Package 'rEHR'

January 5, 2016

Author David Springate [aut, cre],
Evangelos Kontopantelis [ctb],
Ivan Olier [ctb]

Maintainer David Springate <daspringate@gmail.com>

Version 1.0

License GPL-3

Title An R package for manipulating and analysing Electronic Health Record data

Description R tools for processing and extracting clinical information from Electronic Medical Records Databases

Licence GNU GPL3

Depends R (>= 3.0.2), sqldf, RSQLite, dplyr

Imports assertthat, stringr, foreign, parallel, readstata13, combinat, xlsx

Suggests testthat, devtools, roxygen2, knitr, reshape2, survival

VignetteBuilder knitr

LazyData true

R topics documented:

add_to_database *Adds a series of files to a database*

Description

This function can be used to import a CPRD file or files into a SQLite database connection.

Usage

```
add_to_database(db, files, table_name, dateformat = "%d/%m/%Y",
  yob_origin = 1800, practid = TRUE, filenames = FALSE)
```

Arguments

db	a database connection object
files	a character vector of filenames to files to be imported
table_name	a name for the table to import to
dateformat	the format that dates are stored in the CPRD data. If this is wrong it won't break but all dates are likely to be NA
yob_origin	value to add yob values to to get actual year of birth (Generally 1800)
practid	logical should practice id variable be constructed from the patient ids?
filenames	logical should the filename be included as a variable?

Details

Will automatically unzip files before calling them in If practid is TRUE, a practid variable is constructed by converting the last 3 digits of the patient id (if supplied) to a numeric. If filenames is TRUE, source data filenames are included as a variable with the filetypes stripped away.

append_to_temp_table *Appends rows to a temporary table*

Description

This function checks if a table is a temp table and then adds to it based on the select statement The check is to maintain the integrity of the database

Usage

```
append_to_temp_table(db, tab_name, columns, select_query)
```

Arguments

db	a database connection
tab_name	the name of the temporary table being appended to
columns	character vector of columns in tab_name
select_query	SQL query for the selector

`build_cohort` *Converts a longitudinal data set from e.g.* `prev_terms` *to a cohort dataset. The dataset has a single row for each patient and includes only patients in the numerator or denominator for whichever cohort type is chosen. Columns are added for start and end dates and for start and end times as integer differences from the cohort start date. A binary column is added for membership of the case group. All patients with start date > end date are removed from the dataset.*

Description

Converts a longitudinal data set from e.g. `prev_terms` to a cohort dataset. The dataset has a single row for each patient and includes only patients in the numerator or denominator for whichever cohort type is chosen. Columns are added for start and end dates and for start and end times as integer differences from the cohort start date. A binary column is added for membership of the case group. All patients with start date > end date are removed from the dataset.

Usage

```
build_cohort(dat, cohort_type = c("incid", "prev"), cohort_start, cohort_end,
  diagnosis_start = "eventdate")
```

Arguments

`dat`	dataframe as output from a call to `prev_terms`
`cohort_type`	character either 'incid' or 'prev'. This selects the numerators and denominators for that type of cohort
`cohort_start`	ISO date character string for the start of the cohort
`cohort_end`	ISO date character string for the end of the cohort
`diagnosis_start`	character string for the name of the diagnosis variable used to define the start dates, or NULL if the diagnosis date is not to be included in the definition of start dates.

`build_date_fn` *Function to build start/enddate helper fuctions*

Description

This builds functions identical to qof_years but they can be customised to the user's preferences

Usage

```
build_date_fn(start, end)
```

Arguments

`start`	list containing the offset in years, month and day as numerics for the start date
`end`	list containing the offset in years, month and day as numerics for the end date

Value

a function taking a year as an argument and returning a list of startdates and enddates

---

clinical_codes *Clinical codes for 17 QOF conditions, smoking and HbA1c*

Description

This data comes from http://www.clinicalcodes.org

Usage

```
clinical_codes
```

Format

A dataframe with 1283 observations:

medcode Unique internal code for the medical term selected by the healthcare professional

readcode Unique Read Code for the medical term selected by the healthcare professional

desc Description of the code

list Name of the clinical code list this code belongs to

Source

http://www.clinicalcodes.org

---

compress *Compresses a dataframe to make more efficient use of resources*

Description

Converts date variables in a dataframe to integers Integers represent time in days from the supplied origin Converts specified numeric values to integer This function is useful for keeping file sizes down and is used by the to_stata command to save to Stata files.

Usage

```
compress(dat, origin = "1970-01-01", format = "%Y-%m-%d",
  date_fields = c("eventdate", "sysdate", "lcd", "uts", "frd", "crd", "tod",
  "deathdate"), integer_fields = c("yob", "practid"))
```

Arguments

dat	a dataframe
origin	ISO string representation of the dat of origin. default is UNIX start date
format	character: format of the date string. Default is ISO standard
date_fields	character vector of column names representing dates
integer_fields	character vector of column names that should be integers

Value

dataframe

`convert_dates` *converts date fields from ISO character string format to R Date format*

Description

Date fields are determined by the date-fields element in the .ehr definition. Extra date fields can be added to the extras argument or by setting '.ehr$date_fields'.

Usage

```
convert_dates(dat, extras = NULL)
```

Arguments

`dat`	a dataframe
`extras`	= a character vector of extra columns to convert or NULL for no extras

See Also

get_EHR_value set_EHR_value

`cprd_uniform_hba1c_values`
Transforms hba1c values to mmol/mol

Description

This function converts hba1c and fructosamine values to comon mmol/mol values conversions: mmol/L, mg/dL, fructosamine Assumes doctor miscoding for values < 20 after previous transformations

Usage

```
cprd_uniform_hba1c_values(df)
```

Arguments

`df`	a dataframe of hba1c test scores

Details

The input dataframe must include the data2, data3 and enttype columns and the enttype must equal either 275 (HbA1C) or 356 (Fructosamine)

Value

dataframe with all hba1c values recoded to mmol/mol

cut_tv *cut_tv - Cuts a survival dataset on a time varying variable*

Description

Survival datasets often have time-varying covariates that need to be dealt with. For example a drug exposure may occur after the entry into the cohort and you are interested in how this might affect your outcome.

