Package: CohortContrast (via r-universe)

May 22, 2026

Type Package

Title Enrichment Analysis of Clinically Relevant Concepts in Common Data Model Cohort Data

Version 1.0.0

Maintainer Markus Haug <markus.haug@ut.ee>

Description Identifies clinically relevant concepts in Observational Medical Outcomes Partnership Common Data Model cohorts using an enrichment-based workflow. Defines target and control cohorts and extracts medical interventions that are over-represented in the target cohort during the observation period. Users can tune filtering and selection thresholds. The workflow includes chi-squared tests for two proportions with Yates continuity correction, logistic tests, and hierarchy and correlation mappings for relevant concepts. The results can be optionally explored using the bundled graphical user interface. For workflow details and examples, see <https://healthinformaticsut.github.io/CohortContrast/>.

License Apache License 2.0

Encoding UTF-8

RoxygenNote 7.3.3

SystemRequirements Python (>= 3.8)

Imports dplyr (>= 1.0.4), tibble, purrr, tidyr (>= 1.0.0), stringr, stats, utils, CDMConnector (>= 2.0.0), CohortConstructor (>= 0.6.0), omopgenerics (>= 1.3.0), lubridate (>= 1.9.0), doParallel, parallel, foreach, data.table (>= 1.14.0), cli, jsonlite (>= 1.8.0), nanoparquet (>= 0.4.0)

Suggests testthat (>= 3.0.0), PatientProfiles (>= 1.1.0), icd.data, duckdb, DBI (>= 1.2.0), RPostgres, readr, knitr, rmarkdown, processx, bit64, reshape2, igraph, Matrix, cluster, vegan, reticulate (>= 1.26)

Config/testthat/edition 3

Config/testthat/parallel false

URL https://healthinformaticsut.github.io/CohortContrast/

BugReports https://github.com/HealthInformaticsUT/CohortContrast/issues

VignetteBuilder knitr

Config/pak/sysreqs libicu-dev libssl-dev python3 libx11-dev

Repository https://healthinformaticsut.r-universe.dev

Date/Publication 2026-04-22 14:39:55 UTC

RemoteUrl https://github.com/healthinformaticsut/cohortcontrast

RemoteRef HEAD

RemoteSha bc5d0b2de19c11cbb04b440a2295d3c0a03cde67

Contents

`automaticCorrelationCombineConcepts` *Function for automatically combining concepts by hierarchy mapping*

Description

Function for automatically combining concepts by hierarchy mapping

Usage

```
automaticCorrelationCombineConcepts(
  data,
  abstractionLevel = -1,
  minCorrelation = 0.7,
  maxDaysInBetween = 1,
  heritageDriftAllowed = FALSE
)
```

Arguments

`data`	CohortContrastObject
`abstractionLevel`	abstraction level to use for mapping
`minCorrelation`	minimum correlation to use for automatic concept combining
`maxDaysInBetween`	minimum days between concepts to use for automatic concept combining
`heritageDriftAllowed`	boolean for allowing heritage drift (combining concepts from differing heritages)

Value

A CohortContrastObject with correlation-based concept merges applied. The returned object keeps the same overall structure as the input, while updating the patient-, feature-, and cohort-level tables together with the complementary mapping table to reflect the executed correlation mappings.

Examples

```
study <- structure(
  list(
    data_initial = data.frame(
      COHORT_DEFINITION_ID = c(rep("target", 4), rep("control", 4)),
      SUBJECT_ID = 1:8,
      COHORT_START_DATE = as.Date(rep("2020-01-01", 8)),
      COHORT_END_DATE = as.Date(rep("2020-01-10", 8))
    ),
    data_patients = data.frame(
```

```
      COHORT_DEFINITION_ID = c(
        "target", "target", "target", "target", "target", "target",
        "control", "control", "control"
      ),
      PERSON_ID = c(1, 1, 2, 2, 3, 4, 5, 6, 7),
      CONCEPT_ID = c(1, 2, 1, 2, 1, 2, 1, 2, 1),
      CONCEPT_NAME = c(
        "Concept A", "Concept B", "Concept A", "Concept B", "Concept A",
        "Concept B", "Concept A", "Concept B", "Concept A"
      ),
      HERITAGE = rep("drug_exposure", 9),
      ABSTRACTION_LEVEL = rep(-1, 9),
      PREVALENCE = rep(1, 9),
      TIME_TO_EVENT = I(list(0, 1, 0, 1, 0, 1, 0, 1, 0))
    ),
    data_features = data.frame(
      CONCEPT_ID = c(1, 2),
      CONCEPT_NAME = c("Concept A", "Concept B"),
      ABSTRACTION_LEVEL = c(-1, -1),
      TARGET_SUBJECT_COUNT = c(3, 3),
      CONTROL_SUBJECT_COUNT = c(2, 1),
      TIME_TO_EVENT = I(list(c(0, 0, 0), c(1, 1, 1))),
      TARGET_SUBJECT_PREVALENCE = c(0.75, 0.75),
      CONTROL_SUBJECT_PREVALENCE = c(0.5, 0.25),
      PREVALENCE_DIFFERENCE_RATIO = c(1.5, 3),
      CHI2Y = c(TRUE, TRUE),
      CHI2Y_P_VALUE = c(0.1, 0.01),
      LOGITTEST = c(FALSE, FALSE),
      LOGITTEST_P_VALUE = c(1, 1),
      HERITAGE = c("drug_exposure", "drug_exposure")
    ),
    data_person = data.frame(
      PERSON_ID = 1:8,
      YEAR_OF_BIRTH = 1980:1987,
      GENDER_CONCEPT_ID = c(8507, 8532, 8507, 8532, 8507, 8532, 8507, 8532)
    ),
    complementaryMappingTable = data.frame(
      CONCEPT_ID = integer(),
      CONCEPT_NAME = character(),
      NEW_CONCEPT_ID = integer(),
      NEW_CONCEPT_NAME = character(),
      ABSTRACTION_LEVEL = integer(),
      TYPE = character()
    ),
    config = list(
      runChi2YTests = TRUE,
      runLogitTests = FALSE,
      presenceFilter = 0,
      prevalenceCutOff = 0
    )
  ),
  class = "CohortContrastObject"
)
```

```
combined <- automaticCorrelationCombineConcepts(
  study,
  abstractionLevel = -1,
  minCorrelation = 0.5,
  maxDaysInBetween = 2
)
combined$data_features
combined$complementaryMappingTable
```

