*Texas A&M University*

# FINC 689 : Systematic Trading Strategies

## Homework 4 Tutorial Submission (Spring 2023)

Instructor

1. Dr. Michael Ketzenberg

Team 4

1. Sankalp Chapalgaonkar (233005696)
2. Kushagra Jain (532008695)
3. Kailash Hariharan (933003377)

***Introduction***

Welcome to this tutorial on backtesting a trend-following or mean-reverting trading strategy using the S&P 500 energy sector stocks as your stock universe. Backtesting is a crucial step in the development and evaluation of trading strategies. It involves simulating the performance of a trading strategy on historical data to assess its potential profitability and risk.

In this tutorial, we will cover the steps to develop and backtest a daily trading strategy that buys and/or sells on the open, but allows positions to be carried over from one period to the next. We will use the S&P 500 energy sector stocks as our stock universe and a starting cash on hand of $100,000. We will allow long and short positions and implement either a two-trend or three-trend system, depending on the strategy chosen.

Our performance measures for the backtest will include all of the performance measures used in Homework 3. We will also discuss any optimization conducted and how we arrived at the backtest settings (model parameters) used. After completing the backtesting, we will rerun the strategy on another industrial sector of our choice and run the energy sector of stocks backtest on the period from January 2019 through December of 2021.

***Learning Objectives***

Before we dive into the problems we are going to address we want to outline the major learnings you can expect to be gaining at the end of this Tutorial. Here is the list of Learning Objectives for this tutorial:

- Understand the basics of backtesting and its importance in evaluating trading strategies
- Develop a trend-following or mean-reverting trading strategy for the S&P 500 energy sector stocks
- Backtest your strategy using historical data
- Interpret and evaluate the performance metrics of your strategy
- Conduct optimization to improve the performance of your strategy
- Analyze and compare the performance of the energy sector stocks strategy across different backtesting periods

***Two-trend Strategy Overview***

Summarized below is the overview of our trading strategy:

- Implemented a two-trend strategy using 5-day and 10-day exponential moving averages
- To reduce risk of false signals,we combined two-trend analysis with bollinger band analysis and relative strength index
- The strategy enters into a long trade when fast exponential moving average crosses over slow exponential moving average and moving average signal crosses over upper band; exit when fast exponential moving average crosses below slow exponential moving average and get a down trend signal from bollinger band
- For short trades, enter when fast exponential moving average crosses below slow exponential moving average and moving average crosses below lower band; exit when see an uptrend signal on bollinger bands and fast exponential moving average crosses above slow exponential moving average
- Use Relative Strength Index to decide on trades to pick to satisfy max trades constraint, preferred trades in ascending order of Relative Strength Index

***Preprocessing: Loading the Data***

As done in previous assignments, we start off by setting the working directory and installing the required packages. We will also load the universe data from Rdata file. We have updated universe data file to include the sectors information for each symbol. Using this sectors information, we first filter out the stocks for Energy sector and then implement two trend strategy to maximize the portfolio stats.

*Steps and Implementation*

- First we load the rstudioapi, which will be used to set the current path in working directory. We clear all the environment variables using rm function.
- Then we load two R packages: **quantmod** and **TTR**. The quantmod package provides a suite of functions for financial data analysis, including downloading historical stock prices, charting, and technical analysis and the TTR package provides technical analysis functions for financial data.
- Then we load the universe.rdata file containing the stocks OHLC and sectors data. Then we define the start and end dates for the analysis. Specifically, the **from** variable is set to **December 17th, 2020** and the **to** variable is set to **December 31st, 2022**.
- At this point we subset the universe data frame to only include stocks from the **Energy** sector that have a date within the specified range using the subset() function which is used to filter the universe data frame to only include rows where the Sector column is "Energy" and the date column falls within the specified from and to dates.

```
# **************SET WORKING DIRECTORY AND CLEAR ENVIRONMENT ********************
library(rstudioapi)
current_path = rstudioapi::getActiveDocumentContext()$path
setwd(dirname(current_path ))
rm(list=ls())
options(scipen=999)

# **************GET DATA AND SET TRADING DATE RANGE ***************************

library(quantmod)
library(TTR)
load("universe.rdata")
from<-as.Date("2020-12-17")
to<-as.Date("2022-12-31")
universe<-subset(universe,universe$Sector=="Energy"&
                   universe$date>=from&universe$date<=to)
```

***Parameter Initialization***

We set the parameters that will be used later in the strategy. Specifically we set initial equity of 100,000 $ and set a limit on maximum number of trades allowed per day as 11. We also limit the amount of the equity traded in one transaction as 11,000$

```
################################ Configuration Setting ######################

symbols<-unique(universe$symbol)
initialequity<-100000                # starting money
maxtrade<-11000                      # maximum value of any single trade
maxdaytrades<-11                     # maximum trades in one day
```

***Generating Indicators***

Here we define the **genIndicators** function, which takes a stock symbol as input and generates technical indicators for that stock. We have developed a two trend strategy which takes into account the bollinger bands, fast and slow Exponential Moving Average (EMA), and Relative Strength Index (RSI) index to quantify the risk element. In this function, we initialize the all these parameters, that is generating the bollinger bands for the energy stocks, deriving the two moving averages and calculating the RSI index. We will use these three parameters later in the code to make decisions on indicators and signals generated.

*Steps and Implementation*

1. We first subset the universe data to obtain only the data for the input stock symbol. We then create an xts object from the stock data and create a copy of it as temp.xts

```
genIndicators <- function(sym) {
  print(paste('Generating Indicators for symbol: ', sym))

  stock <- subset(universe, universe$symbol == sym)
  stock.xts <- xts(stock[, c(3:7)], stock$date)

  temp.xts <- stock.xts
  temp.xts$high <- stats::lag(stock.xts$high, n = 1)
  temp.xts$low <- stats::lag(stock.xts$low)
  temp.xts$close <- stats::lag(stock.xts$close)
```

2. Then we use BBands function from the TTR package to calculate Bollinger Bands for the stock, using a 5-day simple moving average and a standard deviation of 2. If there is an error or warning during this calculation, the resulting bb object is set to NULL. Then we set the parameters such as down, moving average and up in the xts object to the corresponding indicators from bollinger bands.

```
45  bb <- tryCatch({
46    BBands(HLC(temp.xts), n = 5, maType = "SMA", sd = 2)
47  }, warning = function(w) {
48    bb <- NULL
49  }, error = function(e) {
50    bb <- NULL
51  })
52
53  if (is.null(bb)) {
54    stock.xts$dn <- NA
55    stock.xts$mavg <- NA
56    stock.xts$up <- NA
57    stock.xts$range <- NA
58  } else {
59    stock.xts$dn <- bb$dn
60    stock.xts$mavg <- bb$mavg
61    stock.xts$up <- bb$up
62  }
```

