Package 'EHR'

December 19, 2022

Version 0.4-11

Date 2022-12-15

Title Electronic Health Record (EHR) Data Processing and Analysis Tool

Maintainer Leena Choi <leena.choi@vanderbilt.edu>

Description Process and analyze electronic health record (EHR) data. The 'EHR' package provides modules to perform diverse medication-related studies using data from EHR databases. Especially, the package includes modules to perform pharmacokinetic/pharmacodynamic (PK/PD) analyses using EHRs, as outlined in Choi, Beck, McNeer, Weeks, Williams, James, Niu, Abou-Khalil, Birdwell, Roden, Stein, Bejan, Denny, and Van Driest (2020) <doi:10.1002/cpt.1787>. Additional modules will be added in future. In addition, this package provides various functions useful to perform Phenome Wide Association Study (PheWAS) to explore associations between drug exposure and phenotypes obtained from EHR data, as outlined in Choi, Carroll, Beck, Mosley, Roden, Denny, and Van Driest (2018) <doi:10.1093/bioinformatics/bty306>.

Depends R (>= 2.10)

License GPL (>= 3)

URL https://choileena.github.io/

Imports stats, utils, data.table, methods, lubridate, pkdata

Suggests glmnet, logistf, medExtractR (>= 0.4.1), knitr, rmarkdown, ggplot2, markdown

NeedsCompilation no

RoxygenNote 7.1.1

VignetteBuilder knitr

Author Leena Choi [aut, cre] (<https://orcid.org/0000-0002-2544-7090>), Cole Beck [aut] (<https://orcid.org/0000-0002-6849-6255>), Hannah Weeks [aut] (<https://orcid.org/0000-0002-0262-6790>), Elizabeth McNeer [aut], Nathan James [aut] (<https://orcid.org/0000-0001-7079-9151>), Michael Williams [aut]

Repository CRAN

Date/Publication 2022-12-19 18:20:02 UTC

R topics documented:

EHR-package . . . 3
addLastDose . . . 4
analysisPheWAS . . . 5
buildDose . . . 7
collapseDose . . . 9
dataTransformation . . . 10
dd . . . 10
dd.baseline . . . 11
dd.baseline.small . . . 11
dd.small . . . 12
extractMed . . . 12
freqNum . . . 13
idCrosswalk . . . 14
lam_metadata . . . 15
lam_mxr_parsed . . . 16
Logistf . . . 17
makeDose . . . 18
parseCLAMP . . . 19
parseMedEx . . . 20
parseMedExtractR . . . 21
parseMedXN . . . 22
processLastDose . . . 23
pullFakeId . . . 24
pullRealId . . . 25
readTransform . . . 26
run_Build_PK_IV . . . 26
run_Build_PK_Oral . . . 30
run_Demo . . . 31
run_DrugLevel . . . 33
run_Labs . . . 35
run_MedStrI . . . 36
run_MedStrII . . . 39
stdzDose . . . 40
stdzDoseChange . . . 41
stdzDoseSchedule . . . 41
stdzDuration . . . 42
stdzFreq . . . 42
stdzRoute . . . 43
stdzStrength . . . 43
tac_lab . . . 44
tac_metadata . . . 44
tac_mxr_parsed . . . 45
zeroOneTable . . . 46

Index **47**

EHR-package *Electronic Health Record (EHR) Data Processing and Analysis Tool*

Description

The 'EHR' package provides modules to perform diverse medication-related studies using data from EHR databases.

Details

Package functionality:

- Process and analyze Electronic Health Record (EHR) data.
- Implement modules to perform diverse medication-related studies using data from EHR databases. Especially, the package includes modules to perform pharmacokinetic/pharmacodynamic (PK/PD) analyses using EHRs, as outlined in Choi et al. (2020).
- Implement three statistical methods for Phenome Wide Association Study (PheWAS). Contingency tables for many binary outcomes (e.g., phenotypes) and a binary covariate (e.g., drug exposure) can be efficiently generated by `zeroOneTable`, and three commonly used statistical methods to analyze data for PheWAS are implement by `analysisPheWAS`.

Author(s)

Maintainer: Leena Choi <leena.choi@vanderbilt.edu> (ORCID)

Authors:

- Cole Beck <cole.beck@vumc.org> (ORCID)
- Hannah Weeks <hannah.l.weeks@vanderbilt.edu> (ORCID)
- Elizabeth McNeer <elizabeth.mcneer@vumc.org>
- Nathan James <nathan.t.james@vanderbilt.edu> (ORCID)
- Michael Williams <michael.l.williams@vanderbilt.edu>

References

1. Development of a system for postmarketing population pharmacokinetic and pharmacodynamic studies using real-world data from electronic health records.
Choi L, Beck C, McNeer E, Weeks HL, Williams ML, James NT, Niu X, Abou-Khalil BW, Birdwell KA, Roden DM, Stein CM, Bejan CA, Denny JC, Van Driest SL.
Clin Pharmacol Ther. 2020 Apr;107(4):934-943. doi: 10.1002/cpt.1787.
2. Evaluating statistical approaches to leverage large clinical datasets for uncovering therapeutic and adverse medication effects.
Choi L, Carroll RJ, Beck C, Mosley JD, Roden DM, Denny JC, Van Driest SL.
Bioinformatics. 2018 Sep 1;34(17):2988-2996. doi: 10.1093/bioinformatics/bty306.

See Also

Useful links:

- https://choileena.github.io/

`addLastDose` *Add Lastdose Data*

Description

Add lastdose data to data set from the `buildDose` process.

Usage

```
addLastDose(buildData, lastdoseData)
```

Arguments

`buildData`	data.frame, output of `buildDose` function.
`lastdoseData`	data.frame with columns filename, ld_start, lastdose, raw_time, time_type

Details

Lastdose is a datetime string associated with dose data. Information on time of last dose can be extracted within the `extractMed` function (i.e., `medExtractR`) using the argument `lastdose=TRUE`. Raw extracted times should first be processed using the `processLastDose` function to convert to datetime format before providing to `addLastDose`. This function then combines the processed last dose times with output from the `buildDose` process by file name to pair last dose times with dosing regimens based on position. Alternatively, the user can provide their own table of lastdose data. In this case, with position information absent, the lastdose data should be restricted to one unique last dose time per unique patient ID-date identifier.

In the case where `lastdoseData` is output from `processLastDose`, it is possible to have more than one extracted last dose time. In this case, rules are applied to determine which time should be kept. First, we give preference to an explicit time expression (e.g., "10:30pm") over a duration expression (e.g., "14 hour level"). Then, we pair last dose times with drug regimens based on minimum distance between last dose time start position and drug name start position.

See EHR Vignette for Extract-Med and Pro-Med-NLP for details.

Value

a data.frame with the 'lastdose' column added.

Examples

```
# Get build data
data(tac_mxr_parsed)
# don't combine lastdose at this stage
tac_build <- buildDose(tac_mxr_parsed, preserve = 'lastdose')
# Get processed last dose data
tac_mxr <- read.csv(system.file("examples", "tac_mxr.csv", package = "EHR"))
data(tac_metadata)
data(tac_lab)
ld_data <- processLastDose(tac_mxr, tac_metadata, tac_lab)

addLastDose(tac_build, ld_data)
```

---

analysisPheWAS *Statistical Analysis for PheWAS*

Description

Implement three commonly used statistical methods to analyze data for Phenome Wide Association Study (PheWAS)

Usage

```
analysisPheWAS(
  method = c("firth", "glm", "lr"),
  adjust = c("PS", "demo", "PS.demo", "none"),
  Exposure,
  PS,
  demographics,
  phenotypes,
  data
)
```

Arguments

method	define the statistical analysis method from 'firth', 'glm', and 'lr'. 'firth': Firth's penalized-likelihood logistic regression; 'glm': logistic regression with Wald test, 'lr': logistic regression with likelihood ratio test.
adjust	define the adjustment method from 'PS','demo','PS.demo', and 'none'. 'PS': adjustment of PS only; 'demo': adjustment of demographics only; 'PS.demo': adjustment of PS and demographics; 'none': no adjustment.
Exposure	define the variable name of exposure variable.
PS	define the variable name of propensity score.
demographics	define the list of demographic variables.
phenotypes	define the list of phenotypes that need to be analyzed.
data	define the data.

Details

Implements three commonly used statistical methods to analyze the associations between exposure (e.g., drug exposure, genotypes) and various phenotypes in PheWAS. Firth's penalized-likelihood logistic regression is the default method to avoid the problem of separation in logistic regression, which is often a problem when analyzing sparse binary outcomes and exposure. Logistic regression with likelihood ratio test and conventional logistic regression with Wald test can be also performed.

Value

estimate	the estimate of log odds ratio.
stdError	the standard error.
statistic	the test statistic.
pvalue	the p-value.

