Package 'TreatmentPatterns'

November 13, 2023

Type Package

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

Version 2.6.0

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-dev/TreatmentPatterns

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

Depends R (>= 4.0)

Imports checkmate, dplyr, stringr, stringi, utils, rjson, googleVis, stats, Andromeda, tidyr, R6, sunburstR, networkD3, htmlwidgets

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

License Apache License 2.0

Encoding UTF-8

RoxygenNote 7.2.3

VignetteBuilder knitr

Config/testthat/edition 3

Config/testthat/parallel true

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

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 2023-11-13 11:13:26 UTC

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$fetchCohortTable()
- CDMInterface$fetchMetadata()
- CDMInterface$destroy()
- CDMInterface$clone()

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

Usage:

```
CDMInterface$new(
  connectionDetails = NULL,
  cdmSchema = NULL,
  resultSchema = NULL,
  tempEmulationSchema = 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`.

`tempEmulationSchema` Schema used to emulate temp tables.

`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(
  cohorts,
  cohortTableName,
  andromeda,
  andromedaTableName,
  minEraDuration = NULL
)
```

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.

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

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

`andromedaTableName (character(1))`
: Name of the table in the Andromeda object where the data will be loaded.

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

Returns: `(data.frame)`

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

Usage:

```
CDMInterface$fetchMetadata(andromeda)
```

Arguments:

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

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

Returns: `(invisible(NULL))`

Method `destroy()`**:** Destroys instance

Usage:

```
CDMInterface$destroy()
```

Returns: (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,
  tempEmulationSchema = 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.
tempEmulationSchema	Schema used to emulate temp tables
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.

Examples

```
ableToRun <- invisible(all(
  require("Eunomia", character.only = TRUE),
  require("CirceR", character.only = TRUE),
  require("CohortGenerator", character.only = TRUE),
  require("dplyr", character.only = TRUE)
))

if (ableToRun) {
  # CohortGenerator example
  connectionDetails <- Eunomia::getEunomiaConnectionDetails()
  cdmDatabaseSchema <- "main"
  resultSchema <- "main"
  cohortTable <- "CohortTable"

  cohortsToCreate <- CohortGenerator::createEmptyCohortDefinitionSet()

  cohortJsonFiles <- list.files(
```

```
  system.file(
    package = "TreatmentPatterns",
    "exampleCohorts"),
    full.names = TRUE)

for (i in seq_len(length(cohortJsonFiles))) {
  cohortJsonFileName <- cohortJsonFiles[i]
  cohortName <- tools::file_path_sans_ext(basename(cohortJsonFileName))
  cohortJson <- readChar(cohortJsonFileName, file.info(
    cohortJsonFileName)$size)

  cohortExpression <- CirceR::cohortExpressionFromJson(cohortJson)

  cohortSql <- CirceR::buildCohortQuery(
    cohortExpression,
    options = CirceR::createGenerateOptions(generateStats = FALSE))

  cohortsToCreate <- rbind(
    cohortsToCreate,
    data.frame(
      cohortId = i,
      cohortName = cohortName,
      sql = cohortSql,
      stringsAsFactors = FALSE))
}

cohortTableNames <- CohortGenerator::getCohortTableNames(
  cohortTable = cohortTable)

CohortGenerator::createCohortTables(
  connectionDetails = connectionDetails,
  cohortDatabaseSchema = resultSchema,
  cohortTableNames = cohortTableNames)

# Generate the cohorts
cohortsGenerated <- CohortGenerator::generateCohortSet(
  connectionDetails = connectionDetails,
  cdmDatabaseSchema = cdmDatabaseSchema,
  cohortDatabaseSchema = resultSchema,
  cohortTableNames = cohortTableNames,
  cohortDefinitionSet = cohortsToCreate)

# Select Viral Sinusitis
targetCohorts <- cohortsGenerated %>%
  filter(cohortName == "ViralSinusitis") %>%
  select(cohortId, cohortName)

# Select everything BUT Viral Sinusitis cohorts
eventCohorts <- cohortsGenerated %>%
  filter(cohortName != "ViralSinusitis" & cohortName != "Death") %>%
  select(cohortId, cohortName)

exitCohorts <- cohortsGenerated %>%
```

```
  filter(cohortName == "Death") %>%
  select(cohortId, cohortName)

cohorts <- dplyr::bind_rows(
  targetCohorts %>% mutate(type = "target"),
  eventCohorts %>% mutate(type = "event"),
  exitCohorts %>% mutate(type = "exit")
)

computePathways(
  cohorts = cohorts,
  cohortTableName = cohortTable,
  connectionDetails = connectionDetails,
  cdmSchema = cdmDatabaseSchema,
  resultSchema = resultSchema
)
}
```

