Package: Athlytics (via r-universe)

May 29, 2026

Title Sports Physiology Analysis from Local 'Strava' Data

Version 1.0.5

Description Tools for reproducible, offline analysis of endurance-training data exported from 'Strava'. Provides data import, quality-control, cohort-reference, and visualization helpers for sports-science indicators including acute:chronic workload ratio, aerobic efficiency, cardiovascular decoupling, exposure, and personal-best profiles.

License MIT + file LICENSE

URL https://docs.ropensci.org/Athlytics/, https://github.com/ropensci/Athlytics

BugReports https://github.com/ropensci/Athlytics/issues

Encoding UTF-8

LazyData true

Depends R (>= 4.1.0)

Imports dplyr (>= 1.0.0), ggplot2, lubridate, rlang (>= 0.4.0), tidyr, zoo, tools, utils

Suggests FITfileR, knitr, pkgdown, R.utils, readr, rmarkdown, testthat (>= 3.1.5), vdiffr, xml2

Additional_repositories https://grimbough.r-universe.dev

VignetteBuilder knitr

Roxygen list(markdown = TRUE)

Config/testthat/edition 3

NeedsCompilation no

Config/roxygen2/version 8.0.0

Config/pak/sysreqs libicu-dev

Repository https://ropensci.r-universe.dev

Date/Publication 2026-05-29 16:18:55 UTC

RemoteUrl https://github.com/ropensci/Athlytics

RemoteRef main

RemoteSha ccf55c4d6abee1c05f4ebb89655ac7f5c6029804

Contents

`add_reference_bands` *Add Cohort Reference Bands to Existing Plot*

Description

Adds percentile reference bands from a cohort to an individual's metric plot.

Usage

```
add_reference_bands(
  p,
  reference_data,
  bands = c("p25_p75", "p05_p95", "p50"),
  alpha = 0.15,
  colors = list(p25_p75 = "#4DBBD5", p05_p95 = "#E64B35", p50 = "#3C5488")
)
```

Arguments

p	A ggplot object (typically from plot_acwr or similar).
reference_data	A data frame from `calculate_cohort_reference()`.
bands	Character vector specifying which bands to plot. Options: "p25_p75" (inner quartiles), "p05_p95" (outer 5-95 range), "p50" (median). Default c("p25_p75", "p05_p95", "p50").
alpha	Transparency for reference bands (0-1). Default 0.15.
colors	Named list of colors for bands. Default uses Nature-inspired palette colors.

Value

A ggplot object with added reference bands.

Examples

```
# Example: add reference bands to an ACWR plot
set.seed(42)
n <- 60
dates <- seq(as.Date("2024-01-01"), by = "day", length.out = n)
base_acwr <- 1.0 + cumsum(rnorm(n, 0, 0.03))
individual <- data.frame(
  date = dates, atl = runif(n, 30, 60), ctl = runif(n, 35, 55),
  acwr = base_acwr, acwr_smooth = base_acwr
)
cohort <- dplyr::bind_rows(
  dplyr::mutate(individual, athlete_id = "A1"),
  dplyr::mutate(individual, athlete_id = "A2",
    acwr_smooth = acwr_smooth * runif(n, 0.9, 1.1))
)
ref <- suppressWarnings(
  calculate_cohort_reference(cohort, metric = "acwr_smooth", min_athletes = 2)
)
p <- suppressMessages(plot_acwr(individual, highlight_zones = FALSE))
p_ref <- add_reference_bands(p, reference_data = ref)
print(p_ref)
```

athlytics_palette_nature

Nature-Inspired Color Palette

Description

Professional, colorblind-friendly palette based on Nature journal's visualization guidelines. Suitable for multi-series plots.

Usage

```
athlytics_palette_nature()
```

Value

A character vector of 9 hex color codes

Examples

```
# View the palette colors
athlytics_palette_nature()

# Display as color swatches
barplot(rep(1, 9), col = athlytics_palette_nature(), border = NA)
```

athlytics_palette_vibrant

Vibrant High-Contrast Palette

Description

High-saturation palette optimized for presentations and posters. Maximum visual impact while maintaining colorblind accessibility.

Usage

```
athlytics_palette_vibrant()
```

Value

A character vector of 8 hex color codes

Examples

```
# View the palette colors
athlytics_palette_vibrant()
```

`calculate_acwr` *Calculate Acute:Chronic Workload Ratio (ACWR)*

Description

This function calculates daily training load and derives acute (short-term) and chronic (long-term) load averages, then computes their ratio (ACWR). The ACWR helps identify periods when recent load is elevated relative to chronic load.

Key Concepts:

- **Acute Load (ATL)**: Rolling average of recent training (default: 7 days)
- **Chronic Load (CTL)**: Rolling average of longer-term training (default: 28 days)
- **ACWR**: Ratio of ATL to CTL (ATL / CTL)
- **Reference Band**: ACWR between 0.8-1.3 (commonly used load-balance band)
- **Elevated ACWR Band**: ACWR between 1.3-1.5
- **High ACWR Band**: ACWR > 1.5 (load-spike threshold)

Usage

```
calculate_acwr(
  activities_data,
  activity_type,
  load_metric = "duration_mins",
  acute_period = 7,
  chronic_period = 28,
  start_date = NULL,
  end_date = Sys.Date(),
  user_ftp = NULL,
  user_max_hr = NULL,
  user_resting_hr = NULL,
  smoothing_period = 7,
  missing_load = c("zero", "na"),
  verbose = FALSE
)
```

Arguments

`activities_data`
: A data frame of activities from `load_local_activities()`. Must contain columns: `date`, `distance`, `moving_time`, `elapsed_time`, `average_heartrate`, `average_watts`, `type`, `elevation_gain`.

`activity_type`
: **Required** character vector. Filter activities by type (e.g., `"Run"`, `"Ride"`). **Must specify** to avoid mixing incompatible load metrics.

`load_metric`
: Character string specifying the load calculation method:
  - `"duration_mins"`: Training duration in minutes (default)

- "distance_km": Distance in kilometers
- "elapsed_time_mins": Total elapsed time including stops
- "tss": Training Stress Score approximation using NP/FTP ratio (requires user_ftp)
- "hrss": Heart Rate Stress Score approximation using simplified TRIMP (requires user_max_hr and user_resting_hr)
- "elevation_gain_m": Elevation gain in meters

acute_period Integer. Number of days for the acute load window (default: 7). Represents recent training stimulus. Common values: 3-7 days.

chronic_period Integer. Number of days for the chronic load window (default: 28). Represents fitness/adaptation level. Must be greater than acute_period. Common values: 21-42 days.

start_date Optional. Analysis start date (YYYY-MM-DD string, Date, or POSIXct). Defaults to one year before end_date. Earlier data is used for calculating initial chronic load.

end_date Optional. Analysis end date (YYYY-MM-DD string, Date, or POSIXct). Defaults to current date (Sys.Date()).

user_ftp Numeric. Your Functional Threshold Power in watts. Required only when load_metric = "tss". Used to normalize power-based training stress.

user_max_hr Numeric. Your maximum heart rate in bpm. Required only when load_metric = "hrss". Used for heart rate reserve calculations.

user_resting_hr
Numeric. Your resting heart rate in bpm. Required only when load_metric = "hrss". Used for heart rate reserve calculations.

smoothing_period
Integer. Number of days for smoothing the ACWR using a rolling mean (default: 7). Reduces day-to-day noise for clearer trend visualization.

missing_load How to treat training days on which the chosen load_metric could not be computed (e.g. an HRSS call on an activity with no HR samples, or a TSS call without FTP). Options:

- "zero" (default): coalesce to 0, matching the historical behaviour. This keeps backwards compatibility but conflates missing-data training days with genuine rest days.
- "na": keep as NA. Genuine rest days (no activity on record) are still imputed to 0; only days that *had* an activity whose load was not computable remain NA. Rolling means propagate the NA, clearly surfacing the data gap.

verbose Logical. If TRUE, prints progress messages. Default FALSE.

Details

Computes the Acute:Chronic Workload Ratio (ACWR) from local Strava activity data using rolling average methods. ACWR is a key metric for monitoring training load balance in athletes (Gabbett, 2016; Hulin et al., 2016).

Algorithm:

1. **Daily Aggregation**: Sum all activities by date to compute daily load
2. **Complete Time Series**: Fill missing days with zero load (critical for ACWR accuracy)
3. **Acute Load (ATL)**: Rolling mean over `acute_period` days (default: 7)
4. **Chronic Load (CTL)**: Rolling mean over `chronic_period` days (default: 28)
5. **ACWR Calculation**: ATL / CTL (set to NA when CTL < 0.01 to avoid division by zero)
6. **Smoothing**: Optional rolling mean over `smoothing_period` days for visualization

Data Requirements: The function automatically fetches additional historical data (chronic_period days before start_date) to ensure accurate chronic load calculations at the analysis start point. Ensure your Strava export contains sufficient historical activities.

Load Metric Implementations:

- `"tss"`: Uses normalized power (NP) and FTP to approximate Training Stress Score (TSS). Formula: `(duration_sec × NP × IF) / (FTP × 3600) × 100`, where `IF = NP/FTP` (equivalently: `hours × IF^2 × 100`).
- `"hrss"`: HR-based load using heart rate reserve (simplified TRIMP; **not** TrainingPeaks hrTSS). Formula: `duration_sec * (HR - resting_HR) / (max_HR - resting_HR)`.

Descriptive ACWR Bands:

- ACWR < 0.8: Recent load below chronic load
- ACWR 0.8-1.3: Commonly used training-load reference band
- ACWR 1.3-1.5: Elevated recent/chronic load ratio
- ACWR > 1.5: High ACWR ratio / load-spike band

Multi-Athlete Studies: For cohort analyses, add an `athlete_id` column before calculation and use `group_by(athlete_id)` with `group_modify()`. See examples below and vignettes for details.

Value

A tibble with the following columns:

date Date (Date class)

atl Acute Training Load - rolling average over `acute_period` days (numeric)

ctl Chronic Training Load - rolling average over `chronic_period` days (numeric)

acwr Raw ACWR value (atl / ctl) (numeric)

acwr_smooth Smoothed ACWR using `smoothing_period` rolling mean (numeric)

Scientific Considerations

Important: The predictive value of ACWR for injury risk has been debated in recent literature. Some researchers argue that ACWR may have limited utility for predicting injuries (Impellizzeri et al., 2020), and a subsequent analysis has called for dismissing the ACWR framework entirely (Impellizzeri et al., 2021). Users should interpret ACWR zones as descriptive heuristics rather than validated injury predictors.

