Package: ffscrapr (via r-universe)

October 2, 2024

Type Package

Title API Client for Fantasy Football League Platforms

Version 1.4.8.19

Description Helps access various Fantasy Football APIs by handling authentication and rate-limiting, forming appropriate calls, and returning tidy dataframes which can be easily connected to other data sources.

License MIT + file LICENSE

URL https://ffscrapr.ffverse.com, https://github.com/ffverse/ffscrapr

BugReports https://github.com/ffverse/ffscrapr/issues

Depends R (>= 3.6.0)

Imports cachem (>= 1.0.0), checkmate (>= 2.0.0), cli (>= 3.0.0), data.table (>= 1.14.8), dplyr (>= 1.0.0), glue (>= 1.3.0), httr (>= 1.4.0), jsonlite (>= 1.6.0), lifecycle (>= 1.0.0), magrittr (>= 1.5.0), nflreadr (>= 1.2.0), memoise (>= 2.0.0), purrr (>= 0.3.0), rappdirs (>= 0.3.0), ratelimitr (>= 0.4.0), rlang (>= 0.4.0), stringr (>= 1.4.0), tibble (>= 3.0.0), tidyr (>= 1.0.0)

Suggests covr (>= 3.0.0), curl (>= 4.0.0), httptest (>= 3.0.0), knitr (>= 1.0), piggyback (>= 0.1.5), rmarkdown (>= 2.6), testthat (>= 3.0.0), withr (>= 2.4.0)

LazyData true

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.3.1

Roxygen list(markdown = TRUE)

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

RemoteUrl https://github.com/ffverse/ffscrapr

RemoteRef main

RemoteSha 72cba2dd46f81134e58553ba94a5bba029e9ce5b

Contents

clear_cache *Empty Function Cache*

Description

This function will reset the cache for any and all ffscrapr cached functions, as well as nflreadr's cached functions.

Usage

```
clear_cache()
```

dp_clean_html *Remove HTML from string*

Description

Applies some regex to clean html tags from strings. This is useful for platforms such as MFL that interpret HTML in their franchise name fields.

Usage

```
dp_clean_html(names)
```

Arguments

names	a character (or character vector)

Value

a character vector of cleaned strings

Examples

```
c(
  "<b><font color= Cyan>Kevin OBrien (@kevinobrienff) </FONT></B>",
  "<em><font color= Purple> Other fun names</font></em>"
) %>% dp_clean_html()
```

`dp_clean_names` *Clean Player 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

```
dp_clean_names(...)

dp_cleannames(...)
```

Arguments

`...`	Arguments passed on to `nflreadr::clean_player_names` `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

See Also

`nflreadr::clean_player_names()`

Examples

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

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

`dp_playerids` *Import latest DynastyProcess player IDs*

Description

Fetches a copy of the latest DynastyProcess player IDs csv

Usage

```
dp_playerids()
```

Value

a tibble of player IDs

See Also

https://github.com/DynastyProcess/data

Examples

```
try( # try only shown here because sometimes CRAN checks are weird
  dp_playerids()
)
```

dp_values *Import latest DynastyProcess values*

Description

Fetches a copy of the latest DynastyProcess dynasty trade values sheets

Usage

```
dp_values(file = c("values.csv", "values-players.csv", "values-picks.csv"))
```

Arguments

file	one of c("values.csv","values-players.csv","values-picks.csv")

Value

a tibble of trade values from DynastyProcess

See Also

https://github.com/DynastyProcess/data

Examples

```
try( # try only shown here because sometimes CRAN checks are weird
  dp_values()
)
```

espn_connect *Connect to ESPN League*

Description

This function creates a connection object which stores parameters and a user ID if available.

Usage

```
espn_connect(
  season = NULL,
  league_id = NULL,
  swid = NULL,
  espn_s2 = NULL,
  user_agent = NULL,
  rate_limit = TRUE,
  rate_limit_number = NULL,
  rate_limit_seconds = NULL,
  ...
)
```

Arguments

season	Season to access on Fleaflicker - if missing, will guess based on system date (current year if March or later, otherwise previous year)
league_id	League ID
swid	SWID parameter for accessing private leagues - see vignette for details
espn_s2	ESPN_S2 parameter for accessing private leagues - see vignette for details
user_agent	User agent to self-identify (optional)
rate_limit	TRUE by default - turn off rate limiting with FALSE
rate_limit_number	number of calls per rate_limit_seconds, suggested is under 1000 calls per 60 seconds
rate_limit_seconds	number of seconds as denominator for rate_limit
...	other arguments (for other methods, for R compat)

Value

a list that stores ESPN connection objects

Examples

```
conn <- espn_connect(
  season = 2018,
  league_id = 1178049,
  espn_s2 = Sys.getenv("TAN_ESPN_S2"),
  swid = Sys.getenv("TAN_SWID")
)
```

`espn_getendpoint` *GET ESPN fantasy league endpoint*

Description

This function is used to call the ESPN Fantasy API for league-based endpoints.

Usage

```
espn_getendpoint(conn, ..., x_fantasy_filter = NULL)
```

Arguments

`conn`	a connection object created by `espn_connect` or `ff_connect()`
`...`	Arguments which will be passed as "argumentname = argument" in an HTTP query parameter
`x_fantasy_filter`	a JSON-encoded character string that specifies a filter for the data

Details

The ESPN Fantasy API is undocumented and this should be used by advanced users familiar with the API.

It chooses the correct league endpoint based on the year (eg leagueHistory for <2018), checks the x_fantasy_filter for valid JSON input, builds a url with any optional query parameters, and executes the request with authentication and rate limiting.

HTTP query parameters (i.e. arguments to ...) are Case Sensitive.

Please see the vignette for more on usage.

Value

A list object containing the query, response, and parsed content.

See Also

`vignette("espn_getendpoint")`

`espn_getendpoint_raw`

espn_getendpoint_raw *GET ESPN endpoint (raw)*

Description

This function is the lower-level function that powers the API call: it takes a URL and headers and executes the http request with rate-limiting and authentication. It checks for JSON return and any warnings/errors, parses the json, and returns an espn_api object with the parsed content, the raw response, and the actual query.

