Package: socR (via r-universe)

June 9, 2026

Type Package

Title Useful functions for working with occupation coding

Version 0.8.0

Maintainer Daniel E. Russ <druss@mail.nih.gov>

Description A set of functions that I find useful in my research into occupational coding and codes.

License file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

Imports rlang, dplyr, tidyr, magrittr, tibble, purrr, readr, stringr, httr, digest, readxl, DBI, RSQLite, rio, pillar

Suggests knitr, rmarkdown, ggplot2, testthat (>= 3.0.0)

Config/testthat/edition 3

Depends R (>= 2.10)

VignetteBuilder knitr

URL https://danielruss.github.io/socR/

Config/pak/sysreqs make libicu-dev libssl-dev libx11-dev zlib1g-dev

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

Date/Publication 2025-07-22 21:01:26 UTC

RemoteUrl https://github.com/danielruss/socR

RemoteRef main

RemoteSha 0c0a0a6ff15ba860543f41fe28e172cbcb96a784

Contents

as_codingsystem *Create a coding system from a data frame*

Description

Create a coding system from a data frame

Usage

```
as_codingsystem(x, name = "", ...)

## S3 method for class 'data.frame'
as_codingsystem(x, name = "", ...)

## S3 method for class 'codingsystem'
as_codingsystem(x, name = "", ...)
```

Arguments

x	the data frame containing columns "code" and "title"
name	coding system name
...	additional parameters

Value

a codingsystem object.

bin_center *Bin Score and Assign center value*

Description

The `bin_center()` function takes a vector of scores between 0-1 and a number of bins an returns the center of the bin the score falls in.

Usage

```
bin_center(score, n_bins)
```

Arguments

score	The scores (values from 0-1) that are being binned
n_bins	the number of bins the score

Value

the center of the score bin for all scores

codeJobHistory *Auto-code job results with SOCcer using via the soccer API*

Description

This codes one job at a time. In order to code multiple jobs, you can create a tibble (data frame) and use pmap_dfr to produce a results tibble similar to the web-based version of SOCcer (https://soccer.nci.nih.gov)

Usage

```
codeJobHistory(title, task = "", industry = "", ..., n = 10)
```

Arguments

title	The job title
task	tasks performed on the job
industry	industry (SIC 1987 code)
...	(not used)
n	the number of soc codes to return (default)

Details

Please use the web-based version for handling large jobs.

Value

a tibble consisting of the title/task/industry and the top n SOCcer results and scores

Examples

```
## Not run:
soccer_results <- codeJobHistory("epidemiologist")
jobs <- tibble::tibble(title=c("chemist","farmer","data scientist"),
                       task=rep("",3),industry=rep("",3))
soccer_results_3 <- purrr::pmap_dfr(jobs,codeJobHistory,n=20)

## End(Not run)
```

codes *get the codes for a crosswalk,*

Description

returns the all the codes in column code_column

Usage

```
codes(x, code_column)
```

Arguments

x	A crosswalk of class xwalk
code_column	The column names for the desired codes.

codesAgree *Check if codes agree with reviewer*

Description

Compares if a vector of codes is in a vector of reviewer codes.

Usage

```
codesAgree(codes, reviewer)
```

Arguments

codes	codes to compare
reviewer	reviewer's code – "gold" standard

Details

Particularly useful when combined with purrr::map_lgl

Value

TRUE if the codes are in the reviewer otherwise FALSE

Examples

```
x <- '11-1011'
y <- c('11-1011','11-1031')
codesAgree(x,c("11-1011","11-1021"))
codesAgree(y,c("11-1021","11-1031"))
codesAgree(x,c("13-1011","11-1021"))
codesAgree(y,c("13-1011","11-1021"))
```

codingsystem *constructor create a coding system S3 class*

Description

constructor create a coding system S3 class

Usage

```
codingsystem(codes, titles, ..., name = "")
```

Arguments

codes	vector of codes, a dataframe containing the columns "code" (with codes) and "title" (with titles), or a url/file path of a csv file containing the codes and titles with header row containing at least "code" and title. Other columns may be present.
titles	vector of title
...	additional parameters passed into rio::import
name	coding system name

Value

the codingsystem object

Examples

```
url <- "https://danielruss.github.io/codingsystems/naics2022_all.csv"
naic2022 <- codingsystem(url,name = "naics2022",
   colClasses=c(rep("character",2),"integer",rep("character",5)))
```

combine_crosswalks *Combine two crosswalks to produce a new crosswalk*

Description

Takes two concordance tables (xw1 and xw2), where xw1 go from coding system one to an intermediary coding system, and xw2 goes from the intermediary coding system to coding system two. The goal is to make one table that goes from coding system 1 to coding system 2.

