

Package: ovlytics (via r-universe)

October 13, 2026

Title Functions and Algorithms for Volleyball Analytics

Version 0.3.7

Description Analytical functions for volleyball analytics, to be used in conjunction with the datavolley and peranavolley packages.

URL <https://ovlytics.openvolley.org>,
<https://github.com/openvolley/ovlytics>

BugReports <https://github.com/openvolley/ovlytics/issues>

Imports assertthat, cowplot, datavolley (>= 1.0.2), dplyr (>= 1.1.0),
forcats, ggplot2, grid, gt, htmltools, htmlwidgets, MASS,
methods, ovddata, paletteer, patchwork, purrr, reactable,
reactablefmtr, rlang, scales, stringi, stringr, tidyr,
viridisLite

Suggests covr, shiny, testthat

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

Roxygen list(markdown = TRUE)

Remotes openvolley/datavolley, openvolley/ovdata,
kcuilla/reactablefmtr

Config/pak/sysreqs chromium cmake make libicu-dev libjpeg-dev libuv1-dev libxml2-dev libssl-
dev libnode-dev libx11-dev zlib1g-dev

Repository <https://openvolley.r-universe.dev>

Date/Publication 2026-03-27 05:16:51 UTC

RemoteUrl <https://github.com/openvolley/ovlytics>

RemoteRef HEAD

RemoteSha d67a42882ea88ef72a188ee00925a3cfc3ee7bb1

Contents

attack_eff 2

ov_aev 3

ov_aev_reference_props 5

ov_augment_liberos 6

ov_augment_plays 7

ov_create_history_table 9

ov_example_file 10

ov_heatmap_kde 11

ov_infer_player_roles 12

ov_plot_distribution 14

ov_plot_history_table 15

ov_plot_rate 15

ov_plot_sequence_distribution 16

ov_plot_ssd 17

ov_print_history_table 17

ov_print_rate_table 18

ov_season_table 19

ov_setter_repetition 19

ov_simulate_multiple_setter_distribution 21

ov_simulate_setter_distribution 22

ov_sort_attack_codes 24

ov_table_mssd 24

ovlytics 25

Index 27

attack_eff	<i>Various skill performance indicators</i>
------------	---

Description

- attack_eff: (number of kills - number of errors and blocked attacks) / (number of attacks)
- serve_eff: (number of aces and positive serves - number of errors and poor serves) / (number of serves)
- reception_eff: (number of perfect and positive passes - number of errors and overpasses) / (number of passes)

Usage

```
attack_eff(evaluation, skill)

serve_eff(evaluation, skill)

reception_eff(evaluation, skill)
```

Arguments

evaluation	character: vector of skill evaluations ("Winning attack", "Error", etc)
skill	character: (optional) vector of skill values ("Attack", "Block", etc). If provided, it will be used to filter the evaluation vector to the elements corresponding to the correct skill. If not provided, all elements of evaluation will be used

Value

A numeric scalar

Examples

```
## Not run:
library(dplyr)
x <- ovdata_example("mlafin_braslovce_nkbm", as = "parsed")
plays(x) %>% dplyr::filter(skill == "Attack") %>% group_by(player_name) %>%
  dplyr::summarize(N_attacks = n(), att_eff = attack_eff(evaluation))

## End(Not run)
```

ov_aev

Calculate attack evenness

Description

Attack evenness is a measure of how balanced a team's attack is. Teams that rely heavily on one or two attackers will have a relatively low evenness value, whereas teams that use all of their attackers will be higher. See <https://untan.gl/attack-evenness.html> for further background. Evenness is calculated for each lineup used by a team and averaged (weighting by the number of attacks that each lineup made) to get an overall evenness value. Liberos are not expected to attack, and setter attacks (dumps) are ignored. Evenness can be calculated on grouped data: by default, it is calculated by match but averaged over matches when reporting the final result.

Usage

```
ov_aev(
  x,
  team,
  rotation,
  reference_props = NULL,
  calculate_by = "match_id",
  report_by,
  min_N_attacks = 10,
  detail = FALSE
)
```