Usage

```
espn_getendpoint_raw(conn, url_query, ...)
```

Arguments

conn	a connection object created by ff_connect or equivalent - used for authentication
url_query	a fully-formed URL to call
...	any headers or other httr request objects to pass along

Value

object of class espn_api with parsed content, request, and response

See Also

`espn_getendpoint()` - a higher level wrapper that checks JSON and prepares the url query

`vignette("espn_getendpoint")`

espn_players *ESPN players library*

Description

A cached table of ESPN NFL players. Will store in memory for each session! (via memoise in zzz.R)

Usage

```
espn_players(conn = NULL, season = NULL)
```

Arguments

conn	a connection object created by `espn_connect` or `ff_connect()`
season	a season to fetch

Value

a dataframe containing all ~2000+ active players in the ESPN database

Examples

```
try({ # try only shown here because sometimes CRAN checks are weird

  conn <- espn_connect(season = 2020, league_id = 1178049)

  espn_players(conn, season = 2020)
}) # end try
```

---

espn_potentialpoints *ESPN Potential Points*

Description

This function calculates the optimal starters for a given week, using some lineup heuristics.

Usage

```
espn_potentialpoints(conn, weeks = 1:17)
```

Arguments

conn	the list object created by ff_connect()
weeks	a numeric vector for determining which weeks to calculate

Value

a tibble with the best lineup for each team and whether they were started or not

Examples

```
try({ # try only shown here because sometimes CRAN checks are weird
  conn <- espn_connect(season = 2021, league_id = 950665)
  espn_potentialpoints(conn, weeks = 1)
}) # end try
```

---

ffverse_sitrep *ffverse sitrep*

Description

See nflreadr::ffverse_sitrep for details.

ff_connect *Connect to a League*

Description

This function creates a connection object which stores parameters and gets a login-cookie if available - it does so by passing arguments to the appropriate league-based handler.

Usage

```
ff_connect(platform = "mfl", league_id = NULL, ...)
```

Arguments

platform	one of MFL or Sleeper (Fleaflicker, ESPN, Yahoo in approximate priority order going forward)
league_id	league_id (currently assuming one league at a time)
...	other parameters passed to the connect function for each specific platform.

Value

a connection object to be used with ff_* functions

See Also

mfl_connect(), sleeper_connect(), fleaflicker_connect(), espn_connect()

Examples

```
ff_connect(platform = "mfl", season = 2019, league_id = 54040, rate_limit = FALSE)
```

ff_draft *Get Draft Results*

Description

This function gets a tidy dataframe of draft results for the current year. Can handle MFL devy drafts or startup drafts by specifying the custom_players argument

Usage

```
ff_draft(conn, ...)

## S3 method for class 'espn_conn'
ff_draft(conn, ...)

## S3 method for class 'flea_conn'
ff_draft(conn, ...)

## S3 method for class 'mfl_conn'
ff_draft(conn, custom_players = deprecated(), ...)

## S3 method for class 'sleeper_conn'
ff_draft(conn, ...)
```

Arguments

conn	a conn object created by `ff_connect()`
...	args for other methods
custom_players	**[Deprecated]** - now returns custom players by default

Value

A tidy dataframe of draft results

Methods (by class)

- `ff_draft(espn_conn)`: ESPN: returns the current year's draft/auction, including details on keepers
- `ff_draft(flea_conn)`: Fleaflicker: returns a table of drafts for the current year
- `ff_draft(mfl_conn)`: MFL: returns a table of drafts for the current year - can handle devy/startup-rookie-picks by specifying custom_players (slower!)
- `ff_draft(sleeper_conn)`: Sleeper: returns a dataframe of all drafts and draft selections, if available.

Examples

```
try({ # try only shown here because sometimes CRAN checks are weird
  conn <- espn_connect(season = 2020, league_id = 899513)
  ff_draft(conn)
}) # end try



try({ # try only shown here because sometimes CRAN checks are weird
  conn <- fleaflicker_connect(season = 2020, league_id = 206154)
  ff_draft(conn)
}) # end try
```

```
try({ # try only shown here because sometimes CRAN checks are weird
  ssb_conn <- ff_connect(platform = "mfl", league_id = 54040, season = 2020)
  ff_draft(ssb_conn)
}) # end try
```

```
try({ # try only shown here because sometimes CRAN checks are weird
 jml_conn <- ff_connect(platform = "sleeper", league_id = "522458773317046272", season = 2020)
  ff_draft(jml_conn)
}) # end try
```

`ff_draftpicks` *Get Draft Picks*

Description

Returns all draft picks (current and future) that belong to a specific franchise and have not yet been converted into players (i.e. selected.)

Usage

```
ff_draftpicks(conn, ...)

## S3 method for class 'espn_conn'
ff_draftpicks(conn, ...)

## S3 method for class 'flea_conn'
ff_draftpicks(conn, franchise_id = NULL, ...)

## S3 method for class 'mfl_conn'
ff_draftpicks(conn, ...)

## S3 method for class 'sleeper_conn'
ff_draftpicks(conn, ...)
```

Arguments

`conn`	the list object created by `ff_connect()`
`...`	other arguments (currently unused)
`franchise_id`	A list of franchise IDs to pull, if NULL will return all franchise IDs

Value

Returns a dataframe with current and future draft picks for each franchise

Methods (by class)

- `ff_draftpicks(espn_conn)`: ESPN: does not support future/draft pick trades - for draft results, please use ff_draft.
- `ff_draftpicks(flea_conn)`: Fleaflicker: retrieves current and future draft picks, potentially for a specified team.
- `ff_draftpicks(mfl_conn)`: MFL: returns current and future picks
- `ff_draftpicks(sleeper_conn)`: Sleeper: retrieves current and future draft picks

Examples

```
try({ # try only shown here because sometimes CRAN checks are weird
  conn <- espn_connect(
    season = 2018,
    league_id = 1178049,
    espn_s2 = Sys.getenv("TAN_ESPN_S2"),
    swid = Sys.getenv("TAN_SWID")
  )

  ff_draftpicks(conn)
}) # end try



try({ # try only shown here because sometimes CRAN checks are weird
  conn <- fleaflicker_connect(2020, 206154)
  ff_draftpicks(conn, franchise_id = 1373475)
}) # end try



try({ # try only shown here because sometimes CRAN checks are weird
  dlf_conn <- mfl_connect(2020, league_id = 37920)
  ff_draftpicks(conn = dlf_conn)
}) # end try



try({ # try only shown here because sometimes CRAN checks are weird
 jml_conn <- ff_connect(platform = "sleeper", league_id = "522458773317046272", season = 2020)
  ff_draftpicks(jml_conn)
}) # end try
```

`ff_franchises` *Get League Franchises*

Description

Return franchise-level data (including divisions, usernames, etc) - available data may vary slightly based on platform.

