Package 'nflreadr'

October 5, 2021

Title Download 'nflverse' Data

Version 1.1.1

Description A low-level package for downloading data from 'GitHub' repositories of the 'nflverse' project.

License MIT + file LICENSE

URL https://nflreadr.nflverse.com, https://github.com/nflverse/nflreadr

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

Depends R (>= 3.5.0)

Imports cachem (>= 1.0.0), data.table (>= 1.14.0), curl (>= 4.3.0), memoise (>= 2.0.0), qs (>= 0.24.0), rappdirs (>= 0.3.0), rlang (>= 0.4.0), Rcpp (>= 1.0.7), RcppParallel (>= 5.1.4)

Suggests covr (>= 3.0.0), DT (>= 0.15.0), knitr (>= 1.0.0), progressr (>= 0.8.0), rmarkdown (>= 2.6.0), testthat (>= 3.0.0)

Config/testthat/edition 3

Encoding UTF-8

LazyData true

RoxygenNote 7.1.2

VignetteBuilder knitr

NeedsCompilation no

Author Tan Ho [aut, cre, cph] (<https://orcid.org/0000-0001-8388-5155>), Sebastian Carl [aut], John Edwards [ctb], Ben Baldwin [ctb], Thomas Mock [ctb], Lee Sharpe [ctb]

Maintainer Tan Ho <tan@tanho.ca>

Repository CRAN

Date/Publication 2021-10-05 18:00:02 UTC

R topics documented:

.clear_cache *Clear function cache*

Description

This function clears the memoised cache of all functions memoised by `nflreadr`.

Usage

```
.clear_cache()
```

Value

A success message after clearing the cache.

Examples

```
.clear_cache()
```

clean_homeaway *Clean Home/Away in dataframes into Team/Opponent dataframes*

Description

This function converts dataframes with "home_" and "away_" prefixed columns to "team_" and "opponent_", and doubles the rows. This makes sure that there's one row for each team (as opposed to one row for each game).

Usage

```
clean_homeaway(dataframe, invert = NULL)
```

Arguments

`dataframe`	dataframe
`invert`	a character vector of columns that gets inverted when referring to the away team (e.g. home spread = 1 gets converted to away_spread = -1)

Value

a dataframe with one row per team (twice as long as the input dataframe)

Examples

```
# creating a small example dataframe!
cols <- c("season", "week", "home_team", "home_score",
          "away_team", "away_score", "result", "spread_line")

x <- as.data.frame(load_schedules(2020))
x <- utils::head(x[cols])

# how the data looks like
x

clean_homeaway(x, invert = c("result","spread_line"))
```

`clean_player_names` *Create Player Merge Names*

Description

Applies some name-cleaning heuristics to facilitate joins. These heuristics may include:

- removing periods and apostrophes
- removing common suffixes, such as Jr, Sr, II, III, IV
- converting to lowercase
- using `ffscrapr::dp_name_mapping` to do common name substitutions, such as Mitch Trubisky to Mitchell Trubisky

Usage

```
clean_player_names(
  player_name,
  lowercase = FALSE,
  convert_lastfirst = TRUE,
  use_name_database = TRUE
)
```

Arguments

`player_name`	a character vector of player names
`lowercase`	defaults to FALSE - if TRUE, converts to lowercase
`convert_lastfirst`	defaults to TRUE - converts names from "Last, First" to "First Last"
`use_name_database`	uses internal name database to do common substitutions (Mitchell Trubisky to Mitch Trubisky etc)

Details

Equivalent to the operation done by `ffscrapr::dp_clean_names()` and uses the same player name database.

Value

a character vector of cleaned names

Examples

```
clean_player_names(c("A.J. Green", "Odell Beckham Jr.", "Le'Veon Bell Sr."))

clean_player_names(c("Trubisky,    Mitch", "Atwell, Chatarius", "Elliott, Zeke", "Elijah Moore"),
                   convert_lastfirst = TRUE)
```

---

clean_team_abbrs *Standardize NFL Team Abbreviations*

Description

This function standardizes NFL team abbreviations to nflverse defaults. This helps for joins and plotting, especially with the new nflplotR package!

