Package 'nflseedR'

May 9, 2026

Title Functions to Efficiently Simulate and Evaluate NFL Seasons

Version 2.0.2

Description A set of functions to simulate National Football League seasons including the sophisticated tie-breaking procedures.

License MIT + file LICENSE

URL https://nflseedr.com, https://github.com/nflverse/nflseedR

BugReports https://github.com/nflverse/nflseedR/issues

Depends R (>= 4.1.0)

Imports cli, data.table, dplyr, furrr, future, gsubfn, lifecycle, nflreadr (>= 1.1.3), progressr, purrr, rlang, tibble, tidyr

Suggests gt (>= 0.9.0), knitr, nflplotR (>= 1.2.0), rmarkdown, scales, testthat (>= 3.0.0)

Config/testthat/edition 3

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

NeedsCompilation no

Author Sebastian Carl [cre, aut, cph], Lee Sharpe [aut]

Maintainer Sebastian Carl <mrcaseb@gmail.com>

Repository CRAN

Date/Publication 2025-11-16 12:20:02 UTC

Contents

`compute_conference_seeds`

Compute NFL Playoff Seedings using Game Results and Divisional Rankings

---

Description

Compute NFL Playoff Seedings using Game Results and Divisional Rankings

Usage

```
compute_conference_seeds(
  teams,
  h2h = NULL,
  tiebreaker_depth = 3,
  .debug = FALSE,
  playoff_seeds = 7
)
```

Arguments

`teams`	The division standings data frame as computed by `compute_division_ranks`
`h2h`	A data frame that is used for head-to-head tiebreakers across the tie-breaking functions. It is computed by the function `compute_division_ranks`.
`tiebreaker_depth`	A single value equal to 1, 2, or 3. The default is 3. The value controls the depth of tiebreakers that shall be applied. The deepest currently implemented tiebreaker is strength of schedule. The following values are valid: **tiebreaker_depth = 1** Break all ties with a coinflip. Fastest variant.

tiebreaker_depth = 2 Apply head-to-head and division win percentage tiebreakers. Random if still tied.

tiebreaker_depth = 3 Apply all tiebreakers through strength of schedule. Random if still tied.

`.debug` Either `TRUE` or `FALSE`. Controls whether additional messages are printed to the console showing what the tie-breaking algorithms are currently performing.

`playoff_seeds` Number of playoff teams per conference (increased in 2020 from 6 to 7).

Value

A data frame of division standings including playoff seeds and the week in which the season ended for the respective team (`exit`).

A list of two data frames:

standings Division standings including playoff seeds.

h2h A data frame that is used for head-to-head tiebreakers across the tie-breaking functions.

See Also

The examples on the package website

Examples

```
# Change some options for better output
old <- options(list(digits = 3, tibble.print_min = 64))
library(dplyr, warn.conflicts = FALSE)

try({#to avoid CRAN test problems
s <- nflseedR::load_sharpe_games() |>
  dplyr::filter(season %in% 2019:2020) |>
  dplyr::select(sim = season, game_type, week, away_team, home_team, result) |>
  nflseedR::compute_division_ranks()
  nflseedR::compute_conference_seeds(s, h2h = s$h2h) |>
  purrr::pluck("standings")
})

# Restore old options
options(old)
```

`compute_division_ranks` *Compute NFL Division Rankings using Game Results*

Description

Compute NFL Division Rankings using Game Results

Usage

```
compute_division_ranks(
  games,
  teams = NULL,
  tiebreaker_depth = 3,
  .debug = FALSE,
  h2h = NULL
)
```

