Package 'Athlytics'

April 27, 2025

Title Advanced Sports Performance Analysis for Strava Data

Version 0.1.0

Author Ang [aut, cre]

Maintainer Ang <ang@hezhiang.com>

Description Provides a suite of tools for advanced sports performance analysis and modeling, designed to work with activity data retrieved from Strava. It focuses on applying established sports science models and statistical methods to gain deeper insights into training load, performance prediction, recovery status, and identifying key performance factors, extending basic data analysis capabilities.

License MIT + file LICENSE

URL https://github.com/HzaCode/Athlytics

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

Encoding UTF-8

Depends R (>= 3.6.0)

Imports dplyr (>= 1.0.0), ggplot2, lubridate, purrr, rlang (>= 0.4.0), rStrava, tidyr, viridis, zoo

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

RoxygenNote 7.3.2

NeedsCompilation no

Contents

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

Description

Fetches Strava activity data, calculates daily training load, computes ATL, CTL, ACWR, and smoothed ACWR, returning the results as a data frame.

Usage

```
calculate_acwr(
  stoken,
  activity_type = NULL,
  load_metric = "duration_mins",
  acute_period = 7,
  chronic_period = 28,
  start_date = NULL,
  end_date = NULL,
  user_ftp = NULL,
  user_max_hr = NULL,
  user_resting_hr = NULL,
  smoothing_period = 7
)
```

Arguments

`stoken`	A valid Strava token object obtained using `rStrava::strava_oauth()`.
`activity_type`	Character vector or single string specifying the activity type(s) to include (e.g., "Run", "Ride", c("Run", "Ride")). Default is `NULL` (all activities).
`load_metric`	Character string specifying the metric to use for calculating training load. Options include: "duration_mins", "distance_km", "elapsed_time_mins", "tss", "hrss", "elevation_gain_m". Default is `"duration_mins"`.
`acute_period`	Integer, the number of days for the acute (short-term) load window. Default is 7.
`chronic_period`	Integer, the number of days for the chronic (long-term) load window. Default is 28.
`start_date`	Optional start date (YYYY-MM-DD string or Date object) for the analysis period. Defaults to one year before end_date.
`end_date`	Optional end date (YYYY-MM-DD string or Date object) for the analysis period. Defaults to today.
`user_ftp`	Numeric, required if `load_metric = "tss"`.
`user_max_hr`	Numeric, required if `load_metric = "hrss"`.
`user_resting_hr`	Numeric, required if `load_metric = "hrss"`.
`smoothing_period`	Integer, days for rolling mean smoothing of ACWR. Default 7.

Details

This function performs the data fetching and calculation steps used by `plot_acwr`, returning the underlying data frame containing the raw and smoothed ACWR values.

Value

A data frame with columns: date, acwr, acwr_smooth. Includes data for the specified analysis period (from `start_date` to `end_date`).

Examples

```
## Not run:
# Requires authentication first:
# stoken <- rStrava::strava_oauth(..., cache = TRUE)

# Calculate ACWR data for running based on duration
acwr_data_run <- calculate_acwr(stoken = stoken, activity_type = "Run",
                               load_metric = "duration_mins")
print(tail(acwr_data_run))

# Calculate ACWR data for cycling based on TSS (requires FTP)
acwr_data_ride_tss <- calculate_acwr(stoken = stoken, activity_type = "Ride",
                                     load_metric = "tss", user_ftp = 280)
print(tail(acwr_data_ride_tss))

## End(Not run)
```

`calculate_decoupling` *Calculate Aerobic Decoupling Data*

Description

Fetches Strava activity streams, calculates aerobic decoupling based on Pace/HR or Power/HR, and returns the results as a data frame. Requires fetching detailed stream data, which can be slow.

Usage

```
calculate_decoupling(
  stoken,
  activity_type = c("Run", "Ride"),
  decouple_metric = c("Pace_HR", "Power_HR"),
  start_date = NULL,
  end_date = NULL,
  min_duration_mins = 45,
  max_activities = 50,
  stream_df = NULL
)
```

Arguments

`stoken`	A valid Strava token object obtained using `rStrava::strava_oauth()`.
`activity_type`	Character vector or single string specifying activity type(s).
`decouple_metric`	Character string specifying the metric basis ("Pace_HR" or "Power_HR").
`start_date`	Optional start date (YYYY-MM-DD string or Date object). Defaults to one year ago.
`end_date`	Optional end date (YYYY-MM-DD string or Date object). Defaults to today.
`min_duration_mins`	Numeric, minimum activity duration in minutes. Default 45.
`max_activities`	Integer, max number of recent activities to process streams for. Default 50.
`stream_df`	Optional. A data frame containing pre-fetched activity stream data for a SINGLE activity. If provided, `stoken` and other data fetching parameters are ignored, and decoupling is calculated directly for this stream data. Must contain columns: time, heartrate, and either velocity_smooth (or distance) for Pace_HR, or watts for Power_HR.

