Package: respectables (via r-universe)

October 1, 2024

Type Package

Title Create Relationally-Specified Multi-Table Datasets

Description This package provides a general framework and utility functions for simulating or synthesizing database-like multiple-table datasets. This is done by defining and then applying data recipes, which support the simulation or derivation of multiple variables jointly and/or conditional on the value of other already existing or recipe-specified variables. It also supports the simulation or augmentation of data which encodes a 1-to-many relationship with a foreign-key in an existing table.

Version 0.0.4.9000

License Artistic-2.0

Encoding UTF-8

RoxygenNote 7.1.2

Copyright F. Hoffmann-La Roche Ltd

Depends tibble

Imports stats, methods, tibble

Suggests knitr, dplyr, ggplot2, tinytest, dm, rmarkdown

VignetteBuilder knitr

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

RemoteUrl https://github.com/Roche/respectables

RemoteRef HEAD

RemoteSha 64d3e5409dca2206a71a71d09212768732f89f73

Contents

`gen_data_db` *Generate full db from a Cookbook of Recipes*

Description

Generate full db from a Cookbook of Recipes

Usage

```
gen_data_db(
  cbook,
  db = list(),
  ns = setNames(rep(500, NROW(cbook)), cbook$table)
)
```

Arguments

`cbook`	tribble. Columns: table, scaff_ref, data_rec, na_rec
`db`	list. Named list of already existing tables
`ns`	named numeric. Ns for use when generating independent tables.

Value

A named list of tables of generated data.

gen_reljoin_table *Generate synthetic data relationally-linked to existing data*

Description

Generate synthetic data relationally-linked to existing data

Usage

```
gen_reljoin_table(
  joinrec,
  tblrec,
  miss_recipe = NULL,
  db,
  keep = NA_character_
)
```

Arguments

joinrec	tibble. Recipe for synthesizing core/seed data based of a foreign key present in an existing table within db
tblrec	tibble. Recipe for generating the remainder of the new table, via gen_table_data, building on initial table generated using joinrec.
miss_recipe	tibble or NULL. A missingness recipe, if desired, to be applied after data generation via inject_nas.
db	list. A named list of existing tibbles/data.frames. The names will be used to resolve foreign table references in joinrec.
keep	TODO

Details

In relational database terms, this function synthesizes new data in a table which has a foreign key in a table existing already within db. Typically it will not generate data in the same dimension as the foreign table (as in that case the new data could simply be columns added to the existing table). Instead, it generally has the possibility of multiple rows for a particular foreign-key value, the possibility a foreign key value is not present at all, or both. A concrete example of this is Adverse Events being mapped to patients (USUBJID in CDISC terms). Some patients will have multiple adverse events, while many will have none at all.

This is done via 3 steps:

1. Applying the relational join recipe. The "relational join recipe" step should be considered primarily as the mechanism for defining the *dimensions* of the new data table.

2. The main data synthesis step, which is done by applying the tblrec recipe on the scaffolding provided by the newly dimensioned table generated in step 1.

3. Injecting missingness (optional) using missrec.

Value

The newly synthesized data table.

See Also

reljoin_funcs

---

gen_table_data *Generate variables in a table*

Description

Generate variables in a table

Usage

```
gen_table_data(
  N = if (is.null(df)) 400 else NROW(df),
  recipe,
  df = NULL,
  df_keepcols = if (is.null(df)) character() else names(df),
  miss_recipe = NULL
)
```

Arguments

N	numeric(1). Number of rows to generate. Defaults to 400, or the number of rows in df if provided.
recipe	tibble. A recipe for generating variables into the dataset. see Details.
df	data.frame/tibble. Existing partial data which new variables should be added to, or NULL (the default).
df_keepcols	logical. which columns from df should be retained in the resulting dataset (by position). Defaults to all columns present in df.
miss_recipe	tibble. A recipe for generating missingness positions, or NULL (the default).

Details

The recipe parameter should be a tibble made up of one or more rows which define variable recipes via the following columns:

variables (list column as needed) names of variables generated by that row. No empty/length 0 entries allowed

dependencies (list column). Names of variables which must have already been populated for the the variables in this row to be synthesized

func (list column) A character value which can be used to look up a function, or the function object itself, which accepts n, .df, and ... and returns either an atomic vector of length n, or a data.frame/tibble with n rows

func_args (list column) a list of arguments which should be passed to `func` in addition to `n` and `.df`

The algorithm for synthesizing the table from the recipe is as follows:

1. Columns of synthesized data are generated according to recipe rows which have no dependencies in the order they appear in the recipe tibble and appended to the dataset with names for the variables generated
2. Recipe rows containing dependencies are checked in the order they appear in the recipe table for whether their dependencies are met, and if so data is synthesized for the corresponding variables and added to the dataset. This step is repeated until all recipe rows have been resolved, or until a full pass through the unresolved recipe rows does not lead to any new data synthesis.
3. After all data synthesis is complete, columns are then reordered based on any columns of `df` first, followed by newly synthesized variables in the order the appear in the recipe table's `variables` column.