Author(s)

Leena Choi <leena.choi@vanderbilt.edu> and Cole Beck <cole.beck@vumc.org>

Examples

```
## use small datasets to run this example
data(dataPheWASsmall)
## make dd.base with subset of covariates from baseline data (dd.baseline.small)
## or select covariates with upper code as shown below
upper.code.list <- unique(sub("[.][^.]*(.).*", "", colnames(dd.baseline.small)) )
upper.code.list <- intersect(upper.code.list, colnames(dd.baseline.small))
dd.base <- dd.baseline.small[, upper.code.list]
## perform regularized logistic regression to obtain propensity score (PS)
## to adjust for potential confounders at baseline
phenos <- setdiff(colnames(dd.base), c('id', 'exposure'))
data.x <- as.matrix(dd.base[, phenos])
glmnet.fit <- glmnet::cv.glmnet(x=data.x, y=dd.base[,'exposure'],
                                family="binomial", standardize=TRUE,
                                alpha=0.1)
dd.base$PS <- c(predict(glmnet.fit, data.x, s='lambda.min'))
data.ps <- dd.base[,c('id', 'PS')]
dd.all.ps <- merge(data.ps, dd.small, by='id')
demographics <- c('age', 'race', 'gender')
phenotypeList <- setdiff(colnames(dd.small), c('id','exposure','age','race','gender'))
## run with a subset of phenotypeList to get quicker results
phenotypeList.sub <- sample(phenotypeList, 5)
results.sub <- analysisPheWAS(method='firth', adjust='PS', Exposure='exposure',
                              PS='PS', demographics=demographics,
                              phenotypes=phenotypeList.sub, data=dd.all.ps)
## run with the full list of phenotype outcomes (i.e., phenotypeList)

        results <- analysisPheWAS(method='firth', adjust='PS',Exposure='exposure',
                          PS='PS', demographics=demographics,
                          phenotypes=phenotypeList, data=dd.all.ps)
```

buildDose *Combine Dose Data*

Description

Output from parse process is taken and converted into a wide format, grouping drug entity information together based on various steps and rules.

Usage

```
buildDose(
  dat,
  dn = NULL,
  preserve = NULL,
  dist_method,
  na_penalty,
  neg_penalty,
  greedy_threshold,
  checkForRare = FALSE
)
```

Arguments

dat	data.table object from the output of parseMedExtractR, parseMedXN, parseMedEx, or parseCLAMP
dn	Regular expression specifying drug name(s) of interest.
preserve	Column names to include in output, whose values should not be combined with other rows. If present, dosechange is always preserved.
dist_method	Distance method to use for calculating distance of various paths. Alternatively set the 'ehr.dist_method' option, which defaults to 'minEntEnd'.
na_penalty	Penalty for matching extracted entities with NA. Alternatively set the 'ehr.na_penalty' option, which defaults to 32.
neg_penalty	Penalty for negative distances between frequency/intake time and dose amounts. Alternatively set the 'ehr.neg_penalty' option, which defaults to 0.5.
greedy_threshold	Threshold to use greedy matching; increasing this value too high could lead to the algorithm taking a long time to finish. Alternatively set the 'ehr.greedy_threshold' option, which defaults to 1e8.
checkForRare	Indicate if rare values for each entity should be found and displayed.

Details

The buildDose function takes as its main input (dat), a data.table object that is the output of a parse process function (parseMedExtractR, parseMedXN, parseMedEx, or parseCLAMP). Broadly, the parsed extractions are grouped together to form wide, more complete drug regimen information.

This reformatting facilitates calculation of dose given intake and daily dose in the collapseDose process.

The process of creating this output is broken down into multiple steps:

1. Removing rows for any drugs not of interest. Drugs of interest are specified with the dn argument.
2. Determining whether extractions are "simple" (only one drug mention and at most one extraction per entity) or complex. Complex cases can be more straightforward if they contain at most one extraction per entity, or require a pairing algorithm to determine the best pairing if there are multiple extractions for one or more entities.
3. Drug entities are anchored by drug name mention within the parse process. For complex cases, drug entities are further grouped together anchored at each strength (and dose with medExtractR) extraction.
4. For strength groups with multiple extractions for at least one entity, these groups go through a path searching algorithm, which computes the cost for each path (based on a chosen distance method) and chooses the path with the lowest cost.
5. The chosen paths for each strength group are returned as the final pairings. If route is unique within a strength group, it is standardized and added to all entries for that strength group.

The user can specify additional arguments including:

- dist_method: The distance method is the metric used to determine which entity path is the most likely to be correct based on minimum cost.
- na_penalty: NA penalties are incurred when extractions are paired with nothing (i.e., an NA), requiring that entities be sufficiently far apart from one another before being left unpaired.
- neg_penalty: When working with dose amount (DA) and frequency/intake time (FIT), it is much more common for the ordering to be DA followed by FIT. Thus, when we observe FIT followed by DA, we apply a negative penalty to make such pairings less likely.
- greedy threshold: When there are many extractions from a clinical note, the number of possible combinations for paths can get exponentially large, particularly when the medication extraction natural language processing system is incorrect. The greedy threshold puts an upper bound on the number of entity pairings to prevent the function from stalling in such cases.

If none of the optional arguments are specified, then the buildDose process uses the default option values specified in the EHR package documentation. See EHR Vignette for Extract-Med and Pro-Med-NLP as well as Dose Building Using Example Vanderbilt EHR Data for details. For additional details, see McNeer, et al. 2020.

Value

A data.frame object that contains columns for filename (of the clinical note, inherited from the parse output object dat), drugname, strength, dose, route, freq, duration, and drugname_start.

Examples

```
data(lam_mxr_parsed)

buildDose(lam_mxr_parsed)
```

collapseDose *Collapse Dose Data*

Description

Splits drug data and calls makeDose to collapse at the note and date level.

Usage

```
collapseDose(x, noteMetaData, naFreq = "most", ...)
```

Arguments

x	data.frame containing the output of buildDose, or the output of addLastDose if last dose information is being incorporated.
noteMetaData	data.frame containing identifying meta data for each note, including patient ID, date of the note, and note ID. Column names should be set to 'filename', 'pid', 'date', 'note'. Date should have format YYYY-MM-DD.
naFreq	Expression used to replace missing frequencies with, or by default use the most common.
...	drug formulations to split by

Details

If different formulations of the drug (e.g., extended release) exist, they can be separated using a regular expression (e.g., 'xr|er'). This function will call makeDose on parsed and paired medication data to calculate dose intake and daily dose and remove redundancies at the note and date level.

See EHR Vignette for Extract-Med and Pro-Med-NLP as well as Dose Building Using Example Vanderbilt EHR Data for details.

Value

A list containing two dataframes, one with the note level and one with the date level collapsed data.

Examples

```
data(lam_mxr_parsed)
data(lam_metadata)

lam_build_out <- buildDose(lam_mxr_parsed)

lam_collapsed <- collapseDose(lam_build_out, lam_metadata, naFreq = 'most', 'xr|er')
lam_collapsed$note # Note level collapsing
lam_collapsed$date # Date level collapsing
```

dataTransformation *Data Transformation*

Description

Convenience function for making small modifications to a data.frame.

Usage

```
dataTransformation(x, select, rename, modify)
```

Arguments

x	a data.frame
select	columns to select
rename	character vector with names for all columns
modify	list of expressions used to transform data set

Value

The modified data.frame

dd *dd*

Description

Simulated outcome data example from Phenome Wide Association Study (PheWAS) that examines associations between drug exposure and various phenotypes at follow-up after the drug exposure. The dataset includes 1505 variables: subject identification number ('id'), drug exposure ('exposure'), 3 demographic variables ('age', 'race', 'gender'), and 1500 phenotypes.

Usage

```
data(dataPheWAS, package = 'EHR')
```

Format

A data frame with 10000 observations on 1505 variables.

Examples

```
data(dataPheWAS)
```

dd.baseline *dd.baseline*

Description

Simulated baseline data example from a Phenome Wide Association Study (PheWAS) obtained at baseline before drug exposure. The dataset includes 1505 variables: subject identification number ('id'), drug exposure ('exposure'), 3 demographic variables ('age', 'race', 'gender'), and 1500 phenotypes.

Usage

```
data(dataPheWAS, package = 'EHR')
```

Format

A data frame with 10000 observations on 1505 variables.

Examples

```
data(dataPheWAS)
```

dd.baseline.small *dd.baseline.small*

Description

A smaller subset of baseline data example, dd.baseline. The dataset includes 55 variables: subject identification number ('id'), drug exposure ('exposure'), 3 demographic variables ('age', 'race', 'gender'), and 50 phenotypes.

Usage

```
data(dataPheWASsmall, package = 'EHR')
```

Format

A data frame with 2000 observations on 55 variables.

Examples

```
data(dataPheWASsmall)
```

dd.small *dd.small*

Description

A smaller subset of outcome data example, 'dd'. The dataset includes 55 variables: subject identification number ('id'), drug exposure ('exposure'), 3 demographic variables ('age', 'race', 'gender'), and 50 phenotypes.

Usage

```
data(dataPheWASsmall, package = 'EHR')
```

Format

A data frame with 2000 observations on 55 variables.

