Package 'CohortPathways'

March 24, 2025

Type Package

Title Create Pathways from Target to Event Cohorts

Version 0.0.1

Date 2024-10-20

Maintainer Gowtham Rao <rao@ohdsi.org>

Description Software tool designed to compute the temporal relationship defined as pathways between any two instantiated cohorts. The cohorts are input as Target and event cohorts.

Depends DatabaseConnector (>= 5.0.0),
R (>= 4.1.0)

Imports checkmate,
dplyr,
ggplot2,
hrbrthemes,
lifecycle,
rlang,
SqlRender,
tidyr,
viridis

Suggests remotes,
testthat,
withr

License Apache License

RoxygenNote 7.3.2

Roxygen list(markdown = TRUE)

Encoding UTF-8

Language en-US

URL https://github.com/OHDSI/CohortPathways

BugReports https://github.com/OHDSI/CohortPathways/issues

Contents

`calculateSummaryStatistics`

Calculate Summary Statistics by Group

Description

This function computes a variety of summary statistics for a specified numeric column in a data frame. It calculates the mean, standard deviation, median, several quantiles, the mode, the total count, and the count of distinct values. The final output is returned in a long format for easier inspection and downstream processing.

Usage

```
calculateSummaryStatistics(df, value = "value")
```

Arguments

`df`	A data frame containing the data for which the summary statistics will be computed.
`value`	A character string specifying the name of the numeric column to summarize. Defaults to `"value"`.

Details

A helper function `calculateMode` is defined to determine the mode of the numeric column, i.e., the most frequently occurring value.

The data is grouped by the specified `group` column, and summary statistics are computed for the `value` column including:

- `mean`: The average of the values.
- `sd`: The standard deviation of the values.
- `median`: The median value.
- `p01`: The 1st percentile.
- `p05`: The 5th percentile.
- `p25`: The 25th percentile.
- `p75`: The 75th percentile.
- `p95`: The 95th percentile.
- `p99`: The 99th percentile.
- `mode`: The most frequently occurring value.
- `count`: The number of observations.
- `count_distinct`: The number of distinct values.

Finally, the summary table is reshaped into a long format using `tidyr::pivot_longer`, so that each row corresponds to a single statistic for each group.

Value

A tibble (data frame) in long format with two columns:

`statistic`	The name of the summary statistic (e.g., mean, sd, median, quantiles, mode, count, count_distinct).
`value`	The value of the corresponding summary statistic.

Examples

```
## Not run:
# Create example data
df <- data.frame(
  value = rnorm(200)
)

summary_stats <- calculateSummaryStatistics(df, value = "value")
print(summary_stats)

## End(Not run)
```

`createIfNotExist` *Create a Folder if It Does Not Exist*

Description

This function creates a folder (directory) at a specified location if it does not already exist. It also verifies that the folder exists and is accessible by asserting its execute permissions. Any issues or error messages can be collected using an AssertCollection from the `checkmate` package.

Usage

```
createIfNotExist(type, name, recursive = TRUE, errorMessage = NULL)
```

Arguments

`type`	A character string specifying the type of entity to create. Currently, only `"folder"` is supported.
`name`	A character string specifying the name or path where the folder should be created.
`recursive`	A logical value indicating whether to create the directory recursively (i.e., creating any necessary parent directories). Defaults to `TRUE`.
`errorMessage`	An optional AssertCollection object (from the `checkmate` package) used to collect error messages. If `NULL` or not a valid AssertCollection, a new one is created.

Details

The function first ensures that an appropriate AssertCollection object is available. If `type` is not `NULL`, it checks if the `name` parameter is non-empty. If the `type` is `"folder"`, the function removes any trailing slashes from `name` and checks if the directory exists. If the directory does not exist, it is created using `dir.create` with the specified `recursive` setting. A message is printed to indicate the creation of the folder. Finally, it asserts that the directory exists and is executable.

Value

Invisibly returns the AssertCollection object containing any collected error messages.

Examples

```
## Not run:
# Create a folder "data" if it does not already exist
createIfNotExist("folder", "data")

## End(Not run)
```

`createViolinPlot` *Create an Enhanced Violin Plot*

Description

This function generates a violin plot enhanced with overlaid boxplots and symbols for mean and median values. It uses ggplot2 for plotting, viridis for color palettes, and hrbrthemes for themes.

