BACKTESTING

Joanna Grunwald,
Marta Kałużna,
Sebastian Jachimek,
Wojciech Wojnar

Motywacja

CPI (pot. inflacja)
zmiana (%) do analogicznego miesiąca poprzedniego roku

	2017	2018	2019	2020
Styczeń	1.7	1.9	0.7	4.3
Luty	2.2	1.4	1.2	4.7
Marzec	2.0	1.3	1.7	4.6
Kwiecień	2.0	1.6	2.2	3.4
Maj	1.9	1.7	2.4	2.9
Czerwiec	1.5	2.0	2.6	3.3
Lipiec	1.7	2.0	2.9	3.0
Sierpień	1.8	2.0	2.9	2.9
Wrzesień	2.2	1.9	2.6	3.2
Październik	2.1	1.8	2.5	3.1
Listopad	2.5	1.3	2.6	
Grudzień	2.1	1.1	3.4	

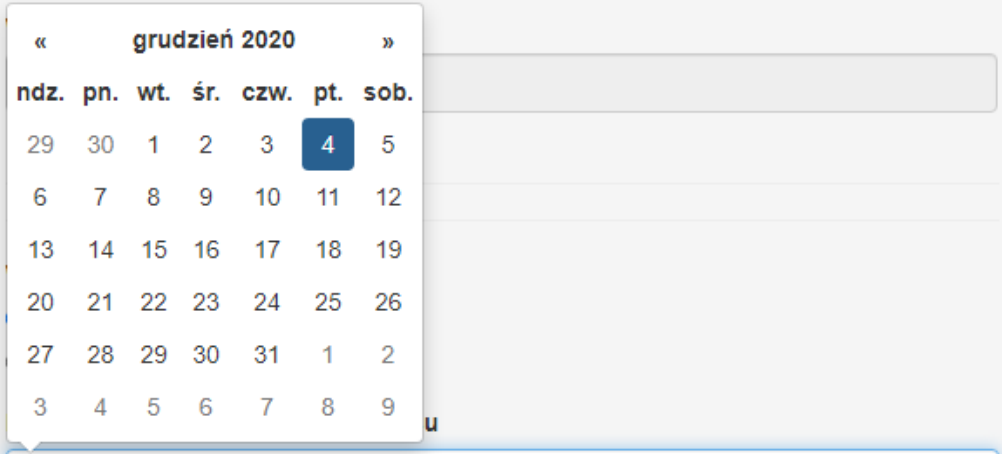
Podstawowe stopy procentowe NBP

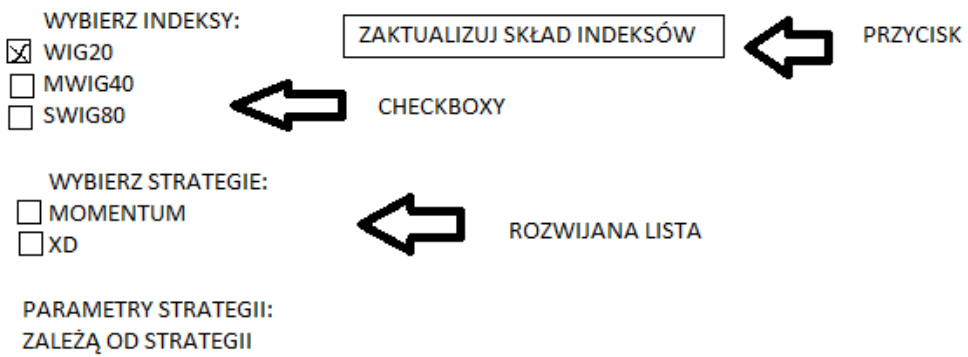
Stopa procentowa:	Oprocentowanie	Obowiązuje od dnia
Stopa referencyjna [1)]	0,10	2020-05-29
Stopa lombardowa	0,50	2020-05-29
Stopa depozytowa	0,00	2020-05-29
Stopa redyskontowa weksli	0,11	2020-05-29
Stopa dyskontowa weksli	0,12	2020-05-29