`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,
  returnHTML = FALSE,
  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.
returnHTML	(logical(1): FALSE) Logical to return HTML or not. TRUE Returns HTML as character(n), does not requie outputPath to be specified. FALSE Returns NULL, but writes the HTML to the specified file instead. Requires outputPath to be specified.
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

```
# treatmentPathways <- read.csv(treatmentPathways.csv)

# 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)
)

outputFile <- tempfile(pattern = "mySankeyDiagram", fileext = "html")

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

`createSankeyDiagram2` *createSankeyDiagram2*

Description

Create sankey diagram, will replace `createSankeyDiagram`.

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.

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)
)

createSankeyDiagram2(treatmentPathways)
```

---

createSunburstPlot *createSunburstPlot*

Description

Generate a sunburst plot from the treatment pathways.

Usage

```
createSunburstPlot(
  treatmentPathways,
  outputFile,
  groupCombinations = FALSE,
  returnHTML = FALSE
)
```

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.
returnHTML	(logical(1): FALSE) Logical to return HTML or not.

TRUE Returns HTML as character(n), does not requie outputPath to be specified.

FALSE Returns NULL, but writes the HTML to the specified file instead. Requires outputPath to be specified.

Value

(NULL)

Examples

```
# treatmentPathways <- read.csv("treatmentPathways.csv")

# 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)
)

outputFile <- tempfile(pattern = "mySunburstPlot", fileext = "html")

createSunburstPlot(
  treatmentPatwhays,
  outputFile
)
```

---

createSunburstPlot2 *createSunburstPlot2*

Description

New sunburstPlot function

Usage

```
createSunburstPlot2(
  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 colors codes.

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)
```

---

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()) personId
2. (numeric()) indexYear
3. (numeric()) eventCohortId
4. (as.Date()) eventStartDate
5. (as.Date()) eventEndDate
6. (character()) type
7. (difftime()) durationEra
8. (difftime()) gapSame

---

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))

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,
  tempEmulationSchema = 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`.

`tempEmulationSchema`
(`character(1)`) Schema to emulate temp tables.

`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

```
# Using CDMConnector
ableToRun <- all(c(
  require("CDMConnector", character.only = TRUE),
  require("DBI", character.only = TRUE),
  require("duckdb", character.only = TRUE),
  require("dplyr", character.only = TRUE)
))
```

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

  CDMConnector::downloadEunomiaData()

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

  cdm$CohortTable <- dplyr::union_all(
  # Viral Sinusitis
  cdm$condition_occurrence %>%
    filter(.data$condition_concept_id == 40481087) %>%
    inner_join(cdm$person, by = join_by(person_id == person_id)) %>%
    select("person_id", "condition_start_date", "condition_end_date") %>%
    mutate(cohort_definition_id = 1) %>%
    rename(
      cohort_start_date = "condition_start_date",
      cohort_end_date = "condition_end_date"
    ),

  # Aspirin
  cdm$drug_era %>%
    filter(.data$drug_concept_id == 1112807) %>%
    inner_join(cdm$person, by = join_by(person_id == person_id)) %>%
    select("person_id", "drug_era_start_date", "drug_era_end_date") %>%
    mutate(cohort_definition_id = 2) %>%
    rename(
      cohort_start_date = "drug_era_start_date",
      cohort_end_date = "drug_era_end_date"
    ),

  # Acetaminophen
  cdm$drug_era %>%
    filter(.data$drug_concept_id == 1125315) %>%
    inner_join(cdm$person, by = join_by(person_id == person_id)) %>%
    select("person_id", "drug_era_start_date", "drug_era_end_date") %>%
    mutate(cohort_definition_id = 3) %>%
    rename(
      cohort_start_date = "drug_era_start_date",
      cohort_end_date = "drug_era_end_date"
    ),

