Package 'dfeR'

September 23, 2026

Type Package

Title Common Department for Education Analysis Tasks

Version 2.0.0

Description Provides preferred methods for common analytical tasks undertaken across the Department for Education in England, including number formatting, project templates and curated reference data.

License GPL (>= 3)

URL https://dfe-analytical-services.github.io/dfeR/, https://github.com/dfe-analytical-services/dfeR

BugReports https://github.com/dfe-analytical-services/dfeR/issues

Depends R (>= 4.1.0)

Imports arrow, cli, DBI, dplyr, emoji, httr2, jsonlite, lifecycle, renv, rlang, rstudio.prefs, stringr, tidyselect, usethis, utils, withr

Suggests knitr, rmarkdown, spelling, testthat (>= 3.1.7), mockery, ggplot2, purrr, scales

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Language en-GB

LazyData true

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Cam Race [aut, cre],
Department for Education, England [cph],
Laura Selby [aut],
Adam Robinson [aut],
Jen Machin [ctb],
Jake Tufts [ctb],
Rich Bielby [ctb] (ORCID: <https://orcid.org/0000-0001-9070-9969>),

Menna Zayed [ctb],
Lauren Snaathorst [ctb],
Lara Garbett [ctb],
Paula Rocha [ctb]

Maintainer Cam Race <cameron.race@education.gov.uk>

Repository CRAN

Date/Publication 2026-09-23 17:50:02 UTC

Contents

`air_install` *Air Install*

Description

checks for air installation status and installs it if required (or if the installed version is older than the minimum supported version), updating the global settings if selected

Usage

```
air_install(update_rstudio_settings = FALSE, verbose = TRUE, force = FALSE)
```

Arguments

`update_rstudio_settings`	auto update RStudio settings
`verbose`	Run in verbose mode
`force`	force (re)installation of Air, even if an up to date version is already installed

Value

No return value, called for side effects

Examples

```
## Not run:
air_install()

## End(Not run)
```

air_style *Air - style code in scripts*

Description

styles the whole project or single file using air

Usage

```
air_style(target = ".", verbose = FALSE)
```

Arguments

target	single file target for formatting
verbose	Run in verbose mode

Value

No return value, called for side effects

Examples

```
## Not run:
air_style()

## End(Not run)
```

check_databricks_odbc *Check Databricks ODBC connection variables*

Description

Checks if the required environment variables for connecting to Databricks are set, and if the odbc package version is sufficient.

Usage

```
check_databricks_odbc()
```

Details

Prints instructions for fixing common problems to the console.

Value

TRUE if the connection is set up correctly, FALSE otherwise.

See Also

Other databricks: `write_df_to_delta()`

Examples

```
check_databricks_odbc()
```

`check_gitconfig_location`

Check the location of the global .gitconfig file

Description

Reports the location Git is using for its global config file. R, RStudio, and the system Git can disagree on this location (e.g. when `HOME` is redirected into OneDrive). The function flags non-standard locations and prints the resolved path so users can sanity-check it against the file they think they are editing.

Usage

```
check_gitconfig_location()
```

Details

Each check returns a list including a `status` field, one of `"pass"`, `"fail"`, `"fixed"` or `"info"`, depending on the outcome of the check.

Value

List object containing `gitconfig_path` (or `NA` if none found) plus a `status` field.

Examples

```
check_gitconfig_location()
```

check_github_pat *Check GITHUB_PAT setting*

Description

If the GITHUB_PAT system variable is set, it can cause issues with R installing packages from GitHub (usually with an error of "ERROR [curl: (22) The requested URL returned error: 401]"). This function checks whether the variable is set and (with clean = TRUE) clears it for the current R session.

The function masks the PAT value both in console output and in the returned list (only its length and last four characters are shown). The raw value is never returned, so it is safe to share the result of diagnostic_test().

Usage

```
check_github_pat(clean = FALSE)
```

Arguments

clean	If TRUE, attempt to clean detected issues. Default FALSE.

Details

Each check returns a list including a status field, one of "pass", "fail", "fixed" or "info", depending on the outcome of the check.

Value

List object containing GITHUB_PAT (masked: empty string when unset, otherwise "..." followed by the last four characters) plus a status field.

Examples

```
check_github_pat()
```

check_git_sslverify *Check Git sslverify setting*

Description

Checks the values of the SSL-verify settings in the global Git config. If they're set to false, the function flips them back to TRUE when called with clean = TRUE. By default it checks http.sslVerify and https.sslVerify.

Usage

```
check_git_sslverify(
  ssl_verify_vars = c("http.sslverify", "https.sslverify"),
  clean = FALSE
)
```

Arguments

ssl_verify_vars	Vector of variables to check for in the Git config.
clean	If TRUE, attempt to clean detected issues. Default FALSE.

Details

Each check returns a list including a `status` field, one of `"pass"`, `"fail"`, `"fixed"` or `"info"`, depending on the outcome of the check.

Value

List of sslverify settings plus a `status` field.

Examples

```
check_git_sslverify()
```

---

check_proxy_settings *Check proxy settings*

Description

Prior to the pandemic, analysts in the DfE would need to add proxy settings to either their Git configuration or system environment variables. These settings now prevent Git from connecting to remote archives on GitHub and Azure DevOps, so this function identifies them and (with `clean = TRUE`) removes them.

Both the Git configuration (keys `http.proxy`, `https.proxy` by default) and the system environment variables (`http_proxy`, `https_proxy`, `no_proxy` by default) are checked.

On Windows, `clean = TRUE` clears the matching environment variables permanently from your user environment (via `setx`) as well as from the current R session, so they do not come back the next time you start R. Windows only applies the change to new processes, so close and reopen RStudio afterwards.

Usage

```
check_proxy_settings(
  proxy_setting_names = c("http.proxy", "https.proxy"),
  proxy_env_names = c("http_proxy", "https_proxy", "no_proxy"),
  clean = FALSE
)
```

Arguments

proxy_setting_names	Vector of Git-config keys to check for. Default: c("http.proxy", "https.proxy")
proxy_env_names	Vector of system environment variable names to check for. Default: c("http_proxy", "https_proxy", "no_proxy")
clean	If TRUE, attempt to clean detected issues. Default FALSE.

Details

Each check returns a list including a status field, one of "pass", "fail", "fixed" or "info", depending on the outcome of the check.

Value

A list with three slots: git (named list of detected Git-config proxy entries, or NULL), system (named list of detected environment-variable proxy entries, or NULL) and a status field.

