Introduction to hisafer 1.5.0

hisafer is an R toolbox for the Hi-sAFe biophysical agroforestry model. It provides functions for defining, building, running, reading, analyzing, plotting, and visualizing Hi-sAFe simulaitons. The first step to using hisafer is to install and load the library, which is currently available from github. You need the devtools package to do this from within R.

```
install.packages("devtools")
library(devtools)
install_github("kevinwolz/hisafer")
```

```
library(hisafer)
```

Six Steps to Hi-sAFe Experimentation

hisafer tools are organized via the six main steps of experimentation with Hi-sAFe: Define, Build, Run, Read, Diagnostics, Analysis. While hisafer can be used to interact with Hi-sAFe at any of these steps independently, enhanced functionality is available when using hisafer to interact with Hi-sAFe from the beginning.

0. Create a TEMPLATE

Prior to defining Hi-sAFe simulations, a critical preliminary step is to create a template Hi-sAFe simulation directory. This template will be the foundation of simulations/experiments that you define in the next step. The template directory contains all files available for use when defining simulations/experiments, and the values of parameters in each file are the default values that will be used if other values are not defined in the next step.

The template directory structure for the latest version of Hi-sAFe (v3.2) has all of the following contained within a single folder:

- `template.sim` - the template .SIM file
- `template.pld` - the template .PLD file
- `weather.wth` - a default .WTH file
- `treeSpecies/` - a folder containing all of the individual .TREE files that will be available for use
- `cropSpecies/` - a folder containing all of the individual .PLT files that will be available for use
- `cropInterventions/` - a folder containing all of the individual .TEC files that will be available for use
- `generalParameters/` - a folder containing the `hisafe.par` and `stics.par` files, which contains general Hi-sAFe and STICS parameters
- `exportParameters/` - a folder containing all of the individual .PRO files that will be available for use

Luckily, hisafer already has several built-in template directories available for you. You can use these in the next step simply by providing the name of the template rather than a path to a custom template directory. The built-in templates are:

- `agroforestry` - a simple alley cropping template, with trees spaced at 5m x 7m and a durum wheat alley crop
- `forestry` - a simple forestry template, with trees spaced at 3m x 3m and an understory of bare soil
- `monocrop` - a simple, single-celled monocrop template with durum-wheat
- `restinclieres_agroforestry_A2` - a template based on the agroforestry calibration simulation of Hi-sAFe in Plot A2 at Restinclieres in Southern France
- `restinclieres_agroforestry_A3` - a template based on the agroforestry calibration simulation of Hi-sAFe in Plot A3 at Restinclieres in Southern France

- `restinclieres_forestry_A4` - a template based on the forestry calibration simulation of Hi-sAFe in Plot A4 at Restinclieres in Southern France
- `restinclieres_monocrop_A2`- a template basd on the monocrop calibration simulation of Hi-sAFe in Plot A2 at Restinclieres in Southern France
- `restinclieres_monocrop_A3`- a template basd on the monocrop calibration simulation of Hi-sAFe in Plot A3 at Restinclieres in Southern France
- `castries_agroforestry`- a template basd on the agroforestry validation simulation of Hi-sAFe at Castries in Southern France

When starting a new project, you will likely want to create your own Hi-sAFe template from which to originate experiments. To do this, it is recommended to create your custom template by copying one of the built-in templates. You can do this using hisafer via `copy_hisafe_template()`. Once you have this copied directory, you can modify the enclosed files for your project using any text editor.

1. DEFINE an experiment

In this step, you will define one or more Hi-sAFe "simulations". Simulations are definied by specifying any number of Hi-SAFe input parameters from the .SIM, .PLD, .TREE, or .PAR files that you wish to vary. Any Hi-SAFe input parameters not specified will simply inherit the default values from the files within the specfied template directory. At this time, hisafer cannot edit paramters in .PLT or .TEC files, so you will have to edit those files in your template directory using a text editor.

Hi-sAFe simluatiosn are defined using `define_hisafe()`. To define a Hi-sAFe simulation that has no modifications from your template directory, you only need to pass a few core arguments to `define_hisafe()`:

```
hip <- define_hisafe(path     = "./simulations",
                     profiles = "all",
                     template = "agroforestry")
```

The `path` argument specifies the path (relative or absolute) where the simulation is to be built in the next step. The `profiles` argument specifies which Hi-sAFe export profiles are to be exported by Hi-sAFe ("all" specifies that all available profiles will be exported). Be careful when using "all", as this will include the very large "cells" and "voxels" outputs, slowing your Hi-sAFe simulations dramatically. To see which Hi-sAFe output profiles are supported and their export frequency, use `hisafe_profiles()`. Finally, the `template` argument specifies which template directory will serve as the foundation for the defined simulation. This can be the name of one of the built-in hisafer templates, or a path to a custom template directory.

We can now see a defined Hi-sAFe simulation, which is contained within a `hip` object (for "Hi-sAFe Input Parmaeters").

```
hip
```

```
## $exp.plan
## # A tibble: 1 x 1
##   SimulationName
##   <chr>
## 1 Sim_1
##
## $template
## [1] "agroforestry"
##
## $profiles
##  [1] "annualCells"      "cells"            "cellsDetail"
##  [4] "climate"          "monthCells"       "monthCellsDetail"
##  [7] "plot"             "plotDetail"       "trees"
## [10] "treesDetail"      "voxels"           "voxels3D"
```

```
## [13] "voxelsDetail"     "voxelsMonth"
##
## $path
## [1] "/Users/kevinwolz/Desktop/RESEARCH/ACTIVE_PROJECTS/HI-SAFE/hisafer/vignettes/simulations"
##
## attr(,"class")
## [1] "hip"  "list"
```

The main feature of a `hip` object is the `exp.plan`, but we'll wait to look at that until we have a more interesting simulation. The other components of the `hip` object contain exactly the three core arguments that we just passed to `define_hisafe()`: the template name/directory, the list of export profiles, and the path where teh simulation is to be built.

