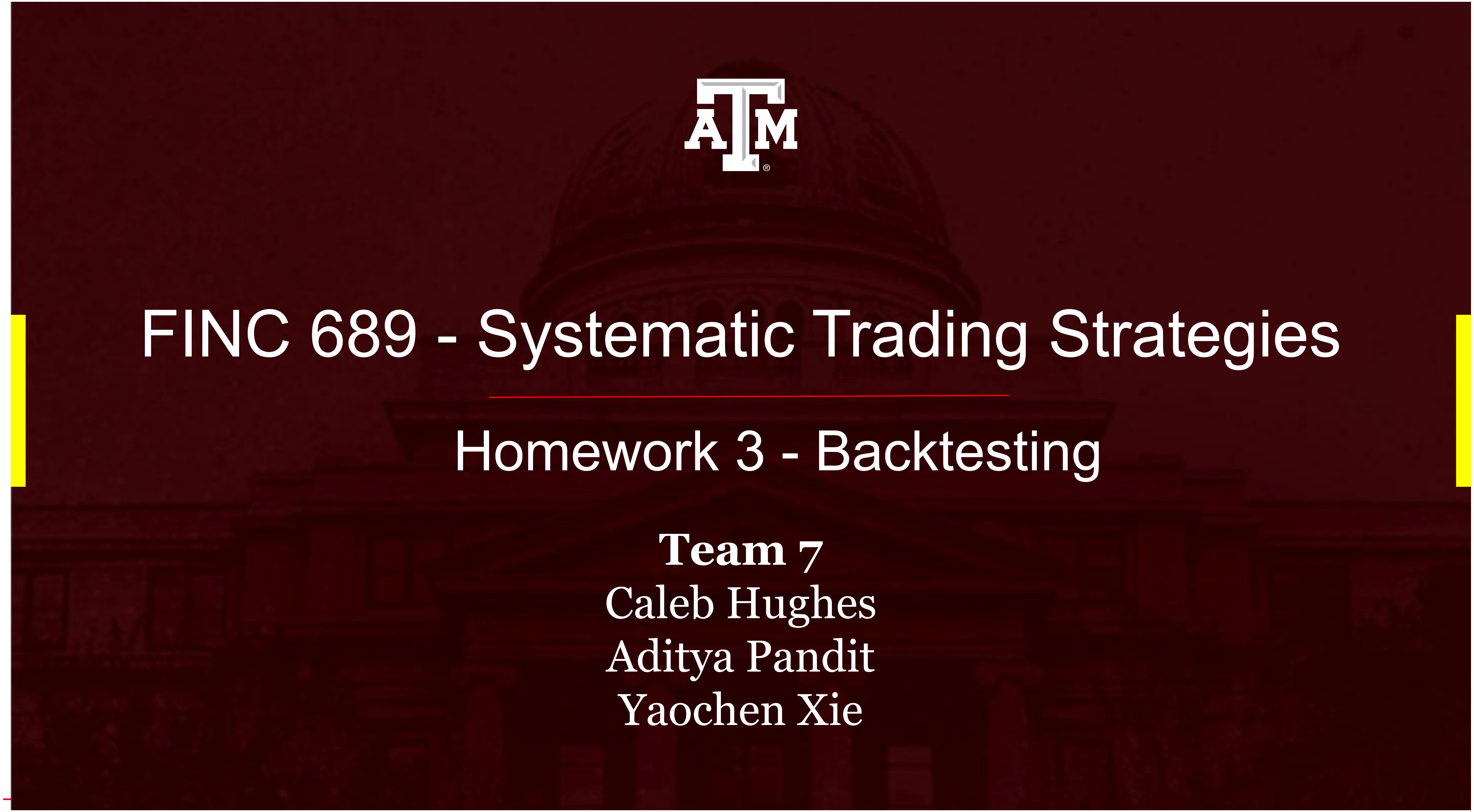


FINC 689 - Systematic Trading Strategies

Homework 3 - Backtesting

Team 7
Caleb Hughes
Aditya Pandit
Yaochen Xie

Index

- Overview
- Learning Outcomes
- Generating signals using time series model
- Framing rules for trading strategy
- Back testing and Performance evaluation
- Conclusion
- Q&A

Overview

- We will use the data from S&P 500 stocks in energy sector to develop and implement a trading strategy.
- We will test multiple time series analysis methods to predict prices based on historical data in a recursive manner
- We will back test the strategy on data of the past 2 years and against some developed rules and constraints
- We evaluate the performance of the strategy based on some financial metrics such as Sharpe ratio, total returns etc.

Learning Outcomes

- Develop a predictable and accurate trading strategy which works on new price data available.
- Filter model predictions based on return level and confidence intervals based on time series models for price prediction.
- Weighing the investments based on predictability and constraints applied as rules.
- Back testing the strategy on the data in subsequent periods in R to calculate metrics on the performance of the trading strategy.
- Evaluating the strategy based on financial metrics which can provide information on the performance and predictability.

Outline of tutorial

- Data preprocessing
- Generating signals/indicators
- Rule-based trading strategy
- Running the strategy, backtesting, and performance evaluation

Developing stock trading strategy

Developing a trading strategy

- A stock trading strategy is a set of rules and guidelines used to make investment decisions in the stock market. It's designed to help traders identify profitable opportunities and manage risk.
- A stock trading strategy should be tested and refined over time using back testing and forward testing to evaluate its effectiveness
- To generate predictions for the strategy we will use financial time series models over our data
- Financial time series models are statistical techniques to identify patterns, trends and relationships in financial data and forecast values
- Some examples of financial time series models are moving average, exponential smoothing, ARIMA, Holt-Winter’s method etc.

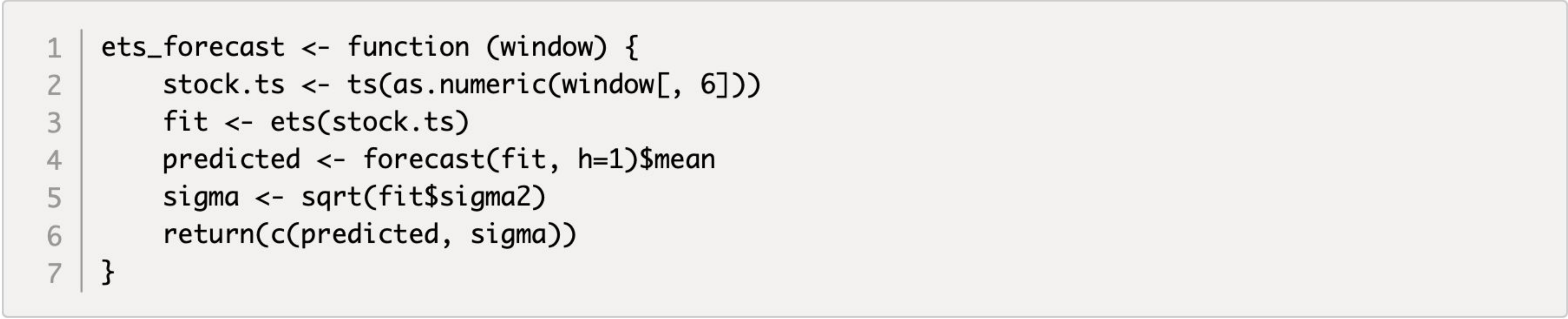
Choice of time series model

- Of the financial time series techniques available, we will choose exponential smoothing method for the following reasons:
  - The method is capable of handling daily/weekly/monthly fluctuations in the prices of the stock
  - Incorporates the trend in the model which enables forecasting prices of stocks which exhibit a trend over time
  - Suitable for forecasting over shorter periods of time since it assigns greater weight to recent observations capturing recent trend.
  - Due to the use of weighted average of the past observation, the impact of outliers and noise in the data is reduced and high leverage points do not distort the forecast.

Implementation in R

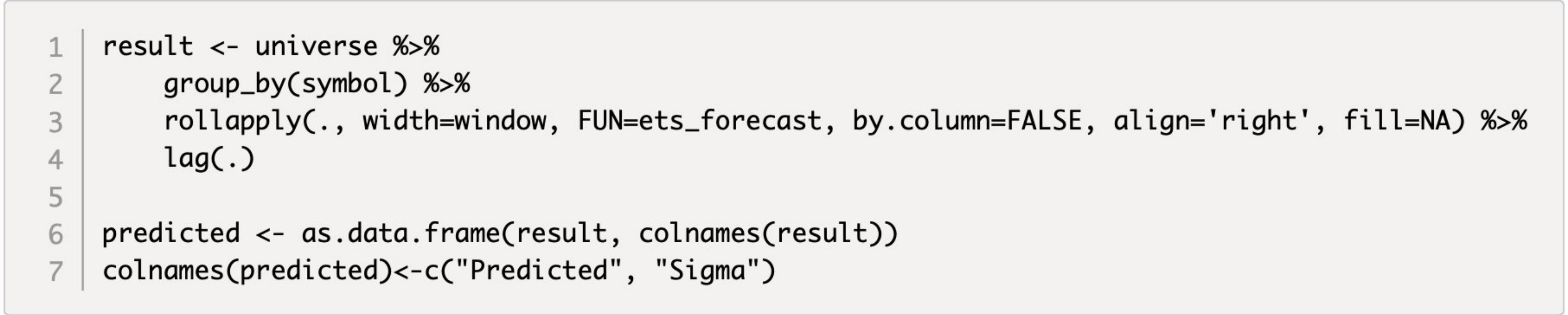
- Using library “forecast”, the function “ets” (Exponential Time Smoothing) is imported into our program.
- In the model parameter, we use to default parameters to implement the exponential smoothing function.
- The mean and the standard deviation of the predicted values is obtained to calculate the confidence interval.
- Calculation of the confidence interval enables us to get the data on which we implement our rules.

