Package ‘TreatmentPatterns’

August 10, 2023

Type Package

Title Analyzes Real-World Treatment Patterns of a Study Population of Interest

Version 2.5.0

Author Aniek F Markus

Maintainer Maarten van Kessel `<m.l.vankessel@erasmusmc.nl>`

Description Computes treatment patterns within a given cohort using the Observational Medical Outcomes Partnership (OMOP) common data model (CDM).

Depends R (>= 4.0)

Imports DatabaseConnector (>= 6.0.0),
checkmate,
dplyr,
SqlRender,
stringr,
stringi,
fs,
glue,
utils,
rjson,
googleVis,
stats,
Andromeda,
tidyr,
data.table,
R6

Suggests knitr,
rmarkdown,
tibble,
testthat (>= 3.0.0),
usethis,
Eunomia,
CDMConnector,
CohortGenerator,
webshot2,
CirceR,
duckdb,
DBI,
withr

License Apache License 2.0

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

VignetteBuilder knitr

Config/testthat/edition 3

Additional_repositories `https://ohdsi.github.io/drat`

Roxygen list(markdown = TRUE)

R topics documented:

`addChild` *addChild*

Description

addChild

Usage

```
addChild(j, children, parts, root)
```

Arguments

`j`	iterator
`children`	children to add
`parts`	labels of treatments
`root`	root list

Value

root list with added childs

`addLabels` *addLabels*

Description

Adds back cohort names to concept ids.

Usage

```
addLabels(andromeda)
```

Arguments

`andromeda`	`(Andromeda::andromeda())`

Value

(invisible(NULL))

`buildHierarchy` *buildHierarchy*

Description

buildHierarchy

Usage

```
buildHierarchy(csv)
```

Arguments

`csv`	matrix

Value

JSON

`CDMInterface` *CDMInterface*

Description

Abstract interface to the CDM, using CDMConnector or DatabaseConnector.

Methods

Public methods:

- `CDMInterface$new()`
- `CDMInterface$validate()`
- `CDMInterface$fetchCohortTable()`
- `CDMInterface$addAge()`
- `CDMInterface$addSex()`
- `CDMInterface$fetchMetadata()`
- `CDMInterface$clone()`

Method `new()`**:** Initializer method

Usage:

```
CDMInterface$new(
  connectionDetails = NULL,
  cdmSchema = NULL,
  resultSchema = NULL,
  cdm = NULL
)
```

Arguments:

`connectionDetails` (`DatabaseConnector::createConnectionDetails()`: NULL)
Optional; In congruence with `cdmSchema` and `resultSchema`. Ignores `cdm`.

`cdmSchema` (`character(1)`: NULL)
Optional; In congruence with `connectionDetails` and `resultSchema`. Ignores `cdm`.

`resultSchema` (`character(1)`: NULL)
Optional; In congruence with `connectionDetails` and `cdmSchema`. Ignores `cdm`.

`cdm` (`CDMConnector::cdm_from_con()`: NULL)
Optional; Ignores `connectionDetails`, `cdmSchema`, and `resultSchema`.

Returns: `(invisible(self))`

Method `validate()`**:** Validation method

Usage:

```
CDMInterface$validate()
```

Returns: `(invisible(self))`

Method `fetchCohortTable()`**:** Fetch specified cohort IDs from a specified cohort table

Usage:

```
CDMInterface$fetchCohortTable(cohortIds, cohortTableName)
```

Arguments:

`cohortIds (integer(1))`
Cohort ID's of cohorts to investigate.

`cohortTableName (character(1))`
Cohort table name.

Returns: `(data.frame)`

Method `addAge()`**:** Stratisfy the treatmentHistory data frame by age.

Usage:

`CDMInterface$addAge(andromeda)`

Arguments:

`andromeda` (`Andromeda::andromeda()`) Andromeda object.

`andromeda` (`Andromeda::andromeda()`) Andromeda object.

`andromeda` (`Andromeda::andromeda()`) Andromeda object.

Returns: `(data.frame())`

Method `addSex()`**:** Stratisfy the treatmentHistory data frame by sex.

Usage:

`CDMInterface$addSex(andromeda)`

Arguments:

`andromeda` (`Andromeda::andromeda()`) Andromeda object.

`andromeda` (`Andromeda::andromeda()`) Andromeda object.

`andromeda` (`Andromeda::andromeda()`) Andromeda object.