Examples

```
library(tibble)
dat <- cbind(model = row.names(mtcars), as_tibble(mtcars))
recipe <- tribble(~variables, ~dependencies, ~func, ~func_args,
        "id", no_deps, "seq_n", NULL,
        "specialid", "id", function(n, .df) paste0("special-", .df$id), NULL)

gen_table_data(10, recipe)
```

`init_new_cols` *Initialize new columns of the correct length*

Description

Initialize new columns of the correct length

Usage

```
init_new_cols(
  n,
  colnames = names(colclasses),
  colclasses = setNames(rep(NA, length(colnames)), colnames)
)
```

Arguments

n	numeric(1). The length (number of rows) to use when initializing.
colnames	character. Vector of column names to use. Can be omitted if colclasses is specified.
colclasses	named character. Optional. Names must be identical to colnames if specified, values are classes such that as(NA, .) will succeed. Defaults to NA for each column, indicating character columns.

Value

A data.frame with the new columns and n rows.

Examples

```
init_new_cols(5, c("col1", "col2"))

init_new_cols(5, colclasses = c(col1 = NA, col2 = "integer"))
```

---

inject_nas *Apply random or systematic missingness to existing data according to recipe*

Description

Apply random or systematic missingness to existing data according to recipe

Usage

```
inject_nas(tbl, recipe)
```

Arguments

tbl	data.frame/tibble. The already generated data to inject missingness into
recipe	tibble. A recipe for generating missingness positions

Details

Unlike in data-generation recipes, the func column in a missingness recipe must return a logical vector of length n or an n x k logical matrix, where n is the number of rows in .df and k is the number of variables listed in this row of the recipe. A vector is only allowed when only one variable is listed in the recipe row (ie k=1). TRUE in the return value indicates that position in the column being processed should be set to missing (NA), while FALSE indicates the value already there should remain unchanged.

Value

tbl, with missingness injected into it as laid out by recipe

Examples

```
library(tibble)
missrec <- tibble(variables = "wt", func = list(function(.df) rep(c(TRUE, FALSE), times = c(3, NROW(.df) - 3))), fun
newdat <- inject_nas(mtcars, missrec)
head(newdat)
```

---

lookup_fun *Lookup function from value you recipe column*

Description

This function is an internal utility exported for its usefulness when debugging recipes. Normal workflows will not involve calling it directly.

Usage

```
lookup_fun(str)
```

Arguments

str	ANY. A function (immediately returned) or a character(1) value of the form func, "pkg::func" or "pkg:::func". Any other value will result in an error.

Value

A function

Examples

```
lookup_fun(rnorm)
lookup_fun("rnorm")
lookup_fun("stats::rnorm")
lookup_fun("stats:::rnorm")
```

---

miss_as_block *Function constructor for injecting "missing together" NAs to columns*

Description

This constructor creates a function which will set all nvars columns to NA together for rows selected for missing data.

Usage

```
miss_as_block(p, nvars)
```

Arguments

p	numeric(1). Proportion of observations to change to missing
nvars	numeric(1). Number of columns this behavior should be applied to.

Value

a function suitable for use in missingness recipe, which when given `.df`, the data will return an N x nvars logical matrix.

---

miss_at_random *Function constructor for injecting independent missing-at-random NAs to columns*

Description

Function constructor for injecting independent missing-at-random NAs to columns

Usage

```
miss_at_random(p, nvars)
```

Arguments

p	numeric(1). Proportion of observations to change to missing
nvars	numeric(1). Number of columns this behavior should be applied to.

Value

a function suitable for use in missingness recipe, which when given `.df`, the data will return an N x nvars logical matrix.

---

no_key *Sentinel Values for Recipes*

Description

Sentinel Values for Recipes

Usage

```
no_key

no_deps

no_rec

no_args
```

Format

An object of class `character` of length 0.

An object of class `character` of length 0.

An object of class `list` of length 0.

An object of class `list` of length 0.

`pct_orig` *Generate a sample of random datetimes*

Description

Generates random datetimes that are, elementwise, between `start` and `end`

Usage

```
pct_orig

secs_per_year

secs_per_day

rand_posixct(
  start,
  end,
  max_duration_secs = NULL,
  multiplier = if (is(start, "Date")) secs_per_day else 1,
  n = max(length(start), length(end))
)
```

Arguments

`start`	POSIXct or Date. Earliest possible datetime for thte sample
`end`	POSIXct or Date. latest possible datetime for thte sample
`max_duration_secs`	numeric. Number of seconds to use to generate alernate end if `end` has a missing value.
`multiplier`	numeric. Used internally.
`n`	numeric. Length of sample. Default to max of `length(start)` and `length(end)`.