3. After bollinger bands we calculate the 5-day and 10-day exponential moving averages (EMAs) for the stock using the EMA function from the TTR package. If there is an error or warning during these calculations, the resulting ema5 and ema10 objects are set to NULL.

```
  ema5 <- tryCatch({
    ema5 <- EMA(temp.xts$close, n = 5)
  }, warning=function(w) {ema5 <- NULL}, error=function(e) {ema5 <- NULL})

  ema10 <- tryCatch({
    ema10 <- EMA(temp.xts$close, n = 10)
  }, warning=function(w) {ema10 <- NULL}, error=function(e) {ema10 <- NULL})

  if (is.null(ema5) | is.null(ema10)) {
    stock.xts$ema5 <- NA
    stock.xts$ema10 <- NA
  } else {
    stock.xts$ema5 <- ema5$EMA
    stock.xts$ema10 <- ema10$EMA
  }

```

4. Finally, we calculate the 14-day relative strength index (RSI) for the stock using the RSI function from the TTR package. We create a data frame from the xts object and add the stock symbol as a column. The resulting data frame is returned as the output of the function.

```
  stock.xts$rsi <- RSI(stock.xts$close, n = 14)
  stock.xts$range<-stock.xts$rsi
  stock <- data.frame(stock.xts)
  date <- as.Date(rownames(stock))
  stock <- cbind(date, stock)
  stock <- cbind(sym, stock)
  names(stock)[1] <- "symbol"
  rownames(stock) <- seq(1, nrow(stock), 1)

  return(stock)
}
```

***Generating Signals***

We use the function **genSignals** to generate trading signals for a given stock symbol using technical indicators such as Bollinger Bands and exponential moving averages. The function takes one argument **sym**, which is the stock symbol for which the signals are to be generated. The function performs the following steps:

1. Here first we subset the indicators data frame to extract the rows corresponding to the given stock symbol and then convert the subsetted data frame into an xts object with the stock prices and the technical indicators as columns.

```
genSignals <- function(sym) {
  print(paste('Generating Signals for symbol: ',sym))
  stock <- subset(indicators, indicators$symbol == sym)
  stock.xts <- xts(stock[, c(3:ncol(stock))], stock$date)
```

2. Calculates the **Bollinger Bands signals** as follows
    a. stock.xts$cross.upper indicates a buy signal when the closing price is above the upper Bollinger Band.
    b. stock.xts$cross.lower indicates a sell signal when the closing price is below the lower Bollinger Band.
    c. stock.xts$cross.trendup indicates a bullish signal when the closing price is above the moving average.
    d. stock.xts$cross.trenddn indicates a bearish signal when the closing price is below the moving average.

```
# Generate Bollinger Bands signals
stock.xts$cross.upper <- ifelse(stock.xts$close > stock.xts$up, 1, 0)
stock.xts$cross.lower <- ifelse(stock.xts$close < stock.xts$dn, 1, 0)
stock.xts$cross.trendup <- ifelse(stock.xts$close > stock.xts$mavg, 1, 0)
stock.xts$cross.trenddn <- ifelse(stock.xts$close < stock.xts$mavg, 1, 0)
```

3. Calculates the **EMA signals** as follows
    a. stock.xts$cross.upper_ema indicates a buy signal when the closing price is above both the 5-day and 10-day EMA.
    b. stock.xts$cross.trenddn_ema indicates a sell signal when the closing price is below the 5-day EMA.

c. stock.xts$cross.lower_ema indicates a sell signal when the closing price is below both the 5-day and 10-day EMA.
d. stock.xts$cross.trendup_ema indicates a buy signal when the closing price is above the 5-day EMA.

```
#Generate EMA
stock.xts$cross.upper_ema <-
  ifelse((stock.xts$close > stock.xts$ema10) & (stock.xts$ema5 > stock.xts$ema10), 1, 0)  # Entry for Buy
stock.xts$cross.trenddn_ema <-
  ifelse(stock.xts$close < stock.xts$ema5, 1, 0)  # Buy order Stop Loss or Exit at Profit

stock.xts$cross.lower_ema <- ifelse((stock.xts$close < stock.xts$ema10) &
                                    (stock.xts$ema5 < stock.xts$ema10), 1, 0)  # Entry for Sell
stock.xts$cross.trendup_ema <-
  ifelse(stock.xts$close > stock.xts$ema5, 1, 0)  # Sell

```

4. Finally we prepare the output data in the desired format, with columns for date, symbol, and the various trading signals generated by the function. We first convert the stock.xts object generated in the previous step to a data frame called stock. We then extract the row names of stock (which correspond to the dates) and convert them to the date format using the as.Date function. The date column is then added to the stock data frame using the cbind function. The first column of stock is then renamed as "symbol" using the names function.

```
  stock <- data.frame(stock.xts)
  date <- as.Date(rownames(stock))
  stock <- cbind(date, stock)
  stock <- cbind(sym, stock)
  names(stock)[1] <- "symbol"
  rownames(stock) <- seq(1, nrow(stock), 1)
  return(stock)
}
```

## ***Close Positions***

The close positions function has been modified to apply rules based on the trading strategy proposed in the beginning of the tutorial. In this case, we are closing positions which are open if it satisfies the following conditions for short and long positions respectively -

*Short positions -*

- The trend of the signal is increasing
- The signal crosses the moving average line and has an increasing trend

This ensures that we recognise that the stock has an upwards trend which is not desirable for short positions. The modified code is given below -

```
161     candidates<-subset(signals,signals$date==day&            # check shorts first
162                        ((signals$cross.trendup==1) & (signals$cross.trendup_ema==1)))[,c(1,2,6)] # grab
163     names(candidates)[2]<-"closedate"                        # keep track of the close date so we
164     names(candidates)[3]<-"outprice"                         # can check how long we hold our positions
165     closeshort<-merge(shortposition,candidates,by="symbol")  # Close only if we have a position
```

*Long positions -*

- The trend of the signal is downwards
- The signal crosses the moving average line and has a downward trend

This ensures that we recognise that the stock has a downward trend and that is going to lose money for a long position. The modified part of the code is given below -

```
166     candidates<-subset(signals,signals$date==day&            # Now do the same for longs
167                        ((signals$cross.trenddn==1) | (signals$cross.trenddn_ema==1)))[,c(1,2,6)]
168     names(candidates)[2]<-"closedate"
169     names(candidates)[3]<-"outprice"
170     closelong<-merge(longposition,candidates,by="symbol")
171     closed<-rbind(closeshort,closelong)                      # put all our positions to close together
```

