Package: nflseedR (via r-universe)

July 19, 2024

Title Functions to Efficiently Simulate and Evaluate NFL Seasons

Version 1.2.0.9001

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 (>= 3.5.0)

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

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

Config/testthat/edition 3

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Repository https://nflverse.r-universe.dev

RemoteUrl https://github.com/nflverse/nflseedR

RemoteRef HEAD

RemoteSha e7458371f184562ecda01f4797c560d7dda72092

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
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(h2h = .$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)

games %>%
  nflseedR::compute_division_ranks() %>%
  nflseedR::compute_conference_seeds(h2h = .$h2h, playoff_seeds = 6) %>%
  nflseedR::compute_draft_order(games = games, h2h = .$h2h)
})

# Restore old options
options(old)
```

`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

```
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)
```

`load_schedules` *Load Lee Sharpe's Games File*

Description

Lee Sharpe maintains an important data set that contains broadly used information on games in the National Football League. This function is a convenient helper to download the file into memory without having to remember the correct url.

Usage

```
load_schedules(...)

load_sharpe_games(...)
```

Arguments

`...`	Arguments passed on to `nflreadr::load_schedules`
	`seasons` a numeric vector of seasons to return, default `TRUE` returns all available data.

Value

A data frame containing the following variables for all NFL games since 1999:

game_id The ID of the game as assigned by the nflverse. Note that this value matches the `game_id` field in nflfastR if you wish to join the data.

season The year of the NFL season. This represents the whole season, so regular season games that happen in January as well as playoff games will occur in the year after this number.

game_type What type of game? One of the following values:

- `REG`: a regular season game
- `WC`: a wildcard playoff game
- `DIV`: a divisional round playoff game
- `CON`: a conference championship
- `SB`: a Super Bowl

week The week of the NFL season the game occurs in. Please note that the `game_type` will differ for weeks >= 18 because of the season expansion in 2021. Please use `game_type` to filter for regular season or postseason.

gameday The date on which the game occurred.

weekday The day of the week on which the game occurred.

gametime The kickoff time of the game. This is represented in 24-hour time and the Eastern time zone, regardless of what time zone the game was being played in.

away_team The away team.

away_score The number of points the away team scored. Is `NA` for games which haven't yet been played.

home_team The home team. Note that this contains the designated home team for games which no team is playing at home such as Super Bowls or NFL International games.

home_score The number of points the home team scored. Is `NA` for games which haven't yet been played.

location Either `Home` if the home team is playing in their home stadium, or `Neutral` if the game is being played at a neutral location. This still shows as `Home` for games between the Giants and Jets even though they share the same home stadium.

result Equals `home_score - away_score`. The number of points the home team scored minus the number of points the away team scored. Is `NA` for games which haven't yet been played. Convenient for evaluating against the spread bets.

total The sum of each team's score in the game. Equals `home_score + away_score`. Is `NA` for games which haven't yet been played. Convenient for evaluating over/under total bets.

overtime Whether the game went into overtime (= 1) or not (= 0).

old_game_id The id of the game issued by the NFL Game Statistics & Information System.

away_rest The number of days since that away team's previous game (7 is used for the team's first game of the season).

home_rest The number of days since that home team's previous game (7 is used for the team's first game of the season).

away_moneyline Odd of the away_team winning the game.

home_moneyline Odd of the home_team winning the game.

spread_line The spread line for the game. A positive number means the home team was favored by that many points, a negative number means the away team was favored by that many points. This lines up with the `result` column.

away_spread_odds Odd of the away_team covering the `spread_line`.

home_spread_odds Odd of the home_team covering the `spread_line`.

total_line The total line for the game.

under_odds Odd of the `total` being under the `total_line`.

over_odds Odd of the `total` being over the `total_line`.

div_game Whether the game was a divisional game (= 1) or not (= 0).

roof What was the status of the stadium's roof? Will be one of the following values:

- `closed`: Stadium has a retractable roof which was closed
- `dome`: An indoor stadium
- `open`: Stadium has a retractable roof which was open
- `outdoors`: An outdoor stadium

surface What type of ground the game was played on.

temp The temperature at the stadium (for `roof` types `outdoors` and `open` only).

wind The speed of the wind in miles/hour (for `roof` types `outdoors` and `open` only).

away_qb_id GSIS ID of the "starting quarterback" of the away team identified as the first quarterback (per roster data) listed as `passer` (in `nflfastR` play by play data) in 2+ plays that game. In the final regular season game it is the QB with the most plays as the `passer`.

home_qb_id GSIS ID of the "starting quarterback" of the home team identified as the first quarterback (per roster data) listed as `passer` (in `nflfastR` play by play data) in 2+ plays that game. In the final regular season game it is the QB with the most plays as the `passer`.

away_qb_name Full name of the "starting quarterback" of the away team identified as the first quarterback (per roster data) listed as `passer` (in `nflfastR` play by play data) in 2+ plays that game. In the final regular season game it is the QB with the most plays as the `passer`.

home_qb_name Full name of the "starting quarterback" of the home team identified as the first quarterback (per roster data) listed as `passer` (in `nflfastR` play by play data) in 2+ plays that game. In the final regular season game it is the QB with the most plays as the `passer`.

away_coach Name of the head coach of the away team.

home_coach Name of the head coach of the home team.

referee Name of the game's referee (head official).

stadium_id Pro Football Reference ID of the stadium.

stadium Name of the stadium.

See Also

The internally called function `nflreadr::load_schedules()`

Examples

```
try({#to avoid CRAN test problems
games <- load_sharpe_games()
dplyr::glimpse(games)
})
```

`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)
})
```

`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