Returns: `(data.frame())`

Method `fetchMetadata()`**:** Fetch metadata from CDM

Usage:

`CDMInterface$fetchMetadata(andromeda)`

Arguments:

`andromeda` (`Andromeda::andromeda()`) Andromeda object.

`andromeda` (`Andromeda::andromeda()`) Andromeda object.

`andromeda` (`Andromeda::andromeda()`) Andromeda object.

Returns: `(invisible(NULL))`

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`CDMInterface$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

computeCounts *computeCounts*

Description

computeCounts

Usage

```
computeCounts(treatmentHistory, minFreq)
```

Arguments

`treatmentHistory`	(`data.frame()`) Patient level Treatment History data.frame
`minFreq`	(`integer(1)`: 5) Minimum frequency required per pathway. Censors data below `x` as <x. This minimum value will carry over to the sankey diagram and sunburst plot.

Value

(`list()`)

computePathways *computePathways*

Description

Compute treatment patterns according to the specified parameters within specified cohorts.

Usage

```
computePathways(
  cohorts,
  cohortTableName,
  cdm = NULL,
  connectionDetails = NULL,
  cdmSchema = NULL,
  resultSchema = NULL,
  includeTreatments = "startDate",
  periodPriorToIndex = 0,
  minEraDuration = 0,
  splitEventCohorts = "",
  splitTime = 30,
  eraCollapseSize = 30,
  combinationWindow = 30,
  minPostCombinationDuration = 30,
  filterTreatments = "First",
  maxPathLength = 5,
  addNoPaths = TRUE
)
```

Arguments

`cohorts`	(`data.frame()`) Data frame containing the following columns and data types: **cohortId** `numeric(1)` Cohort ID's of the cohorts to be used in the cohort table. **cohortName** `character(1)` Cohort names of the cohorts to be used in the cohort table. **type** `character(1)` **["target", "event', "exit"**] Cohort type, describing if the cohort is a target, event, or exit cohort
`cohortTableName`	(`character(1)`) Cohort table name.
`cdm`	(`CDMConnector::cdm_from_con(): NULL`) Optional; Ignores `connectionDetails`, `cdmSchema`, and `resultSchema`.
`connectionDetails`	(`DatabaseConnector::createConnectionDetails(): NULL`) Optional; In congruence with `cdmSchema` and `resultSchema`. Ignores `cdm`.
`cdmSchema`	(`character(1): NULL`) Optional; In congruence with `connectionDetails` and `resultSchema`. Ignores `cdm`.
`resultSchema`	(`character(1): NULL`) Optional; In congruence with `connectionDetails` and `cdmSchema`. Ignores `cdm`.
`includeTreatments`	(`character(1): "startDate"`)
`periodPriorToIndex`	(`integer(1): 0`) Number of days prior to the index date of the target cohort	that event cohorts are allowed to start
`minEraDuration`	(`integer(1): 0`) Minimum time an event era should last to be included in analysis
`splitEventCohorts`	(`character(n): ""`) Specify event cohort to split in acute (< X days) and therapy (>= X days)
`splitTime`	(`integer(1): 30`) Specify number of days (X) at which each of the split event cohorts should be split in acute and therapy
`eraCollapseSize`	(`integer(1): 30`) Window of time between which two eras of the same event cohort are collapsed into one era
`combinationWindow`	(`integer(1): 30`) Window of time two event cohorts need to overlap to be considered a combination treatment
`minPostCombinationDuration`	(`integer(1): 30`) Minimum time an event era before or after a generated combination treatment should last to be included in analysis

`filterTreatments`	(`character(1): "First"` ["first", "Changes", "all"]) Select first occurrence of (‘First’); changes between (‘Changes’); or all event cohorts (‘All’).
`maxPathLength`	(`integer(1): 5`) Maximum number of steps included in treatment pathway
`addNoPaths`	(`logical(1): TRUE`) Select to include untreated persons without treatment pathway in the sunburst plot

Value

(`Andromeda::andromeda()`) andromeda object containing non-sharable patient level data outcomes.

