Generating and calculating causal models for Limpopo

Eike Luedeling, Cory Whitney

This dcument oputlines a collection of holistic modeling techniques aimed at describing the link between river flows and livelihoods. We build our models using environmental flows (eflows) of water provided within a river or wetland to maintain aquatic ecosystems. These can also be thought of as ‘ecological water demand’ effectively a balance between water resources development and the need to protect freshwater-dependent ecosystems. The overall objective of the model we outline here is to simulate how river flow, in particular e-flow, impacts smallholder agriculture. The objective is to make the linkage between sustainable eflows in the rivers, and the water-requirements of sustainable agriculture. The simulation offers insights into the role that river flows play in the ability of agriculture to be sustainable, and the consequent risks to agriculture when river flows are either optimal or when they become marginal. The simulations show a strong positive link between eflows and livelihoods and provide justification for understanding eflows as more than just a mechanism for positive biodiversity outcomes.

Here we generate a holistic model to simulate the contribution of eflows to sustainable agriculture, food security and livelihoods. Spatially, we do this for only a small portion of the basin as a test-case. We apply holistic modeling approaches to generate conceptual impact pathways and quantitative models to forecast decision outcomes (see Do, Luedeling, and Whitney 2020; Lanzanova et al. 2019; Cory Whitney et al. 2018). This includes collaborative model development (C. Whitney, Shepherd, and Luedeling 2018) to assess farming futures given eflow forecasts under different management options. To build these simulations we use functions from the decisionSupport (Luedeling et al. 2021), dplyr (Wickham, François, et al. 2021), nasapower (Sparks 2021), patchwork (Pedersen 2020), tidyverse (Wickham 2021) and Evapotranspiration (Guo, Westra, and Peterson 2020) libraries in the R programming language (R Core Team 2021).

## The model

Decision-makers often wish to have a quantitative basis for their decisions. However,‘hard data’ is often missing or unattainable for many important variables, which can paralyze the decision-making processes or lead decision-makers to conclude that large research efforts are needed before a decision can be made. That is, many variables decision makers must consider cannot be precisely quantified, at least not without unreasonable effort. The major objective of (prescriptive) decision analysis is to support decision-making processes where decision makers are faced with this problem. Following the principles of Decision Analysis can allow us to make forecasts of decision outcomes without precise numbers, as long as probability distributions describing the possible values for all variables can be estimated.

The decisionSupport package implements this as a Monte Carlo simulation, which generates a large number of plausible system outcomes, based on random numbers for each input variable that are drawn from user-specified probability distributions. This approach is useful for determining whether a clearly preferable course of action can be delineated based on the present state of knowledge without the need for further information. If the distribution of predicted system outcomes does not imply a clearly preferable decision option, variables identified as carrying decision-relevant uncertainty can then be targeted by decision-supporting research.

The mcSimulation function from the decisionSupport package can be applied to conduct decision analysis (Luedeling et al. 2021). The function requires three inputs:

1. an estimate of the joint probability distribution of the input variables. These specify the names and probability distributions for all variables used in the decision model. These distributions aim to represent the full range of possible values for each component of the model.
2. a model\_function that predicts decision outcomes based on the variables named in a separate data table. This R function is customized by the user to address a particular decision problem to provide the decision analysis model.
3. numberOfModelRuns indicating the number of times to run the model function.

These inputs are provided as arguments to the mcSimulation function, which conducts a Monte Carlo analysis with repeated model runs based on probability distributions for all uncertain variables. The data table and model are customized to fit the particulars of a specific decision.

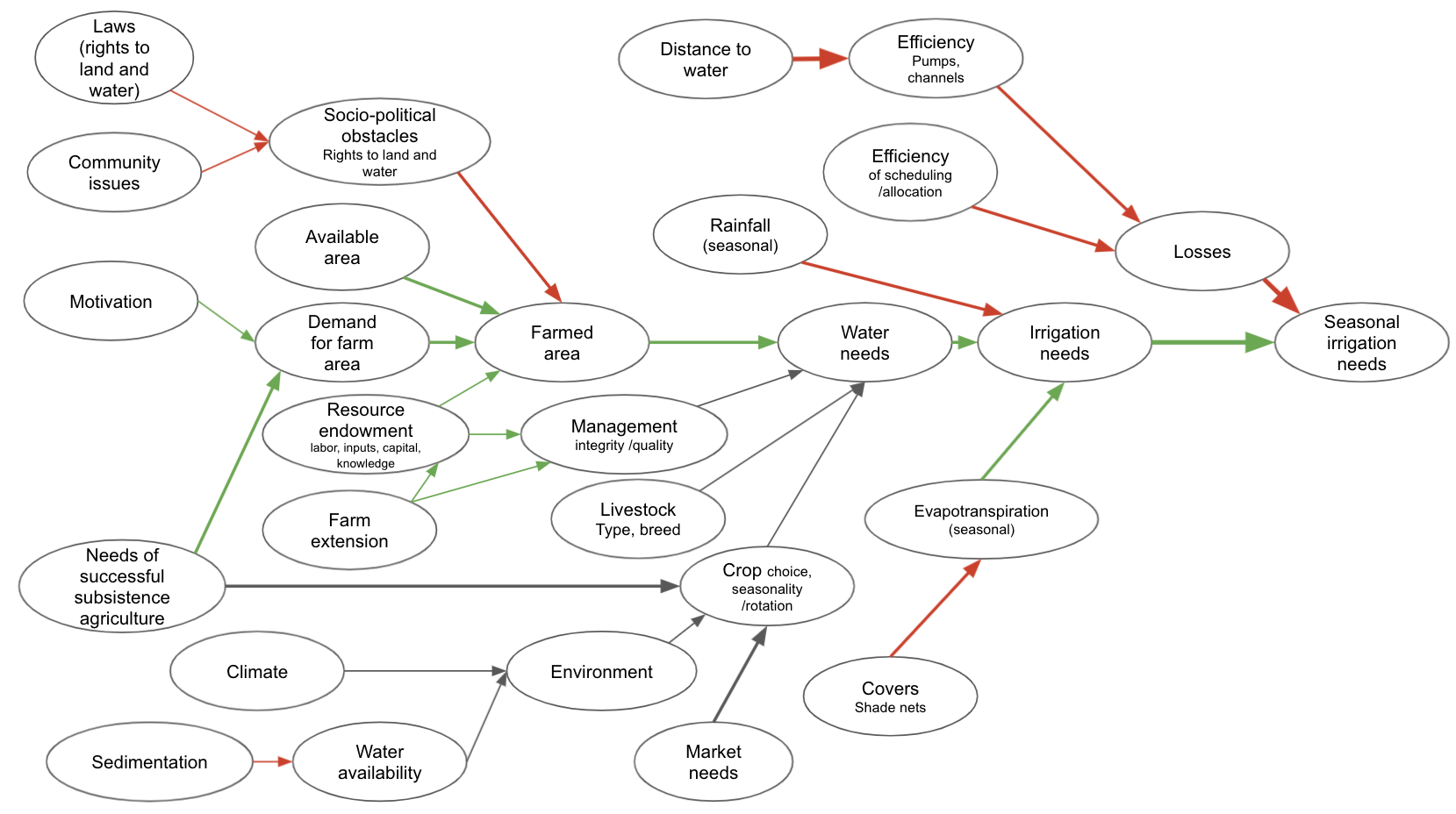
### The estimate

To support the model building process we design an input table to store the estimate values. The table is stored locally as limpopo\_input\_table.csv and contains many of the basic values for the analysis. This table contains all the input variables used in the model. Their distributions are described by 90% confidence intervals, which are specified by lower (5% quantile) and upper (95% quantile) bounds, as well as the shape of the distribution. This model uses four different distributions:

1. const – a constant value
2. norm – a normal distribution
3. tnorm\_0\_1 – a truncated normal distribution that can only have values between 0 and 1 (useful for probabilities; note that 0 and 1, as well as numbers outside this interval are not permitted as inputs)
4. posnorm – a normal distribution truncated at 0 (only positive values allowed)

For a full list of possible distributions, type ?random.estimate1d in your R console. When specifying confidence intervals for truncated distributions, note that approximately 5% of the random values should ‘fit’ within the truncation interval on either side. If there is not enough space, the function will generate a warning (usually it will still work, but the inputs may not look like you intended them to).

### The model\_function



Model of the social effects of altered river flows on the sustainability of livelihoods in the Limpopo Basin

The decision model is coded as an R function which takes in the variables provided in the data table and generates a model output, such as the Net Present Value.

In the following we use of various decisionSupport functions, which use the tidyverse libraries (Wickham et al. 2019) including ggplot2 (Wickham, Chang, et al. 2021), plyr (Wickham 2020) and dplyr (Wickham, François, et al. 2021) among others in the [R programming language](https://www.r-project.org/) (R Core Team 2021).