Examples

```
data(dataPheWASsmall)
```

extractMed *Extract medication information from clinical notes*

Description

This function is an interface to the medExtractR function within the **medExtractR** package, and allows drug dosing information to be extracted from free-text sources, e.g., clinical notes.

Usage

```
extractMed(note_fn, drugnames, drgunit, windowlength, max_edit_dist = 0, ...)
```

Arguments

note_fn	File name(s) for the text file(s) containing the clinical notes. Can be a character string for an individual note, or a vector or list of file names for multiple notes.
drugnames	Vector of drug names for which dosing information should be extracted. Can include various forms (e.g., generic, brand name) as well as abbreviations.
drgunit	Unit of the drug being extracted, e.g., 'mg'
windowlength	Length of the search window (in characters) around the drug name in which to search for dosing entities
max_edit_dist	Maximum edit distance allowed when attempting to extract drugnames. Allows for capturing misspelled drug name information.
...	Additional arguments to medExtractR, for example lastdose=TRUE to extract time of last dose (see **medExtractR** package documentation for details)

Details

Medication information, including dosing data, is often stored in free-text sources such as clinical notes. The `extractMed` function serves as a convenient wrapper for the **medExtractR** package, a natural language processing system written in R for extracting medication data. Within `extractMed`, the `medExtractR` function identifies dosing data for drug(s) of interest, specified by the `drugnames` argument, using rule-based and dictionary-based approaches. Relevant dosing entities include medication strength (identified using the `unit` argument), dose amount, dose given intake, intake time or frequency of dose, dose change keywords (e.g., 'increase' or 'decrease'), and time of last dose. After applying `medExtractR` to extract drug dosing information, `extractMed` appends the file name to results to ensure they are appropriately labeled.

See EHR Vignette for for Extract-Med and Pro-Med-NLP. For more details, see Weeks, et al. 2020.

Value

A data.frame with the extracted dosing information, labeled with file name as an identifier
Sample output:

filename	entity	expr	pos
note_file1.txt	DoseChange	decrease	66:74
note_file1.txt	DrugName	Prograf	78:85
note_file1.txt	Strength	2 mg	86:90
note_file1.txt	DoseAmt	1	91:92
note_file1.txt	Frequency	bid	101:104
note_file1.txt	LastDose	2100	121:125

Examples

```
tac_fn <- list(system.file("examples", "tacpid1_2008-06-26_note1_1.txt", package = "EHR"),
               system.file("examples", "tacpid1_2008-06-26_note2_1.txt", package = "EHR"),
               system.file("examples", "tacpid1_2008-12-16_note3_1.txt", package = "EHR"))

extractMed(tac_fn,
           drugnames = c("tacrolimus", "prograf", "tac", "tacro", "fk", "fk506"),
           drgunit = "mg",
           windowlength = 60,
           max_edit_dist = 2,
           lastdose=TRUE)
```

---

freqNum *Convert Character Frequency to Numeric*

Description

This function converts the frequency entity to numeric.

Usage

```
freqNum(x)
```

Arguments

x	character vector of extracted frequency values

Value

numeric vector

Examples

```
f <- stdzFreq(c('in the morning', 'four times a day', 'with meals'))
freqNum(f)
```

---

idCrosswalk *Create ID Crosswalk*

Description

Link ID columns from multiple data sets. De-identified columns are created to make a crosswalk.

Usage

```
idCrosswalk(data, idcols, visit.id = "subject_id", uniq.id = "subject_uid")
```

Arguments

data	list of data.frames
idcols	list of character vectors, indicating ID columns found in each data set given in 'data'
visit.id	character sting indicating visit-level ID variable (default is "subject_id")
uniq.id	character sting indicating subject-level ID variable (default is "subject_uid")

Details

'visit.id' and 'uniq.id' may occur multiple times, but should have a one-to-one linkage defined by at least one of the input data sets. A new visit number is generated for each repeated 'uniq.id'.

Value

crosswalk of ID columns and their de-identified versions

Examples

```
demo_data <- data.frame(subj_id=c(4.1,4.2,5.1,6.1),
                        pat_id=c(14872,14872,24308,37143),
                        gender=c(1,1,0,1),
                        weight=c(34,42,28,63),
                        height=c(142,148,120,167))

conc_data <- data.frame(subj_id=rep((4:6)+0.1,each=5),
                        event=rep(1:5,times=3),
                        conc.level=15*exp(-1*rep(1:5,times=3))+rnorm(15,0,0.1))

data <- list(demo_data, conc_data)
idcols <- list(c('subj_id', 'pat_id'), 'subj_id')
idCrosswalk(data, idcols, visit.id='subj_id', uniq.id='pat_id')
```

---

lam_metadata *Example of Metadata for Lamotrigine Data*

Description

An example of the metadata needed for the processLastDose, makeDose, and collapseDose functions.

Usage

```
data(lam_metadata, package = 'EHR')
```

Format

A data frame with 5 observations on the following variables.

filename A character vector, filename for the clinical note

pid A character vector, patient ID associated with the filename

date A character vector, date associated with the filename

note A character vector, note ID associated with the filename

Examples

```
data(lam_metadata)
```

`lam_mxr_parsed` *Example of Lamotrigine Output from 'parseMedExtractR'*

Description

The output after running `parseMedExtractR` on 4 example clinical notes.

Usage

```
data(lam_mxr_parsed, package = 'EHR')
```

Format

A data frame with 10 observations on the following variables.

filename A character vector, filename for the clinical note

drugname A character vector, drug name extracted from the clinical note along with start and stop positions

strength A character vector, strengths extracted from the clinical note along with start and stop positions

dose A character vector, dose amounts extracted from the clinical note along with start and stop positions

route A character vector, routes extracted from the clinical note along with start and stop positions

freq A character vector, frequencies extracted from the clinical note along with start and stop positions

dosestr A character vector, dose intakes extracted from the clinical note along with start and stop positions

dosechange A character vector, dose change keywords extracted from the clinical note along with start and stop positions

lastdose A character vector, last dose times extracted from the clinical note along with start and stop positions

Examples

```
data(lam_mxr_parsed)
```

`Logistf` *Firth's penalized-likelihood logistic regression with more decimal places of p-value than* `logistf` *function in the R package 'logistf'*

Description

Adapted from `logistf` in the R package 'logistf', this is the same as `logistf` except that it provides more decimal places of p-value that would be useful for Genome-Wide Association Study (GWAS) or Phenome Wide Association Study (PheWAS).

Usage

```
Logistf(
  formula = attr(data, "formula"),
  data = sys.parent(),
  pl = TRUE,
  alpha = 0.05,
  control,
  plcontrol,
  firth = TRUE,
  init,
  weights,
  plconf = NULL,
  dataout = TRUE,
  ...
)
```

Arguments

`formula`	a formula object, with the response on the left of the operator, and the model terms on the right. The response must be a vector with 0 and 1 or FALSE and TRUE for the outcome, where the higher value (1 or TRUE) is modeled. It is possible to include contrasts, interactions, nested effects, cubic or polynomial splines and all S features as well, e.g. `Y ~ X1*X2 + ns(X3, df=4)`. From version 1.10, you may also include offset() terms.
`data`	a data.frame where the variables named in the formula can be found, i. e. the variables containing the binary response and the covariates.
`pl`	specifies if confidence intervals and tests should be based on the profile penalized log likelihood (`pl=TRUE`, the default) or on the Wald method (`pl=FALSE`).
`alpha`	the significance level (1-α the confidence level, 0.05 as default).
`control`	Controls Newton-Raphson iteration. Default is `control=logistf.control(maxstep, maxit, maxhs, lconv, gconv, xconv)`
`plcontrol`	Controls Newton-Raphson iteration for the estimation of the profile likelihood confidence intervals. Default is `plcontrol=logistpl.control(maxstep, maxit, maxhs, lconv, xconv, ortho, pr)`

firth	use of Firth's penalized maximum likelihood (firth=TRUE, default) or the standard maximum likelihood method (firth=FALSE) for the logistic regression. Note that by specifying pl=TRUE and firth=FALSE (and probably a lower number of iterations) one obtains profile likelihood confidence intervals for maximum likelihood logistic regression parameters.
init	specifies the initial values of the coefficients for the fitting algorithm.
weights	specifies case weights. Each line of the input data set is multiplied by the corresponding element of weights.
plconf	specifies the variables (as vector of their indices) for which profile likelihood confidence intervals should be computed. Default is to compute for all variables.
dataout	If TRUE, copies the data set to the output object.
...	Further arguments to be passed to logistf.

Value

same as logistf except for providing more decimal places of p-value.

Author(s)

Leena Choi <leena.choi@vanderbilt.edu> and Cole Beck <cole.beck@vumc.org>

References

same as those provided in the R package 'logistf'.

Examples

```
data(dataPheWAS)
fit <- Logistf(X264.3 ~ exposure + age + race + gender, data=dd)
summary(fit)
```

---

makeDose *Make Dose Data*

Description

Takes parsed and paired medication data, calculates dose intake and daily dose, and removes redundant information at the note and date level.