Examples

```
check_proxy_settings()
```

---

check_renv_dl_file_method

Check RENV_DOWNLOAD_FILE_METHOD system variable

Description

The legacy proxy.R script used setx RENV_DOWNLOAD_FILE_METHOD wininet to route renv downloads through Windows wininet. This breaks renv outside the DfE network and is no longer needed. This function checks the RENV_DOWNLOAD_FILE_METHOD system environment variable and (with clean = TRUE) unsets it for the current R session.

On Windows, clean = TRUE also clears the variable permanently from your user environment (via setx), so it does not come back the next time you start R. Windows only applies the change to new processes, so close and reopen RStudio afterwards.

Usage

```
check_renv_dl_file_method(clean = FALSE)
```

Arguments

clean	If TRUE, attempt to clean detected issues. Default FALSE.

Details

Each check returns a list including a status field, one of "pass", "fail", "fixed" or "info", depending on the outcome of the check.

Value

List object containing RENV_DOWNLOAD_FILE_METHOD plus a status field.

Examples

```
check_renv_dl_file_method()
```

---

check_renv_download_method

Check renv download method

Description

The renv package can retrieve packages either using curl or wininet, but wininet doesn't work from within the DfE network. This function checks for the parameter controlling which of these is used (RENV_DOWNLOAD_METHOD) in the user's .Renviron and sets it to curl when called with clean = TRUE.

Usage

```
check_renv_download_method(renviron_file = "~/.Renviron", clean = FALSE)
```

Arguments

renviron_file	Location of .Renviron file. Default: ~/.Renviron
clean	If TRUE, attempt to clean detected issues. Default FALSE.

Details

Each check returns a list including a status field, one of "pass", "fail", "fixed" or "info", depending on the outcome of the check.

Value

List object containing RENV_DOWNLOAD_METHOD (with surrounding whitespace and any wrapping quotes stripped, or NA if the variable is not set) plus a status field.

Examples

```
check_renv_download_method()
```

check_renv_rprof_location *Check the location of .Renviron and .Rprofile*

Description

Reports where R will read the user-level `.Renviron` and `.Rprofile` files from, listing every location R consults in precedence order: the `R_ENVIRON_USER` / `R_PROFILE_USER` override, the current working directory, and the home directory. Each file that exists is listed, and the one R actually uses is tagged, so you can spot when more than one copy exists (for example a project-level file shadowing the one in your home directory).

This check is purely informational and never reports a failure.

Usage

```
check_renv_rprof_location()
```

Details

Each check returns a list including a `status` field, one of `"pass"`, `"fail"`, `"fixed"` or `"info"`, depending on the outcome of the check.

Value

List object with a `renviron` and an `rprofile` entry (each a list of the `used` path and the `found` paths that exist), plus a `status` field which is always `"info"`.

Examples

```
check_renv_rprof_location()
```

check_rtools *Check for an RTools / make toolchain*

Description

Some R packages require compilation, which on Windows means RTools must be installed and on `PATH`. This function checks whether `make` is available via `Sys.which("make")`. It does not attempt to install RTools.

Usage

```
check_rtools()
```

Details

Each check returns a list including a `status` field, one of `"pass"`, `"fail"`, `"fixed"` or `"info"`, depending on the outcome of the check.

Value

List object containing `rtools_make_path` plus a `status` field. Missing RTools is reported as `"info"` rather than a failure because many users will not need to compile packages from source.

Examples

```
check_rtools()
```

---

comma_sep *Comma separate*

Description

Adds separating commas to big numbers. If a value is not numeric it will return the value unchanged and as a string.

Usage

```
comma_sep(number, nsmall = 0L)
```

Arguments

`number`	number to be comma separated
`nsmall`	minimum number of digits to the right of the decimal point

Value

string

Examples

```
comma_sep(100)
comma_sep(1000)
comma_sep(3567000)
```

countries *Lookup for valid country names and codes*

Description

A lookup of ONS geography country names and codes, as well as some custom DfE names and codes. This is used as the definitive list for the screening of open data before it is published by the DfE.

Usage

```
countries
```

Format

`countries`:

A data frame with 10 rows and 2 columns:

country_name Country name

country_code Country code

Source

curated by explore.statistics@education.gov.uk, ONS codes sourced from https://geoportal.statistics.gov.uk/search?q=countri

create_project *Creates a pre-populated project for DfE R*

Description

Creates a pre-populated project for DfE R

Usage

```
create_project(
  path,
  init_renv = TRUE,
  include_structure_for_pkg = FALSE,
  create_publication_proj = FALSE,
  include_github_gitignore,
  ...
)
```

Arguments

path	Path of the new project
init_renv	Boolean; initiate renv in the project. Default is set to true.
include_structure_for_pkg	Boolean; Additional folder structure for package development. Default is set to false.
create_publication_proj	Boolean; Should the folder structure be for a publication project. Default is set to false.
include_github_gitignore	Boolean; Should a strict .gitignore file for GitHub be created.
...	Additional parameters, currently not used

Details

This function creates a new project with a custom folder structure. It sets up the R/ folder and template function scripts, initializes {testthat} and adds tests for the function scripts, builds the core project structure, creates a .gitignore file, creates a readme, and optionally initializes {renv}.

Value

No return values, the project and its contents are created

Examples

```
## Not run:

# Call the function to create a new project
dfeR::create_project(
  path = "C:/path/to/your/new/project",
  init_renv = TRUE,
  include_structure_for_pkg = FALSE,
  create_publication_proj = FALSE,
  include_github_gitignore = TRUE
)

## End(Not run)
```

---

diagnostic_test *Diagnostic testing*

Description

Run a set of diagnostic tests to check for common issues found when setting up R on a DfE system. The function runs each check_*() helper in turn and returns a combined list of detected settings. Problems can optionally be fixed by passing clean = TRUE.

The checks include:

- Proxy settings in the Git configuration and system environment (check_proxy_settings())
- Git SSL-verify setting (check_git_sslverify())
- The location of the global .gitconfig file (check_gitconfig_location())
- The GITHUB_PAT system variable (check_github_pat())
- renv download method in .Renviron (check_renv_download_method())
- The RENV_DOWNLOAD_FILE_METHOD system variable (check_renv_dl_file_method())
- Presence of an RTools/make toolchain (check_rtools())
- The location of .Renviron and .Rprofile (check_renv_rprof_location())

Usage

```
diagnostic_test(clean = FALSE, full = FALSE)
```

Arguments

clean	If TRUE, attempt to clean detected issues. Default FALSE.
full	If TRUE, append a session-dump section after the per-check output containing renv::diagnostics(), Sys.getenv() and options(). Useful for sharing a full diagnostic with support. Defaults to FALSE. Environment variables whose names look sensitive (containing e.g. TOKEN, SECRET, KEY, PAT, PASSWORD, CRED) are masked, but the options() and renv::diagnostics() sections are printed as-is, so review the output before sharing it in public channels.