Here we generate a model as a function using decisionSupport library we use the decisionSupport functions vv() to produce time series with variation from a pre-defined mean and coefficient of variation, chance\_event() to simulate whether events occur and discount() to discount values along a time series and generate a Net Present Value for our intervention comparison.

### Scenarios

The following function defines 3 scenarios:

* Scenario 1 - no eflows: This is a scenario without eflows. Farmers extract water according to their irrigation needs. Extractions are only limited by the minimum water level that allows operating the pumps.
* Scenario 2 - restricted extraction: This is an eflow scenario, in which eflows are interpreted in a purely ecological sense. Whenever eflows aren’t achieved, water extraction is curtailed. There are no measures to add water to the river in such events.
* Scenario 3 - dam releases: This is an eflow scenario, in which eflows are interpreted as encompassing the ecological as well as the smallholder irrigation requirement. In case eflows aren’t naturally met, water is released from upstream dams to ensure eflows. Extraction by smallholder farmers is restricted only by the ability to operate the pumps.

limpopo\_decision\_function <- function(x, varnames){  
  
  
# generating boundary conditions for the simulation run   
  
# how much rainwater is available  
# for now we used data from some random climate diagram on the internet  
rainfall<-sapply(1:12,function(x) eval(parse(text=paste0("prec\_",x))))  
  
effective\_rainfall<-sapply(rainfall,function(x) min(x,effprec\_high))  
effective\_rainfall<-sapply(effective\_rainfall,function(x) max(x,effprec\_low))  
  
  
# We compute crop water needs based on ET0 (computed based on the Hargreaves  
# Samani equation, as implemented in the Evapotranspiration package). Input  
# temperature data comes from the NASAPOWER dataset (accessed through the   
# nasapower package)  
# The data will be based on scenarios that represent conditions during real  
# years in the past  
# To get from ET0 to crop water use, we need to multiply ET0 with a crop  
# coefficient (kc), which is estimated for each month  
  
ET0<-sapply(1:12,function(x) eval(parse(text=paste0("ET0\_",x)))) # in mm  
  
kc<-sapply(1:12,function(x) eval(parse(text=paste0("kc\_",x)))) # in mm  
  
cropwat\_need<-ET0\*kc # in mm  
  
irrigation\_need<-cropwat\_need-effective\_rainfall # in mm  
  
  
# define river flow and eflow for each month ####  
# Base river flow data from 1920 to 2010, Letaba River at EWR site EWR4 (Letaba Ranch upstream Little Letaba confluence)  
pre\_livestock\_river\_flow<-sapply(1:12,function(x) eval(parse(text=paste0("river\_flow\_",x)))) # in m3 / month  
eflow<-sapply(1:12,function(x) eval(parse(text=paste0("eflow\_",x)))) # in m3 / month  
  
# watering livestock  
# assuming that this is more or less stable throughout the year, but varies a bit  
livestock\_water\_needs<-vv(livestock\_water\_need,var\_CV,12)  
  
# assuming that the eflows aren't affecting ability to water livestock and that there's always enough  
# water for all the livestock  
river\_flow<-pre\_livestock\_river\_flow-livestock\_water\_needs  
  
# calculating the farmed area  
  
demand\_for\_farm\_area<-n\_subsistence\_farmers\*necessary\_farm\_size\_per\_household  
  
farmed\_area<-min(available\_area, demand\_for\_farm\_area)\*(1-unused\_sociopolit)  
  
total\_cropwater\_need<-cropwat\_need\*farmed\_area\*10 # total water need in m3 (the 10 is the mm to m3/ha conversion)  
total\_effective\_rainfall<-effective\_rainfall\*farmed\_area\*10 # total effective rainfall  
  
# total irrigation need  
total\_irrigation\_need<-total\_cropwater\_need-total\_effective\_rainfall # in m3  
  
# water losses are calculated from the efficiency of the pumps and the water allocation  
efficiency\_pumps<-vv(effi\_pump,var\_CV,12)  
efficiency\_irrig\_scheduling<-vv(effi\_sched,var\_CV,12)  
efficiency\_pumps<-sapply(efficiency\_pumps, function(x) min(x,1))  
efficiency\_pumps<-sapply(efficiency\_pumps, function(x) max(x,0))  
efficiency\_irrig\_scheduling<-sapply(efficiency\_irrig\_scheduling, function(x) min(x,1))  
efficiency\_irrig\_scheduling<-sapply(efficiency\_irrig\_scheduling, function(x) max(x,0))  
  
water\_losses\_share<-(1-efficiency\_pumps\*efficiency\_irrig\_scheduling)  
  
irrigation\_water\_need<-total\_irrigation\_need/(1-water\_losses\_share)  
  
# eflow scenario 1 - no eflows  
  
scen1\_usable\_river\_flow<-sapply(1:12,function(x) max(0,river\_flow[x]-minimum\_flow\_to\_operate\_pumps))  
  
# eflow scenario 2 - eflows as a limit to extraction only  
  
# eflows are to be ensured whenever there is more water in the river than the eflow  
# requirement would mandate, i.e. farmers aren't allowed to extract water beyond  
# the eflow requirement.  
# no measures are taken to ensure that eflows are maintained at times when  
# the present flow is below the eflow requirement.   
  
scen2\_usable\_river\_flow<-sapply(1:12,function(x) max(0,river\_flow[x]-max(eflow[x],minimum\_flow\_to\_operate\_pumps)))  
  
# eflow scenario 3 - eflows are assured by dam releases  
  
# whenever the present flow is below the eflow requirement, water is released  
# from an upstream dam to ensure that the eflows are met.  
  
adj\_river\_flow <- sapply(1:12, function(x)  
 max(river\_flow[x], eflow[x]))  
  
required\_dam\_release <- adj\_river\_flow - river\_flow  
  
scen3\_usable\_river\_flow <-  
 sapply(1:12, function(x)  
 max(0, adj\_river\_flow[x] - minimum\_flow\_to\_operate\_pumps))  
  
# calculate how much water gets extracted from the river  
  
scen1\_extracted\_river\_water <-  
 sapply(1:12, function(x)  
 min(scen1\_usable\_river\_flow[x], irrigation\_water\_need[x]))  
scen2\_extracted\_river\_water <-  
 sapply(1:12, function(x)  
 min(scen2\_usable\_river\_flow[x], irrigation\_water\_need[x]))  
scen3\_extracted\_river\_water <-  
 sapply(1:12, function(x)  
 min(scen3\_usable\_river\_flow[x], irrigation\_water\_need[x]))  
  
# calculate damage to crop production due to lack of irrigation water  
scen1\_water\_shortfall <-  
 sapply(1:12, function (x)  
 max(0, irrigation\_water\_need[x] - scen1\_extracted\_river\_water[x]))  
scen2\_water\_shortfall <-  
 sapply(1:12, function (x)  
 max(0, irrigation\_water\_need[x] - scen2\_extracted\_river\_water[x]))   
scen3\_water\_shortfall <-  
 sapply(1:12, function (x)  
 max(0, irrigation\_water\_need[x] - scen3\_extracted\_river\_water[x]))  
  
scen1\_irrigation\_shortfall<-scen1\_water\_shortfall\*(1-water\_losses\_share)  
scen2\_irrigation\_shortfall<-scen2\_water\_shortfall\*(1-water\_losses\_share)  
scen3\_irrigation\_shortfall<-scen3\_water\_shortfall\*(1-water\_losses\_share)  
  
scen1\_crop\_water\_gap<-scen1\_irrigation\_shortfall/(cropwat\_need\*farmed\_area\*10)  
scen2\_crop\_water\_gap<-scen2\_irrigation\_shortfall/(cropwat\_need\*farmed\_area\*10)  
scen3\_crop\_water\_gap<-scen3\_irrigation\_shortfall/(cropwat\_need\*farmed\_area\*10)  
  
# calculate how much water is left after farmers extracted water  
scen1\_river\_flow\_downstream<-river\_flow-scen1\_extracted\_river\_water  
scen2\_river\_flow\_downstream<-river\_flow-scen2\_extracted\_river\_water  
scen3\_river\_flow\_downstream<-adj\_river\_flow-scen3\_extracted\_river\_water  
  
# calculate outputs and differences   
  
return(list(scen1\_downstream\_river\_flow=scen1\_river\_flow\_downstream,  
 scen2\_downstream\_river\_flow=scen2\_river\_flow\_downstream,  
 scen3\_downstream\_river\_flow=scen3\_river\_flow\_downstream,  
 scen3\_dam\_release=required\_dam\_release,  
 Downstream\_difference\_2\_vs\_1=scen2\_river\_flow\_downstream-scen1\_river\_flow\_downstream,  
 Downstream\_difference\_3\_vs\_1=scen3\_river\_flow\_downstream-scen1\_river\_flow\_downstream,  
 scen1\_crop\_water\_gap=scen1\_crop\_water\_gap,  
 scen2\_crop\_water\_gap=scen2\_crop\_water\_gap,  
 scen3\_crop\_water\_gap=scen3\_crop\_water\_gap,  
 Crop\_water\_gap\_difference\_2\_vs\_1=scen2\_crop\_water\_gap-scen1\_crop\_water\_gap,  
 Crop\_water\_gap\_difference\_3\_vs\_1=scen3\_crop\_water\_gap-scen1\_crop\_water\_gap,  
 Mean\_Crop\_water\_gap\_difference\_2\_vs\_1=mean(scen2\_crop\_water\_gap-scen1\_crop\_water\_gap),  
 Mean\_Crop\_water\_gap\_difference\_3\_vs\_1=mean(scen3\_crop\_water\_gap-scen1\_crop\_water\_gap)))  
   
}

#### Perform a Monte Carlo simulation with scenarios

Using the model function above, we can perform a Monte Carlo simulation with the mcSimulation() function from decisionSupport. This function generates distributions of all variables in the input table as well as the specified model outputs (see return() function above) by calculating random draws in our defined limpopo\_decision\_function(). We run a visual assessment to ensure that all the variables in the input table are included in the model (erroneous variables listed there can cause issues with some of the post-hoc analyses).

