Single studies using the CohortMethod package

Martijn J. Schuemie, Marc A. Suchard and Patrick Ryan

2017-10-30

Contents

1 Introduction

This vignette describes how you can use the `CohortMethod` package to perform a single new-user cohort study. We will walk through all the steps needed to perform an exemplar study, and we have selected the well-studied topic of the effect of coxibs versus non-selective non-steroidal anti-inflammatory drugs (NSAIDs) on gastrointestinal (GI) bleeding-related hospitalization. For simplicity, we focus on one coxib – celecoxib – and one non-selective NSAID – diclofenac.

2 Installation instructions

Before installing the `CohortMethod` package make sure you have Java available. Java can be downloaded from www.java.com. For Windows users, RTools is also necessary. RTools can be downloaded from CRAN.

The `CohortMethod` package is currently maintained in a Github repository, and has dependencies on other packages in Github. All of these packages can be downloaded and installed from within R using the `drat` package:

```
install.packages("drat")
drat::addRepo("OHDSI")
install.packages("CohortMethod")
```

Once installed, you can type `library(CohortMethod)` to load the package.

3 Data extraction

The first step in running the `CohortMethod` is extracting all necessary data from the database server holding the data in the Observational Medical Outcomes Partnership (OMOP) Common Data Model (CDM) format.

3.1 Configuring the connection to the server

We need to tell R how to connect to the server where the data are. `CohortMethod` uses the `DatabaseConnector` package, which provides the `createConnectionDetails` function. Type `?createConnectionDetails` for the specific settings required for the various database management systems (DBMS). For example, one might connect to a PostgreSQL database using this code:

```
connectionDetails <- createConnectionDetails(dbms = "postgresql",
                                             server = "localhost/ohdsi",
                                             user = "joe",
                                             password = "supersecret")

cdmDatabaseSchema <- "my_cdm_data"
resultsDatabaseSchema <- "my_results"
```

The last two lines define the `cdmDatabaseSchema` and `resultSchema` variables. We'll use these later to tell R where the data in CDM format live, and where we want to write intermediate tables. Note that for Microsoft SQL Server, databaseschemas need to specify both the database and the schema, so for example `cdmDatabaseSchema <- "my_cdm_data.dbo"`.

3.2 Preparing the exposures and outcome(s)

We need to define the exposures and outcomes for our study. One could use an external cohort definition tools, but in this example we do this by writing SQL statements against the OMOP CDM that populate a table of events in which we are interested. The resulting table should have the same structure as the `cohort` table in the CDM. This means it should have the fields `cohort_definition_id`, `cohort_start_date`, `cohort_end_date`,and `subject_id`.

For our example study, we have created a file called *coxibVsNonselVsGiBleed.sql* with the following contents:

```
/**********************************
File coxibVsNonselVsGiBleed.sql
**********************************/

IF OBJECT_ID('@resultsDatabaseSchema.coxibVsNonselVsGiBleed', 'U') IS NOT NULL
  DROP TABLE @resultsDatabaseSchema.coxibVsNonselVsGiBleed;

CREATE TABLE @resultsDatabaseSchema.coxibVsNonselVsGiBleed (
  cohort_definition_id INT,
  cohort_start_date DATE,
    cohort_end_date DATE,
```

```
    subject_id BIGINT
    );

INSERT INTO @resultsDatabaseSchema.coxibVsNonselVsGiBleed (
    cohort_definition_id,
    cohort_start_date,
    cohort_end_date,
    subject_id
    )
SELECT 1, -- Exposure
    drug_era_start_date,
    drug_era_end_date,
    person_id
FROM @cdmDatabaseSchema.drug_era
WHERE drug_concept_id = 1118084;-- celecoxib

INSERT INTO @resultsDatabaseSchema.coxibVsNonselVsGiBleed (
    cohort_definition_id,
    cohort_start_date,
    cohort_end_date,
    subject_id
    )
SELECT 2, -- Comparator
    drug_era_start_date,
    drug_era_end_date,
    person_id
FROM @cdmDatabaseSchema.drug_era
WHERE drug_concept_id = 1124300; --diclofenac

INSERT INTO @resultsDatabaseSchema.coxibVsNonselVsGiBleed (
    cohort_definition_id,
    cohort_start_date,
    cohort_end_date,
    subject_id
    )
SELECT 3, -- Outcome
    condition_start_date,
    condition_end_date,
    condition_occurrence.person_id
FROM @cdmDatabaseSchema.condition_occurrence
INNER JOIN @cdmDatabaseSchema.visit_occurrence
    ON condition_occurrence.visit_occurrence_id = visit_occurrence.visit_occurrence_id
WHERE condition_concept_id IN (
        SELECT descendant_concept_id
        FROM @cdmDatabaseSchema.concept_ancestor
        WHERE ancestor_concept_id = 192671 -- GI - Gastrointestinal haemorrhage
        )
    AND visit_occurrence.visit_concept_id IN (9201, 9203);
```

This is parameterized SQL which can be used by the `SqlRender` package. We use parameterized SQL so we do not have to pre-specify the names of the CDM and result schemas. That way, if we want to run the SQL on a different schema, we only need to change the parameter values; we do not have to change the SQL code. By also making use of translation functionality in `SqlRender`, we can make sure the SQL code can be run in many different environments.

```
library(SqlRender)
sql <- readSql("coxibVsNonselVsGiBleed.sql")
sql <- renderSql(sql,
                 cdmDatabaseSchema = cdmDatabaseSchema,
                 resultsDatabaseSchema = resultsDatabaseSchema)$sql
sql <- translateSql(sql, targetDialect = connectionDetails$dbms)$sql

connection <- connect(connectionDetails)
executeSql(connection, sql)
```

In this code, we first read the SQL from the file into memory. In the next line, we replace the two parameter names with the actual values. We then translate the SQL into the dialect appropriate for the DBMS we already specified in the `connectionDetails`. Next, we connect to the server, and submit the rendered and translated SQL.

If all went well, we now have a table with the events of interest. We can see how many events per type:

```
sql <- paste("SELECT cohort_definition_id, COUNT(*) AS count",
             "FROM @resultsDatabaseSchema.coxibVsNonselVsGiBleed",
             "GROUP BY cohort_definition_id")
sql <- renderSql(sql, resultsDatabaseSchema = resultsDatabaseSchema)$sql
sql <- translateSql(sql, targetDialect = connectionDetails$dbms)$sql

querySql(connection, sql)
```

```
#>   cohort_concept_id  count
#> 1                 1 184979
#> 2                 2 798752
#> 3                 3 635619
```

3.3 Extracting the data from the server

Now we can tell `CohortMethod` to define the cohorts based on our events, construct covariates, and extract all necessary data for our analysis.

