

Package: comperes (via r-universe)

November 21, 2024

Title Manage Competition Results

Version 0.2.7.9000

Description Tools for storing and managing competition results. Competition is understood as a set of games in which players gain some abstract scores. There are two ways for storing results: in long (one row per game-player) and wide (one row per game with fixed amount of players) formats. This package provides functions for creation and conversion between them. Also there are functions for computing their summary and Head-to-Head values for players. They leverage grammar of data manipulation from 'dplyr'.

License MIT + file LICENSE

URL <https://github.com/echasnovski/comperes>

BugReports <https://github.com/echasnovski/comperes/issues>

Depends R (>= 3.4.0)

Imports dplyr (>= 0.7.0), magrittr, rlang (>= 0.1.2), tibble, tidyr (>= 0.7.0)

Suggests covr, knitr, rmarkdown, testthat

VignetteBuilder knitr

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE, old_usage = TRUE)

RoxygenNote 7.2.3

Config/pak/sysreqs libicu-dev

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

RemoteUrl <https://github.com/echasnovski/comperes>

RemoteRef HEAD

RemoteSha 171bd05487759faefb2e251f6dd68cf0494c48ab

Contents

comperes-package	2
convert-pair-value	3
get_matchups	5
h2h_funs	6
h2h_long	7
h2h_mat	8
hp_survey	10
item-summary	11
item-summary-join	13
levels2	14
longcr	14
ncaa2005	16
pairgames	17
summary_funs	18
widecr	19
Index	22

comperes-package *comperes: Manage Competition Results*

Description

comperes offers a set of tools for storing and managing competition results. **Competition** is understood as a set of **games** (abstract event) in which **players** (abstract entity) gain some abstract **scores**. The most natural example is sport results, however not the only one. For example, product rating can be considered as a competition between products as "players". Here a "game" is a customer that reviews a set of products by rating them with numerical "score" (stars, points, etc.).

Details

This package provides the following functionality:

- **Store and convert** competition results:
 - In **long format** as a **tibble** with one row per game-player pair.
 - In **wide format** as a tibble with one row per game with fixed amount of players.
- **Summarise**:
 - Compute **item summaries** with functions using **dplyr**'s grammar of data manipulation.
 - Compute and **join** item summaries to data for easy transformation.
 - Use **common item summary functions** with **rlang**'s **unquoting** mechanism.
- **Compute Head-to-Head values** (a summary statistic of direct confrontation between two players) with functions also using dplyr's grammar:
 - Store output in **long format** as a tibble with one row per pair of players.

- Store output in [matrix format](#) as a matrix with rows and columns describing players and entries - Head-to-Head values.
- Use [common Head-to-Head functions](#) with rlang's unquoting mechanism.

To learn more about comperes browse vignettes with `browseVignettes(package = "comperes")`.

Author(s)

Maintainer: Evgeni Chasnovski <evgeni.chasnovski@gmail.com> ([ORCID](#))

See Also

Useful links:

- <https://github.com/echasnovski/comperes>
- Report bugs at <https://github.com/echasnovski/comperes/issues>

convert-pair-value *Convert between long pair-value data and matrix*

Description

Functions for conversion between long pair-value data (data frame with columns for pair identifiers and value column) and matrix.

Usage

```
long_to_mat(tbl, row_key, col_key, value = NULL, fill = NULL,
  silent = FALSE)
```

```
mat_to_long(mat, row_key, col_key, value, drop = FALSE)
```

Arguments

<code>tbl</code>	Data frame with pair-value data.
<code>row_key</code>	String name of column for first key in pair.
<code>col_key</code>	String name of column for second key in pair.
<code>value</code>	String name of column for value (or NULL for <code>long_to_mat()</code>).
<code>fill</code>	Value to fill for missing pairs.
<code>silent</code>	Use TRUE to omit message about guessed value column (see Details).
<code>mat</code>	Matrix with pair-value data.
<code>drop</code>	Use TRUE to drop rows with missing value (see Details).

Details

Pair-value data is commonly used in description of pairs of objects. Pair is described by two keys (usually integer or character) and value is an object of arbitrary nature.