Usage

```
clean_team_abbrs(abbr, current_location = TRUE, keep_non_matches = TRUE)
```

Arguments

`abbr`	a character vector of abbreviations
`current_location`	If `TRUE` (the default), the abbreviation of the most recent team location will be used.
`keep_non_matches`	If `TRUE` (the default) an element of `abbr` that can't be matched to any of the internal mapping vectors will be kept as is. Otherwise it will be replaced with `NA`.

Value

A character vector with the length of `abbr` and cleaned team abbreviations if they are included in `team_abbr_mapping` or `team_abbr_mapping_norelocate` (depending on the value of `current_location`). Non matches may be replaced with `NA` (depending on the value of `keep_non_matches`).

Examples

```
x <- c("PIE", "LAR", "PIT", "CRD", "OAK", "SL")
# use current location and keep non matches
clean_team_abbrs(x)

# keep old location and replace non matches
clean_team_abbrs(x, current_location = FALSE, keep_non_matches = FALSE)
```

`csv_from_url` *Load .csv / .csv.gz file from a remote connection*

Description

This is a thin wrapper on data.table::fread, but memoised & cached for twenty four hours.

Usage

```
csv_from_url(...)
```

Arguments

`...` Arguments passed on to `data.table::fread`

`input` A single character string. The value is inspected and deferred to either `file=` (if no \n present), `text=` (if at least one \n is present) or `cmd=` (if no \n is present, at least one space is present, and it isn't a file name). Exactly one of `input=`, `file=`, `text=`, or `cmd=` should be used in the same call.

`file` File name in working directory, path to file (passed through `path.expand` for convenience), or a URL starting http://, file://, etc. Compressed files with extension '`.gz`' and '`.bz2`' are supported if the `R.utils` package is installed.

`text` The input data itself as a character vector of one or more lines, for example as returned by `readLines()`.

`cmd` A shell command that pre-processes the file; e.g. `fread(cmd=paste("grep",word,"filename"))`. See Details.

`sep` The separator between columns. Defaults to the character in the set `[,\t |;:]` that separates the sample of rows into the most number of lines with the same number of fields. Use `NULL` or `""` to specify no separator; i.e. each line a single character column like `base::readLines` does.

`sep2` The separator *within* columns. A `list` column will be returned where each cell is a vector of values. This is much faster using less working memory than `strsplit` afterwards or similar techniques. For each column `sep2` can be different and is the first character in the same set above `[,\t |;]`, other than `sep`, that exists inside each field outside quoted regions in the sample. NB: `sep2` is not yet implemented.

`nrows` The maximum number of rows to read. Unlike `read.table`, you do not need to set this to an estimate of the number of rows in the file for better speed because that is already automatically determined by `fread` almost instantly using the large sample of lines. `nrows=0` returns the column names and typed empty columns determined by the large sample; useful for a dry run of a large file or to quickly check format consistency of a set of files before starting to read any of them.

`header` Does the first data line contain column names? Defaults according to whether every non-empty field on the first data line is type character. If so, or TRUE is supplied, any empty column names are given a default name.

`na.strings` A character vector of strings which are to be interpreted as `NA` values. By default, `",,"` for columns of all types, including type `character` is read as `NA` for consistency. `,"",` is unambiguous and read as an empty string. To read `,NA,` as `NA`, set `na.strings="NA"`. To read `,,` as blank string `""`, set `na.strings=NULL`. When they occur in the file, the strings in `na.strings` should not appear quoted since that is how the string literal `,"NA",` is distinguished from `,NA,`, for example, when `na.strings="NA"`.

`stringsAsFactors` Convert all character columns to factors?

`verbose` Be chatty and report timings?