The last line of code above combines all rows for the closed long and short positions for the given day after which the below piece of code computes the closed cash, sell and buy prices, profit and close cash.

```
    if (nrow(closed)>0) {
      closed$closecash<-closed$outprice*closed$position      # compute closing calculations of cash
      closed$sellprice<-ifelse(closed$type=="Long",closed$outprice,closed$sellprice)
      closed$buyprice<-ifelse(closed$type=="Short",closed$outprice,closed$buyprice)
      closed$profit<-(closed$sellprice-closed$buyprice)*abs(closed$position)
      cash<-sum(closed$closecash)   # get the aggregate value to add back to currentcash
    } else closed<-NULL
  }
  return(list(closed=closed,cashin=cash))
```

## ***Open Positions***

The open positions function has been modified to apply rules based on the trading strategy proposed in the beginning of the tutorial. In this case, we are entering positions if it satisfies the following conditions for short and long  positions respectively -

*Short positions -*

- The signal crosses the lower trend line
- The signal has a downward trend, is lower than the slow moving average line and the faster moving average trend is lower than the slower moving average trend line

These conditions ensure that the trend of the signal is downwards and hence it is wise to short the position.

```
    candidates<-subset(signals,signals$date==day&                # check shorts first
                         signals$cross.lower==1 & (signals$cross.lower_ema==1))
    temp<-merge(candidates,shortposition,by="symbol",all.x=TRUE)
    openshort<-subset(temp,is.na(dummy))                         # only short if we don't have a position
    if (nrow(openshort)>0) {
      openshort<-openshort[,c(1:ncol(openshort)-1)]              # get rid of dummy column
      openshort$type<-"Short"                                    # we will open a short position
    } else {openshort<-NULL}                                     # if the dataframe is empty, set it to null
```

*Long positions -*

- The signal crosses the upper trend line
- The signal has a upward trend, is greater than the slow moving average line and the faster moving average trend is greater the slower moving average trend line

These conditions ensure that the trend of the signal is upwards and hence it is wise to long the position.

```
206     candidates<-subset(signals,signals$date==day&                # now proceed and do same for longs
207                          signals$cross.upper==1  & (signals$cross.upper_ema==1))
208     temp<-merge(candidates,longposition,by="symbol",all.x=TRUE)
209     openlong<-subset(temp,is.na(dummy))
210     if (nrow(openlong)>0) {
211       openlong<-openlong[,c(1:ncol(openlong)-1)]
212       openlong$type<-"Long"
213     } else {openlong<-NULL}
214     opened<-rbind(openlong,openshort)                            # put all positions to be opened together
```

The last line in the above screenshot shows that we are then combining long and short positions to arrive at all open positions.

Further, in order to ensure that we do not cross the maximum number of trades on a given day, we sort all of the positions based on the RSI values and pick trade positions with lower RSI positions up to the maximum number of trades. This is achieved using the range field that was created in the generate indicators section.

```
228     opened<-opened[order(opened$range),]                         # sort them by the risk
229     numtrades<-nrow(opened)                                      # we will take the best maxtrades to
230     if (numtrades>maxdaytrades) {                                # open - we will not exceed maxtrades
231       opened<-opened[c(1:maxdaytrades),]
232       numtrades<-maxdaytrades
233     }
```

We then move forward to split out trade amount equally between all available stocks and invest the amount accordingly to both long and short positions. The function returns a list of a dataframe opened which contains details of the opened positions and the cash value as cashout.

```
    tradeamount<-max(min(maxtrade,equity/numtrades),0)           # invest equally, but not more than we have
    if (numtrades>0&tradeamount>0) {
      opened$position<-ifelse(opened$type=="Long",               # keep a record of the opening price
                              trunc(tradeamount/opened$open),    # and the size of the position, negative
                              -trunc(tradeamount/opened$open))   # position for shorts
      opened$opencash<-ifelse(opened$type=="Long",               # update our cash position
                              opened$buyprice*opened$position,0)
      opened$opencash<-ifelse(opened$type=="Short",
                              opened$sellprice*opened$position,opened$opencash)
      opened<-subset(opened,opened$position!=0)
      cash<-sum(opened$opencash)
    } else {opened<-NULL}
  }
  return(list(opened=opened,cashout=cash))
}
```

## ***Apply Rules***

The apply rules function will check if there are any open positions and close them by running the close position function which applies all of the rules that have been defined. Similarly, the function then checks whether the signal is already open and if not, it will then apply rules and open positions accordingly using the Open positions function.

```
applyRules=function(currdate,equity,position){
  netopen<-position                                  # netopen will hold all open positions after any close orders
  close.results<-closePositions(currdate,equity,position)  # close any orders for which we have positions and signals
  if (!is.null(close.results$closed)) {              # Did we actually close out any positions
    temp<-close.results$close[,c(1,2)]               # if we we need to remove them from our open positions
    names(temp)[2]<-"dummy"                          # we need one field to check if it is empty after the merge
    temp<-merge(position,temp,by="symbol",all.x=TRUE)      # and we don't want to generate duplicate columns, hence dummy
    netopen<-subset(temp,is.na(temp$dummy))          # so if dummy is NA, then the position is not closed
    netopen<-netopen[,c(1:ncol(netopen)-1)]          # get rid of the dummy column
    equity<-equity+close.results$cashin              # update our equity position with the cash from closing
  }
  open.results<-openPositions(currdate,equity,netopen)     # now check for opening new positions
  return(list(open=open.results$opened,close=close.results$closed,
              posnetofcloses=netopen,cashin=close.results$cash,cashout=open.results$cash))
}
```

## ***CALCULATE PORTFOLIO STATISTICS***

This is a function called "portfolioStats" which takes three arguments:

- trades: a data frame containing information about trades (date, type, closedate, etc.)
- pvalue: a numeric vector representing the portfolio value at each point in time
- tdays: a vector of dates representing the time period

Here we perform several calculations and generates a plot to provide information about the portfolio's performance:

- Calculates the number of days traded, the percentage of days traded, and the total number of trades
- Calculates the daily percentage return and separates the trades into short and long positions
- Calculates the cumulative return, maximum return, and maximum value at each point in time using a for loop

