Australian National University

ASSIGNMENT COVER SHEET

ANU College of Engineering and Computer Science
Australian National University
Canberra ACT 0200 Australia
www.anu.edu.au

This coversheet should be filled electronically when possible and attached as the front page of your report in PDF format.

Student ID For group assignments, list each student's ID	u6451847		
Course Code	COMP1100		
Course Name	Programming as Problem Solving		
Assignment number	2		
Assignment Topic	Stock Market		
Lecturer	Katya Lebedava		
Tutor	David O'Donohue		
Tutorial (day and time)	Monday 15:00–17:00		
Word count	1483	Due Date	September 22
Date Submitted	September 20	Extension Granted	

I declare that this work:

- ☑ upholds the principles of academic integrity, as defined in the University Academic Misconduct Rules;
- ☑ is original, except where collaboration (for example group work) has been authorised in writing by the course convener in the course outline and/or Wattle site;
- ☑ is produced for the purposes of this assessment task and has not been submitted for assessment in any other context, except where authorised in writing by the course convener;
- ☑ gives appropriate acknowledgement of the ideas, scholarship and intellectual property of others insofar as these have been used;
- ☑ in no part involves copying, cheating, collusion, fabrication, plagiarism or recycling.

Signatures

For group assignments, each student must sign. Chucheng Qian

Assignment02- Stock Market

Name: Chucheng Qian ID: u6451847

1. Introduction

This report is written to explain some basic structures about this Stock Market program and elaborate the problems I have met and how I solved them when I was writing the making order part. For this Stock Market program, the simulation and data input and output parts have been given, and all I have to do is filling the order module to give a buying and selling order for stocks. The simulator will implement the order with real stock data and then return the final wealth.One can figure out whether the designed trading strategy is profitable by comparing the final wealth and the initial capital.

2.Description of Given Functions

In this section, I want to describe your understanding of the following functions.

In my opinion, the "calculateWealth" function is used to calculate overall wealth I obtained at present.It takes "Portfolio" and "[StockHistory]" types to input the current amount of cash and stocks and the prices of stocks.Its support function currentValue calculates the value of one stock I am holding now and expressed it as an integer.Then the calculateWealth can use it to obtain the total value of stocks I am holding and add to the cash left now.

Secondly, the purpose of the "getStockPrice" function is to obtain the price of one specific stock.It takes "Stock" and " [StockHistory]" types to input a stock whose present price is wanted and a large pool of stocks and their present prices and historical prices.The "getStock" expression is used to pick out that stock and its list of prices and they will be expressed as a tuple (Stock, [Price]).After that, the "getStockPrice" function is able to pick out the first element which is the current price of this stock from the price list.

Thirdly, the complicated "executeOrders" function is used to execute the orders from the making order part and update my holding list.It is divided into several parts by using "case " and guards.One one hand ,it involves the condition that the orders have not been made and then calculates the loan interests or deposit interests for cash now.On the other hand, if orders exist, there are conditions including that this order is unable to be execute ,short-selling and regular selling and buying order.The order cannot be made refers to that it has reached to the data which means the investable benchmark instead of real stocks ,the condition that short-selling and regular purchase of stocks cannot be made due to that current wealth is less than the sum of the cost and commission, the condition that one has owned too many of one stocks or the loan from bank is larger than -500000.If this order can be executed, then the function will classify it to different conditions by guards and update the new cash an stocks and execute the next order by recursion.

Finally, the "simulate" function can be considered to put "executeOrders" function and "makeOrders" together so that the orders can be used really and got a simulation.It uses built-in function "foldl" so that it can execute one list of orders with current "Portfolio" and " [stockHistory]" first to give out a new Portfolio and then execute the new list of orders based on the new "[stockHistory]" with that updated Portfolio situation and continue to go on.The list of [stockHistory] is made by the supported function "unfoldHistories" which takes in a list of StockHistory and output a list of [StockHistory].

3. Trading Strategy and Making order Structure

First and foremost, I want to introduce some of my concepts for the making order part.

From my point of view, a good trading strategy should be long-term effective and stable.After doing some research, I decided to use the moving average (MA) way to make a decision whether buy or sell the stocks.It is a simple technical analysis tool that smooths out price data by creating a constantly updated average price.Generally, if the price is above a moving average, the trend is up and vice versa.(Mitchell, 2016) Therefore, I write two functions "aboveAverage" and "belowAverage" to group the stocks whose prices are going to rise and fall respectively by comparing their present adjusted close prices with which of past 5 days.

```
aboveAverage :: [StockHistory] -> Double -> [StockHistory]
aboveAverage stocks rate
    | stocks == [] = []
    | getdays < 20 =[]
    | otherwise = take 5 (filter (\x -> getStockPrice (fst x) stocks >
(average (fst x) stocks) * rate) stocks)
    where
        getdays= length(snd (head stocks))

belowAverage :: Holdings -> [StockHistory] -> Holdings
belowAverage holds stocks
    | stocks ==[]= []
    | holds == [] = []
    | getdays < 20 =[]
    | otherwise = filter (\x -> getStockPrice (fst x) stocks < (average
(fst x) stocks)) holds
    where
        getdays= length(snd (head stocks))
```

Then I applied these two important conditions with other supported functions, for example, the "quantity" which calculates the purchase amount of each stock and "getPriceSum" which obtains a list of present prices of the stocks I am going to buy to the "makeOrders" function.The "makeOrders" function will finally give out a list of orders about buying or selling which stocks.

