vignette PlasmodeSim

2022-10-19

Welcome to the vignette about the R package PlasmodeSim. This package is still under development.

installing plasmodeSim using remotes

To install using `remotes` run:

```
#install.packages("remotes")
#remotes::install_github("GidiusVanDeKamp/PlasmodeSim")
```

Setting up

This documents skips some parts, we have skipped the steps to obtain the plpResults and the plpData.

```
# library(dplyr)
# library(PlasmodeSim)

modelSettings <- PatientLevelPrediction::setLassoLogisticRegression()

plpResultLogistic <- PatientLevelPrediction::loadPlpResult(
  "~/R/internshipErasmusMC/simulate-new-patients-outcomes/plp_demolog/Result")

plpData<- PatientLevelPrediction::loadPlpData(
  "~/R/internshipErasmusMC/simulate-new-patients-outcomes/plp_demolog/Data" )
```

Example 1

In this example we obtain new outcomes of a fitted logistic model.

```
plpModelLog <- plpResultLogistic$model
probabilites <- PlasmodeSim::newPropsParametersPlpModel(plpModelLog,
                                                        plpData,
                                                        plpData$cohorts)
```

```
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.198 secs
## Prediction took 0.179 secs
```

The function predictPlp returned this information.

```
newOut <- PlasmodeSim::newOutcomes(200, probabilites)
head(newOut)
```

```
##   rowId newOutcomes
## 1  2370           1
## 2  1155           1
## 3   476           0
## 4   486           0
## 5  1590           0
## 6   304           0
```

In the output of newOut patients are drawed randomly with the same chance, the patients could be drawed multiple times. If this happens they can have a different outcome. The function `newOutcomes` needs a data set where the column that contains the probabilities is called `value`.

Example 2

We here we show how to simulate new outcomes from an unfitted logistic model.

```
Parameters <- plpModelLog$model$coefficients
UnfittedParameters <- Parameters
UnfittedParameters[1,1] <- -0.4
UnfittedParameters[2:4,1] <- 0.4
head(UnfittedParameters)
```

```
##     betas covariateIds
## 1 -0.4000  (Intercept)
## 2  0.4000         6003
## 3  0.4000         8003
## 4  0.4000         9003
## 5  0.0092      8507001
## 6  0.0000     28060210
```

For the logistic model it is necessary that the parameters are stored in a dataset with a column called `betas` and a column called `covariateIds`.

```
plpModelunfitted <- PlasmodeSim::makeLogisticModel(UnfittedParameters)
newprobs <- PatientLevelPrediction::predictPlp(plpModelunfitted,
                                               plpData,
                                               plpData$cohorts)
```

```
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.171 secs
## Prediction took 0.176 secs
```

```
newOut <- PlasmodeSim::newOutcomes(200, newprobs)
head(newOut)
```

```
##   rowId newOutcomes
## 1   684           1
```

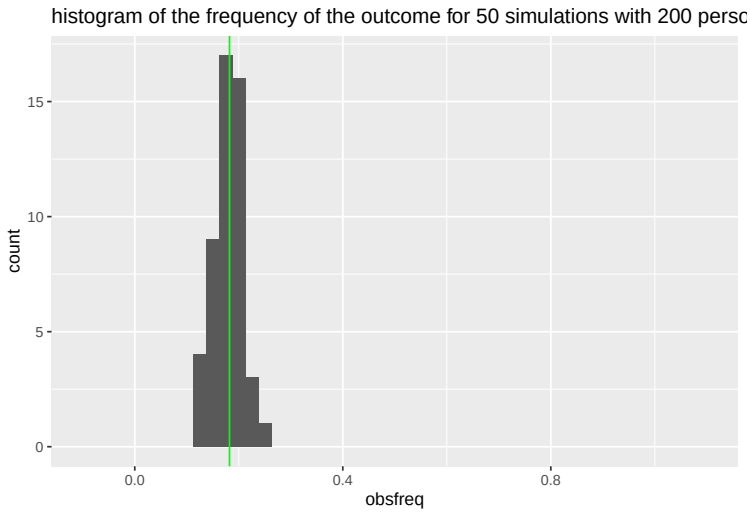
```
## 2  2608           0
## 3  1450           1
## 4  1596           1
## 5  2143           1
## 6  1273           0
```

Visual simulations

The function `visualOutcome` simulated new data and then plots the frequency of the outcome. Right now the function `visualOutcome` only works for a logistic model. The green line in the plots is the average outcome in the original dataset.

