Package ‘Kala’

March 25, 2025

Type Package

Title Tools for Exploring Temporal Relationships Between Cohorts and Their Features

Version 0.0.0

Date 2025-03-24

Description The 'Kala' package provides a streamlined workflow to generate and explore temporal features for cohorts in the OMOP Common Data Model. By leveraging 'FeatureExtraction' and its 'temporalCovariateSettings', 'Kala' produces both tabular and graphical summaries of features across configurable time windows. These outputs allow easy comparisons and time-series analyses for identifying patterns, trends, or changes in clinical features over time.

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

Imports checkmate,
dplyr,
FeatureExtraction,
lifecycle,
ParallelLogger,
readr,
rlang,
stringr,
SqlRender,
tidyr

Suggests Eunomia,
httr,
knitr,
RSQLite,
testthat,
tibble,
withr,
zip

License Apache License

VignetteBuilder knitr

URL https://ohdsi.github.io/Kala, https://github.com/OHDSI/Kala

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

RoxygenNote 7.3.2

Encoding UTF-8

Config/testthat/edition 3

Contents

commaSeparatedStringToIntArray

Convert a Comma-Separated String to a Numeric Vector

Description

This function takes a comma-separated string, splits it into individual elements, removes any empty components, and converts the remaining elements into numeric values.

Usage

```
commaSeparatedStringToIntArray(inputString)
```

Arguments

`inputString`	A character string containing numeric values separated by commas.

Value

A numeric vector with each element corresponding to a number extracted from the input string.

Examples

```
commaSeparatedStringToIntArray("1,2,3,4")
# [1] 1 2 3 4
```

compareTibbles *Compare Two Tibbles for Differences*

Description

This function compares two tibbles (or data frames) and returns a list describing the differences between them. It first checks whether the two tibbles have identical columns (ignoring order) and reports any additional columns found in either tibble. If the columns are identical, it sorts the rows of both tibbles and performs a row-wise comparison. The output includes whether the tibbles are identical, the difference in row counts, and the specific rows that are present in one tibble but not in the other.

Usage

```
compareTibbles(tibble1, tibble2)
```

Arguments

`tibble1`	A tibble or data frame to be compared.
`tibble2`	A tibble or data frame to be compared.

Details

The function works as follows:

1. It extracts and sorts the column names from both tibbles.
2. It identifies additional columns in either tibble and stores them in the result.
3. If the sets of columns differ, it returns immediately after marking the tibbles as not identical.
4. If the columns are identical, it sorts the rows of both tibbles using `do.call(order, tibble)`, then compares the sorted tibbles row-wise.
5. If the tibbles are not identical, it calculates the differences in row counts and identifies the rows that are present in one tibble but not in the other.

Value

A list with the following elements:

`additionalColumnsInFirst`
A character vector of column names present in `tibble1` but not in `tibble2`.

`additionalColumnsInSecond`
A character vector of column names present in `tibble2` but not in `tibble1`.

`identical` A logical value indicating whether the two tibbles are identical after aligning columns and sorting rows.

`additionalRowsInFirst`
(If not identical) The difference in the number of rows in `tibble1` compared to `tibble2`.

`additionalRowsInSecond`
(If not identical) The difference in the number of rows in `tibble2` compared to `tibble1`.

`presentInFirstNotSecond`
: (If not identical) The rows present in `tibble1` but not in `tibble2`.

`presentInSecondNotFirst`
: (If not identical) The rows present in `tibble2` but not in `tibble1`.

Examples

```
## Not run:
library(dplyr)

# Create example tibbles with identical columns in different orders
tib1 <- tibble(x = c(1, 2, 3), y = c("a", "b", "c"))
tib2 <- tibble(y = c("a", "b", "c"), x = c(1, 2, 3))

# Compare tibbles (should be identical)
compareTibbles(tib1, tib2)

# Modify tib2 by adding an extra row
tib2 <- tib2 %>% dplyr::add_row(x = 4, y = "d")

# Compare again (differences in rows will be reported)
compareTibbles(tib1, tib2)

## End(Not run)
```

executeFeatureExtraction

Execute feature extraction using temporal covariate settings.

