```
#* Script name: Functions.R
#-------------------------------------------------------------------------
#
####                    Function to plot cohort trace
####
#-------------------------------------------------------------------------
#
#' Plot cohort trace
#'
#' \code{plot_trace} plots the cohort trace.
#'
#' @param m_M a cohort trace matrix
#' @return a ggplot object - plot of the cohort trace
#'
plot_trace <- function(m_M) {
  df_M      <- data.frame(Cycle = 0:n_cycles, m_M, check.names = F)
  df_M_long <- tidyr::gather(df_M, key = `Health State`, value,
2:ncol(df_M))
  df_M_long$`Health State` <- factor(df_M_long$`Health State`, levels =
v_names_states)
  gg_trace <- ggplot(df_M_long, aes(x = Cycle, y = value,
                            color = `Health State`, linetype = `Health
State`)) +
    geom_line(size = 1) +
    xlab("Cycle") +
    ylab("Proportion of the cohort") +
    scale_x_continuous(breaks = number_ticks(8)) +
    theme_bw(base_size = 14) +
    theme(legend.position  = "bottom",
          legend.background = element_rect(fill = NA))

  return(gg_trace)
}

#-------------------------------------------------------------------------
#
####                    Function to plot cohort trace per strategy
####
#-------------------------------------------------------------------------
#
#' Plot cohort trace per strategy
#'
#' \code{plot_trace} plots the cohort trace for each strategy, split by
health state.
#'
#' @param l_m_M a list containing cohort trace matrices
#' @return a ggplot object - plot of the cohort trace for each strategy
split by health state.
#'
plot_trace_strategy <- function(l_m_M) {
  n_str <- length(l_m_M)
  l_df_M <- lapply(l_m_M, as.data.frame)
  df_M_strategies <- data.table::rbindlist(l_df_M, use.names = T,
                                           idcol = "Strategy")
  df_M_strategies$Cycle <- rep(0:n_cycles, n_str)
  m_M_plot <- tidyr::gather(df_M_strategies, key = `Health State`, value,
                            2:(ncol(df_M_strategies)-1))
```

```
  m_M_plot$`Health State`    <- factor(m_M_plot$`Health State`, levels =
v_names_states)
  m_M_plot$Strategy <- factor(m_M_plot$Strategy, levels = v_names_str)

  p <- ggplot(m_M_plot, aes(x = Cycle, y = value,
                            color = Strategy, linetype = Strategy)) +
    geom_line(size = 1) +
    scale_color_brewer(palette="RdBu") +
    xlab("Cycle") +
    ylab("Proportion of the cohort") +
    theme_bw(base_size = 14) +
    theme(legend.position  = "bottom",
          legend.background = element_rect(fill = NA)) +
    facet_wrap(~ `Health State`)

  return(p)
}

#-----------------------------------------------------------------------------
#
####             Function to calculate survival probabilities
####
#-----------------------------------------------------------------------------
#
#' Calculate survival probabilities
#'
#' \code{calc_surv} calculates the survival probabilities.
#'
#' @param l_m_M a list containing cohort trace matrices
#' @return a dataframe containing survival probabilities for each strategy
#'
calc_surv <- function(l_m_M, v_names_death_states) {
  df_surv <- as.data.frame(lapply(l_m_M,
                                  function(x) {
                                    rowSums(x[, !colnames(x) %in%
v_names_death_states])
                                  }
                                  ))
  colnames(df_surv) <- v_names_str
  df_surv$Cycle     <- 0:n_cycles
  df_surv_long      <- tidyr::gather(df_surv, key = Strategy, Survival,
1:n_str)
  df_surv_long$Strategy <- ordered(df_surv_long$Strategy, levels =
v_names_str)
  df_surv_long <- df_surv_long %>%
    select(Strategy, Cycle, Survival)

  return(df_surv_long)
}

#-----------------------------------------------------------------------------
#
####                Function to calculate state proportions
####
#-----------------------------------------------------------------------------
#
#' Calculate state proportions
#'
```

```
#' \code{calc_surv} calculates the proportions of the cohort in specified
states
#'
#' @param l_m_M a list containing cohort trace matrices
#' @return a dataframe containing proportions in specified states for each
strategy
#'
calc_sick <- function(l_m_M, v_names_sick_states) {
  n_sick_states <- length(v_names_sick_states)
  df_sick <- as.data.frame(lapply(l_m_M,
                                  function(x) {
                                    if (n_sick_states == 1) {
                                      x[, colnames(x) %in%
v_names_sick_states]
                                    } else {
                                      rowSums(x[, colnames(x) %in%
v_names_sick_states])
                                    }
                                  }
  ))
  colnames(df_sick) <- v_names_str
  df_sick$Cycle     <- 0:n_cycles
  df_sick_long      <- tidyr::gather(df_sick, key = Strategy, Sick,
1:n_str)
  df_sick_long$Strategy <- ordered(df_sick_long$Strategy, levels =
v_names_str)
  df_sick_long <- df_sick_long %>%
    select(Strategy, Cycle, Sick)

  return(df_sick_long)
}

#-----------------------------------------------------------------------------
#
####                   Function to calculate prevalence
####
#-----------------------------------------------------------------------------
#
#' Calculate prevalence
#'
#' \code{plot_prevalence} calculate the prevalence for different health
states.
#'
#' @param l_m_M a list containing cohort trace matrices
#' @return a dataframe containing prevalence of specified health states
for each strategy
#'
calc_prevalence <- function(l_m_M, v_names_sick_states,
v_names_dead_states) {
  df_alive      <- calc_surv(l_m_M, v_names_dead_states)
  df_prop_sick  <- calc_sick(l_m_M, v_names_sick_states)
  df_prevalence <- data.frame(Strategy   = df_alive$Strategy,
                              Cycle      = df_alive$Cycle,
                              Prevalence = df_prop_sick$Sick /
df_alive$Survival)
  return(df_prevalence)
}
```

```
#-------------------------------------------------------------------------
#
####           Function to calculate state-in-state proportions
####
#-------------------------------------------------------------------------
#
#' Calculate state-in-state proportions
#'
#' \code{plot_prevalence} calculates the proportion of a speciefied subset
of states among a set of specified states
#'
#' @param l_m_M a list containing cohort trace matrices
#' @return a dataframe containing state-in-state proportions of specified
health states for each strategy
#'
calc_prop_sicker <- function(l_m_M, v_names_sick_states,
v_names_sicker_states) {
  df_prop_sick   <- calc_sick(l_m_M, v_names_sick_states)
  df_prop_sicker <- calc_sick(l_m_M, v_names_sicker_states)
  df_prop_sick_sicker <- data.frame(Strategy   = df_prop_sick$Strategy,
                                    Cycle      = df_prop_sick$Cycle,
                                    `Proportion Sicker` =
                                     df_prop_sicker$Sick /
                                    (df_prop_sick$Sick +
df_prop_sicker$Sick))

  return(df_prop_sick_sicker)
}

#-------------------------------------------------------------------------
#
####                   Function to plot survival curve
####
#-------------------------------------------------------------------------
#
#' Plot survival curve
#'
#' \code{plot_surv} plots the survival probability curve.
#'
#' @param l_m_M a list containing cohort trace matrices
#' @return a ggplot object - plot of the survival curve
#'
plot_surv <- function(l_m_M, v_names_death_states) {
  df_surv <- calc_surv(l_m_M, v_names_death_states)
  df_surv$Strategy <- factor(df_surv$Strategy, levels = v_names_str)
  df_surv$Survival <- round(df_surv$Survival, 2)

  p <- ggplot(df_surv,
              aes(x = Cycle, y = Survival, group = Strategy)) +
    geom_line(aes(linetype = Strategy, col = Strategy), size = 1.2) +
    scale_color_brewer(palette="RdBu") +
    xlab("Cycle") +
    ylab("Proportion") +
    ggtitle("Survival probabilities") +
    theme_bw(base_size = 14) +
    theme()

  return(p)
```

```
}

#-----------------------------------------------------------------------------
#
####                   Function to plot prevalence curve
####
#-----------------------------------------------------------------------------
#
#' Plot prevalence curve
#'
#' \code{plot_prevalence} plots the prevalence curve for specified health
states.
#'
#' @param l_m_M a list containing cohort trace matrices
#' @return a ggplot object - plot of the prevalence curve
#'
plot_prevalence <- function(l_m_M, v_names_sick_states,
v_names_dead_states) {
  df_prevalence <- calc_prevalence(l_m_M, v_names_sick_states,
v_names_dead_states)
  df_prevalence$Strategy <- factor(df_prevalence$Strategy, levels =
v_names_str)
  df_prevalence$Proportion.Sicker <- round(df_prevalence$Prevalence, 2)

  p <- ggplot(df_prevalence,
              aes(x = Cycle, y = Prevalence, group = Strategy)) +
    geom_line(aes(linetype = Strategy, col = Strategy), size = 1.2) +
    scale_color_brewer(palette = "RdBu") +
    xlab("Cycle") +
    ylab("Proportion") +
    ggtitle(paste("Prevalence", "of", paste(v_names_sick_states, collapse
= " "))) +
    theme_bw(base_size = 14) +
    theme()

  return(p)
}

#-----------------------------------------------------------------------------
#
####           Function to plot state-in-state proportion curve
####
#-----------------------------------------------------------------------------
#
#' Plot state-in-state proportion curve
#'
#' \code{plot_prevalence} plots the
#'
#' @param l_m_M a list containing cohort trace matrices
#' @return a ggplot object - plot of state-in-state proportion curve
#'
plot_proportion_sicker <- function(l_m_M, v_names_sick_states,
v_names_sicker_states) {
  df_proportion_sicker <- calc_prop_sicker(l_m_M, v_names_sick_states,
v_names_sicker_states)
  df_proportion_sicker$Strategy <- factor(df_proportion_sicker$Strategy,
levels = v_names_str)
```

```
  df_proportion_sicker$Proportion.Sicker <-
round(df_proportion_sicker$Proportion.Sicker, 2)

  p <- ggplot(df_proportion_sicker,
              aes(x = Cycle, y = Proportion.Sicker, group = Strategy)) +
    geom_line(aes(linetype = Strategy, col = Strategy), size = 1.2, na.rm
= T) +
    scale_color_brewer(palette = "RdBu") +
    xlab("Cycle") +
    ylab("Proportion") +
    ggtitle(paste(paste("Proportion of", v_names_sicker_states),
                  paste(c("among", v_names_sick_states), collapse = " ")))
+
    theme_bw(base_size = 14) +
    theme()

  return(p)
}

#-----------------------------------------------------------------------------
#
####                      Function to format CEA table
####
#-----------------------------------------------------------------------------
#
#' Format CEA table
#'
#' \code{format_table_cea} formats the CEA table.