Usage

```
cut_tv(.data, entry, exit, cut_var, tv_name, cores = 1, id_var,
  on_existing = c("flip", "increment"))
```

Arguments

.data	a dataframe
entry	name of the column in .data that defines entry time to cohort. Column must be numeric.
exit	name of the column in .data that defines exit time from cohort. Column must be numeric
cut_var	name of the column in .data that defines the time of the time-varying covariate event. Column must be numeric.
tv_name	name for the constructed time-varying covariate
cores	number of mc.cores to use.
id_var	name of the variable identifying individual cases
on_existing	see details for cutting behaviour

Details

This function cuts up a dataset based on times supplied for the time-varying covariate. If there is already a variable for the time-varying covariate, you can chose to flip the existing values or increment them. This means the function can be called multiple times to, e.g. deal with drugs starting and stopping and also to deal with progression of treatment.

The function is faster than other cutting methods, does not require conversion to Lexis format, and can be parallelised for large datasets and chained with dply workflows. Arguments should not be quoted.

This function can deal with the following scenarios (see examples):

- "Binary chronic covariates"e.g. The time of diagnosis for a chronic (unresolvable) condition. This requires a single column variable of times from entry in the dataset
- "Binary covariates"e.g. times of starting and stopping medication. This requires more than one column variable in the dataset, one for each start or stop event. The state flips with each new change.
- "Incremental time-varying covariates"e.g. different stages of a condition. This requires a single column variable for each incremental stage
- "Any combination of the above"This is achieved by chaining multiple calls together

Examples

```
# A simple example dataset to be cut
tv_test <- data.frame(id = 1:5, start = rep(0, 5), end = c(1000, 689, 1000, 874, 777),
                      event = c(0,1,0,1,1), drug_1 = c(NA, NA, NA, 340, 460),
                      drug_2 = c(NA, 234, 554, 123, NA),
                      drug_3_start = c(110, 110,111, 109, 110),
                      drug_3_stop = c(400, 400, 400, 400, 400),
                      stage_1 = c(300, NA, NA, NA, NA),
                      stage_2 = c(450, NA, NA, NA, NA))

# Binary chronic covariates:
tv_out1 <- cut_tv(tv_test, start, end, drug_1, id_var = id, drug_1_state)
tv_out1 <- cut_tv(tv_out1, start, end, drug_2, id_var = id, drug_2_state)
# Binary covariates:
tv_out3 <- cut_tv(tv_test, start, end, drug_3_start, id_var = id, drug_3_state)
tv_out3 <- cut_tv(tv_out3, start, end, drug_3_stop, id_var = id, drug_3_state)
# incremental covariates:
inc_1 <- cut_tv(tv_test, start, end, stage_1, id_var = id, disease_stage, on_existing = "
inc_1 <- cut_tv(inc_1, start, end, stage_2, id_var = id, disease_stage, on_existing = "in
# Chaining combinations of the above
## Not run:
library(dplyr)
tv_all <- tv_test %>%
          cut_tv(start, end, drug_1, id_var = id, drug_1_state) %>%
          cut_tv(start, end, drug_2, id_var = id, drug_2_state) %>%
          cut_tv(start, end, drug_3_start, id_var = id, drug_3_state) %>%
          cut_tv(start, end, drug_3_stop, id_var = id, drug_3_state) %>%
          cut_tv(start, end, stage_1, id_var = id, disease_stage, on_existing = "inc") %>
          cut_tv(start, end, stage_2, id_var = id, disease_stage, on_existing = "inc")

## End(Not run)
```

---

database *Wrapper for dbConnect*

Description

Connects to a SQLite database or creates one if it does not already exist

Usage

```
database(dbname)
```

Arguments

dbname	character name path to database file

Details

If the '.sqlite' file extension is ommited from the dbname argument it is automatically added.

Value

SQLiteConnection object

Examples

```
## Not run:
db <- database("mydb")

## End(Not run)
```

`define_EHR` *Construct an EHR_definition object.*

Description

This function creates the object that defines EHR simulations

Usage

```
define_EHR(start_date = "1930-01-01", end_date = "2014-06-30",
  patient = list(num = 10000, comorbidity = list(codes = NULL, prevalence =
  NULL), sim_params = list(transfer_out_prob = 0.2, scale = 25000, weibull_shape
  = 1, censor_type = "noninformative", betas = list(conditions = NULL, gender =
  log(0.7), baseline = 0.01, transfer_out = 3.5))),
  consultation = list(per_year = 2, type = list(code = 0:1, prob = c(0.2,
  0.8))), clinical = list(mean_events = 1), referral = list(mean_events =
  0.1), therapy = list(mean_events = 1), practice = list(num = 100, regions
  = 13, imd_cats = 5, early_lcd_prob = 0.1, early_lcd_range = 2, uts_limit =
  as.Date("1998-01-01"), late_uts_prob = 0.5, late_uts_range = 3))
```

Arguments

`start_date`	character date for earliest birthday of patients in EMR ("%Y-%m-%d")
`end_date`	character date for latest data collection date ("%Y-%m-%d")
`patient`	list of definitions for patient file (see details)
`consultation`	list of definitions for consultation file (see details)
`clinical`	list of definitions for clinical file (see details)
`referral`	list of definitions for referral file (see details)
`therapy`	list of definitions for therapy file (see details)
`practice`	list of definitions for practice file (see details)

Details

The arguments for this constructor function are complex:

Patient element

This is a list containing:

- num number of patients in the EHR simulation
- Comorbidity a list containing :
  - codes a named list of data frames of clinical codes in www.clinicalcodes.org export format

  - prevalence a named list of prevalences for the comorbidities
  - sim_params a list containing:
    * transfer_out_prob - probability of a patient transferring out early
    * scale - scaling up parameter for survival rates
    * weibull_shape - parameter for survival rates
    * censor_type type of censoring employed by `link{surv_sims}`
    * betas - list of values for conditions, gender, baseline and transfer out hazards. The betas element is a named vector of log hazards of length == length(comorbidity$codes)
- to do.