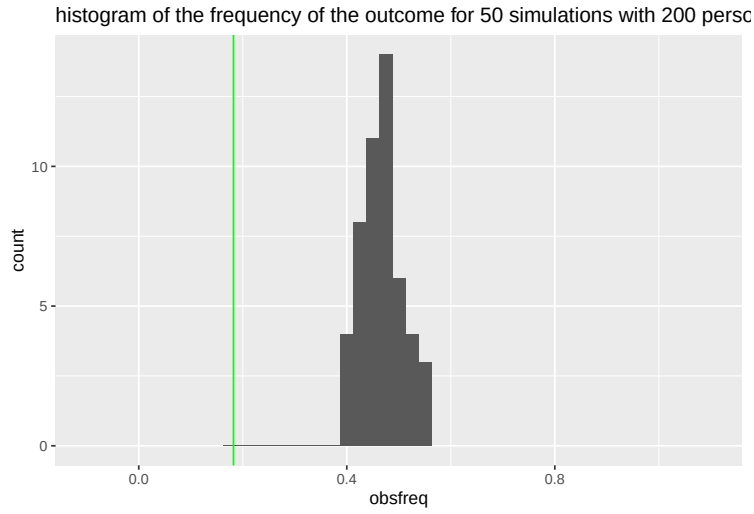
```
PlasmodeSim::visualOutcome(plpData,50,200,Parameters)
```

```
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.175 secs
## Prediction took 0.169 secs
```




```
PlasmodeSim::visualOutcome(plpData,50,200,UnfittedParameters)
```

```
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.177 secs
## Prediction took 0.177 secs
```



Here we have plotted 50 times the frequency of the outcome for a simulated dataset with 200 people.

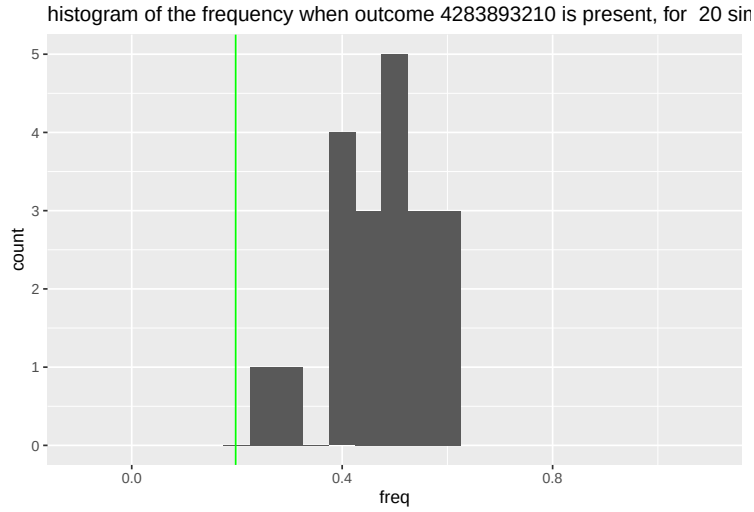
Visual of a specific covariate

```
covariateIdToStudy<- plpResultLogistic$covariateSummary$covariateId[3]
UnfittedParameters[3,]
```

```
##   betas covariateIds
## 3   0.4         8003
```

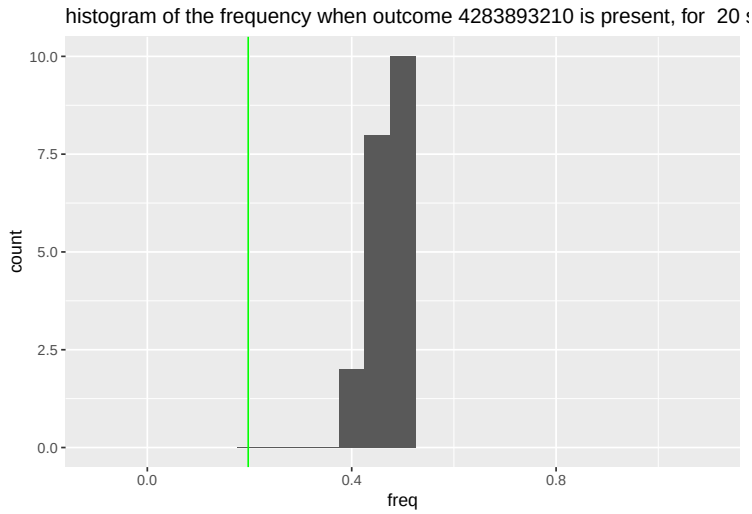
```
PlasmodeSim::visualOutcomeCovariateId(plpData,
                                      covariateIdToStudy,
                                      20,
                                      200,
                                      UnfittedParameters)
```

```
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.199 secs
## Prediction took 0.186 secs
```



```
PlasmodeSim::visualOutcomeCovariateId2(plpData,
                                       covariateIdToStudy,
                                       20,
                                       200,
                                       UnfittedParameters)
```

```
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.179 secs
## Prediction took 0.174 secs
```



