Documentación Ejecutable — Scripts 1–3 (Selección, Construcción y Rebalanceo de Portafolio)

Pipeline multi-activo con selección (Momentum 12–1), optimización MV y comparación de políticas de rebalanceo

Fabian Abarza

2026-02-06

Contents

1 Visión general

Este documento explica y ejecuta un proceso completo para construir y evaluar un portafolio en tres etapas:

1. Selección de activos (Script 1): ordena un conjunto de acciones y elige un grupo pequeño para seguir trabajando.
2. Construcción del portafolio (Script 2): calcula cuánto invertir en cada activo (los pesos objetivo) usando una optimización.
3. Rebalanceo y recomendación (Script 3): prueba distintas reglas para "volver" a los pesos objetivo y compara resultados con un benchmark.

   Nota: Los datos se descargan desde Yahoo Finance. Algunas métricas pueden no estar disponibles para todos los tickers. En ese caso, el proceso sigue funcionando y se apoya más en la señal de precios.

---

2 SCRIPT 1 — Selección de Activos (Yahoo Finance)

2.1 Objetivo

El objetivo es ordenar un universo de acciones y seleccionar `top_n` para construir el portafolio después.

La señal principal se llama Momentum 12–1. En simple: busca acciones que han tenido buen desempeño en un periodo largo, pero evita usar el último mes para reducir ruido de muy corto plazo.

2.2 0) Paquetes

Este bloque carga librerías para:

- Descargar datos desde Yahoo (`quantmod`)
- Trabajar con series de tiempo (`xts`, `zoo`)
- Calcular retornos y métricas (`PerformanceAnalytics`)
- Ordenar y transformar tablas (`dplyr`, `tibble`)

```
suppressPackageStartupMessages({
  library(quantmod)
  library(PerformanceAnalytics)
  library(xts)
  library(zoo)
  library(dplyr)
  library(tibble)
})
```

2.3 1) Configuración (lo que puedes editar)

Aquí defines los parámetros principales:

- `universe`: lista de activos que se evaluarán

- `top_n`: cuántos activos se seleccionarán
- `from_date_prices`: desde cuándo se descargan precios (ideal: 2 años o más)
- `trading_days_month`: aproximación de días hábiles por mes (21 es común)
- `score_weights`: "importancia" de cada criterio en el puntaje final

```
cfg_sel <- list(
  universe = c("AAPL","MSFT","AMZN","GOOGL","NVDA","META","JPM","V","MA","UNH",
               "XOM","COST","PEP","KO","AVGO","LLY","HD","PG","ADBE","TSLA"),
  top_n = 8,
  from_date_prices = "2022-01-01",
  trading_days_month = 21,
  score_weights = list(
    value = 0.35,       # P/E bajo
    momentum = 0.35,    # momentum alto
    size = 0.10,        # market cap alto (tamaño)
    risk = 0.10,        # beta baja
    yield = 0.10        # dividend yield alto
  )
)
```

2.4 2) Utilidades (limpieza y normalización)

En datos financieros es común recibir valores como `"2.3T"` o `"3.5%"`. Este bloque define funciones para:

- Convertir textos a números (por ejemplo, billones/millones o porcentajes)
- Limitar valores extremos (para que un dato raro no distorsione todo)
- Estandarizar variables para combinarlas en un solo puntaje

```
parse_suffix_number <- function(x) {
  if (is.null(x) || length(x) == 0) return(NA_real_)
  x <- as.character(x)
  x <- gsub(",", "", x)
  if (is.na(x) || x == "" || x == "N/A") return(NA_real_)

  mult <- 1
  last <- substr(x, nchar(x), nchar(x))
  if (last %in% c("T","B","M","K")) {
    mult <- switch(last, T=1e12, B=1e9, M=1e6, K=1e3)
    x <- substr(x, 1, nchar(x) - 1)
  }
  val <- suppressWarnings(as.numeric(x))
  if (is.na(val)) NA_real_ else val * mult
}

parse_percent <- function(x) {
  if (is.null(x) || length(x) == 0) return(NA_real_)
  x <- as.character(x)
  x <- gsub("%", "", x)
  x <- gsub(",", "", x)
  val <- suppressWarnings(as.numeric(x))
  if (is.na(val)) NA_real_ else val / 100
}

winsorize <- function(x, p = 0.01) {
  x <- as.numeric(x)
```

```
  if (all(is.na(x))) return(x)
  qs <- stats::quantile(x, probs = c(p, 1 - p), na.rm = TRUE)
  x[x < qs[1]] <- qs[1]
  x[x > qs[2]] <- qs[2]
  x
}

zscore_safe <- function(x) {
  x <- as.numeric(x)
  s <- stats::sd(x, na.rm = TRUE)
  if (is.na(s) || s == 0) return(rep(0, length(x)))
  (x - mean(x, na.rm = TRUE)) / s
}
```

2.5 3) Extracción de datos (Yahoo)

2.5.1 3.1 Precios ajustados

Se descargan precios ajustados (consideran dividendos y splits) para que los cálculos sean comparables en el tiempo. La salida es un `xts` con una columna por ticker.

```
get_prices_yahoo <- function(tickers, from) {
  env <- new.env()
  suppressWarnings(getSymbols(tickers, src = "yahoo", from = from, env = env, auto.assign = TRUE))
  plist <- lapply(tickers, function(tk) Ad(get(tk, envir = env)))
  p <- do.call(merge, plist)
  colnames(p) <- tickers
  na.omit(p)
}
```

2.5.2 3.2 Métricas complementarias (getQuote)

Este bloque intenta traer algunas métricas para complementar el ranking:

- `market_cap`: tamaño de la empresa
- `pe_ttm`: relación precio/utilidad (indicador simple de valoración)
- `beta`: qué tan sensible es frente al mercado
- `div_yield`: rendimiento por dividendos

Si una métrica no aparece, el proceso igual continúa.

```
yahoo_get_quote <- function(tickers) {
  q <- suppressWarnings(getQuote(tickers, src = "yahoo"))
  q <- tibble::rownames_to_column(as.data.frame(q), var = "ticker")
  colnames(q) <- gsub("\\\\.", "_", colnames(q))

  mc_col   <- intersect(c("Market_Cap","MarketCap"), names(q))[1]
  pe_col   <- intersect(c("PE","P_E"), names(q))[1]
  beta_col <- intersect(c("Beta"), names(q))[1]
  yld_col  <- intersect(c("Yield"), names(q))[1]

  q %>%
    mutate(
      market_cap = if (!is.na(mc_col)) parse_suffix_number(.data[[mc_col]]) else NA_real_,
      pe_ttm     = if (!is.na(pe_col)) suppressWarnings(as.numeric(.data[[pe_col]])) else NA_real_,
      beta       = if (!is.na(beta_col)) suppressWarnings(as.numeric(.data[[beta_col]])) else NA_real_,
```

```
      div_yield  = if (!is.na(yld_col)) parse_percent(.data[[yld_col]]) else NA_real_
    ) %>%
    select(ticker, market_cap, pe_ttm, beta, div_yield)
}
```

2.6 4) Señal principal: Momentum 12–1

Momentum 12–1 compara el precio "aproximadamente hace 1 mes" contra el precio "aproximadamente hace 12 meses". El bloque valida que exista historia suficiente para no calcular una señal poco confiable.

```
compute_momentum_12_1 <- function(prices_xts, trading_days_month = 21) {
  n <- nrow(prices_xts)
  need <- 12 * trading_days_month + trading_days_month + 5
  if (n < need) stop("Insuficientes datos para momentum 12-1. Usa from_date_prices más antigua.")
  idx_12m <- 12 * trading_days_month
  idx_1m  <- 1  * trading_days_month
  p_1m  <- as.numeric(prices_xts[n - idx_1m, ])
  p_12m <- as.numeric(prices_xts[n - idx_12m, ])
  mom <- (p_1m / p_12m) - 1
  names(mom) <- colnames(prices_xts)
  mom
}
```

2.7 5) Puntaje y selección

Este bloque:

- Requiere Momentum para evaluar un ticker
- Usa métricas complementarias cuando están disponibles
- Si hay pocos tickers con soporte, usa un "plan B" y rankea principalmente por Momentum
- Calcula un puntaje total y penaliza suavemente cuando faltan datos

```
select_assets_yahoo <- function(cfg) {

  tickers <- unique(cfg$universe)

  message("[1/4] Descargando métricas Yahoo (getQuote)...")
  q <- yahoo_get_quote(tickers)

  message("[2/4] Descargando precios Yahoo para momentum...")
  prices <- get_prices_yahoo(tickers, from = cfg$from_date_prices)

  tickers_ok_prices <- colnames(prices)
  q <- q %>% filter(ticker %in% tickers_ok_prices)

  if (nrow(q) == 0) stop("No se pudo obtener getQuote para ningún ticker con precios válidos.")

  message("[3/4] Calculando momentum 12-1...")
  mom <- compute_momentum_12_1(prices[, q$ticker], cfg$trading_days_month)
  mom_df <- data.frame(ticker = names(mom), momentum_12_1 = as.numeric(mom), stringsAsFactors = FALSE)

  df <- left_join(q, mom_df, by = "ticker")

  message("[4/4] Diagnóstico de NA (cobertura de datos):")
  print(colSums(is.na(df)))
```

```r
df2 <- df %>%
  mutate(
    has_mom = !is.na(momentum_12_1),
    n_support = rowSums(!is.na(across(c(pe_ttm, market_cap, beta, div_yield))))
  ) %>%
  filter(has_mom, n_support >= 1) %>%
  select(-has_mom, -n_support)

if (nrow(df2) < max(5, cfg$top_n)) {
  warning(
    paste0(
      "Pocos tickers con datos de soporte (", nrow(df2), "). ",
      "Relajando filtro: ranking se basará principalmente en Momentum + lo disponible."
    )
  )
  df2 <- df %>% filter(!is.na(momentum_12_1))
  if (nrow(df2) == 0) stop("Ni siquiera hay momentum disponible. Revisa tickers o from_date_prices.")
}

w <- cfg$score_weights

ranked <- df2 %>%
  mutate(
    pe_w   = winsorize(pe_ttm),
    cap_w  = winsorize(market_cap),
    beta_w = winsorize(beta),
    yld_w  = winsorize(div_yield),
    mom_w  = winsorize(momentum_12_1),

    z_value = -zscore_safe(pe_w),
    z_size  =  zscore_safe(cap_w),
    z_risk  = -zscore_safe(beta_w),
    z_yield =  zscore_safe(yld_w),
    z_mom   =  zscore_safe(mom_w),

    score = w$value    * z_value +
      w$size     * z_size  +
      w$risk     * z_risk  +
      w$yield    * z_yield +
      w$momentum * z_mom,

    n_na = rowSums(is.na(across(c(pe_ttm, market_cap, beta, div_yield)))),
    score = score - 0.05 * n_na
  ) %>%
  arrange(desc(score))

selected <- head(ranked$ticker, cfg$top_n)

list(
  raw_table = df2,
  ranked_table = ranked,
  tickers_selected = selected
)
```

```
}
```

2.8 6) Ejecución (resultados del Script 1)

Este bloque ejecuta la selección y muestra:

- El ranking (primeras 15 filas)
- Los tickers seleccionados (se usan en el Script 2)

```
sel <- select_assets_yahoo(cfg_sel)
```

```
## [1/4] Descargando métricas Yahoo (getQuote)...
## [2/4] Descargando precios Yahoo para momentum...
## [3/4] Calculando momentum 12-1...
## [4/4] Diagnóstico de NA (cobertura de datos):
##        ticker    market_cap        pe_ttm          beta     div_yield
##             0            20            20            20            20
## momentum_12_1
##             0
## Warning in select_assets_yahoo(cfg_sel): Pocos tickers con datos de soporte
## (0). Relajando filtro: ranking se basará principalmente en Momentum + lo
## disponible.
```

```
cat("
===== TOP RANKING (primeras 15 filas) =====
")
```

```
##
## ===== TOP RANKING (primeras 15 filas) =====
```

```
print(head(sel$ranked_table, 15))
```

```
##    ticker market_cap pe_ttm beta div_yield momentum_12_1 pe_w cap_w beta_w
## 1    NVDA         NA     NA   NA        NA   0.578518521   NA    NA     NA
## 2    AVGO         NA     NA   NA        NA   0.559485823   NA    NA     NA
## 3   GOOGL         NA     NA   NA        NA   0.529077892   NA    NA     NA
## 4     LLY         NA     NA   NA        NA   0.297682536   NA    NA     NA
## 5     JPM         NA     NA   NA        NA   0.274431051   NA    NA     NA
## 6    MSFT         NA     NA   NA        NA   0.168949911   NA    NA     NA
## 7     XOM         NA     NA   NA        NA   0.141257732   NA    NA     NA
## 8    AAPL         NA     NA   NA        NA   0.132072828   NA    NA     NA
## 9      KO         NA     NA   NA        NA   0.114432812   NA    NA     NA
## 10   TSLA         NA     NA   NA        NA   0.103898424   NA    NA     NA
## 11      V         NA     NA   NA        NA   0.043313370   NA    NA     NA
## 12     MA         NA     NA   NA        NA   0.042398497   NA    NA     NA
## 13    PEP         NA     NA   NA        NA   0.007129734   NA    NA     NA
## 14   AMZN         NA     NA   NA        NA  -0.004668284   NA    NA     NA
## 15   META         NA     NA   NA        NA  -0.058914292   NA    NA     NA
##    yld_w        mom_w z_value z_size z_risk z_yield       z_mom       score
## 1     NA  0.574902309       0      0      0       0  1.89078173  0.46177361
## 2     NA  0.559485823       0      0      0       0  1.82967097  0.44038484
## 3     NA  0.529077892       0      0      0       0  1.70913430  0.39819701
## 4     NA  0.297682536       0      0      0       0  0.79188593  0.07716008
## 5     NA  0.274431051       0      0      0       0  0.69971732  0.04490106
```

```
## 6     NA  0.168949911       0      0      0       0  0.28159137 -0.10144302
## 7     NA  0.141257732       0      0      0       0  0.17181991 -0.13986303
## 8     NA  0.132072828       0      0      0       0  0.13541106 -0.15260613
## 9     NA  0.114432812       0      0      0       0  0.06548625 -0.17707981
## 10    NA  0.103898424       0      0      0       0  0.02372807 -0.19169518
## 11    NA  0.043313370       0      0      0       0 -0.21643035 -0.27575062
## 12    NA  0.042398497       0      0      0       0 -0.22005689 -0.27701991
## 13    NA  0.007129734       0      0      0       0 -0.35986184 -0.32595164
## 14    NA -0.004668284       0      0      0       0 -0.40662904 -0.34232016
## 15    NA -0.058914292       0      0      0       0 -0.62165956 -0.41758085
##    n_na
## 1     4
## 2     4
## 3     4
## 4     4
## 5     4
## 6     4
## 7     4
## 8     4
## 9     4
## 10    4
## 11    4
## 12    4
## 13    4
## 14    4
## 15    4
```

```
cat("
===== TICKERS SELECCIONADOS =====
")
```

```
##
## ===== TICKERS SELECCIONADOS =====
```

```
print(sel$tickers_selected)
```

```
## [1] "NVDA"  "AVGO"  "GOOGL" "LLY"   "JPM"   "MSFT"  "XOM"   "AAPL"
```

```
tickers_eq <- sel$tickers_selected
```

---

3 SCRIPT 2 — Construcción del Portafolio (Optimización Media–Varianza)

3.1 Objetivo

En esta etapa se construye un portafolio objetivo, es decir, una lista de pesos que indica cuánto invertir en cada activo seleccionado.

La optimización sigue reglas simples: - No se usan posiciones cortas - Los pesos deben sumar 1 (se invierte el 100%) - Cada activo tiene un mínimo y máximo para evitar concentraciones excesivas

Opcionalmente se agrega `AGG` como activo defensivo.

3.2 0) Paquetes

```
suppressPackageStartupMessages({
  library(quantmod)
  library(PerformanceAnalytics)
  library(xts)
  library(zoo)
  library(PortfolioAnalytics)
  library(DEoptim)
})
```

3.3 1) Configuración

- `min_w` / `max_w`: límites de peso por activo
- `risk_aversion`: cuánto se penaliza el riesgo (más alto = más conservador)
- `include_defensive`: si se incluye o no el activo defensivo
- `free_risk_min_w`: peso mínimo del defensivo (si se quiere asegurar)

```
cfg_port <- list(
  from_date_prices = "2022-01-01",
  returns_method = "log",
  min_w = 0.00,
  max_w = 0.35,
  risk_aversion = 10,
  seed = 123,
  include_defensive = TRUE,
  free_risk_ticker = "AGG",
  free_risk_min_w = 0.05
)
```