The numberOfModelRuns argument is an integer indicating the number of model runs for the Monte Carlo simulation. Unless the model function is very complex, 10,000 runs is a reasonable choice (for complex models, 10,000 model runs can take a while, so especially when the model is still under development, it often makes sense to use a lower number).

We first make a scenario file, for which we can use data for 1980 to 2020.

# load data from Evapotranspiration  
data("constants")  
  
# use nasapower for evapotranspiration data  
ag\_d <- get\_power(  
 community = "ag",  
 lonlat = c(31.08,-23.7),  
 pars = c("T2M\_MAX", "T2M\_MIN", "PRECTOTCORR"),  
 dates = c("1981-01-01", "2020-12-31"),  
 temporal\_api = "daily"  
)  
  
# choose years of assessment  
years<-1981:2009  
  
# name variables  
colnames(ag\_d)[c(3:5, 8, 9, 10)] <-  
 c("Year", "Month", "Day", "Tmax", "Tmin", "Precipitation")  
  
Inputs <- ReadInputs(c("Tmin", "Tmax"), ag\_d, stopmissing = c(50, 50, 50))  
#> The maximum acceptable percentage of date indices is 50 %  
#> The maximum acceptable percentage of missing data is 50 %  
#> The maximum acceptable percentage of continuous missing data is 50 %  
  
# apply ET.HargreavesSamani from the Evapotranspiration library  
ET <-  
 ET.HargreavesSamani(  
 Inputs,  
 constants,  
 ts = "daily",  
 message = "yes",  
 AdditionalStats = "yes",  
 save.csv = "no"  
 )  
#> Hargreaves-Samani Reference Crop ET  
#> Evaporative surface: reference crop  
#> Timestep: daily  
#> Units: mm  
#> Time duration: 1981-01-01 to 2020-12-31  
#> 14610 ET estimates obtained  
#> Basic stats  
#> Mean: 4.94  
#> Max: 30.45  
#> Min: 1.35  
  
ETdata <- data.frame(year = years)  
ETdata[, month.abb[1:12]] <- NA  
for (yyyy in years)  
 ETdata[which(ETdata$year == yyyy), 2:13] <-  
 ET$ET.Monthly[as.character(yyyy + 0:11 / 12)]  
  
rain <-  
 aggregate(ag\_d$Precipitation,  
 by = list(ag\_d$Year, ag\_d$Month),  
 FUN = sum)  
raindata <- data.frame(year = years)  
raindata[, month.abb[1:12]] <- NA  
for (yyyy in years)  
 raindata[which(raindata[, 1] == yyyy), 2:13] <-  
 rain[which(rain[, 1] == yyyy), 3]  
  
  
scenario\_variables <-  
 c(paste0("river\_flow\_", 1:12),  
 paste0("ET0\_", 1:12),  
 paste0("prec\_", 1:12),  
 paste0("eflow\_", 1:12))  
  
Scenarios <- data.frame(Variable = scenario\_variables, param = "both")  
  
eflows<-read.csv("data/Letaba\_eflows\_exceedence\_m3\_per\_s.csv",fileEncoding="UTF-8-BOM")  
eflowsort <-  
 eflows[, c(1, order(unlist(sapply(colnames(eflows)[2:13], function(x)  
 which(month.abb[1:12] == x)))) + 1)]  
eflow\_exceedance<-eflowsort[which(eflowsort$Exceedence == 80),]  
eflow\_per\_month<-eflow\_exceedance[2:13]\*c(31,28,31,30,31,30,31,31,30,31,30,31)\*3600\*24  
  
# read data of present data   
present\_flows<-read.csv("data/Letaba\_modelled\_present\_flows\_m3\_per\_s.csv",fileEncoding="UTF-8-BOM")  
presentflowsort <-  
 present\_flows[, c(1, order(unlist(sapply(colnames(present\_flows)[2:13], function(x)  
 which(month.abb[1:12] == x)))) + 1)]  
presentflow\_permonth<-data.frame(cbind(presentflowsort[,1],t(t(presentflowsort[,2:13])\*c(31,28,31,30,31,30,31,31,30,31,30,31)\*3600\*24)))  
colnames(presentflow\_permonth)[1]<-"Year"  
  
# The hydrological year in the input file starts in October and runs until September. We're assuming here that the year given for each year corresponds to the first calendar year of this period.  
  
presentflow\_permonth[2:nrow(presentflow\_permonth),month.abb[1:9]]<-  
 presentflow\_permonth[1:(nrow(presentflow\_permonth)-1),month.abb[1:9]]  
  
presentflow\_permonth[1,month.abb[1:9]]<-NA  
  
for (yyyy in years)  
{  
 Scenarios[, paste0("y\_", yyyy)] <- NA  
 for (mm in 1:12)  
 {  
 Scenarios[which(Scenarios$Variable == paste0("ET0\_", mm)), paste0("y\_", yyyy)] <-  
 ETdata[which(ETdata$year == yyyy), 1 + mm]  
 Scenarios[which(Scenarios$Variable == paste0("prec\_", mm)), paste0("y\_", yyyy)] <-  
 raindata[which(raindata$year == yyyy), 1 + mm]  
 Scenarios[which(Scenarios$Variable == paste0("river\_flow\_", mm)), paste0("y\_", yyyy)] <-  
 presentflow\_permonth[which(presentflow\_permonth$Year == yyyy), 1 + mm]  
 Scenarios[which(Scenarios$Variable == paste0("eflow\_", mm)), paste0("y\_", yyyy)] <-  
 eflow\_per\_month[mm]  
 }  
}  
  
# natural flows (this is for information and not used in the model)  
natural\_flows<-read.csv("data/Letaba\_modelled\_natural\_flows\_m3\_per\_s.csv",fileEncoding="UTF-8-BOM")  
  