As one can see `visualOutcomeCovariateId` and `visualOutcomeCovariateId2` are very similiar, they both calculate and plot the frequency for a group with a specific covariate present. The small difference is that `visualOutcomeCovariateId` filters a newly simulated dataset set to only keep the patients where the covariate is present, and `visualOutcomeCovariateId2` only simulates new outcomes for patients that have the covariate present. We see they are almost the identical only `visualOutcomeCovariateId2` is spread out less because the groups for calculating the frequency with are bigger.

Survival times outcomes.

first we make a training set.

```
connectionDetails <- Eunomia::getEunomiaConnectionDetails()

# create the database
Eunomia::createCohorts(
  connectionDetails = connectionDetails,
  cdmDatabaseSchema = 'main',
  cohortDatabaseSchema = 'main',
  cohortTable = 'cohort'
)
```

```
## Creating cohort: Celecoxib
##   |                                                                      |
```

```
## Creating cohort: Diclofenac
##   |                                                                      |
## Creating cohort: GiBleed
##   |                                                                      |
## Creating cohort: NSAIDs
##   |                                                                      |
## Cohorts created in table main.cohort

##   cohortId       name
## 1        1  Celecoxib
## 2        2 Diclofenac
## 3        3    GiBleed
## 4        4     NSAIDs
##                                                                                 description
## 1    A simplified cohort definition for new users of celecoxib, designed specifically for Eunomia.
## 2    A simplified cohort definition for new users ofdiclofenac, designed specifically for Eunomia.
## 3 A simplified cohort definition for gastrointestinal bleeding, designed specifically for Eunomia.
## 4       A simplified cohort definition for new users of NSAIDs, designed specifically for Eunomia.
##   count
## 1  1844
## 2   850
## 3   479
## 4  2694
```

```r
# ------------------------------------------------------------------------------
# Points PatientLevelPredictionPackage to the Eunomia database
# Tells Eunomia to extract the cohort stored with id = 4 as the target cohort
# and cohort with id = 3 as the outcome cohort. The other settings (...Schema)
# tell the database where to look for the target and the outcome cohorts
# ------------------------------------------------------------------------------
databaseDetails <- PatientLevelPrediction::createDatabaseDetails(
  connectionDetails = connectionDetails,
  cdmDatabaseId = "eunomia",
  cdmDatabaseSchema = 'main',
  cdmDatabaseName = 'Eunomia',
  cohortDatabaseSchema = 'main',
  cohortTable = 'cohort',
  target = 4,
  outcomeDatabaseSchema = 'main',
  outcomeTable = 'cohort',
  outcomeId = 3,
  cdmVersion = 5
)

# Use ?FeatureExtraction::createCovariateSettings to see what the options are
# There are a lot...
covariateSettings <- FeatureExtraction::createCovariateSettings(
  useDemographicsGender = TRUE,
  useDemographicsAgeGroup = TRUE,
  useConditionGroupEraLongTerm = TRUE,
  useDrugGroupEraLongTerm = TRUE,
  endDays = -1,
  longTermStartDays = -365
)
```

```
restrictPlpDataSettings <- PatientLevelPrediction::createRestrictPlpDataSettings(
  studyStartDate = '20000101',
  studyEndDate = '20200101',
  firstExposureOnly = TRUE,
  washoutPeriod = 30
)

# issue with studyStartDate/studyEndDate
restrictPlpDataSettings <- PatientLevelPrediction::createRestrictPlpDataSettings(
  firstExposureOnly = TRUE,
  washoutPeriod = 30
)

plpData <- PatientLevelPrediction::getPlpData(
  databaseDetails = databaseDetails,
  covariateSettings = covariateSettings,
  restrictPlpDataSettings = restrictPlpDataSettings
)
```

```
##   |                                                                      |
```

```
## Warning: The 'oracleTempSchema' argument is deprecated. Use 'tempEmulationSchema' instead.
## This warning is displayed once every 8 hours.
```

```
## Constructing features on server
##   |                                                                      |
## Fetching data from server
## Fetching data took 0.177 secs
```