Usage

```
ff_franchises(conn)

## S3 method for class 'espn_conn'
ff_franchises(conn)

## S3 method for class 'flea_conn'
ff_franchises(conn)

## S3 method for class 'mfl_conn'
ff_franchises(conn)

## S3 method for class 'sleeper_conn'
ff_franchises(conn)
```

Arguments

`conn`	a conn object created by `ff_connect()`

Value

A tidy dataframe of franchises, complete with IDs

Methods (by class)

- `ff_franchises(espn_conn)`: ESPN: returns franchise and division information.
- `ff_franchises(flea_conn)`: Fleaflicker: returns franchise and division information.
- `ff_franchises(mfl_conn)`: MFL: returns franchise and division information.
- `ff_franchises(sleeper_conn)`: Sleeper: retrieves a list of franchise information, including user IDs and co-owner IDs.

Examples

```
try({ # try only shown here because sometimes CRAN checks are weird

  conn <- espn_connect(season = 2020, league_id = 1178049)

  ff_franchises(conn)
```

```
}) # end try


try({ # try only shown here because sometimes CRAN checks are weird
  conn <- fleaflicker_connect(season = 2020, league_id = 206154)
  ff_franchises(conn)
}) # end try


try({ # try only shown here because sometimes CRAN checks are weird
  ssb_conn <- ff_connect(platform = "mfl", league_id = 54040, season = 2020)
  ff_franchises(ssb_conn)
}) # end try


try({ # try only shown here because sometimes CRAN checks are weird
 jml_conn <- ff_connect(platform = "sleeper", league_id = "522458773317046272", season = 2020)
  ff_franchises(jml_conn)
}) # end try
```

`ff_league` *Get League Summary*

Description

This function returns a tidy dataframe of common league settings, including details like "1QB" or "2QB/SF", scoring, best ball, team count, IDP etc. This is potentially useful in summarising the features of multiple leagues.

Usage

```
ff_league(conn)

## S3 method for class 'espn_conn'
ff_league(conn)

## S3 method for class 'flea_conn'
ff_league(conn)

## S3 method for class 'mfl_conn'
ff_league(conn)

## S3 method for class 'sleeper_conn'
ff_league(conn)
```

Arguments

conn	the connection object created by `ff_connect()`

Value

A one-row summary of each league's main features.

Methods (by class)

- `ff_league(espn_conn)`: ESPN: returns a summary of league features.
- `ff_league(flea_conn)`: Flea: returns a summary of league features.
- `ff_league(mfl_conn)`: MFL: returns a summary of league features.
- `ff_league(sleeper_conn)`: Sleeper: returns a summary of league features.

Examples

```
try({ # try only shown here because sometimes CRAN checks are weird

  conn <- espn_connect(season = 2020, league_id = 899513)

  ff_league(conn)
}) # end try



try({ # try only shown here because sometimes CRAN checks are weird
  conn <- fleaflicker_connect(2020, 206154)
  ff_league(conn)
}) # end try



try({ # try only shown here because sometimes CRAN checks are weird
  ssb_conn <- ff_connect(platform = "mfl", league_id = 22627, season = 2021)
  ff_league(ssb_conn)
}) # end try



try({ # try only shown here because sometimes CRAN checks are weird
 jml_conn <- ff_connect(platform = "sleeper", league_id = "522458773317046272", season = 2020)
  ff_league(jml_conn)
}) # end try
```

`ff_playerscores` *Get Player Scoring History*

Description

This function returns a tidy dataframe of player scores based on league rules.

Unfortunately, Sleeper has deprecated their player stats endpoint from their supported/open API. Please see `ff_scoringhistory()` for an alternative reconstruction.

Usage

```
ff_playerscores(conn, ...)

## S3 method for class 'espn_conn'
ff_playerscores(conn, limit = 1000, ...)

## S3 method for class 'flea_conn'
ff_playerscores(conn, page_limit = NULL, ...)

## S3 method for class 'mfl_conn'
ff_playerscores(conn, season, week, ...)

## S3 method for class 'sleeper_conn'
ff_playerscores(conn, ...)
```

Arguments

`conn`	the list object created by `ff_connect()`
`...`	other arguments (currently unused)
`limit`	A numeric describing the number of players to return - default 1000
`page_limit`	A numeric describing the number of pages to return - default NULL returns all available
`season`	the season of interest - generally only the most recent 2-3 seasons are available
`week`	a numeric vector (ie 1:17) or one of YTD (year-to-date) or AVG (average to date)

Value

A tibble of historical player scoring

Methods (by class)

- `ff_playerscores(espn_conn)`: ESPN: returns total points for season and average per game, for both current and previous season.

- `ff_playerscores(flea_conn)`: Fleaflicker: returns the season, season average, and standard deviation
- `ff_playerscores(mfl_conn)`: MFL: returns the player fantasy scores for each week (not the actual stats)
- `ff_playerscores(sleeper_conn)`: Sleeper: Deprecated their open API endpoint for player scores

See Also

ff_scoringhistory

Examples

```
try({ # try only shown here because sometimes CRAN checks are weird

  conn <- espn_connect(season = 2020, league_id = 899513)

  ff_playerscores(conn, limit = 5)
}) # end try


try({ # try only shown here because sometimes CRAN checks are weird
  conn <- fleaflicker_connect(2020, 312861)
  ff_playerscores(conn, page_limit = 2)
}) # end try


try({ # try only shown here because sometimes CRAN checks are weird
  sfb_conn <- mfl_connect(2020, league_id = 65443)
  ff_playerscores(conn = sfb_conn, season = 2019, week = "YTD")
}) # end try
```