Arguments

x	data.frame: the plays data.frame as returned by <code>datavolley::read_dv()</code> or <code>peranavolley::pv_read()</code>
team	string: the team to calculate attack evenness for. If not provided, it will be calculated separately for all teams in x
rotation	string: the calculation needs to know what position each player is playing, so that it can work out how many attacks that player should have made under a "perfectly even attack" scenario. This is important for middle hitters, who are usually replaced by the libero in back court and therefore cannot make attacks at those times. The rotation parameter can be one of: • "player_role": use the player roles as specified in the "player_role" column in x. These are typically the player roles defined in the roster, but it is left to the user to populate this column • "SHM": assume a setter-hitter-middle rotation order (i.e. outside hitter is in position 2 when the setter is in 1) and infer the player roles from that • "SMH": assume a setter-middle-hitter rotation order and infer the player roles from that • "none": don't assume player roles, which will mean that under a "perfectly even attack" scenario, each player (excluding the setter) will be expected to make 20% of attacks. This option is probably of limited use
reference_props	data.frame or NULL: if NULL, the default attack profile as returned by <code>ov_aev_reference_props()</code> will be used (recommended). Otherwise, provide a data.frame with the same format as that returned by <code>ov_aev_reference_props()</code>
calculate_by	character: names variables in x to group by when doing the calculations. Note that "lineup" is always used as a calculation grouping variable (it is populated inside the function). See Details
report_by	character: names of variables in x to group by for the final results. Note that results are always effectively grouped by team. Any report_by variables will be used in addition to team. Note that: • report_by can include "lineup" and/or "setter_id" (these variables are used internally in calculations) • report_by variables must also be present in calculate_by, with the exception of "lineup" and "setter_id"
min_N_attacks	integer: minimum number of attacks that must be made in order to be included in the calculations. Attacks are counted by lineup and calculate_by variables (if any). If calculate_by is empty and min_N_attacks is 10, then only lineups that made 10 or more attacks (in total) will be included in the calculations. If min_N_attacks is 10 and calculate_by is "match_id", then calculations will be done match by match, using only lineups in a match that made 10 or more attacks in that match
detail	logical: if TRUE, the returned data frame will include additional columns: "aev_sd" (standard deviation of aev across the calculate_by groups), "rally_win_rate", "kill_rate", "rec_eff" (reception efficiency), "N_rallies" (number of rallies played). Note that including these details makes the calculation noticeably slower

Details

Note that calculation by group and averaging will not generally give the same results as calculating the average result in one step (e.g. calculating for several matches and averaging those results will probably not give the same answer as calculating for all matches pooled). This is expected: say that my team under-utilizes a particular hitter in one match, and over-utilizes her in another. In both of those matches my team's attack was uneven, and so the average of those two results should return a low evenness value indicating that on average my team's attack was uneven. But if the two matches are pooled and the data analyzed all together, the under-utilization in the first match might be balanced by the over-utilization in the second match, giving a higher attack evenness.

Value

A tibble with at least the columns "team", "aev", and "N_attacks". If detail was TRUE, additional columns will also be present (see the detail parameter)

References

<https://untan.gl/attack-evenness.html>

Examples

```
px <- plays(dv_read(ovdata_example()))

## for a single team
ov_aev(px, rotation = "SHM", team = "GKS Katowice")

## for all teams in px, and with extra detail
ov_aev(px, rotation = "SHM", detail = TRUE)

## for a single team, calculated by set number but aggregate results when reporting
ov_aev(px, team = "GKS Katowice", rotation = "SHM", calculate_by = "set_number")

## for a single team, calculated and reported by match and set number
ov_aev(px, team = "GKS Katowice", rotation = "SHM", calculate_by = c("match_id", "set_number"),
       report_by = c("match_id", "set_number"))

## for a single team, calculated by match and setter position, and reported by setter position
px <- ov_augment_plays(px, to_add = "setters", use_existing = FALSE)
ov_aev(px, team = "GKS Katowice", rotation = "SHM",
       calculate_by = c("match_id", "setter_position"), report_by = "setter_position")
```

ov_aev_reference_props

Reference proportions used in attack evenness calculations

Description

Reference proportions used in attack evenness calculations

Usage

```
ov_aev_reference_props()
```

Value

A tibble with columns `player_role` ("middle", "outside", "opposite", "setter", "libero"), `player_front_back` (whether that player is front or back court), and `p_expected` (the proportion of attacks that the player is expected to make under a perfectly even attack scenario)

See Also

```
ov_aev()
```

ov_augment_liberos	<i>Add information about liberos to a datavolley plays dataframe</i>
--------------------	--

Description

Add information about liberos to a datavolley plays dataframe

Usage

```
ov_augment_liberos(x, liberos, middles)
```

Arguments

x	data.frame: the plays data.frame as returned by <code>datavolley::read_dv()</code> or <code>peranavolley::pv_read()</code> . Can be plays from multiple matches combined using <code>rbind</code> or <code>bind_rows</code>
liberos	character: vector of <code>player_id</code> values of liberos. Or a data.frame with <code>match_id</code> , <code>player_id</code> , <code>team</code> , <code>role</code> columns (only rows with <code>role == "libero"</code> will be used as liberos). The latter form will be necessary if x contains data from multiple matches and a player plays as a libero in some matches but other positions in other matches
middles	character: vector of <code>player_id</code> values of middle blockers. Or a data.frame with <code>match_id</code> , <code>player_id</code> , <code>team</code> , <code>role</code> columns (only rows with <code>role == "middle"</code> will be used as middles)

Details

DataVolley (.dvw) files do not include information about the libero on court. This function adds that information where it can. It assumes that the libero replaces the back-court middle blocker, except when a middle blocker is serving.