For simulating new censored survival times we need more than one probability, we use the baselinehazard, stored in the plpModel, we also need a model for the censoring. First we make the traing set.

```
# plpData <- PatientLevelPrediction::loadPlpData(
#   "~/R/internshipErasmusMC/simulate-new-patients-outcomes/plp_democox/Data" )
#


populationSettings <- PatientLevelPrediction::createStudyPopulationSettings(
  binary = TRUE,
  includeAllOutcomes = FALSE,
  firstExposureOnly = FALSE,
  washoutPeriod = 180,
  removeSubjectsWithPriorOutcome = FALSE,
  priorOutcomeLookback = 99999,
  requireTimeAtRisk = TRUE,
  minTimeAtRisk = 1,
  riskWindowStart = 1,
  startAnchor = 'cohort start',
  riskWindowEnd = 7300,
  endAnchor = 'cohort start'
)
executeSettings <- PatientLevelPrediction::createExecuteSettings(
```

```
  runSplitData = TRUE,
  runSampleData = FALSE,
  runfeatureEngineering = FALSE,
  runPreprocessData = TRUE,
  runModelDevelopment = TRUE,
  runCovariateSummary = TRUE
)
splitSettings <- PatientLevelPrediction::createDefaultSplitSetting(
  testFraction = 0.25,
  trainFraction = 0.75,
  splitSeed = 123,
  nfold = 3,
  type = 'stratified'
)
sampleSettings <- PatientLevelPrediction::createSampleSettings(
  type = 'none'
)
featureEngineeringSettings <-
  PatientLevelPrediction::createFeatureEngineeringSettings(
  type = 'none'
)
preprocessSettings <- PatientLevelPrediction::createPreprocessSettings(
  minFraction = 0,
  normalize = TRUE,
  removeRedundancy = TRUE
)
TrainingSet <- PlasmodeSim::MakeTraingSet(
  plpData = plpData,
  executeSettings = executeSettings,
  populationSettings = populationSettings,
  splitSettings = splitSettings,
  sampleSettings = sampleSettings,
  preprocessSettings = preprocessSettings,
  featureEngineeringSettings = featureEngineeringSettings,
  outcomeId = 3
)
```

```
## Outcome is 0 or 1
## seed: 123
## Creating a 25% test and 75% train (into 3 folds) random stratified split by class
## Data split into 656 test cases and 1974 train cases (658, 658, 658)
## Train Set:
## Fold 1 658 patients with 120 outcomes - Fold 2 658 patients with 120 outcomes - Fold 3 658 patients w
## 103 covariates in train data
## Test Set:
## 656 patients with 119 outcomes
## Removing 2 redundant covariates
## Normalizing covariates
## Tidying covariates took 0.523 secs
## Train Set:
## Fold 1 658 patients with 120 outcomes - Fold 2 658 patients with 120 outcomes - Fold 3 658 patients w
## 101 covariates in train data
## Test Set:
```

```
## 656 patients with 119 outcomes
```

Fit the model

To fit the model we make the modelsettings and we run the function `fitModelWithCensoring`.

```
modelSettings <- PatientLevelPrediction::setCoxModel()

fitCensor <- PlasmodeSim::fitModelWithCensoring(
  Trainingset = TrainingSet$Train,
  modelSettings = modelSettings
  # now i have only one model setting
  # should i change this to two seperate settings
)
```

```
## Running Cyclops
## Done.
## GLM fit status:  OK
## Creating variable importance data frame
## Prediction took 0.134 secs
## Running Cyclops
## Done.
## GLM fit status:  OK
## Creating variable importance data frame
## Prediction took 0.139 secs
```