Usage

```
makeDose(x, noteMetaData, naFreq = "most")
```

Arguments

x	data.frame containing the output of buildDose, or the output of addLastDose if last dose information is being incorporated.
noteMetaData	data.frame containing identifying meta data for each note, including patient ID, date of the note, and note ID. Column names should be set to 'filename', 'pid', 'date', 'note'. Date should have format YYYY-MM-DD.
naFreq	Replacing missing frequencies with this value, or by default the most common value across the entire set in x.

Details

This function standardizes frequency, route, and duration entities. Dose amount, strength, and frequency entities are converted to numeric. Rows with only drug name and/or route are removed. If there are drug name changes in adjacent rows (e.g., from a generic to brand name), these rows are collapsed into one row if there are no conflicts. Missing strengths, dose amounts, frequencies, and routes are borrowed or imputed using various rules (see McNeer et al., 2020 for details). Dose given intake and daily dose are calculated. Redundancies are removed at the date and note level. If time of last dose is being used and it is unique within the level of collapsing, it is borrowed across all rows.

Value

A list containing two dataframes, one with the note level and one with the date level collapsed data.

Examples

```
data(lam_mxr_parsed)
data(lam_metadata)

lam_build_out <- buildDose(lam_mxr_parsed)

lam_collapsed <- makeDose(lam_build_out, lam_metadata)
lam_collapsed[[1]] # Note level collapsing
lam_collapsed[[2]] # Date level collapsing
```

---

parseCLAMP *Parse CLAMP NLP Output*

Description

Takes files with the raw medication extraction output generated by the CLAMP natural language processing system and converts it into a standardized format.

Usage

```
parseCLAMP(filename)
```

Arguments

filename	File name for a single file containing CLAMP output.

Details

Output from different medication extraction systems is formatted in different ways. In order to be able to process the extracted information, we first need to convert the output from different systems into a standardized format. Extracted expressions for various drug entities (e.g., drug name, strength, frequency, etc.) each receive their own column formatted as "extracted expression::start position::stop position". If multiple expressions are extracted for the same entity, they will be separated by backticks.

CLAMP output files anchor extractions to a specific drug name extraction through semantic relations.

See EHR Vignette for Extract-Med and Pro-Med-NLP as well as Dose Building Using Example Vanderbilt EHR Data for details.

Value

A data.table object with columns for filename, drugname, strength, dose, route, and freq. The filename contains the file name corresponding to the clinical note. Each of the entity columns are of the format "extracted expression::start position::stop position".

---

parseMedEx *Parse MedEx NLP Output*

Description

Takes files with the raw medication extraction output generated by the MedEx natural language processing system and converts it into a standardized format.

Usage

```
parseMedEx(filename)
```

Arguments

filename	File name for a single file containing MedEx output.

Details

Output from different medication extraction systems is formatted in different ways. In order to be able to process the extracted information, we first need to convert the output from different systems into a standardized format. Extracted expressions for various drug entities (e.g., drug name, strength, frequency, etc.) each receive their own column formatted as "extracted expression::start position::stop position". If multiple expressions are extracted for the same entity, they will be separated by backticks.

MedEx output files anchor extractions to a specific drug name extraction.

See EHR Vignette for Extract-Med and Pro-Med-NLP as well as Dose Building Using Example Vanderbilt EHR Data for details.

Value

A data.table object with columns for filename, drugname, strength, dose, route, and freq. The filename contains the file name corresponding to the clinical note. Each of the entity columns are of the format "extracted expression::start position::stop position".

`parseMedExtractR` *Parse medExtractR NLP Output*

Description

Takes files with the raw medication extraction output generated by the medExtractR natural language processing system and converts it into a standardized format.

Usage

```
parseMedExtractR(filename)
```

Arguments

`filename`	File name for a single file containing medExtractR output.

Details

Output from different medication extraction systems is formatted in different ways. In order to be able to process the extracted information, we first need to convert the output from different systems into a standardized format. Extracted expressions for various drug entities (e.g., drug name, strength, frequency, etc.) each receive their own column formatted as "extracted expression::start position::stop position". If multiple expressions are extracted for the same entity, they will be separated by backticks.

The medExtractR system returns extractions in a long table format, indicating the entity, extracted expression, and start:stop position of the extraction. To perform this initial parsing, entities are paired with the closest preceding drug name. The one exception to this is the dose change entity, which can occur before the drug name (see Weeks, et al. 2020 for details).

See EHR Vignette for Extract-Med and Pro-Med-NLP as well as Dose Building Using Example Vanderbilt EHR Data for details.

Value

A data.table object with columns for filename, drugname, strength, dose, route, freq, dosestr, dosechange and lastdose. The filename contains the file name corresponding to the clinical note. Each of the entity columns are of the format "extracted expression::start position::stop position".

Examples

```
mxr_output <- system.file("examples", "lam_mxr.csv", package = "EHR")
mxr_parsed <- parseMedExtractR(mxr_output)
mxr_parsed
```

`parseMedXN` *Parse MedXN NLP Output*

Description

Takes files with the raw medication extraction output generated by the MedXN natural language processing system and converts it into a standardized format.

Usage

```
parseMedXN(filename, begText = "^[R0-9]+_[0-9-]+_[0-9]+_")
```

Arguments

`filename`	File name for single file containing MedXN output.
`begText`	A regular expression that would indicate the beginning of a new observation (i.e., extracted clinical note).

Details

Output from different medication extraction systems is formatted in different ways. In order to be able to process the extracted information, we first need to convert the output from different systems into a standardized format. Extracted expressions for various drug entities (e.g., drug name, strength, frequency, etc.) each receive their own column formatted as "extracted expression::start position::stop position". If multiple expressions are extracted for the same entity, they will be separated by backticks.

MedXN output files anchor extractions to a specific drug name extraction.

In MedXN output files, the results from multiple clinical notes can be combined into a single output file. The beginning of some lines of the output file can indicate when output for a new observation (or new clinical note) begins. The user should specify the argument `begText` to be a regular expression used to identify the lines where output for a new clinical note begins.

See EHR Vignette for Extract-Med and Pro-Med-NLP as well as Dose Building Using Example Vanderbilt EHR Data for details.

Value

A data.table object with columns for filename, drugname, strength, dose, route, freq, and duration. The filename contains the file name corresponding to the clinical note. Each of the entity columns are of the format "extracted expression::start position::stop position".

Examples

```
mxn_output <- system.file("examples", "lam_medxn.csv", package = "EHR")
mxn_parsed <- parseMedXN(mxn_output, begText = "^ID[0-9]+_[0-9-]+_")
mxn_parsed
```

`processLastDose` *Process and standardize extracted last dose times*

Description

This function takes last dose times extracted using the **medExtractR** system and processes the times into standardized datetime objects using recorded lab data where necessary. The raw output from `extractMed` is filtered to just the LastDose extractions. Time expressions are standardized into HH:MM:SS format based on what category they fall into (e.g., a time represented with AM/PM, 24-hour military time, etc.). When the last dose time is after 12pm, it is assumed to have been taken one day previous to the note's date. For any duration extractions (e.g. "14 hour level"), the last dose time is calculated from the labtime by extracting the appropriate number of hours. The final dataset is returned with last dose time formatted into a POSIXct variable.

Usage

```
processLastDose(mxrData, noteMetaData, labData)
```

Arguments

`mxrData`	data.frame containing output from the `medExtractR` system
`noteMetaData`	data.frame with meta data (`pid` (patient ID) and `date`) for the file names contained within `mxrData`
`labData`	data.frame that contains lab dates and times associated with the file names within `mxrData`. Must contain columns `pid` and `date`, as well as `labtime`. The `date` column must be in the same format as `date` in `noteMetaData`, and `labtime` must be a POSIXct

Details

See EHR Vignette for Extract-Med and Pro-Med-NLP for details.

Value

data.frame with identifying information (e.g., filename, etc) as well as processed and standardized last dose times as a POSIXct column

Examples

```
tac_mxr <- read.csv(system.file("examples", "tac_mxr.csv", package = "EHR"))
data(tac_metadata)
data(tac_lab)

processLastDose(mxrData = tac_mxr, noteMetaData = tac_metadata, labData = tac_lab)
```

`pullFakeId` *Pull Fake/Mod ID*

Description

Replace IDs with de-identified version pulled from a crosswalk.