Usage

```
combine_crosswalks(xw1, xw2)
```

Arguments

xw1	- crosswalk 1, either an xwalk object or a data.frame
xw2	- crosswalk 2, either an xwalk object or a data.frame

Examples

```
# the noc_isco example has an extra column that confuses the parser,
# so I have to specify the parts or skip the last column.
noc_isco <- xwalk("https://danielruss.github.io/codingsystems/noc2011_isco2008.csv",
                  col_types = "cccc-")
isco_soc <- xwalk("https://danielruss.github.io/codingsystems/isco2008_soc2010.csv")
combine_crosswalks(noc_isco,isco_soc)
```

---

createMultiHotEncoder *creates a multihot encoder from a list of labels*

Description

creates a multihot encoder from a list of labels

Usage

```
createMultiHotEncoder(allLabels)
```

Arguments

allLabels	The complete set of labels

Value

a function that preforms multihot encoding

---

crosswalk *Use the concordance table (crosswalk) to convert from one coding system to another.*

Description

Use the concordance table (crosswalk) to convert from one coding system to another.

Usage

```
crosswalk(codes, xwalk, invert = FALSE, unlist = FALSE)
```

Arguments

codes	the vector of codes that will be crosswalked
xwalk	the concordance table.
invert	by default the crosswalk goes from codes1 to codes2 setting invert to TRUE make the crosswalk go from codes2 to codes1
unlist	instead of returning a list, return an unamed vector use it when crosswalking a dataframe column with mutate

Value

an unnamed list of codes in the resulting coding system

---

crosswalk_columns *Crosswalk multiple columns in a tibble/data frame*

Description

If you have a data frame of data with multiple columns that need to be crosswalked, use this in a pipe.

Usage

```
crosswalk_columns(.data, xwalk, new_column_name, ..., unnest_results = TRUE)
```

Arguments

.data	job data
xwalk	crosswalk going from code system 1 to coding system 2
new_column_name	the column name for the results if the results are unnested, the results will be colname_1, colname_2 ... colname_n, otherwise the results are a list column with name new_column_name.
...	Columns that need to be crosswalked
unnest_results	default=TRUE, should the results be separated into individual columns or else as a single list column

Value

a crosswalked tibble.

Examples

```
## Not run:
a <- tibble::tibble(id=c("job-1","job-2"),soc2010_1=c("11-1011","11-2011"),
     soc2010_2=c("11-1021",NA))
xw <- socR::xwalk("https://danielruss.github.io/codingsystems/soc2010_soc2018.csv")
a |> crosswalk_columns(xw,soc2018_xw,soc2010_1,soc2010_2)
a |> crosswalk_columns(xw,soc2018_xw,soc2010_1,soc2010_2,unnest_results=FALSE)

## End(Not run)
```

`dim.codingsystem` *Dimensions of an Object*

Description

Retrieve or set the dimension of an object.

Usage

```
## S3 method for class 'codingsystem'
dim(x)
```

Arguments

`x`	an R object, for example a matrix, array or data frame.

Details

The functions `dim` and `dim<-` are internal generic primitive functions.

`dim` has a method for `data.frame`s, which returns the lengths of the `row.names` attribute of `x` and of `x` (as the numbers of rows and columns respectively).

Value

For an array (and hence in particular, for a matrix) `dim` retrieves the `dim` attribute of the object. It is `NULL` or a vector of mode `integer`.

The replacement method changes the `"dim"` attribute (provided the new value is compatible) and removes any `"dimnames"` *and* `"names"` attributes.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

`ncol`, `nrow` and `dimnames`.

Examples

```
x <- 1:12 ; dim(x) <- c(3,4)
x

# simple versions of nrow and ncol could be defined as follows
nrow0 <- function(x) dim(x)[1]
ncol0 <- function(x) dim(x)[2]
```

`extend_standard_soc1980_codes` *Extended SOC 1980 codes*

Description

Takes valid 1980 standardized codes (the ones in the book) and extends them so that unit codes are always the most detailed (even if it is exactly the same as the parent code.)