Okay, let's now begin to add complexity to our simulation by modifiying parameters away from the defaults in the template. All other arguments to `define_hisafe()` can specify any number of Hi-sAFe input parameters (named, spelled, and capitalized correctly!). The only additional input parameter that is available but not part of the input files is **weatherFile**, which specifies a path to a .WTH file to use.

To see which Hi-SAFe input parameters are modifiable in your template, use `hip_params()`. You can modifty the `template` argument of `hip_params()` to be any of the other hisafer templates or a string to the path of your custom template. By default, `hip_params()` just provides the names of supported parameters. To also see the default values and range of allowed values for a parameter, just pass its name as a character string:

```
hip_params("nbSimulations")
```

```
##
##
##  nbSimulations
##    -- Default: 10
##    -- Definition: Number of simulations (i.e. number of years to simulate)
##    -- Units: NA (integer)
##    -- Accepted Range: [1, NA]
```

It looks like the agroforestry template defaults to running for 10 years. Let's start with a simple modification by changing that to 20 years and then take a look at the resulting `exp.plan`:

```
hip <- define_hisafe(path     = "./simulations",
                     profiles = "all",
                     template = "agroforestry",
                     nbSimulations = 20)
hip$exp.plan
```

```
## # A tibble: 1 x 2
##   SimulationName nbSimulations
##   <chr>                  <dbl>
## 1 Sim_1                     20
```

Each row of the `exp.plan` table describes a single Hi-sAFe simulation, and each column provides the value of a Hi-sAFe input parameter. So, we can see here that we have a single simulation defined, with **nbSimulations** as the only modified parameter. All other Hi-sAFe parameters will remain unchagned from the template directory. If not provided manually, a default **SimulationName** is provided for each simulation within the `hip` object. This will become very important during analysis so we can specify which simulations we want to analyze.

Now, let's instead modifying two input parameters at the same time by customizing **latitude** and **northOrientation**.

```
hip <- define_hisafe(path     = "./simulations",
                     profiles = "all",
```

```
                      template = "agroforestry",
                      latitude         = 60,
                      northOrientation = -90)
```

```
## Error: Hi-sAFe definition errors:
## -- northOrientation - must be betwen 0 and 359
```

Oops! It looks like **northOrientation** must be between 0 and 359. This error message is just one example of a suite of checks that `define_hisafe()` runs when attempting to define a simulation. Informative error messages are provided to help guide you in fixing any issues. Let's double check the details on the **northOrientation** parameter.

```
hip_params("northOrientation")
```

```
##
##
##  northOrientation
##   -- Default: 0
##   -- Definition: Orientation of North. The tree row is always parallel to the Y axis.
##   -- Units: degrees clockwise relative to the the +Y axis of the scene (real)
##   -- Accepted Range: [0, 359]
```

Okay, let's try again using a new value for **northOrientation**.

```
hip_sim <- define_hisafe(path     = "./simulations",
                         profiles = "all",
                         template = "agroforestry",
                         latitude         = 60,
                         northOrientation = 90)
hip_sim$exp.plan
```

```
## # A tibble: 1 x 3
##   SimulationName latitude northOrientation
##   <chr>             <dbl>            <dbl>
## 1 Sim_1                60               90
```

Great, we still have our single simulation, but now with two modified input parameters.

Let's try something more complicated now. Most Hi-sAFe simulations are run in groups that vary one or more parameters over a range of values. A group of multiple, related simulations is called an "experiment". We can also use `define_hisafe()` to define an experiment by simply providing multiple values for one or more parameters. The arguments to `define_hisafe()` are the same when defining an experiment, except that there is also an optional argument to name the experiment via `exp.name`. If not provided, a default name is used.

There are two methods for defining an experiment, depending on if the `define_hisafe()` agrument `factorial` is `TRUE` or `FALSE`.

If `factorial` is `FALSE`, the default, then parameter values are recycled (i.e. such as for default behavior of `data.frame()`).

```
hip <- define_hisafe(path      = "./simulations",
                     profiles  = "all",
                     template  = "agroforestry",
                     exp.name  = "lat-orient",
                     factorial = FALSE,
                     latitude         = c(30, 60),
                     northOrientation = c(0,  90))
hip$exp.plan
```

```
## # A tibble: 2 x 3
##   SimulationName latitude northOrientation
##   <chr>             <dbl>            <dbl>
## 1 Sim_1                30                0
## 2 Sim_2                60               90
```

In this case, you can see that there are two simulations in the resulting `hip`, one with **latitude** = 30 and **northOrientation** = 0, and the other with **latitude** = 60 and **northOrientation** = 90.

If `factorial` is `TRUE`, then a factorial experiment is created, in which a simulation is defined for each possible combination of supplied values.

```
hip <- define_hisafe(path      = "./simulations",
                     profiles  = "all",
                     template  = "agroforestry",
                     exp.name  = "lat-orient",
                     factorial = TRUE,
                     latitude         = c(30, 60),
                     northOrientation = c(0,  90))
hip$exp.plan
```

```
## # A tibble: 4 x 3
##   SimulationName latitude northOrientation
##   <chr>             <dbl>            <dbl>
## 1 Sim_1                30                0
## 2 Sim_2                60                0
## 3 Sim_3                30               90
## 4 Sim_4                60               90
```

In this case, you can see that there are four simulations in the resulting `hip`, one for each possible combination of the supplied values of **latitude** and **northOrientation**.