Details

If `stream_df` is provided, the function calculates decoupling for that single activity stream. Otherwise, it fetches activities and their streams via Strava API. This function performs the data fetching (including streams) and calculation steps used by `plot_decoupling`, returning the underlying data frame. See `?plot_decoupling` for calculation details and warnings about performance.

Value

A data frame with columns: date, decoupling (percentage). If `stream_df` is provided, returns a single numeric value: the decoupling percentage.

Examples

```
## Not run:
# Requires authentication first:
# stoken <- rStrava::strava_oauth(..., cache = TRUE)

# Calculate Pace/HR decoupling data for recent Runs
decoupling_data_run <- calculate_decoupling(
    stoken = stoken,
    activity_type = "Run",
    decouple_metric = "Pace_HR",
    max_activities = 10 # Use small number for quick example
)
print(tail(decoupling_data_run))

## End(Not run)
```

`calculate_ef` *Calculate Efficiency Factor (EF) Data*

Description

Fetches Strava activity data, calculates a specified Efficiency Factor (EF) metric (Pace/HR or Power/HR), and returns the results as a data frame.

Usage

```
calculate_ef(
  stoken,
  activity_type = c("Run", "Ride"),
  ef_metric = c("Pace_HR", "Power_HR"),
  start_date = NULL,
  end_date = NULL,
  min_duration_mins = 20
)
```

Arguments

`stoken`	A valid Strava token object obtained using `rStrava::strava_oauth()`.
`activity_type`	Character vector or single string specifying activity type(s).
`ef_metric`	Character string specifying the EF metric ("Pace_HR" or "Power_HR").
`start_date`	Optional start date (YYYY-MM-DD string or Date object). Defaults to one year ago.
`end_date`	Optional end date (YYYY-MM-DD string or Date object). Defaults to today.
`min_duration_mins`	Numeric, minimum activity duration in minutes. Default 20.

Details

This function performs the data fetching and calculation steps used by `plot_ef`, returning the underlying data frame.

Value

A data frame with columns: date, activity_type, ef_value.

Examples

```
## Not run:
# Requires authentication first:
# stoken <- rStrava::strava_oauth(..., cache = TRUE)

# Calculate Pace/HR EF data for Runs
ef_data_run <- calculate_ef(stoken = stoken, activity_type = "Run", ef_metric = "Pace_HR")
print(tail(ef_data_run))

# Calculate Power/HR EF data for Rides
ef_data_ride <- calculate_ef(stoken = stoken, activity_type = "Ride", ef_metric = "Power_HR")
```

```
print(tail(ef_data_ride))

## End(Not run)
```

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

Description

Fetches Strava activity data, calculates daily training load based on a specified metric, computes Acute Training Load (ATL), Chronic Training Load (CTL), and Acute:Chronic Workload Ratio (ACWR), and returns the results as a data frame.

Usage

```
calculate_exposure(
  stoken,
  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 = NULL
)
```

Arguments

`stoken`	A valid Strava token object obtained using `rStrava::strava_oauth()`.
`activity_type`	Character vector specifying activity type(s).
`load_metric`	Character string for the load metric ("duration_mins", "distance_km", "tss", "hrss", etc.).
`acute_period`	Integer, days for acute load window. Default 7.
`chronic_period`	Integer, days for chronic load window. Default 42.
`user_ftp`	Numeric, required if `load_metric = "tss"`.
`user_max_hr`	Numeric, required if `load_metric = "hrss"`.
`user_resting_hr`	Numeric, required if `load_metric = "hrss"`.
`end_date`	Optional end date (YYYY-MM-DD string or Date object) for the analysis period. Defaults to today.

Details

This function performs the data fetching and calculation steps used by `plot_exposure`, but returns the underlying data instead of a plot. It fetches activities for a period longer than the chronic window to ensure accurate calculation of initial CTL values.