`skip` If 0 (default) start on the first line and from there finds the first row with a consistent number of columns. This automatically avoids irregular header information before the column names row. `skip>0` means ignore the first `skip` rows manually. `skip="string"` searches for `"string"` in the file (e.g. a substring of the column names row) and starts on that line (inspired by read.xls in package gdata).

`select` A vector of column names or numbers to keep, drop the rest. `select` may specify types too in the same way as `colClasses`; i.e., a vector of `colname=type` pairs, or a `list` of `type=col(s)` pairs. In all forms of `select`, the order that the columns are specified determines the order of the columns in the result.

`drop` Vector of column names or numbers to drop, keep the rest.

`colClasses` As in `utils::read.csv`; i.e., an unnamed vector of types corresponding to the columns in the file, or a named vector specifying types for a subset of the columns by name. The default, `NULL` means types are inferred from the data in the file. Further, `data.table` supports a named `list` of vectors of column names *or numbers* where the `list` names are the class names; see examples. The `list` form makes it easier to set a batch of columns to be a particular class. When column numbers are used in the `list` form, they refer to the column number in the file not the column number after `select` or `drop` has been applied. If type coercion results in an error, introduces `NA`s, or would result in loss of accuracy, the coercion attempt is aborted for that column with warning and the column's type is left unchanged. If you really desire data loss (e.g. reading `3.14` as `integer`) you have to truncate such columns afterwards yourself explicitly so that this is clear to future readers of your code.

`integer64` "integer64" (default) reads columns detected as containing integers larger than 2^31 as type `bit64::integer64`. Alternatively, `"double"|"numeric"`

reads as utils::read.csv does; i.e., possibly with loss of precision and if so silently. Or, "character".

dec The decimal separator as in utils::read.csv. If not "." (default) then usually ",". See details.

col.names A vector of optional names for the variables (columns). The default is to use the header column if present or detected, or if not "V" followed by the column number. This is applied after check.names and before key and index.

check.names default is FALSE. If TRUE then the names of the variables in the data.table are checked to ensure that they are syntactically valid variable names. If necessary they are adjusted (by make.names) so that they are, and also to ensure that there are no duplicates.

encoding default is "unknown". Other possible options are "UTF-8" and "Latin-1". Note: it is not used to re-encode the input, rather enables handling of encoded strings in their native encoding.

quote By default ("\""), if a field starts with a double quote, fread handles embedded quotes robustly as explained under Details. If it fails, then another attempt is made to read the field *as is*, i.e., as if quotes are disabled. By setting quote="", the field is always read as if quotes are disabled. It is not expected to ever need to pass anything other than \"\" to quote; i.e., to turn it off.

strip.white default is TRUE. Strips leading and trailing whitespaces of unquoted fields. If FALSE, only header trailing spaces are removed.

fill logical (default is FALSE). If TRUE then in case the rows have unequal length, blank fields are implicitly filled.

blank.lines.skip logical, default is FALSE. If TRUE blank lines in the input are ignored.

key Character vector of one or more column names which is passed to setkey. It may be a single comma separated string such as key="x,y,z", or a vector of names such as key=c("x","y","z"). Only valid when argument data.table=TRUE. Where applicable, this should refer to column names given in col.names.

index Character vector or list of character vectors of one or more column names which is passed to setindexv. As with key, comma-separated notation like index="x,y,z" is accepted for convenience. Only valid when argument data.table=TRUE. Where applicable, this should refer to column names given in col.names.

showProgress TRUE displays progress on the console if the ETA is greater than 3 seconds. It is produced in fread's C code where the very nice (but R level) txtProgressBar and tkProgressBar are not easily available.

data.table TRUE returns a data.table. FALSE returns a data.frame. The default for this argument can be changed with options(datatable.fread.datatable=FALSE).

nThread The number of threads to use. Experiment to see what works best for your data on your hardware.

logical01 If TRUE a column containing only 0s and 1s will be read as logical, otherwise as integer.

keepLeadingZeros If TRUE a column containing numeric data with leading zeros will be read as character, otherwise leading zeros will be removed and converted to numeric.