Let's now go back to the default template simulation and try to change the crop rotation. This is a more complex modification than just providing a simple number. First of all, to change the crop rotation, there are two input parameters that must be modified: **mainCropSpecies** specifying which .PLT file to use and **mainCropItk** specifying which .TEC file to use. So, let's keep our single simulation but change the crop from just durum wheat to a rotation of durum wheat and rape.

```
hip <- define_hisafe(path      = "./simulations",
                     profiles  = "all",
                     template  = "agroforestry",
                     mainCropSpecies = c("durum-wheat.plt", "rape.plt"),
                     mainCropItk     = c("durum-wheat.tec", "rape.tec"))
hip$exp.plan
```

```
## # A tibble: 2 x 3
##   SimulationName mainCropSpecies mainCropItk
##   <chr>          <chr>           <chr>
## 1 Sim_1          durum-wheat.plt durum-wheat.tec
## 2 Sim_2          rape.plt        rape.tec
```

Oops! We accidentally created **two** simluations, rather than modifying the crop rotation within our single simulation. What happened here? Let's check the documentation: `?define_hisafe()`. Ah, so `define_hisafe()` just did the same here as for when we passed two values each for **latitude** and **northOrientation** above. We need `define_hisafe()` to consider `c("durum-wheat.plt", "rape.plt")` as a single "unit". To do this, we must pass the values as a list.

```
hip <- define_hisafe(path      = "./simulations",
                     profiles  = "all",
                     template  = "agroforestry",
                     mainCropSpecies = list(c("durum-wheat.plt", "rape.plt")),
                     mainCropItk     = list(c("durum-wheat.tec", "rape.tec")))
hip$exp.plan
```

```
## # A tibble: 1 x 3
##   SimulationName mainCropSpecies mainCropItk
##   <chr>          <list>          <list>
## 1 Sim_1          <chr [2]>       <chr [2]>
```

Great, now we have our single simulation back, and we can look more closely to see if the crop rotation was captured correctly:

```
hip$exp.plan$mainCropSpecies
```

```
## [[1]]
## [1] "durum-wheat.plt" "rape.plt"
```

There are few, but important, Hi-sAFe input parameters that are passed as a multiple values. For these parameters, we must group inputs within a list to ensure the multiple values remain together. In this example, if we wanted to create two simluations, one with just wheat, and one with the wheat-rape rotation, we would do this:

```
hip <- define_hisafe(path      = "./simulations",
                     profiles  = "all",
                     template  = "agroforestry",
                     mainCropSpecies = list("durum-wheat.plt", c("durum-wheat.plt", "rape.plt")),
                     mainCropItk     = list("durum-wheat.tec", c("durum-wheat.tec", "rape.tec")))
hip$exp.plan$mainCropSpecies
```

```
## [[1]]
## [1] "durum-wheat.plt"
##
## [[2]]
## [1] "durum-wheat.plt" "rape.plt"
```

So far we have only tried to edit Hi-sAFe parameters in the .SIM and .PLD files. `define_hisafe()` can also be used to edit parameters in the .TREE and .PLT files. You must be careful, though, if your simluations have more than one tree species or more than one crop species, as any .TREE or .PLT parameters passed to `define_hisafe()` will be applied to **all** .TREE and .PLT files. Let's define a simple simulation where we edit `lueMax` in the .TREE file:

```
hip <- define_hisafe(path      = "./simulations",
                     profiles  = c("plot", "trees", "climate", "monthCells", "cells", "voxels"),
                     template  = "agroforestry",
                     exp.name  = "lue",
                     lueMax    = c(0.6, 0.7))
```

We'll use this `hip` object for moving on to the next step.

You can also use `define_hisafe_file()` to directly read an `exp.plan` object from a *csv* file. For more information on this approach, see `?define_hisafe_file`.

2. BUILD simulation folders/files

The `build_hisafe()` function takes a `hip` object as an input and builds the actual folders/files on your computer that Hi-sAFe will read to run. These directories will be created in the location specififed in the `path` portion of the `hip` object.

```
build_hisafe(hip = hip)
```

In addition to building the folder/file structure, `build_hisafe()` also creates a map of each scene within the respective simulation folders. It is highly recommended to check these maps prior to running your simulations to ensure that the scene was built as intended. To create scene maps of simulations within a `hip` object prior to building, you can use `plot_hisafe_scene()` directly.

3. RUN simulations

Once the Hi-sAFe folders/files are created on your comupter, you are ready to run Hi-sAFe! Simulations are run using `run_hisafe()`. To run our previous experiment on **lueMax** we will use `run_hisafe()` by providing:

- The `hip` object.
- A character vector of the **SimulationName** of which simulations to run using `simu.names`, or "all" to run all simulations in the experiment.
- A logical `parallel` indicating whether or not to run multiple simulations in parallel on your computer. The default is `FALSE`, but you are highly encouraged to use parallel processing if your computer has the capacity! If `parallel` is `TRUE`, one fewer than the total number of available cores will be used. Alternatively, the number of cores to use can be specified directly via `num.cores`.
- The path to the Capsis folder on your computer via `capsis.path`. This is where hisafer can find Capsis and the Hi-sAFe model.

```
run_hisafe(hip         = hip,
           simu.names  = "all",
           parallel    = TRUE,
           capsis.path = "/Applications/Capsis/")
```

`run_hisafe()`can also be used without a `hip` object to run Hi-sAFe simulations that were not created using hisafer or when the hip is not available in the workspace. To do this, you must supply the `path` to the directory where the simulation folders are located. Just make sure the simulation folder/file structures are correct!

```
run_hisafe(path        = "./simulations/lue"
           simu.names  = c("Sim_1", "Sim_2"),
           parallel    = TRUE,
           capsis.path = "/Applications/Capsis/")
```

4. READ output data

Hi-sAFe can generate quite a lot of output data. All of this data is created within a set of text files, which can be difficult to navigate without hisafer. During this step, hisafer reads all of this output data into R for easy manipulation and analysis. To read Hi-sAFe output data use `read_hisafe()`. The only required input is a `hip` object or, alternatively, a `path` to the directory containing the simulation folders can by manually provided. You can also specify a subset of simulations to read via `simu.names`. Finally, you can specify which Hi-sAFe output profiles to read in using `profiles`. While the default is to read all profiles, you may not want to read, for example, the large "cells" or "voxels" profiles if you simply want to analyze daily plot data.

