Package 'healthdb'

May 5, 2024

Type Package

Title Working with Healthcare Databases

Version 0.2.0

Description A system for identifying diseases or events from healthcare databases and preparing data for epidemiological studies. It includes capabilities not supported by 'SQL', such as matching strings by 'stringr' style regular expressions, and can compute comorbidity scores (Quan et al. (2005) <doi:10.1097/01.mlr.0000182534.19832.83>) directly on a database server. The implementation is based on 'dbplyr' with full 'tidyverse' compatibility.

License MIT + file LICENSE

Encoding UTF-8

Imports clock, data.table, dbplyr (>= 2.5.0), dplyr (>= 1.1.0), glue, lubridate, magrittr, purrr, rlang, stringr (>= 1.5.0), tidyr

RoxygenNote 7.3.1

URL https://github.com/KevinHzq/healthdb, https://kevinhzq.github.io/healthdb/

BugReports https://github.com/KevinHzq/healthdb/issues

Suggests knitr, rmarkdown, testthat (>= 3.0.0), withr, RSQLite

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation no

Author Kevin Hu [aut, cre, cph] (<https://orcid.org/0000-0003-0254-5277>)

Maintainer Kevin Hu <kevin.hu@bccdc.ca>

Repository CRAN

Date/Publication 2024-05-05 21:20:02 UTC

R topics documented:

`bind_source` *Row-bind a list of data.frames or remote tables*

Description

Row bind a list of data.frames or remote tables with variable selection and renaming.

Usage

```
bind_source(data, ..., force_proceed = getOption("healthdb.force_proceed"))
```

Arguments

`data`	A list of data.frame or remote tables, e.g., output from `execute_def()`.
`...`	Named arguments for each variable included in the output. The argument name should be the new name in the output, and the right hand side of the argument is a character vector of the original names. The name vector and the list elements in `data` will be matched by position. if an output variable only came from some of the sources, fill the name vector to a length equal to the number of sources with NA, e.g., `var` only come from the second out of three sources, then `var = c(NA, 'nm_in_src2', NA)`.
`force_proceed`	A logical for whether to ask for user input in order to proceed when remote tables are needed to be collected for binding. The default is FALSE to let user be aware of that the downloading process may be slow. Use `options(healthdb.force_proceed = TRUE)` to suppress the prompt once and for all.

Value

A data.frame or remote table containing combined rows of the input list with variables specified by ...

Examples

```
df1 <- subset(iris, Species == "setosa")
df2 <- subset(iris, Species == "versicolor")
df3 <- subset(iris, Species == "virginica")

bind_source(list(df1, df2, df3),
  s_l = "Sepal.Length",
  s_w = "Sepal.Width",
  p_l_setosa = c("Petal.Length", NA, NA),
  p_l_virginica = c(NA, NA, "Petal.Length")
) %>%
head()
```

---

build_def *Build case definition function calls*

Description

This function assembles function calls from the supplied functions and their required arguments, leaving the data argument empty for easy re-use of the definition calls with different data and batch execution (see `execute_def()` for detail). It is useful for defining multiple diseases/events across multiple sources.

Usage

```
build_def(def_lab, src_labs, def_fn = define_case, fn_args)
```

Arguments

`def_lab`	A single character label for the definition, e.g., some disease.
`src_labs`	A character vector of place-holder names for the data sources that will be used to execute the definition.
`def_fn`	A list of functions (default: `define_case()`) that will filter the source data sets and keep clients met the case definition. The length of the list should be either 1 or equal to the length of `src_labs`. If length = 1, the same function will be applied to all sources; otherwise, `def_fn` should match `src_lab` by position. User can supply custom functions but must put input data as the first argument and name it `data`.
`fn_args`	A named list of arguments passing to the `def_fn`. Each element in the list should have the same name as an argument in the source-specific `def_fn`, and the element length should also be either 1 or equal to the number of sources. If you have `def_fn` functions taking different sets of arguments, include the union in one list.

Value

A tibble with a number of rows equal to the length of `src_labs`, containing the input arguments and the synthetic function call in the `fn_call` column.

Examples

```
sud_def <- build_def("SUD", # usually a disease name
 src_lab = c("src1", "src2"), # identify from multiple sources, e.g., hospitalization, ED visits.
  # functions that filter the data with some criteria,
  # including mean here for src2 as a trivial example
  # to show only valid arguments will be in the call
  def_fn = list(define_case, mean),
  fn_args = list(
    vars = list(starts_with("diagx"), "diagx_2"),
    match = "start", # "start" will be applied to all sources as length = 1
    vals = list(c("304"), c("305")),
    clnt_id = "clnt_id",
    # c() can be used in place of list
    # if this argument only takes one value for each source
    n_per_clnt = c(2, 3),
    x = list(1:10) # src2 with mean as def_fn will only accept this argument
  )
)

# the result is a tibble
sud_def

# the fn_call column stores the code that can be ran with execute_def
sud_def#fn_call
```

---

collapse_episode *Grouping records into episodes by date range*

Description

This function is useful for collapsing, e.g., medication dispensation or hospitalization, records into episodes if the records' dates are no more than n days gap apart. The length of the gap can be relaxed by another grouping variable. This function is implemented for data.frame input only.