The procedure is broadly:

- all recorded ball touches are checked on a per-set basis for each team: if there were no ball touches by any libero (for a given team) during a set, it is assumed that no libero was used in that set by that team. If ball touches were recorded only for one libero, it is assumed that only a single libero was used by that team in that set, even if that team has multiple liberos available
- this information is then refined on a rally-by-rally basis: in a rally where the libero made a ball touch we can be sure that this libero was on court

The function returns a copy of `x` with two columns added:

- the `libero_on_court` column will contain `TRUE` if a libero was on court for that data row, or `FALSE` if not on court, or `NA` if unknown
- the `libero_id` column will contain the `player_id` of the libero that was on court for that data row. If no libero was on court, the value `"NO_LIBERO"` will be used. If a libero was known to be on court but we could not identify who it was, the value `"MULTIPLE_LIBEROS"` will be used

Value

A data.frame of `x` augmented with columns `libero_id` and `libero_on_court`. See Details for the contents of these columns

Examples

```
x <- ovdata_example("190301_kats_beds", as = "parsed")

## identify our liberos and middles
liberos <- c(x$meta$players_h$player_id[which(x$meta$players_h$role == "libero")],
            x$meta$players_v$player_id[which(x$meta$players_v$role == "libero")])
middles <- c(x$meta$players_h$player_id[which(x$meta$players_h$role == "middle")],
            x$meta$players_v$player_id[which(x$meta$players_v$role == "middle")])

## infer libero info
px <- ov_augment_liberos(plays(x), liberos = liberos, middles = middles)
```

ov_augment_plays

Add some extra columns to a plays object

Description

Add some extra columns to a plays object

Usage

```
ov_augment_plays(
  x,
  to_add = c("receiving_team", "touch_summaries", "setters"),
  rotation = "SHM",
  use_existing = TRUE
)
```

Arguments

- | | |
|----------|--|
| x | data.frame: the plays data.frame as returned by <code>datavolley::read_dv()</code> or <code>peranavolley::pv_read()</code> . Can be plays from multiple matches combined using <code>rbind</code> or <code>bind_rows</code> |
| to_add | <p>character: columns to add</p> <ul style="list-style-type: none"> • "receiving_team" adds the columns "receiving_team" (team name) and "receiving_team_id" • "winners" adds the columns "set_won_by", "set_won_by_id" (the name and ID of the team that won the current set), "match_won_by", "match_won_by_id" (the name and ID of the team that won the current match), "team_won_set" and "team_won_match" (did the team making the action in the current row win the set/match), and "home_sets_won" and "visiting_sets_won" (the number of sets won by the home and visiting teams) • "touch_summaries" adds a number of columns named "ts_*" that summarize a team touch (e.g. columns "ts_pass_quality", "ts_pass_evaluation_code" give the pass quality and pass evaluation code of the reception or dig associated with a given team touch). "touch_summaries" also adds a column named <code>freeball_over</code>, which disambiguates the action of putting a freeball over the net from the action of digging such a ball. Many scouts code both of these as a "Freeball". The <code>freeball_over</code> column will be TRUE if it was a freeball being put over the net, and FALSE otherwise (including freeball digs). Freeballs over and freeball digs will still both have "Freeball" in their skill column • "setters" adds the columns "home_setter_id", "visiting_setter_id" (the player IDs of the home and visiting setter on court), and "setter_id", "setter_position", and "setter_front_back" (the player ID and position of the setter from the team performing the current action) • "followed" adds the columns "followed_timeout", "followed_technical_timeout", and "followed_sub" • "player_role" add the column "player_role" which gives the role (outside, middle, opposite, setter) for the active player on each row of x. This assumes a standard rotation as specified by <code>rotation</code>. Note that <code>player_role</code> does NOT include libero, although this can be inferred from the meta component of a full <code>datavolley</code> object • "all" is a shortcut for all of the above |
| rotation | string: (only relevant when <code>to_add</code> includes "player_role") either "SHM" (assume a setter-hitter-middle rotation order, i.e. outside hitter is in position 2 when the setter is in 1), or "SMH" (setter-middle-hitter) |

use_existing logical: if TRUE and all of the columns associated with a given to_add choice are already present in x, they won't be re-generated

Value

x with the extra columns added

ov_create_history_table

Create a prior table from a dvw or a directory of dvw files

Description

Create a prior table from a dvw or a directory of dvw files

Usage

```
ov_create_history_table(
  dvw,
  play_phase = c("Reception", "Transition"),
  attack_by = "code",
  setter_position_by = "rotation",
  exclude_attacks = c("PR"),
  normalize_parameters = TRUE
)
```

Arguments

dvw string: path to one or more datavolley files, a list of one or more datavolley objects, or a directory containing datavolley files

play_phase character: one or both of "Reception", "Transition"

attack_by string: either "code", "zone", "tempo" or "setter call"

setter_position_by string: either "rotation", or "front_back"

exclude_attacks character: vector of attack codes to exclude

normalize_parameters logical: reduce the prior parameter values

Value

A list, currently with one component named "prior_table"