Dane

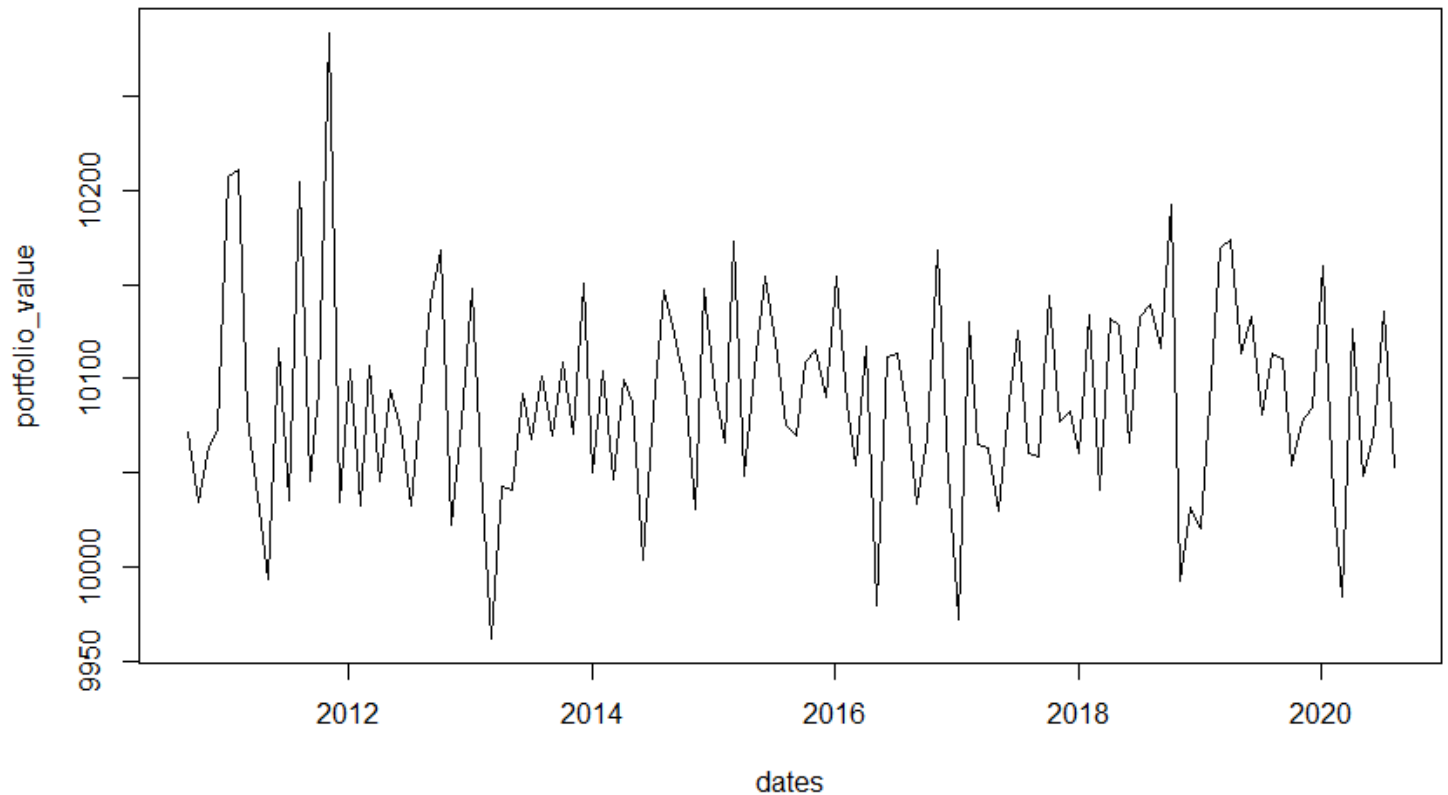
Źródło: stooq.pl
Format: .csv
Rodzaj: OHLCV
Przekształcenia: zależne od testowanej strategii (głównie różne statystyki dotyczące cen zamknięcia, zmienności w ciągu dnia itp.)