In **long format** there are at least three columns: for first key in pair, for second key and for value (might be more). In **matrix format** pair-value data is represented as matrix of values with row names as character representation of first key, column names - second key.

`long_to_mat()` works as follows:

- Pair identifiers are taken from columns with names `row_key` (to be used as row names) and `col_key` (to be used as column names). Unique identifiers (and future dimension names) are determined with `levels2()`. This is a way to target function on specific set of pairs by using factor columns. **Note** that NAs are treated as single unknown key and put on last place (in case of non-factor).
- Values are taken from column with name `value`. **Note** that if `value` has length 0 (typically NULL) then `long_to_mat()` will take first non-key column. If there is no such column, it will use vector of dummy values (NAs or fills). In both cases a message is given if `silent = FALSE`.
- Output is a matrix with described row and column names. Value of pair "key_1" and "key_2" is stored at intersection of row "key_1" and "key_2". **Note** that in case of duplicated pairs the value from first occurrence is taken.

`mat_to_long()` basically performs inverse operation to `long_to_mat()` but pair identifiers are always character. If `drop = TRUE` it drops rows with values (but not keys) being missing.

Value

`long_to_mat()` returns a matrix with selected values where row names indicate first key in pair, col names - second.

`mat_to_long()` returns a tibble with three columns: the one for first key in pair, the one for second, and the one for value.

Examples

```
long_data <- data.frame(
  key_1 = c("a", "a", "b"),
  key_2 = c("c", "d", "c"),
  val = 1:3,
  stringsAsFactors = FALSE
)

mat_data <- long_data %>% long_to_mat("key_1", "key_2", "val")
print(mat_data)

# Converts to tibble
mat_data %>% mat_to_long("new_key_1", "new_key_2", "new_val")

# Drops rows with values missing
mat_data %>% mat_to_long("new_key_1", "new_key_2", "new_val", drop = TRUE)
```

get_matchups	<i>Get matchups from competition results</i>
--------------	--

Description

This function powers computing Head-to-Head values (both [long](#) and [matrix](#)).

Usage

```
get_matchups(cr_data)
```

Arguments

cr_data Competition results ready for [as_longcr\(\)](#).

Details

get_matchups() returns a [tibble](#) of all matchups (pairs of players from one game) **actually present** in cr_data (including matchups of players with themselves). It has following columns:

- game - game identifier of matchup.
- player1 - identifier of first player in matchup.
- score1 - score of the first player in matchup.
- player2 - identifier of second player in matchup.
- score2 - score of the second player in matchup.

Important notes:

- Matchups are not symmetrical: matchup "player1"- "player2" is considered different from "player2"- "player1" in order to except more advanced, not symmetrical Head-to-Head values.
- Missing values in player column after conversion to longcr are treated as separate players. It allows operating with games where multiple players' identifiers are not known. However, when computing Head-to-Head values they treated as single player.

Value

A [widecr](#) for games with two players.

See Also

[Long format](#) of Head-to-Head values.

[Matrix format](#) of Head-to-Head values.

Examples

```
get_matchups(ncaa2005)
```

h2h_funs

Common Head-to-Head functions

Description

List of commonly used functions for computing Head-to-Head values.

Usage

h2h_funs

Format

An object of class `list` of length 9.

Details

h2h_funs is a named list of [expressions](#) representing commonly used expressions of Head-to-Head functions for computing Head-to-Head values with [h2h_long\(\)](#) or [h2h_mat\(\)](#). Names of the elements will be used as Head-to-Head value names. To use them inside [h2h_long\(\)](#) or [h2h_mat\(\)](#) use [unquoting](#) mechanism from [rlang](#) package.

Currently present functions:

- `mean_score_diff` - computes mean score difference of player1 compared to player2.
- `mean_score_diff_pos` - equivalent to `mean_score_diff` but returns 0 if result is negative.
- `mean_score` - computes mean score of player1.
- `sum_score_diff` - computes sum of score differences of player1 compared to player2.
- `sum_score_diff_pos` - equivalent to `sum_score_diff` but returns 0 if result is negative.
- `sum_score` - computes sum of scores of player1.
- `num_wins` - computes number of matchups player1 scored **more** than player2. Draws (determined by `dplyr::near()`) are omitted.
- `num_wins2` - computes number of matchups player1 scored **more** than player2 **plus** half the number of matchups where they had draw. **Note** that for equal player1 and player2 there might be non-zero output.
- `num` - computes number of matchups.