Description

This function executes the feature extraction on one or more cohortId in a cohort table and returns covariate data.

Usage

```
executeFeatureExtraction(
  connectionDetails = NULL,
  connection = NULL,
  cdmDatabaseSchema,
  cohortDatabaseSchema,
  cohortIds,
  cohortTable,
  covariateSettings = NULL,
  addCohortBasedTemporalCovariateSettings = TRUE,
  covariateCohortDatabaseSchema = cohortDatabaseSchema,
  includeCovariateIds = NULL,
  covariateCohortTable = cohortTable,
  covariateCohortDefinitionSet = NULL,
  cohortCovariateAnalysisId = 150,
  tempEmulationSchema = getOption("sqlRenderTempEmulationSchema"),
  outputFolder,
  aggregated = TRUE,
  rowIdField = "subject_id",
  incremental = TRUE
)
```

Arguments

`connectionDetails`
: An object of type `connectionDetails` as created using `createConnectionDetails`. Can be left NULL if `connection` is provided. Both cannot be NULL.

`connection`
: An object of type `connection` as created using `connect`. Can be left NULL if `connectionDetails` is provided. Both cannot be NULL.

`cdmDatabaseSchema`
: Schema name where your patient-level data in OMOP CDM format resides. For SQL Server, include both database and schema name (e.g., 'cdm_data.dbo').

`cohortDatabaseSchema`
: Schema name where your cohort tables reside. For SQL Server, include both database and schema name (e.g., 'scratch.dbo').

`cohortIds`
: Vector of cohort IDs for which covariate extraction is to be performed.

`cohortTable`
: Name of the table containing cohort data.

`covariateSettings`
: A FeatureExtraction covariateSettings object. If NULL, cohort-based temporal covariate settings will be used.

`addCohortBasedTemporalCovariateSettings`
: Logical flag indicating whether to add cohort-based temporal covariate settings. Default is TRUE.

`covariateCohortDatabaseSchema`
: Schema name where covariate cohort data resides. Default is the same as `cohortDatabaseSchema`.

`includeCovariateIds`
: Optional vector of covariate IDs to include.

`covariateCohortTable`
: Name of the table containing covariate cohort data. Default is the same as `cohortTable`.

`covariateCohortDefinitionSet`
: A data frame defining covariate cohort definitions used to generate covariate IDs.

`cohortCovariateAnalysisId`
: An integer identifier used to generate covariate IDs. Default is 150.

`tempEmulationSchema`
: Some database platforms (e.g., Oracle, Impala) do not support temporary tables. Provide a schema with write privileges where temporary tables can be created. Default is obtained from `getOption("sqlRenderTempEmulationSchema")`.

`outputFolder`
: Name of the local folder to place results. Use forward slashes (/); avoid network drives.

`aggregated`
: Logical flag indicating whether to aggregate covariate data. Default is TRUE.

`rowIdField`
: Field name used as the row identifier (e.g., "subject_id" or "row_id"). Default is "subject_id".

`incremental`
: Logical flag indicating whether to skip processing if output for a cohort already exists. Default is TRUE.

Value

No explicit return value. The function saves the covariate data files to the specified output folder.

Examples

```
## Not run:
executeFeatureExtraction(
  connectionDetails = myConnectionDetails,
  cdmDatabaseSchema = "cdm_data.dbo",
  cohortDatabaseSchema = "scratch.dbo",
  cohortIds = c(1, 2, 3),
  cohortTable = "cohort_table",
  outputFolder = "results/output",
  covariateSettings = myCovariateSettings
)

## End(Not run)
```

formatCountPercent *Format Count and Percentage as a Single String*

Description

This function formats a count and a percentage value into a single string. The count is formatted with commas using `formatIntegerWithComma`, and the percentage is formatted using `formatPercent` with a specified number of decimal digits. The resulting string is in the format: "formattedCount (formattedPercent)".

