Package: phsmethods (via r-universe)

August 30, 2024

Title Standard Methods for Use in Public Health Scotland

Version 1.0.2

Description A collection of methods for commonly undertaken analytical tasks, primarily developed for Public Health Scotland (PHS) analysts, but the package is also generally useful to others working in the healthcare space, particularly since it has functions for working with Community Health Index (CHI) numbers. The package can help to make data manipulation and analysis more efficient and reproducible.

License GPL (>= 2)

URL https://github.com/Public-Health-Scotland/phsmethods, https://public-health-scotland.github.io/phsmethods/

BugReports https://github.com/Public-Health-Scotland/phsmethods/issues

Depends R (>= 3.6)

Imports cli, dplyr, lifecycle, lubridate, magrittr, readr, rlang, scales (>= 1.0.0), stringr, tibble, utils

Suggests covr, ggplot2, here, knitr, rmarkdown, spelling, testthat (>= 3.0.0)

RdMacros lifecycle

Config/testthat/edition 3

Encoding UTF-8

Language en-GB

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

VignetteBuilder knitr

Repository https://public-health-scotland.r-universe.dev

RemoteUrl https://github.com/public-health-scotland/phsmethods

RemoteRef HEAD

RemoteSha f38d440e58aedec32196c443859a8a549dd3c51b

Contents

`age_calculate` *Calculate age between two dates*

Description

This function calculates the age between two dates using functions in `lubridate`. It calculates age in either years or months.

Usage

```
age_calculate(
  start,
  end = if (lubridate::is.Date(start)) Sys.Date() else Sys.time(),
  units = c("years", "months"),
  round_down = TRUE
)
```

Arguments

`start`	A start date (e.g. date of birth) which must be supplied with `Date` or `POSIXct` or `POSIXlt` class. `base::as.Date()`, `lubridate::dmy()` and `as.POSIXct()` are examples of functions which can be used to store dates as an appropriate class.
`end`	An end date which must be supplied with `Date` or `POSIXct` or `POSIXlt` class. Default is `Sys.Date()` or `Sys.time()` depending on the class of `start`.
`units`	Type of units to be used. years and months are accepted. Default is `years`.
`round_down`	Should returned ages be rounded down to the nearest whole number. Default is `TRUE`.

Value

A numeric vector representing the ages in the given units.

Examples

```
library(lubridate)
birth_date <- lubridate::ymd("2020-02-29")
end_date <- lubridate::ymd("2022-02-21")
age_calculate(birth_date, end_date)
age_calculate(birth_date, end_date, units = "months")

# If the start day is leap day (February 29th), age increases on 1st March
# every year.
leap1 <- lubridate::ymd("2020-02-29")
leap2 <- lubridate::ymd("2022-02-28")
leap3 <- lubridate::ymd("2022-03-01")

age_calculate(leap1, leap2)
age_calculate(leap1, leap3)
```

---

age_from_chi *Extract age from the CHI number*

Description

`age_from_chi` takes a CHI number or a vector of CHI numbers and returns the age as implied by the CHI number(s). If the Date of Birth (DoB) is ambiguous it will return NA. It uses `dob_from_chi()`.

Usage

```
age_from_chi(
  chi_number,
  ref_date = NULL,
  min_age = 0,
  max_age = NULL,
  chi_check = TRUE
)
```

Arguments

`chi_number`	a CHI number or a vector of CHI numbers with `character` class.
`ref_date`	calculate the age at this date, default is to use `Sys.Date()` i.e. today.
`min_age, max_age`	optional min and/or max dates that the DoB could take as the century needs to be guessed. Must be either length 1 for a 'fixed' age or the same length as `chi_number` for an age per CHI number. `min_age` can be age based on common sense in the dataset, whilst `max_age` can be age when an event happens such as the age at discharge.

chi_check — logical, optionally skip checking the CHI for validity which will be faster but should only be used if you have previously checked the CHI(s), the default (TRUE) will to check the CHI numbers.