yaml If TRUE, fread will attempt to parse (using yaml.load) the top of the input as YAML, and further to glean parameters relevant to improving the performance of fread on the data itself. The entire YAML section is returned as parsed into a list in the yaml_metadata attribute. See Details.

autostart Deprecated and ignored with warning. Please use skip instead.

tmpdir Directory to use as the tmpdir argument for any tempfile calls, e.g. when the input is a URL or a shell command. The default is tempdir() which can be controlled by setting TMPDIR before starting the R session; see base::tempdir.

tz Relevant to datetime values which have no Z or UTC-offset at the end, i.e. *unmarked* datetime, as written by utils::write.csv. The default tz="UTC" reads unmarked datetime as UTC POSIXct efficiently. tz="" reads unmarked datetime as type character (slowly) so that as.POSIXct can interpret (slowly) the character datetimes in local timezone; e.g. by using "POSIXct" in colClasses=. Note that fwrite() by default writes datetime in UTC including the final Z and therefore fwrite's output will be read by fread consistently and quickly without needing to use tz= or colClasses=. If the TZ environment variable is set to "UTC" (or "" on non-Windows where unset vs '""' is significant) then the R session's timezone is already UTC and tz="" will result in unmarked datetimes being read as UTC POSIXct. For more information, please see the news items from v1.13.0 and v1.14.0.

Value

a dataframe as created by data.table::fread()

Examples

```
csv_from_url("https://github.com/nflverse/nfldata/raw/master/data/games.csv")
```

---

dictionary_draft_picks

Data Dictionary: Draft Picks

Description

A dataframe containing the data dictionary for load_draft_picks()

Usage

```
dictionary_draft_picks
```

Format

An object of class data.frame with 10 rows and 3 columns.

See Also

vignette("Data Dictionary -Draft Picks")

https://nflreadr.nflverse.com/articles/dictionary_draft_picks.html

---

dictionary_ff_playerids

Data Dictionary: Fantasy Player IDs

Description

A dataframe containing the data dictionary for load_ff_playerids()

Usage

```
dictionary_ff_playerids
```

Format

An object of class data.frame with 34 rows and 3 columns.

See Also

vignette("Data Dictionary -FF Player IDs")

https://nflreadr.nflverse.com/articles/dictionary_ff_playerids.html

---

dictionary_ff_rankings

Data Dictionary: Fantasy Football Rankings

Description

A dataframe containing the data dictionary for load_ff_rankings()

Usage

```
dictionary_ff_rankings
```

Format

An object of class data.frame with 25 rows and 3 columns.

See Also

vignette("Data Dictionary -FF Rankings")

https://nflreadr.nflverse.com/articles/dictionary_ff_rankings.html

---

dictionary_nextgen_stats

Data Dictionary: Next Gen Stats

Description

A dataframe containing the data dictionary for load_nextgen_stats()

Usage

```
dictionary_nextgen_stats
```

Format

An object of class data.frame with 61 rows and 3 columns.

See Also

vignette("Data Dictionary -Next Gen Stats")

https://nflreadr.nflverse.com/articles/dictionary_nextgen_stats.html

---

dictionary_pbp

Data Dictionary: Play by Play

Description

A dataframe containing the data dictionary for load_pbp()

Usage

```
dictionary_pbp
```

Format

An object of class data.frame with 372 rows and 2 columns.

See Also

vignette("Data Dictionary -PBP")

https://nflreadr.nflverse.com/articles/dictionary_pbp.html

`dictionary_pfr_passing`

Data Dictionary: PFR Passing

Description

A dataframe containing the data dictionary for `load_pfr_passing()`

Usage

```
dictionary_pfr_passing
```

Format

An object of class `data.frame` with 28 rows and 3 columns.