Implementation in R



```r
ets_forecast <- function (window) {
    stock.ts <- ts(as.numeric(window[, 6]))
    fit <- ets(stock.ts)
    predicted <- forecast(fit, h=1)$mean
    sigma <- sqrt(fit$sigma2)
    return(c(predicted, sigma))
}
```

Code Snippet: Developing a time series prediction function in R



```r
result <- universe %>%
    group_by(symbol) %>%
    rollapply(., width=window, FUN=ets_forecast, by.column=FALSE, align='right', fill=NA) %>%
    lag(.)

predicted <- as.data.frame(result, colnames(result))
colnames(predicted)<-c("Predicted", "Sigma")
```

Code Snippet: Application of the function to the data and displaying results as a column

Rules based on indicators

- We developed a system of rules to trade based on the indicators with the goal of investing in **predictable stocks**
- Using our exponential smoothing time-series method, we are able to generate indicators for stocks:
  - Predicted - predicted value of stock price
  - Sigma - standard deviation of stock price

Rules based on indicators

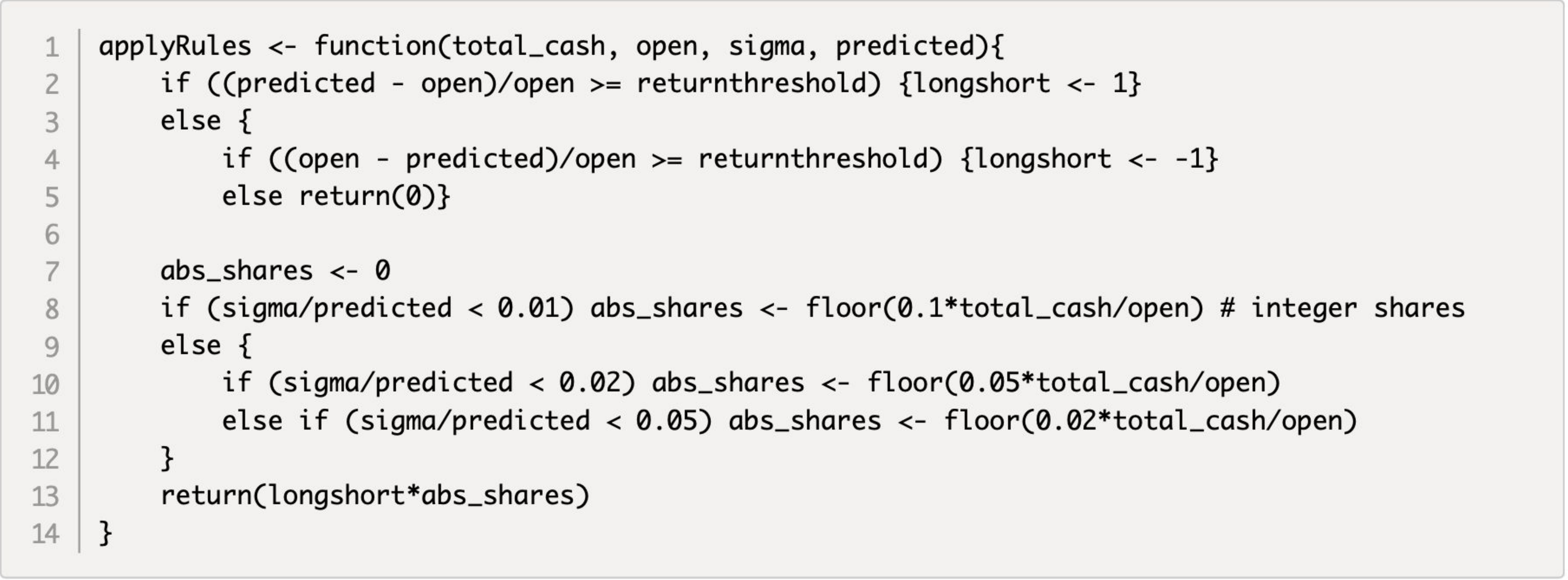
- As our predicted values were given at 95% confidence, we satisfy our overall trading strategy of investing in predictable stocks with high confidence
- Our first rule was to filter and sort our predictions by a given return threshold (between open and predicted values)
- Stocks that exceeded this threshold positively were sorted into a “long” group, while exceeding the threshold negatively resulted in a “short” designation

Rules based on indicators

- In order to allocate the most capital to the most predictable stocks, we used the sigma indicator to determine the number of shares to be traded.
- For an individual stock, if (stock sigma/predicted price) is less than .01, we invest 10% of capital
- If (stock sigma/predicted price) is between .01 and .02, we invest 5% of capital
- If (stock sigma/predicted price) is between .02 and .05, we invest 2% of capital

Rules based on indicators - R

- Our applyrules function takes in our stock indicators and outputs the number of shares to buy



```r
applyRules <- function(total_cash, open, sigma, predicted){
    if ((predicted - open)/open >= returnthreshold) {longshort <- 1}
    else {
        if ((open - predicted)/open >= returnthreshold) {longshort <- -1}
        else return(0)}

    abs_shares <- 0
    if (sigma/predicted < 0.01) abs_shares <- floor(0.1*total_cash/open) # integer shares
    else {
        if (sigma/predicted < 0.02) abs_shares <- floor(0.05*total_cash/open)
        else if (sigma/predicted < 0.05) abs_shares <- floor(0.02*total_cash/open)
    }
    return(longshort*abs_shares)
}
```

Apply constraints, adjustments, and run the strategy

Maximum long/short trades

- We are given the maximum number of long/short trades per day, *e.g.*, *K long trades* and *L short trades*. When the strategy gives more qualified trades, we select the top-K long and top-L short trades based on the expected return of each trade on the day.

Maximum fund per day

- To control the risk, we want to make sure the invested fund per day does not exceed *a certain cap T*. When the total required funds by qualified trades given by the strategy exceed the cap, we linearly shrunk every trade.
- In our case, we let the maximum fund to be proportional to the total fund we have (it can also be constant). It will increase as we profit over time.