Important: The target and comparator drug must not be included in the covariates, including any descendant concepts. If the `targetId` and `comparatorId` arguments represent real concept IDs, you can set the `excludeDrugsFromCovariates` argument of the `getDbCohortMethodData` function to TRUE and automatically the drugs and their descendants will be excluded from the covariates. However, if the `targetId` and `comparatorId` arguments do not represent concept IDs, such as in the example above, you will need to manually add the drugs and descendants to the `excludedCovariateConceptIds` of the covariate settings. In this example code we exclude all NSAIDs from the covariates by pointing to the concept ID of the NSAID class and specifying `addDescendantsToExclude = TRUE`.

```
nsaids <- 21603933

# Define which types of covariates must be constructed:
covSettings <- createDefaultCovariateSettings(excludedCovariateConceptIds = nsaids,
                                              addDescendantsToExclude = TRUE)

#Load data:
cohortMethodData <- getDbCohortMethodData(connectionDetails = connectionDetails,
                                          cdmDatabaseSchema = cdmDatabaseSchema,
                                          oracleTempSchema = resultsDatabaseSchema,
                                          targetId = 1,
```

```
                                          comparatorId = 2,
                                          outcomeIds = 3,
                                          studyStartDate = "",
                                          studyEndDate = "",
                                          exposureDatabaseSchema = resultsDatabaseSchema,
                                          exposureTable = "coxibVsNonselVsGiBleed",
                                          outcomeDatabaseSchema = resultsDatabaseSchema,
                                          outcomeTable = "coxibVsNonselVsGiBleed",
                                          cdmVersion = cdmVersion,
                                          excludeDrugsFromCovariates = FALSE,
                                          firstExposureOnly = TRUE,
                                          removeDuplicateSubjects = TRUE,
                                          restrictToCommonPeriod = FALSE,
                                          washoutPeriod = 180,
                                          covariateSettings = covSettings)
cohortMethodData
```

```
#> CohortMethodData object
#>
#> Treatment concept ID: 1
#> Comparator concept ID: 2
#> Outcome concept ID(s): 3
```

There are many parameters, but they are all documented in the `CohortMethod` manual. The `createDefaultCovariateSettings` function is described in the `FeatureExtraction` package. In short, we are pointing the function to the table created earlier and indicating which concept IDs in that table identify the target, comparator and outcome. We instruct that the default set of covariates should be constructed, including covariates for all conditions, drug exposures, and procedures that were found on or before the index date. To customize the set of covariates, please refer to the `FeatureExtraction` package vignette by typing `vignette("UsingFeatureExtraction", package="FeatureExtraction")`.

All data about the cohorts, outcomes, and covariates are extracted from the server and stored in the `cohortMethodData` object. This object uses the package `ff` to store information in a way that ensures R does not run out of memory, even when the data are large.

We can use the generic `summary()` function to view some more information of the data we extracted:

```
summary(cohortMethodData)
```

```
#> CohortMethodData object summary
#>
#> Treatment concept ID: 1
#> Comparator concept ID: 2
#> Outcome concept ID(s): 3
#>
#> Treated persons: 49102
#> Comparator persons: 348558
#>
#> Outcome counts:
#>   Event count Person count
#> 3       27636        18141
#>
#> Covariates:
#> Number of covariates: 54112
#> Number of non-zero covariate values: 161150531
```

3.3.1 Saving the data to file

Creating the `cohortMethodData` file can take considerable computing time, and it is probably a good idea to save it for future sessions. Because `cohortMethodData` uses `ff`, we cannot use R's regular save function. Instead, we'll have to use the `saveCohortMethodData()` function:

```
saveCohortMethodData(cohortMethodData, "coxibVsNonselVsGiBleed")
```

We can use the `loadCohortMethodData()` function to load the data in a future session.

3.3.2 Defining new users

Typically, a new user is defined as first time use of a drug (either target or comparator), and typically a washout period (a minimum number of days prior first use) is used to make sure it is truly first use. When using the `CohortMethod` package, you can enforce the necessary requirements for new use in three ways:

1. When creating the cohorts in the database, for example when using a cohort definition tool.
2. When loading the cohorts using the `getDbCohortMethodData` function, you can use the `firstExposureOnly`, `removeDuplicateSubjects`, `restrictToCommonPeriod`, and `washoutPeriod` arguments. (As shown in the example above).
3. When defining the study population using the `createStudyPopulation` function (see below) using the `firstExposureOnly`, `removeDuplicateSubjects`, `restrictToCommonPeriod`, and `washoutPeriod` arguments.

The advantage of option 1 is that the input cohorts are already fully defined outside of the `CohortMethod` package, and for example external cohort characterization tools can be used on the same cohorts used in this package. The advantage of options 2 and 3 is that it saves you the trouble of limiting to first use yourself, for example allowing you to directly use the `drug_era` table in the CDM. Option 2 is more efficient than 3, since only data for first use will be fetched, while option 3 is less efficient but allows you to compare the original cohorts to the study population.

4 Defining the study population

Typically, the exposure cohorts and outcome cohorts will be defined independently of each other. When we want to produce an effect size estimate, we need to further restrict these cohorts and put them together, for example by removing exposed subjects that had the outcome prior to exposure, and only keeping outcomes that fall within a defined risk window. For this we can use the `createStudyPopulation` function:

```
studyPop <- createStudyPopulation(cohortMethodData = cohortMethodData,
                                  outcomeId = 3,
                                  firstExposureOnly = FALSE,
                                  restrictToCommonPeriod = FALSE,
                                  washoutPeriod = 0,
                                  removeDuplicateSubjects = FALSE,
                                  removeSubjectsWithPriorOutcome = TRUE,
                                  minDaysAtRisk = 1,
                                  riskWindowStart = 0,
                                  addExposureDaysToStart = FALSE,
                                  riskWindowEnd = 30,
                                  addExposureDaysToEnd = TRUE)
```

Note that we've set `firstExposureOnly` and `removeDuplicateSubjects` to FALSE, and `washoutPeriod` to zero because we already filtered on these arguments when using the `getDbCohortMethodData` function. During loading we set `restrictToCommonPeriod` to FALSE, and we do the same here because we do not want

to force the comparison to restrict only to time when both drugs are recorded. We specify the outcome ID we will use, and that people with outcomes prior to the risk window start date will be removed. The risk window is defined as starting at the index date (`riskWindowStart = 0` and `addExposureDaysToStart = FALSE`), and the risk windows ends 30 days after exposure ends (`riskWindowEnd = 30` and `addExposureDaysToEnd = TRUE`). Note that the risk windows are truncated at the end of observation or the study end date. We also remove subjects who have no time at risk. To see how many people are left in the study population we can always use the `getAttritionTable` function:

```
getAttritionTable(studyPop)
```

```
#>                                                                description
#> 1                                                          Original cohorts
#> 2 First exp. only & removed subs in both cohorts & 180 days of obs. prior
#> 3                                                          No prior outcome
#> 4                                                 Have at least 1 days at risk
#>   treatedPersons comparatorPersons treatedExposures comparatorExposures
#> 1         104225             511194           184979              798752
#> 2          49102             348558            49102              348558
#> 3          47411             339992            47411              339992
#> 4          47377             339648            47377              339648
```

One additional filtering step that is often used is matching or trimming on propensity scores, as will be discussed next.