- Calculates the drawdown and drawdown percentage, as well as the maximum drawdown and its percentage
- Calculates the average holding period, mean return, and Sharpe ratio
- Generates a plot of the portfolio's cumulative return, maximum return, and daily return over time
- Returns a list of performance metrics including total trades, long and short trades, cumulative return, mean return, Sharpe ratio, maximum drawdown, maximum drawdown percentage, length of the longest drawdown streak, and average holding period

Details:

1. The code calculates some basic statistics related to the trading history, such as the number of unique trading days, total trading days, percentage of days traded, total number of trades made, and the number of short and long trades. The code achieves this by first counting the number of unique trading days in the trades data frame and the total number of trading days in the tdays vector. It then calculates the percentage of days traded by dividing the number of unique trading days by the total number of trading days. Finally, it determines the total number of trades made and the number of short and long trades by subsetting the trades data frame based on the trade type. We also calculate the cumulative return of the portfolio over time and the maximum return achieved up to that point. It does this by initializing two vectors, cumreturn and maxreturn, with length equal to totaldays, and setting all their values to 1. The code then calculates the cumulative return up to each day in the tdays vector using the prod function and saves the result in the corresponding position of the cumreturn vector. It also updates the maxreturn vector with the maximum cumulative return achieved up to that point. Finally, it calculates the percentage drawdown, i.e., the percentage decline in the portfolio's value from its peak, by subtracting maxreturn from cumreturn and dividing the result by maxreturn

```
portfolioStats=function(trades,pvalue,tdays){
  tradedays<-length(unique(trades$date))
  totaldays<-length(tdays)
  pctdaystraded<-tradedays/totaldays
  totaltrades<-nrow(trades)
  pdiff<-c(0,diff(pvalue))
  preturn<-pdiff/pvalue+1
  shorttrades<-nrow(subset(trades,type=="Short"))
  longtrades<-totaltrades-shorttrades
  cumreturn<-rep(1,length(totaldays))
  maxvalue<-cumreturn
  maxreturn<-cumreturn
  for (i in c(1:totaldays)){
    cumreturn[i]<-prod(preturn[c(1:i)],na.rm=TRUE)
    maxreturn[i]<-max(cumreturn[c(1:i)],na.rm=TRUE)
    maxvalue[i]<-max(pvalue[c(1:i)],na.rm=TRUE)
  }
  down<-pvalue-maxvalue
  downpct<-(pvalue-maxvalue)/maxvalue
```

2. This section calculates the maximum drawdown and drawdown duration of the portfolio using a for loop. It initializes two variables, streak and maxstreak, to 0. The for loop iterates over all the days and increments the streak variable if the portfolio return is negative. If the portfolio return is positive or zero, the streak is reset to 0. The maximum streak is updated if the current streak is greater than the previous maximum. Finally, the function plots the cumulative return of the portfolio, maximum return, and portfolio return using the plot and lines functions.

```
streak<-0
maxstreak<-0
for (i in c(1:totaldays)){
  streak<-ifelse(down[i]<0,streak+1,0)
  maxstreak<-ifelse(streak>maxstreak,streak,maxstreak)
}
plot(cumreturn,type="l",col="black",lwd=2,xlab="Time Period",ylim=c(0.5,1.5),
     ylab="Portfolio Return",main="Portfolio Results")
lines(maxreturn,co=2,lw=2)
lines(preturn,co=4,lw=2)
```

3. This section of code computes several performance metrics for a portfolio based on the input of trading data. It calculates the cumulative return of the portfolio, the mean return, the Sharpe ratio, the maximum drawdown, and the length of the longest drawdown streak. The results are stored in a list and returned for further
   Analysis.

```
  cumreturn<-cumreturn[totaldays]
  meanreturn<-mean(preturn,na.rm=TRUE)
  sharpe<-(meanreturn-1)/sd(preturn,na.rm=TRUE)*sqrt(252)
  maxdraw<-min(down)
  maxdrawpct<-min(downpct)*100
  performance<-list(totaltrades=totaltrades,longtrades=longtrades,shorttrades=shorttrades,cumreturn=cumreturn,
                    meanreturn=meanreturn,sharpe=sharpe,maxdraw=maxdraw,maxdrawpct=maxdrawpct,drawlength=maxstreak,
                    meanhold=meanhold)
  return(performance)
}
```

## ***Run Strategy***

Now that we have built the basic framework for our trading strategy, it is now time to run the strategy and test the output. The final output is the performance of the trading strategy in terms of the calculated statistics.

*Implementation*

1. **Data Preperation**

This includes generating indicators for a list of stocks and generating signals required to be able to apply different conditions to run the trading strategy.

```
indicators<-NULL                          # we will put all OHLC data and our generated
for (sym in symbols) {                    # indicators into a dataframe named "indicators"
  temp<-genIndicators(sym)                # by looping through all the symbols in our
  indicators<-rbind(indicators,temp)
}

signals<-NULL                             # signals will be added to indicators and dumped
for (sym in symbols) {                    # into a separate dataframe called "signals"
  temp<-genSignals(sym)
  signals<-rbind(signals,temp)
}
```

- We first create indicator as an empty dataframe that will be used to store the technical indicators generated for each stock.

- Using for loop we iterate over each stock symbol in symbols and generates technical indicators using the genIndicators function explained above.
- If stock is empty, temp is assigned to it. Otherwise, temp is appended to stock using the **rbind()** function.
- Similarly, we generate signals by using a for loop and iterating through each symbol present in the stock universe.

2. **Trading Simulation**

This involves looping through each trading day and applying the trading rules to determine which stocks to buy or sell. We first compute the results for each day by applying rules using the apply rules function and then use that to determine current cash. This current cash is then carried forward to the next day to be able to apply rules using this money. The value pvalue monitors the daily portfolio value which can also be used to check the portfolio value and can be used as a performance metric.

```
tdays<-unique(signals$date)                          # Now process (apply rules) for each trading day in
position<-NULL                                       # order... keeping track of open "positions" and
closed<-NULL                                         # "closed" positions as we proceed.
pvalue<-rep(0,length(tdays))                         # Each day we will keep track of our portfolio value
currentcash<-initialequity                           # that includes current cash, plus our investments.
for (day in 1:length(tdays)) {                       # Now backtest throughout the trading period
  currdate<-tdays[day]
  print(currdate)                                    # simple update to screen on our progress
  results<-applyRules(currdate,currentcash,position) # our state variables are the date and cash available
  position<-rbind(results$posnetofcloses,results$open) # open positions - what we didn't close+ new positions
  closed<-rbind(closed,results$close)                # keep track of all our closed positions
  currentcash<-currentcash+results$cashin-results$cashout  # update our cash position at end of day
  if (!is.null(position)) {                            # update the value of our investments
    temp<-subset(indicators,indicators$date==currdate)[,c(1,6)]
    names(temp)[2]<-"currprice"
    currpos<-merge(position,temp)
    currpos$value<-currpos$position*currpos$currprice
    pvalue[day]<-sum(currpos$value,na.rm=TRUE)          # should not be missing values...
  } else pvalue[day]<-0
  pvalue[day]<-pvalue[day]+currentcash
}
```