---

ff_rosters *Get League Rosters*

Description

This function returns a tidy dataframe of team rosters

Usage

```
ff_rosters(conn, ...)

## S3 method for class 'espn_conn'
ff_rosters(conn, week = NULL, ...)

## S3 method for class 'flea_conn'
```

```
ff_rosters(conn, ..., week = NULL)

## S3 method for class 'mfl_conn'
ff_rosters(conn, custom_players = deprecated(), week = NULL, ...)

## S3 method for class 'sleeper_conn'
ff_rosters(conn, ...)
```

Arguments

conn	a conn object created by ff_connect()
...	arguments passed to other methods (currently none)
week	a numeric that specifies which week to return
custom_players	"**[Deprecated]**" - now returns custom players by default

Value

A tidy dataframe of rosters, joined to basic player information and basic franchise information

Methods (by class)

- ff_rosters(espn_conn): ESPN: Returns all roster data.
- ff_rosters(flea_conn): Fleaflicker: Returns roster data (minus age as of right now)
- ff_rosters(mfl_conn): MFL: returns roster data
- ff_rosters(sleeper_conn): Sleeper: Returns all roster data.

Examples

```
try({ # try only shown here because sometimes CRAN checks are weird
  conn <- espn_connect(season = 2020, league_id = 899513)
  ff_league(conn)
}) # end try


try({ # try only shown here because sometimes CRAN checks are weird
  joe_conn <- ff_connect(platform = "fleaflicker", league_id = 312861, season = 2020)

  ff_rosters(joe_conn)
}) # end try


try({ # try only shown here because sometimes CRAN checks are weird
  ssb_conn <- ff_connect(platform = "mfl", league_id = 54040, season = 2020)
  ff_rosters(ssb_conn)
}) # end try



try({ # try only shown here because sometimes CRAN checks are weird
```

```
 jml_conn <- ff_connect(platform = "sleeper", league_id = "522458773317046272", season = 2020)
 ff_rosters(jml_conn)
}) # end try
```

`ff_schedule` *Get Schedule*

Description

This function returns a tidy dataframe with one row for every team for every weekly matchup

Usage

```
ff_schedule(conn, ...)

## S3 method for class 'espn_conn'
ff_schedule(conn, ...)

## S3 method for class 'flea_conn'
ff_schedule(conn, week = 1:17, ...)

## S3 method for class 'mfl_conn'
ff_schedule(conn, ...)

## S3 method for class 'sleeper_conn'
ff_schedule(conn, ...)
```

Arguments

`conn`	a conn object created by `ff_connect()`
`...`	for other platforms
`week`	a numeric or numeric vector specifying which weeks to pull

Value

A tidy dataframe with one row per game per franchise per week

Methods (by class)

- `ff_schedule(espn_conn)`: ESPN: returns schedule data, one row for every franchise for every week. Completed games have result data.
- `ff_schedule(flea_conn)`: Flea: returns schedule data, one row for every franchise for every week. Completed games have result data.
- `ff_schedule(mfl_conn)`: MFL: returns schedule data, one row for every franchise for every week. Completed games have result data.
- `ff_schedule(sleeper_conn)`: Sleeper: returns all schedule data

Examples

```
try({ # try only shown here because sometimes CRAN checks are weird
  espn_conn <- espn_connect(season = 2020, league_id = 899513)
  ff_schedule(espn_conn)
}) # end try



try({ # try only shown here because sometimes CRAN checks are weird
  conn <- fleaflicker_connect(season = 2019, league_id = 206154)
  ff_schedule(conn, week = 2:4)
}) # end try



try({ # try only shown here because sometimes CRAN checks are weird
  ssb_conn <- ff_connect(platform = "mfl", league_id = 54040, season = 2020)
  ff_schedule(ssb_conn)
}) # end try



try({ # try only shown here because sometimes CRAN checks are weird
 jml_conn <- ff_connect(platform = "sleeper", league_id = "522458773317046272", season = 2020)
  ff_schedule(jml_conn)
}) # end try
```

---

ff_scoring *Get League Scoring Settings*

Description

This function returns a dataframe with detailed scoring settings for each league - broken down by event, points, and (if available) position.

Usage

```
ff_scoring(conn)

## S3 method for class 'espn_conn'
ff_scoring(conn)

## S3 method for class 'flea_conn'
ff_scoring(conn)

## S3 method for class 'mfl_conn'
ff_scoring(conn)
```

```
## S3 method for class 'sleeper_conn'
ff_scoring(conn)

## S3 method for class 'template_conn'
ff_scoring(conn)
```

Arguments

conn	a conn object created by `ff_connect()`

Value

A tibble of league scoring rules for each position defined.

Methods (by class)

- `ff_scoring(espn_conn)`: ESPN: returns scoring settings in a flat table, override positions have their own scoring.
- `ff_scoring(flea_conn)`: Fleaflicker: returns scoring settings in a flat table, one row per position per rule.
- `ff_scoring(mfl_conn)`: MFL: returns scoring settings in a flat table, one row per position per rule.
- `ff_scoring(sleeper_conn)`: Sleeper: returns scoring settings in a flat table, one row per position per rule.
- `ff_scoring(template_conn)`: Template: returns MFL style scoring settings in a flat table, one row per position per rule.

See Also

http://www03.myfantasyleague.com/2020/scoring_rules#rules

Examples

```
try({ # try only shown here because sometimes CRAN checks are weird
  conn <- espn_connect(season = 2020, league_id = 899513)
  ff_scoring(conn)
}) # end try


try({ # try only shown here because sometimes CRAN checks are weird
  joe_conn <- ff_connect(platform = "fleaflicker", league_id = 312861, season = 2020)
  ff_scoring(joe_conn)
}) # end try


try({ # try only shown here because sometimes CRAN checks are weird
  ssb_conn <- ff_connect(platform = "mfl", league_id = 54040, season = 2020)
```

```
  ff_scoring(ssb_conn)
}) # end try



try({ # try only shown here because sometimes CRAN checks are weird
 jml_conn <- ff_connect(platform = "sleeper", league_id = "522458773317046272", season = 2020)
  ff_scoring(jml_conn)
}) # end try



template_ppr <- ff_template(scoring_type = "ppr")
ff_scoring(template_ppr)
```

`ff_scoringhistory` *Get League-Specific Scoring History*

Description

(Experimental!) This function reads your league's ff_scoring rules and maps them to nflfastr week-level data. Not all of the scoring rules from your league may have nflfastr equivalents, but most of the common ones are available!