5 Propensity scores

The `CohortMethod` can use propensity scores to adjust for potential confounders. Instead of the traditional approach of using a handful of predefined covariates, `CohortMethod` typically uses thousands to millions of covariates that are automatically constructed based on conditions, procedures and drugs in the records of the subjects.

5.1 Fitting a propensity model

We can fit a propensity model using the covariates constructed by the `getDbcohortMethodData()` function:

```
ps <- createPs(cohortMethodData = cohortMethodData, population = studyPop)
```

The `createPs()` function uses the `Cyclops` package to fit a large-scale regularized logistic regression.

To fit the propensity model, `Cyclops` needs to know the hyperparameter value which specifies the variance of the prior. By default `Cyclops` will use cross-validation to estimate the optimal hyperparameter. However, be aware that this can take a really long time. You can use the `prior` and `control` parameters of the `createPs()` to specify `Cyclops` behavior, including using multiple CPUs to speed-up the cross-validation.

5.2 Propensity score diagnostics

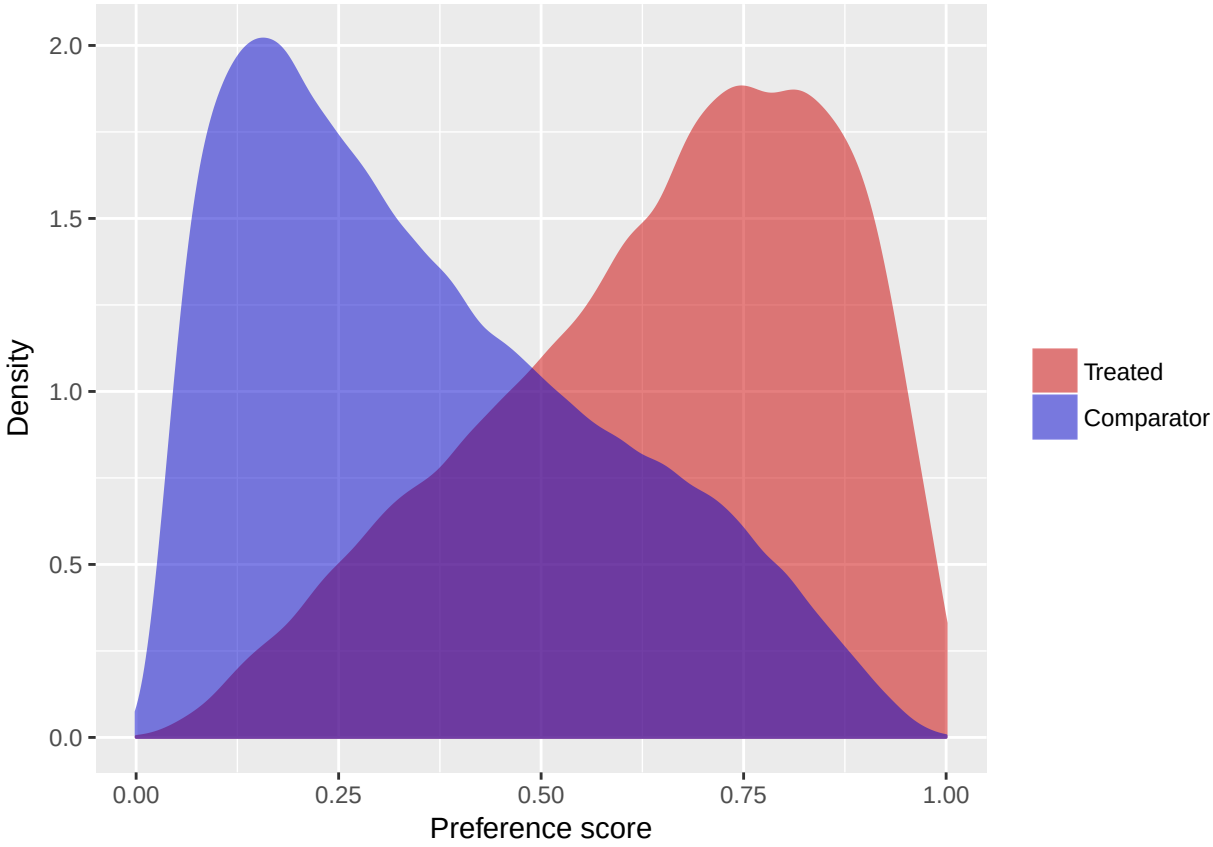
We can compute the area under the receiver-operator curve (AUC) for the propensity score model:

```
computePsAuc(ps)
```

```
#> [1] 0.8167895
```

We can also plot the propensity score distribution, although we prefer the preference score distribution:

```
plotPs(ps, scale = "preference")
```



It is also possible to inspect the propensity model itself by showing the covariates that have non-zero coefficients:

```
propensityModel <- getPsModel(ps, cohortMethodData)
head(propensityModel)
```