3. **Portfolio Evaluation**

This includes calling the **portfolioStats** function to calculate portfolio statistics such as total returns, Sharpe ratio, maximum drawdown, etc. based on the trade information and daily returns stores the results of the evaluation in the **performance** dataframe.

```
performance<-portfolioStats(closed,pvalue,tdays)
performance
```

***Conclusion***

Our strategy of combining two trend system with bollinger bands and using RSI to mitigate risky trades helps to reduce the trades on false signals and hence we a significant improvement in the performance of our trade shown below:

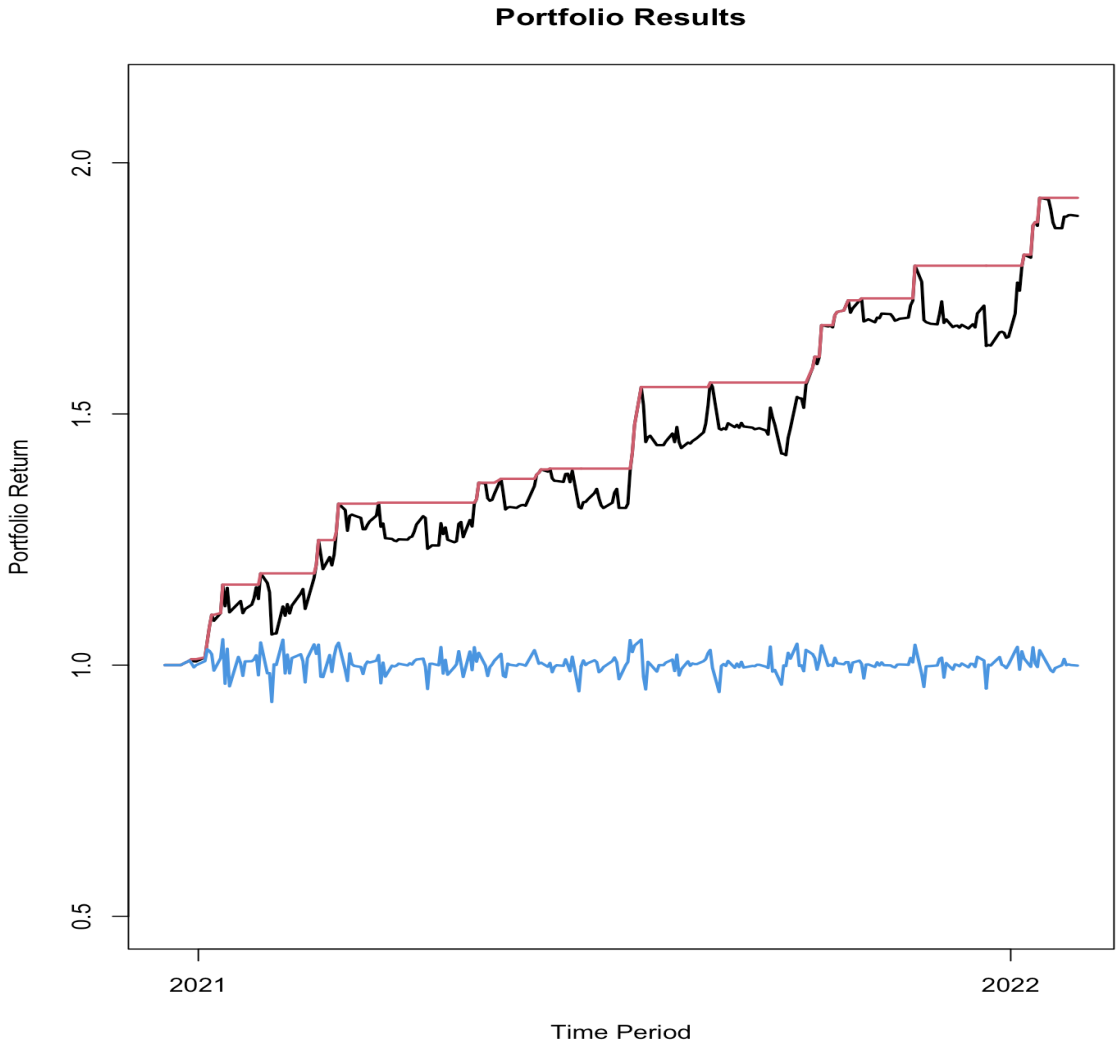
```
$totaltrades
[1] 514

$longtrades
[1] 270

$shorttrades
[1] 244

$cumreturn
[1] 1.894225

$meanreturn
[1] 1.002436

$sharpe
[1] 2.103954

$maxdraw
[1] -16311.26

$maxdrawpct
[1] -9.757428

$drawlength
[1] 31

$meanhold
[1] 7.830739
```

Here's what each metric means and whether it indicates good performance or not:

- `$totaltrades`: This is the total number of trades taken by the strategy. It doesn't provide much information about performance on its own.

- `$longtrades` and `$shorttrades`: These are the number of long and short trades taken by the strategy, respectively. Depending on the strategy, one of these numbers may be higher than the other. Again, this doesn't provide much information about performance on its own.
- `$cumreturn`: This is the cumulative return of the strategy over the period being analyzed. A value of 1.0 would mean that the strategy didn't make or lose any money, while a value higher than 1.0 indicates that the strategy made a profit. In this case, the value is 1.894225, which means that the strategy made a profit.
- `$meanreturn`: This is the average return per trade. A value higher than 1.0 means that the strategy is profitable on average. In this case, the value is 1.002436, which indicates that the strategy is profitable.
- `$sharpe`: This is the Sharpe ratio, which measures the risk-adjusted performance of a strategy. A value higher than 1.0 means that the strategy is generating returns that are higher than its risk. In this case, the value is 2.103954, which is a good Sharpe ratio.
- `$maxdraw`: This is the largest drawdown experienced by the strategy. A drawdown is the amount of money lost from a peak to a trough in the strategy's equity curve. A large drawdown indicates that the strategy is risky. In this case, the value is -16311.26, which is quite large.
- `$maxdrawpct`: This is the percentage of the largest drawdown relative to the starting equity. A value higher than -20% is generally considered to be bad. In this case, the value is -9.757428, which is a relatively small drawdown compared to the starting equity.
- `$drawlength`: This is the length of the longest drawdown period in terms of the number of trades taken. The longer the drawdown period, the more difficult it is for the strategy to recover. In this case, the drawdown period lasted for 31 trades.
- `$meanhold`: This is the average holding period of the trades taken by the strategy. Depending on the strategy, a shorter or longer holding period may be preferable. In this case, the average holding period is 7.830739.