Usage

```
formatCountPercent(count, percent, percentDigits = 1)
```

Arguments

`count`	A numeric value representing the count. It is formatted using `formatIntegerWithComma`.
`percent`	A numeric value representing the percentage. It is formatted using `formatPercent`.
`percentDigits`	An integer specifying the number of digits to display for the percentage. Defaults to `1`.

Details

The function concatenates the outputs of these helpers with additional formatting.

Value

A character string combining the formatted count and percentage in the format: `"formattedCount (formattedPercent)"`.

Examples

```
## Not run:
# Example assuming the helper functions are defined:
result <- formatCountPercent(123456, 0.789, percentDigits = 2)
# Might return "123,456 (78.90%)"

## End(Not run)
```

---

formatDecimalWithComma

Format a Decimal Number with Commas and Fixed Decimal Places

Description

This function formats a decimal number by splitting it into an integer part and a decimal part, formatting the integer part with commas as thousand separators, and formatting the decimal part to a fixed number of digits. The decimal part can either be rounded or truncated 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 splits the number into its integer and decimal components. The integer part is formatted using `formatC` with commas inserted as thousand separators. The decimal part is processed either by rounding or truncation, then converted to a string with fixed decimal places. Finally, the two parts are concatenated with a period separator.

Value

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

Examples

```
formatDecimalWithComma(1234567.8912)
# Might return "1,234,567.9"

formatDecimalWithComma(1234567.8912, decimalPlaces = 2, round = FALSE)
# Might return "1,234,567.89"
```

formatIntegerWithComma

Format an Integer with Comma Separators

Description

This function formats a numeric value by first truncating any fractional part. If the input number is a whole number (i.e., it has no fractional part), the function inserts commas as thousand separators using `formatC` with `format = "d"` and `big.mark = ","`. For non-whole numbers, only the integer portion is returned as a character string without commas.

Usage

```
formatIntegerWithComma(number)
```

Arguments

`number`	A numeric value to be formatted as an integer.

Value

A character string representing the formatted integer with commas.

Examples

```
formatIntegerWithComma(1234567)
# [1] "1,234,567"
```

getCovariateSettingsTimeWindows

Get Covariate Settings Time Windows

Description

This function extracts time window information from a covariate settings object. It creates a tibble using the `temporalStartDays` and `temporalEndDays` provided in the covariate settings and performs a left join with the default time windows from `getFeatureExtractionDefaultTimeWindows()`.

Usage

```
getCovariateSettingsTimeWindows(covariateSettings)
```

Arguments

`covariateSettings`	An object (e.g., a list) containing the covariate settings. It must include the elements `temporalStartDays` and `temporalEndDays` that define the start and end days for the time windows.

Details

The function works by first constructing a tibble from the `temporalStartDays` and `temporalEndDays` in the provided covariate settings. It then merges this tibble with the default time window definitions using a left join on the `startDay` and `endDay` columns.

Value

A tibble with the time window definitions. The tibble contains columns `startDay` and `endDay`, along with any additional columns from the default time windows as provided by `getFeatureExtractionDefaultTimeW`

Examples

```
## Not run:
# Example covariate settings list with temporal start and end days
covariateSettings <- list(
  temporalStartDays = c(-365, -30, 1),
  temporalEndDays = c(-1, 0, 30)
)

# Retrieve the time windows based on the covariate settings
timeWindows <- getCovariateSettingsTimeWindows(covariateSettings)
print(timeWindows)

## End(Not run)
```

---

getFeatureExtractionDefaultTemporalCohortCovariateSettings

Get Default Temporal Cohort Covariate Settings for Feature Extraction

---

Description

This function creates default temporal covariate settings for cohort-based feature extraction. It constructs temporal windows from the provided time window definitions and sets up the necessary parameters to extract covariate data for cohorts. If no specific covariate IDs are provided, it uses the cohort IDs from the provided cohort definition set.

Usage

```
getFeatureExtractionDefaultTemporalCohortCovariateSettings(
  timeWindows = getFeatureExtractionDefaultTimeWindows(),
  analysisId = 150,
  covariateCohortDatabaseSchema,
  covariateCohortTable,
  covariateCohortDefinitionSet,
  includedCovariateIds = NULL,
  valueType = "binary"
)
```

Arguments

`timeWindows` A tibble or list containing time window definitions with at least `startDay` and `endDay` elements. Defaults to the output of `getFeatureExtractionDefaultTimeWindows()`.