#'
#' @param table_cea a dataframe object - table with CEA results
#' @return a dataframe object - formatted CEA table
#'
format_table_cea <- function(table_cea) {
  colnames(table_cea)[colnames(table_cea)
                      %in% c("Cost",
                             "Effect",
                             "Inc_Cost",
                             "Inc_Effect",
                             "ICER")] <-

                          c("Costs ($)",
                            "QALYs",
                            "Incremental Costs ($)",
                            "Incremental QALYs",
                            "ICER ($/QALY)")

  table_cea$`Costs ($)` <- scales::comma(round(table_cea$`Costs ($)`, 0))
  table_cea$`Incremental Costs ($)` <-
scales::comma(round(table_cea$`Incremental Costs ($)`, 0))
  table_cea$QALYs <- round(table_cea$QALYs, 2)
  table_cea$`Incremental QALYs` <- round(table_cea$`Incremental QALYs`, 2)
  table_cea$`ICER ($/QALY)` <- scales::comma(round(table_cea$`ICER ($/
QALY)`, 0))
  return(table_cea)
}

#############################################################################
```

```
################## FUNCTIONS INCLUDED IN DARTHTOOLS
##############################
################################################################################
#' Within-cycle correction (WCC)
#'
#' \code{gen_wcc} generates a vector of within-cycle corrections (WCC).
#'
#' @param n_cycles number of cycles
#' @param method The method to be used for within-cycle correction.
#'
#' @return A vector of length \code{n_cycles + 1} with within-cycle
corrections
#'
#' @details
#' The default method is an implementation of Simpson's 1/3rd rule that
#' generates a vector with the first and last entry with 1/3 and the odd
and
#' even entries with 4/3 and 2/3, respectively.
#'
#' Method "\code{half-cycle}" is the half-cycle correction method that
#' generates a vector with the first and last entry with 1/2 and the rest
equal
#' to 1.
#'
#' Method "\code{none}" does not implement any within-cycle correction and
#' generates a vector with ones.
#'
#' @references
#' \enumerate{
#' \item Elbasha EH, Chhatwal J. Myths and misconceptions of within-cycle
#' correction: a guide for modelers and decision makers.
Pharmacoeconomics.
#' 2016;34(1):13-22.
#' \item Elbasha EH, Chhatwal J. Theoretical foundations and practical
#' applications of within-cycle correction methods. Med Decis Mak.
#' 2016;36(1):115-131.
#' }
#'
#' @examples
#' # Number of cycles
#' n_cycles <- 10
#' gen_wcc(n_cycles = n_cycles, method = "Simpson1/3")
#' gen_wcc(n_cycles = n_cycles, method = "half-cycle")
#' gen_wcc(n_cycles = n_cycles, method = "none")
#'
#' @export
gen_wcc <- function (n_cycles, method = c("Simpson1/3", "half-cycle",
"none"))
{
  if (n_cycles <= 0) {
    stop("Number of cycles should be positive")
  }
  method <- match.arg(method)
  n_cycles <- as.integer(n_cycles)
  if (method == "Simpson1/3") {
    v_cycles <- seq(1, n_cycles + 1)
    v_wcc <- ((v_cycles%%2) == 0) * (2/3) + ((v_cycles%%2) !=
                                               0) * (4/3)
```

```
    v_wcc[1] <- v_wcc[n_cycles + 1] <- 1/3
  }
  if (method == "half-cycle") {
    v_wcc <- rep(1, n_cycles + 1)
    v_wcc[1] <- v_wcc[n_cycles + 1] <- 0.5
  }
  if (method == "none") {
    v_wcc <- rep(1, n_cycles + 1)
  }
  return(v_wcc)
}

function (r, t = 1)
{
  if ((sum(r < 0) > 0)) {
    stop("rate not greater than or equal to 0")
  }
  p <- 1 - exp(-r * t)
  return(p)
}

#' Convert a rate to a probability
#'
#' \code{rate_to_prob} convert a rate to a probability.
#'
#' @param r rate
#' @param t time/ frequency
#' @return a scalar or vector with probabilities
#' @examples
#' # Annual rate to monthly probability
#' r_year  <- 0.3
#' r_month <- rate_to_prob(r = r_year, t = 1/12)
#' r_month
#' @export
rate_to_prob <- function(r, t = 1){
  if ((sum(r < 0) > 0)){
    stop("rate not greater than or equal to 0")
  }
  p <- 1 - exp(- r * t)
  return(p)
}

#' Check if transition array is valid
#'
#' \code{check_transition_probability} checks if transition probabilities
are in \[0, 1\].
#'
#' @param a_P A transition probability array/ matrix.
#' @param err_stop Logical variable to stop model run if set up as TRUE.
Default = FALSE.
#' @param verbose Logical variable to indicate print out of messages.
#' Default = FALSE
#'
#' @return
#' This function stops if transition probability array is not valid and
shows
#' what are the entries that are not valid
#' @export
```

```
check_transition_probability <- function(a_P,
                                         err_stop = FALSE,
                                         verbose = FALSE) {

  a_P <- as.array(a_P)

  # Verify if a_P is 2D or 3D matrix
  n_dim <- length(dim(a_P))
  # If a_P is a 2D matrix, convert to a 3D array
  if (n_dim < 3){
    a_P <- array(a_P, dim = list(nrow(a_P), ncol(a_P), 1),
                 dimnames = list(rownames(a_P), colnames(a_P), "Time
independent"))
  }
  # Check which entries are not valid
  m_indices_notvalid <- arrayInd(which(a_P < 0 | a_P > 1),
                                 dim(a_P))

  if(dim(m_indices_notvalid)[1] != 0){
    v_rows_notval   <- rownames(a_P)[m_indices_notvalid[, 1]]
    v_cols_notval   <- colnames(a_P)[m_indices_notvalid[, 2]]
    v_cycles_notval <- dimnames(a_P)[[3]][m_indices_notvalid[, 3]]

    df_notvalid <- data.frame(`Transition probabilities not valid:` =
                                matrix(paste0(paste(v_rows_notval,
v_cols_notval, sep = "->"),
                                              "; at cycle ",
                                              v_cycles_notval), ncol = 1),
                              check.names = FALSE)

    if(err_stop) {
      stop("Not valid transition probabilities\n",
           paste(capture.output(df_notvalid), collapse = "\n"))
    }

    if(verbose){
      warning("Not valid transition probabilities\n",
              paste(capture.output(df_notvalid), collapse = "\n"))
    }
  }
}

#' Check if the sum of transition probabilities equal to one.
#'
#' \code{check_sum_of_transition_array} checks if each of the rows of the
#' transition matrices sum to one.
#'
#' @param a_P A transition probability array/ matrix.
#' @param n_states Number of health states in a Markov trace, appropriate
for Markov models.
#' @param n_rows Number of rows (individuals), appropriate for
microsimulation models.
#' @param n_cycles Number of cycles.
#' @param err_stop Logical variable to stop model run if set up as TRUE.
#' Default = TRUE.
#' @param verbose Logical variable to indicate print out of messages.
#' Default = TRUE
#' @return
```

```
#' The transition probability array and the cohort trace matrix.
#' @export
check_sum_of_transition_array <- function(a_P,
                                          n_rows = NULL,
                                          n_states = NULL,
                                          n_cycles,
                                          err_stop = TRUE,
                                          verbose  = TRUE) {

  if (!is.null(n_rows) & !is.null(n_states)) {
    stop("Pick either n_rows or n_states, not both.")
  }

  if (is.null(n_rows) & is.null(n_states)) {
    stop("Need to specify either n_rows or n_states, but not both.")
  }

  if (!is.null(n_rows)) {
    n_states <- n_rows
  }

  a_P <- as.array(a_P)
  d <- length(dim(a_P))
  # For matrix
  if (d == 2) {
    valid <- sum(rowSums(a_P))
    if (abs(valid - n_states) > 0.01) {
      if(err_stop) {
        stop("This is not a valid transition matrix")
      }

      if(verbose){
        warning("This is not a valid transition matrix")
      }
    }
  } else {
    # For array
    valid <- (apply(a_P, d, function(x) sum(rowSums(x))) == n_states)
    if (!isTRUE(all.equal(as.numeric(sum(valid)), as.numeric(n_cycles))))
{
      if(err_stop) {
        stop("This is not a valid transition array")
      }

      if(verbose){
        warning("This is not a valid transition array")
      }
    }
  }
}

################################################################################
################## FUNCTIONS INCLUDED IN DAMPACK
################################
################################################################################

# For a more detailed description and current versions of the functions
above,
```

```
# please go to dampack's package Github repo (https://github.com/DARTH-
git/dampack)

#' Cost-Effectiveness Acceptability Curve (CEAC)
#'
#' \code{ceac} is used to compute and plot the cost-effectiveness
acceptability
#' curves (CEAC) from a probabilistic sensitivity analysis (PSA) dataset.
#'
#' @param wtp numeric vector with willingness-to-pay (WTP) thresholds
#' @param psa psa object from \code{\link{make_psa_obj}}
#' @keywords cost-effectiveness acceptability curves
#' @details
#' \code{ceac} computes the probability of each of the strategies being
#' cost-effective at each \code{wtp} threshold. The returned object has
classes
#' \code{ceac} and \code{data.frame}, and has its own plot method
(\code{\link{plot.ceac}}).