Usage

```
collapse_episode(
  data,
  clnt_id,
  start_dt,
  end_dt = NULL,
  gap,
  overwrite = NULL,
  gap_overwrite = Inf,
```

```
  .dt_trans = data.table::as.IDate,
  ...
)
```

Arguments

data	A data.frame that contains the id and date variables.
clnt_id	Column name of subject/person ID.
start_dt	Column name of the starting date of records.
end_dt	Column name of the end date of records. The default is NULL assuming the record last one day and only the start date will be used to calculate the gaps between records.
gap	A number in days that will be used to separate episodes. For example, gap = 7 means collapsing records no more than 7 days apart. Note that the number of days apart will be calculated as numeric difference between two days, so that Monday and Sunday is considered as 6 days apart.
overwrite	Column name of a grouping variable determining whether the consecutive records are related and should have a different gap value. For example, dispensing records may have the same original prescription number, and a different gap value can be assigned for such situation, e.g., the days between two records is > gap, but these records still belong to the same prescription.
gap_overwrite	A different gap value used for related records. The default is Inf, which means all records with the same overwrite variable will be collapsed.
.dt_trans	Function to transform start_dt/end_dt. Default is `data.table::as.IDate()`.
...	Additional arguments passing to the .dt_trans function.

Value

The original data.frame with new columns indicating episode grouping. The new variables include:

- epi_id: unique identifier of episodes across the whole data set
- epi_no: identifier of episodes within a client/group
- epi_seq: identifier of records within an episode
- epi_start/stop_dt: start and end dates corresponding to epi_id

Examples

```
# make toy data
df <- make_test_dat() %>%
dplyr::select(clnt_id, dates)

head(df)

# collapse records no more than 90 days apart
# end_dt could be absent then it is assumed to be the same as start_dt
collapse_episode(df, clnt_id, start_dt = dates, gap = 90)
```

`compute_comorbidity` *Compute Elixhauser Comorbidity Index*

Description

This function computes unweighted Elixhauser Comorbidity Index for both data.frame and remote table input. The ICD codes used to identify the 31 disease categories is from Quan et al. (2005).

Usage

```
compute_comorbidity(
  data,
  vars,
  icd_ver = c("ICD-10", "ICD-9-CM-3digits", "ICD-9-CM-5digits"),
  clnt_id,
  uid = NULL,
  sum_by = c("row", "clnt"),
  excl = NULL
)
```

Arguments

`data`	Data.frames or remote tables (e.g., from `dbplyr::tbl_sql()`)
`vars`	An expression passing to `dplyr::select()`. It can be quoted/unquoted column names, or helper functions, such as `dplyr::starts_with()`.
`icd_ver`	One of `c("ICD-10", "ICD-9-CM-3digits", "ICD-9-CM-5digits")`. Specify the ICD code version used in `data`. The ICD-10 and ICD-9-CM 5 digits version are from Quan et al. (2005). The ICD-9-CM 3 digits version is adopted from Manitoba Centre for Health Policy. It uses BOTH 3-digit and 5-digit codes in search. See their web page for cautions and limitations of the 3 digit version if your data only has 3-digit codes (http://mchp-appserv.cpe.umanitoba.ca/viewConcept.php?printer=Y&conceptID=1436#CAUTIONS).
`clnt_id`	Grouping variable (quoted/unquoted).
`uid`	Variable name for a unique row identifier. It is necessary for SQL to produce consistent result based on sorting.
`sum_by`	One of "row" or "clnt". The "row" option computes total score for each row (default), and the "clnt" option summarizes total score by `clnt_id`. Each disease categories will be counted only once in the calculation regardless of multiple records in a category.
`excl`	A character vector of disease categories labels that should be excluded in the total score calculation. This is useful when some of the categories are the exposure/outcome of interest, and the goal is to measure comorbidity excluding these disease. See detail for a list of the categories and labels.

Details

List of disease categories - labels (in quote):

1. Congestive Heart Failure - "chf"
2. Cardiac Arrhythmia - "arrhy"
3. Valvular Disease - "vd"
4. Pulmonary Circulation Disorders - "pcd"
5. Peripheral Vascular Disorders - "pvd"
6. Hypertension Uncomplicated - "hptn_nc"
7. Hypertension complicated - "hptn_C"
8. Paralysis - "para"
9. Other Neurological Disorders - "Othnd"
10. Chronic Pulmonary Disease - "copd"
11. Diabetes Uncomplicated - "diab_nc"
12. Diabetes Complicated - "diab_c"
13. Hypothyroidism - "hptothy"
14. Renal Failure - "rf"
15. Liver Disease - "ld"
16. Peptic Ulcer Disease excluding bleeding - "pud_nb"
17. AIDS/HIV - "hiv"
18. Lymphoma - "lymp"
19. Metastatic Cancer - "mets"
20. Solid Tumor without Metastasis - "tumor"
21. Rheumatoid Arthritis/collagen - "rheum_a"
22. Coagulopathy - "coag"
23. Obesity - "obesity"
24. Weight Loss - "wl"
25. Fluid and Electrolyte Disorders - "fluid"
26. Blood Loss Anemia - "bla"
27. Deficiency Anemia - "da"
28. Alcohol Abuse - "alcohol"
29. Drug Abuse - "drug"
30. Psychoses - "psycho"
31. Depression - "dep"

Value

A data.frame or remote table with binary indicators for each categories as columns.

References

Quan H, Sundararajan V, Halfon P, Fong A, Burnand B, Luthi JC, Saunders LD, Beck CA, Feasby TE, Ghali WA. Coding algorithms for defining comorbidities in ICD-9-CM and ICD-10 administrative data. Med Care 2005;43(11):1130-1139.