# write the scenarios file  
write.csv(Scenarios, "data/scenarios\_1980\_2020.csv", row.names = FALSE)

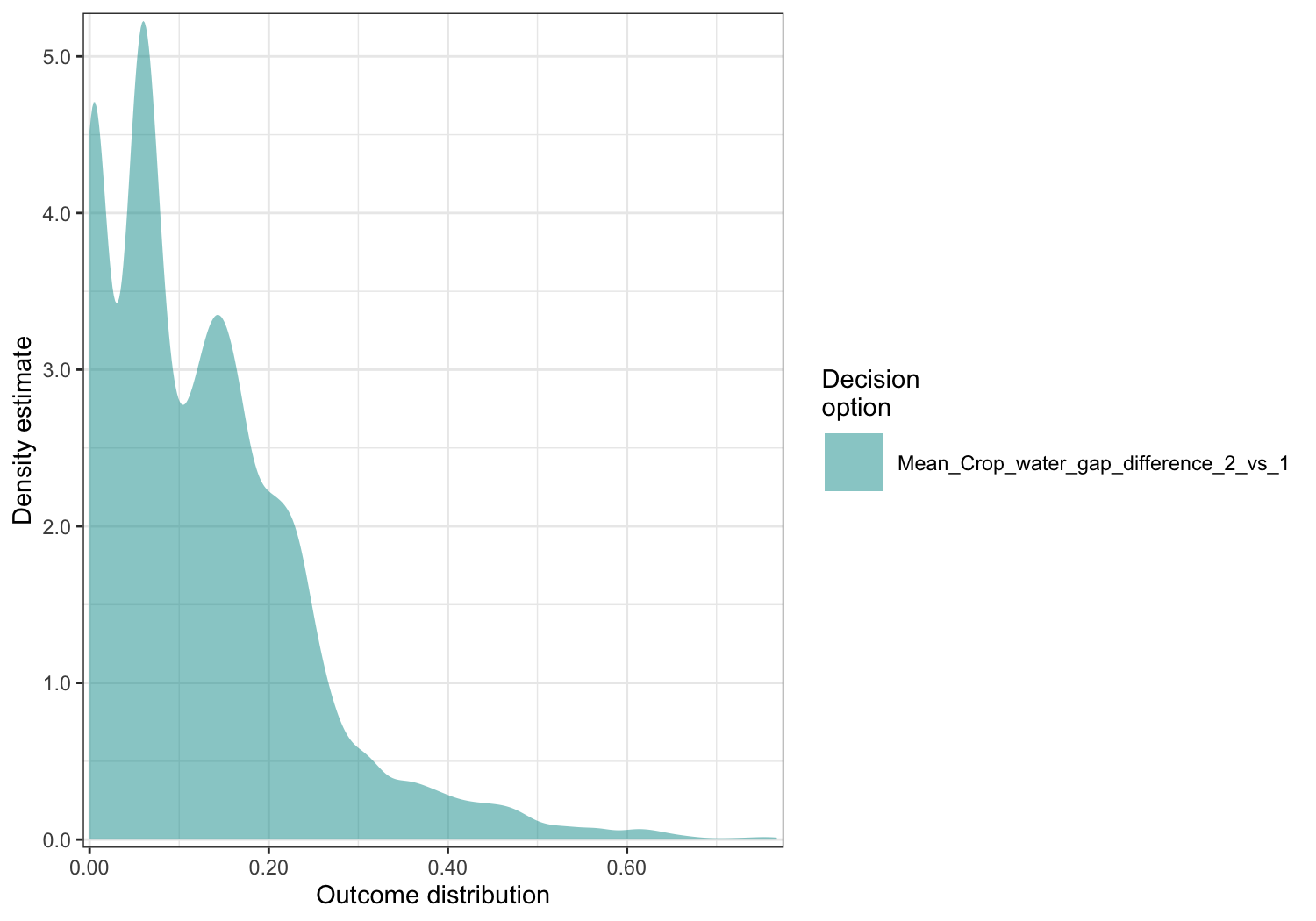
# run the model with the scenario\_mc function (a monte carlo with data from the scenarios for some inputs)  
mcSimulation\_results <-  
 scenario\_mc(  
 base\_estimate = decisionSupport::estimate\_read\_csv("data/limpopo\_input\_table.csv"),  
 scenarios = read.csv("data/scenarios\_1980\_2020.csv", fileEncoding =  
 "UTF-8-BOM"),  
 model\_function = limpopo\_decision\_function,  
 numberOfModelRuns = 2e2, #run 1,000 times  
 functionSyntax = "plainNames"  
 )

#### Plot Net Present Value (NPV) distributions

We can use the plot\_distributions() function to produce one of the several plotting options for distribution outputs. This shows us an overlay of the full results of the Monte Carlo model of the decision options, i.e. the expected NPV if we choose to do the intervention Interv\_NPV or not do the intervention NO\_Interv\_NPV.

Here’s the difference between the crop water gap in eflow scenario 2 vs. the baseline (no eflows):

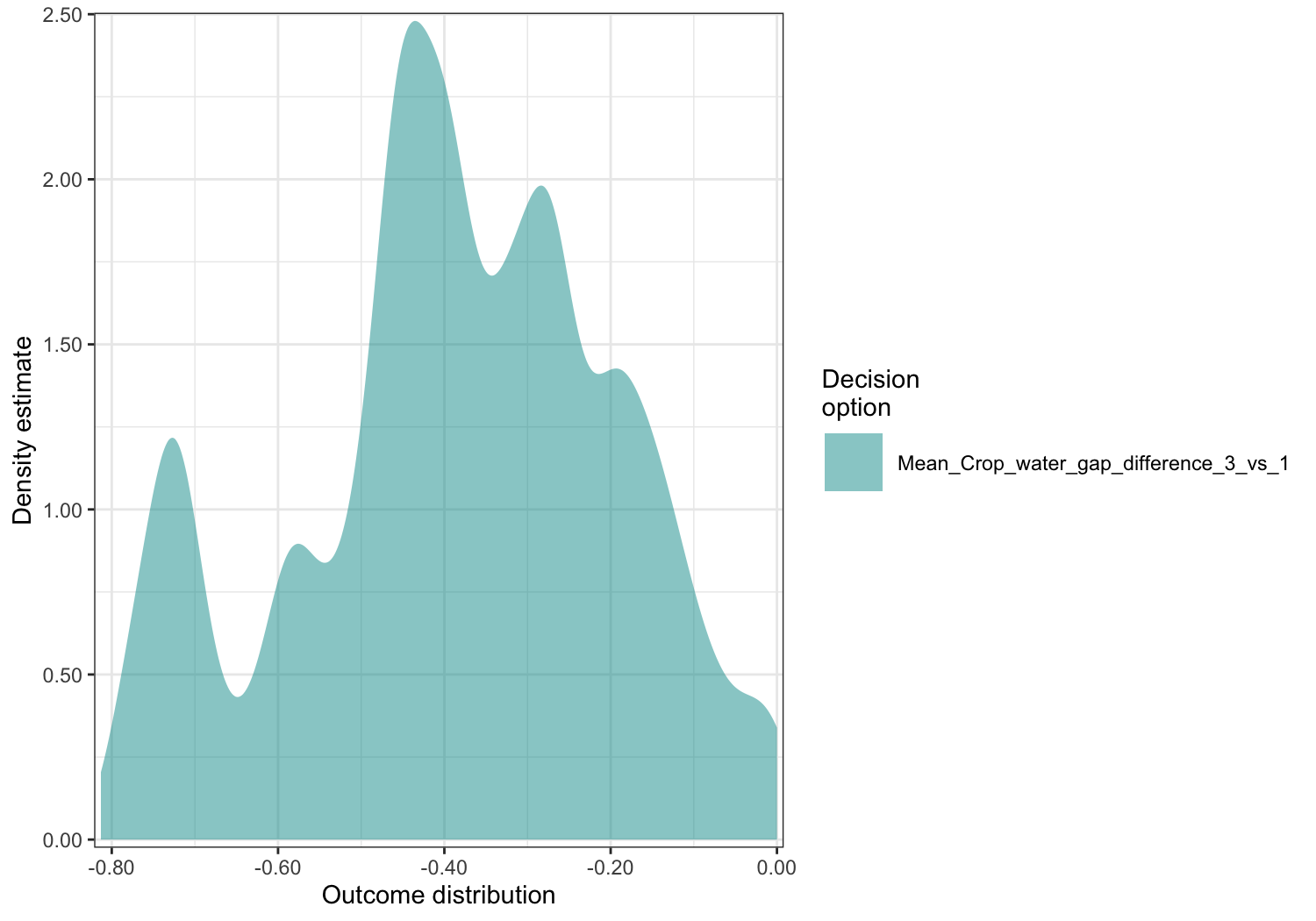
decisionSupport::plot\_distributions(mcSimulation\_object = mcSimulation\_results,  
 vars = c("Mean\_Crop\_water\_gap\_difference\_2\_vs\_1"),  
 method = 'smooth\_simple\_overlay',  
 base\_size = 7)



In short, an eflow scenario that simply curtails extraction but doesn’t alleviate sub-eflow flows, is likely to cause considerable irrigation water shortages.

Here’s the difference between the crop water gap in eflow scenario 3 (incl. dam releases) vs. the baseline (no eflows):

decisionSupport::plot\_distributions(mcSimulation\_object = mcSimulation\_results,  
 vars = c("Mean\_Crop\_water\_gap\_difference\_3\_vs\_1"),  
 method = 'smooth\_simple\_overlay',  
 base\_size = 7)