`analysisId` A numeric identifier for the analysis. Defaults to `150`.

`covariateCohortDatabaseSchema`
A character string specifying the database schema where the covariate cohorts reside.

`covariateCohortTable`
A character string specifying the table name containing the covariate cohorts.

`covariateCohortDefinitionSet`
A data frame or tibble that includes the cohort definitions, with at least the columns `cohortId` and `cohortName`.

`includedCovariateIds`
An optional vector of covariate IDs to include. If `NULL`, the function defaults to using all `cohortId` values from `covariateCohortDefinitionSet`.

`valueType` A character string indicating the type of value to be used in the covariate settings (e.g., "binary"). Defaults to `"binary"`.

Details

The function first constructs a tibble of distinct temporal windows by extracting `startDay` and `endDay` from the provided `timeWindows`. If `includedCovariateIds` is not specified, it is set to the `cohortId` values from the `covariateCohortDefinitionSet`. These parameters are then passed to the `FeatureExtraction::createCohortBasedTemporalCovariateSettings` function, which sets up the covariate extraction process based on the specified temporal windows and cohort information.

Value

An object representing the cohort-based temporal covariate settings, as created by `FeatureExtraction::createCohort`

Examples

```
## Not run:
# Assume default time windows and a cohort definition set are available
covariateCohortDefinitionSet <- dplyr::tibble(
  cohortId = c(1, 2, 3),
  cohortName = c("Cohort A", "Cohort B", "Cohort C")
)

settings <- getFeatureExtractionDefaultTemporalCohortCovariateSettings(
  covariateCohortDatabaseSchema = "my_schema",
  covariateCohortTable = "my_cohort_table",
  covariateCohortDefinitionSet = covariateCohortDefinitionSet
)
print(settings)

## End(Not run)
```

getFeatureExtractionDefaultTemporalCovariateSettings
Get Feature Extraction Default Temporal Covariate Settings

Description

This function creates a set of default temporal covariate settings for feature extraction by configuring various covariate flags and specifying temporal windows. It uses the provided time windows (or the default from `getFeatureExtractionDefaultTimeWindows()`) to construct a tibble of distinct temporal intervals, which is then passed along with the specified covariate settings to `FeatureExtraction::createTemporalCovariateSettings`.

Usage

```
getFeatureExtractionDefaultTemporalCovariateSettings(
  timeWindows = getFeatureExtractionDefaultTimeWindows(),
  useConditionOccurrence = TRUE,
  useProcedureOccurrence = TRUE,
  useDrugEraStart = TRUE,
  useMeasurement = TRUE,
  useConditionEraStart = TRUE,
  useConditionEraOverlap = TRUE,
  useVisitCount = TRUE,
  useVisitConceptCount = TRUE,
  useConditionEraGroupStart = TRUE,
  useConditionEraGroupOverlap = TRUE,
  useDrugExposure = FALSE,
  useDrugEraOverlap = TRUE,
  useDrugEraGroupStart = TRUE,
  useDrugEraGroupOverlap = TRUE,
  useObservation = TRUE,
  useDeviceExposure = TRUE
)
```

Arguments

`timeWindows` A list or tibble containing the temporal window definitions, with elements `startDay` and `endDay`. Defaults to the output of `getFeatureExtractionDefaultTimeWindows()`.

`useConditionOccurrence` Logical indicating whether to include condition occurrence covariates. Defaults to `TRUE`.

`useProcedureOccurrence` Logical indicating whether to include procedure occurrence covariates. Defaults to `TRUE`.

`useDrugEraStart` Logical indicating whether to include drug era start covariates. Defaults to `TRUE`.

`useMeasurement` Logical indicating whether to include measurement covariates. Defaults to `TRUE`.

`useConditionEraStart` Logical indicating whether to include condition era start covariates. Defaults to `TRUE`.

`useConditionEraOverlap`
: Logical indicating whether to include condition era overlap covariates. Defaults to `TRUE`.