Examples

```
## use this file to create the priors
hist_dvw <- ovdata_example("NCA-CUB")
history_table <- ov_create_history_table(dvw = hist_dvw, attack_by = "attacker_name",
                                         setter_position_by = "front_back")

## use it on another file (here, the same file for demo purposes)
## usually the history would be from a reference set of previous matches

dvw <- ovdata_example("NCA-CUB")
setter <- ov_simulate_setter_distribution(dvw = dvw, play_phase = "Reception", n_sim = 100,
                                         attack_by = "attacker_name", attack_options = "use_history",
                                         setter_position_by = "front_back",
                                         history_table = history_table, filter_sim = TRUE)

## plot the results
ov_plot_ssd(setter, overlay_set_number = TRUE)
ov_plot_distribution(setter)
```

ov_example_file

Example DataVolley files provided as part of the ovlytics package

Description

Example DataVolley files provided as part of the ovlytics package

Usage

```
ov_example_file(choice = "190301_kats_beds")
```

Arguments

choice string: which data file to return?

- "190301_kats_beds" - a match between GKS Katowice and MKS Bedzin during the 2018/19 Polish Plus Liga

Value

path to the file

Examples

```
myfile <- ov_example_file()
x <- dv_read(myfile)
summary(x)
```

ov_heatmap_kde	<i>Kernel density estimates for volleyball heatmaps</i>
----------------	---

Description

Kernel density estimates for volleyball heatmaps

Usage

```
ov_heatmap_kde(
  x,
  y,
  N = NULL,
  resolution = "coordinates",
  bw,
  n,
  court = "full",
  auto_flip = FALSE
)
```

Arguments

x	: either a numeric vector of x-locations, or a three-column data.frame or matrix with columns x, y, and optionally N. If x is a grouped tibble, the kernel density estimates will be calculated separately for group
y	numeric: (unless x is a data.frame or matrix) a numeric vector of y-locations
N	numeric: (unless x is a data.frame or matrix) a numeric vector of counts associated with each location (the corresponding location was observed N times)
resolution	string: the resolution of the locations, either "coordinates" or "subzones"
bw	numeric: a vector of bandwidths to use in the x- and y-directions (see MASS::kde2d()). If not provided, default values will be used based on the location resolution
n	integer: (scalar or a length-2 integer vector) the number of grid points in each direction. If not provided, 60 points in the x-direction and 60 (for half-court) or 120 points in the y-direction will be used
court	string: "full" (generate the kernel density estimate for the full court) or "lower" or "upper" (only the lower or upper half of the court)
auto_flip	logical: if TRUE, and court is either "lower" or "upper", then locations corresponding to the non-selected half of the court will be flipped. This might be appropriate if, for example, the heatmap represents attack end locations that were scouted with coordinates (because these aren't necessarily all aligned to the same end of the court by default)

Value

A data.frame with columns x, y, and density

Examples

```

library(ggplot2)
library(datavolley)

## Example 1 - by coordinates
## generate some fake coordinate data
Na <- 20
set.seed(17)
px <- data.frame(x = c(runif(Na, min = 0.4, max = 1.2), runif(Na, min = 2, max = 3)),
                 y = c(runif(Na, min = 4.5, max = 6.6), runif(Na, min = 4.9, max = 6.6)))

## plot as points
ggplot(px, aes(x, y)) + ggcourt(labels = NULL, court = "upper") +
  geom_point(colour = "dodgerblue")

## or as a heatmap
hx <- ov_heatmap_kde(px, resolution = "coordinates", court = "upper")
ggplot(hx, aes(x, y, fill = density)) +
  scale_fill_distiller(palette = "Purples", direction = 1, labels = NULL,
                      name = "Attack\ndensity") +
  geom_raster() + ggcourt(labels = NULL, court = "upper")

## Example 2 - by subzones, with data from two attackers
## generate some fake data
Na <- 20
set.seed(17)
px <- data.frame(zone = sample(c(1, 5:9), Na * 2, replace = TRUE),
                 subzone = sample(c("A", "B", "C", "D"), Na * 2, replace = TRUE),
                 attacker = c(rep("Attacker 1", Na), rep("Attacker 2", Na)))

## convert to x, y coordinates
px <- cbind(px, dv_xy(zones = px$zone, end = "upper", subzone = px$subzone))

## plot as tiles
library(dplyr)
ggplot(count(px, attacker, x, y), aes(x, y, fill = n)) + geom_tile() +
  facet_wrap(~attacker) + ggcourt(labels = NULL, court = "upper")

## or as a heatmap, noting that we group the data by attacker first
hx <- ov_heatmap_kde(group_by(px, attacker), resolution = "subzones", court = "upper")
ggplot(hx, aes(x, y, fill = density)) + facet_wrap(~attacker) +
  scale_fill_distiller(palette = "Purples", direction = 1, labels = NULL,
                      name = "Attack\ndensity") +
  geom_raster() + ggcourt(labels = NULL, court = "upper")