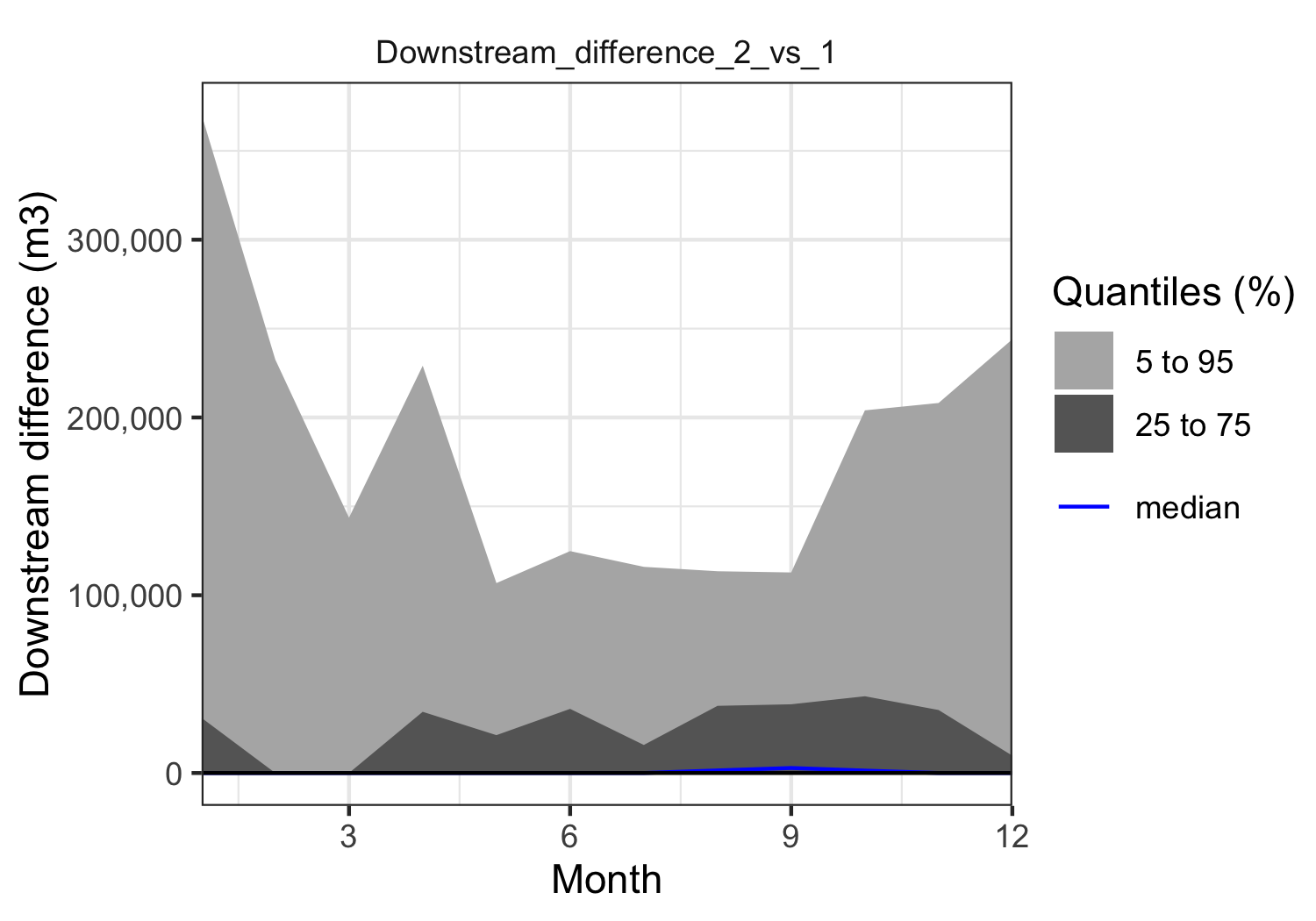
An eflow scenario that includes dam releases to ensure eflows (scenario 3) benefits farmers by effectively precluding irrigation water shortfalls.

#### Flow analysis

Here we plot the distribution of downstream flow difference over the entire simulated year. For this we use the plot\_cashflow() function, which uses the Downstream\_difference output from the mcSimulation() function to show the difference in downstream river flow over time.

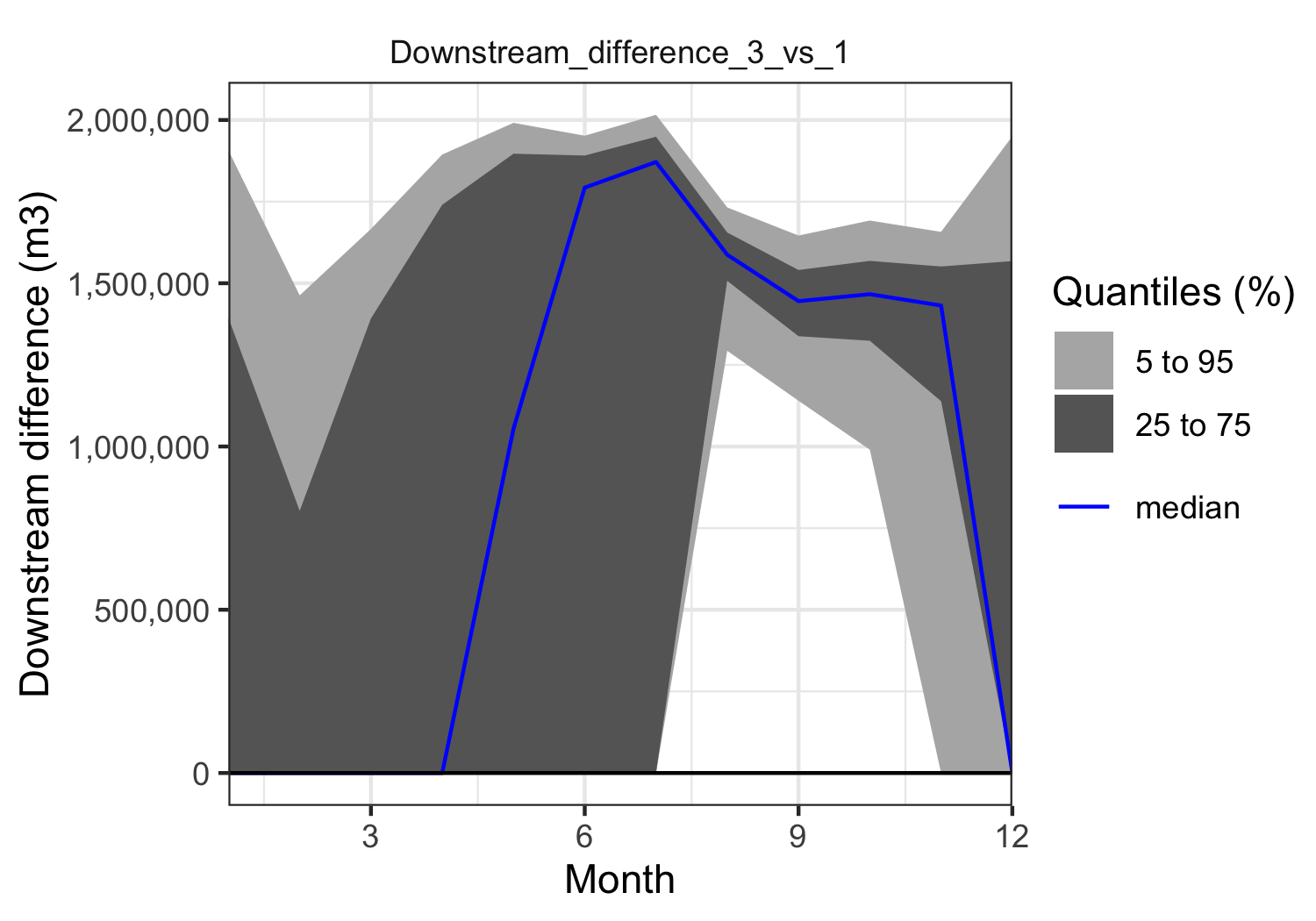
First for scenario 2 vs. the no-eflow baseline:

plot\_cashflow(mcSimulation\_object = mcSimulation\_results,   
 cashflow\_var\_name = "Downstream\_difference\_2\_vs\_1",  
 y\_axis\_name = "Downstream difference (m3)",  
 x\_axis\_name = "Month")



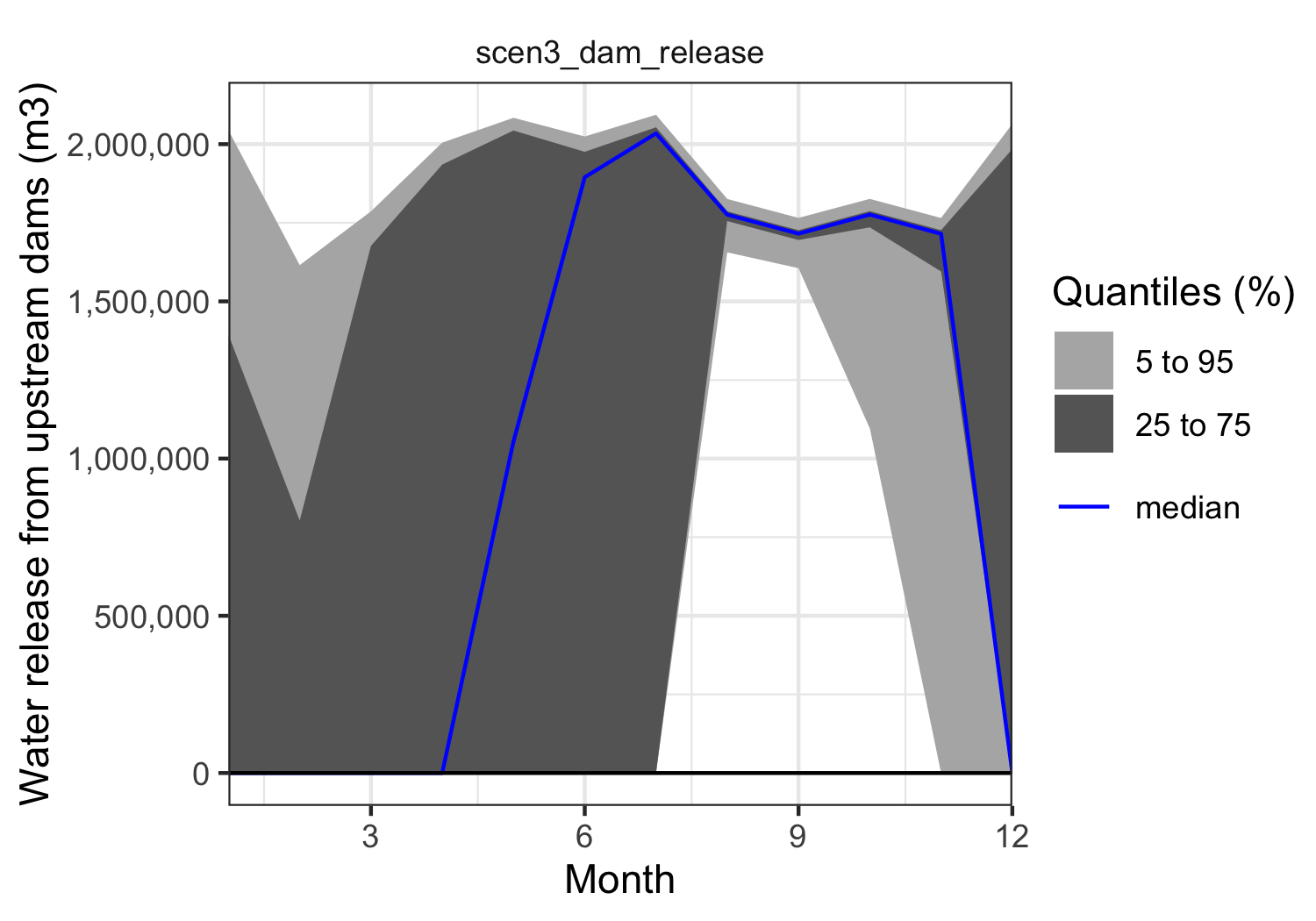
Now for scenario 3 vs. the no-eflow baseline:

plot\_cashflow(mcSimulation\_object = mcSimulation\_results,   
 cashflow\_var\_name = "Downstream\_difference\_3\_vs\_1",  
 y\_axis\_name = "Downstream difference (m3)",  
 x\_axis\_name = "Month")



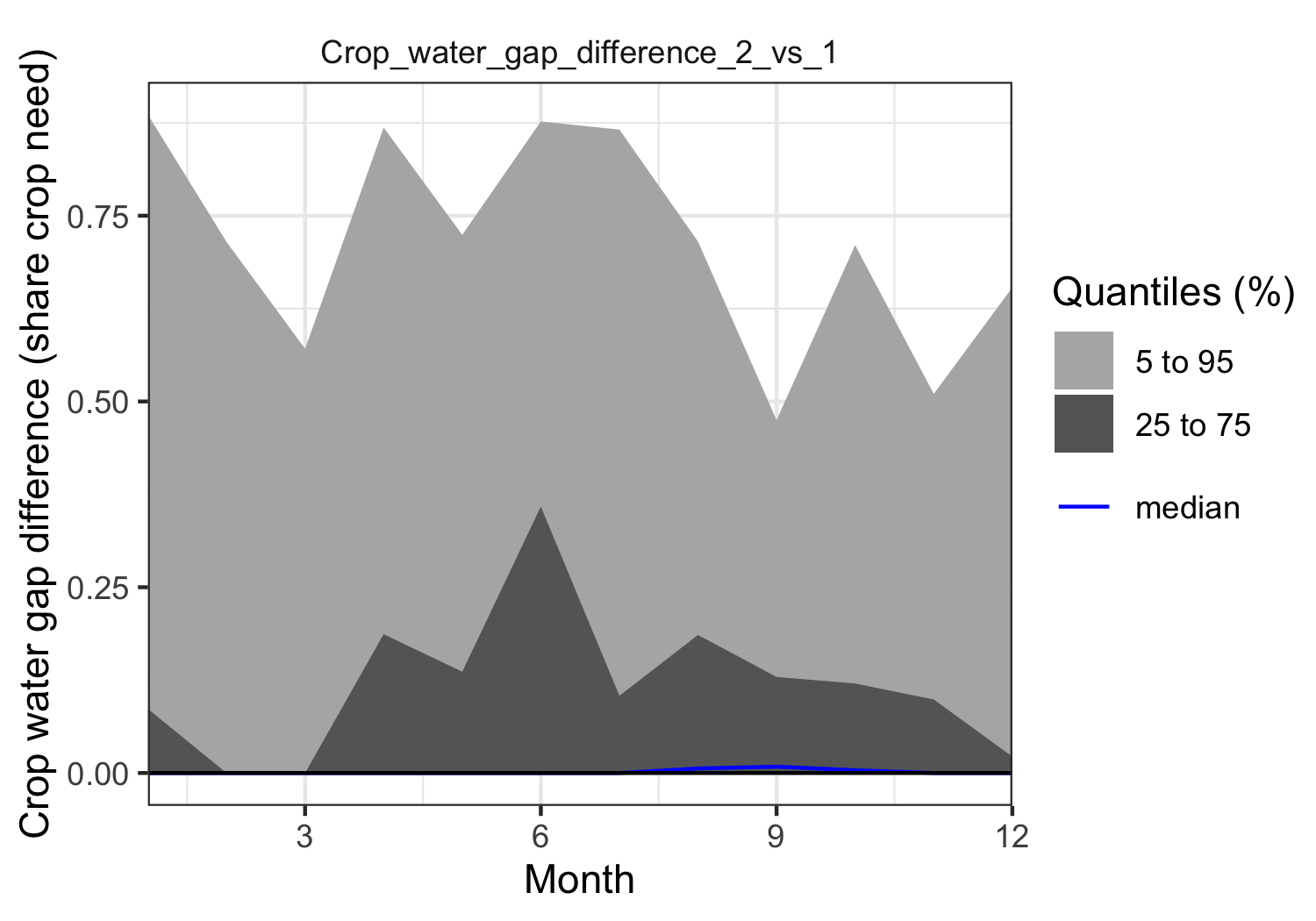
And these are the dam releases required to maintain scenario 3:

plot\_cashflow(mcSimulation\_object = mcSimulation\_results,   
 cashflow\_var\_name = "scen3\_dam\_release",  
 y\_axis\_name = "Water release from upstream dams (m3)",  
 x\_axis\_name = "Month")

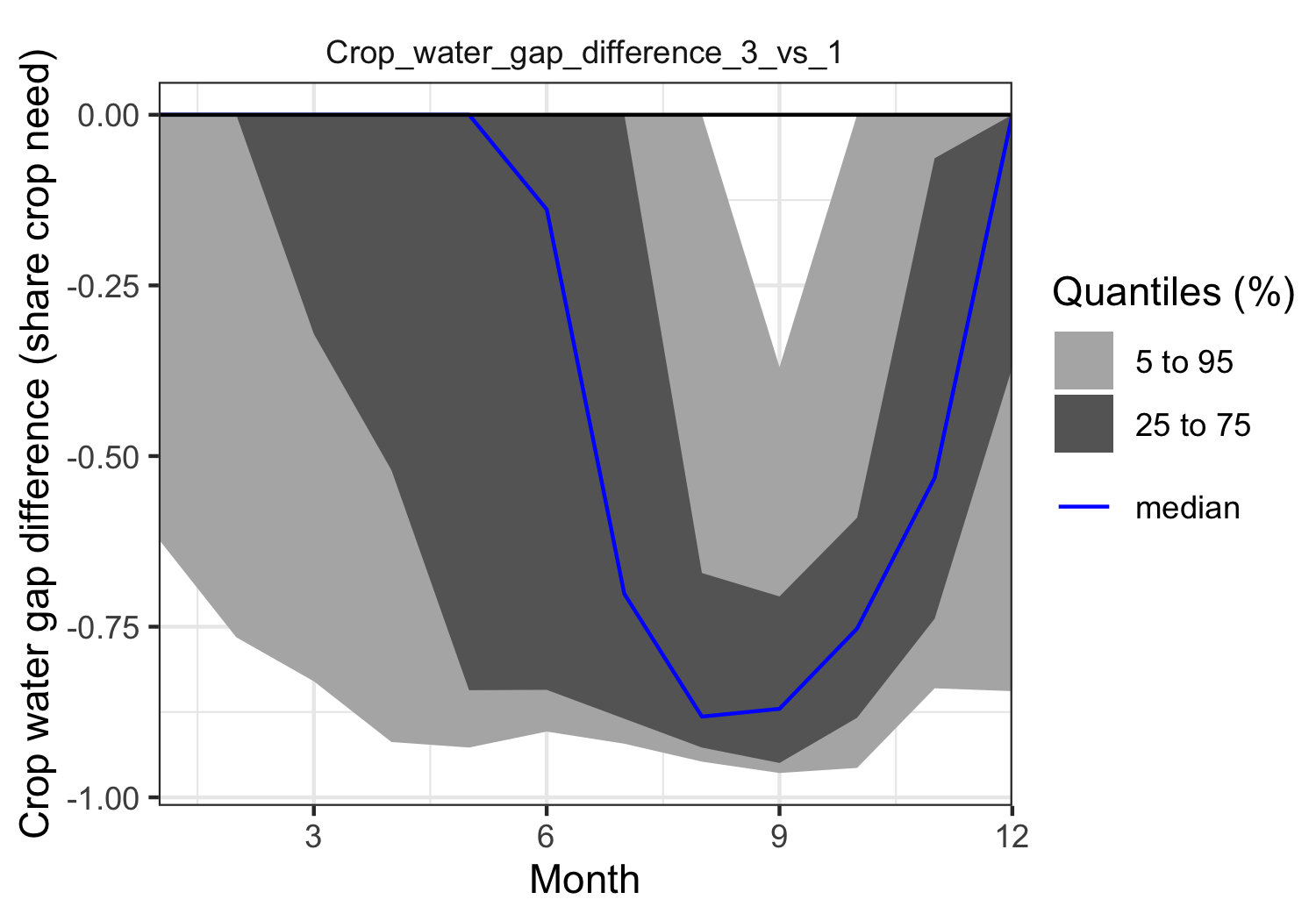


Here a similar plot of Crop\_water\_gap\_difference to show the crop water gap over time.