`useVisitCount`
: Logical indicating whether to include visit count covariates. Defaults to `TRUE`.

`useVisitConceptCount`
: Logical indicating whether to include visit concept count covariates. Defaults to `TRUE`.

`useConditionEraGroupStart`
: Logical indicating whether to include condition era group start covariates. Defaults to `TRUE`.

`useConditionEraGroupOverlap`
: Logical indicating whether to include condition era group overlap covariates. Defaults to `TRUE`.

`useDrugExposure`
: Logical indicating whether to include drug exposure covariates. Defaults to `FALSE` due to potential overabundance of concept IDs.

`useDrugEraOverlap`
: Logical indicating whether to include drug era overlap covariates. Defaults to `TRUE`.

`useDrugEraGroupStart`
: Logical indicating whether to include drug era group start covariates. Defaults to `TRUE`.

`useDrugEraGroupOverlap`
: Logical indicating whether to include drug era group overlap covariates. Defaults to `TRUE`.

`useObservation`
: Logical indicating whether to include observation covariates. Defaults to `TRUE`.

`useDeviceExposure`
: Logical indicating whether to include device exposure covariates. Defaults to `TRUE`.

Details

This function first constructs a tibble, `feTemporalDays`, by extracting the `startDay` and `endDay` from the provided `timeWindows` and ensuring that the intervals are distinct and ordered. These temporal windows are then used to set the `temporalStartDays` and `temporalEndDays` parameters when calling `FeatureExtraction::createTemporalCovariateSettings` along with other covariate flags.

Value

An object containing the temporal covariate settings, as created by `FeatureExtraction::createTemporalCovariateS`

Examples

```
## Not run:
# Retrieve default temporal covariate settings for feature extraction
temporalSettings <- getFeatureExtractionDefaultTemporalCovariateSettings()
print(temporalSettings)

## End(Not run)
```

getFeatureExtractionDefaultTimeWindows

Extract default time windows for feature extraction

Description

This function reads a CSV file containing time windows for feature extraction and filters the time windows based on the ‘cumulative‘, ‘periodTypes‘, and ‘selectedcumulative‘ parameters. The ‘cumulative‘ parameter filters the time windows by matching the ‘sequencecumulative‘ column, while ‘periodTypes‘ allows filtering by period types (e.g., "month" or "year"). Additionally, if ‘selectedcumulative‘ is TRUE, the function further restricts the results to a predefined subset of cumulative time windows.

Usage

```
getFeatureExtractionDefaultTimeWindows(
  cumulative = NULL,
  periodTypes = NULL,
  selectedcumulative = NULL
)
```

Arguments

`cumulative` A logical value indicating whether cumulative time windows should be returned. TRUE returns only cumulative time windows, FALSE returns non-cumulative windows, and NULL (default) returns all records.

`periodTypes` A character vector specifying the types of periods to filter by. Valid values are "month" or "year". If NULL (default), all period types are returned.

`selectedcumulative` A logical flag that, if TRUE, filters the results to include only a selected subset of cumulative time windows based on specific startDay and endDay criteria.

Value

A data frame containing the filtered time windows with columns: startDay, endDay, periodName, and windowType.

getFeatureExtractionReportByTimeWindows

Generate Feature Extraction Reports by Time Windows

Description

This function generates detailed reports from feature extraction data, analyzing covariates across different time windows. It processes both binary and continuous covariates, and can handle time-varying and non-time-varying features. The function supports filtering by covariate IDs, formatting according to table specifications, and pivoting results for easier interpretation.