Usage

```
ff_scoringhistory(conn, season, ...)

## S3 method for class 'espn_conn'
ff_scoringhistory(conn, season = 1999:nflreadr::most_recent_season(), ...)

## S3 method for class 'flea_conn'
ff_scoringhistory(conn, season = 1999:nflreadr::most_recent_season(), ...)

## S3 method for class 'mfl_conn'
ff_scoringhistory(conn, season = 1999:nflreadr::most_recent_season(), ...)

## S3 method for class 'sleeper_conn'
ff_scoringhistory(conn, season = 1999:nflreadr::most_recent_season(), ...)

## S3 method for class 'template_conn'
ff_scoringhistory(conn, season = 1999:nflreadr::most_recent_season(), ...)
```

Arguments

conn	a conn object created by `ff_connect()`

season	season a numeric vector of seasons (earliest available year is 1999)
...	other arguments

Value

A tidy dataframe of weekly fantasy scoring data, one row per player per week

Methods (by class)

- `ff_scoringhistory(espn_conn)`: ESPN: returns scoring history in a flat table, one row per player per week.
- `ff_scoringhistory(flea_conn)`: Fleaflicker: returns scoring history in a flat table, one row per player per week.
- `ff_scoringhistory(mfl_conn)`: MFL: returns scoring history in a flat table, one row per player per week.
- `ff_scoringhistory(sleeper_conn)`: Sleeper: returns scoring history in a flat table, one row per player per week.
- `ff_scoringhistory(template_conn)`: template: returns scoring history in a flat table, one row per player per week.

See Also

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

Examples

```
try({ # try only shown here because sometimes CRAN checks are weird
  conn <- espn_connect(season = 2020, league_id = 899513)
  ff_scoringhistory(conn, season = 2020)
}) # end try



try({ # try only shown here because sometimes CRAN checks are weird
  conn <- fleaflicker_connect(2020, 312861)
  ff_scoringhistory(conn, season = 2020)
}) # end try



try({ # try only shown here because sometimes CRAN checks are weird
  ssb_conn <- ff_connect(platform = "mfl", league_id = 54040, season = 2020)
  ff_scoringhistory(ssb_conn, season = 2020)
}) # end try



try({ # try only shown here because sometimes CRAN checks are weird
 conn <- ff_connect(platform = "sleeper", league_id = "522458773317046272", season = 2020)
  ff_scoringhistory(conn, season = 2020)
```

```
}) # end try


try({ # try only shown here because sometimes CRAN checks are weird
  template_conn <- ff_template(scoring_type = "sfb11", roster_type = "sfb11")
  ff_scoringhistory(template_conn, season = 2020)
}) # end try
```

`ff_standings` *Get Standings*

Description

This function returns a tidy dataframe of season-long fantasy team stats, including H2H wins as well as points, potential points, and all-play.

Usage

```
ff_standings(conn, ...)

## S3 method for class 'espn_conn'
ff_standings(conn, ...)

## S3 method for class 'flea_conn'
ff_standings(conn, include_allplay = TRUE, include_potentialpoints = TRUE, ...)

## S3 method for class 'mfl_conn'
ff_standings(conn, ...)

## S3 method for class 'sleeper_conn'
ff_standings(conn, ...)
```

Arguments

`conn`	a conn object created by `ff_connect()`
`...`	arguments passed to other methods (currently none)
`include_allplay`	TRUE/FALSE - return all-play win pct calculation? defaults to TRUE
`include_potentialpoints`	TRUE/FALSE - return potential points calculation? defaults to TRUE.

Value

A tidy dataframe of standings data

Methods (by class)

- `ff_standings(espn_conn)`: ESPN: returns standings and points data.
- `ff_standings(flea_conn)`: Fleaflicker: returns H2H/points/all-play/best-ball data in a table.
- `ff_standings(mfl_conn)`: MFL: returns H2H/points/all-play/best-ball data in a table.
- `ff_standings(sleeper_conn)`: Sleeper: returns all standings and points data and manually calculates allplay results.

Examples

```
try({ # try only shown here because sometimes CRAN checks are weird
  espn_conn <- espn_connect(season = 2020, league_id = 899513)
  ff_standings(espn_conn)
}) # end try



try({ # try only shown here because sometimes CRAN checks are weird
  conn <- fleaflicker_connect(season = 2020, league_id = 206154)
  x <- ff_standings(conn)
}) # end try



try({ # try only shown here because sometimes CRAN checks are weird
  ssb_conn <- ff_connect(platform = "mfl", league_id = 54040, season = 2020)
  ff_standings(ssb_conn)
}) # end try



try({ # try only shown here because sometimes CRAN checks are weird
 jml_conn <- ff_connect(platform = "sleeper", league_id = "522458773317046272", season = 2020)
  ff_standings(jml_conn)
}) # end try
```

---

ff_starters *Get Starting Lineups*

Description

This function returns a tidy dataframe with one row for every starter (and bench) for every week and their scoring, if available.

Usage

```
ff_starters(conn, ...)

## S3 method for class 'espn_conn'
ff_starters(conn, weeks = 1:17, ...)

## S3 method for class 'flea_conn'
ff_starters(conn, week = 1:17, ...)

## S3 method for class 'mfl_conn'
ff_starters(conn, week = 1:17, season = NULL, ...)

## S3 method for class 'sleeper_conn'
ff_starters(conn, week = 1:17, ...)
```

Arguments

conn	the list object created by `ff_connect()`
...	other arguments (currently unused)
weeks	which weeks to calculate, a number or numeric vector
week	a numeric or one of YTD (year-to-date) or AVG (average to date)
season	the season of interest - generally only the most recent 2-3 seasons are available

Value

A tidy dataframe with every player for every week, including a flag for whether they were started or not

Methods (by class)

- `ff_starters(espn_conn)`: ESPN: returns who was started as well as what they scored.
- `ff_starters(flea_conn)`: Fleaflicker: returns who was started as well as what they scored.
- `ff_starters(mfl_conn)`: MFL: returns the player fantasy scores for each week (not the actual stats)
- `ff_starters(sleeper_conn)`: Sleeper: returns only "who" was started, without any scoring/stats data. Only returns season specified in initial connection object.