Impellizzeri, F. M., Tenan, M. S., Kempton, T., Novak, A., & Coutts, A. J. (2020). Acute:chronic workload ratio: conceptual issues and fundamental pitfalls. *International Journal of Sports Physiology and Performance*, 15(6), 907-913. doi:10.1123/ijspp.20190864

Impellizzeri, F. M., Woodcock, S., Coutts, A. J., Fanchini, M., McCall, A., & Vigotsky, A. D. (2021). What role do chronic workloads play in the acute to chronic workload ratio? Time to dismiss ACWR and its underlying theory. *Sports Medicine*, 51(3), 581-592. doi:10.1007/s40279-020013786

References

Gabbett, T. J. (2016). The training-injury prevention paradox: should athletes be training smarter and harder? *British Journal of Sports Medicine*, 50(5), 273-280. doi:10.1136/bjsports2015095788

Hulin, B. T., Gabbett, T. J., Lawson, D. W., Caputi, P., & Sampson, J. A. (2016). The acute:chronic workload ratio predicts injury: high chronic workload may decrease injury risk in elite rugby league players. *British Journal of Sports Medicine*, 50(4), 231-236. doi:10.1136/bjsports2015094817

See Also

`plot_acwr()` for visualization, `calculate_acwr_ewma()` for EWMA-based ACWR, `load_local_activities()` for data loading, `calculate_cohort_reference()` for multi-athlete comparisons

Examples

```
# Example using simulated data (Note: sample data is pre-calculated, shown for demonstration)
data(sample_acwr)
print(head(sample_acwr))

# Runnable example with dummy data:
end <- Sys.Date()
dates <- seq(end - 59, end, by = "day")
dummy_activities <- data.frame(
  date = dates,
  type = "Run",
  moving_time = rep(3600, length(dates)), # 1 hour
  distance = rep(10000, length(dates)), # 10 km
  average_heartrate = rep(140, length(dates)),
  suffer_score = rep(50, length(dates)),
  tss = rep(50, length(dates)),
  stringsAsFactors = FALSE
)

# Calculate ACWR
result <- calculate_acwr(
  activities_data = dummy_activities,
  activity_type = "Run",
  load_metric = "distance_km",
  acute_period = 7,
  chronic_period = 28,
  end_date = end
)
print(head(result))
```

```
## Not run:
# Example using local Strava export data
# Step 1: Download your Strava data export
# Go to Strava.com > Settings > My Account > Download or Delete Your Account
# You'll receive a ZIP file via email (e.g., export_12345678.zip)

# Step 2: Load activities directly from ZIP (no extraction needed!)
activities <- load_local_activities("export_12345678.zip")

# Or from extracted CSV
activities <- load_local_activities("strava_export_data/activities.csv")

# Step 3: Calculate ACWR for Runs (using distance)
run_acwr <- calculate_acwr(
  activities_data = activities,
  activity_type = "Run",
  load_metric = "distance_km"
)
print(tail(run_acwr))

# Calculate ACWR for Rides (using TSS, requires FTP)
ride_acwr_tss <- calculate_acwr(
  activities_data = activities,
  activity_type = "Ride",
  load_metric = "tss",
  user_ftp = 280
)
print(tail(ride_acwr_tss))

# Plot the results
plot_acwr(run_acwr, highlight_zones = TRUE)

# Multi-athlete cohort analysis

# Load data for multiple athletes and add athlete_id
athlete1 <- load_local_activities("athlete1_export.zip") %>%
  dplyr::mutate(athlete_id = "athlete1")

athlete2 <- load_local_activities("athlete2_export.zip") %>%
  dplyr::mutate(athlete_id = "athlete2")

# Combine all athletes
cohort_data <- dplyr::bind_rows(athlete1, athlete2)

# Calculate ACWR for each athlete using group_modify()
cohort_acwr <- cohort_data %>%
  dplyr::group_by(athlete_id) %>%
  dplyr::group_modify(~ calculate_acwr(.x,
    activity_type = "Run",
    load_metric = "duration_mins"
  )) %>%
  dplyr::ungroup()
```

```
# View results
print(cohort_acwr)

## End(Not run)
```

`calculate_acwr_ewma` *Calculate ACWR using EWMA Method with Confidence Bands*

Description

Calculates the Acute:Chronic Workload Ratio (ACWR) using Exponentially Weighted Moving Average (EWMA) with optional bootstrap confidence bands.

Usage

```
calculate_acwr_ewma(
  activities_data,
  activity_type,
  load_metric = "duration_mins",
  method = c("ra", "ewma"),
  acute_period = 7,
  chronic_period = 28,
  half_life_acute = 3.5,
  half_life_chronic = 14,
  start_date = NULL,
  end_date = Sys.Date(),
  user_ftp = NULL,
  user_max_hr = NULL,
  user_resting_hr = NULL,
  smoothing_period = 7,
  missing_load = c("zero", "na"),
  ci = FALSE,
  B = 200,
  block_len = 7,
  conf_level = 0.95
)
```

Arguments

`activities_data`	A data frame of activities from `load_local_activities()`.
`activity_type`	Required. Filter activities by type, such as `"Run"` or `"Ride"`. Explicit filtering avoids mixing incompatible sport contexts.
`load_metric`	Method for calculating daily load. Default "duration_mins".
`method`	ACWR calculation method: "ra" (rolling average) or "ewma". Default "ra".

acute_period	Days for acute window (for RA method). Default 7.
chronic_period	Days for chronic window (for RA method). Default 28.
half_life_acute	Half-life for acute EWMA in days. Default 3.5.
half_life_chronic	Half-life for chronic EWMA in days. Default 14.
start_date	Optional. Analysis start date. Defaults to one year ago.
end_date	Optional. Analysis end date. Defaults to today.
user_ftp	Required if load_metric = "tss".
user_max_hr	Required if load_metric = "hrss".
user_resting_hr	Required if load_metric = "hrss".
smoothing_period	Days for smoothing ACWR. Default 7.
missing_load	How to treat training days on which the chosen load_metric could not be computed ("zero" for historical behaviour or "na" to keep missing-data training days visibly NA). See calculate_acwr() for details.
ci	Logical. Whether to calculate confidence bands (EWMA only). Default FALSE.
B	Number of bootstrap iterations (if ci = TRUE). Default 200.
block_len	Block length for moving-block bootstrap (days). Default 7.
conf_level	Confidence level (0-1). Default 0.95 (95\% CI).

Details

This function extends the basic ACWR calculation with two methods:

- **RA (Rolling Average)**: Traditional rolling mean approach (default).
- **EWMA (Exponentially Weighted Moving Average)**: Uses exponential decay with configurable half-lives. More responsive to recent changes.

EWMA Formula: The smoothing parameter alpha is calculated from half-life: `alpha = 1 - exp(-ln(2) / half_life)`. The EWMA update is: `E_t = alpha * L_t + (1-alpha) * E_{t-1}` where L_t is daily load and E_t is the exponentially weighted average.

Confidence Bands: When `ci = TRUE` and `method = "ewma"`, uses **moving-block bootstrap** to estimate uncertainty. The daily load sequence is resampled in overlapping weekly blocks (preserving within-week correlation), ACWR is recalculated, and percentiles form the confidence bands. This accounts for temporal correlation in training load patterns.

Value

A data frame with columns: `date`, `atl`, `ctl`, `acwr`, `acwr_smooth`, and if `ci = TRUE` and `method = "ewma"`: `acwr_lower`, `acwr_upper`.

Examples

```
# Example using pre-calculated sample data
data("sample_acwr", package = "Athlytics")
head(sample_acwr)

## Not run:
# Full workflow with real data - Load local activities
activities <- load_local_activities("export_12345678.zip")

# Calculate ACWR using Rolling Average (RA)
acwr_ra <- calculate_acwr_ewma(activities, activity_type = "Run", method = "ra")

# Calculate ACWR using EWMA with confidence bands
acwr_ewma <- calculate_acwr_ewma(
  activities,
  activity_type = "Run",
  method = "ewma",
  half_life_acute = 3.5,
  half_life_chronic = 14,
  ci = TRUE,
  B = 200
)

# Compare both methods
head(acwr_ewma)

## End(Not run)
```

---

calculate_cohort_reference

Calculate Cohort Reference Percentiles

Description

Calculates reference percentiles for a metric across a cohort of athletes, stratified by specified grouping variables (e.g., sport, sex, age band).

Usage

```
calculate_cohort_reference(
  data,
  metric = "acwr_smooth",
  by = c("sport"),
  probs = c(0.05, 0.25, 0.5, 0.75, 0.95),
  min_athletes = 5,
  allow_unknown_athlete = FALSE,
  date_col = "date"
)
```

```
cohort_reference(
  data,
  metric = "acwr_smooth",
  by = c("sport"),
  probs = c(0.05, 0.25, 0.5, 0.75, 0.95),
  min_athletes = 5,
  allow_unknown_athlete = FALSE,
  date_col = "date"
)
```

Arguments

`data`	A data frame containing metric values for multiple athletes. Must include columns: `date`, `athlete_id`, and the metric column.
`metric`	Name of the metric column to calculate percentiles for (e.g., "acwr", "acwr_smooth", "ef", "decoupling"). Default "acwr_smooth".
`by`	Character vector of grouping variables. Options: "sport", "sex", "age_band", "athlete_id". Default c("sport").
`probs`	Numeric vector of probabilities for percentiles (0-1). Default c(0.05, 0.25, 0.50, 0.75, 0.95) for 5th, 25th, 50th, 75th, 95th percentiles.
`min_athletes`	Minimum number of athletes required per group to calculate valid percentiles. Default 5.
`allow_unknown_athlete`	Logical. Controls the behaviour when `data` has no `athlete_id` column. `FALSE` (default) raises an error, because a cohort reference is by definition a distribution *across athletes*, and silently pooling the rows under a single synthetic `"unknown"` athlete produces a band that looks multi-athlete but is not. Set to `TRUE` only if you really are computing a pseudo-cohort from a single athlete and understand that the resulting percentiles are row-level quantiles within that athlete, not a peer-group distribution. In that mode a warning is emitted and an `athlete_id = "unknown"` column is added, matching the pre-fix behaviour.
`date_col`	Name of the date column. Default "date".