Usage

```
getFeatureExtractionReportByTimeWindows(
  covariateData,
  startDays = NULL,
  endDays = NULL,
  includeNonTimeVarying = FALSE,
  minAverageValue = 0.01,
  includedCovariateIds = NULL,
  excludedCovariateIds = NULL,
  table1Specifications = NULL,
  cohortId,
  databaseId = NULL,
  cohortName = NULL,
  reportName = NULL,
  format = TRUE,
  distributionStatistic = c("averageValue", "standardDeviation", "medianValue",
    "p25Value", "p75Value"),
  pivot = TRUE
)
```

Arguments

`covariateData` A covariateData object containing feature extraction results with components:

- analysisRef (analysisId, analysisName, domainId, isBinary, missingMeansZero)
- covariateRef (covariateId, covariateName, analysisId, conceptId, valueAsConceptId, collisions)
- covariates (cohortDefinitionId, covariateId, timeId, sumValue, averageValue)
- covariatesContinuous (cohortDefinitionId, covariateId, countValue, minValue, maxValue, averageValue, standardDeviation, medianValue, p10Value, p25Value, p75Value, p90Value, timeId)
- timeRef (timeId, startDay, endDay)

`startDays` Vector of start days for time windows to include in the report. If NULL, all available time windows from covariateData$timeRef will be used.

`endDays` Vector of end days for time windows to include in the report. If NULL, all available time windows from covariateData$timeRef will be used.

`includeNonTimeVarying`
Boolean indicating whether to include non-time-varying covariates. Default is FALSE.

`minAverageValue`
Minimum average value threshold for including covariates. Default is 0.01.

`includedCovariateIds`
Vector of covariate IDs to include. If NULL, all covariates will be included (subject to other filters).

`excludedCovariateIds`
Vector of covariate IDs to exclude. If NULL, no covariates will be explicitly excluded.

`table1Specifications`
Optional data frame with specifications for formatting as a Table 1, containing columns for label and covariateIds.

`cohortId` The cohort definition ID to generate the report for.

`databaseId` Optional database ID to include in the report header.

`cohortName` Optional cohort name to include in the report header.

`reportName` Optional report name to include in the report header.

`format` Boolean indicating whether to format the values in the report (e.g., as percentages). Default is TRUE.

`distributionStatistic`
Character vector of statistics to include for continuous variables. Default includes "averageValue", "standardDeviation", "medianValue", "p25Value", "p75Value".

`pivot` Boolean indicating whether to pivot the report to have time periods as columns. Default is TRUE.

Details

The function processes both binary and continuous covariates from the provided covariateData object. For binary covariates, it reports counts and percentages. For continuous covariates, it reports the specified distribution statistics. The function can filter covariates based on minimum average value and specific inclusion/exclusion lists.

Time windows are specified using startDays and endDays parameters. If these are NULL, all time windows in the covariateData will be used. Non-time-varying covariates can be included by setting includeNonTimeVarying to TRUE.

The table1Specifications parameter allows for organizing covariates into logical groups in the report, similar to a Table 1 in clinical papers.

Value

A list with two components:

- raw: The raw data frame with all covariate information before formatting
- formatted: The formatted report, either in long format or pivoted (if pivot = TRUE)

Examples

```
## Not run:
# Load covariate data
covariateData <- FeatureExtraction::loadCovariateData("path/to/covariateData")

# Generate report for specific time windows
report <- getFeatureExtractionReportByTimeWindows(
  covariateData = covariateData,
  startDays = c(-365, -30, 0),
  endDays = c(-1, -1, 0),
  includeNonTimeVarying = TRUE,
  cohortId = 1
)

# View the formatted report
View(report$formatted)

## End(Not run)
```

getFeatureExtractionReportCommonSequentialTimePeriods

Get Common Sequential Time Periods for Feature Extraction Reports

Description

This function constructs and returns a tibble of common sequential time periods used in feature extraction reports. The returned tibble defines prior monthly periods, post monthly periods, and a specific day period ("on day of"), each characterized by a unique time identifier, a start day, and an end day.

Usage

```
getFeatureExtractionReportCommonSequentialTimePeriods()
```

Details

These periods are combined and arranged in ascending order based on the `timeId` column.

Value

A tibble with the following columns:

`timeId`	A numeric identifier for the time period.
`startDay`	The start day of the time period relative to a reference date.
`endDay`	The end day of the time period relative to a reference date.