`computeStatsTherapy` *computeStatsTherapy*

Description

computeStatsTherapy

Usage

```
computeStatsTherapy(treatmentHistory)
```

Arguments

`treatmentHistory`	(`data.frame()`) Patient level Treatment History data.frame

Value

(`data.frame()`)

`computeTreatmentPathways` *computeTreatmentPathways*

Description

computeTreatmentPathways

Usage

```
computeTreatmentPathways(treatmentHistory, ageWindow, minFreq)
```

Arguments

`treatmentHistory`	(`data.frame()`) Patient level Treatment History data.frame
`ageWindow`	(`integer(1)`: `10`) Number of years to bin age groups into.
`minFreq`	(`integer(1)`: `5`) Minimum frequency required per pathway. Censors data below `x` as `<x`. This minimum value will carry over to the sankey diagram and sunburst plot.

Value

`(data.frame())`

`constructPathways` *constructPathway*

Description

Constructs the pathways.

Usage

```
constructPathways(settings, cohorts, andromeda)
```

Arguments

`settings`	(`data.frame`)
`cohorts`	(`data.frame`)
`andromeda`	(`Andromeda::andromeda()`)

Value

`invisible(NULL)`

`createSankeyDiagram` *createSankeyDiagram*

Description

Writes the Sankey diagram to a HTML-file, to a specified file path.

Usage

```
createSankeyDiagram(
  treatmentPathways,
  outputFile,
  groupCombinations = FALSE,
  minFreq = 5
)
```

Arguments

`treatmentPathways`	(`data.frame()`) The contents of the treatmentPathways.csv-file as a data.frame().
`outputFile`	(`character(1)`) Path where the Sankey diagram should be written to.
`groupCombinations`	(`logical(1): FALSE`) `TRUE` Group all combination treatments in category `"Combination"`. `FALSE` Do not group combination treatments.
`minFreq`	(`integer(1): 5`) Minimum frequency required per pathway. Censors data below `x` as `<x`. This minimum value will carry over to the sankey diagram and sunburst plot.

Value

invisible(NULL)

Examples

```
if (FALSE) {
  treatmentPathways <- read.csv(treatmentPathways.csv)
  outputFile <- "mySankeyDiagram.html"
  groupCombinations <- FALSE
  minFreq <- 5

  createSankeyDiagram(
    treatmentPathways,
    outputFile,
    groupCombinations,
    minFreq
  )
}
```

`createSunburstPlot` *createSunburstPlot*

Description

Generate a sunburst plot from the treatment pathways.

Usage

```
createSunburstPlot(treatmentPathways, outputFile)
```

Arguments

`treatmentPathways`	(`data.frame()`) The contents of the treatmentPathways.csv-file as a data.frame().
`outputFile`	(`character(1)`) Path where the Sankey diagram should be written to.

Value

`(NULL)`

Examples

```
if (FALSE) {
  treatmentPathways <- read.csv("treatmentPathways.csv")
  outputFile <- "mySunburstPlot.html"

  createSunburstPlot(
    treatmentPatwhays,
    outputFile
  )
}
```

`createTreatmentHistory` *createTreatmentHistory*

Description

createTreatmentHistory

Usage

```
createTreatmentHistory(
  andromeda,
  targetCohortIds,
  eventCohortIds,
  exitCohortIds,
  periodPriorToIndex,
  includeTreatments
)
```

Arguments

`andromeda`	(`Andromeda::andromeda()`) Andromeda object.
`targetCohortIds`	(`numeric(n)`)
`eventCohortIds`	(`numeric(n)`)
`exitCohortIds`	(`numeric(n)`)
`periodPriorToIndex`	(`integer(1)`: `0`) Number of days prior to the index date of the target cohort | that event cohorts are allowed to start
`includeTreatments`	(`character(1)`: `"startDate"`)

Value

`(data.frame())`

1. (`numeric()`) person_id
2. (`numeric()`) index_year
3. (`numeric()`) event_cohort_id
4. (`as.Date()`) event_start_date
5. (`as.Date()`) event_end_date
6. (`character()`) type
7. (`difftime()`) duration_era
8. (`difftime()`) gap_same

`createTreatmentPathways` *createTreatmentPathways*

Description

createTreatmentPathways

Usage

```
createTreatmentPathways(treatmentHistory)
```

Arguments

`treatmentHistory`	`(data.frame())`

Value

`(data.frame())`

`depth` *depth*

Description

Function to find depth of a list element.