`automaticHierarchyCombineConcepts`
Function for automatically combining concepts by hierarchy mapping

---

Description

Function for automatically combining concepts by hierarchy mapping

Usage

```
automaticHierarchyCombineConcepts(
  data,
  abstractionLevel = -1,
  minDepthAllowed = 0,
  allowOnlyMinors = FALSE
)
```

Arguments

`data`	CohortContrastObject
`abstractionLevel`	abstraction level to use for mapping
`minDepthAllowed`	integer for restricting the mapping, if a concept is part of a hierarchy tree then minDepthAllowed value will prune the tree from said depth value upwards
`allowOnlyMinors`	allows only mappping if child has smaller prevalence than parent

Value

A CohortContrastObject with hierarchy-based concept merges applied. The returned object keeps the same overall structure as the input, while updating the patient-, feature-, and cohort-level tables together with the complementary mapping table to reflect the executed hierarchy mappings.

Examples

```
study <- structure(
  list(
    data_initial = data.frame(
      COHORT_DEFINITION_ID = c("target", "target", "control", "control"),
      SUBJECT_ID = 1:4,
      COHORT_START_DATE = as.Date(rep("2020-01-01", 4)),
      COHORT_END_DATE = as.Date(rep("2020-01-10", 4))
    ),
    data_patients = data.frame(
      COHORT_DEFINITION_ID = c(
        "target", "target", "target", "target", "target", "target",
        "control", "control", "control", "control", "control", "control"
      ),
      PERSON_ID = c(1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4),
      CONCEPT_ID = c(1, 2, 10, 1, 2, 10, 1, 2, 10, 1, 2, 10),
      CONCEPT_NAME = c(
        "Concept A", "Concept B", "Parent concept",
        "Concept A", "Concept B", "Parent concept",
        "Concept A", "Concept B", "Parent concept",
        "Concept A", "Concept B", "Parent concept"
      ),
      HERITAGE = rep("drug_exposure", 12),
      ABSTRACTION_LEVEL = rep(-1, 12),
      PREVALENCE = rep(1, 12),
      TIME_TO_EVENT = I(rep(list(c(0, 2)), 12))
    ),
    data_features = data.frame(
      CONCEPT_ID = c(1, 2, 10),
      CONCEPT_NAME = c("Concept A", "Concept B", "Parent concept"),
      ABSTRACTION_LEVEL = c(-1, -1, -1),
      TARGET_SUBJECT_COUNT = c(2, 2, 2),
      CONTROL_SUBJECT_COUNT = c(2, 2, 2),
      TIME_TO_EVENT = I(list(c(0, 2), c(0, 2), c(0, 2))),
      TARGET_SUBJECT_PREVALENCE = c(1, 1, 1),
      CONTROL_SUBJECT_PREVALENCE = c(1, 1, 1),
      PREVALENCE_DIFFERENCE_RATIO = c(1, 1, 1),
      CHI2Y = c(TRUE, TRUE, TRUE),
      CHI2Y_P_VALUE = c(1, 1, 1),
      LOGITTEST = c(FALSE, FALSE, FALSE),
      LOGITTEST_P_VALUE = c(1, 1, 1),
      HERITAGE = c("drug_exposure", "drug_exposure", "drug_exposure")
    ),
    data_person = data.frame(),
    conceptsData = list(
      concept = data.frame(
        concept_id = c(1, 2, 10),
        concept_name = c("Concept A", "Concept B", "Parent concept"),
        invalid_reason = c(NA, NA, NA)
      ),
      concept_ancestor = data.frame(
        ancestor_concept_id = c(1, 2, 10, 10, 10),
```

```
        descendant_concept_id = c(1, 2, 10, 1, 2),
        min_levels_of_separation = c(0, 0, 0, 1, 1),
        max_levels_of_separation = c(0, 0, 0, 1, 1)
      )
    ),
    complementaryMappingTable = data.frame(
      CONCEPT_ID = integer(),
      CONCEPT_NAME = character(),
      NEW_CONCEPT_ID = integer(),
      NEW_CONCEPT_NAME = character(),
      ABSTRACTION_LEVEL = integer(),
      TYPE = character()
    )
  ),
  class = "CohortContrastObject"
)

combined <- suppressWarnings(
  automaticHierarchyCombineConcepts(study, abstractionLevel = -1)
)
combined$data_features
combined$complementaryMappingTable
```

---

checkDataMode *Check if a Data Directory Contains Summary Mode Data*

Description

Determines whether a data directory contains pre-computed summary data (summary mode) or patient-level data (patient mode).

Usage

```
checkDataMode(dataDir)
```

Arguments

dataDir	Path to the data directory

Value

A list with:

mode	"summary" or "patient"
has_clustering	Logical, whether clustering data is available
clusterKValues	Vector of available k values for clustering

Examples

```
studyPath <- system.file("example", "st", "lc500s", package = "CohortContrast")
info <- checkDataMode(studyPath)
info$mode
```

`checkPythonDeps` *Check Python Dependencies*

Description

Checks if all required Python packages are installed.

Usage

```
checkPythonDeps()
```

Value

A data frame with package names and their installation status.