#'
#' @return An object of class \code{ceac} that can be visualized with
\code{plot}. The \code{ceac}
#' object is a data.frame that shows the proportion of PSA samples for
which each strategy at each
#' WTP threshold is cost-effective. The final column indicates whether or
not the strategy at a
#' particular WTP is on the cost-efficient frontier.
#'
#' @examples
#' # psa input provided with package
#' data("example_psa")
#' example_psa_obj <- make_psa_obj(example_psa$cost,
example_psa$effectiveness,
#'                       example_psa$parameters, example_psa$strategies)
#'
#' # define wtp threshold vector (can also use a single wtp)
#' wtp <- seq(1e4, 1e5, by = 1e4)
#' ceac_obj <- ceac(wtp, example_psa_obj)
#' plot(ceac_obj) # see ?plot.ceac for options
#'
#' # this is most useful when there are many strategies
#' # warnings are printed to describe strategies that
#' # have been filtered out
#' plot(ceac_obj, min_prob = 0.5)
#'
#' # standard ggplot layers can be used
#' plot(ceac_obj) +
#'     labs(title = "CEAC", y = "Pr(Cost-effective) at WTP")
#'
#' # the ceac object is also a data frame
#' head(ceac_obj)
#'
#' # summary() tells us the regions of cost-effectiveness for each
strategy.
#' # Note that the range_max column is an open parenthesis, meaning that
the
#' # interval over which that strategy is cost-effective goes up to but
does not include
#' # the value in the range_max column.
```

```
#' summary(ceac_obj)
#'
#' @seealso
#' \code{\link{plot.ceac}}, \code{\link{summary.ceac}}
#'
#'
#' @importFrom tidyr pivot_longer
#' @export
ceac <- function(wtp, psa) {
  # check that psa has class 'psa'
  check_psa_object(psa)

  # define needed variables
  strategies <- psa$strategies
  n_strategies <- psa$n_strategies
  effectiveness <- psa$effectiveness
  cost <- psa$cost
  n_sim <- psa$n_sim

  # number of willingness to pay thresholds
  n_wtps <- length(wtp)

  # matrix to store probability optimal for each strategy
  cea <- matrix(0, nrow = n_wtps, ncol = n_strategies)
  colnames(cea) <- strategies

  # vector to store strategy at the cost-effectiveness acceptability
frontier
  frontv <- rep(0, n_wtps)

  for (l in 1:n_wtps) {
    # calculate net monetary benefit at wtp[l]
    lth_wtp <- wtp[l]
    nmb <-  calculate_outcome("nmb", cost, effectiveness, lth_wtp)

    # find the distribution of optimal strategies
    max.nmb <- max.col(nmb)
    opt <- table(max.nmb)
    cea[l, as.numeric(names(opt))] <- opt / n_sim

    # calculate point on CEAF
    # the strategy with the highest expected nmb
    frontv[l] <- which.max(colMeans(nmb))
  }

  # make cea df
  cea_df <- data.frame(wtp, cea, strategies[frontv],
                       stringsAsFactors = FALSE)
  colnames(cea_df) <- c("WTP", strategies, "fstrat")

  # Reformat df to long format
  ceac <- tidyr::pivot_longer(
    data = cea_df,
    cols = !c("WTP", "fstrat"),
    names_to = "Strategy",
    values_to = "Proportion"
  )
```

```
  # boolean for on frontier or not
  ceac$On_Frontier <- (ceac$fstrat == ceac$Strategy)

  # drop fstrat column
  ceac$fstrat <- NULL

  # order by WTP
  ceac <- ceac[order(ceac$WTP), ]

  # remove rownames
  rownames(ceac) <- NULL

  # make strategies in ceac object into ordered factors
  ceac$Strategy <- factor(ceac$Strategy, levels = strategies, ordered =
TRUE)

  # define classes
  # defining data.frame as well allows the object to use print.data.frame,
for example
  class(ceac) <- c("ceac", "data.frame")

  return(ceac)
}

#' Plot of Cost-Effectiveness Acceptability Curves (CEAC)
#'
#' Plots the CEAC, using the object created by \code{\link{ceac}}.
#'
#' @param x object of class \code{ceac}.
#' @param frontier whether to plot acceptability frontier (TRUE) or not
(FALSE)
#' @param points whether to plot points (TRUE) or not (FALSE)
#' @param currency string with currency used in the cost-effectiveness
analysis (CEA).
#'Defaults to \code{$}, but can be any currency symbol or word (e.g., £,
€, peso)
#' @param min_prob minimum probability to show strategy in plot.
#' For example, if the min_prob is 0.05, only strategies that ever
#' exceed Pr(Cost Effective) = 0.05 will be plotted. Most useful in
situations
#' with many strategies.
#' @inheritParams add_common_aes
#'
#' @keywords internal
#'
#' @details
#' \code{ceac} computes the probability of each of the strategies being
#' cost-effective at each \code{wtp} value.
#' @return A \code{ggplot2} plot of the CEAC.
#'
#' @import ggplot2
#' @import dplyr
#'
#' @export
plot.ceac <- function(x,
                      frontier = TRUE,
                      points = TRUE,
                      currency = "$",
```

```
                      min_prob = 0,
                      txtsize = 12,
                      n_x_ticks = 10,
                      n_y_ticks = 8,
                      xbreaks = NULL,
                      ybreaks = NULL,
                      ylim = NULL,
                      xlim = c(0, NA),
                      col = c("full", "bw"),
                      ...) {
  wtp_name <- "WTP"
  prop_name <- "Proportion"
  strat_name <- "Strategy"
  x$WTP_thou <- x[, wtp_name] / 1000

  # removing strategies with probabilities always below `min_prob`
  # get group-wise max probability
  if (min_prob > 0) {
    max_prob <- x %>%
      group_by(.data$Strategy) %>%
      summarize(maxpr = max(.data$Proportion)) %>%
      filter(.data$maxpr >= min_prob)
    strat_to_keep <- max_prob$Strategy
    if (length(strat_to_keep) == 0) {
      stop(
        paste("no strategies remaining. you may want to lower your
min_prob value (currently ",
              min_prob, ")", sep = "")
      )
    }
    # report filtered out strategies
    old_strat <- unique(x$Strategy)
    diff_strat <- setdiff(old_strat, strat_to_keep)
    n_diff_strat <- length(diff_strat)
    if (n_diff_strat > 0) {
      # report strategies filtered out
      cat("filtered out ", n_diff_strat, " strategies with max prob below
", min_prob, ":\n",
          paste(diff_strat, collapse = ","), "\n", sep = "")

      # report if any filtered strategies are on the frontier
      df_filt <- filter(x, .data$Strategy %in% diff_strat &
.data$On_Frontier)
      if (nrow(df_filt) > 0) {
        cat(paste0("WARNING - some strategies that were filtered out are
on the frontier:\n",
                   paste(unique(df_filt$Strategy), collapse = ","), "\n"))
      }
    }

    # filter dataframe
    x <- filter(x, .data$Strategy %in% strat_to_keep)
  }

  # Drop unused strategy names
  x$Strategy <- droplevels(x$Strategy)

  p <- ggplot(data = x, aes_(x = as.name("WTP_thou"),
```

```
                             y = as.name(prop_name),
                             color = as.name(strat_name))) +
    geom_line() +
    xlab(paste("Willingness to Pay (Thousand ", currency, " / QALY)", sep
= "")) +
    ylab("Pr Cost-Effective")

  if (points) {
    p <- p + geom_point(aes_(color = as.name(strat_name)))
  }

  if (frontier) {
    front <- x[x$On_Frontier, ]
    p <- p + geom_point(data = front, aes_(x = as.name("WTP_thou"),
                                           y = as.name(prop_name),
                                           shape =
as.name("On_Frontier")),
                        size = 3, stroke = 1, color = "black") +
      scale_shape_manual(name = NULL, values = 0, labels = "Frontier") +
      guides(color = guide_legend(order = 1),
             shape = guide_legend(order = 2))
  }
  col <- match.arg(col)
  add_common_aes(p, txtsize, col = col, col_aes = "color",
                 continuous = c("x", "y"), n_x_ticks = n_x_ticks,
n_y_ticks = n_y_ticks,
                 xbreaks = xbreaks, ybreaks = ybreaks,
                 ylim = ylim, xlim = xlim)
}


#' Summarize a ceac
#'
#' Describes cost-effective strategies and their
#' associated intervals of cost-effectiveness
#'
#' @param object object returned from the \code{ceac} function
#' @param ... further arguments (not used)
#' @return data frame showing the interval of cost effectiveness for each
#' interval. The intervals are open on the right endpoint -
#' i.e., [\code{range_min}, \code{range_max})
#'
#' @keywords internal
#'
#' @export
summary.ceac <- function(object, ...) {
  front <- object[object$On_Frontier == TRUE, ]
  front$Strategy <- as.character(front$Strategy)
  wtp <- front$WTP
  wtp_range <- range(wtp)
  n_wtps <- length(wtp)

  # get the indices where the CE strategy isn't the same as the following
CE strategy
  strat_on_front <- front$Strategy
  lagged_strat <- c(strat_on_front[-1], strat_on_front[n_wtps])
  switches <- which(strat_on_front != lagged_strat) + 1
  n_switches <- length(switches)
```

```
  # strat_on_front[switches] are the optimal strategies at wtp[switches]
  if (n_switches == 0) {
    wtp_min <- wtp_range[1]
    wtp_max <- wtp_range[2]
    one_strat <- unique(front$Strategy)
    sum_df <- data.frame(wtp_min,
                         wtp_max,
                         one_strat)
  } else {
    # build up summary data frame
    sum_df <- NULL
    for (i in 1:n_switches) {
      if (i == 1) {
        sum_df_row_first <- data.frame(wtp_range[1],
                                       wtp[switches],
                                       strat_on_front[switches - 1],
                                       fix.empty.names = FALSE,
                                       stringsAsFactors = FALSE)
        sum_df <- rbind(sum_df, sum_df_row_first)
      }
      if (i == n_switches) {
        sum_df_row_last <- data.frame(wtp[switches],
                                      wtp_range[2],
                                      strat_on_front[switches],
                                      fix.empty.names = FALSE,
                                      stringsAsFactors = FALSE)
        sum_df <- rbind(sum_df, sum_df_row_last)
      }
      if (i > 1) {
        sum_df_row_middle <- data.frame(wtp[switches[i]],
                                        wtp[switches[i + 1]],
                                        strat_on_front[switches[i]],
                                        fix.empty.names = FALSE,
                                        stringsAsFactors = FALSE)
        sum_df <- rbind(sum_df, sum_df_row_middle)
      }
    }
  }
  names(sum_df) <- c("range_min", "range_max", "cost_eff_strat")
  sum_df
}

#' Calculate incremental cost-effectiveness ratios (ICERs)
#'
#' @description
#' This function takes in strategies and their associated cost and effect,
assigns them
#' one of three statuses (non-dominated, extended dominated, or
dominated), and
#' calculates the incremental cost-effectiveness ratios for the non-
dominated strategies
#'
#' The cost-effectiveness frontier can be visualized with \code{plot},
which calls \code{\link{plot.icers}}.