3.4 2) Inputs desde Script 1

Este bloque verifica que `sel` exista y define el universo final para optimizar.

```
if (!exists("sel")) stop("No existe objeto 'sel'. Ejecuta Script 1 primero (sel$tickers_selected).")

tickers_eq <- sel$tickers_selected

tickers <- if (cfg_port$include_defensive) {
  unique(c(tickers_eq, cfg_port$free_risk_ticker))
} else {
  tickers_eq
}
```

3.5 3) Datos: precios y retornos

Se descargan precios ajustados y se convierten a retornos. Por defecto se usan retornos log para mantener consistencia matemática al componer rendimientos a lo largo del tiempo.

```
get_prices_yahoo <- function(tickers, from) {
  env <- new.env()
  suppressWarnings(
    getSymbols(tickers, src = "yahoo", from = from, env = env, auto.assign = TRUE)
  )
  plist <- lapply(tickers, function(tk) Ad(get(tk, envir = env)))
```

```
  p <- do.call(merge, plist)
  colnames(p) <- tickers
  na.omit(p)
}

prices_to_returns <- function(prices, method = "log") {
  na.omit(Return.calculate(prices, method = method))
}

prices  <- get_prices_yahoo(tickers, from = cfg_port$from_date_prices)
returns <- prices_to_returns(prices, method = cfg_port$returns_method)

if (nrow(returns) < 60) warning("Pocos datos de retornos (<60 obs). Resultados pueden ser inestables.")
if (anyNA(returns)) returns <- na.omit(returns)
```

3.6 4) Optimización

Este bloque define restricciones y optimiza buscando un equilibrio entre retorno esperado y riesgo (volatilidad).

```
build_target_portfolio_mv <- function(returns,
                                      min_w,
                                      max_w,
                                      risk_aversion,
                                      seed,
                                      free_risk_ticker = NULL,
                                      free_risk_min_w = 0.0) {

  set.seed(seed)
  assets <- colnames(returns)

  port <- PortfolioAnalytics::portfolio.spec(assets)

  port <- PortfolioAnalytics::add.constraint(port, type = "full_investment")
  port <- PortfolioAnalytics::add.constraint(port, type = "long_only")
  port <- PortfolioAnalytics::add.constraint(port, type = "box", min = min_w, max = max_w)

  if (!is.null(free_risk_ticker) &&
      free_risk_ticker %in% assets &&
      free_risk_min_w > 0) {

    port <- PortfolioAnalytics::add.constraint(
      port,
      type = "box",
      min = ifelse(assets == free_risk_ticker, free_risk_min_w, min_w),
      max = max_w
    )
  }

  port <- PortfolioAnalytics::add.objective(port, type = "risk",   name = "StdDev", multiplier = risk_a
  port <- PortfolioAnalytics::add.objective(port, type = "return", name = "mean",   multiplier = 1)

  if (!exists(".storage", envir = .GlobalEnv)) .storage <<- new.env(parent = emptyenv())
```

```
  opt <- PortfolioAnalytics::optimize.portfolio(
    R = returns,
    portfolio = port,
    optimize_method = "DEoptim",
    trace = FALSE
  )

  w <- PortfolioAnalytics::extractWeights(opt)
  w <- w / sum(w)

  list(
    opt = opt,
    weights_target = w,
    portfolio_spec = port
  )
}
```

3.7 5) Ejecución del Script 2

Este bloque calcula los pesos objetivo y deja todo listo para el rebalanceo.

```
port <- build_target_portfolio_mv(
  returns = returns,
  min_w = cfg_port$min_w,
  max_w = cfg_port$max_w,
  risk_aversion = cfg_port$risk_aversion,
  seed = cfg_port$seed,
  free_risk_ticker = if (cfg_port$include_defensive) cfg_port$free_risk_ticker else NULL,
  free_risk_min_w = cfg_port$free_risk_min_w
)
```

```
## Leverage constraint min_sum and max_sum are restrictive,
##               consider relaxing. e.g. 'full_investment' constraint should be min_sum=0.99 and max_sum
```

```
## Iteration: 1 bestvalit: 0.095095 bestmemit:    0.034000    0.000000    0.020000    0.010000    0.002
## Iteration: 2 bestvalit: 0.095095 bestmemit:    0.034000    0.000000    0.020000    0.010000    0.002
## Iteration: 3 bestvalit: 0.095095 bestmemit:    0.034000    0.000000    0.020000    0.010000    0.002
## Iteration: 4 bestvalit: 0.095095 bestmemit:    0.034000    0.000000    0.020000    0.010000    0.002
## Iteration: 5 bestvalit: 0.095095 bestmemit:    0.034000    0.000000    0.020000    0.010000    0.002
## Iteration: 6 bestvalit: 0.095095 bestmemit:    0.034000    0.000000    0.020000    0.010000    0.002
## Iteration: 7 bestvalit: 0.095095 bestmemit:    0.034000    0.000000    0.020000    0.010000    0.002
## Iteration: 8 bestvalit: 0.095095 bestmemit:    0.034000    0.000000    0.020000    0.010000    0.002
## Iteration: 9 bestvalit: 0.095095 bestmemit:    0.034000    0.000000    0.020000    0.010000    0.002
## Iteration: 10 bestvalit: 0.095095 bestmemit:    0.034000    0.000000    0.020000    0.010000    0.00
## Iteration: 11 bestvalit: 0.095095 bestmemit:    0.034000    0.000000    0.020000    0.010000    0.00
## Iteration: 12 bestvalit: 0.095095 bestmemit:    0.034000    0.000000    0.020000    0.010000    0.00
## Iteration: 13 bestvalit: 0.095095 bestmemit:    0.034000    0.000000    0.020000    0.010000    0.00
## Iteration: 14 bestvalit: 0.095095 bestmemit:    0.034000    0.000000    0.020000    0.010000    0.00
## Iteration: 15 bestvalit: 0.095095 bestmemit:    0.034000    0.000000    0.020000    0.010000    0.00
## Iteration: 16 bestvalit: 0.095095 bestmemit:    0.034000    0.000000    0.020000    0.010000    0.00
## [1] 0.034 0.000 0.020 0.010 0.002 0.048 0.214 0.350 0.322
```

```
cat("
===== PESOS OBJETIVO (weights_target) =====
")
```

```
##
## ===== PESOS OBJETIVO (weights_target) =====
```

```
print(round(port$weights_target, 4))
```

```
##  NVDA  AVGO GOOGL   LLY   JPM  MSFT   XOM  AAPL   AGG
## 0.034 0.000 0.020 0.010 0.002 0.048 0.214 0.350 0.322
```

```
cat("Suma pesos:", round(sum(port$weights_target), 6), "
")
```

```
## Suma pesos: 1
```

```
port$prices <- prices
port$returns_log <- returns
port$tickers <- tickers

if (any(port$weights_target < -1e-8)) warning("Existen pesos negativos (no esperado en long-only).")
if (abs(sum(port$weights_target) - 1) > 1e-6) warning("Los pesos no suman 1 (revisa restricciones).")

cat("
Listo. Usa port$weights_target en el Script 3.
")
```

```
##
## Listo. Usa port$weights_target en el Script 3.
```