Generate new outcomes from a population.

```
population <- PatientLevelPrediction::createStudyPopulation(
  plpData = plpData,
  outcomeId = 3,
  populationSettings = populationSettings
)
```

```
## Outcome is 0 or 1
```

```
NewOutcomes <- PlasmodeSim::simulateSurvivaltimesWithCensoring(
  censorModel = fitCensor,
  plpData = plpData,
  population = population,
  populationSettings = populationSettings,
  numberToSimulate = 10
)
```

```
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.182 secs
## Prediction took 0.28 secs
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.179 secs
## Prediction took 0.272 secs
```

```
head(NewOutcomes)
```

```
##   rowId survivalTime outcomeCount
## 1  2367           18            1
## 2   500         7300            0
## 3  2086           20            1
## 4  2279         7300            0
## 5   287         7300            0
## 6   290         5862            0
```

```
#for simulation the uncensored data

newdata <- PlasmodeSim::simulateSurvivaltimes(
  plpModel = fitCensor$outcomesModel,
  plpData = plpData,
  numberToSimulate = 10,
  population = population,
  populationSettings = populationSettings
)
```

```
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.177 secs
## Prediction took 0.272 secs
```

```
head(newdata)
```

```
##   rowId outcome
## 1  1961    7300
## 2  1134    7300
## 3  1720    7300
## 4  2369    7300
## 5   418    7300
## 6  2599    7300
```

Make an unfitted model

```
# makeCoxModel.

plpModel <- fitCensor$outcomesModel
coeff <- plpModel$model$coefficients
survival <- plpModel$model$baselineSurvival$surv
times <- plpModel$model$baselineSurvival$time

unfittedmodel <- PlasmodeSim::makeCoxModel(
  coefficients = coeff,
  baselinehazard = survival,
  timesofbaselinhazard = times,
  featureEngineering = NULL #  = NULL is the standart setting.
)
```

```
newdata <- PlasmodeSim::simulateSurvivaltimes(
  plpModel = unfittedmodel,
  plpData = plpData,
  numberToSimulate = 10,
  population = population,
  populationSettings = populationSettings
)
```

```
## Prediction took 0.191 secs
```

```
head(newdata)
```

```
##   rowId outcome
## 1    54      67
## 2  1229    7300
## 3   552    7300
## 4  2376    7300
## 5  1447    7300
## 6  1709      30
```

Make an unfitted model with censoring

```
#we can swap outcomes with censoring.
unfittedcensor<- list(censorModel = unfittedmodel,
                      outcomesModel = fitCensor$outcomesModel)

NewOutcomes <- PlasmodeSim::simulateSurvivaltimesWithCensoring(
  censorModel = unfittedcensor,
  plpData = plpData,
  population =  population,
  populationSettings = populationSettings,
  numberToSimulate = 200
)
```

```
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.172 secs
## Prediction took 0.272 secs
## Prediction took 0.19 secs
```

```
head(NewOutcomes)
```

```
##   rowId survivalTime outcomeCount
## 1   658           51            0
## 2  1864         7300            0
## 3  1317         7300            0
## 4  1193           86            1
## 5  1518         7300            0
## 6  1883         7300            0
```

Adjust the BaselineSurvival

```
adjustedModel <- PlasmodeSim::AdjustBaselineSurvival(
  plpModel = plpModel,
  TrainingSet = TrainingSet$Train,
  plpData = plpData,
  populationSettings = populationSettings,
  timeTofixat = 3592,
  proptofixwith = 0.87,
  intervalSolution= c(-100,100)
)
```

```
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.183 secs
## Prediction took 0.256 secs
```

```
NewOutcomes <- PlasmodeSim::simulateSurvivaltimesWithCensoring(
  censorModel = list(censorModel = fitCensor$outcomesModel,
                     outcomesModel = adjustedModel),
  plpData = plpData,
  population =  population,
  populationSettings = populationSettings,
  numberToSimulate = 2000
)
```

```
## Prediction took 0.197 secs
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.177 secs
## Prediction took 0.27 secs
```

```
head(NewOutcomes)
```

```
##   rowId survivalTime outcomeCount
## 1   705           56            1
## 2   306           14            1
## 3  1066         7300            0
## 4   678         7300            0
## 5   532           64            0
## 6  1942           58            1
```

plotting the survival

the function `kaplanMeierPlot` visualised the kamplanmeier estimate of a given dataset. It works with ggplot. we can easily compare the simulated data sets with the real dataset by putting them in one plot. For the true data set we set the colour to red.

```
NewOutcomes <- PlasmodeSim::simulateSurvivaltimesWithCensoring(
  censorModel = fitCensor,
  plpData = plpData,
```

```
  population = population,
  populationSettings = populationSettings,
  numberToSimulate = 2000
)
```

```
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.178 secs
## Prediction took 0.262 secs
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.174 secs
## Prediction took 0.272 secs
```

```
NewOutcomes2 <- PlasmodeSim::simulateSurvivaltimesWithCensoring(
  censorModel = fitCensor,
  plpData = plpData,
  population = population,
  populationSettings = populationSettings,
  numberToSimulate = 2000
)
```