#'
#' An efficent way to get from a probabilistic sensitivity analysis to an
ICER table
```

```
#' is by using \code{summary} on the PSA object and then using its columns
as
#' inputs to \code{calculate_icers}.
#'
#' @param cost vector of cost for each strategy
#' @param effect vector of effect for each strategy
#' @param strategies string vector of strategy names
#' With the default (NULL), there is no reference strategy, and the
strategies
#' are ranked in ascending order of cost.
#'
#' @return A data frame and \code{icers} object of strategies and their
associated
#' status, incremental cost, incremental effect, and ICER.
#'
#' @seealso \code{\link{plot.icers}}
#'
#' @examples
#' ## Base Case
#' # if you have a base case analysis, can use calculate_icers on that
#' data(hund_strat)
#' hund_icers <- calculate_icers(hund_strat$Cost,
#'                               hund_strat$QALYs,
#'                               hund_strat$Strategy)
#'
#' plot(hund_icers)
#' # we have so many strategies that we may just want to plot the frontier
#' plot(hund_icers, plot_frontier_only = TRUE)
#' # see ?plot.icers for more options
#'
#' ## Using a PSA object
#' data(psa_cdiff)
#'
#' # summary() gives mean cost and effect for each strategy
#' sum_cdiff <- summary(psa_cdiff)
#'
#' # calculate icers
#' icers <- calculate_icers(sum_cdiff$meanCost,
#'                          sum_cdiff$meanEffect,
#'                          sum_cdiff$Strategy)
#' icers
#'
#' # visualize
#' plot(icers)
#'
#' # by default, only the frontier is labeled
#' # if using a small number of strategies, you can label all the points
#' # note that longer strategy names will get truncated
#' plot(icers, label = "all")
#' @export
calculate_icers <- function(cost, effect, strategies) {
  # checks on input
  n_cost <- length(cost)
  n_eff <- length(effect)
  n_strat <- length(strategies)
  if (n_cost != n_eff | n_eff != n_strat) {
    stop("cost, effect, and strategies must all be vectors of the same
length", call. = FALSE)
```

```
}

# coerce to character, in case they are provided as numeric
char_strat <- as.character(strategies)

# create data frame to hold data
df <- data.frame("Strategy" = char_strat,
                 "Cost" = cost,
                 "Effect" = effect,
                 stringsAsFactors = FALSE)
nstrat <- nrow(df)

# if only one strategy was provided, return df with NAs for incremental
if (nstrat == 1) {
  df[, c("ICER", "Inc_Cost", "Inc_Effect")] <- NA
  return(df)
}

# three statuses: dominated, extended dominated, and non-dominated
d <- NULL

# detect dominated strategies
# dominated strategies have a higher cost and lower effect
df <- df %>%
  arrange(.data$Cost, desc(.data$Effect))

# iterate over strategies and detect (strongly) dominated strategies
# those with higher cost and equal or lower effect
for (i in 1:(nstrat - 1)) {
  ith_effect <- df[i, "Effect"]
  for (j in (i + 1):nstrat) {
    jth_effect <- df[j, "Effect"]
    if (jth_effect <= ith_effect) {
      # append dominated strategies to vector
      d <- c(d, df[j, "Strategy"])
    }
  }
}

# detect weakly dominated strategies (extended dominance)
# this needs to be repeated until there are no more ED strategies
ed <- vector()
continue <- TRUE  # ensure that the loop is run at least once
while (continue) {
  # vector of all dominated strategies (strong or weak)
  dom <- union(d, ed)

  # strategies declared to be non-dominated at this point
  nd <- setdiff(strategies, dom)

  # compute icers for nd strategies
  nd_df <- df[df$Strategy %in% nd, ] %>%
    compute_icers()

  # number non-d
  n_non_d <- nrow(nd_df)

  # if only two strategies left, we're done
```

```
    if (n_non_d <= 2) {
      break
    }

    # strategy identifiers for non-d
    nd_strat <- nd_df$Strategy

    # now, go through non-d strategies and detect any
    # with higher ICER than following strategy
    ## keep track of whether any ED strategies are picked up
    # if not, we're done - exit the loop
    new_ed <- 0
    for (i in 2:(n_non_d - 1)) {
      if (nd_df[i, "ICER"] > nd_df[i + 1, "ICER"]) {
        ed <- c(ed, nd_strat[i])
        new_ed <- new_ed + 1
      }
    }
    if (new_ed == 0) {
      continue <- FALSE
    }
  }

  # recompute icers without weakly dominated strategies
  nd_df_icers <- nd_df[!(nd_df$Strategy %in% dom), ] %>%
    mutate(Status = "ND") %>%
    compute_icers()

  # dominated and weakly dominated
  d_df <- df[df$Strategy %in% d, ] %>%
    mutate(ICER = NA, Status = "D")

  ed_df <- df[df$Strategy %in% ed, ] %>%
    mutate(ICER = NA, Status = "ED")

  # when combining, sort so we have ref,ND,ED,D
  results <- bind_rows(d_df, ed_df, nd_df_icers) %>%
    arrange(desc(.data$Status), .data$Cost, desc(.data$Effect))

  # re-arrange columns
  results <- results %>%
    select(.data$Strategy, .data$Cost, .data$Effect,
           .data$Inc_Cost, .data$Inc_Effect, .data$ICER, .data$Status)

  # declare class of results
  class(results) <- c("icers", "data.frame")
  return(results)
}

#' Calculate incremental cost-effectiveness ratios from a \code{psa}
object.
#'
#' @description The mean costs and QALYs for each strategy in a PSA are
used
#' to conduct an incremental cost-effectiveness analysis.
\code{\link{calculate_icers}} should be used
#' if costs and QALYs for each strategy need to be specified manually,
whereas \code{calculate_icers_psa}
```

```
#' can be used if mean costs and mean QALYs from the PSA are assumed to
represent a base case scenario for
#' calculation of ICERS.
#'
#' Optionally, the \code{uncertainty} argument can be used to provide the
2.5th and 97.5th
#' quantiles for each strategy's cost and QALY outcomes based on the
variation present in the PSA.
#' Because the dominated vs. non-dominated status and the ordering of
strategies in the ICER table are
#' liable to change across different samples of the PSA, confidence
intervals are not provided for the
#' incremental costs and QALYs along the cost-effectiveness acceptability
frontier.
#' \code{link{plot.psa}} does not show the confidence intervals in the
resulting plot
#' even if present in the ICER table.
#'
#' @param psa \code{psa} object from \code{link{make_psa_object}}
#' @param uncertainty whether or not 95% quantiles for the cost and QALY
outcomes should be included
#' in the resulting ICER table. Defaults to \code{FALSE}.
#'
#' @return A data frame and \code{icers} object of strategies and their
associated
#' status, cost, effect, incremental cost, incremental effect, and ICER.
If \code{uncertainty} is
#' set to \code{TRUE}, four additional columns are provided for the 2.5th
and 97.5th quantiles for
#' each strategy's cost and effect.
#' @seealso \code{\link{plot.icers}}
#' @seealso \code{\link{calculate_icers}}
#' @importFrom tidyr pivot_longer
#' @export
calculate_icers_psa <- function(psa, uncertainty = FALSE) {

  # check that psa has class 'psa'
  check_psa_object(psa)

  # Calculate mean outcome values
  psa_sum <- summary(psa)

  # Supply mean outcome values to calculate_icers
  icers <- calculate_icers(cost = psa_sum$meanCost,
                           effect = psa_sum$meanEffect,
                           strategies = psa_sum$Strategy)

  if (uncertainty == TRUE) {

    # extract cost and effect data.frames from psa object
    cost <- psa$cost
    effect <- psa$effectiveness

    # Calculate quantiles across costs and effects
    cost_bounds <- cost %>%
      pivot_longer(cols = everything(), names_to = "Strategy") %>%
      group_by(.data$Strategy) %>%
```

```
      summarize(Lower_95_Cost = quantile(.data$value, probs = 0.025, names
= FALSE),
                Upper_95_Cost = quantile(.data$value, probs = 0.975, names
= FALSE))

    effect_bounds <- effect %>%
      pivot_longer(cols = everything(), names_to = "Strategy") %>%
      group_by(.data$Strategy) %>%
      summarize(Lower_95_Effect = quantile(.data$value, probs = 0.025,
names = FALSE),
                Upper_95_Effect = quantile(.data$value, probs = 0.975,
names = FALSE))

    # merge bound data.frames into icers data.frame
    icers <- icers %>%
      left_join(cost_bounds, by = "Strategy") %>%
      left_join(effect_bounds, by = "Strategy") %>%
      select(.data$Strategy, .data$Cost, .data$Lower_95_Cost,
.data$Upper_95_Cost,
             .data$Effect, .data$Lower_95_Effect, .data$Upper_95_Effect,
             .data$Inc_Cost, .data$Inc_Effect, .data$ICER, .data$Status)
  }

  return(icers)
}

#' compute icers for non-dominated strategies
#'
#' @param non_d a data frame of non-dominated strategies, with columns
#' "Strategy", "Cost", and "Effect"
#'
#' @return the input dataframe with columns "Inc_Cost",
#' "Inc_Effect", and "ICER" appended
#'
#' @keywords internal
compute_icers <- function(non_d) {
  if (nrow(non_d) > 1) {
    non_d[1, "ICER"] <- NA
    for (i in 2:nrow(non_d)) {
      inc_cost <- (non_d[i, "Cost"] - non_d[i - 1, "Cost"])
      inc_effect <- (non_d[i, "Effect"] - non_d[i - 1, "Effect"])
      non_d[i, "Inc_Cost"] <- inc_cost
      non_d[i, "Inc_Effect"] <- inc_effect
      non_d[i, "ICER"] <- inc_cost / inc_effect
    }
  } else {
    # don't calculate ICER if only one strategy
    non_d[1, c("ICER", "Inc_Cost", "Inc_Effect")] <- NA
  }
  return(non_d)
}

#' Plot of ICERs
#'
#' Plots the cost-effectiveness plane for a ICER object, calculated with
\code{\link{calculate_icers}}
#' @param x Object of class \code{icers}.