As we can infer from the definitions, the performance of the strategy seems good based on the Sharpe ratio, average return per trade, and cumulative return. However, the large drawdown and relatively long drawdown period suggest that the strategy may be risky and could benefit from risk management techniques.



We also applied our strategy on the “Utilities” sector to check the applicability of our strategy in a different environment.

```
$totaltrades
[1] 700

$longtrades
[1] 408

$shorttrades
[1] 292

$cumreturn
[1] 1.492548

$meanreturn
[1] 1.001465

$sharpe
[1] 2.485747

$maxdraw
[1] -8684.891

$maxdrawpct
[1] -5.677712

$drawlength
[1] 80

$meanhold
[1] 7.7
```

Here again we can see we perform decently in terms of sharpe ratio and our strategy was able to make 49.25% over the course of our trading period.

We also observed the improvement combining the two strategies gave us over just using the two trend system. Attached below is the performance metric for strategy just using the ema signals:

```
$totaltrades
[1] 1200

$longtrades
[1] 635

$shorttrades
[1] 565

$cumreturn
[1] 0.5544338

$meanreturn
[1] 0.9984423

$sharpe
[1] -0.769297

$maxdraw
[1] -53900.98

$maxdrawpct
[1] -46.17607

$drawlength
[1] 265

$meanhold
[1] 5.0525
```

We can clearly see how bad the performance is compared to our strategy of combining multiple signals to perform a trade and using RSI for risk ordering.

