Package ‘TreatmentPatterns’

November 27, 2024

Type Package

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

Version 2.7.0

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>.

URL https://github.com/darwin-eu/TreatmentPatterns

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

Language en-US

Depends R (>= 4.2)

Imports checkmate, dplyr, stringr, utils, stats, Andromeda, tidyr, R6, sunburstR, networkD3, htmlwidgets, shiny, shinydashboard, ggplot2

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

License Apache License (>= 2)

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'

'attrition.R' 'computePathways.R' 'constructPathways.R'
'createSankeyDiagram.R' 'createSunburstPlot.R'
'executeTreatmentPatterns.R' 'export.R' 'plotEventDuration.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 2024-11-27 11:50:02 UTC

Contents

`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 *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,
  tempEmulationSchema = NULL,
  includeTreatments = "startDate",
  indexDateOffset = 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.
`indexDateOffset`	(`integer(1)`: `0`) Offset the index date of the `Target` cohort.
`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

```
ableToRun <- all(
  require("CirceR", character.only = TRUE, quietly = TRUE),
  require("CDMConnector", character.only = TRUE, quietly = TRUE),
  require("TreatmentPatterns", character.only = TRUE, quietly = TRUE),
  require("dplyr", character.only = TRUE, quietly = TRUE)
)

if (ableToRun) {
  library(TreatmentPatterns)
  library(CDMConnector)
  library(dplyr)

  withr::local_envvar(
    R_USER_CACHE_DIR = tempfile(),
    EUNOMIA_DATA_FOLDER = Sys.getenv("EUNOMIA_DATA_FOLDER", unset = tempfile())
  )

  tryCatch({
    if (Sys.getenv("skip_eunomia_download_test") != "TRUE") {
      CDMConnector::downloadEunomiaData(overwrite = TRUE)
    }
  }, error = function(e) NA)

  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)
}
```

`createSankeyDiagram` *createSankeyDiagram*

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)
```

---

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)
```

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

```
ableToRun <- all(
  require("CirceR", character.only = TRUE, quietly = TRUE),
  require("CDMConnector", character.only = TRUE, quietly = TRUE),
  require("TreatmentPatterns", character.only = TRUE, quietly = TRUE),
  require("dplyr", character.only = TRUE, quietly = TRUE)
)

if (require("CirceR", character.only = TRUE, quietly = TRUE)) {
  library(TreatmentPatterns)
  library(CDMConnector)
  library(dplyr)

  withr::local_envvar(
    R_USER_CACHE_DIR = tempfile(),
    EUNOMIA_DATA_FOLDER = Sys.getenv("EUNOMIA_DATA_FOLDER", unset = tempfile())
  )

  tryCatch({
    if (Sys.getenv("skip_eunomia_download_test") != "TRUE") {
      CDMConnector::downloadEunomiaData(overwrite = TRUE)
    }
  },
    error = function(e) NA)

  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. `c(0, 18, 150)` which will results 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 `x` as `<x`. This minimum value will carry over to the sankey diagram and sunburst plot.
`censorType`	(`character(1)`) `"minCellCount"` Censors pathways <`minCellCount` to `minCellCount`. `"remove"` Censors pathways <`minCellCount` by removing them completely. `"mean"` Censors pathways <`minCellCount` to the mean of all frequencies below `minCellCount`
`archiveName`	(`character(1)`: `NULL`) If not `NULL` adds the exported files to a ZIP-file with the specified archive name.

Value

`(invisible(NULL))`

Examples

```
ableToRun <- all(
  require("CirceR", character.only = TRUE, quietly = TRUE),
  require("CDMConnector", character.only = TRUE, quietly = TRUE),
  require("TreatmentPatterns", character.only = TRUE, quietly = TRUE),
  require("dplyr", character.only = TRUE, quietly = TRUE)
)

if (ableToRun) {
  library(TreatmentPatterns)
  library(CDMConnector)
  library(dplyr)

  withr::local_envvar(
    R_USER_CACHE_DIR = tempfile(),
    EUNOMIA_DATA_FOLDER = Sys.getenv("EUNOMIA_DATA_FOLDER", unset = tempfile())
  )

  tryCatch({
    if (Sys.getenv("skip_eunomia_download_test") != "TRUE") {
      CDMConnector::downloadEunomiaData(overwrite = TRUE)
    }
  }, error = function(e) NA)

  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()
}
```

`plotEventDuration` *plotEventDuration*

Description

plotEventDuration

Usage

```
plotEventDuration(
  eventDurations,
  minCellCount = 0,
  treatmentGroups = "both",
  eventLines = NULL,
  includeOverall = TRUE
)
```

Arguments

`eventDurations`	(`data.frame`) Contents of `summaryEventDuration.csv` file.
`minCellCount`	(`numeric(1)`: `0`) Min Cell Count per event group.
`treatmentGroups`	(`character(1)`: `"both"`) `"group"`: Only mono-, and combination-events. `"individual"`: Only individual (combination) events. `"both"`: Both mono-, and combination-events, and individual (combination) events.
`eventLines`	(`numeric(n)`: `NULL`) Event lines to include, i.e. `c(1, 2, 3)` includes first (1), second (2), and third (3) lines of events. `NULL` will include all `eventLines`.
`includeOverall`	(`logical(1)`: `TRUE`) `TRUE`: Include an overall column with the `eventLines`. `FALSE`: Exclude the overall column.

Value

ggplot

Examples

```
ableToRun <- all(
  require("CirceR", character.only = TRUE, quietly = TRUE),
  require("CDMConnector", character.only = TRUE, quietly = TRUE),
  require("TreatmentPatterns", character.only = TRUE, quietly = TRUE),
  require("dplyr", character.only = TRUE, quietly = TRUE)
)
```

```
if (ableToRun) {
  withr::local_envvar(
    R_USER_CACHE_DIR = tempfile(),
    EUNOMIA_DATA_FOLDER = Sys.getenv("EUNOMIA_DATA_FOLDER", unset = tempfile())
  )

  tryCatch({
    if (Sys.getenv("skip_eunomia_download_test") != "TRUE") {
      CDMConnector::downloadEunomiaData(overwrite = TRUE)
    }
  }, error = function(e) NA)

  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()
  )

  eventDurations <- read.csv(file.path(tempdir(), "summaryEventDuration.csv"))

  plotEventDuration(
    eventDurations = eventDurations,
    minCellCount = 5,
    treatmentGroups = "group",
    eventLines = 1:4,
    includeOverall = FALSE
  )

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

`SankeyDiagram` *SankeyDiagram*

Description

Class to handle the Sankey diagram of TreatmentPatterns.

Super classes

`TreatmentPatterns::ShinyModule` -> `TreatmentPatterns::InteracitvePlot` -> `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::InteracitvePlot` -> `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