```

Description

Infer the role of each player

Usage

```
ov_infer_player_roles(
  x,
  target_team,
  method,
  fall_back = TRUE,
  setter_tip_codes = c("PP", "PR", "P2")
)
```

Arguments

x	: a datavolley object (as returned by <code>datavolley::dv_read()</code>), a list of datavolley objects, or the plays component of a datavolley object
target_team	string or function: team to report on. If this is a function, it should return TRUE when passed the target team name
method	string: one of • "meta" - rely on player metadata, i.e. the team rosters • "SHM" - assume a setter-hitter-middle rotation order. When the setter is in position 1 the outside hitters are in 2 and 5, and middles in 3 and 6 • "SMH" - setter-middle-hitter rotation order. When the setter is in position 1 the middles are in 2 and 5, and outside hitters in 3 and 6 • or "data" - try and identify player roles based on the ball touches that each player makes. Note that this assumes certain play characteristics: outsides hit pipes and in front row hit from the left except in P1 reception, opposites hit from the right (left in P1 reception), middles hit quick balls or slides or from zone 3 Method "meta" is the default if a datavolley object or list of objects is provided. Note that "SHM" and "SMH" cannot identify liberos, because they do not appear in the point-by-point on-court lineups.
fall_back	logical: if TRUE and method is "meta" and x is a single datavolley object BUT player roles are not provided in the DataVolley file metadata section, fall back to method "data"
setter_tip_codes	character: vector of attack combination codes that correspond to setter tips and other attacks that do not tend to be made from a consistent start zone. Only relevant if method is "data"

Details

The inference procedure assumes that each player plays the same role across the entire data set x. If x is drawn from multiple matches, and a player changes role from match to match, it will likely give misleading results for that player. In that case, it is probably better to run this inference on each match separately.

Value

A data.frame with columns player_id, role, and score (the approximate confidence in the role assignment, in the range 0 to 1)

Examples

```
x <- ovdata_example("mlafin_braslovce_nkbm", as = "parsed")
## guess roles according to the actions that the players made
rx <- ov_infer_player_roles(x, target_team = "Nova KBM Branik", method = "data")
```

ov_plot_distribution *Court plot of a real and simulated setter distribution*

Description

Court plot of a real and simulated setter distribution

Usage

```
ov_plot_distribution(
  ssd,
  label_setters_by = "id",
  font_size = 11,
  title_wrap = NA,
  output = "plot"
)
```

Arguments

ssd	simulated setter distribution output as returned by ov_simulate_setter_distribution()
label_setters_by	string: either "id" or "name"
font_size	numeric: font size
title_wrap	numeric: if non-NA, use strwrap() to break the title into lines of this width
output	string: either "plot" or "list"

Examples

```
dvw <- ovdata_example("NCA-CUB")
setter <- ov_simulate_setter_distribution(dvw = dvw, play_phase = c("Reception", "Transition"),
                                         n_sim = 100, attack_by = "code")
ov_plot_distribution(setter)
```

ov_plot_history_table *Plot the prior table*

Description

Plot the prior table

Usage

```
ov_plot_history_table(history_table, team, setter_id)
```

Arguments

history_table	data.frame: the prior_table component of the object returned by ov_create_history_table()
team	string: team name
setter_id	string: setter_id

Examples

```
hist_dvw <- ovdata_example("NCA-CUB")
history_table <- ov_create_history_table(dvw = hist_dvw, setter_position_by = "front_back",
                                         normalize_parameters = FALSE)

team = unique(history_table$prior_table$team)[1]
setter_id = unique(history_table$prior_table$setter_id)[1]
ov_plot_history_table(history_table, team, setter_id)
```

ov_plot_rate *Plot the rates*

Description

Plot the rates

Usage

```
ov_plot_rate(ssd, team, setter_id, range = c(0.05, 0.95))
```

Arguments

ssd	simulated setter distribution output as returned by ov_simulate_setter_distribution()
team	string: team name
setter_id	string: setter_id
range	vector of maximum and minimum quantile value description

Examples

```

dvw <- ovdata_example("NCA-CUB")
system.time({
  ssd <- ov_simulate_setter_distribution(dvw = dvw, play_phase = c("Reception", "Transition"),
                                       n_sim = 150, setter_position_by = "front_back",
                                       attack_by = "zone")

  team <- ssd$raw_data$meta$teams$team[1]
  setter_id <- ssd$raw_data$meta$players_h$player_id[which(ssd$raw_data$meta$players_h$role == "setter")][2]
  ov_plot_rate(ssd, team, setter_id)
})

```

ov_plot_sequence_distribution

Plot a simulated setter distribution sequence

Description

Plot a simulated setter distribution sequence

Usage

```

ov_plot_sequence_distribution(
  ssd,
  label_setters_by = "id",
  font_size = 11,
  title_wrap = NA,
  split_set = FALSE,
  output = "plot"
)

