Package ‘relatable’

October 14, 2022

Type Package

Title Functions for Mapping Key-Value Pairs, Many-to-Many, One-to-Many, and Many-to-One Relations

Version 1.0.0

Description Functions to safely map from a vector of keys to a vector of values, determine properties of a given relation, or ensure a relation conforms to a given type, such as many-to-many, one-to-many, injective, surjective, or bijective. Permits default return values for use similar to a vectorised switch statement, as well as safely handling large vectors, NAs, and duplicate mappings.

Depends R (>= 3.4)

Imports compare (>= 0.2-6)

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 6.0.1

Suggests testthat, knitr, rmarkdown, tibble

VignetteBuilder knitr

URL https://github.com/domjarkey/relatable

NeedsCompilation no

Author Dominic Jarkey [aut, cre]

Maintainer Dominic Jarkey <dominic.jarkey@gmail.com>

Repository CRAN

Date/Publication 2018-01-30 16:56:41 UTC

R topics documented:

  elements .......................................................... 2
  relate ........................................................... 2

Index 8
elements

Data from the periodic table of elements

Description

A dataset containing atomic numbers, chemical symbols, and names of 118 elements.

Usage

elements

Format

A data frame with 118 rows and 3 variables:

- **Z** Atomic number
- **Symbol** Chemical symbol
- **Name** Name of element

Source

https://en.wikipedia.org/wiki/Symbol_(chemistry)

relate

Map inputs from a vector of keys to a vector of values.

Description

relate returns a vector \( Y = F(X) \) where \( F \) maps each element of input vector \( X \) from its position in vector \( A \) to its corresponding position in vector \( B \). Can be applied as a vectorised key-value dictionary with an optional default return value. Additional options restrict mapping types so relation \( F \) must be a function, injective, surjective, etc.

relation returns a reusable function \( F \) that performs the same operation as relate. In addition to providing a reusable function, if handle_duplicate_mappings = TRUE, relation checks for and eliminates duplicate mappings that would be invalid inputs for relate. If report_properties = TRUE, relation also prints the restrictions the mapping from \( A \) to \( B \) conforms to.
Usage

relate(X, A, B, default = NA, atomic = TRUE, named = FALSE,
allow_default = TRUE, heterogeneous_outputs = FALSE,
handle_duplicate_mappings = FALSE, report_properties = FALSE,
relation_type = "func", restrictions = list(),
map_error_response = "warn")

relation(A, B, default = NA, atomic = TRUE, named = FALSE,
allow_default = TRUE, heterogeneous_outputs = FALSE,
handle_duplicate_mappings = FALSE, report_properties = FALSE,
relation_type = "func", restrictions = list(),
map_error_response = "warn")

Arguments

X  A vector of inputs
A  A vector possible inputs ordered to correspond to desired outputs given by B.
B  A vector possible outputs ordered to correspond to each input to the relation
given by A.
default The default value to return if the value of \( F(x) \) is undefined.
atomic If TRUE, the return vector \( Y \) will be atomic; If TRUE \( Y \) will be a list vector. To
allow for multiple outputs from a single input, atomic must be set to FALSE if
relation_type = "many_to_many" or "one_to_many", or if relation_type
= NULL and max_one_y_per_x = FALSE is an element of restrictions list.
named The elements of the returned vector \( Y \) will be named by to their corresponding
inputs in X.
allow_default If TRUE, the provided default will be returned when \( F(x) \) is undefined; other-
wise invalid mappings will return an error determined by the map_error_response
argument.
heterogeneous_outputs
By default, elements \( y \) of the output vector \( Y \) will be returned as atomic vectors.
In many-to-many and one-to-many relations, if the elements in the codomain
are not all of the same type, this will coerce outputs to the same type. Set
heterogeneous_outputs = TRUE to return each \( y \) as a list vector. This will
avoid coercion of individual outputs to the same type, but may also result in
messy nested list vectors.
handle_duplicate_mappings
If TRUE, each possible input/output pair in the returned function \( F \) for duplicate
mappings and removes them. This may increase the runtime for larger map-
ings, but only for the first instance of relation. The function returned by
relation does not need to re-check these properties, so will run more quickly.
If handle_duplicate_mappings = FALSE, duplicate mappings from A to B in
relate or relation will return multiple instances of the same output. See Ex-
amples.
report_properties
If TRUE, relation reports which restrictions \( F \) conforms to. See Details.
relation_type  Ensure that the relation is restricted to a certain type, e.g. "bijection". See Details.

restrictions  A named list of logicals imposing constraints on the relation. These will only be used if relation_type is NULL. See Details.

map_error_response  How to deal with mapping errors caused by violated restrictions. Takes values "ignore", "warn", or "throw".