Value

A data frame containing columns for date, daily_load, atl, ctl, and acwr within the analysis period (last `chronic_period` days ending on `end_date`).

Examples

```
## Not run:
# Requires authentication first:
# stoken <- rStrava::strava_oauth(..., cache = TRUE)

# Calculate exposure data using TSS for Rides
exposure_data_tss <- calculate_exposure(
    stoken = stoken,
    activity_type = "Ride",
    load_metric = "tss",
    user_ftp = 280,
    acute_period = 7,
    chronic_period = 28
)
print(head(exposure_data_tss))

# Calculate exposure data using HRSS for Runs
exposure_data_hrss <- calculate_exposure(
    stoken = stoken,
    activity_type = "Run",
    load_metric = "hrss",
    user_max_hr = 190,
    user_resting_hr = 50,
    acute_period = 7,
    chronic_period = 42
)
print(tail(exposure_data_hrss))

## End(Not run)
```

`calculate_pbs` *Calculate Personal Best (PB) Data for Running Distances*

Description

Fetches Strava activities, extracts best efforts for specified running distances, calculates cumulative PBs, and returns the results as a data frame.

Usage

```
calculate_pbs(
  stoken,
  activity_type = "Run",
  distance_meters,
  max_activities = 500,
  date_range = NULL
)
```

Arguments

`stoken`	A valid Strava authentication token object obtained using `rStrava::strava_oauth()`.
`activity_type`	Character vector specifying activity type(s). Currently, processing logic primarily supports "Run".
`distance_meters`	Numeric vector specifying the distances in meters for PB calculation.
`max_activities`	Integer, max number of recent activities to fetch. Default 500.
`date_range`	Optional character vector c('YYYY-MM-DD', 'YYYY-MM-DD') to filter activities.

Details

This function performs the data fetching and processing steps used by `plot_pbs`, returning the underlying data frame containing all best efforts found for the specified distances and whether each effort constituted a new personal best at that time. It requires fetching detailed activity data for each run, which can be slow.

Value

A data frame containing columns like activity_id, activity_date, distance, time_seconds (elapsed time), cumulative_pb_seconds, is_pb (boolean indicating if that effort set a new PB), distance_label, and time_period.

Examples

```
## Not run:
# Requires authentication first:
# stoken <- rStrava::strava_oauth(..., cache = TRUE)

# Calculate PB data for 1k, 5k, 10k
pb_data <- calculate_pbs(stoken = stoken, distance_meters = c(1000, 5000, 10000))
print(head(pb_data))
new_pbs <- pb_data[pb_data$is_pb, ]
print(new_pbs)

## End(Not run)
```

`fetch_strava_activities` *Fetch and Compile Strava Activities*

Description

Retrieves activity data from Strava API for a given date range, optionally fetching detailed data for each activity if needed for specific load metrics (e.g., power for TSS).

Usage

```
fetch_strava_activities(
  stoken,
  start_date = NULL,
  end_date = NULL,
  fetch_details = FALSE,
  required_cols = c("average_watts", "average_heartrate"),
  delay = 1
)
```

Arguments

`stoken`	A valid Strava token object obtained using `rStrava::strava_oauth()`.
`start_date`	Optional start date (YYYY-MM-DD string or Date object) for fetching activities. If NULL, defaults to a date far in the past (e.g., Strava inception). Be cautious, fetching all data can be slow.
`end_date`	Optional end date (YYYY-MM-DD string or Date object) for fetching activities. If NULL, defaults to the current date.
`fetch_details`	Logical, whether to fetch detailed data for each activity using `get_activity()`. This is necessary for metrics like TSS requiring power or HR data, but significantly increases execution time and API calls. Default is `FALSE`.
`required_cols`	Character vector of column names expected in the final output. If `fetch_details = TRUE`, these columns will be attempted to be extracted from the detailed activity data.
`delay`	Numeric, seconds to wait between calls to `get_activity` when `fetch_details = TRUE`, to help avoid API rate limits. Default is 1.

Details

This function first uses `rStrava::get_activity_list()` to get a summary list of activities within the date range. It then uses `rStrava::compile_activities()` to convert this list into a base data frame.