Usage

```
pullFakeId(
  dat,
  xwalk,
  firstCols = NULL,
  orderBy = NULL,
  uniq.id = "subject_uid"
)
```

Arguments

`dat`	a data.frame
`xwalk`	a data.frame providing linkage for each ID, e.g. output from `idCrosswalk`
`firstCols`	name of columns to put at front of output data set
`orderBy`	name of columns used to reorder output data set
`uniq.id`	character string indicating subject-level id variable (default is "subject_uid")

Value

The modified data.frame

Examples

```
demo_data <- data.frame(subj_id=c(4.1,4.2,5.1,6.1),
                        pat_id=c(14872,14872,24308,37143),
                        gender=c(1,1,0,1),
                        weight=c(34,42,28,63),
                        height=c(142,148,120,167))

# crosswalk w/ same format as idCrosswalk() output
xwalk <- data.frame(subj_id=c(4.1,4.2,5.1,6.1),
```

```
                    pat_id=c(14872,14872,24308,37143),
                    mod_visit=c(1,2,1,1),
                    mod_id=c(1,1,2,3),
                    mod_id_visit=c(1.1,1.2,2.1,3.1))

demo_data_deident <- pullFakeId(demo_data, xwalk,
                                firstCols = c('mod_id','mod_id_visit','mod_visit'),
                                uniq.id='pat_id')
```

`pullRealId` *Pull Real ID*

Description

Replace de-identified IDs with identified version pulled from a crosswalk.

Usage

```
pullRealId(dat, xwalk = NULL, remove.mod.id = FALSE)
```

Arguments

`dat`	a data.frame
`xwalk`	a data.frame providing linkage for each ID, e.g. output from `idCrosswalk`; if NULL, the crosswalk will be pulled from the 'pkxwalk' option, or otherwise the unmodified data.frame.
`remove.mod.id`	logical, should the de-identified IDs – mod_id, mod_visit, mod_id_visit – be removed (default=FALSE)

Value

The modified data.frame

Examples

```
demo_data_deident <- data.frame(mod_id=c(1,1,2,3),
                                mod_id_visit=c(1.1,1.2,2.1,3.1),
                                mod_visit=c(1,2,1,1),
                                gender=c(1,1,0,1),
                                weight=c(34,42,28,63),
                                height=c(142,148,120,167))

# crosswalk w/ same format as idCrosswalk() output
xwalk <- data.frame(subj_id=c(4.1,4.2,5.1,6.1),
                    pat_id=c(14872,14872,24308,37143),
                    mod_visit=c(1,2,1,1),
                    mod_id=c(1,1,2,3),
                    mod_id_visit=c(1.1,1.2,2.1,3.1))
```

```
pullRealId(demo_data_deident, xwalk)
pullRealId(demo_data_deident, xwalk, remove.mod.id=TRUE)
```

`readTransform` *Read and Transform*

Description

Convenience function for reading in a CSV file, and making small modifications to a data.frame.

Usage

```
readTransform(file, ...)
```

Arguments

`file`	filename of a CSV file
`...`	additional information passed to `dataTransformation`

Details

If `read.csv` needs additional arguments (or the file is in a different format), the user should load the data first, then directly call `dataTransformation`.

Value

The modified data.frame

`run_Build_PK_IV` *Build-PK-IV Module*

Description

This module builds PK data for intravenously (IV) administered medications.

Usage

```
run_Build_PK_IV(
  conc,
  conc.columns = list(),
  dose,
  dose.columns = list(),
  censor = NULL,
  censor.columns = list(),
  demo.list = NULL,
  demo.columns = list(),
  lab.list = NULL,
  lab.columns = list(),
  dosePriorWindow = 7,
  labPriorWindow = 7,
  postWindow = NA,
  pk.vars = NULL,
  drugname = NULL,
  check.path = NULL,
  missdemo_fn = "-missing-demo",
  faildupbol_fn = "DuplicateBolus-",
  date.format = "%m/%d/%y %H:%M:%S",
  date.tz = "America/Chicago",
  isStrict = FALSE
)
```

Arguments

conc	concentration data, the output of run_DrugLevel, a filename (CSV, RData, RDS), or a correctly formatted data.frame
conc.columns	a named list that should specify columns in concentration data; 'id', 'datetime', 'druglevel' are required. 'idvisit' may also be specified; 'idvisit' can be used when there are multiple visits (i.e., several occasions) for the same subject. 'datetime' is date and time for concentration measurement, which can refer to a single date-time variable (datetime = 'date_time') or two variables holding date and time separately (e.g., datetime = c('Date', 'Time')).
dose	dose data, the output of run_MedStrI, a filename (CSV, RData, RDS), or a correctly formatted data.frame
dose.columns	a named list that should specify columns in dose data; 'id' is required. 'infuseDatetime' and 'infuseDose' should be set if infusion dose data is present. 'infuseTimeExact' may also be specified for infusion data – this variable represents an precise time, if for example the 'infuseDatetime' variable is rounded. 'bolusDatetime' and 'bolusDose' should be set if bolus dose data is present. A generic 'date' variable may be provided, agnostic to either infusion or bolus dosing. 'gap' and 'weight' column names may also be set. Any of the date-time variables can be specified as a single date-time variable (infuseDatetime = 'date_time') or two variables holding date and time separately (e.g., infuseDatetime = c('Date', 'Time')).

censor	censoring information, if available; this will censor concentration and dose data for dates occuring after the censor datetime variable.
censor.columns	a named list that should specify columns in censoring data; 'id', and 'datetime' are required. 'datetime' is the date and time when data should be censored. This can refer to a single date-time variable (datetime = 'date_time') or two variables holding date and time separately (e.g., datetime = c('Date', 'Time')).
demo.list	demographic information, if available; the output from run_Demo or a correctly formatted data.frame
demo.columns	a named list that should specify columns in demographic data; 'id' is required. 'weight' and 'idvisit' may also be used to specify columns for weight or the unique idvisit. Any other columns present in the demographic data are treated as covariates.
lab.list	lab data, if available; the output from run_Labs or a correctly formatted list
lab.columns	a named list that should specify columns in lab data; 'id', and 'datetime' are required. 'datetime' is the date and time when the lab data was obtained, which can refer to a single date-time variable (datetime = 'date_time') or two variables holding date and time separately (e.g., datetime = c('Date', 'Time')). Any other columns present in lab data are treated as lab values.
dosePriorWindow	Dose data is merged with drug level data. This value sets the time frame window with the number of days prior to the first drug level data; defaults to 7.
labPriorWindow	Lab data is merged with drug level data. This value sets the time frame window with the number of days prior to the first drug level data; defaults to 7.
postWindow	Data is merged with drug level data. This postWindow can set the end time for the drug level data, being the number of days after the first drug level data. The default (NA) will use the date of the last drug level data.
pk.vars	variables to include in the returned PK data. The variable 'date' is a special case; when included, it maps the 'time' offset to its original date-time. Other named variables will be merged from the concentration data set. For example, rather than being separate data sets, labs or demographics may already be present in the concentration data. These columns should be named here.
drugname	drug of interest, included in filename of check files. The default (NULL) will produce filenames without drugname included.
check.path	path to 'check' directory, where check files are created. The default (NULL) will not produce any check files.
missdemo_fn	filename for checking NA frequency among demographic data
faildupbol_fn	filename for duplicate bolus data
date.format	output format for 'date' variable
date.tz	output time zone for 'date' variable
isStrict	logical; when TRUE dose amount totals are strictly summed rather than repeated hourly until stopped

Details

See EHR Vignette for Structured Data.

Regarding the 'gap' variable in the dose dataset, if 'gap' is specified in 'dose.columns', it allows a continuous infusion given when there are missing records between infusion dosing records. For example, suppose that 'gap' = 60 is defined (which is typical gap size when infusion dosing is supposed to be recorded hourly for inpatients) and time between two records (i.e., gap) are greater than 1 hour (i.e., missing records). If the gap between the two records is less or equal to twice of the gap (i.e., 2*60 = 120 min), a continuous infusion is assumed until the 2nd dose record; otherwise, the first infusion is assumed to be stopped (i.e., add zero doses) after 60 min (i.e., equal to the gap size) and a new infusion (the 2nd record) starts at its recorded time.

Value

PK data set

Examples

```
# make fake data
set.seed(6543)

build_date <- function(x) format(seq(x, length.out=5, by="1 hour"), "%Y-%m-%d %H:%M")
dates <- unlist(lapply(rep(Sys.time(),3), build_date))

plconc <- data.frame(mod_id = rep(1:3,each=5),
                     mod_id_visit = rep(1:3,each=5)+0.1,
                     event = rep(1:5,times=3),
                     conc.level = 15*exp(-1*rep(1:5,times=3))+rnorm(15,0,0.1),
                     date.time = as.POSIXct(dates))

ivdose <- data.frame(mod_id = 1:3,
                      date.dose = substr(dates[seq(1,15,by=5)],1,10),
                      infuse.time.real = NA, infuse.time = NA, infuse.dose = NA,
                      bolus.time = as.POSIXct(dates[seq(1,15,by=5)])-300,
                      bolus.dose = 90,
                      maxint = 0L,
                      weight = 45)


run_Build_PK_IV(conc = plconc,
                conc.columns = list(id = 'mod_id', datetime = 'date.time',
                  druglevel = 'conc.level', idvisit = 'mod_id_visit'),
                dose = ivdose,
                dose.columns = list(id = 'mod_id', date = 'date.dose',
                  bolusDatetime = 'bolus.time', bolusDose = 'bolus.dose',
                  gap = 'maxint', weight = 'weight'),
                pk.vars = 'date')
```

`run_Build_PK_Oral` *Build-PK-Oral Module*

Description

This module builds PK data for orally administered medications.