`definition_search` *This function is used to build new definition lists based on medical definitions*

Description

This function is used to build new definition lists based on medical definitions

Usage

```
definition_search(def, medical_table = NULL, test_table = NULL,
  drug_table = NULL, lookup = NULL)
```

Arguments

`def`	an object of class MedicalDefinition
`medical_table`	Dataframe lookup table of clinical codes
`test_table`	dataframe lookup table of test codes
`drug_table`	dataframe lookup table of medication product codes
`lookup`	list containing elements: "codes", "terms", "tests", "drugs", "drugcodes" (see details)

Details

You may get an invalid multibyte string error, in which case, set fileEncoding="latin1" on read.delim when reading in the lookup tables Lookup tables are

Examples

```
## Not run:
medical_table <- read.delim("medical.txt", fileEncoding="latin1", stringsAsFactors = FALS
drug_table <- read.delim("product.txt", fileEncoding="latin1", stringsAsFactors = FALSE)
def2 <- import_definition_lists(system.file("extdata", "example_search.csv",
                                            package = "rpcdsearch"))
draft_lists <- definition_search(def2, medical_table, drug_table = drug_table)

## End(Not run)
```

drop_all_temp_tables

drops all temporary tables from the database

Description

This is useful for temporary storage/memory management

Usage

```
drop_all_temp_tables(db)
```

Arguments

db	a database connection

drop_temp_table

Checks if a temporary table exists and then deletes if it does

Description

Checks if a temporary table exists and then deletes if it does

Usage

```
drop_temp_table(db, tab_name)
```

Arguments

db	a database connection
tab_name	character the name of the table of interest

ehr_def

An example EHR_definition object for defining parameters for simulating EHR data

Description

An example EHR_definition object for defining parameters for simulating EHR data

Usage

```
ehr_def
```

Format

object of type 'EHR_definition' including the following elements

patient list of parameters defining the simulation of the patient table

clinical list of parameters defining the simulation of the clinical table

therapy list of parameters defining the simulation of the therapy table

practice list of parameters defining the simulation of the patient table

start_date start_date for patients - earliest possible birthday

end_date Latest possible date in the ehr for data collection

---

entity *A sample of 6 clinical tests and meaures used in UK primary care*

Description

A sample of 6 clinical tests and meaures used in UK primary care

Usage

```
entity
```

Format

A dataframe with 6 observations:

enttype Unique internal code for the entity term

description Description of the code

filetype The table in the EHR the entity applies to

category Category of clinical entity

data1 first data variable

data2 second data variable

data3 third data variable

data4 fourth data variable

Source

http://www.clinicalcodes.org

expand_string *Reads strings and expands sections wrapped in dotted parentheses*

Description

This is a kind of inverse of bquote

Usage

```
expand_string(s, level = 3)
```

Arguments

s	a string
level	integer sets the parent frame level for evaluation

Examples

```
a <- runif(10)
 expand_string("The r code is .(a)")
```

export_definition_search *Exports definition searches to an excel file*

Description

Exports definition searches to an excel file

Usage

```
export_definition_search(definition_search, out_file)
```

Arguments

definition_search	a list of dataframes as produced by build_definition_lists
out_file	file path to the excel file to be exported

Examples

```
## Not run:
medical_table <- read.delim("medical.txt", fileEncoding="latin1", stringsAsFactors = FALS
drug_table <- read.delim("product.txt", fileEncoding="latin1", stringsAsFactors = FALSE)
def2 <- import_definition_lists(system.file("extdata", "example_search.csv",
                                            package = "rpcdsearch"))
draft_lists <- definition_search(def2, medical_table, drug_table = drug_table)
out_file <- "def_searches.xlsx"
export_definition_search(draft_lists, out_file)

## End(Not run)
```

export_fn *Exports to a variety of formats based on the file type argument*

Description

Exports to a variety of formats based on the file type argument

Usage

```
export_fn(x, file, ...)
```

Arguments

x	object to be exported
file	character path to the file to be exported to
...	arguments to be passed to the export functions File type is based on the file suffix and can be one of "txt", "csv", "rda", "dta". dta files use foreign::write.dta. If a match is not found, the file is written to std.out

extract_keywords *Function to extract rows from a lookup table based on keywords*

Description

This function can be used to build draft clinical code lists based on a clinical or product lookup table and a set of keywords.

Usage

```
extract_keywords(lookup, keywords, keyword_field = "desc")
```

Arguments

lookup	a dataframe containing a lookup table
keywords	character vector containing the keyword terms to search for
keyword_field	character identifying the field in the lookup table to be searched for keywords

Details

See www.clinicalcodes.org for clinical code lists that have been used in previous studies

All keywords are collapsed together in an OR statement

Value

a data frame subsetted by keyword

Examples

```
## Not run:
keywords <- c('oral ulceration', 'mouth ulceration', 'aphthous ulceration',
'oral aphthous ulceration','oral ulcer[s]?', 'mouth ulcer[s]?', 'aphthous ulcer[s]?',
'aphthous stomatitis', "stomatitis", "aphthae", 'oral aphthous stomatitis',
'oral aphthous ulcers', 'recurrent oral ulcers', 'recurrent mouth ulcers',
'recurrent oral aphthous ulcers', 'recurrent aphthous ulcers', 'recurrent aphthous stomat
'recurrent oral aphthous stomatitis')
a <- extract_keywords(medical, keywords)

## End(Not run)
```

---

first_events *Selects the earliest event grouped by patient*

Description

This function runs a select_events() query and then groups by patient id and picks only the earliest event for each patient

Usage

```
first_events(db = NULL, tab, columns = "eventdate", where = NULL,
  sql_only = FALSE, group_column = "patid", date_column = "eventdate")
```