If `fetch_details` is TRUE, it iterates through each activity ID in the base data frame and calls `rStrava::get_activity()` to retrieve detailed information. The specified `required_cols` (like 'average_watts') are extracted from the detailed list and merged back into the data frame. This process can be slow and hit API rate limits, hence the `delay` parameter. Activities for which detailed fetching fails will have NA for the detail columns.

Dates are converted to POSIXct timestamps for the Strava API call.

Value

A tibble (data frame) containing activity data. Columns included depend on whether details were fetched. Basic columns include 'id', 'start_date_local', 'type', 'distance', 'moving_time', 'elapsed_time'. Detailed columns might include 'average_watts', 'average_heartrate', etc.

plot_acwr *Plot Acute:Chronic Workload Ratio (ACWR) Trend*

Description

Fetches Strava activity data, calculates a specified training load metric, computes the ACWR over time, and generates a plot visualizing the trend.

Usage

```
plot_acwr(
  stoken,
  activity_type = NULL,
  load_metric = "duration_mins",
  acute_period = 7,
  chronic_period = 28,
  start_date = NULL,
  end_date = NULL,
  user_ftp = NULL,
  user_max_hr = NULL,
  user_resting_hr = NULL,
  smoothing_period = 7,
  highlight_zones = TRUE,
  acwr_df = NULL
)
```

Arguments

`stoken` A valid Strava token object obtained using `rStrava::strava_oauth()`.

`activity_type` Character vector or single string specifying the activity type(s) to include (e.g., "Run", "Ride", c("Run", "Ride")). Default is `NULL` (all activities).

`load_metric` Character string specifying the metric to use for calculating training load. Options include:

- `"duration_mins"`: Activity duration in minutes.
- `"distance_km"`: Activity distance in kilometers.
- `"elapsed_time_mins"`: Activity elapsed time in minutes.
- `"tss"`: Training Stress Score (requires `user_ftp` and power data).
- `"hrss"`: Heart Rate Stress Score (requires heart rate data and `user_max_hr`, `user_resting_hr`). *(Note: Implementation uses a simplified, gender-neutral formula)*
- `"elevation_gain_m"`: Activity total elevation gain in meters.

Default is `"duration_mins"`.

`acute_period` Integer, the number of days for the acute (short-term) load window. Default is 7.

`chronic_period` Integer, the number of days for the chronic (long-term) load window. Default is 28.

`start_date` Optional start date (YYYY-MM-DD string or Date object) to filter activities. If `NULL` (default), analysis starts approximately one year before `end_date`.

`end_date`	Optional end date (YYYY-MM-DD string or Date object) to filter activities. If `NULL` (default), analysis ends on the current date.
`user_ftp`	Numeric, the user's Functional Threshold Power (FTP) in Watts. Required if `load_metric = "tss"`. Default is `NULL`.
`user_max_hr`	Numeric, the user's maximum heart rate. Required if `load_metric = "hrss"`. Default is `NULL`.
`user_resting_hr`	Numeric, the user's resting heart rate. Required if `load_metric = "hrss"`. Default is `NULL`.
`smoothing_period`	Integer, the number of days to apply rolling mean smoothing to the ACWR ratio before plotting. Default is 7.
`highlight_zones`	Logical, whether to add background shading for typical ACWR risk zones (e.g., 0.8-1.3 as "Sweet Spot"). Default is `TRUE`.
`acwr_df`	Optional. A data frame containing pre-calculated ACWR data, typically the output of `calculate_acwr`. If provided, `stoken` and other data fetching parameters are ignored, and this data is plotted directly.

Details

This function first fetches activity data using `rStrava`. It then calculates the daily sum of the chosen `load_metric`. Rolling means are applied using the specified `acute_period` and `chronic_period` to calculate acute and chronic load. The ACWR is the ratio of acute load to chronic load.

The function requires authentication with Strava via `rStrava`. Ensure the provided `stoken` has the necessary permissions (e.g., "activity:read_all").

Calculation of TSS requires power data from Strava activities and the user's FTP. HR TRIMP calculation requires heart rate data and user-specific parameters. The exact TRIMP formula used may vary.

Value

A ggplot object showing the ACWR trend over time.