Arguments

games	A data frame containing real or simulated game scores. The following variables are required: **sim** A simulation ID. Normally 1 - n simulated seasons. **game_type** One of 'REG', 'WC', 'DIV', 'CON', 'SB' indicating if a game was a regular season game or one of the playoff rounds. **week** The week of the corresponding NFL season. **away_team** Team abbreviation of the away team (please see divisions for valid team abbreviations). **home_team** Team abbreviation of the home team (please see divisions for valid team abbreviations). **result** Equals home score - away score.
teams	This parameter is optional. If it is NULL the function will compute it internally, otherwise it has to be a data frame of all teams contained in the games data frame repeated for each simulation ID (sim). The following variables are required: **sim** A simulation ID. Normally 1 - n simulated seasons. **team** Team abbreviation of the team (please see divisions for valid team abbreviations). **conf** Conference abbreviation of the team (please see divisions for valid team abbreviations). **division** Division of the team (please see divisions for valid division names).
tiebreaker_depth	A single value equal to 1, 2, or 3. The default is 3. The value controls the depth of tiebreakers that shall be applied. The deepest currently implemented tiebreaker is strength of schedule. The following values are valid: **tiebreaker_depth = 1** Break all ties with a coinflip. Fastest variant. **tiebreaker_depth = 2** Apply head-to-head and division win percentage tiebreakers. Random if still tied. **tiebreaker_depth = 3** Apply all tiebreakers through strength of schedule. Random if still tied.
.debug	Either TRUE or FALSE. Controls whether additional messages are printed to the console showing what the tie-breaking algorithms are currently performing.
h2h	A data frame that is used for head-to-head tiebreakers across the tie-breaking functions. It is computed by the function compute_division_ranks.

Value

A list of two data frames:

standings Division standings.

h2h A data frame that is used for head-to-head tiebreakers across the tie-breaking functions.

See Also

The examples on the package website

Examples

```
# Change some options for better output
old <- options(list(digits = 3, tibble.print_min = 64))
library(dplyr, warn.conflicts = FALSE)

try({#to avoid CRAN test problems
nflseedR::load_sharpe_games() |>
  dplyr::filter(season %in% 2019:2020) |>
  dplyr::select(sim = season, game_type, week, away_team, home_team, result) |>
  nflseedR::compute_division_ranks() |>
  purrr::pluck("standings")
})

# Restore old options
options(old)
```

---

compute_draft_order *Compute NFL Draft Order using Game Results and Divisional Rankings*

Description

Compute NFL Draft Order using Game Results and Divisional Rankings

Usage

```
compute_draft_order(
  teams,
  games,
  h2h = NULL,
  tiebreaker_depth = 3,
  .debug = FALSE
)
```

Arguments

`teams`	The division standings data frame including playoff seeds as computed by `compute_conference_seeds`
`games`	A data frame containing real or simulated game scores. The following variables are required: **sim** A simulation ID. Normally 1 - n simulated seasons. **game_type** One of 'REG', 'WC', 'DIV', 'CON', 'SB' indicating if a game was a regular season game or one of the playoff rounds. **week** The week of the corresponding NFL season. **away_team** Team abbreviation of the away team (please see `divisions` for valid team abbreviations). **home_team** Team abbreviation of the home team (please see `divisions` for valid team abbreviations). **result** Equals home score - away score.
`h2h`	A data frame that is used for head-to-head tiebreakers across the tie-breaking functions. It is computed by the function `compute_division_ranks`.
`tiebreaker_depth`	A single value equal to 1, 2, or 3. The default is 3. The value controls the depth of tiebreakers that shall be applied. The deepest currently implemented tiebreaker is strength of schedule. The following values are valid: **tiebreaker_depth = 1** Break all ties with a coinflip. Fastest variant. **tiebreaker_depth = 2** Apply head-to-head and division win percentage tiebreakers. Random if still tied. **tiebreaker_depth = 3** Apply all tiebreakers through strength of schedule. Random if still tied.
`.debug`	Either `TRUE` or `FALSE`. Controls whether additional messages are printed to the console showing what the tie-breaking algorithms are currently performing.

Value

A data frame of standings including the final draft pick number and the variable `exit` which indicates the week number of the teams final game (Super Bowl Winner is one week higher).

See Also

The examples on the package website

Examples

```
# Change some options for better output
old <- options(list(digits = 3, tibble.print_min = 64))
library(dplyr, warn.conflicts = FALSE)

try({#to avoid CRAN test problems
games <-
  nflseedR::load_sharpe_games() |>
  dplyr::filter(season %in% 2018:2019) |>
```

```
  dplyr::select(sim = season, game_type, week, away_team, home_team, result)

s <- games |> nflseedR::compute_division_ranks()
s <- nflseedR::compute_conference_seeds(s, h2h = s$h2h, playoff_seeds = 6)
nflseedR::compute_draft_order(s, games = games, h2h = s$h2h)
})

# Restore old options
options(old)
```

`dictionary_games` *Data Dictionary: Simulations | Games*

Description

A dataframe containing the data dictionary of the simulation output table "games"

Usage

```
dictionary_games
```

Format

An object of class `data.frame` with 9 rows and 2 columns.

