Package 'BAwiR'

February 27, 2026

Type Package

Title Analysis of Basketball Data

Version 1.5

Date 2026-02-27

Description Collection of tools to work with European basketball data. Functions available are related to friendly
web scraping, data management and visualization. Data were obtained from <https://www.euroleaguebasketball.net/euroleague/>,
<https://www.euroleaguebasketball.net/eurocup/> and <https://www.acb.com/>, following the instructions
of their respectives robots.txt files, when available. Box score data are available for the three leagues.
Play-by-play and spatial shooting data are also available for the Spanish league. Methods for analysis include a
population pyramid, 2D plots, circular plots of players' percentiles, plots of players' monthly/yearly stats,
team heatmaps, team shooting plots, team four factors plots, cross-tables with the results of regular season games,
maps of nationalities, combinations of lineups, possessions-related variables, timeouts,
performance by periods, personal fouls, offensive rebounds and different types of shooting charts.
Please see Vinue (2020) <doi:10.1089/big.2018.0124> and Vinue (2024) <doi:10.1089/big.2023.0177>.

License GPL (>= 2)

URL https://www.uv.es/vivigui/basketball_platform.html,
https://www.uv.es/vivigui/, https://www.R-project.org

Depends R (>= 3.5.0)

Imports plyr, dplyr, ggplot2, ggpubr, grid, httr, janitor, jsonlite,
lubridate, magrittr, polite, purrr, reshape2, robotstxt, rvest,
stringi, stringr, tibble, tidyr, xml2

Suggests Anthropometry, cowplot, ggimage, ggrepel, ggtext, ggupset,
knitr, markdown, packageRank, png, qdapRegex, RCurl, readr,
rmarkdown, rworldmap, scales

VignetteBuilder knitr

LazyData true

RoxygenNote 7.3.2

Encoding UTF-8

NeedsCompilation no

Author Guillermo Vinue [aut, cre]

Maintainer Guillermo Vinue <`guillermo.vinue@uv.es`>

Repository CRAN

Date/Publication 2026-02-27 16:52:10 UTC

Contents

`acb_age_profile_data_2526` *ACB age profile, 2025-2026, Valencia Basket*

Description

Roster of the ACB team Valencia Basket for the 2025-2026 season and its players' age profiles.

Usage

```
acb_age_profile_data_2526
```

Format

Data frame with 15 rows and 7 columns.

Source

`https://www.acb.com/`

`acb_combs` *Lineups in a given game.*

Description

Data set for illustrative purposes with the Unicaja five-men and one-man lineups in the game 104459 from the ACB 2025-2026 regular season.

Usage

```
acb_combs
```

Format

Data frame with 150 rows and 38 columns.

Source

`https://www.acb.com/`

`acb_games_1718` *ACB games 2017-2018*

Description

Games of the first seventeen days of the ACB 2017-2018 season.

Usage

```
acb_games_1718
```

Format

Data frame with 3939 rows and 38 columns.

Source

`https://www.acb.com/`

`acb_games_2223_coach` *ACB coaches in the 2022-2023 season.*

Description

Coach for each team in all the games of the ACB 2022-2023 season.

Usage

```
acb_games_2223_coach
```

Format

Data frame with 612 rows and 4 columns.

Note

The **game_code** column allows us to detect the source website, for example, `https://live.acb.com/es/partidos/103389/jugadas`.

Source

`https://www.acb.com/`

`acb_games_2223_info` *ACB games 2022-2023, days and codes.*

Description

Game codes, games and days from the ACB 2022-2023 season.

Usage

```
acb_games_2223_info
```

Format

Data frame with 306 rows and 3 columns.

Note

The **game_code** column allows us to detect the source website, for example, `https://live.acb.com/es/partidos/103389/jugadas`.

Source

`https://www.acb.com/`

`acb_players_1718` *ACB players 2017-2018*

Description

Players corresponding to the games of the first seventeen days of the ACB 2017-2018 season.

Usage

```
acb_players_1718
```

Format

Data frame with 255 rows and 7 columns.

Source

`https://www.acb.com/`

`acb_players_2425` *ACB players 2024-2025*

Description

Player unique identifiers of a sample of 30 players from the ACB 2024-2025 season.

Usage

```
acb_players_2425
```

Format

Data frame with 30 rows and 5 columns.

Source

https://www.acb.com/

`acb_shields` *Shields of the ACB teams*

Description

Links to the official shields of the ACB teams.

Usage

```
acb_shields
```

Format

Data frame with 20 rows and 2 columns.

Source

https://www.acb.com/

`acb_shooting_data_2425` *ACB shooting data, 2024-2025*

Description

Spatial shooting data from a sample of 30 players from the ACB 2024-2025 season. See also `acb_players_2425`.

Usage

```
acb_shooting_data_2425
```

Format

Data frame with 4277 rows and 23 columns.

Source

https://www.acb.com/

`acb_sticker_data_2526` *ACB sticker data, 2025-2026*

Description

This data frame contains an example of the usage and context statistics for the Real Madrid player Alberto Abalde to illustrate the creation of the sticker plot.

Usage

```
acb_sticker_data_2526
```

Format

Data frame with 11 rows and 9 columns.

Source

https://www.acb.com/

acb_usage_act_data_2526 *ACB usage action data, 2025-2026*

Description

Sample of possessions played and ended for a game from the ACB 2025-2026 season and the particular number of actions used by the player to end the possession.

Usage

```
acb_usage_act_data_2526
```

Format

Data frame with 75 rows and 8 columns.

Source

https://www.acb.com/

acb_usage_data_2526 *ACB usage data, 2025-2026*

Description

Sample of possessions played and ended for a game from the ACB 2025-2026 season. For each player we have the number of possessions he played and the ones he ended.

Usage

```
acb_usage_data_2526
```

Format

Data frame with 62 rows and 9 columns.

Source

https://www.acb.com/

acb_vbc_cz_pbp_2223 *ACB play-by-play data, 2022-2023, Valencia Basket-Casademont Zaragoza*

Description

Play-by-play data from the game Valencia Basket-Casademont Zaragoza from the ACB 2022-2023 season.

Usage

```
acb_vbc_cz_pbp_2223
```

Format

Data frame with 466 rows and 9 columns.

Note

Actions are given in Spanish. A bilingual basketball vocabulary (Spanish/English) is provided in `https://www.uv.es/vivigui/docs/basketball_dictionary.xlsx`. The **game_code** column allows us to detect the source website, namely, `https://live.acb.com/es/partidos/103389/jugadas`.

Source

`https://www.acb.com/`

acb_vbc_cz_sl_2223 *ACB starting lineups, 2022-2023, Valencia Basket-Casademont Zaragoza*

Description

Starting lineups in each period from the game Valencia Basket-Casademont Zaragoza from the ACB 2022-2023 season.

Usage

```
acb_vbc_cz_sl_2223
```

Format

Data frame with 40 rows and 9 columns.

Note

The **action** column refers to starting lineup (*Quinteto inicial*, in Spanish). The initial score in each period does not really matter for the creation of this data set. The **game_code** column allows us to detect the source website, for example, `https://live.acb.com/es/partidos/103389/jugadas`.

Source

`https://www.acb.com/`

`capit_two_words` *Capitalize two-word strings*

Description

Ancillary function to capitalize the first letter of both words in a two-word string. This can be used for example to capitalize the teams names for the plots title.

Usage

```
capit_two_words(two_word_string)
```

Arguments

`two_word_string`	Two-word string.

Value

Vector with the two words capitalized.

Author(s)

Guillermo Vinue

Examples

```
capit_two_words("valencia basket")
```

`do_add_adv_stats` *Advanced statistics*

Description

This function adds to the whole data frame the advanced statistics for every player in every game.

Usage

```
do_add_adv_stats(df)
```

Arguments

`df`	Data frame with the games and the players info.

Details

The advanced statistics computed are as follows:

- GameSc: Game Score.
- PIE: Player Impact Estimate.
- EFGPerc: Effective Field Goal Percentage.
- ThreeRate: Three points attempted regarding the total field goals attempted.
- FRate: Free Throws made regarding the total field goals attempted.
- STL_TOV: Steal to Turnover Ratio.
- AST_TOV: Assist to Turnover Ratio.
- PPS: Points Per Shot.
- OE: Offensive Efficiency.
- EPS: Efficient Points Scored.

The detailed definition of some of these stats can be found at `https://www.basketball-reference.com/about/glossary.html` and `https://www.nba.com/stats/help/glossary/`.

Value

Data frame.

Author(s)

Guillermo Vinue

See Also

`do_OE`, `do_EPS`

Examples

```
df <- do_join_games_bio("ACB", acb_games_1718, acb_players_1718)

df1 <- do_add_adv_stats(df)
```

`do_best_zones` *Best players by zone*

Description

Creates a visualization of the players who shoot little and score a lot in several zones at the same time.

Usage

```
do_best_zones(data_best_archetypoid)
```

Arguments

`data_best_archetypoid`	Best players by zone computed with the archetypoid algorithm.

Value

A plot.

Author(s)

Guillermo Vinue

See Also

archetypoids

Examples

```
## Not run:
library(dplyr)
library(Anthropometry)

zones_court <- metrics_player_zone %>%
  distinct(location) %>%
  pull()

numArch <- 10
numRep <- 20
numArchoid <- 2 # Number of archetypoids.
```

```
data_arch <- data.frame()

# Run the algorithm for each zone one by one and save the archetypoid
# with least shots and highest percentage.
i <- 1

zone <- metrics_player_zone %>%
  filter(location == zones_court[i]) %>%
  select(-pps_player)

zone_num <- zone %>%
  select(total, perc_player)

lass <- stepArchetypesRawData(data = zone_num, numArch = 1:numArch,
                              numRep = numRep, verbose = FALSE)

res_ns <- archetypoids(numArchoid, zone_num, huge = 200, step = FALSE,
                       ArchObj = lass, nearest = "cand_ns",sequ = TRUE)
zone[res_ns$cases, ]

# Here [1, ] indicates the archetypoid of interest. Change it accordingly.
# Here 4 indicates the number of similar players to the archetypoid. Change it accordingly.
arch_targ <- zone[order(res_ns$alphas[1, ], decreasing = TRUE)[1:4], ]
data_arch <- rbind(data_arch, arch_targ)

i <- 2

zone <- metrics_player_zone %>%
  filter(location == zones_court[i]) %>%
  select(-pps_player)

zone_num <- zone %>%
  select(total, perc_player)

lass <- stepArchetypesRawData(data = zone_num, numArch = 1:numArch,
                              numRep = numRep, verbose = FALSE)

res_ns <- archetypoids(numArchoid, zone_num, huge = 200, step = FALSE,
                       ArchObj = lass, nearest = "cand_ns",sequ = TRUE)

arch_targ <- zone[order(res_ns$alphas[2, ], decreasing = TRUE)[1:10], ]
data_arch <- rbind(data_arch, arch_targ)

do_best_zones(data_arch)

## End(Not run)
```

do_clutch_time *Get games with clutch time*

Description

Obtain the games that have clutch time. The clutch time is generally defined as the game situation when the scoring margin is within 5 points with five or fewer minutes remaining in a game. In order to allow more clutch situations, the user will be able to set the minutes left and the scoring difference.

Usage

```
do_clutch_time(data, min_left = "05:00", score_diff = 5)
```

Arguments

`data`	Source play-by-play data.
`min_left`	Minutes left to finish the game. Default 5 minutes.
`score_diff`	Scoring difference. Default 5 points.

Value

Data frame of the game that has clutch time.

Author(s)

Guillermo Vinue

Examples

```
df0 <- do_clutch_time(acb_vbc_cz_pbp_2223)
#df0 # If no rows, that means that the game did not have clutch time.
```

`do_divide_court_zones` *Zones of the basketball court*

Description

Divide the basketball court into 10 zones.

Usage

```
do_divide_court_zones(data_shots)
```

Arguments

`data_shots`	Shooting data frame.

Value

The shooting data frame with a new column called "location" indicating the zone from which each shot was taken.

Author(s)

Guillermo Vinue

Examples

```
## Not run:
do_divide_court_zones(acb_shooting_data_2425)

## End(Not run)
```

---

do_EPS *Efficient Points Scored (EPS)*

Description

A limitation of `do_OE` is that it doesn't rely on the quantity of the player's offense production, that's to say, whether the player provides a lot of offense or not. In addition, it does not give credit for free-throws. An extension of `do_OE` has been defined: the Efficient Points Scored (EPS), which is the result of the product of OE and points scored. Points scored counts free-throws, two-point and three-point field goals. A factor F is also added to put the adjusted total points on a points scored scale. With the factor F, the sum of the EPS scores for all players in a given season is equal to the sum of the league total points scored in that season.

Usage

```
do_EPS(df)
```

Arguments

`df`	Data frame with the games and the players info.

Value

EPS values.

Author(s)

Guillermo Vinue

References

Shea, S., Baker, C., (2013). Basketball Analytics: Objective and Efficient Strategies for Understanding How Teams Win. Lake St. Louis, MO: Advanced Metrics, LLC.

See Also

do_OE, do_add_adv_stats

Examples

```
df <- do_join_games_bio("ACB", acb_games_1718, acb_players_1718)
df1 <- do_add_adv_stats(df)
do_EPS(df1)[1]
```

---

do_filter_data *Filter shooting data*

Description

Filter the shooting data with the team or player of interest, and also by periods, minutes and game place. If neither team nor player is given, the data from the whole league is used.