Details

This function creates cohort-level reference bands for comparing individual athlete metrics to their peers. Common use cases:

- Compare an athlete's ACWR trend to team averages
- Identify outliers (athletes outside P5-P95 range)
- Track team-wide trends over time

Important: Percentile bands represent **population variability**, not statistical confidence intervals for individual values.

Value

A long-format data frame with columns:

date Date

... Grouping variables (as specified in by)

percentile Percentile label (e.g., "p05", "p25", "p50", "p75", "p95")

value Metric value at that percentile

n_athletes Number of athletes contributing to this percentile

Examples

```
# Example using sample data to create a mock cohort
data("sample_acwr", package = "Athlytics")

# Simulate a cohort by duplicating with different athlete IDs
cohort_mock <- dplyr::bind_rows(
  dplyr::mutate(sample_acwr, athlete_id = "A1", sport = "Run"),
  dplyr::mutate(sample_acwr,
    athlete_id = "A2", sport = "Run",
    acwr_smooth = acwr_smooth * runif(nrow(sample_acwr), 0.9, 1.1)
  ),
  dplyr::mutate(sample_acwr,
    athlete_id = "A3", sport = "Run",
    acwr_smooth = acwr_smooth * runif(nrow(sample_acwr), 0.85, 1.15)
  )
)

# Calculate reference percentiles (min_athletes = 2 for demo)
reference <- calculate_cohort_reference(cohort_mock,
  metric = "acwr_smooth",
  by = "sport", min_athletes = 2
)
head(reference)

## Not run:
# Full workflow with real data - Load activities for multiple athletes
athlete1 <- load_local_activities("athlete1_export.zip") %>%
  mutate(athlete_id = "athlete1")
athlete2 <- load_local_activities("athlete2_export.zip") %>%
  mutate(athlete_id = "athlete2")
athlete3 <- load_local_activities("athlete3_export.zip") %>%
  mutate(athlete_id = "athlete3")

# Combine data
cohort_data <- bind_rows(athlete1, athlete2, athlete3)

# Calculate ACWR for each athlete
cohort_acwr <- cohort_data %>%
  group_by(athlete_id) %>%
  group_modify(~ calculate_acwr_ewma(.x, activity_type = "Run")) %>%
  ungroup() %>%
```

```
  mutate(sport = "Run")

# Calculate reference percentiles
reference <- calculate_cohort_reference(
  cohort_acwr,
  metric = "acwr_smooth",
  by = c("sport"),
  probs = c(0.05, 0.25, 0.5, 0.75, 0.95)
)

# Plot individual against cohort
plot_with_reference(
  individual = cohort_acwr %>% filter(athlete_id == "athlete1"),
  reference = reference
)

## End(Not run)
```

`calculate_decoupling` *Calculate Aerobic Decoupling*

Description

Calculates aerobic decoupling for Strava activities from local export data.

Usage

```
calculate_decoupling(
  activities_data = NULL,
  export_dir = "strava_export_data",
  activity_type = c("Run", "Ride"),
  decouple_metric = c("speed_hr", "power_hr"),
  start_date = NULL,
  end_date = Sys.Date(),
  min_duration_mins = 40,
  min_steady_minutes = 40,
  steady_cv_threshold = 0.08,
  min_hr_coverage = 0.9,
  quality_control = c("filter", "flag", "off"),
  stream_df = NULL,
  return_diagnostics = FALSE,
  verbose = FALSE
)
```

Arguments

`activities_data`
: A data frame from `load_local_activities()`. Required unless `stream_df` is provided.

`export_dir` Base directory of Strava export containing the activities folder. Default is "strava_export_data".

`activity_type` Type(s) of activities to analyze (e.g., "Run", "Ride").

`decouple_metric`
Basis for calculation: "speed_hr" or "power_hr". Note: `"pace_hr"` is accepted as a deprecated alias for `"speed_hr"`.

`start_date` Optional. Analysis start date (YYYY-MM-DD string or Date). Defaults to one year ago.

`end_date` Optional. Analysis end date (YYYY-MM-DD string or Date). Defaults to today.

`min_duration_mins`
Minimum activity duration (minutes) to include. Default 40.

`min_steady_minutes`
Minimum duration (minutes) for steady-state segment (default: 40). Activities shorter than this are automatically rejected for decoupling calculation.

`steady_cv_threshold`
Coefficient of variation threshold for steady-state (default: 0.08 = 8%). Activities with higher variability are rejected as non-steady-state.

`min_hr_coverage`
Minimum HR data coverage threshold (default: 0.9 = 90%). Activities with lower HR coverage are rejected as insufficient data quality.

`quality_control`
Quality control mode: "off" (no filtering), "flag" (mark issues), or "filter" (exclude flagged data). Default "filter" for scientific rigor.

`stream_df` Optional. A pre-fetched data frame for a *single* activity's stream. If provided, decoupling is calculated directly from this stream, and the activities-level arguments (`activities_data`, `export_dir`, `activity_type`, `start_date`, `end_date`, `min_duration_mins`) are not used. The stream-level quality-control and steady-state parameters — `quality_control`, `min_steady_minutes`, `steady_cv_threshold`, `min_hr_coverage` — are honoured and forwarded into the internal calculator, so callers can tune rejection behaviour per call. Must include columns: `time`, `heartrate`, and either `velocity_smooth`/`distance` (for speed_hr) or `watts` (for power_hr).

`return_diagnostics`
Logical. Only consulted when `stream_df` is supplied. When `FALSE` (default) the function returns a bare numeric decoupling value for backward compatibility with early releases. When `TRUE` it returns a one-row data frame with the same columns as the activities-level path (`decoupling`, `status`, `quality_score`, `hr_coverage`, `steady_duration_minutes`, `sampling_interval_seconds`) so callers can distinguish rejection reasons from a genuine `NA` decoupling value.

`verbose` Logical. If TRUE, prints progress messages. Default FALSE.

Details

Calculates aerobic decoupling (HR drift relative to speed/power) using detailed activity stream data from local FIT/TCX/GPX files.

Provides data for `plot_decoupling`. Compares output/HR efficiency between first and second halves of activities. Positive values indicate HR drift (cardiovascular drift).

Best practice: Use `load_local_activities()` to load data, then pass to this function.

The function parses FIT/TCX/GPX files from your Strava export to extract detailed stream data (time, heartrate, distance/power). Activities are split into two halves, and the efficiency factor (output/HR) is compared between halves.

Steady-State Detection Method:

Before computing decoupling, the function applies a rolling coefficient of variation (CV) filter to identify steady-state segments:

1. A sliding window (default 300 s) computes the rolling mean and standard deviation of the output metric (velocity or power).
2. The CV (= rolling SD / rolling mean) is calculated at each time point.
3. Time points with CV < `steady_cv_threshold` (default 8 %) are classified as steady-state.
4. At least `min_steady_minutes` of contiguous steady-state data must be present; otherwise the activity is marked `"insufficient_steady_duration"`.
5. Decoupling is then calculated by comparing the EF (output / HR) of the first half vs. the second half of the steady-state segment.

This ensures that measured decoupling reflects true cardiovascular drift rather than pacing variability or interval efforts (Coyle & González-Alonso, 2001). The rolling CV approach is a standard signal-processing technique for detecting stationarity in physiological time series.

Value

Returns a data frame with columns:

date Activity date (Date class)

decoupling Decoupling percentage (\%). Positive = HR drift, negative = improved efficiency

status Character status code describing the outcome of the calculation. See **Status vocabulary** below.

quality_score Numeric in [0, 1]. Fraction of stream samples that passed quality-control range checks. `NA` if the activity was rejected before the QC stage.

hr_coverage Numeric in [0, 1]. Time-weighted fraction of the stream that carried a valid heart-rate sample.

steady_duration_minutes Wall-clock duration of the contiguous steady-state block the decoupling was derived from. `NA` when no qualifying block existed.

sampling_interval_seconds Observed median sampling interval of the stream (seconds). Useful for auditing whether the rolling-CV window was well-calibrated.

When `stream_df` is provided the default return is a single numeric decoupling value (backward-compatible with early releases). Pass `return_diagnostics = TRUE` to get the full one-row diagnostics frame instead.

Status vocabulary

- `"ok"`: Decoupling computed from a contiguous steady-state block.
- `"missing_hr_data"`: Stream lacked a `heartrate` / `heart_rate` column.
- `"missing_time_data"`: Stream lacked a `time` column.
- `"missing_velocity_data"` / `"missing_power_data"`: Stream lacked the column required by the chosen `decouple_metric`.
- `"insufficient_hr_data"`: Time-weighted HR coverage < `min_hr_coverage`.
- `"insufficient_data_points"`: Fewer than 100 non-NA stream samples.
- `"insufficient_valid_data"`: Fewer than 100 samples survived basic positivity filtering (`velocity > 0` or `watts > 0`).
- `"insufficient_data_after_quality_filter"`: `quality_control = "filter"` removed enough out-of-range samples to drop the stream below 100 rows.
- `"insufficient_steady_duration"`: No contiguous steady-state block met the `min_steady_minutes` threshold.
- `"non_steady"`: No rolling-CV windows cleared `steady_cv_threshold`, or time-midpoint split produced an empty half.
- `"calculation_failed"`: Median first-half EF was non-positive or not finite, so decoupling could not be expressed as a percentage.
- `"calculation_error"`: An exception was raised during per-activity processing (caught by the outer loop).

References

Coyle, E. F., & González-Alonso, J. (2001). Cardiovascular drift during prolonged exercise: New perspectives. *Exercise and Sport Sciences Reviews*, 29(2), 88-92. doi:10.1097/00003677200104000-00009

Examples

```
# Example using simulated data
data(sample_decoupling)
print(head(sample_decoupling))

# Runnable example with dummy stream data (single activity analysis):
dummy_stream <- data.frame(
  time = 1:3600, # 1 hour
  heartrate = rep(140, 3600),
  velocity_smooth = rep(3, 3600), # 3 m/s
  watts = rep(200, 3600),
  distance = cumsum(rep(3, 3600)),
  stringsAsFactors = FALSE
)

# Calculate decoupling for this specific activity stream
result <- calculate_decoupling(
  stream_df = dummy_stream,
```

```
  decouple_metric = "speed_hr"
)
print(result)

## Not run:
# Load local activities
activities <- load_local_activities("strava_export_data/activities.csv")

# Calculate Speed/HR decoupling for recent runs
run_decoupling <- calculate_decoupling(
  activities_data = activities,
  export_dir = "strava_export_data",
  activity_type = "Run",
  decouple_metric = "speed_hr",
  start_date = "2024-01-01"
)
print(tail(run_decoupling))

# Calculate for a single activity stream
# stream_data <- parse_activity_file("strava_export_data/activities/12345.fit")
# single_decoupling <- calculate_decoupling(stream_df = stream_data, decouple_metric = "speed_hr")

## End(Not run)
```

`calculate_ef` *Calculate Efficiency Factor (EF)*

Description

Efficiency Factor measures how much work you perform per unit of cardiovascular effort. Higher EF indicates better aerobic fitness - you're able to maintain faster pace or higher power at the same heart rate. Tracking EF over time helps monitor aerobic base development and training effectiveness.