Examples

```
if (requireNamespace("reticulate", quietly = TRUE) &&
    (nzchar(Sys.which("python3")) || nzchar(Sys.which("python")))) {
  configurePython(createVenv = FALSE)
  checkPythonDeps()
}
```

`CohortContrast` *Run CohortContrast Analysis*

Description

Run CohortContrast Analysis

Usage

```
CohortContrast(
  cdm,
  targetTable = NULL,
  controlTable = NULL,
  pathToResults = getwd(),
 domainsIncluded = c("Drug", "Condition", "Measurement", "Observation", "Procedure",
    "Visit", "Visit detail", "Death"),
  prevalenceCutOff = 10,
  topK = FALSE,
  presenceFilter = 0.005,
  complementaryMappingTable = NULL,
  runChi2YTests = TRUE,
  runLogitTests = TRUE,
  getAllAbstractions = FALSE,
  getSourceData = FALSE,
  maximumAbstractionLevel = 5,
  createOutputFiles = TRUE,
  complName = NULL,
  runRemoveTemporalBias = FALSE,
  removeTemporalBiasArgs = list(),
  runAutomaticHierarchyCombineConcepts = FALSE,
  automaticHierarchyCombineConceptsArgs = list(),
  runAutomaticCorrelationCombineConcepts = FALSE,
  automaticCorrelationCombineConceptsArgs = list(),
  numCores = max(1L, ceiling(0.2 * parallel::detectCores()), na.rm = TRUE)
)
```

Arguments

cdm	Connection to database
targetTable	Table for target cohort (tbl)
controlTable	Table for control cohort (tbl)
pathToResults	Path to the results folder, can be project's working directory
domainsIncluded	list of CDM domains to include
prevalenceCutOff	numeric > if set, removes all of the concepts which are not present (in target) more than prevalenceCutOff times
topK	numeric > if set, keeps this number of features in the analysis. Maximum number of features exported.
presenceFilter	numeric > if set, removes all features represented less than the given percentage
complementaryMappingTable	Mappingtable for mapping concept_ids if present, columns CONCEPT_ID, CONCEPT_NAME, NEW_CONCEPT_ID, NEW_CONCEPT_NAME, ABSTRACTION_LEVEL, TYPE

`runChi2YTests` Boolean for running the CHI2Y test (chi-squared test for two proportions with Yates continuity correction).

`runLogitTests` boolean for logit-tests

`getAllAbstractions`
boolean for creating abstractions' levels for the imported data, this is useful when using GUI and exploring data

`getSourceData` boolean for fetching source data

`maximumAbstractionLevel`
Maximum level of abstraction allowed

`createOutputFiles`
Boolean for creating output files, the default value is TRUE

`complName` Name of the output study directory

`runRemoveTemporalBias`
Logical; when 'TRUE', runs 'removeTemporalBias()' as an optional post-processing step.

`removeTemporalBiasArgs`
A list of additional arguments passed to 'removeTemporalBias()' (for example 'ratio', 'alpha', 'domainsIncluded', 'removeIdentified'). Missing arguments default to 'ratio = 1', 'alpha = 0.05', 'domainsIncluded = NULL', and 'removeIdentified = TRUE'.

`runAutomaticHierarchyCombineConcepts`
Logical; when 'TRUE', runs 'automaticHierarchyCombineConcepts()' after temporal-bias processing.

`automaticHierarchyCombineConceptsArgs`
A list of additional arguments passed to 'automaticHierarchyCombineConcepts()' (for example 'abstractionLevel', 'minDepthAllowed', 'allowOnlyMinors'). Missing arguments default to 'abstractionLevel = -1', 'minDepthAllowed = 0', and 'allowOnlyMinors = TRUE'.

`runAutomaticCorrelationCombineConcepts`
Logical; when 'TRUE', runs 'automaticCorrelationCombineConcepts()' after hierarchy combining.

`automaticCorrelationCombineConceptsArgs`
A list of additional arguments passed to 'automaticCorrelationCombineConcepts()' (for example 'abstractionLevel', 'minCorrelation', 'maxDaysInBetween', 'heritageDriftAllowed'). Missing arguments default to 'abstractionLevel = -1', 'minCorrelation = 0.7', 'maxDaysInBetween = 1', and 'heritageDriftAllowed = FALSE'.

`numCores` Number of cores to allocate to parallel processing. Defaults to 20 percent of detected cores (minimum 1).

Value

A CohortContrastObject. This is a list with the main analysis tables data_patients, data_initial, data_person, data_features, conceptsData, complementaryMappingTable, selectedFeatureData, trajectoryDataList, and config. Together these components contain the processed cohort-level, person-level, feature-level, optional mapping, and configuration outputs produced by the workflow.

Examples

```
if (requireNamespace("CDMConnector", quietly = TRUE) &&
    requireNamespace("DBI", quietly = TRUE) &&
    requireNamespace("duckdb", quietly = TRUE) &&
    nzchar(Sys.getenv("EUNOMIA_DATA_FOLDER")) &&
    isTRUE(tryCatch(
      CDMConnector::eunomiaIsAvailable("GiBleed"),
      error = function(...) FALSE
    ))) {
  pathToJSON <- system.file(
    "example", "example_json", "diclofenac",
    package = "CohortContrast"
  )
  con <- DBI::dbConnect(
    duckdb::duckdb(),
    dbdir = CDMConnector::eunomiaDir("GiBleed")
  )
  cdm <- CDMConnector::cdmFromCon(
    con = con,
    cdmName = "eunomia",
    cdmSchema = "main",
    writeSchema = "main"
  )

  targetTable <- cohortFromJSON(pathToJSON = pathToJSON, cdm = cdm)
  controlTable <- createControlCohortInverse(cdm = cdm, targetTable = targetTable)

  result <- CohortContrast(
    cdm = cdm,
    targetTable = targetTable,
    controlTable = controlTable,
    pathToResults = tempdir(),
    prevalenceCutOff = 1,
    topK = 3,
    presenceFilter = FALSE,
    runChi2YTests = TRUE,
    runLogitTests = TRUE,
    createOutputFiles = FALSE,
    numCores = 1
  )

  head(result$data_features)
  DBI::dbDisconnect(con, shutdown = TRUE)
}
```

Description

Functions to configure Python environment and launch the CohortContrast Viewer dashboard

---

`cohortFromCohortTable` *Read cohort from database cohort table*

Description

Read cohort from database cohort table

Usage

```
cohortFromCohortTable(
  cdm,
  db,
  tableName = NULL,
  schemaName = NULL,
  cohortId = NULL
)
```