See Also

https://nflseedr.com/articles/articles/nflsim2.html#simulation-output

---

`dictionary_game_summary`
Data Dictionary: Simulations | Game Summary

Description

A dataframe containing the data dictionary of the simulation output table "game_summary"

Usage

```
dictionary_game_summary
```

Format

An object of class `data.frame` with 11 rows and 2 columns.

See Also

https://nflseedr.com/articles/articles/nflsim2.html#simulation-output

dictionary_overall *Data Dictionary: Simulations | Overall*

Description

A dataframe containing the data dictionary of the simulation output table "overall"

Usage

```
dictionary_overall
```

Format

An object of class `data.frame` with 11 rows and 2 columns.

See Also

https://nflseedr.com/articles/articles/nflsim2.html#simulation-output

dictionary_standings *Data Dictionary: Simulations | Standings*

Description

A dataframe containing the data dictionary of the simulation output table "standings"

Usage

```
dictionary_standings
```

Format

An object of class `data.frame` with 21 rows and 2 columns.

See Also

https://nflseedr.com/articles/articles/nflsim2.html#simulation-output

dictionary_team_wins *Data Dictionary: Simulations | Team Wins*

Description

A dataframe containing the data dictionary of the simulation output table "team_wins"

Usage

```
dictionary_team_wins
```

Format

An object of class `data.frame` with 4 rows and 2 columns.

See Also

https://nflseedr.com/articles/articles/nflsim2.html#simulation-output

divisions *NFL team names and the conferences and divisions they belong to*

Description

NFL team names and the conferences and divisions they belong to

Usage

```
divisions
```

Format

A data frame with 36 rows and 4 variables containing NFL team level information, including franchises in multiple cities:

team Team abbreviation

conf Conference abbreviation

division Division name

sdiv Division abbreviation

This data frame is created using the `teams_colors_logos` data frame of the `nflfastR` package. Please see `data-raw/divisions.R` for the code to create this data.

Examples

```
str(divisions)
```

fmt_pct_special *Format Numerical Values to Special Percentage Strings*

Description

This function formats numeric vectors with values between 0 and 1 into percentage strings with special specifications. Those specifications are:

- 0 and 1 are converted to "0%" and "100%" respectively (takes machine precision into account)
- all other values < 0.01 are converted to "<1%"
- all other values between 0.01 and 0.995 are rounded to percentages without decimals
- values between 0.995 and 0.999 are rounded to percentages with 1 decimal
- values between 0.999 and 1 are converted to ">99.9%" unless closer to 1 than machine precision.

Usage

```
fmt_pct_special(x)
```

Arguments

x	A vector of numerical values

Value

A character vector

Examples

```
x <- c(0, 0.004, 0.009, 0.011, 0.9, 0.98, 0.994,
       .995, .9989, .999, .9991, .99999999)
fmt <- fmt_pct_special(x)
data.frame(x = x, fmt = fmt)
```

nfl_simulations *Simulate an NFL Season*

Description

Simulate NFL games based on a user provided games/schedule object that holds matchups with and without results. Missing results are computed using the argument `compute_results` and possible further arguments to `compute_results` in `...` (please see simulations_verify_fct for further information.).

It is possible to let the function calculate playoff participants and simulate the post-season. The code is also developed for maximum performance and allows parallel computation by splitting the number of simulations into chunks and calling the appropriate future::plan. Progress updates can be activated by calling progressr::handlers before the start of the simulations. Please see the below given section "Details" for further information.

Usage

```
nfl_simulations(
  games,
  compute_results = nflseedR_compute_results,
  ...,
  playoff_seeds = 7L,
  simulations = 10000L,
  chunks = 8L,
  byes_per_conf = 1L,
  tiebreaker_depth = c("SOS", "PRE-SOV", "RANDOM"),
  sim_include = c("DRAFT", "REG", "POST"),
  verbosity = c("MIN", "MAX", "NONE")
)
```

Arguments

`games` A data frame containing real or simulated game scores. Outside of simulations, this is simply the output of nflreadr::load_schedules. The following variables are required as a minimum:

sim or season A season or simulation ID. Normally 1 - n simulated seasons. If both sim and season are included, a warning is triggered and work continues with sim.

game_type One of 'REG', 'WC', 'DIV', 'CON', 'SB' indicating if a game was a regular season game or one of the playoff rounds.

week The week of the corresponding NFL season.

away_team Team abbreviation of the away team (please see `divisions` for valid team abbreviations).

home_team Team abbreviation of the home team (please see `divisions` for valid team abbreviations).

result Equals home score - away score.