Examples

```
# make ICD-9 toy data
df <- data.frame(
  uid = 1:10, clnt_id = sample(1:3, 10, replace = TRUE),
  diagx_1 = c("193", "2780", "396", "4254", "4150", "401", "401", "0932", "5329", "2536"),
  diagx_2 = c(NA, NA, "72930", "V6542", "493", "405", "5880", "2409", "714", NA)
)

# compute Elixhauser Comorbidity Index by row
# uid is needed for by row calculation
# 3 categories were excluded in total_eci
compute_comorbidity(df,
  vars = starts_with("diagx"),
  icd_ver = "ICD-9-CM-5digits",
  clnt_id = clnt_id, uid = uid,
  excl = c("drug", "psycho", "dep")
)

# compute ECI by person
compute_comorbidity(df,
  vars = starts_with("diagx"),
  icd_ver = "ICD-9-CM-5digits",
  clnt_id = clnt_id,
  sum_by = "clnt"
)
```

---

compute_duration *Compute duration between two dates*

Description

This function is meant to be for data frame input only and used with `dplyr::mutate()` to compute age or duration between two character or Date columns. If a vector of breaks is given, the output will be converted to factor with labels generated automatically.

Usage

```
compute_duration(
  from,
  to,
  lower_brks = NULL,
  unit = c("year", "day", "week", "month"),
  trans = FALSE,
```

```
  .transfn = lubridate::ymd,
  verbose = getOption("healthdb.verbose"),
  ...
)
```

Arguments

from	A character or Date vector for start dates.
to	A character or Date vector for end dates.
lower_brks	A numeric vector for lower breaks passing to the base base::cut() function to convert the numeric result to a factor. The level labels will be auto generated. For example, the level labels are c("<19", "19-24", "25-34", "35-44", "45-54", "55+") for lower_brks = c(0, 19, 25, 35, 45, 55). Default is NULL (no conversion).
unit	A character string specifying the unit of the output. One of "year" (default), "day", "week", or "month".
trans	A logical for whether transform both from and to with the .transfn function
.transfn	A function for transforming the inputs. Default is lubridate::ymd().
verbose	A logical for whether print summary of the out and warning for missing values. Default is fetching from options. Use options(healthdb.verbose = FALSE) to suppress once and for all.
...	Additional arguments passing to base::cut().

Value

A numeric or factor vector of the duration.

Examples

```
# toy data
n <- 5
df <- data.frame(id = 1:n,
start_dt = sample(seq(as.Date("1970-01-01"), as.Date("2000-12-31"), by = 1), size = n),
end_dt = sample(seq(as.Date("2001-01-01"), as.Date("2023-12-31"), by = 1), size = n))

# get age group at a cut-off
df %>% dplyr::mutate(
 age_grp = compute_duration(start_dt, "2023-01-01", lower_brks = c(0, 19, 25, 35, 45, 55))
)

# compute gaps between two dates in weeks
df %>% dplyr::mutate(
 gap_wks = compute_duration(start_dt, end_dt, unit = "week")
)
```

cut_period *Cut the time period in one row into multiple rows by interval*

Description

This function is for cutting time periods into segments, which could be useful for subsequent overlap joins. Each original period (per row) will be expanded to multiple rows by weeks, months, etc. Only data.frame input is accepted as the output size is greater than the input. Thus, remote tables should be collected before running this function for optimal performance.

Usage

```
cut_period(
  data,
  start,
  end,
  len,
  unit = c("day", "week", "month", "quarter", "year"),
  .dt_trans = NULL
)
```

Arguments

data	Input data.frame that each row has start and end dates
start	Record start date column (unquoted)
end	Record end date column (unquoted)
len	An integer, the interval that would be used to divide the record duration
unit	One of "day" (default), "week", "month", "quarter, or "year" used in combination of len to specify the time length of the interval.
.dt_trans	Function to transform start/end, such as lubridate::ymd(). Default is NULL.

Value

Data frame that each row is now a segment of the period defined by c(start, end) in the original row. Original variables are retained and repeated for each segment plus new variables defining the segment interval.

Examples

```
# toy data
df <- data.frame(sample_id = 1, period_id = 1, start_date = "2015-01-01", end_date = "2019-12-31")

# divide period into segments (multiple rows per period)
df_seg <- cut_period(
  data = df, start = start_date, end = end_date,
  len = 1,
```

```
  unit = "year",
  .dt_trans = lubridate::ymd
)

# categorize segment_id as factor
df_seg$segment <- cut(df_seg$segment_id,
  breaks = c(0, 1, 2, Inf),
  labels = c("< 1 year", "1 - 2 years", "Remainder")
)

head(df_seg)
```

`define_case` *Identify diseases/events from administrative records*

Description

This function is a composite of `identify_row()`, `exclude()`, `restrict_n()`, and `restrict_date()`. It is aimed to implement common case definitions in epidemiological studies using administrative database as a one-shot big query. The intended use case is for definitions in the form of, e.g., two or more physician visits with some diagnostic code at least 30 days apart within two years. The component functions mentioned above are chained in the following order if all arguments were supplied: `identify_row(vals) %>% exclude(identify_row(excl_vals), by = clnt_id) %>% restrict_n() %>% restrict_date()`. Only necessary steps in the chain will be ran if some arguments are missing, see the verbose output for what was done. Note that if `date_var` is supplied, `n_per_clnt` will be counted by distinct dates instead of number of records.