```

Arguments

ssd	simulated setter distribution output as returned by ov_simulate_setter_distribution()
label_setters_by	string: either "id" or "name"
font_size	numeric: font size
title_wrap	numeric: if non-NA, use strwrap() to break the title into lines of this width
split_set	boolean: if TRUE, separate the distribution sequence by set
output	string: either "plot" or "list"

Examples

```

dvw <- ovdata_example("NCA-CUB")
ssd <- ov_simulate_setter_distribution(dvw = dvw, play_phase = c("Reception"),
                                       n_sim = 100, attack_by = "zone",
                                       setter_position_by = "front_back")

ov_plot_sequence_distribution(ssd)

```

ov_plot_ssd

Plot a simulated setter distribution

Description

Plot a simulated setter distribution

Usage

```
ov_plot_ssd(
  ssd,
  overlay_set_number = FALSE,
  label_setters_by = "name",
  font_size = 11
)
```

Arguments

ssd simulated setter distribution output as returned by [ov_simulate_setter_distribution\(\)](#)

overlay_set_number boolean: if TRUE, overlay set number and score in the plot

label_setters_by string: either "id" or "name"

font_size numeric: font size

Examples

```
dvw <- ovdata_example("NCA-CUB")
setter <- ov_simulate_setter_distribution(dvw = dvw,
                                         n_sim = 150, attack_by = "zone")
ov_plot_ssd(setter, overlay_set_number = TRUE)
```

ov_print_history_table

Print the prior table

Description

Print the prior table

Usage

```
ov_print_history_table(history_table, team, setter_id)
```

Arguments

history_table data.frame: the prior_table component of the object returned by [ov_create_history_table\(\)](#)

team string: team name

setter_id string: setter_id

Examples

```
hist_dvw <- ovdata_example("NCA-CUB")
history_table <- ov_create_history_table(dvw = hist_dvw, attack_by = "zone")
team = history_table$prior_table$team[1]
setter_id = history_table$prior_table$setter_id[1]
ov_print_history_table(history_table, team, setter_id)
```

ov_print_rate_table *Print the rate table*

Description

Print the rate table

Usage

```
ov_print_rate_table(ssd, team, setter_id)
```

Arguments

ssd simulated setter distribution output as returned by [ov_simulate_setter_distribution\(\)](#)

team string: team name

setter_id string: setter_id

Examples

```
dvw <- ovdata_example("NCA-CUB")
system.time({
  ssd <- ov_simulate_setter_distribution(dvw = dvw, play_phase = "Reception",
                                         n_sim = 100, setter_position_by = "front_back")
  team <- ssd$raw_data$meta$teams$team[1]
  setter_id <- ssd$raw_data$meta$players_h$player_id[which(ssd$raw_data$meta$players_h$role == "setter")][2]
  ov_print_rate_table(ssd, team, setter_id)
})
```

ov_season_table	Create a summary table of a team's matches in a season
-----------------	--

Description

Create a summary table of a team's matches in a season

Usage

```
ov_season_table(xl, target_team, target_team_id, show_by = "match date")
```

Arguments

xl	list: list of datavolley objects (each as returned by <code>datavolley::dv_read()</code>)
target_team	string: the name of the target team. Only one of target_team or target_team_id is required
target_team_id	string: the team ID of the target team. Ignored if target_team has been provided
show_by	string: either "match date" (show each match according to its date) or "filename" (show each match according to its filename. This might be useful if the match dates are being parsed incorrectly by <code>datavolley::dv_read()</code>)

Value

A tibble with columns "Opponent", "Date" (or "File"), "Result", "Set scores", and one column for sets 1 to 5

Examples

```
## trivial example of a single-match "season"
library(datavolley)
x <- dv_read(dv_example_file())
ov_season_table(list(x), target_team = home_team(x))
```

ov_setter_repetition	Tabulate setter repeat patterns
----------------------	---------------------------------

Description

Note: analysis is done on the basis of attack actions, and simply assumes that the setter on court made the set.

Usage

```
ov_setter_repetition(
  x,
  setter_id,
  setter_name,
  exclude_attacks = c("PP", "PR", "P2"),
  exclude_negative_reception = TRUE,
  exclude_highballs = FALSE
)
```

Arguments

<code>x</code>	data.frame: the plays data.frame as returned by <code>datavolley::read_dv()</code> or <code>peranavolley::pv_read()</code>
<code>setter_id</code>	string: (optional) the player ID of the setter to analyze (or provide <code>setter_name</code>). If neither <code>setter_id</code> nor <code>setter_name</code> are provided, all setters will be analyzed separately, and collated results returned
<code>setter_name</code>	string: (optional) the name of the setter to analyze (ignored if <code>setter_id</code> is provided). If neither <code>setter_id</code> nor <code>setter_name</code> are provided, all setters will be analyzed separately, and collated results returned
<code>exclude_attacks</code>	character: vector of attack codes to exclude
<code>exclude_negative_reception</code>	logical: if TRUE, exclude attacks following poor reception (likely to be out-of-system and therefore might not represent attacks on which the setter had genuine options)
<code>exclude_highballs</code>	logical: if TRUE, exclude highball attacks (likely to be out-of-system and therefore might not represent attacks on which the setter had genuine options)

