Follow the Trend!

Stefan Nicov

2024-07-12

What would be the easiest trading strategy one could think of? It seems that it is to buy when prices go up and sell when prices go down. Why? Because when the prices are going up - we "believe" that they will continue to go up! and when they go down - we "believe" that they will continue to go down!

By analogy with physics, it is kinda saying that Newton's First Law holds: "An object at rest remains at rest, or if in motion, remains in motion at a constant velocity unless acted on by a net external force". In this strategy, we make use of the model in which price changes, once initiated, are continuing the movement by inertia. In industry, such strategies are called momentum trading strategies. The name momentum would be quite fitted here, since a body's momentum $\vec{p}$ is changed when there is an external force acting. $\vec{F} = \frac{d\vec{p}}{dt}$ and therefore for the momentum to change direction - we need to wait for a certain amount of time until the external force acts enough to change it's direction.

In this post we will try to implement such a strategy. As direct inspiration served Dean Markwick's **post**. I will work in R and use the Yahoo Finance historical data through the `yahoofinancer` library.

Load the necessary libraries:

```
library(yahoofinancer) #for historical data
library(ggplot2) #for plots
library(zoo) #for rolling values calculation
```

For strategy backtesting we will use three ETF's: SPY for the stock class, BND for the bond asset class, and GLD for gold. We expect these three to be "independent".

Let's obtain our market data for the past 10 years:

```
spy_ticker <- Ticker$new('SPY')
bnd_ticker <- Ticker$new('BND')
gld_ticker = Ticker$new('GLD')

spy <- spy_ticker$get_history(start = "2014-07-12", end = "2024-07-12", interval = "1d")

bnd <- bnd_ticker$get_history(start = "2014-07-12", end = "2024-07-12", interval = "1d")

gld <- gld_ticker$get_history(start = "2014-07-12", end = "2024-07-12", interval = "1d")
```

Now let's do a bit of data cleaning, by keeping the date, open and close column. For this I will create a function that returns a cleaned df

```
clean_df <- function(df) {
  df$volume <- NULL
  df$high <- NULL
  df$low <- NULL
```

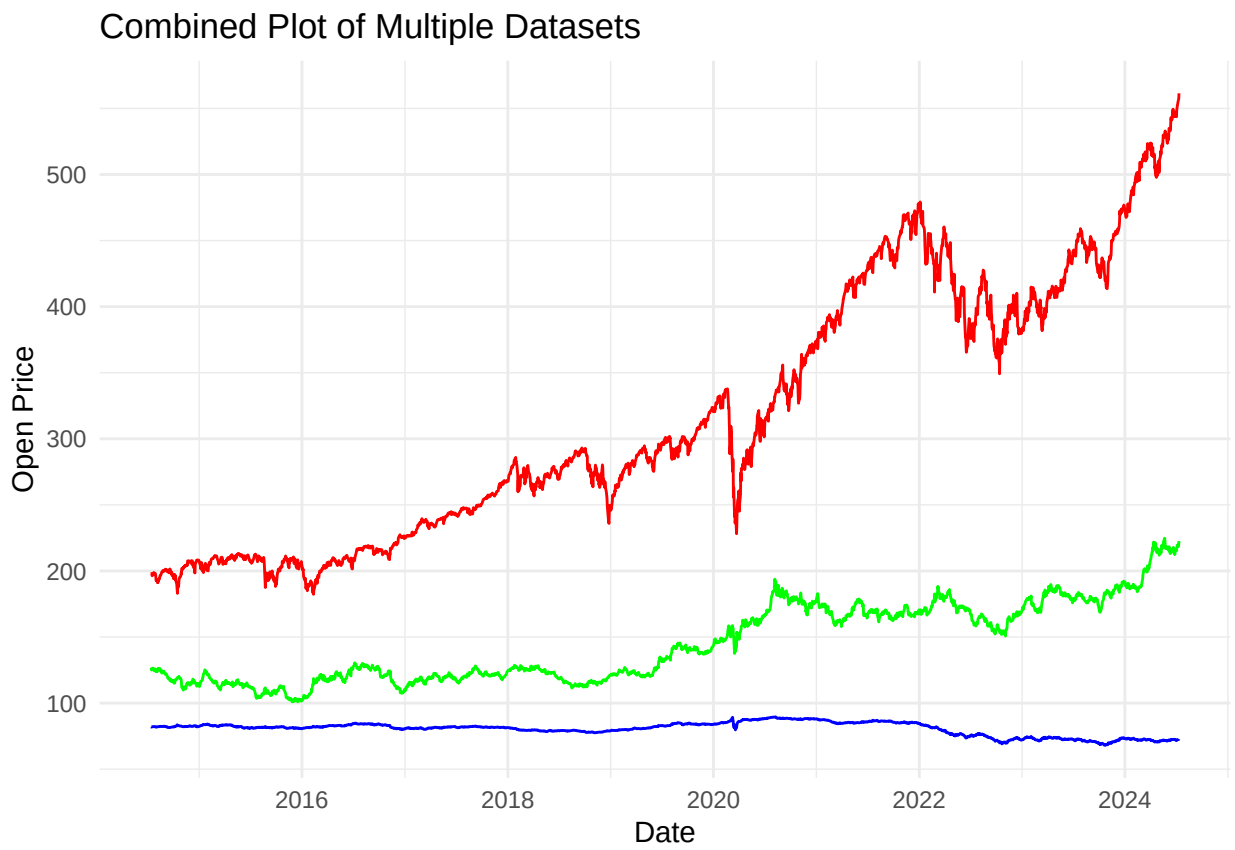
```
  df$adj_close <- NULL
  df$date <- as.Date(df$date, format = "%Y-%m-%d %H:%M")
  return(df)

}
```