Examples

```
try({ # try only shown here because sometimes CRAN checks are weird
  conn <- espn_connect(season = 2020, league_id = 1178049)
  ff_starters(conn, weeks = 1:3)
}) # end try



try({ # try only shown here because sometimes CRAN checks are weird
```

```
  conn <- fleaflicker_connect(season = 2020, league_id = 206154)
  ff_starters(conn)
}) # end try



try({ # try only shown here because sometimes CRAN checks are weird
  dlf_conn <- mfl_connect(2020, league_id = 37920)
  ff_starters(conn = dlf_conn)
}) # end try



try({ # try only shown here because sometimes CRAN checks are weird
  jml_conn <- sleeper_connect(league_id = "522458773317046272", season = 2020)
  ff_starters(jml_conn, week = 3)
}) # end try
```

`ff_starter_positions` *Get Starting Lineup Settings*

Description

This function returns a tidy dataframe with positional lineup rules.

Usage

```
ff_starter_positions(conn, ...)

## S3 method for class 'espn_conn'
ff_starter_positions(conn, ...)

## S3 method for class 'flea_conn'
ff_starter_positions(conn, ...)

## S3 method for class 'mfl_conn'
ff_starter_positions(conn, ...)

## S3 method for class 'sleeper_conn'
ff_starter_positions(conn, ...)

## S3 method for class 'template_conn'
ff_starter_positions(conn, ...)
```

Arguments

`conn`	the list object created by `ff_connect()`
`...`	other arguments (currently unused)

Value

A tidy dataframe of positional lineup rules, one row per position with minimum and maximum starters as well as total starter calculations.

Methods (by class)

- `ff_starter_positions(espn_conn)`: ESPN: returns min/max starters for each main player position
- `ff_starter_positions(flea_conn)`: Fleaflicker: returns minimum and maximum starters for each player position.
- `ff_starter_positions(mfl_conn)`: MFL: returns minimum and maximum starters for each player position.
- `ff_starter_positions(sleeper_conn)`: Sleeper: returns minimum and maximum starters for each player position.
- `ff_starter_positions(template_conn)`: Template: returns minimum and maximum starters for each player position.

Examples

```
try({ # try only shown here because sometimes CRAN checks are weird
  conn <- espn_connect(season = 2020, league_id = 1178049)
  ff_starter_positions(conn)
}) # end try



try({ # try only shown here because sometimes CRAN checks are weird
  conn <- fleaflicker_connect(season = 2020, league_id = 206154)
  ff_starter_positions(conn)
}) # end try



try({ # try only shown here because sometimes CRAN checks are weird
  dlfidp_conn <- mfl_connect(2020, league_id = 33158)
  ff_starter_positions(conn = dlfidp_conn)
}) # end try



try({ # try only shown here because sometimes CRAN checks are weird
  jml_conn <- sleeper_connect(league_id = "652718526494253056", season = 2021)
  ff_starter_positions(jml_conn)
}) # end try



template_conn <- ff_template(roster_type = "idp")
ff_starter_positions(template_conn)
```

`ff_template` *Default* conn *objects*

Description

This function creates a connection to a few league templates, and can be used instead of a real conn object in the following functions: `ff_scoring()`, `ff_scoringhistory()`, `ff_starterpositions()`.

Usage

```
ff_template(
  scoring_type = c("ppr", "half_ppr", "zero_ppr", "sfb11"),
  roster_type = c("1qb", "superflex", "sfb11", "idp")
)
```

Arguments

`scoring_type`	One of c("default", "ppr", "half_ppr", "zero_ppr", "te_prem", "sfb11")
`roster_type`	One of c("1qb", "superflex","sfb11", "idp")

Details

Scoring types defined here are:

- `ppr`: 6 pt passing/rushing/receiving touchdowns, 0.1 for rushing/receiving yards, 1 point per reception, -2 for fumbles/interceptions
- `half_ppr`: same as `ppr` but with 0.5 points per reception
- `zero_ppr`: same as `ppr` but with 0 points per reception
- `te_prem`: same as `ppr` but TEs get 1.5 points per reception
- `sfb11`: SFB11 scoring as defined by https://scottfishbowl.com

Roster settings defined here are:

- `1qb`: Starts 1 QB, 2 RB, 3 WR, 1 TE, 2 FLEX
- `superflex`: Starts 1 QB, 2 RB, 3 WR, 1 TE, 2 FLEX, 1 SUPERFLEX
- `sfb11`: Starts 1 QB, 2 RB, 3 WR, 1 TE, 3 FLEX, 1 SUPERFLEX (flex positions can also start a kicker)
- `idp`: Starts same as 1QB but also starts 3 DL, 3 LB, 3 DB, and two IDP FLEX

Value

a connection object that can be used with `ff_scoring()`, `ff_scoringhistory()`, and `ff_starterpositions()`

`ff_transactions` *Get League Transactions*

Description

This function returns a tidy dataframe of transactions - generally one row per player per transaction per team. Each trade is represented twice, once per each team.

Usage

```
ff_transactions(conn, ...)

## S3 method for class 'espn_conn'
ff_transactions(conn, limit = 1000, ...)

## S3 method for class 'flea_conn'
ff_transactions(conn, franchise_id = NULL, ...)

## S3 method for class 'mfl_conn'
ff_transactions(conn, transaction_type = "*", ...)

## S3 method for class 'sleeper_conn'
ff_transactions(conn, week = 1:17, ...)
```

Arguments

`conn`	the list object created by `ff_connect()`
`...`	additional args for other methods
`limit`	number of most recent transactions to return
`franchise_id`	fleaflicker returns transactions grouped by franchise id, pass a list here to filter
`transaction_type`	parameter to return transactions of the specified type. Types are: `WAIVER`, `BBID_WAIVER`, `FREE_AGENT`, `TRADE`, `IR`, `TAXI`, `AUCTION_INIT`, `AUCTION_BID`, `AUCTION_WON`, or `*` for all. Can also pass a comma-separated string. Defaults to `*`. Note that only the types listed above are actually cleaned and processed by ffscrapr - you will need to make a custom api request with `mfl_getendpoint()` to receive other things.
`week`	A week filter for transactions - 1 returns all offseason transactions. Default 1:17 returns all transactions.