#' @inheritParams add_common_aes
```

```
#' @param currency string. with currency used in the cost-effectiveness
analysis (CEA).
#' @param effect_units string. unit of effectiveness
#' @param label whether to label strategies on the efficient frontier, all
strategies, or none.
#' defaults to frontier.
#' @param label_max_char max number of characters to label the strategies
- if not NULL (the default)
#' longer strategies are truncated to save space.
#' @param plot_frontier_only only plot the efficient frontier
#' @param alpha opacity of points
#' @inheritParams ggrepel::geom_label_repel
#'
#' @return a ggplot2 object which can be modified by adding additional
geoms
#'
#' @importFrom stringr str_sub
#' @importFrom ggrepel geom_label_repel
#' @export
plot.icers <- function(x,
                       txtsize = 12,
                       currency = "$",
                       effect_units = "QALYs",
                       label = c("frontier", "all", "none"),
                       label_max_char = NULL,
                       plot_frontier_only = FALSE,
                       alpha = 1,
                       n_x_ticks = 6,
                       n_y_ticks = 6,
                       xbreaks = NULL,
                       ybreaks = NULL,
                       xlim = NULL,
                       ylim = NULL,
                       xexpand = expansion(0.1),
                       yexpand = expansion(0.1),
                       max.iter = 20000,
                       ...) {
  if (ncol(x) > 7) {
    # reformat icers class object if uncertainty bounds are present
    x <- x %>%
      select(.data$Strategy, .data$Cost, .data$Effect,
             .data$Inc_Cost, .data$Inc_Effect,
             .data$ICER, .data$Status)
  }

  # type checking
  label <- match.arg(label)

  # this is so non-dominated strategies are plotted last (on top)
  x <- arrange(x, .data$Status)

  # change status text in data frame for plotting
  d_name <- "Dominated"
  ed_name <- "Weakly Dominated"
  nd_name <- "Efficient Frontier"

  status_expand <- c("D" = d_name, "ED" = ed_name,
                     "ND" = nd_name, "ref" = nd_name)
```

```
  x$Status <- factor(status_expand[x$Status], ordered = FALSE,
                     levels = c(d_name, ed_name, nd_name))

  # linetype
  plot_lines <- c("Dominated" = "blank",
                  "Weakly Dominated" = "blank",
                  "Efficient Frontier" = "solid")

  # names to refer to in aes_
  stat_name <- "Status"
  strat_name <- "Strategy"
  eff_name <- "Effect"
  cost_name <- "Cost"

  # frontier only
  if (plot_frontier_only) {
    plt_data <- x[x$Status == nd_name, ]
  } else {
    plt_data <- x
  }

  # make plot
  icer_plot <- ggplot(plt_data, aes_(x = as.name(eff_name), y =
as.name(cost_name),
                                     shape = as.name(stat_name))) +
    geom_point(alpha = alpha, size = 2) +
    geom_line(aes_(linetype = as.name(stat_name), group =
as.name(stat_name))) +
    scale_linetype_manual(name = NULL, values = plot_lines) +
    scale_shape_discrete(name = NULL) +
    labs(x = paste0("Effect (", effect_units, ")"),
         y = paste0("Cost (", currency, ")"))

  icer_plot <- add_common_aes(icer_plot, txtsize, col = "none",
                              continuous = c("x", "y"),
                              n_x_ticks = n_x_ticks, n_y_ticks =
n_y_ticks,
                              xbreaks = xbreaks, ybreaks = ybreaks,
                              xlim = xlim, ylim = ylim,
                              xexpand = xexpand, yexpand = yexpand)

  # labeling
  if (label != "none") {
    if (!is.null(label_max_char)) {
      plt_data[, strat_name] <- str_sub(plt_data[, strat_name],
                                        start = 1L, end = label_max_char)
    }
    if (label == "all") {
      lab_data <- plt_data
    }
    if (label == "frontier") {
      lab_data <- plt_data[plt_data$Status == nd_name, ]
    }

    icer_plot <- icer_plot +
      ggrepel::geom_label_repel(data = lab_data,
                        aes_(label = as.name(strat_name)),
                        size = 3,
```

```
                      show.legend = FALSE,
                      max.iter = max.iter,
                      direction = "both")
  }
  return(icer_plot)
}

#' Create a PSA object
#'
#' @description
#' Creates an object to hold probabilistic sensitivity analysis data,
#' while checking the data for validity. The object can then be
#' used for many standard cost-effectiveness analyses (see Details below).
#'
#' @param parameters Data frame with values for each simulation (rows) and
parameter (columns).
#' The column names should be the parameter names.
#' @param cost For the data.frame, each simulation should be a row and
each strategy should be a column.
#' Naming the columns of the data frames is not necessary, as they will be
renamed with
#' the \code{strategies} vector.
#' @param effectiveness For the data.frame, each simulation should be a
row and each strategy should be a column.
#' Naming the columns of the data frames is not necessary, as they will be
renamed with
#' the \code{strategies} vector.
#' @param other_outcome data.frame containing values for another user-
defined outcome.
#' Each simulation should be a row of the data frame, and each strategy
should be a column.
#' Naming the columns of the data frames is not necessary, as they will be
renamed with
#' the \code{strategies} vector.
#' @param strategies vector with the names of the strategies. Due to
requirements in
#' certain uses of this vector, this function uses
\code{\link{make.names}} to modify
#' strategy names as necessary. It is strongly suggested that you follow
the rules
#' in the \code{\link{make.names}} help page, to avoid unexpected errors.
#'
#' @param currency symbol for the currency being used (ex. "$", "£")
#'
#' @details
#' The PSA object forms the backbone of one part of the \code{dampack}
package.
#'
#' A scatterplot of the cost-effectiveness plane may be shown by running
\code{plot}
#' on the output of \code{make_psa_obj}.
#'
#' Using this object, you may calculate:
#' \itemize{
#'   \item Cost-effectiveness acceptability curves (\code{\link{ceac}})
#'   \item Expected value of perfect information (\code{\link{calc_evpi}})
#'   \item Expected loss (\code{\link{calc_exp_loss}})
#'   \item One-way sensitivity analysis (\code{\link{owsa}})
```

```
#'   \item Two-way sensitivity analysis (\code{\link{twsa}})
#'   \item Metamodels (\code{\link{metamodel}})
#' }
#'
#' In addition, the PSA may be converted to a base-case analysis by using
\code{summary}
#' on the PSA object. The output of \code{summary} can be used in
\code{\link{calculate_icers}}.
#'
#'
#' @return An object of class \code{psa}
#'
#' @seealso \code{\link{summary.psa}}, \code{\link{plot.psa}}
#'
#' @examples
#' # psa input provided with package
#' data("example_psa")
#' psa <- make_psa_obj(example_psa$cost, example_psa$effectiveness,
#'                     example_psa$parameters, example_psa$strategies)
#'
#' # custom print and summary methods
#' print(psa)
#' summary(psa)
#'
#' # custom plot method; see ?plot.psa for options
#' plot(psa)
#'
#' @importFrom stringr str_replace
#' @export
make_psa_obj <- function(cost, effectiveness, parameters = NULL,
                         strategies = NULL, currency = "$", other_outcome
= NULL) {

  # parameter names
  parnames <- names(parameters)

  # define psa as a named list
  psa_obj <- create_sa(parameters, parnames, effectiveness, strategies,
                       cost, currency, other_outcome)

  # give classes "psa" and "sa"
  class(psa_obj) <- c("psa", class(psa_obj))
  return(psa_obj)
}

check_psa_object <- function(psa) {
  if (!inherits(psa, "psa")) {
    stop(paste0("The psa results parameter must be an object of class
`psa`.\n",
                "Please run the make_psa() function to create this
object."))
  }
}

check_df_and_coerce <- function(obj) {
  obj_name <- deparse(substitute(obj))
  if (!inherits(obj, "data.frame")) {
```

```
    warning(paste0("\'", obj_name, "\'", " is not a data frame. coercing
to data frame"))
    df <- as.data.frame(obj)
  } else {
    df <- as.data.frame(obj)
  }
  return(df)
}

#' summarize a psa object across all simulations
#'
#' @param object the psa object
#' @param calc_sds whether or not to calculate the standard deviations.
Defaults to FALSE
#' @param ... further arguments to summary (not used)
#'
#' @importFrom stats sd
#' @return a \code{data.frame} containing the mean cost and effectiveness
for each strategy and, if requested,
#' the standard deviations of the cost and effectiveness for each
strategy.