See Also

https://nflreadr.nflverse.com/articles/dictionary_pfr_passing.html

```
vignette("Data Dictionary -PFR Passing")
```

`dictionary_player_stats`

Data Dictionary: Player Stats

Description

A dataframe containing the data dictionary for `load_player_stats()`

Usage

```
dictionary_player_stats
```

Format

An object of class `data.frame` with 41 rows and 2 columns.

See Also

```
vignette("Data Dictionary -Player Stats")
```

https://nflreadr.nflverse.com/articles/dictionary_player_stats.html

dictionary_rosters *Data Dictionary: Rosters*

Description

A dataframe containing the data dictionary for `load_rosters()`

Usage

```
dictionary_rosters
```

Format

An object of class `data.frame` with 24 rows and 3 columns.

See Also

```
vignette("Data Dictionary -Rosters")
```

https://nflreadr.nflverse.com/articles/dictionary_rosters.html

dictionary_schedules *Data Dictionary: Schedules*

Description

A dataframe containing the data dictionary for `load_schedules()`

Usage

```
dictionary_schedules
```

Format

An object of class `data.frame` with 27 rows and 2 columns.

See Also

```
vignette("Data Dictionary -Schedules")
```

https://nflreadr.nflverse.com/articles/dictionary_schedules.html

dictionary_snap_counts *Data Dictionary: Snap Counts*

Description

A dataframe containing the data dictionary for load_snap_counts()

Usage

```
dictionary_snap_counts
```

Format

An object of class data.frame with 12 rows and 3 columns.

See Also

```
vignette("Data Dictionary -Snap Counts")
```

https://nflreadr.nflverse.com/articles/dictionary_snap_counts.html

load_combine *Load Combine Data from PFR*

Description

Loads combine data since 2000 courtesy of PFR.

Usage

```
load_combine(seasons = TRUE)
```

Arguments

seasons	a numeric vector of seasons to return, default TRUE returns all available data

Value

A tibble of NFL combine data provided by Pro Football Reference.

Examples

```
load_combine()
```

load_depth_charts *Load Weekly Depth Charts*

Description

Loads depth charts for each NFL team for each week back to 2001.

Usage

```
load_depth_charts(seasons = most_recent_season())
```

Arguments

seasons	a numeric vector specifying what seasons to return, if TRUE returns all available data. Defaults to latest season.

Value

A tibble of week-level depth charts for each team.

See Also

https://github.com/nflverse/nflfastR-roster

Examples

```
load_depth_charts(2020)
```

load_draft_picks *Load Draft Picks from PFR*

Description

Loads every draft pick since 1980 courtesy of PFR.

Usage

```
load_draft_picks(seasons = TRUE)
```

Arguments

seasons	a numeric vector of seasons to return, default TRUE returns all available data

Value

A tibble of NFL draft picks provided by Pro Football Reference.

See Also

dictionary_draft_picks for the data dictionary

https://nflreadr.nflverse.com/articles/dictionary_draft_picks.html

Examples

```
load_draft_picks()
```

---

load_espn_qbr *Load ESPN's QBR*

Description

Load ESPN's QBR

Usage

```
load_espn_qbr(
  league = c("nfl", "college"),
  seasons = most_recent_season(),
  summary_type = c("season", "weekly")
)
```

Arguments

league	One of "nfl" or "college", defaults to "nfl"
seasons	a numeric vector of seasons to return, data available since 2006. Defaults to latest season available. TRUE will select all seasons.
summary_type	One of "season" or "weekly", defaults to season

Value

a tibble of season-level injury report data.

See Also

https://github.com/nflverse/espnscrapeR-data

Examples

```
load_espn_qbr("nfl",2020)
```

`load_ff_playerids` *Load Fantasy Player IDs*

Description

Accesses DynastyProcess.com's database of fantasy football player IDs, which help connect nflverse to various other platforms and IDs.

Usage

```
load_ff_playerids()
```

Value

a dataframe of player IDs

See Also

https://github.com/dynastyprocess/data

Examples

```
load_ff_playerids()
```

`load_ff_rankings` *Load Latest FantasyPros Rankings*

Description

Accesses DynastyProcess.com's repository of the latest FP expert consensus rankings - updated on a weekly basis.