Note that it is generally better to subset `h2h_funs` using names rather than indices because the order of elements might change in future versions.

See Also

[Long format](#) of Head-to-Head values.

[Matrix format](#) of Head-to-Head values.

Examples

```
ncaa2005 %>% h2h_long(!!!h2h_funs)

ncaa2005 %>% h2h_mat(!!!h2h_funs["num_wins2"])
```

h2h_long

Compute long format of Head-to-Head values

Description

Functions to compute Head-to-Head values in long pair-value format.

Usage

```
h2h_long(cr_data, ..., fill = list())

to_h2h_long(mat, value = "h2h_value", drop = FALSE)

## S3 method for class 'h2h_long'
as_tibble(x, ...)
```

Arguments

cr_data	Competition results ready for as_longcr() .
...	Name-value pairs of Head-to-Head functions (see Details).
fill	A named list that for each variable supplies a single value to use instead of NA for missing pairs (see tidyr's complete()).
mat	Matrix of Head-to-Head values.
value	String name to be used for column with Head-to-Head value.
drop	Use TRUE to drop rows with missing Head-to-Head values (see Details).
x	Object to be converted to tibble .

Details

`h2h_long()` computes Head-to-Head values in [long](#) format. It allows computation of multiple Head-to-Head values at the same time by supplying multiple summary functions in `...`. If no function is supplied in `...`, it returns all appropriate combinations of [matchups](#) (see next paragraph).

After computing Head-to-Head values of actually present matchups, they are aligned with "levels" (see [levels2\(\)](#)) of player vector (after applying [as_longcr\(\)](#)). This is a way to target function on fixed set of players by using factor columns. The procedure is:

- Implicit missing matchups are turned into explicit (by adding corresponding rows with filling values in Head-to-Head columns) by using [tidyr's complete\(\)](#).
- All matchups not containing players from "levels" are removed.

Use `fill` as in `complete()` to control filling values. To drop those rows use `tidyr`'s `drop_na()`.

`to_h2h_long()` takes **object of `h2h_mat` structure** and converts it into `h2h_long` object with value column named as stored in `value`. Use `drop = TRUE` to remove rows with missing values in `value` column (but not in `players`).

Value

An object of class `h2h_long` which is a [tibble](#) with columns `player1`, `player2` and those, produced by Head-to-Head functions (for `h2h_long()` maybe none).

[as_tibble\(\)](#) applied to `h2h_long` object drops `h2h_long` class.

Head-to-Head value

Head-to-Head value is a summary statistic of direct confrontation between two players. It is assumed that this value can be computed based only on the players' [matchups](#). In other words, every game is converted into series of "subgames" between ordered pairs of players (including selfplay) which is stored as [widecr](#) object. After that, summary of item, defined by columns `player1` and `player2`, is computed using [summarise_item\(\)](#).

That said, name-value pairs of Head-to-Head functions should be defined as for `summarise_item()` applied to data with columns `game`, `player1`, `score1`, `player2`, `score2`.

See Also

[Matrix format](#) of Head-to-Head values.

[Common Head-to-Head functions](#).

Examples

```
ncaa2005 %>%
  h2h_long(
    mean_score = mean(score1),
    mean_abs_score = mean(abs(score1 - score2))
  )
```

```
ncaa2005[-(1:2), ] %>%
  h2h_long(
    mean_score = mean(score1),
    fill = list(mean_score = 0)
  )
```

h2h_mat

Compute matrix format of Head-to-Head values

Description

Functions to compute Head-to-Head values in matrix pair-value format.

