Package ‘PKPDmisc’

December 17, 2017

Type Package

Title Pharmacokinetic and Pharmacodynamic Data Management Functions

Version 2.1.1

Date 2017-12-15

Description A toolbox for data management common to pharmacokinetic and pharmacokinetic modeling and simulation, such as resampling, area-under-the-curve calculation, data chunking, custom csv output, and project scaffolding.

URL https://github.com/dpastoor/PKPDmisc

BugReports https://github.com/dpastoor/PKPDmisc/issues

Depends R (>= 3.2.2)

LinkingTo Rcpp, BH

Imports dplyr (>= 0.5.0), data.table, ggplot2, lazyeval, magrittr, parallel, purrr, Rcpp (>= 0.12.10), readr (>= 1.0.0), rlang, stringr, tibble

Suggests devtools, knitr, rmarkdown, testthat

LazyData TRUE

License MIT + file LICENSE

VignetteBuilder knitr

RoxygenNote 6.0.1

NeedsCompilation yes

Author Devin Pastoor [aut, cre]

Maintainer Devin Pastoor <devin.pastoor@gmail.com>

Repository CRAN

Date/Publication 2017-12-17 22:47:16 UTC
R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>as_numeric</td>
<td>3</td>
</tr>
<tr>
<td>auc_inf</td>
<td>4</td>
</tr>
<tr>
<td>auc_partial</td>
<td>4</td>
</tr>
<tr>
<td>auc_partial_cpp</td>
<td>5</td>
</tr>
<tr>
<td>basename_sans_ext</td>
<td>6</td>
</tr>
<tr>
<td>base_theme</td>
<td>6</td>
</tr>
<tr>
<td>bring_to_front</td>
<td>7</td>
</tr>
<tr>
<td>calculate_wam</td>
<td>7</td>
</tr>
<tr>
<td>capitalize_names</td>
<td>8</td>
</tr>
<tr>
<td>capture_groups</td>
<td>9</td>
</tr>
<tr>
<td>char_to_numeric</td>
<td>9</td>
</tr>
<tr>
<td>chunk_df</td>
<td>10</td>
</tr>
<tr>
<td>color_pallete</td>
<td>10</td>
</tr>
<tr>
<td>cols_to_factor</td>
<td>11</td>
</tr>
<tr>
<td>cols_to_numeric</td>
<td>12</td>
</tr>
<tr>
<td>cor_to_cov</td>
<td>12</td>
</tr>
<tr>
<td>cov_to_cor</td>
<td>13</td>
</tr>
<tr>
<td>dapa_IV_oral</td>
<td>13</td>
</tr>
<tr>
<td>file_rows</td>
<td>14</td>
</tr>
<tr>
<td>fill_backward</td>
<td>14</td>
</tr>
<tr>
<td>fill_forward</td>
<td>15</td>
</tr>
<tr>
<td>has_ext</td>
<td>15</td>
</tr>
<tr>
<td>ids_per_plot</td>
<td>16</td>
</tr>
<tr>
<td>is_dir</td>
<td>17</td>
</tr>
<tr>
<td>jprint</td>
<td>18</td>
</tr>
<tr>
<td>lowercase_names</td>
<td>18</td>
</tr>
<tr>
<td>max_through</td>
<td>19</td>
</tr>
<tr>
<td>min_through</td>
<td>20</td>
</tr>
<tr>
<td>nca</td>
<td>20</td>
</tr>
<tr>
<td>new_report</td>
<td>21</td>
</tr>
<tr>
<td>nm_description</td>
<td>21</td>
</tr>
<tr>
<td>nm_notes</td>
<td>22</td>
</tr>
<tr>
<td>nonmem_report</td>
<td>22</td>
</tr>
<tr>
<td>ordinal_to_binary_</td>
<td>23</td>
</tr>
<tr>
<td>pad_left</td>
<td>24</td>
</tr>
<tr>
<td>peek</td>
<td>24</td>
</tr>
<tr>
<td>phx_report</td>
<td>25</td>
</tr>
<tr>
<td>print_plots</td>
<td>26</td>
</tr>
<tr>
<td>read_nonmem</td>
<td>27</td>
</tr>
<tr>
<td>read_phx</td>
<td>28</td>
</tr>
<tr>
<td>rename_table_pattern</td>
<td>29</td>
</tr>
<tr>
<td>replace_dots</td>
<td>29</td>
</tr>
<tr>
<td>replace_empty</td>
<td>30</td>
</tr>
<tr>
<td>replace_from_template</td>
<td>31</td>
</tr>
<tr>
<td>replace_list_elements</td>
<td>31</td>
</tr>
<tr>
<td>replace_values</td>
<td>32</td>
</tr>
</tbody>
</table>
as_numeric

convert to numeric passing through character for safety

Description

convert to numeric passing through character for safety

Usage

as_numeric(x, ...)

Arguments

x vector

... additional argument to as.character

Examples

# factor with weird levels that we don't want to keep
ex <- factor(c(1, 2, 3, 4), levels = c(2, 3, 1, 4))
ex

# keeps information about the levels, oh no!
as.numeric(ex)

# keeps the labelled values
as_numeric(ex)
### auc_inf

**Calculate AUC_{t-infty}**

**Description**

Calculate AUC_{t-infty}

**Usage**

```r
auc_inf(idv, dv, last_points = c(3, 4, 5), na.rm = TRUE)
```

**Arguments**

- **idv**: column name for independent variable such as time
- **dv**: column name for dependent variable such as concentration
- **last_points**: vector of amount of points in terminal phase that will be evaluated for extrapolation
- **na.rm**: remove any NAs from the idv/dv column before calculating AUC

**Details**

last_points defaults to 3, 4, 5 see auc_partial for other details

### auc_partial

**Calculate partial AUC**

**Description**

Calculate partial AUC

**Usage**

```r
auc_partial(idv, dv, range = c(0, Inf))
```

**Arguments**

- **idv**: independent variable (such as time)
- **dv**: dependent variable (such as concentration)
- **range**: time range for pauc calculation

**Details**

default range is 0 to tmax is recommended to be used alongside dplyr for ease of calculation if an individual does not have any value within the specified range a warning will be issued and an NA value will be returned. This is important if some individuals dropped out early and do not have all observations other individuals have.
**Description**

Calculate partial AUC.