Examples

```
## Not run:
# Requires authentication first:
# stoken <- rStrava::strava_oauth(..., cache = TRUE)

# Plot ACWR for running based on duration
plot_acwr(stoken = stoken, activity_type = "Run", load_metric = "duration_mins")

# Plot ACWR for cycling based on TSS (requires FTP)
plot_acwr(stoken = stoken, activity_type = "Ride", load_metric = "tss", user_ftp = 280)

# Plot ACWR for all activities based on distance, with different periods
plot_acwr(stoken = stoken,
          load_metric = "distance_km",
          acute_period = 10,
          chronic_period = 40,
          smoothing_period = 10)
```

```
# Plot pre-calculated data
# mock_acwr_data <- ... (generate data frame)
# plot_acwr(acwr_df = mock_acwr_data, load_metric = "duration_mins")

## End(Not run)
```

plot_decoupling   *Plot Pace/Power vs. Heart Rate Decoupling Trend*

Description

Analyzes Strava activities to calculate aerobic decoupling (heart rate drift relative to pace or power) and plots the trend over time. Requires fetching detailed activity stream data, which can be slow.

Usage

```
plot_decoupling(
  stoken,
  activity_type = c("Run", "Ride"),
  decouple_metric = c("Pace_HR", "Power_HR"),
  start_date = NULL,
  end_date = NULL,
  min_duration_mins = 45,
  max_activities = 50,
  add_trend_line = TRUE,
  smoothing_method = "loess",
  decoupling_df = NULL
)
```

Arguments

`stoken`	A valid Strava token object obtained using `rStrava::strava_oauth()`.
`activity_type`	Character vector or single string specifying the activity type(s) to analyze (e.g., "Run", "Ride"). Typically applied to steady-state aerobic activities.
`decouple_metric`	Character string specifying the metric basis for decoupling. Options: • `"Pace_HR"`: Compares speed/HR ratio between first and second half. Requires velocity/distance and heart rate streams. • `"Power_HR"`: Compares power/HR ratio between first and second half. Requires power and heart rate streams.
`start_date`	Optional start date (YYYY-MM-DD string or Date object) for the analysis period. Defaults to one year before end_date.
`end_date`	Optional end date (YYYY-MM-DD string or Date object) for the analysis period. Defaults to today.
`min_duration_mins`	Numeric, the minimum activity duration in minutes to include in the analysis. Decoupling is more meaningful in longer activities. Default is 45.
`max_activities`	Integer, the maximum number of recent activities to fetch streams for and analyze. Limits processing time due to slow stream fetching. Default is 50. Reduce for faster testing.

`add_trend_line`	Logical, whether to add a smoothed trend line (`geom_smooth`) to the plot. Default is `TRUE`.
`smoothing_method`	Character string specifying the smoothing method used by `geom_smooth`. Default is "loess".
`decoupling_df`	Optional. A data frame containing pre-calculated decoupling data, typically the output of `calculate_decoupling`. If provided, `stoken` and other data fetching parameters are ignored, and this data is plotted directly.

Details

Aerobic decoupling measures how much heart rate increases for the same output (pace or power) over the duration of an activity, often indicating fatigue or insufficient aerobic fitness. It is calculated by comparing the Efficiency Factor (EF = Output/HR) of the first half of the activity to the second half: Decoupling

Warning: This function requires fetching detailed time-series data (streams) for each activity using `rStrava::get_activity_streams`, which can be significantly slower than other functions and subject to API rate limits. Processing many activities may take considerable time.

The function filters activities by type and minimum duration. It attempts to fetch time, heartrate, velocity_smooth (preferred), distance, and watts streams as needed. Activities without the required streams or sufficient data points will be skipped.

If `decoupling_df` is provided, it is plotted directly. Otherwise, `calculate_decoupling` is called to fetch and process the data.

Value

A ggplot object showing the decoupling percentage trend over time. Lower percentages generally indicate better aerobic endurance.

Examples

```
## Not run:
# Requires authentication first:
# stoken <- rStrava::strava_oauth(..., cache = TRUE)

# Plot Pace/HR decoupling for recent Runs (max 30 activities)
plot_decoupling(stoken = stoken,
                activity_type = "Run",
                decouple_metric = "Pace_HR",
                max_activities = 30)

# Plot Power/HR decoupling for recent Rides
plot_decoupling(stoken = stoken,
                activity_type = "Ride",
                decouple_metric = "Power_HR")

## End(Not run)
```

plot_ef *Plot Efficiency Factor (EF) Trend*

Description

Fetches Strava activity data, calculates a specified Efficiency Factor (EF) metric (either Pace/HR or Power/HR), and plots its trend over time.