4. Solving Problems

During the process of designing the making order part, some conceptual and technical problems show up.

When I started writing the function "makeOrders", I make it like that

```
makeOrders :: Portfolio -> [StockHistory] -> [Order]
makeOrders (cash, holds) history =
    case (belowAverage holds history) of
        x:xs -> [Order (fst x) (-(snd x))] ++ makeOrders (cash, xs) history
        [] -> case (aboveAverage history 1.005) of
              [] -> []
              b:_ -> [Order (fst b) (quantity cash (fst b) history)] ++
case (tail(aboveAverage history 1.005)) of
                  []->[]
                  c:_->[Order (fst c) (quantity cash (fst c) history)] ++
makeOrders (cash, holds) history
```

However, this function will obtain repeated order lists in the recursion process as the "case" expression " (aboveAverage history 0.05) " will always match the same thing.It is unpractical to use recursion here.For this problem, I decided to use guards instead of the case expression and write another function " makeOrders2 " to achieve the buying orders list by using recursion individually.

```
makeOrders :: Portfolio -> [StockHistory] -> [Order]
makeOrders (cash, holds) history =
    case (belowAverage holds history) of
        x:xs -> [Order (fst x) (-(snd x))] ++ makeOrders (cash, xs) history
        []
         |(aboveAverage history 1.005)==[]->[]
         |otherwise -> makeOrders2 (cash, holds) history (aboveAverage
history 1.005)


makeOrders2 :: Portfolio -> [StockHistory] ->[StockHistory]-> [Order]
makeOrders2 (cash, holds) history a = case a of
         []->[]
         c:cs ->[Order (fst c) (quantity cash history)] ++ makeOrders2
(cash, holds) history cs
```

This kind of expression will be more clear and feasible.

4. Testing

The method I used to test my program is doctest and my program has passed these tests.

```
-- | Stock Market
--
-- >>> average "TLS" [("TLS",[6,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5]),
("NNT",[8,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10]),
("CCS",[9,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8]),("HHA",
[5,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4]),("KKE",
[6,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7]),("GGA",
[4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4]),("JJY",
[9,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6]),("NNA",
[4,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7])]
-- 5.0
--
-- >>> aboveAverage [("TLS",[6,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5]),
("NNT",[8,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10]),
("CCS",[9,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8]),("HHA",
[5,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4]),("KKE",
[6,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7]),("GGA",
[4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4]),("JJY",
[9,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6]),("NNA",
[4,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7])]  1.05
-- [("TLS",
[6.0,5.0,5.0,5.0,5.0,5.0,5.0,5.0,5.0,5.0,5.0,5.0,5.0,5.0,5.0,5.0,5.0,5.0,5.
0,5.0,5.0]),("CCS",
[9.0,8.0,8.0,8.0,8.0,8.0,8.0,8.0,8.0,8.0,8.0,8.0,8.0,8.0,8.0,8.0,8.0,8.0,8.
0,8.0,8.0,8.0]),("HHA",
[5.0,4.0,4.0,4.0,4.0,4.0,4.0,4.0,4.0,4.0,4.0,4.0,4.0,4.0,4.0,4.0,4.0,4.0,4.
0,4.0,4.0,4.0]),("JJY",
[9.0,6.0,6.0,6.0,6.0,6.0,6.0,6.0,6.0,6.0,6.0,6.0,6.0,6.0,6.0,6.0,6.0,6.0,6.
0,6.0,6.0,6.0])]
--
-- >>> belowAverage [("TLS",18),("NNT",50),("CCS",88),("KKE",5),
("GGA",100),("NNA",50)] [("TLS",
[6,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5]),("NNT",
[8,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10]),("CCS",
[9,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8]),("HHA",
[5,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4]),("KKE",
[6,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7]),("GGA",
[4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4]),("JJY",
[9,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6]),("NNA",
[4,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7])]
-- [("NNT",50),("KKE",5),("NNA",50)]
--
```

```
-- >>> belowAverage [] [("TLS",
[6,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5]),("NNT",
[8,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10]),("CCS",
[9,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8]),("HHA",
[5,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4]),("KKE",
[6,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7]),("GGA",
[4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4]),("JJY",
[9,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6]),("NNA",
[4,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7])]
--[]
--
-- >>> makeOrders (100000, [("TLS",18),("NNT",50),("CCS",88),("KKE",5),
("GGA",100),("NNA",50)])  [("TLS",
[6.5,5,5.5,5,5.6,5.8,5.5,5,5.6,5.8,5.5,5,5.6,5.8,5.5,5,5.6,5.8,5.5,5,5.6,5.
8]),("NNT",
[8.3,10,9.4,8.6,8.5,8.8,9.4,8.6,8.5,8.8,9.4,8.6,8.5,8.8,9.4,8.6,8.5,8.8,9.4
,8.6,8.5,8.8]),("CCS",
[9.3,7.9,8.8,8.6,8,8,8.8,8.6,8,8,8.8,8.6,8,8,8.8,8.6,8,8,8.8,8.6,8,8]),
("HHA",[5,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4]),("KKE",
[6,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7]),("GGA",
[4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4]),("JJY",
[9,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6]),("NNA",
[8,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9]),("GGG",
[9,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10]),("YYY",
[5,6.5,7.8,7.7,7.4,4.3,7.8,7.7,7.4,4.3,7.8,7.7,7.4,4.3,7.8,7.7,7.4,4.3,7.8,
7.7,7.4,4.3]),("KKK",[4,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7]),("NNG",
[9,7.5,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7])]
-- [Order "NNT" (-50),Order "KKE" (-5),Order "NNA" (-50),Order "TLS"
2577,Order "CCS" 2577,Order "HHA" 2577,Order "JJY" 2577,Order "NNG" 2577]
--
-- >>> makeOrders (5000000,[]) [("TLS",[6.5,5,5.5]),("NNT",[8.3,10,9.4]),
("CCS",[9.3,7.9,8.8]),("HHA",[5,4,4]),("KKE",[6,7,7]),("GGA",[4,4,4]),
("JJY",[9,6,6]),("NNA",[8,9,9]),("GGG",[9,10,10]),("YYY",[5,6.5,7.8]),
("KKK",[4,7,7]),("NNG",[9,7.5,7])]
--[]
```