## ***ANNEXURE***

```
# ********************************************************************************
#Solution to Homework 4 Team 4
# **************SET WORKING DIRECTORY AND CLEAR ENVIRONMENT *********************
library(rstudioapi)
current_path = rstudioapi::getActiveDocumentContext()$path
setwd(dirname(current_path ))
rm(list=ls())
options(scipen=999)

# ***************GET DATA AND SET TRADING DATE RANGE ***************************

library(quantmod)
library(TTR)
```

```
load("universe.rdata")
from<-as.Date("2020-12-17")
to<-as.Date("2022-12-31")
universe<-subset(universe,universe$Sector=="Energy"&
                 universe$date>=from&universe$date<=to)

################################## Configuration Setting ######################

symbols<-unique(universe$symbol)
initialequity<-100000              # starting money
maxtrade<-11000                    # maximum value of any single trade
maxdaytrades<-11                    # maximum trades in one day

# ************************************* GENERATE INDICATORS ******************
# The indicators for the function are simply the output from the BBands function
# in the TTR library.  Since we are potentially trading on the open based on
# prior days indicators, we need to lag the BBands indicators to they are
# accessible on the day we will trade.
# ****************************************************************************

genIndicators <- function(sym) {
  print(paste('Generating Indicators for symbol: ', sym))

  stock <- subset(universe, universe$symbol == sym)
  stock.xts <- xts(stock[, c(3:7)], stock$date)

  temp.xts <- stock.xts
  temp.xts$high <- stats::lag(stock.xts$high, n = 1)
  temp.xts$low <- stats::lag(stock.xts$low)
  temp.xts$close <- stats::lag(stock.xts$close)

  bb <- tryCatch({
    BBands(HLC(temp.xts), n = 5, maType = "SMA", sd = 2)
  }, warning = function(w) {
    bb <- NULL
  }, error = function(e) {
    bb <- NULL
  })

  if (is.null(bb)) {
    stock.xts$dn <- NA
    stock.xts$mavg <- NA
    stock.xts$up <- NA
    stock.xts$range <- NA
  } else {
    stock.xts$dn <- bb$dn
```

```
  stock.xts$mavg <- bb$mavg
  stock.xts$up <- bb$up
 }

 ema5 <- tryCatch({
  ema5 <- EMA(temp.xts$close, n = 5)
 }, warning=function(w) {ema5 <- NULL}, error=function(e) {ema5 <- NULL})

 ema10 <- tryCatch({
  ema10 <- EMA(temp.xts$close, n = 10)
 }, warning=function(w) {ema10 <- NULL}, error=function(e) {ema10 <- NULL})

 if (is.null(ema5) | is.null(ema10)) {
  stock.xts$ema5 <- NA
  stock.xts$ema10 <- NA
 } else {
  stock.xts$ema5 <- ema5$EMA
  stock.xts$ema10 <- ema10$EMA
 }

 stock.xts$rsi <- RSI(stock.xts$close, n = 14)
 stock.xts$range<-stock.xts$rsi
 stock <- data.frame(stock.xts)
 date <- as.Date(rownames(stock))
 stock <- cbind(date, stock)
 stock <- cbind(sym, stock)
 names(stock)[1] <- "symbol"
 rownames(stock) <- seq(1, nrow(stock), 1)

 return(stock)
}


# ************************************ GENERATE SIGNALS
*****************************************************************
# Generate signals is a function that will check for any cross overs.  We have
# a sell short signal if price closes below the lower band.  We have a go long
# signal if prices closes above the upper band.  For exiting, we can check to
# see which side of the trend line prices close.  But we will have to check our
# position (long or short) to see if price crossed the trend line against the
# trend.  We will do this latter part when we apply rules.
# ********************************************************************************
genSignals <- function(sym) {
 print(paste('Generating Signals for symbol: ',sym))
 stock <- subset(indicators, indicators$symbol == sym)
 stock.xts <- xts(stock[, c(3:ncol(stock))], stock$date)
```

```
 # Generate Bollinger Bands signals
 stock.xts$cross.upper <- ifelse(stock.xts$close > stock.xts$up, 1, 0)
 stock.xts$cross.lower <- ifelse(stock.xts$close < stock.xts$dn, 1, 0)
 stock.xts$cross.trendup <- ifelse(stock.xts$close > stock.xts$mavg, 1, 0)
 stock.xts$cross.trenddn <- ifelse(stock.xts$close < stock.xts$mavg, 1, 0)

 #Generate EMA
 stock.xts$cross.upper_ema <-
  ifelse((stock.xts$close > stock.xts$ema10) & (stock.xts$ema5 > stock.xts$ema10), 1, 0)  # Entry for Buy
 stock.xts$cross.trenddn_ema <-
  ifelse(stock.xts$close < stock.xts$ema5, 1, 0)  # Buy order Stop Loss or Exit at Profit

 stock.xts$cross.lower_ema <- ifelse((stock.xts$close < stock.xts$ema10) &
                              (stock.xts$ema5 < stock.xts$ema10), 1, 0)  # Entry for Sell
 stock.xts$cross.trendup_ema <-
  ifelse(stock.xts$close > stock.xts$ema5, 1, 0)  # Sell

 # Generate RSI signals
 stock.xts$rsi <- RSI(stock.xts$close, n = 10)
 stock.xts$rsi.buy <- ifelse(stock.xts$rsi <= 30 | stock.xts$rsi <= 75, 1, 0)
 stock.xts$rsi.sell <- ifelse(stock.xts$rsi >= 30 | stock.xts$rsi >= 75, 1, 0)


 stock <- data.frame(stock.xts)
 date <- as.Date(rownames(stock))
 stock <- cbind(date, stock)
 stock <- cbind(sym, stock)
 names(stock)[1] <- "symbol"
 rownames(stock) <- seq(1, nrow(stock), 1)
 return(stock)
}


# **************************CLOSE POSITIONS ************************************
# Here we will check our exit signals and compare them to the list of open
# positions, separately for longs and shorts.  If we have a short position
# and price closes above the trend line, then we close it.   If we have a long
# position and prices closes below the trend line, then we close it.  Note we
# will note simultaneously hold long and short positions with this strategy.
# We will only open if we don't already have an open position in a stock.
# ******************************************************************************
closePositions=function(day,equity,position){
 cash<-0
 closed<-NULL
 if (!is.null(position)) {
```

```
   longposition<-subset(position,type=="Long")             # check long and short separately
   shortposition<-subset(position,type=="Short")
   candidates<-subset(signals,signals$date==day&           # check shorts first
                      ((signals$cross.trendup==1) & (signals$cross.trendup_ema==1)))[,c(1,2,6)] # grab symbol (1),
date(2), and price (6)
   names(candidates)[2]<-"closedate"                       # keep track of the close date so we
   names(candidates)[3]<-"outprice"                        # can check how long we hold our positions
   closeshort<-merge(shortposition,candidates,by="symbol")  # Close only if we have a position
   candidates<-subset(signals,signals$date==day&           # Now do the same for longs
                      ((signals$cross.trenddn==1) & (signals$cross.trenddn_ema==1)))[,c(1,2,6)]
   names(candidates)[2]<-"closedate"
   names(candidates)[3]<-"outprice"
   closelong<-merge(longposition,candidates,by="symbol")
   closed<-rbind(closeshort,closelong)                     # put all our positions to close together
   if (nrow(closed)>0) {
     closed$closecash<-closed$outprice*closed$position     # compute closing calculations of cash
     closed$sellprice<-ifelse(closed$type=="Long",closed$outprice,closed$sellprice)
     closed$buyprice<-ifelse(closed$type=="Short",closed$outprice,closed$buyprice)
     closed$profit<-(closed$sellprice-closed$buyprice)*abs(closed$position)
     cash<-sum(closed$closecash)   # get the aggregate value to add back to currentcash
   } else closed<-NULL
  }
  return(list(closed=closed,cashin=cash))
}

# ************************** OPEN POSITIONS ***********************************
# Now we are going to check our entry signals and only enter a position if we
# don't already have a position in the stock.  So we have a signal to open, we
# need to check for the absence of the position in the set of open positions
# *****************************************************************************
#day<-currdate
#position<-netopen
openPositions=function(day,equity,position){
  cash=0
  opened<-NULL
  if (!is.null(position)) {                          # only need to check if we have open positions
    longposition<-subset(position,type=="Long")[,c(1,2)]        # check long and shorts separately
    names(longposition)[2]<-"dummy"                            # use dummy again, see Apply Rules function
    shortposition<-subset(position,type=="Short")[,c(1,2)]      # for further explanation
    names(shortposition)[2]<-"dummy"
    candidates<-subset(signals,signals$date==day&              # check shorts first
                       signals$cross.lower==1 & (signals$cross.lower_ema==1))
    temp<-merge(candidates,shortposition,by="symbol",all.x=TRUE)
    openshort<-subset(temp,is.na(dummy))                       # only short if we don't have a position
    if (nrow(openshort)>0) {
      openshort<-openshort[,c(1:ncol(openshort)-1)]            # get rid of dummy column
```

```
    openshort$type<-"Short"                          # we will open a short position