---

4 SCRIPT 3 — Rebalanceo (Backtest + Diagnóstico + Recomendación)

4.1 Objetivo

Esta etapa compara reglas de rebalanceo para decidir cuál funciona mejor en la práctica. La comparación busca equilibrio entre:

- Resultado (retorno y riesgo)
- Diferencia frente al benchmark
- Esfuerzo operativo (cuántas veces hay que rebalancear)

Reglas evaluadas: - Periodic: rebalancea en fechas fijas - Bands: rebalancea solo si el portafolio se aleja del objetivo más de un umbral - Glidepath (+ bands): ajusta gradualmente el peso en acciones vs defensivo con el tiempo

4.2 0) Pre-checks + paquetes

```
if (!exists("port")) stop("No existe objeto 'port'. Ejecuta Script 2 primero.")

suppressPackageStartupMessages({
  library(xts)
  library(zoo)
  library(PerformanceAnalytics)
  library(dplyr)
  library(quantmod)
})
```

```r
returns_log <- port$returns_log
w_target    <- port$weights_target

stopifnot(is.xts(returns_log))
stopifnot(is.numeric(w_target), !is.null(names(w_target)))
stopifnot(all(names(w_target) %in% colnames(returns_log)))
```

4.3 1) Configuración

```r
cfg_reb <- list(
  returns_are_log = TRUE,
  band = 0.05,
  k_periodic = 3,
  k_bands    = 1,
  use_glide = TRUE,
  free_risk_ticker = "AGG",
  horizon_years = 20,
  start_equity  = 0.80,
  end_equity    = 0.30,
  k_glide = 3,
  use_bands_glide = TRUE,
  benchmark = "60_40",
  bench_from = "2022-01-01",
  tc_bps = 10,
  risk_window_days = 126,
  risk_tol = 0.02,
  n_sims = 1500
)
```

4.4 2) Helpers: riqueza, deriva de pesos y retornos

```r
wealth_step <- function(wealth, port_ret, returns_are_log) {
  if (returns_are_log) wealth * exp(port_ret) else wealth * (1 + port_ret)
}

weight_drift <- function(w, r_vec, returns_are_log) {
  gross <- if (returns_are_log) exp(r_vec) else (1 + r_vec)
  w_new <- w * gross
  w_new / sum(w_new)
}

to_discrete_returns_from_wealth <- function(wealth_xts) {
  na.omit(Return.calculate(wealth_xts, method = "discrete"))
}
```

4.5 3) Reglas de rebalanceo

```r
rebalance_periodic <- function(current_w, target_w) {
  list(
    rebalance = TRUE,
    new_w = target_w,
    reason = "Periódico (fecha programada)",
```

```
    max_dev = max(abs(current_w - target_w))
  )
}

rebalance_bands <- function(current_w, target_w, band) {
  dev <- abs(current_w - target_w)
  mx <- max(dev)
  need <- any(dev > band)
  list(
    rebalance = need,
    new_w = if (need) target_w else current_w,
    reason = if (need) paste0("Bandas: max dev=", round(mx,4), " > ", band)
    else      paste0("Bandas: max dev=", round(mx,4), " <= ", band),
    max_dev = mx
  )
}
```