Arguments

db	A database connection object
tab	the database table to extract from
columns	The other columns to be extracted
where	sting representation of the selection criteria
sql_only	logical should the function just return a string of the SQL query?
group_column	column to group by. Default is patid
date_column	the column to sort by. default is eventdate

Value

a dataframe or a string representing an sql query

Examples

```
## Not run:
b1 <- first_events(db, tab = "Clinical", columns = c("eventdate", "medcode"),
where = "medcode %in% .(a$medcode)")
first_events(tab = "Clinical", columns = c("eventdate", "medcode"),
where = "medcode %in% c(1, 2, 3, 4)", sql_only = TRUE)

## End(Not run)
```

flat_files *Exports flat files from the database. One file per practice*

Description

Exports flat files from the database. One file per practice

Usage

```
flat_files(db, table = "Consultation", practice_table = "Practice", out_dir,
  file_type = c("txt", "csv", "rda", "dta"), ...)
```

Arguments

db	a database connection
table	character the table to be exported
practice_table	the table that the practice definitions can be found
out_dir	a directory to output to. This will be created if it does not already exist
file_type	the type of file to be saved. This can be one of "txt", "csv", "rda", "dta".
...	arguments to be passed to export_fn

Defaults to exporting consultation tables for use by `match_on_index()`. the full path to `out_dir` will be created if it does not already exist.

See Also

match_on_index export_fn

get_EHR_attribute *Return the value of an attribute in the .ehr environment*

Description

Return the value of an attribute in the .ehr environment

Usage

```
get_EHR_attribute(x = NULL)
```

Arguments

x	an attribute name

Examples

```
{
set_CPRD()
get_EHR_attribute()
get_EHR_attribute(patient_id)
}
```

`get_matches` *Find matched controls for a set of cases*

Description

This function will provide a set of matched controls for a given set of cases.

Usage

```
get_matches(cases, control_pool, n_controls, match_vars, extra_vars,
  extra_conditions = NULL, cores = 1, track = TRUE,
  tracker = function(case_num) ".", method = c("incidence_density",
  "exact"), diagnosis_date = NULL)
```

Arguments

`cases`	dataframe of cases
`control_pool`	dataframe of potential controls to be used for matching
`n_controls`	number of controls to match to each case
`match_vars`	character vector of variables in the dataframes to be used to perform the matching
`extra_vars`	character vector of other variables to be used in the matching to define other conditions
`extra_conditions`	a character vector of length 1 defining further restrictions on matching
`cores`	number of cpu cores to be used by multicore (windows users should leave set to 1)
`track`	logical should a dot be printed to std.out for each case?
`tracker`	function to track progress of the function (See details)
`method`	The method of selection of controls (see details)
`diagnosis_date`	character the name of the variable in the cases and control_pool datasets containing the date of diagnosis (or other event to base the IDM method on). If there is no diagnosis date for a patient, this should be represented by NA

Details

Setting method to "exact" means that the matched controls are removed from the control pool after each case has been matched. This makes this method not thread safe and so will only run on a single core (and more slowly). Setting method to "incidence_density" is thread safe as the same controls can be used for more than one case. See Richardson (2004) Occup Environ Med 2004;61:e59 doi:10.1136/oem.2004.014472 for a description of IDS matching. Also see the introduction vignette. The tracker variable allows for different outputs to track the progress of the function. This is currently set to ouput a dot for every case matched. A function can be added to the argument For a more verbose tracking, e.g. to track number of cases, set `tracker = function() paste0(case_num, ",")`

head.SQLiteConnection *head for* SQLiteConnection *object*

Description

If just a database connection is selected, returns a dataframe of table names If a table name is also supplied, the first n rows from this table are output

Usage

```
## S3 method for class 'SQLiteConnection'
head(x, table = NULL, n = 6L, temp = FALSE,
  ...)
```

Arguments

x	A SQLiteConnection object
table	character specifying a table
n	integer: Number of rows to output
temp	logical should the function list the temp tables
...	Additional arguments

import_CPRD_data *Imports all selected CPRD data into an sqlite database*

Description

This function can import from both cohorts downloaded via the CPRD online tool and CPRD GOLD builds

Usage

```
import_CPRD_data(db, data_dir, filetypes = c("Additional", "Clinical",
  "Consultation", "Immunisation", "Patient", "Practice", "Referral", "Staff",
  "Test", "Therapy"), dateformat = "%d/%m/%Y", yob_origin = 1800,
  regex = "PET", recursive = TRUE, ...)
```

Arguments

db	a database connection
data_dir	the directory containing the CPRD cohort data
filetypes	character vector of filetypes to be imported
dateformat	the format that dates are stored in the CPRD data. If this is wrong it won't break but all dates are likely to be NA
yob_origin	value to add yob values to to get actual year of birth (Generally 1800)
regex	character regular expression to identify data files in the directory. This is separated from the filetype by an underscore. e.g. 'p[0-9]3' in CPRD GOLD
recursive	logical should files be searched for recursively under the data_dir?
...	arguments to be passed to add_to_database

Details

Note that if you chose to import all the filetype, you may end up with a very large database file. You may then chose only to import the files you want to use. You can always import the rest of the files later. This function may take a long time to process because it unzips (potentially large) files, reads into R where it converts the date formats before importing to SQLite. However, this initial data preparation step will greatly accelerate downstream processing.