Ticker	Date	Open	High	Low	Close	Volume
CDR	1994-08-02	8.2789	8.2789	8.2789	8.2789	347109
CDR	1994-08-03	9.0931	9.0931	9.0931	9.0931	325196
CDR	1994-08-04	9.9950	9.9950	9.9950	9.9950	355499
CDR	1994-08-08	9.0931	9.0931	9.0931	9.0931	263619
CDR	1994-08-09	10.7320	10.7320	10.7320	10.7320	135441
CDR	1994-08-10	10.1230	10.1230	10.1230	10.1230	156263
CDR	1994-08-11	9.9170	9.9170	9.9170	9.9170	127846
CDR	1994-08-16	10.5360	10.5360	10.5360	10.5360	81439
CDR	1994-08-17	11.0650	11.0650	11.0650	11.0650	82459
CDR	1994-08-18	9.9560	9.9560	9.9560	9.9560	54547
CDR	1994-08-22	10.3300	10.3300	10.3300	10.3300	105020
CDR	1994-08-23	9.5050	9.5050	9.5050	9.5050	68436
CDR	1994-08-24	9.2990	9.2990	9.2990	9.2990	75990
CDR	1994-08-25	8.3575	8.3575	8.3575	8.3575	64062
CDR	1994-08-29	9.0538	9.0538	9.0538	9.0538	88751
CDR	1994-08-30	9.0146	9.0146	9.0146	9.0146	57584


« grudzień 2020 »
ndz. pn. wt. śr. czw. pt. sob.
29 30 1 2 3 4 5
6 7 8 9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31 1 2
3 4 5 6 7 8 9
2020-12-04
Proszę podać datę początku symulacji
2020-12-04
Proszę podać datę końca symulacji
2020-12-04


WYBIERZ INDEKSY:
WIG20
MWIG40
SWIG80
CHECKBOXY
WYBIERZ STRATEGIE:
MOMENTUM
XD
ROZWIJANA LISTA
PARAMETRY STRATEGII:
ZALEŻĄ OD STRATEGII


"ZAKŁADKI"
KRZYWA KAPITAŁU
DANE SZCZEGÓŁOWE
HISTORIA PORTFELA


ZAKTUALIZUJ SKŁAD INDEKSÓW
PRZYCISK
portfolio_value
9950
10000
10100
10200
2012
2014
2016
2018
2020
dates

Przykładowa funkcja: pobieranie aktualnych składów indeksów giełdowych (fragment)

Source on Save | Run | Sour

```r
library(rvest)
library(stringr)
library(stringi)

dir.create(paste0(c(getwd(), 'data'), collapse='/'))

pages = c('akcje', 'indeksy-gpw', 'new-connect', 'futures', 'opcje', 'obligacje')
url = 'https://www.bankier.pl/gielda/notowania/'

### POBRANIE INFORMACJI O AKCJACH-GPW, INDEKSACH GPW, AKCJACH-NEWCONNECT, FUTURESACH, OPCJACH I OBLIGACJACH

lista_tabel = list()

for (page in pages){
  print(page)
  tmp_url = paste0(url,page,collapse = '')

  tabela = read_html(tmp_url) %>%
    html_nodes(xpath = '//*[@id="boxQuotes"]/div[2]/table[1]') %>%
    html_table(fill=TRUE)
  tabela = tabela[[1]]
  tabela = rbind(tabela[1:9, 1:ncol(tabela)], tabela[11:nrow(tabela), 1:ncol(tabela)])
  lista_tabel[[page]] = tabela
}
```

Plan:

- Pobieranie danych z bazy,
- Implementacja strategii,
- Stworzenie 'środowiska' do testowania oraz:
  - Wykres krzywej kapitału,
  - Generowanie tabelki z podsumowaniem,
  - Historii transakcji