Usage

```
createViolinPlot(
  data,
  xName = "groupLabel",
  yName = "value",
  fillName = NULL,
  title = "Distribution Plot",
  colorMean = "black",
  colorMedian = "black",
  sizeMean = 3,
  sizeMedian = 3,
  shapeMean = 18,
  shapeMedian = 16
)
```

Arguments

`data`	A data frame containing the variables to be plotted.
`xName`	The name of the categorical variable (as a string) to plot on the x-axis.
`yName`	The name of the numeric variable (as a string) to plot on the y-axis.
`fillName`	(Optional) The name of the variable (as a string) to use for fill colors. Default is NULL.
`title`	The title of the plot. Default is "Distribution Plot".
`colorMean`	The color to use for the mean symbol. Default is "blue".
`colorMedian`	The color to use for the median symbol. Default is "green".
`sizeMean`	The size of the mean symbol. Default is 3.
`sizeMedian`	The size of the median symbol. Default is 3.
`shapeMean`	The shape of the mean symbol. Default is 18.
`shapeMedian`	The shape of the median symbol. Default is 16.

Value

A ggplot object representing the violin plot.

Examples

```
createViolinPlot(
  data = myData, xName = "group", yName = "value", fillName = "category",
  title = "My Violin Plot", colorMean = "red", colorMedian = "darkgreen",
  sizeMean = 4, sizeMedian = 4
)
```

---

executeCohortPathways *Execute cohort pathway analysis.*

Description

This function reproduces the logic behind "Cohort Pathways" (sometimes referred to as "Treatment Pathways") as defined in ATLAS. A *pathway* is the sequence of events that occur after a cohort entry, often used to compare or visualize patients' treatment patterns. The function takes a set of target cohorts (the index cohorts) and event cohorts (the "steps" in the pathway) and computes the frequency and order in which these events occur.

Usage

```
executeCohortPathways(
  connectionDetails = NULL,
  connection = NULL,
  cohortDatabaseSchema,
  cohortTableName = "cohort",
  tempEmulationSchema = getOption("sqlRenderTempEmulationSchema"),
  targetCohortIds,
  eventCohortIds,
  minCellCount = 5,
  allowRepeats = FALSE,
  maxDepth = 5,
  collapseWindow = 30
)
```

Arguments

`connectionDetails`
: An object of type `createConnectionDetails` (from the **DatabaseConnector** package) providing the details to connect to the database. If `NULL`, then `connection` must be provided.

`connection`
: An existing database connection (as created by `connect`). If `NULL`, a new connection will be established using `connectionDetails` at the start of the function, and closed upon completion.

`cohortDatabaseSchema`
: Schema where the target and event cohorts are stored. For SQL Server, this should include both the database and schema name (e.g., "mydatabase.dbo").

`cohortTableName`
: Name of the table containing your cohorts (default = "cohort").

`tempEmulationSchema`
: For databases without true temp table support (e.g., Oracle), specify a schema where you have write privileges to emulate temp tables.

`targetCohortIds`
: Integer vector of cohort definition IDs representing one or more *target* cohorts (the index populations whose pathways you want to explore).

`eventCohortIds`
: Integer vector of cohort definition IDs representing one or more *event* cohorts (the steps in the pathways, e.g., treatments or clinical events).

`minCellCount`
: The minimum cell count to display (default = 5). Any group or count below this threshold should be redacted or masked if you're following a strict privacy policy.

`allowRepeats`
: Boolean; if `TRUE`, an event cohort can reappear multiple times in the same patient's pathway sequence (default = `FALSE`).

`maxDepth`
: Maximum number of steps in any pathway (default = 5). Pathways longer than this threshold are truncated.

`collapseWindow`
: An integer indicating the number of days within which events are considered the same "step" (default = 30 days). This window effectively groups closely occurring events to reduce spurious variability.

Details

Overview Cohort Pathways are commonly used to visualize how patients move through different treatments or events over time. Within ATLAS, these pathways are depicted as "sunburst" diagrams and flow charts. This function provides a programmatic way to replicate that logic outside of ATLAS, especially useful when you need to run a pathway analysis in a fully scripted R environment.