Usage

```
depth(x, thisdepth = 0)
```

Arguments

`x`	input list (element)
`thisdepth`	current list depth

Value

the depth of the list element

`doCombinationWindow` *Combine overlapping events into combinations*

Description

doCombinationWindow is an internal function that combines overlapping events into combination events. It accepts a treatmentHistory dataframe and returns a modified treatmentHistory dataframe. The returned treatmentHistory dataframe always has the property that a person is only in one event cohort, which might be a combination event cohort, at any point time.

Usage

```
doCombinationWindow(andromeda, combinationWindow, minPostCombinationDuration)
```

Arguments

`andromeda`	`(Andromeda::andromeda())`
`combinationWindow`	`(integer(1))`
`minPostCombinationDuration`	`(integer(1))`

Value

`(invisible(NULL))`

`doEraCollapse` *doEraCollapse*

Description

Updates the treatmentHistory data.frame where if gapSame is smaller than the specified era collapse size (eraCollapseSize) are collapsed

Usage

```
doEraCollapse(andromeda, eraCollapseSize)
```

Arguments

`andromeda`	`(Andromeda::andromeda())`
`eraCollapseSize`	`(integer(1))`

Value

`(invisible(NULL))`

`doEraDuration` *doEraDuration*

Description

doEraDuration

Usage

```
doEraDuration(andromeda, minEraDuration)
```

Arguments

`andromeda`	`(Andromeda::andromeda())`
`minEraDuration`	`(integer(1))`

Value

`(invisible(NULL))`

`doFilterTreatments` *doFilterTreatments*

Description

Updates the treatmentHistory data.frame where the desired event cohorts are maintained for the visualizations

Usage

```
doFilterTreatments(andromeda, filterTreatments)
```

Arguments

`andromeda`	`(Andromeda::andromeda())`
`filterTreatments`	`(character(1))`

Value

`(invisible(NULL))`

`doSplitEventCohorts` *doSplitEventCohorts*

Description

Splits the treatmentHistory data.frame based on event cohorts into ‘acute’ and ‘therapy’ cohorts.

Usage

```
doSplitEventCohorts(andromeda, splitEventCohorts, splitTime)
```

Arguments

`andromeda`	`(Andromeda::andromeda())`
`splitEventCohorts`	`(character(n))`
`splitTime`	`(integer(1))`

Value

`(invisible(NULL))`

`executeTreatmentPatterns` *executeTreatmentPatterns*

Description

Compute treatment patterns according to the specified parameters within specified cohorts. For more customization, or investigation of patient level outcomes, you can run computePathways and export separately.

Usage

```
executeTreatmentPatterns(
  cohorts,
  cohortTableName,
  outputPath,
  cdm = NULL,
  connectionDetails = NULL,
  cdmSchema = NULL,
  resultSchema = NULL,
  includeTreatments = "startDate",
  periodPriorToIndex = 0,
  minEraDuration = 0,
  splitEventCohorts = "",
  splitTime = 30,
  eraCollapseSize = 30,
  combinationWindow = 30,
  minPostCombinationDuration = 30,
  filterTreatments = "First",
  maxPathLength = 5,
  minFreq = 5,
  addNoPaths = TRUE
)
```

Arguments

`cohorts`
: `(data.frame())`
Data frame containing the following columns and data types:

  cohortId `numeric(1)` Cohort ID's of the cohorts to be used in the cohort table.

  cohortName `character(1)` Cohort names of the cohorts to be used in the cohort table.

  type `character(1)` **["target", "event', "exit"**] Cohort type, describing if the cohort is a target, event, or exit cohort

`cohortTableName`
: `(character(1))`
Cohort table name.

`outputPath`
: `(character(1))`

`cdm`
: `(CDMConnector::cdm_from_con(): NULL)`
Optional; Ignores `connectionDetails`, `cdmSchema`, and `resultSchema`.

`connectionDetails`
: `(DatabaseConnector::createConnectionDetails(): NULL)`
Optional; In congruence with `cdmSchema` and `resultSchema`. Ignores `cdm`.

`cdmSchema`
: `(character(1): NULL)`
Optional; In congruence with `connectionDetails` and `resultSchema`. Ignores `cdm`.

`resultSchema`
: `(character(1): NULL)`
Optional; In congruence with `connectionDetails` and `cdmSchema`. Ignores `cdm`.

