```r
#* Script name: cSTM_time_indep.R
# Appendix code to time-independent cSTMs in R ----

#* This code forms the basis for the state-transition model of the
tutorial:
#* 'An Introductory Tutorial to Cohort State-Transition Models in R for
#* Cost-Effectiveness Analysis'
#* Authors:
#* - Fernando Alarid-Escudero <fernando.alarid@cide.edu>
#* - Eline Krijkamp
#* - Eva A. Enns
#* - Alan Yang
#* - M.G. Myriam Hunink
#* - Petros Pechlivanoglou
#* - Hawre Jalal
#* Please cite the article when using this code
#*
#* To program this tutorial we used:
#* R version 4.0.5 (2021-03-31)
#* Platform: 64-bit operating system, x64-based processor
#* Running under: Mac OS 12.2.1
#* RStudio: Version 1.4.1717 2009-2021 RStudio, Inc

#* Implements a time-independent Sick-Sicker cSTM model that evaluates
four
#* strategies:
#* - Standard of Care (SoC): best available care for the patients with the
#*   disease. This scenario reflects the natural history of the disease
#*   progression.
#* - Strategy A: treatment A is given to patients in the Sick and Sicker
states,
#*   but does only improves the quality of life  of those in the Sick
state.
#* - Strategy B: treatment B is given to all sick patients and reduces
disease
#*   progression from the Sick to Sicker state.
#* - Strategy AB: This strategy combines treatment A and treatment B. The
disease
#*   progression is reduced and individuals in the Sick state have an
improved
#*   quality of life.

#******************************************************************************#
# Initial setup ----
rm(list = ls())    # remove any variables in R's memory

## Install required packages ----
# install.packages("dplyr")      # to manipulate data
# install.packages("tidyr")      # to manipulate data
# install.packages("reshape2")   # to manipulate data
# install.packages("ggplot2")    # to visualize data
# install.packages("ggrepel")    # to visualize data
# install.packages("ellipse")    # to visualize data
# install.packages("scales")     # for dollar signs and commas
# install.packages("dampack")    # for CEA and calculate ICERs
# install.packages("devtools")   # to install packages from GitHub
# devtools::install_github("DARTH-git/darthtools") # to install darthtools
from GitHub using devtools
```

```r
# install.packages("doParallel") # to handle parallel processing

## Load packages ----
library(dplyr)
library(tidyr)
library(reshape2)   # For melting data
library(ggplot2)    # For plotting
library(ggrepel)    # For plotting
library(ellipse)    # For plotting
library(scales)     # For dollar signs and commas
# library(dampack)  # Uncomment to use CEA and PSA visualization
functionality from dampack instead of the functions included in this
repository
# library(darthtools) # Uncomment to use WCC, parameter transformation,
and matrix checks from darthtools instead of the functions included in
this repository
# library(doParallel) # For running PSA in parallel

## Load supplementary functions ----
source("R/Functions.R")

# Model input ----
## General setup ----
cycle_length <- 1       # cycle length equal to one year (use 1/12 for
monthly)
n_age_init <- 25        # age at baseline
n_age_max  <- 100       # maximum age of follow up
n_cycles <- (n_age_max - n_age_init)/cycle_length # time horizon, number
of cycles
v_names_states <- c("H",  # the 4 health states of the model:
                    "S1", # Healthy (H), Sick (S1), Sicker (S2), Dead (D)
                    "S2",
                    "D")

n_states <- length(v_names_states)     # number of health states

### Discounting factors ----
d_c <- 0.03 # annual discount rate for costs