Key Steps

1. **Identify Cohorts**: You must already have *target* and *event* cohorts instantiated in your results schema (i.e., the cohorts exist in the `cohortTableName` table).
2. **Specify Pathway Parameters**: Options such as `allowRepeats`, `collapseWindow`, and `maxDepth` control how the event sequences are aggregated:
   - `allowRepeats`: If `TRUE`, an event cohort can appear more than once at different time points in the pathway sequence for the same person.
   - `collapseWindow`: Events that occur within this many days of each other (default = 30 days) will be considered part of the same step, reducing the "noise" of very frequent transitions.
   - `maxDepth`: The maximum length of a pathway (default = 5 steps). Longer sequences are truncated to keep the analysis and visualization more interpretable.
3. **Analyze**: The function creates internal temporary tables (or uses your specified `tempEmulationSchema` if necessary) to calculate how many people experience each distinct pathway. It then returns several data frames capturing:
   - `pathwayAnalysisStatsData`: Summary statistics by target cohort, including total counts and the distribution of event occurrences.
   - `pathwaysAnalysisPathsData`: The distinct sequences (paths) observed, with counts of how many persons followed each sequence.
   - `pathwaysAnalysisEventsData`: A more granular breakdown of how events contribute to different pathway steps.

- pathwayAnalysisCodesData and related tables: Helper mappings and indices that translate event cohorts into pathway "codes."

Cell Count Restrictions The argument minCellCount allows you to enforce a minimum cell count, consistent with privacy rules. Any group or count below minCellCount can be masked or handled downstream according to your organization's policies.

Reference

- *The Book of OHDSI* (Chapter: Characterization, Section: Cohort Pathways in ATLAS)
- OHDSI Forums: https://forums.ohdsi.org/t/cohort-pathways-in-atlas-faq/9511/2 https://forums.ohdsi.org/t/reproducing-a-treatment-pathway-study-atlas-json-files/9450

Value

A named list of data frames capturing key pathway results:

pathwayAnalysisStatsData High-level summary statistics about the number of people in each target cohort and the distribution of events.

pathwaysAnalysisPathsData Rows describing each distinct pathway (sequence of events) observed, along with counts of persons following each path.

pathwaysAnalysisEventsData Detailed event-level information, allowing you to see how each event fits into a given path.

pathwayAnalysisCodesData Mapping of event cohort IDs to pathway "codes" for easy interpretation of combinatorial steps.

pathwaycomboIds, pathwayAnalysisCodesLong, isCombo Additional helper tables describing combos of events.

Examples

```
## Not run:
# Example: Execute a pathway analysis for two target cohorts and three event cohorts.

# 1) Create connection details or use an existing connection.
connDetails <- DatabaseConnector::createConnectionDetails(
  dbms = "postgresql",
  server = "myserver/mydb",
  user = "someuser",
  password = "DontSharePasswords"
)

# 2) Run the function
result <- executeCohortPathways(
  connectionDetails = connDetails,
  cohortDatabaseSchema = "my_results_schema",
  cohortTableName = "my_cohort",
  targetCohortIds = c(123, 456), # e.g., 'diabetes' and 'hypertension' cohorts
  eventCohortIds = c(789, 1011, 1213), # e.g., 'metformin', 'insulin', 'ace_inhibitors'
  allowRepeats = TRUE,
  maxDepth = 4,
  collapseWindow = 14
)

# 3) Inspect the returned data frames for pathway summaries and details
```

```
names(result)
head(result$pathwaysAnalysisPathsData)

## End(Not run)
```

`extractBitSum` *Extract Bit Sum*

Description

This function takes a positive integer and returns a numeric vector of bit positions (using 0-indexing) that represent the powers of 2 summing to the input integer. Essentially, it decomposes the number into its binary representation and returns the indices of the bits that are set to 1.

Usage

```
extractBitSum(x)
```

Arguments

`x`	A positive integer. The number to be decomposed into a sum of powers of 2.

Details

The function first computes an upper bound for the exponent needed to represent `x` in binary using `round(log(x, base = 2) + 2)`. It then creates a vector `series` containing powers of 2 from 2^0 up to $2^{lengthVar}$. In the loop, it repeatedly finds the largest power of 2 (from the series) that is less than or equal to the current remainder, subtracts that value, and records the corresponding bit position (adjusted for 0-indexing).

Value

A numeric vector containing the bit positions (0-indexed) corresponding to the powers of 2 that sum up to `x`.