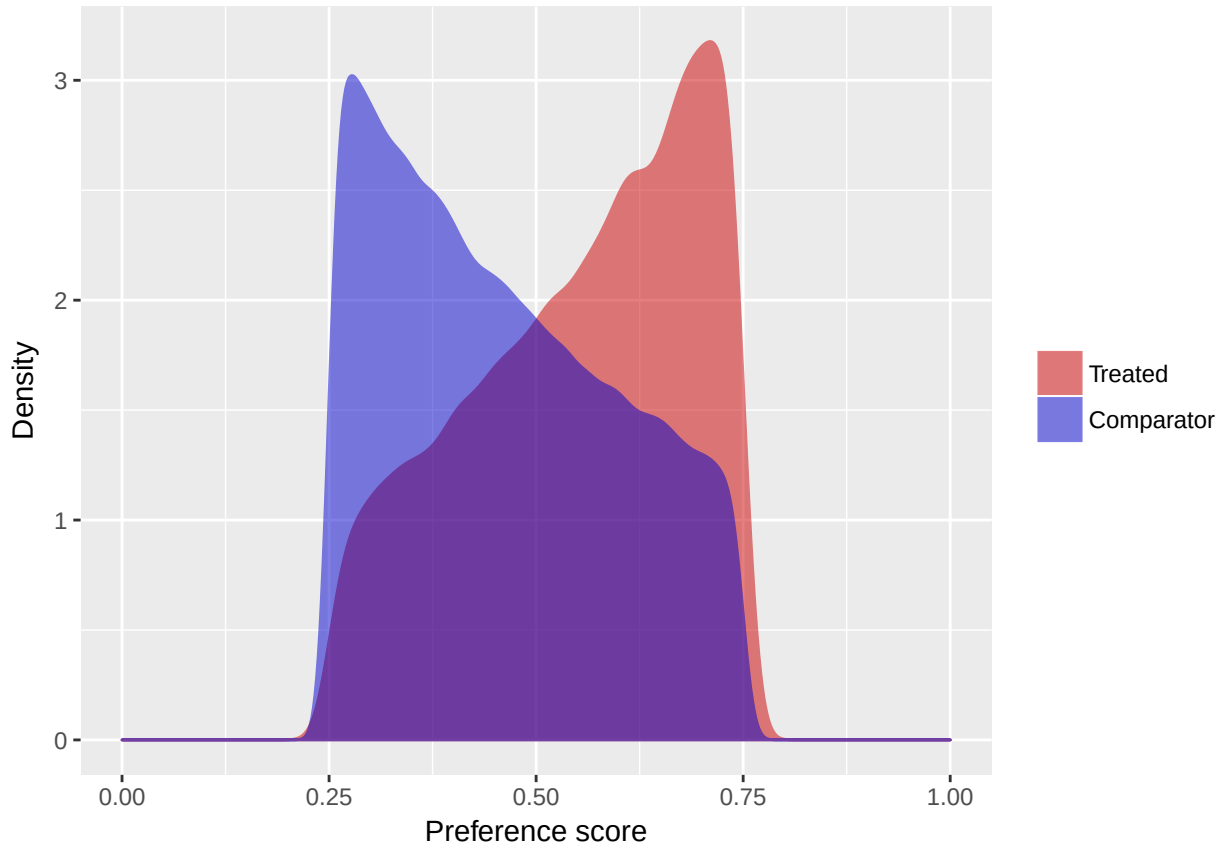
```
#>      coefficient         id                                covariateName
#> 27     1.4373697    2007006                           index year: 2007
#> 955    1.3397763 2101660504 ...s on knee joint; total knee arthroplasty
#> 2084   1.2042032 4253901210 ... to index: Juvenile rheumatoid arthritis
#> 5     -1.1650736       3003                           age group: 15-19
#> 28     1.0002664    2008006                           index year: 2008
#> 3     -0.9868114       2003                           age group: 10-14
```

One advantage of using the regularization when fitting the propensity model is that most coefficients will shrink to zero and fall out of the model. It is a good idea to inspect the remaining variables for anything that should not be there, for example variations of the drugs of interest that we forgot to exclude.

5.3 Using the propensity score

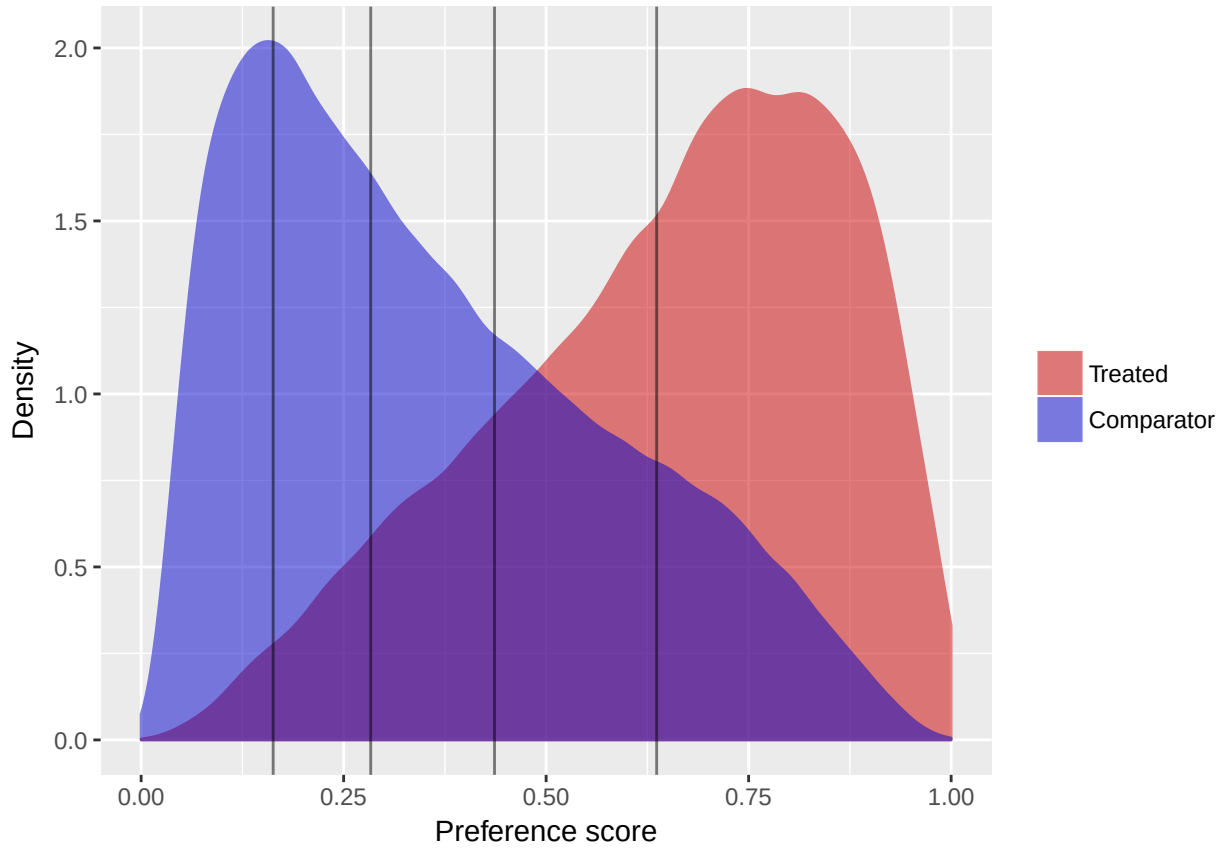
We can use the propensity scores to trim, stratify, match, or weigh our population. For example, one could trim to equipoise, meaning only subjects with a preference score between 0.25 and 0.75 are kept:

```
trimmedPop <- trimByPsToEquipoise(ps)
plotPs(trimmedPop, ps, scale = "preference")
```