Usage

```
h2h_mat(cr_data, ..., fill = NULL)

to_h2h_mat(tbl, value = NULL, fill = NULL)
```

Arguments

cr_data	Competition results ready for as_longcr() .
...	Name-value pairs of Head-to-Head functions (see Details).
fill	A single value to use instead of NA for missing pairs.
tbl	Data frame in long format of Head-to-Head values.
value	String name for column with Head-to-Head value.

Details

`h2h_mat()` computes Head-to-Head values in [matrix](#) format. It allows multiple Head-to-Head functions in ... but only first (if present) will be used. Basically, it uses supplied function to compute long format of Head-to-Head values and then [transforms](#) it naturally to matrix, filling missing values with `fill`.

`to_h2h_mat()` takes **object of [h2h_long](#) structure** and converts it into `h2h_mat` using column with name `value` for values and filling data for implicitly missing (not explicitly provided in `tbl`) player pairs with `fill`. If `value` is `NULL` it takes first non-player column. If there is no such column, it will use vector of dummy values (NAs or fills).

Value

An object of class `h2h_mat` which is a [matrix](#) with row names indicating first player in matchup, col names - second and values - Head-to-Head values.

Head-to-Head value

Head-to-Head value is a summary statistic of direct confrontation between two players. It is assumed that this value can be computed based only on the players' [matchups](#). In other words, every game is converted into series of "subgames" between ordered pairs of players (including selfplay) which is stored as [widecr](#) object. After that, summary of item, defined by columns `player1` and `player2`, is computed using [summarise_item\(\)](#).

That said, name-value pairs of Head-to-Head functions should be defined as for [summarise_item\(\)](#) applied to data with columns `game`, `player1`, `score1`, `player2`, `score2`.

See Also

[Long format](#) of Head-to-Head values.

[Common Head-to-Head functions](#).

Examples

```
# Only first function is used
ncaa2005 %>%
  h2h_mat(
    mean_score = mean(score1),
    mean_abs_score = mean(abs(score1 - score2))
  )

ncaa2005[-(1:2), ] %>%
  h2h_mat(mean_score = mean(score1), fill = 0)
```

hp_survey

Results of Harry Potter Books Survey

Description

hp_survey contains results of the survey with a goal to collect data enough to rate Harry Potter books.

Usage

```
hp_survey
```

Format

A **tibble** with answers from 182 respondents and the following columns:

- **person** <int>: Identifier of a person.
- **book** <chr>: Identifier of a Harry Potter book. Its values are of the form "HP_x" where "x" represents book's number in the series (from 1 to 7).
- **score** <chr>: Book's score. Can be one of "1 - Poor", "2 - Fair", "3 - Good", "4 - Very Good", "5 - Excellent".

Rows are ordered by person and then by book identifier.

Details

Survey was done via [Google Forms](#) service. To participate in it, respondent is asked to log in into her/his Google account (to ensure that one person takes part only once). It was popularized mostly among R users via [R-bloggers](#) and [Twitter](#).

At the beginning of the survey, there was the following text:

This is a survey with goal to collect data enough to rate Harry Potter books. Data will be made public with complete anonymity of respondents. Please, take part only if you have read all seven original J. K. Rowling Harry Potter books and are willing to give an honest feedback about your impressions.

Analyzed books were coded with the following names:

- “HP and the Philosopher’s (Sorcerer’s) Stone (#1)”.
- “HP and the Chamber of Secrets (#2)”.
- “HP and the Prisoner of Azkaban (#3)”.
- “HP and the Goblet of Fire (#4)”.
- “HP and the Order of the Phoenix (#5)”.
- “HP and the Half-Blood Prince (#6)”.
- “HP and the Deathly Hallows (#7)”.

Survey had the following procedure:

- At first, respondent is asked to choose the first element in the randomly shuffled list of number from 1 to 127. This simulates the random generation of books subset in the next question.
- Next he/she is presented with a question "What is your impression of these Harry Potter BOOKS?" (singular if there is one book) and the following question grid:
 - Rows represent randomly shuffled subset of books corresponding to the number chosen in the first step.
 - Columns contain the following scale of answers: “1 - Poor”, “2 - Fair”, “3 - Good”, “4 - Very Good”, “5 - Excellent”. Respondent is asked and allowed to choose only one answer per book (every book should be rated).

item-summary

Compute item summary

Description

Functions for computing item summary, i.e. some summary measurements (of arbitrary nature) of item (one or more columns) present in data frame.