Usage

```
load_ff_rankings(type = c("draft", "week"))
```

Arguments

type	one of "draft" or "week", for preseason or inseason-weekly rankings

Value

a dataframe of expert consensus rankings

See Also

https://github.com/dynastyprocess/data

https://www.fantasypros.com

Examples

```
load_ff_rankings()
```

---

load_injuries *Load Injury Reports*

Description

Load Injury Reports

Usage

```
load_injuries(
  seasons = most_recent_season(),
  file_type = getOption("nflreadr.prefer", default = "qs")
)
```

Arguments

seasons	a numeric vector of seasons to return, data available since 2009. Defaults to latest season available.
file_type	One of "rds" or "qs". Can also be set globally with options(nflreadr.prefer)

Value

a tibble of season-level injury report data.

See Also

https://github.com/nflverse/nflfastR-roster

Examples

```
load_injuries(2020)
```

`load_nextgen_stats` *Load Player Level Weekly NFL Next Gen Stats*

Description

Loads player level weekly stats provided by NFL Next Gen Stats starting with the 2016 season. Three different stat types are available and the current season's data updates every night.

Usage

```
load_nextgen_stats(
  seasons = TRUE,
  stat_type = c("passing", "receiving", "rushing"),
  file_type = getOption("nflreadr.prefer", default = "qs")
)
```

Arguments

`seasons`	a numeric vector specifying what seasons to return, if `TRUE` returns all available data
`stat_type`	one of `"passing"`, `"receiving"`, or `"rushing"`
`file_type`	One of `"rds"` or `"qs"`. Can also be set globally with options(nflreadr.prefer)

Value

A tibble of week-level player statistics provided by NFL Next Gen Stats. Regular season summary is given for `week == 0`.

See Also

https://nextgenstats.nfl.com/stats/passing for `stat_type = "passing"`

https://nextgenstats.nfl.com/stats/receiving for `stat_type = "receiving"`

https://nextgenstats.nfl.com/stats/rushing for `stat_type = "rushing"`

`dictionary_nextgen_stats` for the data dictionary

https://nflreadr.nflverse.com/articles/dictionary_nextgen_stats.html for a web version of the dictionary

Examples

```
load_nextgen_stats(stat_type = "passing")
load_nextgen_stats(stat_type = "receiving")
load_nextgen_stats(stat_type = "rushing")
```

---

load_pbp *Load Play By Play*

Description

Loads multiple seasons from the nflfastR data repository

Usage

```
load_pbp(
  seasons = most_recent_season(),
  file_type = getOption("nflreadr.prefer", default = "qs")
)
```

Arguments

seasons	A numeric vector of 4-digit years associated with given NFL seasons - defaults to latest season. If set to TRUE, returns all available data since 1999.
file_type	One of "rds" or "qs". Can also be set globally with options(nflreadr.prefer)

Value

The complete nflfastR dataset as returned by nflfastR::build_nflfastR_pbp() (see below) for all given seasons

See Also

dictionary_pbp for the data dictionary as a dataframe

https://nflreadr.nflverse.com/articles/dictionary_pbp.html for a web version of the data dictionary

https://www.nflfastr.com/reference/build_nflfastR_pbp.html for the nflfastR function nflfastR::build_nflfastR_pbp()

Examples

```
load_pbp(2019:2020)
```

load_pfr_advstats *Load Advanced Stats from PFR*

Description

Loads player level season stats provided by Pro Football Reference starting with the 2018 season, primarily to augment existing nflverse data.