Usage

```
plot_ef(
  stoken,
  activity_type = c("Run", "Ride"),
  ef_metric = c("Pace_HR", "Power_HR"),
  start_date = NULL,
  end_date = NULL,
  min_duration_mins = 20,
  add_trend_line = TRUE,
  smoothing_method = "loess",
  ef_df = NULL
)
```

Arguments

`stoken`	A valid Strava token object obtained using `rStrava::strava_oauth()`.
`activity_type`	Character vector or single string specifying the activity type(s) to include (e.g., "Run", "Ride"). The function will filter activities to match the requirements of the chosen `ef_metric`.
`ef_metric`	Character string specifying the EF metric to calculate. Options: • `"Pace_HR"`: Speed (m/s) / Average Heart Rate. Typically used for Running. Requires distance and heart rate data. • `"Power_HR"`: Average Power (Watts) / Average Heart Rate. Typically used for Cycling. Requires power and heart rate data.
`start_date`	Optional start date (YYYY-MM-DD string or Date object) for the analysis period. Defaults to one year before end_date.
`end_date`	Optional end date (YYYY-MM-DD string or Date object) for the analysis period. Defaults to today.
`min_duration_mins`	Numeric, the minimum activity duration in minutes to include in the analysis. Helps filter out very short activities where EF might be noisy. Default is 20.
`add_trend_line`	Logical, whether to add a smoothed trend line (`geom_smooth`) to the plot. Default is `TRUE`.
`smoothing_method`	Character string specifying the smoothing method used by `geom_smooth` if `add_trend_line` is `TRUE`. Default is "loess".
`ef_df`	Optional. A data frame containing pre-calculated Efficiency Factor data, typically the output of `calculate_ef`. If provided, `stoken` and other data fetching parameters are ignored, and this data is plotted directly.

Details

Efficiency Factor (EF) is a measure of output relative to input. For Pace/HR, higher values indicate faster speed for a given heart rate. For Power/HR, higher values indicate more power output for a given heart rate. An upward trend in EF generally suggests improved aerobic fitness.

The function fetches activities using `rStrava::get_activity_list` and filters them based on type, duration, and data availability (average heart rate, and average power if `ef_metric = "Power_HR"`). Activities without the required data for the chosen metric are excluded.

If `ef_df` is provided, it is plotted directly. Otherwise, `calculate_ef` is called.

Value

A ggplot object showing the EF trend over time.

Examples

```
## Not run:
# Requires authentication first:
# stoken <- rStrava::strava_oauth(..., cache = TRUE)

# Plot Pace/HR EF for Runs over the last 6 months
plot_ef(stoken = stoken,
        activity_type = "Run",
        ef_metric = "Pace_HR",
        start_date = Sys.Date() - months(6))

# Plot Power/HR EF for Rides
plot_ef(stoken = stoken,
        activity_type = "Ride",
        ef_metric = "Power_HR")

# Plot Pace/HR EF for Runs and VirtualRuns, without trend line
plot_ef(stoken = stoken,
        activity_type = c("Run", "VirtualRun"),
        ef_metric = "Pace_HR",
        add_trend_line = FALSE)

## End(Not run)
```

`plot_exposure` *Plot Training Load Exposure*

Description

Calculates and visualizes the relationship between acute training load (ATL) and chronic training load (CTL) based on a chosen load metric. This helps assess training readiness and potential injury risk.

Usage

```
plot_exposure(
  stoken,
  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 = NULL,
  risk_zones = TRUE,
  exposure_df = NULL
)
```