Details

relate returns vector of outputs \( Y = F(X) \) where the \( F \) is a relation defined by the collection of ordered pairs \((a_i, b_i)\) where \(a_i, b_i\) are the \(i\)th elements of \(A\) and \(B\) respectively. If \(F(x)\) is undefined because \(x\) is not in \(A\) or it does not map to an element of \(B\), relate will either return default if allow_default = TRUE. Otherwise the function will throw an error.

The relation \(F\) can be restricted so it conforms to a particular type specified, for example relation_type = "one_to_many". If relation_type = NULL, the properties are determined by restrictions specified with a named list, for example restrictions = list(min_one_y_per_x = TRUE). For all relations where min_one_y_per_x = FALSE, only a list vector can be returned, so an error will be thrown if atomic = TRUE. If \(A\) and \(B\) do not produce a relation that conforms to the specified type or restrictions, the value of map_error_response will determine whether the relate ignores the error, reports it, or throws it. The full list of restrictions and relation types are listed below:

Restrictions

NB: 1) The restrictions argument is only used if relation_type = NULL; 2) If relation is allowed to return multiple values, i.e. max_one_y_per_x = FALSE, then atomic must be set to FALSE, otherwise an error will be throw; 3) All unspecified restrictions are assumed false, e.g. restrictions = list() is equivalent to restrictions = list("min_one_y_per_x" = FALSE,"min_one_x_per_y" = FALSE,"max_one_y_per_x" = FALSE,"max_one_x_per_y" = FALSE)

min_one_y_per_x  Guarantees at least one \(y = F(x)\) in \(B\) exists for each \(x\) in \(A\). Returns an error if \(B\) is longer than \(A\).

min_one_x_per_y  Guarantees at least one \(x\) in \(A\) exists for each \(y\) in \(B\) such that \(y = F(x)\). Returns an error if \(A\) is longer than \(B\).

max_one_y_per_x  Guarantees no more than one \(y = F(x)\) in \(B\) exists for each \(x\) in \(A\). Returns an error if \(A\) contains duplicate elements.

max_one_x_per_y  Guarantees no more than one \(x\) in \(A\) exists for each \(y\) in \(B\) such that \(y = F(x)\). Returns an error if \(B\) contains duplicate elements.

Relation types

relation_type = "one_to_one"  One-to-one relations require that each element in the domain to map to at most one element in the codomain, and each element of the codomain to map from the only one element in the domain. There may still be elements in \(A\) that do not have a mapping to an element in \(B\), and vice versa. This is equivalent to restrictions = list("min_one_y_per_x" = FALSE,"min_one_x_per_y" = FALSE,"max_one_y_per_x" = TRUE,"max_one_x_per_y" = TRUE)

relation_type = "many_to_many"  Many-to-many relations allow multiple elements in the domain to map to the same element of the codomain, and multiple elements of the codomain to
map from the same element of the domain. This is equivalent to
restrictions = list("min_one_y_per_x" = FALSE,
"min_one_x_per_y" = FALSE,
"max_one_y_per_x" = FALSE,
"max_one_x_per_y" = FALSE)

relation_type = "one_to_many"  One-to-many relations require each element of the domain to
map to a distinct set of one or more elements in the codomain. This is equivalent to
restrictions = list("min_one_y_per_x" = FALSE,
"min_one_x_per_y" = FALSE,
"max_one_y_per_x" = FALSE,
"max_one_x_per_y" = TRUE)

relation_type = "many_to_one"  Many-to-one relations allows sets of one or more elements in
the domain to map to the same distinct element in the codomain. This is equivalent to
restrictions = list("min_one_y_per_x" = FALSE,
"min_one_x_per_y" = FALSE,
"max_one_y_per_x" = TRUE,
"max_one_x_per_y" = FALSE)

relation_type = "func"  Functions map each element in the domain to exactly one element in the
codomain. This is equivalent to
restrictions = list("min_one_y_per_x" = TRUE,
"min_one_x_per_y" = FALSE,
"max_one_y_per_x" = TRUE,
"max_one_x_per_y" = FALSE)

relation_type = "injection"  A function is injective if every element of the domain maps to a
unique element of the codomain. This is equivalent to
restrictions = list("min_one_y_per_x" = TRUE,
"min_one_x_per_y" = FALSE,
"max_one_y_per_x" = TRUE,
"max_one_x_per_y" = TRUE)