Usage

```
extend_standard_soc1980_codes(codes)
```

Arguments

`codes`	The codes we are extending

Value

extended codes.

`filter.xwalk` *Using dplyr verbs with crosswalks*

Description

These methods allow you to work with the underlying data within a crosswalk as if was a tibble.

Usage

```
## S3 method for class 'xwalk'
filter(.data, ..., .by = NULL, .preserve = FALSE)

## S3 method for class 'xwalk'
arrange(.data, ..., .by_group = FALSE)

## S3 method for class 'xwalk'
as_tibble(
```

```
  x,
  ...,
  .rows = NULL,
  .name_repair = c("check_unique", "unique", "universal", "minimal"),
  rownames = pkgconfig::get_config("tibble::rownames", NULL)
)
```

Arguments

.data	The crosswalk
...	<data-masking> Expressions that return a logical value, and are defined in terms of the variables in .data. If multiple expressions are included, they are combined with the & operator. Only rows for which all conditions evaluate to TRUE are kept.
.by	**[Experimental]** <tidy-select> Optionally, a selection of columns to group by for just this operation, functioning as an alternative to group_by(). For details and examples, see ?dplyr_by.
.preserve	Relevant when the .data input is grouped. If .preserve = FALSE (the default), the grouping structure is recalculated based on the resulting data, otherwise the grouping is kept as is.
.by_group	If TRUE, will sort first by grouping variable. Applies to grouped data frames only.
x	A data frame, list, matrix, or other object that could reasonably be coerced to a tibble.
.rows	The number of rows, useful to create a 0-column tibble or just as an additional check.
.name_repair	Treatment of problematic column names: • "minimal": No name repair or checks, beyond basic existence, • "unique": Make sure names are unique and not empty, • "check_unique": (default value), no name repair, but check they are unique, • "universal": Make the names unique and syntactic • "unique_quiet": Same as "unique", but "quiet" • "universal_quiet": Same as "universal", but "quiet" • a function: apply custom name repair (e.g., .name_repair = make.names for names in the style of base R). • A purrr-style anonymous function, see rlang::as_function() This argument is passed on as repair to vctrs::vec_as_names(). See there for more details on these terms and the strategies used to enforce them.
rownames	How to treat existing row names of a data frame or matrix: • NULL: remove row names. This is the default. • NA: keep row names.

- A string: the name of a new column. Existing rownames are transferred into this column and the `row.names` attribute is deleted. No name repair is applied to the new column name, even if `x` already contains a column of that name. Use `as_tibble(rownames_to_column(...))` to safeguard against this case.

Read more in rownames.

`format.codingsystem` *formats a codingsystem*

Description

formats a codingsystem

Usage

```
## S3 method for class 'codingsystem'
format(x, ...)
```

Arguments

`x`	- the codingsystem
`...`	not currently used

Value

a formatted character vector

`get_codes` *Get a list of codes from a coding system*

Description

Get a list of codes from a coding system

Usage

```
get_codes(.codingsystem)
```

Arguments

`.codingsystem`	either a codingsystem or a tibble that has a a column named "code".

Value

a vector of codes

`head.codingsystem` *Return the First or Last Parts of an Object*

Description

Returns the first or last parts of a vector, matrix, table, data frame or function. Since `head()` and `tail()` are generic functions, they may also have been extended to other classes.

Usage

```
## S3 method for class 'codingsystem'
head(x, ...)
```

Arguments

`x`	an object
`...`	arguments to be passed to or from other methods.

Details

For vector/array based objects, `head()` (`tail()`) returns a subset of the same dimensionality as `x`, usually of the same class. For historical reasons, by default they select the first (last) 6 indices in the first dimension ("rows") or along the length of a non-dimensioned vector, and the full extent (all indices) in any remaining dimensions. `head.matrix()` and `tail.matrix()` are exported.

The default and array(/matrix) methods for `head()` and `tail()` are quite general. They will work as is for any class which has a `dim()` method, a `length()` method (only required if `dim()` returns `NULL`), and a `[` method (that accepts the `drop` argument and can subset in all dimensions in the dimensioned case).

For functions, the lines of the deparsed function are returned as character strings.