`includeTreatments`
: `(character(1): "startDate")`

`periodPriorToIndex`
: `(integer(1): 0)`
Number of days prior to the index date of the target cohort | that event cohorts are allowed to start

`minEraDuration`
: `(integer(1): 0)`
Minimum time an event era should last to be included in analysis

`splitEventCohorts`
: `(character(n): "")`
Specify event cohort to split in acute (< X days) and therapy (>= X days)

`splitTime`
: `(integer(1): 30)`
Specify number of days (X) at which each of the split event cohorts should be split in acute and therapy

`eraCollapseSize`
: `(integer(1): 30)`
Window of time between which two eras of the same event cohort are collapsed into one era

`combinationWindow`	(`integer(1)`: 30) Window of time two event cohorts need to overlap to be considered a combination treatment
`minPostCombinationDuration`	(`integer(1)`: 30) Minimum time an event era before or after a generated combination treatment should last to be included in analysis
`filterTreatments`	(`character(1)`: "First"` ["first", "Changes", "all"]) Select first occurrence of ('First'); changes between ('Changes'); or all event cohorts ('All').
`maxPathLength`	(`integer(1)`: 5) Maximum number of steps included in treatment pathway
`minFreq`	(`integer(1)`: 5) Minimum frequency required per pathway. Censors data below `x` as `<x`. This minimum value will carry over to the sankey diagram and sunburst plot.
`addNoPaths`	(`logical(1)`: TRUE) Select to include untreated persons without treatment pathway in the sunburst plot

Value

`(invisible(NULL))`

Examples

```
if (FALSE) {
  # Using CDMConnector

  con <- DBI::dbConnect(duckdb::duckdb(), dbdir = eunomia_dir())
  cdm <- cdm_from_con(con, cdm_schema = "main")

  # <Code to compute cohorts into Cohort Table>

  cohortTableName <- "CohortTable"

  cohorts <- data.frame(
    cohortId = c(1, 2, 3, 4, 5),
   cohortName = c("ViralSinusitis", "Acetaminophen", "Aspirin", "Clavulanate", "Death"),
    type = c("target", "event", "event", "event", "exit")
  )

  outputPath <- "./output/"

  executeTreatmentPatterns(
    cohorts,
    cohortTableName,
    outputPath,
    cdm
  )

  # Using DatabaseConnector
  connectionDetails <- Eunomia::getEunomiaConnectionDetails()
  cdmSchema <- "main"
  resultSchema <- "main"

  executeTreatmentPatterns(
    cohorts,
    cohortTableName,
    outputPath,
    connectionDetails,
    cdmSchema,
    resultSchema
  )
}
```

`export` *export*

Description

Export andromeda generated by computePathways object to sharable csv-files and/or a zip archive.

Usage

```
export(
  andromeda,
  outputPath = ".",
  ageWindow = 10,
  minFreq = 5,
  archiveName = NULL
)
```

Arguments

`andromeda`	(`Andromeda::andromeda()`) Andromeda object.
`outputPath`	(`character(1)`)
`ageWindow`	(`integer(1): 10`) Number of years to bin age groups into.
`minFreq`	(`integer(1): 5`) Minimum frequency required per pathway. Censors data below `x` as `<x`. This minimum value will carry over to the sankey diagram and sunburst plot.
`archiveName`	(`character(1): NULL`) If not `NULL` adds the exported files to a ZIP-file with the specified archive name.

Value

`(invisible(NULL))`

Examples

```
if (FALSE) {
  andromeda <- computePathways(
    cohorts,
    cohortTableName,
    cdm
  )

  export(
    andromeda,
    outputPath = "./",
    ageWindow = 10,
    minFreq = 5,
    archiveName = "output.zip"
  )
}
```

`PathwayConstructor` *PathwayConstructor*

Description

PathwayConstructor R6 object.

Methods

Public methods:

- `PathwayConstructor$new()`
- `PathwayConstructor$validate()`
- `PathwayConstructor$construct()`
- `PathwayConstructor$getAndromeda()`
- `PathwayConstructor$editSettings()`
- `PathwayConstructor$getSettings()`
- `PathwayConstructor$clone()`

Method `new()`**:** Initialize method called by `PathwayConstructor$new()`.

Choose the way you interface with the CDM, either through `DatabaseConnector` or `CDMConnector`.