Format

An object of class `character` of length 1.

An object of class `numeric` of length 1.

An object of class `numeric` of length 1.

Value

A POSIXct vector of datetimes.

Examples

```
rand_posixct("2020-01-01", "2021-01-01", n = 2)
rand_posixct(c("1995-04-01", "2000-01-01"), c("2000-04-01", "2000-01-30"))
```

---

rep_per_key *Convenience helper functions for defining relational-join recipes*

Description

These function constructors create functions suitable for use in relational-join recipes that expand or contract the row-dimension of the incoming data.

Usage

```
rep_per_key(keyvar, tblnm, count, prop_present = 1)

rand_per_key(keyvar, tblnm, mincount = 1, maxcount = 20, prop_present = 0.5)
```

Arguments

keyvar	character(1). The name of the column to treat as a foreign key.
tblnm	character(1). The name of the table in the database in which to find the keyvar
count	numeric(1). The number of times each foreign-key value should appear in the scaffold data.
prop_present	numeric(1). Proportion of the key values in the foreign table to include rows for in the dimension-scaffold. Defaults to 1 (all values present).
mincount	numeric(1). Minimum replications for a present key
maxcount	numeric(1). Maximum replications for a pressent key

Details

rep_per_key creates functions which generate dimension-scaffolds that contain a constant number of rows per key value (ie row of the incoming data), e.g., the the map from ADSL requires 3 rows per patient (foreign key) to synthesize the long-form PARAMCD-based ADTTE data.

rand_per_key creates functions which generate dimension-scaffolds which contain a uniformly distributed random number of rows per key value. An example of this would be that for adverse events, a patient can have anywehre from 0 to 20 adverse events, each of which is a separate row in the new dimensions.

Examples

```
foreign_tbl <- data.frame(id = 1:5)
perkey_fun <- rep_per_key("id", tblnm = "foreign_tbl",  2, .6)
perkey_fun(.db = list(foreign_tbl = foreign_tbl))

randrep_fun <- rand_per_key("id", tblnm = "foreign_tbl", mincount = 1, maxcount = 5)
randrep_fun(.db = list(foreign_tbl = foreign_tbl))
```

`sample_fct` *Create a factor with random elements of x*

Description

Sample elements from x with replacing and build a factor

Usage

```
sample_fct(x, n, ...)

sample_yn(n)

rep_n(val, n, ...)

seq_n(n, ...)
```

Arguments

`x`	character vector or factor, if character vector then it is also used as levels of the returned factor, otherwise if it is a factor then the levels get used as the new levels
`n`	number of observations to sample.
`...`	arguments passed on to `sample`
`val`	ANY. Single value to be repeated n times

Value

a factor of length N

Examples

```
sample_fct(letters[1:3], 10)
sample_fct(iris$Species, 10)

sample_yn(3)

rep_n("aaa", 5)
rep_n(1:5, 2)
```

```
seq_n(10)
```

`subjid_func` *Generate sequence of "subject id"s*

Description

Generate sequence of "subject id"s

Usage

```
subjid_func(n, prefix = "id", suffix = NULL, sep = "-")
```

Arguments

`n`	numeric(1). number of ids to generate. Values will be padded with leading 0s so all resulting ids have equal width
`prefix`	character(1). Prefix to prepend to the generated numeric ids. Defaults to `"id"`
`suffix`	character(1). Suffix to append to generated ids. Defautls to `NULL` (no suffix).
`sep`	character(1). String to use as separator when combining `prefix`, number, and `suffix`.

Value

sequence from 1 to `n`, prepended with `prefix`, and appended with `suffix`, separated by `sep`

Examples

```
subjid_func(5)
subjid_func(3, suffix = "x")
```

`table_spec` *Construct a table specification from one or more recipes*

Description

Construct a table specification from one or more recipes

Usage

```
table_spec(data_rec, scaffold_rec = NULL, missing_rec = NULL)
```

Arguments

data_rec	tibble. A recipe for the data of the table
scaffold_rec	tibble or NULL. A scaffolding join recipe.
missing_rec	tibble or NULL. A recipe for injecting missingness

Value

An object representing the collection of recipes corresponding this single table. Currently a named list.

---

validate_recipe_deps *Validate recipe for circular dependencies*

Description

Validate recipe for circular dependencies

Usage

```
validate_recipe_deps(recipe, seed_df = NULL)
```

Arguments

recipe	tibble. Table recipe.
seed_df	tibble. Seed/pre-existing data.frame or NULL.

Value

TRUE if successful, throws an error if not

Index