The pipeline to enforce constraints

TEXAS A&M UNIVERSITY

1. Compute all candidate trades and their expected return

⬇️

2. Select top-K/L trades with the highest expected return
(and update trades in the dataframe)

⬇️

3. Shrunk the traded shares linearly based on updated trades
(and update trades in the dataframe)

⬇️

4. Compute actual return and update total fund/max fund

The pipeline to enforce constraints

1. Compute all candidate trades and their expected return

⬇️

2. Select top-K/L trades with the highest expected return (and update trades in the dataframe)

⬇️

3. Shrunk the traded shares linearly based on updated trades (and update trades in the dataframe)

⬇️

4. Compute actual return and update total fund/max fund

The pipeline to enforce constraints

1. Compute all candidate trades and their expected return

⬇️

2. Select top-K/L trades with the highest expected return
(and update trades in the dataframe)

⬇️

3. Shrunk the traded shares linearly based on updated trades
(and update trades in the dataframe)

⬇️

4. Compute actual return and update total fund/max fund

The pipeline to enforce constraints

1. Compute all candidate trades and their expected return

⬇️

2. Select top-K/L trades with the highest expected return
(and update trades in the dataframe)

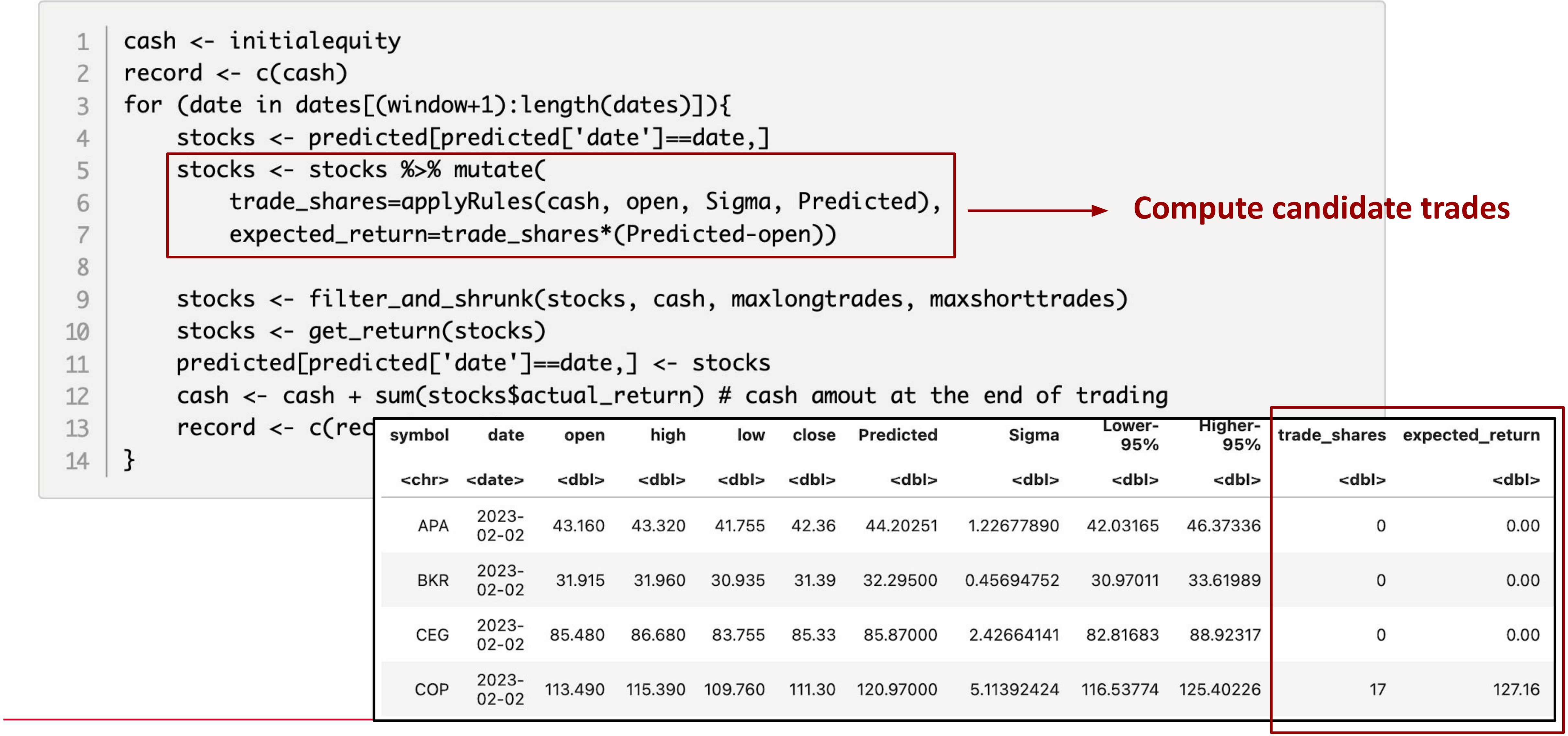
⬇️

3. Shrunk the traded shares linearly based on updated trades
(and update trades in the dataframe)

⬇️

4. Compute actual return and update total fund/max fund

Pipeline implementation



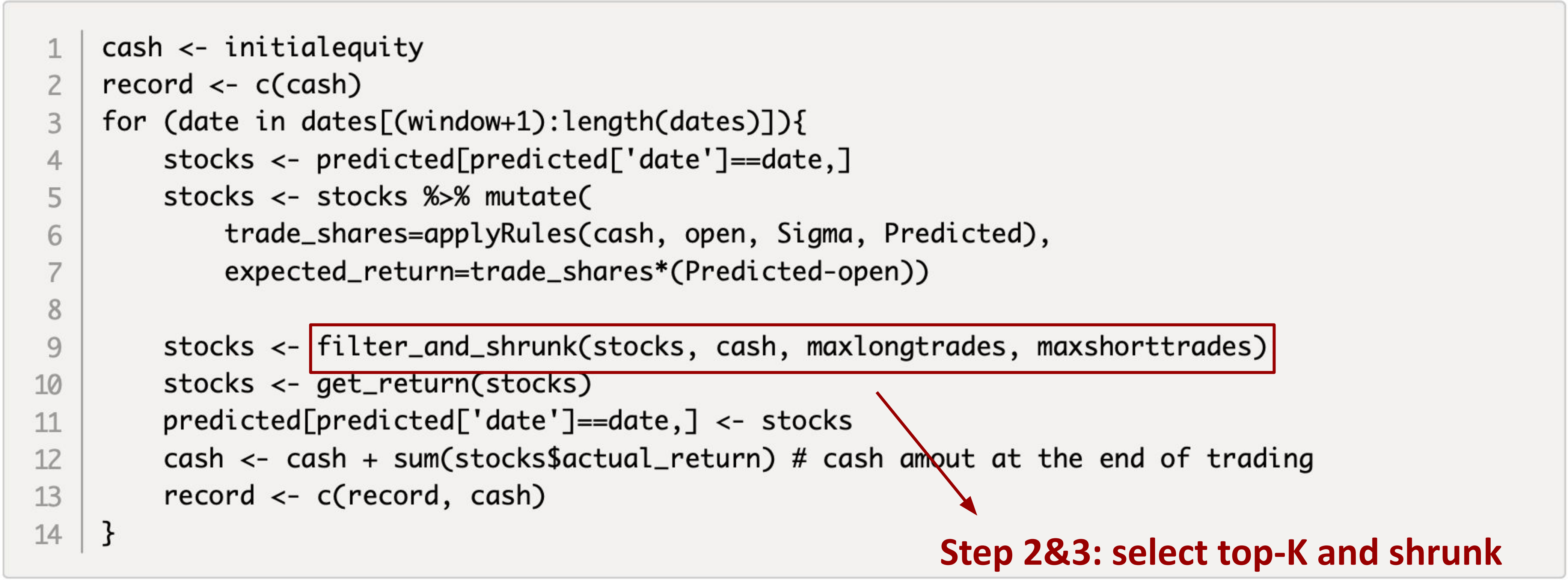
```
cash <- initialequity
record <- c(cash)
for (date in dates[(window+1):length(dates)]){
    stocks <- predicted[predicted['date']==date,]
    stocks <- stocks %>% mutate(
        trade_shares=applyRules(cash, open, Sigma, Predicted),
        expected_return=trade_shares*(Predicted-open))

    stocks <- filter_and_shrunk(stocks, cash, maxlongtrades, maxshorttrades)
    stocks <- get_return(stocks)
    predicted[predicted['date']==date,] <- stocks
    cash <- cash + sum(stocks$actual_return) # cash amout at the end of trading
    record <- c(rec
}
```

Compute candidate trades

symbol	date	open	high	low	close	Predicted	Sigma	Lower-95%	Higher-95%	trade_shares	expected_return
<chr>	<date>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
APA	2023-02-02	43.160	43.320	41.755	42.36	44.20251	1.22677890	42.03165	46.37336	0	0.00
BKR	2023-02-02	31.915	31.960	30.935	31.39	32.29500	0.45694752	30.97011	33.61989	0	0.00
CEG	2023-02-02	85.480	86.680	83.755	85.33	85.87000	2.42664141	82.81683	88.92317	0	0.00
COP	2023-02-02	113.490	115.390	109.760	111.30	120.97000	5.11392424	116.53774	125.40226	17	127.16

Pipeline implementation

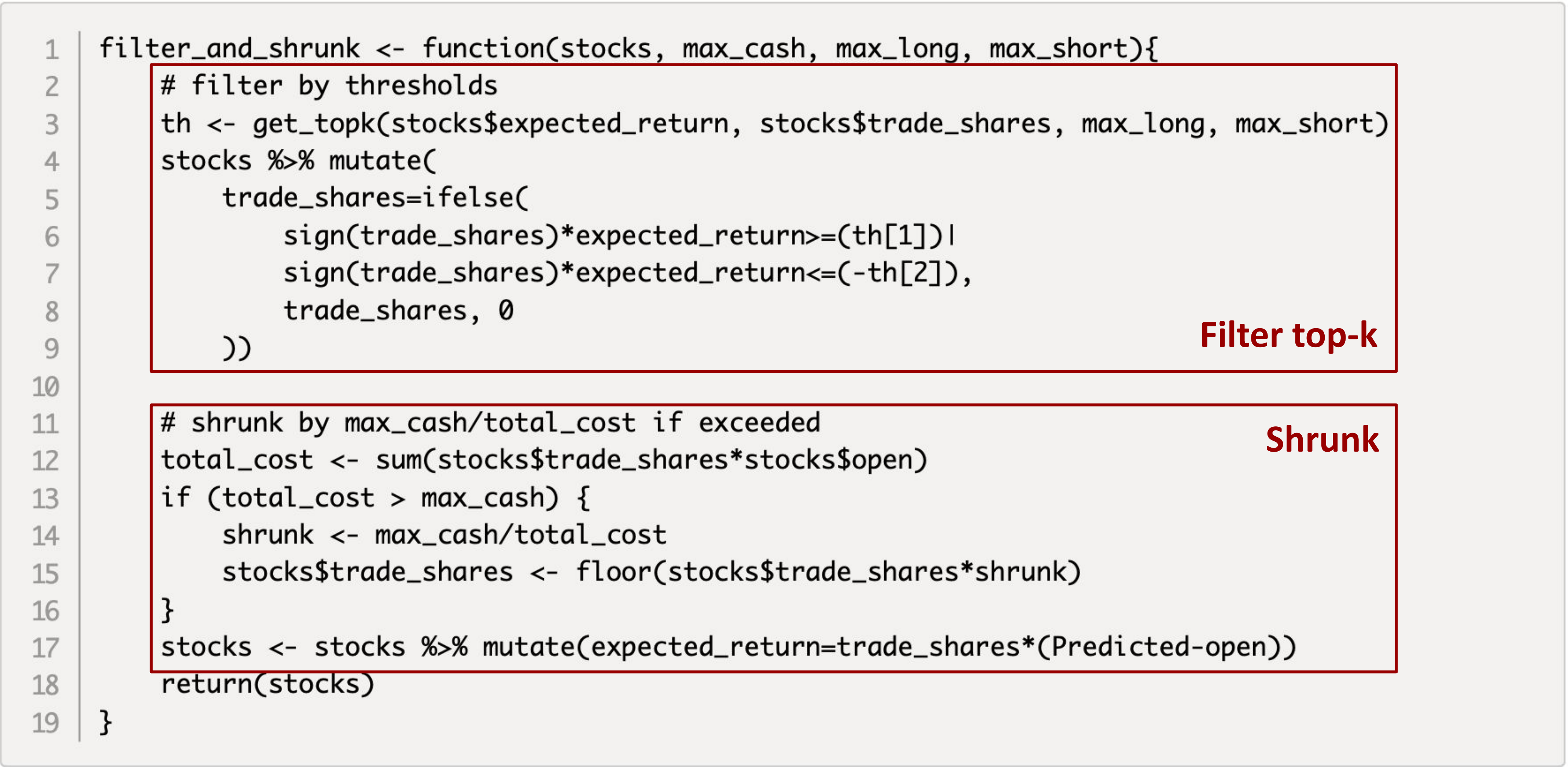


```
cash <- initialequity
record <- c(cash)
for (date in dates[(window+1):length(dates)]){
    stocks <- predicted[predicted['date']==date,]
    stocks <- stocks %>% mutate(
        trade_shares=applyRules(cash, open, Sigma, Predicted),
        expected_return=trade_shares*(Predicted-open))

    stocks <- filter_and_shrunk(stocks, cash, maxlongtrades, maxshorttrades)
    stocks <- get_return(stocks)
    predicted[predicted['date']==date,] <- stocks
    cash <- cash + sum(stocks$actual_return) # cash amout at the end of trading
    record <- c(record, cash)
}
```