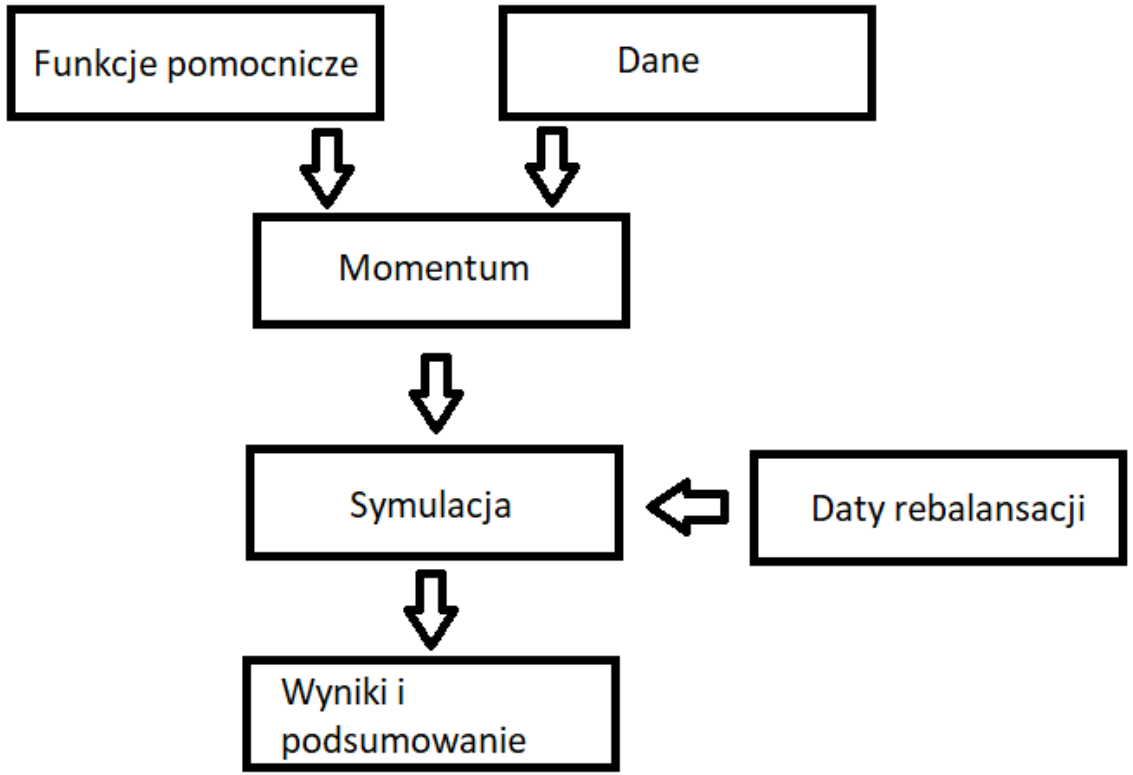
Źródła

- https://www.nbp.pl/home.aspx?f=/dzienne/stopy.htm
- https://stat.gov.pl/
- https://natallijen.shinyapps.io/projekt_1_02/

Schemat:

- 1. Pobranie danych z przedziału czasowego, który podał użytkownik.
- 2. Wybór parametrów do testowania przez użytkownika
- 3. Przeprowadzenie symulacji
- 4. Wyniki

Schemat cd.


Funkcje pomocnicze
Dane
Momentum
Symulacja
Daty rebalansacji
Wyniki i podsumowanie

Pobranie danych

```r
get_data_from_db <- function(ticker, start_date, end_date){
  url_code <- paste(c(Sys.getenv("DB_URL"),
                      ticker,"&startDate=", as.character(start_date - 365),
                      "&endDate=", as.character(end_date)), collapse = "")
  json <- httr::GET(url_code)
  json_string <- content(json, 'text', encoding = "UTF-8")
  frame <- fromJSON(json_string)
  if (class(frame) == 'character'){
    # print(frame)
    return(data.table())
  } else{
    frame <- fromJSON(json_string)
    result_dt = data.table(frame)
    setnames(result_dt, 'hight', 'high')
    result_dt[, `:=` (date = ymd(date),
                      open = as.numeric(open),
                      high = as.numeric(high),
                      low = as.numeric(low),
                      close = as.numeric(close),
                      volume = as.numeric(volume)
                      )]
    return(result_dt)
  }
}

get_all_data <- function(tickers, start_date, end_date){
  prices <- lapply(tickers, get_data_from_db, start_date, end_date)
  return(data.table::rbindlist(prices))
}
```

Funkcje pomocnicze do wyboru spółek

```r
is_weekend <- function(date){
  ifelse(weekdays(date) %in% c("sobota", "niedziela"), TRUE, FALSE)
}

numdays <- function(num, date){
  vec <- vector()
  while (length(vec) < num){
    if (is_weekend(as.Date(date)) == FALSE){
      vec <- c(as.Date(vec), as.Date(date))
    }
    else{
      vec <- vec
    }
    date = date - 1
  }
  rev(vec)
}
```

```r
slope_of_regression <- function(data, name){
  model <- lm(log(close)~date, data = data[ticker == name, ])
  return(c(model$coefficient[2], summary(model)$r.squared))
}

score_function <- function(data, lst){
  slopevec <- c()
  rsqvec <- c()
  for (name in lst){
    slopevec <- c(slopevec,slope_of_regression(data, name)[1])
    rsqvec <- c(rsqvec, slope_of_regression(data, name)[2])
  }
  score <- ((exp(slopevec))^252 - 1) * 100 * rsqvec
  return (score)
}
```

