

Package ‘TreatmentPatterns’

April 16, 2024

Type Package

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

Version 2.6.6

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). As described in Markus, Verhamme, Kors, and Rijnbeek (2022) <[doi:10.1016/j.cmpb.2022.107081](https://doi.org/10.1016/j.cmpb.2022.107081)>.

URL <https://github.com/darwin-eu-dev/TreatmentPatterns>

BugReports <https://github.com/darwin-eu-dev/TreatmentPatterns/issues>

Language en-US

Depends R (>= 4.2)

Imports checkmate, dplyr, stringr, utils, stats, Andromeda, tidyr, R6, sunburstR, networkD3, htmlwidgets, shiny, shinydashboard

Suggests knitr, rmarkdown, tibble, testthat (>= 3.0.0), usethis, Eunomia, CDMConnector, DatabaseConnector (>= 6.0.0), SqlRender, CohortGenerator, webshot2, CirceR, duckdb, DBI, withr, plotly, ggplot2, Capr

License Apache License 2.0

Encoding UTF-8

RoxygenNote 7.3.1

VignetteBuilder knitr

Config/testthat/edition 3

Config/testthat/parallel true

Additional_repositories <https://ohdsi.github.io/drat>

Collate 'CDMInterface.R' 'ShinyModule.R' 'CharacterizationPlots.R' 'InputHandler.R' 'InteractivePlots.R' 'SankeyDiagram.R' 'ShinyApp.R' 'SunburstPlot.R' 'TreatmentPatterns-package.R' 'computePathways.R' 'constructPathways.R' 'createSankeyDiagram.R' 'createSunburstPlot.R' 'executeTreatmentPatterns.R' 'export.R'

NeedsCompilation no

Author Aniek Markus [aut] (<<https://orcid.org/0000-0001-5779-4794>>),
Maarten van Kessel [cre] (<<https://orcid.org/0009-0006-8832-6030>>)

Repository CRAN

Date/Publication 2024-04-16 15:10:06 UTC

R topics documented:

CharacterizationPlots	2
computePathways	4
createSankeyDiagram	6
createSankeyDiagram2	8
createSunburstPlot	9
createSunburstPlot2	10
executeTreatmentPatterns	11
export	13
InputHandler	15
launchResultsExplorer	17
SankeyDiagram	17
ShinyModule	18
SunburstPlot	19

Index **21**

CharacterizationPlots *CharacterizationPlots*

Description

Class to handle the characterization plots.

Super class

[TreatmentPatterns::ShinyModule](#) -> CharacterizationPlots

Methods

Public methods:

- [CharacterizationPlots\\$uiMenu\(\)](#)
- [CharacterizationPlots\\$uiBody\(\)](#)
- [CharacterizationPlots\\$server\(\)](#)
- [CharacterizationPlots\\$clone\(\)](#)

Method [uiMenu\(\)](#): Method to include a [menuItem](#) to link to the body.

Usage:

```
CharacterizationPlots$uiMenu(  
  label = "Characteristics",  
  tag = "characteristics"  
)
```

Arguments:

label (character(1))
Label to show for the menuItem.
tag (character(1))
Tag to use internally in input.

Returns: (menuItem)

Method uiBody(): Method to include a [tabItem](#) to include the body.

Usage:

```
CharacterizationPlots$uiBody()
```

Returns: (tabItem)

Method server(): Method to handle the back-end.

Usage:

```
CharacterizationPlots$server(input, output, session, inputHandler)
```

Arguments:

input (input)
Input from the server function.
output (output)
Output from the server function.
session (session)
Session from the server function.
inputHandler (inputHandler)
[InputHandler](#) class.

Returns: (NULL)

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

computePathways	<i>computePathways</i>
-----------------	------------------------

Description

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

Usage

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

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.
tempEmulationSchema	Schema used to emulate temp tables
includeTreatments	(character(1): "startDate") "startDate" Include treatments after the target cohort start date and onwards. "endDate" Include treatments before target cohort end date and before.
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

Value

(Andromeda::andromeda()) [andromeda](#) object containing non-sharable patient level data outcomes.