Usage

```
define_case(
  data,
  vars,
  match = "in",
  vals,
  clnt_id,
  n_per_clnt = 1,
  date_var = NULL,
  apart = NULL,
  within = NULL,
  uid = NULL,
  excl_vals = NULL,
  excl_args = NULL,
  keep = c("all", "first", "last"),
  if_all = FALSE,
  mode = c("flag", "filter"),
  force_collect = FALSE,
  verbose = getOption("healthdb.verbose"),
  ...
)
```

Arguments

data	Data.frames or remote tables (e.g., from dbplyr::tbl_sql())
vars	An expression passing to dplyr::select(). It can be quoted/unquoted column names, or helper functions, such as dplyr::starts_with().
match	One of "in", "start", "regex", "like", "between", and "glue_sql". It determines how values would be matched. See identify_row() for detail.
vals	Depending on match, it takes different input. See identify_row().
clnt_id	Grouping variable (quoted/unquoted).
n_per_clnt	A single number specifying the minimum number of group size.
date_var	Variable name (quoted/unquoted) for the dates to be interpreted.
apart	An integer specifying the minimum gap (in days) between adjacent dates in a draw.
within	An integer specifying the maximum time span (in days) of a draw.
uid	Variable name for a unique row identifier. It is necessary for SQL to produce consistent result based on sorting.
excl_vals	Same as vals but clients/groups with these values are going to be removed from the result. This is intended for exclusion criteria of a case definition.
excl_args	A named list of arguments passing to the second identify_row() call for excl_vals. If not supplied, var, match and if_all of the first call will be re-used.
keep	One of: • "first" (keeping each client's earliest record), • "last" (keeping the latest), • and "all" (keeping all relevant records, default). • Note that "first"/"last" should not be used with "flag" mode.
if_all	A logical for whether combining the predicates (if multiple columns were selected by vars) with AND instead of OR. Default is FALSE, e.g., var1 in vals OR var2 in vals.
mode	Either: • "flag" - add new columns starting with "flag_" indicating if the client met the condition, • or "filter" - remove clients that did not meet the condition from the data. • This will be passed to both restrict_n() AND restrict_date(). Default is "flag".
force_collect	A logical for whether force downloading the result table if it is not a local data.frame. Downloading data could be slow, so the user has to opt in; default is FALSE.
verbose	A logical for whether printing explanation for the operation. Default is fetching from options. Use options(healthdb.verbose = FALSE) to suppress once and for all.
...	Additional arguments, e.g., mode, passing to restrict_date().

Value

A subset of input data satisfied the specified case definition.

Examples

```
sample_size <- 30
df <- data.frame(
  clnt_id = rep(1:3, each = 10),
  service_dt = sample(seq(as.Date("2020-01-01"), as.Date("2020-01-31"), by = 1),
    size = sample_size, replace = TRUE
  ),
  diagx = sample(letters, size = sample_size, replace = TRUE),
  diagx_1 = sample(c(NA, letters), size = sample_size, replace = TRUE),
  diagx_2 = sample(c(NA, letters), size = sample_size, replace = TRUE)
)

# define from one source
define_case(df,
  vars = starts_with("diagx"), "in", vals = letters[1:4],
  clnt_id = clnt_id, date_var = service_dt,
  excl_args = list(if_all = TRUE),
  # remove non-case
  mode = "filter",
  # keeping the first record
  keep = "first"
)

# multiple sources with purrr::pmap
# arguments with length = 1 will be recycle to match the number of sources
# wrap expressions/unquoted variables with bquote(),
# or rlang:exprs() to prevent immediate evaluation,
# or just use quoted variable names
purrr::pmap(
  list(
    data = list(df, df),
    vars = rlang::exprs(starts_with("diagx")),
    match = c("in", "start"),
    vals = list(letters[1:4], letters[5:10]),
    clnt_id = list(bquote(clnt_id)), n_per_clnt = c(2, 3),
    date_var = "service_dt",
    excl_vals = list(letters[11:13], letters[14:16]),
    excl_args = list(list(if_all = TRUE), list(if_all = FALSE))
  ),
  define_case
)
```

---

exclude *Remove rows based on conditions or another data set*

Description

This function combines dplyr::anti_join(), and negation of dplyr::filter(). When a second data set is supplied through the excl argument, anti join would be performed; otherwise, data would be filtered with the expression given via the condition argument, and the filter result would in turn be removed using dplyr::setdiff().

Usage

```
exclude(
  data,
  excl = NULL,
  by = NULL,
  condition = NULL,
  verbose = getOption("healthdb.verbose"),
  report_on = NULL,
  ...
)
```

Arguments

data	Data.frames or remote tables (e.g., from dbplyr::tbl_sql()). A subset will be removed from this data.
excl	Data frames or remote tables (e.g., from 'dbplyr'). Rows/values present in it will be removed from data if there is a match. This will be passed to dplyr::anti_join() as the second argument.
by	Column names that should be matched by dplyr::anti_join(), or a expressions with dplyr::join_by(). See dplyr::anti_join()'s by argument for detail. Default NULL is the same as setdiff(data, excl).
condition	An expression that will be passed to dplyr::filter(). The rows that satisfy condition are those to be removed from data.
verbose	A logical for whether printing explanation for the operation. Default is fetching from options. Use options(healthdb.verbose = FALSE) to suppress once and for all.
report_on	A quoted/unquoted column name for counting how many of its distinct values were removed from data, e.g., counting how many client IDs were removed. Default is NULL.
...	Additional arguments passing to dplyr::filter()/dplyr::anti_join() for finer control of matching, e.g., na action, by-group filtering, etc.

Value

A data frame or remote table that is a subset of data.

Examples

```
# exclude with condition
cyl_not_4 <- exclude(mtcars, condition = cyl == 4, report_on = cyl)
```

```
# exclude with another data
exclude(mtcars, cyl_not_4, dplyr::join_by(cyl), report_on = cyl)
```

`execute_def` *Execute parameterized case definitions*

Description

This function executes the function calls stored in the output tibble from [build_def()] with data objects supplied through a named list and returns the results as a list. It is intended to facilitate re-use of pre-defined calls with different data.