Arguments

`cdm`	CDMConnector object
`db`	Database instance (DBI)
`tableName`	Name of the table where the cohort is defined
`schemaName`	Name of the schema where the cohort table is defined
`cohortId`	The id for cohort in cohorts' table, if NULL whole table will be imported

Value

a tbl object for further CohortContrast usage

Examples

```
if (requireNamespace("DBI", quietly = TRUE) &&
    requireNamespace("duckdb", quietly = TRUE) &&
    requireNamespace("CDMConnector", quietly = TRUE)) {
  db <- DBI::dbConnect(duckdb::duckdb(), dbdir = ":memory:")
  DBI::dbExecute(db, "CREATE SCHEMA example")

  cohort <- data.frame(
    cohort_definition_id = c(1L, 2L),
    subject_id = c(101L, 202L),
    cohort_start_date = as.Date(c("2020-01-01", "2020-02-01")),
    cohort_end_date = as.Date(c("2020-01-10", "2020-02-10"))
  )
  DBI::dbWriteTable(db, DBI::SQL('"example"."cohort"'), cohort)
```

```
  targetTable <- cohortFromCohortTable(
    cdm = NULL,
    db = db,
    tableName = "cohort",
    schemaName = "example",
    cohortId = 2
  )
  targetTable

  DBI::dbDisconnect(db, shutdown = TRUE)
}
```

---

cohortFromCSV *Read cohort from CSV*

Description

Read cohort from CSV

Usage

```
cohortFromCSV(pathToCsv, cohortId = NULL)
```

Arguments

pathToCsv	Path to the cohort data CSV file
cohortId	The id for cohort in cohorts' table, if NULL whole table will be imported

Value

a tbl object for further CohortContrast usage

Examples

```
if (requireNamespace("readr", quietly = TRUE)) {
  pathToCsv <- tempfile(fileext = ".csv")
  cohort <- data.frame(
    cohort_definition_id = c(1L, 2L),
    subject_id = c(101L, 202L),
    cohort_start_date = as.Date(c("2020-01-01", "2020-02-01")),
    cohort_end_date = as.Date(c("2020-01-10", "2020-02-10"))
  )
  readr::write_csv(cohort, pathToCsv)

  targetTable <- cohortFromCSV(pathToCsv = pathToCsv, cohortId = 2)
  targetTable
}
```

cohortFromDataTable *Read cohort from data.frame object*

Description

Read cohort from data.frame object

Usage

```
cohortFromDataTable(data, cohortId = NULL)
```

Arguments

data	A data frame with cohort data
cohortId	The id for cohort in cohorts' table, if NULL whole table will be imported

Value

a tbl object for further CohortContrast usage

Examples

```
data <- tibble::tribble(
  ~cohort_definition_id, ~subject_id, ~cohort_start_date, ~cohort_end_date,
  1, 4804, '1997-03-23', '2018-10-29',
  1, 4861, '1982-06-02', '2019-05-23',
  1, 1563, '1977-06-25', '2019-04-20',
  1, 2830, '2006-08-11', '2019-01-14',
  1, 1655, '2004-09-29', '2019-05-24',
  2, 5325, '1982-06-02', '2019-03-17',
  2, 3743, '1997-03-23', '2018-10-07',
  2, 2980, '2004-09-29', '2018-04-01',
  2, 1512, '2006-08-11', '2017-11-29',
  2, 2168, '1977-06-25', '2018-11-22'
)
targetTable <- cohortFromDataTable(data = data, cohortId = 2)
targetTable
```

cohortFromJSON *Read cohort from JSON*

Description

Read cohort from JSON

Usage

```
cohortFromJSON(pathToJSON, cdm, cohortId = NULL)
```

Arguments

pathToJSON	Path to the cohort data JSON file
cdm	Connection to the database (package CDMConnector)
cohortId	The id for cohort in cohorts' table, if NULL whole table will be imported

Value

a tbl object for further CohortContrast usage

Examples

```
if (requireNamespace("CDMConnector", quietly = TRUE) &&
    requireNamespace("DBI", quietly = TRUE) &&
    requireNamespace("duckdb", quietly = TRUE) &&
    nzchar(Sys.getenv("EUNOMIA_DATA_FOLDER")) &&
    isTRUE(tryCatch(
      CDMConnector::eunomiaIsAvailable("GiBleed"),
      error = function(...) FALSE
    ))) {
  pathToJSON <- system.file(
    "example", "example_json", "diclofenac",
    package = "CohortContrast"
  )
  con <- DBI::dbConnect(
    duckdb::duckdb(),
    dbdir = CDMConnector::eunomiaDir("GiBleed")
  )
  cdm <- CDMConnector::cdmFromCon(
    con = con,
    cdmName = "eunomia",
    cdmSchema = "main",
    writeSchema = "main"
  )

  targetTable <- cohortFromJSON(pathToJSON = pathToJSON, cdm = cdm)
  targetTable

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

`configurePython` *Configure Python Environment for CohortContrast Viewer*

Description

Sets up the Python environment for running the CohortContrast Viewer dashboard. This function can create a new virtual environment or use an existing Python installation.

Usage

```
configurePython(
  pythonPath = NULL,
  virtualenvName = "r-cohortcontrast-viewer",
  createVenv = TRUE,
  force = FALSE
)
```

Arguments

`pythonPath`	Optional path to a specific Python executable. If NULL, reticulate will auto-detect Python.
`virtualenvName`	Name for the virtual environment. Default is "r-cohortcontrast-viewer".
`createVenv`	Logical. If TRUE, creates a new virtual environment. Default is TRUE. Set to FALSE on systems without python3-venv package.
`force`	Logical. If TRUE, recreates the virtual environment even if it exists. Default is FALSE.

Value

Invisibly returns TRUE if configuration was successful.

Examples

```
if (requireNamespace("reticulate", quietly = TRUE) &&
    (nzchar(Sys.which("python3")) || nzchar(Sys.which("python")))) {
  configurePython(createVenv = FALSE)
  getPythonInfo()
}
```

createControlCohortInverse *Function for creating automatic matches based on inverse control logic*

Description

Function for creating automatic matches based on inverse control logic

Usage