Usage:

`PathwayConstructor$new(cohorts, cohortTableName, cdmInterface)`

Arguments:

`cohorts (data.frame())`
Data frame containing the following columns and data types:
cohortId `numeric(1)` Cohort ID's of the cohorts to be used in the cohort table.
cohortName `character(1)` Cohort names of the cohorts to be used in the cohort table.
type `character(1)` **["target", "event', "exit"**] Cohort type, describing if the cohort is a target, event, or exit cohort

`cohortTableName (character(1))`
Cohort table name.

`cdmInterface` (`TreatmentPatterns::cdmInterface`)
  A cdmInterface object created internally

Returns: `(invisible(self))`

Method `validate()`**:** Validation method

Usage:

```
PathwayConstructor$validate()
```

Returns: `(invisible(self))`

Method `construct()`**:** Construct the pathways. Generates `Andromeda::andromeda()` objects, which can be fetched using `self$getAndromeda()`.

Usage:

```
PathwayConstructor$construct()
```

Method `getAndromeda()`**:** Gets the `Andromeda::andromeda()` objects in a list.

Usage:

```
PathwayConstructor$getAndromeda()
```

Returns: `(list())`

Method `editSettings()`**:** Edit settings

Usage:

```
PathwayConstructor$editSettings(
  includeTreatments = "startDate",
  periodPriorToIndex = 0,
  minEraDuration = 0,
  splitEventCohorts = "",
  splitTime = 30,
  eraCollapseSize = 30,
  combinationWindow = 30,
  minPostCombinationDuration = 30,
  filterTreatments = "First",
  maxPathLength = 5,
  addNoPaths = TRUE
)
```

Arguments:

`includeTreatments (character(1): "startDate")`

`periodPriorToIndex (integer(1): 0)`
  Number of days prior to the index date of the target cohort | that event cohorts are allowed to start

`minEraDuration (integer(1): 0)`
  Minimum time an event era should last to be included in analysis

`splitEventCohorts (character(n): "")`
  Specify event cohort to split in acute (< X days) and therapy (>= X days)

`splitTime (integer(1): 30)`
  Specify number of days (X) at which each of the split event cohorts should be split in acute and therapy

`eraCollapseSize (integer(1): 30)`
  Window of time between which two eras of the same event cohort are collapsed into one era

`combinationWindow` (integer(1): 30)
: Window of time two event cohorts need to overlap to be considered a combination treatment

`minPostCombinationDuration` (integer(1): 30)
: Minimum time an event era before or after a generated combination treatment should last to be included in analysis

`filterTreatments` (character(1): "First" ["first", "Changes", "all"])
: Select first occurrence of ('First'); changes between ('Changes'); or all event cohorts ('All').

`maxPathLength` (integer(1): 5)
: Maximum number of steps included in treatment pathway

`addNoPaths` (logical(1): TRUE)
: Select to include untreated persons without treatment pathway in the sunburst plot

Returns: `(data.frame())`

Method `getSettings()`**:** Getter method to get the specified settings

Usage:

```
PathwayConstructor$getSettings()
```

Returns: `(data.frame())`

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PathwayConstructor$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`prepData` *prepData*

Description

prepData

Usage

```
prepData(treatmentHistory, year)
```

Arguments

`treatmentHistory`	`(data.frame())`
`year`	`(integer(1))`

Value

`(data.frame())`

`selectRowsCombinationWindow` *selectRowsCombinationWindow*

Description

Help function for doCombinationWindow that selects one overlapping drug era per person to modify in next iteration of the combination window.

Usage

```
selectRowsCombinationWindow(andromeda)
```

Arguments

`andromeda`	(`Andromeda::andromeda()`)

Value

`(invisible(NULL))`

`stratisfy` *stratisfy*

Description

stratisfy

Usage

```
stratisfy(treatmentHistory, years, ages)
```

Arguments

`treatmentHistory`	(`data.frame()`) Patient level Treatment History data.frame
`years`	(`vector("character")`)
`ages`	(`vector("character")`)

Value

(`data.frame()`)

`stripname` *stripname*

Description

Recursive function to remove name from all levels of list.

Usage

```
stripname(x, name)
```

Arguments

`x`	input list
`name`	the name of the list item from which the names will be removed

Value

list with removed names

`transformCSVtoJSON` *transformCSVtoJSON*

Description

Help function to transform data in csv format to required JSON format for HTML.

Usage

```
transformCSVtoJSON(data, outcomes)
```

Arguments

`data`	`(data.frame())`
`outcomes`	`(c())`

Value

the transformed csv as a json string

Index