Package ‘epiCleanr’

September 28, 2023

Type Package

Title A Tidy Solution for Epidemiological Data

Version 0.2.0

Description Offers a tidy solution for epidemiological data. It houses a range of functions for epidemiologists and public health data wizards for data management and cleaning.

License MIT + file LICENSE

URL `https://github.com/truenomad/epiCleanr`, `https://truenomad.github.io/epiCleanr/`

BugReports `https://github.com/truenomad/epiCleanr/issues`

Encoding UTF-8

Depends R (>= 3.6)

Imports rio, dplyr, tidyr, tools, withr, ggplot2, stringr, tidyselect, rlang, glue, crayon, countrycode, purrr, tibble, janitor

Suggests testthat, knitr, haven, readxl, openxlsx, foreign, yaml, clipr, xml2, readODS, rmatio, jsonlite, fst, feather, arrow, xts, data.table, R.utils, stringdist, stringi, ggtext, ggdist, zoo, wesanderson, scales, spsUtil, rmarkdown, ggh4x

RoxygenNote 7.2.3

VignetteBuilder knitr

NeedsCompilation no

Author Mohamed A. Yusuf [aut, cre] (<`https://orcid.org/0000-0002-9339-4613`>)

Maintainer Mohamed A. Yusuf <`mohamedayusuf87@gmail.com`>

Repository CRAN

Date/Publication 2023-09-28 12:20:05 UTC

R topics documented:

`clean_admin_names` *Clean and Match Administrative Names*

Description

This function takes administrative names and cleans them using various matching and string distance algorithms. It can also match the cleaned names with a base list provided by the user or fetched from ‘GeoNames‘, which is a official repository of standard spellings of all foreign geographic names.

Usage

```
clean_admin_names(
  admin_names_to_clean,
  country_code,
  admin_level = "adm2",
  user_base_admin_names = NULL,
  user_base_only = FALSE,
  report_mode = FALSE
)
```

Arguments

`admin_names_to_clean`	A character vector of administrative names to clean.
`country_code`	sed if ‘use_get_admin_names‘ is TRUE. A character string or numerical value of the country code (e.g., "KE"). This can be in various formats such as country name, ISO codes, UN codes, etc., see `countrycode::codelist()` for the full list of codes and naming conventions used.
`admin_level`	A character string indicating the administrative level (e.g., "adm2").
`user_base_admin_names`	A character of of administrative names that the use would like to use as reference. This is no necessary, downloaded ‘GeoNames‘ will be used if missing.
`user_base_only`	A logical indicating whether to use only the user-provided base administrative names (‘user_base_admin_names‘) for matching. If TRUE, ‘country_code‘ and ‘admin_names_to_clean‘ are not required. Default is FALSE.
`report_mode`	A logical indicating whether to return a detailed report. Default is FALSE.

Value

If ‘report_mode‘ is set to TRUE, a data frame containing the original admin names and the matched and cleaned admin names with inormation of the source of data used to clean including the algorithm used, else a cleaned list of names is returned.

See Also

`countrycode::codelist()` for the full list of codes and naming conventions.

Examples

```
# Example with country code
base_names <- c(
  "Paris", "Marseille", "Lyon",
  "Toulouse", "Nice", "Nantes", "Strasbourg",
  "Montpellier", "Bordeaux", "Lille"
)

unclean_names <- c(
  "Pariis", "Marseill", "Lyone",
  "Toulous", "Niice", "Nantees", "Strasbourgh",
  "Montpeelier", "Bordeuax", "Lilie"
)

france_new <- clean_admin_names(
  country_code = "Fr",
  user_base_admin_names = base_names,
  admin_names_to_clean = unclean_names
)

print(france_new)
```

`clean_names_strings` *Clean variable names or column names in various styles*

Description

This function transforms variable names or column names into one of the standard cleaned formats specified by the ‘style‘ argument. It offers more flexibility than `janitor::clean_names()` function by supporting individual strings and providing multiple naming styles.