The resulting object created by `read_hisafe()` is an obeject of class "hop" (for "Hi-sAFe OutPut"). This is a list of 12 data frames (tibbles):

- `trees`
- `plot`
- `climate`
- `cells`
- `monthCells`
- `annualCells`
- `voxels`
- `plot.info` - general plot data for each simulation
- `tree.info` - general tree species data for each simulation
- `exp.plan` - the `exp.plan` of the hip object that generated the simulation
- `metadata` - the metadata of each simulations (path to simulation folders, model versions, run date , run duration)
- `path` - the path to the experiment/simulation folders

If any Hi-sAFe output profiles specified by `profiles` were not created when Hi-sAFe was run, a warning will notify you. For any profiles not created by Hi-sAFe or not selected for reading, the associated list components will be empty tibbles.

WARNING: Depending on the number of simulations, length of simulations, and which output profiles are specified, reading in Hi-sAFe data can take some time!

To read in the example experiment included in hisafer, we will use `read_hisafe_example()` as follows:

```
hop.raw <- hop <- read_hisafe_example()
```

```
## Warning: hip not provided, and no experiment summary to read. This
## experiment may not have been created with hisafer.
```

```
##
##
## Reading: monocrop
## Profiles: plot, plotDetail, trees, treesDetail, cells, cellsDetail, voxelsMonth, climate, monthCells
##    -- reading:  plot
##    -- reading:  plotDetail
##    -- reading:  trees
```

```
## Warning: monocrop_trees.txt exists but contains no data
```

```
##
##    -- reading:  treesDetail
```

```
## Warning: monocrop_treesDetail.txt exists but contains no data
```

```
##
##    -- reading:  cells
##    -- reading:  cellsDetail
##    -- reading:  voxelsMonth
##    -- reading:  climate
##    -- reading:  monthCells
##    -- reading:  annualCells
##
## Reading: agroforestry
## Profiles: plot, plotDetail, trees, treesDetail, cells, cellsDetail, voxelsMonth, climate, monthCells
##    -- reading:  plot
##    -- reading:  plotDetail
##    -- reading:  trees
```

```
##    -- reading:  treesDetail
##    -- reading:  cells
##    -- reading:  cellsDetail
##    -- reading:  voxelsMonth
##    -- reading:  climate
##    -- reading:  monthCells
##    -- reading:  annualCells
##
## Reading: forestry
## Profiles: plot, plotDetail, trees, treesDetail, cells, cellsDetail, voxelsMonth, climate, monthCells
##    -- reading:  plot
##    -- reading:  plotDetail
##    -- reading:  trees
##    -- reading:  treesDetail
##    -- reading:  cells
##    -- reading:  cellsDetail
##    -- reading:  voxelsMonth
##    -- reading:  climate
##    -- reading:  monthCells
##    -- reading:  annualCells
```

Once you have the `hop` object, you can easily access any of the data within using the `$` operator. For example, to access the "metadata" data of the experiment:

```
hop$metadata
```

```
## # A tibble: 3 x 7
##   SimulationName path  hisafe.version stics.version capsis.version
##   <chr>          <chr> <chr>          <chr>         <chr>
## 1 monocrop       /Lib~ 3.5            8.5           4.2.4
## 2 agroforestry   /Lib~ 3.5            8.5           4.2.4
## 3 forestry       /Lib~ 3.5            8.5           4.2.4
## # ... with 2 more variables: simulation.start <dttm>,
## #   simulation.seconds <dbl>
```

To learn about the output variables contained in the `hop` object, use `hop_params()`. Analagous to `hip_params()` during the Define phase, `hop_params()` displays the definitions and units of the output variables.

Sometimes, you will realize that the values of **SimulationName** that you provided during the define phase are not quite what you wanted. To rename each **SimulationName** in your `hop` object, you can use `hop_rename()`. Lets rename our simulations to better describe what was actually manipulated:

```
hop <- hop_rename(hop       = hop,
                  old.names = c("monocrop", "agroforestry", "forestry"),
                  new.names = c("MC", "AF", "PF"))
```

You may also realize that you want to filter your hop to specific SimulationNames, tree ids, or dates. This can be done using `hop_filter()`.

```
hop <- hop_filter(hop        = hop,
                  simu.names = "all",
                  tree.ids   = "all",
                  date.min   = "1995-01-01",
                  date.max   = "2004-01-01")
```

Similarly, you may realize that you want to merge several `hop` objects together for collective analysis. This can be done using `hop_merge()`.

As you create your own analysis scripts to explore your `hop` object, there are several helper functions that allow you check: the validity of `hip` and `hop` objects, the presence of a specific output profile within a `hop` object, and the presence of specific output variables within a `hop` object. These are: `is_hip()`, `is_hop()`, `check_profile()`, and `check_variable()`.

5. Data DIAGNOSTICS

Prior to performing any analyses using your `hop` object, it is highly recommended to check some basic diagnostics on your simluations to ensure that the simulations ran as expected. To do so, hisafer provides some basic diagnostic functions that allow you to compare your various simulations side by side:

- `diag_hisafe_ts()` - plots a timeseries plot for each variable in the "annualTrees", "annualPlot", "trees", "plot", and "climate" profiles of a `hop` object.
- `diag_hisafe_cells()` - plots a full range of tile plots for each variable in the "cells" profile of a `hop` object.
- `diag_hisafe_monthcells()` - plots a full range of tile plots for each variable in the "monthCells" profile of a `hop` object.
- `diag_hisafe_annualcells()` - plots a full range of tile plots for each variable in the "annualCells" profile of a `hop` object.
- `diag_hisafe_voxels()` - plots a timeseries for each variable in the "voxels" profile of a `hop` object.

You can also easily apply the various diagnostics functions using the shortcut function `diag_hisafe()`.