Usage

```
execute_def(
  def,
  with_data,
  bind = FALSE,
  force_proceed = getOption("healthdb.force_proceed")
)
```

Arguments

`def`	A tibble created by [build_def()].
`with_data`	A named list which the elements are in the form of src_lab = data, where 'src_lab' corresponds to the src_labs argument from [build_def()] and 'data' is the data object that will be passed to calls stored in def. The names (and length) of 'with_data' must match the unique values of src_labs in 'def'.
`bind`	A logical for whether row-binding records from multiple sources into one table. Note that the binding may fail in ways that are difficult to anticipate in advance, such as data type conflict (e.g., Date vs. character) between variables in the same name from different sources. The default is FALSE. If TRUE, the behavior is to try and return the unbinded result when failed.
`force_proceed`	A logical for whether to ask for user input in order to proceed when remote tables are needed to be collected for binding. The default is FALSE to let user be aware of that the downloading process may be slow. Use options(healthdb.force_proceed = TRUE) to suppress the prompt once and for all.

Value

A single (if bind = TRUE) or a list of data.frames or remote tables.

See Also

[bind_sources()] for binding the output with convenient renaming features.

Examples

```
# toy data
sample_size <- 30
df <- data.frame(
  clnt_id = rep(1:3, each = 10),
  service_dt = sample(seq(as.Date("2020-01-01"), as.Date("2020-01-31"), by = 1),
    size = sample_size, replace = TRUE
  ),
  diagx = sample(letters, size = sample_size, replace = TRUE),
  diagx_1 = sample(c(NA, letters), size = sample_size, replace = TRUE),
  diagx_2 = sample(c(NA, letters), size = sample_size, replace = TRUE)
)

# make df a database table
db <- dbplyr::tbl_memdb(df)

# use build_def to make a toy definition
sud_def <- build_def("SUD", # usually a disease name
 src_lab = c("src1", "src2"), # identify from multiple sources, e.g., hospitalization, ED visits.
  # functions that filter the data with some criteria
  def_fn = define_case,
  fn_args = list(
    vars = starts_with("diagx"),
    match = "start", # "start" will be applied to all sources as length = 1
    vals = list(c("304"), c("305")),
    clnt_id = "clnt_id", # list()/c() could be omitted for single element
    # c() can be used in place of list
    # if this argument only takes one value for each source
    n_per_clnt = c(2, 3)
  )
)

# save the definition for re-use
# saveRDS(sud_def, file = some_path)

sud_def %>% execute_def(with_data = list(src1 = df, src2 = db), force_proceed = TRUE)
```

`fetch_var` *Get variables from multiple tables with common ID columns*

Description

This function fetches variables from different tables that linked by common IDs. It calls `dplyr::left_join()` multiple times with different source tables (y argument of the join) to gather variables. It is not meant to replace left_join() but simplify syntax for the situation that you started off a table of study sample and wanted to gather covariates from different sources linked by common client IDs, which is often the case when working with healthcare databases. That said, this function is to replace repetitions of simple joins and only allows one-to-one matching.

Usage

```
fetch_var(data, keys, linkage, verbose = getOption("healthdb.verbose"), ...)
```

Arguments

data	A local data.frame, or tibble. It would be used as the x argument in left_join().
keys	A vector of quoted/unquoted variable names, or 'tidyselect' expression (see dplyr::select()). These variables must be present in data and would be used as the by argument in left_join(). The y tables must have a subset of these if not all.
linkage	A list of formulas in the form of "from_tab ~ get_vars|by_keys": • source table on the left-hand-side • variables on the right-hand-side • If a source table does not have all the variables in keys, use "|" on RHS to specify the subset of keys to be used. For example, given keys has 3 variables, list(y1 ~ tidyselect_expr1, y2 ~ tidyselect_expr2|key1 + key2) meaning: 1. from table y1 get variables picked by the tidyselect expression matching on all 3 keys; 2. from table y2 get variables matching on only key1 and key2.
verbose	A logical for whether report the number of rows after joining for each source. Default is getting from options. Use options(healthdb.verbose = FALSE) to suppress once and for all.
...	Additional arguments passing to left_join().

Value

A data.frame or tibble containing all original columns of x and new variables matched from other tables based on the specified linkage.

Examples

```
# make toy data
size <- 30
n <- 10
df1 <- data.frame(
  id = sample(1:n, size = size, replace = TRUE),
  service_dt = sample(seq(as.Date("2020-01-01"), as.Date("2022-01-31"), by = 1),
    size = size
  )
) %>%
  dplyr::mutate(year = lubridate::year(service_dt))
df2 <- data.frame(
  id = rep(1:n, size / n), year = rep(2020:2022, each = n),
  status_1 = sample(0:1, size = size, replace = TRUE),
  status_2 = sample(0:1, size = size, replace = TRUE)
```

```
)
df3 <- data.frame(id = 1:n, sex = sample(c("F", "M"), size = n, replace = TRUE))

# simple joins
# note that for left_join(df1, df2), boths keys have to be used,
# otherwise, error as the relation would not be one-to-one
fetch_var(df1,
  keys = c(id, year),
  linkage = list(
    df2 ~ starts_with("s"), # match both keys without '|'
    df3 ~ sex | id
  ) # match by id only; otherwise failed because df3 has no year
)

# example if some y is remote
# make df2 as database table
db2 <- dbplyr::tbl_memdb(df2)

fetch_var(df1,
  keys = c(id, year),
  linkage = list(
    db2 ~ starts_with("s"),
    df3 ~ sex | id
  ),
  copy = TRUE # pass to left_join for forced collection of remote table
)
```

`identify_row` *Identify rows with a match*

Description

Filter rows which values satisfy the specified conditions. The functionality is identical to `dplyr::filter()` combined with `dplyr::if_any()` or `dplyr::if_all()`, but it used the 'data.table' package `vignette("datatable-intro" package = "data.table")` for data.frame method, and has regular regular expression support for remote database tables. The motivation is to take away some pain when working with databases which often do not support regular expression and 'LIKE' operator with multiple string patterns.