4.6 4) Motor de backtest con logs

El turnover se aproxima como `sum(|Δw|)/2`, una forma estándar de medir cuánto "se mueve" el portafolio cuando se rebalancea.

```
backtest_rebalance <- function(returns, w_target, rule = c("periodic","bands"),
                               k = 1, band = 0.05,
                               returns_are_log = TRUE, verbose = FALSE) {

  rule <- match.arg(rule)
  returns <- returns[, names(w_target)]
  w <- as.numeric(w_target); names(w) <- names(w_target)

  wealth <- 1
  wealth_xts <- xts(rep(NA_real_, nrow(returns)), order.by = index(returns))

  ep <- endpoints(returns, on = "months", k = k)
  reb_idx <- rep(FALSE, nrow(returns))
  reb_idx[ep] <- TRUE
  reb_idx[1] <- TRUE

  log_df <- data.frame(
    date=as.Date(character()),
    rule=character(),
    rebalanced=logical(),
    max_dev=numeric(),
    turnover=numeric(),
    reason=character(),
    stringsAsFactors = FALSE
  )

  for (t in 1:nrow(returns)) {
    r <- as.numeric(returns[t, ])
    port_ret <- sum(w * r)

    wealth <- wealth_step(wealth, port_ret, returns_are_log)
    wealth_xts[t] <- wealth
```

```
    w <- weight_drift(w, r, returns_are_log)

    if (reb_idx[t]) {
      d <- as.Date(index(returns)[t])

      dec <- if (rule == "periodic") {
        rebalance_periodic(w, w_target)
      } else {
        rebalance_bands(w, w_target, band)
      }

      turnover <- if (dec$rebalance) sum(abs(dec$new_w - w))/2 else 0
      w <- dec$new_w

      log_df <- rbind(log_df, data.frame(
        date=d, rule=rule, rebalanced=dec$rebalance,
        max_dev=dec$max_dev, turnover=turnover, reason=dec$reason,
        stringsAsFactors = FALSE
      ))

      if (verbose) cat(sprintf("[%s] %s | reb=%s
", d, dec$reason, ifelse(dec$rebalance,"SÍ","NO")))
    }
  }

  colnames(wealth_xts) <- paste0("Wealth_", rule)
  list(wealth = wealth_xts, log = log_df)
}
```

4.7 5) Glidepath + (opcional) bandas

El glidepath ajusta el peso objetivo en acciones desde `start_equity` a `end_equity` durante `horizon_years`. Es útil si quieres que el portafolio sea más agresivo al inicio y más conservador con el tiempo.

```
glide_path_equity_weight <- function(t, T_total, start_equity, end_equity) {
  w <- start_equity + (end_equity - start_equity) * (t / T_total)
  pmin(pmax(w, 0), 1)
}

make_target_weights_glide <- function(equity_tickers, free_risk_ticker, equity_total_weight) {
  eq_n <- length(equity_tickers)
  w <- rep(0, eq_n + 1)
  names(w) <- c(equity_tickers, free_risk_ticker)
  w[equity_tickers] <- equity_total_weight / eq_n
  w[free_risk_ticker] <- 1 - equity_total_weight
  w
}

backtest_glidepath <- function(returns, equity_tickers, free_risk_ticker,
                               horizon_years, start_equity, end_equity,
                               k = 3, band = 0.05, use_bands = TRUE,
                               returns_are_log = TRUE, verbose = FALSE) {
```

```r
stopifnot(free_risk_ticker %in% colnames(returns))
returns <- returns[, c(equity_tickers, free_risk_ticker)]

ep <- endpoints(returns, on="months", k=k)
reb_idx <- rep(FALSE, nrow(returns))
reb_idx[ep] <- TRUE
reb_idx[1] <- TRUE

start_date <- as.Date(index(returns)[1])
t_years <- as.numeric(as.Date(index(returns)) - start_date) / 365.25

w_target <- make_target_weights_glide(
  equity_tickers, free_risk_ticker,
  glide_path_equity_weight(0, horizon_years, start_equity, end_equity)
)
w <- as.numeric(w_target); names(w) <- names(w_target)

wealth <- 1
wealth_xts <- xts(rep(NA_real_, nrow(returns)), order.by=index(returns))

log_df <- data.frame(
  date=as.Date(character()),
  equity_target=numeric(),
  rebalanced=logical(),
  max_dev=numeric(),
  turnover=numeric(),
  reason=character(),
  stringsAsFactors=FALSE
)

for (i in 1:nrow(returns)) {
  r <- as.numeric(returns[i, ])
  port_ret <- sum(w * r)

  wealth <- wealth_step(wealth, port_ret, returns_are_log)
  wealth_xts[i] <- wealth

  w <- weight_drift(w, r, returns_are_log)

  if (reb_idx[i]) {
    d <- as.Date(index(returns)[i])

    eq_target <- glide_path_equity_weight(t_years[i], horizon_years, start_equity, end_equity)
    w_new_target <- make_target_weights_glide(equity_tickers, free_risk_ticker, eq_target)

    dev <- abs(w - w_new_target)
    mx <- max(dev)

    do_reb <- if (use_bands) any(dev > band) else TRUE
    reason <- if (use_bands) {
      if (do_reb) paste0("Glide+bandas: max dev=", round(mx,4), " > ", band, " | eq=", round(eq_targe
      else       paste0("Glide+bandas: max dev=", round(mx,4), " <= ", band, " | eq=", round(eq_targe
    } else {
```

```
        paste0("Glide periódico | eq=", round(eq_target,3))
      }

      turnover <- if (do_reb) sum(abs(w_new_target - w))/2 else 0
      if (do_reb) w <- w_new_target

      log_df <- rbind(log_df, data.frame(
        date=d, equity_target=eq_target, rebalanced=do_reb,
        max_dev=mx, turnover=turnover, reason=reason,
        stringsAsFactors=FALSE
      ))

      if (verbose) cat(sprintf("[%s] %s
", d, reason))
    }
  }

  colnames(wealth_xts) <- "Wealth_Glide"
  list(wealth=wealth_xts, log=log_df)
}
```

4.8 6) Benchmark y tracking

Se construye un benchmark con `SPY` o con una mezcla 60/40 (SPY/AGG). Luego se calculan dos métricas para comparar:

- Tracking Error: qué tan variable es la diferencia frente al benchmark
- Tracking Difference: diferencia promedio frente al benchmark

```
get_prices_yahoo <- function(tickers, from) {
  env <- new.env()
  suppressWarnings(getSymbols(tickers, src="yahoo", from=from, env=env, auto.assign=TRUE))
  plist <- lapply(tickers, function(tk) Ad(get(tk, envir=env)))
  p <- do.call(merge, plist); colnames(p) <- tickers
  na.omit(p)
}
prices_to_returns <- function(prices, method="discrete") na.omit(Return.calculate(prices, method=method

tracking_error_annualized <- function(port_ret, bench_ret, scale=252) {
  x <- na.omit(merge(port_ret, bench_ret))
  active <- x[,1] - x[,2]
  as.numeric(sd(active) * sqrt(scale))
}

tracking_difference_annualized <- function(port_ret, bench_ret, scale=252) {
  x <- na.omit(merge(port_ret, bench_ret))
  active <- x[,1] - x[,2]
  as.numeric(mean(active) * scale)
}
```

4.9 7) KPIs y métricas de desempeño

```
rebalance_kpis <- function(log_df) {
  if (is.null(log_df) || nrow(log_df) == 0) {
```

```
    return(data.frame(Rebalance_Count=0, Rebalance_Rate=0, Avg_Turnover=0, Avg_MaxDev=NA_real_))
  }
  data.frame(
    Rebalance_Count = sum(log_df$rebalanced),
    Rebalance_Rate  = mean(log_df$rebalanced),
    Avg_Turnover    = mean(log_df$turnover[log_df$rebalanced], na.rm=TRUE),
    Avg_MaxDev      = mean(log_df$max_dev, na.rm=TRUE),
    stringsAsFactors = FALSE
  )
}

portfolio_summary <- function(ret_xts, name) {
  ret_xts <- na.omit(ret_xts)
  data.frame(
    Portfolio = name,
    Annualized_Return = as.numeric(Return.annualized(ret_xts)),
    Annualized_Vol    = as.numeric(StdDev.annualized(ret_xts)),
    Sharpe            = as.numeric(SharpeRatio.annualized(ret_xts, Rf=0)),
    Max_Drawdown      = as.numeric(maxDrawdown(ret_xts)),
    stringsAsFactors = FALSE
  )
}
```

4.10 8) Ejecución de políticas

```
res_periodic <- backtest_rebalance(
  returns_log, w_target,
  rule="periodic", k=cfg_reb$k_periodic, band=cfg_reb$band,
  returns_are_log=cfg_reb$returns_are_log, verbose=FALSE
)

res_bands <- backtest_rebalance(
  returns_log, w_target,
  rule="bands", k=cfg_reb$k_bands, band=cfg_reb$band,
  returns_are_log=cfg_reb$returns_are_log, verbose=FALSE
)

res_glide <- NULL
if (cfg_reb$use_glide && cfg_reb$free_risk_ticker %in% colnames(returns_log)) {
  equity_tickers <- setdiff(colnames(returns_log), cfg_reb$free_risk_ticker)
  res_glide <- backtest_glidepath(
    returns_log, equity_tickers, cfg_reb$free_risk_ticker,
    cfg_reb$horizon_years, cfg_reb$start_equity, cfg_reb$end_equity,
    k=cfg_reb$k_glide, band=cfg_reb$band, use_bands=cfg_reb$use_bands_glide,
    returns_are_log=cfg_reb$returns_are_log, verbose=FALSE
  )
}
```

4.11 9) Wealth → retornos discretos

```
r_periodic <- to_discrete_returns_from_wealth(res_periodic$wealth)
r_bands    <- to_discrete_returns_from_wealth(res_bands$wealth)
```

```
r_glide    <- if (!is.null(res_glide)) to_discrete_returns_from_wealth(res_glide$wealth) else NULL
```

4.12 10) Benchmark

```
bench_spy <- prices_to_returns(get_prices_yahoo("SPY", cfg_reb$bench_from), "discrete")
colnames(bench_spy) <- "SPY"

bench_6040_prices <- get_prices_yahoo(c("SPY", cfg_reb$free_risk_ticker), cfg_reb$bench_from)
bench_6040_rets <- prices_to_returns(bench_6040_prices, "discrete")

bench_6040 <- na.omit(0.60*bench_6040_rets[,"SPY"] + 0.40*bench_6040_rets[,cfg_reb$free_risk_ticker])
colnames(bench_6040) <- "Bench_60_40"

bench_use <- if (cfg_reb$benchmark == "SPY") bench_spy else bench_6040
```

4.13 11) Resultados

```
tbl <- bind_rows(
  cbind(portfolio_summary(r_periodic, "Periodic"), rebalance_kpis(res_periodic$log)),
  cbind(portfolio_summary(r_bands,    "Bands"),    rebalance_kpis(res_bands$log)),
  if (!is.null(r_glide)) cbind(portfolio_summary(r_glide, "GlidePath"), rebalance_kpis(res_glide$log)) e
) %>%
  mutate(
    TE_Ann = c(
      tracking_error_annualized(r_periodic, bench_use),
      tracking_error_annualized(r_bands,    bench_use),
      if (!is.null(r_glide)) tracking_error_annualized(r_glide, bench_use) else NULL
    ),
    TD_Ann = c(
      tracking_difference_annualized(r_periodic, bench_use),
      tracking_difference_annualized(r_bands,    bench_use),
      if (!is.null(r_glide)) tracking_difference_annualized(r_glide, bench_use) else NULL
    )
  ) %>%
  mutate(across(where(is.numeric), ~ round(.x, 4)))

cat("
===== RESULTADOS =====
")
```

```
##
## ===== RESULTADOS =====
```

```
print(tbl)
```

```
##   Portfolio Annualized_Return Annualized_Vol Sharpe Max_Drawdown
## 1  Periodic            0.1257         0.1493 0.8417       0.1599
## 2     Bands            0.1174         0.1494 0.7859       0.1667
## 3 GlidePath            0.2024         0.1739 1.1639       0.2114
##   Rebalance_Count Rebalance_Rate Avg_Turnover Avg_MaxDev TE_Ann TD_Ann
## 1              18         1.0000       0.0405     0.0316 0.0823 0.0583
## 2               5         0.0980       0.0603     0.0261 0.0810 0.0509
## 3               6         0.3333       0.0943     0.0373 0.0860 0.1283
```