  # Clavunate
  cdm$drug_era %>%
    filter(.data$drug_concept_id == 1759842) %>%
    inner_join(cdm$person, by = join_by(person_id == person_id)) %>%
    select("person_id", "drug_era_start_date", "drug_era_end_date") %>%
    mutate(cohort_definition_id = 4) %>%
    rename(
      cohort_start_date = "drug_era_start_date",
      cohort_end_date = "drug_era_end_date"
```

```
    ),

  # Death
  cdm$observation %>%
    filter(.data$observation_concept_id == 4306655) %>%
    inner_join(cdm$person, by = join_by(person_id == person_id)) %>%
    select("person_id", "observation_date") %>%
    mutate(
      cohort_definition_id = 5,
      cohort_end_date = observation_date) %>%
    rename(
      cohort_start_date = "observation_date"
    )
  )

  cdm$CohortTable <- cdm$CohortTable %>%
    rename(
      SUBJECT_ID = "person_id",
      COHORT_START_DATE = "cohort_start_date",
      COHORT_END_DATE = "cohort_end_date",
      COHORT_DEFINITION_ID = "cohort_definition_id"
    )

  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")
  )

  try(
    executeTreatmentPatterns(
      cohorts,
      cohortTableName,
      tempdir(),
      cdm
    )
  )

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

# Using DatabaseConnector
ableToRun <- invisible(all(
  require("Eunomia", character.only = TRUE),
  require("CirceR", character.only = TRUE),
  require("CohortGenerator", character.only = TRUE),
  require("dplyr", character.only = TRUE)
))

if (ableToRun) {
  # CohortGenerator example
```

```
connectionDetails <- Eunomia::getEunomiaConnectionDetails()
cdmDatabaseSchema <- "main"
resultSchema <- "main"
cohortTable <- "CohortTable"

cohortsToCreate <- CohortGenerator::createEmptyCohortDefinitionSet()

cohortJsonFiles <- list.files(
  system.file(
    package = "TreatmentPatterns",
    "exampleCohorts"),
    full.names = TRUE)

for (i in seq_len(length(cohortJsonFiles))) {
  cohortJsonFileName <- cohortJsonFiles[i]
  cohortName <- tools::file_path_sans_ext(basename(cohortJsonFileName))
  cohortJson <- readChar(cohortJsonFileName, file.info(
    cohortJsonFileName)$size)

  cohortExpression <- CirceR::cohortExpressionFromJson(cohortJson)

  cohortSql <- CirceR::buildCohortQuery(
    cohortExpression,
    options = CirceR::createGenerateOptions(generateStats = FALSE))

  cohortsToCreate <- rbind(
    cohortsToCreate,
    data.frame(
      cohortId = i,
      cohortName = cohortName,
      sql = cohortSql,
      stringsAsFactors = FALSE))
}

cohortTableNames <- CohortGenerator::getCohortTableNames(
  cohortTable = cohortTable)

CohortGenerator::createCohortTables(
  connectionDetails = connectionDetails,
  cohortDatabaseSchema = resultSchema,
  cohortTableNames = cohortTableNames)

# Generate the cohorts
cohortsGenerated <- CohortGenerator::generateCohortSet(
  connectionDetails = connectionDetails,
  cdmDatabaseSchema = cdmDatabaseSchema,
  cohortDatabaseSchema = resultSchema,
  cohortTableNames = cohortTableNames,
  cohortDefinitionSet = cohortsToCreate)

# Select Viral Sinusitis
targetCohorts <- cohortsGenerated %>%
  filter(cohortName == "ViralSinusitis") %>%
```

```
    select(cohortId, cohortName)