---

import_definitions *Imports definitions to be searched from a csv file into a MedicalDefinition object*

Description

Imports definitions to be searched from a csv file into a MedicalDefinition object

Usage

```
import_definitions(input_file)
```

Arguments

input_file	character path to the input file

Examples

```
def2 <- import_definitions(system.file("extdata", "example_search.csv",
                                       package = "rpcdsearch"))
```

---

last_events *Selects the earliest event grouped by patient*

Description

This function runs a select_events() query and then groups by patient id and picks only the latest event for each patient

Usage

```
last_events(db = NULL, tab, columns = "eventdate", where = NULL,
  sql_only = FALSE, group_column = "patid", date_column = "eventdate")
```

Arguments

db	A database connection object
tab	the database table to extract from
columns	The other columns to be extracted
where	sting representation of the selection criteria
sql_only	logical should the function just return a string of the SQL query?
group_column	column to group by. Default is patid
date_column	the column to sort by. default is eventdate

Value

a dataframe or a string representing an sql query

Examples

```
## Not run:
b2 <- last_events(db, tab = "Clinical", other_columns = c("eventdate", "medcode"),
where = "medcode %in% .(a$medcode)")

## End(Not run)
```

---

list_EHR_attributes *Lists all of the EHR attribute names in .ehr*

Description

Lists all of the EHR attribute names in .ehr

Usage

```
list_EHR_attributes()
```

---

match_case *Selected controls matching a list of variables from a case*

Description

Helper function for get_matches() This function wil perform incidence density sampling or exact sampling up to a supplied number of matched controls

Usage

```
match_case(matcher, control_pool, n_controls, id, replace)
```

Arguments

matcher	list of character strings defining the matching conditions
control_pool	dataframe of potential controls for matching
n_controls	The number of controls for each case. If replace == FALSE this is a maximum value
id	named vector of length 1 for the variable name and value of the case identifier
replace	logical should sampling of matched controls be with replacement (incident density sampling) or not.

Value

A dataframe of matched controls or NULL if no controls could be found

`match_on_index` *Function for performing matching of controls to cases using the consultation files to generate a dummy index date for controls.*

Description

Controls are matched on an arbitrary number of categrorical variables and on continuous variables via the `extra_conditions` argument. Also the date at `index_var` is matched to the eventdate in the consultation files, providing a dummy index date for controls of a consultaton within +/- `index_diff_limit` days of the index date.

Usage

```
match_on_index(cases, control_pool, index_var, match_vars,
  extra_conditions = "", index_diff_limit = 90, consult_path,
  n_controls = 5, cores = 1, import_fn = read.delim, ...)
```

Arguments

`cases`	A dataframe of cases to which to match controls
`control_pool`	A dataframe of possible contols to match to cases
`index_var`	character string of the name of the variable containing index dates
`match_vars`	character vector detailing the common variables in `cases` and `control_pool` to match on
`extra_conditions`	character string detailing other matching constraints (see details)
`index_diff_limit`	integer number of days before or after the case index date that dummy index dates can be picked from the consultation files
`consult_path`	path to directory containing consultation files
`n_controls`	integer the number of controls to attempt to match to each case
`cores`	integer the number of processor cores to be used in processing
`import_fn`	function name stipulating the function used to read the consultation files
`...`	extra arguments to be passed to import_fn

Details

Note that the consultaton files must be in flat-file format (i.e. not as part of the database, but as text (or other filetype, e.g stata dta) files). Set the `import_fn` argument to use different file formats (e.g. `foreign::read.dta` or `readstata13::read.dta13`)

The `extra_conditions` argument can add extra condtions to the matching criteria on top of the matching vars for example you could add "year > 1990". You can wrap calls to expressions in dotted brackets to automatically expand them. This is particularly useful when you want to find the value for each individual case. Each case is denoted by `CASE` e.g. "start_date < .(CASE$start_date)" will ensure the start date for controls is prior to the start date for the matched case.

Value

a dataframe of matched controls

medcodes_to_read *Translate CPRD medcodes to Read/Oxmis*

Description

This function accepts a data frame with a column for CPRD medcodes and merges with a medical lookup table to give columns for Read/OXMIS codes and optional descriptions

Usage

```
medcodes_to_read(medcodes_data, lookup_table, medcodes_name = "medcode",
  lookup_readcodes = "readcode", lookup_medcodes = "medcode",
  description = TRUE)
```

Arguments

medcodes_data	a dataframe with a column matching medcodes_name
lookup_table	a dataframe with columns matching lookup_readcodes and lookup_medcodes
medcodes_name	character name of the CPRD medcodes column in medcodes_data
lookup_readcodes	character name of the Read codes column in the lookup_table
lookup_medcodes	character name of the CPRD medcodes column in the lookup_table
description	logical Should description and other categories from the lookup table also be included?

Details

Note that if the names of the medcodes columns are different in the data and the lookup table, the name in the data is retained To maintain sanity, a warning will be given to inform of are any name conflicts between the input data and the lookup

Value

a data frame matching the input medcodes_data with the Read codes and optional description columns merged in.

MedicalDefinition *Constructor function for MedicalDefinition class*

Description

Constructor function for MedicalDefinition class

Usage

```
MedicalDefinition(terms = NULL, codes = NULL, tests = NULL,
  drugs = NULL, drugcodes = NULL)
```

Arguments

terms	list of character vectors or NULL
codes	list of character vectors or NULL
tests	list of character vectors or NULL
drugs	list of character vectors or NULL
drugcodes	list of character vectors or NULL

Details

Elements marked with a "-" are excluded. Elements marked with a "r

Examples

```
def <- MedicalDefinition(terms = list(c("angina", "unstable"), c("angina", "Crescendo "),
                                      c("angina", "Refractory")),
                         codes = list("G33..00", "G330.00", "%r212H", "-G617"))
class(def)
```

---

patients_in_window *Select patients alive and registered between certain dates*

Description

This function selects patients from the patient table of a CPRD database who are alive and registered within a supplied window

Usage

```
patients_in_window(db, startdate, enddate, qs = TRUE,
  registration_buffer = 0, patient_tablename = "Patient")
```

Arguments

db	a database connection
startdate	character for the start of the window. format %Y-%m-%d
enddate	character for the end of the window. format %Y-%m-%d
qs	logical should only patients deemed to be of acceptable quality standard be selected? Most downloaded cohorts will be up to standard by default
registration_buffer	numeric how many days must patients be registered for prior to the startdate to be included? Setting a posive value can reduce information bias
patient_tablename	The name of the patient table. default is Patient

Details

criteria are that crd is before start of window, tod is after end of window deathdate is after end of window

`patients_per_medcode` *Produce a dataset of CPRD medcodes with frequencies of patients in the clinical table*

Description

This function aggregates all distinct patients matching each CPRD medcode in the clinical table

Usage

```
patients_per_medcode(db, clinical_table = "Clinical", patid = "patid",
  medcode = "medcode")
```

Arguments

`db`	a database connection
`clinical_table`	name of the clinical table in the database
`patid`	name of the patid field
`medcode`	name of the medcode field

Details

Note that this does not translate to Read/OXMIS codes. This function should be fast because all of the heavy lifting happens in SQLite before the data is exported to R

Examples

```
## Not run:
medcode_counts <- patients_per_medcode(db)
head(medcode_counts)

## End(Not run)
```

`prev_terms` *This function adds columns enabling one to calculate numerators and denominators for prevalence and incidence.*

Description

See the vignette for more details.