d_e <- 0.03 # annual discount rate for QALYs

### Strategies ----
v_names_str <- c("Standard of care",      # store the strategy names
                 "Strategy A",
                 "Strategy B",
                 "Strategy AB")
n_str       <- length(v_names_str)        # number of strategies

## Within-cycle correction (WCC) using Simpson's 1/3 rule ----
v_wcc <- gen_wcc(n_cycles = n_cycles,  # Function included in "R/
Functions.R". The latest version can be found in `darthtools` package
                 method = "Simpson1/3") # vector of wcc

### Transition rates (annual), and hazard ratios (HRs) ----
r_HD   <- 0.002 # constant annual rate of dying when Healthy (all-cause
mortality)
r_HS1  <- 0.15  # constant annual rate of becoming Sick when Healthy
r_S1H  <- 0.5   # constant annual rate of becoming Healthy when Sick
r_S1S2 <- 0.105 # constant annual rate of becoming Sicker when Sick
```

```
hr_S1  <- 3     # hazard ratio of death in Sick vs Healthy
hr_S2  <- 10    # hazard ratio of death in Sicker vs Healthy

### Effectiveness of treatment B ----
hr_S1S2_trtB <- 0.6  # hazard ratio of becoming Sicker when Sick under
treatment B

### State rewards ----
#### Costs ----
c_H    <- 2000  # annual cost of being Healthy
c_S1   <- 4000  # annual cost of being Sick
c_S2   <- 15000 # annual cost of being Sicker
c_D    <- 0     # annual cost of being dead
c_trtA <- 12000 # annual cost of receiving treatment A
c_trtB <- 13000 # annual cost of receiving treatment B
#### Utilities ----
u_H    <- 1     # annual utility of being Healthy
u_S1   <- 0.75  # annual utility of being Sick
u_S2   <- 0.5   # annual utility of being Sicker
u_D    <- 0     # annual utility of being dead
u_trtA <- 0.95  # annual utility when receiving treatment A

### Discount weight for costs and effects ----
v_dwc  <- 1 / ((1 + (d_e * cycle_length)) ^ (0:n_cycles))
v_dwe  <- 1 / ((1 + (d_c * cycle_length)) ^ (0:n_cycles))

# Process model inputs ----
## Cycle-specific transition probabilities to the Dead state ----
#* compute mortality rates
r_S1D <- r_HD * hr_S1 # annual mortality rate in the Sick state
r_S2D <- r_HD * hr_S2 # annual mortality rate in the Sicker state
#* transform rates to probabilities
#* Function included in "R/Functions.R". The latest version can be found
in `darthtools` package
p_HS1  <- rate_to_prob(r = r_HS1, t = cycle_length) # constant annual
probability of becoming Sick when Healthy conditional on surviving
p_S1H  <- rate_to_prob(r = r_S1H, t = cycle_length) # constant annual
probability of becoming Healthy when Sick conditional on surviving
p_S1S2 <- rate_to_prob(r = r_S1S2, t = cycle_length)# constant annual
probability of becoming Sicker when Sick conditional on surviving
p_HD   <- rate_to_prob(r = r_HD, t = cycle_length)  # annual mortality
risk in the Healthy state
p_S1D  <- rate_to_prob(r = r_S1D, t = cycle_length) # annual mortality
risk in the Sick state
p_S2D  <- rate_to_prob(r = r_S2D, t = cycle_length) # annual mortality
risk in the Sicker state

## Annual transition probability of becoming Sicker when Sick for
treatment B ----
#* Apply hazard ratio to rate to obtain transition rate of becoming Sicker
when
#* Sick for treatment B
r_S1S2_trtB <- r_S1S2 * hr_S1S2_trtB
#* Transform rate to probability to become Sicker when Sick under
treatment B
#* conditional on surviving
#* (Function included in "R/Functions.R". The latest version can be found
in
#* `darthtools` package)
p_S1S2_trtB <- rate_to_prob(r = r_S1S2_trtB, t = cycle_length)

# Construct state-transition models ----
## Initial state vector ----
#* All starting healthy