When `x` is an array(/matrix) of dimensionality two and more, `tail()` will add dimnames similar to how they would appear in a full printing of `x` for all dimensions `k` where `n[k]` is specified and non-missing and `dimnames(x)[[k]]` (or `dimnames(x)` itself) is `NULL`. Specifically, the form of the added dimnames will vary for different dimensions as follows:

`k=1` **(rows):** `"[n,]"` (right justified with whitespace padding)

`k=2` **(columns):** `"[,n]"` (with *no* whitespace padding)

`k>2` **(higher dims):** `"n"`, i.e., the indices as *character* values

Setting `keepnums = FALSE` suppresses this behaviour.

As `data.frame` subsetting ('indexing') keeps `attributes`, so do the `head()` and `tail()` methods for data frames.

Value

An object (usually) like `x` but generally smaller. Hence, for `array`s, the result corresponds to `x[..., drop=FALSE]`. For `ftable` objects `x`, a transformed `format(x)`.

Note

For array inputs the output of `tail` when `keepnums` is `TRUE`, any dimnames vectors added for dimensions `>2` are the original numeric indices in that dimension *as character vectors*. This means that, e.g., for 3-dimensional array `arr`, `tail(arr, c(2,2,-1))[ , , 2]` and `tail(arr, c(2,2,-1))[ , , "2"]` may both be valid but have completely different meanings.

Author(s)

Patrick Burns, improved and corrected by R-Core. Negative argument added by Vincent Goulet. Multi-dimension support added by Gabriel Becker.

Examples

```
head(letters)
head(letters, n = -6L)

head(freeny.x, n = 10L)
head(freeny.y)

head(iris3)
head(iris3, c(6L, 2L))
head(iris3, c(6L, -1L, 2L))

tail(letters)
tail(letters, n = -6L)

tail(freeny.x)
## the bottom-right "corner" :
tail(freeny.x, n = c(4, 2))
tail(freeny.y)

tail(iris3)
tail(iris3, c(6L, 2L))
tail(iris3, c(6L, -1L, 2L))

## iris with dimnames stripped
a3d <- iris3 ; dimnames(a3d) <- NULL
tail(a3d, c(6, -1, 2)) # keepnums = TRUE is default here!
tail(a3d, c(6, -1, 2), keepnums = FALSE)

## data frame w/ a (non-standard) attribute:
treeS <- structure(trees, foo = "bar")
(n <- nrow(treeS))
stopifnot(exprs = { # attribute is kept
    identical(htS <- head(treeS), treeS[1:6, ])
    identical(attr(htS, "foo") , "bar")
    identical(tlS <- tail(treeS), treeS[(n-5):n, ])
    ## BUT if I use "useAttrib(.)", this is *not* ok, when n is of length 2:
    ## --- because [i,j]-indexing of data frames *also* drops "other" attributes ..
    identical(tail(treeS, 3:2), treeS[(n-2):n, 2:3] )
})
```

```
tail(library) # last lines of function

head(stats::ftable(Titanic))

## 1d-array (with named dim) :
a1 <- array(1:7, 7); names(dim(a1)) <- "O2"
stopifnot(exprs = {
  identical( tail(a1, 10), a1)
  identical( head(a1, 10), a1)
  identical( head(a1, 1), a1 [1 , drop=FALSE] ) # was a1[1] in R <= 3.6.x
  identical( tail(a1, 2), a1[6:7])
  identical( tail(a1, 1), a1 [7 , drop=FALSE] ) # was a1[7] in R <= 3.6.x
})
```

`is.codingsystem` *checks if an object is a coding system*

Description

Is this object a coding system

Usage

```
is.codingsystem(x)
```

Arguments

x	object to test

`is.xwalk` *checks if an object is a crosswalk*

Description

Is this object a crosswalk

Usage

```
is.xwalk(x)
```

Arguments

x	A crosswalk of class xwalk

is_url — *Check if a value is a url by looking for the http(s):// .Works with vectors...*

Description

Check if a value is a url by looking for the http(s):// .Works with vectors...

Usage

```
is_url(x)
```

Arguments

x	String to check

Value

logical vector TRUE if the x is a url False otherwise

is_valid — *Check if a set of codes are valid for a coding system*

Description

Check if a set of codes are valid for a coding system

Usage

```
is_valid(code, system)
```

Arguments

code	vector of codes to check
system	the coding system

Value

boolean vector corresponding to whether the codes are in the coding system

level *Get the code Level*

Description

Gets the levels for a vector of codes from a codingsystem The type returned depends on the data.