**Usage**

```r
auc_partial_cpp(time, dv, range)
```

**Arguments**

- `time`: vector of time values
- `dv`: concentration measurements
- `range`: vector of min and max value of the partial auc range

**Examples**

```r
## Not run:
library(dplyr)
sd_oral_richpk %>% group_by(ID) %>%
  summarize(pauc0_12 = auc_partial_cpp(Time, Conc, c(0, 12)))

## End(Not run)
```
**basename_sans_ext**

*get the basename of a filepath, minus any extensions*

**Description**

get the basename of a filepath, minus any extensions

**Usage**

`basename_sans_ext(.x)`

**Arguments**

- `.x`  filepath

---

**base_theme**

*A theme with better default values for pharmacometric plots, especially conc-time*

**Description**

A theme with better default values for pharmacometric plots, especially conc-time

**Usage**

`base_theme(legend_text = 10, legend_title = 12, axis_title_x = 12, axis_title_y = axis_title_x, axis_text_x = 10, axis_text_y = axis_text_x, strip_text_x = 12, strip_text_y = strip_text_x)`

**Arguments**

- `legend_text`  size of legend text
- `legend_title`  legend title
- `axis_title_x`  size of X axis title
- `axis_title_y`  size of Y axis title
- `axis_text_x`  size of X axis text
- `axis_text_y`  size of Y axis text
- `strip_text_x`  size of strip text for X axis
- `strip_text_y`  size of strip text for Y axis
Examples

```r
## Not run:
ggplot() + base_theme_obs()
made for viewing in the plot window and copying out

## End(Not run)
```

---

**bring_to_front**

*bring select columns to the front of a dataframe*

**Description**

bring select columns to the front of a dataframe

**Usage**

```r
bring_to_front(Ndf, NNN)
```

**Arguments**

- `.df` : dataframe with column order to adjust
- `...` : columns to bring to front

**Examples**

```r
head(Theoph)
head(bring_to_front(Theoph, conc, Time))
```

---

**calculate_wam**

*internal WAM function*

**Description**

internal WAM function

**Usage**

```r
calculate_wam(k, p, n, theta, cov)
```

**Arguments**

- `k` : number of covariate thetas
- `p` : total number of parameters in model (theta, omega, sigma)
- `n` : number of observations
- `theta` : vector of covariate parameter values
- `cov` : variance-covariance matrix for covariate parameters
capitalize_names

capitalize all names for a dataframe

details

returns model rank, the LRT, SBC, which covariates selected for internal use with wam()

value

data.frame

description

capitalize all names for a dataframe

usage

capitalize_names(df)

arguments

df
data frame to capitalize names

details

is a simple wrapper function to reduce typing and more easily pass data as it is read from a file

examples

names(Theoph)
cTheoph <- capitalize_names(Theoph)
names(cTheoph)
## Not run:
df <- capitalize_names(df)
# make sure all names are capitalized as a file is read
df <- capitalize_names(read.csv(...))

## End(Not run)
capture_groups

wrapper to capture groups from a grouped data frame

Description

wrapper to capture groups from a grouped data frame

Usage

capture_groups(df)

Arguments

df frame to determine groups

Examples

## Not run:
gTheoph <- dplyr::group_by(Theoph, Subject)
capture_groups(gTheoph)

## End(Not run)

char_to_numeric

convert all columns to numeric

Description

defaults to convert all character columns except named 'C' to numeric useful for nonmem ready datasets with '.' for missing values

Usage

char_to_numeric(df, exclude_cols = "C")

Arguments

df dataframe to convert character columns to numeric
exclude_cols vector of column names to be skipped from conversion

Value

dataframe
Examples

```r
## Not run:
nm_dat <- char_to_numeric(nm_dat)
  # if 'C' col is 0/1 rather than typical 'C' or '.'
nm_dat <- char_to_numeric(nm_dat, exclude_cols = NULL)
## End(Not run)
```

```r
color_pallete

Description
design-quality color palletes to use in ggplot2

Usage

color_pallete(pallette)
```
cols_to_factor

Arguments

pallate          pallate number or name

Examples

library(PKPDmisc)
library(ggplot2)
library(dplyr)
ggplot(sd_oral_richpk %>% filter(ID < 10),
aes(x = Time,
y= Conc,
group = ID,
color = Gender)) +
  geom_line(size = 1.5) + scale_color_manual(values =color_pallete(1)) +
  base_theme()

---

cols_to_factor                  convert column to type factor

Description

convert column to type factor

Usage

cols_to_factor(df, col_names)

Arguments

df                    dataframe
col_names             vector of column names or indices to convert to factor

Examples

## Not run:
df <- cols_to_factor(df, c("DOSE", "TRT")) # will convert DOSE and TRT columns to factor
## End(Not run)
cols_to_numeric  
convert column to type numeric

**Description**

convert column to type numeric

**Usage**

cols_to_numeric(df, col_names)