If tiebreakers beyond SOS are to be used, then the actual scores of the home (`home_score`) and away (`away_score`) teams must also be available.

`compute_results`

Defaults to the nflseedR function `nflseedR_compute_results`. A function to compute results of games. Uses team, schedule, and week number as arguments. Please see simulations_verify_fct for further information.

`...` Additional parameters passed on to the function `compute_results`.

`playoff_seeds` If `NULL` (the default), will compute all 16 conference ranks. This means, the function applies conference tiebreakers to all conference ranks. For better performance, it is possible to set this to a value < 16 to make the function skip tiebreakers of those conference ranks.

`simulations` Equals the number of times the given NFL season shall be simulated

`chunks` The number of chunks `simulations` should be split into and potentially be processed parallel. This parameter controls the number of simulations per chunk. There is no obvious way to determine the ideal number of chunks in advance

because there are too many dependencies on the hardware. Too many chunks can be just as slow as too few. It is therefore up to the user to determine the optimum number themselves.

`byes_per_conf` The number of teams with a playoff bye week per conference. This number influences the number of wildcard games that are simulated.

`tiebreaker_depth`
One of `"SOS"`, `"PRE-SOV"`, `"POINTS"` or `"RANDOM"`. Controls which tiebreakers are to be applied. The implemented tiebreakers are documented here https://nflseedr.com/articles/tiebreaker.html. The values mean:

- `"SOS"` (default): Apply all tiebreakers through Strength of Schedule. If there are still remaining ties, break them through coin toss.
- `"PRE-SOV"`: Apply all tiebreakers before Strength of Victory. If there are still remaining ties, break them through coin toss. Why Pre SOV? It's the first tiebreaker that requires knowledge of how OTHER teams played.
- `"POINTS"`: Apply all tiebreakers through point differential. If there are still remaining ties, break them through coin toss. This will go beyond SOS and requires knowledge of points scored and points allowed. As this is not usually part of season simulations, caution is advised in this case. These tiebreakers should only be used if the scores are real or are deliberately simulated.
- `"RANDOM"`: Breaks all tiebreakers with a coin toss. I don't really know, why I allow this...

`sim_include` One of `"REG"`, `"POST"`, `"DRAFT"` (the default). Simulation will behave as follows:

- `"REG"`: Simulate the regular season and compute standings, division ranks, and playoff seeds
- `"POST"`: Do `"REG"` + simulate the postseason
- `"DRAFT"` (default): Do `"POST"` + compute draft order

`verbosity` One of `"MIN"`, `"MAX"`, or `"NONE"` allowing the user to set the grade of verbosity of status reports. They mean:

- `"MIN"` (default): Prints main steps of the process.
- `"MAX"`: Prints all steps of the complete tiebreaking process.
- `"NONE"`: No status reports at all. Do this to maximize the performance.

Details

More Speed Using Parallel Processing:

We recommend choosing a default parallel processing method and saving it as an environment variable in the R user profile to make sure all futures will be resolved with the chosen method by default. This can be done by following the below given steps.

First, run the below line and the user profile should be opened automatically. If you haven't saved any environment variables yet, this will be an empty file.

```
usethis::edit_r_environ()
```

In the opened file add the next line, then save the file and restart your R session. Please note that this example sets "multisession" as default. For most users this should be the appropriate plan but please make sure it truly is.

```
R_FUTURE_PLAN="multisession"
```

After the session is freshly restarted please check if the above method worked by running the next line. If the output is `FALSE` you successfully set up a default non-sequential `future::plan()`. If the output is `TRUE` all functions will behave like they were called with `purrr::map()` and **NOT** in multisession.

```
inherits(future::plan(), "sequential")
```

For more information on possible plans please see the future package Readme.

Get Progress Updates while Functions are Running:
nflseedR is able to show progress updates using `progressr::progressor()` if they are turned on before the function is called. There are at least two basic ways to do this by either activating progress updates globally (for the current session) with

```
progressr::handlers(global = TRUE)
```

or by piping the function call into `progressr::with_progress()`:

```
nflseedR::nfl_simulations(
  games = nflseedR::sims_games_example,
  simulations = 4,
  chunks = 2
) |>
  progressr::with_progress()
```

For more information how to work with progress handlers please see progressr::progressr.