Details

Each check returns a list including a status field, one of "pass", "fail", "fixed" or "info", depending on the outcome of the check.

Value

Invisibly, a named list keyed by check (proxy, sslverify, gitconfig, github_pat, renv_download, renv_download_file, rtools, renviron_rprofile). Each entry is the result list returned by the corresponding check_* helper, plus a status field.

Examples

```
diagnostic_test()
```

fetch *Fetch Westminster parliamentary constituencies*

Description

Fetch a data frame of all Westminster Parliamentary Constituencies for a given year and country based on the dfeR::geo_hierarchy file.

Usage

```
fetch_pcons(year = "All", countries = "All")
```

Arguments

year	year to filter the locations to, default is "All", options of 2017, 2019, 2020, 2021, 2022, 2023, 2024, 2025
countries	vector of desired countries to filter the locations to, default is "All", or can be a vector with options of "England", "Scotland", "Wales" or "Northern Ireland"

Value

data frame of unique location names and codes

Examples

```
# Using head() to show only top 5 rows for examples
head(fetch_wards())

head(fetch_pcons())

head(fetch_pcons(2023))

head(fetch_pcons(countries = "Scotland"))

head(fetch_pcons(year = 2023, countries = c("England", "Wales")))

head(fetch_mayoral())

head(fetch_lsips())

head(fetch_lsips(2025))

fetch_lads(2024, "Wales")

fetch_las(2022, "Northern Ireland")

# The following have no specific years available and return all values
fetch_regions()
fetch_countries()
```

`fetch_countries` *Fetch countries*

Description

Fetch a data frame of all countries based on the dfeR::countries file.

Usage

```
fetch_countries()
```

Value

data frame of unique location names and codes

See Also

Other fetch_locations: `fetch_lads()`, `fetch_las()`, `fetch_lsips()`, `fetch_mayoral()`, `fetch_regions()`, `fetch_wards()`

Examples

```
# Using head() to show only top 5 rows for examples
head(fetch_wards())

head(fetch_pcons())

head(fetch_pcons(2023))

head(fetch_pcons(countries = "Scotland"))

head(fetch_pcons(year = 2023, countries = c("England", "Wales")))

head(fetch_mayoral())

head(fetch_lsips())

head(fetch_lsips(2025))

fetch_lads(2024, "Wales")

fetch_las(2022, "Northern Ireland")

# The following have no specific years available and return all values
fetch_regions()
fetch_countries()
```

`fetch_lads` *Fetch local authority districts*

Description

Fetch a data frame of all local authority districts for a given year and country based on the dfeR::geo_hierarchy file.

Usage

```
fetch_lads(year = "All", countries = "All")
```

Arguments

`year`	year to filter the locations to, default is "All", options of 2017, 2019, 2020, 2021, 2022, 2023, 2024, 2025
`countries`	vector of desired countries to filter the locations to, default is "All", or can be a vector with options of "England", "Scotland", "Wales" or "Northern Ireland"

Value

data frame of unique location names and codes

See Also

Other fetch_locations: `fetch_countries()`, `fetch_las()`, `fetch_lsips()`, `fetch_mayoral()`, `fetch_regions()`, `fetch_wards()`

Examples

```
# Using head() to show only top 5 rows for examples
head(fetch_wards())

head(fetch_pcons())

head(fetch_pcons(2023))

head(fetch_pcons(countries = "Scotland"))

head(fetch_pcons(year = 2023, countries = c("England", "Wales")))

head(fetch_mayoral())

head(fetch_lsips())

head(fetch_lsips(2025))

fetch_lads(2024, "Wales")
```

```
fetch_las(2022, "Northern Ireland")

# The following have no specific years available and return all values
fetch_regions()
fetch_countries()
```

`fetch_las` *Fetch local authorities*

Description

Fetch a data frame of all local authorities for a given year and country based on the dfeR::geo_hierarchy file.

Usage

```
fetch_las(year = "All", countries = "All")
```

Arguments

`year`	year to filter the locations to, default is "All", options of 2017, 2019, 2020, 2021, 2022, 2023, 2024, 2025
`countries`	vector of desired countries to filter the locations to, default is "All", or can be a vector with options of "England", "Scotland", "Wales" or "Northern Ireland"

Value

data frame of unique location names and codes

See Also

Other fetch_locations: `fetch_countries()`, `fetch_lads()`, `fetch_lsips()`, `fetch_mayoral()`, `fetch_regions()`, `fetch_wards()`

Examples

```
# Using head() to show only top 5 rows for examples
head(fetch_wards())

head(fetch_pcons())

head(fetch_pcons(2023))

head(fetch_pcons(countries = "Scotland"))

head(fetch_pcons(year = 2023, countries = c("England", "Wales")))

head(fetch_mayoral())
```

```
head(fetch_lsips())

head(fetch_lsips(2025))

fetch_lads(2024, "Wales")

fetch_las(2022, "Northern Ireland")

# The following have no specific years available and return all values
fetch_regions()
fetch_countries()
```

`fetch_lsips` *Fetch Local Skills Improvement Plan (LSIP) areas lookup*

Description

Fetch a data frame of Local Skills Improvement Plan (LSIP) areas for a given year based on `dfeR::lsip_lad`.

Usage

```
fetch_lsips(year = "All")
```

Arguments

`year`	year to filter the locations to, default is "All", options of 2023 or 2025. ONS did not publish a 2024 lookup, so 2024 is not a valid option

Value

data frame of LSIP for a given year.

See Also

Other fetch_locations: `fetch_countries()`, `fetch_lads()`, `fetch_las()`, `fetch_mayoral()`, `fetch_regions()`, `fetch_wards()`

Examples

```
# Using head() to show only top 5 rows for examples
head(fetch_wards())

head(fetch_pcons())

head(fetch_pcons(2023))

head(fetch_pcons(countries = "Scotland"))
```

```
head(fetch_pcons(year = 2023, countries = c("England", "Wales")))

head(fetch_mayoral())

head(fetch_lsips())

head(fetch_lsips(2025))

fetch_lads(2024, "Wales")

fetch_las(2022, "Northern Ireland")

# The following have no specific years available and return all values
fetch_regions()
fetch_countries()
```

`fetch_mayoral` *Fetch mayoral combined authorities*

Description

Fetch a data frame of all mayoral combined authorities for a given year and country based on the dfeR::geo_hierarchy file.