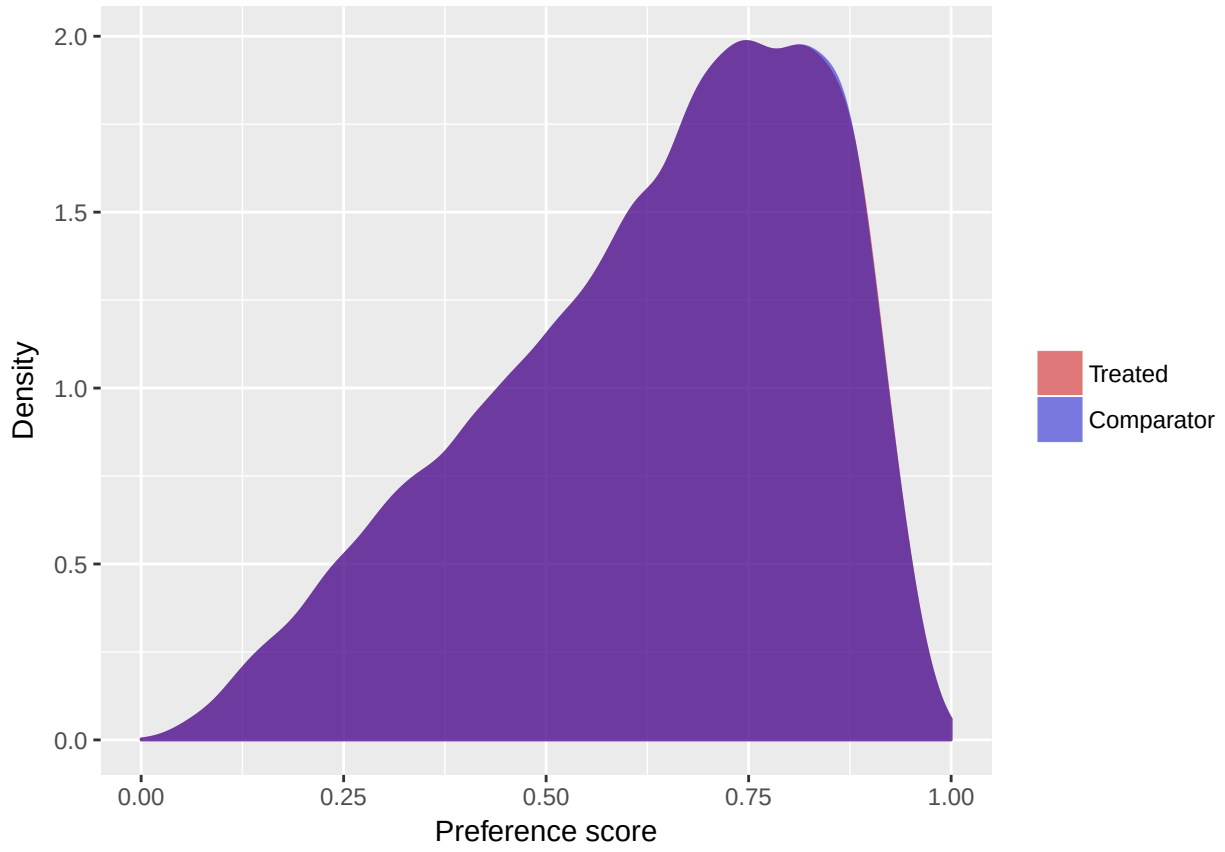
Instead (or additionally), we could stratify the population based on the propensity score:

```
stratifiedPop <- stratifyByPs(ps, numberOfStrata = 5)
plotPs(stratifiedPop, ps, scale = "preference")
```



We can also match subjects based on propensity scores. In this example, we're using one-to-one matching:

```
matchedPop <- matchOnPs(ps, caliper = 0.2, caliperScale = "standardized logit",
    maxRatio = 1)
plotPs(matchedPop, ps)
```



Note that for both stratification and matching it is possible to specify additional matching criteria such as age and sex using the `stratifyByPsAndCovariates()` and `matchOnPsAndCovariates()` functions, respectively.

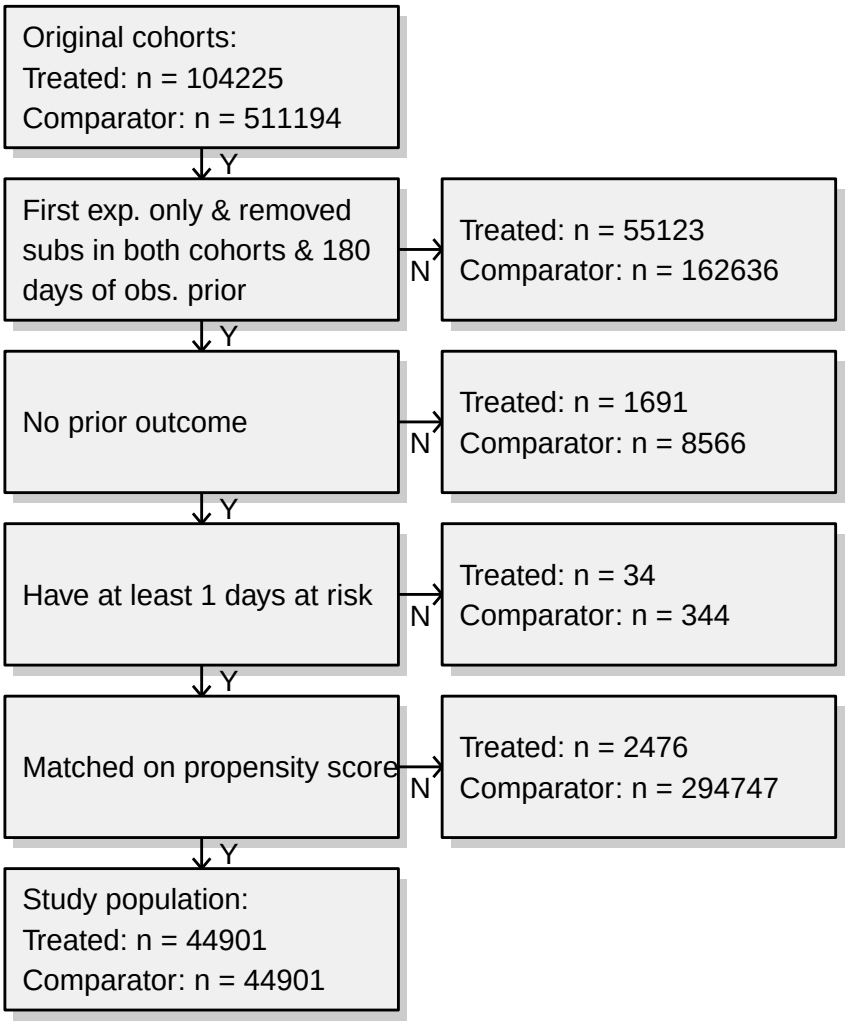
We can see the effect of trimming and/or matching on the population using the `getAttritionTable` function:

```
getAttritionTable(matchedPop)
```

```
#>                                                          description
#> 1                                                    Original cohorts
#> 2 First exp. only & removed subs in both cohorts & 180 days of obs. prior
#> 3                                                    No prior outcome
#> 4                                              Have at least 1 days at risk
#> 5                                               Matched on propensity score
#>   treatedPersons comparatorPersons treatedExposures comparatorExposures
#> 1         104225            511194           184979              798752
#> 2          49102            348558            49102              348558
#> 3          47411            339992            47411              339992
#> 4          47377            339648            47377              339648
#> 5          44901             44901            44901               44901
```