Examples

```
extractBitSum(10)
# Returns: c(3, 1) because 10 = 2^3 + 2^1 (binary 1010)
```

`formatDecimalWithComma` *Format a Number with Commas and Specified Decimal Places*

Description

This function formats a numeric value into a string by inserting commas as thousand separators and formatting the decimal portion to a specified number of digits. It can either round or truncate the decimal part based on the `round` parameter.

Usage

```
formatDecimalWithComma(number, decimalPlaces = 1, round = TRUE)
```

Arguments

`number`	A numeric value to be formatted.
`decimalPlaces`	An integer specifying the number of digits to display after the decimal point. Defaults to `1`.
`round`	A logical value indicating whether the decimal portion should be rounded. If `FALSE`, the decimal part will be truncated instead. Defaults to `TRUE`.

Details

The function separates the number into its integer and decimal components. The integer part is formatted with commas using `formatC`, while the decimal part is either rounded or truncated according to the `decimalPlaces` parameter. The decimal portion is then extracted from the formatted string and concatenated with the formatted integer part to produce the final output.

Value

A character string representing the formatted number with commas as thousand separators and a period separating the integer and decimal parts.

Examples

```
# Example with rounding (default)
formatDecimalWithComma(1234567.8912)
# Might return "1,234,567.9"

# Example with truncation and two decimal places
formatDecimalWithComma(1234567.8912, decimalPlaces = 2, round = FALSE)
# Might return "1,234,567.89"
```

`timeToFeatureCohort` *Calculate Time to Feature Cohort*

Description

This function calculates the time (in days) from the start date of a target cohort to the start date of a feature cohort for subjects in the specified cohorts. It generates a summary table of the time differences as well as a violin plot visualizing the distribution of these differences.

Usage

```
timeToFeatureCohort(
  targetCohortIds,
  featureCohortIds,
  targetCohortTableName,
  featureCohortTableName = targetCohortTableName,
  minDays = NULL,
  maxDays = NULL,
  tempEmulationSchema = getOption("sqlRenderTempEmulationSchema")
)
```

Arguments

`targetCohortIds`	A vector of one or more Cohort Ids corresponding to target cohort(s). These records are collapsed to a unique combination of subject_id and cohort_start_date.
`featureCohortIds`	A vector of one or more Cohort Ids corresponding to feature cohort(s). These records are collapsed to a unique combination of subject_id and cohort_start_date.
`targetCohortTableName`	The name of the target cohort table.
`featureCohortTableName`	The name of the feature cohort table. Default is the same as targetCohortTable-Name.
`minDays`	(Optional) Minimum number of days between the target and feature cohort start dates. If provided, only records with a time difference greater than or equal to minDays are included.
`maxDays`	(Optional) Maximum number of days between the target and feature cohort start dates. If provided, only records with a time difference less than or equal to maxDays are included.
`tempEmulationSchema`	Some database platforms (e.g., Oracle, Impala) do not support temporary tables natively. To emulate temp tables, provide a schema with write privileges where temporary tables can be created. Default is obtained from `getOption("sqlRenderTempEmulationSc`

Details

The function executes a SQL query that:

- Extracts unique subject start dates for the target cohort and assigns a sequence number for each subject.

- Joins the target cohort with the feature cohort to find the first occurrence (minimum start date) in the feature cohort that occurs on or after the target start date.
- Calculates the time difference (in days) between the target and feature cohort start dates.
- Applies optional filtering based on `minDays` and `maxDays`.

The resulting data is then transformed into a long format (expanding counts by the number of subjects) and used to create both a violin plot and a table of summary statistics.

Value

A list with two components:

summaryStatistics A data frame containing summary statistics for the time differences (in days) grouped by the database identifier.

violinPlot A violin plot (created by `createViolinPlot`) that visualizes the distribution of the time differences.

Examples

```
## Not run:
results <- timeToFeatureCohort(
  targetCohortIds = c(1, 2),
  featureCohortIds = c(3),
  targetCohortTableName = "target_cohort",
  featureCohortTableName = "feature_cohort",
  minDays = 30,
  maxDays = 365,
  tempEmulationSchema = "temp_schema"
)

# Access the summary statistics
summaryStats <- results$summaryStatistics

# Display the violin plot
print(results$violinPlot)

## End(Not run)
```

Index