Usage

```
do_filter_data(data_shots_zones, season_str, team, period, minute_vect, place, player)
```

Arguments

data_shots_zones	Shooting data with the court zones.
season_str	String with the season.
team	String with the team's full name. Nothing to filter if "".
period	Number with the periods (1, 2, 3 and 4 for the common four quarters, 5 for the first overtime and 6 for the second overtime). Nothing to filter if "".
minute_vect	Vector with the minutes to filter by. Nothing to filter if "".
place	String. If "Home" or "Casa", the local games are filtered. Nothing to filter if "".
player	String with the player's name. Nothing to filter if "".

Value

A data frame with the filters applied.

Author(s)

Guillermo Vinue

See Also

do_divide_court_zones

Examples

```
## Not run:
df0 <- do_divide_court_zones(acb_shooting_data_2425)

# Data for the whole league:
df1 <- do_filter_data(df0, "2024-2025", "", "", "", "", "")

# Data for a team:
df1 <- do_filter_data(df0, "2024-2025", "UCAM Murcia", "", "", "", "")

# Data for a player:
df1 <- do_filter_data(df0, "2024-2025", "", "", "", "", "D. Ennis")

# Other filters:
# By minutes:
df1 <- do_filter_data(df0, "2024-2025", "", "", c(8,10), "", "D. Ennis")

## End(Not run)
```

`do_four_factors_df` *Four factors for teams*

Description

This function computes team's offense and defense four factors.

The four factors are the effective field goal percentage (EFGP), the turnover percentage (TOVP), the offensive rebound percentage (ORBP) and the free throws rate (FTRate). They are well defined at `http://www.rawbw.com/~deano/articles/20040601_roboscout.htm` and `https://www.basketball-reference.com/about/factors.html`.

As a summary, EFGP is a measure of shooting efficiency; TOVP is the percentage of possessions where the team missed the ball, see `https://www.nba.com/thunder/news/stats101.html` to read about the 0.44 coefficient; ORBP measures how many rebounds were offensive from the total of available rebounds, and FTRate is a measure of how often a team gets to the line.

Usage

```
do_four_factors_df(df_games, teams, data_team_reb_tov)
```

Arguments

`df_games`	Data frame with the games, players info, advanced stats and eventually recoded teams names.
`teams`	Teams names.
`data_team_reb_tov`	Additional data with rebounds and turnovers directly assigned to teams. Nothing to do if NULL.

Details

Instead of defining the Offensive and Defensive Rebound Percentage as mentioned in the previous links, I have computed just the Offensive Rebound Percentage for the team and for its rivals. This makes easier to have four facets, one per factor, in the ggplot.

In order to establish the team rankings, we have to consider these facts: In defense (accumulated statistics of the opponent teams to the team of interest), the best team in each factor is the one that allows the smallest EFGP, the biggest TOVP, the smallest ORBP and the smallest FTRate, respectively.

In offense (accumulated statistics of the team of interest), the best team in each factor is the one that has the biggest EFGP, the smallest TOVP, the biggest ORBP and the biggest FTRate, respectively.

Value

A list with two data frames, `df_rank` and `df_no_rank`. Both have the same columns:

- Team: Team name.
- Type: Either Defense or Offense.
- EFGP, ORBP, TOVP and FTRate.

The `df_rank` data frame contains the team ranking label for each statistic between parentheses. Therefore, `df_no_rank` is used to create the ggplot with the numerical values and `df_rank` is used to add the ranking labels.

Author(s)

Guillermo Vinue

See Also

`get_four_factors_plot`

Examples

```
df <- do_join_games_bio("ACB", acb_games_1718, acb_players_1718)

df1 <- do_add_adv_stats(df)

# When only one team is selected the rankings between parentheses
# do not reflect the real rankings regarding all the league teams.
# The rankings are computed with respect to the number of teams
# passed as an argument.
df_four_factors <- do_four_factors_df(df1, "Valencia", NULL)
```

do_four_factors_df_lineup

Four factors for lineups

Description

This function computes offense and defense four factors for a given lineup. The four factors are the effective field goal percentage (EFGP), the turnover percentage (TOVP), the offensive rebound percentage (ORBP) and the free throws rate (FTRate). They are well defined at `http://www.rawbw.com/~deano/articles/20040601_roboscout.htm` and `https://www.basketball-reference.com/about/factors.html`.

As a summary, EFGP is a measure of shooting efficiency; TOVP is the percentage of possessions where the team missed the ball; ORBP measures how many rebounds were offensive from the total of available rebounds, and FTRate is a measure of how often a team gets to the line.

Usage

```
do_four_factors_df_lineup(data_combs, team_name, type_lineup, type_period, type_opponent)
```

Arguments

`data_combs`	Data frame with all the combinations of lineups.
`team_name`	Name of the team.
`type_lineup`	Type of lineups to analyze. Options are 'quintet', 'quartet', 'trio', 'duo' and 'single'.
`type_period`	Period of interest. Options are xC, where x starts from 1. Common periods are from 1 to 4. Overtimes are labeled with the next numbers, such as 5C for the first overtime and 6C for the second one.
`type_opponent`	Name of the opponent teams.

Details

Instead of defining the Offensive and Defensive Rebound Percentage as mentioned in the previous links, I have computed just the Offensive Rebound Percentage for the team and for its rivals. This makes easier to have four facets, one per factor, in the ggplot.

Value

A data frame with the four factors in defense and in offense for a given lineup.

Author(s)

Guillermo Vinue

See Also

`do_four_factors_df`

Examples

```
## Not run:
do_four_factors_df_lineup(acb_combs, "Unicaja", "quintet", NULL, NULL)

## End(Not run)
```

`do_ft_fouls` *Compute free throw fouls*

Description

Compute how many 1-,2- and 3-free throw fouls has committed or received every player.

Usage

```
do_ft_fouls(data, type)
```

Arguments

`data`	Play-by-play data.
`type`	Either 'comm' (for committed) or 'rec' (for received).

Value

Data frame with the following columns:

team: Name of the team. **player**: Name of the player. **n_ft_fouls_x**: Number of free throw fouls committed or received. **n_ft_x**: Number of free throws given or got. **n_ft_char**: Type of free throw. Options can be 1TL, 2TL and 3TL. **n**: Number of free throws of each type.

Author(s)

Guillermo Vinue

Examples

```
df01 <- do_ft_fouls(acb_vbc_cz_pbp_2223, "comm")
#df01

df02 <- do_ft_fouls(acb_vbc_cz_pbp_2223, "rec")
#df02
```

`do_join_games_bio` *Join games and players' info*

Description

This function calls the needed ancillary functions to join the games played by all the players in the desired competition (currently ACB, Euroleague and Eurocup) with their personal details.

Usage

```
do_join_games_bio(competition, df_games, df_rosters)
```

Arguments

`competition`	String. Options are "ACB", "Euroleague" and "Eurocup".
`df_games`	Data frame with the games.
`df_rosters`	Data frame with the biography of the roster players.

Value

Data frame.

Author(s)

Guillermo Vinue

See Also

`join_players_bio_age_acb`, `join_players_bio_age_euro`

Examples

```
df <- do_join_games_bio("ACB", acb_games_1718, acb_players_1718)
```

`do_lineup` *Compute ACB lineups*

Description

Compute all the lineups that a given team shows during a game.

Usage

```
do_lineup(data, day_num, game_code, team_sel, verbose)
```

Arguments

`data`	Play-by-play prepared data from a given game.
`day_num`	Day number.
`game_code`	Game code.
`team_sel`	One of the teams' names involved in the game.
`verbose`	Logical to indicate if the information of the computations must be provided.

Value

Data frame. Each row is a different lineup. This is the meaning of the columns that might not be explanatory by themselves:

> **team_in**: Time point when that lineup starts playing together. **team_out**: Time point when that lineup stops playing together (because there is a substitution). **num_players**: Number of players forming the lineup (must be 5 in this case). **time_seconds**: Total of seconds that the lineup played. **diff_points**: Game score in the time that the lineup played. **plus_minus**: Plus/minus achieved by the lineup. This is the difference between the game score of the previous lineup and of the current one. **plus_minus_poss**: Plus/minus per possession.

Note

A possession lasts 24 seconds in the ACB league.

Author(s)

Guillermo Vinue

Examples

```
library(dplyr)
df0 <- acb_vbc_cz_pbp_2223

day_num <- unique(acb_vbc_cz_pbp_2223$day)
game_code <- unique(acb_vbc_cz_pbp_2223$game_code)

# Starting players:
acb_games_2223_sl <- acb_vbc_cz_sl_2223 %>%
  filter(period == "1C")

# Prepare data:
df1 <- do_prepare_data(df0, day_num,
                       acb_games_2223_sl, acb_games_2223_info,
                       game_code)

df2 <- do_lineup(df1, day_num, game_code, "Valencia Basket", FALSE)
```

do_lineup_metrics *Lineups-related information*

Description

For every lineup, compute the number of possessions, points, assists, turnovers, field goals, rebounds and assisted field goals, both in defense and offense.

Usage

```
do_lineup_metrics(data_possess, team_sel, team_opp)
```

Arguments

data_possess	Play-by-play data with the start of possessions.
team_sel	One of the teams involved in the game.
team_opp	Opponent team.

Value

Data frame. Each row is a different lineup.

Author(s)

Guillermo Vinue

Examples

```
## Not run:
library(dplyr)
df0 <- acb_vbc_cz_pbp_2223

day_num <- unique(acb_vbc_cz_pbp_2223$day)
game_code <- unique(acb_vbc_cz_pbp_2223$game_code)

# Starting players:
acb_games_2223_sl <- acb_vbc_cz_sl_2223 %>%
  dplyr::filter(period == "1C")

# Prepare data:
df1 <- do_prepare_data(df0, day_num,
                       acb_games_2223_sl, acb_games_2223_info,
                       game_code)

teams_game <- sort(unique(df1$team))
team_sel <- teams_game[1]

data <- df1
data <- data %>%
  mutate(row_num = row_number()) %>%
  mutate(time_point = ifelse(nchar(time_point) < 5, paste0("0", time_point), time_point))

# Filter by team:
data1 <- data %>%
  filter(team == team_sel)

# Set also the opponent team:
team_opp <- setdiff(unique(data$team), team_sel)

# Add the last row of games' data to have the real final
# game score in case it is not available:
last_row_game <- data[nrow(data),]

last_row_game$time_point <- "00:00"
last_row_game$player <- NA
last_row_game$action <- NA
last_row_game$team <- team_sel

data1 <- bind_rows(data1, last_row_game)

# Get players out:
pl_out <- c(1, which(data1$action == "Sale de la pista"), nrow(data1))

i <- 1
data2 <- data1 %>%
  slice(pl_out[i]:pl_out[i + 1])

nr <- nrow(data2)

# Lineup:
lineup <- data2 %>%
  filter(action != "Sale de la pista") %>%
  # Avoid actions that are assigned to teams:
  filter(player != team_sel) %>%
  distinct(player) %>%
  pull()

# Identify when the possessions start:
data2_rival <- data %>%
  filter(team == team_opp) %>%
  filter(between(row_num, data2$row_num[1], data2$row_num[nr]))

data3 <- rbind(data2, data2_rival) %>%
  arrange(row_num) %>%
  na.omit()

data4 <- do_possession(data3, NULL, "10:00")

data4[data4$action == "Mate", "action"] <- "Tiro de 2 anotado"

# Obtain metrics for the lineup:
data5 <- do_lineup_metrics(data4, team_sel, team_opp)
```

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

`do_map_nats` *Data frame for the nationalities map*

Description

This function prepares the data frame with the nationalities to be mapped with `get_map_nats`. It is used inside it.

Usage

```
do_map_nats(df_stats)
```

Arguments

`df_stats`	Data frame with the statistics and the corrected nationalities.

Value

List with the following elements:

- df_all: Data frame with each country, its latitudes and longitudes and whether it must be coloured or not (depending on if there are players from that country).
- countr_num: Vector with the countries from where there are players and the number of them.
- leng: Number of countries in the world.

Author(s)

Guillermo Vinue

See Also

`get_map_nats`

do_OE *Offensive Efficiency (OE)*

Description

Offensive Efficiency (OE) is a measure to evaluate the quality of offense produced. OE counts the total number of successful offensive possessions the player was involved in, regarding the player's total number of potential ends of possession.

This measure is used in the definition of `do_EPS`.

Usage

```
do_OE(df)
```

Arguments

`df`	Data frame with the games and the players info.

Value

OE values.

Note

When either both the numerator and denominator of the OE expression are 0 or just the denominator is 0, the function returns a 0.

Author(s)

Guillermo Vinue

References

Shea, S., Baker, C., (2013). Basketball Analytics: Objective and Efficient Strategies for Understanding How Teams Win. Lake St. Louis, MO: Advanced Metrics, LLC.

See Also

`do_EPS`, `do_add_adv_stats`

Examples

```
df <- do_join_games_bio("ACB", acb_games_1718, acb_players_1718)

df1 <- do_add_adv_stats(df)

# Players with OE = 0:
# df1[55, c("Player.x", "FG", "AST", "FGA", "ORB", "TOV")]
# Player.x     FG  AST  FGA  ORB  TOV
# Triguero, J.  0    0    0    0    0

# OE can be greater than 1, for example:
# df1[17, c("Player.x", "FG", "AST", "FGA", "ORB", "TOV")]
# Player.x        FG  AST  FGA  ORB  TOV
# Diagne, Moussa   3    0    3    1    0

do_OE(df1[1,])
```

`do_offensive_fouls` *Compute offensive fouls*

Description

Compute how many offensive fouls has committed or received every player.

Usage

```
do_offensive_fouls(data, type)
```

Arguments

`data`	Play-by-play data.
`type`	Either 'comm' (for committed) or 'rec' (for received).

Value

Data frame with the following columns:

team: Name of the team. **player**: Name of the player. **n_offensive_fouls_x**: Number of offensive fouls.