Let's obtained our cleaned dataframes:

```
bnd_clean <- clean_df(bnd)
gld_clean <- clean_df(gld)
spy_clean <- clean_df(spy)
```

Let's plot the prices of the ETF's.

```
ggplot()+
  geom_line(data = bnd_clean, aes(x = date, y = open), color = "blue") +
  geom_line(data = spy_clean, aes(x = date, y = open), color = "red") +
  geom_line(data = gld_clean, aes(x = date, y = open), color = "green") +

  labs(title = "Combined Plot of Multiple Datasets",
       x = "Date",
       y = "Open Price") +
  theme_minimal()
```



We can see that they differ a lot both by price and price volatility: BND didn't change much and stays in the range 70 to 80, GLD is in the range 100-250 so far and has more volatility, and SPY is in the range

200-600 with the highest volatility. To take into account different scales and ranges of these values, we will standardize the data. For this, we will work with log of the returns of the close to close move of individual ETS's, and then calculate a running standard deviation of the prices that represents the volatility we will use to standardize the log og the returns.

```
RunVar <- function(v, windowsize){

  running_variance <- rollapply(v, width = windowsize,
                                FUN = sd,
                                by.column = FALSE,
                                align = "right")

  running_variance <- c(rep(NA, windowsize - 1), running_variance)

  return(running_variance)
}



addValues <- function(df){
  df$return <- c(NA, diff(log(df$close)))
  df$RunVol <- sqrt(RunVar(df$return, 256))
  df$rhat <- df$return/df$RunVol
  df$AvgNormReturn <- mean(na.omit(df$rhat))
  df$StdNormReturn <- sd(na.omit(df$rhat))
  return(df)
}
```

Now let's apply these functions to our dataframes and then remove all NA values caused by the running volatility calculation.

```
spy_new <- na.omit(addValues(spy_clean))
gld_new <- na.omit(addValues(gld_clean))
bnd_new <- na.omit(addValues(bnd_clean))
```