**Arguments**

df  
dataframe

col_names  
vector of column names or indices to convert to numeric

**Examples**

```r
## Not run:
df <- cols_to_numeric(df, c("DOSE", "TRT")) # will convert dose and TRT columns to numeric
## End(Not run)
```

cor_to_cov  
correlation to covariance matrix

**Description**

correlation to covariance matrix

**Usage**

cor_to_cov(.cor_mat, .sd)

**Arguments**

_.cor_mat_  
correlation matrix

_.sd_  
vector of standard deviations
**cov_to_cor**

**Description**

covariance to correlation

**Usage**

cov_to_cor(.cov_mat)

**Arguments**

.cov_mat    covariance matrix

**Value**

list of unit correlation matrix and vector of standard deviations

**dapa_IV_oral**

**IV and oral pharmacokinetic data for daptomycin**

**Description**

A dataset containing simulated dapagliflozin PK. A single IV dose followed by 3 escalating oral doses

**Usage**

data(dapa_IV_oral)

**Format**

A data frame with 1536 rows and 11 variables

**Details**

- ID. Numerical ID (1-24)
- TIME. Nominal Time after first dose (hrs)
- TAD. Time After Dose (hrs)
- COBS. Observed Concentration (ug/L)
- AMT_IV. IV dose amount when given (ug)
- AMT_ORAL. Oral dose amount when given (ug)
- OCC. Occasion, associated with each dosing event
• AGE. Age (years)
• WEIGHT Weight (kg)
• GENDER. Gender flag (female=1, male=0)
• FORMULATION Formulation associated with dose (IV or oral)

---

file_rows

**count number of rows in a file without needing to read file into R**

**Description**

count number of rows in a file without needing to read file into R

**Usage**

`file_rows(file)`

**Arguments**

- `file` file path to be read in

**Details**

uses `wc -l` system call to count rows

---

fill_backward

given NA values fill them with the next non-na value

**Description**

given NA values fill them with the next non-na value

**Usage**

`fill_backward(x)`

**Arguments**

- `x` A numeric vector of values

**Details**

Works very well in context of dplyr to carry out backwards imputation
fill_forward

Examples

```r
fill_backward(c(1.0, NA, 2))
fill_backward(c(NA, 1, NA, 2))
library(dplyr)
df <- data.frame(id = c(1, 1, 2, 2), obs = c(1.2, 4.8, 2.5, NA))
df %>% group_by(id) %>% mutate(obs_imp = fill_backward(obs))
```

Description

given NA values fill them with the final non-na value

Usage

```r
fill_forward(x)
```

Arguments

- `x` A numeric vector of values

Details

Works very well in context of dplyr to carry out last-observation-carried-forward for different individuals. It will NOT replace leading NA's

Examples

```r
fill_forward(c(1.0, NA, 2))
fill_forward(c(NA, 1, NA, 2))
library(dplyr)
df <- data.frame(id = c(1, 1, 2, 2), obs = c(1.2, 4.8, 2.5, NA))
df %>% group_by(id) %>% mutate(obs_locf = fill_forward(obs))
```

has_ext

Description

check for if the passed name has a file extension

Usage

```r
has_ext(name, ext, match_case = TRUE)
```
Argument

- **name**: string name to check for extension
- **ext**: string of extension to check
- **match_case**: logical whether to match the case when checking extension. Defaults to TRUE

Details

This is not a particularly robust checker, but serves its purpose

Examples

```r
## Not run:
has_ext("test.rmd", ".rmd") # TRUE
has_ext("test.Rmd", ".rmd", match_case=F) # TRUE
has_ext("testmd", ".rmd") # FALSE

## End(Not run)
```

### ids_per_plot

**split IDs into groups to use for subsequent plotting**

#### Description

split IDs into groups to use for subsequent plotting

#### Usage

```r
ids_per_plot(id, id_per_plot = 9)
chunk(.x, .nchunk = parallel::detectCores())
chunk_grp(.x, .nchunk = parallel::detectCores())
chunk_list(.x, .nchunk = parallel::detectCores())
chunk_grp_list(.x, .nchunk = parallel::detectCores())
```

#### Arguments

- **id**: vector of ids (eg id column)
- **id_per_plot**: number of ids per plot. Default to 9
- **.x**: vector of values
- **.nchunk**: number of chunks to identify
is_dir

detect if a filepath is for a directory

Description

detect if a filepath is for a directory

Usage

is_dir(x)

Arguments

x vector of file paths
jprint

print multiple values joined together

Description

print multiple values joined together

Usage

jprint(..., sep_atomic = " ", sep_vector = ", ")

Arguments

... variadic parameter to of objects to combine
sep_atomic separator value between each atomic element
sep_vector separator value for collapsed vectors

Details

print can only take one value to be printed, this function j(oined)print provides a wrapper to obviate
the need to do print(paste0(...)) type work-arounds
This is especially valuable for messages where output can be single element or a vector/list (for
example, a list of missing column names), as the internal vector/list can be collapsed to a single
string via a different pattern than how the various elements are combined.