EF Metrics:

- **speed_hr** (for running): Speed (m/s) / Average HR
  - Higher values = faster speed at same HR = better fitness
- **power_hr** (for cycling): Average Power (watts) / Average HR
  - Higher values = more power at same HR = better fitness

What Improves EF?

- Aerobic base building (Zone 2 training)
- Improved running/cycling economy
- Enhanced cardiovascular efficiency
- Increased mitochondrial density

Usage

```
calculate_ef(
  activities_data,
  activity_type = c("Run", "Ride"),
  ef_metric = c("speed_hr", "gap_hr", "power_hr"),
  start_date = NULL,
  end_date = Sys.Date(),
  min_duration_mins = 20,
  min_steady_minutes = 20,
  steady_cv_threshold = 0.08,
  min_hr_coverage = 0.9,
  quality_control = c("filter", "flag", "off"),
  export_dir = NULL
)
```

Arguments

`activities_data`
: A data frame of activities from `load_local_activities()`. Must contain columns: `date`, `type`, `moving_time`, `distance`, `average_heartrate`, and `average_watts` (for power_hr metric).

`activity_type`
: Character vector or single string specifying activity type(s) to analyze. Common values: `"Run"`, `"Ride"`, or `c("Run", "Ride")`. Default: `c("Run", "Ride")`.

`ef_metric`
: Character string specifying the efficiency metric:

  - `"speed_hr"`: Speed-based efficiency (for running). Formula: speed (m/s) / avg_HR. Units: m/s/bpm (higher = better fitness)
  - `"gap_hr"`: Grade Adjusted Speed efficiency (for running on hilly terrain). Formula: GAP speed (m/s) / avg_HR. Accounts for elevation changes. Units: m/s/bpm
  - `"power_hr"`: Power-based efficiency (for cycling). Formula: avg_watts / avg_HR. Units: W/bpm (higher = better fitness) Default: `c("speed_hr", "power_hr")` (uses first matching metric for activity type). Note: `"pace_hr"` is accepted as a deprecated alias for `"speed_hr"` for backward compatibility.

`start_date`
: Optional. Analysis start date (YYYY-MM-DD string, Date, or POSIXct). Defaults to one year before `end_date`.

`end_date`
: Optional. Analysis end date (YYYY-MM-DD string, Date, or POSIXct). Defaults to current date (Sys.Date()).

`min_duration_mins`
: Numeric. Minimum activity duration in minutes to include in analysis (default: 20). Filters out very short activities that may not represent steady-state aerobic efforts.

`min_steady_minutes`
: Numeric. Minimum duration (minutes) for steady-state segment (default: 20). Activities shorter than this are automatically rejected for EF calculation.

`steady_cv_threshold`
: Numeric. Coefficient of variation threshold for steady-state (default: 0.08 = 8%). Activities with higher variability are rejected as non-steady-state.

`min_hr_coverage`
: Numeric. Minimum HR data coverage threshold (default: 0.9 = 90%). Activities with lower HR coverage are rejected as insufficient data quality.

`quality_control`
: Character. Quality control mode: "off" (no filtering), "flag" (mark issues), or "filter" (exclude flagged data). Default "filter" for scientific rigor.

`export_dir`
: Optional. Path to a Strava export directory or ZIP file containing activity files. When provided, enables stream data analysis for more accurate steady-state detection; when omitted, EF falls back to activity-summary averages.

Details

Computes Efficiency Factor (EF) for endurance activities, quantifying the relationship between performance output (speed or power) and heart rate. EF is a key indicator of aerobic fitness and training adaptation (Allen et al., 2019).

Algorithm:

1. Filter activities by type, date range, and minimum duration
2. For each activity, calculate:
   - speed_hr: (distance / moving_time) / average_heartrate
   - power_hr: average_watts / average_heartrate
3. Return one EF value per activity

Steady-State Detection Method:

When stream data is available (via `export_dir`), the function applies a rolling coefficient of variation (CV) approach to identify steady-state periods and then enforces *contiguous* block duration so scattered "steady" islands are not averaged together:

1. **Sampling-interval awareness**: The observed median sampling interval is estimated via `diff(time)` so the rolling window targets a fixed number of seconds regardless of recording frequency (1 Hz, 0.5 Hz smart-recording, multi-Hz). This removes the implicit 1 Hz assumption from earlier versions.
2. **Rolling window**: A sliding window targeting 300 seconds (capped by `nrow / 4`, floor at 60 seconds' worth of samples) computes rolling mean and SD of the output metric.
3. **CV calculation**: CV = rolling SD / rolling mean at each time point.
4. **Continuous blocks**: All time points with CV < `steady_cv_threshold` are marked steady, then grouped into contiguous runs via `rle()`. A block qualifies only if its wall-clock span is >= `min_steady_minutes` AND it has >= 100 samples. If no block qualifies, status is `"insufficient_steady_duration"` (activities with no steady CV window at all are marked `"non_steady"`).
5. **EF computation**: The longest qualifying block is selected; EF is the median output/HR ratio across that block. `steady_duration_minutes`, `n_steady_blocks` and `sampling_interval_seconds` are returned alongside the EF value for auditability.

This mirrors the block-based steady-state logic used in `calculate_decoupling()` and follows the principle that valid EF comparisons require quasi-constant output intensity, as outlined by Coyle & González-Alonso (2001), who demonstrated that cardiovascular drift is meaningful only under steady-state exercise conditions. The rolling CV method is a common signal-processing technique for detecting stationarity in physiological time series.

Data Quality Considerations:

- Requires heart rate data (activities without HR are excluded)
- power_hr requires power meter data (cycling with power)
- Best for steady-state endurance efforts (tempo runs, long rides)
- Interval workouts may give misleading EF values
- Environmental factors (heat, altitude) can affect EF

Interpretation:

- **Upward trend**: Improving aerobic fitness
- **Stable**: Maintenance phase
- **Downward trend**: Possible overtraining, fatigue, or environmental stress
- **Sudden drop**: Check for illness, equipment change, or data quality

Typical EF Ranges (speed_hr for running):

- Beginner: 0.01 - 0.015 (m/s per bpm)
- Intermediate: 0.015 - 0.020
- Advanced: 0.020 - 0.025
- Elite: > 0.025

Note: EF values are relative to individual baseline. Focus on personal trends rather than absolute comparisons with other athletes.

Value

A tibble with the following columns:

date Activity date (Date class)

activity_type Activity type (character: "Run" or "Ride")

ef_value Efficiency Factor value (numeric). Higher = better fitness. Units: m/s/bpm for speed_hr, W/bpm for power_hr.

status Character status code describing the outcome of the calculation. See **Status vocabulary** below.

ef_metric_requested The metric the user asked for (`"speed_hr"`, `"gap_hr"`, or `"power_hr"`). Mirrors the `ef_metric` argument.

ef_metric_used The metric that was actually computed. Usually matches `ef_metric_requested`; for `gap_hr` inputs processed through the stream path (where no GAP channel is available) this is `"speed_hr"` and the row is also marked with status `"gap_stream_unavailable_fallback_to_speed"`.

quality_score Numeric in [0, 1]. Fraction of stream samples that passed quality-control range checks (HR, velocity, watts). NA when no stream was parsed or when the activity was rejected before QC.

hr_coverage Numeric in [0, 1]. Time-weighted fraction of the stream that carried a valid heart-rate sample. NA for activity-level paths (status = "no_streams").

steady_duration_minutes Wall-clock duration of the contiguous steady-state block the EF value was derived from. NA for activity-level paths or when no qualifying block existed.

n_steady_blocks Number of contiguous steady-state blocks that met the min_steady_minutes threshold. NA for activity-level paths.

sampling_interval_seconds Observed median sampling interval of the stream (seconds). NA for activity-level paths.

Status vocabulary

- "ok": Full steady-state analysis succeeded on stream data.
- "no_streams": No stream file was available; ef_value was computed from activity-level averages. QC and HR-coverage metrics are NA.
- "gap_stream_unavailable_fallback_to_speed": Caller requested ef_metric = "gap_hr" but the stream does not expose a grade-adjusted channel, so ef_value was computed from plain speed/HR. ef_metric_used is "speed_hr" on these rows so downstream code can filter them out.
- "missing_velocity_data": Stream lacked both distance and velocity_smooth when ef_metric = "speed_hr".
- "missing_power_data": Stream lacked watts when ef_metric = "power_hr".
- "missing_hr_data": Stream lacked a heartrate / heart_rate column.
- "insufficient_hr_data": Time-weighted HR coverage < min_hr_coverage.
- "insufficient_data_points": Fewer than 100 non-NA stream samples.
- "insufficient_valid_data": Fewer than 100 samples survived the basic positivity filter (velocity > 0 or watts > 0).
- "insufficient_data_after_quality_filter": quality_control = "filter" removed enough out-of-range samples to drop the stream below 100 rows.
- "poor_hr_quality": Activity-level path with out-of-range average_heartrate while quality_control = "filter".
- "poor_hr_quality_flagged": Activity-level path with out-of-range average_heartrate kept for inspection because quality_control = "flag".
- "too_short": Activity or steady-state window duration is below min_steady_minutes.
- "non_steady": No rolling-CV window cleared steady_cv_threshold.
- "insufficient_steady_duration": No *contiguous* steady-state block lasted at least min_steady_minutes.
- "calculation_failed": Median ratio was non-positive or not finite.

References

Allen, H., Coggan, A. R., & McGregor, S. (2019). *Training and Racing with a Power Meter* (3rd ed.). VeloPress.

