Package: dfeR (via r-universe)

January 14, 2025

Type Package

Title Common DfE R tasks

Version 0.6.1.9000

Description This package contains R functions to allow DfE analysts to re-use code for common analytical tasks that are undertaken across the Department.

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 (>= 2.10)

Imports dplyr, emoji, httr, jsonlite, lifecycle, magrittr, renv, rlang, tidyselect, usethis, utils, withr

Suggests knitr, readxl, rmarkdown, spelling, stringr, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Language en-GB

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Config/pak/sysreqs git make libgit2-dev libicu-dev libssl-dev libx11-dev

Repository https://dfe-analytical-services.r-universe.dev

RemoteUrl https://github.com/dfe-analytical-services/dfer

RemoteRef HEAD

RemoteSha 6695f5d8be563fc3891f02161706c496ca44be8e

Contents

`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)
```

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::wd_pcon_lad_la_rgn_ctry 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
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")))

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 countries

Usage

```
fetch_countries()
```

Value

data frame of unique location names and codes

See Also

Other fetch_locations: `fetch_lads()`, `fetch_las()`, `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")))

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 local authority districts

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
`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_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")))

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 local authorities

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
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_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")))

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_regions *Fetch regions*

Description

Fetch regions

Usage

```
fetch_regions()
```

Value

data frame of unique location names and codes

See Also

Other fetch_locations: fetch_countries(), fetch_lads(), fetch_las(), 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")))

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 wards

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
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_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")))

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

Details

It accepts both numerical and character arguments.

Value

Character vector of formatted academic year

See Also

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

Examples

```
format_ay(201617)
format_ay("201617")
```

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

Details

It accepts character arguments.

Value

Unformatted 6 digit year as string

See Also

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

Examples

```
format_ay_reverse("2016/17")
```

---

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

Details

It accepts both numerical and character arguments.

Value

Character vector of formatted financial year

See Also

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

Examples

```
format_fy(201617)
format_fy("201617")
```

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

Details

It accepts character arguments.

Value

Unformatted 6 digit year as string

See Also

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

Examples

```
format_fy_reverse("2016-17")
```

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. Get guidance on time and geography data.

`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 250 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 query URL so for every row included in the batch, and especial if those Id's go into the 1,000s or 10,000s they will increase the size of the URL 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

```
if (interactive()) {
  # Specify some parameters
  get_ons_api_data(
    data_id = "LAD23_RGN23_EN_LU",
    query_params =
      list(outFields = "column1, column2", outSR = "4326", f = "json")
  )

  # Just fetch everything
  get_ons_api_data(data_id = "LAD23_RGN23_EN_LU")
}
```

---

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 7 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_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
)
```

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.

Value

string featuring prettified value

See Also

comma_sep() round_five_up() as.numeric()

Other prettying: pretty_filesize(), pretty_num_table(), 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")

# 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().

See Also

pretty_num()

Other prettying: pretty_filesize(), pretty_num(), 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
pretty_num_table(df, dp = 2)

# 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_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()`

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

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)
```

---

wd_pcon_lad_la_rgn_ctry

Ward to Constituency to LAD to LA to Region to Country lookup

---

Description

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

Usage

```
wd_pcon_lad_la_rgn_ctry
```

Format

`wd_pcon_lad_la_rgn_ctry`**:**
A data frame with 24,629 rows and 14 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

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
new_la_code 9 digit local authority code
region_code 9 digit region code
country_code 9 digit country code

Details

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.

Source

https://geoportal.statistics.gov.uk/search?tags=lup_wd_pcon_lad_utla and https://geoportal.statistics.gov.uk/search?q=lup_w

---

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. Get more guidance on 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