Examples

result <- "world"
jprint("hello", result)

missing.cols <- c("WT", "HT", "OCC") #would normally actually calculate this
jprint("missing columns: ", missing.cols)
jprint("missing columns: ", missing.cols, sep_vector="; ")
jprint("missing cols", missing.cols, sep_atomic=". ")

lowercase_names

lowercase all names for a dataframe

Description

lowercase all names for a dataframe

Usage

lowercase_names(df)
max_through

Arguments

  df  data frame to lowercase names

Details

  is a simple wrapper function to reduce typing and more easily pass data as it is read from a file

Examples

  ## Not run:
  df <- lowercase_names(df)
  # make sure all names are lowered as a file is read
  df <- lowercase_names(read.csv(...))
  # or via dplyr pipe syntax
  df <- read.csv(...) %>% lowercase_names

  ## End(Not run)

max_through

  give the max value up to that point

Description

  give the max value up to that point

Usage

  max_through(x)

Arguments

  x  A numeric vector of values

Details

  useful for safety analyses where an event may be defined as a certain change in a biomarker, so need to see how the current measurement compares to the maximum value up to that point

Examples

  max_through(c(4, 3, 3, 2, 5, 1))
  max_through(c(NA, 2, 1, 4, 2))
**min_through**

give the min value up to that point

**Description**

give the min value up to that point

**Usage**

min_through(x)

**Arguments**

x 
A numeric vector of values

**Details**

useful for safety analyses where an event may be defined as a certain change in a biomarker, so need to see how the current measurement compares to the minimum value up to that point

**Examples**

min_through(c(4, 3, 2, 4, 1))
min_through(c(NA, 2))

---

**nca**

calculate basic NCA parameters t1/2, Cl, Cmax etc

**Description**

calculate basic NCA parameters t1/2, Cl, Cmax etc

**Usage**

nca(.time, dv, dose, last_times = c(3, 4, 5), digits = 2)

NCA(...)

**Arguments**

.time vector of times
dv vector of observations (concentrations)
dose vector or single value of Dose given
last_times vector of numbers of time points to evaluate for AUCinf extrapolation default 3-5
digits number of digits to round to. Can use NULL for no rounding
... arguments to pass to nca
new_report

**new_report**

*a better new rmd template for reports*

**Description**

a better new rmd template for reports

**Usage**

`new_report(report)`

**Arguments**

- `report`: file name and (optionally) path to subdirectory

**Examples**

```r
## Not run:
new_report("lab-notebook/Report.Rmd")
new_report("Report.Rmd")
## End(Not run)
```

nm_description

**nm_description**

*show description output from .mod file*

**Description**

show description output from .mod file

**Usage**

`nm_description(filename)`

**Arguments**

- `filename`: name of .mod file

**Details**

return line containing description file

**Value**

string

**See Also**

`nm_notes`
**nm_notes**

*show notes output from .mod file*

**Description**

show notes output from .mod file

**Usage**

```
nm_notes(filename)
```

**Arguments**

filename name of .mod file

**Details**

returns all lines between ;NOTES and ;ENDNOTES in .mod file if filename not in current wd, need to give path as well

**Value**

string

**See Also**

nm_description

**Examples**

````
## Not run:
nm_notes("run102.mod")
```

```
## End(Not run)
```

**nonmem_report**

*create basic folder structure for a nonmem report*

**Description**

create basic folder structure for a nonmem report

**Usage**

```
nonmem_report(project = NULL, secondary_folders = list("scripts",
             "modeling", "lab-notebook", "data", "reports"))
```
ordinal_to_binary

Arguments

project          name of top level project folder
secondary_folders    list of subfolders

Details

A placeholder.txt file is created in each subfolder to allow for initial folder structure to be maintained if using a version control system such as git (as empty directories are not kept)

Currently only works if there is no project folder with the same name, and does not do any checking (will just error out)

Examples

```r
## Not run:
nonmem_report("Drug-x")
nonmem_report("Drug-x", list("Rscripts", "nonmem","CTS", "data"))

## End(Not run)
```

---

**ordinal_to_binary_**   convert a column of categorical covariates into a number of columns with a binary flag for each category

Description

convert a column of categorical covariates into a number of columns with a binary flag for each category

Usage

`ordinal_to_binary_(df, col_name, prefix = NULL, overwrite = FALSE)`

Arguments

- **df**          data frame
- **col_name**    column name to split into multiple binary flags
- **prefix**      string name if want to specify a prefix to label the binary flag columns other than the original col_name
- **overwrite**   overwrite any existing columns if the newly generated columns share the same name
Examples

```r
library(dplyr)
df <- data_frame(OCC = c(1, 1, 2, 3))
df %>% ordinal_to_binary("OCC")
df %>% ordinal_to_binary("OCC", prefix = "OCCasion")

df2 <- data_frame(OCC = c(1, 1, 2, 3), OCC1 = 999)
df2 %>% ordinal_to_binary("OCC")
df2 %>% ordinal_to_binary("OCC", overwrite=T)
```

---

**pad_left**  
*add left padding to a vector of values*

**Description**

add left padding to a vector of values

**Usage**

```r
pad_left(.x, .n, padding_char = "0")
```

**Arguments**

- `.x` vector
- `.n` total number of characters result should have
- `padding_char` padding char to use, defaults to 0

**Examples**

```r
pad_left(1, 3)
pad_left(c(1, 10, 100), 4)
```

---

**peek**  
*peek at the results in a dplyr pipeline*

**Description**

peek at the results in a dplyr pipeline

**Usage**

```r
peek(df, n = 5, message = NULL)
```
Arguments

- **df**: dataframe in pipeline
- **n**: number of rows to show
- **message**: give a message along with peeking at the data

Details

A wrapper around giving a head and tail command but only as a side effect so the original data frame is passed along to continue on for further manipulation in the pipeline.