plot\_cashflow(mcSimulation\_object = mcSimulation\_results,   
 cashflow\_var\_name = "Crop\_water\_gap\_difference\_2\_vs\_1",  
 y\_axis\_name = "Crop water gap difference (share crop need)",  
 x\_axis\_name = "Month")



plot\_cashflow(mcSimulation\_object = mcSimulation\_results,   
 cashflow\_var\_name = "Crop\_water\_gap\_difference\_3\_vs\_1",  
 y\_axis\_name = "Crop water gap difference (share crop need)",  
 x\_axis\_name = "Month")



#### Projection to Latent Structures (PLS) analysis

Projection to Latent Structures (PLS), also sometimes known as Partial Least Squares regression is a multivariate statistical technique that can deal with multiple colinear dependent and independent variables (Wold, Sjöström, and Eriksson 2001). It can be used as another means to assess the outcomes of a Monte Carlo model. Read more in [‘A Simple Explanation of Partial Least Squares’ by Kee Siong Ng](http://users.cecs.anu.edu.au/~kee/pls.pdf).

Variable Importance in Projection (VIP) scores estimate the importance of each variable in the projection used in a PLS mode. VIP is a parameter used for calculating the cumulative measure of the influence of individual -variables on the model. For a given PLS dimension, , the squared PLS weight of that term is multiplied by the explained sum of squares () of that dimension; and the value obtained is then divided by the total explained by the PLS model and multiplied by the number of terms in the model. The final is the square root of that number.

The VIP is a weighted combination overall components of the squared PLS weights (), where is the sum of squares of explained by component , is the total number of components, and is the total number of variables. The average VIP is equal to 1 because the of all VIP values is equal to the number of variables in . A variable with a VIP Score close to or greater than 1 (one) can be considered important. The input is a PLS model and the output is a set of column vectors equal in length to the number of variables included in the model. See Galindo-Prieto, Eriksson, and Trygg (2014) for a detailed description of variations of VIP analysis.

We apply a post-hoc analysis to the mcSimulation() outputs with plsr.mcSimulation() to determine the *Variable Importance in the Projection (VIP)* score and coefficients of a *Projection to Latent Structures (PLS)* regression model. This function uses the outputs of the mcSimulation() selecting all the input variables from the decision analysis function in the parameter object and then runs a PLS regression with an outcome variable defined in the parameter resultName. We use the code names(mcSimulation\_results$y)[73] to select the outcome variable scen1\_crop\_water\_gap1, which is an element of the list y in our mcSimulation\_results outputs (this must be a character element).

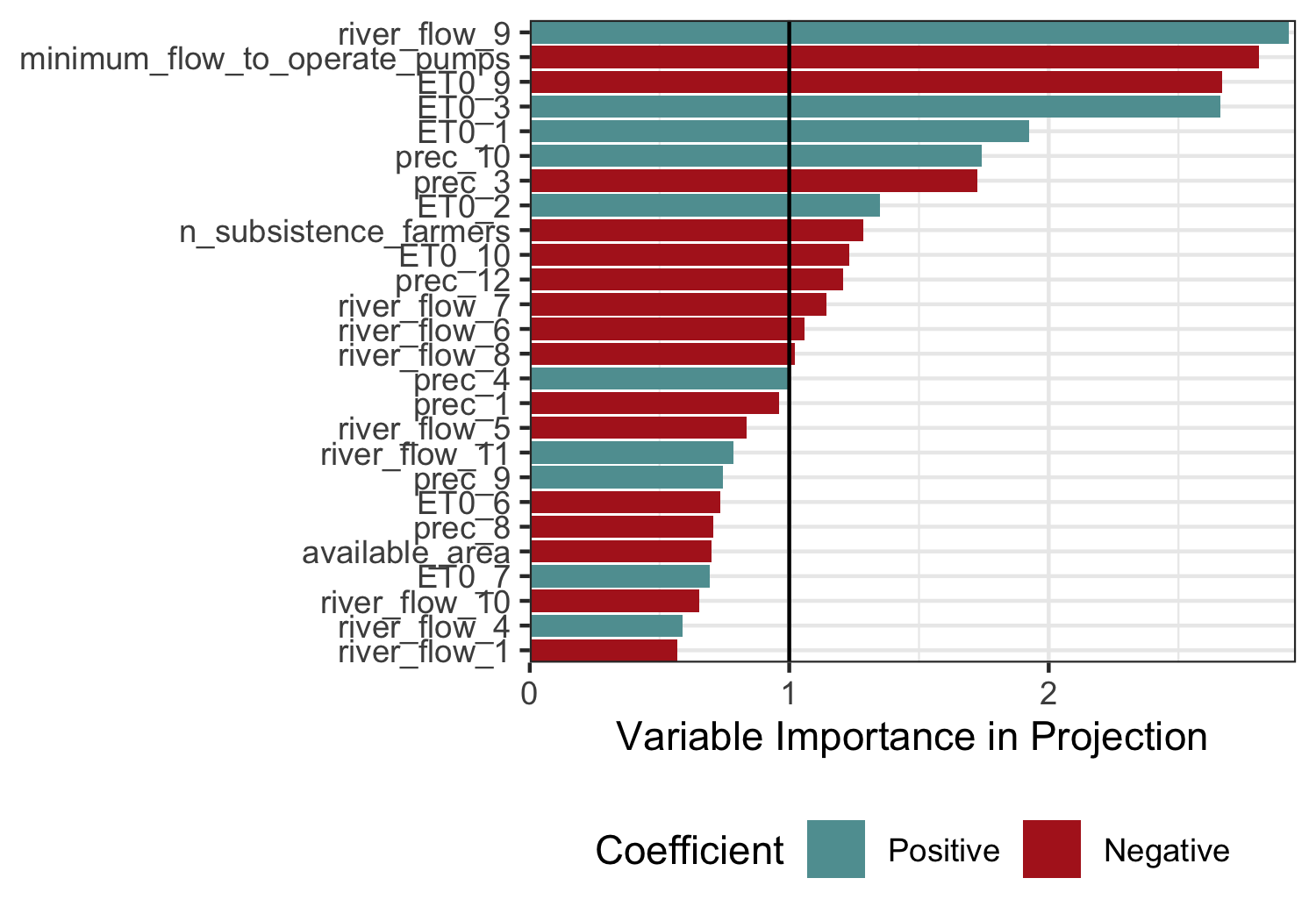
mcSimulation\_pls<-mcSimulation\_results  
mcSimulation\_pls$x<-mcSimulation\_pls$x[, !names(mcSimulation\_pls$x) == "Scenario"]  
  
pls\_result <- plsr.mcSimulation(object = mcSimulation\_pls,  
 resultName = "Mean\_Crop\_water\_gap\_difference\_2\_vs\_1", ncomp = 1)

We run the plot\_pls() on the results from plsr.mcSimulation() with a number of standard settings. The length of the bars is equal to VIP with a vertical line at ‘1’ on the x-axis indicating a standard cut-off for VIP used for variable selection. The overall plot only shows those variables with a VIP > 0.8, which is the common threshold for variable selection Luedeling and Shepherd (2016). The colors of the bars represent the positive or negative coefficient of the given input variable with the output variable.

Here we import the input table again to replace the labels for the variables on the y-axis. The input table can include a label and variable column. The standard labels (from the variable column) are usually computer readable and not very nice for a plot. The plot\_pls() function uses the text in the label column as replacement for the default text in the variable column.