Step 2&3: select top-K and shrunk

Pipeline implementation



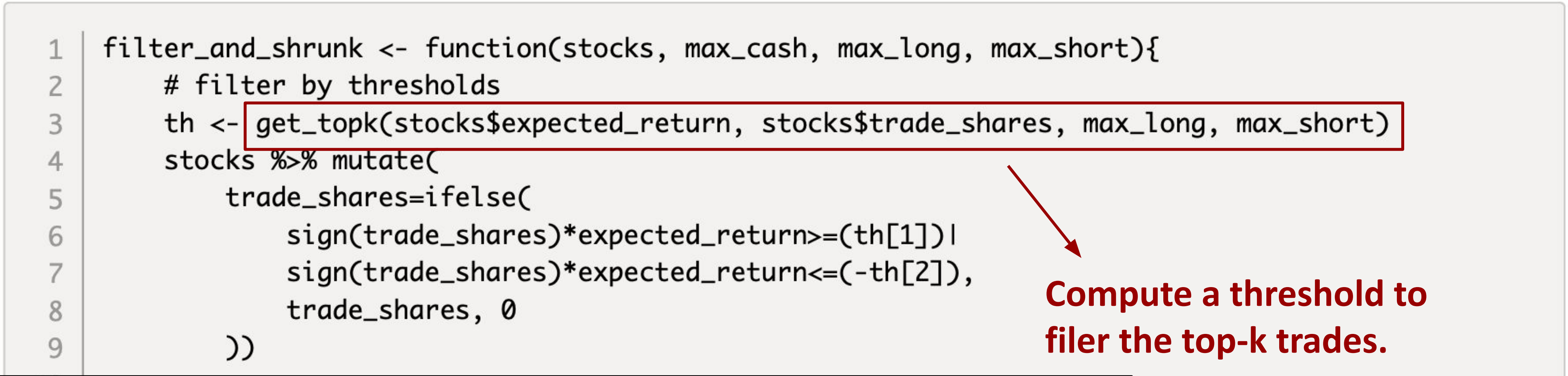
```
filter_and_shrunk <- function(stocks, max_cash, max_long, max_short){
    # filter by thresholds
    th <- get_topk(stocks$expected_return, stocks$trade_shares, max_long, max_short)
    stocks %>% mutate(
        trade_shares=ifelse(
            sign(trade_shares)*expected_return>=(th[1])|
            sign(trade_shares)*expected_return<=(-th[2]),
            trade_shares, 0
        ))

    # shrunk by max_cash/total_cost if exceeded
    total_cost <- sum(stocks$trade_shares*stocks$open)
    if (total_cost > max_cash) {
        shrunk <- max_cash/total_cost
        stocks$trade_shares <- floor(stocks$trade_shares*shrunk)
    }
    stocks <- stocks %>% mutate(expected_return=trade_shares*(Predicted-open))
    return(stocks)
}
```

Filter top-k

Shrunk

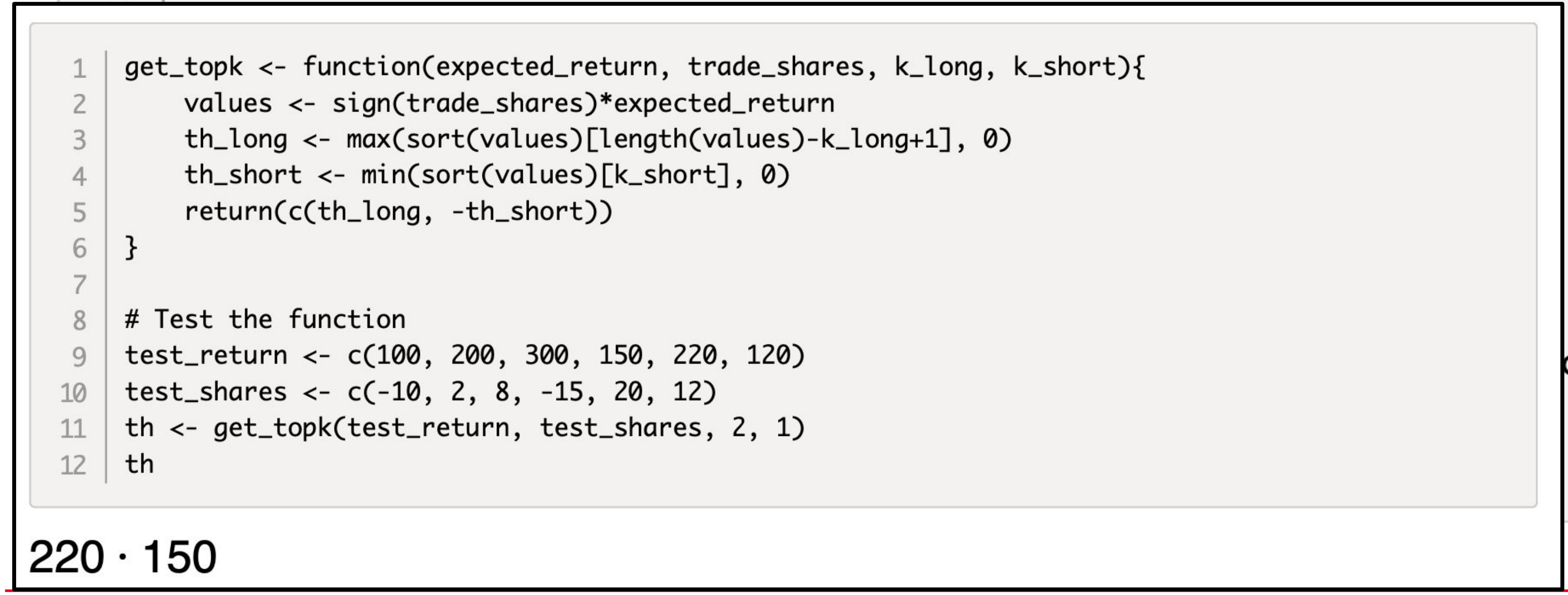
Pipeline implementation



```
filter_and_shrunk <- function(stocks, max_cash, max_long, max_short){
    # filter by thresholds
    th <- get_topk(stocks$expected_return, stocks$trade_shares, max_long, max_short)
    stocks %>% mutate(
        trade_shares=ifelse(
            sign(trade_shares)*expected_return>=(th[1])|
            sign(trade_shares)*expected_return<=(-th[2]),
            trade_shares, 0
        ))
```

```
cted-open))
```

Compute a threshold to filer the top-k trades.

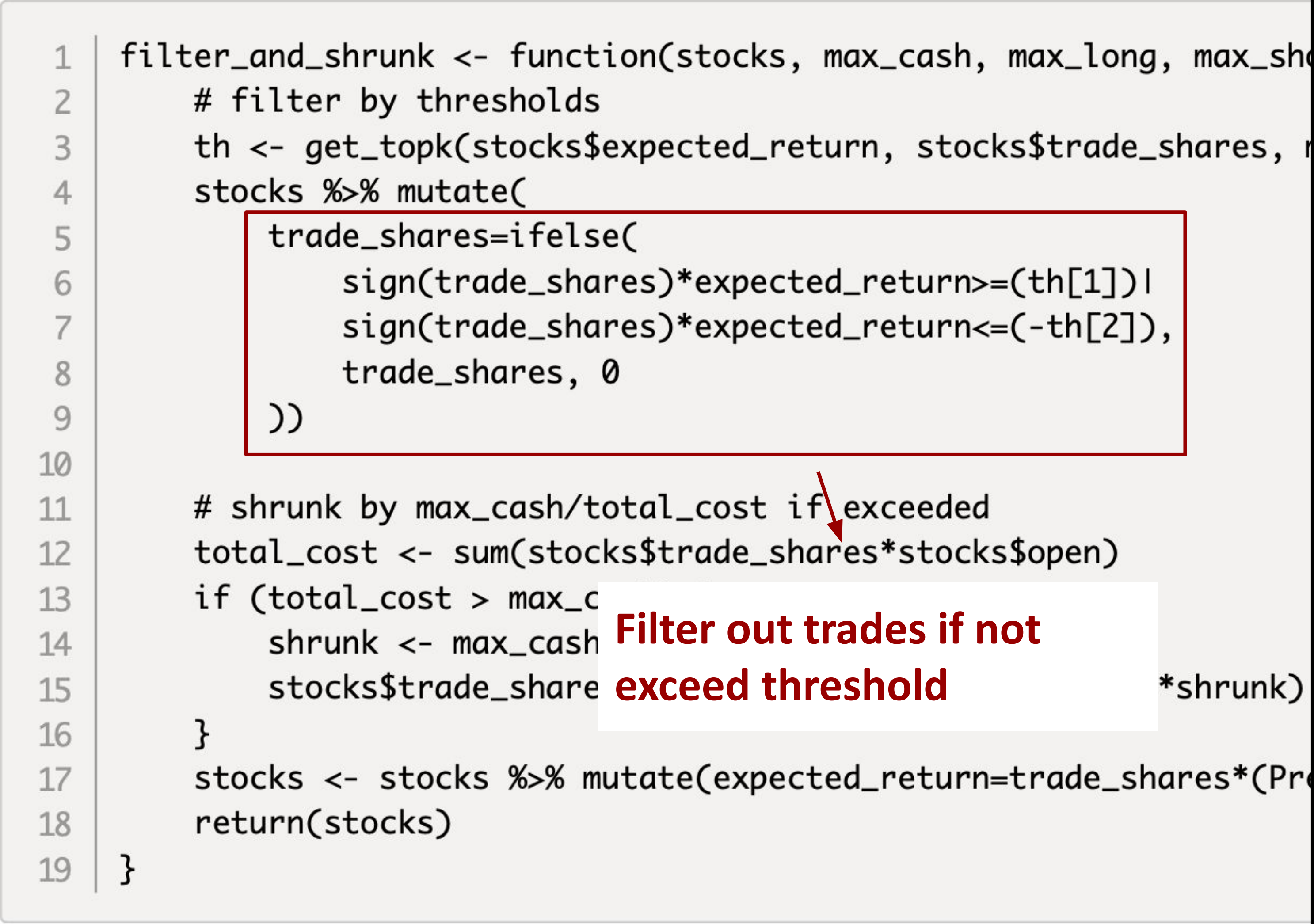


```
get_topk <- function(expected_return, trade_shares, k_long, k_short){
    values <- sign(trade_shares)*expected_return
    th_long <- max(sort(values)[length(values)-k_long+1], 0)
    th_short <- min(sort(values)[k_short], 0)
    return(c(th_long, -th_short))
}

# Test the function
test_return <- c(100, 200, 300, 150, 220, 120)
test_shares <- c(-10, 2, 8, -15, 20, 12)
th <- get_topk(test_return, test_shares, 2, 1)
th
```

```
220 · 150
```

