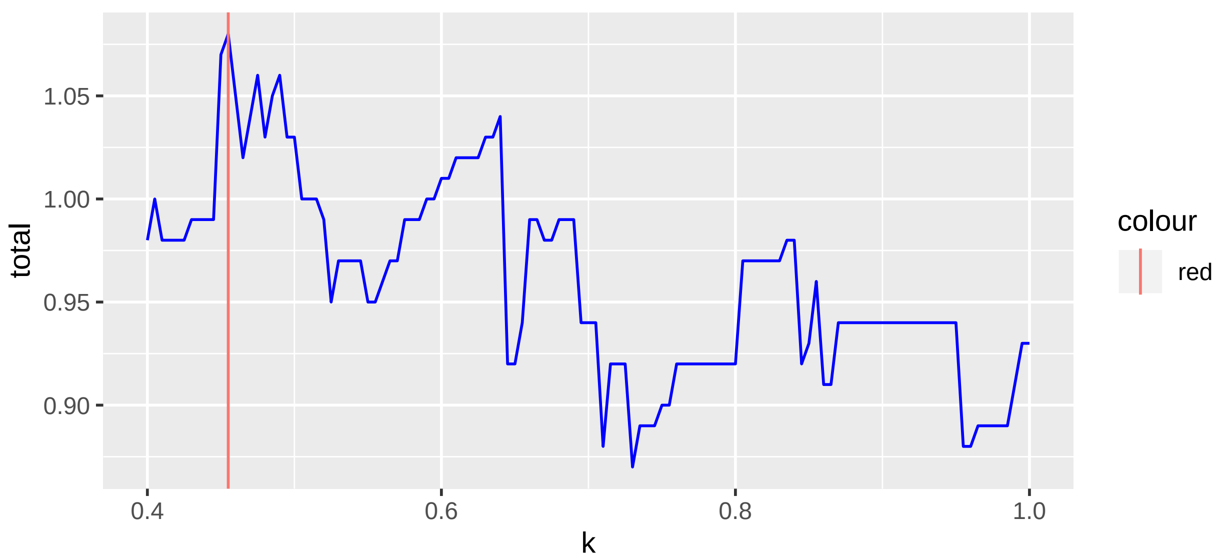
According to the instructions, I create three datasets for training, validation, and testing. Below is the code for creating the datasets:

```r
training_start_date <- as.Date("1990-01-01")
training_end_date <- as.Date("1994-12-31")
q3_training_data <- stocks %>%
  filter(date >= training_start_date, date <= training_end_date) %>%
  select(date, PEP, CVX) %>%
  mutate(ratio = PEP / CVX)
validation_start_date <- as.Date("1995-01-01")
validation_end_date <- as.Date("1999-12-31")
q3_validation_data <- stocks %>%
  filter(date >= validation_start_date, date <= validation_end_date) %>%
  select(date, PEP, CVX) %>%
  mutate(ratio = PEP / CVX)
testing_start_date <- as.Date("2000-01-01")
testing_end_date <- as.Date("2019-09-30")
q3_testing_data <- stocks %>%
  filter(date >= testing_start_date, date <= testing_end_date) %>%
  select(date, PEP, CVX) %>%
  mutate(ratio = PEP / CVX)
```

Second, I estimate the m and s based on training data. Then, I assess the strategy based on the validation data to see for which **"k"** I get the maximum profit.



According to the results, a k of **0.5** yields the highest profit. To have a more precise estimate, I estimate the k with steps of 0.005 in 0.4 to 1 interval.



This would result in a **"k"** of **0.455** which will be used for estimating the final profit.

By using m and s estimated on validation data and **"k = 0.455"**, I can compute the total profit for this strategy.

Therefore, the **"Total Profit"** is equal to **1.389**.