Value

an integer vector of ages in years truncated to the nearest year. It will be the same length as chi_number.

Examples

```
age_from_chi("0101336489")

library(tibble)
library(dplyr)
data <- tibble(chi = c(
  "0101336489",
  "0101405073",
  "0101625707"
), dis_date = as.Date(c(
  "1950-01-01",
  "2000-01-01",
  "2020-01-01"
)))

data %>%
  mutate(chi_age = age_from_chi(chi))

data %>%
  mutate(chi_age = age_from_chi(chi, min_age = 18, max_age = 65))

data %>%
  mutate(chi_age = age_from_chi(chi,
    ref_date = dis_date
  ))
```

---

area_lookup — *Codes and names of Scottish geographical and administrative areas.*

Description

A dataset containing Scotland's geography codes and associated area names. It is used within match_area().

Usage

```
area_lookup
```

Format

A `tibble::tibble()` with 2 variables and over 17,000 rows:

geo_code Standard geography code - 9 characters

area_name Name of the area the code represents

Details

`geo_code` contains geography codes pertaining to Health Boards, Council Areas, Health and Social Care Partnerships, Intermediate Zones, Data Zones (2001 and 2011), Electoral Wards, Scottish Parliamentary Constituencies, UK Parliamentary Constituencies, Travel to work areas, National Parks, Community Health Partnerships, Localities (S19), Settlements (S20) and Scotland.

Source

https://statistics.gov.scot/

See Also

The script used to create the `area_lookup` dataset on GitHub.

`chi_check` *Check the validity of a CHI number*

Description

`chi_check` takes a CHI number or a vector of CHI numbers with `character` class. It returns feedback on the validity of the entered CHI number and, if found to be invalid, provides an explanation as to why.

Usage

```
chi_check(x)
```

Arguments

`x`	a CHI number or a vector of CHI numbers with `character` class.

Details

The Community Health Index (CHI) is a register of all patients in NHS Scotland. A CHI number is a unique, ten-digit identifier assigned to each patient on the index.

The first six digits of a CHI number are a patient's date of birth in DD/MM/YY format.

The ninth digit of a CHI number identifies a patient's sex: odd for male, even for female. The tenth digit is a check digit, denoted `checksum`.

While a CHI number is made up exclusively of numeric digits, it cannot be stored with `numeric` class in R. This is because leading zeros in numeric values are silently dropped, a practice not

exclusive to R. For this reason, `chi_check` accepts input values of `character` class only. A leading zero can be added to a nine-digit CHI number using `chi_pad()`.

`chi_check` assesses whether an entered CHI number is valid by checking whether the answer to each of the following criteria is `Yes`:

- Does it contain no non-numeric characters?
- Is it ten digits in length?
- Do the first six digits denote a valid date?
- Is the checksum digit correct?

Value

`chi_check` returns a character string. Depending on the validity of the entered CHI number, it will return one of the following:

- `Valid CHI`
- `Invalid character(s) present`
- `Too many characters`
- `Too few characters`
- `Invalid date`
- `Invalid checksum`
- `Missing (NA)`
- `Missing (Blank)`

Examples

```
chi_check("0101011237")
chi_check(c("0101201234", "3201201234"))

library(dplyr)
df <- tibble(chi = c(
  "3213201234",
  "123456789",
  "12345678900",
  "010120123?",
  NA
))
df %>%
  mutate(validity = chi_check(chi))
```

chi_pad *Add a leading zero to nine-digit CHI numbers*

Description

chi_pad takes a nine-digit CHI number with character class and prefixes it with a zero. Any values provided which are not a string comprised of nine numeric digits remain unchanged.

Usage

```
chi_pad(x)
```

Arguments

x	a CHI number or a vector of CHI numbers with character class.

Details

The Community Health Index (CHI) is a register of all patients in NHS Scotland. A CHI number is a unique, ten-digit identifier assigned to each patient on the index.