Value

A tidy dataframe of transaction data

Methods (by class)

- `ff_transactions(espn_conn)`: ESPN: returns adds, drops, and trades. Requires private/auth-cookie.
- `ff_transactions(flea_conn)`: Fleaflicker: returns all transactions, including free agents, waivers, and trades.
- `ff_transactions(mfl_conn)`: MFL: returns all transactions, including auction, free agents, IR, TS, waivers, and trades.
- `ff_transactions(sleeper_conn)`: Sleeper: returns all transactions, including free agents, waivers, and trades.

Examples

```
## Not run:
# Marked as don't run because this endpoint requires private authentication

conn <- espn_connect(
  season = 2020,
  league_id = 1178049,
  swid = Sys.getenv("TAN_SWID"),
  espn_s2 = Sys.getenv("TAN_ESPN_S2")
)
ff_transactions(conn)

## End(Not run)


try({ # try only shown here because sometimes CRAN checks are weird
  conn <- fleaflicker_connect(season = 2020, league_id = 312861)
  ff_transactions(conn)
}) # end try



try({ # try only shown here because sometimes CRAN checks are weird
  dlf_conn <- mfl_connect(2019, league_id = 37920)
  ff_transactions(dlf_conn)
}) # end try


try({ # try only shown here because sometimes CRAN checks are weird
 jml_conn <- ff_connect(platform = "sleeper", league_id = "522458773317046272", season = 2020)
  ff_transactions(jml_conn, week = 1:2)
}) # end try
```

`ff_userleagues` *Get User Leagues*

Description

This function returns a tidy dataframe with one row for every league a user is in. This requries authentication cookies for MFL usage.

Usage

```
ff_userleagues(conn, ...)

## S3 method for class 'espn_conn'
ff_userleagues(conn = NULL, ...)

## S3 method for class 'flea_conn'
ff_userleagues(conn = NULL, user_email = NULL, season = NULL, ...)

## S3 method for class 'mfl_conn'
ff_userleagues(conn, season = NULL, ...)

## S3 method for class 'sleeper_conn'
ff_userleagues(conn = NULL, user_name = NULL, season = NULL, ...)
```

Arguments

`conn`	a connection object created by `ff_connect()`
`...`	arguments that may be passed to other methods (for method consistency)
`user_email`	the username to look up - defaults to user created in conn if available
`season`	the season to look up leagues for
`user_name`	the username to look up - defaults to user created in conn if available

Value

A tidy dataframe with one row for every league a user is in

Methods (by class)

- `ff_userleagues(espn_conn)`: ESPN: does not support a lookup of user leagues by email or user ID at this time.
- `ff_userleagues(flea_conn)`: flea: returns a listing of leagues for a given user_email
- `ff_userleagues(mfl_conn)`: MFL: With username/password, it will return a list of user leagues.
- `ff_userleagues(sleeper_conn)`: Sleeper: returns a listing of leagues for a given user_id or user_name

See Also

`fleaflicker_userleagues()` to call this function for flea leagues without first creating a connection object.

`sleeper_userleagues()` to call this function for Sleeper leagues without first creating a connection object.

`fleaflicker_connect` *Connect to Fleaflicker League*

Description

This function creates a connection object which stores parameters and a user ID if available.

Usage

```
fleaflicker_connect(
  season = NULL,
  league_id = NULL,
  user_email = NULL,
  user_agent = NULL,
  rate_limit = TRUE,
  rate_limit_number = NULL,
  rate_limit_seconds = NULL,
  ...
)
```

Arguments

`season`	Season to access on Fleaflicker - if missing, will guess based on system date (current year if March or later, otherwise previous year)
`league_id`	League ID
`user_email`	Optional - attempts to get user's user ID by email
`user_agent`	User agent to self-identify (optional)
`rate_limit`	TRUE by default - turn off rate limiting with FALSE
`rate_limit_number`	number of calls per `rate_limit_seconds`, suggested is under 1000 calls per 60 seconds
`rate_limit_seconds`	number of seconds as denominator for rate_limit
`...`	other arguments (for other methods, for R compat)

Value

a list that stores Fleaflicker connection objects

fleaflicker_getendpoint *GET any Fleaflicker endpoint*

Description

The endpoint names and HTTP parameters (i.e. argument names) are CASE SENSITIVE and should be passed in exactly as displayed on the Fleaflicker API reference page.

Usage

```
fleaflicker_getendpoint(endpoint, ...)
```

Arguments

endpoint	a string defining which endpoint to return from the API
...	Arguments which will be passed as "argumentname = argument" in an HTTP query parameter

Details

Check out the vignette for more details and example usage.

Value

A list object containing the query, response, and parsed content.

See Also

https://www.fleaflicker.com/api-docs/index.html

```
vignette("fleaflicker_getendpoint")
```

fleaflicker_players *Fleaflicker players library*

Description

A cached table of Fleaflicker NFL players. Will store in memory for each session! (via memoise in zzz.R)

Usage

```
fleaflicker_players(conn, page_limit = NULL)
```

Arguments

conn	a conn object created by ff_connect()
page_limit	A number limiting the number of players to return, or NULL (default) returns all

Value

a dataframe containing all ~7000+ players in the Fleaflicker database

Examples

```
try({ # try only shown here because sometimes CRAN checks are weird
  conn <- fleaflicker_connect(2020, 312861)
  player_list <- fleaflicker_players(conn, page_limit = 2)
}) # end try
```

---

fleaflicker_userleagues

Fleaflicker - Get User Leagues

Description

This function returns the leagues that a specific user is in. This variant can be used without first creating a connection object.