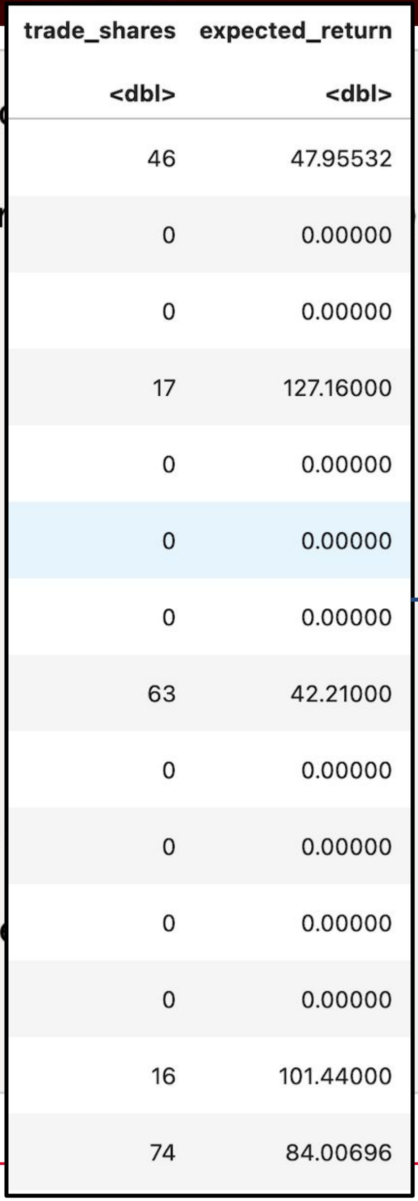
Pipeline implementation



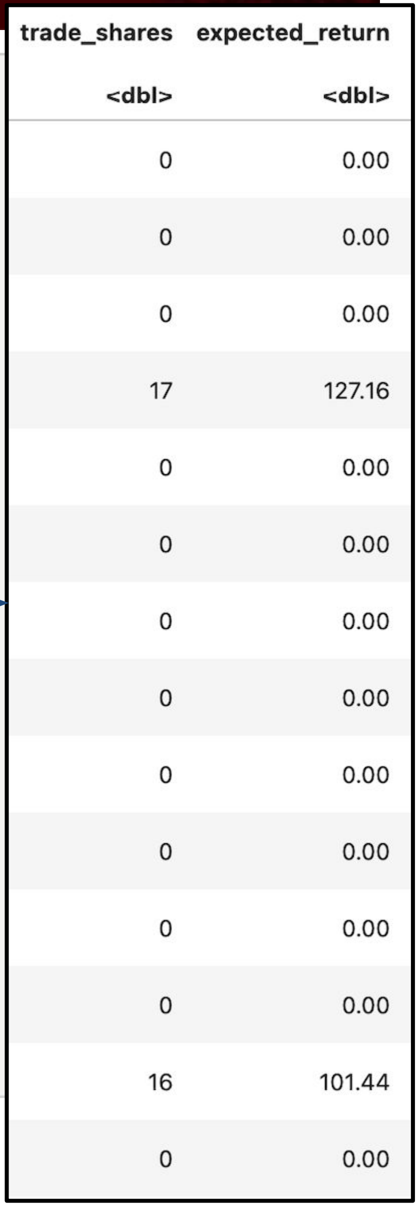
```
filter_and_shrunk <- function(stocks, max_cash, max_long, max_sh
    # filter by thresholds
    th <- get_topk(stocks$expected_return, stocks$trade_shares, r
    stocks %>% mutate(
        trade_shares=ifelse(
            sign(trade_shares)*expected_return>=(th[1])|
            sign(trade_shares)*expected_return<=(-th[2]),
            trade_shares, 0
        ))

    # shrunk by max_cash/total_cost if exceeded
    total_cost <- sum(stocks$trade_shares*stocks$open)
    if (total_cost > max_c
        shrunk <- max_cash
        stocks$trade_share                *shrunk)
    }
    stocks <- stocks %>% mutate(expected_return=trade_shares*(Pr
    return(stocks)
}
```

Filter out trades if not exceed threshold

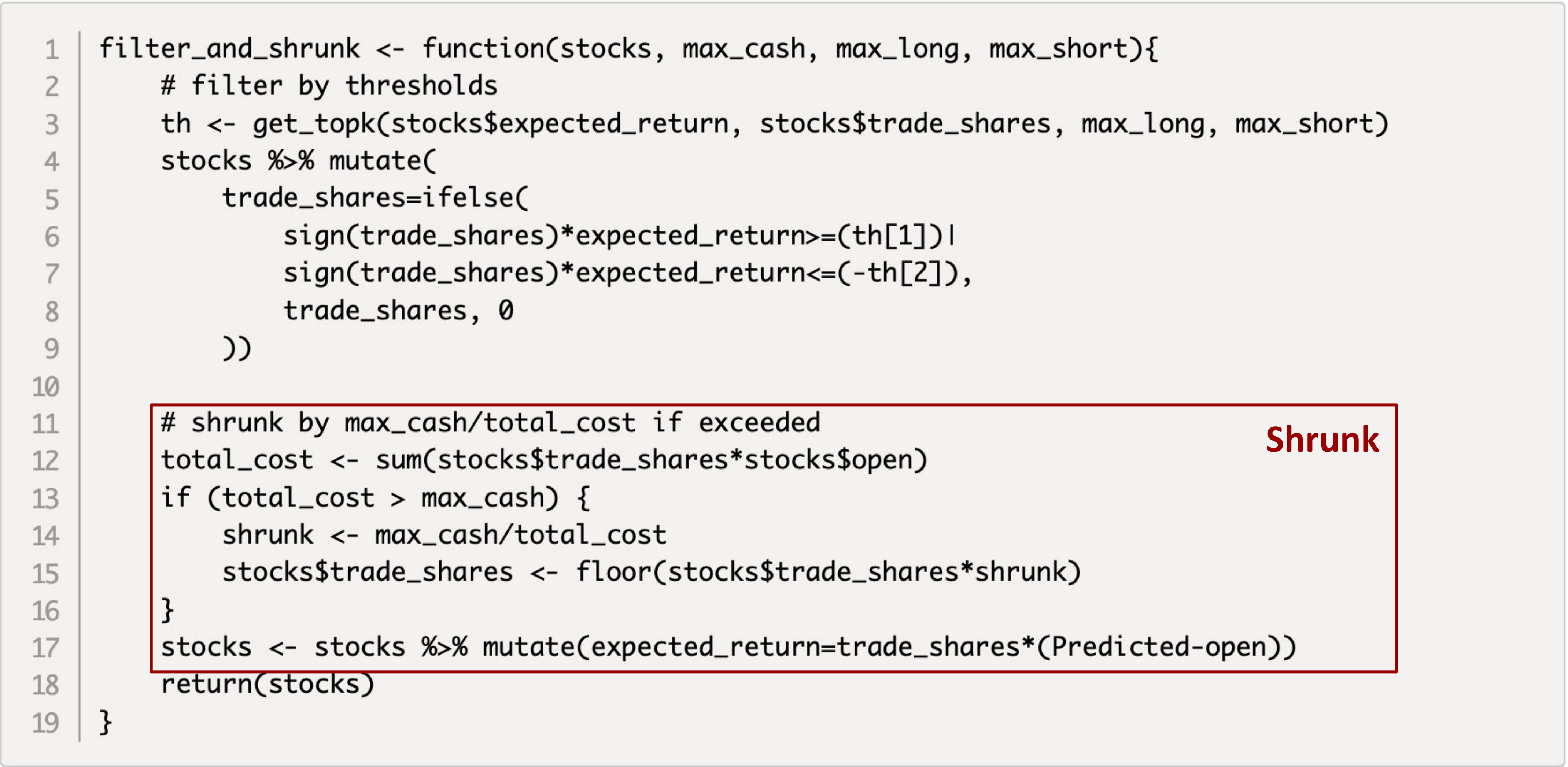


trade_shares	expected_return
<dbl>	<dbl>
46	47.95532
0	0.00000
0	0.00000
17	127.16000
0	0.00000
0	0.00000
0	0.00000
63	42.21000
0	0.00000
0	0.00000
0	0.00000
0	0.00000
16	101.44000
74	84.00696



trade_shares	expected_return
<dbl>	<dbl>
0	0.00
0	0.00
0	0.00
17	127.16
0	0.00
0	0.00
0	0.00
0	0.00
0	0.00
0	0.00
0	0.00
0	0.00
16	101.44
0	0.00

