Wrangling data for input to simulation engine

Introduction

`cash_backtest` is a reasonably well-optimised event-driven backtesting engine. In testing, it processed an 80-asset, 2300-day backtest in less than 150 milliseconds.

It was built with speed in mind, which required trading off certain conveniences such as holding weights and prices in long-format dataframes indexed by a human-readable timestamp. Instead, it requires the user to ensure their input data meets some fairly strict requirements.

The purpose of this vignette is to provide an example of how to prepare input data for `cash_backtest`.

Input Data Requirements

Price and weight matrixes

`cash_backtest` requires two matrixes of identical dimensions. Both matrix's first column needs to be a timestamp or date in Unix format.

The timestamp should be aligned with the weights and prices such that on a single row, the price is the price you assume you can trade into the weight at. This may require lagging of signals or weights upstream of the simulation and is up to the user. In another vignette, we provide an example of a simple workflow for lagging weights with respect to prices.

The first input matrix contains prices, one column for each asset or product in the strategy's universe.

The second matrix contains theoretical or ideal weights, again, one column for each asset in the strategy's universe.

Columns must map between the two matrixes: * Column 1 is always the date or timestamp column * Column 2 contains the prices and weights for the first asset * Column 3 contains the prices and weights for the second asset * *etc*

Let's run through an example of how you might wrangle such input data using tools from the tidyverse.

```
library(rsims)
library(tidyverse)
#> Warning: package 'tidyverse' was built under R version 4.0.5
#> -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
#> v ggplot2 3.3.3     v purrr   0.3.4
#> v tibble  3.1.2     v dplyr   1.0.6
#> v tidyr   1.1.3     v stringr 1.4.0
#> v readr   1.4.0     v forcats 0.5.1
#> Warning: package 'ggplot2' was built under R version 4.0.3
#> Warning: package 'tibble' was built under R version 4.0.5
#> Warning: package 'tidyr' was built under R version 4.0.5
#> Warning: package 'readr' was built under R version 4.0.3
#> Warning: package 'dplyr' was built under R version 4.0.5
#> Warning: package 'forcats' was built under R version 4.0.5
```

```
#> -- Conflicts ---------------------------------------- tidyverse_conflicts() --
#> x dplyr::filter() masks stats::filter()
#> x dplyr::lag()    masks stats::lag()
library(glue)
#>
#> Attaching package: 'glue'
#> The following object is masked from 'package:dplyr':
#>
#>     collapse
```

First, let's assume you have a long dataframe consisting of columns for ticker, date, price, and weight:

```
# load("~/rsims/data/backtest_df_long.RData")
head(backtest_df_long)
#> # A tibble: 6 x 4
#> # Groups:   date [1]
#>   ticker date        price_usd theo_weight
#>   <chr>  <date>          <dbl>       <dbl>
#> 1 BTC    2015-04-22 234.             0.1
#> 2 DASH   2015-04-22   3.24          -0.1
#> 3 DGB    2015-04-22   0.000110      -0.06
#> 4 DOGE   2015-04-22   0.000109      -0.02
#> 5 LTC    2015-04-22   1.44           0.02
#> 6 MAID   2015-04-22   0.0233         0.18
```

How you arrived at the weights for each product for each day is up to you. `backtest_df_long` contains weights for a simple cross-sectional momentum strategy.

Recall that we need to end up with two wide matrixes of date and prices and date and weights, and that the columns of each matrix must map column-wise.

One easy way to do that is use `tidyr::pivot_wider`, which will guarantee that prices and weights will be mapped correctly:

```
backtest_df <- backtest_df_long %>%
  pivot_wider(names_from = ticker, values_from = c(price_usd, theo_weight))

head(backtest_df)
#> # A tibble: 6 x 37
#> # Groups:   date [6]
#>   date       price_usd_BTC price_usd_DASH price_usd_DGB price_usd_DOGE
#>   <date>             <dbl>          <dbl>         <dbl>          <dbl>
#> 1 2015-04-22          234.           3.24      0.000110      0.000109
#> 2 2015-04-23          236.           3.67      0.000119      0.000111
#> 3 2015-04-24          231.           3.20      0.000133      0.000105
#> 4 2015-04-25          226.           3.09      0.000122      0.0000997
#> 5 2015-04-26          221.           3.05      0.000123      0.0000976
#> 6 2015-04-27          227.           2.98      0.000120      0.000105
#> # ... with 32 more variables: price_usd_LTC <dbl>, price_usd_MAID <dbl>,
#> #   price_usd_VTC <dbl>, price_usd_XEM <dbl>, price_usd_XMR <dbl>,
#> #   price_usd_XRP <dbl>, price_usd_ETH <dbl>, price_usd_XLM <dbl>,
#> #   price_usd_DCR <dbl>, price_usd_LSK <dbl>, price_usd_ETC <dbl>,
#> #   price_usd_REP <dbl>, price_usd_ZEC <dbl>, price_usd_WAVES <dbl>,
```

```
#> #   theo_weight_BTC <dbl>, theo_weight_DASH <dbl>, theo_weight_DGB <dbl>,
#> #   theo_weight_DOGE <dbl>, theo_weight_LTC <dbl>, theo_weight_MAID <dbl>, ...
```