Usage

```
level(data, codes)

## S3 method for class 'codingsystem'
level(data, codes)
```

Arguments

data	- a codingsystem
codes	- a vector of codes to check

Value

a vector of Levels

Examples

```
level(soc1980_all,"99-99") # "division"
level(soc2010_all,c("11-1011","11-2010")) # c(6,5)
```

load_socassign_db *load_socassign_db*

Description

load the data from a SOCAssign SQLite database

Usage

```
load_socassign_db(fname, addSrc = FALSE)
```

Arguments

fname	the SOCAssign db file
addSrc	should I add the file name as an src column

Value

tibble with coder results

lookup_code *Look up code*

Description

Look up code

Usage

```
lookup_code(x, system)
```

Arguments

x	list of codes to lookup
system	the coding system

Value

a vector of titles for the codes

make_code_str *convert a list column of codes to vector of string for display*

Description

convert a list column of codes to vector of string for display

Usage

```
make_code_str(x)
```

Arguments

x	codes column

Value

a vector a string concatenating all the codes

Examples

```
df <- tibble::tibble(soc2010_codes = list(c("11-1011","11-1021"),c("11-1000")))
df <- dplyr::mutate(df,code_str=make_code_str(soc2010_codes))
```

name *Returns the user assigned name of the coding system*

Description

Returns the user assigned name of the coding system

Usage

```
name(system)
```

Arguments

system	coding system

Value

the name of the coding system (may be blank)

noc2011_4digit *noc2011 4 digit classification system*

Description

Canadian 4 digit National Occupational Classification (NOC) 2011

Usage

```
noc2011_4digit
```

Format

noc_code a 4 digit code formated like '0011', be careful must be a string not an integer

title a short definition of the code

Source

https://danielruss.github.io/codingsystems/noc_2011_4d.csv

https://www.statcan.gc.ca/eng/subjects/standard/noc/2011/index

noc2011_all *noc2011 4 digit classification system*

Description

Canadian 4 digit National Occupational Classification (NOC) 2011

Usage

```
noc2011_all
```

Format

code a 1-4-digit code formated like '0011', be careful must be a string not an integer

title a short definition of the code

Level Unofficial name for the level in the hierarchy (number of digits) for the code, 1, 2, 3, or 4

Hierarchical_structure Official name for the level in the hierarchy

noc1d the 1-digit noc code associated with the code

noc2d the 2-digit noc code associated with the code, is NA for 1-digit codes

noc3d the 3-digit noc code associated with the code, is NA for 1- or 2-digit codes

noc4d the 4-digit noc code associated with the code, is NA for 1-, 2-, or 3-digit codes

Source

https://danielruss.github.io/codingsystems/noc_2011_4d.csv

https://www.statcan.gc.ca/eng/subjects/standard/noc/2011/index

print.codingsystem *prints a codingsystem*

Description

prints a codingsystem

Usage

```
## S3 method for class 'codingsystem'
print(x, ...)
```

Arguments

x	- the codingsystem
...	parameter for format, not currently used

`select.codingsystem` *Use Coding system with dplyr*

Description

These methods allow you to use the codingsystem like a tibble. When using select, make sure you keep the code/title or else you can break the functionality of the codingsystem.

Usage

```
## S3 method for class 'codingsystem'
select(.data, ...)

## S3 method for class 'codingsystem'
filter(.data, ..., .by = NULL, .preserve = FALSE, name = NULL)

## S3 method for class 'codingsystem'
mutate(.data, ...)

## S3 method for class 'codingsystem'
arrange(.data, ..., .by_group = FALSE)

## S3 method for class 'codingsystem'
as_tibble(x, ..., .rows = NULL, .name_repair = NULL, rownames = NULL)

## S3 method for class 'codingsystem'
count(x, ..., wt = NULL, sort = FALSE, name = NULL)
```

Arguments

`.data`	the coding system
`...`	parts of the coding system
`.by`	passed to dplyr::filter
`.preserve`	passed to dplyr::filter
`name`	name for the filtered coding system
`.by_group`	passed to dplyr::arrange
`x`	the coding system
`.rows`	passed to dplyer::as_tibble
`.name_repair`	passed to dplyer::as_tibble
`rownames`	passed to dplyer::as_tibble

Value

a new codingsystem

soc1980_all *SOC 1980 complete classification system*

Description

US SOC 1980 classification system

Usage

```
soc1980_all
```

Format

soc1980_code the n-digit soc 1980

title a short definition of the code

Source

https://danielruss.github.io/codingsystems/soc1980.csv

soc1980_detailed *Detailed SOC 1980 classification system*

Description

The US SOC 1980 classification system can have higher level (major or minor codes) codes without any children. This data contains all the most detailed codes regardless of the code level.