Or, if we like, we can plot an attrition diagram:

```
drawAttritionDiagram(matchedPop)
```



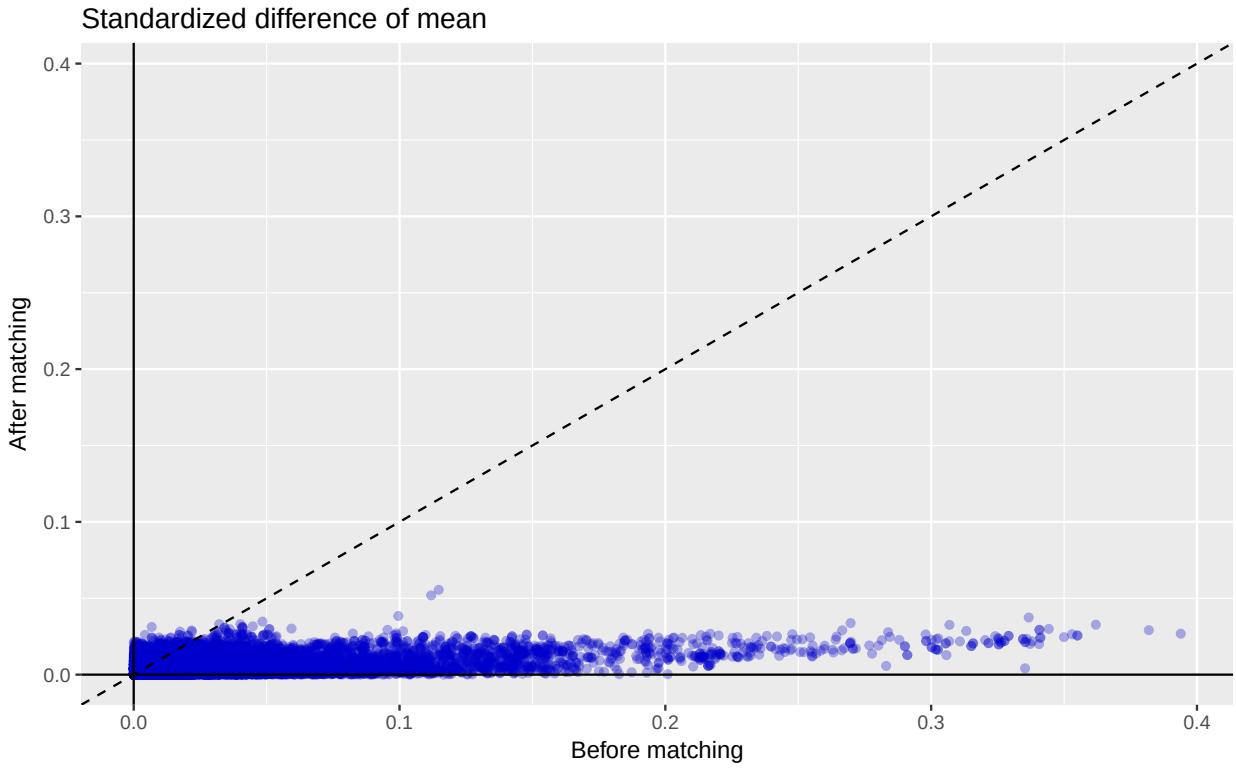
5.4 Evaluating covariate balance

To evaluate whether our use of the propensity score is indeed making the two cohorts more comparable, we can compute the covariate balance before and after trimming, matching, and/or stratifying:

```
balance <- computeCovariateBalance(matchedPop, cohortMethodData)
```

```
plotCovariateBalanceScatterPlot(balance)
```

```
#> Warning: Removed 10591 rows containing missing values (geom_point).
```




```
plotCovariateBalanceOfTopVariables(balance)
```

The 'before matching' population is the population as extracted by the `getDbCohortMethodData` function, so before any further filtering steps.

5.5 Inserting the population cohort in the database

For various reasons it might be necessary to insert the study population back into the database, for example because we want to use an external cohort characterization tool. We can use the `insertDbPopulation` function for this purpose:

```
insertDbPopulation(population = matchedPop,
                   cohortIds = c(101,100),
                   connectionDetails = connectionDetails,
                   cohortDatabaseSchema = resultsDatabaseSchema,
                   cohortTable = "coxibVsNonselVsGiBleed",
                   createTable = FALSE,
                   cdmVersion = cdmVersion)
```

This function will store the population in a table with the same structure as the `cohort` table in the CDM, in this case in the same table where we had created our original cohorts.

6 Outcome models

The outcome model is a model describing which variables are associated with the outcome.

6.1 Considering follow-up and power

Before we start fitting an outcome model, we might be interested to know whether we have sufficient power to detect a particular effect size. It makes sense to perform these power calculations once the study population has been fully defined, so taking into account loss to the various inclusion and exclusion criteria (such as no prior outcomes), and loss due to matching and/or trimming. Since the sample size is fixed in retrospective studies (the data has already been collected), and the true effect size is unknown, the CohortMethod package provides a function to compute the minimum detectable relative risk (MDRR) instead:

```
computeMdrr(population = studyPop,
            modelType = "cox",
            alpha = 0.05,
            power = 0.8,
            twoSided = TRUE)
```

```
#>   targetPersons comparatorPersons targetExposures comparatorExposures
#> 1         47377            339648           47377              339648
#>   targetDays comparatorDays totalOutcomes     mdrr         se
#> 1    6609395       23763641          1215 1.277903 0.08752912
```

In this example we used the `studyPop` object, so the population before any matching or trimming. If we want to know the MDRR after matching, we use the `matchedPop` object we created earlier instead:

```
computeMdrr(population = matchedPop,
            modelType = "cox",
            alpha = 0.05,
            power = 0.8,
            twoSided = TRUE)
```

```
#>   targetPersons comparatorPersons targetExposures comparatorExposures
#> 1         44901             44901           44901                44901
#>   targetDays comparatorDays totalOutcomes     mdrr        se
#> 1    6196468        3929370           442 1.305408 0.0951303
```

Even thought the MDRR in the matched population is higher, meaning we have less power, we should of course not be fooled: matching most likely eliminates confounding, and is therefore preferred to not matching.