relation_type = "surjection"  A function is surjective if every element of the codomain maps
from an element of the domain. This is equivalent to
restrictions = list("min_one_y_per_x" = TRUE,
"min_one_x_per_y" = TRUE,
"max_one_y_per_x" = TRUE,
"max_one_x_per_y" = FALSE)

relation_type = "bijection"  A function is bijective if it is both injective and surjective, i.e. a
complete one-to-one mapping. This is equivalent to
restrictions = list("min_one_y_per_x" = TRUE,
"min_one_x_per_y" = TRUE,
"max_one_y_per_x" = TRUE,
"max_one_x_per_y" = TRUE)

Examples

## Map from one vector to another
relate(c("a", "e", "i", "o", "u"), letters, LETTERS)
# [1] "A" "E" "I" "O" "U"
## or
caps <- relation(letters, LETTERS)
caps("t")
# [1] "T"
caps(c("p", "q", "r"))
# [1] "P" "Q" "R"

## Create a new column in a data frame
df <- data.frame(
  name = c("Alice", "Bob", "Charlotte", "Dan", "Elise", "Frank"),
  position = c("right", "lean-left", "left", "left", "lean-right", "no response"))
positions <- c("left", "lean-left", "independent", "lean-right", "right")
colours <- c("darkblue", "lightblue", "green", "lightred", "darkred")
df$colour <- relate(df$position, positions, colours, default = "gray")
df
# name position colour
# 1  Alice  right  darkred
# 2   Bob  lean-left  lightblue
# 3 Charlotte   left  darkblue
## Authors have a many-to-many relation with books:
## a book can have multiple authors and authors can write multiple books

```r
my_library <- data.frame(
  author = c(  
    "Arendt",  
    "Austen-Smith",  
    "Austen-Smith",  
    "Austen-Smith",  
    "Banks",  
    "Banks",  
    "Camus",  
    "Camus",  
    "Arendt",  
    "Dryzek",  
    "Dunleavy"  
  ),
  work = c(  
    "The Human Condition",  
    "Social Choice and Voting Models",  
    "Information Aggregation, Rationality, and the Condorcet Jury Theorem",  
    "Positive Political Theory I",  
    "Information Aggregation, Rationality, and the Condorcet Jury Theorem",  
    "Positive Political Theory I",  
    "The Myth of Sisyphus",  
    "The Rebel",  
    "The Origins of Totalitarianism",  
    "Theories of the Democratic State",  
    "Theories of the Democratic State"  
  ),
  stringsAsFactors = FALSE  
)
relate(  
  X = c("Arendt", "Austen-Smith", "Banks", "Dryzek", "Dunleavy"),  
  A = my_library$author,  
  B = my_library$work,  
  atomic = FALSE,  
  named = TRUE,  
  relation_type = "many_to_many"
)
# $Arendt
# [1] "The Human Condition"  "The Origins of Totalitarianism"
#
# $Austen-Smith
# [1] "Social Choice and Voting Models"
# [2] "Information Aggregation, Rationality, and the Condorcet Jury Theorem"
# [3] "Positive Political Theory I"
#
# $Banks
# [1] "Information Aggregation, Rationality, and the Condorcet Jury Theorem"
```
# [2] "Positive Political Theory I"
# $Dryzek
# [1] "Theories of the Democratic State"
# $Dunleavy
# [1] "Theories of the Democratic State"

## Duplicate mappings will return multiple copies by default:
relate(
  X = 1:3,
  A = c(1, 2, 2, 3, 4, 5),
  B = c('a', 'b', 'b', 'c', 'd', 'e'),
  relation_type = "many_to_many",
  atomic = FALSE
)

## Use handle_duplicate_mappings = TRUE to ignore these and avoid mapping errors.
nums_to_letters <- relation(
  A = c(1, 2, 3, 4, 5),
  B = c('a', 'b', 'b', 'c', 'd', 'e'),
  relation_type = "bijection",
  handle_duplicate_mappings = TRUE
)
nums_to_letters(X = c(1, 2, 3))

## Use relation with report_properties = TRUE to determine the properties of specified relation
domain <- -3:3
image <- domain^2
relation(domain, image, report_properties = TRUE)
# Relation properties:
# min_one_y_per_x min_one_x_per_y max_one_y_per_x max_one_x_per_y
# TRUE TRUE FALSE FALSE
Index

* datasets
  elements, 2

elements, 2

relate, 2

relation (relate), 2