Coyle, E. F., & González-Alonso, J. (2001). Cardiovascular drift during prolonged exercise: New perspectives. *Exercise and Sport Sciences Reviews*, 29(2), 88-92. doi:10.1097/00003677200104000-00009

See Also

`plot_ef()` for visualization with trend lines, `calculate_decoupling()` for within-activity efficiency analysis, `load_local_activities()` for data loading

Examples

```
# Example using simulated data
data(sample_ef)
print(head(sample_ef))

# Runnable example with dummy data:
end <- Sys.Date()
dates <- seq(end - 29, end, by = "day")
dummy_activities <- data.frame(
  date = dates,
  type = "Run",
  moving_time = rep(3600, length(dates)), # 1 hour
  distance = rep(10000, length(dates)), # 10 km
  average_heartrate = rep(140, length(dates)),
  average_watts = rep(200, length(dates)),
  weighted_average_watts = rep(210, length(dates)),
  filename = "",
  stringsAsFactors = FALSE
)

# Calculate EF (Speed/HR)
ef_result <- calculate_ef(
  activities_data = dummy_activities,
  activity_type = "Run",
  ef_metric = "speed_hr",
  end_date = end
)
print(head(ef_result))

## Not run:
# Example using local Strava export data
activities <- load_local_activities("strava_export_data/activities.csv")

# Calculate Speed/HR efficiency factor for Runs
ef_data_run <- calculate_ef(
  activities_data = activities,
  activity_type = "Run",
  ef_metric = "speed_hr"
```

```
)
print(tail(ef_data_run))

# Calculate Power/HR efficiency factor for Rides
ef_data_ride <- calculate_ef(
  activities_data = activities,
  activity_type = "Ride",
  ef_metric = "power_hr"
)
print(tail(ef_data_ride))

## End(Not run)
```

`calculate_ef_from_stream` *Calculate EF from Stream Data with Steady-State Analysis*

Description

Calculate efficiency factor (EF) from detailed stream data using steady-state analysis. This function analyzes heart rate and power/pace data to find periods of steady effort and calculates the efficiency factor for those periods.

Usage

```
calculate_ef_from_stream(
  stream_data,
  activity_date,
  act_type,
  ef_metric,
  min_steady_minutes = 20,
  steady_cv_threshold = 0.08,
  min_hr_coverage = 0.9,
  quality_control = "filter"
)
```

Arguments

`stream_data`	Data frame with stream data (time, heartrate, watts/distance columns)
`activity_date`	Date of the activity
`act_type`	Activity type (e.g., "Run", "Ride")
`ef_metric`	Efficiency metric to calculate ("speed_hr" or "power_hr")
`min_steady_minutes`	Minimum duration for steady-state analysis (minutes)
`steady_cv_threshold`	Coefficient of variation threshold for steady state

min_hr_coverage
: Minimum heart rate data coverage required

quality_control
: Quality control setting ("off", "flag", "filter")

Value

Data frame with EF calculation results

Examples

```
# Example with synthetic stream data
set.seed(42)
n <- 3600
stream <- data.frame(
  time = 0:(n - 1),
  heartrate = round(150 + rnorm(n, 0, 2)),
  velocity_smooth = 3.0 + rnorm(n, 0, 0.05),
  distance = cumsum(rep(3.0, n))
)
result <- calculate_ef_from_stream(
  stream_data = stream,
  activity_date = as.Date("2025-01-15"),
  act_type = "Run",
  ef_metric = "speed_hr",
  min_steady_minutes = 10,
  steady_cv_threshold = 0.1,
  min_hr_coverage = 0.8,
  quality_control = "off"
)
print(result)
```

---

calculate_exposure *Calculate Training Load Exposure (ATL, CTL, ACWR)*

Description

Calculates training load metrics like ATL, CTL, and ACWR from local Strava data.

Usage

```
calculate_exposure(
  activities_data,
  activity_type = c("Run", "Ride", "VirtualRide", "VirtualRun"),
  load_metric = "duration_mins",
  acute_period = 7,
  chronic_period = 42,
  user_ftp = NULL,
```

```
  user_max_hr = NULL,
  user_resting_hr = NULL,
  end_date = Sys.Date(),
  missing_load = c("zero", "na"),
  verbose = FALSE
)
```

Arguments

activities_data	A data frame of activities from load_local_activities(). Must contain columns: date, distance, moving_time, elapsed_time, average_heartrate, average_watts, type, elevation_gain.
activity_type	Type(s) of activities to include (e.g., "Run", "Ride"). Default includes common run/ride types.
load_metric	Method for calculating daily load (e.g., "duration_mins", "distance_km", "tss", "hrss"). Default "duration_mins".
acute_period	Days for the acute load window (e.g., 7).
chronic_period	Days for the chronic load window (e.g., 42). Must be greater than acute_period.
user_ftp	Required if load_metric = "tss". Your Functional Threshold Power.
user_max_hr	Required if load_metric = "hrss". Your maximum heart rate.
user_resting_hr	Required if load_metric = "hrss". Your resting heart rate.
end_date	Optional. Analysis end date (YYYY-MM-DD string or Date). Defaults to today. The analysis period covers the chronic_period days ending on this date.
missing_load	How to treat training days on which the chosen load_metric could not be computed (e.g. HRSS with missing HR, TSS without FTP). "zero" (default) matches the historical behaviour and coalesces these days to 0, conflating them with genuine rest days. "na" keeps them visibly NA; only days with no recorded activity at all are treated as rest (0).
verbose	Logical. If TRUE, prints progress messages. Default FALSE.

Details

Calculates daily load, ATL, CTL, and ACWR from Strava activities based on the chosen metric and periods.

Provides data for plot_exposure. Requires extra prior data for accurate initial CTL. Requires FTP/HR parameters for TSS/HRSS metrics.

Value

A data frame with columns: date, daily_load, atl (Acute Load), ctl (Chronic Load), and acwr (Acute:Chronic Ratio) for the analysis period.

Examples

```
# Example using simulated data
data(sample_exposure)
print(head(sample_exposure))

# Runnable example with dummy data:
end <- Sys.Date()
dates <- seq(end - 59, end, by = "day")
dummy_activities <- data.frame(
  date = dates,
  type = "Run",
  moving_time = rep(3600, length(dates)), # 1 hour
  distance = rep(10000, length(dates)), # 10 km
  average_heartrate = rep(140, length(dates)),
  suffer_score = rep(50, length(dates)),
  tss = rep(50, length(dates)),
  stringsAsFactors = FALSE
)

# Calculate Exposure (ATL/CTL)
exposure_result <- calculate_exposure(
  activities_data = dummy_activities,
  activity_type = "Run",
  load_metric = "distance_km",
  acute_period = 7,
  chronic_period = 28,
  end_date = end
)
print(head(exposure_result))

## Not run:
# Example using local Strava export data
activities <- load_local_activities("strava_export_data/activities.csv")

# Calculate training load for Rides using TSS
ride_exposure_tss <- calculate_exposure(
  activities_data = activities,
  activity_type = "Ride",
  load_metric = "tss",
  user_ftp = 280,
  acute_period = 7,
  chronic_period = 28
)
print(head(ride_exposure_tss))

# Calculate training load for Runs using HRSS
run_exposure_hrss <- calculate_exposure(
  activities_data = activities,
  activity_type = "Run",
  load_metric = "hrss",
  user_max_hr = 190,
  user_resting_hr = 50
```

```
)
print(tail(run_exposure_hrss))

## End(Not run)
```

`calculate_pbs` *Calculate Personal Bests (PBs) from Local Strava Data*

Description

Tracks personal best times for standard distances (1k, 5k, 10k, half marathon, marathon) by analyzing detailed activity files from Strava export data.

Usage

```
calculate_pbs(
  activities_data,
  export_dir = "strava_export_data",
  activity_type = "Run",
  start_date = NULL,
  end_date = Sys.Date(),
  distances_m = c(1000, 5000, 10000, 21097.5, 42195),
  verbose = FALSE
)
```

Arguments

`activities_data`	A data frame of activities from `load_local_activities()`. Must contain columns: date, type, filename, distance. Optional `id`, `name`, and `start_date_local` columns are used when available; otherwise row number, filename, and date fallbacks are used.
`export_dir`	Path to a Strava export directory or ZIP file containing the activities folder. Default is "strava_export_data".
`activity_type`	Type of activities to analyze (typically "Run"). Default "Run".
`start_date`	Optional start date for analysis (YYYY-MM-DD). If NULL, defaults to 365 days before `end_date`.
`end_date`	End date for analysis (YYYY-MM-DD). Default `Sys.Date()` (today).
`distances_m`	Target distances in meters to track. Default: c(1000, 5000, 10000, 21097.5, 42195) for 1k, 5k, 10k, half, full marathon.
`verbose`	Logical. If TRUE, prints progress messages. Default FALSE.

Details

This function analyzes detailed activity files (FIT/TCX/GPX) to find the fastest efforts at specified distances. It tracks cumulative personal bests over time, showing when new PBs are set.

Personal best tracking is a standard approach in endurance sport performance monitoring. Systematic PB analysis over multiple distances helps identify fitness improvements, training phase effectiveness, and performance peaks (Matveyev, 1981). The multi-distance approach enables athletes to assess both speed (shorter distances) and endurance (longer distances) progression simultaneously.

References:

- Matveyev, L. P. (1981). *Fundamentals of Sports Training*. Moscow: Progress Publishers.

Note: Requires detailed activity files from your Strava export. Activities must be long enough to contain the target distance segments.

Value

A data frame with columns: `activity_id`, `activity_date`, `distance`, `elapsed_time`, `moving_time`, `time_seconds`, `cumulative_pb_seconds`, `is_pb`, `distance_label`, `time_period`, and `time_basis` (always `"moving"` in the current implementation; see **PB time semantics** below).

PB time semantics

`find_best_effort()` selects the fastest interval whose cumulative *distance* increases strictly monotonically. It builds a compressed moving-time axis from those increasing-distance samples, so samples where the distance counter plateaus (traffic stops, laps pausing the watch, signal dropouts) are excluded from the candidate window and from the reported duration. That makes the reported times *moving-time* best efforts rather than elapsed-time best efforts.

The authoritative field is `time_basis`, which is hard-coded to `"moving"` in the current implementation. For backward compatibility with earlier releases the output still exposes two columns: `elapsed_time` and `moving_time`. Both are populated with the same numeric `time_seconds` value — they are *compatibility columns*, not two independently-computed quantities. Filter on `time_basis` rather than relying on `elapsed_time != moving_time` to tell the two apart, because the current implementation never produces that difference.

If you need an elapsed-time PB (i.e. including paused seconds), use the raw stream with a separate tool; the current implementation intentionally does not attempt to reconstruct paused segments from FIT laps.