As the making order part contains several functions, I just want to test some important and comprehensive ones.

The "average" function is not in the origin skeleton and I use it to calculate the 20-day moving average of one stock.It is a relatively simple function so I just input a stock and its price list and it gets the correct output.

For the "aboveAverage" function, the inputs are a list of stockHistory and a rate which represents the increasing percentage of the stock price.The situation is that there are only 4 stocks have current prices which are 0.5% larger than their MAs.Although this function is designed to expect output as a list with 5 elements, it actually outputs a list with 4 elements now.

In addition, for "belowAverage" and "makeOrders" functions, I set two tests for each of them.If the input holding lists are empty, there is no stock whose current price is lower than its MA, therefore the output will be an empty list for the "belowAverage" function.Similarly, for the second test of "makeOrders" function, every stock in [stockHistoty] only has prices for 3 days, thus they do not have MA.As the order is made based on the MA, the function will output an empty list.The second test of "belowAverage" and the first test of "makeOrders" are just regular tests which input the normal data and see if they can output expected values.

Since I used doctest to test the problem instead of QuickCheck, it cannnot cover all the possible cases for each data types, and it may cause some potential failure for some inputs.However, the QuickCheck part may be a bit of complicated in this part.As the QuickCheck generates the input values automatically according to the input types, it is difficult to write the expectation in many different situations to make them match and pass all tests.Doctest may be more desirable due to the time limit.

4. Assumptions and Changes

Firstly, I suppose that the ideal increasing percentage of prices of stocks is larger than 0.5%, and all stocks which gots this increasing percentage are those I intended to buy, therefore I just use the aboveAverage function to pick out any 5 of them.

Besides, if I am able to rewrite the program, I will try to use short-selling as part of my strategy.As a result of lacking the financial knowledge, I am still not sure whether I understand how short-selling works very clearly even though after reading some materials.A further research may be required.

5. Potential Problems

As I mentioned before, the ideal increasing percentage of prices of stocks I set is larger than 0.5% and the function will buy any 5 stocks which satisfy this condition.However, it may cause many problems. For example, if one of the stocks has increased dramatically to the peak, but my function does not consider this and will still buy it with a high cost.In addition, if a stock shows a continuing downward trend but gets a small growth in one day, my function will also buy it, however, after that, the price will continue to drop.It will cause a wastage.

6. Conclusion

For this program, I designed the making order part in the most basic way, that the function buys stocks when it shows an upward trend and sells when it shows a sown trend.It is unrealistic to elaborate every problem I have met and write exhaustive tests for all functions due to the time concern. The program may also cause large losses in some special situations. I will keep working on it to find out how to get it more effective, practical and with fewer problems.

7. Reference

1. Cory ,Mitchell. (2016, July 8). How To Use A Moving Average To Buy Stocks.Retrieved 19

September 2017,from http://www.investopedia.com/articles/active-trading/052014/how-use-moving-average-buy-stocks.asp