Usage

```
identify_row(
  data,
  vars,
  match = c("in", "start", "regex", "like", "between", "glue_sql"),
  vals,
  if_all = FALSE,
  verbose = getOption("healthdb.verbose"),
  query_only = TRUE,
  ...
)
```

Arguments

data	Data.frames or remote tables (e.g., from dbplyr::tbl_sql())
vars	An expression passing to dplyr::select(). It can be quoted/unquoted column names, or helper functions, such as dplyr::starts_with().
match	One of "in", "start", "regex", "like", "between", and "glue_sql". It determines how values would be matched. The operations under each type: • "in": var %in% vals (This is default) • "regex": stringr::str_detect(var, vals). For remote tables, unique values in vars are collected locally before matching (may be slow). • "like": stringr::str_like(var, vals). For remote tables, WHERE var LIKE val. • "start": same as regex or LIKE with modified vals, e.g., "^val1|^val2" or "va1%|val2%" • "between": dplyr::between(var, val1, val2) • "glue_sql": For remote table only, this gives full control of the WHERE clause using dplyr::filter(dbplyr::sql(glue::glue_sql(...)))
vals	Depending on match, it takes different input: • "in": a vector of values (numeric/character/Date) • "start": a vector of numeric/character that would be modified into a regex or LIKE pattern string by adding "^" in front or "%" at the end • "regex"/"like": a string of the expression • "between": a vector of numeric or date with exactly two elements, e.g., c(lower, upper) • "glue_sql": a string of a SQL WHERE clause, which will be passed to glue::glue_sql(). See examples for detail.
if_all	A logical for whether combining the predicates (if multiple columns were selected by vars) with AND instead of OR. Default is FALSE, e.g., var1 in vals OR var2 in vals.
verbose	A logical for whether printing explanation and result overview for the query. Default is fetching from options. Use options(healthdb.verbose = FALSE) to suppress once and for all. Result overview is not for remote tables as the query is not executed immediately, thus no result is available for summary without adding an extra run (may be slow) of the query.
query_only	A logical for whether keeping the output as remote table (Default TRUE) or downloading the query result as a tibble (FALSE). The argument is ignored when the input data is a data.frame/tibble.
...	For remote table method only. Additional arguments passing to glue::glue_sql() for parameterized queries.

Value

A data.frame or tbl_sql object depending on the input.

Examples

```
#applying to data.frame; both sepal length and width in range 3-5
identify_row(iris, starts_with("Sepal"), "between", c(3, 5), if_all = TRUE)

#applying to remote table; species starts with se or ends with ca
iris_db <- dbplyr::memdb_frame(iris)
identify_row(iris_db, Species, "like", c("se%", "%ca"))

#using glue_sql to write the WHERE clause
#use {`vars`} to refer to the variables selected by vars
#supply additional values required in the query through '...'
#note that if you use LIKE here, you cannot supply multiple patterns in what
identify_row(iris_db, Species, "glue_sql",
 "{`vars`} LIKE {what}",
  what = "se%")

#add * after a vector
identify_row(iris_db, Species, "glue_sql",
 "{`vars`} IN ({what*})",
 what = c("setosa", "virginica"))
```

---

if_date *Interpret if n dates drawn from a vector could be some days apart within some years*

Description

Given a vector of dates x, interpret if there could be at least one set of n elements taken from x satisfy that adjacent elements in the set are at least certain days apart AND the dates in the set are within the specified time span. When identifying events/diseases from administrative data, definitions often require, e.g., n diagnoses that are at least some days apart within some years. This function is intended for such use and optimized to avoid looping through all n-size combinations in x. This function does not work with remote table input.

Usage

```
if_date(
  x,
  n,
  apart = NULL,
  within = NULL,
  detail = FALSE,
  align = c("left", "right"),
  dup.rm = TRUE,
  ...
)
```

Arguments

x	A character or Date vector
n	An integer for the size of a draw
apart	An integer specifying the minimum gap (in days) between adjacent dates in a draw.
within	An integer specifying the maximum time span (in days) of a draw.
detail	Logical for whether return result per element of x.The default is FALSE, which returns one logical summarized by any(). Detail is not available if `apart` was supplied without `within` because sets that satisfied the condition could overlap, and records within a set may be far apart; thus, no unambiguous way to label by element.
align	Character, define if the time span for each record should start ("left") or end ("right") at its current date. Defaults to "left". See 'flag_at' argument in `restrict_date()` for detail.
dup.rm	Logical for whether multiple records on the same date should be count as one in calculation. Only applicable when `within` is supplied without `apart`; duplicated dates have no impact when `apart` is present as the n dates must be distinct if they were apart. Default is TRUE.
...	Additional argument passing to `data.table::as.IDate()` for date conversion.

Value

Single or a vector of logical for whether there is any draw from x satisfied the conditions

See Also

`restrict_date()`

Examples

```
dates_of_records <- sample(seq(as.Date("2015-01-01"), as.Date("2021-12-31"), 7), 10)

# whether there is any 3 records at least 30 days apart within 2 years
if_date(dates_of_records, n = 3, apart = 30, within = 365 * 2)

# specified either apart or within or both
if_date(dates_of_records, n = 2, within = 365)
```

lookup *Find value corresponding to input vector using a look-up table*

Description

Find value corresponding to input vector using a look-up table

Usage

```
lookup(x, link, lu, verbose = getOption("healthdb.verbose"))
```

Arguments

x	A variable name in a data.frame; this function should be called inside dplyr::mutate().
link	A formula in the form: name_of_x_in_lu ~ name_of_target_value. The left-hand-side can be omitted if x's name is also x in the look-up.
lu	Look-up table in data.frame class.
verbose	A logical for whether warn for missing values in the output.