Usage

```
fetch_mayoral(year = "All")
```

Arguments

`year`	year to filter the locations to, default is "All", options of 2017, 2019, 2020, 2021, 2022, 2023, 2024, 2025

Details

Note that mayoral combined authorities only exist for England.

Mayoral combined authorities are also known as English Devolved Areas, as we add in the Greater London Authority to the combined authority lookup published by ONS.

Value

data frame of unique location names and codes

See Also

Other fetch_locations: `fetch_countries()`, `fetch_lads()`, `fetch_las()`, `fetch_lsips()`, `fetch_regions()`, `fetch_wards()`

Examples

```
# Using head() to show only top 5 rows for examples
head(fetch_wards())

head(fetch_pcons())

head(fetch_pcons(2023))

head(fetch_pcons(countries = "Scotland"))

head(fetch_pcons(year = 2023, countries = c("England", "Wales")))

head(fetch_mayoral())

head(fetch_lsips())

head(fetch_lsips(2025))

fetch_lads(2024, "Wales")

fetch_las(2022, "Northern Ireland")

# The following have no specific years available and return all values
fetch_regions()
fetch_countries()
```

`fetch_mp_lookup` *Fetch Westminster constituency to MP lookup*

Description

Fetch a data frame with one row per Westminster parliamentary constituency and the MP currently sitting for it. Alongside the constituency name and code, it gives the MP's name, Parliament member ID, party, email address and 2024 general election result summary, plus the local authority districts, local authorities, mayoral authorities, region and country that each constituency maps to.

Usage

```
fetch_mp_lookup(verbose = TRUE)
```

Arguments

`verbose`	TRUE or FALSE boolean. TRUE by default. FALSE will turn off the messages to the console that update on what the function is doing

Details

The lookup is maintained in the mp-lookup repository. It updates automatically as new results are published, and is read directly from its GitHub-hosted CSV so that users don't need to know or find the URL themselves.

Note that this is a lookup of sitting MPs, not a set of candidate-level results, so unsuccessful candidates are not included.

Value

data frame with one row per Westminster parliamentary constituency and its sitting MP

Examples

```
head(fetch_mp_lookup())
```

---

fetch_regions *Fetch regions*

Description

Fetch a data frame of all regions based on the dfeR::regions file.

Usage

```
fetch_regions()
```

Value

data frame of unique location names and codes

See Also

Other fetch_locations: `fetch_countries()`, `fetch_lads()`, `fetch_las()`, `fetch_lsips()`, `fetch_mayoral()`, `fetch_wards()`

Examples

```
# Using head() to show only top 5 rows for examples
head(fetch_wards())

head(fetch_pcons())

head(fetch_pcons(2023))

head(fetch_pcons(countries = "Scotland"))

head(fetch_pcons(year = 2023, countries = c("England", "Wales")))
```

```
head(fetch_mayoral())

head(fetch_lsips())

head(fetch_lsips(2025))

fetch_lads(2024, "Wales")

fetch_las(2022, "Northern Ireland")

# The following have no specific years available and return all values
fetch_regions()
fetch_countries()
```

`fetch_wards` *Fetch wards*

Description

Fetch a data frame of all wards for a given year and country based on the dfeR::geo_hierarchy file.

Usage

```
fetch_wards(year = "All", countries = "All")
```

Arguments

`year`	year to filter the locations to, default is "All", options of 2017, 2019, 2020, 2021, 2022, 2023, 2024, 2025
`countries`	vector of desired countries to filter the locations to, default is "All", or can be a vector with options of "England", "Scotland", "Wales" or "Northern Ireland"

Value

data frame of unique location names and codes

See Also

Other fetch_locations: `fetch_countries()`, `fetch_lads()`, `fetch_las()`, `fetch_lsips()`, `fetch_mayoral()`, `fetch_regions()`

Examples

```
# Using head() to show only top 5 rows for examples
head(fetch_wards())

head(fetch_pcons())
```

```
head(fetch_pcons(2023))

head(fetch_pcons(countries = "Scotland"))

head(fetch_pcons(year = 2023, countries = c("England", "Wales")))

head(fetch_mayoral())

head(fetch_lsips())

head(fetch_lsips(2025))

fetch_lads(2024, "Wales")

fetch_las(2022, "Northern Ireland")

# The following have no specific years available and return all values
fetch_regions()
fetch_countries()
```

`format_ay` *Format academic year*

Description

This function formats academic year variables for reporting purposes. It will convert an academic year input from 201516 format to 2015/16 format.

Usage

```
format_ay(year)
```

Arguments

`year`	Academic year, a single value or a vector

Details

It accepts both numerical and character arguments.

Value

Character vector of formatted academic years

See Also

Other format: `format_ay_reverse()`, `format_fy()`, `format_fy_reverse()`

Examples

```
format_ay(201617)
format_ay("201617")
format_ay(c(201617, 201718))
```

`format_ay_reverse` *Undo academic year formatting*

Description

This function converts academic year variables back into 201617 format.

Usage

```
format_ay_reverse(year)
```

Arguments

`year`	Academic year, a single value or a vector

Details

It accepts character arguments.

Value

Character vector of unformatted 6 digit years

See Also

Other format: `format_ay()`, `format_fy()`, `format_fy_reverse()`

Examples

```
format_ay_reverse("2016/17")
format_ay_reverse(c("2016/17", "2017/18"))
```

format_fy *Format financial year*

Description

This function formats financial year variables for reporting purposes. It will convert an year input from 201516 format to 2015-16 format.

Usage

```
format_fy(year)
```

Arguments

year	Financial year, a single value or a vector

Details

It accepts both numerical and character arguments.

Value

Character vector of formatted financial years

See Also

Other format: format_ay(), format_ay_reverse(), format_fy_reverse()

Examples

```
format_fy(201617)
format_fy("201617")
format_fy(c(201617, 201718))
```

format_fy_reverse *Undo financial year formatting*

Description

This function converts financial year variables back into 201617 format.

Usage

```
format_fy_reverse(year)
```

Arguments

year	Financial year, a single value or a vector

Details

It accepts character arguments.