Usage

```
prev_terms(dat, event_date = "eventdate", year_fn = standard_years)
```

Arguments

dat	dataframe of longitudinal data
event_date	character name of the column used to identify clinical events
year_fn	function that determines how year start and end dates are calculated

Value

longitudinal data frame with incidence, prevalence and followup columns

---

prev_totals *Calculates the prevalence totals for the output of a data frame of events/patients etc.*

Description

e.g. Run on the output of a call to prev_terms

Usage

```
prev_totals(dat, included_totals = c("year", .ehr$practice_id),
  time_var = "year", person_years = 100)
```

Arguments

dat	a dataframe
included_totals	character vector describing which aggregates should be included e.g. c("year", "practid")
time_var	name of the variable determining timepoints
person_years	numeric multiplier for presentation of prevalence and incidence numbers

Details

Outputs totals by aggregated by a time variable and other variables Updated to use dplyr to do the aggregation - Now 40x faster!

Value

list of aggregates and/or the original data

`print.EHR_definition` *Tools for describing EMR_description objects.*

Description

Tools for describing EMR_description objects.

Usage

```
## S3 method for class 'EHR_definition'
print(x, element = NULL, level = 3, ...)
```

Arguments

`x`	A `EHR_definition` object
`element`	an element name
`level`	nesting level for display purposes
`...`	Additional arguments

`print.MedicalDefinition` *Basic print method for medical definition classes*

Description

Basic print method for medical definition classes

Usage

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

Arguments

`x`	an object of class "medical_definition"
`...`	Potential further arguments (required for method/generic reasons)

product *A sample of 500 medicines used in UK primary care*

Description

A sample of 500 medicines used in UK primary care

Usage

```
product
```

Format

A dataframe with 500 observations:

prodcode Unique internal code for the entity term

productname Description of the code

bnfcode BNF code for the medicine

bnfchapter BNF chapter heading

Source

http://www.bnf.org/

qof_15_months *Helper function providing startdate and enddate for a given year*

Description

Start and end dates matching QOF year start/ends

Usage

```
qof_15_months(year)
```

Arguments

year	integer

qof_years — *Helper function providing startdate and enddate for a given year*

Description

Start and end dates matching QOF year start/ends

Usage

```
qof_years(year)
```

Arguments

year	integer

random_dates — *Generates random dates between a start and end day.*

Description

dates are in usual R as.Date() format

Usage

```
random_dates(n, start_day, end_day)
```

Arguments

n	Number of dates to be returned
start_day	string representation of a start day
end_day	string representation of a start day

Details

Enter start and end dates in ISO format, e.g. "2001-02-04"

Value

a vector of dates

read_to_medcodes *Translate Read/Oxmis codes to CPRD medcodes*

Description

This function accepts a data frame with a column for Read/Oxmis codes and merges with a medical lookup table to give columns for CPRD medcodes and optional descriptions

Usage

```
read_to_medcodes(readcodes_data, lookup_table, readcodes_name = "readcode",
  lookup_readcodes = "readcode", lookup_medcodes = "medcode", description)
```

Arguments

readcodes_data	a dataframe with a column matching medcodes_name
lookup_table	a dataframe with columns matching lookup_readcodes and lookup_medcodes
readcodes_name	character name of the Read codes column in readcodes_data
lookup_readcodes	character name of the Read codes column in the lookup_table
lookup_medcodes	character name of the CPRD medcodes column in the lookup_table
description	logical Should description and other categories from the lookup table also be included?

Details

Note that if the names of the Read/Oxmis codes columns are different in the data and the lookup table, the name in the data is retained To maintain sanity, a warning will be given to inform of are any name conflicts between the input data and the lookup

Value

a data frame matching the input medcodes_data with the Read codes and optional description columns merged in.

read_zip *Reads a zipped data file to a dataframe*

Description

This function will unzip a zipped text file and read it in to an R data frame

Usage

```
read_zip(file, ...)
```

Arguments

`file`	character a file to read in
`...`	extra arguments to pass to read.delim

Details

Default behaviour is to read in as a standard read.delim call. extra arguments to read.delim can be passed to the function

Value

a dataframe

`rEHR` *The rEHR package.*

Description

The rEHR package.

`repsample_example` *An example dataset to demonstrate the repsample function. 2474 theroetical UK GP Practices.*

Description

An example dataset to demonstrate the repsample function. 2474 theroetical UK GP Practices.

Usage

```
repsample_example
```

Format

A dataframe with 2474 observations:

practicecode ID for the GP practice
postcode postcode for the practice
shacode Strategic Health Authorty Code
shaname Strategic Health Authorty name
pctcode Primary care trust code
pctname Primary care trust name
listsize number of patients registered at practice
ftes ftes
soaimd07 2007 IMD deprivation score
ruralvar Rurality

Source

http://www.jstatsoft.org/v55/c01/paper

`select_by_year` *Runs a series of selects over a year range and collects in a list of dataframes*

Description

This function applies a database select over a range of years and outputs as a list or a dataframe The function can be parallelised using `parallel`.