Reproducible Random Number Generation (RNG):
It is to be expected that some form of random number generation is required in the function in argument `compute_results`. For better performance, nflseedR uses the furrr package to parallelize chunks. furrr functions are guaranteed to generate the exact same sequence of random numbers given the same initial seed if, and only if, the initial seed is of the type "L'Ecuyer-CMRG". So if you want a consistent seed to be used across all chunks, you must ensure that the correct type is specified in `set.seed`, e.g. with the following code

```
set.seed(5, "L'Ecuyer-CMRG")
```

It is sufficient to set the seed before nfl_simulations is called. To check that the type has been set correctly, you can use the following code.

```
RNGkind()
"L'Ecuyer-CMRG" "Inversion"     "Rejection"

# Should be a integer vector of length 7
.Random.seed
10407  1157214768 -1674567567 -1532971138 -1249749529  1302496508  -253670963
```

For more information, please see the section "Reproducible random number generation (RNG)" in furrr::furrr_options.

Value

An `nflseedR_simulation` object containing a list of 6 data frames with the results of all simulated games, the final standings in each simulated season, summary statistics across all simulated seasons, and the simulation parameters. For a full list, please see the package website.

See Also

The examples on the package website

The method `summary.nflseedR_simulation()` that creates a pretty html summary table.

Examples

```
library(nflseedR)

# Activate progress updates
# progressr::handlers(global = TRUE)

# Parallel processing can be activated via the following line
# future::plan("multisession")

sim <- nflseedR::nfl_simulations(
  games = nflseedR::sims_games_example,
  simulations = 4,
  chunks = 2
)

# Overview output
str(sim, max.level = 3)
```

---

nfl_standings *Compute NFL Standings*

Description

Compute NFL Standings

Usage

```
nfl_standings(
  games,
  ...,
  ranks = c("CONF", "DIV", "DRAFT", "NONE"),
  tiebreaker_depth = c("SOS", "PRE-SOV", "POINTS", "RANDOM"),
  playoff_seeds = NULL,
  verbosity = c("MIN", "MAX", "NONE")
)
```

Arguments

`games` A data frame containing real or simulated game scores. Outside of simulations, this is simply the output of nflreadr::load_schedules. The following variables are required as a minimum:

sim or season A season or simulation ID. Normally 1 - n simulated seasons. If both sim and season are included, a warning is triggered and work continues with sim.

game_type One of 'REG', 'WC', 'DIV', 'CON', 'SB' indicating if a game was a regular season game or one of the playoff rounds.

week The week of the corresponding NFL season.

away_team Team abbreviation of the away team (please see `divisions` for valid team abbreviations).

home_team Team abbreviation of the home team (please see `divisions` for valid team abbreviations).

result Equals home score - away score.

If tiebreakers beyond SOS are to be used, then the actual scores of the home (`home_score`) and away (`away_score`) teams must also be available.

`...` currently not used

`ranks` One of `"DIV"`, `"CONF"`, `"DRAFT"`, or `"NONE"` to specify which ranks - and thus the associated tiebreakers - are to be determined.

- `"DIV"`: Adds the division ranking variable `div_rank`
- `"CONF"` (default): `"DIV"` + the conference variable `conf_rank`. For better performance, it is possible to set `playoff_seeds` to a value < 16 to make the function skip tiebreakers of irrelevant conference ranks.
- `"DRAFT"`: `"CONF"` + the draft variable `draft_rank`. This is the actual pick in the draft based off game results. No trades of course.

`tiebreaker_depth`

One of `"SOS"`, `"PRE-SOV"`, `"POINTS"` or `"RANDOM"`. Controls which tiebreakers are to be applied. The implemented tiebreakers are documented here https://nflseedr.com/articles/tiebreaker.html. The values mean:

- `"SOS"` (default): Apply all tiebreakers through Strength of Schedule. If there are still remaining ties, break them through coin toss.
- `"PRE-SOV"`: Apply all tiebreakers before Strength of Victory. If there are still remaining ties, break them through coin toss. Why Pre SOV? It's the first tiebreaker that requires knowledge of how OTHER teams played.
- `"POINTS"`: Apply all tiebreakers through point differential. If there are still remaining ties, break them through coin toss. This will go beyond SOS and requires knowledge of points scored and points allowed. As this is not usually part of season simulations, caution is advised in this case. These tiebreakers should only be used if the scores are real or are deliberately simulated.
- `"RANDOM"`: Breaks all tiebreakers with a coin toss. I don't really know, why I allow this...