Examples

```

library(TreatmentPatterns)
library(CDMConnector)
library(dplyr)

if (require("CirceR", character.only = TRUE, quietly = TRUE)) {
  withr::local_envvar(
    EUNOMIA_DATA_FOLDER = Sys.getenv("EUNOMIA_DATA_FOLDER", unset = tempfile())
  )

  downloadEunomiaData(overwrite = TRUE)

  con <- DBI::dbConnect(duckdb::duckdb(), dbdir = eunomia_dir())
  cdm <- cdmFromCon(con, cdmSchema = "main", writeSchema = "main")

  cohortSet <- readCohortSet(
    path = system.file(package = "TreatmentPatterns", "exampleCohorts")
  )

  cdm <- generateCohortSet(
    cdm = cdm,
    cohortSet = cohortSet,
    name = "cohort_table"
  )

  cohorts <- cohortSet %>%
    # Remove 'cohort' and 'json' columns
    select(-"cohort", -"json") %>%
    mutate(type = c("event", "event", "event", "event", "exit", "event", "event", "target")) %>%
    rename(
      cohortId = "cohort_definition_id",
      cohortName = "cohort_name",
    ) %>%
    select("cohortId", "cohortName", "type")

  outputEnv <- computePathways(
    cohorts = cohorts,
    cohortTableName = "cohort_table",
    cdm = cdm
  )

  Andromeda::close(outputEnv)
  DBI::dbDisconnect(con, shutdown = TRUE)
}

```

Description

Create sankey diagram.

Usage

```
createSankeyDiagram(  
  treatmentPathways,  
  groupCombinations = FALSE,  
  colors = NULL,  
  ...  
)
```

Arguments

treatmentPathways	(data.frame()) The contents of the treatmentPathways.csv-file as a data.frame().
groupCombinations	(logical(1): FALSE) TRUE Group all combination treatments in category "Combination". FALSE Do not group combination treatments.
colors	(character(n)) Vector of hex color codes.
...	Paramaters for sankeyNetwork .

Value

(htmlwidget)

Examples

```
# Dummy data, typically read from treatmentPathways.csv  
treatmentPathways <- data.frame(  
  path = c("Acetaminophen", "Acetaminophen-Amoxicillin+Clavulanate",  
           "Acetaminophen-Aspirin", "Amoxicillin+Clavulanate", "Aspirin"),  
  freq = c(206, 6, 14, 48, 221),  
  sex = rep("all", 5),  
  age = rep("all", 5),  
  index_year = rep("all", 5)  
)  
  
createSankeyDiagram(treatmentPathways)
```

createSankeyDiagram2 *createSankeyDiagram2*

Description

DEPRECATED Create sankey diagram.

Usage

```
createSankeyDiagram2(
  treatmentPathways,
  groupCombinations = FALSE,
  colors = NULL,
  ...
)
```

Arguments

treatmentPathways
 (data.frame())
 The contents of the treatmentPathways.csv-file as a data.frame().

groupCombinations
 (logical(1): FALSE)
 TRUE Group all combination treatments in category "Combination".
 FALSE Do not group combination treatments.

colors
 (character(n)) Vector of hex color codes.

... Paramaters for [sankeyNetwork](#).

Value

(htmlwidget)

Examples

```
# Dummy data, typically read from treatmentPathways.csv
treatmentPathways <- data.frame(
  path = c("Acetaminophen", "Acetaminophen-Amoxicillin+Clavulanate",
           "Acetaminophen-Aspirin", "Amoxicillin+Clavulanate", "Aspirin"),
  freq = c(206, 6, 14, 48, 221),
  sex = rep("all", 5),
  age = rep("all", 5),
  index_year = rep("all", 5)
)

createSankeyDiagram(treatmentPathways)
```

createSunburstPlot *createSunburstPlot*

Description

New sunburstPlot function

Usage

```
createSunburstPlot(treatmentPathways, groupCombinations = FALSE, ...)
```

Arguments

treatmentPathways
(data.frame())
The contents of the treatmentPathways.csv-file as a data.frame().

groupCombinations
(logical(1): FALSE)

TRUE Group all combination treatments in category "Combination".
FALSE Do not group combination treatments.