Usage

```
soc1980_detailed
```

Format

soc1980_code the soc 1980 code

title a short definition of the code

Level the level of the soc 1980 code

parent the parent of the soc 1980 code, note: at the division level, the parent is 0000

division for any soc 1980 code, what is the division code

major for any soc 1980 code, what is the major code. Is NA for division codes.

minor for any soc 1980 code, what is the minor code. Is NA for division and major codes.

unit for any soc 1980 code, what is the unit code. Is NA for non-unit codes.

Source

https://danielruss.github.io/codingsystems/soc1980_most_detailed.csv

soc1980_extended *Extended SOC 1980 complete classification system*

Description

The US SOC 1980 classification system can have higher level (major or minor codes) codes without any children. We extended the SOC 1980 classification system to require all major codes (2-digit code) to have at least 1 minor code (3-digit code), and every minor codes to have at least 1 unit code (4-digit code). The most detailed code is now always a unit code.

Usage

```
soc1980_extended
```

Format

soc1980_code the soc 1980 code

title a short definition of the code

Level the level of the soc 1980 code

parent the parent of the soc 1980 code, note: at the division level, the parent is 0000

division for any soc 1980 code, what is the division code

major for any soc 1980 code, what is the major code. Is NA for division codes.

minor for any soc 1980 code, what is the minor code. Is NA for division and major codes.

unit for any soc 1980 code, what is the unit code. Is NA for non-unit codes.

Source

https://danielruss.github.io/codingsystems/soc_1980_extended.csv

soc2010_6digit *soc2010 6 digit classification system*

Description

Downloaded by Daniel Russ

Usage

```
soc2010_6digit
```

Format

code a 6 digit code formated like '11-1011'

title a short definition of the code

Source

https://danielruss.github.io/codingsystems/soc_2010_6digit.csv

https://www.bls.gov/soc/2010/2010_major_groups.htm

`soc2010_all` *Complete SOC 2010 classification system*

Description

The complete US SOC 2010 classification system. This data contains all the codes regardless of the code level.

Usage

```
soc2010_all
```

Format

code the soc 2010 code

title a short definition of the code

Level The number of significant digits in the code

Hierarchical_structure The name of the level

parent the parent of the soc code, note: 2 digit soc code dont have parents

soc2d for any soc code, what is the 2-digit code

soc3d for any soc code, what is the 3-digit code. Is NA for 2-digit codes.

soc5d for any soc code, what is the 5-digit code. Is NA for 2- and 3-digit codes.

soc6d for any soc code, what is the 6-digit code. Is NA for 2-, 3-, and 5-digit codes.

Source

https://danielruss.github.io/codingsystems/soc2010_all.csv

soc2018_all *Complete SOC 2018 classification system*

Description

The complete US SOC 2018 classification system. This data contains all the codes regardless of the code level.

Usage

```
soc2018_all
```

Format

code the soc 2018 code

title a short definition of the code

Level The number of significant digits in the code

Hierarchical_structure The name of the level

parent the parent of the soc code, note: 2 digit soc code dont have parents

soc2d for any soc code, what is the 2-digit code

soc3d for any soc code, what is the 3-digit code. Is NA for 2-digit codes.

soc5d for any soc code, what is the 5-digit code. Is NA for 2- and 3-digit codes.

soc6d for any soc code, what is the 6-digit code. Is NA for 2-, 3-, and 5-digit codes.

Source

https://danielruss.github.io/codingsystems/soc2018_all.csv

split_data *Split data*

Description

A simple deterministic mechanism for splitting data into training, development, and test data based on the MD5 hash of a unused string parameters.