Examples

```r
library(dplyr)
theoph %>% select(Subject, Time, conc) %>% peek %>% group_by(Subject) %>% summarize(cmax = max(conc))

theoph %>% select(Subject, Time, conc) %>% peek(message = "after select") %>%
  group_by(Subject) %>%
  summarize(cmax = max(conc))

# nice for saving full objects but still seeing their output
cmax_theoph <- theoph %>% select(Subject, Time, conc) %>%
  peek(message = "after select") %>%
  group_by(Subject) %>%
  summarize(cmax = max(conc)) %>% peek

cmax_theoph # still saves full output
```

---

**phx_report**

create basic folder structure for a phoenix report

**Description**

create basic folder structure for a phoenix report

**Usage**

```r
phx_report(project, secondary_folders = list("Rscripts", "phoenix-output", "for-phoenix", "lab-notebook", "data"))
```

**Arguments**

- **project**: name of top level project folder
- **secondary_folders**: list of subfolders
print_plots

description

create a list of plots cleanly with extra pdf functionality

usage

print_plots(.ggplot_list, .start_page_number = NULL, .start_break = TRUE, .end_break = TRUE)

arguments

.ggpplot_list       list of ggplot plots
.start_page_number  pdf-only starting page number for plots
.start_break        whether to add a page break before starting to print plots
.end_break          whether to add a page break after the plot output

details

Especially for pdf, this can allow the generation of clean pdf pages with only plots, no code, warnings, etc. for all pages related to the plots. In addition, by controlling the start number, you can further trim the pdf to slice out the extra pages generated from the output but keep a nicely numbered plot appendix.
Examples

```r
## Not run:
library(dplyr)
library(PKPmisc)

# given we may only plot a subset of individuals per plot
# and generate multiple plots, lets split the dataframe
list_of_ids <- sd_oral_richpk %>% capitalize_names() %>%
mutable(plotnum = ids_per_plot(ID)) %>% # default 9 per plot
  split(.splitnum)

# now we want to plot each subplot
plot_list <- list_of_ids %>%
  lapply(function(df) {
    df %>%
      ggplot(aes(x = TIME, y = CONC, group = ID)) +
      geom_line() + facet_wrap(~ID)
  })

# to print these out (with one plot per page on pdf)
print_plots(plot_list)

## End(Not run)
```

---

**Description**

read nonmem files easily

**Usage**

```r
read_nonmem(path, header = TRUE, sep = "auto", example_name = NULL)
```

**Arguments**

- `path` path to file
- `header` whether header with column names exists
- `sep` automatically detected by default, however can tell by default.
- `example_name` name of column to detect which rows contain header(s)

**Details**

This function is designed specifically for handling nonmem’s nonstandard output format, and is especially useful for simulation tables output with NSUB as it will appropriately parse out the additional TABLE and column name rows.
HOWEVER, for tables with standard formatting (eg comma separated with FORMAT=,1PE11.4) and no NSUB, then the ‘read_phx()’ function will likely be slightly faster. This should only be an issue for large (at least 20 MB) files, else the difference will be imperceptible.

```
read_phx to more easily read data with a 2nd row with units common to phx data
```

**Description**

to more easily read data with a 2nd row with units common to phx data

**Usage**

```r
read_phx(data, skip = 0, header = TRUE, sep = "auto",
    stringsAsFactors = FALSE, has_units = FALSE, fread = TRUE,
    data.table = FALSE, ...)
```

**Arguments**

- `data` the name of the csv
- `skip` rows to skip before header row
- `header` logical value indicating whether the file contains the names of variables as it’s first line
- `sep` field separator character. With fread defaults to 'auto', else defaults to ",,"
- `stringsAsFactors` logical value whether to include string columns as factors
- `has_units` logical value whether has a units row below the header (eg phx nlme data)
- `fread` logical value. Use fread from data.table for much faster reading
- `data.table` logical value. When using fread, whether to return data.table (TRUE) or data.frame (FALSE)
- `...` additional arguments to read.csv functions

**Details**

helpful function to handle situations where the second row is a units row as often seen with phoenix-style datasets

**Value**

data frame of read-in csv

**Examples**

```r
## Not run:
read_phx("example.csv")
read_phx("example.csv", skip = 1) # will ignore 1st line, good for comment lines

## End(Not run)
```
rename_table_pattern

```
rename_table_pattern  simple table name replacement for a given pattern
```

### Description

simple table name replacement for a given pattern

### Usage

```
rename_table_pattern(df, pattern = " ", replacement = "_")
```

### Arguments

- **df**: dataframe to rename
- **pattern**: pattern to replace
- **replacement**: value to replace pattern

### Details

a helper to easily wrap around reading tables to perform simple replacement

### Examples

```
# Not run:
# rename all spaces to underscore
df <- rename_table_pattern(read_table("data/data.csv"))
# rename "-" to "_"
df <- rename_table_pattern(read_table("data/data.csv"), "-", "_")
# rename an existing df
df <- rename_table_pattern(df, "-", "_")

# End(Not run)
```

---

replace_dots

```
replace_dots  Convert "." values into missing values.
```

### Description

Convert "." values into missing values.

### Usage

```
replace_dots(x)
```

### Arguments

- **x**: A character vector
Details

good for nonmem preprocessing as missing values are represented with '.' in NONMEM

Value

A character vector with '.' replaced by NA values.

Examples

```r
x <- c(".", "1", "1")
replace_dots(x)
```

---

**replace_empty**

*Convert empty strings into missing values.*

Description

Convert empty strings into missing values.

Usage

```r
replace_empty(x)
```

Arguments

- `x` A character vector

Value

A character vector with empty strings replaced by missing values.

Examples

```r
x <- c("a", "", "c")
replace_empty(x)
```
**replace_from_template**  
*replace values from a template file*

**Description**

replace values from a template file

**Usage**

```r
replace_from_template(file, patterns, replacement_df, output_dir = NULL,  
file_name = NULL, ...)
```

**Arguments**

- `file`: template file
- `patterns`: patterns to match
- `replacement_df`: dataframe of replacements
- `output_dir`: set directory if not current directory
- `file_name`: column in replacement containing requested output file name
- `...`: additional arguments for readLines

**Details**

the pattern will first check for names, and if matching names in replacement df will follow, however if no names are detected, then will match by position

**replace_list_elements**  
*replace list elements by name*

**Description**

replace list elements by name

**Usage**

```r
replace_list_elements(.list, replacement, add = TRUE)
```

**Arguments**

- `.list`: original list
- `replacement`: replacement list
- `add`: add replacement values if they don’t exist, default = TRUE
**replace_values**

Details

Finds and replaces list elements by name and throws an error if an element is not available in the original list.