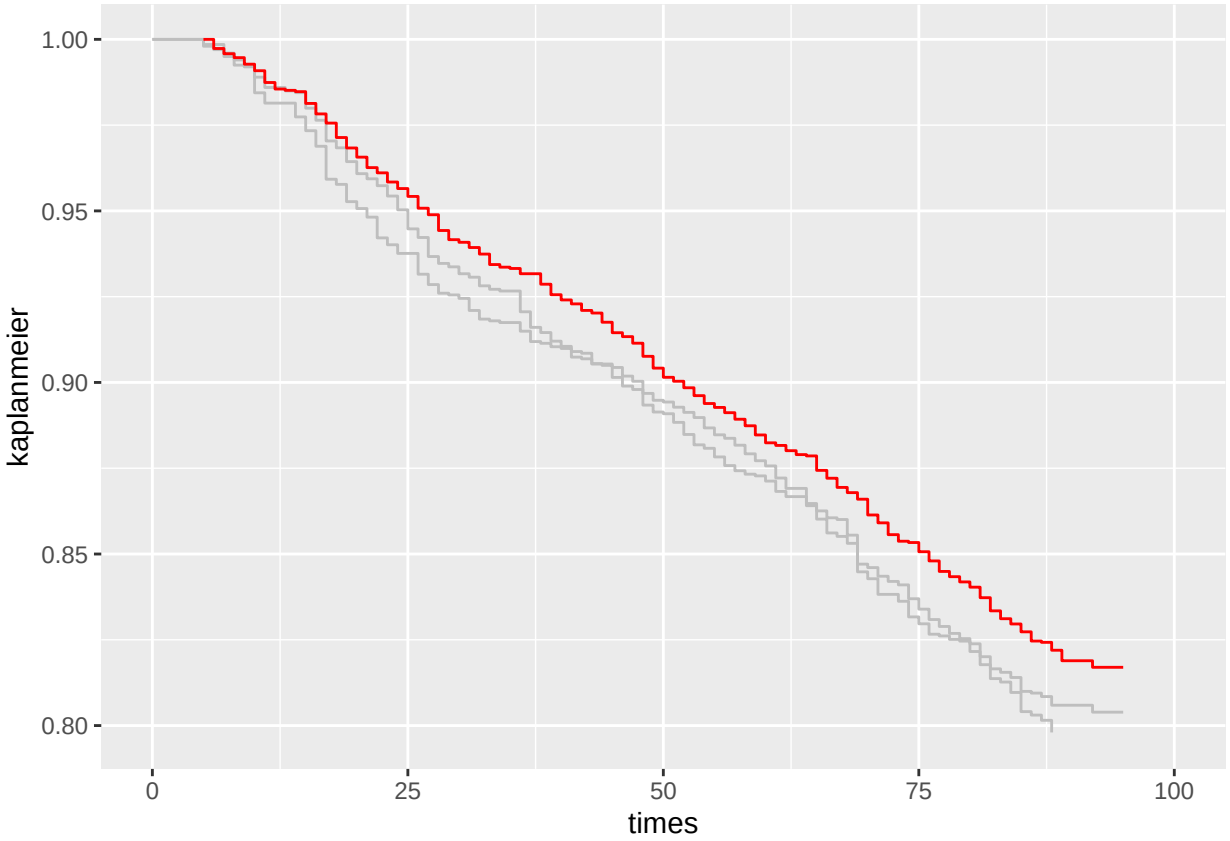
```
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.18 secs
## Prediction took 0.268 secs
## Removing infrequent and redundant covariates and normalizing
## Removing infrequent and redundant covariates covariates and normalizing took 0.179 secs
## Prediction took 0.269 secs
```

```
ggplot2::ggplot()+
  PlasmodeSim::KaplanMeierPlot( NewOutcomes )+
  PlasmodeSim::KaplanMeierPlot( NewOutcomes2 )+
  PlasmodeSim::KaplanMeierPlot( population, colour = 'red' )+
  ggplot2::xlim(c(0,100))
```

```
## Warning: Removed 542 rows containing missing values (‘geom_step()‘).

## Warning: Removed 558 rows containing missing values (‘geom_step()‘).

## Warning: Removed 1036 rows containing missing values (‘geom_step()‘).
```


1.00
0.95
0.90
0.85
0.80
kaplanmeier
0
25
50
75
100
times