v_m_init <- c(H = 1, S1 = 0, S2 = 0, D = 0) # initial state vector
v_m_init

## Initialize cohort traces ----
### Initialize cohort trace for SoC ----
m_M <- matrix(NA,
              nrow = (n_cycles + 1), ncol = n_states,
              dimnames = list(0:n_cycles, v_names_states))
#* Store the initial state vector in the first row of the cohort trace
m_M[1, ] <- v_m_init

### Initialize cohort trace for strategies A, B, and AB ----
#* Structure and initial states are the same as for SoC
m_M_strA  <- m_M # Strategy A
m_M_strB  <- m_M # Strategy B
m_M_strAB <- m_M # Strategy AB

## Create transition probability matrices for strategy SoC ----
### Initialize transition probability matrix for strategy SoC ----
#* All transitions to a non-death state are assumed to be conditional on
survival
m_P <- matrix(0,
              nrow = n_states, ncol = n_states,
              dimnames = list(v_names_states,
                              v_names_states)) # define row and column
names
### Fill in matrix ----
#* From H
m_P["H", "H"]   <- (1 - p_HD) * (1 - p_HS1)
m_P["H", "S1"]  <- (1 - p_HD) * p_HS1
m_P["H", "D"]   <- p_HD
#* From S1
m_P["S1", "H"]  <- (1 - p_S1D) * p_S1H
m_P["S1", "S1"] <- (1 - p_S1D) * (1 - (p_S1H + p_S1S2))
m_P["S1", "S2"] <- (1 - p_S1D) * p_S1S2
m_P["S1", "D"]  <- p_S1D
#* From S2
m_P["S2", "S2"] <- 1 - p_S2D
m_P["S2", "D"]  <- p_S2D
#* From D
m_P["D", "D"]   <- 1

### Initialize transition probability matrix for strategy A as a copy of
SoC's ----
m_P_strA <- m_P

### Initialize transition probability matrix for strategy B ----
m_P_strB <- m_P
#* Update only transition probabilities from S1 involving p_S1S2
m_P_strB["S1", "S1"] <- (1 - p_S1D) * (1 - (p_S1H + p_S1S2_trtB))
m_P_strB["S1", "S2"] <- (1 - p_S1D) * p_S1S2_trtB
```

```
### Initialize transition probability matrix for strategy AB as a copy of
B's ----
m_P_strAB <- m_P_strB

## Check if transition probability matrices are valid ----
#* Functions included in "R/Functions.R". The latest version can be found
in `darthtools` package
### Check that transition probabilities are [0, 1] ----
check_transition_probability(m_P,      verbose = TRUE)  # m_P >= 0 && m_P
<= 1
check_transition_probability(m_P_strA, verbose = TRUE)  # m_P_strA >= 0 &&
m_P_strA <= 1
check_transition_probability(m_P_strB, verbose = TRUE)  # m_P_strB >= 0 &&
m_P_strB <= 1
check_transition_probability(m_P_strAB, verbose = TRUE) # m_P_strAB >= 0
&& m_P_strAB <= 1
### Check that all rows sum to 1 ----
check_sum_of_transition_array(m_P,      n_states = n_states, n_cycles =
n_cycles, verbose = TRUE)  # rowSums(m_P) == 1
check_sum_of_transition_array(m_P_strA, n_states = n_states, n_cycles =
n_cycles, verbose = TRUE)  # rowSums(m_P_strA) == 1
check_sum_of_transition_array(m_P_strB, n_states = n_states, n_cycles =
n_cycles, verbose = TRUE)  # rowSums(m_P_strB) == 1
check_sum_of_transition_array(m_P_strAB, n_states = n_states, n_cycles =
n_cycles, verbose = TRUE) # rowSums(m_P_strAB) == 1

#  Run Markov model ----
#* Iterative solution of time-independent cSTM
for(t in 1:n_cycles){
  # For SoC
  m_M[t + 1, ] <- m_M[t, ] %*% m_P
  # For strategy A
  m_M_strA[t + 1, ] <- m_M_strA[t, ] %*% m_P_strA
  # For strategy B
  m_M_strB[t + 1, ] <- m_M_strB[t, ] %*% m_P_strB
  # For strategy AB
  m_M_strAB[t + 1, ] <- m_M_strAB[t, ] %*% m_P_strAB
}

## Store the cohort traces in a list ----
l_m_M <- list(m_M,
              m_M_strA,
              m_M_strB,
              m_M_strAB)
names(l_m_M) <- v_names_str

# Plot Outputs ----
#* Plot the cohort trace for strategies SoC and A
#* (Function included in "R/Functions.R"; depends on the `ggplot2`
package)