From this point, we can split our single wide matrix into two matrixes. Note that since matrixes must hold a common data type, our date column will be converted to a Unix-style timestamp.

```
# get weights as a wide matrix
# note that date column will get converted to unix timestamp
backtest_theo_weights <- backtest_df %>%
  select(date, starts_with("theo_weight_")) %>%
  data.matrix()
```

We have some `NA` weights where we didn't have a weight for an asset on a particular day in our long dataframe. It makes sense to replace these with zero.

```
# NA weights should be zero
backtest_theo_weights[is.na(backtest_theo_weights)] <- 0

head(backtest_theo_weights, c(5, 5))
#>       date theo_weight_BTC theo_weight_DASH theo_weight_DGB theo_weight_DOGE
#> [1,] 16547           0.10           -0.10          -0.06           -0.02
#> [2,] 16548           0.14           -0.06          -0.10            0.06
#> [3,] 16549           0.14           -0.10           0.10           -0.06
#> [4,] 16550           0.10           -0.10           0.06           -0.06
#> [5,] 16551           0.06           -0.06           0.14           -0.02
```

We do the same thing for our prices, but this time where an asset didn't have a price (for example because it wasn't in existence on particular day), we leave the existing `NA`:

```
# get prices as a wide matrix
# note that date column will get converted to unix timestamp
backtest_prices <- backtest_df %>%
  select(date, starts_with("price_")) %>%
  data.matrix()

head(backtest_prices, c(5, 5))
#>       date price_usd_BTC price_usd_DASH price_usd_DGB price_usd_DOGE
#> [1,] 16547      233.8224       3.241223  0.0001098965   1.091501e-04
#> [2,] 16548      235.9333       3.667605  0.0001194309   1.113668e-04
#> [3,] 16549      231.4586       3.203421  0.0001334753   1.046427e-04
#> [4,] 16550      226.4460       3.093542  0.0001222808   9.972296e-05
#> [5,] 16551      220.5034       3.054431  0.0001227349   9.762035e-05
```

At this point, we are ready to simulate trading according to our weights:

```
# simulation parameters
initial_cash <- 10000
capitalise_profits <- FALSE  # remain fully invested?
trade_buffer <- 0.
commission_pct <- 0.

# simulation
```

```
results_df <- cash_backtest(
  backtest_prices,
  backtest_theo_weights,
  trade_buffer,
  initial_cash,
  commission_pct,
  capitalise_profits
)

head(results_df)
#> # A tibble: 6 x 8
#>   ticker         Date            Close Position Value  Trades TradeValue Commission
#>   <chr>          <date>          <dbl>    <dbl> <dbl>   <dbl>      <dbl>      <dbl>
#> 1 Cash           2015-04-22   1           1    e4 10000 NA              NA          0
#> 2 price_usd_BTC  2015-04-22 234.          4.28e0  1000  4.28e0       1000          0
#> 3 price_usd_DASH 2015-04-22   3.24       -3.09e2 -1000 -3.09e2      -1000          0
#> 4 price_usd_DGB  2015-04-22   0.000110   -5.46e6  -600 -5.46e6       -600          0
#> 5 price_usd_DOGE 2015-04-22   0.000109   -1.83e6  -200 -1.83e6       -200          0
#> 6 price_usd_LTC  2015-04-22   1.44        1.39e2   200  1.39e2        200          0
```