Examples

```r
olist <- list(dv = "dv", idv = "idv", notreplace = "somethingelse")
replacement <- list(dv = "conc")
combined_list <- replace_list_elements(olist, replacement)
```

---

**Description**

replace symbols or other character flags

**Usage**

```r
replace_values(x, flag_df, nonflag = NULL, as_numeric = TRUE)
```

**Arguments**

- `x`: vector to replace
- `flag_df`: dataframe with first column being the flag and second the replacement value
- `nonflag`: what to do with non-flagged elements
- `as_numeric`: whether to return column as numeric

**Details**

because of R’s coercion rules being somewhat unpredictable regarding character vs factor the behavior of replace_values is to always treat as character data, even if passed in as a factor

**See Also**

- `unique_non_numerics`: to identify which non-numeric values must be replaced before column can be safely coerced to a numeric

**Examples**

```r
df <- data.frame(ID = 1, DV = c(1, "BQL", ".", 5))
rflags <- data.frame(flag = c("BQL", "."), replacement = -99)
D$DVR <- replace_values(D$DV, rflags)

library(dplyr)
df <- df %>% mutate(DVR2 = replace_values(DV, rflags))
```
# powerful with unique_non_numerics
```r
df <- df %>% mutate(DV3 = replace_values(DV,
  data.frame(values = unique_non_numerics(DV),
  replacement = NA))
```

## resample_df

### Description

Resampling

### Usage

```r
resample_df(df, key_cols, strat_cols = NULL, n = NULL,
  key_col_name = "KEY", replace = TRUE)
```

### Arguments

- **df**: data frame
- **key_cols**: key columns to resample on
- **strat_cols**: columns to maintain proportion for stratification
- **n**: number of unique sampled keys, defaults to match dataset
- **key_col_name**: name of outputted key column. Default to "KEY"
- **replace**: whether to stratify with replacement

### Details

This function is valuable when generating a large simulated population where your goal is to create resampled sub-populations in addition to being able to maintain certain stratifications of factors like covariate distributions.

A new keyed column will be created (defaults to name 'KEY') that contains the uniquely created new samples. This allows one to easily compare against the key’d columns. Eg, if you would like to see how many times a particular individual was resampled you can check the original ID column against the number of key’s associated with that ID number.

### Examples

```r
library(PKPDmisc)
library(dplyr, quiet = TRUE)

# simple example resampling by ID maintaining Gender distribution, with 10 individuals
resample_df(sd_oral_richpk, key_cols = "ID", strat_cols = "Gender", n = 10)

# for a more complex example lets resample "simulated" data with multiple replicates
subset_data <- sd_oral_richpk %>%
  filter(ID < 20)
```
rf51_factory

factory to create read fort51 file

Description

factory to create read fort51 file

Usage

rf51_factory(.names)

Arguments

.names names of columns in the fort51 file

Examples

rf51 <- rf51_factory(c("Iteration", "ID", "CL", "v", "LNTVCL", "LNTVV", "CLEGFR", "nCL", "nV")

# could now do rf51("path/to/fort.51")
**round_columns**

**Description**

round columns

**Usage**

round_columns(df, col_list)

**Arguments**

df  
data frame

col_list  
list of vectors of columns and number of digits to round each

**Details**

col_list should contain a list of list tuples, where the first element is a vector of column names to round, and the second value should be the number of digits to round to

**Examples**

```r
## Not run:
col_list <- list(
  list("AGE", 1),
  list(c("WEIGHT", "CREATinine", "SERUMALT", "DELDBP", "DELSBP"), 2)
)
aht_trial_rounded <- round_columns(aht_trial, col_list)

## End(Not run)
```

---

**sd_oral_richpk**

One-compartment pharmacokinetic data given single oral dose

**Description**

A dataset containing dose, plasma concentration and time data, as well as demographic data of Age, Weight, Gender, and Race for each individual
Usage

data(sd_oral_richpk)

Format

A data frame with 4300 rows and 9 variables

Details

• ID. Numerical ID (1–50)
• Time. Time in hours (0–24)
• Amt. Amount of drug given, time dependent, in milligrams
• Conc. Plasma concentration in mg/L
• Age. Age in years
• Weight. Weight in kg
• Gender. Male or Female gender identification
• Race. Ethnicity
• Dose. Dose given to each individual in milligrams

---

set_bins

given a set of bin ranges, assign each value to a bin

Description

given a set of bin ranges, assign each value to a bin

Usage

set_bins(x, breaks = stats::quantile(x, na.rm = T), lower_bound = -Inf,
upper_bound = Inf, quiet = TRUE, between = NULL, inclusive = TRUE)

Arguments

x numeric vector to assign bins
breaks breaks for each bin, defaults to quantiles
lower_bound set a lower bound for the first bin, defaults to -Inf
upper_bound set an upper bound for the last bin, defaults to Inf
quiet whether to give additional information regarding bins and assigned range for each
between defaults to NULL, a special case of setting all inside the specified range
inclusive include max value of largest user defined bin even though lower bins are non-inclusive
Details

Given a set of quantiles/bins/etc established from a separate dataset, it can be useful to assign the same bins to new or simulated data for comparisons or to do additional analysis such as assign dropouts etc. This function can be used to take the breakpoints to establish bins quickly and easily.