Usage

```
run_Build_PK_Oral(
  x,
  idCol = "id",
  dtCol = "dt",
  doseCol = "dose",
  concCol = "conc",
  ldCol = NULL,
  first_interval_hours = 336,
  imputeClosest = NULL
)
```

Arguments

`x`	a data.frame or file saved as either CSV, RData, or RDS
`idCol`	data.frame id column name
`dtCol`	data.frame date column name
`doseCol`	dose column name
`concCol`	concentration column name
`ldCol`	last-dose time column name
`first_interval_hours`	number of hours before the first concentration to start time=0; the default is 336 hours = 14 days
`imputeClosest`	columns to impute missing data with next observation propagated backward; this is in addition to all covariates receving imputation using last observation carried forward

Details

See EHR Vignette for Build-PK-Oral.

Value

data.frame

Examples

```
## Data Generating Function
mkdat <- function() {
  npat <- 3
  visits <- floor(runif(npat, min=2, max=6))
  id <- rep(1:npat, visits)
  dt_samp <- as.Date(sort(sample(700, sum(visits))), origin = '2019-01-01')
  tm_samp <- as.POSIXct(paste(dt_samp, '10:00:00'), tz = 'UTC')
  dt <- tm_samp + rnorm(sum(visits), 0, 1*60*60)
  dose_morn <- sample(c(2.5,5,7.5,10), sum(visits), replace = TRUE)
  conc <- round(rnorm(sum(visits), 1.5*dose_morn, 1),1)
  ld <- dt - sample(10:16, sum(visits), replace = TRUE) * 3600
  ld[rnorm(sum(visits)) < .3] <- NA
  age <- rep(sample(40:75, npat), visits)
  gender <- rep(sample(0:1, npat, replace=TRUE), visits)
  weight <- rep(round(rnorm(npat, 180, 20)),visits)
  hgb <- rep(rnorm(npat, 10, 2), visits)
  data.frame(id, dt, dose_morn, conc, ld, age, gender, weight, hgb)
}

# Make raw data
set.seed(30)
dat <- mkdat()

#Process data without last-dose times
run_Build_PK_Oral(x = dat,
                  idCol = "id",
                  dtCol = "dt",
                  doseCol = "dose_morn",
                  concCol = "conc",
                  ldCol = NULL,
                  first_interval_hours = 336,
                  imputeClosest = NULL)

#Process data with last-dose times
run_Build_PK_Oral(x = dat, doseCol = "dose_morn", ldCol = "ld")
```

---

run_Demo *Run Demographic Data*

Description

This module will load and modify demographic data.

Usage

```
run_Demo(demo.path, demo.columns = list(), toexclude, demo.mod.list)
```

Arguments

demo.path	filename of demographic file (CSV, RData, RDS) or data.frame
demo.columns	a named list that should specify columns in demo data; 'id', is required.
toexclude	expression that should evaluate to a logical, indicating if the observation should be excluded
demo.mod.list	list of expressions, giving modifications to make

Details

See EHR Vignette for Structured Data.

Value

list with two components

demo	demographic data
exclude	vector of excluded visit IDs

Examples

```
set.seed(2525)
dateSeq <- seq(as.Date('2019/01/01'), as.Date('2020/01/01'), by="day")
demo <- data.frame(mod_id_visit = 1:10,
                   weight.lbs = rnorm(10,160,20),
                   age = rnorm(10, 50, 10),
                   enroll.date = sample(dateSeq, 10))
tmpfile <- paste0(tempfile(), '.rds')
saveRDS(demo, file = tmpfile)

# exclusion functions
exclude_wt <- function(x) x < 150
exclude_age <- function(x) x > 60
ind.risk <- function(wt, age) wt>170 & age>55
exclude_enroll <- function(x) x < as.Date('2019/04/01')

# make demographic data that:
# (1) excludes ids with weight.lbs < 150, age > 60, or enroll.date before 2019/04/01
# (2) creates new 'highrisk' variable for subjects with weight.lbs>170 and age>55
out <- run_Demo(demo.path = tmpfile, demo.columns = list(id = 'mod_id_visit'),
               toexclude = expression(
                 exclude_wt(weight.lbs)|exclude_age(age)|exclude_enroll(enroll.date)
               ),
               demo.mod.list = list(highrisk = expression(ind.risk(weight.lbs, age))))

out
```

run_DrugLevel *Run Drug Level Data*

Description

This module will load and modify drug-level data.

Usage

```
run_DrugLevel(
  conc.path,
  conc.columns = list(),
  conc.select,
  conc.rename,
  conc.mod.list = NULL,
  samp.path = NULL,
  samp.columns = list(),
  samp.mod.list = NULL,
  check.path = NULL,
  failmiss_fn = "MissingConcDate-",
  multsets_fn = "multipleSetsConc-",
  faildup_fn = "DuplicateConc-",
  drugname = NULL,
  LLOQ = NA,
  demo.list = NULL,
  demo.columns = list()
)
```

Arguments

conc.path	filename of concentration data (CSV, RData, RDS), or data.frame
conc.columns	a named list that should specify columns in concentration data. 'id' and 'conc' are required. 'idvisit' may also be specified. If linking with sampling data, 'samplinkid' is required. Otherwise 'datetime' is required. This is the date and time when blood samples were obtained. This can refer to a single date-time variable (datetime = 'date_time') or two variables holding date and time separately (e.g., datetime = c('Date', 'Time')).
conc.select	columns to select from concentration data
conc.rename	new column names for concentration data
conc.mod.list	list of expressions, giving modifications to make
samp.path	filename of data with sampling time (CSV, RData, RDS), or data.frame
samp.columns	a named list that should specify columns in sampling data. 'conclinkid' and 'datetime' are required to link sampling data to concentration data. 'conclinkid' should match the id variable provided as 'samplinkid' in the 'conc.columns' argument. 'datetime' is the date and time when blood samples were obtained.

	This can refer to a single date-time variable (datetime = 'date_time') or two variables holding date and time separately (e.g., datetime = c('Date', 'Time')).
samp.mod.list	list of expressions, giving modifications to make
check.path	path to 'check' directory, where check files are created. The default (NULL) will not produce any check files.
failmiss_fn	filename for data missing concentration date
multsets_fn	filename for data with multiple concentration sets
faildup_fn	filename for data with duplicate concentration observations
drugname	drug of interest, included in filename of check files. The default (NULL) will produce filenames without drugname included.
LLOQ	lower limit of concentration values; values below this are invalid
demo.list	demographic information; if available, concentration records must have a valid demo record
demo.columns	a named list that should specify columns in demographic data; 'id', is required. If 'idvisit' is present in the concentration data, then it is required here too.

Details

See EHR Vignette for Structured Data.

Value

drug-level data set

Examples

```
# concentrations
conc_data <- data.frame(mod_id = rep(1:3,each=4),
                        mod_visit = rep(c(2,1,1),each=4),
                        mod_id_visit = as.numeric(paste(rep(1:3,each=4),
                                                        rep(c(2,1,1),each=4), sep=".")),
                        samp = rep(1:4,times=3),
                        drug_calc_conc=15*exp(-1*rep(1:4,times=3))+rnorm(12,0,0.1))

# sample times
build_date <- function(x) format(seq(x, length.out=4, by="1 hour"), "%Y-%m-%d %H:%M")
dates <- unlist(lapply(rep(Sys.time(),3), build_date))

samp_data <- data.frame(mod_id = rep(1:3,each=4),
                        mod_visit = rep(c(2,1,1),each=4),
                        mod_id_visit = as.numeric(paste(rep(1:3,each=4),
                                                         rep(c(2,1,1),each=4), sep=".")),
                        samp = rep(1:4,times=3),
                        Sample.Collection.Date.and.Time = dates)

run_DrugLevel(
  conc.path = conc_data,
  conc.columns = list(
```

```
   id = 'mod_id', idvisit = 'mod_id_visit', samplinkid = 'mod_id_event', conc = 'conc.level'
  ),
  conc.select = c('mod_id','mod_id_visit','samp','drug_calc_conc'),
  conc.rename = c(drug_calc_conc= 'conc.level', samp='event'),
  conc.mod.list = list(mod_id_event = expression(paste(mod_id_visit, event, sep = "_"))),
  samp.path = samp_data,
  samp.columns = list(conclinkid = 'mod_id_event', datetime = 'Sample.Collection.Date.and.Time'),
  samp.mod.list = list(mod_id_event = expression(paste(mod_id_visit, samp, sep = "_"))),
  drugname = 'drugnm',
  LLOQ = 0.05
)

# minimal example with data in required format
conc_data <- conc_data[,c('mod_id','mod_id_visit','samp','drug_calc_conc')]
conc_data[,'mod_id_event'] <- paste(conc_data[,'mod_id_visit'], conc_data[,'samp'], sep = "_")
names(conc_data)[3:4] <- c('event','conc.level')
samp_data[,'mod_id_event'] <- paste(samp_data[,'mod_id_visit'], samp_data[,'samp'], sep = "_")
conc_samp_link <- match(conc_data[,'mod_id_event'], samp_data[,'mod_id_event'])
conc_date <- samp_data[conc_samp_link, 'Sample.Collection.Date.and.Time']
conc_data[,'date.time'] <- as.POSIXct(conc_date)
run_DrugLevel(conc_data, conc.columns = list(
  id = 'mod_id', idvisit = 'mod_id_visit', datetime = 'date.time', conc = 'conc.level'
))
```

`run_Labs` *Run Lab Data*

Description

This module will load and modify laboratory data.