Usage

```
split_data(x, pTrain = 0.9, pDev = 0.09, pTest = 0.01)
```

Arguments

x	unused string data used to split the data
pTrain	approximate percent of the training split
pDev	approximate percent of the development split
pTest	approximate percent of the test split

Value

a vector of factors (Train,Dev,Test) denoting the data split

Examples

```
split_data(rownames(mtcars))
```

---

standardize_soc1980_codes

Standardize US SOC 1980 codes

Description

US SOC 1980 codes are often written in none stand form (e.g 4600 instead of 46-47). This function attempt to standardize some of the ways SOC 1980 codes are written.

Usage

```
standardize_soc1980_codes(codes)
```

Arguments

codes	vector of US SOC 1980 codes

Details

the function trims leading and trailing zeros ("up to 2 trailing zero - 20 is a valid soc code)

Value

standardized US SOC 1980 codes

Examples

```
standardize_soc1980_codes(c("2000",'7600'))
```

`tail.codingsystem` *Return the First or Last Parts of an Object*

Description

Returns the first or last parts of a vector, matrix, table, data frame or function. Since `head()` and `tail()` are generic functions, they may also have been extended to other classes.

Usage

```
## S3 method for class 'codingsystem'
tail(x, ...)
```

Arguments

`x`	an object
`...`	arguments to be passed to or from other methods.

Details

For vector/array based objects, `head()` (`tail()`) returns a subset of the same dimensionality as `x`, usually of the same class. For historical reasons, by default they select the first (last) 6 indices in the first dimension ("rows") or along the length of a non-dimensioned vector, and the full extent (all indices) in any remaining dimensions. `head.matrix()` and `tail.matrix()` are exported.

The default and array(/matrix) methods for `head()` and `tail()` are quite general. They will work as is for any class which has a `dim()` method, a `length()` method (only required if `dim()` returns `NULL`), and a `[` method (that accepts the `drop` argument and can subset in all dimensions in the dimensioned case).

For functions, the lines of the deparsed function are returned as character strings.

When `x` is an array(/matrix) of dimensionality two and more, `tail()` will add dimnames similar to how they would appear in a full printing of `x` for all dimensions `k` where `n[k]` is specified and non-missing and `dimnames(x)[[k]]` (or `dimnames(x)` itself) is `NULL`. Specifically, the form of the added dimnames will vary for different dimensions as follows:

`k=1` **(rows):** `"[n,]"` (right justified with whitespace padding)

`k=2` **(columns):** `"[,n]"` (with *no* whitespace padding)

`k>2` **(higher dims):** `"n"`, i.e., the indices as *character* values

Setting `keepnums = FALSE` suppresses this behaviour.

As `data.frame` subsetting ('indexing') keeps `attributes`, so do the `head()` and `tail()` methods for data frames.

Value

An object (usually) like `x` but generally smaller. Hence, for `array`s, the result corresponds to `x[..., drop=FALSE]`. For `ftable` objects `x`, a transformed `format(x)`.

Note

For array inputs the output of `tail` when `keepnums` is `TRUE`, any dimnames vectors added for dimensions `>2` are the original numeric indices in that dimension *as character vectors*. This means that, e.g., for 3-dimensional array `arr`, `tail(arr, c(2,2,-1))[ , , 2]` and `tail(arr, c(2,2,-1))[ , , "2"]` may both be valid but have completely different meanings.

Author(s)

Patrick Burns, improved and corrected by R-Core. Negative argument added by Vincent Goulet. Multi-dimension support added by Gabriel Becker.

Examples

```
head(letters)
head(letters, n = -6L)

head(freeny.x, n = 10L)
head(freeny.y)

head(iris3)
head(iris3, c(6L, 2L))
head(iris3, c(6L, -1L, 2L))

tail(letters)
tail(letters, n = -6L)

tail(freeny.x)
## the bottom-right "corner" :
tail(freeny.x, n = c(4, 2))
tail(freeny.y)

tail(iris3)
tail(iris3, c(6L, 2L))
tail(iris3, c(6L, -1L, 2L))

## iris with dimnames stripped
a3d <- iris3 ; dimnames(a3d) <- NULL
tail(a3d, c(6, -1, 2)) # keepnums = TRUE is default here!
tail(a3d, c(6, -1, 2), keepnums = FALSE)

## data frame w/ a (non-standard) attribute:
treeS <- structure(trees, foo = "bar")
(n <- nrow(treeS))
stopifnot(exprs = { # attribute is kept
    identical(htS <- head(treeS), treeS[1:6, ])
    identical(attr(htS, "foo") , "bar")
    identical(tlS <- tail(treeS), treeS[(n-5):n, ])
    ## BUT if I use "useAttrib(.)", this is *not* ok, when n is of length 2:
    ## --- because [i,j]-indexing of data frames *also* drops "other" attributes ..
    identical(tail(treeS, 3:2), treeS[(n-2):n, 2:3] )
})
```

```
tail(library) # last lines of function

head(stats::ftable(Titanic))

## 1d-array (with named dim) :
a1 <- array(1:7, 7); names(dim(a1)) <- "O2"
stopifnot(exprs = {
  identical( tail(a1, 10), a1)
  identical( head(a1, 10), a1)
  identical( head(a1, 1), a1 [1 , drop=FALSE] ) # was a1[1] in R <= 3.6.x
  identical( tail(a1, 2), a1[6:7])
  identical( tail(a1, 1), a1 [7 , drop=FALSE] ) # was a1[7] in R <= 3.6.x
})
```