Author(s)

Guillermo Vinue

Examples

```
df01 <- do_offensive_fouls(acb_vbc_cz_pbp_2223, "comm")
#df01

df02 <- do_offensive_fouls(acb_vbc_cz_pbp_2223, "rec")
#df02
```

`do_possession` *Compute when possessions start*

Description

Compute when the possession starts for each team in a game.

Usage

```
do_possession(data, period_sel, time_point_start)
```

Arguments

`data`	Play-by-play prepared data from a given game.
`period_sel`	Period of interest. Options can be "xC", where x=1,2,.... If NULL, no filtering is done.
`time_point_start`	Starting time point of the lineup.

Value

Data frame. This is the meaning of the columns that might not be explanatory by themselves:

time_start: Time point when the action starts. **time_end**: Time point when the action ends. **poss_time**: Duration of the possession. **possession**: Indicates when the possession starts. This is encoded with the Spanish word *inicio* (*start*, in English). **points**: Number of points scored from a given action.

Note

1. A possession lasts 24 seconds in the ACB league.

2. Actions are given in Spanish. A bilingual basketball vocabulary (Spanish/English) is provided in `https://www.uv.es/vivigui/docs/basketball_dictionary.xlsx`.

3. The **game_code** column allows us to detect the source website, for example, `https://live.acb.com/es/partidos/103389/jugadas`.

Author(s)

Guillermo Vinue

Examples

```
## Not run:
library(dplyr)
df0 <- acb_vbc_cz_pbp_2223

day_num <- unique(acb_vbc_cz_pbp_2223$day)
game_code <- unique(acb_vbc_cz_pbp_2223$game_code)
```

```
# Starting players:
acb_games_2223_sl <- acb_vbc_cz_sl_2223 %>%
  dplyr::filter(period == "1C")

# Prepare data:
df1 <- do_prepare_data(df0, day_num,
                       acb_games_2223_sl, acb_games_2223_info,
                       game_code)

teams_game <- sort(unique(df1$team))
team_sel <- teams_game[1]

data <- df1
data <- data %>%
  mutate(row_num = row_number()) %>%
  mutate(time_point = ifelse(nchar(time_point) < 5, paste0("0", time_point), time_point))

# Filter by team:
data1 <- data %>%
  filter(team == team_sel)

# Set also the opponent team:
team_opp <- setdiff(unique(data$team), team_sel)

# Add the last row of games' data to have the real final
# game score in case it is not available:
last_row_game <- data[nrow(data),]

last_row_game$time_point <- "00:00"
last_row_game$player <- NA
last_row_game$action <- NA
last_row_game$team <- team_sel

data1 <- bind_rows(data1, last_row_game)

# Get players out:
pl_out <- c(1, which(data1$action == "Sale de la pista"), nrow(data1))

i <- 1
data2 <- data1 %>%
  slice(pl_out[i]:pl_out[i + 1])

nr <- nrow(data2)

# Lineup:
lineup <- data2 %>%
  filter(action != "Sale de la pista") %>%
  # Avoid actions that are assigned to teams:
  filter(player != team_sel) %>%
  distinct(player) %>%
  pull()
```

```
# Identify when the possessions start:
data2_rival <- data %>%
  filter(team == team_opp) %>%
  filter(between(row_num, data2$row_num[1], data2$row_num[nr]))

data3 <- rbind(data2, data2_rival) %>%
  arrange(row_num) %>%
  na.omit()

data4 <- do_possession(data3, NULL, "10:00")

## End(Not run)
```

do_possession_stats *Possessions-related statistics for teams*

Description

Compute the possessions-related statistics for teams. These statistics are offensive rating, defensive rating, net rating, pace and number of possessions.

Usage

```
do_possession_stats(data_possess, season = "2025-2026")
```

Arguments

`data_possess`	Data frame with the beginning of each possession obtained with `do_possession`.
`season`	Season string.

Details

See `https://www.basketball-reference.com/about/glossary.html` for formulas and explanations.

Both teams in the same game share the same pace. Pace reflects the tempo of the game itself, not just one team's style. Over many games, a team's average pace reflects how fast they usually play, but any individual game's pace is shared with their opponent.

Value

A data frame with the possessions statistics for each team.

Author(s)

Guillermo Vinue

See Also

do_possession

Examples

```
## Not run:
library(dplyr)
df0 <- acb_vbc_cz_pbp_2223

day_num <- unique(acb_vbc_cz_pbp_2223$day)
game_code <- unique(acb_vbc_cz_pbp_2223$game_code)

# Starting players:
acb_games_2223_sl <- acb_vbc_cz_sl_2223 %>%
  dplyr::filter(period == "1C")

# Prepare data:
df1 <- do_prepare_data(df0, day_num,
                       acb_games_2223_sl, acb_games_2223_info,
                       game_code)

teams_game <- sort(unique(df1$team))

# Study the lineups:
data_res <- do_study_lineup(df1, day_num, game_code_num, teams_game[1], FALSE)
data_all_posse <- data_res$data_pos %>% select(-row_num)

# Statistics of possessions:
data_poss_st <- do_possession_stats(data_all_posse, "2022-2023")

## End(Not run)
```

do_possession_stats_lineup

Possessions-related statistics for lineups

Description

Compute the possessions-related statistics for lineups. These statistics are offensive rating, defensive rating, net rating, pace and number of possessions.

Usage

```
do_possession_stats_lineup(data_combs, team_name, type_lineup, type_period, type_opponent,
                              cols_group = c("team", "lineup"))
```

Arguments

`data_combs`	Data frame with all the combinations of lineups.
`team_name`	Name of the team.
`type_lineup`	Type of lineups to analyze. Options are 'quintet', 'quartet', 'trio', 'duo' and 'single'.
`type_period`	Period of interest. Options are xC, where x starts from 1. Common periods are from 1 to 4. Overtimes are labeled with the next numbers, such as 5C for the first overtime and 6C for the second one. Nothing to do if NULL.
`type_opponent`	Name of the opponent teams. Nothing to do if NULL.
`cols_group`	Group of columns to apply the computations. Default is c("team", "lineup") to compute the metrics just for the players on court. To compute them for the players both on and off court, use c("team", "lineup", "status").

Details

See `https://www.basketball-reference.com/about/glossary.html` for formulas and explanations.

Value

A data frame with the possessions statistics for each lineup.

Author(s)

Guillermo Vinue

See Also

`do_possession_stats`

Examples

```
## Not run:
do_possession_stats_lineup(acb_combs, "Unicaja", "quintet", NULL, NULL)

## End(Not run)
```

`do_prepare_data` *Prepare ACB play-by-play data*

Description

Prepare the ACB play-by-play data to be analyzed in further steps. It involves correcting some inconsistencies and filtering some unnecessary information.

Usage

```
do_prepare_data(data, day_num, data_gsl, data_ginfo, game_code_excel)
```

Arguments

`data`	Source play-by-play data from a given game.
`day_num`	Day number.
`data_gsl`	Games' starting lineups.
`data_ginfo`	Games' basic information.
`game_code_excel`	Game code.

Value

Data frame. Each row represents the action happened in the game. It has associated a player, a time point and the game score. The **team** column refers to the team to which the player belongs.

Note

1. Actions are given in Spanish. A bilingual basketball vocabulary (Spanish/English) is provided in https://www.uv.es/vivigui/docs/basketball_dictionary.xlsx.

2. The **game_code** column allows us to detect the source website, for example, https://live.acb.com/es/partidos/103389/jugadas.

Author(s)

Guillermo Vinue

Examples

```
library(dplyr)
df0 <- acb_vbc_cz_pbp_2223

day_num <- unique(acb_vbc_cz_pbp_2223$day)
game_code <- unique(acb_vbc_cz_pbp_2223$game_code)

acb_games_2223_sl <- acb_vbc_cz_sl_2223 %>%
  filter(period == "1C")

df1 <- do_prepare_data(df0, day_num,
                       acb_games_2223_sl, acb_games_2223_info,
                       game_code)
#df1
```

`do_prepare_data_gradient`
Prepare the data for the gradient shooting plots

Description

Prepare the data for the gradient shooting visualizations at a player level.

Usage

```
do_prepare_data_gradient(all_shots_pl, summary_shots_zone_pl, summary_shots_zone_league)
```

Arguments

`all_shots_pl`	Shooting data frame associated with the filters given to `do_shots_stats` for the player of interest.
`summary_shots_zone_pl`	Summary of the player's shots by zone.
`summary_shots_zone_league`	Summary of the league's shots by zone.

Value

A list with the following three elements:

- all_shots_comp_data: Summary of the shooting data of the player and of the league.
- all_shots_comp_viz: Summary of the shooting data prepared for the visualization.
- player: Player's name.

Author(s)

Guillermo Vinue

See Also

`do_divide_court_zones`, `do_shots_stats`

Examples

```
## Not run:
library(dplyr)

df0 <- do_divide_court_zones(acb_shooting_data_2425)

df1 <- do_filter_data(df0, "2024-2025", "", "", "", "", "")

# LEAGUE METRICS:
shots_stats <- do_shots_stats(df1, df0)
```

```
summary_shots_zone_lg <- shots_stats$summary_shots_zone

summary_shots_zone_league <- summary_shots_zone_lg %>%
  mutate(pps_league = ifelse(location_color == "2pt",
                             (2 * count) / total,
                             (3 * count) / total)) %>%
  select(location, perc_league = perc, pps_league)

# PLAYER METRICS:
df1 <- do_filter_data(df0, "2024-2025", "", "", "", "", "D. Ennis")

shots_stats <- do_shots_stats(df1, df0)

all_shots_pl <- shots_stats$all_shots

summary_shots_zone_pl <- shots_stats$summary_shots_zone

res_grad <- do_prepare_data_gradient(all_shots_pl, summary_shots_zone_pl, summary_shots_zone_league)

## End(Not run)
```

do_prepare_data_or *Prepare data for the offensive rebounds computation*

Description

The computation of the scoring after offensive rebounds requires a specifical data preparation. This function does this data processing.

Usage

```
do_prepare_data_or(data, rm_overtime, data_ginfo)
```

Arguments

`data`	Source play-by-play data from a given game.
`rm_overtime`	Logical. Decide to remove overtimes or not.
`data_ginfo`	Games' basic information. If NULL, nothing to add.

Value

Data frame. Each row represents the action happened in the game. The **points** column is added to transform the action that finished in scoring into numbers.

Note

1. Actions are given in Spanish. A bilingual basketball vocabulary (Spanish/English) is provided in https://www.uv.es/vivigui/docs/basketball_dictionary.xlsx.

2. The **game_code** column allows us to detect the source website, for example, https://live.acb.com/es/partidos/103389/jugadas.

Author(s)

Guillermo Vinue

See Also

do_reb_off_success

Examples

```
df0 <- acb_vbc_cz_pbp_2223

df1 <- do_prepare_data_or(df0, TRUE, acb_games_2223_info)
#df1
```

`do_prepare_data_to` *Prepare data for the timeouts computation*

Description

The computation of the successful timeouts requires a specific data preparation. This function does this data processing.

Usage

```
do_prepare_data_to(data, rm_overtime, data_ginfo, data_gcoach)
```

Arguments

data	Source play-by-play data from a given game.
rm_overtime	Logical. Decide to remove overtimes or not.
data_ginfo	Games' basic information. If NULL, nothing to do.
data_gcoach	Coach of each team in each day.

Value

Data frame. Each row represents the action happened in the game. The **team** column refers in this case both to the team to which the player belongs and the coach of that team. In addition, a **points** column is added to transform the action that finished in scoring into numbers .

Note

1. Actions are given in Spanish. A bilingual basketball vocabulary (Spanish/English) is provided in https://www.uv.es/vivigui/docs/basketball_dictionary.xlsx.
2. The **game_code** column allows us to detect the source website, for example, https://live.acb.com/es/partidos/103389/jugadas.

Author(s)

Guillermo Vinue

See Also

do_time_out_success

Examples

```
df0 <- acb_vbc_cz_pbp_2223

df1 <- do_prepare_data_to(df0, TRUE, acb_games_2223_info, acb_games_2223_coach)
#df1
```

---

do_preproc_period *Data preprocessing for periods*

Description

Preprocess the data that will be needed for computing statistics per period.

Usage

```
do_preproc_period(data, team_sel, period_sel, data_sl)
```

Arguments

data	Prepared data from a given game.
team_sel	One of the teams' names involved in the game.
period_sel	Period of interest. Options can be "xC", where x=1,2,3,4.
data_sl	Data with the starting lineups.

Author(s)

Guillermo Vinue

Examples

```
team_sel <- "Valencia Basket"
period_sel <- "1C"

pre_per <- do_preproc_period(acb_vbc_cz_pbp_2223, team_sel, period_sel, acb_vbc_cz_sl_2223)

df2 <- pre_per$df2
df0_inli_team <- pre_per$df0_inli_team
```

`do_process_acb_pbp` *Processing of the ACB website play-by-play data*

Description

This function disentangles the play-by-play data coming from the ACB website and creates a common data structure in R.

Usage

```
do_process_acb_pbp(game_elem, day, game_code, period, acb_shields, verbose)
```

Arguments

`game_elem`	Character with the tangled play-by-play data.
`day`	Day of the game.
`game_code`	Game code.
`period`	Period of the game.
`acb_shields`	Data frame with the links to the shields of the ACB teams.
`verbose`	Logical to display processing information.

Value

Data frame with eight columns:

- period: Period of the game.
- time_point: Time point when the basketball action happens.
- player: Player who performs the action.
- action: Basketball action.
- local_score: Local score at that time point.
- visitor_score: Visitor score at that time point.
- day: Day of the game.
- game_code: Game code.