```
createControlCohortInverse(cdm, targetTable)
```

Arguments

cdm	Connection to the database (package CDMConnector)
targetTable	A cohort tibble which contains subjects' cohort data

Value

A data frame representing inverse control time windows for the target subjects. The returned table contains cohort_definition_id, subject_id, cohort_start_date, and cohort_end_date columns, where each row captures an observation-period segment outside the target cohort interval.

Examples

```
if (requireNamespace("CDMConnector", quietly = TRUE) &&
    requireNamespace("DBI", quietly = TRUE) &&
    requireNamespace("duckdb", quietly = TRUE) &&
    nzchar(Sys.getenv("EUNOMIA_DATA_FOLDER")) &&
    isTRUE(tryCatch(
      CDMConnector::eunomiaIsAvailable("GiBleed"),
      error = function(...) FALSE
    ))) {
  pathToJSON <- system.file(
    "example", "example_json", "diclofenac",
    package = "CohortContrast"
  )
  con <- DBI::dbConnect(
    duckdb::duckdb(),
    dbdir = CDMConnector::eunomiaDir("GiBleed")
  )
  cdm <- CDMConnector::cdmFromCon(
    con = con,
    cdmName = "eunomia",
    cdmSchema = "main",
    writeSchema = "main"
  )
```

```
  targetTable <- cohortFromJSON(pathToJSON = pathToJSON, cdm = cdm)
  controlTable <- createControlCohortInverse(cdm = cdm, targetTable = targetTable)
  head(controlTable)

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

`createControlCohortMatching` *Function for creating automatic matches based on age and sex*

Description

Function for creating automatic matches based on age and sex

Usage

```
createControlCohortMatching(
  cdm,
  targetTable,
  ratio = 1,
  max = NULL,
  min = NULL
)
```

Arguments

`cdm`	Connection to the database (package CDMConnector)
`targetTable`	A cohort tibble which contains subjects' cohort data
`ratio`	ratio for the number of matches generated
`max`	Maximum ratio to use
`min`	Minimum ratio to use

Value

A data frame representing the matched control cohort. The returned table contains cohort_definition_id, subject_id, cohort_start_date, and cohort_end_date columns, with one row per matched control interval aligned to the target cohort follow-up logic.

Examples

```
if (requireNamespace("CDMConnector", quietly = TRUE) &&
    requireNamespace("DBI", quietly = TRUE) &&
    requireNamespace("duckdb", quietly = TRUE) &&
    nzchar(Sys.getenv("EUNOMIA_DATA_FOLDER")) &&
    isTRUE(tryCatch(
```

```
    CDMConnector::eunomiaIsAvailable("GiBleed"),
    error = function(...) FALSE
  ))) {
pathToJSON <- system.file(
  "example", "example_json", "diclofenac",
  package = "CohortContrast"
)
con <- DBI::dbConnect(
  duckdb::duckdb(),
  dbdir = CDMConnector::eunomiaDir("GiBleed")
)
cdm <- CDMConnector::cdmFromCon(
  con = con,
  cdmName = "eunomia",
  cdmSchema = "main",
  writeSchema = "main"
)

targetTable <- cohortFromJSON(pathToJSON = pathToJSON, cdm = cdm)
controlTable <- createControlCohortMatching(
  cdm = cdm,
  targetTable = targetTable,
  ratio = 1
)
head(controlTable)

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

`generateMappingTable` *Create a mapping table with predefined maximum abstraction level*

Description

Create a mapping table with predefined maximum abstraction level

Usage

```
generateMappingTable(
  abstractionLevel = 10,
  data = NULL,
  maxMinDataFrame = NULL
)
```

Arguments

`abstractionLevel`
: Maximum abstraction level allowed

data CohortContrastObject returned by CohortContrast function

maxMinDataFrame
Optional data frame describing the maximum available abstraction depth per descendant concept. Must contain columns ‘descendant_concept_id‘ and ‘maximum_minimal_separation‘. If ‘NULL‘, the summary is computed automatically from ‘data$conceptsData$concept_ancestor‘.

Value

A data frame describing hierarchy-based concept mappings. Each row maps an original concept to a replacement concept through the columns CONCEPT_ID, CONCEPT_NAME, NEW_CONCEPT_ID, NEW_CONCEPT_NAME, ABSTRACTION_LEVEL, and TYPE. An empty data frame is returned if no concepts need to be mapped at the requested abstraction level.

Examples

```
if (requireNamespace("nanoparquet", quietly = TRUE)) {
  studyDir <- system.file("example", "st", package = "CohortContrast")
  study <- loadCohortContrastStudy("lc500", pathToResults = studyDir)

  maxMinDataFrame <- data.frame(
    descendant_concept_id = c(
      4008211, 4176729, 2107967, 2107968, 2108158, 32280, 32815
    ),
    maximum_minimal_separation = c(0, 1, 0, 1, 1, 0, 1)
  )

  mappingTable <- generateMappingTable(
    abstractionLevel = 0,
    data = study,
    maxMinDataFrame = maxMinDataFrame
  )
  mappingTable
}
```

---

getPythonInfo *Get Python Configuration Information*

Description

Returns information about the current Python configuration for the CohortContrast Viewer.

Usage

```
getPythonInfo()
```

Value

A list with Python configuration details

Examples

```
if (requireNamespace("reticulate", quietly = TRUE) &&
    (nzchar(Sys.which("python3")) || nzchar(Sys.which("python")))) {
  configurePython(createVenv = FALSE)
  getPythonInfo()
}
```

`getTopSeparatingConcepts` *Get Top Concepts That Best Separate Clusters*

Description

Returns the top ‘n‘ main concepts ranked by standard deviation of concept prevalence across clusters (same ranking idea as the UI "Top N by SD" filter).

Usage