6. ANALYZE data

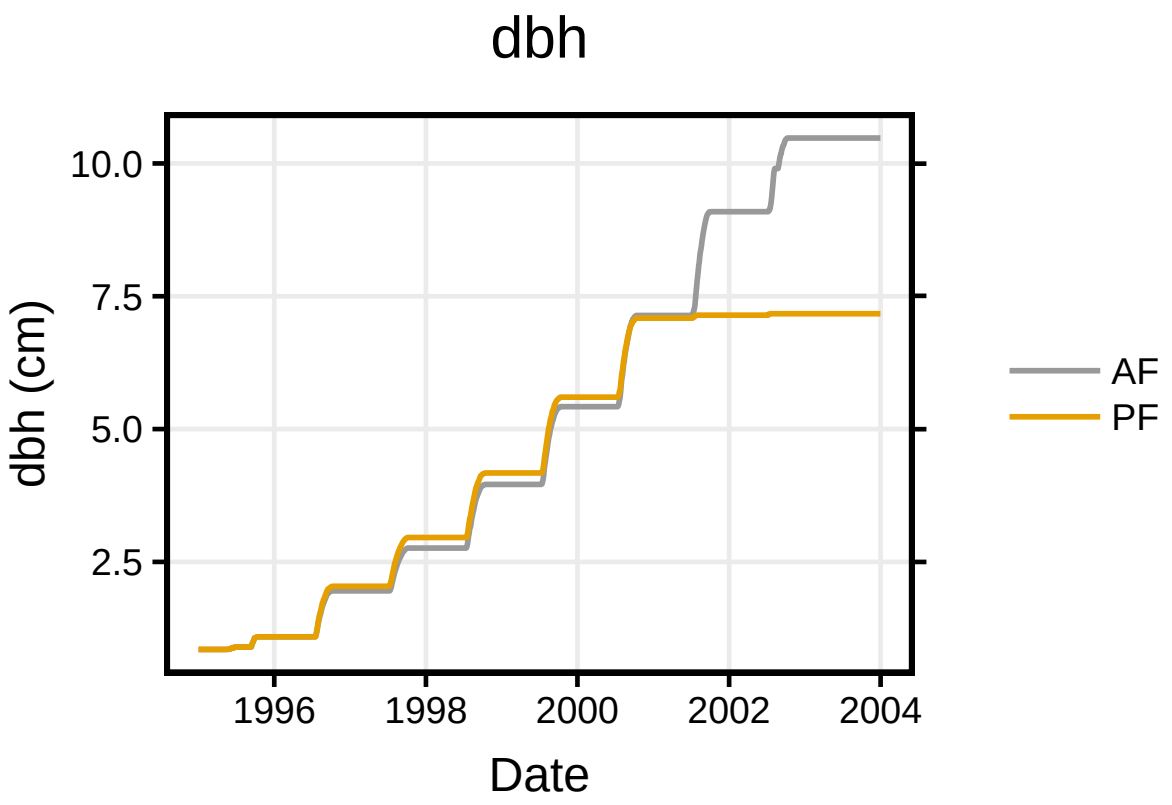
There are many different types of analyses that can be performed with Hi-sAFe output data. hisafer contains a large suite of data analysis functions. By default, most hisafer analysis function return a `ggplot` object. Each function, however, has an arguement `plot`, whch can be set to `FALSE` to instead return the data (a tibble) that would create the plot.

6.1 Basic plots

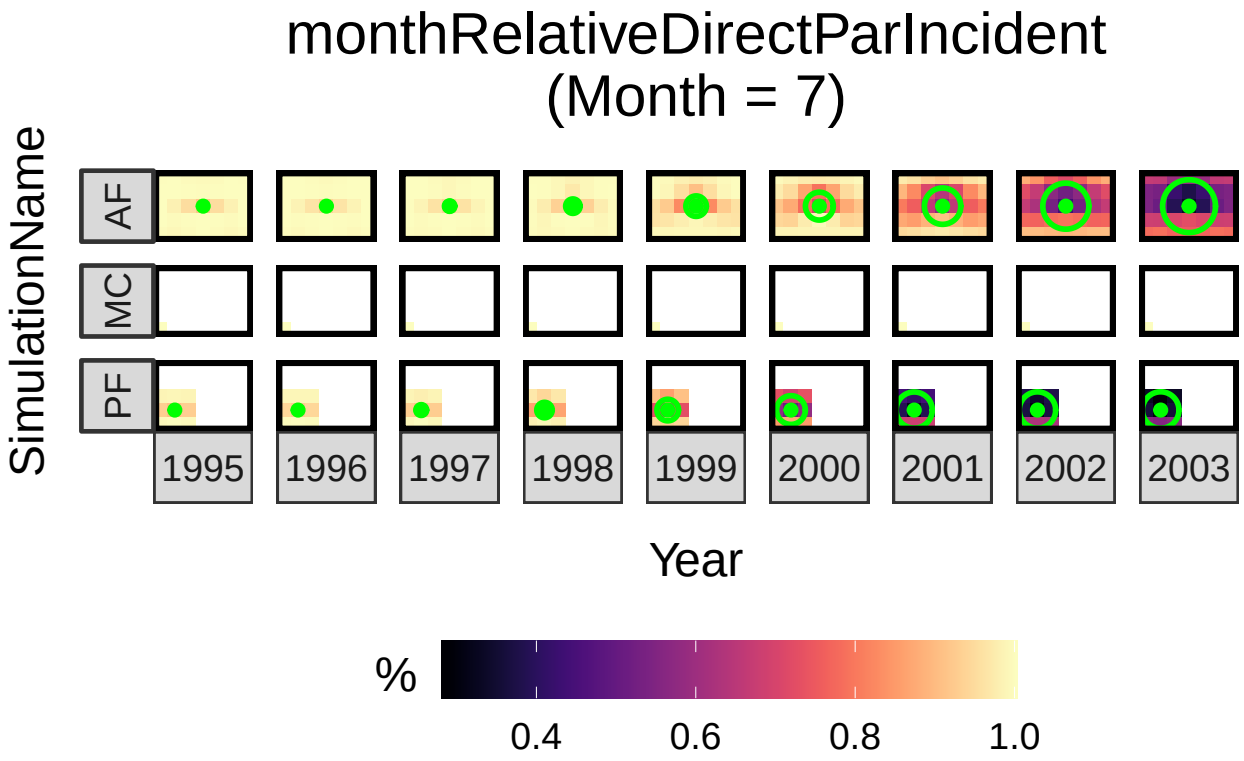
Here are some basic plotting functions that can plot individual variables:

- `plot_hisafe_ts()` - plots "annualTrees", "annualPlot", "trees", "plot", or "climate" timeseries data
- `plot_hisafe_monthcells()` - plots tile plots of "monthCells" data
- `plot_hisafe_cells()` - plots tile plots of "cells" data
- `plot_hisafe_voxels()` - plots timeseries plots of "voxels" data

```
plot_hisafe_ts(hop        = hop,
               variable   = "dbh",
               profile    = "trees",
               facet.year = FALSE)
```



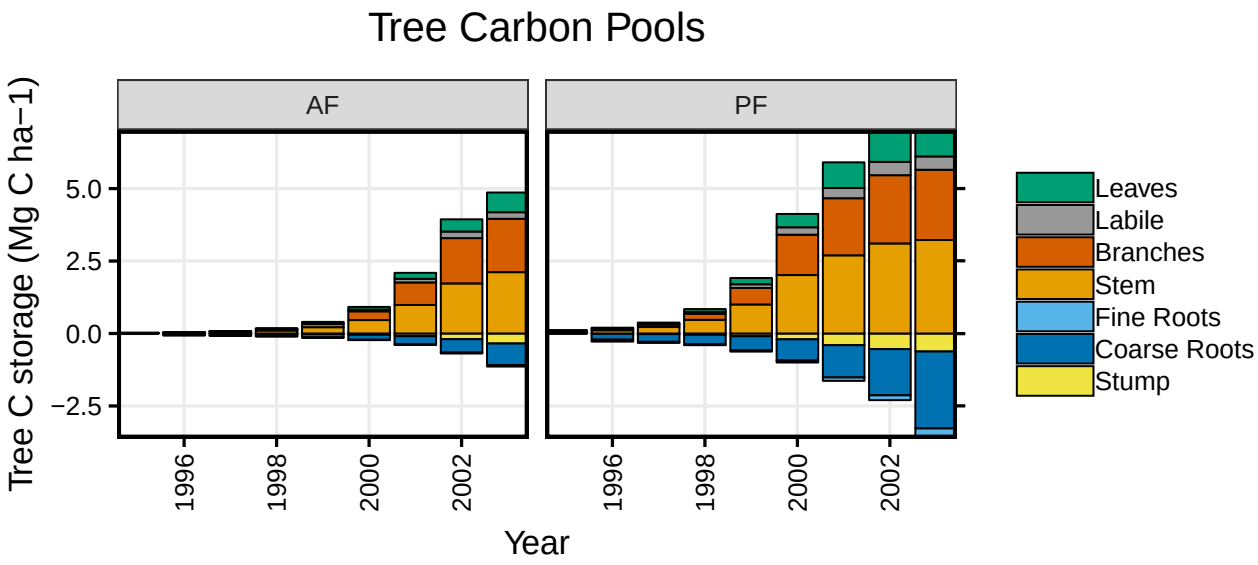

```
plot_hisafe_monthcells(hop        = hop,
                       variable   = "monthRelativeDirectParIncident",
                       colfacet   = "Year",
                       rowfacet   = "SimulationName",
                       years      = 1995:2003,
                       months     = 7)
```



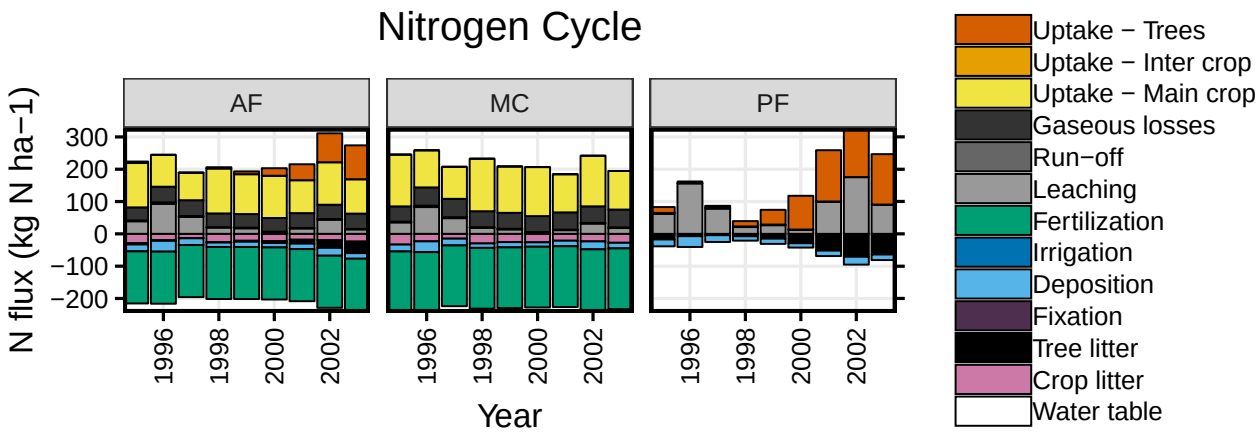
6.2 Cycle plots

You can also compare the major Hi-sAFe "cycles" among your simulations by using `plot_hisafe_cycle_annual()` and `plot_hisafe_cycle_daily()`. Supported "cycles" include carbon, water, nitrogen, and light.