Note

1. Actions are given in Spanish. A bilingual basketball vocabulary (Spanish/English) is provided in `https://www.uv.es/vivigui/docs/basketball_dictionary.xlsx`.

2. The **game_code** column allows us to detect the source website, for example, `https://live.acb.com/es/partidos/103389/jugadas`.

Author(s)

Guillermo Vinue

Examples

```
## Not run:
# Load packages required:
library(RSelenium)

# Provide the day and game code:
day <- "24"
game_code <- "103170"

# Open an Internet server:
rD <- rsDriver(browser = "firefox", chromever = NULL)

# Follow this procedure on the server:
# 1. Copy and paste the game link https://jv.acb.com/es/103170/jugadas
# 2. Click on each period, starting with 1C.
# 3. Scroll down to the first row of data.
# 4. Go back to R and run the following code:

# Set the remote driver:
remDr <- rD$client

# Get the play-by-play data:
game_elem <- remDr$getPageSource()[[1]]

# Close the client and the server:
remDr$close()
rD$server$stop()

period <- "1C"
data_game <- do_process_acb_pbp(game_elem, day, game_code,
                                period, acb_shields, FALSE)

## End(Not run)
```

`do_reb_off_success` *Check if scoring after offensive rebounds*

Description

For each team and player, locate the position of offensive rebounds and check if they resulted in scoring points.

Usage

```
do_reb_off_success(data, day_num, game_code, team_sel, verbose)
```

Arguments

`data`	Play-by-play prepared data from a given game.
`day_num`	Day number.
`game_code`	Game code.
`team_sel`	One of the teams' names involved in the game.
`verbose`	Logical. Decide if information of the computations must be provided or not.

Value

List with two data frames, one for the results for the team (**stats_team**) and the other for the players (**stats_player**). The team data frame shows the number of offensive rebounds, the number of those that finished in scoring (and the percentage associated) and the total of points scored. The player data frame shows the player who grabbed the offensive rebound, the player who scored and how many points.

Author(s)

Guillermo Vinue

See Also

`do_prepare_data_or`

Examples

```
df0 <- acb_vbc_cz_pbp_2223

day_num <- unique(acb_vbc_cz_pbp_2223$day)
game_code <- unique(acb_vbc_cz_pbp_2223$game_code)

df1 <- do_prepare_data_or(df0, TRUE, acb_games_2223_info)

df2 <- do_reb_off_success(df1, day_num, game_code, "Valencia Basket", FALSE)
#df2
```

`do_rpackage_stats` *R package downloads*

Description

Counts the number of times that a given R package was downloaded in a given year.

Usage

```
do_rpackage_stats(r_packages, year, verbose)
```

Arguments

`r_packages`	Vector with the names of the R packages.
`year`	String with the year.
`verbose`	Should R report information on progress? TRUE or FALSE.

Value

A data frame.

Author(s)

Guillermo Vinue

See Also

cranDownloads

Examples

```
## Not run:
do_rpackage_stats(c("BAwiR", "BasketballAnalyzeR"), 2025, TRUE)

## End(Not run)
```

do_scrape_days_acb *ACB day game codes*

Description

Obtain the game codes of any regular season day from any ACB season. These game codes will be used to define the target url from which collecting the shooting data of every game.

Usage

```
do_scrape_days_acb(season, analyst_name, verbose, num_days, edition_id)
```

Arguments

`season`	String with the starting year of the season. For example, "2024" refers to the 2024-2025 season.
`analyst_name`	Name to identify the user when doing web scraping. This is a polite way to do web scraping and certify that the user is working as transparently as possible with a research purpose.
`verbose`	Should R report information on progress? TRUE or FALSE.
`num_days`	Number of days to obtain.
`edition_id`	Identifier of the league edition. For 2024 is 975 and for 2025 is 979. For coming seasons, check it at the ACB website, such as `https://acb.com/calendario/index/temporada_id/2025` and click on any of the days to see which url appears.

Value

A data frame with two columns, one with the days and the other with the game codes.

Note

Before starting the web scraping, we must visit `https://www.acb.com/robots.txt` to check for permissions.

Author(s)

Guillermo Vinue

See Also

`do_scrape_shots_acb`

Examples

```
## Not run:
data_days <- do_scrape_days_acb("2024", "analyst_name", TRUE, 2, 975)

## End(Not run)
```

---

do_scrape_shots_acb *ACB shooting data*

Description

Obtain the shooting data from the ACB website and creates a common R data structure. Each shot is described with its (x, y) coordinates and other additional information, such as the outcome of the shot (made or missed) or the player who took that shot.

Usage

```
do_scrape_shots_acb(data_days, verbose, user_agent_def, x_apikey)
```

Arguments

`data_days`	Data frame with the game codes of each day. It is obtained with `do_scrape_days_acb`.
`verbose`	Should R report information on progress? TRUE or FALSE.
`user_agent_def`	String with the user agent.
`x_apikey`	String with the X-APIKEY.

Value

A data frame with the shooting data.

Note

The original codes of the playType column have the following meaning: 92: ft made; 93: 2pt made; 94: 3pt made; 96: ft missed. 97: 2pt missed; 98: 3pt missed; 100: dunk.

Author(s)

Guillermo Vinue

See Also

`do_scrape_days_acb`

Examples

```
## Not run:
data_days <- do_scrape_days_acb("2024", "analyst_name", TRUE, 2, 975)

data_shots <- do_scrape_shots_acb(data_days[1:2, ], TRUE, "user_agent_def", "x_apikey")

## End(Not run)
```

`do_scraping_games` *Player game finder data*

Description

This function calls the needed ancillary functions to scrape the player game finder data for the desired competition (currently, ACB, Euroleague and Eurocup).

Usage

```
do_scraping_games(competition, type_league, nums, year, verbose, accents, r_user)
```

Arguments

`competition`	String. Options are "ACB", "Euroleague" and "Eurocup".
`type_league`	String. If `competition` is ACB, to scrape ACB league games ("ACB"), Copa del Rey games ("CREY") or Supercopa games ("SCOPA").
`nums`	Numbers corresponding to the website from which scraping.
`year`	If `competition` is either Euroleague or Eurocup, the year when the season starts is needed. 2017 refers to 2017-2018 and so on.
`verbose`	Should R report information on progress? Default TRUE.
`accents`	If `competition` is ACB, should we keep the Spanish accents? The recommended option is to remove them, so default FALSE.
`r_user`	Email to identify the user when doing web scraping. This is a polite way to do web scraping and to certify that the user is working as transparently as possible with a research purpose.

Value

A data frame with the player game finder data for the competition selected.

Author(s)

Guillermo Vinue

See Also

`scraping_games_acb`, `scraping_games_euro`

Examples

```
## Not run:
# Not needed to scrape every time the package is checked, built and installed.
df1 <- do_scraping_games(competition = "ACB", type_league = "ACB", nums = 62001,
                         year = "2017-2018", verbose = TRUE, accents = FALSE,
                         r_user = "guillermo.vinue@uv.es")

df1_eur <- do_scraping_games(competition = "Euroleague", nums = 1,
                         year = "2017", verbose = TRUE,
                         r_user = "guillermo.vinue@uv.es")

## End(Not run)
```

`do_scraping_rosters` *Players profile data*

Description

This function calls the needed ancillary functions to scrape the players' profile data for the desired competition (currently, ACB, Euroleague and Eurocup).

Usage

```
do_scraping_rosters(competition, pcode, verbose, accents, year, r_user)
```

Arguments

`competition`	String. Options are "ACB", "Euroleague" and "Eurocup".
`pcode`	Code corresponding to the player's website to scrape.
`verbose`	Should R report information on progress? Default TRUE.
`accents`	If `competition` is ACB, should we keep the Spanish accents? The recommended option is to remove them, so default FALSE.
`year`	If `competition` is either Euroleague or Eurocup, the year when the season starts is needed. 2017 refers to 2017-2018 and so on.
`r_user`	Email to identify the user when doing web scraping. This is a polite way to do web scraping and to certify that the user is working as transparently as possible with a research purpose.

Value

A data frame with the players' information.

Author(s)

Guillermo Vinue

See Also

scraping_games_acb, scraping_rosters_euro

Examples

```
## Not run:
# Not needed to scrape every time the package is checked, built and installed.
df_bio <- do_scraping_rosters(competition = "ACB", pcode = "56C",
                              verbose = TRUE, accents = FALSE,
                              r_user = "guillermo.vinue@uv.es")

df_bio_eur <- do_scraping_rosters(competition = "Euroleague", pcode = "007969",
                              year = "2017", verbose = TRUE,
                              r_user = "guillermo.vinue@uv.es")

## End(Not run)
```

`do_shots_stats` *Shots statistics*

Description

Compute both the total and by zone two-point and threes statistics.

Usage

```
do_shots_stats(data_filter, data_shots_zones)
```

Arguments

`data_filter`	Shooting filtered data obtained with do_filter_data.
`data_shots_zones`	Shooting data with the court zones.

Value

A list with the following three elements:

- all_shots: Shooting data frame associated with the filters given to the function.
- summary_shots: Summary of the shots as a whole.
- summary_shots_zone: Summary of the shots by zone.

Author(s)

Guillermo Vinue

See Also

do_divide_court_zones, do_filter_data

Examples

```
## Not run:
df0 <- do_divide_court_zones(acb_shooting_data_2425)

df1 <- do_filter_data(df0, "2024-2025", "", "", "", "", "")

shots_stats <- do_shots_stats(df1, df0)

all_shots <- shots_stats$all_shots

summary_shots <- shots_stats$summary_shots

summary_shots_zone <- shots_stats$summary_shots_zone

## End(Not run)
```

do_stats *Accumulated or average statistics*

Description

This function computes either the total or the average statistics.

Usage

```
do_stats(df_games, type_stats = "Total", season, competition, type_season)
```

Arguments

df_games	Data frame with the games, players info, advanced stats and eventually recoded teams names.
type_stats	String. In English, the options are "Total" and "Average" and in Spanish, the options are "Totales" and "Promedio".
season	String indicating the season, for example, 2017-2018.
competition	String. Options are "ACB", "Euroleague" and "Eurocup".
type_season	String with the round of competition, for example regular season or playoffs and so on.

Value

Data frame.

Author(s)

Guillermo Vinue

Examples

```
compet <- "ACB"
df <- do_join_games_bio(compet, acb_games_1718, acb_players_1718)

df1 <- do_add_adv_stats(df)

df2 <- do_stats(df1, "Total", "2017-2018", compet, "Regular Season")
```

`do_stats_per_period` *Compute stats per period*

Description

Compute time played and points scored for a player of interest in any period of the game.

Usage

```
do_stats_per_period(data, day_num, game_code, team_sel, period_sel, player_sel)
```

Arguments

`data`	Prepared data from a given game.
`day_num`	Day number.
`game_code`	Game code.
`team_sel`	One of the teams' names involved in the game.
`period_sel`	Period of interest. Options can be "xC", where x=1,2,3,4.
`player_sel`	Player of interest.

Value

Data frame with one row and mainly time played (seconds and minutes) and points scored by the player of interest in the period of interest.

Note

The **game_code** column allows us to detect the source website, for example, `https://live.acb.com/es/partidos/103389/jugadas`.

Author(s)

Guillermo Vinue

Examples

```
day_num <- unique(acb_vbc_cz_pbp_2223$day)
game_code <- unique(acb_vbc_cz_pbp_2223$game_code)

team_sel <- "Valencia Basket"
period_sel <- "1C"
player_sel <- "Webb"

pre_per <- do_preproc_period(acb_vbc_cz_pbp_2223, team_sel, period_sel, acb_vbc_cz_sl_2223)

df2 <- pre_per$df2
df0_inli_team <- pre_per$df0_inli_team

df3 <- do_prepare_data(df2, day_num,
                       df0_inli_team, acb_games_2223_info,
                       game_code)

df4 <- do_stats_per_period(df3, day_num, game_code, team_sel, period_sel, player_sel)
#df4
```

`do_stats_teams` *Accumulated and average statistics for teams*

Description

This function computes the total and average statistics for every team.

Usage

```
do_stats_teams(df_games, season, competition, type_season)
```

Arguments

df_games	Data frame with the games, players info, advanced stats and eventually recoded teams names.
season	String indicating the season, for example, 2017-2018.
competition	String. Options are "ACB", "Euroleague" and "Eurocup".
type_season	String with the round of competition, for example regular season or playoffs and so on.

Value

A list with two elements:

- df_team_total: Data frame with the total statistics for every team.
- df_team_mean: Data frame with the average statistics for every team.

Author(s)

Guillermo Vinue

Examples

```
compet <- "ACB"

df <- do_join_games_bio(compet, acb_games_1718, acb_players_1718)
df$Compet <- compet

df_teams <- do_stats_teams(df, "2017-2018", "ACB", "Regular Season")

# Total statistics:
#df_teams$df_team_total

# Average statistics:
#df_teams$df_team_mean
```

`do_study_lineup` *Compute lineups with their statistics*

Description

This is an improvement of `do_lineup` to obtain all the information related to the lineups that a given team shows during a game.

Usage

```
do_study_lineup(data, day_num, game_code_num, team_sel, season = "2025-2026", verbose)
```

Arguments

`data`	Play-by-play prepared data from a given game.
`day_num`	Day number.
`game_code_num`	Game code.
`team_sel`	One of the teams involved in the game.
`season`	Season string.
`verbose`	Logical to indicate if the information of the computations must be provided.

Value

A list with four data frames:

data_lin: Statistics obtained by every lineup. **data_pos**: Start of each possession. **data_usg**: Possessions ended by each player. **data_usg_act**: Actions that ended each possession per player.

Note

A possession lasts 24 seconds in the ACB league.