If there is concern over data being outside the range of the assigned breaks, one can assign -Inf to lower and/or Inf to upper to make sure all values will be assigned to a bin.

To use the between functionality, you must specify the range you wish to bin between, and those values will be assigned to bin 1, with all values below as 0 and all values above as 2. See the examples for more details.

---

set_bins_cpp

*given a set of bin ranges, assign each value to a bin*

---

Description

given a set of bin ranges, assign each value to a bin

Usage

```r
set_bins_cpp(x, left, right)
```

Arguments

- `x`: A numeric vector of values
- `left, right`: Boundary values

Details

Given a set of quantiles/bins/etc established from a separate dataset, it can be useful to assign the same bins to new or simulated data for comparisons or to do additional analysis such as assign dropouts etc. This function can be used to take the breakpoints to establish bins quickly and easily.

---

set_bins_df

*given a set of bin ranges, assign each value to a bin and provide the label*

---

Description

given a set of bin ranges, assign each value to a bin and provide the label

Usage

```r
set_bins_df(.df, .x, breaks = stats::quantile(.df[[.x]], na.rm = T), .name = NULL, .label = NULL, lower_bound = -Inf, upper_bound = Inf, quiet = TRUE, between = NULL, inclusive = TRUE)
```
set_groups

Arguments

- `.df` data frame
- `.x` name of column
- `breaks` breaks for each bin, defaults to quantiles
- `.name` name of new binned column, defaults to appending `_bin` to column name
- `.label` name of the new label column, defaults to appending `_label` to bin column name
- `lower_bound` set a lower bound for the first bin, defaults to `-Inf`
- `upper_bound` set an upper bound for the last bin, defaults to `Inf`
- `quiet` whether to give additional information regarding bins and assigned range for each
- `between` defaults to `NULL`, a special case of setting all inside the specified range
- `inclusive` include max value of largest user defined bin even though lower bins are non-inclusive

Details

`set_bins_df` offers the ability to create bins from a dataframe and get both the binning column as well as a label column with the range of values associated with a given bin.

See Also

- `set_bins`

Description

set groups

Usage

`set_groups(df, groups)`

Arguments

- `df` data frame
- `groups` list of groups to pass to `group_by`

Details

useful in tandem with `capture_groups` when concerned about modification of groups by a function, for example when summarizing with `dplyr` Can easily capture and reset groups to maintain original grouping
Examples

```r
## Not run:
gTheoph <- dplyr::group_by(Theoph, Subject)
grps <- capture_groups(gTheoph) # capture subject
theoph_cmax <- summarize(gTheoph, cmax = max(conc)) # lose Subject grouping
theoph_cmax <- set_groups(theoph_cmax, grps) # resets the original "Subject" grouping

## End(Not run)
```

Description

strip additional attributes that make dplyr fail

Usage

```
strip_attributes(df, attr_names)
```

Arguments

- **df**
  - dataframe
- **attr_names**
  - names of attributes that you want to remove

Details

dplyr as of 0.4 still does not handle columns with non-generic attributes and will error out rather than ignoring them etc. This function will allow one to strip attribute names to allow the data frame to be used within the dplyr pipeline without issue.

This type of data is common when dealing with SAS datasets

Examples

```r
## Not run:
foo <- data.frame(a = 1:5, b = 1:5, c=letters[1:5])
df <- foo
attr(df$a, "label") <- "col a"
attr(df$b, "label") <- "col b"
attr(df$c, "label") <- "col c"

library(dplyr)
df %>% filter(a %in% c(1, 2)) # will throw an error
df %>% strip_attributes("label") %>% filter(a %in% c(1, 2))

attr(df$a, "notes") <- "a note"
# now column a has attributes of label and notes
df %>% strip_attributes(c("label", "notes")) %>% filter(a %in% c(1, 2))

## End(Not run)
```
**strip_curves**

`basic curve stripping to get initial estimates`

**Description**

basic curve stripping to get initial estimates

**Usage**

```r
strip_curves(.time, .dv, dose, number_terminal_points, oral = FALSE,
round = 2)
```

**Arguments**

- `.time` column for time
- `.dv` column for DV (concentration) values
- `dose` Dose value or column
- `number_terminal_points` number of points in terminal phase
- `oral` whether data is oral (instead of IV)
- `round` number of decimals to round, default to 2

**Details**

for oral stripping, if multiple cmax values found per ID, will use the first

**Examples**

```r
## Not run:
strip_curves(df$TIME, df$DV, dose =1000, 5, oral=TRUE)
df %>% group_by(ID) %>% do(data.frame(strip_curves(.TIME, .DV, 1000, 5, TRUE)))

## End(Not run)
```

---

**s_pauc_**

`summarize paucs`

**Description**

summarize paucs

**Usage**

```r
s_pauc_(df, idv, dv, paucs, digits = Inf)
s_pauc(df, idv, dv, paucs, digits = Inf)
```
**s_quantiles_**

**Arguments**

- **df**: data frame
- **idv**: string name for time column for pauc slice
- **dv**: string name for dependent variable column (e.g., dv or cobs)
- **paucs**: list of ranges for pauc calculation
- **digits**: number of decimals to round result before returning

**Examples**

```r
library(dplyr)
sd_oral_richpk %>% group_by(ID) %>% s_pauc(Time, Conc, list(c(0, 8), c(8, 24)))
sd_oral_richpk %>% group_by(ID) %>% s_pauc("Time", "Conc", list(c(0, 8), c(8, 24)))
```