Usage

```
select_by_year(dbname = NULL, db = NULL, tables, columns = "*", where,
  year_range, year_fn = qof_years, as_list = FALSE,
  selector_fn = select_events, cores = 1L, ...)
```

Arguments

`dbname`	path to the database file
`db`	a database connection
`tables`	character vector of table names
`columns`	character vector of columns to be selected from the tables
`where`	character string representation of the selection criteria
`year_range`	integer vector of years to be queried
`year_fn`	function that determines how year start and end dates are calculated
`as_list`	logical: Should the results be returned as a list of years? If not, the data is collapsed into a dataframe
`selector_fn`	function to select from the database. See notes.
`cores`	integer: The number of processor cores available.
`...`	extra arguments to be passed to the `selector_fn`

Details

Because the same database connection cannot be used across threads, the input is a path to a database rather than a database connection itself and a new connection is made with every fork.

`columns` can take a character vector of arbitrary length. This means you can use it to insert SQL clauses e.g. "DISTINCT patid".

Year start and year end criteria can be added to the where argument as 'STARTDATE' and 'ENDDATE'. These will get translated to the correct start and end dates specified by year_fn

Note that if you are working with temprary tables, you need to set `cores` to 1 and specify the open database connection with db This is because the use of `mclapply` means that new database connections need to be started for each fork and temporary files can only be seen inside the same connection

The `selector_fn` argument determines how the database select operates. Default is the `select_events` function. Alternatives are `first_events` and `last_events`

Examples

```
## Not run:
# Output from a single table
where_q <- "crd < STARTDATE & (is.null(tod) | tod > ENDDATE) & accept == 1"
ayears <- select_by_year(db, "Patient", columns = c("patid", "yob", "tod"),
                         where = where_q, year_range = 2000:2003)
# Output from multiple tables
load("data/medical.RData")
a <- read.csv("data/chronic-renal-disease.csv")
a <- read_to_medcodes(a, medical, "code", lookup_readcodes= "readcode",
                      lookup_medcodes="medcode", description = T)
where_q <- "eventdate >= STARTDATE & eventdate <= ENDDATE & medcode %in% .(a$medcode)"
byears <- byears <- select_by_year("~/rOpenHealth/CPRD_test/Coupland/Coupland",
                                   c("Clinical", "Referral"),
columns = c("patid", "eventdate", "medcode"),
where = where_q, year_range = 2000:2003, as_list = FALSE,
cores = 10)

## End(Not run)
```

---

select_events *Extracts From the database*

Description

This is a generic function for extracting EHR data from the database

Usage

```
select_events(db = NULL, tab, columns = "*", where = NULL,
  sql_only = FALSE, convert_dates = FALSE)
```

Arguments

db	a database connection
tab	the database table to extract from
columns	character vector of columns to extract from the table "*" means all tables
where	sting representation of the selection criteria
sql_only	logical should the function just return a string of the SQL query?
convert_dates	logical should date fields be converted to R date format?

Details

The function is the base function for a range of others It can either extract by itself or generate the SQL to make a query. In this way it can be combined to make compound queries. The where argument is equivalent to the WHERE clause in sql The elements are converted to SQL using dplyr::translate_sql_q If an element is wrapped in a '.()', the element is expanded. Dates should be entered as strings in ISO format (

Value

a dataframe or a string representing an sql query

Examples

```
## Not run:
# medical lookup tables are provided with CPRD
load("data/medical.RData")
a <- read.csv("data/chronic-renal-disease.csv")
a <- read_to_medcodes(a, medical, "code", lookup_readcodes= "readcode",
lookup_medcodes="medcode", description = T)
b <- select_events(db, tab = "Referral", columns = c("patid", "eventdate", "medcode"),
where = "medcode %in% .(a$medcode) & eventdate < '2000-01-01'")
b1 <- select_events(db, tab = "Clinical", columns = c("patid", "eventdate", "medcode"),
where = "medcode %in% .(a$medcode) & eventdate < '2000-01-01'")

## End(Not run)
```

---

set_CPRD *Sets EHR metadata to CPRD format When this is run, most functions in rEHR act as though the EHR database is CPRD*

Description

Sets EHR metadata to CPRD format When this is run, most functions in rEHR act as though the EHR database is CPRD

Usage

```
set_CPRD()
```

Details

Metadata on EHR type is stored in the .ehr environment. This allows the same functions to work across different data sources. The .ehr environment is not desgined to be accessible to the user, but accessor functions are provided. CPRD is the default EHR setting.

See Also

get_EHR_value set_EHR_value

set_EHR_attribute *Sets the value of an attribute in the .ehr environment*

Description

Sets the value of an attribute in the .ehr environment

Usage

```
set_EHR_attribute(x, value)
```

Arguments

x	an ehr attribute name
value	the value to set to the attribute

Examples

```
set_CPRD()
set_EHR_attribute(practice_id, "pracid")
```

simulate_ehr_consultations *Generates simulated GP consultation tables.*

Description

This function generates simulated GP consultations based on an EHR_definition object and a patient table, as generated by simulate_ehr_patients. Multicore functionality is implemented via mclapply

Usage

```
simulate_ehr_consultations(ehr_def, patient_table, cores = 1)
```

Arguments

ehr_def	an object of class link{EHR_definition}
patient_table	a dataframe of simulated patient EHR data
cores	number of processor cores to use to run the analysis

Value

data frame of simulated GP consultations

Examples

```
## Not run:  patient <- simulate_ehr_patients(ehr_definition) cons <-
  simulate_ehr_consultations(ehr_def, patient_table = patient, cores = 4)
## End(Not run)
```

simulate_ehr_events

Generate simulated events tables

Description

This function can generate events for clinical, referral and therapy tables. These are based on the consultation tables generated by simulate_ehr_consultations.