Wybór spółek:

```
momentum_function <- function(data, date_of_analysis, number_of_days,
                              cash, min_momentum, max_stocks, min_inv_vola, ...){
  if (weekdays(date_of_analysis) == "sobota"){}
  if (weekdays(date_of_analysis) == "niedziela"){}
  else{}
  vec_of_days <- numdays(number_of_days, date_of_analysis)
  window <- data[date %in% vec_of_days, ]
  ticker_vec <- unique(window$ticker)
  score <- score_function(window, ticker_vec)
  momentum_tab <- data.table(ticker_vec, score)
  setnames(momentum_tab, c("ticker", "momentum"))
  momentum_tab <- momentum_tab[order(-momentum), ]
  momentum_tab <- head(momentum_tab, max_stocks)
  price <- window[ticker %in% momentum_tab$ticker & date == date_of_analysis,
                  mean(c(open, high, low, close), na.rm = TRUE), by = ticker]
  volatility <- window[ticker %in% momentum_tab$ticker,
                       sd((high - low)/close), by = ticker]
  tmp_table <- data.table(inner_join(price, volatility, by = "ticker"))
  setnames(tmp_table, c("ticker", "price", "volatility"))
  tmp_table[, inv_volatility := 1/volatility]
  momentum_tab <- data.table(inner_join(momentum_tab, tmp_table, by = "ticker"))
  momentum_tab[, weight := inv_volatility/sum(inv_volatility)]
  momentum_tab <- momentum_tab[momentum > min_momentum
                               & inv_volatility > min_inv_vola, ]
  momentum_tab[, quantity := floor((cash * weight) / price)]
  momentum_tab[, value := quantity * price]
  momentum_tab[, date := date_of_analysis]
  setcolorder(momentum_tab, c("ticker", "date", "momentum", "volatility",
                              "inv_volatility", "weight", "price",
                              "quantity", "value"))
  return (momentum_tab)
}
```

Daty rebalansacji i uzupełnienie braków danych

```r
get_rebalance_dates = function(start_date, end_date, rebalance_frequency, period){
  rebalance_dates = seq(ymd(start_date), ymd(end_date),
                        by = paste(rebalance_frequency, period))
  rebalance_dates = unique(adjust('Poland', rebalance_dates, OL))
  return(rebalance_dates)
}
```

```r
fill_missing_prices_for_ticker = function(dates_dt, single_stock_data){
  tkr = unique(single_stock_data[, ticker])
  merged_single_stock = merge(dates_dt, single_stock_data, all.x = TRUE)
  first_non_na = which.min(is.na(merged_single_stock[, ticker]))
  result = merged_single_stock[first_non_na:nrow(merged_single_stock), ]
  setnafill(result, 'nocb', cols = c('open', 'high', 'low', 'close', 'volume'))
  result[, ticker := tkr]
  return(result)
}

fill_missing_prices = function(dates_dt, data){
  tickers = unique(data[, ticker])
  filled_data_list = vector("list", length(tickers))
  for (i in 1:length(tickers)){
    filled_data_list[[i]] = fill_missing_prices_for_ticker(dates_dt, data[ticker == tickers[i]])
  }
  return(rbindlist(filled_data_list))
}
```