Value

A vector of matched values.

Examples

```
df <- data.frame(drug_code = 1:10)
lu <- data.frame(drug_id = 1:20, drug_code = as.character(1:10), drug_name = sample(letters, 20))

df %>% dplyr::mutate(
  drug_nm = lookup(drug_code, drug_id ~ drug_name, lu),
  # this will work as lu also has drug_code column
  drug_nm = lookup(drug_code, ~ drug_name, lu)
)
```

make_test_dat *Make test data*

Description

Make a toy data set for testing and demo. This is for internal use purpose and not intended to be called by users.

Usage

```
make_test_dat(
  vals_kept = c("304", "305", 3040:3049, 3050:3059),
  noise_val = "999",
  IDs = 1:50,
  date_range = seq(as.Date("2015-01-01"), as.Date("2020-12-31"), by = 1),
  nrows = 100,
  n_any = 50,
  n_all = 10,
  seed = NULL,
  answer_id = NULL,
  type = c("data.frame", "database")
)
```

Arguments

vals_kept	A vector of values that suppose to be identified.
noise_val	A vector of values that are not meant to be identified.
IDs	A vector of client IDs.
date_range	A vector of all possible dates in the data.
nrows	Number of rows of the output.
n_any	Number of rows to be identified if the criteria is that if any target column contains certain values.
n_all	Number of rows to be identified if the criteria is that if all target columns contain certain values.
seed	Seed for random number generation.
answer_id	Column name for the indicator of how the row should be identified: any, all, and noise.
type	Output type, "data.frame" or "database".

Value

A data.frame or remote table from 'dbplyr'.

Examples

```
make_test_dat() %>% head()
```

`pool_case` *Pooling qualified clients from multiple sources*

Description

This function filters and pools, i.e., row bind, qualified clients/groups from different source with an option to summarize by client. Unlike `bind_source()`, no need to supply variable names; the function will guess what should be included and their names from the supplied definition from `build_def()`. Whether a client is qualified relies on the flag variables set by `define_case()`. Therefore, this function is intended to be use only with the built-in `define_case()` as `def_fn` in `build_def()`.

Usage

```
pool_case(
  data,
  def,
  output_lvl = c("raw", "clnt"),
  include_src = c("all", "has_valid", "n_per_clnt"),
  ...
)
```

Arguments

`data`	A list of data.frame or remote table which should be output from `execute_def()`.
`def`	A tibble of case definition generated by `build_def()`.
`output_lvl`	Either: • "raw" - output all records (default), • or "clnt" - output one record per client with summaries including date of first valid record ('first_valid_date'), date of the latest record ('last_entry_date'), and sources that contain valid records.
`include_src`	Character. It determines records from which sources should be included. This matters when clients were identified only from, not all, but some of the sources. This choice will not impact the number of client that would be identified but has impact on the number of records and the latest entry date. The options are one of: • "all" - records from all sources are included; • "has_valid" - for each client, records from sources that contain at least one valid record are included; • "n_per_clnt" - for each client, if they had fewer than `n_per_clnt` records in a source (see `restrict_n()`), then records from that source are removed.
`...`	Additional arguments passing to `bind_source()`

Value

A data.frame or remote table with clients that satisfied the predefined case definition. Columns started with "raw_in_" are source-specific counts of raw records, and columns started with "valid_in_" are the number of valid entries (or the number of flags) in each source.

Examples

```
# toy data
df1 <- make_test_dat()
df2 <- make_test_dat()

# use build_def to make a toy definition
sud_def <- build_def("SUD", # usually a disease name
 src_lab = c("src1", "src2"), # identify from multiple sources, e.g., hospitalization, ED visits.
  # functions that filter the data with some criteria
  def_fn = define_case,
  fn_args = list(
    vars = starts_with("diagx"),
    match = "start", # "start" will be applied to all sources as length = 1
    vals = list(c("304"), c("305")),
    clnt_id = "clnt_id", # list()/c() could be omitted for single element
    # c() can be used in place of list
    # if this argument only takes one value for each source
    n_per_clnt = c(2, 3)
  )
)

# save the definition for re-use
# saveRDS(sud_def, file = some_path)

# execute definition
sud_by_src <- sud_def %>% execute_def(with_data = list(src1 = df1, src2 = df2))

# pool results from src1 and src2 together at client level
pool_case(sud_by_src, sud_def, output_lvl = "clnt")
```

`report_n` *Report number of distinct value in a column across data frames*

Description

This function is intended to mimic `dplyr::n_distinct()` for multiple inputs. It is useful to report the number of clients through out a series of inclusion or exclusion steps. An use case could be getting the Ns for the sample definition flowchart in an epidemiological study. It is also useful for inline reporting of Ns in a Rmarkdown document.

Usage

```
report_n(..., on, force_proceed = getOption("healthdb.force_proceed"))
```

Arguments

...	Data frames or remote tables (e.g., from 'dbplyr')
on	The column to report on. It must be present in all data sources.
force_proceed	A logical for whether to ask for user input in order to proceed when the data is not local data.frames, and a query needs to be executed before reporting. The default is fetching from options (FALSE). Use `options(healthdb.force_proceed = TRUE)` to suppress the prompt once and for all.