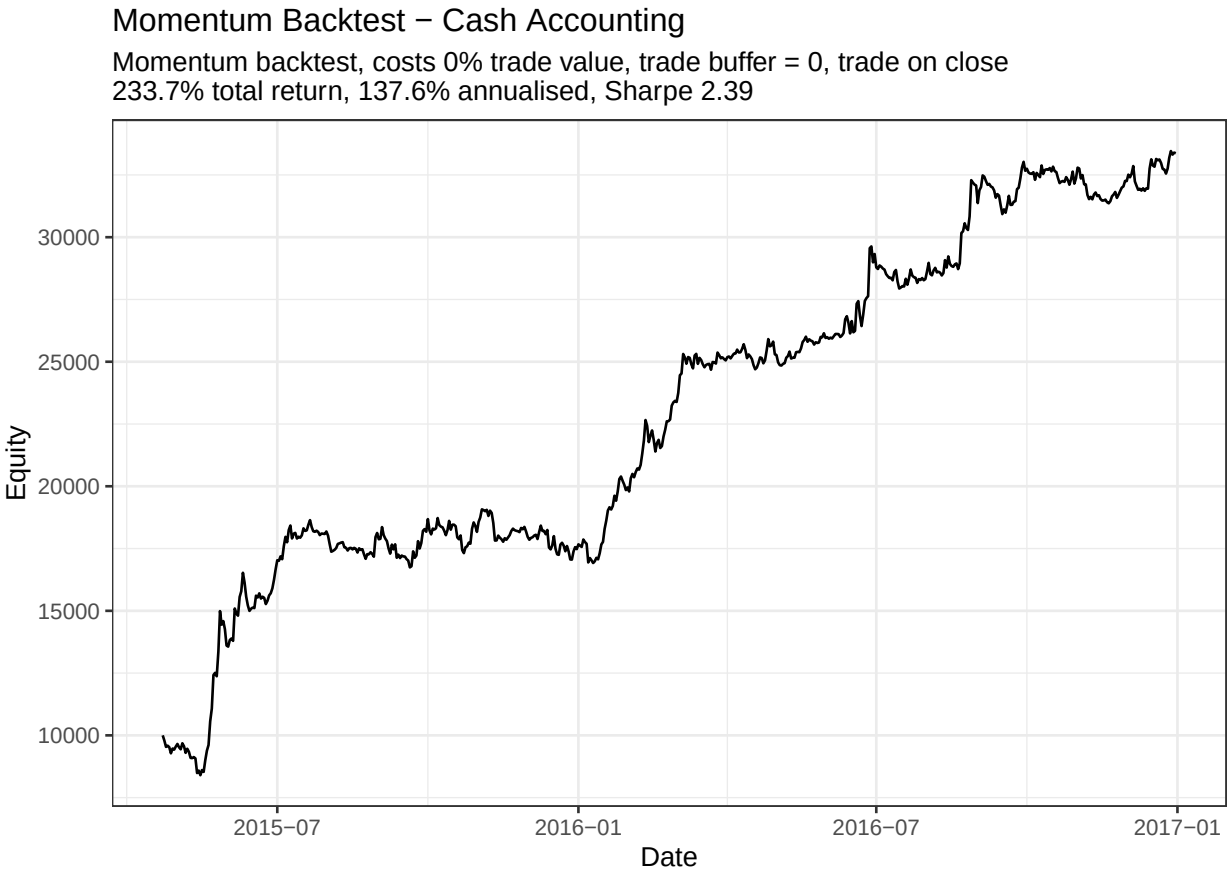
```
equity_curve <- results_df %>%
  group_by(Date) %>%
  summarise(Equity = sum(Value, na.rm = TRUE))

fin_eq <- equity_curve %>%
  tail(1) %>%
  pull(Equity)

init_eq <- equity_curve %>%
  head(1) %>%
  pull(Equity)

total_return <- (fin_eq/init_eq - 1) * 100
days <- nrow(equity_curve)
ann_return <- total_return * 365/days
sharpe <- equity_curve %>%
  mutate(returns = Equity/lag(Equity)- 1) %>%
  na.omit() %>%
  summarise(sharpe = sqrt(355)*mean(returns)/sd(returns)) %>%
  pull()

equity_curve %>%
  ggplot(aes(x = Date, y = Equity)) +
    geom_line() +
    labs(
      title = "Momentum Backtest - Cash Accounting",
      subtitle = glue(
        "Momentum backtest, costs {commission_pct*100}% trade value, trade buffer = {trade_buffer}, trad
        {round(total_return, 1)}% total return, {round(ann_return, 1)}% annualised, Sharpe {round(sharpe
      )
    ) +
  theme_bw()
```


Momentum Backtest − Cash Accounting
Momentum backtest, costs 0% trade value, trade buffer = 0, trade on close
233.7% total return, 137.6% annualised, Sharpe 2.39
Equity
30000
25000
20000
15000
10000
2015−07
2016−01
2016−07
2017−01
Date