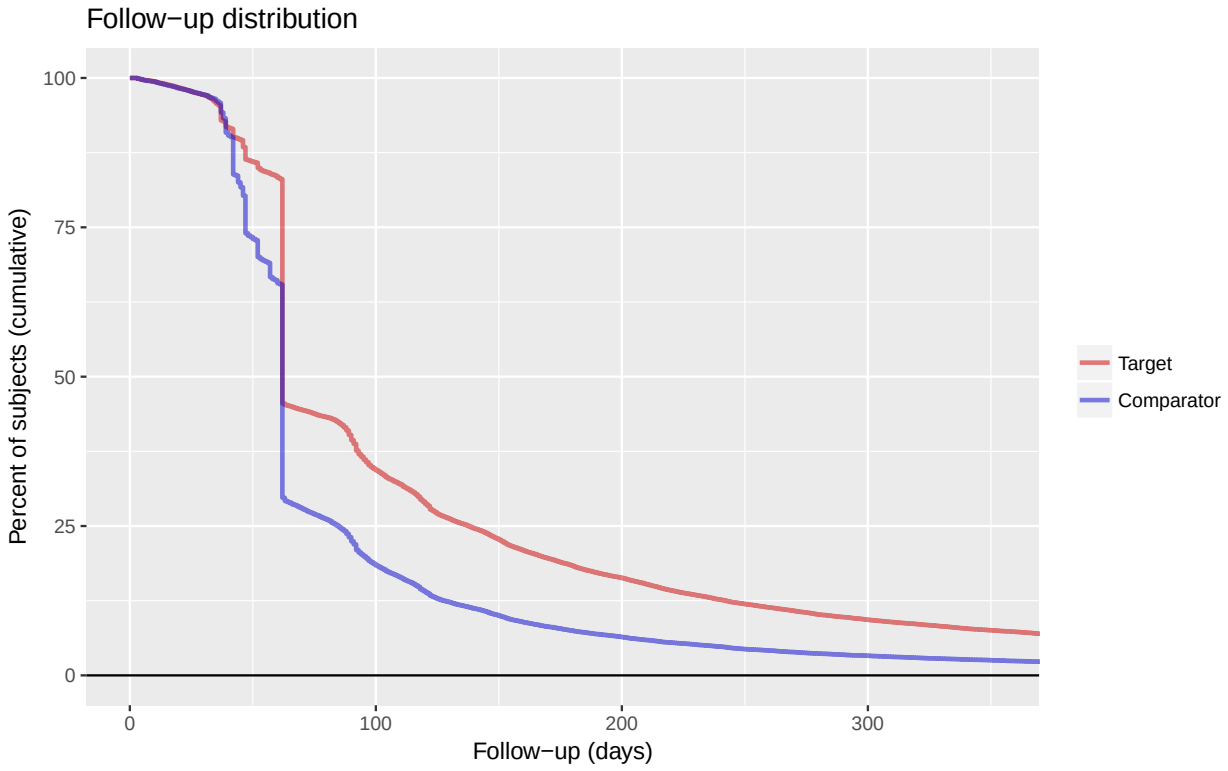
To gain a better understanding of the amount of follow-up available we can also inspect the distribution of follow-up time. We defined follow-up time as time at risk, so not censored by the occurrence of the outcome. The `getFollowUpDistribution` can provide a simple overview:

```
getFollowUpDistribution(population = matchedPop)
```

```
#>   100% 75% 50% 25%   0% Treatment
#> 1    2  61  61 137 3764         1
#> 2    2  46  61  84 3026         0
```

The output is telling us number of days of follow-up each quantile of the study population has. We can also plot the distribution:

```
plotFollowUpDistribution(population = matchedPop)
```



6.2 Fitting the outcome model

In theory we could fit an outcome model without using the propensity scores. In this example we are fitting an outcome model using a Cox regression:

```
outcomeModel <- fitOutcomeModel(population = studyPop,
                                modelType = "cox",
                                stratified = FALSE,
                                useCovariates = FALSE)
outcomeModel
```

```
#> Model type: cox
#> Stratified: FALSE
```

```
#> Use covariates: FALSE
#> Use inverse probability of treatment weighting: FALSE
#> Status: OK
#>
#>           Estimate lower .95 upper .95    logRr seLogRr
#> treatment 1.038993  0.899015  1.196863 0.038252   0.073
```

But of course we want to make use of the matching done on the propensity score:

```
outcomeModel <- fitOutcomeModel(population = matchedPop,
                                modelType = "cox",
                                stratified = TRUE,
                                useCovariates = FALSE)
outcomeModel
```

```
#> Model type: cox
#> Stratified: TRUE
#> Use covariates: FALSE
#> Use inverse probability of treatment weighting: FALSE
#> Status: OK
#>
#>           Estimate lower .95 upper .95    logRr seLogRr
#> treatment  0.80292   0.62382   1.03105 -0.21950  0.1282
```

Note that we define the sub-population to be only those in the `matchedPop` object, which we created earlier by matching on the propensity score. We also now use a stratified Cox model, conditioning on the propensity score match sets.

One final refinement would be to use the same covariates we used to fit the propensity model to also fit the outcome model. This way we are more robust against misspecification of the model, and more likely to remove bias. For this we use the regularized Cox regression in the `Cyclops` package. (Note that the treatment variable is automatically excluded from regularization.)

```
outcomeModel <- fitOutcomeModel(population = matchedPop,
                                cohortMethodData = cohortMethodData,
                                modelType = "cox",
                                stratified = TRUE,
                                useCovariates = TRUE)
outcomeModel
```

```
#> Model type: cox
#> Stratified: TRUE
#> Use covariates: TRUE
#> Use inverse probability of treatment weighting: FALSE
#> Status: OK
#> Prior variance: 0.0417214627426502
#>
#>           Estimate lower .95 upper .95    logRr seLogRr
#> treatment  0.85190   0.63285   1.14949 -0.16029  0.1523
```

6.3 Inpecting the outcome model

We can inspect more details of the outcome model:

```
summary(outcomeModel)
```

```
#> Model type: cox
```

```
#> Stratified: TRUE
#> Use covariates: TRUE
#> Use inverse probability of treatment weighting: FALSE
#> Status: OK
#> Prior variance: 0.0417214627426502
#>
#>           Estimate lower .95 upper .95    logRr seLogRr
#> treatment  0.85190   0.63285   1.14949 -0.16029  0.1523
#>
#> Population counts
#>       treatedPersons comparatorPersons treatedExposures
#> Count          44840             44840            44840
#>       comparatorExposures
#> Count               44840
#>
#> Outcome counts
#>       treatedPersons comparatorPersons treatedExposures
#> Count            245               196              245
#>       comparatorExposures
#> Count                 196
#>
#> Time at risk
#>      treatedDays comparatorDays
#> Days     6144026        3904722
```

```
exp(coef(outcomeModel))
```

```
#> [1] 0.8518951
```

```
exp(confint(outcomeModel))
```

```
#> [1] 0.6328451 1.1494942
```