Usage

```
summarise_item(tbl, item, ..., .prefix = "")
```

```
summarise_game(tbl, ..., .prefix = "")
```

```
summarise_player(tbl, ..., .prefix = "")
```

```
summarize_item(tbl, item, ..., .prefix = "")
```

```
summarize_game(tbl, ..., .prefix = "")
```

```
summarize_player(tbl, ..., .prefix = "")
```

Arguments

<code>tbl</code>	Data frame.
<code>item</code>	Character vector of columns to group by.
<code>...</code>	Name-value pairs of summary functions (as in dplyr::summarise).
<code>.prefix</code>	A string to be added to all summary functions' names.

Details

Basically, `summarise_item()` performs the following steps:

- Group `tbl` by columns stored in `item`. **Note** that starting from 0.8.0 version of `dplyr` this might give a warning in case of implicit NAs in factor columns (NA present in column values but not in its levels) suggesting to add NA to levels.
- Apply `dplyr`'s `summarise()`.
- Ungroup result.
- Convert to [tibble](#).
- Add `.prefix` to names of summary functions.

`summarise_game()` and `summarise_player()` are wrappers for `summarise_item()` using `item = "game"` and `item = "player"` respectively.

Value

Output of `summarise()` as not grouped tibble.

See Also

Common item [summary functions](#) for competition results.

[Join item summary](#)

Examples

```
ncaa2005 %>%
  dplyr::mutate(game_type = game %% 2) %>%
  summarise_item(c("game_type", "player"), mean_score = mean(score))

ncaa2005 %>%
  summarise_game(mean_score = mean(score), min_score = min(score))
```

item-summary-join	<i>Join item summary</i>
-------------------	--------------------------

Description

Functions for joining summary data to data frame. They perform respective variant of [summarise item functions](#) and then [left join](#) to the input its result (by `item` columns).

Usage

```
join_item_summary(tbl, item, ..., .prefix = "")
```

```
join_game_summary(tbl, ..., .prefix = "")
```

```
join_player_summary(tbl, ..., .prefix = "")
```

Arguments

<code>tbl</code>	Data frame.
<code>item</code>	Character vector of columns to group by.
<code>...</code>	Name-value pairs of summary functions (as in dplyr::summarise).
<code>.prefix</code>	A string to be added to all summary functions' names.

Details

`join_game_summary()` and `join_player_summary()` are wrappers for `join_item_summary()` using `item = "game"` and `item = "player"` respectively.

Value

Result of `left_join()` to the input data frame.

See Also

[Compute item summary](#)

Common item [summary functions](#) for competition results.

Examples

```
ncaa2005 %>% join_player_summary(player_mean_score = mean(score))
```

levels2	<i>Levels of vector</i>
---------	-------------------------

Description

Extension of `levels()` function. If `levels(x)` is not NULL, it is returned. Otherwise, character representation of unique sorted values is returned (with NA treated based on `na.last` as in `sort()`).

Usage

```
levels2(x, na.last = TRUE)
```

Arguments

<code>x</code>	An object of interest.
<code>na.last</code>	Argument for controlling the treatment of NAs. See <code>sort()</code> .

Examples

```
fac_vec <- factor(c("a", "b"), levels = c("a", "b", "c"))
levels2(fac_vec)

levels2(c(10, 1, 2, NA, 11))
```

longcr	<i>Long format of competition results</i>
--------	---

Description

Functions for dealing with competition results in long format.

Usage

```
is_longcr(cr_data)

as_longcr(cr_data, repair = TRUE, ...)

## S3 method for class 'longcr'
as_tibble(x, ...)
```

Arguments

<code>cr_data</code>	Data of competition results (convertible to tabular).
<code>repair</code>	Whether to repair input.
<code>...</code>	Additional arguments to be passed to or from methods.
<code>x</code>	Object to be converted to <code>tibble</code> .