`playoff_seeds` If `NULL` (the default), will compute all 16 conference ranks. This means, the function applies conference tiebreakers to all conference ranks. For better performance, it is possible to set this to a value < 16 to make the function skip tiebreakers of those conference ranks.

`verbosity` One of `"MIN"`, `"MAX"`, or `"NONE"` allowing the user to set the grade of verbosity of status reports. They mean:

- `"MIN"` (default): Prints main steps of the process.
- `"MAX"`: Prints all steps of the complete tiebreaking process.
- `"NONE"`: No status reports at all. Do this to maximize the performance.

Details

nflseedR does not support all levels of tie-breakers at the moment. The deepest tie-breaker currently is "best net points in all games". After that, the decision is made at random. However, the need for the last level ("best net touchdowns in all games") is extremely unlikely in practice. Deeper levels than strength of schedule have never actually been needed to resolve season-end standings since the NFL expanded to 32 teams.

Value

A data.table of NFL standings including the ranks selected in the argument `ranks`

See Also

For more information on the implemented tiebreakers, see https://nflseedr.com/articles/tiebreaker.html

Examples

```
try({#to avoid CRAN test problems
  games <- nflreadr::load_schedules(2021:2022)
})
standings <- nflseedR::nfl_standings(games)
print(standings, digits = 3)
```

---

nfl_standings_prettify

Compute Pretty NFL Standings Table

Description

Uses the R package gt to create a pretty html table of NFL standings.

Usage

```
nfl_standings_prettify(
  standings,
  ...,
  grp_by = c("div", "conf", "nfl"),
  order_by = c("div_rank", "conf_rank", "draft_rank"),
  reverse = FALSE
)
```

Arguments

standings	A table of NFL standings. Usually computed by nfl_standings()
...	Currently unused. The function errors if objects are passed to the dots, i.e. when unnamed arguments are provided.
grp_by	Group the output table by Division ("div"), Conference ("conf"), or complete league ("nfl")
order_by	Order teams by division rank, conference rank, or draft rank
reverse	Teams are sorted by the argument order_by in ascending order by default. If reverse is set to TRUE, order will be reversed.

Value

An object of class gt_tbl.

Output of below examples

Examples

```
# Calculate standings
s <- nflreadr::load_schedules(2024) |>
  nflseedR::nfl_standings(ranks = "DRAFT")

# Create table
tbl1 <- nfl_standings_prettify(s, grp_by = "conf", order_by = "conf_rank")
tbl2 <- nfl_standings_prettify(s, grp_by = "nfl", order_by = "draft_rank")

# The output of tbl1 and tbl2 is given in the above images.
```

sims_games_example *Example Games Data used in NFL Simulations*

Description

Example Games Data used in NFL Simulations

Usage

```
sims_games_example
```

Format

A data frame with 284 rows and 9 variables containing NFL schedule information.

Details

Please see `data-raw/sim_examples.R` for the code to create this data.

Examples

```
str(sims_games_example)
```

sims_teams_example *Example Teams Data used in NFL Simulations*

Description

Example Teams Data used in NFL Simulations

Usage

```
sims_teams_example
```

Format

A data frame with 64 rows and 5 variables containing team name and division information.

Details

Please see `data-raw/sim_examples.R` for the code to create this data.

Examples

```
str(sims_teams_example)
```

`simulate_nfl` *Simulate an NFL Season*

Description

This function simulates a given NFL season multiple times using custom functions to estimate and simulate game results and computes the outcome of the given season including playoffs and draft order. It is possible to run the function in parallel processes by calling the appropriate plan. Progress updates can be activated by calling handlers before the start of the simulations. Please see the below given section "Details" for further information.