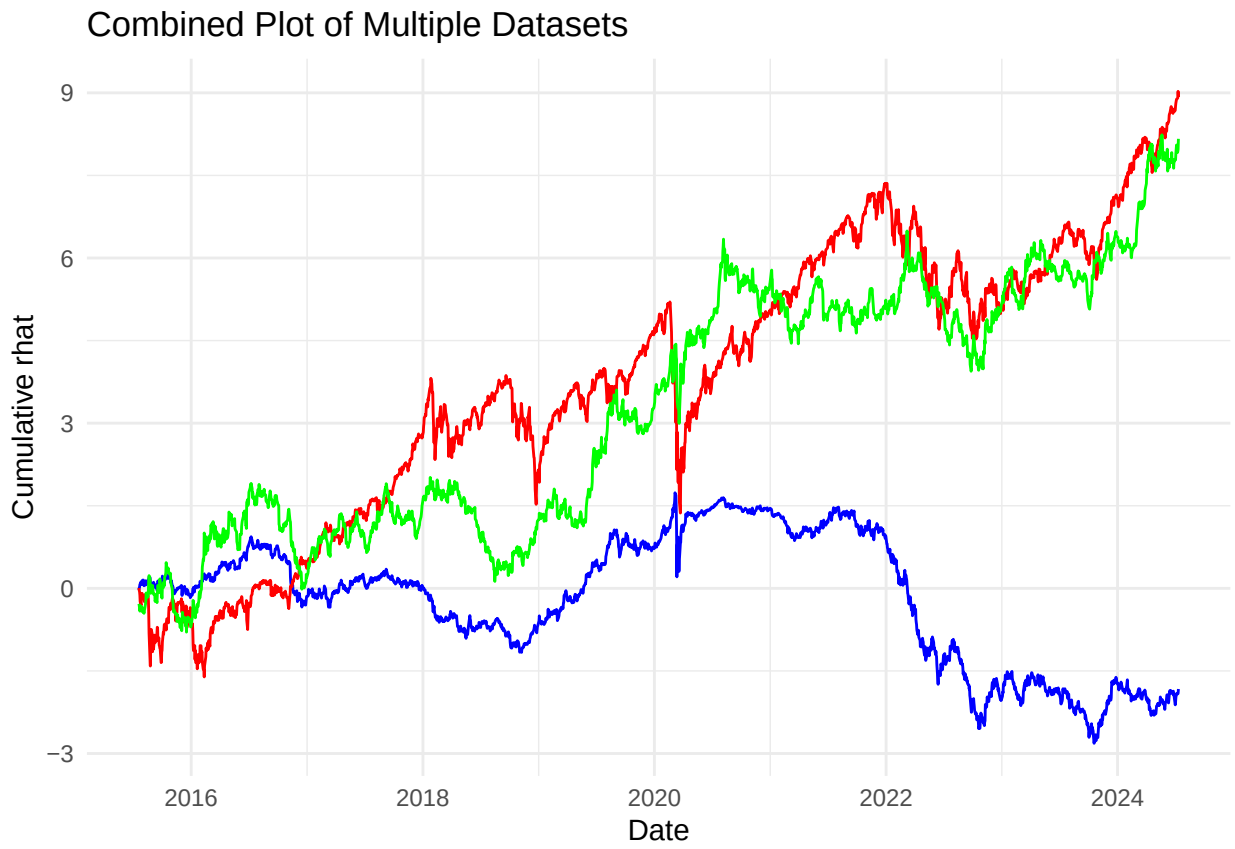
Now we will calculate the cumulative sums of the rhat and return columns.

```
spy_new$rhatC <- cumsum(spy_new$rhat)
spy_new$returnC <- cumsum(spy_new$return)

gld_new$rhatC <- cumsum(gld_new$rhat)
gld_new$returnC <- cumsum(gld_new$return)

bnd_new$rhatC <- cumsum(bnd_new$rhat)
bnd_new$returnC <- cumsum(bnd_new$return)
```

Now let's plot these!

```
ggplot()+
  geom_line(data = bnd_new, aes(x = date, y = rhatC), color = "blue") +
  geom_line(data = spy_new, aes(x = date, y = rhatC), color = "red") +
  geom_line(data = gld_new, aes(x = date, y = rhatC), color = "green") +
```

```
  labs(title = "Combined Plot of Multiple Datasets",
       x = "Date",
       y = "Cumulative rhat") +
  theme_minimal()
```



Now, it's time to build the signal. The most simple version of it consists of looking at the 100-day simple moving average of the `rhat` value: long if it is positive and short if it is negative.