Usage

```
simulate_ehr_events(ehr_def, consultation, event_type = c("clinical",
  "referral", "therapy"), cores = 1, therapy_lookup = NULL)
```

Arguments

ehr_def	an object of class link{EHR_definition}
consultation	a dataframe of simulated patient consultations
event_type	Type of events to be generated
cores	number of processor cores to use in generating the data
therapy_lookup	lookup table for drug therapy events e.g. link{product}

Details

This function is relatively basic - for clinical and referral tables, it generates events according to the comorbidities defined in the ehr_def, with the mean number of events for each consultation being defined in the ehr_def for that event_type. For therapy events, the function simply samples the therapy_lookup table, with the mean number of events for each consultation being defined in the ehr_def for therapy. Therefore, at the moment, the therapies bear no relationship to the conditions the patient has and are only for the purposes of explaining the functioning of the package. The random sampling is based on a poisson distribution

Value

dataframe

simulate_ehr_patients

Generate a dataframe of simulated patients with exit dates based on presented comorbidities.

Description

The definitions of the patient file are all in the ehr_def object

Usage

```
simulate_ehr_patients(ehr_def)
```

Arguments

ehr_def	an object of class EHR_definition

Details

Patients must have transferred out after the ealiest possible collection date ehr_def$practice$uts_limit

Value

a dataframe of simulated patients

---

simulate_ehr_practices

generates a simulated dataframe of primary care practices in the same format as is used in CPRD

Description

generates a simulated dataframe of primary care practices in the same format as is used in CPRD

Usage

```
simulate_ehr_practices(ehr_def)
```

Arguments

ehr_def	an object of class ehr_def

Details

The definitions of the practice file are all in the ehr_def object

Value

a dataframe of simulated practices

---

standard_years

Helper function providing startdate and enddate for a given year

---

Description

Standard years

Usage

```
standard_years(year)
```

Arguments

year	integer

surv_sims *Function to simulate survival data.*

Description

Model: proportional hazards, h(t; cov_mat, beta) = exp(cov_mat indicators for Type I censoring (common censoring time 'tc').

Usage

```
surv_sims(cov_mat, beta, cens_type = c("typeI", "noninformative"),
  baseline_hazard, cens_hazard = 0.04, cens_prob = 0, scale = 1,
  weibull_shape = 1)
```

Arguments

cov_mat	n x p matrix of cov_matiates
beta	p-vector of regression coefficients
cens_type	typeI censoring or non-informative based on exponential distribution
baseline_hazard	for modelling death dates
cens_hazard	log(hazard) for non-informative censoring
cens_prob	expected censoring fraction (0 <= cens_prob < 1). Used for typeI censoring
scale	value to scale up the time variable by
weibull_shape	shape parameter for the weibull distribution. 1 is the same as an exponential

Details

Weibull_shape is the k (shape) parameter from a weibull distribution

- A value of k < 1 indicates that the mortality rate decreases over time. This happens if there is significant infant mortality
- A value of k = 1 indicates that the mortality rate is constant over time. This might suggest random external events are causing mortality. This is the same as an exponential distribution
- A value of k > 1 indicates that the mortality rate increases with time. This happens if there is an aging process.

Value

Censored exponential survival times and censoring

temp_location *Sets location of the db temporary store for temporary tables*

Description

By default, sqlite stores temp tables in /tmp (Or windows equivalent). If you are building large temporary tables and don't have a large /tmp directory, you can get "database or disk is full" errors. If you have a lot of RAM you can set `store` to "RAM" and the temp files will be stored in RAM rather than in /tmp. This could also speed things up.

Usage

```
temp_location(db, store = c("tmp", "RAM"))
```

Arguments

`db`	a database connection
`store`	character vector either "tmp" or "RAM"

temp_table *Creates a temporary table in the database*

Description

This function is useful if most of your work is on a subset of the database

Usage

```
temp_table(db, tab_name, select_query)
```

Arguments

`db`	a database connection object
`tab_name`	character name for the teporary table
`select_query`	character the query that specifies the temporary table

Details

The table will exist for as long as the database connection is kept open The Select_query argument will take the output from a select_events(sql_only = TRUE) based function

Examples

```
## Not run:
db <- database("myCPRDdb")
temp_table(db, tab_name = "post_2005",
           select_query = select_events(db, tab = "Referral",
                                        columns = c("patid", "eventdate", "medcode"),
                                        where = "eventdate > '2005-01-01'",
                                        sql_only = TRUE))

## End(Not run)
```

`to_stata` *Compresses a dataframe and saves in stata format. Options to save as Stata 12 or 13.*

Description

Automatically compresses data to reduce file size

Usage

```
to_stata(dat, fname, stata13 = FALSE, ...)
```

Arguments

`dat`	dataframe
`fname`	character string: filepath to save to
`stata13`	logical Save as Stata13 compatible format?
`...`	arguments to be passed to compress

Details

Defaults to saving compressed dates to integer days from 1960-01-01 which is the standard in stata.

`to_temp_table` *Send a dataframe to a temporary table in the database*

Description

The table is a temporary database and is linked only to the current connection object

Usage

```
to_temp_table(db, tab_name, dat, overwrite = FALSE)
```

Arguments

`db`	a database connection
`tab_name`	character name for the new temporary database table
`dat`	dataframe to send to the temporary database table
`overwrite`	logical if a table already exists with the same name should it be dropped?

`wrap_sql_query` *combines strings and vectors in a sensible way for select queries*

Description

This function is a variant of the sprintf function. In the query, can be placed identifier tags which are a hash character followed by a number e.g. #1 The number in the tag reflects the position of the arguments after the query The resut of evaluating that argument will then be inserted in place of the tag. If the result of evaluating the argument is a vector of length 1, it is inserted as is. If it is a vector of length > 1, it is wrapped in parentheses and comma separated.

Usage

```
wrap_sql_query(query, ...)
```

Arguments

`query`	a character string with identifier tags (#[number]) for selecting the argument in ...
`...`	optional arguments selected by the identifier tags

Details

Note that this function is for help in constructing raw SQL queries and should not be used as an input to the `where` argument in `select_event` calls. This is because these calls use translate_sql_q to translate from R code to SQL

Examples

```
medcodes1 <- 1:5
practice <- 255
wrap_sql_query("eventdate >= STARTDATE & eventdate <= ENDDATE & medcode %in% #1 &
   practice == #2", medcodes1, practice)
```

Index