Examples

```
## Not run:
# Retrieve common sequential time periods for feature extraction reports
timePeriods <- getFeatureExtractionReportCommonSequentialTimePeriods()
print(timePeriods)

## End(Not run)
```

getFeatureExtractionReportNonTimeVarying

Get Non-Time-Varying Feature Extraction Report

Description

This function generates a non-time-varying feature extraction report by calling `getFeatureExtractionReportInParall` with predefined settings and then filtering the formatted output to remove specific covariates based on a regular expression pattern.

Usage

```
getFeatureExtractionReportNonTimeVarying(
  covariateDataPath,
  cohortId,
  remove = paste(c("Visit Count", "Chads 2 Vasc", "Demographics Index Month",
    "Demographics Post Observation Time", "Visit Concept Count", "Chads 2",
   "Demographics Prior Observation Time", "Dcsi", "Demographics Time In Cohort",
    "Demographics Index Year Month"), collapse = "|")
)
```

Arguments

`covariateDataPath`
: A character string specifying the file path to the covariate data.

`cohortId`
: A numeric or character identifier for the cohort. This ID is used both to filter covariate data and as part of the file name pattern for retrieving covariate data.

`remove`
: A character string containing a regular expression pattern used to filter out certain covariates from the formatted report. The default pattern removes labels related to visit counts, certain risk scores, and demographic time metrics. Defaults to `'Visit Count|Chads 2 Vasc|Demographics Index Month|Demographics Post Observation Time|Visit Concept Count|Chads 2|Demographics Prior Observation Time|Dcsi|Demographics Time In Cohort|Demographics Index Year Month'`.

Details

After generating the report, if a non-`NULL` `remove` pattern is provided, the function writes a message indicating that it is removing matching labels from the formatted report, and then filters out any rows in the `formatted` report whose `label` matches the pattern.

Value

A list containing the feature extraction report, including both the full formatted output (`formattedFull`) and a filtered version (`formatted`) where rows matching the `remove` pattern in the `label` column have been excluded. If the report generation returns `NULL`, then `NULL` is returned.

Examples

```
## Not run:
# Generate a non-time-varying feature extraction report for a given cohort
report <- getFeatureExtractionReportNonTimeVarying(
  cdmSources = myCdmSources,
  covariateDataPath = "path/to/covariateData",
  cohortId = 123,
  cohortDefinitionSet = myCohortDefinitionSet
)
# View the formatted report
print(report$formatted)

## End(Not run)
```

`getFeatureExtractionStandardizedDifference` *Compute Standardized Difference Between Two Covariate Data Sets*

Description

This function computes the standardized difference between covariate data for two cohorts by loading covariate data from provided file paths and comparing them over defined time windows. The function iterates over each time window to calculate the standardized difference using `FeatureExtraction::computeSt` Optionally, it also computes the standardized difference for non-time-varying covariates when `includeNonTimeVarying` is set to `TRUE`.

Usage

```
getFeatureExtractionStandardizedDifference(
  covariateData1Path = NULL,
  covariateData2Path = NULL,
  cohortId1,
  cohortId2,
  includeNonTimeVarying = TRUE,
  timeRef = NULL
)
```

Arguments

`covariateData1Path`
A character string specifying the file path to the first covariate data set.

`covariateData2Path`
A character string specifying the file path to the second covariate data set.

`cohortId1` A numeric or character identifier for the first cohort (used to filter covariate data).

`cohortId2` A numeric or character identifier for the second cohort (used to filter covariate data).

`includeNonTimeVarying`
A logical value indicating whether to include non-time-varying covariates in the calculation. Defaults to `TRUE`.

`timeRef` An optional data frame that defines the time windows, containing at least the columns `startDay` and `endDay`. If `NULL`, the time reference will be derived from the covariate data.

Value

A tibble containing the computed standardized differences. For each time window, the output includes information such as `startDay`, `endDay`, `covariateId`, `covariateName`, and the calculated standardized difference. If non-time-varying covariates are included, their standardized difference is appended to the result.