Pipeline implementation

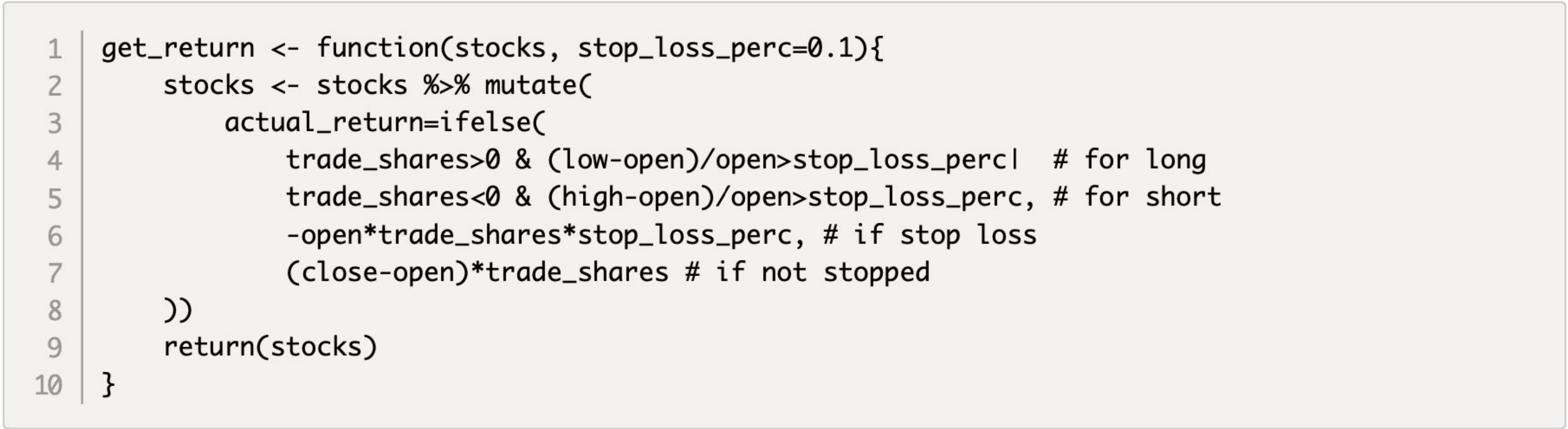


```
filter_and_shrunk <- function(stocks, max_cash, max_long, max_short){
    # filter by thresholds
    th <- get_topk(stocks$expected_return, stocks$trade_shares, max_long, max_short)
    stocks %>% mutate(
        trade_shares=ifelse(
            sign(trade_shares)*expected_return>=(th[1])|
            sign(trade_shares)*expected_return<=(-th[2]),
            trade_shares, 0
        ))

    # shrunk by max_cash/total_cost if exceeded
    total_cost <- sum(stocks$trade_shares*stocks$open)
    if (total_cost > max_cash) {
        shrunk <- max_cash/total_cost
        stocks$trade_shares <- floor(stocks$trade_shares*shrunk)
    }
    stocks <- stocks %>% mutate(expected_return=trade_shares*(Predicted-open))
    return(stocks)
}
```

Shrunk

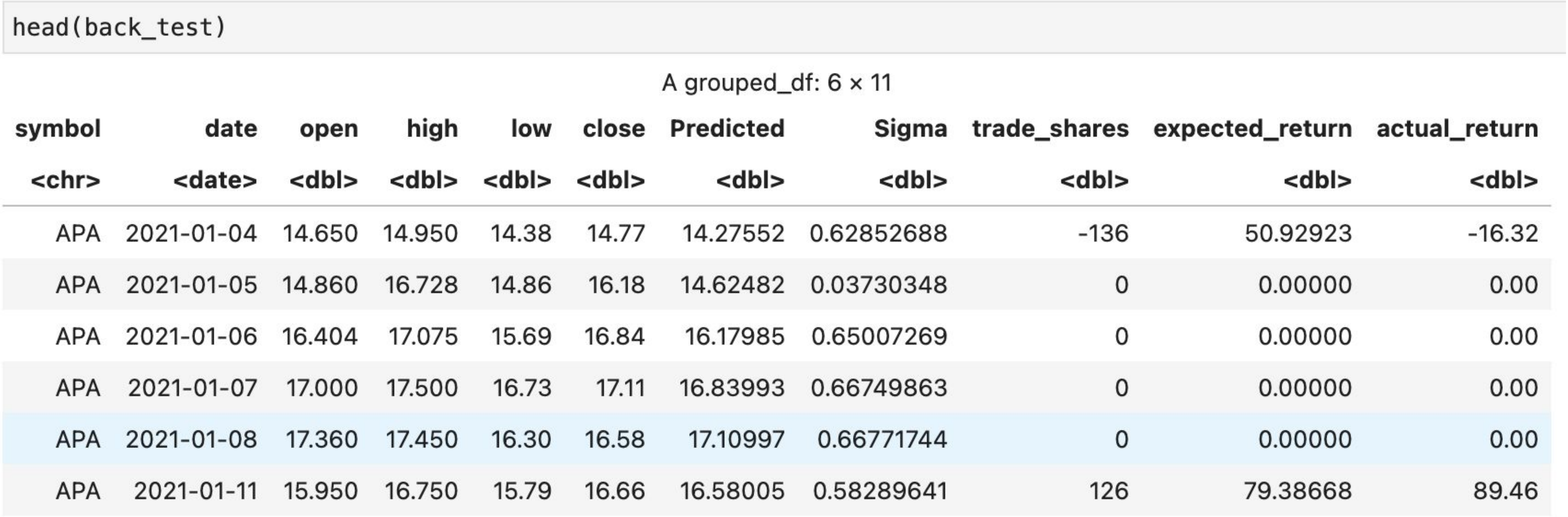
Pipeline implementation



```
get_return <- function(stocks, stop_loss_perc=0.1){
    stocks <- stocks %>% mutate(
        actual_return=ifelse(
            trade_shares>0 & (low-open)/open>stop_loss_perc|  # for long
            trade_shares<0 & (high-open)/open>stop_loss_perc, # for short
            -open*trade_shares*stop_loss_perc, # if stop loss
            (close-open)*trade_shares # if not stopped
    ))
    return(stocks)
}
```

We need to consider our stop-loss strategy when computing the return.

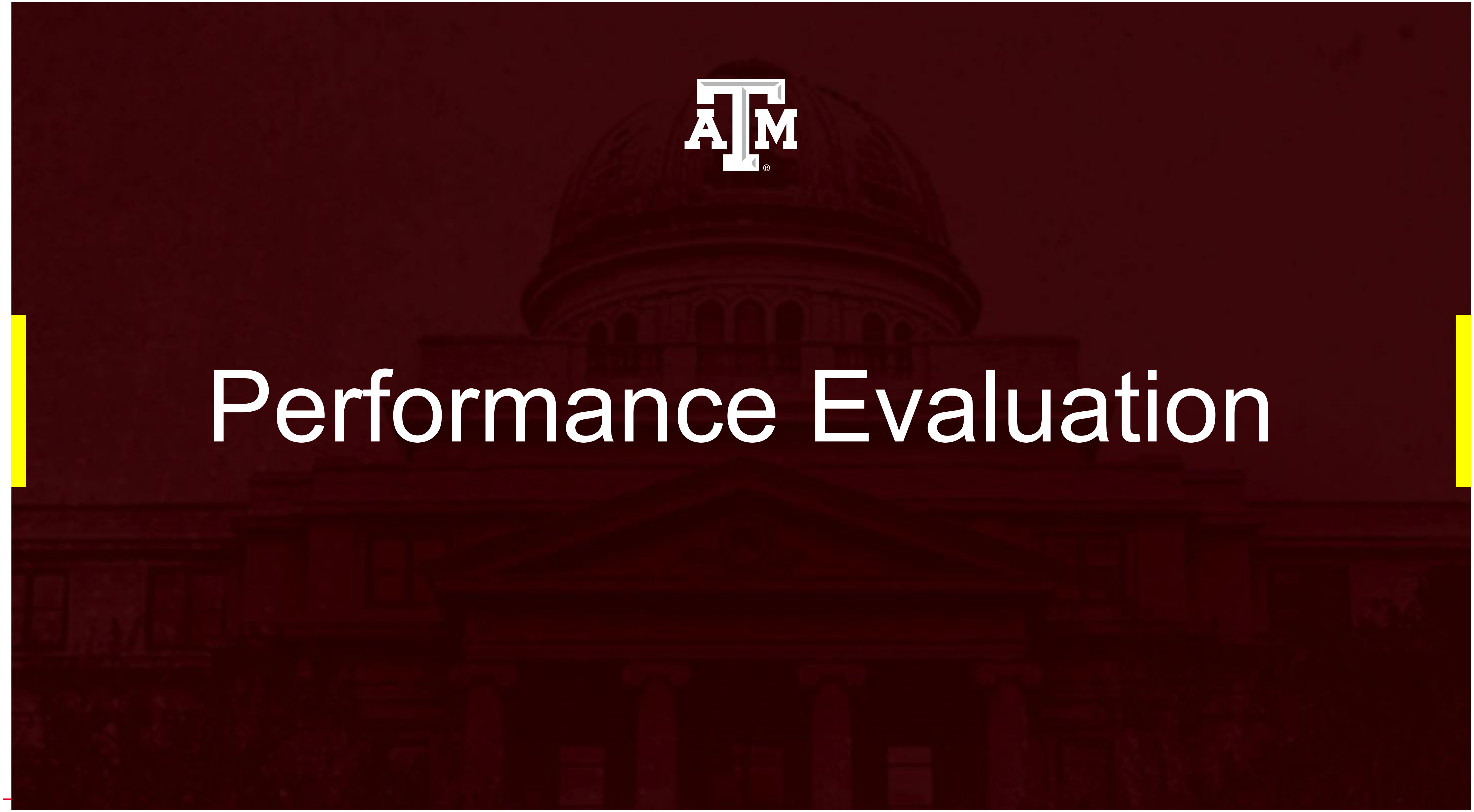
Results after this step



```
head(back_test)
```