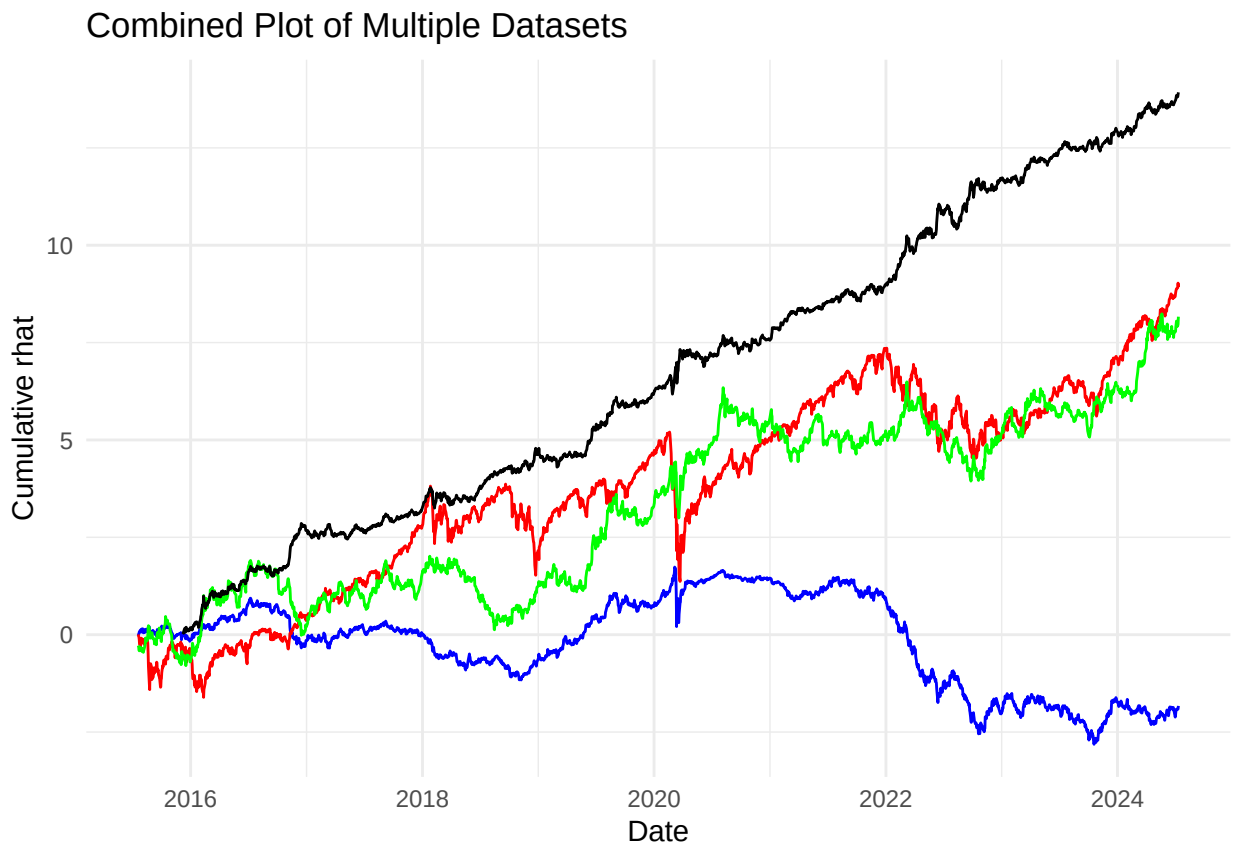
Let's calculate the signal column for each dataframe and then calculate the return out of this signal. Note that we will use a 1/3 factor to account that our budget is evenly splitted across these 3 assets.

```
addSignal <- function(df){

  running_mean <- rollapply(df$rhat, width = 100,
                            FUN = mean,
                            by.column = FALSE,
                            align = "right")

  df$signal <- c(rep(NA, 100 - 1), sign(running_mean))

  return(df)

}


bnd_final <- na.omit(addSignal(bnd_new))
```

```r
gld_final <- na.omit(addSignal(gld_new))
spy_final <- na.omit(addSignal(spy_new))



result_bnd <- sum(bnd_final$signal*bnd_final$rhat)
result_gld <- sum(gld_final$signal*gld_final$rhat)
result_spy <- sum(spy_final$signal*spy_final$rhat)

resultC = 1/3*(cumsum(bnd_final$signal*bnd_final$rhat) +
               cumsum(gld_final$signal*gld_final$rhat) +
               cumsum(spy_final$signal*spy_final$rhat))

resultC <- resultC

result_df <- data.frame(date = bnd_final$date, result = resultC)

print(1/3*(result_gld+result_spy+result_bnd))
```

```
## [1] 13.92546
```

```r
ggplot()+
  geom_line(data = bnd_new, aes(x = date, y = rhatC), color = "blue") +
  geom_line(data = spy_new, aes(x = date, y = rhatC), color = "red") +
  geom_line(data = gld_new, aes(x = date, y = rhatC), color = "green") +
  geom_line(data = result_df, aes(x = date, y = result), color = "black") +
  labs(title = "Combined Plot of Multiple Datasets",
       x = "Date",
       y = "Cumulative rhat") +
  theme_minimal()
```



As we observe, we significantly outperformed every asset! A simple rule therefore is more than enough to perform well on the market. Then why so few firms are outperforming the market in reality?

There are several reasons. First, the price we can trade at is not the close price as in this model, since we will use instead the open price of the next day for real-life trading. Therefore, our signal will be triggered when the observed price will be different from the theoretical, calculated price. Taking just that into account already provides an inferior strategy.

Moreover, there are actual costs, like brokerage fees whenever you buy or sell an ETF, borrow fees whenever you short an ETF (since shorting is basically borrowing the asset for the time period you're shorting). With all these fees, there will be now a minimum budget you can implement this strategy with. Moreover, taking the fees into account, as well produces an inferior model that is not guaranteed to significantly outperform the market.

As further playing with the signal, I tried to vary the window size of the running mean inside the signal calculation. Surprisingly, the returns only were growing when decreasing the window size, with a peak at the 2-day moving average. I asked the author of the post that I am replicating about this seemingly odd behavior and this is the answer I obtained:

> Optimising the window size can sometimes lead to overfitting and harder to verify out of sample perfomrnace. Practically, window size is driven by trading costs, shorter windows means more trading and thus trading costs, so your strategy becomes sensitive to market impact