plot_trace(m_M)

# State Rewards ----
## Scale by the cycle length ----
#* Vector of state utilities under strategy SoC
v_u_SoC    <- c(H  = u_H,
                S1 = u_S1,
                S2 = u_S2,
                D  = u_D) * cycle_length
#* Vector of state costs under strategy SoC
v_c_SoC    <- c(H  = c_H,
                S1 = c_S1,
                S2 = c_S2,
                D  = c_D) * cycle_length
#* Vector of state utilities under strategy A
v_u_strA   <- c(H  = u_H,
                S1 = u_trtA,
                S2 = u_S2,
                D  = u_D) * cycle_length
#* Vector of state costs under strategy A
v_c_strA   <- c(H  = c_H,
                S1 = c_S1 + c_trtA,
                S2 = c_S2 + c_trtA,
                D  = c_D)
#* Vector of state utilities under strategy B
v_u_strB   <- c(H  = u_H,
                S1 = u_S1,
                S2 = u_S2,
                D  = u_D) * cycle_length
#* Vector of state costs under strategy B
v_c_strB   <- c(H  = c_H,
                S1 = c_S1 + c_trtB,
                S2 = c_S2 + c_trtB,
                D  = c_D) * cycle_length
#* Vector of state utilities under strategy AB
v_u_strAB  <- c(H  = u_H,
                S1 = u_trtA,
                S2 = u_S2,
                D  = u_D) * cycle_length
#* Vector of state costs under strategy AB
v_c_strAB  <- c(H  = c_H,
                S1 = c_S1 + (c_trtA + c_trtB),
                S2 = c_S2 + (c_trtA + c_trtB),
                D  = c_D) * cycle_length

## Store state rewards ----
#* Store the vectors of state utilities for each strategy in a list
l_u   <- list(SQ = v_u_SoC,
              A  = v_u_strA,
              B  = v_u_strB,
              AB = v_u_strAB)
#* Store the vectors of state cost for each strategy in a list
l_c   <- list(SQ = v_c_SoC,
              A  = v_c_strA,
              B  = v_c_strB,
              AB = v_c_strAB)

#* assign strategy names to matching items in the lists
names(l_u) <- names(l_c) <- v_names_str

# Compute expected outcomes ----
#* Create empty vectors to store total utilities and costs
v_tot_qaly <- v_tot_cost <- vector(mode = "numeric", length = n_str)
names(v_tot_qaly) <- names(v_tot_cost) <- v_names_str

## Loop through each strategy and calculate total utilities and costs ----
```

```
for (i in 1:n_str) {
  v_u_str <- l_u[[i]]   # select the vector of state utilities for the i-
th strategy
  v_c_str <- l_c[[i]]   # select the vector of state costs for the i-th
strategy

  ###* Expected QALYs and costs per cycle
  ##* Vector of QALYs and Costs
  #* Apply state rewards
  v_qaly_str <- l_m_M[[i]] %*% v_u_str # sum the utilities of all states
for each cycle
  v_cost_str <- l_m_M[[i]] %*% v_c_str # sum the costs of all states for
each cycle

  ###* Discounted total expected QALYs and Costs per strategy and apply
within-cycle correction if applicable
  #* QALYs
  v_tot_qaly[i] <- t(v_qaly_str) %*% (v_dwe * v_wcc)
  #* Costs
  v_tot_cost[i] <- t(v_cost_str) %*% (v_dwc * v_wcc)
}

# Cost-effectiveness analysis (CEA) ----
## Incremental cost-effectiveness ratios (ICERs) ----
#* Function included in "R/Functions.R"; depends on the `dplyr` package
#* The latest version can be found in `dampack` package
df_cea <- calculate_icers(cost       = v_tot_cost,
                          effect     = v_tot_qaly,
                          strategies = v_names_str)
df_cea

## CEA table in proper format ----
table_cea <- format_table_cea(df_cea) # Function included in "R/
Functions.R"; depends on the `scales` package
table_cea

## CEA frontier -----
#* Function included in "R/Functions.R"; depends on the `ggplot2`  and
`ggrepel` packages.