Usage

```
fleaflicker_userleagues(user_email, season = NULL)
```

Arguments

user_email	the username to look up
season	the season to return leagues from - defaults to current year based on heuristics

Value

a dataframe of leagues for the specified user

See Also

ff_userleagues()

`mfl_connect` *Connect to MFL League*

Description

This function creates a connection object which stores parameters and gets a login-cookie if available

Usage

```
mfl_connect(
  season = NULL,
  league_id = NULL,
  APIKEY = NULL,
  user_name = NULL,
  password = NULL,
  user_agent = NULL,
  rate_limit = TRUE,
  rate_limit_number = NULL,
  rate_limit_seconds = NULL,
  ...
)
```

Arguments

`season`	Season to access on MFL - if missing, will guess based on system date (current year if March or later, otherwise previous year)
`league_id`	league_id Numeric ID parameter for each league, typically found in the URL
`APIKEY`	APIKEY - optional - allows access to private leagues. Key is unique for each league and accessible from Developer's API page (currently assuming one league at a time)
`user_name`	MFL user_name - optional - when supplied in conjunction with a password, will attempt to retrieve authentication token
`password`	MFL password - optional - when supplied in conjunction with user_name, will attempt to retrieve authentication token
`user_agent`	A string representing the user agent to be used to identify calls - may find improved rate_limits if verified token
`rate_limit`	TRUE by default, pass FALSE to turn off rate limiting
`rate_limit_number`	number of calls per `rate_limit_seconds`, suggested is 60 calls per 60 seconds
`rate_limit_seconds`	number of seconds as denominator for rate_limit
`...`	silently swallows up unused arguments

Value

a connection object to be used with `ff_*` functions

Examples

```
mfl_connect(season = 2020, league_id = 54040)
mfl_connect(season = 2019, league_id = 54040, rate_limit = FALSE)
```

---

mfl_getendpoint *GET any MFL endpoint*

Description

Create a GET request to any MFL export endpoint.

Usage

```
mfl_getendpoint(conn, endpoint, ...)
```

Arguments

`conn`	the list object created by `mfl_connect()`
`endpoint`	a string defining which endpoint to return from the API
`...`	Arguments which will be passed as "argumentname = argument" in an HTTP query parameter

Details

This function will read the connection object and automatically pass in the rate-limiting, league ID (L), authentication cookie, and/or API key (APIKEY) if configured in the connection object.

The endpoint names and HTTP parameters (i.e. argument names) are CASE SENSITIVE and should be passed in exactly as displayed on the MFL API reference page.

Check out the vignette for more details and example usage.

Value

A list object containing the query, response, and parsed content.

See Also

https://api.myfantasyleague.com/2020/api_info?STATE=details

`vignette("mfl_getendpoint")`

mfl_players *MFL players library*

Description

A cached table of MFL players. Will store in memory for each session! (via memoise in zzz.R)

Usage

```
mfl_players(conn = NULL)
```

Arguments

conn	optionally, pass in a conn object generated by ff_connect to receive league-specific custom players

Value

a dataframe containing all ~2000+ players in the MFL database

Examples

```
try({ # try only shown here because sometimes CRAN checks are weird
  player_list <- mfl_players()
  dplyr::sample_n(player_list, 5)
}) # end try
```

nflfastr_rosters *Import nflfastr roster data*

Description

Deprecated in favour of `nflreadr::load_rosters()`

Usage

```
nflfastr_rosters(...)
```

Arguments

...	deprecated

nflfastr_stat_mapping *Mappings for nflfastr to fantasy platform scoring*

Description

A small helper dataframe for connecting nflfastr to specific fantasy platform rules.

Usage

```
nflfastr_stat_mapping
```

Format

A data frame with ~85 rows and 3 variables:

nflfastr_event the column name of the statistic in the nflfastr_weekly dataset

platform specific platform that this mapping applies to

ff_event name of the statistic for that platform

nflfastr_weekly *Import latest nflfastr weekly stats*

Description

Deprecated in favour of `nflreadr::load_player_stats()`

Usage

```
nflfastr_weekly(...)
```

Arguments

...	deprecated

`sleeper_connect` *Connect to Sleeper League*

Description

This function creates a connection object which stores parameters and a user ID if available.

Usage

```
sleeper_connect(
  season = NULL,
  league_id = NULL,
  user_name = NULL,
  user_agent = NULL,
  rate_limit = TRUE,
  rate_limit_number = NULL,
  rate_limit_seconds = NULL,
  ...
)
```

Arguments

`season`	Season to access on Sleeper - if missing, will guess based on system date (current year if March or later, otherwise previous year)
`league_id`	League ID (currently assuming one league at a time)
`user_name`	Sleeper user_name - optional - attempts to get user's user ID
`user_agent`	User agent to self-identify (optional)
`rate_limit`	TRUE by default - turn off rate limiting with FALSE
`rate_limit_number`	number of calls per `rate_limit_seconds`, suggested is under 1000 calls per 60 seconds
`rate_limit_seconds`	number of seconds as denominator for rate_limit
`...`	other arguments (for other methods)

Value

a list that stores Sleeper connection objects

sleeper_draft *Get Sleeper Draft*

Description

This function retrieves drafts by sleeper's draft ID. This better supports mock drafts.

Usage

```
sleeper_draft(draft_id)
```

Arguments

draft_id	draft ID as found in URL e.g. "https://sleeper.com/draft/nfl/draft_id"

Value

draft dataframe

sleeper_getendpoint *GET any Sleeper endpoint*

Description

The endpoint names and HTTP parameters (i.e. argument names) are CASE SENSITIVE and should be passed in exactly as displayed on the Sleeper API reference page.

Usage

```
sleeper_getendpoint(endpoint, ...)
```

Arguments

endpoint	a string defining which endpoint to return from the API
...	Arguments which will be passed as "argumentname = argument" in an HTTP query parameter

Details

Check out the vignette for more details and example usage.

Value

A list object containing the query, response, and parsed content.

See Also

https://docs.sleeper.com

vignette("sleeper_getendpoint")

`sleeper_players` *Sleeper players library*

Description

A cached table of Sleeper NFL players. Will store in memory for each session! (via memoise in zzz.R)

Usage

```
sleeper_players()
```

Value

a dataframe containing all ~7000+ players in the Sleeper database

Examples

```
try({ # try only shown here because sometimes CRAN checks are weird
  x <- sleeper_players()
  dplyr::sample_n(x, 5)
}) # end try
```

`sleeper_userleagues` *Sleeper - Get User Leagues*

Description

This function returns the leagues that a specific user is in. This variant can be used without first creating a connection object.

Usage

```
sleeper_userleagues(user_name, season = NULL)
```

Arguments

`user_name`	the username to look up
`season`	the season to return leagues from - defaults to current year based on heuristics

Value

a dataframe of leagues for the specified user

See Also

`ff_userleagues()`

`%>%` *Pipe operator*

Description

See `magrittr::%>%` for details.

Index