Author(s)

Guillermo Vinue

See Also

`do_lineup`, `do_possession`, `do_lineup_metrics`, `do_usage`

Examples

```
## Not run:
library(dplyr)
df0 <- acb_vbc_cz_pbp_2223

day_num <- unique(acb_vbc_cz_pbp_2223$day)
game_code <- unique(acb_vbc_cz_pbp_2223$game_code)

# Starting players:
acb_games_2223_sl <- acb_vbc_cz_sl_2223 %>%
  dplyr::filter(period == "1C")

# Prepare data:
df1 <- do_prepare_data(df0, day_num,
                       acb_games_2223_sl, acb_games_2223_info,
                       game_code)

teams_game <- sort(unique(df1$team))

# Study the lineups:
data_res <- do_study_lineup(df1, day_num, game_code, teams_game[1], "2022-2023", FALSE)

## End(Not run)
```

`do_sub_lineup` *Compute sub-lineups*

Description

Compute all the sub-lineups that a given team shows during a game. They can be made up of four, three, two or one player(s).

Usage

```
do_sub_lineup(data, elem_choose, col_diff_points = FALSE)
```

Arguments

`data`	Data frame with the lineups (quintets).
`elem_choose`	Numeric: 4, 3, 2 or 1.
`col_diff_points`	Logical to indicate if `data` contains a column called diff_points.

Value

Data frame. Each row is a different sub-lineup. This is the meaning of the columns that might not be explanatory by themselves:

team_in: Time point when the sub-lineup starts playing together. **team_out**: Time point when the sub-lineup stops playing together (because there is a substitution). **time_seconds**: Total of seconds that the sub-lineup played.

Note

A possession lasts 24 seconds in the ACB league.

Author(s)

Guillermo Vinue

Examples

```
library(dplyr)
df0 <- acb_vbc_cz_pbp_2223

day_num <- unique(acb_vbc_cz_pbp_2223$day)
game_code <- unique(acb_vbc_cz_pbp_2223$game_code)

acb_games_2223_sl <- acb_vbc_cz_sl_2223 %>%
  filter(period == "1C")

df1 <- do_prepare_data(df0, day_num,
                       acb_games_2223_sl, acb_games_2223_info,
                       game_code)

df2 <- do_lineup(df1, day_num, game_code, "Valencia Basket", FALSE)

df3 <- do_sub_lineup(df2, 4)
#df3
```

`do_time_out_success` *Check if timeouts resulted in scoring*

Description

For each team, locate the position of timeouts and check if they resulted in scoring points.

Usage

```
do_time_out_success(data, day_num, game_code, team_sel, verbose)
```

Arguments

`data`	Prepared data from a given game.
`day_num`	Day number.
`game_code`	Game code.
`team_sel`	One of the teams' names involved in the game.
`verbose`	Logical. Decide if information of the computations must be provided or not.

Value

Data frame. This is the meaning of the columns:

day: Day number. **game_code**: Game code. **team**: Name of the corresponding team and coach. **times_out_requested**: Number of timeouts requested in the game. **times_out_successful**: Number of timeouts that resulted in scoring. **times_out_successful_perc**: Percentage of successful timeouts. **points_scored**: Total of points achieved after the timeouts.

Author(s)

Guillermo Vinue

See Also

`do_prepare_data_to`

Examples

```
df0 <- acb_vbc_cz_pbp_2223

day_num <- unique(acb_vbc_cz_pbp_2223$day)
game_code <- unique(acb_vbc_cz_pbp_2223$game_code)

df1 <- do_prepare_data_to(df0, TRUE, acb_games_2223_info, acb_games_2223_coach)

# sort(unique(df1$team))
# "Casademont Zaragoza_Porfirio Fisac" "Valencia Basket_Alex Mumbru"
```

```
df2 <- do_time_out_success(df1, day_num, game_code,
                           "Casademont Zaragoza_Porfirio Fisac", FALSE)
#df2
```

do_time_out_success_altern

Alternative timeouts

Description

This is an alternative to `do_time_out_success` from season 2025-2026 because to my best of knowledge the timeouts are not directly collected from web scraping and I have to check manually when they were called.

Usage

```
do_time_out_success_altern(data, day_num, game_code, team_sel, data_to_list, verbose)
```

Arguments

`data`	Prepared data from a given game.
`day_num`	Day number.
`game_code`	Game code.
`team_sel`	One of the teams' names involved in the game.
`data_to_list`	List with all the timeouts called from all coaches.
`verbose`	Logical. Decide if information of the computations must be provided or not.

Value

Data frame. This is the meaning of the columns:

day: Day number. **game_code**: Game code. **team**: Name of the corresponding team and coach. **times_out_requested**: Number of timeouts requested in the game. **times_out_successful**: Number of timeouts that resulted in scoring. **times_out_successful_perc**: Percentage of successful timeouts. **points_scored**: Total of points achieved after the timeouts.

Author(s)

Guillermo Vinue

See Also

`do_time_out_success`

do_usage *Players' usage*

Description

This function computes the players' usage to indicate how many possessions each player ended. A possession ends with a field-goal or free-throw attempt, or with a turnover.

Usage

```
do_usage(data_all_posse, team_name, lineup_curr, season = "2025-2026")
```

Arguments

`data_all_posse`	Data frame with the start of each possession.
`team_name`	Name of the team.
`lineup_curr`	Lineup currently playing in the game interval under analysis.
`season`	Season string.

Value

A list with two data frames:

data_all: Possessions ended by each player. **data_all_act**: Actions that ended each possession per player.

Author(s)

Guillermo Vinue

Examples

```
## Not run:
library(dplyr)
df0 <- acb_vbc_cz_pbp_2223

day_num <- unique(acb_vbc_cz_pbp_2223$day)
game_code <- unique(acb_vbc_cz_pbp_2223$game_code)

# Starting players:
acb_games_2223_sl <- acb_vbc_cz_sl_2223 %>%
  dplyr::filter(period == "1C")

# Prepare data:
df1 <- do_prepare_data(df0, day_num,
                       acb_games_2223_sl, acb_games_2223_info,
                       game_code)

teams_game <- sort(unique(df1$team))
```

```
team_sel <- teams_game[1]

data <- df1
data <- data %>%
  mutate(row_num = row_number()) %>%
  mutate(time_point = ifelse(nchar(time_point) < 5, paste0("0", time_point), time_point))

# Filter by team:
data1 <- data %>%
  filter(team == team_sel)

# Set also the opponent team:
team_opp <- setdiff(unique(data$team), team_sel)

# Add the last row of games' data to have the real final
# game score in case it is not available:
last_row_game <- data[nrow(data),]

last_row_game$time_point <- "00:00"
last_row_game$player <- NA
last_row_game$action <- NA
last_row_game$team <- team_sel

data1 <- bind_rows(data1, last_row_game)

# Get players out:
pl_out <- c(1, which(data1$action == "Sale de la pista"), nrow(data1))

i <- 1
data2 <- data1 %>%
  slice(pl_out[i]:pl_out[i + 1])

nr <- nrow(data2)

# Lineup:
lineup <- data2 %>%
  filter(action != "Sale de la pista") %>%
  # Avoid actions that are assigned to teams:
  filter(player != team_sel) %>%
  distinct(player) %>%
  pull()

# Identify when the possessions start:
data2_rival <- data %>%
  filter(team == team_opp) %>%
  filter(between(row_num, data2$row_num[1], data2$row_num[nr]))

data3 <- rbind(data2, data2_rival) %>%
  arrange(row_num) %>%
  na.omit()

data4 <- do_possession(data3, NULL, "10:00")
```

```
data4[data4$action == "Mate", "action"] <- "Tiro de 2 anotado"

data4_usg <- do_usage(data4, team_sel, lineup, "2022-2023")

## End(Not run)
```

`do_violin_box_plots` *Plots of data distributions*

Description

Create violin plots and boxplots to analyze the distribution of the two-point, three-point and total shots. Violin plots show the distribution shape, while boxplots give a compact statistical summary.

Usage

```
do_violin_box_plots(data_shots, data_players)
```

Arguments

`data_shots`	Shooting data frame.
`data_players`	Players' identifiers data frame.

Value

A plot.

Author(s)

Guillermo Vinue

Examples

```
## Not run:
df0 <- do_divide_court_zones(acb_shooting_data_2425)

df1 <- do_filter_data(df0, "2024-2025", "", "", "", "", "")

do_violin_box_plots(df1, acb_players_2425)

## End(Not run)
```

`do_viz_shots_gradient` *Visualization of the shots statistics with advanced features*

Description

Create a visualization of the left half of the court and compare either the field goal percentage or the points per shot of a given player with respect to the league. In addition, it can also show a heatmap with the zones where the player takes the shots.

Usage

```
do_viz_shots_gradient(data_filter, type, metric, data_shots_zones, language = "English")
```

Arguments

`data_filter`	Shooting filtered data obtained with `do_filter_data`.
`type`	Options are 'team' for team statistics, 'player' for player statistics and 'all' for the whole league.
`metric`	Options are 'fg' for the field goal percentage, 'pps' for the points per shot and 'none' if plotting a heatmap is preferred.
`data_shots_zones`	Shooting data with the court zones.
`language`	Language of the titles. Valid options are 'English' and 'Spanish' so far.

Value

A plot.

Author(s)

Guillermo Vinue

See Also

`do_divide_court_zones`, `do_filter_data`, `do_shots_stats`, `do_prepare_data_gradient`

Examples

```
## Not run:
df0 <- do_divide_court_zones(acb_shooting_data_2425)

df1 <- do_filter_data(df0, "2024-2025", "", "", "", "", "")

do_viz_shots_gradient(df1, "all", "none", df0)

df1 <- do_filter_data(df0, "2024-2025", "", "", "", "", "D. Ennis")
```

```
do_viz_shots_gradient(df1, "player", "none", df0)
do_viz_shots_gradient(df1, "player", "fg", df0)

df1 <- do_filter_data(df0, "2024-2025", "Valencia Basket", "", "", "", "")

do_viz_shots_gradient(df1, "team", "none", df0)

## End(Not run)
```

`do_viz_shots_scatter` *Visualization of the shots statistics*

Description

Create a visualization of the left half of the court and annotates both the total and by zone shooting statistics. It can also show the location of each individual shot, with color-coding for makes and misses.

Usage

```
do_viz_shots_scatter(shots_stats, type, draw, size_lab_box = 2.8, size_lab_court = 3,
                        size_point = 3, language = "English")
```

Arguments

`shots_stats`	Shooting data associated with the filters given to `do_shots_stats`.
`type`	Options are 'team' for team statistics, 'player' for player statistics and 'all' for the whole league.
`draw`	Logical. TRUE to add the shots in their coordinates. FALSE to add just the number of mades and attempted field goals.
`size_lab_box`	Size of the text indicating the overall percentages (they are inside a box).
`size_lab_court`	Size of the text indicating the percentages by zone.
`size_point`	Size of the points.
`language`	Language of the titles. Valid options are 'English' and 'Spanish' so far.

Value

A plot.

Author(s)

Guillermo Vinue

See Also

`do_divide_court_zones`, `do_filter_data`, `do_shots_stats`

Examples

```
## Not run:
df0 <- do_divide_court_zones(acb_shooting_data_2425)

df1 <- do_filter_data(df0, "2024-2025", "", "", "", "", "")

shots_stats <- do_shots_stats(df1, df0)

do_viz_shots_scatter(shots_stats, "all", FALSE)
do_viz_shots_scatter(shots_stats, "all", TRUE)

df1 <- do_filter_data(df0, "2024-2025", "", "", "", "", "D. Ennis")

shots_stats <- do_shots_stats(df1, df0)

do_viz_shots_scatter(shots_stats, "player", FALSE)
do_viz_shots_scatter(shots_stats, "player", TRUE)
do_viz_shots_scatter(shots_stats, "player", TRUE, language = "Spanish")

## End(Not run)
```

`do_volume_threes` *Volume of three-point shots*

Description

This function computes the three-point shots volume, both in offense in defense. This volume is defined as the percentage of three-point shots attempted with respect to the total field-goal attempts.

Usage

```
do_volume_threes(df)
```

Arguments

`df`	Data frame with the games and the players info.

Value

A data frame with the volume statistics.

Author(s)

Guillermo Vinue

Examples

```
library(dplyr)

df0 <- do_join_games_bio("ACB", acb_games_1718, acb_players_1718)

df1 <- df0 %>% rename(game_code = Game)

data_volume <- do_volume_threes(df1)

data_volume$data_volume_threes
data_volume$data_volume_threes_comp
```

`eurocup_games_1718` *Eurocup games 2017-2018*

Description

Games of the ten days of regular season and the first three days of top 16 of the Eurocup 2017-2018 season.

Usage

```
eurocup_games_1718
```

Format

Data frame with 3604 rows and 38 columns.

Source

https://www.euroleaguebasketball.net/eurocup/

`eurocup_players_1718` *Eurocup players 2017-2018*

Description

Players corresponding to the games of the ten days of regular season and the first three days of top 16 of the Eurocup 2017-2018 season.

Usage

```
eurocup_players_1718
```

Format

Data frame with 351 rows and 7 columns.

Source

https://www.euroleaguebasketball.net/eurocup/

`euroleague_games_1718` *Euroleague games 2017-2018*

Description

Games of the first nineteen days of the Euroleague 2017-2018 season.

Usage

```
euroleague_games_1718
```

Format

Data frame with 3932 rows and 38 columns.

Source

https://www.euroleaguebasketball.net/euroleague/

`euroleague_players_1718` *Euroleague players 2017-2018*

Description

Players corresponding to the games of the first nineteen days of the Euroleague 2017-2018 season.

Usage

```
euroleague_players_1718
```

Format

Data frame with 245 rows and 7 columns.