   } else {openshort<-NULL}                          # if the dataframe is empty, set it to null
   candidates<-subset(signals,signals$date==day&             # now proceed and do same for longs
                      signals$cross.upper==1 & (signals$cross.upper_ema==1))
   temp<-merge(candidates,longposition,by="symbol",all.x=TRUE)
   openlong<-subset(temp,is.na(dummy))
   if (nrow(openlong)>0) {
    openlong<-openlong[,c(1:ncol(openlong)-1)]
    openlong$type<-"Long"
   } else {openlong<-NULL}
   opened<-rbind(openlong,openshort)                     # put all positions to be opened together
   if (!is.null(opened)) {                           # convert empty dataframe to null
    if (nrow(opened)==0) opened<-NULL                    # so we don't have to check for both !null
   }                                                 # and that the number of rows>0
  } else {
   opened<-subset(signals,signals$date==day&                # no open positions so grab all signals to open
                  (signals$cross.lower==1|signals$cross.upper==1))
   if (nrow(opened)==0) {opened<-NULL} else {
    opened$type<-ifelse(opened$cross.lower==1,           # set the type of position (long, short)
                        "Short","Long")}
  }
  if (!is.null(opened)) {                            # open if we have positions to open
   opened$buyprice<-ifelse(opened$type=="Long",opened$open,NA)
   opened$sellprice<-ifelse(opened$type=="Short",opened$open,NA)
   opened<-opened[order(opened$range),]                  # sort them by the risk
   numtrades<-nrow(opened)                           # we will take the best maxtrades to
   if (numtrades>maxdaytrades) {                      # open - we will not exceed maxtrades
    opened<-opened[c(1:maxdaytrades),]
    numtrades<-maxdaytrades
   }
   tradeamount<-max(min(maxtrade,equity/numtrades),0)       # invest equally, but not more than we have
   if (numtrades>0&tradeamount>0) {
    opened$position<-ifelse(opened$type=="Long",         # keep a record of the opening price
                            trunc(tradeamount/opened$open),  # and the size of the position, negative
                            -trunc(tradeamount/opened$open))  # position for shorts
    opened$opencash<-ifelse(opened$type=="Long",         # update our cash position
                            opened$buyprice*opened$position,0)
    opened$opencash<-ifelse(opened$type=="Short",
                            opened$sellprice*opened$position,opened$opencash)
    opened<-subset(opened,opened$position!=0)
    cash<-sum(opened$opencash)
   } else {opened<-NULL}
  }
  return(list(opened=opened,cashout=cash))
}
```

```
# ************************************** APPLY RULES *************************
# Apply rules will first check the signals and apply rules for closing out
# positions then open any new positions.  We  won't add to existing positions so
# we will only open if we don't already have an open position in a stock.
# ****************************************************************************
#results<-applyRules(currdate,currentcash,position)    # our state variables are the date and cash available
#equity<-currentcash

applyRules=function(currdate,equity,position){
  netopen<-position                                  # netopen will hold all open positions after any close orders
  close.results<-closePositions(currdate,equity,position)  # close any orders for which we have positions and
signals
  if (!is.null(close.results$closed)) {                # Did we actually close out any positions
    temp<-close.results$close[,c(1,2)]                   # if we we need to remove them from our open positions
    names(temp)[2]<-"dummy"                              # we need one field to check if it is empty after the merge
    temp<-merge(position,temp,by="symbol",all.x=TRUE)      # and we don't want to generate duplicate columns,
hence dummy
    netopen<-subset(temp,is.na(temp$dummy))              # so if dummy is NA, then the position is not closed
    netopen<-netopen[,c(1:ncol(netopen)-1)]              # get rid of the dummy column
    equity<-equity+close.results$cashin                  # update our equity position with the cash from closing
  }
  open.results<-openPositions(currdate,equity,netopen)     # now check for opening new positions
  return(list(open=open.results$opened,close=close.results$closed,
              posnetofcloses=netopen,cashin=close.results$cash,cashout=open.results$cash))
}
# ************************** CALCULATE PORTFOLIO STATISTICS ********************
# Calculate various portfolio statistics such as period returns, cumulative
# returns, number of trades, max drawdown period, max drawdown percent, and
# annualized sharpe ratio.  The only change from what we have seen before
# with backtestings is that there is a new calculation to compute the mean
# number of days that we hold a position.  This provides an indicate of trading
# frequency.
# ****************************************************************************
portfolioStats=function(trades,pvalue,tdays){
  tradedays<-length(unique(trades$date))
  totaldays<-length(tdays)
  pctdaystraded<-tradedays/totaldays
  totaltrades<-nrow(trades)
  pdiff<-c(0,diff(pvalue))
  preturn<-pdiff/pvalue+1
  shorttrades<-nrow(subset(trades,type=="Short"))
  longtrades<-totaltrades-shorttrades
  cumreturn<-rep(1,length(totaldays))
  maxvalue<-cumreturn
  maxreturn<-cumreturn
```

```
 for (i in c(1:totaldays)){
  cumreturn[i]<-prod(preturn[c(1:i)],na.rm=TRUE)
  maxreturn[i]<-max(cumreturn[c(1:i)],na.rm=TRUE)
  maxvalue[i]<-max(pvalue[c(1:i)],na.rm=TRUE)
 }
 down<-pvalue-maxvalue
 downpct<-(pvalue-maxvalue)/maxvalue
 streak<-0
 maxstreak<-0
 for (i in c(1:totaldays)){
  streak<-ifelse(down[i]<0,streak+1,0)
  maxstreak<-ifelse(streak>maxstreak,streak,maxstreak)
 }
 maxy<-max(cumreturn+0.2)
 plot(tdays,cumreturn,type="l",col="black",lwd=2,xlab="Time Period",ylim=c(0.5,maxy),ylab="Portfolio
Return",main="Portfolio Results")
 lines(tdays,maxreturn,co=2,lw=2)
 lines(tdays,preturn,co=4,lw=2)
 trades$holdperiod<-as.numeric(trades$closedate-trades$date)
 meanhold<-mean(trades$holdperiod,na.rm=TRUE)
 cumreturn<-cumreturn[totaldays]
 meanreturn<-mean(preturn,na.rm=TRUE)
 sharpe<-(meanreturn-1)/sd(preturn,na.rm=TRUE)*sqrt(252)
 maxdraw<-min(down)
 maxdrawpct<-min(downpct)*100

performance<-list(totaltrades=totaltrades,longtrades=longtrades,shorttrades=shorttrades,cumreturn=cumretu
rn,

meanreturn=meanreturn,sharpe=sharpe,maxdraw=maxdraw,maxdrawpct=maxdrawpct,drawlength=maxstrea
k,
              meanhold=meanhold)
 return(performance)
}


# ********************************  RUN STRATEGY **********************************
indicators<-NULL                      # we will put all OHLC data and our generated
for (sym in symbols) {                 # indicators into a dataframe named "indicators"
 temp<-genIndicators(sym)                  # by looping through all the symbols in our
 indicators<-rbind(indicators,temp)
}

signals<-NULL                        # signals will be added to indicators and dumped
for (sym in symbols) {                 # into a separate dataframe called "signals"
 temp<-genSignals(sym)
```

```
  signals<-rbind(signals,temp)
}

tdays<-unique(signals$date)                    # Now process (apply rules) for each trading day in
position<-NULL                                 # order... keeping track of open "positions" and
closed<-NULL                                   # "closed" positions as we proceed.
pvalue<-rep(0,length(tdays))                   # Each day we will keep track of our portfolio value
currentcash<-initialequity                     # that includes current cash, plus our investments.
for (day in 1:length(tdays)) {                 # Now backtest throughout the trading period
  currdate<-tdays[day]
  print(currdate)                              # simple update to screen on our progress
  results<-applyRules(currdate,currentcash,position)    # our state variables are the date and cash available
  position<-rbind(results$posnetofcloses,results$open)  # open positions - what we didn't close+ new positions
  closed<-rbind(closed,results$close)          # keep track of all our closed positions
  currentcash<-currentcash+results$cashin-results$cashout  # update our cash position at end of day
  if (!is.null(position)) {                    # update the value of our investments
    temp<-subset(indicators,indicators$date==currdate)[,c(1,6)]
    names(temp)[2]<-"currprice"
    currpos<-merge(position,temp)
    currpos$value<-currpos$position*currpos$currprice
    pvalue[day]<-sum(currpos$value,na.rm=TRUE)          # should not be missing values...
  } else pvalue[day]<-0
  pvalue[day]<-pvalue[day]+currentcash
}

performance<-portfolioStats(closed,pvalue,tdays)
performance
```