Usage

```
clean_names_strings(input, style = "snake_case")
```

Arguments

`input`	A data frame, tibble, matrix, list, or character vector representing the names to be cleaned.
`style`	A character string specifying the naming style to use. Available options are "snake_case" (default), "camel_case", and "simple_clean".

Value

The object with cleaned names.

See Also

`janitor::clean_names()`

Examples

```
library(data.table)
library(zoo)
library(xts)

# For data frame with snake_case (default)
data("iris")
cleaned_iris <- clean_names_strings(iris)
colnames(cleaned_iris)

# For data frame with camel_case
cleaned_iris_camel <- clean_names_strings(iris, style = "camel_case")
colnames(cleaned_iris_camel)

# For character vector
original_names <- c("Some Column", "Another-Column!", "Yet Another Column")
cleaned_names <- clean_names_strings(original_names, style = "simple_clean")
print(cleaned_names)

# For matrix
mat <- matrix(1:4, ncol = 2)
colnames(mat) <- c("Some Column", "Another Column")
cleaned_mat <- clean_names_strings(mat)
colnames(cleaned_mat)

# For list
lst <- list("Some Column" = 1, "Another Column" = 2)
cleaned_lst <- clean_names_strings(lst)
names(cleaned_lst)

# For xts object
xts_obj <- xts(x = matrix(1:4, ncol = 2),
               order.by = as.Date('2021-01-01') + 0:1)
colnames(xts_obj) <- c("Some Column", "Another Column")
cleaned_xts <- clean_names_strings(xts_obj)
```

```
print(colnames(cleaned_xts))

zoo_obj <- zoo(matrix(1:4, ncol = 2), order.by = 1:2)
colnames(zoo_obj) <- c("Some Column", "Another Column")
cleaned_zoo <- clean_names_strings(zoo_obj)
print(colnames(cleaned_zoo))

# for Data table
dt <- data.table("Some Column" = 1:2, "Another Column" = 3:4)
cleaned_dt <- clean_names_strings(dt)
print(names(cleaned_dt))
```

`consistency_check` *Consistency Check Function*

Description

This function performs a consistency check to ensure that the number of tests is greater than the number of cases for given columns in a dataset. It returns a `ggplot2` object visualizing the results.

Usage

```
consistency_check(data, tests, cases)
```

Arguments

`data`	A data frame containing the test and case data.
`tests`	A character vector specifying the column names for the test data.
`cases`	A character vector specifying the column names for the case data. The length of 'tests' and 'cases' must be the same, and each element in 'tests' corresponds to an element in 'cases'.

Value

A `ggplot2::ggplot()` object showing the consistency between the number of tests and cases. The x-axis represents the cases, and the y-axis represents the tests. Each facet represents a disease, and the diagonal line shows where the number of tests equals the number of cases.

Examples

```
# check the consistency between malaria tests and cases

# get path
path <- system.file(
        "extdata",
        "fake_epi_df_togo.rds",
         package = "epiCleanr")
```

```
fake_epi_df_togo <- import(path)

consistency_check(fake_epi_df_togo,
                  tests = c("malaria_tests","cholera_tests"),
                  cases = c("malaria_cases", "cholera_cases"))
```

`create_test` *Create Test Function*

Description

This function creates a test function to perform various data validation checks. The returned function can be applied to a dataset to perform the specified tests.

Usage

```
create_test(
  dimension_test = NULL,
  combinations_test = NULL,
  row_duplicates = FALSE,
  col_duplicates = FALSE,
  min_threshold_test = NULL,
  max_threshold_test = NULL
)
```

Arguments

`dimension_test`	A vector of two integers specifying the expected number of rows and columns.
`combinations_test`	A list with the elements ‘variables‘ (character vector of variable names) and ‘expectation‘ (integer specifying the expected number of unique combinations for each column).
`row_duplicates`	Logical. If TRUE, checks for duplicate rows.
`col_duplicates`	Logical. If TRUE, checks for duplicate columns.
`min_threshold_test`	Named list of minimum threshold values for specified columns.
`max_threshold_test`	Named list of maximum threshold values for specified columns.

Value

A function to be applied to the dataset.