Value

Character vector of unformatted 6 digit years

See Also

Other format: format_ay(), format_ay_reverse(), format_fy()

Examples

```
format_fy_reverse("2016-17")
format_fy_reverse(c("2016-17", "2017-18"))
```

---

geog_time_identifiers *Potential names for geography and time columns*

Description

Potential names for geography and time columns in line with the ones used for the explore education statistics data screener.

Usage

```
geog_time_identifiers
```

Format

geog_time_identifiers:
A character vector with 38 potential column names in snake case format.

Source

curated by explore.statistics@education.gov.uk. Guidance on time and geography data.

`geo_hierarchy` *Geography hierarchy lookup*

Description

A lookup showing the hierarchy of ward to Westminster parliamentary constituency to local authority district to local authority to combined mayoral authority to region to country for years 2017, 2019, 2020, 2021, 2022, 2023, 2024 and 2025.

Usage

```
geo_hierarchy
```

Format

`geo_hierarchy`**:**

A data frame with 26,057 rows and 17 columns:

first_available_year_included First year in the lookups that we see this location

most_recent_year_included Last year in the lookups that we see this location

ward_name Ward name

pcon_name Parliamentary constituency name

lad_name Local authority district name

la_name Local authority name

english_devolved_area_name Mayoral authority name

region_name Region name

country_code Country name

ward_code 9 digit ward code

pcon_code 9 digit westminster constituency code

lad_code 9 digit local authority district code

old_la_code old 3 digit local authority code

new_la_code 9 digit local authority code

english_devolved_area_code 9 digit combined authority code

region_code 9 digit region code

country_code 9 digit country code

Details

Note that combined mayoral authorities only exist in England, and we use `english_devolved_area_name` and `english_devolved_area_code` to refer to mayoral authorities in line with the standards set in DfE official statistics. Greater London Authority is not included in the ONS combined authority lookup, we have added that in for all local authority districts with codes that start with `E090....`

Changes we've made to the original lookup:

1. The original lookup from ONS uses the Upper Tier Local Authority, we then update this so that where there is a metropolitan local authority we use the local authority district as the local authority to match how DfE publish data for local authorities.
2. We have noticed that in the 2017 version, the Glasgow East constituency had a code of S1400030 instead of the usual S14000030, we've assumed this was an error and have change this in our data so that Glasgow East is S14000030 in 2017.
3. We have joined on regions using the Ward to LAD to County to Region file.
4. We have joined on countries based on the E / N / S / W at the start of codes.
5. Scotland had no published regions in 2017, so given the rest of the years have Scotland as the region, we've forced that in for 2017 too to complete the data set.
6. We've added the Greater London Authority as the overarching mayoral authority (English devolved area) for all Local authority districts with codes that start `E090...`.

Source

https://geoportal.statistics.gov.uk/search?tags=lup_wd_pcon_lad_utla https://geoportal.statistics.gov.uk/search?q=lup_wd_la
https://geoportal.statistics.gov.uk/search?tags=LUP_LAD_CAUTH https://get-information-schools.service.gov.uk/Guidance
and https://tinyurl.com/EESScreenerLAs

`get_clean_sql` *Get a cleaned SQL script into R*

Description

This function cleans a SQL script, ready for using within R in the DfE.

Usage

```
get_clean_sql(filepath, additional_settings = FALSE)
```

Arguments

`filepath`	path to a SQL script
`additional_settings`	TRUE or FALSE boolean for the addition of settings at the start of the SQL script

Value

Cleaned string containing SQL query

Examples

```
# This assumes you have already set up a database connection
# and that the filepath for the function exists
# For more details see the vignette on connecting to SQL

# Pull a cleaned version of the SQL file into R
if (file.exists("your_script.sql")) {
  sql_query <- get_clean_sql("your_script.sql")
}
```

`get_ons_api_data` *Fetch ONS Open Geography API data*

Description

Helper function that takes a data set id and parameters to query and parse data from the ONS Open Geography API. Technically uses a POST request rather than a GET request.

Usage

```
get_ons_api_data(
  data_id,
  query_params = list(where = "1=1", outFields = "*", outSR = "4326", f = "json"),
  batch_size = 200,
  verbose = TRUE
)
```

Arguments

`data_id`	the id of the data set to query, can be found from the Open Geography Portal
`query_params`	query parameters to pass into the API, see the ESRI documentation for more information on query parameters - ESRI Query (Feature Service/Layer)
`batch_size`	the number of rows per query. This is 200 by default, if you hit errors then try lowering this. The API has a limit of 1000 to 2000 rows per query, and in truth, the actual limit for our method is lower as every ObjectId queried is pasted into the request, so larger batches, and especially Ids that go into the 1,000s or 10,000s, increase the size of each request and risk hitting the limit.
`verbose`	TRUE or FALSE boolean. TRUE by default. FALSE will turn off the messages to the console that update on what the function is doing

Details

It does a pre-query to understand the ObjectIds for the query you want, and then does a query to retrieve those Ids directly in batches before then stacking the whole thing back together to work around the row limits for a single query.

On the Open Geography Portal, find the data set you're interested in and then use the query explorer to find the information for the query.

This function has been mostly developed for ease of use for dfeR maintainers if you're interested in getting data from the Open Geography Portal more widely you should also look at the boundr package.

Value

parsed data.frame of geographic names and codes

Examples

```
# Fetch everything from a data set
dfeR::get_ons_api_data(data_id = "LAD23_RGN23_EN_LU")

# Specify the columns you want
dfeR::get_ons_api_data(
  "RGN_DEC_2023_EN_NC",
  query_params = list(
    where = "1=1",
    outFields = "RGN23CD,RGN23NM",
    outSR = 4326,
    f = "json"
  )
)
```

`lsip_lad` *Local Skills Improvement Plan (LSIP) areas to Local Authority District (LAD) Lookup*

Description

A lookup table mapping Local Skills Improvement Plan (LSIP) areas to Local Authority Districts (LADs) in England. This dataset provides a mapping between LSIP areas and LADs as provided by the ONS Geography Portal.