Source

https://www.euroleaguebasketball.net/euroleague/

`get_bubble_plot` *Basketball bubble plot*

Description

This plot is a representation of the percentiles of all statistics for a particular player. The figure shows four cells. The first box contains the percentiles between 0 and 24. The second, between 25 and 49. The third, between 50 and 74 and the fourth, between 75 and 100. The percentiles are computed with the function `percentilsArchetypoid`. Boxes of the same percentile category are in the same color in the interests of easy understanding.

This type of visualization allows the user to analyze each player in a very simple way, since a general idea of those aspects of the game in which the player excels can be obtained.

Usage

```
get_bubble_plot(df_stats, player, descr_stats, size_text, size_text_x, size_legend)
```

Arguments

`df_stats`	Data frame with the statistics.
`player`	Player.
`descr_stats`	Description of the statistics for the legend.
`size_text`	Text size inside each box.
`size_text_x`	Stats labels size.
`size_legend`	Legend size.

Details

In the example shown below, it can be seen that Alberto Abalde has a percentile of x in free throws percentage. This means that the x percent of league players has a fewer percentage than him, while there is a (100-x) percent who has a bigger percentage.

Value

Graphical device.

Author(s)

This function has been created using the code from this website: `https://www.r-bloggers.com/2017/01/visualizing-the-best/`.

See Also

`percentilsArchetypoid`

Examples

```
## Not run:
compet <- "ACB"
df <- do_join_games_bio(compet, acb_games_1718, acb_players_1718)

df1 <- do_add_adv_stats(df)

df2 <- do_stats(df1, "Total", "2017-2018", compet, "Regular Season")

# When choosing a subset of stats, follow the order in which they appear
# in the data frame.
stats <- c("GP", "MP", "PTS", "FGA", "FGPerc", "ThreePA", "ThreePPerc",
           "FTA", "FTPerc", "TRB", "ORB", "AST", "STL", "TOV")

df2_1 <- df2[, c(1:5, which(colnames(df2) %in% stats), 46:49)]

descr_stats <- c("Games played", "Minutes played", "Points",
                 "Field goals attempted", "Field goals percentage",
                 "3-point field goals attempted", "3-point percentage",
                 "FTA: Free throws attempted", "Free throws percentage",
                 "Total rebounds", "Offensive rebounds",
                 "Assists", "Steals", "Turnovers")

get_bubble_plot(df2_1, "Abalde, Alberto", descr_stats, 6, 10, 12)

## End(Not run)
```

`get_donut_usage` *Donut chart usage possessions*

Description

This function creates a donut chart that displays the percentage of possessions that each player of a team ends while being on the court.

Usage

```
get_donut_usage(data_usage, team_sel, size_play, size_perc)
```

Arguments

`data_usage`	Data frame with the number of possessions that each player played and the number that he ended.
`team_sel`	String with the team's full name.
`size_play`	Size of the players' labels.
`size_perc`	Size of the percentages labels.

Details

Donut charts are an alternative for pie charts, which have a hole in the middle, making them cleaner to read than pie charts.

Value

A donut chart.

Author(s)

Guillermo Vinue with the help of ChatGPT.

See Also

`get_donut_usage_action`

Examples

```
## Not run:
get_donut_usage(acb_usage_data_2526, "Valencia Basket", 3, 4)

## End(Not run)
```

---

get_donut_usage_action *Donut chart usage possessions action*

Description

This function creates a donut chart that displays the percentage of possessions that each player of a team ends with a particular action (turnover, field goal attempted or free throw attempted) while being on the court.

Usage

```
get_donut_usage_action(data_usage_act, team_sel, type_play,
                       language, min_poss, min_perc, size_orl,
                       size_irl = 4, vjust_title = 1)
```

Arguments

`data_usage_act`	Data frame with the number of possessions that each player ended and the particular action used.
`team_sel`	String with the team's full name.
`type_play`	Play type. Options are 'one' for free throws, 'two' for two-point field goals and 'three' for three-point field goals and 'tov' for turnovers.

`language`	Language of the titles. Valid options are 'English' and 'Spanish' so far.
`min_poss`	Minimum number of possessions played.
`min_perc`	Minimum percentage achieved.
`size_orl`	Size of the outer ring labels.
`size_irl`	Size of the inner ring labels.
`vjust_title`	Adjust the title vertically when representing turnovers.

Details

Donut charts are an alternative for pie charts, which have a hole in the middle, making them cleaner to read than pie charts.

Value

A donut chart.

Author(s)

Guillermo Vinue with the help of ChatGPT.

See Also

`get_donut_usage`

Examples

```
## Not run:
get_donut_usage_action(acb_usage_act_data_2526, "Valencia Basket", "two", "English", 1, 1, 3)
# For example, the interpretation here is that Sako finishes the 66.67% of his possessions
# scoring a two-point shot.

get_donut_usage_action(acb_usage_act_data_2526, "Valencia Basket", "tov", "English", 1, 1, 5)

## End(Not run)
```

`get_four_factors_plot` *Four factors plot*

Description

Once computed the team's factors and its rankings with `do_four_factors_df`, this function represents them.

Usage

```
get_four_factors_plot(df_rank, df_no_rank, team, language, scope = "def_off")
```

Arguments

`df_rank`	Data frame with the team's offense and defense four factors and its ranking labels.
`df_no_rank`	Data frame with the team's offense and defense four factors.
`team`	Team name. Multiple teams can be chosen.
`language`	Language labels. Current options are 'en' for English and 'es' for Spanish.
`scope`	Plot both defense and offense or just one of them. Options are "def_off", "def", "off".

Value

Graphical device.

Author(s)

Guillermo Vinue

See Also

`do_four_factors_df`

Examples

```
## Not run:
df <- do_join_games_bio("ACB", acb_games_1718, acb_players_1718)

df1 <- do_add_adv_stats(df)

team <- "Valencia"

df_four_factors <- do_four_factors_df(df1, team, NULL)

# If only one team is represented the ranking between parentheses is just one.
get_four_factors_plot(df_four_factors$df_rank, df_four_factors$df_no_rank, team, "en")

# Example with only defense:
get_four_factors_plot(df_four_factors$df_rank, df_four_factors$df_no_rank, team,
                      "en", "def") +
 ggplot2::theme(legend.position = "none")

## End(Not run)
```

get_games_rosters *Get all games and rosters*

Description

This function is to get all the games and rosters of the competition selected.

Usage

```
get_games_rosters(competition, type_league, nums, verbose = TRUE,
                          accents = FALSE, r_user, df0, df_bio0)
```

Arguments

competition	String. Options are "ACB", "Euroleague" and "Eurocup".
type_league	String. If competition is ACB, to scrape ACB league games ("ACB"), Copa del Rey games ("CREY") or Supercopa games ("SCOPA").
nums	Numbers corresponding to the website from which scraping.
verbose	Should R report information on progress? Default TRUE.
accents	If competition is ACB, should we keep the Spanish accents? The recommended option is to remove them, so default FALSE.
r_user	Email to identify the user when doing web scraping. This is a polite way to do web scraping and to certify that the user is working as transparently as possible with a research purpose.
df0	Data frame to save the games data.
df_bio0	Data frame to save the rosters data.

Value

Data frame.

Author(s)

Guillermo Vinue

Examples

```
## Not run:
library(readr)
# 1. The first time, all the historical data until the last games played can be
# directly scraped.

# ACB seasons available and corresponding games numbers:
acb_nums <- list(30001:30257, 31001:31262, 32001:32264, 33001:33492, 34001:34487,
                 35001:35494, 36001:36498, 37001:37401, 38001:38347, 39001:39417,
                 40001:40415, 41001:41351, 42001:42350, 43001:43339, 44001:44341,
                 45001:45339, 46001:46339, 47001:47339, 48001:48341, 49001:49341,
                 50001:50339, 51001:51340, 52001:52327, 53001:53294, 54001:54331,
                 55001:55331, 56001:56333, 57001:57333, 58001:58332, 59001:59331,
                 60001:60332, 61001:61298,
                 62001:62135)
names(acb_nums) <- paste(as.character(1985:2017), as.character(1986:2018), sep = "-")

df0 <- data.frame()
df_bio0 <- data.frame(CombinID = NA, Player = NA, Position = NA,
                      Height = NA, Date_birth = NA,
                      Nationality = NA, Licence = NA, Website_player = NA)

# All the games and players:
get_data <- get_games_rosters(competition = "ACB", type_league = "ACB",
                              nums = acb_nums, verbose = TRUE, accents = FALSE,
                              r_user = "guillermo.vinue@uv.es",
                              df0 = df0, df_bio0 = df_bio0)
acb_games <- get_data$df0
acb_players <- get_data$df_bio0
write_csv(acb_games, path = "acb_games.csv")
write_csv(acb_players, path = "acb_players.csv")

# 2. Then, in order to scrape new games as they are played, the df0 and df_bio0 objects are
# the historical games and rosters:
acb_nums <- list(62136:62153)
names(acb_nums) <- "2017-2018"
df0 <- read_csv("acb_games.csv", guess_max = 1e5)
df_bio0 <- read_csv("acb_players.csv", guess_max = 1e3)
get_data <- get_games_rosters(competition = "ACB", type_league = "ACB",
                              nums = acb_nums, verbose = TRUE, accents = FALSE,
                              r_user = "guillermo.vinue@uv.es",
                              df0 = df0, df_bio0 = df_bio0)

# -----

# ACB Copa del Rey seasons available and corresponding games numbers (rosters were
already downloaded with the ACB league):
acb_crey_nums <- list(50001:50004, 51001:51007, 52001:52007, 53033:53039,
                      54033:54039, 55033:55040, 56033:56040, 57029:57036,
                      58025:58032, 59038:59045, 60001:60008, 61001:61007,
                      62001:62007, 63001:63007, 64001:64007, 65001:65007,
                      66001:66007, 67001:67007, 68001:68007, 69001:69007,
                      70001:70007, 71001:71007, 72001:72007, 73001:73007,
                      74001:74007, 75001:75007, 76001:76007, 77001:77007,
                      78001:78007, 79001:79007, 80001:80007, 81001:81007)
names(acb_crey_nums) <- paste(as.character(1985:2016), as.character(1986:2017), sep = "-")

df0 <- data.frame()
get_data <- get_games_rosters(competition = "ACB", type_league = "CREY",
                              nums = acb_crey_nums, verbose = TRUE, accents = FALSE,
                              r_user = "guillermo.vinue@uv.es",
                              df0 = df0, df_bio0 = NULL)
acb_crey_games <- get_data$df0
```

```
write_csv(acb_crey_games, path = "acb_crey_games.csv")

# -----

# ACB Supercopa seasons available and corresponding games numbers (rosters were
already downloaded with the ACB league):
acb_scopa_nums <- list(1001, 2001, 3001, 4001, 5001:5004, 6001:6004,
                       7001:7003, 9001:9003, 10001:10003, 11001:11003,
                       12001:12003, 13001:13003, 14001:14003, 15001:15003,
                       16001:16003, 17001:17003, 18001:18003, 19001:19003)
# I haven't found the data for the supercopa in Bilbao 2007 ; 8001:8003
# http://www.acb.com/fichas/SCOPA8001.php
names(acb_scopa_nums) <- c(paste(as.character(1984:1987), as.character(1985:1988), sep = "-"),
                      paste(as.character(2004:2006), as.character(2005:2007), sep = "-"),
                      paste(as.character(2008:2018), as.character(2009:2019), sep = "-"))

df0 <- data.frame()
get_data <- get_games_rosters(competition = "ACB", type_league = "SCOPA",
                              nums = acb_scopa_nums, verbose = TRUE, accents = FALSE,
                              r_user = "guillermo.vinue@uv.es",
                              df0 = df0, df_bio0 = NULL)
acb_scopa_games <- get_data$df0
write_csv(acb_scopa_games, path = "acb_scopa_games.csv")

# -----

# Euroleague seasons available and corresponding games numbers:
euroleague_nums <- list(1:128,
                        1:263, 1:250, 1:251, 1:253, 1:253, 1:188, 1:189,
                        1:188, 1:188, 1:231, 1:231, 1:231, 1:229, 1:220,
                        1:220, 1:275, 1:169)
names(euroleague_nums) <- 2017:2000

df0 <- data.frame()
df_bio0 <- data.frame(CombinID = NA, Player = NA, Position = NA,
                     Height = NA, Date_birth = NA,
                     Nationality = NA, Website_player = NA)
get_data <- get_games_rosters(competition = "Euroleague", nums = euroleague_nums,
                              verbose = TRUE, r_user = "guillermo.vinue@uv.es",
                              df0 = df0, df_bio0 = df_bio0)
euroleague_games <- get_data$df0
euroleague_players <- get_data$df_bio0
write_csv(euroleague_games, path = "euroleague_games.csv")
write_csv(euroleague_players, path = "euroleague_players.csv")

# -----

# Eurocup seasons available and corresponding games numbers:
eurocup_nums <- list(1:128,
                     2:186, 1:306, 1:306, 1:366, 1:157, 1:156, 1:156, 1:156,
                     1:151, 1:326, 1:149, 1:149, 1:239, 1:209, 1:150)
names(eurocup_nums) <- 2017:2002
```

```
df0 <- data.frame()
df_bio0 <- data.frame(CombinID = NA, Player = NA, Position = NA,
                      Height = NA, Date_birth = NA,
                      Nationality = NA, Website_player = NA)
get_data <- get_games_rosters(competition = "Eurocup", nums = eurocup_nums,
                              verbose = TRUE, r_user = "guillermo.vinue@uv.es",
                              df0 = df0, df_bio0 = df_bio0)
eurocup_games <- get_data$df0
eurocup_players <- get_data$df_bio0
write_csv(eurocup_games, path = "eurocup_games.csv")
write_csv(eurocup_players, path = "eurocup_players.csv")


## End(Not run)
```

get_heatmap_bb *Basketball heatmap*

Description

The heatmap created with this function allows the user to easily represent the stats for each player. The more intense the color, the more the player highlights in the statistic considered. The plot can be ordered by any statistic. If all the statistics are represented, the offensive statistics are grouped in red, the defensive in green, the rest in purple and the advanced in pink. Otherwise, the default color is red.