Details

`as_longcr()` is S3 method for converting data to `longcr`. When using **default** method if `repair` is TRUE it also tries to fix possible problems (see "Repairing"). If `repair` is FALSE it converts `cr_data` to `tibble` and adds `longcr` class to it.

When applying `as_longcr()` to proper (check via `is_widecr()` is made) `widecr` object, conversion is made:

- If there is column `game` then it is used as game identifier. Else treat every row as separate game data.
- Every "player"-`"score"` pair for every game is converted to separate row with adding the appropriate extra columns.
- Result is arranged by game and identifier of a "player"-`"score"` pair (extra symbols after "player" and "score" strings in input column names) in increasing order.
- If `repair` is TRUE then repair is done.

For appropriate `longcr` objects `as_longcr()` returns its input and throws error otherwise.

Value

`is_longcr()` returns TRUE if its argument is appropriate object of class `longcr`: it should inherit classes `longcr`, `tbl_df` (in other words, to be `tibble`) and have "game", "player", "score" among column names.

`as_longcr()` returns an object of class `longcr`.

`as_tibble()` applied to `longcr` object drops `longcr` class.

Long format of competition results

It is assumed that competition consists from multiple games (matches, comparisons, etc.). One game can consist from **variable** number of players. Inside a game all players are treated equally. In every game every player has some score: the value of arbitrary nature that fully characterizes player's performance in particular game (in most cases it is some numeric value).

`longcr` inherits from `tibble`. Data should have at least three columns with the following names:

- `game` - game identifier.
- `player` - player identifier.
- `score` - score of particular player in particular game.

Extra columns are allowed. **Note** that if object is converted to `widecr`, they will be dropped. So it is better to store extra information about "game"-`"player"` pair as list-column "score" which will stay untouched.

Repairing

Option `repair = TRUE` (default) in `as_longcr()` means that its result is going to be repaired with following actions:

- Detect columns exactly matching "game", "player" or "score". Those are used in the output. If all are detected matched columns are put in the beginning. Other columns are preserved.

- If not all columns were exactly matched, detect first columns with names containing "game", "player" or "score" (ignoring case). If there are many matching names for one output name then the first one is used. In case of imperfect match, message is given. All other columns are treated as "extra".
- If some legitimate names aren't detected, respective columns are created and filled with NA_integer_. Also a message is given.
- If in one game some player listed more than once, the first record is taken.
- Return the tibble with at least 3 appropriate for longcr columns and column names.

See Also

[Wide format](#)

Examples

```
# Repairing example
cr_data <- data.frame(
  playerscoregame_ID = rep(1:5, times = 2),
  gameId = rep(1:5, each = 2),
  scoreS = 31:40,
  scoreSS = 41:50
)
cr_data_long <- as_longcr(cr_data, repair = TRUE)

is_longcr(cr_data_long)

as_tibble(cr_data_long)
```

ncaa2005

Example competition results from 2005 NCAA football season

Description

ncaa2005 is an example competition results of an isolated group of Atlantic Coast Conference teams provided in book "Who's #1" by Langville and Meyer.

Usage

```
ncaa2005
```

Format

An object of class [longcr](#) containing information about 10 games.

pairgames

Competition results with games between two players

Description

Functions for competition results with games between two players.

Usage

```
to_pairgames(cr_data)
```

```
is_pairgames(cr_data)
```

Arguments

`cr_data` Competition results in format ready for `as_longcr()`.

Details

Pairgames is a term for competition results with games between two players.

`to_pairgames()` is a function that converts competition results into pairwise games: it drops games with one player and for every game with 3 and more players this function transforms it into set of separate games between unordered pairs of players. In other words the result is a set of unordered **matchups (as different games)** between different players.

Important notes:

- New game identifiers are integers, order of which respects order of games stored in `cr_data` (based on first occurrence in long format). There is no particular order in subgames of games with 3 and more players.
- Order in which players are assigned to `player1` or `player2` column in general shouldn't agree with any order in `cr_data`.
- Any column except `game`, `player` and `score` will be dropped after conversion to `longcr`.
- NA and NaN in `players` are allowed. They are treated as different players.
- `to_pairgames()` is rather compute-intensive and can take much time for competition results with many games.