Value

A data.frame with columns "team", "setter_name", "setter_id", "player_name", "player_id", "category", "opportunities", "repeats", "repeat%"

Examples

```
x <- plays(ovdata_example("NCA-CUB", as = "parsed"))
set_reps <- ov_setter_repetition(x, setter_name = "LOLETTE RODRIGUEZ")

library(ggplot2)
ggplot(set_reps, aes(x = player_name, y = `repeat%`)) + geom_col() +
  geom_text(aes(x = player_name, label = paste0("N=", opportunities)),
    angle = 90, y = 100, hjust = 1, inherit.aes = FALSE) +
  facet_wrap(~category) +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 60, vjust = 1, hjust = 1)) +
  labs(x = NULL, y = "Repeat percentage")
```

ov_simulate_multiple_setter_distribution

Simulate a Bayesian Bandit choice for a given set of probabilities and a number of points for multiple games

Description

Simulate a Bayesian Bandit choice for a given set of probabilities and a number of points for multiple games

Usage

```
ov_simulate_multiple_setter_distribution(
  list_dv,
  play_phase = c("Reception", "Transition"),
  n_sim = 500,
  priors = list(name = "beta", par1 = 1, par2 = 1),
  epsilon = 1,
  filter_sim = FALSE,
  attack_options = "use_data",
  killRate_grouping = NULL,
  setter_position_by = "rotation",
  history_table = NULL,
  attack_by = "code",
  exclude_attacks = c("PR"),
  rotation = "SHM",
  shiny_progress = NULL
)
```

Arguments

list_dv	list: list of datavolley object as returned by <code>datavolley::dv_read()</code>
play_phase	character: one or both of "Reception", "Transition"
n_sim	integer: number of simulations
priors	numeric: prior distribution of the kill rate for the different attacking options
epsilon	numeric: reward size
filter_sim	logical:
attack_options	string: either "use_data" or "use_history"
killRate_grouping	string: Default to NULL, it will use 'attack by' grouping variables. Otherwise a set of additional grouping variables to calculate the kill rate.
setter_position_by	string: either "rotation" or "front_back"
history_table	list: (only if attack_options is "use_history") the object returned by <code>ov_create_history_table()</code>

attack_by	string: either "code", "zone", "tempo", "setter call", "attacker_name", "player_role"
exclude_attacks	character: vector of attack codes to exclude
rotation	string: (only relevant when attack_by is "player_role") either "SHM" (assume a setter-hitter-middle rotation order), or "SMH" (setter-middle-hitter)
shiny_progress	numeric: an optional two-element vector. If not NULL or NA, shiny::setProgress() calls will be made during simulation with values in this range

See Also

[ov_simulate_setter_distribution\(\)](#)

Examples

```
list_dv <- list(dv_read(ovdata_example("NCA-CUB")), dv_read(ovdata_example("NCA-CUB")))
system.time({
  mssd <- ov_simulate_multiple_setter_distribution(list_dv = list_dv, play_phase = "Reception",
    n_sim = 100, setter_position_by = "front_back")
})
```

ov_simulate_setter_distribution

Simulate a Bayesian Bandit choice for a given set of probabilities and a number of points

Description

Simulate a Bayesian Bandit choice for a given set of probabilities and a number of points

Usage

```
ov_simulate_setter_distribution(
  dvw,
  play_phase = c("Reception", "Transition"),
  n_sim = 500,
  priors = list(name = "beta", par1 = 1, par2 = 1),
  epsilon = 1,
  filter_sim = FALSE,
  attack_options = "use_data",
  killRate_grouping = NULL,
  setter_position_by = "rotation",
  history_table = NULL,
  attack_by = "code",
  exclude_attacks = c("PR"),
  rotation = "SHM",
  shiny_progress = NULL
)
```

Arguments

<code>dvw</code>	string or datavolley: a datavolley object as returned by <code>datavolley::dv_read()</code> or a path to datavolley file
<code>play_phase</code>	character: one or both of "Reception", "Transition"
<code>n_sim</code>	integer: number of simulations
<code>priors</code>	numeric: prior distribution of the kill rate for the different attacking options
<code>epsilon</code>	numeric: reward size
<code>filter_sim</code>	logical:
<code>attack_options</code>	string: either "use_data" or "use_history"
<code>killRate_grouping</code>	string: Default to NULL, it will use 'attack by' grouping variables. Otherwise a set of additional grouping variables to calculate the kill rate.
<code>setter_position_by</code>	string: either "rotation" or "front_back"
<code>history_table</code>	list: (only if <code>attack_options</code> is "use_history") the object returned by <code>ov_create_history_table()</code>
<code>attack_by</code>	string: either "code", "zone", "tempo", "setter call", "attacker_name", "player_role"
<code>exclude_attacks</code>	character: vector of attack codes to exclude
<code>rotation</code>	string: (only relevant when <code>attack_by</code> is "player_role") either "SHM" (assume a setter-hitter-middle rotation order), or "SMH" (setter-middle-hitter)
<code>shiny_progress</code>	numeric: an optional two-element vector. If not NULL or NA, <code>shiny::setProgress()</code> calls will be made during simulation with values in this range