Usage

```
get_heatmap_bb(df_stats, team, levels_stats = NULL, stat_ord, base_size = 9, title)
```

Arguments

`df_stats`	Data frame with the statistics.
`team`	Team.
`levels_stats`	Statistics classified in several categories to plot. If this is NULL, all the statistics are included in the data frame. Otherwise, the user can define a vector with the variables to represent.
`stat_ord`	To sort the heatmap on one particular statistic.
`base_size`	Sets the font size in the theme used. Default 9.
`title`	Plot title.

Value

Graphical device.

Author(s)

This function has been created using the code from these websites: https://learnr.wordpress.com/2010/01/26/ggplot2-quick-heatmap-plotting/ and https://stackoverflow.com/questions/13016022/ggplot2-heatmaps-using-different-gradients-for-categories/13016912

Examples

```
## Not run:
compet <- "ACB"
df <- do_join_games_bio(compet, acb_games_1718, acb_players_1718)
df1 <- do_add_adv_stats(df)
df2 <- do_stats(df1, "Total", "2017-2018", compet, "Regular Season")
teams <- as.character(rev(sort(unique(df2$Team))))
get_heatmap_bb(df2, teams[6], NULL, "MP", 9, paste(compet, "2017-2018", "Total", sep = " "))

## End(Not run)
```

`get_map_nats` *Nationalities map*

Description

A world map is represented. The countries from where there are players in the competition selected are in green color.

Usage

```
get_map_nats(df_stats)
```

Arguments

`df_stats`	Data frame with the statistics and the corrected nationalities.

Value

Graphical device.

Author(s)

Guillermo Vinue

See Also

`do_map_nats`

Examples

```
## Not run:
compet <- "ACB"
df <- do_join_games_bio(compet, acb_games_1718, acb_players_1718)
df1 <- do_add_adv_stats(df)
df2 <- do_stats(df1, "Total", "2017-2018", compet, "Regular Season")
get_map_nats(df2)

## End(Not run)
```

`get_net_rtg_on_off` *Net Rating On/Off*

Description

The Net Rating On/Off measures a team's net rating when a specific player is on the court versus when they are on the bench. It highlights the player's impact by comparing the team's efficiency both in offense and defense. A positive differential indicates a positive, higher-impact player.

Usage

```
get_net_rtg_on_off(data_combs, team_name, type_lineup, type_period, type_opponent,
                   filter_players, language, asp_ratio = 1)
```

Arguments

`data_combs`	Data frame with all the combinations of lineups.
`team_name`	Name of the team.
`type_lineup`	Type of lineups to analyze. Options are 'quintet', 'quartet', 'trio', 'duo' and 'single'.
`type_period`	Period of interest. Options are xC, where x starts from 1. Common periods are from 1 to 4. Overtimes are labeled with the next numbers, such as 5C for the first overtime and 6C for the second one. Nothing to do if NULL.
`type_opponent`	Name of the opponent teams. Nothing to do if NULL.
`filter_players`	String with the players' names to filter. Nothing to do if NULL.
`language`	Language of the legends and titles.
`asp_ratio`	Aspect ratio of the plot. Default 1.

Details

- Net Rating: difference between Offensive Rating and Defensive Rating.
- On-Court Rating: team's net rating while the player is in the game.
- Off-Court Rating: team's net rating while the player is on the bench.
- On/Off Differential: difference between the on-court and off-court net ratings, which gives a first idea of how much better/worse the team performs with that player.

Value

A plot with the players' net ratings and differentials.

Author(s)

Guillermo Vinue

See Also

`do_possession_stats_lineup`

Examples

```
## Not run:
get_net_rtg_on_off(acb_combs, "Unicaja", "single", NULL, NULL, NULL, "Spanish")

## End(Not run)
```

---

get_plot_monthly_stats

Monthly stats

Description

In all the available basketball websites, the stats are presented for the whole number of games played. This function represents the players' stats for each month, which is very useful to analyze the players' evolution. The plot can be either a bar plot or a line plot.

Usage

```
get_plot_monthly_stats(df_stats, title, size_text = 2.5, type_plot, language,
                          same_team = FALSE, hjust_val = 2, vjust_val = 0.5)
```

Arguments

`df_stats`	Data frame with the statistics.
`title`	Plot title.
`size_text`	Label size for each bar. Default 2.5.
`type_plot`	String, either 'bar_plot' or 'line_plot'.
`language`	String, either 'English' or 'Spanish'. Needed for the line plot.
`same_team`	Logical to specify if players selected belong to the same team. If so, the facet labels in the line plot can be abbreviated.
`hjust_val`	Adjust horizontally the text in the line plot.
`vjust_val`	Adjust vertically the text in the line plot.

Value

Graphical device.

Author(s)

Guillermo Vinue

See Also

capit_two_words

Examples

```
## Not run:
library(dplyr)

compet <- "ACB"

df <- do_join_games_bio(compet, acb_games_1718, acb_players_1718)
df1 <- do_add_adv_stats(df)

months <- c(df %>% distinct(Month))$Month
months_order <- c("septiembre", "octubre", "noviembre", "diciembre", "enero")
months_plot <- match(months_order, months)
months_plot1 <- months_plot[!is.na(months_plot)]
months_plot2 <- months[months_plot1]

df3_m <- df1 %>%
filter(Team == "Real_Madrid",
      Player.x == "Doncic, Luka") %>%
 group_by(Month) %>%
 do(do_stats(., "Average", "2017-2018", "ACB", "Regular Season")) %>%
 ungroup() %>%
 mutate(Month = factor(Month, levels = months_plot2)) %>%
 arrange(Month)

stats <- c("GP", "MP", "PTS", "FGA", "FGPerc", "ThreePA",
           "ThreePPerc", "FTA", "FTPerc",
           "TRB", "ORB", "AST", "TOV", "STL")

df3_m1 <- df3_m %>%
  select(1:5, all_of(stats), 46:50) %>%
  mutate(Month = plyr::mapvalues(Month,
                                from = c("octubre", "noviembre", "diciembre", "enero"),
                                to = c("October", "November", "December", "January")))

get_plot_monthly_stats(df3_m1, paste("ACB", "2017-2018", "Average", sep = " ; "),
                       2.5, "bar_plot", "English")

get_plot_monthly_stats(df3_m1, paste("ACB", "2017-2018", "Average", sep = " ; "),
                       2.5, "line_plot", "English")
```

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

get_pop_pyramid *Population pyramid*

Description

This is the code to get a population pyramid with the number of both Spanish and foreigner players along the seasons for the ACB league. This aids in discussion of nationality imbalance.

Usage

```
get_pop_pyramid(df, title, language)
```

Arguments

`df`	Data frame that contains the ACB players' nationality.
`title`	Title of the plot
`language`	String, "eng" for English labels; "esp" for Spanish labels.

Value

Graphical device.

Author(s)

Guillermo Vinue

Examples

```
## Not run:
 # Load the data_app_acb file with the ACB games
 # from seasons 1985-1986 to 2017-2018:
 load(url("http://www.uv.es/vivigui/softw/data_app_acb.RData"))
 title <- " Number of Spanish and foreign players along the ACB seasons \n Data from www.acb.com"
 get_pop_pyramid(data_app_acb, title, "eng")

## End(Not run)
```

get_roster_age_profile *Roster age profile*

Description

For the players of the same team, show their age at time of joining the team, their current year and how many years they have spent in the team.

Usage

```
get_roster_age_profile(data_age_team, team_sel, language)
```

Arguments

`data_age_team`	Data frame with the team's age profile.
`team_sel`	Team.
`language`	Language labels. Current options are 'en' for English and 'es' for Spanish.

Value

Graphical device.

Author(s)

Guillermo Vinue

Examples

```
## Not run:
get_roster_age_profile(acb_age_profile_data_2526, "Valencia Basket", "es")

## End(Not run)
```

get_shooting_plot *Shooting plot*

Description

This plot represents the number of shots attempted and scored by every player of the same team, together with the scoring percentage. The players are sortered by percentage.

Usage

```
get_shooting_plot(df_stats, team, type_shot, min_att, title, language,
                          size_summ = 5, size_add = 16)
```

Arguments

`df_stats`	Data frame with the statistics.
`team`	Team.
`type_shot`	Numeric with values 1-2-3: 1 refers to free throws, 2 refers to two point shots and 3 refers to three points shots.
`min_att`	Minimum number of attempts by the player to be represented in the plot.
`title`	Plot title.
`language`	Language labels. Current options are 'en' for English and 'es' for Spanish.
`size_summ`	Size of the text summarizing the total shots and the percentage.
`size_add`	Size of the additional axis and legends.

Value

Graphical device.

Author(s)

Guillermo Vinue

Examples

```
## Not run:
compet <- "ACB"
df <- do_join_games_bio(compet, acb_games_1718, acb_players_1718)

df1 <- do_add_adv_stats(df)

df2 <- do_stats(df1, "Total", "2017-2018", compet, "Regular Season")

get_shooting_plot(df2, "Valencia", 3, 1, paste("Valencia", compet, "2017-2018", sep = " "), "en")

## End(Not run)
```

`get_similar_players` *Similar players to archetypoids*

Description

Similar players to the archetypoids computed with `archetypoids` according to a similarity threshold.

Usage

```
get_similar_players(atype, threshold, alphas, cases, data, variables, compet, season)
```

Arguments

atype	Number assigned to the archetypoid (1:length(cases)) from which searching the players who most resemble to it.
threshold	Similarity threshold.
alphas	Alpha values of all the players.
cases	Archetypoids.
data	Data frame with the statistics.
variables	Statistics used to compute the archetypoids.
compet	Competition.
season	Season.

Value

Data frame with the features of the similar players.

Author(s)

Guillermo Vinue

See Also

archetypoids

Examples

```
(s0 <- Sys.time())
# Turn off temporarily some negligible warnings from the
# archetypes package to avoid missunderstandings. The code works well.
library(Anthropometry)
df <- do_join_games_bio("ACB", acb_games_1718, acb_players_1718)
df1 <- do_add_adv_stats(df)
df2 <- do_stats(df1, "Total", "2017-2018", "ACB", "Regular Season")
df3 <- df2[which(df2$Position == "Guard")[1:31], c("MP", "PTS", "Name")]
preproc <- preprocessing(df3[,1:2], stand = TRUE, percAccomm = 1)
set.seed(4321)
suppressWarnings(lass <- stepArchetypesRawData(preproc$data, 1:2,
                numRep = 20, verbose = FALSE))
res <- archetypoids(2, preproc$data, huge = 200, step = FALSE, ArchObj = lass,
                    nearest = "cand_ns", sequ = TRUE)
# The S3 class of anthrCases from Anthropometry has been updated.
cases <- anthrCases(res)
df3[cases,] # https://github.com/r-quantities/units/issues/225
alphas <- round(res$alphas, 4)
df3_aux <- df2[which(df2$Position == "Guard")[1:31], ]
get_similar_players(1, 0.99, alphas, cases, df3_aux, c("MP", "PTS"),
                    unique(df3_aux$Compet), unique(df3_aux$Season))
s1 <- Sys.time() - s0
s1
```

get_similar_teams *Similar teams to archetypoids*

Description

Similar teams to the archetypoids computed with archetypoids according to a similarity threshold.

Usage

```
get_similar_teams(atype, threshold, alphas, cases, data, variables)
```

Arguments

atype	Number assigned to the archetypoid (1:length(cases)) from which searching the players who most resemble to it.
threshold	Similarity threshold.
alphas	Alpha values of all the players.
cases	Archetypoids.
data	Data frame with the statistics.
variables	Statistics used to compute the archetypoids.

Value

Data frame with the features of the similar teams.

Author(s)

Guillermo Vinue

See Also

archetypoids

Examples

```
## Not run:
(s0 <- Sys.time())
library(Anthropometry)
df <- do_join_games_bio("ACB", acb_games_1718, acb_players_1718)
df$Compet <- "ACB"
df_teams <- do_stats_teams(df, "2017-2018", "ACB", "Regular Season")
df_team_total <- df_teams$df_team_total

df3 <- df_team_total[, c("PTS", "PTSrv", "Team")]
preproc <- preprocessing(df3[,1:2], stand = TRUE, percAccomm = 1)
set.seed(4321)
lass <- stepArchetypesRawData(preproc$data, 1:2, numRep = 20, verbose = FALSE)
```

```
res <- archetypoids(2, preproc$data, huge = 200, step = FALSE, ArchObj = lass,
                    nearest = "cand_ns", sequ = TRUE)
cases <- anthrCases(res)
df3[cases,]
alphas <- round(res$alphas, 4)

get_similar_teams(1, 0.95, alphas, cases, df_team_total, c("PTS", "PTSrv"))
s1 <- Sys.time() - s0
s1

## End(Not run)
```

get_stats_seasons *Season-by-season stats*

Description

This function represents the average values of a set of statistics for certain players in every season where the players played. It gives an idea of the season-by-season performance.

Usage

```
get_stats_seasons(df, competition, player, variabs, type_season, add_text, show_x_axis)
```

Arguments

`df`	Data frame with the games and the players info.
`competition`	Competition.
`player`	Players's names.
`variabs`	Vector with the statistics to plot.
`type_season`	String with the round of competition, for example regular season or playoffs and so on.
`add_text`	Boolean. Should text be added to the plot points?
`show_x_axis`	Boolean. Should x-axis labels be shown in the plot?