Examples

```
# get path
path <- system.file(
        "extdata",
        "fake_epi_df_togo.rds",
         package = "epiCleanr")

fake_epi_df_togo <- import(path)

# Set up unit-test function
my_tests <- create_test(
  # For checking the dimension of the data
  dimension_test = c(900, 9),
  # For expected number of combinations in data
 combinations_test = list(
   variables = c("month", "year", "district"),
   expectation = 12 * 5 * 15),
  # Check repeated cols, rows and max and min thresholds
  row_duplicates = TRUE, col_duplicates = TRUE,
  max_threshold_test = list(malaria_tests = 1000, cholera_tests = 1000),
 min_threshold_test = list(cholera_cases = 0, cholera_cases = 0)
)

result <- my_tests(fake_epi_df_togo)
```

`export` *Export Data to Various File Formats*

Description

This function provides a unified interface for exporting data to various file formats supported by the `rio::export()` function. The format is automatically detected from the file extension to simplify the exporting process.

Usage

```
export(data, file_path, ...)
```

Arguments

`data`	The dataset to be exported.
`file_path`	Character string specifying the path to the output file.
`...`	Additional arguments to be passed to the underlying write functions. These arguments are specific to the file format being exported. Please refer to the documentation of each package used for more information.

Value

No return value, called for side effects.

See Also

`rio::export()`, which this function is based on.

Examples

```
# Create temporary account
tmpdir <- tempfile()
dir.create(tmpdir)

# Export a CSV file
export(mtcars, file_path = file.path(tmpdir, "file.csv"))

# Export an Excel file
export(mtcars, file_path = file.path(tmpdir, "file.xlsx"))

# Export a Stata DTA file
export(mtcars, file_path = file.path(tmpdir, "file.dta"))

# Export an RDS file
export(mtcars, file_path = file.path(tmpdir, "file.rds"))

# Export an RData file
export(list(mtcars = mtcars, iris = iris),
       file_path = file.path(tmpdir, "file.RData"))

# Remove the temporary directory and its contents
unlink(tmpdir, recursive = TRUE)
```

`get_admin_names` *Retrieve Administrative Names from GeoNames*

Description

This function grabs administrative region names (such as districts, provinces, etc.) for a given country from the ‘GeoNames‘ website. It accepts both country names and various country coding schemes.

Usage

```
get_admin_names(country_name_or_code, silent_mode = TRUE)
```

Arguments

`country_name_or_code`	Character or numeric. The name or code of the country for which administrative names are to be retrieved. This can be in various formats such as country name, ISO codes, UN codes, etc., see ‘countrycode::codelist()‘ for the full list of codes and naming conventions used.
`silent_mode`	A logical indicating whether to suppress messages. Default is TRUE.

Value

A list containing administrative region names and details for different administrative levels (e.g., ADM1, ADM2, etc.). Each element of the list corresponds to a different administrative level and contains a data frame with columns such as country_code, ascii name, alternate names, latitude, longitude, and date last updated.

See Also

‘Geonames‘ website for the source of admin names data

Examples

```
# example using different naming/code conventions
three_digit <- get_admin_names("TGO")   # using 3 digit iso codes
two_digit <- get_admin_names("TG")      # using 2 digit iso codes
un_code <- get_admin_names(768)         # using UN codes
full_name <-  get_admin_names("Togo")   # using full names

str(full_name$adm2)
```

`handle_outliers` *Detect and Handle Outliers in Dataset*

Description

This function identifies and handles outliers in a given dataset using various methods including Z-Score, Modified Z-Score, and Inter-Quartile Range (IQR). It also provides options to treat the identified outliers, using mean, median, rolling mean by group and inter-quartile range. It also has the option to generate a summary report and a plot.