Usage

```
run_Labs(lab.path, lab.select, lab.mod.list)
```

Arguments

`lab.path`	filename of a lab file (CSV, RData, RDS), or data.frame
`lab.select`	columns to select
`lab.mod.list`	list of expressions giving modifications to make; passed to `dataTransformation`

Details

See EHR Vignette for Structured Data.

Value

lab data set

Examples

```
lab_data <- data.frame(mod_id=rep(1:3,each=3),
                       date=rep(c("01/12/17","05/05/18","11/28/16"),each=3),
                       time=rep(c("1:30","2:30","3:30"),3),
                       creat=rnorm(9,0.5,0.05))
run_Labs(lab_data, lab.mod.list=list(log_creat=expression(log(creat))))
```

`run_MedStrI` *Run Str Data I*

Description

This module will load and modify structured intravenous (IV) infusion and bolus medication data.

Usage

```
run_MedStrI(
  mar.path,
  mar.columns = list(),
  medGivenReq = FALSE,
  flow.path = NULL,
  flow.columns = list(),
  medchk.path = NULL,
  demo.list = NULL,
  demo.columns = list(),
  missing.wgt.path = NULL,
  wgt.columns = list(),
  check.path = NULL,
  failflow_fn = "FailFlow",
  failnounit_fn = "NoUnit",
  failunit_fn = "Unit",
  failnowgt_fn = "NoWgt",
  censor_date_fn = "CensorTime",
  infusion.unit = "mcg/kg/hr",
  bolus.unit = "mcg",
  bol.rate.thresh = Inf,
  rateunit = "mcg/hr",
  ratewgtunit = "mcg/kg/hr",
  weightunit = "kg",
  drugname = NULL
)
```

Arguments

`mar.path`	filename of MAR data (CSV, RData, RDS), or data.frame

`mar.columns` a named list that should specify columns in MAR data; 'id', 'datetime' and 'dose' are required. 'drug', 'weight', 'given' may also be specified. 'datetime' is date and time for data measurement, which can refer to a single date-time variable (datetime = 'date_time') or two variables holding date and time separately (e.g., datetime = c('Date', 'Time')). 'dose' can also be given as a single variable or two variables. If given as a single column, the column's values should contain dose and units such as '25 mcg'. If given as two column names, the dose column should come before the unit column (e.g., dose = c('doseamt', 'unit')). 'drug' can provide list of acceptable drug names. If 'drug' is present, the 'medchk.path' argument should also be provided. The 'given' is a variable that flags whether the medication (inpatient) was given. When it is given, values shoule be "Given"; should be used in conjunction with the 'medGivenReq' argument.

`medGivenReq` if TRUE, values in 'given' column in MAR data should equal "Given"; if this is FALSE (the default), NA values are also acceptable.

`flow.path` filename of flow data (CSV, RData, RDS), or data.frame

`flow.columns` a named list that should specify columns in flow data; 'id', 'datetime', 'finalunits', 'unit', 'rate', 'weight' are required. 'idvisit' may also be specified. 'datetime' is date and time for data measurement, which can refer to a single datetime variable (datetime = 'date_time') or two variables holding date and time separately (e.g., datetime = c('Date', 'Time')).

`medchk.path` filename containing data set (CSV, RData, RDS), or data.frame; should have the column 'medname' with list of acceptable drug names (e.g., brand and generic name, abbreviations) to subset drugs of interest using 'drug' column in MAR data. This argument can be used when MAR data contains different drugs that should be excluded.

`demo.list` demographic information; if available, the output from 'run_Demo' or a correctly formatted data.frame, which can be used to impute weight when missing

`demo.columns` a named list that should specify columns in demographic data; 'id', 'datetime', and 'weight' are required. 'datetime' is the date and time when the demographic data were obtained, which can refer to a single date-time variable (datetime = 'date_time') or two variables holding date and time separately (e.g., datetime = c('Date', 'Time')).

`missing.wgt.path` filename containing additional weight data (CSV, RData, RDS), or data.frame. The variables in this file should be defined in the 'wgt.columns' argument.

`wgt.columns` a named list that should specify columns in additional weight data; 'id', 'datetime', and 'weight' are required. 'datetime' is date and time for weight measurement, which can refer to a single date-time variable (datetime = 'date_time') or two variables holding date and time separately (e.g., datetime = c('Date', 'Time')).

`check.path` path to 'check' directory, where check files are created. The default (NULL) will not produce any check files.

`failflow_fn` filename for duplicate flow data with rate zero

`failnounit_fn` filename for MAR data with missing unit

`failunit_fn` filename for MAR data with invalid unit

failnowgt_fn	filename for infusion data with missing weight where unit indicates weight is required
censor_date_fn	filename containing censor times created with invalid dose data
infusion.unit	acceptable unit for infusion data
bolus.unit	acceptable unit for bolus data
bol.rate.thresh	upper limit for bolus rate; values above this are invalid
rateunit	acceptable unit for hourly rate; defaults to 'mcg/hr'
ratewgtunit	acceptable unit for hourly rate by weight; defaults to 'mcg/kg/hr'
weightunit	acceptable unit for weight; defaults to 'kg'
drugname	drug of interest, included in filename of check files. The default (NULL) will produce filenames without drugname included.

Details

See EHR Vignette for Structured Data.

Value

structured data set

Examples

```
# flow data for 'Fakedrug1'
flow <- data.frame(mod_id=c(1,1,2,2,2),
                   mod_id_visit=c(46723,46723,84935,84935,84935),
                   record.date=c("07/05/2019 5:25","07/05/2019 6:01",
                                 "09/04/2020 3:21", "09/04/2020 4:39",
                                 "09/04/2020 5:32"),
                   Final.Weight=c(6.75,6.75,4.5,4.5,4.5),
                   Final.Rate=c(rep("1 mcg/kg/hr",2),
                                rep("0.5 mcg/kg/hr",3)),
                   Final.Units=c("3.375","6.5",
                                 "2.25","2.25","2.25"))
flow[,'Perform.Date'] <- pkdata::parse_dates(flow[,'record.date'])
flow[,'unit'] <- sub('.*[ ]', '', flow[,'Final.Rate'])
flow[,'rate'] <- as.numeric(sub('([0-9.]+).*', '\\1', flow[,'Final.Rate']))

# mar data for 4 fake drugs
mar <- data.frame(mod_id=rep(1,5),
                  Date=rep("2019-07-05",5),
                  Time=c("07:12","07:31","08:47","09:16","10:22"),
                  `med:mDrug`=c("Fakedrug2","Fakedrug1","Fakedrug2",
                                "Fakedrug3","Fakedrug4"),
                  `med:dosage`=c("30 mg","0.5 mcg","1 mg",
                                 "20 mg","3 mcg/kg/min"),
                  `med:route`=rep("IV",5),
                  `med:given`=rep("Given",5),
                  check.names=FALSE)
```

```
# medcheck file for drug of interest ('Fakedrug1')
medcheck <- data.frame(medname="Fakedrug1",freq=4672)

run_MedStrI(mar.path = mar,
            mar.columns = list(id = 'mod_id', datetime = c('Date','Time'),
                               dose = 'med:dosage', drug = 'med:mDrug', given = 'med:given'),
            flow.path = flow,
            flow.columns = list(id = 'mod_id', datetime = 'Perform.Date',
                                finalunits = 'Final.Units', unit = 'unit',
                                rate = 'rate', weight = 'Final.Weight'),
            medchk.path = medcheck,
            check.path = tempdir(),
            drugname = 'fakedrg1')
```

`run_MedStrII` *Run Structured E-Prescription Data*

Description

This module will load and modify structured e-prescription data.

Usage

```
run_MedStrII(file, dat.columns = list())
```

Arguments

`file`	filename of prescription data (CSV, RData, RDS), or data.frame
`dat.columns`	a named list that should specify columns in data; 'id', 'dose', 'freq', 'date', and 'str' are required. 'desc' may also be specified.

Details

See EHR Vignette for Structured Data.