The first six digits of a CHI number are a patient's date of birth in DD/MM/YY format. The first digit of a CHI number must, therefore, be 3 or less. Depending on the source, CHI numbers are sometimes missing a leading zero.

While a CHI number is made up exclusively of numeric digits, it cannot be stored with numeric class in R. This is because leading zeros in numeric values are silently dropped, a practice not exclusive to R. For this reason, chi_pad accepts input values of character class only, and returns values of the same class. It does not assess the validity of a CHI number - please see chi_check() for that.

Value

The original character vector with CHI numbers padded if applicable.

Examples

```
chi_pad(c("101011237", "101201234"))
```

`create_age_groups` *Create age groups*

Description

`create_age_groups()` takes a numeric vector and assigns each age to the appropriate age group.

Usage

```
create_age_groups(x, from = 0, to = 90, by = 5, as_factor = FALSE)
```

Arguments

`x`	a vector of numeric values
`from`	the start of the smallest age group. The default is `0`.
`to`	the end point of the age groups. The default is `90`.
`by`	the size of the age groups. The default is `5`.
`as_factor`	The default behaviour is to return a character vector. Use `TRUE` to return a factor vector instead.

Details

The `from`, `to` and `by` values are used to create distinct age groups. `from` dictates the starting age of the lowest age group, and `by` indicates how wide each group should be. `to` stipulates the cut-off point at which all ages equal to or greater than this value should be categorised together in a `to+` group. If the specified value of `to` is not a multiple of `by`, the value of `to` is rounded down to the nearest multiple of `by`.

The default values of `from`, `to` and `by` correspond to the European Standard Population age groups.

Value

A character vector, where each element is the age group for the corresponding element in `x`. If `as_factor = TRUE`, a factor vector is returned instead.

Examples

```
age <- c(54, 7, 77, 1, 26, 101)

create_age_groups(age)
create_age_groups(age, from = 0, to = 80, by = 10)

# Final group may start below 'to'
create_age_groups(age, from = 0, to = 65, by = 10)

# To get the output as a factor:
create_age_groups(age, as_factor = TRUE)
```

dob_from_chi *Extract Date of Birth (DoB) from the CHI number*

Description

dob_from_chi takes a CHI number or a vector of CHI numbers and returns the Date of Birth (DoB) as implied by the CHI number(s). If the DoB is ambiguous it will return NA.

Usage

```
dob_from_chi(chi_number, min_date = NULL, max_date = NULL, chi_check = TRUE)
```

Arguments

chi_number	a CHI number or a vector of CHI numbers with character class.
min_date, max_date	optional min and/or max dates that the DoB could take as the century needs to be guessed. Must be either length 1 for a 'fixed' date or the same length as chi_number for a date per CHI number. min_date can be date based on common sense in the dataset, whilst max_date can be date when an event happens such as discharge date.
chi_check	logical, optionally skip checking the CHI for validity which will be faster but should only be used if you have previously checked the CHI(s). The default (TRUE) will check the CHI numbers.

Value

a date vector of DoB. It will be the same length as chi_number.

Examples

```
dob_from_chi("0101336489")

library(tibble)
library(dplyr)
data <- tibble(chi = c(
  "0101336489",
  "0101405073",
  "0101625707"
), adm_date = as.Date(c(
  "1950-01-01",
  "2000-01-01",
  "2020-01-01"
)))

data %>%
  mutate(chi_dob = dob_from_chi(chi))
```

```
data %>%
  mutate(chi_dob = dob_from_chi(chi,
    min_date = as.Date("1930-01-01"),
    max_date = adm_date
  ))
```

`extract_fin_year` *Extract the formatted financial year from a date*

Description

`extract_fin_year` takes a date and extracts the correct financial year in the PHS specified format from it.