... Paramaters for [sunburst](#).

Value

(htmlwidget)

Examples

```
# Dummy data, typically read from treatmentPathways.csv
treatmentPatwhays <- data.frame(
  path = c("Acetaminophen", "Acetaminophen-Amoxicillin+Clavulanate",
           "Acetaminophen-Aspirin", "Amoxicillin+Clavulanate", "Aspirin"),
  freq = c(206, 6, 14, 48, 221),
  sex = rep("all", 5),
  age = rep("all", 5),
  index_year = rep("all", 5)
)

createSunburstPlot(treatmentPatwhays)
```

createSunburstPlot2 *createSunburstPlot2*

Description

DEPRECATED New sunburstPlot function

Usage

```
createSunburstPlot2(treatmentPathways, groupCombinations = FALSE, ...)
```

Arguments

treatmentPathways
(data.frame())
The contents of the treatmentPathways.csv-file as a data.frame().

groupCombinations
(logical(1): FALSE)

TRUE Group all combination treatments in category "Combination".
FALSE Do not group combination treatments.

... Paramaters for [sunburst](#).

Value

(htmlwidget)

Examples

```
# Dummy data, typically read from treatmentPathways.csv
treatmentPatwhays <- data.frame(
  path = c("Acetaminophen", "Acetaminophen-Amoxicillin+Clavulanate",
           "Acetaminophen-Aspirin", "Amoxicillin+Clavulanate", "Aspirin"),
  freq = c(206, 6, 14, 48, 221),
  sex = rep("all", 5),
  age = rep("all", 5),
  index_year = rep("all", 5)
)

createSunburstPlot2(treatmentPatwhays)
```

 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,
  tempEmulationSchema = NULL,
  minEraDuration = 0,
  eraCollapseSize = 30,
  combinationWindow = 30,
  minCellCount = 5
)
```

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.
tempEmulationSchema	(character(1)) Schema to emulate temp tables.
minEraDuration	(integer(1): 0) Minimum time an event era should last to be included in analysis
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
minCellCount	(integer(1): 5) Minimum count 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

```
library(TreatmentPatterns)
library(CDMConnector)
library(dplyr)
if (require("CirceR", character.only = TRUE, quietly = TRUE)) {
  withr::local_envvar(
    EUNOMIA_DATA_FOLDER = Sys.getenv("EUNOMIA_DATA_FOLDER", unset = tempfile())
  )

  downloadEunomiaData(overwrite = TRUE)

  con <- DBI::dbConnect(duckdb::duckdb(), dbdir = eunomia_dir())
  cdm <- cdmFromCon(con, cdmSchema = "main", writeSchema = "main")

  cohortSet <- readCohortSet(
    path = system.file(package = "TreatmentPatterns", "exampleCohorts")
  )

  cdm <- generateCohortSet(
    cdm = cdm,
    cohortSet = cohortSet,
    name = "cohort_table"
  )
}
```

```

)

cohorts <- cohortSet %>%
  # Remove 'cohort' and 'json' columns
  select(-"cohort", -"json") %>%
  mutate(type = c("event", "event", "event", "event", "exit", "event", "event", "target")) %>%
  rename(
    cohortId = "cohort_definition_id",
    cohortName = "cohort_name",
  ) %>%
  select("cohortId", "cohortName", "type")

executeTreatmentPatterns(
  cohorts = cohorts,
  cohortTableName = "cohort_table",
  cdm = cdm,
  outputPath = tempdir()
)

DBI::dbDisconnect(con, shutdown = TRUE)
}

```

 export

export

Description

Export andromeda generated by [computePathways](#) object to sharable csv-files and/or a zip archive.

Usage