A grouped_df: 6 × 11

symbol	date	open	high	low	close	Predicted	Sigma	trade_shares	expected_return	actual_return
<chr>	<date>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
APA	2021-01-04	14.650	14.950	14.38	14.77	14.27552	0.62852688	-136	50.92923	-16.32
APA	2021-01-05	14.860	16.728	14.86	16.18	14.62482	0.03730348	0	0.00000	0.00
APA	2021-01-06	16.404	17.075	15.69	16.84	16.17985	0.65007269	0	0.00000	0.00
APA	2021-01-07	17.000	17.500	16.73	17.11	16.83993	0.66749863	0	0.00000	0.00
APA	2021-01-08	17.360	17.450	16.30	16.58	17.10997	0.66771744	0	0.00000	0.00
APA	2021-01-11	15.950	16.750	15.79	16.66	16.58005	0.58289641	126	79.38668	89.46

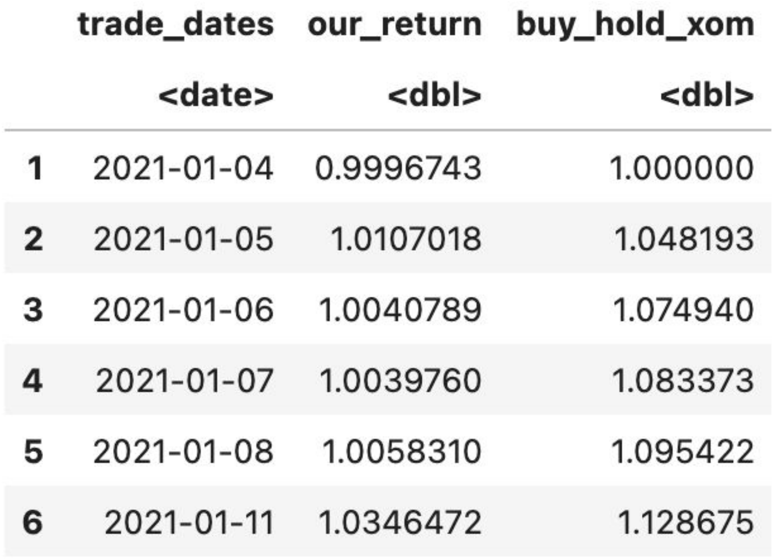


Performance Evaluation

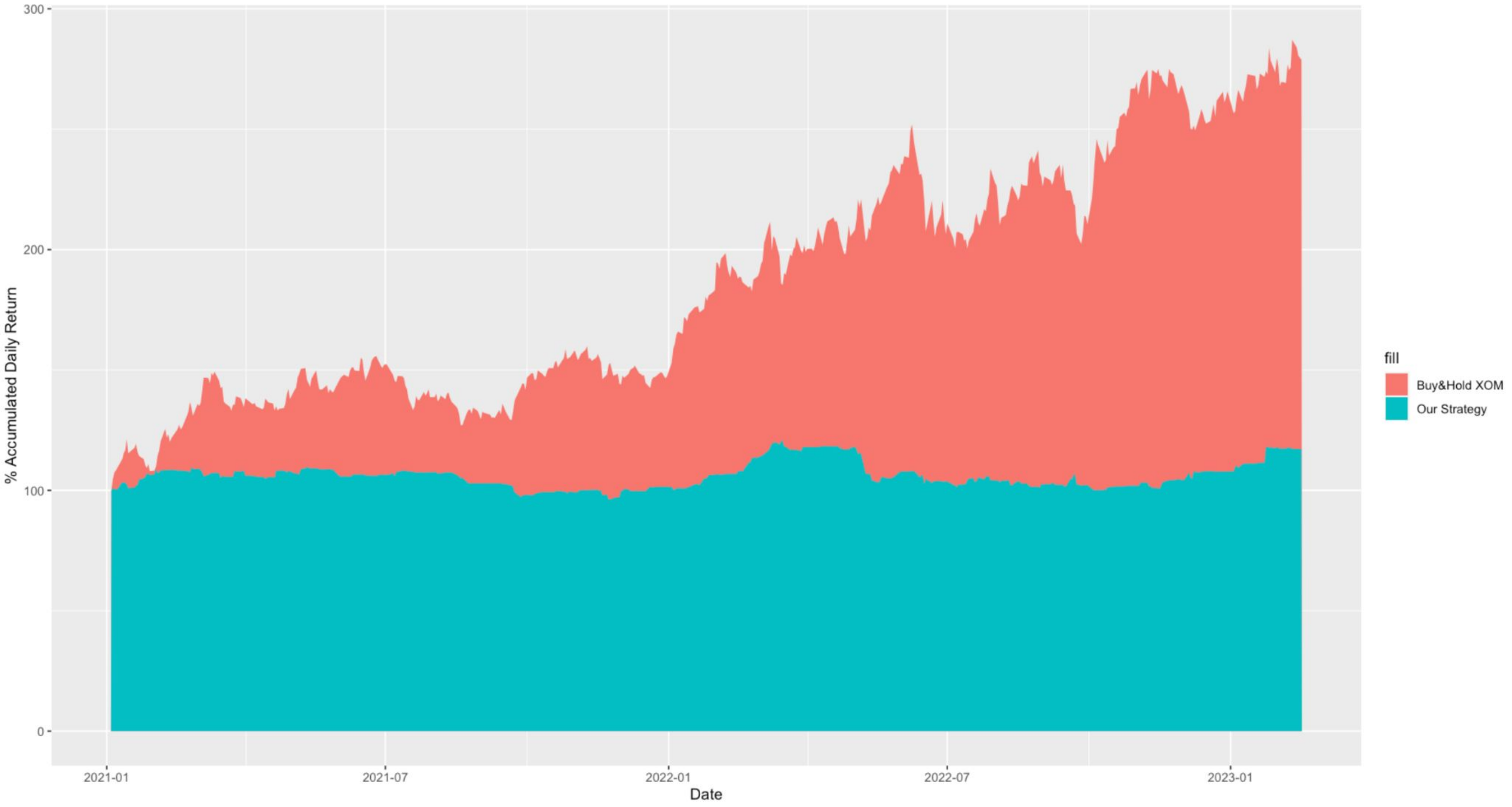
Performance Evaluation

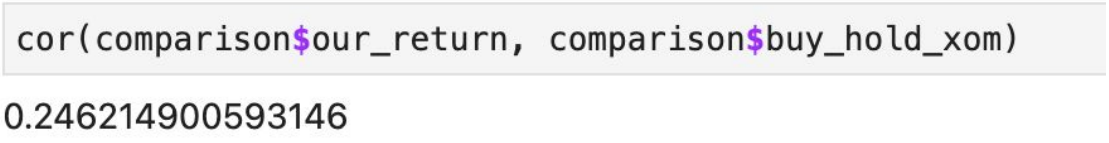
TEXAS A&M UNIVERSITY

Summation of the actual return of each trade by date, over initial fund:



	trade_dates <date>	our_return <dbl>	buy_hold_xom <dbl>
1	2021-01-04	0.9996743	1.000000
2	2021-01-05	1.0107018	1.048193
3	2021-01-06	1.0040789	1.074940
4	2021-01-07	1.0039760	1.083373
5	2021-01-08	1.0058310	1.095422
6	2021-01-11	1.0346472	1.128675



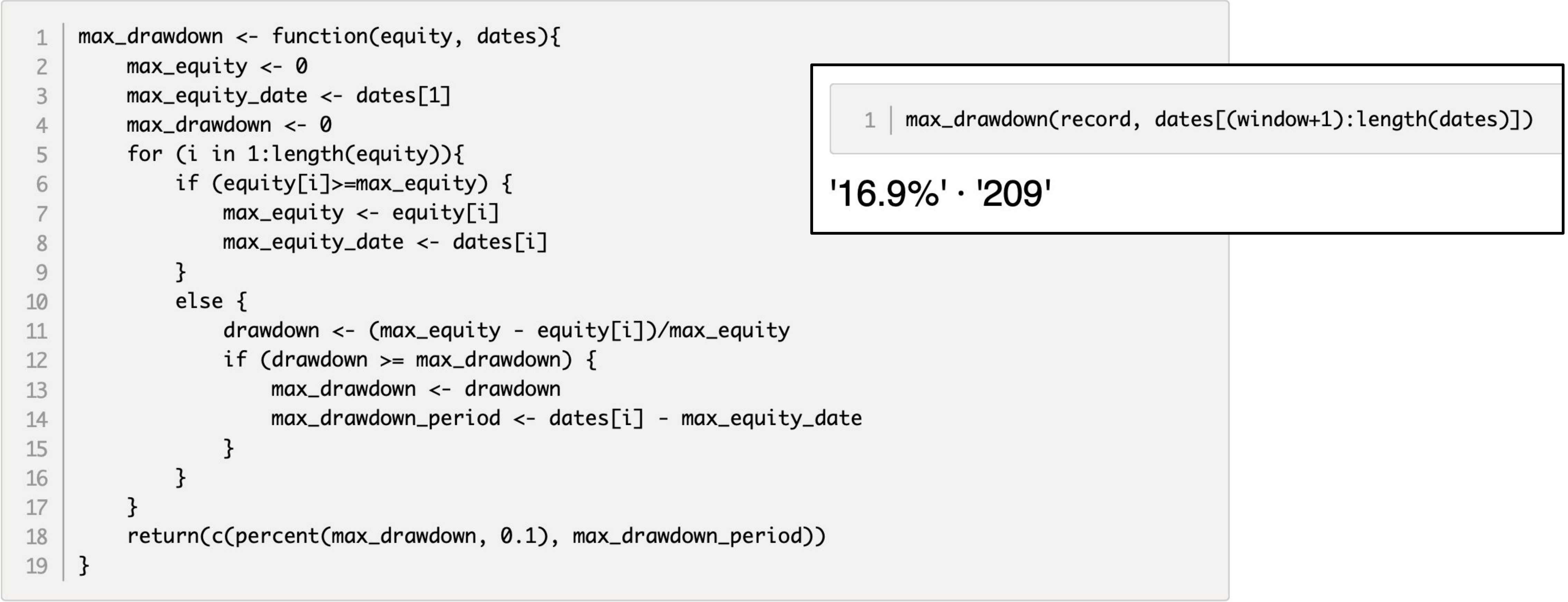


```
cor(comparison$our_return, comparison$buy_hold_xom)
```

```
0.246214900593146
```

Max Drawdown

- An O(N) algorithm using **two-pointers**.