---

**s_quantiles_**

summarize quantiles for a column

**Description**

summarize quantiles for a column

**Usage**

`s_quantiles_(.data, x, probs, na_rm = TRUE)`

**Arguments**

- **.data**: data frame
- **x**: column to calculate quantiles for
- **probs**: probabilities to calculate quantiles for
- **na_rm**: remove na’s before calculating value for quantile

**Examples**

```r
library(dplyr)
sd_oral_richpk %>% group_by(Gender, Time) %>% s_quantiles(Conc, c(0.05, 0.5, 0.95))
```
**unique_non_numerics**  
*find all unique non-numeric values*

**Description**

*find all unique non-numeric values*

**Usage**

```r
unique_non_numerics(x, na.rm = TRUE)
```

**Arguments**

- `x`  
  vector to check on
- `na.rm`  
  remove existing na values before checking

**Details**

This function is especially useful for figuring out what non-numeric unique values are in in a column that should be numeric so one can easily replace them with another flag. This function can work well with `replace_char_flags` instead of using nested ifelse statements

**See Also**

`replace_values`: to use to replace non-numeric values in a dataframe.

**Examples**

```r
dv <- c(1, 2, 4, "88 (excluded)", "bql", "*")
unique_non_numerics(dv)
df <- tibble::data_frame(ID = 1:3, DV = c("BQL", 0.5, 9))
unique_non_numerics(df$DV)

# using dplyr
library(dplyr)
df %>% filter(!(DV %in% unique_non_numerics(DV)))
```

---

**view_creator**  
*create view commands that save rds files to where a shiny app is listening for them*

**Description**

create view commands that save rds files to where a shiny app is listening for them
Usage

view_creator(path, save_non_interactive = FALSE)

Arguments

path path to shiny app directory
save_non_interactive whether to save data as RDS file while non-interactive mode such as knitting

Details

will, by default, set up to save RDS files to the shiny app to render them, and can take some optional arguments in the resulting function. Namely, can rename the file via name, and can determine whether the function should return a dataframe with return = T/F. The value of returning a function is view<x> can be embedded in a data pipeline to output intermediate results as well, while continuing on the pipeline

Value

a function with the path set

Examples

```r
## Not run:
path <- "~/Repos/dataView" # dataView shiny app location
view2 <- view_creator(path)
View2(Theoph) # will save a file Theoph.rds in dataView
View2(Theoph, "theoph1") # will save a file theoph1.rds
View2(Theoph, return = F) # will not return Theoph as well

## End(Not run)
```

wam function

Description

wam function

Usage

wam(ncovtheta, cov_theta_nums, npar, nobs, run_num, dir = NULL)
write_nonmem

Arguments

- `ncovtheta`: number of covariate thetas
- `cov_theta_nums`: <- vector of theta numbers for covariate thetas
- `npar`: number of parameters in model (theta, omega, sigma)
- `nobs`: number of observations
- `run_num`: run number
- `dir`: directory where .cov and .ext files located

Details

for runnum it is recommended to pass as a string or leading 0's will be stripped

---

write_nonmem

 easily write a csv file compatible with nonmem

---

Description

easily write a csv file compatible with nonmem

Usage

```r
write_nonmem(x, file, sep = ',', row.names = FALSE, na = '.',
              quote = FALSE, ...)
```

Arguments

- `x`: dataframe to be written to csv
- `file`: character string naming a file or connection open for writing.
- `sep`: field string separator, defaults to comma (";")
- `row.names`: logical value whether to include row names
- `na`: value for NA
- `quote`: whether character or factor columns should be surrounded by double quotes
- `...`: remaining arguments passed to data.table::fwrite

Details

nonmem uses '.' for NA values, does not like quotes in column names and does not handle row names, so these are all presets

Examples

```r
## Not run:
write_nonmem(nonmemdat, 'folder/nonmemdat.csv')
```

## End(Not run)
Index

*Topic datasets
  dapa_IV_oral, 13
  sd_oral_richpk, 35

as_numeric, 3
auc_inf, 4
auc_partial, 4
auc_partial_cpp, 5
base_theme, 6
basename_sans_ext, 6
bring_to_front, 7
calculate_wam, 7
capitalize_names, 8
capture_groups, 9
char_to_numeric, 9
chunk(ids_per_plot), 16
chunk_df, 10
chunk_grp(ids_per_plot), 16
chunk_grp_list(ids_per_plot), 16
chunk_list(ids_per_plot), 16
color_pallete, 10
cols_to_factor, 11
cols_to_numeric, 12
cor_to_cov, 12
cov_to_cor, 13
dapa_IV_oral, 13
file_rows, 14
fill_backward, 14
fill_forward, 15
has_ext, 15
ids_per_plot, 16
is_dir, 17
jprint, 18
lowercase_names, 18
max_through, 19
min_through, 20
NCA (nca), 20
nca, 20
new_report, 21
nm_description, 21
nm_notes, 22
nonmem_report, 22
ordinal_to_binary_, 23
pad_left, 24
peek, 24
phx_report, 25
print_plots, 26
read_nonmem, 27
read_phx, 28
rename_table_pattern, 29
replace_dots, 29
replace_empty, 30
replace_from_template, 31
replace_list_elements, 31
replace_values, 32, 42
resample_df, 33
rf51_factory, 34
round_columns, 35
s_pauc, 5
s_pauc (s_pauc_), 40
s_pauc_, 40
s_quantiles (s_quantiles_), 41
s_quantiles_, 41
sd_oral_richpk, 35
set_bins, 36, 38
set_bins_cpp, 37
set_bins_df, 37
set_groups, 38
strip_attributes, 39
strip_curves, 40
unique_non_numerics, 32, 42

view_creator, 42

wam, 43

write_nonmem, 44