```

export(
  andromeda,
  outputPath,
  ageWindow = 10,
  minCellCount = 5,
  censorType = "minCellCount",
  archiveName = NULL
)

```

Arguments

andromeda (Andromeda::andromeda()) Andromeda object.
 outputPath (character(1))

ageWindow	(integer(n): 10) Number of years to bin age groups into. It may also be a vector of integers. I.e. <code>c(0, 18, 150)</code> which will result in age group 0-18 which includes subjects < 19. And age group 18-150 which includes subjects > 18.
minCellCount	(integer(1): 5) Minimum count required per pathway. Censors data below <code>x</code> as <code><x</code> . This minimum value will carry over to the sankey diagram and sunburst plot.
sensorType	(character(1)) "minCellCount" Censors pathways <code><minCellCount</code> to <code>minCellCount</code> . "remove" Censors pathways <code><minCellCount</code> by removing them completely. "mean" Censors pathways <code><minCellCount</code> to the mean of all frequencies below <code>minCellCount</code>
archiveName	(character(1): NULL) If not NULL adds the exported files to a ZIP-file with the specified archive name.

Value

(invisible(NULL))

Examples

```
library(TreatmentPatterns)
library(CDMConnector)
library(dplyr)

if (require("CirceR", character.only = TRUE, quietly = TRUE)) {
  withr::local_envvar(
    EUNOMIA_DATA_FOLDER = Sys.getenv("EUNOMIA_DATA_FOLDER", unset = tempfile())
  )

  downloadEunomiaData(overwrite = TRUE)

  con <- DBI::dbConnect(duckdb::duckdb(), dbdir = eunomia_dir())
  cdm <- cdmFromCon(con, cdmSchema = "main", writeSchema = "main")

  cohortSet <- readCohortSet(
    path = system.file(package = "TreatmentPatterns", "exampleCohorts")
  )

  cdm <- generateCohortSet(
    cdm = cdm,
    cohortSet = cohortSet,
    name = "cohort_table"
  )

  cohorts <- cohortSet %>%
    # Remove 'cohort' and 'json' columns
    select(-"cohort", -"json") %>%
```

```

mutate(type = c("event", "event", "event", "event", "exit", "event", "event", "target")) %>%
  rename(
    cohortId = "cohort_definition_id",
    cohortName = "cohort_name",
  ) %>%
  select("cohortId", "cohortName", "type")

outputEnv <- computePathways(
  cohorts = cohorts,
  cohortTableName = "cohort_table",
  cdm = cdm
)

export(
  andromeda = outputEnv,
  outputPath = tempdir()
)

Andromeda::close(outputEnv)
DBI::dbDisconnect(con, shutdown = TRUE)
}

```

InputHandler

InputHandler

Description

Class to handle input from the user. Supports direct paths or input fields through `setDataPath()`.

Super class

[TreatmentPatterns::ShinyModule](#) -> InputHandler

Active bindings

`reactiveValues` (`reactiveValues`)
`reactiveValues` class created by [reactiveValues](#).

Methods

Public methods:

- [InputHandler\\$uiMenu\(\)](#)
- [InputHandler\\$uiBody\(\)](#)
- [InputHandler\\$server\(\)](#)
- [InputHandler\\$uiDatabaseSelector\(\)](#)

- [InputHandler\\$setDataPath\(\)](#)
- [InputHandler\\$clone\(\)](#)

Method uiMenu(): Method to include a [menuItem](#) to link to the body.

Usage:

```
InputHandler$uiMenu(label = "File upload", tag = "fileUpload")
```

Arguments:

label (character(1))

Label to show for the menuItem.

tag (character(1))

Tag to use internally in input.

Returns: (menuItem)

Method uiBody(): Method to include a [tabItem](#) to include the body.

Usage:

```
InputHandler$uiBody()
```

Returns: (tabItem)

Method server(): Method to handle the back-end.

Usage:

```
InputHandler$server(input, output, session)
```

Arguments:

input (input)

Input from the server function.

output (output)

Output from the server function.

session (session)

Session from the server function.

Returns: (NULL)

Method uiDatabaseSelector(): Method to include a [uiOutput](#) to select between multiple uploaded files.

Usage:

```
InputHandler$uiDatabaseSelector()
```

Returns: (uiOutput)

Method setDataPath(): Method to dictate where the data is coming from, either from the input through the shiny application, or from a specified path. When one is provided, the other is ignored.

Usage:

```
InputHandler$setDataPath(tag = "uploadField", input = NULL, path = NULL)
```

Arguments:

tag (character(1))

Tag to use internally in input.

input (input)
 Input from the server function of the shiny app.

path (character(1))
 Path to a zip-file containing TreatmentPatterns output files.

Returns: (invisible(self))

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

Usage:
 InputHandler\$clone(deep = FALSE)

Arguments:
 deep Whether to make a deep clone.

launchResultsExplorer *launchResultsExplorer*

Description

Launches the ResultExplorer shinyApp.

Usage

```
launchResultsExplorer()
```

Value

(shinyApp)

Examples

```
if (interactive()) {
  launchResultsExplorer()
}
```

SankeyDiagram *SankeyDiagram*

Description

Class to handle the Sankey diagram of TreatmentPatterns.

Super classes

[TreatmentPatterns::ShinyModule](#) -> TreatmentPatterns::InteractivePlot -> SankeyDiagram

Methods**Public methods:**

- [SankeyDiagram\\$clone\(\)](#)

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

Usage:

`SankeyDiagram$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

ShinyModule

ShinyModule

Description

ShinyModule super class

Active bindings

`namespace` Namespace of the module.

Methods**Public methods:**

- [ShinyModule\\$new\(\)](#)
- [ShinyModule\\$validate\(\)](#)
- [ShinyModule\\$uiMenu\(\)](#)
- [ShinyModule\\$uiBody\(\)](#)
- [ShinyModule\\$server\(\)](#)
- [ShinyModule\\$clone\(\)](#)

Method `new()`: Initializer method

Usage:

`ShinyModule$new(namespace)`

Arguments:

`namespace` (character(1))

Returns: (invisible(self))

Method `validate()`: Validator method

Usage:

`ShinyModule$validate()`

Returns: (invisible(self))

Method `uiMenu()`: Method to include a [menuItem](#) to link to the body.

Usage:

```
ShinyModule$uiMenu(label, tag)
```

Arguments:

`label` (character(1))

Label to show for the menuItem.

`tag` (character(1))

Tag to use internally in input.

Returns: (menuItem)

Method `uiBody()`: Method to include a [tabItem](#) to include the body.

Usage:

```
ShinyModule$uiBody()
```

Returns: (tabItem)

Method `server()`: Method to handle the back-end.

Usage:

```
ShinyModule$server(input, output, session)
```

Arguments:

`input` (input)

Input from the server function.

`output` (output)

Output from the server function.

`session` (session)

Session from the server function.

Returns: (NULL)

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

SunburstPlot

SunburstPlot

Description

Class to handle the Sunburst plot of TreatmentPatterns.

Super classes

[TreatmentPatterns::ShinyModule](#) -> [TreatmentPatterns::InteractivePlot](#) -> [SunburstPlot](#)

Methods**Public methods:**

- [SunburstPlot\\$clone\(\)](#)

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Index

andromeda, [5](#)

CharacterizationPlots, [2](#)

computePathways, [4](#), [11](#), [13](#)

createSankeyDiagram, [6](#)

createSankeyDiagram2, [8](#)

createSunburstPlot, [9](#)

createSunburstPlot2, [10](#)

executeTreatmentPatterns, [11](#)

export, [11](#), [13](#)

InputHandler, [3](#), [15](#)

launchResultsExplorer, [17](#)

menuItem, [2](#), [16](#), [19](#)

reactiveValues, [15](#)

SankeyDiagram, [17](#)

sankeyNetwork, [7](#), [8](#)

ShinyModule, [18](#)

sunburst, [9](#), [10](#)

SunburstPlot, [19](#)

tabItem, [3](#), [16](#), [19](#)

TreatmentPatterns::ShinyModule, [2](#), [15](#),
[17](#), [19](#)

uiOutput, [16](#)