Examples

```
## Not run:
# Compute standardized differences between two covariate data sets for cohorts 1 and 2
stdDiff <- getFeatureExtractionStandardizedDifference(
  covariateData1Path = "path/to/covariateData1.rds",
  covariateData2Path = "path/to/covariateData2.rds",
  cohortId1 = 1,
  cohortId2 = 2,
  includeNonTimeVarying = TRUE
)
print(stdDiff)

## End(Not run)
```

getTable1SpecificationsFromCovariateData

Generate Table 1 Specifications from Covariate Data

Description

This function generates specification rows for Table 1 based on covariate data and reference tables. It extracts distinct analyses from the analysis reference and, for each analysis, collects the associated covariate IDs from the covariate reference. A specification row is then created for each analysis using `getTable1SpecificationsRow`. If `covariateData` is provided, it is used to extract both the covariate reference (`covariateRef`) and the analysis reference (`analysisRef`).

Usage

```
getTable1SpecificationsFromCovariateData(
  covariateData = NULL,
  covariateRef = NULL,
  analysisRef = NULL
)
```

Arguments

`covariateData`	An optional object (typically a list) that contains `covariateRef` and `analysisRef`. If provided, these elements will be collected and used in place of the separately supplied `covariateRef` and `analysisRef` parameters.
`covariateRef`	A data frame or tibble containing covariate reference data. This parameter is overridden if `covariateData` is provided.
`analysisRef`	A data frame or tibble containing analysis reference data. This parameter is overridden if `covariateData` is provided.

Details

The function follows these steps:

1. If `covariateData` is provided, extract and collect `covariateRef` and `analysisRef` from it.
2. Select distinct `analysisId` and `analysisName` from the analysis reference.

3. For each analysis, filter the covariate reference to retrieve unique covariate IDs corresponding to that analysis.
4. Create a Table 1 specification row using `getTable1SpecificationsRow` with the collected covariate IDs.
5. Bind all specification rows into a single tibble.

Value

A tibble with one row per analysis containing Table 1 specification details. The tibble includes:

`label`	A formatted label for the analysis (converted from camelCase to Title Case).
`analysisId`	The analysis ID.
`covariateIds`	A comma-separated string of unique covariate IDs associated with the analysis.

Examples

```
## Not run:
# Assuming myCovariateData is a list containing covariateRef and analysisRef:
table1Specs <- getTable1SpecificationsFromCovariateData(covariateData = myCovariateData)
print(table1Specs)

# Alternatively, if covariateRef and analysisRef are provided separately:
table1Specs <- getTable1SpecificationsFromCovariateData(
  covariateRef = myCovariateRef,
  analysisRef = myAnalysisRef
)
print(table1Specs)

## End(Not run)
```

`getTable1SpecificationsRow` *Generate Table 1 Specification Row for Analysis*

Description

This function creates a specification row for Table 1 using analysis information. It takes an analysis ID, concept IDs, covariate IDs, and a label, then produces a tibble with the analysis ID, label, and a comma-separated string of covariate IDs. If concept IDs are provided, they are combined with the analysis ID to generate additional covariate IDs.

Usage

```
getTable1SpecificationsRow(
  analysisId,
  conceptIds = NULL,
  covariateIds = NULL,
  label = "Feature cohorts"
)
```

Arguments

`analysisId`	A numeric value representing the analysis ID. Must be of length one.
`conceptIds`	An optional numeric vector representing concept IDs. Defaults to `NULL`.
`covariateIds`	An optional numeric vector representing covariate IDs. Defaults to `NULL`. At least one of `conceptIds` or `covariateIds` must be provided.
`label`	A character string specifying a label for the feature cohorts. Defaults to `"Feature cohorts"`.

Details

The function validates that at least one of `conceptIds` or `covariateIds` is provided, and ensures that both `analysisId` and `label` are single values. If `conceptIds` is provided, additional covariate IDs are computed using the formula `(conceptIds * 1000) + analysisId`. The resulting covariate IDs (including any provided via `covariateIds`) are then made unique and concatenated into a comma-separated string.

Value

A tibble (data frame) with a single row and the following columns:

`label`	The label for the feature cohorts.
`analysisId`	The provided analysis ID.
`covariateIds`	A comma-separated string of unique covariate IDs derived from the input `covariateIds` and `conceptIds` processed as `(conceptIds * 1000) + analysisId`.

Index