Value

A sequence of the number of distinct on for each data frames

Examples

```
# some exclusions
iris_1 <- subset(iris, Petal.Length > 1)
iris_2 <- subset(iris, Petal.Length > 2)

# get n at each operation
n <- report_n(iris, iris_1, iris_2, on = Species)
n

# get the difference at each step
diff(n)
# data in a list
iris_list <- list(iris_1, iris_2)
report_n(rlang::splice(iris_list), on = Species)
# if you loaded tidyverse, this will also work
# report_n(!!!iris_list, on = Species)
```

---

restrict_date *Removes or flags groups failed to meet conditions based on dates*

Description

For each client or group, interpret if they have n records that are at least certain days apart AND within a specified time span. When identifying events/diseases from administrative data, definitions often require, e.g., n diagnoses that are at least some days apart within some years. This function is intended for such use and optimized to avoid looping through all n-size combinations of dates per client.

Usage

```
restrict_date(
  data,
  clnt_id,
  date_var,
```

```
  n,
  apart = NULL,
  within = NULL,
  uid = NULL,
  mode = c("flag", "filter"),
  flag_at = c("left", "right"),
  dup.rm = TRUE,
  force_collect = FALSE,
  verbose = getOption("healthdb.verbose"),
  ...
)
```

Arguments

`data`	Data frames or remote tables (e.g., from `dbplyr::tbl_sql()`)
`clnt_id`	Grouping variable (quoted/unquoted).
`date_var`	Variable name (quoted/unquoted) for the dates to be interpreted.
`n`	An integer for the size of a draw.
`apart`	An integer specifying the minimum gap (in days) between adjacent dates in a draw.
`within`	An integer specifying the maximum time span (in days) of a draw.
`uid`	Variable name for a unique row identifier. It is necessary for SQL to produce consistent result based on sorting.
`mode`	Either: • "flag" - add a new column 'flag_restrict_date' indicating if the condition was met (flag = 1 if the time period starting or ending at the current record satisfied the apart-within condition), • or "filter" - remove clients without any qualified record from the data. Default is "flag".
`flag_at`	Character, define if the flag should be placed at the start ("left") or end ("right") of a time period that contains n qualified records. Defaults to "left". Note that this would impact the first and last qualified/diagnosed dates of a client, e.g., using "right" will have the first flag not at the earliest but the date which the client became qualified. For example, if the condition was 2 records within a year, for `c("2023-01-01", "2023-04-01", "2024-05-01")`, flag will be `c(0, 1, 0)` for "right" while `c(1,0,0)` for "left".
`dup.rm`	Logical for whether multiple records on the same date should be count as one in calculation. Only applicable when `within` is supplied without `apart`; duplicated dates have no impact when `apart` is present as the n dates must be distinct if they were apart. Default is TRUE.
`force_collect`	A logical for whether force downloading remote table if `apart` is not NULL. For remote table only, because `apart` is implemented for local data frame only. Downloading data could be slow, so the user has to opt in; default FALSE will stop with error.

verbose	A logical for whether to explain the query and report how many groups were removed. Default is fetching from options. Use `options(healthdb.verbose = FALSE)` to suppress once and for all. Reporting is not for remote tables as the query is not executed immediately, thus no result is available for summary without adding an extra run (may be slow) of the query.
...	Additional argument passing to `data.table::as.IDate()` for date conversion.

Value

A subset of input data satisfied the dates requirement, or raw input data with an new flag column.

See Also

`if_date()`

Examples

```
sample_size <- 30
df <- data.frame(
  clnt_id = sample(1:sample_size, sample_size, replace = TRUE),
  service_dt = sample(seq(as.Date("2020-01-01"), as.Date("2020-01-31"), by = 1),
    size = sample_size, replace = TRUE
  ),
  diagx = sample(letters, size = sample_size, replace = TRUE),
  diagx_1 = sample(c(NA, letters), size = sample_size, replace = TRUE),
  diagx_2 = sample(c(NA, letters), size = sample_size, replace = TRUE)
)

# Keep clients with 2 records that were 1 week apart within 1 month
restrict_date(df, clnt_id, service_dt, n = 2, apart = 7, within = 30)
```

---

restrict_n *Removes or flags groups with n less than some number*

Description

Remove or flags groups or clients that have less than some number of rows or some number of distinct values in a variable. For example, it can be used to remove clients that had less than n visits to some service on different dates from some administrative records. It offers filtering with `dplyr::n_distinct()` functionality for database input.

Usage

```
restrict_n(
  data,
  clnt_id,
  n_per_clnt,
  count_by = NULL,
```

```
  mode = c("flag", "filter"),
  verbose = getOption("healthdb.verbose")
)
```

Arguments

data	Data.frames or remote tables (e.g., from dbplyr::tbl_sql())
clnt_id	Grouping variable (quoted/unquoted).
n_per_clnt	A single number specifying the minimum number of group size.
count_by	Another variable dictating the counting unit of n_per_clnt. The default is NULL meaning the inclusion criteria is the number of row, i.e., dplyr::n() >= n_per_clnt. If it is not NULL, the criteria becomes equivalent to dplyr::n_distinct(count_by) >= n_per_clnt.
mode	Either "flag" - add a new column 'flag_restrict_n' indicating if the client met the condition (all rows from a qualified client would have flag = 1), or "filter" - remove clients that did not meet the condition from the data. Default is "flag".
verbose	A logical for whether to explain the query and report how many groups were removed. Default is fetching from options. Use options(healthdb.verbose = FALSE) to suppress once and for all. Reporting is not for remote tables as the query is not executed immediately, thus no result is available for summary without adding an extra run (may be slow) of the query.

Value

A subset of input data satisfied the group size requirement, or raw input data with an new flag column.

See Also

dplyr::n(), dplyr::n_distinct()

Examples

```
# flag cyl groups with less than 8 cars
restrict_n(mtcars, clnt_id = cyl, n_per_clnt = 8, mode = "flag") %>%
head()

#remove cyl groups with less than 2 types of gear boxes
restrict_n(mtcars, clnt_id = cyl, n_per_clnt = 3, count_by = gear, mode = "filter")
```

Index