#' @export
summary.psa <- function(object, calc_sds = FALSE, ...) {

  mean_cost <- colMeans(object$cost)
  mean_effect <- colMeans(object$effectiveness)
  strat <- object$strategies
  sum_psa <- data.frame("Strategy" = strat,
                        "meanCost" = mean_cost,
                        "meanEffect" = mean_effect,
                        stringsAsFactors = FALSE)
  if (calc_sds) {
    sd_cost <- apply(object$cost, 2, sd)
    sd_effect <- apply(object$effectiveness, 2, sd)
    sum_psa[, "sdCost"] <- sd_cost
    sum_psa[, "sdEffect"] <- sd_effect
  }
  rownames(sum_psa) <- seq_len(nrow(sum_psa))
  sum_psa
}

#' Plot the psa object
#'
#' @param x the psa object
#' @param center plot the mean cost and effectiveness for each strategy.
defaults to TRUE
#' @param ellipse plot an ellipse around each strategy. defaults to TRUE
#' @param alpha opacity of the scatterplot points.
#' 0 is completely transparent, 1 is completely opaque
#' @inheritParams add_common_aes
#'
#' @importFrom ellipse ellipse
#' @import dplyr
#' @import ggplot2
#' @importFrom scales dollar_format
#' @return A \code{ggplot2} plot of the PSA, showing the distribution of
each PSA sample and strategy
#' on the cost-effectiveness plane.
```

```
#' @importFrom tidyr pivot_longer
#' @export
plot.psa <- function(x,
                     center = TRUE, ellipse = TRUE,
                     alpha = 0.2, txtsize = 12, col = c("full", "bw"),
                     n_x_ticks = 6, n_y_ticks = 6,
                     xbreaks = NULL,
                     ybreaks = NULL,
                     xlim = NULL,
                     ylim = NULL,
                     ...) {

  effectiveness <- x$effectiveness
  cost <- x$cost
  strategies <- x$strategies
  currency <- x$currency

  # expect that effectiveness and costs have strategy column names
  # removes confusing 'No id variables; using all as measure variables'
  df_cost <- suppressMessages(
    pivot_longer(cost,
                 everything(),
                 names_to = "Strategy",
                 values_to = "Cost")
  )
  df_effect <- suppressMessages(
    pivot_longer(effectiveness,
                 cols = everything(),
                 names_to = "Strategy",
                 values_to = "Effectiveness")
  )
  ce_df <- data.frame("Strategy" = df_cost$Strategy,
                      "Cost" = df_cost$Cost,
                      "Effectiveness" = df_effect$Effectiveness)

  # make strategies in psa object into ordered factors
  ce_df$Strategy <- factor(ce_df$Strategy, levels = strategies, ordered =
TRUE)

  psa_plot <- ggplot(ce_df, aes_string(x = "Effectiveness", y = "Cost",
color = "Strategy")) +
    geom_point(size = 0.7, alpha = alpha, shape = 21) +
    ylab(paste("Cost (", currency, ")", sep = ""))

  # define strategy-specific means for the center of the ellipse
  if (center) {
    strat_means <- ce_df %>%
      group_by(.data$Strategy) %>%
      summarize(Cost.mean = mean(.data$Cost),
                Eff.mean = mean(.data$Effectiveness))
    # make strategies in psa object into ordered factors
    strat_means$Strategy <- factor(strat_means$Strategy, levels =
strategies, ordered = TRUE)
    psa_plot <- psa_plot +
      geom_point(data = strat_means,
                 aes_string(x = "Eff.mean", y = "Cost.mean", fill =
"Strategy"),
                 size = 8, shape = 21, color = "black")
```

```
  }

  if (ellipse) {
    # make points for ellipse plotting
    df_list_ell <- lapply(strategies, function(s) {
      strat_specific_df <- ce_df[ce_df$Strategy == s, ]
      els <-  with(strat_specific_df,
                   ellipse::ellipse(cor(Effectiveness, Cost),
                           scale = c(sd(Effectiveness), sd(Cost)),
                           centre = c(mean(Effectiveness), mean(Cost))))
      data.frame(els, group = s, stringsAsFactors = FALSE)
    })
    df_ell <- bind_rows(df_list_ell)
    # draw ellipse lines
    psa_plot <- psa_plot + geom_path(data = df_ell,
                                     aes_string(x = "x", y = "y", colour =
"group"),
                                     size = 1, linetype = 2, alpha = 1)
  }

  # add common theme
  col <- match.arg(col)
  add_common_aes(psa_plot, txtsize, col = col, col_aes = c("color",
"fill"),
                 continuous = c("x", "y"),
                 n_x_ticks = n_x_ticks, n_y_ticks = n_y_ticks,
                 xbreaks = xbreaks, ybreaks = ybreaks,
                 xlim = xlim, ylim = ylim)
}

' A generic sensitivity analysis object
#'
#' @description This function is called by \code{\link{make_psa_obj}},
#' \code{\link{create_dsa_oneway}},
#' and \code{\link{create_dsa_oneway}}, and checks the structure of
#' each of the arguments before creating an SA object.
#'
#' @param parameters a data frame with parameter values for each model
run. Each
#' column should represent a different parameter, and each row should
represent a
#' simulation (in the same order as \code{cost} and \code{effectiveness})
#' @param parnames names for the parameters.
#' @param cost,effectiveness,other_outcome data frames containing data for
costs,
#' effectiveness or another outcome (user-defined), respectively.
#' Each simulation should be a row of the data frame, and each strategy
should be a column.
#' Naming the columns of the data frames is not necessary, as they will be
renamed with
#' the \code{strategies} vector.
#' @param strategies vector with the names of the strategies. Due to
requirements in
#' certain uses of this vector, this function uses
\code{\link{make.names}} to modify
#' strategy names as necessary. It is strongly suggested that you follow
the rules
#' in the \code{\link{make.names}} help page, to avoid unexpected errors.
```

```
#'
#' @param currency symbol for the currency being used (ex. "$", "£")
#' @return returns "sa" sensitivity analysis object.
#' @keywords internal
create_sa <- function(parameters, parnames, effectiveness, strategies,
                      cost, currency, other_outcome) {
  # checks that each is a dataframe
  if (!is.null(cost)) {
    cost <- check_df_and_coerce(cost)
  }

  if (!is.null(other_outcome)) {
    other_outcome <- check_df_and_coerce(other_outcome)
  }

  if (!is.null(effectiveness)) {
    effectiveness <- check_df_and_coerce(effectiveness)
  }

  if (!is.null(parameters)) {
    parameters <- check_df_and_coerce(parameters)
  }

  ### argument checks and definitions of other variables ###

  # costs, effectiveness, and parameters have same number of rows
  n_sim_ls <- list(effectiveness, cost, parameters, other_outcome)
  if (length(unique(unlist(lapply(n_sim_ls[!unlist(lapply(n_sim_ls,
is.null))], nrow)))) != 1) {
    stop("Among those provided, the cost, effectiveness, parameter,
         and other_outcome dataframes must all have the same number of
rows.")
  }

  # define n_sim
  n_sim <- unique(unlist(lapply(n_sim_ls[!unlist(lapply(n_sim_ls,
is.null))], nrow)))

  # costs and effectiveness have same number of columns (strategies)
  n_strategies_ls <- list(effectiveness, cost, other_outcome)
  if (length(unique(unlist(lapply(n_strategies_ls[!
unlist(lapply(n_strategies_ls, is.null))], ncol)))) != 1) {
    stop("Among those provided, the cost, effectiveness,
         and other_outcome dataframes must all have the same number of
columns.")
  }

  # define n_strategies
  n_strategies <- unique(unlist(lapply(n_strategies_ls[!
unlist(lapply(n_strategies_ls, is.null))], ncol)))

  # If the strategy names are not provided, generate a generic vector
  # with strategy names
  if (is.null(strategies)) {
    strategies <- paste(rep("Strategy_", n_strategies), seq(1,
n_strategies), sep = "")
  } else {
```

```
    # correct strategy names. they are used as data.frame column names and
in lm()
    # so they need to be syntactically valid
    new_strategies <- make.names(strategies, unique = TRUE)

    # write warning to console, so user knows that strategy name was
changed
    for (i in 1:n_strategies) {
      old_strat <- strategies[i]
      new_strat <- new_strategies[i]
      if (new_strat != old_strat) {
        warning(paste0("strategy name '", old_strat, "' was converted to
'", new_strat,
                       "' for compatibility. See ?make.names"), call. =
FALSE)
      }
    }
    # update strategies
    strategies <- new_strategies

    # make sure strategies is the same length as the number of columns
    if (n_strategies != length(strategies)) {
      stop(
        paste0("The number of columns in the cost and effectiveness",
               "matrices is different from the number of strategies
provided"))
    }
  }

  # define cost and effectiveness column names using strategies
  if (!is.null(cost)) {
    names(cost) <- strategies
  }
  if (!is.null(effectiveness)) {
    names(effectiveness) <- strategies
  }

  # define sa as a named list
  sa <- list("n_strategies" = n_strategies,
             "strategies" = strategies,
             "n_sim" = n_sim,
             "cost" = cost,
             "effectiveness" = effectiveness,
             "other_outcome" = other_outcome,
             "parameters" = parameters,
             "parnames" = parnames,
             "currency" = currency)
  class(sa) <- "sa"
  return(sa)
}

#' print a psa object
#'
#' @param x the psa object
#' @param all_strat whether or not to print the full list of strategies.
defaults to FALSE, which truncates
#' the strategy list to 5
#' @param ... further arguments to print (not used)
```

```
#'
#' @return None (invisible NULL).