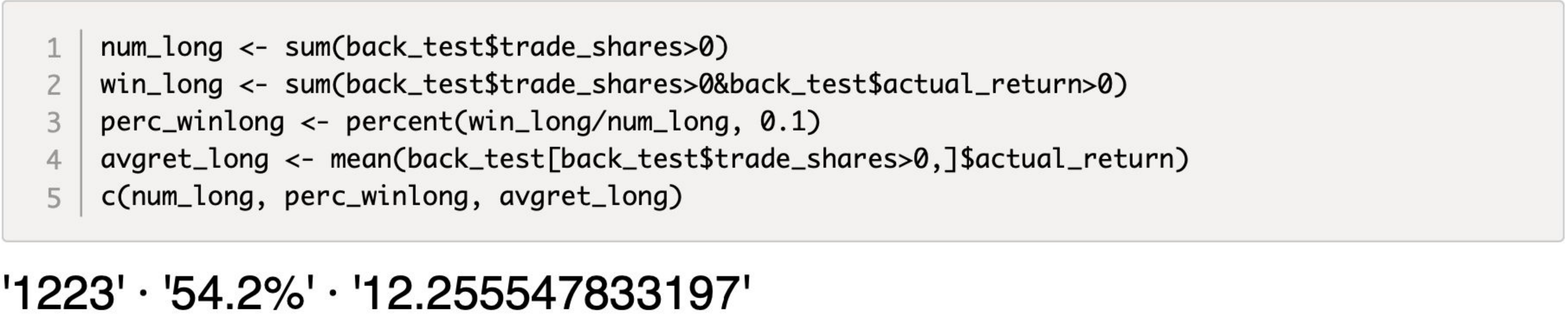
```
max_drawdown <- function(equity, dates){
    max_equity <- 0
    max_equity_date <- dates[1]
    max_drawdown <- 0
    for (i in 1:length(equity)){
        if (equity[i]>=max_equity) {
            max_equity <- equity[i]
            max_equity_date <- dates[i]
        }
        else {
            drawdown <- (max_equity - equity[i])/max_equity
            if (drawdown >= max_drawdown) {
                max_drawdown <- drawdown
                max_drawdown_period <- dates[i] - max_equity_date
            }
        }
    }
    return(c(percent(max_drawdown, 0.1), max_drawdown_period))
}
```

```
max_drawdown(record, dates[(window+1):length(dates)])
```

```
'16.9%' · '209'
```

Return statistics

The number of long trades, percentage of winning long trades, and the average return of long trades:

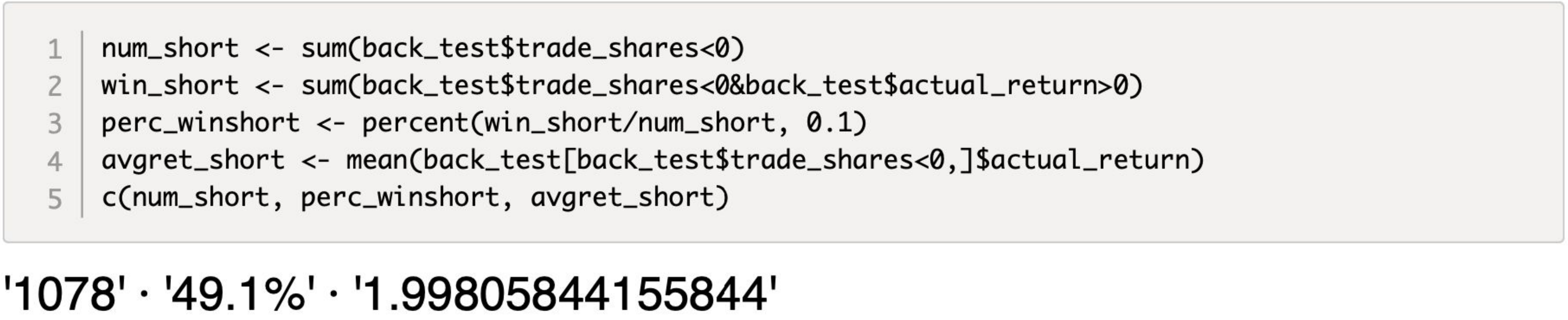


```
num_long <- sum(back_test$trade_shares>0)
win_long <- sum(back_test$trade_shares>0&back_test$actual_return>0)
perc_winlong <- percent(win_long/num_long, 0.1)
avgret_long <- mean(back_test[back_test$trade_shares>0,]$actual_return)
c(num_long, perc_winlong, avgret_long)
```

```
'1223' · '54.2%' · '12.255547833197'
```

Return statistics

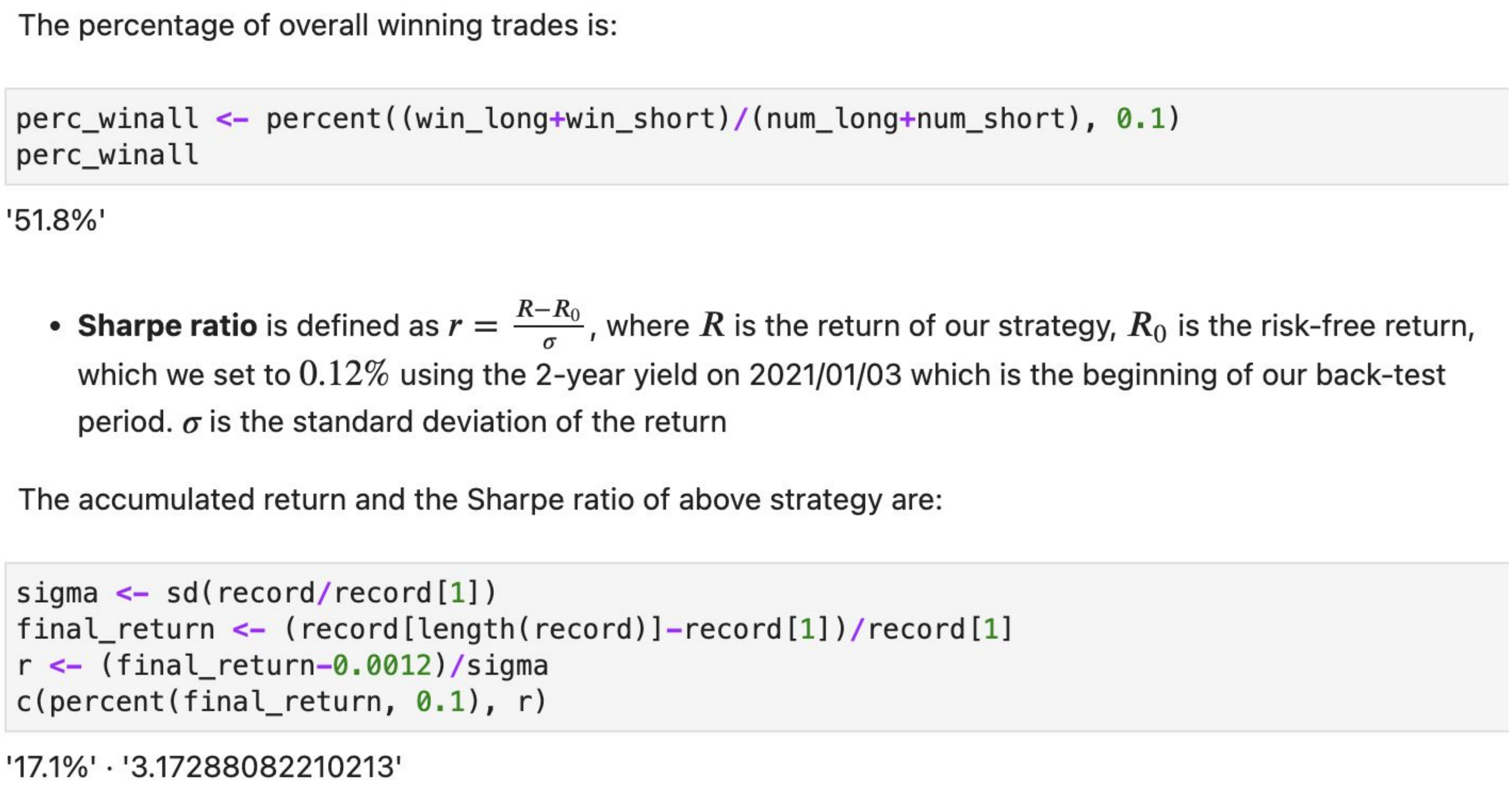
The number of short trades, percentage of winning short trades, and the average return of short trades:



```
num_short <- sum(back_test$trade_shares<0)
win_short <- sum(back_test$trade_shares<0&back_test$actual_return>0)
perc_winshort <- percent(win_short/num_short, 0.1)
avgret_short <- mean(back_test[back_test$trade_shares<0,]$actual_return)
c(num_short, perc_winshort, avgret_short)
```

```
'1078' · '49.1%' · '1.99805844155844'
```

Return statistics



The percentage of overall winning trades is:

```
perc_winall <- percent((win_long+win_short)/(num_long+num_short), 0.1)
perc_winall
```

'51.8%'

- **Sharpe ratio** is defined as $r = \frac{R-R_0}{\sigma}$, where R is the return of our strategy, R_0 is the risk-free return, which we set to 0.12% using the 2-year yield on 2021/01/03 which is the beginning of our back-test period. σ is the standard deviation of the return

The accumulated return and the Sharpe ratio of above strategy are:

```
sigma <- sd(record/record[1])
final_return <- (record[length(record)]-record[1])/record[1]
r <- (final_return-0.0012)/sigma
c(percent(final_return, 0.1), r)
```

'17.1%' · '3.17288082210213'