Usage

```
load_pfr_advstats(
  seasons = most_recent_season(),
  stat_type = c("pass", "rush", "rec", "def")
)
```

Arguments

seasons	a numeric vector specifying what seasons to return, if TRUE returns all available data
stat_type	one of "pass", "rush", "rec", "def"

Value

A tibble of week-level player statistics provided by Pro Football Reference to supplement data in nflverse

See Also

https://www.pro-football-reference.com/years/2021/passing_advanced.htm

Examples

```
load_pfr_advstats()
```

load_pfr_passing *Load Advanced Passing Stats from PFR*

Description

Loads player level season stats provided by Pro Football Reference starting with the 2019 season.

Usage

```
load_pfr_passing(seasons = TRUE)
```

Arguments

seasons	a numeric vector specifying what seasons to return, if TRUE returns all available data

Value

A tibble of season-level player statistics provided by Pro Football Reference.

See Also

https://www.pro-football-reference.com/years/2020/passing_advanced.htm

dictionary_pfr_passing for the data dictionary

https://nflreadr.nflverse.com/articles/dictionary_pfr_passing.html

Examples

```
load_pfr_passing()
```

---

load_player_stats *Load Player Level Weekly Stats*

Description

Load Player Level Weekly Stats

Usage

```
load_player_stats(
  seasons = most_recent_season(),
  stat_type = c("offense", "kicking"),
  file_type = getOption("nflreadr.prefer", default = "qs")
)
```

Arguments

seasons	a numeric vector of seasons to return, defaults to most recent season. If set to TRUE, returns all available data.
stat_type	one of offense or kicking
file_type	One of "rds" or "qs". Can also be set globally with options(nflreadr.prefer)

Value

A tibble of week-level player statistics that aims to match NFL official box scores.

See Also

https://www.nflfastr.com/reference/load_player_stats.html

https://github.com/nflverse/nflfastr-data

vignette("Data Dictionary -Player Stats") for the data dictionary

Examples

```
load_player_stats()
load_player_stats(stat_type = "kicking")
```

---

load_rosters *Load Rosters*

Description

Load Rosters

Usage

```
load_rosters(seasons = most_recent_season(roster = TRUE))
```

Arguments

seasons	a numeric vector of seasons to return, defaults to returning this year's data if it is March or later. If set to TRUE, will return all available data.

Value

A tibble of season-level roster data.

See Also

dictionary_rosters for the data dictionary as a dataframe

https://github.com/nflverse/nflfastR-roster

https://www.nflfastr.com/reference/fast_scraper_roster.html

Examples

```
load_rosters(2020)
```

load_schedules *Load Game/Schedule Data*

Description

This returns game/schedule information as maintained by Lee Sharpe.

Usage

```
load_schedules(seasons = TRUE)
```

Arguments

seasons	a numeric vector of seasons to return, default TRUE returns all available data.

Value

A tibble of game information for past and/or future games.

See Also

https://github.com/nflverse/nfldata/blob/master/DATASETS.md#games

dictionary_schedules for the data dictionary as a dataframe

vignette("Data Dictionary -Schedules") for the data dictionary as a vignette

Examples

```
load_schedules(2020)
```

load_snap_counts *Load Snap Counts from PFR*

Description

Loads game level snap counts stats provided by Pro Football Reference starting with the 2013 season.

Usage

```
load_snap_counts(seasons = most_recent_season())
```

Arguments

seasons	a numeric vector specifying what seasons to return, if TRUE returns all available data

Value

A tibble of season-level player statistics provided by Pro Football Reference.

See Also

dictionary_snap_counts for the data dictionary

https://nflreadr.nflverse.com/articles/dictionary_snap_counts.html

Examples

```
load_snap_counts()
```

---

load_teams *Load NFL Team Graphics, Colors, and Logos*

Description

Loads team graphics, colors, and logos - useful for plots!

Usage

```
load_teams()
```

Value

A tibble of team-level image URLs and hex color codes.

See Also

https://github.com/nflverse/nflfastr-data

Examples

```
load_teams()
```

load_trades *Load Trades*

Description

This returns a table of historical trades as maintained by Lee Sharpe.

Usage

```
load_trades(seasons = TRUE)
```

Arguments

seasons	a numeric vector of seasons to return, default TRUE returns all available data.