#' @export
print.sa <- function(x, all_strat = FALSE, ...) {
  xclass <- class(x)
  is_ow_dsa <- "dsa_oneway" %in% xclass
  is_tw_dsa <- "dsa_twoway" %in% xclass
  is_psa <- "psa" %in% xclass
  cat("\n")
  if (is_ow_dsa) {
    cat("One-way Deterministic SA Object", "\n")
  }
  if (is_tw_dsa) {
    cat("Two-way Deterministic SA Object", "\n")
  }
  if (is_psa) {
    cat("PSA object", "\n")
  }
  cat("-----------------------------------------------", "\n")

  # cost
  cat("number of strategies (n_strategies):", x$n_strategies, "\n")
  n_trunc <- 5
  if (all_strat | (x$n_strategies <= n_trunc)) {
    s2print <- x$strategies
    msg <- ""
  } else {
    s2print <- c(x$strategies[1:n_trunc], "...")
    msg <- paste("(truncated at", n_trunc, ")")
  }
  s_collapsed <- paste(s2print, collapse = ", ")
  cat("strategies:", s_collapsed, msg, "\n")
  if (is_psa) {
    cat("number of simulations (n_sim):", x$n_sim, "\n")
  }
  cat("cost: a data frame with", nrow(x$cost), "rows and", ncol(x$cost),
"columns.", "\n")
  cat("effectiveness: a data frame with",
      nrow(x$effectiveness), "rows and",
      ncol(x$effectiveness), "columns.", "\n")
  cat("parameters: a data frame with",
      nrow(x$parameters), "rows and",
      ncol(x$parameters), "columns", "\n")
  cat("parameter names (parnames): ", paste(x$parnames, collapse = ", "),
"\n")
  cat("currency:", x$currency, "\n")
}

#' A function that is used to calculate all outcomes
#'
#' @param outcome choice of outcome
#' @param cost data frame with costs
#' @param effect data frame with effects
#' @param wtp willingness-to-pay threshold
#' @return a data.frame of the desired outcome values for each strategy
#' @keywords internal
calculate_outcome <- function(outcome = c("nhb", "nmb", "eff", "cost",
"nhb_loss",
```

```
                                          "nmb_loss", "nhb_loss_voi",
"nmb_loss_voi"),
                                cost, effect, wtp) {
  outcome <- match.arg(outcome)
  n_sim <- nrow(cost)
  if (outcome == "eff") {
    y <- effect
  } else if (outcome == "cost") {
    y <- cost
  } else {
    if (is.null(wtp)) {
      # the call. = FALSE makes the error message more clear
      stop("wtp must be provided for NHB and NMB",  call. = FALSE)
    }
    if (is.null(cost)) {
      stop("must provide cost for NHB and NMB.",  call. = FALSE)
    }
    if (outcome == "nhb") {
      y <- effect - cost / wtp
    }
    if (outcome == "nmb") {
      y <- effect * wtp - cost
    }
    if (outcome == "nhb_loss" | outcome == "nmb_loss") {
      if (outcome == "nhb_loss") {
        net_outcome <- "nhb"
      }
      if (outcome == "nmb_loss") {
        net_outcome <- "nmb"
      }
      netben <- calculate_outcome(net_outcome, cost, effect, wtp)
      max_str_rowwise <- max.col(netben)
      y <-  netben[cbind(1:n_sim, max_str_rowwise)] - netben
    }
    if (outcome == "nhb_loss_voi" | outcome == "nmb_loss_voi") {
      if (outcome == "nhb_loss_voi") {
        net_outcome <- "nhb"
      }
      if (outcome == "nmb_loss_voi") {
        net_outcome <- "nmb"
      }
      netben <- calculate_outcome(net_outcome, cost, effect, wtp)
      max_str <- which.max(colMeans(netben))
      y <- netben - netben[cbind(1:n_sim), max_str]
    }
  }
  return(y)
}

#' Calculate the expected loss at a range of willingness-to-pay thresholds
#'
#' @description
#' The expected loss is the quantification of the foregone benefits
#' when choosing a suboptimal strategy given current evidence.
#'
#' @param wtp vector of willingness to pay thresholds
#' @param psa object of class \code{psa}, produced by function
#' \code{\link{make_psa_obj}}
```

```
#'
#' @details
#' Visualize the expected loss at a variety of WTP thresholds using
\code{\link{plot.exp_loss}}.
#'
#' @return object with classes \code{exp_loss} and \code{data.frame}
#'
#' @seealso \code{\link{plot.exp_loss}}, \code{\link{make_psa_obj}}
#'
#' @references
#' \enumerate{
#' \item Alarid-Escudero F, Enns EA, Kuntz KM, Michaud TL, Jalal H.
#' "Time Traveling Is Just Too Dangerous" But Some Methods Are Worth
Revisiting:
#' The Advantages of Expected Loss Curves Over Cost-Effectiveness
Acceptability
#' Curves and Frontier. Value Health. 2019;22(5):611-618.
#' \item Eckermann S, Briggs A, Willan AR. Health technology assessment in
the
#' cost- disutility plane. Med Decis Making. 2008;28(2):172–181.
#' }
#' @examples
#' data("example_psa_obj")
#' wtp <- seq(1e4, 1e5, by = 1e4)
#' exp_loss <- calc_exp_loss(example_psa_obj, wtp)
#'
#' # can use head(), summary(), print(), etc.
#' head(exp_loss)
#'
#' # plot an expected loss curve (ELC)
#' plot(exp_loss)
#'
#' # the y axis is on a log scale by default
#' plot(exp_loss, log_y = FALSE)
#' @importFrom tidyr pivot_longer
#' @export
calc_exp_loss <- function(psa, wtp) {
  check_psa_object(psa)
  cost <- psa$cost
  effectiveness <- psa$effectiveness
  strategies <- psa$strategies
  n_str  <- psa$n_strategies
  exp_loss <- matrix(0, nrow = length(wtp), ncol = n_str)
  for (i in seq_len(length(wtp))) {
    ith_wtp <- wtp[i]
    loss <- calculate_outcome("nmb_loss", cost, effectiveness, ith_wtp)
    exp_loss[i, ] <- colMeans(loss)
  }
  # optimal strategy based on lowest expected loss (max of negative
expected loss)
  # this was done because min.col isn't a function
  optimal_str <- max.col(-exp_loss)

  # Format expected loss for plotting
  exp_loss_df <- data.frame(wtp, exp_loss, strategies[optimal_str])
  colnames(exp_loss_df) <- c("WTP", strategies, "fstrat")

  # Reformat df to long format
```

```
  exp_loss_df_melt <- tidyr::pivot_longer(
    data = exp_loss_df,
    cols = !c("WTP", "fstrat"),
    names_to = "Strategy",
    values_to = "Expected_Loss"
  )

  # boolean for on frontier or not
  exp_loss_df_melt$On_Frontier <- (exp_loss_df_melt$fstrat ==
exp_loss_df_melt$Strategy)

  # drop fstrat column
  exp_loss_df_melt$fstrat <- NULL

  # order by WTP
  exp_loss_df_melt <- exp_loss_df_melt[order(exp_loss_df_melt$WTP), ]

  # remove rownames
  rownames(exp_loss_df_melt) <- NULL

  # make strategies in exp_loss object into ordered factors
  exp_loss_df_melt$Strategy <- factor(exp_loss_df_melt$Strategy, levels =
strategies, ordered = TRUE)

  class(exp_loss_df_melt) <- c("exp_loss", "data.frame")
  return(exp_loss_df_melt)
}


#' Plot of Expected Loss Curves (ELC)
#'
#' @param x object of class \code{exp_loss}, produced by function
#'  \code{\link{calc_exp_loss}}
#' @param currency string with currency used in the cost-effectiveness
analysis (CEA).
#'  Default: $, but it could be any currency symbol or word (e.g., £, €,
peso)
#' @param effect_units units of effectiveness. Default: QALY
#' @param log_y take the base 10 log of the y axis
#' @param frontier indicate the frontier (also the expected value of
perfect information).
#' To only plot the EVPI see \code{\link{calc_evpi}}.
#' @param points whether to plot points on the curve (TRUE) or not (FALSE)
#' @param lsize line size. defaults to 1.
#' @inheritParams add_common_aes
#'
#' @return A \code{ggplot2} object with the expected loss
#' @import ggplot2
#' @importFrom scales comma
#' @export
plot.exp_loss <- function(x,
                          log_y = TRUE,
                          frontier = TRUE,
                          points = TRUE,
                          lsize = 1,
                          txtsize = 12,
                          currency = "$",
                          effect_units = "QALY",
```

```
                          n_y_ticks = 8,
                          n_x_ticks = 20,
                          xbreaks = NULL,
                          ybreaks = NULL,
                          xlim = c(0, NA),
                          ylim = NULL,
                          col = c("full", "bw"),
                          ...) {
  wtp_name <- "WTP_thou"
  loss_name <- "Expected_Loss"
  strat_name <- "Strategy"
  x[, wtp_name] <- x$WTP / 1000

  # split into on frontier and not on frontier
  nofront <- x
  front <- x[x$On_Frontier, ]

  # Drop unused levels from strategy names
  nofront$Strategy <- droplevels(nofront$Strategy)
  front$Strategy <- droplevels(front$Strategy)
  # formatting if logging the y axis
  if (log_y) {
    tr <- "log10"
  } else {
    tr <- "identity"
  }

  p <- ggplot(data = nofront, aes_(x = as.name(wtp_name),
                                   y = as.name(loss_name))) +
    xlab(paste0("Willingness to Pay (Thousand ", currency, "/",
effect_units, ")")) +
    ylab(paste0("Expected Loss (", currency, ")"))

  # color
  col <- match.arg(col)
  ## change linetype too if color is black and white
  if (col == "full") {
    if (points) {
      p <- p + geom_point(aes_(color = as.name(strat_name)))
    }
    p <- p +
      geom_line(size = lsize, aes_(color = as.name(strat_name)))

  }
  if (col == "bw") {
    if (points) {
      p <- p + geom_point()
    }
    p <- p +
      geom_line(aes_(linetype = as.name(strat_name)))
  }

  p <- add_common_aes(p, txtsize, col = col, col_aes = c("color", "line"),
                      continuous = c("x", "y"),
                      n_x_ticks = n_x_ticks, n_y_ticks = n_y_ticks,
                      xbreaks = xbreaks, ybreaks = ybreaks,
                      xlim = xlim, ylim = ylim,
                      ytrans = tr)
```

```
  if (frontier) {
    p <- p + geom_point(data = front, aes_(x = as.name(wtp_name),
                                           y = as.name(loss_name),
                                           shape =
as.name("On_Frontier")),
                        size = 3, stroke = 1, color = "black") +
      scale_shape_manual(name = NULL, values = 0, labels = "Frontier &
EVPI") +
      guides(color = guide_legend(order = 1),
             linetype = guide_legend(order = 1),
             shape = guide_legend(order = 2))
  }
  return(p)
}

#' Expected Value of Perfect Information (EVPI)
#'
#' \code{calc_evpi} is used to compute the expected value of perfect
information
#' (EVPI) from a probabilistic sensitivity analysis (PSA) dataset.