We can also see the covariates that ended up in the outcome model:

```
fullOutcomeModel <- getOutcomeModel(outcomeModel, cohortMethodData)
head(fullOutcomeModel)
```

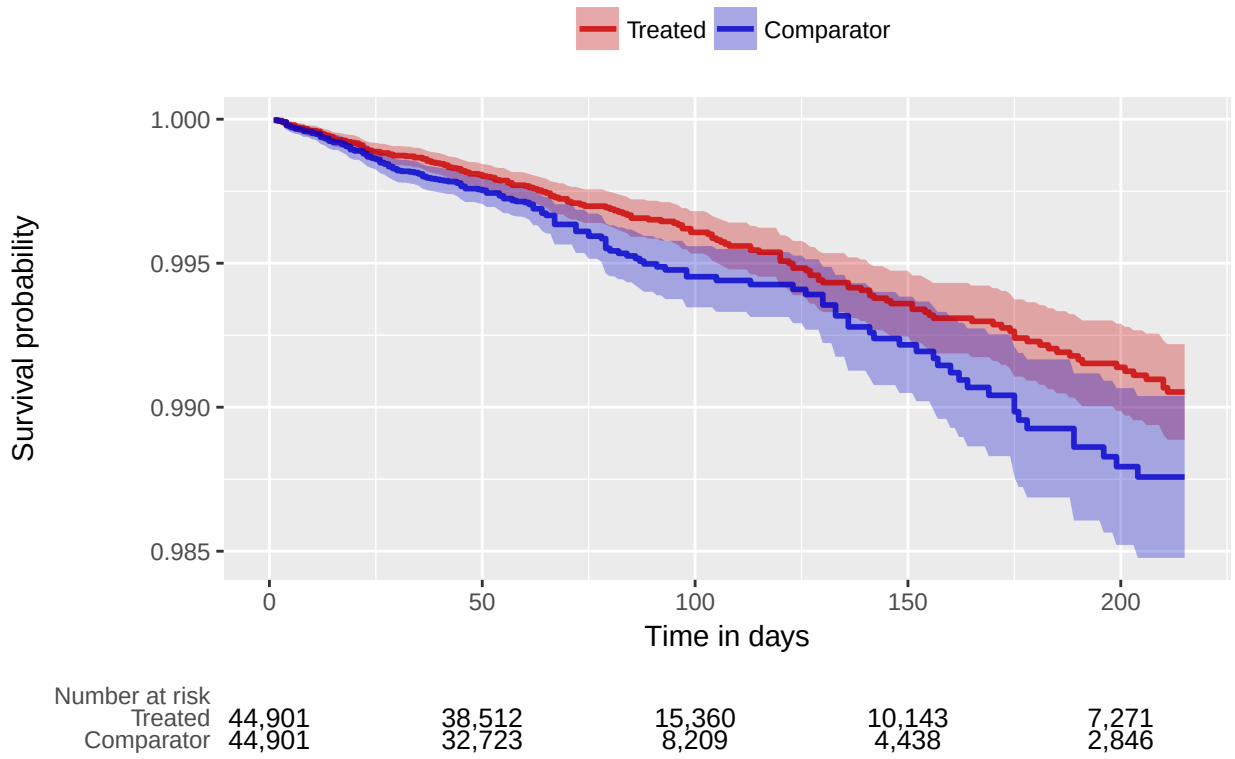
```
#>             coefficient          id
#> 198124210     0.5499190   198124210
#> 8507001       0.4828838     8507001
#> 2514408502    0.3252432  2514408502
#> 21601664412   0.3017819 21601664412
#> 4329041212    0.2805469  4329041212
#> 321588210     0.2762147   321588210
#>                                          covariateName
#> 198124210   ...0 days relative to index: Kidney disease
#> 8507001                                    gender = MALE
#> 2514408502  ...nation; Medical decision making of moder
#> 21601664412 ... relative to index: BETA BLOCKING AGENTS
#> 4329041212  ...0 through 0 days relative to index: Pain
#> 321588210   ... 0 days relative to index: Heart disease
```

6.4 Kaplan-Meier plot

We can create the Kaplan-Meier plot:

```
plotKaplanMeier(matchedPop, includeZero = FALSE)
```

```
#> Warning in plotKaplanMeier(matchedPop, includeZero = FALSE): The population
#> has strata, but the stratification is not visible in the plot
```



7 Acknowledgments

Considerable work has been dedicated to provide the `CohortMethod` package.

```
citation("CohortMethod")
```

```
#>
#> To cite package 'CohortMethod' in publications use:
#>
#>   Martijn J. Schuemie, Marc A. Suchard and Patrick B. Ryan (2017).
#>   CohortMethod: New-user cohort method with large scale propensity
#>   and outcome models. R package version 2.4.4.
#>
#> A BibTeX entry for LaTeX users is
#>
#>   @Manual{,
#>     title = {CohortMethod: New-user cohort method with large scale propensity and outcome models},
#>     author = {Martijn J. Schuemie and Marc A. Suchard and Patrick B. Ryan},
#>     year = {2017},
#>     note = {R package version 2.4.4},
```

```
#>   }
#>
#> ATTENTION: This citation information has been auto-generated from
#> the package DESCRIPTION file and may need manual editing, see
#> 'help("citation")'.
```

Further, `CohortMethod` makes extensive use of the `Cyclops` package.

```
citation("Cyclops")
```

```
#>
#> To cite Cyclops in publications use:
#>
#> Suchard MA, Simpson SE, Zorych I, Ryan P and Madigan D (2013).
#> "Massive parallelization of serial inference algorithms for
#> complex generalized linear models." _ACM Transactions on Modeling
#> and Computer Simulation_, *23*, pp. 10. <URL:
#> http://dl.acm.org/citation.cfm?id=2414791>.
#>
#> A BibTeX entry for LaTeX users is
#>
#>   @Article{,
#>     author = {M. A. Suchard and S. E. Simpson and I. Zorych and P. Ryan and D. Madigan},
#>     title = {Massive parallelization of serial inference algorithms for complex generalized linear m
#>     journal = {ACM Transactions on Modeling and Computer Simulation},
#>     volume = {23},
#>     pages = {10},
#>     year = {2013},
#>     url = {http://dl.acm.org/citation.cfm?id=2414791},
#>   }
```

This work is supported in part through the National Science Foundation grant IIS 1251151.