Value

str data set

Examples

```
erx_data <- data.frame(GRID=paste0("ID",c(1,1,2,2,2,2)),
                       MED_NAME=c("fakedrug","fakedrug","fakedrug",
                                  "Brandname","fakedrug","fakedrug"),
                       RX_DOSE=c(1,2,1,'2 tabs',1,'1+1.5+1'),
                       FREQUENCY=c(rep("bid",3),"qam","bid",
                                   "brkfst,lunch,dinner"),
```

```
                        ENTRY_DATE=c("2018-02-15","2018-03-14","2017-07-01",
                                     "2017-07-01","2017-09-15","2017-11-01"),
                        STRENGTH_AMOUNT=c("100","100","200",
                                          "100mg","100","100"),
                        DESCRIPTION=c("fakedrug 100 mg tablet","fakedrug 100 mg tablet",
                                      "fakedrug 200 mg tablet (also known as brandname)",
                                      "Brandname 100mg tablet", "fakedrug 100 mg tablet",
                                      "fakedrug 100 mg tablet"))

run_MedStrII(erx_data, list(id = 'GRID', dose = 'RX_DOSE', freq = 'FREQUENCY',
             date = 'ENTRY_DATE', str = 'STRENGTH_AMOUNT', desc = 'DESCRIPTION'))
```

`stdzDose` *Standardize Dose Entity*

Description

This function standardizes the dose entity.

Usage

```
stdzDose(x)
```

Arguments

`x`	character vector of extracted dose values

Details

Some dose strings may include multiple values and additional interpretation may be needed. For example '2-1' likely indicates a dose of 2 followed by a dose of 1. Currently it would be converted to the average of 1.5.

Value

numeric vector

Examples

```
stdzDose(c('one tablet', '1/2 pill', '1-3 tabs'))
```

stdzDoseChange *Standardize Dose Change Entity*

Description

This function standardizes the dose change entity.

Usage

```
stdzDoseChange(x)
```

Arguments

x	character vector of extracted dose change values

Value

character vector

Examples

```
stdzDoseChange(c('decreasing','dropped','increased'))
```

stdzDoseSchedule *Standardize Dose Schedule Entity*

Description

This function standardizes the dose schedule entity.

Usage

```
stdzDoseSchedule(x)
```

Arguments

x	character vector of extracted dose schedule values

Value

character vector

Examples

```
stdzDoseSchedule(c('tapered','weaned','TAPER'))
```

stdzDuration *Standardize Duration Entity*

Description

This function standardizes the duration entity.

Usage

```
stdzDuration(x)
```

Arguments

x	character vector of extracted duration values

Value

character vector

Examples

```
stdzDuration(c('1 month', 'three days', 'two-weeks'))
```

stdzFreq *Standardize Frequency Entity*

Description

This function standardizes the frequency entity.

Usage

```
stdzFreq(x)
```

Arguments

x	character vector of extracted frequency values

Value

character vector

Examples

```
stdzFreq(c('in the morning', 'four times a day', 'with meals'))
```

stdzRoute *Standardize Route Entity*

Description

This function standardizes the route entity.

Usage

```
stdzRoute(x)
```

Arguments

x	character vector of extracted route values

Value

character vector

Examples

```
stdzRoute(c('oral', 'po', 'subcut'))
```

stdzStrength *Standardize Strength Entity*

Description

This function standardizes the strength entity.

Usage

```
stdzStrength(str, freq)
```

Arguments

str	character vector of extracted strength values
freq	character vector of extracted frequency values

Details

Some strength strings may include multiple values and additional interpretation may be needed. For example '2-1' likely indicates a strength of 2 followed by a strength of 1. Thus a single element may need to be standarized into two elements. This can only happen if the frequency entity is missing or in agreement ('bid' for example). See the 'addl_data' attribute of the returned vector.

Value

numeric vector

Examples

```
stdzStrength(c('1.5', '1/2', '1/1/1'))
stdzStrength(c('1.5', '1/2', '1/1/1'), c('am', 'daily', NA))
stdzStrength(c('1.5', '1/2', '1/1/1'), FALSE)
```

---

tac_lab *Example of Lab Time Data for Tacrolimus*

Description

An example dataset used in processLastDose that contains lab time data. This dataset should have one row per patient ID-date pair, and contain the time a lab was performed as a datetime variable.

Usage

```
data(tac_lab, package = 'EHR')
```

Format

A data frame with 2 observations on the following variables.

pid A character vector, patient ID associated with the lab value

date A character vector, date associated with the lab value

labtime A POSIXct vector, datetime at which the lab was performed formatted as YYYY-MM-DD HH:MM:SS

Examples

```
data(tac_lab)
```

---

tac_metadata *Example of Metadata for Tacrolimus Data*

Description

An example of the metadata needed for the processLastDose, makeDose, and collapseDose functions.

Usage

```
data(tac_metadata, package = 'EHR')
```

Format

A data frame with 5 observations on the following variables.

filename A character vector, filename for the clinical note

pid A character vector, patient ID associated with the filename

date A character vector, date associated with the filename

note A character vector, note ID associated with the filename

Examples

```
data(tac_metadata)
```

---

tac_mxr_parsed *Example of Tacrolimus Output from 'parseMedExtractR'*

Description

The output after running `parseMedExtractR` on 3 example clinical notes.

Usage

```
data(tac_mxr_parsed, package = 'EHR')
```

Format

A data frame with 7 observations on the following variables.

filename A character vector, filename for the clinical note

drugname A character vector, drug name extracted from the clinical note along with start and stop positions

strength A character vector, strengths extracted from the clinical note along with start and stop positions

dose A character vector, dose amounts extracted from the clinical note along with start and stop positions

route A character vector, routes extracted from the clinical note along with start and stop positions

freq A character vector, frequencies extracted from the clinical note along with start and stop positions

dosestr A character vector, dose intakes extracted from the clinical note along with start and stop positions

dosechange A character vector, dose change keywords extracted from the clinical note along with start and stop positions

lastdose A character vector, last dose times extracted from the clinical note along with start and stop positions

Examples

```
data(tac_mxr_parsed)
```

zeroOneTable *Make Zero One Contingency Tables*

Description

Make contingency tables for many binary outcomes and a binary covariate

Usage

```
zeroOneTable(EXPOSURE, phenotype)
```

Arguments

EXPOSURE	binary covariate (e.g., exposure).
phenotype	binary outcome (e.g., phenotype).

Details

Generates frequency and contingency tables for many binary outcomes (e.g., large number of phenotypes) and a binary covariate (e.g., drug exposure, genotypes) more efficiently.

Value

t00	frequency for non-exposed group and non-case outcome.
t01	frequency for non-exposed group and case outcome.
t10	frequency for exposed group and non-case outcome.
t11	frequency for exposed group and case outcome.

Author(s)

Leena Choi <leena.choi@vanderbilt.edu> and Cole Beck <cole.beck@vumc.org>

Examples

```
## full example data
data(dataPheWAS)
demo.covariates <- c('id','exposure','age','race','gender')
phenotypeList <- setdiff(colnames(dd), demo.covariates)
tablePhenotype <- matrix(NA, ncol=4, nrow=length(phenotypeList),
dimnames=list(phenotypeList, c("n.nocase.nonexp", "n.case.nonexp",
"n.nocase.exp", "n.case.exp")))
for(i in seq_along(phenotypeList)) {
    tablePhenotype[i, ] <- zeroOneTable(dd[, 'exposure'], dd[, phenotypeList[i]])
}
```

Index

∗ **EHR**
  EHR-package, 3
∗ **PheWAS**
  EHR-package, 3
∗ **datasets**
  dd, 10
  dd.baseline, 11
  dd.baseline.small, 11
  dd.small, 12
  lam_metadata, 15
  lam_mxr_parsed, 16
  tac_lab, 44
  tac_metadata, 44
  tac_mxr_parsed, 45
∗ **process**
  EHR-package, 3

addLastDose, 4, *9*, *19*
analysisPheWAS, *3*, 5

buildDose, *4*, 7, *9*, *19*

collapseDose, *8*, 9, *15*, *44*

dataTransformation, 10, *26*, *35*
dd, 10
dd.baseline, 11
dd.baseline.small, 11
dd.small, 12

EHR *(*EHR-package*)*, 3
EHR-package, 3
extractMed, *4*, 12, *23*

freqNum, 13

idCrosswalk, 14, *24, 25*

lam_metadata, 15
lam_mxr_parsed, 16
Logistf, 17

makeDose, *9*, *15*, 18, *44*
medExtractR, *4*, *12, 13*, *23*

parseCLAMP, *7*, 19
parseMedEx, *7*, 20
parseMedExtractR, *7*, *16*, 21, *45*
parseMedXN, *7*, 22
processLastDose, *4*, *15*, 23, *44*
pullFakeId, 24
pullRealId, 25

read.csv, *26*
readTransform, 26
run_Build_PK_IV, 26
run_Build_PK_Oral, 30
run_Demo, *28*, 31
run_DrugLevel, *27*, 33
run_Labs, *28*, 35
run_MedStrI, *27*, 36
run_MedStrII, 39

stdzDose, 40
stdzDoseChange, 41
stdzDoseSchedule, 41
stdzDuration, 42
stdzFreq, 42
stdzRoute, 43
stdzStrength, 43

tac_lab, 44
tac_metadata, 44
tac_mxr_parsed, 45

zeroOneTable, *3*, 46