```
getTopSeparatingConcepts(
  studyPath,
  n = 10,
  k = "auto",
  precomputeIfNeeded = TRUE,
  clusterKValuesWhenAuto = c(2, 3, 4, 5),
  conceptLimit = 60,
  ...
)
```

Arguments

`studyPath`	Path to a CohortContrast study folder (summary or patient).
`n`	Number of top concepts to return.
`k`	Cluster count as an integer, or ‘"auto"‘ to select the best precomputed ‘k‘ from ‘metadata.json‘.
`precomputeIfNeeded`	Logical; if ‘TRUE‘ and clustering summaries are not available, run [precomputeSummary()] into a temporary directory.
`clusterKValuesWhenAuto`	Integer vector of ‘k‘ values to precompute when ‘k = "auto"‘ and precomputation is needed.
`conceptLimit`	Maximum concept count passed to [precomputeSummary()] when precomputation is needed.
`...`	Additional arguments forwarded to [precomputeSummary()] when precomputation is needed.

Details

The function accepts either: - A summary-mode directory (with ‘metadata.json‘ and ‘concept_summaries.parquet‘) - A patient-level study directory (with ‘data_patients.parquet‘)

For patient-level inputs, clustering summaries are generated on the fly with [precomputeSummary()] when needed.

Value

A ‘data.frame‘ with columns:

id Concept ID

name Concept name

enrichment Target/control prevalence ratio

target_prevalence Target cohort prevalence

The selected ‘k‘ is attached as ‘attr(result, "k")‘.

Examples

```
if (requireNamespace("nanoparquet", quietly = TRUE)) {
  summaryPath <- system.file("example", "st", "lc500s", package = "CohortContrast")

  top_auto <- getTopSeparatingConcepts(summaryPath, n = 5, k = "auto")
  top_k3 <- getTopSeparatingConcepts(summaryPath, n = 5, k = 3)

  head(top_auto)
  head(top_k3)
}
```

---

installPythonDeps *Install Python Dependencies*

Description

Installs the required Python packages for the CohortContrast Viewer.

Usage

```
installPythonDeps(upgrade = FALSE, quiet = FALSE, user = NULL)
```

Arguments

upgrade	Logical. If TRUE, upgrades existing packages. Default is FALSE.
quiet	Logical. If TRUE, suppresses pip output. Default is FALSE.
user	Logical. If TRUE, installs to user site-packages (–user flag). Useful when not using a virtual environment. Default is NULL (auto-detect).

Value

Invisibly returns TRUE if installation was successful.

Examples

```
## Not run:
if (interactive() &&
    requireNamespace("reticulate", quietly = TRUE) &&
    (nzchar(Sys.which("python3")) || nzchar(Sys.which("python")))) {
  configurePython(createVenv = FALSE)
  installPythonDeps(user = TRUE)
}

## End(Not run)
```

`installPythonDepsOffline` *Install Python Dependencies Offline*

Description

Installs Python packages from pre-downloaded wheel files (offline installation). This is useful for air-gapped servers without internet access.

Usage

```
installPythonDepsOffline(
  packagesDir = NULL,
  platform = "linux_x86_64",
  cleanup = TRUE
)
```

Arguments

`packagesDir`	Path to directory containing wheel files or a .zip archive. This argument is required in CRAN builds.
`platform`	Platform identifier. Default is "linux_x86_64".
`cleanup`	Logical. If TRUE, removes extracted files after installation. Default is TRUE.

Details

The function supports two formats:

- A .zip file (e.g., "linux_x86_64.zip") containing wheel files
- A directory containing .whl files directly

For offline installation:

1. Download wheel files on a machine with internet access
2. Copy the packages folder to the offline server
3. Run this function to install from local files

Value

Invisibly returns TRUE if installation was successful.

Examples

```
## Not run:
packagesDir <- Sys.getenv("COHORT_CONTRAST_WHEELS")
if (interactive() &&
    requireNamespace("reticulate", quietly = TRUE) &&
    nzchar(packagesDir) &&
    dir.exists(packagesDir)) {
  configurePython(createVenv = FALSE)
  installPythonDepsOffline(packagesDir = packagesDir)
}

## End(Not run)
```

---

loadCohortContrastStudy

Load a Saved CohortContrast Study

Description

Loads a saved study folder produced by ‘CohortContrast()‘ and reconstructs a ‘CohortContrastObject‘ for downstream post-processing.

Usage

```
loadCohortContrastStudy(studyName, pathToResults = getwd())
```

Arguments

studyName	Name of the study folder inside ‘pathToResults‘.
pathToResults	Path to the parent directory containing study folders. Defaults to current working directory.

Value

A ‘CohortContrastObject‘.

Examples

```
if (requireNamespace("nanoparquet", quietly = TRUE)) {
  studyDir <- system.file("example", "st", package = "CohortContrast")
  study <- loadCohortContrastStudy("lc500", pathToResults = studyDir)
  class(study)
}
```

`matchCohortsByAge` *Function for matching the control to target by age*

Description

Function for matching the control to target by age

Usage

```
matchCohortsByAge(cdm, sourceTable, tableToMatch, maxAllowedAgeDifference = 0)
```

Arguments

`cdm`	Connection to the database (package CDMConnector)
`sourceTable`	Table which is used as reference for matching
`tableToMatch`	Table which is matched
`maxAllowedAgeDifference`	Value for maximum allowed age difference to be mapped to

Value

A list with two data frames named source and tableToMatch. Each data frame contains the matched rows from the corresponding input table after age-based sampling, preserving the original column structure of the input cohort tables.

Examples

```
cdm <- list(
  person = tibble::tibble(
    person_id = 1:13,
    year_of_birth = c(1980L, 1981L, 1982L, 1983L, 1984L,
                      1980L, 1981L, 1982L, 1985L, 1986L, 1982L, 1983L, 1984L)
  )
)
sourceTable <- tibble::tibble(
  cohort_definition_id = 1L,
  subject_id = c(1L, 2L, 3L, 4L, 5L),
  cohort_start_date = as.Date(rep("2020-01-01", 5)),
  cohort_end_date = as.Date(rep("2020-01-10", 5))
)
tableToMatch <- tibble::tibble(
```