Value

`to_pairgames()` returns a competition results of pairwise games as `widecr` object with two players.

`is_pairgames()` returns a boolean value of whether `cr_data` contains only games between two players.

Examples

```
cr_data <- data.frame(
  game = c(rep(1:5, each = 3), 6),
  player = c(rep(1:5, times = 3), 1),
  score = 101:116,
  extraCol = -(1:16)
)

to_pairgames(cr_data)

# Missing values
cr_data_na <- data.frame(
  game = rep(1L, 3),
  player = c(1, NA, NA),
  score = 1:3
)
to_pairgames(cr_data_na)

# Checks
is_pairgames(cr_data)
is_pairgames(to_pairgames(cr_data))
```

summary_funs

Common item summary functions

Description

List of commonly used functions for summarising competition results.

Usage

```
summary_funs
```

Format

An object of class `list` of length 8.

Details

`summary_funs` is a named list of [expressions](#) representing commonly used expressions of summary functions for summarising competition results with `summarise_item()`. Names of the elements will be used as summary names. It is designed primarily to be used with [long format](#) of competition results. To use them inside `summarise_item()` use [unquoting](#) mechanism from `rlang` package.

Currently present functions:

- **min_score** - `min(score)`.
- **max_score** - `max(score)`.
- **mean_score** - `mean(score)`.

- **median_score** - median(score).
- **sd_score** - sd(score).
- **sum_score** - sum(score).
- **num_games** - length(unique(game)).
- **num_players** - length(unique(player)).

Note that it is generally better to subset `summary_funs` using names rather than indices because the order of elements might change in future versions.

See Also

[Compute item summary](#), [Join item summary](#)

Examples

```
ncaa2005 %>% summarise_game(!!!summary_funs, .prefix = "game_")
```

widecr

Wide format of competition results

Description

Functions for dealing with competition results in wide format.

Usage

```
is_widecr(cr_data)

as_widecr(cr_data, repair = TRUE, ...)

## S3 method for class 'widecr'
as_tibble(x, ...)
```

Arguments

<code>cr_data</code>	Data of competition results (convertible to tabular).
<code>repair</code>	Whether to repair input.
<code>...</code>	Additional arguments to be passed to or from methods.
<code>x</code>	Object to be converted to tibble .

Details

`as_widecr()` is S3 method for converting data to widecr. When using **default** method if `repair` is TRUE it also tries to fix possible problems (see "Repairing"). If `repair` is FALSE it converts `cr_data` to [tibble](#) and adds widecr class to it.

When applying `as_widecr()` to proper (check via `is_longcr()` is made) longcr object, conversion is made:

- All columns except "game", "player" and "score" are dropped.
- Conversion from long to wide format is made. The number of "player"- "score" pairs is taken as the maximum number of players in game. If not all games are played between the same number of players then there will be NA's in some pairs. Column game is preserved in output and is used for arranging in increasing order.

For appropriate widecr objects `as_widecr` returns its input and throws error otherwise.

Value

`is_widecr()` returns TRUE if its argument is appropriate object of class widecr: it should inherit classes widecr, `tbl_df` (in other words, to be [tibble](#)) and have complete pairs of "player"- "score" columns where pair is detected by **digits** after strings "player" and "score" respectively. Columns of "player" and "score" types shouldn't have any extra symbols except type name and digits after it. All other columns are considered as "extra columns".

`as_widecr()` returns an object of class widecr.

[as_tibble\(\)](#) applied to widecr object drops widecr class.

Wide format of competition results

It is assumed that competition consists from multiple games (matches, comparisons, etc.). One game can consist only from **constant** number of players. Inside a game all players are treated equally. In every game every player has some score: the value of arbitrary nature that fully characterizes player's performance in particular game (in most cases it is some numeric value).

widecr inherits from `tibble`. Data should be organized in pairs of columns "player"- "score". Identifier of a pair should go after respective keyword and consist only from digits. For example: `player1, score1, player2, score2`. Order doesn't matter. Extra columns are allowed.