Arguments

`stoken`	A valid Strava authentication token object obtained using `rStrava::strava_oauth()`.
`activity_type`	Character or vector of characters. The type(s) of activities to include in the load calculation (e.g., "Run", "Ride", c("Run", "Ride")). Defaults to c("Run", "Ride", "VirtualRide", "VirtualRun").
`load_metric`	Character. The metric used to quantify daily training load. Options include: • `"duration_mins"`: Total activity duration in minutes. • `"distance_km"`: Total activity distance in kilometers. • `"hrss"`: **Approximate** Heart Rate Stress Score (TRIMP variation), calculated using **average heart rate**. Requires `user_max_hr` and `user_resting_hr`. (Note: Accurate HRSS typically requires heart rate stream data for time-in-zones, which would require fetching individual activity streams and be significantly slower). • `"tss"`: **Approximate** Training Stress Score, calculated using **average power** as a proxy for Normalized Power (NP). Requires power data for rides/virtual rides and `user_ftp`. (Note: Accurate TSS requires NP, which needs fetching individual activity power streams and is significantly slower. This approximation may be inaccurate for variable intensity activities). • `"elevation_gain_m"`: Total elevation gain in meters. Defaults to "duration_mins".
`acute_period`	Integer. The number of days for the acute (short-term) load window. Defaults to 7.
`chronic_period`	Integer. The number of days for the chronic (long-term) load window. Defaults to 42.
`user_ftp`	Numeric. Functional Threshold Power (FTP) in Watts. Required if `load_metric = "tss"`.
`user_max_hr`	Numeric. Maximum heart rate. Required if `load_metric = "hrss"`.
`user_resting_hr`	Numeric. Resting heart rate. Required if `load_metric = "hrss"`.
`end_date`	Character. The end date (\'YYYY-MM-DD\') for the analysis period. Defaults to today. The analysis will cover `chronic_period` days prior to this date.
`risk_zones`	Logical. If TRUE, adds background colors representing typical ACWR-based risk zones (e.g., Sweet Spot, Danger Zone). Defaults to TRUE.
`exposure_df`	Optional. A data frame containing pre-calculated exposure data, typically the output of `calculate_exposure`. If provided, `stoken` and other data fetching parameters are ignored, and this data is plotted directly. Must contain columns: date, atl, ctl.

Value

A ggplot object showing acute load vs. chronic load, potentially with risk zones.

Examples

```
## Not run:
# Ensure you have authenticated and have a valid token 'stoken'

# Plot exposure using duration for Runs and Rides (default periods)
plot_exposure(stoken = my_token)

# Plot exposure using TSS for Rides, providing FTP (7-day acute, 28-day chronic)
plot_exposure(stoken = my_token, activity_type = "Ride", load_metric = "tss",
              user_ftp = 280, acute_period = 7, chronic_period = 28)

# Plot exposure using HRSS for Runs, providing HR parameters
plot_exposure(stoken = my_token, activity_type = "Run", load_metric = "hrss",
              user_max_hr = 190, user_resting_hr = 50)

## End(Not run)
```

---

plot_pbs — *Plot Personal Best Trends for Running Distances*

Description

Fetches running activities from Strava, extracts best efforts for specified standard distances (e.g., 1k, 5k, 10k), and plots the trend of these best times over time, highlighting personal bests (PBs).

Usage

```
plot_pbs(
  stoken,
  activity_type = "Run",
  distance_meters,
  max_activities = 500,
  date_range = NULL,
  pbs_df = NULL
)
```

Arguments

`stoken`	A valid Strava authentication token object obtained using `rStrava::strava_oauth()`.
`activity_type`	A character vector specifying the activity types to filter (e.g., "Run", "Ride"). Common types include "Run", "Ride", "Swim", etc.
`distance_meters`	A numeric vector specifying the distances in meters for which to plot PBs (e.g., `c(1000, 5000, 10000)` for 1k, 5k, 10k). Common distances like 400, 800, 1000, 1609 (mile), 5000, 10000, 21097 (half), 42195 (marathon) are typically available in `best_efforts`.

`max_activities`	Integer, the maximum number of recent activities to fetch and process. Defaults to 500. Processing many activities can be time-consuming due to individual API calls. Reduce this number for faster results on recent data.
`date_range`	Optional. A character vector of length 2 specifying the start and end date ('YYYY-MM-DD') to filter activities. Defaults to NULL (all activities).
`pbs_df`	Optional. A data frame containing pre-calculated PB data, typically the output of `calculate_pbs`. If provided, `stoken` and other data fetching parameters are ignored, and this data is plotted directly.

Value

A ggplot object showing the PB trends for the specified distances.

Examples

```
## Not run:
# Ensure you have authenticated and have a valid token 'stoken'
# Plot PBs for 1k, 5k, and 10k
plot_pbs(stoken = my_token, distance_meters = c(1000, 5000, 10000))

# Plot PBs for mile and half marathon for the year 2023
plot_pbs(stoken = my_token,
         distance_meters = c(1609, 21097),
         date_range = c("2023-01-01", "2023-12-31"))

## End(Not run)
```

Index