Usage

```
lsip_lad
```

Format

`lsip_lad`**:**

A data frame with one row per LAD to LSIP pairing per lookup, currently 466 rows covering 298 LADs and 51 LSIP codes. As codes are reused for renamed areas, those 51 codes give 56 distinct code and name pairings, which is what `fetch_lsips()` returns. The columns are:

lsip_code 9-character code for the LSIP area

lsip_name Name of the Local Skills Improvement Plan (LSIP) area

lad_code 9-character code for the Local Authority District

lad_name Name of the Local Authority District

most_recent_year_included The most recent year in which this location appears in the lookup

first_available_year_included The first year in which this location appears in the lookup

Details

- Within a given year, each LAD is assigned to a single LSIP area. The LSIP a LAD belongs to can change between lookups though, so across the data set as a whole a LAD may appear against more than one LSIP area.
- Mappings may change over time and can be tracked using the `most_recent_year_included` and `first_available_year_included` columns.
- LSIP codes are not stable over time. A code can be kept but the area renamed (E69000001 was 'Brighton and Hove, East Sussex, West Sussex' in 2023 and 'Sussex and Brighton' in 2025), and a name can be kept but the code changed ('North East' was E69000023 in 2023 and E69000045 in 2025). Joining on `lsip_code` alone across years will silently mismatch, so use `lsip_code` and `lsip_name` together to identify an area within a lookup.
- ONS have published LSIP lookups for 2023 and 2025 only, there was no 2024 lookup.

Source

https://geoportal.statistics.gov.uk/search?q=lad%20lsip

---

ons_geog_shorthands *Lookup for ONS geography columns shorthands*

Description

A lookup of ONS geography shorthands and their respective column names in line with DfE open data standards.

Usage

```
ons_geog_shorthands
```

Format

`ons_geog_shorthands`**:**

A data frame with 10 rows and 3 columns:

ons_level_shorthands ONS shorthands used in their lookup files

name_column DfE names for geography name columns

code_column DfE names for geography code columns

Details

GOR (Government Office Region) was the predecessor to RGN.

Source

curated by explore.statistics@education.gov.uk

`pretty_filesize` *Pretty numbers into readable file size*

Description

Converts a raw file size from bytes to a more readable format.

Usage

```
pretty_filesize(filesize)
```

Arguments

`filesize`	file size in bytes

Details

Designed to be used in conjunction with the file.size() function in base R.

Presents in kilobytes, megabytes or gigabytes.

Shows as bytes until 1 KB, then kilobytes up to 1 MB, then megabytes until 1GB, then it will show as gigabytes for anything larger.

Rounds the end result to 2 decimal places.

Using base 10 (decimal), so 1024 bytes is 1,024 KB.

Value

string containing prettified file size

See Also

`comma_sep()` `round_five_up()`

Other prettying: `pretty_num()`, `pretty_num_table()`, `pretty_time()`, `pretty_time_taken()`

Examples

```
pretty_filesize(2)
pretty_filesize(549302)
pretty_filesize(9872948939)
pretty_filesize(1)
pretty_filesize(1000)
pretty_filesize(1000^2)
pretty_filesize(10^9)
```

pretty_num *Prettify big numbers into a readable format*

Description

Uses `as.numeric()` to force a numeric value and then formats prettily for easy presentation in console messages, reports, or dashboards.

This rounds to 0 decimal places by default, and adds in comma separators.

Expect that this will commonly be used for adding the pound symbol, the percentage symbol, or to have a +/- prefixed based on the value.

If applying over multiple or unpredictable values and you want to preserve a non-numeric symbol such as "x" or "c" for data not available, use the `ignore_na = TRUE` argument to return those values unaffected.

If you want to customise what NA values are returned as, use the `alt_na` argument.

This function silences the warning around NAs being introduced by coercion.

Usage

```
pretty_num(
  value,
  prefix = "",
  gbp = FALSE,
  suffix = "",
  dp = 0,
  ignore_na = FALSE,
  alt_na = FALSE,
  nsmall = NULL,
  dynamic_dp_value = NULL,
  abbreviate = TRUE
)
```

Arguments

`value`	value to be prettified
`prefix`	prefix for the value, if "+/-" then it will automatically assign + or - based on the value
`gbp`	whether to add the pound symbol or not, defaults to not
`suffix`	suffix for the value, e.g. "%"
`dp`	number of decimal places to round to, 0 by default.
`ignore_na`	whether to skip function for strings that can't be converted and return original value
`alt_na`	alternative value to return in place of NA, e.g. "x"

nsmall — minimum number of digits to the right of the decimal point. If NULL, the value of dp will be used. If the value of dp is less than 0, then nsmall will automatically be set to 0.

dynamic_dp_value — Integer. Default = NULL. Overrides the dp setting and dynamically adjusts decimal places based on value magnitude. For values >= 1 million or >= 1 billion, the function checks the scaled value (e.g., value / 1e6 or value / 1e9): if the scaled value is a whole number, it sets decimal places to 0; otherwise, it adds precision as specified here. This approach improves clarity without unnecessary formatting.

abbreviate — whether to abbreviate large numbers to nearest million (where 1e6 <= value < 1e9) or billion (where value >= 1e9).

Value

string featuring prettified value

See Also

comma_sep() round_five_up() as.numeric()

Other prettying: pretty_filesize(), pretty_num_table(), pretty_time(), pretty_time_taken()

Examples

```
# On individual values
pretty_num(5789, gbp = TRUE)
pretty_num(564, prefix = "+/-")
pretty_num(567812343223, gbp = TRUE, prefix = "+/-")
pretty_num(11^9, gbp = TRUE, dp = 3)
pretty_num(-11^8, gbp = TRUE, dp = -1)
pretty_num(43.3, dp = 1, nsmall = 2)
pretty_num("56.089", suffix = "%")
pretty_num("x")
pretty_num("x", ignore_na = TRUE)
pretty_num("nope", alt_na = "x")
pretty_num(7.8e9, abbreviate = FALSE)
# dynamic_dp_value enabled for a billion value not divisible by 10
pretty_num(3e9, dynamic_dp_value = 2)
# dynamic_dp_value enabled for a billion value divisible by 10
pretty_num(10e9, dynamic_dp_value = 2)
# dynamic_dp_value enabled for a million value not divisible by 10
pretty_num(3e6, dynamic_dp_value = 3)
# dynamic_dp_value enabled for a million value divisible by 10
pretty_num(10e6, dynamic_dp_value = 3)
# dynamic_dp_value enabled with GBP and suffix
pretty_num(1.5e9,
  gbp = TRUE, suffix = "%",
  dynamic_dp_value = 1
)
#' # Applied over an example vector
vector <- c(3998098008, -123421421, "c", "x")
```

```
pretty_num(vector)
pretty_num(vector, prefix = "+/-", gbp = TRUE)

# Return original values if NA
pretty_num(vector, ignore_na = TRUE)

# Return alternative value in place of NA
pretty_num(vector, alt_na = "z")
```

`pretty_num_table` *Format a data frame with* `dfeR::pretty_num()`.