To account for R standard string ordering, identifiers of pairs should be formatted with leading zeros (when appropriate). For example: `player01, score01, ..., player10, score10`.

Column `game` for game identifier is optional. If present it will be used in conversion to longcr format via `as_longcr()`.

Repairing

Option `repair = TRUE` (default) in `as_widecr()` means that its result is going to be repaired with following actions:

- Detect columns with names containing "player" or "score" (ignoring case). All other columns are treated as "extra".

- Extract first occurrence of "player" or "score" (ignoring case) from names of detected columns. Everything after extracted word is treated as identifier of "player"- "score" pair.
- Convert these identifiers to numeric form with `as.integer(as.factor(...))`.
- Convert identifiers once again to character form with possible leading zeros (to account for R standard string ordering).
- Spread pairs to appropriate columns with possible column adding (which were missed in original pairs based on information of pair identifier) with `NA_integer_`.
- **Note** that if there is column game (exactly matched) it is placed as first column. **Note** that the order (and numeration) of pairs can change.

See Also

[Long format](#)

Examples

```
cr_data <- data.frame(
  playerA = 1:10,
  playerB = 2:11,
  scoreC = 11:20,
  scoreB = 12:21,
  otherColumn = 101:110
)
cr_data_wide <- as_widecr(cr_data, repair = TRUE)

is_widecr(cr_data_wide)

as_tibble(cr_data_wide)
```

Index

* datasets

- h2h_funs, 6
 - hp_survey, 10
 - ncaa2005, 16
 - summary_funs, 18
- as_longcr (longcr), 14
- as_longcr(), 5, 7, 9, 17, 20
- as_tibble(), 8, 15, 20
- as_tibble.h2h_long (h2h_long), 7
- as_tibble.longcr (longcr), 14
- as_tibble.widecr (widecr), 19
- as_widecr (widecr), 19
- Common Head-to-Head functions, 8, 9
- common Head-to-Head functions, 3
- common item summary functions, 2
- comperes (comperes-package), 2
- comperes-package, 2
- complete(), 7
- Compute item summary, 13, 19
- convert-pair-value, 3
- dplyr::near(), 6
- dplyr::summarise, 12, 13
- drop_na(), 8
- expressions, 6, 18
- get_matchups, 5
- h2h_funs, 6
- h2h_long, 7, 9
- h2h_long(), 6
- h2h_mat, 8, 8
- h2h_mat(), 6
- hp_survey, 10
- is_longcr (longcr), 14
- is_longcr(), 20
- is_pairgames (pairgames), 17
- is_widecr (widecr), 19
- is_widecr(), 15
- item summaries, 2
- item-summary, 11
- item-summary-join, 13
- join, 2
- Join item summary, 12, 19
- join_game_summary (item-summary-join), 13
- join_item_summary (item-summary-join), 13
- join_player_summary (item-summary-join), 13
- left join, 13
- levels(), 14
- levels2, 14
- levels2(), 4, 7
- long, 5, 7
- Long format, 5, 6, 9, 21
- long format, 2, 9, 18
- long_to_mat (convert-pair-value), 3
- longcr, 14, 16, 17
- mat_to_long (convert-pair-value), 3
- matchups, 7–9, 17
- matrix, 5, 9
- Matrix format, 5, 6, 8
- matrix format, 3
- ncaa2005, 16
- pairgames, 17
- sort(), 14
- summarise item functions, 13
- summarise_game (item-summary), 11
- summarise_item (item-summary), 11
- summarise_item(), 8, 9, 18
- summarise_player (item-summary), 11

summarize_game (item-summary), 11
summarize_item (item-summary), 11
summarize_player (item-summary), 11
summary functions, 12, 13
summary_funs, 18

tibble, 2, 5, 7, 8, 10, 12, 14, 15, 19, 20
to_h2h_long (h2h_long), 7
to_h2h_mat (h2h_mat), 8
to_pairgames (pairgames), 17
transforms, 9

unquoting, 2, 6, 18

Wide format, 16
wide format, 2
widecr, 5, 8, 9, 15, 17, 19