Examples

```
# Example using simulated data
data(sample_pbs)
print(head(sample_pbs))

## Not run:
# Load local activities
activities <- load_local_activities("strava_export_data/activities.csv")

# Calculate PBs for standard running distances
pbs_data <- calculate_pbs(
```

```
  activities_data = activities,
  export_dir = "strava_export_data",
  activity_type = "Run"
)
print(head(pbs_data))

# Calculate PBs for custom distances (e.g., 400m, 800m, 1500m for track)
track_pbs <- calculate_pbs(
  activities_data = activities,
  export_dir = "strava_export_data",
  activity_type = "Run",
  distances_m = c(400, 800, 1500, 3000) # Custom distances in meters
)

## End(Not run)
```

`flag_quality` *Flag Data Quality Issues in Activity Streams*

Description

Detects and flags potential data quality issues in activity stream data, including HR/power spikes, GPS drift, and identifies steady-state segments suitable for physiological metrics calculation.

Usage

```
flag_quality(
  streams,
  sport = "Run",
  hr_range = c(30, 220),
  pw_range = c(0, 1500),
  max_run_speed = 7,
  max_ride_speed = 25,
  max_accel = 3,
  max_hr_jump = 10,
  max_pw_jump = 300,
  min_steady_minutes = 20,
  steady_cv_threshold = 0.08
)
```

Arguments

`streams`	A data frame containing activity stream data with time-series measurements. Expected columns: `time` (seconds), `heartrate` (bpm), `watts` (W), `velocity_smooth` or `speed` (m/s), `distance` (m). `heart_rate` (FIT/TCX parser output) is accepted as an alias for `heartrate`, and `power` as an alias for `watts`.
`sport`	Type of activity (e.g., "Run", "Ride"). Default "Run".
`hr_range`	Valid heart rate range as c(min, max). Default c(30, 220).

`pw_range` Valid power range as c(min, max). Default c(0, 1500).

`max_run_speed` Maximum plausible running speed in m/s. Default 7.0 (approx. 2:23/km).

`max_ride_speed` Maximum plausible riding speed in m/s. Default 25.0 (approx. 90 km/h).

`max_accel` Maximum plausible acceleration in m/s². Default 3.0.

`max_hr_jump` Maximum plausible HR change **per second** (bpm/s). Default 10. The value is compared against `|dHR/dt|` (i.e. the per-second rate of change), which makes the threshold meaningful on streams that are not 1 Hz. Earlier versions compared raw sample-to-sample differences, silently changing the effective threshold on 0.5 Hz smart-recording or higher-frequency (e.g. Bluetooth) data.

`max_pw_jump` Maximum plausible power change **per second** (W/s). Default 300. Rate-based in the same sense as `max_hr_jump`.

`min_steady_minutes`
Minimum duration (minutes) for steady-state segment. Default 20.

`steady_cv_threshold`
Coefficient of variation threshold for steady-state, as a dimensionless fraction in (0, 1]. Default 0.08 (i.e. 8\ auto-normalized with a deprecation warning. This brings `flag_quality()` in line with `calculate_ef()` and `calculate_decoupling()`, which have always used fraction-space thresholds.

Details

This function performs several quality checks:

- **HR/Power Spikes**: Flags values outside physiological ranges or with sudden per-second jumps (|dHR/dt| > `max_hr_jump`, |dP/dt| > `max_pw_jump`). Rate computation uses `diff(time)` so the thresholds are sampling-rate invariant.
- **GPS Drift**: Flags implausible speeds or accelerations based on sport type.
- **Steady-State Detection**: Identifies segments with low variability (CV < `steady_cv_threshold`) lasting >= `min_steady_minutes` of *wall-clock time* (not rows), suitable for EF/decoupling calculations.

The function is sport-aware and adjusts thresholds accordingly. All thresholds are configurable to accommodate different athlete profiles and data quality.

Value

A data frame identical to `streams` with additional flag columns:

flag_hr_spike Logical. TRUE if HR is out of range or has excessive jump.

flag_pw_spike Logical. TRUE if power is out of range or has excessive jump.

flag_gps_drift Logical. TRUE if speed or acceleration is implausible.

flag_any Logical. TRUE if any quality flag is raised.

is_steady_state Logical. TRUE if segment meets steady-state criteria.

quality_score Numeric 0-1. Activity-level proportion of clean data (1 = perfect). This is *not* a per-row score, it is the single summary `1 - mean(flag_any)` broadcast to every row for backward-compatible column semantics. The same value is also stored on the returned frame as `attr(result, "activity_quality_score")` so downstream code that wants a single number can read it without assuming row constancy.

Examples

```
# Create sample activity stream data
set.seed(42)
stream_data <- data.frame(
  time = seq(0, 3600, by = 1),
  heartrate = pmax(60, pmin(200, rnorm(3601, mean = 150, sd = 10))),
  watts = pmax(0, rnorm(3601, mean = 200, sd = 20)),
  velocity_smooth = pmax(0, rnorm(3601, mean = 3.5, sd = 0.3))
)

# Flag quality issues
flagged_data <- flag_quality(stream_data, sport = "Run")

# Check summary
cat("Quality score range:", range(flagged_data$quality_score), "\n")
cat("Flagged points:", sum(flagged_data$flag_any), "\n")
```

---

load_local_activities *Load Activities from Local Strava Export*

Description

Reads and processes activity data from a local Strava export, supporting both direct CSV files and compressed ZIP archives. This function converts Strava export data to a format compatible with all Athlytics analysis functions. Designed to work with Strava's official bulk data export (Settings > My Account > Download or Delete Your Account > Get Started).

Usage

```
load_local_activities(
  path = "strava_export_data/activities.csv",
  start_date = NULL,
  end_date = NULL,
  activity_types = NULL
)
```

Arguments

path	Path to activities.csv file OR a .zip archive from Strava export. Supports both CSV and ZIP formats. If a .zip file is provided, the function will automatically extract and read the activities.csv file from within the archive. Default is "strava_export_data/activities.csv".
start_date	Optional. Start date (YYYY-MM-DD or Date/POSIXct) for filtering activities. Defaults to NULL (no filtering).
end_date	Optional. End date (YYYY-MM-DD or Date/POSIXct) for filtering activities. Defaults to NULL (no filtering).
activity_types	Optional. Character vector of activity types to include (e.g., c("Run", "Ride")). Defaults to NULL (include all types).

Details

This function reads the activities.csv file from a Strava data export and transforms the data to match the structure expected by Athlytics analysis functions. The transformation includes:

- Standardizing column names for analysis functions
- Parsing dates into POSIXct format
- Converting distances to meters
- Converting times to seconds
- Filtering by date range and activity type if specified

Language Note: Strava export language must be set to **English** for proper CSV parsing. Change this in Strava Settings > Display Preferences > Language before requesting your data export. If `load_local_activities()` reports missing Strava export columns, first check that the export was requested after changing this setting.

Privacy Note: This function processes local export data only and does not connect to the internet. Ensure you have permission to analyze the data and follow applicable privacy regulations when using this data for research purposes.

Value

A tibble of activity data with standardized column names compatible with Athlytics functions. Key columns include:

- `id`: Activity ID (numeric)
- `name`: Activity name
- `type`: Activity type (Run, Ride, etc.)
- `start_date_local`: Activity start datetime (POSIXct)
- `date`: Activity date (Date)
- `distance`: Distance in meters (numeric)
- `moving_time`: Moving time in seconds (integer)
- `elapsed_time`: Elapsed time in seconds (integer)
- `average_heartrate`: Average heart rate (numeric)
- `average_watts`: Average power in watts (numeric)
- `elevation_gain`: Elevation gain in meters (numeric)

Examples

```
# Example using built-in sample CSV
csv_path <- system.file("extdata", "activities.csv", package = "Athlytics")
if (nzchar(csv_path)) {
  activities <- load_local_activities(csv_path)
  head(activities)
}

## Not run:
```

```
# Load from a local Strava export ZIP archive
activities <- load_local_activities("export_12345678.zip")

# Filter by date and activity type
activities <- load_local_activities(
  path = "export_12345678.zip",
  start_date = "2023-01-01",
  end_date = "2023-12-31",
  activity_types = "Run"
)

## End(Not run)
```

`parse_activity_file` *Parse Activity File (FIT, TCX, or GPX)*

Description

Parse activity files from Strava export data. TCX and GPX parsing requires the suggested `xml2` package; FIT parsing requires the optional `FITfileR` package. `.gz` compressed files require the suggested `R.utils` package.

Usage

```
parse_activity_file(file_path, export_dir = NULL)
```

Arguments

`file_path`	Path to the activity file (can be .fit, .tcx, .gpx, or .gz compressed)
`export_dir`	Base directory of the Strava export (for resolving relative paths)

Value

A data frame with columns: time, latitude, longitude, elevation, heart_rate, power, cadence, speed (all optional depending on file content). Returns NULL if file cannot be parsed or does not exist.

Examples

```
# Parse a built-in example TCX file
tcx_path <- system.file("extdata", "activities", "example.tcx", package = "Athlytics")
if (nzchar(tcx_path)) {
  streams <- parse_activity_file(tcx_path)
  if (!is.null(streams)) head(streams)
}

## Not run:
# Parse a FIT file from a Strava export
streams <- parse_activity_file("activity_12345.fit", export_dir = "strava_export/")
```

```
## End(Not run)
```

`plot_acwr` *Plot ACWR Trend*

Description

Visualizes the Acute:Chronic Workload Ratio (ACWR) trend over time.

Usage

```
plot_acwr(
  data,
  highlight_zones = TRUE,
  sweet_spot_min = 0.8,
  sweet_spot_max = 1.3,
  high_risk_min = 1.5,
  group_var = NULL,
  group_colors = NULL,
  title = NULL,
  subtitle = NULL,
  ...
)
```

Arguments

`data`	A data frame from `calculate_acwr()` or `calculate_acwr_ewma()`. Must contain `date` and `acwr_smooth` columns.
`highlight_zones`	Logical, whether to highlight descriptive ACWR bands (e.g., reference band, high ACWR) on the plot. Default `TRUE`.
`sweet_spot_min`	Lower bound for the reference ACWR band. Default 0.8.
`sweet_spot_max`	Upper bound for the reference ACWR band. Default 1.3.
`high_risk_min`	Lower bound for the high-ACWR band. Default 1.5.
`group_var`	Optional. Column name for grouping/faceting (e.g., "athlete_id").
`group_colors`	Optional. Named vector of colors for groups.
`title`	Optional. Custom title for the plot.
`subtitle`	Optional. Custom subtitle for the plot.
`...`	Additional arguments. Arguments `activity_type`, `load_metric`, `acute_period`, `chronic_period`, `start_date`, `end_date`, `user_ftp`, `user_max_hr`, `user_resting_hr`, `smoothing_period`, `acwr_df` are deprecated and ignored.