Value

List with two elements:

- gg Graphical device.
- df_gg Data frame associated with the plot.

Author(s)

Guillermo Vinue

Examples

```
## Not run:
competition <- "ACB"

df <- do_join_games_bio("ACB", acb_games_1718, acb_players_1718)
df$Compet <- competition

player <- "Carroll, Jaycee"

variabs <- c("GP", "MP", "PTS", "EFGPerc", "TRB", "AST", "TOV", "PIR")

plot_yearly <- get_stats_seasons(df, competition, player, variabs, "All", TRUE, TRUE)
plot_yearly$gg

# There are only games from the regular season in this demo data frame.
plot_yearly1 <- get_stats_seasons(df, competition, player, variabs, "Regular Season",
                                  TRUE, TRUE)
plot_yearly1$gg

## End(Not run)
```

`get_sticker` *Player's sticker*

Description

This function creates players' cards (a kind of sticker) that bring together for each player three of the concepts considered most important in basketball analytics. These are efficiency (how many points the team scores and receives per 100 possessions with the player on the floor), the shooting context (from where and with what percentage they shoot) and the use of possessions (how they end the possessions they execute).

Usage

```
get_sticker(data_player_eff, data_team, player_sel, language = "English",
            change_hjust_perc = FALSE, size_head = c(0.4, 0.3, 5.6, 3.5),
            size_eff = 2, size_cont_us = c(2.3, 1.7), size_plot_tit = 8,
            hjust_title = 0.1, xjust_vt = c(0.392, 0.3), size_netrtg = 20)
```

Arguments

`data_player_eff`	Data frame with the efficiency statistics.
`data_team`	Data frame with the context and usage statistics.
`player_sel`	Player of interest.
`language`	Language of the titles. Valid options are 'English' and 'Spanish' so far.

`change_hjust_perc`	Logical to change the position of the win percentage sentence.
`size_head`	Vector with the sizes of headers text.
`size_eff`	Size of the text related to the winning percentage and player's efficiencies.
`size_cont_us`	Vector with the sizes of context and usage text.
`size_plot_tit`	Size of the plot titles.
`hjust_title`	Adjust the title of the net efficiency.
`xjust_vt`	Adjust the text of the team's winning percentage and efficiencies.
`size_netrtg`	Size of the colored net efficiency value.

Value

A plot.

Author(s)

Guillermo Vinue

Examples

```
## Not run:
# The efficiency data frame must have this type of structure:
data_player_eff <- data.frame(team = "Real Madrid",
                              win_perc = "83.3% (5/6)",
                              pts_poss = 114,
                              pts_poss_opp = 104,
                              net_rtg = 10)

get_sticker(data_player_eff, acb_sticker_data_2526, "A. Abalde", language = "English")
get_sticker(data_player_eff, acb_sticker_data_2526, "A. Abalde", language = "Spanish", TRUE)

## End(Not run)
```

`get_table_results` *League cross table*

Description

The league results are represented with a cross table.

Usage

```
get_table_results(df, competition, season)
```

Arguments

`df`	Data frame with the games and the players info.
`competition`	Competition.
`season`	Season.

Value

List with these two elements:

- plot_teams Graphical device with the cross table.
- wins_teams Vector with the team wins.

Author(s)

Guillermo Vinue

Examples

```
## Not run:
df <- do_join_games_bio("ACB", acb_games_1718, acb_players_1718)
df$Compet <- "ACB"

gg <- get_table_results(df, "ACB", "2017-2018")

gg$wins_teams
gg$plot_teams

## End(Not run)
```

`join_players_bio_age_acb` *Join ACB games and players' info*

Description

This function joins the ACB games with the players' bio and computes the players' age at each game.

Usage

```
join_players_bio_age_acb(df_games, df_rosters)
```

Arguments

`df_games`	Data frame with the games.
`df_rosters`	Data frame with the biography of the roster players.

Value

Data frame.

Author(s)

Guillermo Vinue

See Also

`do_join_games_bio`

Examples

```
df <- join_players_bio_age_acb(acb_games_1718, acb_players_1718)
```

---

join_players_bio_age_euro

Join Euroleague and Eurocup games and players' info

Description

This function joins the Euroleague/Eurocup games with the players' bio and computes the players' age at each game.

Usage

```
join_players_bio_age_euro(df_games, df_rosters)
```

Arguments

`df_games`	Data frame with the games.
`df_rosters`	Data frame with the biography of the roster players.

Value

Data frame.

Author(s)

Guillermo Vinue

See Also

`do_join_games_bio`

Examples

```
df <- join_players_bio_age_euro(euroleague_games_1718, euroleague_players_1718)
```

`metrics_player_zone` *ACB players 2024-2025*

Description

Metrics (total shots, field goal percentages and points per shot) at some of the court zones for some of the 30 players contained in the data frame `acb_players_2425`.

Usage

```
metrics_player_zone
```

Format

Data frame with 8 rows and 5 columns.

Source

https://www.acb.com/

`scraping_games_acb` *ACB player game finder data*

Description

This is the new function to obtain the ACB box score data.

Usage

```
scraping_games_acb(code, game_id, season = "2020-2021",
                   type_season = "Regular Season",
                   user_email, user_agent_goo)
```

Arguments

`code`	Game code.
`game_id`	Game id.
`season`	Season, e.g. 2022-2023.
`type_season`	Type of season, e.g. 'Regular season'.
`user_email`	Email's user to identify the user when doing web scraping. This is a polite way to do web scraping and to certify that the user is working as transparently as possible with a research purpose.
`user_agent_goo`	User-agent to identify the user when doing web scraping. This is a polite way to do web scraping and to certify that the user is working as transparently as possible with a research purpose.

Value

A data frame with the player game finder data (box score data).

Author(s)

Guillermo Vinue

See Also

`scraping_games_acb_old`

Examples

```
## Not run:
# Not needed to scrape every time the package is checked, built and installed.
user_email <- "yours"
user_agent_goo <- "yours"
df1 <- scraping_games_acb("103350", 1, "2022_2023", "Regular Season",
                          user_email, user_agent_goo)

## End(Not run)
```

`scraping_games_acb_old` *Old ACB player game finder data*

Description

This function allowed us to get all the player game finder data for all the desired ACB seasons available from: `https://www.acb.com`. It was an old version that worked before the internal structure of the ACB website changed. The updated function is now `scraping_games_acb`.

Usage

```
scraping_games_acb_old(type_league, nums, year, verbose = TRUE,
                       accents = FALSE, r_user = "guillermo.vinue@uv.es")
```

Arguments

`type_league`	String. If `competition` is ACB, to scrape ACB league games ("ACB"), Copa del Rey games ("CREY") or Supercopa games ("SCOPA").
`nums`	Numbers corresponding to the website to scrape.
`year`	Season, e.g. 2017-2018.
`verbose`	Should R report information on progress? Default TRUE.
`accents`	Should we keep the Spanish accents? The recommended option is to remove them, so default FALSE.

`r_user`	Email to identify the user when doing web scraping. This is a polite way to do web scraping and to certify that the user is working as transparently as possible with a research purpose.

Details

The official website of the Spanish basketball league ACB used to present the statistics of each game in a php website, such as: https://www.acb.com/fichas/LACB62090.php.

In some cases, https://www.acb.com/fichas/LACB60315.php didn't exist, so for these cases is where we can use the `httr` package.

Value

A data frame with the player game finder data.

Note

In addition to use the email address to stay identifiable, the function also contains two headers regarding the R platform and version used.

Furthermore, even though in the robots.txt file at `https://www.acb.com/robots.txt`, there is no information about scraping limitations and all robots are allowed to have complete access, the function also includes the command `Sys.sleep(2)` to pause between requests for 2 seconds. In this way, we don't bother the server with multiple requests and we do carry out a friendly scraping.

Author(s)

Guillermo Vinue

See Also

`do_scraping_games`

Examples

```
## Not run:
# Not needed to scrape every time the package is checked, built and installed.
df1 <- scraping_games_acb_old(type_league = "ACB", nums = 62001:62002, year = "2017-2018",
                              verbose = TRUE, accents = FALSE,
                              r_user = "guillermo.vinue@uv.es")

## End(Not run)
```

scraping_games_euro *Euroleague and Eurocup player game finder data*

Description

This function should allow us to get all the player game finder data for all the desired Euroleague and Eurocup seasons available from https://www.euroleaguebasketball.net/euroleague/game-center/ and https://www.euroleaguebasketball.net/eurocup/game-center/, respectively.

NOTE (2023): The Euroleague and Eurocup websites have changed their format, so this function will need to be updated.

Usage

```
scraping_games_euro(competition, nums, year, verbose = TRUE,
                    r_user = "guillermo.vinue@uv.es")
```

Arguments

`competition`	String. Options are "Euroleague" and "Eurocup".
`nums`	Numbers corresponding to the website from which scraping.
`year`	Year when the season starts. 2017 refers to 2017-2018 and so on.
`verbose`	Should R report information on progress? Default TRUE.
`r_user`	Email to identify the user when doing web scraping. This is a polite way to do web scraping and to certify that the user is working as transparently as possible with a research purpose.

Details

See the examples in `get_games_rosters` to see the game numbers to scrape in each season.

Value

A data frame with the player game finder data.

Note

In addition to use the email address to stay identifiable, the function also contains two headers regarding the R platform and version used.

Furthermore, in the robots.txt file located at https://www.euroleaguebasketball.net/robots.txt there is no Crawl-delay field. However, we assume crawlers to pause between requests for 15 seconds. This is done by adding to the function the command `Sys.sleep(15)`.

Author(s)

Guillermo Vinue

See Also

do_scraping_games

Examples

```
## Not run:
# Not needed to scrape every time the package is checked, built and installed.
# It takes 15 seconds.
df1 <- do_scraping_games(competition = "Euroleague", nums = 1:2,
                         year = "2017", verbose = TRUE, r_user =
                         "guillermo.vinue@uv.es")

## End(Not run)
```

scraping_rosters_acb *ACB players' profile*

Description

This function allows us to obtain the basic information of each player, including his birth date. Then, we will be able to compute the age that each player had in the date that he played each game. The website used to collect information is https://www.acb.com.

Usage

```
scraping_rosters_acb(pcode, verbose = TRUE, accents = FALSE,
                     r_user = "guillermo.vinue@uv.es")
```

Arguments

pcode	Code corresponding to the player's website to scrape.
verbose	Should R report information on progress? Default TRUE.
accents	Should we keep the Spanish accents? The recommended option is to remove them, so default FALSE.
r_user	Email user to identify the user when doing web scraping. This is a polite way to do web scraping and to certify that the user is working as transparently as possible with a research purpose.

Details

Some players have a particular licence, which does not necessarily match with their nationality, in order not to be considered as a foreign player, according to the current ACB rules.

Value

Data frame with eight columns:

- CombinID: Unique ID to identify the players.
- Player: Player's name.
- Position: Player's position on the court.
- Height: Player's height.
- Date_birth: Player's birth date.
- Nationality: Player's nationality.
- Licence: Player's licence.
- Website_player: Website.

Note

In addition to use the email address to stay identifiable, the function also contains two headers regarding the R platform and version used.

Furthermore, even though in the robots.txt file at `https://www.acb.com/robots.txt`, there is no information about scraping limitations and all robots are allowed to have complete access, the function also includes the command `Sys.sleep(2)` to pause between requests for 2 seconds. In this way, we don't bother the server with multiple requests and we do carry out a friendly scraping.

Author(s)

Guillermo Vinue

See Also

`do_scraping_rosters`

Examples

```
## Not run:
# Not needed to scrape every time the package is checked, built and installed.
df_bio <- scraping_rosters_acb("56C", verbose = TRUE, accents = FALSE,
                               r_user = "guillermo.vinue@uv.es")

## End(Not run)
```

`scraping_rosters_euro` *Euroleague and Eurocup players' profile*

Description

This function should allow us to obtain the basic information of each Euroleague/Eurocup player, including his birth date. Then, we will be able to compute the age that each player had in the date that he played each game. The websites used to collect information are `https://www.euroleaguebasketball.net/euroleague/` and `https://www.euroleaguebasketball.net/eurocup/`.

Usage

```
scraping_rosters_euro(competition, pcode, year, verbose = TRUE,
                      r_user = "guillermo.vinue@uv.es")
```

Arguments

`competition`	String. Options are "Euroleague" and "Eurocup".
`pcode`	Code corresponding to the player's website to scrape.
`year`	Year when the season starts. 2017 refers to 2017-2018 and so on.
`verbose`	Should R report information on progress? Default TRUE.
`r_user`	Email user to identify the user when doing web scraping. This is a polite way to do web scraping and to certify that the user is working as transparently as possible with a research purpose.

Value

Data frame with seven columns:

- CombinID: Unique ID to identify the players.
- Player: Player's name.
- Position: Player's position on the court.
- Height: Player's height.
- Date_birth: Player's birth date.
- Nationality Player's nationality.
- Website_player: Website.

Note

In addition to use the email address to stay identifiable, the function also contains two headers regarding the R platform and version used.

`https://www.euroleaguebasketball.net/robots.txt` there is no Crawl-delay field. However, we assume crawlers to pause between requests for 15 seconds. This is done by adding to the function the command `Sys.sleep(15)`.

Author(s)

Guillermo Vinue

See Also

do_scraping_rosters

Examples

```
## Not run:
# Not needed to scrape every time the package is checked, built and installed.
# It takes 15 seconds.
df_bio <- scraping_rosters_euro("Euroleague", "005791", "2017", verbose = TRUE,
                                 r_user = "guillermo.vinue@uv.es")

## End(Not run)
```

Index