`to_level` *to_level*

Description

A utility function for converting occupational codes to higher levels in the hierarchy.

Usage

```
to_level(codingsystem, level)
```

Arguments

`codingsystem`	The coding system we are using
`level`	The level in the coding system we want. Should be a column name in the codingsystem table.

Value

a function that converts a vector of codes from a lower level to a the level input.

Examples

```
to_soc2010_2d <- to_level(soc2010_all, soc2d)
to_soc2010_2d(c("11-1011","15-1110"))
```

to_list_column *Create a list column from multiple columns*

Description

This function replaces a set of input columns that you pass in with a list column containing the values of input column on a row-by-row basis.

Usage

```
to_list_column(df, colname, ...)
```

Arguments

df	the data frame you are modifying
colname	the name of the new column
...	the columns you are combining into a list column

Value

The original data frame with a new list column ‘colname‘ replacing the columns given

Examples

```
df <- tibble::tibble(a_1=1:3,a_2=2:4,a_3=3:5,b=4:6) |> to_list_column(a,a_1,a_2,a_3)
```

valid_code *Is valid code*

Description

check whether a code is valid

Usage

```
valid_code(codeList)

is_valid_6digit_soc2010(code)

is_valid_soc1980(code)

is_most_detailed_soc1980(code)

is_valid_extended_soc1980(code)

is_most_detailed_extended_soc1980(code)
```

Arguments

codeList	a vector of valid codes
code	codes to compare

Details

valid_code is a functional that create a function that check if a vector of codes is valid

is_valid_4digit_soc1980, is_valid_6digit_soc2010 and is_valid_4digit_noc2011 were made using valid_code functional.

Value

valid_code returns a function. The functions (e.g. is_valid_soc2010) take a code or a vector of codes and returns a logic vector representing if the codes are valid.

See Also

[standardize_soc1980_codes()]

Examples

```
is_valid_toy <- valid_code(c("A","B","C"))
is_valid_toy(c("X","A","Z","B"))
```

---

xwalk *xwalk class constructor*

Description

takes a data frame (the crosswalk) and which columns are the codes and titles and create an xwalk object that can perform crosswalks...

Usage

```
xwalk(
  dta,
  codes1,
  titles1,
  codes2,
  titles2,
  col_types = ifelse(grepl("\\.xlsx?$", dta), "text", "c"),
  ...
)
```

Arguments

dta	the data frame of the crosswalk, or the filename/URL of a csv crosswalk file or the filename of an excel file.
codes1	Codes for the (Default) input coding system for crosswalking
titles1	Titles for the (Default) input coding system.
codes2	Codes for the (Default) output coding system for crosswalking
titles2	Titles for the (Default) output coding system.
col_types	set the default col_type parameter for read_csv/read_excel
...	additional parameters passed to read_csv

---

xwalk_entropy *Calculates the Shannon entropy for a Crosswalk*

Description

The more potential codes that a crosswalk will allow an intial code to become, the higher the entropy. The entropy (S) is given by

$$S = -\Sigma\Sigma plogp$$

where p = 1/n and n is the number of potential codes. a single code can map to, natural logs are used in the calculation. The inner summation can be is the sum of n iteration of 1/n, so the equation can be simplified to

$$S = -\Sigma logp$$

Usage

```
xwalk_entropy(x)
```

Arguments

x	The crosswalk

Value

the entropy

Index