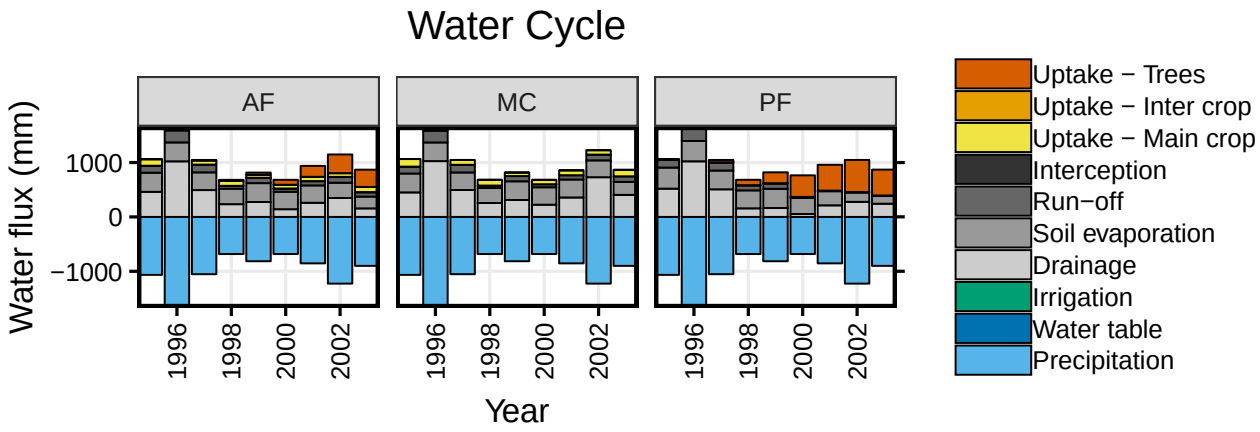
```
plot_hisafe_cycle_annual(hop = hop, cycle = "carbon")
```



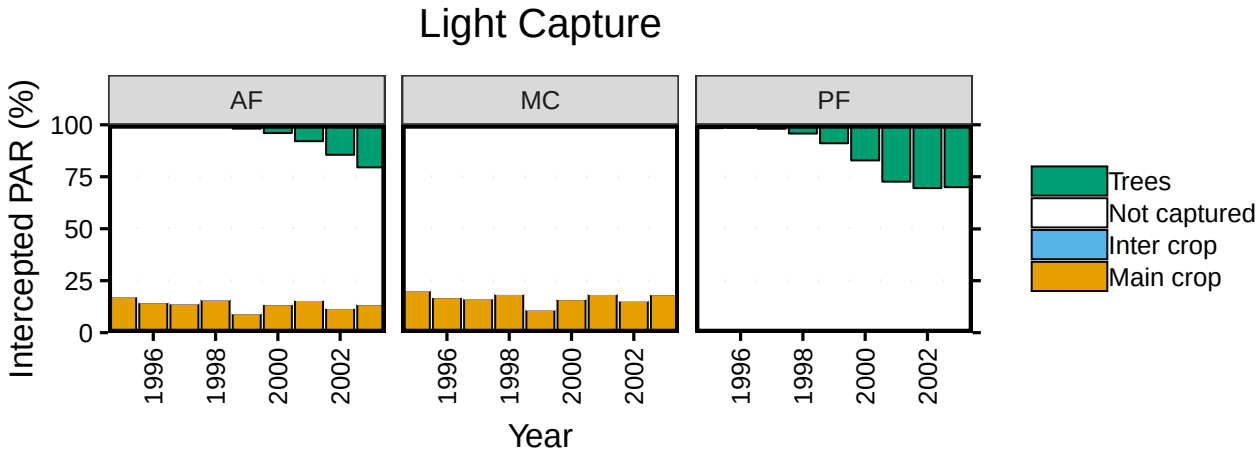
```
plot_hisafe_cycle_annual(hop = hop, cycle = "nitrogen")
```



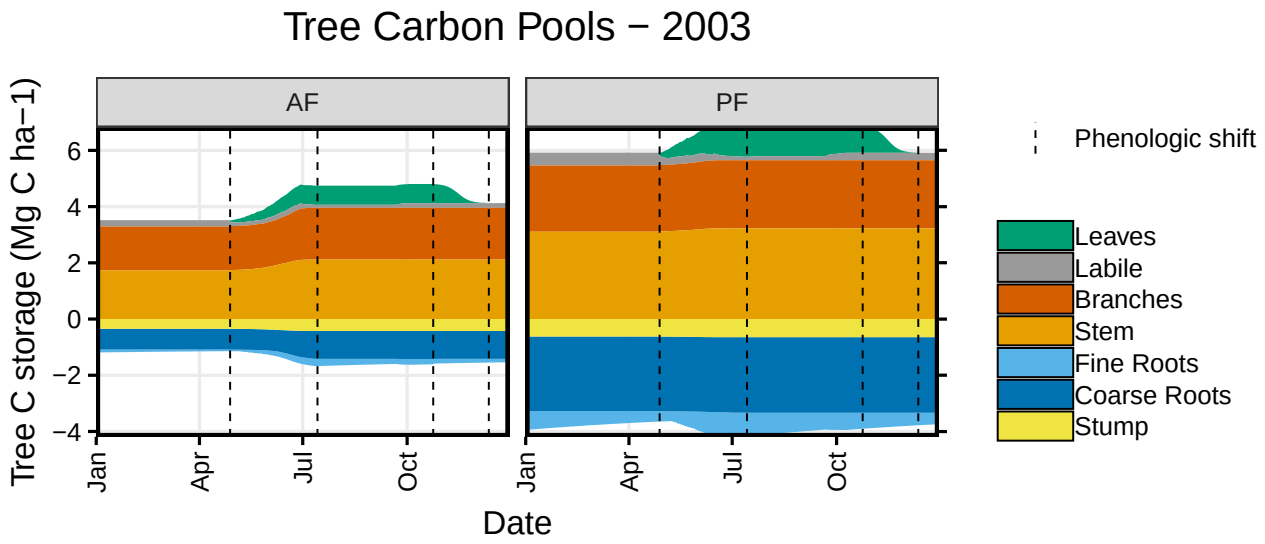

```
plot_hisafe_cycle_annual(hop = hop, cycle = "water")
```