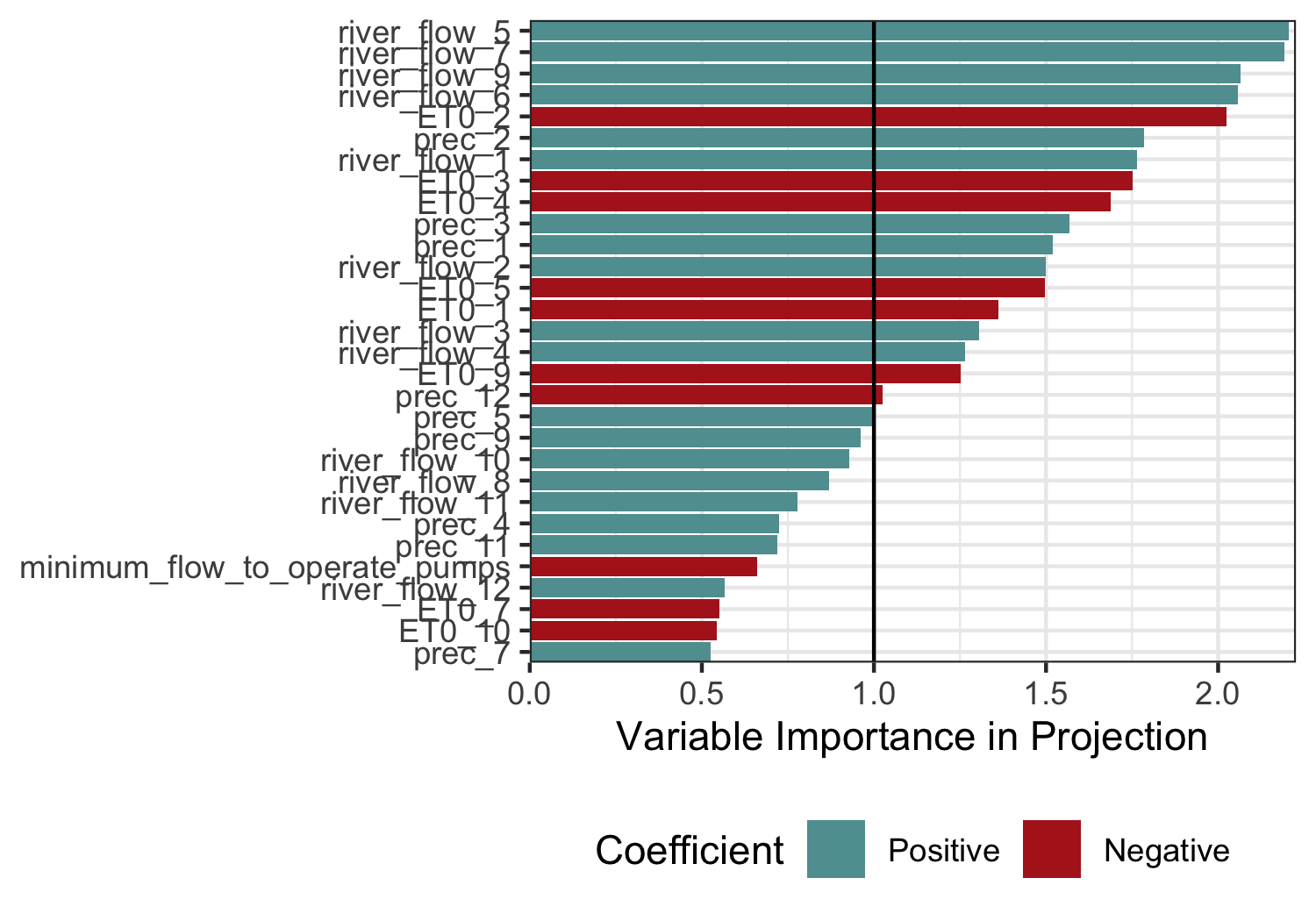
Here’s the plot for a PLS analysis for the outcome variable Mean\_Crop\_water\_gap\_difference\_2\_vs\_1:

input\_table <- read.csv("data/limpopo\_input\_table.csv")  
  
plot\_pls(pls\_result, input\_table = input\_table, threshold = 0.5)



Here’s the plot for a PLS analysis for the outcome variable Mean\_Crop\_water\_gap\_difference\_3\_vs\_1:

input\_table <- read.csv("data/limpopo\_input\_table.csv")  
  
pls\_result <- plsr.mcSimulation(object = mcSimulation\_pls,  
 resultName = "Mean\_Crop\_water\_gap\_difference\_3\_vs\_1", ncomp = 1)  
  
plot\_pls(pls\_result, input\_table = input\_table, threshold = 0.5)



### Addendum

The objective of the procedures used in the decisionSupport package is to make it easier for analysts to produce decision-relevant information that adequately reflects the imperfect nature of the information we usually have. Adding probabilistic elements to a simulation adds substantial value to an analysis. Mostly, it avoids making spurious assumptions, replacing uncertainty with ‘best bets’ and producing results that do not reflect the knowledge limitations that make decision-making so challenging. More information on all this is contained in the [decisionSupport manual](https://cran.r-project.org/web/packages/decisionSupport/decisionSupport.pdf), especially under welfareDecisionAnalysis.

### References

Do, Hoa, Eike Luedeling, and Cory Whitney. 2020. “Decision Analysis of Agroforestry Options Reveals Adoption Risks for Resource-Poor Farmers.” *Agronomy for Sustainable Development* 40 (3): 20. <https://doi.org/10.1007/s13593-020-00624-5>.

Galindo-Prieto, Beatriz, Lennart Eriksson, and Johan Trygg. 2014. “Variable Influence on Projection (VIP) for Orthogonal Projections to Latent Structures (OPLS): Variable Influence on Projection for OPLS.” *Journal of Chemometrics* 28 (8): 623–32. <https://doi.org/10.1002/cem.2627>.

Guo, Danlu, Seth Westra, and Tim Peterson. 2020. *Evapotranspiration: Modelling Actual, Potential and Reference Crop Evapotranspiration*. <https://CRAN.R-project.org/package=Evapotranspiration>.

Lanzanova, Denis, Cory Whitney, Keith Shepherd, and Eike Luedeling. 2019. “Improving Development Efficiency Through Decision Analysis: Reservoir Protection in Burkina Faso.” *Environmental Modelling & Software* 115 (May): 164–75. <https://doi.org/10.1016/j.envsoft.2019.01.016>.

Luedeling, Eike, Lutz Goehring, Katja Schiffers, Cory Whitney, and Eduardo Fernandez. 2021. *decisionSupport: Quantitative Support of Decision Making Under Uncertainty*. <http://www.worldagroforestry.org/>.

Luedeling, Eike, and Keith Shepherd. 2016. “Decision-Focused Agricultural Research.” *Solutions* 7 (5): 46–54. https://doi.org/<https://www.thesolutionsjournal.com/article/decision-focused-agricultural-research/>.

Pedersen, Thomas Lin. 2020. *Patchwork: The Composer of Plots*. <https://CRAN.R-project.org/package=patchwork>.

R Core Team. 2021. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. <https://www.R-project.org/>.

Sparks, Adam H. 2021. *Nasapower: NASA POWER API Client*. <https://docs.ropensci.org/nasapower/>.

Whitney, Cory, D. Lanzanova, C. Muchiri, K. Shepherd, T. Rosenstock, M. Krawinkel, J. R. S. Tabuti, and E. Luedeling. 2018. “Probabilistic Decision Tools for Determining Impacts of Agricultural Development Policy on Household Nutrition.” *Earth’s Future* 6 (3): 359–72. <https://doi.org/10.1002/2017EF000765/full>.

Whitney, C., K. Shepherd, and E. Luedeling. 2018. “Decision Analysis Methods Guide; Agricultural Policy for Nutrition.” *World Agroforestry (ICRAF)* Working Paper series (275): 40.

Wickham, Hadley. 2020. *Plyr: Tools for Splitting, Applying and Combining Data*. <https://CRAN.R-project.org/package=plyr>.

———. 2021. *Tidyverse: Easily Install and Load the Tidyverse*. <https://CRAN.R-project.org/package=tidyverse>.

Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D’Agostino McGowan, Romain François, Garrett Grolemund, et al. 2019. “Welcome to the tidyverse.” *Journal of Open Source Software* 4 (43): 1686. <https://doi.org/10.21105/joss.01686>.

Wickham, Hadley, Winston Chang, Lionel Henry, Thomas Lin Pedersen, Kohske Takahashi, Claus Wilke, Kara Woo, Hiroaki Yutani, and Dewey Dunnington. 2021. *Ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. <https://CRAN.R-project.org/package=ggplot2>.

Wickham, Hadley, Romain François, Lionel Henry, and Kirill Müller. 2021. *Dplyr: A Grammar of Data Manipulation*. <https://CRAN.R-project.org/package=dplyr>.

Wold, Svante, Michael Sjöström, and Lennart Eriksson. 2001. “PLS-Regression: A Basic Tool of Chemometrics.” *Chemometrics and Intelligent Laboratory Systems*, PLS Methods, 58 (2): 109–30. <https://doi.org/10.1016/S0169-7439(01)00155-1>.