Symulacja:

```
backtest_simulation = function(dt_from_db, commission_rate, rebalance_dates,
                               number_of_days, min_momentum,
                               max_stocks, min_inv_vola, cash){
  #' dt_from_db - result from get_all_data()
  #' rest - strategy params
  date_tmp = seq(min(rebalance_dates) - 365, max(rebalance_dates), 1)
  dates_dt = data.table(date = date_tmp[isBusinessDay("Poland", date_tmp)])
  dt_with_filled_missing_prices = fill_missing_prices(dates_dt, dt_from_db)
  momentum_tables_history = list()
  cash_history = c()
  stocks_value_history = c()
  total_value_history = c()
  commissions = c()
  current_momentum_table = data.table(matrix(nrow=0, ncol=9))
  setnames(current_momentum_table, colnames(current_momentum_table),
           c("ticker", "date", "momentum", "volatility", "inv_volatility", "weight",
             "price", "quantity", "value"))
  old_momentum_table = data.table(matrix(nrow = 0, ncol = 9))
  setnames(old_momentum_table, colnames(old_momentum_table),
           c("ticker", "date", "momentum", "volatility", "inv_volatility", "weight",
             "price", "quantity", "value"))
  for (date_number in 1:length(rebalance_dates)){
    analysis_date = rebalance_dates[date_number]
    current_prices = dt_with_filled_missing_prices[ticker %in% old_momentum_table[, ticker] &
                                 date == analysis_date][order(ticker), close]
    current_stocks_value = sum(current_prices * old_momentum_table[order(ticker),
                                                                   quantity])
    commission_sell = current_stocks_value * commission_rate
    cash = cash + current_stocks_value - commission_sell
    current_momentum_table = momentum_function(data = dt_with_filled_missing_prices,
                                               date_of_analysis = analysis_date,
                                               number_of_days = number_of_days,
                                               min_momentum = min_momentum,
                                               max_stocks = max_stocks,
                                               min_inv_vola = min_inv_vola,
                                               cash = cash)
    stocks_value = sum(current_momentum_table[, value])
    momentum_tables_history[[as.character(analysis_date)]] = current_momentum_table
    commission_buy = commission_rate * stocks_value
    cash = cash - stocks_value - commission_buy
    old_momentum_table = current_momentum_table
    cash_history = c(cash_history, cash)
    stocks_value_history = c(stocks_value_history, stocks_value)
    total_value_history = c(total_value_history, cash + stocks_value)
    commissions = c(commissions, commission_buy + commission_sell)
  }
  history_dt = data.table(date = rebalance_dates,
                          cash = cash_history,
                          stocks_value = stocks_value_history,
                          total_value = total_value_history,
                          commissions = commissions)
  return(list(equity_history = history_dt, portfolio_history = momentum_tables_history))
}
```

Podsumowanie:

```r
equity_plot = function(history_dt, plot_type){
  history_dt_long = pivot_longer(history_dt, cols = c('cash', 'stocks_value', 'total_value'))
  if (plot_type == 'point'){
    return(ggplot(history_dt_long) +
             geom_point(aes(x = date, y = value, col = name)))
  }
  if (plot_type == 'col'){
    return(ggplot(history_dt_long) +
             geom_col(aes(x = date, y = value, fill = name), position = 'dodge'))
  }
}

strat_summary = function(history_dt){
  returns = CalculateReturns(history_dt[, c("date", "total_value")])
  start = as.Date(history_dt[, date][1])
  end = as.Date(history_dt[, date][nrow(history_dt)])
  start_value = round(history_dt[, total_value][1], 2)
  end_value = round(history_dt[, total_value][nrow(history_dt)], 2)
  commissions_value = round(sum(history_dt[, commissions]))
  cagr = round(Return.annualized(returns)[1], 3)
  sortino = round(SortinoRatio(returns)[1], 3)
  std_dev = round(StdDev(returns)[1], 3)
  max_dd = round(maxDrawdown(returns), 3)
  profitable_rebablances = sum(returns$total_value > 0, na.rm = TRUE)
  loss_rebalances = sum(returns$total_value < 0, na.rm = TRUE)
  summary_dt = data.table(list("Początek symulacji", "Koniec symulacji", "Początkowa wartość porfela",
                               "Końcowa wartość portfela", "Suma prowizji", "CAGR",
                               "Wskaźnik Sortino", "Odchylenie standardowe portfela",
                               "Maksymalne obsunięcie kapitału", "Zyskowne rebalansacje",
                               "Stratne rebalansacje"),
                          list(start, end, start_value, end_value, commissions_value,
                               cagr, sortino, std_dev, max_dd,
                               profitable_rebablances, loss_rebalances)
  )
  setnames(summary_dt, colnames(summary_dt), c("Nazwa", "Wartość"))
  return(summary_dt)
}
```