Usage

```
extract_fin_year(date)
```

Arguments

`date`	A date which must be supplied with `Date`, `POSIXct`, `POSIXlt` or `POSIXt` class. `base::as.Date()`, `lubridate::dmy()` and `as.POSIXct()` are examples of functions which can be used to store dates as an appropriate class.

Details

The PHS accepted format for financial year is YYYY/YY e.g. 2017/18.

Value

A character vector of financial years in the form '2017/18'.

Examples

```
x <- lubridate::dmy(c(21012017, 04042017, 17112017))
extract_fin_year(x)
```

`file_size` *Calculate file size*

Description

`file_size` takes a filepath and an optional regular expression pattern. It returns the size of all files within that directory which match the given pattern.

Usage

```
file_size(filepath = getwd(), pattern = NULL)
```

Arguments

`filepath`	A character string denoting a filepath. Defaults to the working directory, `getwd()`.
`pattern`	An optional character string denoting a `regular expression()` pattern. Only file names which match the regular expression will be returned. See the **See Also** section for resources regarding how to write regular expressions.

Details

The sizes of files with certain extensions are returned with the type of file prefixed. For example, the size of a 12 KB `.xlsx` file is returned as `Excel 12 KB`. The complete list of explicitly catered-for file extensions and their prefixes are as follows:

- `.xls`, `.xlsb`, `.xlsm` and `.xlsx` files are prefixed with `Excel`
- `.csv` files are prefixed with `CSV`
- `.sav` and `.zsav` files are prefixed with `SPSS`
- `.doc`, `.docm` and `.docx` files are prefixed with `Word`
- `.rds` files are prefixed with `RDS`
- `.txt` files are prefixed with `Text`,
- `.fst` files are prefixed with `FST`,
- `.pdf` files are prefixed with `PDF`,
- `.tsv` files are prefixed with `TSV`,
- `.html` files are prefixed with `HTML`,
- `.ppt`, `.pptm` and `.pptx` files are prefixed with `PowerPoint`,
- `.md` files are prefixed with `Markdown`

Files with extensions not contained within this list will have their size returned with no prefix. To request that a certain extension be explicitly catered for, please create an issue on GitHub.

File sizes are returned as the appropriate multiple of the unit byte (bytes (B), kilobytes (KB), megabytes (MB), etc.). Each multiple is taken to be 1,024 units of the preceding denomination.

Value

A tibble::tibble() listing the names of files within filepath which match pattern and their respective sizes. The column names of this tibble are name and size. If no pattern is specified, file_size returns the names and sizes of all files within filepath. File names and sizes are returned in alphabetical order of file name. Sub-folders contained within filepath will return a file size of 0 B.

If filepath is an empty folder, or pattern matches no files within filepath, file_size returns NULL.

See Also

For more information on using regular expressions, see this Jumping Rivers blog post and this vignette from the stringr() package.

Examples

```
# Name and size of all files in working directory
file_size()

# Name and size of .xlsx files only in working directory
file_size(pattern = "\\.xlsx$")

# Size only of alphabetically first file in working directory
library(magrittr)
file_size() %>%
  dplyr::pull(size) %>%
  extract(1)
```

---

format_postcode *Format a postcode*

Description

format_postcode takes a character string or vector of character strings. It extracts the input values which adhere to the standard UK postcode format (with or without spaces), assigns the appropriate amount of spacing to them (for both pc7 and pc8 formats) and ensures all letters are capitalised.

Usage

```
format_postcode(x, format = c("pc7", "pc8"), quiet = FALSE)
```

Arguments

x	A character string or vector of character strings. Input values which adhere to the standard UK postcode format may be upper or lower case and will be formatted regardless of existing spacing. Any input values which do not adhere to the standard UK postcode format will generate an NA and a warning message - see **Value** section for more information.

format	A character string denoting the desired output format. Valid options are pc7 and pc8. The default is pc7. See **Value** section for more information on the string length of output values.
quiet	(optional) If quiet is TRUE all messages and warnings will be suppressed. This is useful in a production context and when you are sure of the data or you are specifically using this function to remove invalid postcodes. This will also make the function a bit quicker as fewer checks are performed.