Details

Plots the ACWR trend over time. **Best practice: Use** `calculate_acwr()` **first, then pass the result to this function.** ACWR is calculated as acute load / chronic load. The default 0.8-1.3 band is a commonly used reference band.

When `highlight_zones = TRUE`, all zone labels (High ACWR, Elevated ACWR, Reference Band, Low ACWR) are **always displayed** regardless of whether data falls within each zone. The y-axis is automatically extended to ensure all zone annotations remain visible. Zone boundaries can be customised via `sweet_spot_min`, `sweet_spot_max`, and `high_risk_min`.

Note: The predictive value of ACWR for injury outcomes is debated in the literature (Impellizzeri et al., 2020). Zone labels should be interpreted as descriptive workload heuristics, not validated injury predictors. See `calculate_acwr()` documentation for full references.

Value

A ggplot object showing the ACWR trend.

Examples

```
# Example using pre-calculated sample data
data("sample_acwr", package = "Athlytics")
p <- plot_acwr(sample_acwr)
print(p)
```

---

plot_acwr_comparison *Compare RA and EWMA Methods Side-by-Side*

Description

Creates a faceted plot comparing Rolling Average and EWMA ACWR calculations.

Usage

```
plot_acwr_comparison(
  acwr_ra,
  acwr_ewma,
  title = "ACWR Method Comparison: RA vs EWMA"
)
```

Arguments

`acwr_ra`	A data frame from `calculate_acwr_ewma(..., method = "ra")`.
`acwr_ewma`	A data frame from `calculate_acwr_ewma(..., method = "ewma")`.
`title`	Plot title. Default "ACWR Method Comparison: RA vs EWMA".

Value

A ggplot object with faceted comparison.

Examples

```
# Example using sample data
data("sample_acwr", package = "Athlytics")
if (!is.null(sample_acwr) && nrow(sample_acwr) > 0) {
  # Create two versions for comparison (simulate RA vs EWMA)
  acwr_ra <- sample_acwr
  acwr_ewma <- sample_acwr
  acwr_ewma$acwr_smooth <- acwr_ewma$acwr_smooth * runif(nrow(acwr_ewma), 0.95, 1.05)

  p <- plot_acwr_comparison(acwr_ra, acwr_ewma)
  print(p)
}

## Not run:
activities <- load_local_activities("export.zip")

acwr_ra <- calculate_acwr_ewma(activities, activity_type = "Run", method = "ra")
acwr_ewma <- calculate_acwr_ewma(activities, activity_type = "Run", method = "ewma")

plot_acwr_comparison(acwr_ra, acwr_ewma)

## End(Not run)
```

---

plot_acwr_enhanced *Enhanced ACWR Plot with Confidence Bands and Reference*

Description

Creates a comprehensive ACWR visualization with optional confidence bands and cohort reference percentiles.

Usage

```
plot_acwr_enhanced(
  acwr_data,
  reference_data = NULL,
  show_ci = TRUE,
  show_reference = TRUE,
  reference_bands = c("p25_p75", "p05_p95", "p50"),
  highlight_zones = TRUE,
  title = NULL,
  subtitle = NULL,
  method_label = NULL,
  caption = default_acwr_zone_caption(highlight_zones)
)
```

Arguments

acwr_data	A data frame from calculate_acwr_ewma() containing ACWR values.
reference_data	Optional. A data frame from calculate_cohort_reference() for adding cohort reference bands.
show_ci	Logical. Whether to show confidence bands (if available in data). Default TRUE.
show_reference	Logical. Whether to show cohort reference bands (if provided). Default TRUE.
reference_bands	Which reference bands to show. Default c("p25_p75", "p05_p95", "p50").
highlight_zones	Logical. Whether to highlight descriptive ACWR zones. Default TRUE.
title	Plot title. Default NULL (auto-generated).
subtitle	Plot subtitle. Default NULL (auto-generated).
method_label	Optional label for the method used (e.g., "RA", "EWMA"). Default NULL.
caption	Plot caption. Set to NULL to remove. Defaults to zone description when highlight_zones = TRUE.

Details

This enhanced plot function combines multiple visualization layers:

- ACWR zone shading (reference band: 0.8-1.3, elevated ACWR: 1.3-1.5, high ACWR: >1.5)
- Cohort reference percentile bands (if provided)
- Bootstrap confidence bands (if available in data)
- Individual ACWR trend line

The layering order (bottom to top):

1. ACWR zones (background)
2. Cohort reference bands (P5-P95, then P25-P75)
3. Confidence intervals (individual uncertainty)
4. ACWR line (individual trend)

Note: The predictive value of ACWR for injury outcomes is debated in the literature (Impellizzeri et al., 2020). Zone labels should be interpreted as descriptive workload heuristics, not validated injury predictors. See calculate_acwr() documentation for full references.

Value

A ggplot object.

Examples

```
# Example using sample data
data("sample_acwr", package = "Athlytics")
if (!is.null(sample_acwr) && nrow(sample_acwr) > 0) {
  p <- plot_acwr_enhanced(sample_acwr, show_ci = FALSE)
  print(p)
}

## Not run:
# Load activities
activities <- load_local_activities("export.zip")

# Calculate ACWR with EWMA and confidence bands
acwr <- calculate_acwr_ewma(
  activities,
  activity_type = "Run",
  method = "ewma",
  ci = TRUE,
  B = 200
)

# Basic enhanced plot
plot_acwr_enhanced(acwr)

# With cohort reference
reference <- calculate_cohort_reference(cohort_data, metric = "acwr_smooth")
plot_acwr_enhanced(acwr, reference_data = reference)

## End(Not run)
```

---

plot_decoupling *Plot Aerobic Decoupling Trend*

Description

Visualizes the trend of aerobic decoupling over time.

Usage

```
plot_decoupling(
  data,
  add_trend_line = TRUE,
  smoothing_method = "loess",
  caption = NULL,
  title = NULL,
  subtitle = NULL,
  ...
)
```

Arguments

data	A data frame from calculate_decoupling(). Must contain 'date' and 'decoupling' columns.
add_trend_line	Add a smoothed trend line (geom_smooth)? Default TRUE.
smoothing_method	Smoothing method for trend line (e.g., "loess", "lm"). Default "loess".
caption	Plot caption. Default NULL (no caption).
title	Optional. Custom title for the plot.
subtitle	Optional. Custom subtitle for the plot.
...	Additional arguments. Arguments activity_type, decouple_metric, start_date, end_date, min_duration_mins, decoupling_df are deprecated and ignored.

Details

Plots decoupling percentage ((EF_1st_half - EF_2nd_half) / EF_1st_half * 100). Positive values mean HR drifted relative to output. A 5\% threshold line is often used as reference. **Best practice: Use** calculate_decoupling() **first, then pass the result to this function.**

Value

A ggplot object showing the decoupling trend.

Examples

```
# Example using pre-calculated sample data
data("sample_decoupling", package = "Athlytics")
p <- plot_decoupling(sample_decoupling)
print(p)

# Runnable example with a manually created decoupling data frame:
decoupling_df <- data.frame(
  date = seq(Sys.Date() - 29, Sys.Date(), by = "day"),
  decoupling = rnorm(30, mean = 5, sd = 2)
)
plot_decoupling(data = decoupling_df)
```

---

plot_ef *Plot Efficiency Factor (EF) Trend*

Description

Visualizes the trend of Efficiency Factor (EF) over time.

Usage

```
plot_ef(
  data,
  add_trend_line = TRUE,
  smoothing_method = "loess",
  smooth_per_activity_type = FALSE,
  group_var = NULL,
  group_colors = NULL,
  title = NULL,
  subtitle = NULL,
  ...
)
```

Arguments

data	A data frame from calculate_ef(). Must contain date, ef_value, and activity_type columns.
add_trend_line	Add a smoothed trend line (geom_smooth)? Default TRUE.
smoothing_method	Smoothing method for trend line (e.g., "loess", "lm"). Default "loess".
smooth_per_activity_type	Logical. If TRUE and add_trend_line = TRUE, draws separate trend lines for each activity type. Default FALSE (single trend line for all data). Note: this parameter only applies when group_var = NULL. When group_var is set, smoothing is always done per group and this parameter is ignored with a warning.
group_var	Optional. Column name for grouping/faceting (e.g., "athlete_id").
group_colors	Optional. Named vector of colors for groups.
title	Optional. Custom title for the plot.
subtitle	Optional. Custom subtitle for the plot.
...	Additional arguments. Arguments activity_type, ef_metric, start_date, end_date, min_duration_mins, ef_df are deprecated and ignored.

Details

Plots EF (output/HR based on activity averages). **Best practice: Use** calculate_ef() **first, then pass the result to this function.**

Value

A ggplot object showing the EF trend.

Examples

```
# Example using pre-calculated sample data
data("sample_ef", package = "Athlytics")
p <- plot_ef(sample_ef)
print(p)
```

plot_exposure *Plot Training Load Exposure (ATL vs CTL)*

Description

Visualizes the relationship between Acute and Chronic Training Load.

Usage

```
plot_exposure(
  data,
  risk_zones = TRUE,
  show_date_color = TRUE,
  caption = NULL,
  axis_limit = NULL,
  title = NULL,
  subtitle = NULL,
  ...
)
```

Arguments

data	A data frame from `calculate_exposure()`. Must contain `date`, `atl`, and `ctl` columns.
risk_zones	Add background shading for typical descriptive ACWR zones? Default `TRUE`.
show_date_color	Logical. Whether to color points by date (gradient). Default `TRUE`. The date gradient helps visualize the training trajectory over time: lighter colors represent earlier dates and darker colors represent more recent dates, so you can trace how training state has evolved across a season. Set to `FALSE` for a simpler single-color plot (useful when the temporal ordering is less important than the overall distribution).
caption	Plot caption. Default NULL (no caption).
axis_limit	Optional. Numeric value to set both x and y axis limits (0 to axis_limit). Useful when plotting ACWR zones without data or with sparse data. Default NULL (auto-scale).
title	Optional. Custom title for the plot.
subtitle	Optional. Custom subtitle for the plot.
...	Additional arguments. Arguments `activity_type`, `load_metric`, `acute_period`, `chronic_period`, `user_ftp`, `user_max_hr`, `user_resting_hr`, `end_date`, `exposure_df` are deprecated and ignored.