#* The latest version can be found in `dampack` package
plot(df_cea, label = "all", txtsize = 16) +
  expand_limits(x = max(table_cea$QALYs) + 0.1) +
  theme(legend.position = c(0.8, 0.2))

#******************************************************************************#
# Probabilistic Sensitivity Analysis (PSA) -----
## Load model, CEA and PSA functions ----
source("R/Functions_cSTM_time_indep.R")
source("R/Functions.R")

## List of input parameters -----
l_params_all <- list(
  # Transition probabilities (per cycle), hazard ratios
  r_HD        = 0.002, # constant rate of dying when Healthy (all-cause
mortality)
  r_HS1       = 0.15,  # probability to become Sick when Healthy
conditional on surviving
```

```
  r_S1H       = 0.5,   # probability to become Healthy when Sick
conditional on surviving
  r_S1S2      = 0.105, # probability to become Sicker when Sick
conditional on surviving
  hr_S1       = 3,     # hazard ratio of death in Sick vs Healthy
  hr_S2       = 10,    # hazard ratio of death in Sicker vs Healthy
  # Effectiveness of treatment B
  hr_S1S2_trtB = 0.6,  # hazard ratio of becoming Sicker when Sick under
treatment B
  ## State rewards
  # Costs
  c_H    = 2000,  # cost of remaining one cycle in Healthy
  c_S1   = 4000,  # cost of remaining one cycle in Sick
  c_S2   = 15000, # cost of remaining one cycle in Sicker
  c_D    = 0,     # cost of being dead (per cycle)
  c_trtA = 12000, # cost of treatment A
  c_trtB = 13000, # cost of treatment B
  # Utilities
  u_H    = 1,     # utility when Healthy
  u_S1   = 0.75,  # utility when Sick
  u_S2   = 0.5,   # utility when Sicker
  u_D    = 0,     # utility when Dead
  u_trtA = 0.95, # utility when being treated with A
  # Initial and maximum ages
  n_age_init = 25,
  n_age_max = 100,
  # Discount rates
  d_c = 0.03, # annual discount rate for costs
  d_e = 0.03, # annual discount rate for QALYs,
  # Cycle length
  cycle_length = 1
)

#* Store the parameter names into a vector
v_names_params <- names(l_params_all)

## Test functions to generate CE outcomes and PSA dataset ----
#* Test function to compute CE outcomes
calculate_ce_out(l_params_all) # Function included in "R/
Functions_cSTM_time_indep.R"

#* Test function to generate PSA input dataset
generate_psa_params(10) # Function included in "R/
Functions_cSTM_time_indep.R"

## Generate PSA dataset ----
#* Number of simulations
n_sim <- 1000

#* Generate PSA input dataset
df_psa_input <- generate_psa_params(n_sim = n_sim)
#* First six observations
head(df_psa_input)

### Histogram of PSA dataset ----
ggplot(melt(df_psa_input, variable.name = "Parameter"),
       aes(x = value)) +
  facet_wrap(~Parameter, scales = "free") +
  geom_histogram(aes(y = ..density..)) +
  scale_x_continuous(breaks = number_ticks(4)) +
  ylab("") +
  theme_bw(base_size = 16) +
  theme(axis.text = element_text(size = 6),
        axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        axis.text.y  = element_blank(),
        axis.ticks.y = element_blank())

## Run PSA ----
#* Initialize data.frames with PSA output
#* data.frame of costs
df_c <- as.data.frame(matrix(0,
                             nrow = n_sim,
                             ncol = n_str))
colnames(df_c) <- v_names_str
#* data.frame of effectiveness
df_e <- as.data.frame(matrix(0,
                             nrow = n_sim,
                             ncol = n_str))
colnames(df_e) <- v_names_str

#* Conduct probabilistic sensitivity analysis
#* Run Markov model on each parameter set of PSA input dataset
n_time_init_psa_series <- Sys.time()
for(i in 1:n_sim){
  l_psa_input <- update_param_list(l_params_all, df_psa_input[i,])