Details

The standard UK postcode format (without spaces) is:

- 1 or 2 letters, followed by
- 1 number, followed by
- 1 optional letter or number, followed by
- 1 number, followed by
- 2 letters

UK government regulations mandate which letters and numbers can be used in specific sections of a postcode. However, these regulations are liable to change over time. For this reason, format_postcode does not validate whether a given postcode actually exists, or whether specific numbers and letters are being used in the appropriate places. It only assesses whether the given input is consistent with the above format and, if so, assigns the appropriate amount of spacing and capitalises any lower case letters.

Value

When format is set equal to pc7, format_postcode returns a character string of length 7. 5 character postcodes have two spaces after the 2nd character; 6 character postcodes have 1 space after the 3rd character; and 7 character postcodes have no spaces.

When format is set equal to pc8, format_postcode returns a character string with maximum length 8. All postcodes, whether 5, 6 or 7 characters, have one space before the last 3 characters.

Any input values which do not adhere to the standard UK postcode format will generate an NA output value and a warning message. A warning is generated rather than an error so as not to let one erroneously recorded postcode in a large input vector prevent the remaining entries from being appropriately formatted.

Any input values which do adhere to the standard UK postcode format but contain lower case letters will generate a warning message explaining that these letters will be capitalised.

Examples

```
format_postcode("G26QE")
format_postcode(c("KA89NB", "PA152TY"), format = "pc8")

library(dplyr)
df <- tibble(postcode = c("G429BA", "G207AL", "DD37JY", "DG98BS"))
df %>%
  mutate(postcode = format_postcode(postcode))
```

match_area *Translate geography codes into area names*

Description

match_area takes a geography code or vector of geography codes. It matches the input to the corresponding value in the area_lookup() dataset and returns the corresponding area name.

Usage

```
match_area(x)
```

Arguments

x	A geography code or vector of geography codes.

Details

match_area relies predominantly on the standard 9 digit geography codes. The only exceptions are:

- RA2701: No Fixed Abode
- RA2702: Rest of UK (Outside Scotland)
- RA2703: Outside the UK
- RA2704: Unknown Residency

match_area caters for both current and previous versions of geography codes (e.g 2014 and 2019 Health Boards).

It can account for geography codes pertaining to Health Boards, Council Areas, Health and Social Care Partnerships, Intermediate Zones, Data Zones (2001 and 2011), Electoral Wards, Scottish Parliamentary Constituencies, UK Parliamentary Constituencies, Travel to work areas, National Parks, Community Health Partnerships, Localities (S19), Settlements (S20) and Scotland.

match_area returns a non-NA value only when an exact match is present between the input value and the corresponding variable in the area_lookup() dataset. These exact matches are sensitive to both case and spacing. It is advised to inspect area_lookup() in the case of unexpected results, as these may be explained by subtle differences in transcription between the input value and the corresponding value in the lookup dataset.

Value

Each geography code within Scotland is unique, and consequently match_area returns a single area name for each input value. Any input value without a corresponding value in the area_lookup() dataset will return an NA output value.

Examples

```
match_area("S20000010")

library(dplyr)
df <- tibble(code = c("S02000656", "S02001042", "S08000020", "S12000013"))
df %>% mutate(name = match_area(code))
```

---

phsmethods phsmethods *package*

Description

Standard Methods for use in PHS.

Details

See the README on GitHub.