Usage

```
handle_outliers(
  data,
  vars = NULL,
  method = NULL,
  zscore_threshold = 3,
  mod_zscore_threshold = 3.5,
  iqr_k_value = 1.5,
  treat_method = "none",
  grouping_vars = NULL,
  report_mode = FALSE
)
```

Arguments

`data`	Dataframe containing the variables to be checked for outliers.
`vars`	Character vector of variable names to check for outliers. Default is NULL, which selects all numeric columns.
`method`	Character indicating the method for outlier detection. Options are "zscore", "mod_zscore", and "iqr_method". Default is NULL, which applies all methods.
`zscore_threshold`	Numeric value for Z-Score threshold. Default is 3.
`mod_zscore_threshold`	Numeric value for Modified Z-Score threshold. Default is 3.5.
`iqr_k_value`	Numeric value for IQR multiplier. Default is 1.5.
`treat_method`	Character indicating how to treat outliers. Options are "none", "remove", "mean", "median", "grouped_mean", and "quantile". Default is "none".
`grouping_vars`	Character vector of grouping variables for "grouped_mean". Required only if treat_method is "grouped_mean".
`report_mode`	Logical, if TRUE, the function returns a summary report and a plot. Default is FALSE.

Value

If report_mode is TRUE, a list containing a summary dataframe and a ggplot object. Otherwise, a dataframe with outliers treated according to treat_method.

Examples

```
# get path
path <- system.file(
        "extdata",
        "fake_epi_df_togo.rds",
         package = "epiCleanr")

fake_epi_df_togo <- import(path)

variables <- c("malaria_tests", "malaria_cases",
               "cholera_tests", "cholera_cases")
result <- handle_outliers(fake_epi_df_togo, vars = variables,
               method = "zscore", report_mode = TRUE)
```

```
print(result$report)

print(result$plot)
```

---

import *Import Data from Various File Formats*

Description

This function provides a unified interface for importing data from various file formats supported by the rio package. The format is automatically detected from the file extension to simplify the importing process.

Usage

```
import(file_path, ...)
```

Arguments

file_path	Character string specifying the path to the input file or a URL pointing to the dataset.
...	Additional arguments to be passed to the underlying read functions. These arguments are specific to the file format being imported. Please refer to the documentation of each package used for more information.

Value

A data frame or appropriate R object containing the imported data.

See Also

rio::import(), which this function is based on.

Examples

```
# Locate test data directory
path <-  system.file("extdata",
                     package = "epiCleanr")

# Import a CSV file
data_csv <- import(file_path = file.path(path, "test_data.csv"))

# Import an Excel file
data_excel <- import(file_path = file.path(path, "test_data.xlsx"))

# Import a Stata DTA file
data_dta <- import(file_path = file.path(path, "test_data.dta"))
```

```
# Import an RDS file
data_rds <- import(file_path = file.path(path, "test_data.rds"))

# Import an RData file
data_rdata <- import(file_path = file.path(path, "test_data.RData"))

# Import an SPSS file
data_spss <- import(file_path = file.path(path, "test_data.sav"))
```

---

missing_plot *Plot Missing data over time*

Description

This function visualizes the proportion of missing data or reporting rate for specified variables in a dataset. It creates a tile plot using ggplot2; where the x-axis can represent any categorical time such as time (e.g., year, month), and the y-axis can represents either variables or groupings (e.g., state). The output can further be manipulated to one's needs.

Usage

```
missing_plot(data, x_var, y_var = NULL, miss_vars = NULL, use_rep_rate = FALSE)
```

Arguments

data	A data frame containing the data to be visualized. Must include columns specified in 'x_var', 'y_var', and 'vars'.
x_var	A character string specifying the time variable in 'data' (e.g., "year", "month"). Must be provided.
y_var	An optional character string specifying the grouping variable in 'data' (e.g., "state"). If provided, only one variable can be specified in 'vars'.
miss_vars	An optional character vector specifying the variables to be visualized in 'data'. If NULL, all variables except 'x_var' and 'y_var' will be used.
use_rep_rate	A logical value. If TRUE, the reporting rate is visualized; otherwise, the proportion of missing data is visualized. Defaults to FALSE

Value

A ggplot2 object representing the tile plot.

Examples

```
# get path
path <- system.file(
        "extdata",
        "fake_epi_df_togo.rds",
         package = "epiCleanr")

fake_epi_df_togo <- import(path)

# Check misisng data by year
result <- missing_plot(fake_epi_df_togo,
             x_var = "year", use_rep_rate = FALSE)
```

Index