  l_out_temp <- calculate_ce_out(l_psa_input)
  df_c[i, ] <- l_out_temp$Cost
  df_e[i, ] <- l_out_temp$Effect
  # Display simulation progress
  if(i/(n_sim/10) == round(i/(n_sim/10), 0)) { # display progress every
10%
    cat('\r', paste(i/n_sim * 100, "% done", sep = " "))
  }
}
n_time_end_psa_series <- Sys.time()
n_time_total_psa_series <- n_time_end_psa_series - n_time_init_psa_series
print(paste0("PSA with ", scales::comma(n_sim), " simulations run in
series in ",
             round(n_time_total_psa_series, 2), " ",
             units(n_time_total_psa_series)))

### Run Markov model on each parameter set of PSA input dataset in
parallel
# ## Get OS
# os <- get_os()
# print(paste0("Parallelized PSA on ", os))
#
# no_cores <- parallel::detectCores() - 1
#
# n_time_init_psa <- Sys.time()
#
# ## Run parallelized PSA based on OS
# if(os == "osx"){
#   # Initialize cluster object
#   cl <- parallel::makeForkCluster(no_cores)
#   # Register clusters
#   doParallel::registerDoParallel(cl)
#   # Run parallelized PSA
#   df_ce <- foreach::foreach(i = 1:n_sim, .combine = rbind) %dopar% {
#     l_out_temp <- calculate_ce_out(df_psa_input[i, ])
#     df_ce <- c(l_out_temp$Cost, l_out_temp$Effect)
#   }
#   # Extract costs and effects from the PSA dataset
#   df_c <- df_ce[, 1:n_str]
#   df_e <- df_ce[, (n_str+1):(2*n_str)]
#   # Register end time of parallelized PSA
#   n_time_end_psa <- Sys.time()
# }
# if(os == "windows"){
#   # Initialize cluster object
#   cl <- parallel::makeCluster(no_cores)
#   # Register clusters
#   doParallel::registerDoParallel(cl)
#   opts <- list(attachExportEnv = TRUE)
#   # Run parallelized PSA
#   df_ce <- foreach::foreach(i = 1:n_samp, .combine = rbind,
#                             .export = ls(globalenv()),
#                             .packages=c("dampack"),
#                             .options.snow = opts) %dopar% {
#                               l_out_temp <-
calculate_ce_out(df_psa_input[i, ])
#                               df_ce <- c(l_out_temp$Cost,
l_out_temp$Effect)
#                             }
#   # Extract costs and effects from the PSA dataset
#   df_c <- df_ce[, 1:n_str]
#   df_e <- df_ce[, (n_str+1):(2*n_str)]
#   # Register end time of parallelized PSA
#   n_time_end_psa <- Sys.time()
# }
# if(os == "linux"){
#   # Initialize cluster object
#   cl <- parallel::makeCluster(no_cores)
#   # Register clusters
#   doParallel::registerDoMC(cl)
#   # Run parallelized PSA
#   df_ce <- foreach::foreach(i = 1:n_sim, .combine = rbind) %dopar% {
#     l_out_temp <- calculate_ce_out(df_psa_input[i, ])
#     df_ce <- c(l_out_temp$Cost, l_out_temp$Effect)
#   }
#   # Extract costs and effects from the PSA dataset
#   df_c <- df_ce[, 1:n_str]
#   df_e <- df_ce[, (n_str+1):(2*n_str)]
#   # Register end time of parallelized PSA
#   n_time_end_psa <- Sys.time()
# }
# # Stop clusters
# stopCluster(cl)
# n_time_total_psa <- n_time_end_psa - n_time_init_psa
# print(paste0("PSA with ", scales:: comma(n_sim), " simulations run in
series in ",
#              round(n_time_total_psa, 2), " ",
#              units(n_time_total_psa_series)))
```

```r
## Visualize PSA results and CEA ----
### Create PSA object ----
#* Function included in "R/Functions.R" The latest version can be found in
`dampack` package
l_psa <- make_psa_obj(cost          = df_c,
                      effectiveness = df_e,
                      parameters    = df_psa_input,
                      strategies    = v_names_str)
l_psa$strategies <- v_names_str
colnames(l_psa$effectiveness)<- v_names_str
colnames(l_psa$cost)<- v_names_str

#* Vector with willingness-to-pay (WTP) thresholds.