Usage

```
simulate_nfl(
  nfl_season = NULL,
  process_games = NULL,
  ...,
  playoff_seeds = ifelse(nfl_season >= 2020, 7, 6),
  if_ended_today = FALSE,
  fresh_season = FALSE,
  fresh_playoffs = FALSE,
  tiebreaker_depth = 3,
  test_week = NULL,
  simulations = 1000,
  sims_per_round = max(ceiling(simulations/future::availableCores() * 2), 100),
  .debug = FALSE,
  print_summary = FALSE,
  sim_include = c("DRAFT", "REG", "POST")
)
```

Arguments

`nfl_season`	Season to simulate
`process_games`	A function to estimate and simulate the results of games. Uses team, schedule, and week number as arguments.
`...`	Additional parameters passed on to the function `process_games`.
`playoff_seeds`	Number of playoff teams per conference (increased in 2020 from 6 to 7).
`if_ended_today`	Either `TRUE` or `FALSE`. If TRUE, ignore remaining regular season games and cut to playoffs based on current regular season data.
`fresh_season`	Either `TRUE` or `FALSE`. Whether to blank out all game results and simulate the the season from scratch (TRUE) or take game results so far as a given and only simulate the rest (FALSE).
`fresh_playoffs`	Either `TRUE` or `FALSE`. Whether to blank out all playoff game results and simulate the postseason from scratch (TRUE) or take game results so far as a given and only simulate the rest (FALSE).

`tiebreaker_depth`
: A single value equal to 1, 2, or 3. The default is 3. The value controls the depth of tiebreakers that shall be applied. The deepest currently implemented tiebreaker is strength of schedule. The following values are valid:

  tiebreaker_depth = 1 Break all ties with a coinflip. Fastest variant.

  tiebreaker_depth = 2 Apply head-to-head and division win percentage tiebreakers. Random if still tied.

  tiebreaker_depth = 3 Apply all tiebreakers through strength of schedule. Random if still tied.

`test_week`
: Aborts after the simulator reaches this week and returns the results from your process games call.

`simulations`
: Equals the number of times the given NFL season shall be simulated

`sims_per_round`
: The number of `simulations` can be split into multiple rounds and be processed parallel. This parameter controls the number of simulations per round. The default value determines the number of locally available cores and calculates the number of simulations per round to be equal to half of the available cores (various benchmarks showed this results in optimal performance).

`.debug`
: Either `TRUE` or `FALSE`. Controls whether additional messages are printed to the console showing what the tie-breaking algorithms are currently performing.

`print_summary`
: If `TRUE`, prints the summary statistics to the console.

`sim_include`
: One of `"REG"`, `"POST"`, `"DRAFT"` (the default). Simulation will behave as follows:

  REG Simulate the regular season and compute standings, division ranks, and playoff seeds

  POST Do REG + simulate the postseason

  DRAFT Do POST + compute draft order

Details

More Speed Using Parallel Processing:

We recommend choosing a default parallel processing method and saving it as an environment variable in the R user profile to make sure all futures will be resolved with the chosen method by default. This can be done by following the below given steps.

First, run the following line and the user profile should be opened automatically. If you haven't saved any environment variables yet, this will be an empty file.

```
usethis::edit_r_environ()
```

In the opened file add the next line, then save the file and restart your R session. Please note that this example sets "multisession" as default. For most users this should be the appropriate plan but please make sure it truly is.

```
R_FUTURE_PLAN="multisession"
```

After the session is freshly restarted please check if the above method worked by running the next line. If the output is `FALSE` you successfully set up a default non-sequential `future::plan()`. If the output is `TRUE` all functions will behave like they were called with `purrr::map()` and NOT in multisession.

```
inherits(future::plan(), "sequential")
```

For more information on possible plans please see the future package Readme.

Get Progress Updates while Functions are Running:
Most nflfastR functions are able to show progress updates using `progressr::progressor()` if they are turned on before the function is called. There are at least two basic ways to do this by either activating progress updates globally (for the current session) with

```
progressr::handlers(global = TRUE)
```

or by piping the function call into `progressr::with_progress()`:

```
simulate_nfl(2020, fresh_season = TRUE) |>
  progressr::with_progress()
```

For more information how to work with progress handlers please see progressr::progressr.

Value

An `nflseedR_simulation` object containing a list of 6 data frames data frames with the results of all simulated games, the final standings in each simulated season (incl. playoffs and draft order), summary statistics across all simulated seasons, and the simulation parameters. For a full list, please see the package website.

See Also

The examples on the package website

The method `summary.nflseedR_simulation()` that creates a pretty html summary table.