```
  cohort_definition_id = 2L,
  subject_id = c(6L, 7L, 8L, 9L, 10L, 11L, 12L, 13L),
  cohort_start_date = as.Date(rep("2020-01-01", 8)),
  cohort_end_date = as.Date(rep("2020-01-10", 8))
)

matched <- matchCohortsByAge(
  cdm = cdm,
  sourceTable = sourceTable,
  tableToMatch = tableToMatch,
  maxAllowedAgeDifference = 1
)
nrow(tableToMatch)
matched
```

`nGramClusterSummarization` *Function for summarizing ngrams from nGramDiscovery output*

Description

Function for summarizing ngrams from nGramDiscovery output

Usage

```
nGramClusterSummarization(result, top_n = 5)
```

Arguments

`result`	output from nGramDiscovery
`top_n`	integer for number of most frequent concept to show per cluster

Value

A data frame with one row per n-gram cluster. The table reports the number of n-grams in each cluster, average number of persons per n-gram, total unique patients represented in the cluster, average timing, and a concatenated label of the most frequent concepts in that cluster.

Examples

```
if (requireNamespace("nanoparquet", quietly = TRUE) &&
    requireNamespace("Matrix", quietly = TRUE) &&
    requireNamespace("vegan", quietly = TRUE) &&
    requireNamespace("cluster", quietly = TRUE)) {
  studyDir <- system.file("example", "st", package = "CohortContrast")
  study <- loadCohortContrastStudy("lc500", pathToResults = studyDir)
  ngrams <- nGramDiscovery(study)
  clusterSummary <- suppressWarnings(nGramClusterSummarization(ngrams))
  head(clusterSummary)
```

```
}
```

`nGramDiscovery` *Function for discovering ngrams from the extracted data from CohortContrast function*

Description

Function for discovering ngrams from the extracted data from CohortContrast function

Usage

```
nGramDiscovery(data, collapse_size = 0)
```

Arguments

`data`	CohortContrastObject returned by CohortContrast or GUI snapshot
`collapse_size`	integer for days to use for collapse same concept ids

Value

A data frame of enriched n-grams, or NULL if no significant n-grams are found. The returned table contains one row per detected n-gram with cluster assignment, concept identifiers and names, observed and expected counts, over-representation statistics, person counts, timing summaries, and p-values used to flag significant patterns.

Examples

```
if (requireNamespace("nanoparquet", quietly = TRUE) &&
    requireNamespace("Matrix", quietly = TRUE) &&
    requireNamespace("vegan", quietly = TRUE) &&
    requireNamespace("cluster", quietly = TRUE)) {
  studyDir <- system.file("example", "st", package = "CohortContrast")
  study <- loadCohortContrastStudy("lc500", pathToResults = studyDir)
  ngrams <- nGramDiscovery(study)
  head(ngrams)
}
```

precomputeSummary *Pre-compute Summary Data for a Study*

Description

Generates aggregated summary data from patient-level parquet files, removing all individual patient information. This creates a "summary mode" dataset that can be shared without privacy concerns.

Usage

```
precomputeSummary(
  studyPath,
  outputPath = NULL,
  clusterKValues = c(2, 3, 4, 5),
  conceptLimit = 60,
  minCellCount = 0,
  maxParallelJobs = 1,
  minibatchKMeansCutoffPatients = 50000
)
```

Arguments

studyPath	Path to directory containing patient-level parquet files (data_patients.parquet, data_features.parquet, etc.)
outputPath	Path for output summary files. Default is studyPath + "_summary"
clusterKValues	Vector of k values to pre-compute clustering for. Default is c(2, 3, 4, 5).
conceptLimit	Maximum number of concepts to use for clustering. Default is 60.
minCellCount	Minimum patient count for small cell suppression (privacy). Counts between 1 and (minCellCount-1) are rounded up to minCellCount. Default is 0 (disabled). Set to e.g. 5 to apply suppression.
maxParallelJobs	Maximum number of parallel clustering jobs. Default is 1 (sequential) to avoid out-of-memory errors on servers. Set to 2-4 on machines with ample RAM for faster execution.
minibatchKMeansCutoffPatients	If target patient count is greater than this value, clustering uses MiniBatchKMeans instead of KMedoids. Default is 50000.

Details

The function pre-computes: - Concept-level time distribution statistics (for violin/box plots) - Age and gender statistics per concept - Ordinal concept summaries (1st, 2nd, 3rd occurrences) - Clustering results for k=2,3,4,5 clusters - Cluster overlap and differentiation metrics

Value

A list with:

outputPath	Path to the generated summary directory.
files	Named list of generated file paths
metadata	Study metadata including demographics and clustering info

Examples

```
if (requireNamespace("reticulate", quietly = TRUE) &&
    (nzchar(Sys.which("python3")) || nzchar(Sys.which("python")))) {
  configurePython(createVenv = FALSE)
  deps <- checkPythonDeps()
  if (all(deps$installed)) {
    studyPath <- system.file("example", "st", "lc500", package = "CohortContrast")
    outputPath <- file.path(tempdir(), "lc500_summary")
    result <- precomputeSummary(
      studyPath = studyPath,
      outputPath = outputPath,
      clusterKValues = c(2, 3),
      conceptLimit = 20,
      maxParallelJobs = 1
    )
    result$outputPath
  }
}
```

---

removeTemporalBias *Remove Temporal Bias from CohortContrast Analysis*

Description

This function identifies and optionally removes concepts that may represent temporal bias in a CohortContrast analysis. It works by creating age/sex matched controls from the general population for the same time periods as the target cohort, then using a proportion test to identify concepts where the matched cohort has greater or equal prevalence compared to the target. These concepts likely represent temporal trends (e.g., seasonal effects, healthcare changes) rather than condition-specific features.

Usage

```
removeTemporalBias(
  data,
  cdm,
  ratio = 1,
  alpha = 0.05,
```

```
  domainsIncluded = NULL,
  removeIdentified = FALSE
)
```

Arguments

data	A CohortContrast result object (returned from CohortContrast function)
cdm	Connection to the database (package CDMConnector)
ratio	Matching ratio for control cohort generation (default: 1)
alpha	Significance level for the proportion test before Bonferroni correction (default: 0.05)
domainsIncluded	Domains to analyze for temporal bias (default: same as original analysis)
removeIdentified	If TRUE, automatically remove identified temporal bias concepts from the data (default: FALSE)

Details

The function applies Bonferroni correction for multiple testing, adjusting the significance level by dividing alpha by the number of concepts being tested.

Value

A list containing:

temporal_bias_concepts	A data frame of concepts identified as potential temporal bias
data	The original or filtered CohortContrast data object (if removeIdentified = TRUE)
matched_control_prevalences	Prevalence data from the matched control cohort

Examples