v_wtp <- seq(0, 200000, by = 5000)

### Cost-Effectiveness Scatter plot ----
#* Function included in "R/Functions.R"; depends on `tidyr` and `ellipse`
packages.
#* The latest version can be found in `dampack` package
plot.psa(l_psa) +
  ggthemes::scale_color_colorblind() +
  ggthemes::scale_fill_colorblind() +
  xlab("Effectiveness (QALYs)") +
  guides(col = guide_legend(nrow = 2)) +
  theme(legend.position = "bottom")

### Incremental cost-effectiveness ratios (ICERs) with probabilistic
output ----
#* Compute expected costs and effects for each strategy from the PSA
#* Function included in "R/Functions.R". The latest version can be found
in `dampack` package
df_out_ce_psa <- summary.psa(l_psa)

#* Function included in "R/Functions.R"; depends on the `dplyr` package
#* The latest version can be found in `dampack` package
df_cea_psa <- calculate_icers(cost       = df_out_ce_psa$meanCost,
                              effect     = df_out_ce_psa$meanEffect,
                              strategies = df_out_ce_psa$Strategy)
df_cea_psa

### Plot cost-effectiveness frontier with probabilistic output ----
#* Function included in "R/Functions.R"; depends on the `ggplot2`  and
`ggrepel` packages.
#* The latest version can be found in `dampack` package
plot.icers(df_cea_psa)

## Cost-effectiveness acceptability curves (CEACs) and frontier (CEAF) ---
#* Functions included in "R/Functions.R". The latest versions can be found
in `dampack` package
ceac_obj <- ceac(wtp = v_wtp, psa = l_psa)
#* Regions of highest probability of cost-effectiveness for each strategy
summary.ceac(ceac_obj)
#* CEAC & CEAF plot
plot.ceac(ceac_obj) +
  ggthemes::scale_color_colorblind() +
  ggthemes::scale_fill_colorblind() +
  theme(legend.position = c(0.82, 0.5))
```

```
## Expected Loss Curves (ELCs) ----
#* Function included in "R/Functions.R".The latest version can be found in
`dampack` package
elc_obj <- calc_exp_loss(wtp = v_wtp, psa = l_psa)
elc_obj
#* ELC plot
plot.exp_loss(elc_obj, log_y = FALSE,
     txtsize = 16, xlim = c(0, NA), n_x_ticks = 14,
     col = "full") +
  ggthemes::scale_color_colorblind() +
  ggthemes::scale_fill_colorblind() +
  # geom_point(aes(shape = as.name("Strategy"))) +
  scale_y_continuous("Expected Loss (Thousand $)",
                     breaks = number_ticks(10),
                     labels = function(x) x/1000) +
  theme(legend.position = c(0.4, 0.7))

## Expected value of perfect information (EVPI) ----
#* Function included in "R/Functions.R". The latest version can be found
in `dampack` package
evpi <- calc_evpi(wtp = v_wtp, psa = l_psa)
#* EVPI plot
plot.evpi(evpi, effect_units = "QALY")
```