Examples

```
library(nflseedR)

# Activate progress updates
# progressr::handlers(global = TRUE)

# Parallel processing can be activated via the following line
# future::plan("multisession")

try({#to avoid CRAN test problems
# Simulate the season 4 times in 2 rounds
sim <- nflseedR::simulate_nfl(
  nfl_season = 2020,
  fresh_season = TRUE,
  simulations = 4,
  sims_per_round = 2
)

# Overview output
dplyr::glimpse(sim)
})
```

`simulations_verify_fct` *Verify Custom NFL Result Simulation Function*

Description

nflseedR supports custom functions to compute results in season simulations through the argument `compute_results` in the season simulation function nfl_simulations. To ensure that custom functions work as nflseedR expects them to, it is recommended to verify their behavior. This function first checks the structure of the output and then whether game results are changed as expected. Whenever a problem is found, the function will error with a hint to the problem (this means that you might be required to iterate over all problems until the function stops erroring). See below detail section for more information on expected behavior.

Usage

```
simulations_verify_fct(
  compute_results,
  ...,
  games = nflseedR::sims_games_example,
  teams = nflseedR::sims_teams_example
)
```

Arguments

`compute_results`	A function to compute results of games. See below detail section for more information on expected behavior.
`...`	Further arguments passed on to `compute_results`.
`games`	An NFL schedule where some results are missing. `compute_results` is supposed to compute those results on a weekly base. Defaults to sims_games_example. Please see this example to understand the required data structure.
`teams`	A list of teams by simulation number. This is usually calculated automatically and not user facing. It can be used to "transport" team information like elo ratings from one simulated week to the next. Defaults to sims_teams_example. Please see this example to understand the required data structure.

Details

The following sections detail the requirements for the `compute_results` function. If anything is unclear, please see the source code of nflseedR's default function `nflseedR_compute_results`.

Required Function Arguments of `compute_results`:

The function passed to `compute_results` is required to support the arguments `"teams"`, `"games"`, and `"week_num"`. The two leading ones are already described above. The latter is a factor with a length of 1, which identifies the current week. Regular season weeks are labeled `"1"`, `"2"`, etc. Playoff weeks are labeled `"WC"`, `"DIV"`, `"CON"`, and `"SB"`.

Required Output Structure of `compute_results`**:**

The function passed to `compute_results` is required to return a list of the two objects `"teams"` and `"games"` as passed to it in the arguments of the same name. The function must not remove rows or columns. So the last line of `compute_results` usually looks like

```
list("teams" = teams, "games" = games)
```

Required Behavior of `compute_results` **when Computing Game Results:**

nflseedR calls `compute_results` for every week where a `result` is missing in `games`. The variable `result` is defined as the point differential between the home team and the away team. If the home team loses, the value is therefore < 0, if it wins > 0 and if it ties == 0. To support elo-based simulations, this is done in a loop so that elo ratings can be updated based on the results and "transported" from week to week. You can "transport" ratings or other information by joining them to the `"teams"` table. This behavior requires that `compute_results` only changes the results of the current week - called `week_num`. And only if there is not already a result. So `compute_results` must only compute a result when

```
week == week_num & is.na(result)
```

For the playoffs, there is also the special case that matches cannot end in a tie (`result == 0`). In most cases, ties are not simulated anyway because they occur so rarely. But in the event that they are simulated, they must not be in the playoffs.

Value

Returns `TRUE` invisibly if no problems are found.

Examples

```
simulations_verify_fct(nflseedR_compute_results)
```

`summary.nflseedR_simulation`

Compute Pretty Simulations Summary Table

Description

Uses the R package gt to create a pretty html table of the nflseedR simulation summary data frame.

Usage

```
## S3 method for class 'nflseedR_simulation'
summary(object, ...)
```

Arguments

`object`	an object for which a summary is desired.
`...`	additional arguments passed on to the methods (currently not used).

Output of below example

Examples

```
library(nflseedR)
# set seed for recreation,
# internal parallelization requires a L'Ecuyer-CMRG random number generator
set.seed(19980310, kind = "L'Ecuyer-CMRG")

# Simulate the season 20 times in 1 round
sim <- nflseedR::simulate_nfl(
  nfl_season = 2021,
  fresh_season = TRUE,
  simulations = 20
)

# Create Summary Tables
tbl <- summary(sim)

# The output of tbl is given in the above image.
```

Index