Value

A tibble of game information for past and/or future games.

See Also

https://github.com/nflverse/nfldata/blob/master/DATASETS.md#trades

Examples

```
load_trades(2020)
```

player_name_mapping *Alternate player name mappings*

Description

A named character vector mapping common alternate names, re-exported from ffscrapr.

Usage

```
player_name_mapping
```

Format

A named character vector

name attribute The "alternate" name.

value attribute The "correct" name.

Details

You can suggest additions to this table by opening an issue in ffscrapr.

Examples

```
player_name_mapping[c("Chatarius Atwell", "Robert Kelley")]
```

`progressively` *Progressively*

Description

This function helps add progress-reporting to any function - given function `f()` and progressor `p()`, it will return a new function that calls `f()` and then (on-exiting) will call `p()` after every iteration.

Usage

```
progressively(f, p = NULL)
```

Arguments

`f`	a function to add progressr functionality to.
`p`	a progressor function as created by `progressr::progressor()`

Details

This is inspired by purrr's `safely`, `quietly`, and `possibly` function decorators.

Value

a function that does the same as `f` but it calls `p()` after iteration.

See Also

`https://nflreadr.nflverse.com/articles/exporting_nflreadr.html` for vignette on exporting nflreadr in packages

Examples

```
read_rosters <- function(){
 urls <- c("https://github.com/nflverse/nflfastR-roster/raw/master/data/seasons/roster_2020.csv",
        "https://github.com/nflverse/nflfastR-roster/raw/master/data/seasons/roster_2021.csv")

  p <- progressr::progressor(along = urls)
  lapply(urls, progressively(read.csv, p))
}

progressr::with_progress(read_rosters())
```

`qs_from_url` *Load .qs file from a remote connection*

Description

Load .qs file from a remote connection

Usage

```
qs_from_url(url)
```

Arguments

`url`	a character url

Value

a dataframe as parsed by `qs::qdeserialize()`

Examples

```
qs_from_url(
"https://github.com/nflverse/nflfastR-data/raw/master/data/play_by_play_2020.qs"
)
```

raw_from_url *Load raw filedata from a remote connection*

Description

This function allows you to retrieve data from a URL into raw format, which can then be passed into the appropriate file-reading function, such as `arrow::read_parquet()`

Usage

```
raw_from_url(url)
```

Arguments

`url`	a character url

Value

a raw vector

Examples

```
head(raw_from_url(
  "https://github.com/nflverse/nflfastR-data/raw/master/data/play_by_play_2020.parquet"
  ),
50)
```

rds_from_url *Load .rds file from a remote connection*

Description

Load .rds file from a remote connection

Usage

```
rds_from_url(url)
```

Arguments

`url`	a character url

Value

a dataframe as created by `readRDS()`

Examples

```
rds_from_url("https://github.com/nflverse/nfldata/raw/master/data/games.rds")
```

`team_abbr_mapping` *Alternate team abbreviation mappings*

Description

A named character vector mapping common alternate team abbreviations.

Usage

```
team_abbr_mapping
```

Format

A named character vector

name attribute The "alternate" name.

value attribute The "correct" name.

Details

You can suggest additions to this table by opening an issue in nflreadr.

See Also

`team_abbr_mapping_norelocate` for the same thing but relocations stay in their original cities.

Examples

```
team_abbr_mapping[c("STL", "OAK","CRD","BLT", "CLV")]
```

`team_abbr_mapping_norelocate` *Alternate team abbreviation mappings, no relocation*

Description

A named character vector mapping common alternate team abbreviations, but does not follow relocations to their current city.

Usage

```
team_abbr_mapping_norelocate
```

Format

A named character vector

name attribute The "alternate" name.

value attribute The "correct" name.

Details

You can suggest additions to this table by opening an issue in nflreadr.

Examples

```
team_abbr_mapping_norelocate[c("STL", "OAK","CRD","BLT", "CLV")]
```

Index