#' @param wtp numeric vector with willingness-to-pay (WTP) thresholds
#' @param psa psa object from \code{\link{make_psa_obj}}
#' @param pop scalar that corresponds to the total population
#' @keywords expected value of perfect information; net monetary benefit
#' @section Details:
#' \code{evpi} calculates the value of eliminating all the uncertainty of
a
#' cost-effectiveness analysis at each WTP threshold.
#' @return A data frame and \code{evpi} object with the EVPI at each WTP
threshold.
#' @seealso \code{\link{plot.evpi}}, \code{\link{make_psa_obj}}
#' @examples
#' # load psa object provided with package
#' data("example_psa_obj")
#'
#' # define wtp threshold vector (can also use a single wtp)
#' wtp <- seq(1e4, 1e5, by = 1e4)
#' evpi <- calc_evpi(example_psa_obj, wtp)
#' plot(evpi) # see ?plot.evpi for options
#'
#' # can use plot options (# see ?plot.evpi for details)
#' plot(evpi, effect_units = "QALE")
#'
#' # or can use ggplot layers
#' plot(evpi) + ggtitle("Expected Value of Perfect Information")
#' @export
calc_evpi <- function(psa, wtp, pop = 1) {
  check_psa_object(psa)
  cost <- psa$cost
  effectiveness <- psa$effectiveness
  if (ncol(effectiveness) < 2) {
    stop("You need at least two different strategies to compute EVPI.")
  }
  # number of wtp thresholds
  n_wtps <- length(wtp)
  # vector to store evpi
  evpi <- rep(0, n_wtps)
  # Estimate the Loss matrix and EVPI at each WTP threshold
```

```
  for (l in 1:n_wtps) {
    ## Calculate the opportunity loss from choosing d.star for each
strategy
    loss <- calculate_outcome("nmb_loss", cost, effectiveness, wtp[l])

    ## Compute EVPI
    evpi[l] <- min(apply(loss, 2, mean)) * pop
  }

  # Data frame to store EVPI for each WTP threshold
  df_evpi <- data.frame("WTP" = wtp, "EVPI" = evpi)

  # declare class as both evpi (plotting) and data.frame (printing)
  class(df_evpi) <- c("evpi", "data.frame")
  return(df_evpi)
}

#' Plot of Expected Value of Perfect Information (EVPI)
#'
#' @description
#' Plots the \code{evpi} object created by \code{\link{calc_evpi}}.
#'
#' @param x object of class \code{evpi}, produced by function
#'  \code{\link{calc_evpi}}
#' @param currency string with currency used in the cost-effectiveness
analysis (CEA).
#'  Default: $, but it could be any currency symbol or word (e.g., £, €,
peso)
#' @param effect_units units of effectiveness. Default: QALY
#' @inheritParams add_common_aes
#' @keywords expected value of perfect information
#' @return A \code{ggplot2} plot with the EVPI
#' @seealso \code{\link{calc_evpi}}
#' @import ggplot2
#' @importFrom scales comma
#' @export
plot.evpi <- function(x,
                      txtsize = 12,
                      currency = "$",
                      effect_units = "QALY",
                      n_y_ticks = 8,
                      n_x_ticks = 20,
                      xbreaks = NULL,
                      ybreaks = NULL,
                      xlim = c(0, NA),
                      ylim = NULL,
                      ...) {
  x$WTP_thou <- x$WTP / 1000
  g <- ggplot(data = x,
              aes_(x = as.name("WTP_thou"), y = as.name("EVPI"))) +
    geom_line() +
    xlab(paste("Willingness to Pay (Thousand ", currency, "/",
effect_units, ")", sep = "")) +
    ylab(paste("EVPI (", currency, ")", sep = ""))
  add_common_aes(g, txtsize, continuous = c("x", "y"),
                 n_x_ticks = n_x_ticks, n_y_ticks = n_y_ticks,
                 xbreaks = xbreaks, ybreaks = ybreaks,
                 xlim = xlim, ylim = ylim)
```

```
}

#' Adds aesthetics to all plots to reduce code duplication
#'
#' @param gplot a ggplot object
#' @param txtsize base text size
#' @param scale_name how to name scale. Default inherits from variable
name.
#' @param col either none, full color, or black and white
#' @param col_aes which aesthetics to modify with \code{col}
#' @param lval color lightness - 0 to 100
#' @param greystart between 0 and 1. used in greyscale only. smaller
numbers are lighter
#' @param greyend between 0 and 1, greater than greystart.
#' @param continuous which axes are continuous and should be modified by
this function
#' @param n_x_ticks,n_y_ticks number of axis ticks
#' @param xbreaks,ybreaks vector of axis breaks.
#' will override \code{n_x_ticks} and/or \code{n_y_ticks} if provided.
#' @param facet_lab_txtsize text size for plot facet labels
#' @param xlim,ylim vector of axis limits, or NULL, which sets limits
automatically
#' @param xtrans,ytrans transformations for the axes. See
\code{\link[ggplot2]{scale_continuous}} for details.
#' @param xexpand,yexpand Padding around data. See \code{\link[ggplot2]
{scale_continuous}} for details.
#' The default behavior in ggplot2 is \code{expansion(0.05)}. See
\code{\link[ggplot2]{expansion}}
#' for how to modify this.
#' @param ... further arguments to plot.
#' This is not used by \code{dampack} but required for generic
consistency.
#' @return a \code{ggplot2} plot updated with a common aesthetic
#'
#' @import ggplot2
#' @keywords internal
add_common_aes <- function(gplot, txtsize, scale_name = waiver(),
                           col = c("none", "full", "bw"),
                           col_aes = c("fill", "color"),
                           lval = 50,
                           greystart = 0.2,
                           greyend = 0.8,
                           continuous = c("none", "x", "y"),
                           n_x_ticks = 6,
                           n_y_ticks = 6,
                           xbreaks = NULL,
                           ybreaks = NULL,
                           xlim = NULL,
                           ylim = NULL,
                           xtrans = "identity",
                           ytrans = "identity",
                           xexpand = waiver(),
                           yexpand = waiver(),
                           facet_lab_txtsize = NULL,
                           ...) {
  p <- gplot +
    theme_bw() +
    theme(legend.title = element_text(size = txtsize),
```

```
          legend.text = element_text(size = txtsize - 3),
          title = element_text(face = "bold", size = (txtsize + 2)),
          axis.title.x = element_text(face = "bold", size = txtsize - 1),
          axis.title.y = element_text(face = "bold", size = txtsize - 1),
          axis.text.y = element_text(size = txtsize - 2),
          axis.text.x = element_text(size = txtsize - 2),
          strip.text.x = element_text(size = facet_lab_txtsize),
          strip.text.y = element_text(size = facet_lab_txtsize))

  col <- match.arg(col)
  col_aes <- match.arg(col_aes, several.ok = TRUE)
  if (col == "full") {
    if ("color" %in% col_aes) {
      p <- p +
        scale_color_discrete(name = scale_name, l = lval,
                             aesthetics = "color",
                             drop = FALSE)
    }
    if ("fill" %in% col_aes) {
      p <- p +
        scale_fill_discrete(name = scale_name, l = lval,
                            aesthetics = "fill",
                            drop = FALSE)
    }
  }
  if (col == "bw") {
    if ("color" %in% col_aes) {
      p <- p +
        scale_color_grey(name = scale_name, start = greystart, end =
greyend,
                         aesthetics = "color",
                         drop = FALSE)
    }
    if ("fill" %in% col_aes) {
      p <- p +
        scale_fill_grey(name = scale_name, start = greystart, end =
greyend,
                        aesthetics = "fill",
                        drop = FALSE)
    }
  }

  # axes and axis ticks
  continuous <- match.arg(continuous, several.ok = TRUE)

  if ("x" %in% continuous) {
    if (!is.null(xbreaks)) {
      xb <- xbreaks
    } else {
      xb <- number_ticks(n_x_ticks)
    }
    p <- p +
      scale_x_continuous(breaks = xb,
                         labels = labfun,
                         limits = xlim,
                         trans = xtrans,
                         expand = xexpand)
  }
```

```
  if ("y" %in% continuous) {
    if (!is.null(ybreaks)) {
      yb <- ybreaks
    } else {
      yb <- number_ticks(n_y_ticks)
    }
    p <- p +
      scale_y_continuous(breaks = yb,
                         labels = labfun,
                         limits = ylim,
                         trans = ytrans,
                         expand = yexpand)
  }
  return(p)
}

#' used to automatically label continuous scales
#' @keywords internal
#' @param x axis breaks
#' @return  a character vector giving a label for each input value
labfun <- function(x) {
  if (any(x > 999, na.rm = TRUE)) {
    scales::comma(x)
  } else {
    x
  }
}

#' Number of ticks for \code{ggplot2} plots
#'
#' Function for determining number of ticks on axis of \code{ggplot2}
plots.
#' @param n integer giving the desired number of ticks on axis of
#' \code{ggplot2} plots. Non-integer values are rounded down.
#' @section Details:
#' Based on function \code{pretty}.
#' @return a vector of axis-label breaks
#' @export
number_ticks <- function(n) {
  function(limits) {
    pretty(limits, n + 1)
  }
}

# Function that returns the lower case name of the operating system we're
running on
# Source: https://www.r-bloggers.com/identifying-the-os-from-r/
get_os <- function(){
  sysinf <- Sys.info()
  if (!is.null(sysinf)){
    os <- sysinf['sysname']
    if (os == 'Darwin')
      os <- "osx"
  } else { ## mystery machine
    os <- .Platform$OS.type
    if (grepl("^darwin", R.version$os))
      os <- "osx"
    if (grepl("linux-gnu", R.version$os))
```

```
      os <- "linux"
  }
  tolower(os)
}
```