4.14 12) Recomendación (Decision Score)

Este puntaje intenta preferir reglas que: - tengan buen desempeño (Sharpe), - no se alejen demasiado del benchmark (TE), - no obliguen a rebalancear muy seguido, - y no generen demasiada rotación (turnover).

```
tbl <- tbl %>%
  mutate(
    Decision_Score = Sharpe - 0.5*TE_Ann - 0.1*Rebalance_Rate - 0.2*Avg_Turnover,
    Decision_Score = round(Decision_Score, 4)
  )

best_policy <- tbl %>%
  arrange(desc(Decision_Score)) %>%
  slice(1) %>%
  pull(Portfolio)

cat("
===== RECOMENDACIÓN =====
")
```

```
##
## ===== RECOMENDACIÓN =====
```

```
print(tbl[, c("Portfolio","Sharpe","TE_Ann","Rebalance_Rate","Avg_Turnover","Decision_Score")])
```

```
##   Portfolio Sharpe TE_Ann Rebalance_Rate Avg_Turnover Decision_Score
## 1  Periodic 0.8417 0.0823         1.0000       0.0405         0.6924
## 2     Bands 0.7859 0.0810         0.0980       0.0603         0.7235
## 3 GlidePath 1.1639 0.0860         0.3333       0.0943         1.0687
```

```
cat("Política recomendada:", best_policy, "
")
```

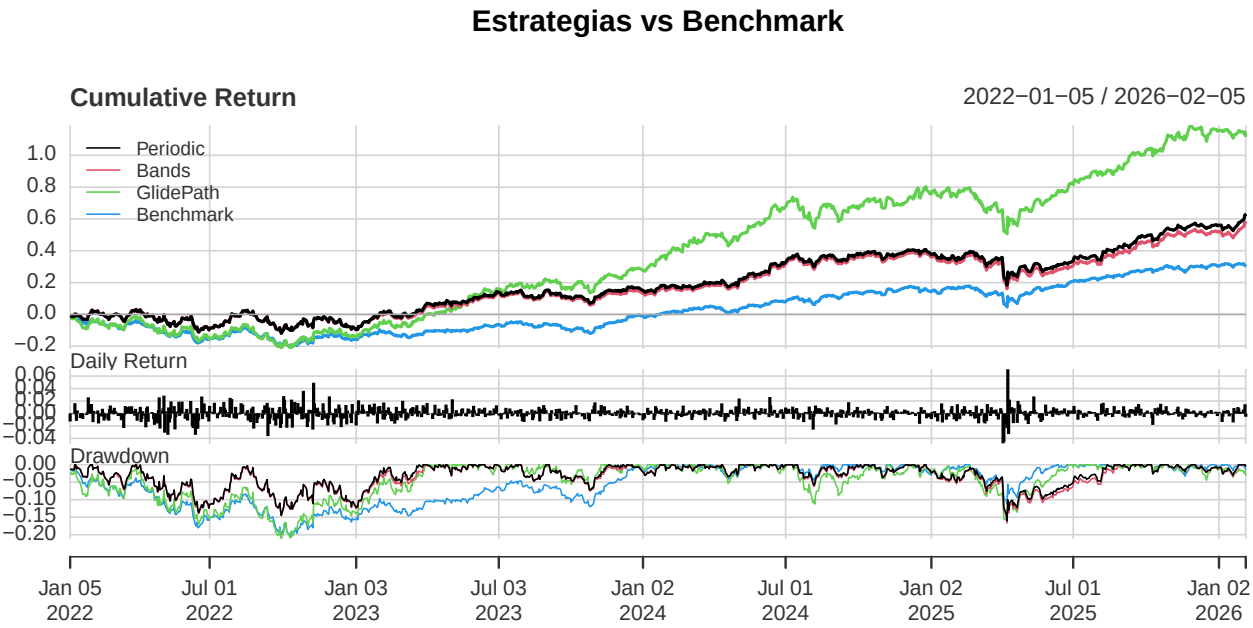
```
## Política recomendada: GlidePath
```

4.15 13) Gráfico comparativo

```
R_comp <- na.omit(merge(r_periodic, r_bands, bench_use))
colnames(R_comp) <- c("Periodic","Bands","Benchmark")

if (!is.null(r_glide)) {
  R_comp <- na.omit(merge(r_periodic, r_bands, r_glide, bench_use))
  colnames(R_comp) <- c("Periodic","Bands","GlidePath","Benchmark")
}

charts.PerformanceSummary(R_comp, main="Estrategias vs Benchmark")
```



---

5 Consideraciones prácticas y límites

- Costos reales: aquí usamos el turnover como señal de cuánto se mueve el portafolio. En un caso real conviene sumar comisiones y spreads.
- Cambios de mercado: una estrategia puede funcionar mejor en algunos periodos y peor en otros. Por eso conviene mirar el gráfico y el drawdown.
- Calidad de datos: algunas métricas pueden faltar. El proceso está preparado para seguir funcionando aun con datos incompletos.
- Benchmark: ayuda a comparar la estrategia con una referencia simple.

---

6 Cómo reproducir

1. Ejecuta el documento de arriba hacia abajo.
2. Ajusta `cfg_sel`, `cfg_port` y `cfg_reb` según lo que quieras probar.
3. Salidas más importantes:
   - `sel$tickers_selected` (activos seleccionados),
   - `port$weights_target` (pesos objetivo),
   - `tbl` y `best_policy` (tabla y recomendación).