Description

You can format number and character values in a data frame by passing arguments to `dfeR::pretty_num()`. Use parameters `include_columns` or `exclude_columns` to specify columns for formatting.

Usage

```
pretty_num_table(data, include_columns = NULL, exclude_columns = NULL, ...)
```

Arguments

`data`	A data frame containing the columns to be formatted.
`include_columns`	A character vector specifying which columns to format. If `NULL` (default), all columns will be considered for formatting.
`exclude_columns`	A character vector specifying columns to exclude from formatting. If `NULL` (default), no columns will be excluded. If both `include_columns` and `exclude_columns` are provided , `include_columns` takes precedence.
`...`	Additional arguments passed to `dfeR::pretty_num()` , such as `dp` (decimal places) for controlling the number of decimal points.

Details

The function first checks if any columns are specified for inclusion via `include_columns`. If none are provided, it checks if columns are specified for exclusion via `exclude_columns`. If neither is specified, all columns in the data frame are formatted.

Value

A data frame with columns formatted using `dfeR::pretty_num()`.

Warning

Any selected column that cannot be coerced to numeric - including character columns such as geography or school names - is returned as NA by default. Pass `ignore_na = TRUE` (via `...`, forwarded to `dfeR::pretty_num()`) to leave those values unchanged instead; this is the safe way to format a table that also contains non-numeric columns.

See Also

`pretty_num()`

Other prettying: `pretty_filesize()`, `pretty_num()`, `pretty_time()`, `pretty_time_taken()`

Examples

```
# Example data frame
df <- data.frame(
  a = c(1.234, 5.678, 9.1011),
  b = c(10.1112, 20.1314, 30.1516),
  c = c("A", "B", "C")
)

# Apply formatting to all columns - note column c is character, so it
# cannot be coerced to numeric and comes back as NA (see Warning section)
pretty_num_table(df, dp = 2)

# Apply formatting to all columns, leaving non-numeric columns like c
# unchanged instead of turning them into NA
pretty_num_table(df, dp = 2, ignore_na = TRUE)

# Apply formatting to only selected columns
pretty_num_table(df, include_columns = c("a"), dp = 2)

# Apply formatting to all columns except specified ones
pretty_num_table(df, exclude_columns = c("b"), dp = 2)

# Apply formatting to all columns except specified ones and
# provide alternative value for NAs
pretty_num_table(df, alt_na = "[z]", exclude_columns = c("b"), dp = 2)
```

---

pretty_time *Pretty time*

Description

Convert seconds into a human readable format

Usage

```
pretty_time(seconds)
```

Arguments

seconds	number of seconds to prettify

Details

Recognises when to present as:

- seconds
- minutes and seconds
- hours, minutes and seconds

It will show seconds until 119 seconds, then minutes until 119 minutes, then hours. It doesn't do days or higher yet, but could be adapted to do so if there's demand.

Value

string containing the 'pretty' time

See Also

comma_sep() round_five_up() as.POSIXct()

Other prettying: pretty_filesize(), pretty_num(), pretty_num_table(), pretty_time_taken()

Examples

```
pretty_time(1)
pretty_time(8)
pretty_time(888)
pretty_time(88888888)
pretty_time(c(60, 2, 88, 88888888))
```

---

pretty_time_taken *Calculate elapsed time between two points and present prettily*

Description

Converts a start and end value to a readable time format.

Usage

```
pretty_time_taken(start_time, end_time)
```

Arguments

start_time	start time readable by as.POSIXct
end_time	end time readable by as.POSIXct

Details

Designed to be used with Sys.time() when tracking start and end times.

Shows as seconds up until 119 seconds, then minutes until 119 minutes, then hours for anything larger.

Input start and end times must be convertible to POSIXct format.

Value

string containing prettified elapsed time

See Also

`comma_sep()` `round_five_up()` `as.POSIXct()`

Other prettying: `pretty_filesize()`, `pretty_num()`, `pretty_num_table()`, `pretty_time()`

Examples

```
pretty_time_taken(
  "2024-03-23 07:05:53 GMT",
  "2024-03-23 12:09:56 GMT"
)

# Track the start and end time of a process
start <- Sys.time()
Sys.sleep(0.1)
end <- Sys.time()

# Use this function to present it prettily
pretty_time_taken(start, end)
```

---

regions *Lookup for valid region names and codes*

Description

A lookup of ONS geography region names and codes for England. In their lookups Northern Ireland, Scotland and Wales are regions.

Usage

```
regions
```

Format

`regions`:
A data frame with 16 rows and 2 columns:

region_name Region name

region_code Region code

Details

Also included inner and outer London county split as DfE frequently publish those as regions, as well as some custom DfE names and codes. This is used as the definitive list for the screening of open data before it is published by the DfE.

Source

curated by explore.statistics@education.gov.uk, ONS codes sourced from https://geoportal.statistics.gov.uk/search?q=NAC_I

---

round_five_up *Round five up*

Description

Round any number to a specified number of places, with 5's being rounded up.

Usage

```
round_five_up(number, dp = 0)
```

Arguments

number	number to be rounded
dp	number of decimal places to round to, default is 0

Details

Rounds to 0 decimal places by default.

You can use a negative value for the decimal places. For example: -1 would round to the nearest 10 -2 would round to the nearest 100 and so on.

This is as an alternative to round in base R, which uses a bankers round. For more information see the round() documentation.

Value

Rounded number

Examples

```
# No dp set
round_five_up(2485.85)

# With dp set
round_five_up(2485.85, 2)
round_five_up(2485.85, 1)
round_five_up(2485.85, 0)
round_five_up(2485.85, -1)
round_five_up(2485.85, -2)
```

`toggle_message` *Controllable console messages*

Description

Quick expansion to the `message()` function aimed for use in functions for an easy addition of a global verbose TRUE / FALSE argument to toggle the messages on or off

Usage

```
toggle_message(..., verbose)
```

Arguments

`...`	any message you would normally pass into `message()`. See `message` for more details
`verbose`	logical, usually a variable passed from the function you are using this within

Value

No return value, called for side effects

Examples

```
# Usually used in a function
my_function <- function(count_fingers, verbose) {
  toggle_message("I have ", count_fingers, " fingers", verbose = verbose)
  fingers_thumbs <- count_fingers + 2
  toggle_message("I have ", fingers_thumbs, " digits", verbose = verbose)
}

my_function(5, verbose = FALSE)
my_function(5, verbose = TRUE)

# Can be used in isolation
toggle_message("I want the world to read this!", verbose = TRUE)
toggle_message("I ain't gonna show this message!", verbose = FALSE)

count_fingers <- 5
toggle_message("I have ", count_fingers, " fingers", verbose = TRUE)
```

`write_df_to_delta` *Write a Data Frame to Delta Lake with COPY INTO*

Description

This function writes a large R data frame or tibble (`df`) to a Delta Lake table (`target_table`) on Databricks using Databricks Volumes and the `COPY INTO` SQL command.