  # Select everything BUT Viral Sinusitis cohorts
  eventCohorts <- cohortsGenerated %>%
    filter(cohortName != "ViralSinusitis" & cohortName != "Death") %>%
    select(cohortId, cohortName)

  exitCohorts <- cohortsGenerated %>%
    filter(cohortName == "Death") %>%
    select(cohortId, cohortName)

  cohorts <- dplyr::bind_rows(
    targetCohorts %>% mutate(type = "target"),
    eventCohorts %>% mutate(type = "event"),
    exitCohorts %>% mutate(type = "exit")
  )

  try(
    executeTreatmentPatterns(
      cohorts = cohorts,
      cohortTableName = cohortTable,
      outputPath = tempdir(),
      connectionDetails = connectionDetails,
      cdmSchema = cdmDatabaseSchema,
      resultSchema = 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

```
ableToRun <- invisible(all(
  require("Eunomia", character.only = TRUE),
  require("CirceR", character.only = TRUE),
  require("CohortGenerator", character.only = TRUE),
  require("dplyr", character.only = TRUE)
))

if (ableToRun) {
  # CohortGenerator example
  connectionDetails <- Eunomia::getEunomiaConnectionDetails()
  cdmDatabaseSchema <- "main"
  resultSchema <- "main"
  cohortTable <- "CohortTable"

  cohortsToCreate <- CohortGenerator::createEmptyCohortDefinitionSet()

  cohortJsonFiles <- list.files(
    system.file(
      package = "TreatmentPatterns",
      "exampleCohorts"),
      full.names = TRUE)

  for (i in seq_len(length(cohortJsonFiles))) {
    cohortJsonFileName <- cohortJsonFiles[i]
    cohortName <- tools::file_path_sans_ext(basename(cohortJsonFileName))
    cohortJson <- readChar(cohortJsonFileName, file.info(
      cohortJsonFileName)$size)

    cohortExpression <- CirceR::cohortExpressionFromJson(cohortJson)

    cohortSql <- CirceR::buildCohortQuery(
      cohortExpression,
      options = CirceR::createGenerateOptions(generateStats = FALSE))

    cohortsToCreate <- rbind(
      cohortsToCreate,
      data.frame(
        cohortId = i,
        cohortName = cohortName,
```

```
      sql = cohortSql,
      stringsAsFactors = FALSE))
}

cohortTableNames <- CohortGenerator::getCohortTableNames(
  cohortTable = cohortTable)

CohortGenerator::createCohortTables(
  connectionDetails = connectionDetails,
  cohortDatabaseSchema = resultSchema,
  cohortTableNames = cohortTableNames)

# Generate the cohorts
cohortsGenerated <- CohortGenerator::generateCohortSet(
  connectionDetails = connectionDetails,
  cdmDatabaseSchema = cdmDatabaseSchema,
  cohortDatabaseSchema = resultSchema,
  cohortTableNames = cohortTableNames,
  cohortDefinitionSet = cohortsToCreate)

# Select Viral Sinusitis
targetCohorts <- cohortsGenerated %>%
  filter(cohortName == "ViralSinusitis") %>%
  select(cohortId, cohortName)

# Select everything BUT Viral Sinusitis cohorts
eventCohorts <- cohortsGenerated %>%
  filter(cohortName != "ViralSinusitis" & cohortName != "Death") %>%
  select(cohortId, cohortName)

exitCohorts <- cohortsGenerated %>%
  filter(cohortName == "Death") %>%
  select(cohortId, cohortName)

cohorts <- dplyr::bind_rows(
  targetCohorts %>% mutate(type = "target"),
  eventCohorts %>% mutate(type = "event"),
  exitCohorts %>% mutate(type = "exit")
)

andromeda <- computePathways(
  cohorts = cohorts,
  cohortTableName = cohortTable,
  connectionDetails = connectionDetails,
  cdmSchema = cdmDatabaseSchema,
  resultSchema = resultSchema
)

try(
  TreatmentPatterns::export(
    andromeda = andromeda,
    outputPath = tempdir(),
    ageWindow = 2,
```

```
      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(minEraDuration)
```

Arguments:

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

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

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

`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

---

toFile *toFile*

Description

toFile

Usage

```
toFile(json, treatmentPathways, outputFile)
```

Arguments

json	(character(1))
treatmentPathways	(data.frame())
outputFile	(character(1))

Value

(NULL)

toList *toList*

Description

toList

Usage

```
toList(json)
```

Arguments

json	(character(1))

Value

(list())

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