```
if (requireNamespace("CDMConnector", quietly = TRUE) &&
    requireNamespace("DBI", quietly = TRUE) &&
    requireNamespace("duckdb", quietly = TRUE) &&
    nzchar(Sys.getenv("EUNOMIA_DATA_FOLDER")) &&
    isTRUE(tryCatch(
      CDMConnector::eunomiaIsAvailable("GiBleed"),
      error = function(...) FALSE
    ))) {
  pathToJSON <- system.file(
    "example", "example_json", "diclofenac",
    package = "CohortContrast"
  )
  con <- DBI::dbConnect(
    duckdb::duckdb(),
    dbdir = CDMConnector::eunomiaDir("GiBleed")
```

```
  )
  cdm <- CDMConnector::cdmFromCon(
    con = con,
    cdmName = "eunomia",
    cdmSchema = "main",
    writeSchema = "main"
  )

  targetTable <- cohortFromJSON(pathToJSON = pathToJSON, cdm = cdm)
  controlTable <- createControlCohortInverse(cdm = cdm, targetTable = targetTable)
  data <- CohortContrast(
    cdm = cdm,
    targetTable = targetTable,
    controlTable = controlTable,
    pathToResults = tempdir(),
    prevalenceCutOff = 1,
    topK = 3,
    presenceFilter = FALSE,
    runChi2YTests = TRUE,
    runLogitTests = TRUE,
    createOutputFiles = FALSE,
    numCores = 1
  )

  result_filtered <- removeTemporalBias(
    data = data,
    cdm = cdm,
    ratio = 1,
    removeIdentified = TRUE
  )
  head(result_filtered$data$data_features)

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

`resolveCohortTableOverlaps` *Resolve overlaps inside the cohort table*

Description

Resolve overlaps inside the cohort table

Usage

```
resolveCohortTableOverlaps(cohortTable, cdm)
```

Arguments

cohortTable	A table with cohort table characteristics
cdm	CDMConnector object: connection to the database

Value

A dataframe like cohort table with resolved overlaps

Examples

```
cohortTable <- data.frame(
  cohort_definition_id = c(1L, 1L, 1L),
  subject_id = c(1L, 1L, 2L),
  cohort_start_date = as.Date(c("2020-01-01", "2020-01-05", "2020-02-01")),
  cohort_end_date = as.Date(c("2020-01-10", "2020-01-20", "2020-02-10"))
)
cdm <- list(
  observation_period = data.frame(
    person_id = c(1L, 2L),
    observation_period_start_date = as.Date(c("2020-01-03", "2020-01-15")),
    observation_period_end_date = as.Date(c("2020-01-18", "2020-02-20"))
  )
)

resolveCohortTableOverlaps(cohortTable, cdm)
```

---

runCohortContrastViewer

Run CohortContrast Viewer Dashboard

Description

Launches the CohortContrast Viewer interactive dashboard.

Usage

```
runCohortContrastViewer(
  dataDir = NULL,
  port = 8050,
  host = "127.0.0.1",
  mode = c("simple", "server", "debug"),
  logFile = NULL,
  allowExports = TRUE,
  openBrowser = TRUE,
  background = TRUE
)
```

Arguments

dataDir	Path to the directory containing parquet data files. If NULL, defaults to the current working directory (‘getwd()‘).
port	Port number for the Dash server. Default is 8050.
host	Host address. Default is "127.0.0.1" (localhost).
mode	Run mode. One of ‘"simple"‘, ‘"server"‘, ‘"debug"‘. ‘"simple"‘ hides debug-style output and is the default. ‘"server"‘ enables file logging suitable for hosted/server runs. ‘"debug"‘ enables maximum debug features.
logFile	Optional log file path for ‘"server"‘ (or ‘"debug"‘) mode. If NULL in ‘"server"‘ mode, defaults to ‘file.path(dataDir, "contrast_viewer.log")‘.
allowExports	Logical. If FALSE, disables export actions (TSV/PNG) in the UI. Default is TRUE.
openBrowser	Logical. If TRUE, opens the dashboard in the default browser. Default is TRUE.
background	Logical. If TRUE, runs the server in the background. Default is TRUE.

Value

If background is TRUE, invisibly returns the process object. If FALSE, blocks until the server is stopped.

Examples

```
if (interactive() &&
    requireNamespace("reticulate", quietly = TRUE) &&
    requireNamespace("processx", quietly = TRUE) &&
    (nzchar(Sys.which("python3")) || nzchar(Sys.which("python")))) {
  configurePython(createVenv = FALSE)
  deps <- checkPythonDeps()
  if (all(deps$installed)) {
    summaryDir <- system.file("example", "st", "lc500s", package = "CohortContrast")
    runCohortContrastViewer(
      dataDir = summaryDir,
      openBrowser = FALSE,
      background = TRUE
    )
    stopCohortContrastViewer()
  }
}
```

`stopCohortContrastViewer` *Stop CohortContrast Viewer Dashboard*

Description

Stops a running CohortContrast Viewer dashboard instance.

Usage

```
stopCohortContrastViewer()
```

Value

Invisibly returns TRUE if the server was stopped.

Examples

```
if (interactive() &&
    requireNamespace("reticulate", quietly = TRUE) &&
    requireNamespace("processx", quietly = TRUE) &&
    (nzchar(Sys.which("python3")) || nzchar(Sys.which("python")))) {
  configurePython(createVenv = FALSE)
  deps <- checkPythonDeps()
  if (all(deps$installed)) {
    summaryDir <- system.file("example", "st", "lc500s", package = "CohortContrast")
    runCohortContrastViewer(
      dataDir = summaryDir,
      openBrowser = FALSE,
      background = TRUE
    )
    stopCohortContrastViewer()
  }
}
```

Index