Usage

```
write_df_to_delta(
  df,
  target_table,
  db_conn,
  column_types_schema = NULL,
  volume_dir,
  copy_options = "'mergeSchema' = 'true'",
  overwrite_table = FALSE,
  chunk_size_bytes = 5 * 1024^3
)
```

Arguments

`df`	A `data.frame` or `tibble` containing the data to be written to Delta Lake.
`target_table`	A character string specifying the name of the Delta table. Can be unqualified (`"table"`), partially qualified (`"schema.table"`), or fully qualified (`"catalog.schema.table"`). If not fully qualified, the function resolves the catalog and/or schema from the active database connection.
`db_conn`	A valid DBI connection object to Databricks. This connection is used to interact with the Delta table.
`column_types_schema`	An optional Arrow Schema object (created via `arrow::schema(...)`) used to explicitly enforce precise data types during Parquet conversion. Defaults to `NULL`. If `NULL`, data types are inferred from the R data frame. Type mapping reference (Arrow schema field → Spark SQL type): • Arrow int8/int16 → TINYINT/SMALLINT • Arrow int32/int64 → INT/BIGINT • Arrow float/double → FLOAT/DOUBLE • Arrow string/large_string → STRING • Arrow bool → BOOLEAN • Arrow `date32[day]` → DATE • Arrow `timestamp['s' | 'ms' | 'us', ...]` → TIMESTAMP_NTZ (no timezone) • Arrow `timestamp['s' | 'ms' | 'us', tz = ...]` → TIMESTAMP (with timezone)

- Arrow decimal(P,S) → DECIMAL(P,S)

Important notes for schema use:

- Factors: If the R data frame contains a `factor` column, the corresponding Arrow schema field must be `utf8()` or `large_utf8()`. The categorical labels are automatically converted and mapped to a `STRING` type in the Delta table.
- Timestamp precision: Second (`s`), millisecond (`ms`), and microsecond (`us`) precisions are supported. Nanosecond (`ns`) precision may be incompatible with the current Databricks runtime environment.

`volume_dir` A character string specifying the path to the target Databricks Volume where the Parquet file will be uploaded.

`copy_options` A character string specifying options for the `COPY INTO` command, e.g., `'mergeSchema' = 'true'`. Defaults to `"'mergeSchema' = 'true'"`.

`overwrite_table`
Logical; if `TRUE`, deletes and recreates the Delta table before import. If `FALSE` and the table does not exist, the function will throw an error. Defaults to `FALSE`.

`chunk_size_bytes`
An integer specifying the size of each data chunk in bytes. This is used to split the data frame into smaller chunks for uploading. Defaults to 5GB.

Details

The function performs the following steps:

- Optionally overwrites the target table.
- Uploads the data as Parquet file(s) to a specified Databricks Volume.
- Executes a `COPY INTO` command to load the file(s) into Delta Lake.
- Deletes the temporary file(s) from the Volume after loading is complete.

Data is chunked into segments during upload to accommodate the Databricks REST API limit of 5 GB per single file upload.

Value

Invisibly returns the result of the `COPY INTO` execution.

Note

To use this function, users must ensure that they have appropriate Databricks permissions:

- **Catalog/schema**: `USE CATALOG` on the target catalog and `USE SCHEMA` on the target schema.
- **Table creation**: `CREATE TABLE` on the target schema (if `overwrite_table = TRUE`) or `MODIFY` and `SELECT` on the existing table.
- **Volume access**: `READ VOLUME` and `WRITE VOLUME` on the Databricks Volume used for staging (specified in `volume_dir`).

Moreover, this function requires valid .Renviron variables for authentication, specifically DATABRICKS_TOKEN and DATABRICKS_HOST.

DATABRICKS_HOST may be supplied with or without a scheme. A bare host has https:// prepended automatically, and http:// is upgraded to https://. Both adb-1234.cloud.databricks.com and https://adb-1234.cloud.databricks.com are accepted. A trailing slash is stripped if present.

See Also

Other databricks: check_databricks_odbc()

Examples

```
## Not run:
# Setup connection using environment variables
con <- DBI::dbConnect(odbc::databricks(),
  httpPath = Sys.getenv("DATABRICKS_SQL_PATH")
)

write_df_to_delta(
  df = my_data,
  target_table = "catalog.schema.my_table",
  db_conn = con,
  volume_dir = "/Volumes/catalog/schema",
  overwrite_table = TRUE
)

## End(Not run)
```

---

z_replace *Replaces* NA *values in tables*

Description

Replaces NA values in tables except for ones in time and geography columns that must be included in DfE official statistics. Guidance on our Open Data Standards.

Usage

```
z_replace(data, replacement_alt = NULL, exclude_columns = NULL)
```

Arguments

data	name of the table that you want to replace NA values in
replacement_alt	optional - if you want the NA replacement value to be different to "z"

exclude_columns
: optional - additional columns to exclude from NA replacement. Column names that match ones found in `dfeR::geog_time_identifiers` will always be excluded because any missing data for these columns need more explicit codes to explain why data is not available.

Details

Names of geography and time columns that are used in this function can be found in `dfeR::geog_time_identifiers`.

Value

table with "z" or an alternate replacement value instead of `NA` values for columns that are not for time or geography.

See Also

geog_time_identifiers

Examples

```
# Create a table for the example

df <- data.frame(
  time_period = c(2022, 2022, 2022),
  time_identifier = c("Calendar year", "Calendar year", "Calendar year"),
  geographic_level = c("National", "Regional", "Regional"),
  country_code = c("E92000001", "E92000001", "E92000001"),
  country_name = c("England", "England", "England"),
  region_code = c(NA, "E12000001", "E12000002"),
  region_name = c(NA, "North East", "North West"),
  mystery_count = c(42, 25, NA)
)

z_replace(df)

# Use a different replacement value
z_replace(df, replacement_alt = "c")
```

Index