See Also

`ov_create_history_table()`

Examples

```
dvw <- ovdata_example("NCA-CUB")
system.time({
  ssd <- ov_simulate_setter_distribution(dvw = dvw, play_phase = "Reception",
                                         n_sim = 100, attack_by = "setter call",
                                         setter_position_by = "front_back", filter_sim = TRUE)
})
```

ov_sort_attack_codes	<i>Sort DataVolley attack codes</i>
----------------------	-------------------------------------

Description

Sort DataVolley attack codes

Usage

```
ov_sort_attack_codes(ac, by = "XV", na.last = NA)
```

Arguments

ac	character: character vector of attack codes to sort
by	string: method to use, currently only "XV" (any other value will default back to using sort without modification). "XV" will place X and V codes first (in numerical order, with each X preceding its matching V) then everything else in alphabetical order after that
na.last	logical: passed to sort

Value

Sorted character vector

Examples

```
ov_sort_attack_codes(c("V5", "V1", "X6", "CF", "X5"))

## Not run:
## sorting might be useful for controlling the plot order when facetting
## a `ggplot` by attack code
mydata$attack_code <- factor(mydata$attack_code,
  levels = ov_sort_attack_codes(unique(na.omit(mydata$attack_code))))
ggplot(mydata, ...) + facet_wrap(~attack_code)

## End(Not run)
```

ov_table_mssd	<i>Table of a simulated multi-game setter distribution sequence</i>
---------------	---

Description

Table of a simulated multi-game setter distribution sequence

Usage

```
ov_table_mssd(
  mssd,
  label_setters_by = "name",
  team = NULL,
  nrows = 50,
  groupBy = TRUE
)
```

Arguments

<code>mssd</code>	simulated multi-game setter distribution output as returned by <code>ov_simulate_multiple_setter_distribution</code>
<code>label_setters_by</code>	string: either "id" or "name"
<code>team</code>	NULL or string: if non-NULL, show sequence just for this team name
<code>nrows</code>	integer: number of rows per page in the table
<code>groupBy</code>	boolean: if TRUE, will group the rows by Opponent

Examples

```
## Not run:
list_dv <- list(dv_read(ovdata_example("NCA-CUB"))) # would normally be multiple games
mssd <- ov_simulate_multiple_setter_distribution(list_dv = list_dv,
  play_phase = c("Reception", "Transition"), attack_by = "attacker_name",
  n_sim = 100, setter_position_by = "front_back")

res <- ov_table_mssd(mssd, team = "NICARAGUA")

## End(Not run)
```

 ovlytics

 ovlytics

Description

Analytical functions for volleyball analytics, to be used in conjunction with the `datavolley` and `peranavolley` packages.

Author(s)

Maintainer: Ben Raymond <ben@untan.gl>

Authors:

- Adrien Ickowicz

Other contributors:

- openvolley.org [originator]

See Also

Useful links:

- <https://ovlytics.openvolley.org>
- <https://github.com/openvolley/ovlytics>
- Report bugs at <https://github.com/openvolley/ovlytics/issues>

Index

attack_eff, 2

datavolley::dv_read(), 13, 19, 21, 23
datavolley::read_dv(), 4, 6, 8, 20

MASS::kde2d(), 11

ov_aev, 3
ov_aev(), 6
ov_aev_reference_props, 5
ov_aev_reference_props(), 4
ov_augment_liberos, 6
ov_augment_plays, 7
ov_create_history_table, 9
ov_create_history_table(), 15, 18, 21, 23
ov_example_file, 10
ov_heatmap_kde, 11
ov_infer_player_roles, 12
ov_plot_distribution, 14
ov_plot_history_table, 15
ov_plot_rate, 15
ov_plot_sequence_distribution, 16
ov_plot_ssd, 17
ov_print_history_table, 17
ov_print_rate_table, 18
ov_season_table, 19
ov_setter_repetition, 19
ov_simulate_multiple_setter_distribution,
21
ov_simulate_multiple_setter_distribution(),
25
ov_simulate_setter_distribution, 22
ov_simulate_setter_distribution(),
14–18, 22
ov_sort_attack_codes, 24
ov_table_mssd, 24
ovlytics, 25
ovlytics-package (ovlytics), 25

peranavolley::pv_read(), 4, 6, 8, 20

reception_eff (attack_eff), 2

serve_eff (attack_eff), 2
shiny::setProgress(), 22, 23
sort, 24
strwrap(), 14, 16