Author(s)

Maintainer: Tina Fu <Yuyan.Fu2@phs.scot>

Authors:

- David Caldwell <David.Caldwell@phs.scot>
- Jack Hannah <jack.hannah2@phs.scot>
- Ciara Gribben <Ciara.Gribben@phs.scot>
- Chris Deans <Chris.Deans2@phs.scot>
- Jaime Villacampa <Jaime.Villacampa@phs.scot>
- Graeme Gowans <Graeme.Gowans@phs.scot>
- James McMahon <James.McMahon@phs.scot> (ORCID)
- Nicolaos Christofidis <nicolaos.christofidis@phs.scot>

Other contributors:

- Public Health Scotland <phs.datascience@phs.scot> [copyright holder]
- Lucinda Lawrie <Lucinda.Lawrie@phs.scot> [reviewer]
- Alice Byers [contributor]
- Alan Yeung <Alan.Yeung@phs.scot> [contributor]

See Also

Useful links:

- https://github.com/Public-Health-Scotland/phsmethods
- https://public-health-scotland.github.io/phsmethods/
- Report bugs at https://github.com/Public-Health-Scotland/phsmethods/issues

qtr *Assign a date to a quarter*

Description

The qtr functions take a date input and calculate the relevant quarter-related value from it. They all return the year as part of this value.

- `qtr` returns the current quarter
- `qtr_end` returns the last month in the quarter
- `qtr_next` returns the next quarter
- `qtr_prev` returns the previous quarter

Usage

```
qtr(date, format = c("long", "short"))

qtr_end(date, format = c("long", "short"))

qtr_next(date, format = c("long", "short"))

qtr_prev(date, format = c("long", "short"))
```

Arguments

`date`	A date which must be supplied with `Date` or `POSIXct`
`format`	A `character` string specifying the format the quarter should be displayed in. Valid options are `long` (January to March 2018) and `short` (Jan-Mar 2018). The default is `long`.

Details

Quarters are defined as:

- January to March (Jan-Mar)
- April to June (Apr-Jun)
- July to September (Jul-Sep)
- October to December (Oct-Dec)

Value

A character vector of financial quarters in the specified format.

Examples

```
x <- lubridate::dmy(c(26032012, 04052012, 23092012))
qtr(x)
qtr_end(x, format = "short")
qtr_next(x)
qtr_prev(x, format = "short")
```

`sex_from_chi` *Extract sex from the CHI number*

Description

`sex_from_chi` takes a CHI number or a vector of CHI numbers and returns the sex as implied by the CHI number(s). The default return type is an integer but this can be modified.

Usage

```
sex_from_chi(
  chi_number,
  male_value = 1L,
  female_value = 2L,
  as_factor = FALSE,
  chi_check = TRUE
)
```

Arguments

`chi_number`	a CHI number or a vector of CHI numbers with `character` class.
`male_value, female_value`	optionally supply custom values for Male and Female. Note that that these must be of the same class.
`as_factor`	logical, optionally return as a factor with labels `'Male'` and `'Female'`. Note that this will override any custom values supplied with `male_value` or `female_value`.
`chi_check`	logical, optionally skip checking the CHI for validity which will be faster but should only be used if you have previously checked the CHI(s).

Details

The Community Health Index (CHI) is a register of all patients in NHS Scotland. A CHI number is a unique, ten-digit identifier assigned to each patient on the index.

The ninth digit of a CHI number identifies a patient's sex: odd for men, even for women.

The default behaviour for `sex_from_chi` is to first check the CHI number is valid using `check_chi` and then to return 1 for male and 2 for female.

There are options to return custom values e.g. `'M'` and `'F'` or to return a factor which will have labels `'Male'` and `'Female'`)

Value

a vector with the same class as `male_value` and `female_value`, (integer by default) unless `as_factor` is `TRUE` in which case a factor will be returned.

Examples

```
sex_from_chi("0101011237")
sex_from_chi(c("0101011237", "0101336489", NA))
sex_from_chi(
  c("0101011237", "0101336489", NA),
  male_value = "M",
  female_value = "F"
)
sex_from_chi(c("0101011237", "0101336489", NA), as_factor = TRUE)

library(dplyr)
df <- tibble(chi = c("0101011237", "0101336489", NA))
df %>% mutate(chi_sex = sex_from_chi(chi))
```

Index