Details

Visualizes training state by plotting ATL vs CTL (related to PMC charts). Points are colored by date, latest point is highlighted (red triangle). Optional ACWR zones (based on thresholds ~0.8, 1.3, 1.5) can be shaded. **Best practice: Use** `calculate_exposure()` **first, then pass the result to this function.**

Value

A ggplot object showing ATL vs CTL.

Examples

```
# Example using simulated data
data(sample_exposure)
# Ensure exposure_df is named and other necessary parameters like activity_type are provided
p <- plot_exposure(sample_exposure)
print(p)

# Runnable example with dummy data:
end <- Sys.Date()
dates <- seq(end - 59, end, by = "day")
dummy_activities <- data.frame(
  date = dates,
  type = "Run",
  moving_time = rep(3600, length(dates)), # 1 hour
  distance = rep(10000, length(dates)), # 10 km
  average_heartrate = rep(140, length(dates)),
  suffer_score = rep(50, length(dates)),
  tss = rep(50, length(dates)),
  stringsAsFactors = FALSE
)

# Calculate Exposure (ATL/CTL)
exposure_result <- calculate_exposure(
  activities_data = dummy_activities,
  activity_type = "Run",
  load_metric = "distance_km",
  acute_period = 7,
  chronic_period = 28,
  end_date = end
)
plot_exposure(exposure_result)
```

---

plot_pbs *Plot Personal Best (PB) Trends*

Description

Visualizes the trend of personal best times for specific running distances.

Usage

```
plot_pbs(
  data,
  add_trend_line = TRUE,
  caption = NULL,
  facet_ncol = 2,
  title = NULL,
  subtitle = NULL,
  ...
)
```

Arguments

data	A data frame from calculate_pbs(). Must contain activity_date, distance, time_seconds.
add_trend_line	Logical. Whether to add a trend line to the plot. Default TRUE.
caption	Plot caption. Default NULL (no caption).
facet_ncol	Integer. Number of columns for faceted plots when multiple distances are shown. Default 2 for better aspect ratio. Set to 1 for vertical stacking.
title	Optional. Custom title for the plot.
subtitle	Optional. Custom subtitle for the plot.
...	Additional arguments. Arguments activity_type, distance_meters, max_activities, date_range, pbs_df are deprecated and ignored.

Details

Visualizes data from calculate_pbs. Points show best efforts; solid points mark new PBs. Y-axis is MM:SS. **Best practice: Use** calculate_pbs() **first, then pass the result to this function.**

Value

A ggplot object showing PB trends, faceted by distance if multiple are plotted.

Examples

```
# Example using the built-in sample data
data("sample_pbs", package = "Athlytics")

if (!is.null(sample_pbs) && nrow(sample_pbs) > 0) {
  # Plot PBs using the package sample data directly
  p <- plot_pbs(sample_pbs)
  print(p)
}

## Not run:
# Example using local Strava export data
activities <- load_local_activities("strava_export_data/activities.csv")
```

```
# Calculate PBs first
pb_data_run <- calculate_pbs(
  activities_data = activities,
  activity_type = "Run",
  distances_m = c(1000, 5000, 10000)
)

if (nrow(pb_data_run) > 0) {
  plot_pbs(pb_data_run)
}

## End(Not run)
```

`plot_with_reference` *Plot Individual Metric with Cohort Reference*

Description

Creates a complete plot showing an individual's metric trend with cohort reference percentile bands.

Usage

```
plot_with_reference(
  individual,
  reference,
  metric = "acwr_smooth",
  date_col = "date",
  title = NULL,
  bands = c("p25_p75", "p05_p95", "p50"),
  caption = NULL
)
```

Arguments

`individual`	A data frame with individual athlete data (from calculate_acwr, etc.)
`reference`	A data frame from `calculate_cohort_reference()`.
`metric`	Name of the metric to plot. Default "acwr_smooth".
`date_col`	Name of the date column. Default "date".
`title`	Plot title. Default NULL (auto-generated).
`bands`	Which reference bands to show. Default c("p25_p75", "p05_p95", "p50").
`caption`	Plot caption. Default NULL (no caption).

Value

A ggplot object.

Examples

```
# Example with weekly data for smooth curves
set.seed(123)
n_weeks <- 40
dates <- seq(as.Date("2023-01-01"), by = "week", length.out = n_weeks)

# Individual athlete data with realistic ACWR fluctuation
individual_data <- data.frame(
  date = dates,
  acwr_smooth = 1.0 + cumsum(rnorm(n_weeks, 0, 0.03))
)

# Cohort reference percentiles with gradual variation
base_trend <- 1.0 + cumsum(rnorm(n_weeks, 0, 0.015))
reference_data <- data.frame(
  date = rep(dates, each = 5),
  percentile = rep(c("p05", "p25", "p50", "p75", "p95"), n_weeks),
  value = as.vector(t(outer(base_trend, c(-0.35, -0.15, 0, 0.15, 0.35), "+")))
)

p <- plot_with_reference(
  individual = individual_data,
  reference = reference_data,
  metric = "acwr_smooth"
)
print(p)

## Not run:
plot_with_reference(
  individual = athlete_acwr,
  reference = cohort_ref,
  metric = "acwr_smooth"
)

## End(Not run)
```

`sample_acwr` *Sample ACWR Data for Athlytics*

Description

A dataset containing pre-calculated Acute:Chronic Workload Ratio (ACWR) and related metrics, derived from simulated Strava data. Used in examples and tests.

Usage

```
sample_acwr
```

Format

A tibble with 365 rows and 5 variables:

date Date of the metrics, as a Date object.

atl Acute Training Load, as a numeric value.

ctl Chronic Training Load, as a numeric value.

acwr Acute:Chronic Workload Ratio, as a numeric value.

acwr_smooth Smoothed ACWR, as a numeric value.

Source

Simulated data generated for package examples.

`sample_decoupling` *Sample Aerobic Decoupling Data for Athlytics*

Description

A dataset containing pre-calculated aerobic decoupling percentages, derived from simulated Strava data. Used in examples and tests.

Usage

```
sample_decoupling
```

Format

A tibble with 365 rows and 2 variables:

date Date of the activity, as a Date object.

decoupling Calculated decoupling percentage, as a numeric value.

Source

Simulated data generated for package examples.

sample_ef *Sample Efficiency Factor (EF) Data for Athlytics*

Description

A dataset containing pre-calculated Efficiency Factor (EF) values, derived from simulated Strava data. Used in examples and tests.

Usage

```
sample_ef
```

Format

A data.frame with 50 rows and 3 variables:

date Date of the activity, as a Date object.

activity_type Type of activity (e.g., "Run", "Ride"), as a character string.

ef_value Calculated Efficiency Factor, as a numeric value.

Source

Simulated data generated for package examples.

sample_exposure *Sample Training Load Exposure Data for Athlytics*

Description

This dataset contains daily training load, ATL, CTL, and ACWR, derived from simulated Strava data. Used in examples and tests, particularly for `plot_exposure`.

Usage

```
sample_exposure
```

Format

A tibble with 365 rows and 5 variables:

date Date of the metrics, as a Date object.

daily_load Calculated daily training load, as a numeric value.

ctl Chronic Training Load, as a numeric value.

atl Acute Training Load, as a numeric value.

acwr Acute:Chronic Workload Ratio, as a numeric value.

Source

Simulated data generated for package examples.

`sample_pbs` *Sample Personal Bests (PBs) Data for Athlytics*

Description

A dataset containing pre-calculated Personal Best (PB) times for various distances, derived from simulated Strava data. Used in examples and tests.

Usage

```
sample_pbs
```

Format

A tibble with 26 rows and 11 variables:

activity_id ID of the activity where the effort occurred, as a character string.

activity_date Date and time of the activity, as a POSIXct object.

distance Target distance in meters for the best effort, as a numeric value.

elapsed_time Elapsed time for the effort in seconds, as a numeric value.

moving_time Moving time for the effort in seconds, as a numeric value.

time_seconds Reported PB duration in seconds, as a numeric value. Interpret this using `time_basis`.

cumulative_pb_seconds The personal best time for that distance up to that date, in seconds, as a numeric value.

is_pb Logical, TRUE if this effort set a new personal best.

distance_label Factor representing the distance (e.g., "1k", "5k").

time_period Formatted time of the effort, as a Period object from lubridate.

time_basis Time basis for the effort, currently `"moving"`.

Source

Simulated data generated for package examples.

summarize_quality *Get Quality Summary Statistics*

Description

Provides a summary of quality flags and steady-state segments.

Usage

```
summarize_quality(flagged_streams)

quality_summary(flagged_streams)
```

Arguments

flagged_streams	A data frame returned by flag_quality().

Value

A list with summary statistics:

total_points Total number of data points

flagged_points Number of flagged points

flagged_pct Percentage of flagged points

steady_state_points Number of steady-state points

steady_state_pct Percentage in steady-state

quality_score Overall quality score (0-1)

hr_spike_pct Percentage with HR spikes

pw_spike_pct Percentage with power spikes

gps_drift_pct Percentage with GPS drift

Examples

```
# Create sample stream and summarize quality
set.seed(42)
stream_data <- data.frame(
  time = seq(0, 3600, by = 1),
  heartrate = pmax(60, pmin(200, rnorm(3601, mean = 150, sd = 10))),
  watts = pmax(0, rnorm(3601, mean = 200, sd = 20)),
  velocity_smooth = pmax(0, rnorm(3601, mean = 3.5, sd = 0.3))
)
flagged_data <- flag_quality(stream_data, sport = "Run")
summarize_quality(flagged_data)
```

`theme_athlytics` *Get Athlytics Theme*

Description

Publication-ready ggplot2 theme with sensible defaults for scientific figures.

Usage

```
theme_athlytics(base_size = 13, base_family = "")
```

Arguments

`base_size`	Base font size (default: 12)
`base_family`	Font family (default: "")

Value

A ggplot2 theme object that can be added to plots

Examples

```
# Apply theme to a plot
ggplot2::ggplot(mtcars, ggplot2::aes(mpg, wt)) +
  ggplot2::geom_point() +
  theme_athlytics()
```

Index