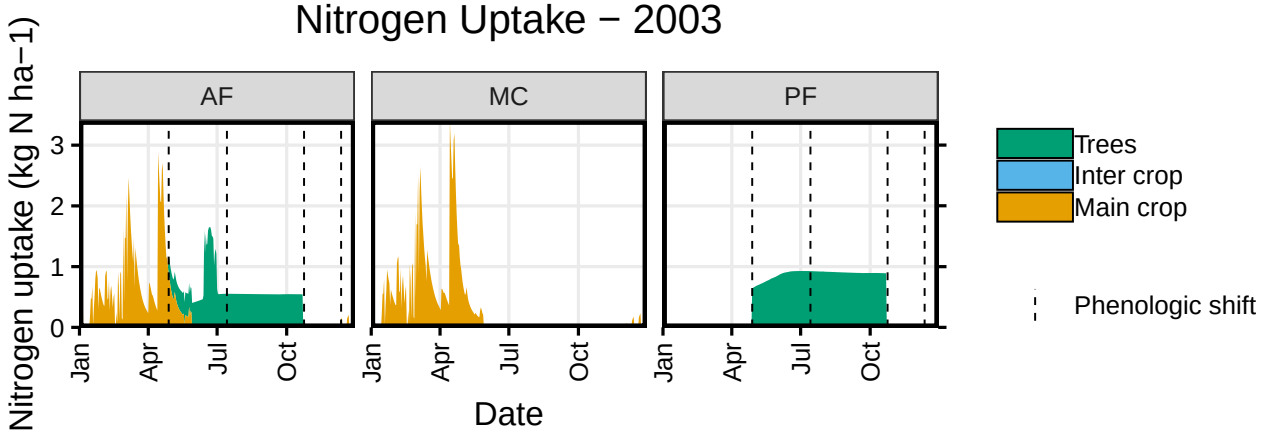

```
plot_hisafe_cycle_annual(hop = hop, cycle = "light")
```



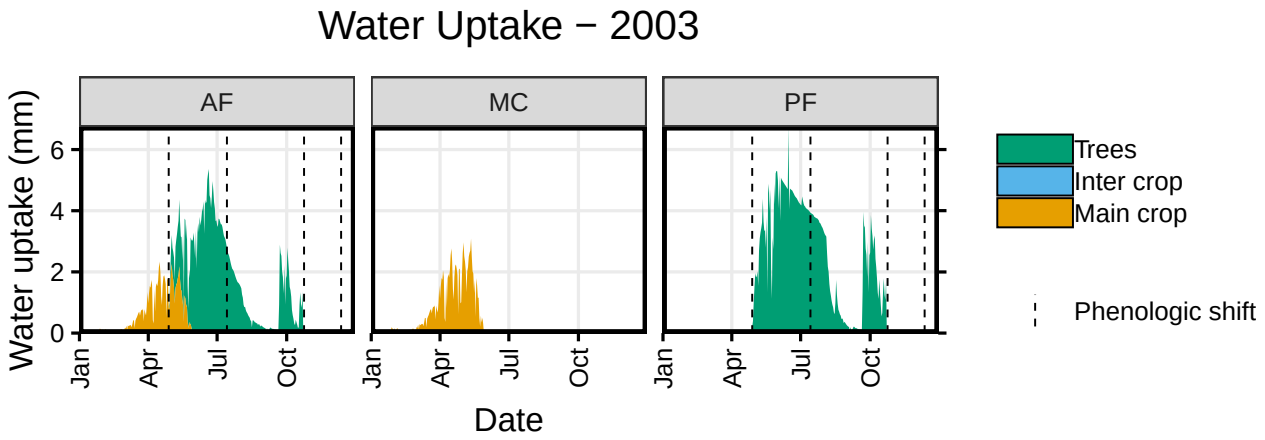

```
plot_hisafe_cycle_daily(hop = hop, cycle = "carbon", years = 2003)
```



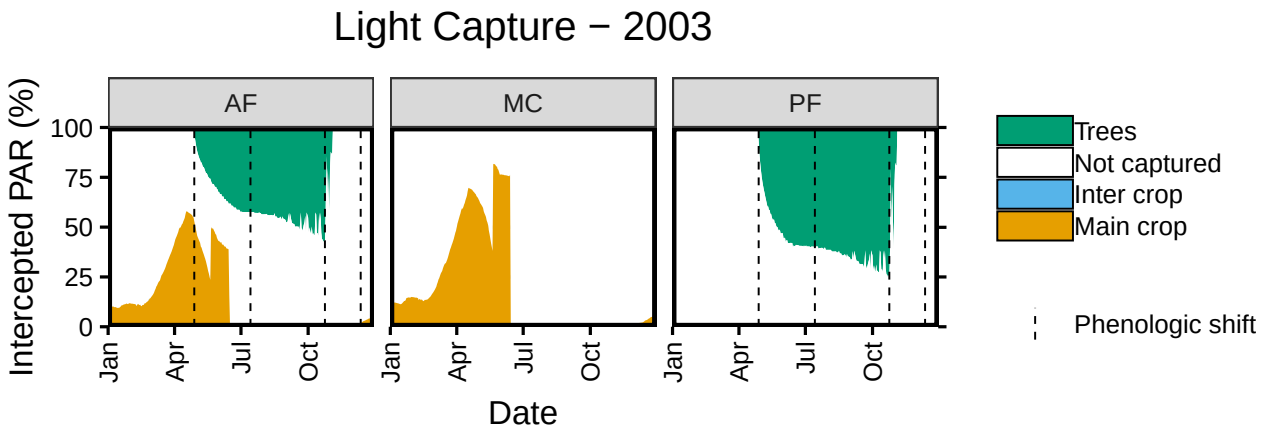

```
plot_hisafe_cycle_daily(hop = hop, cycle = "nitrogen", years = 2003)
```



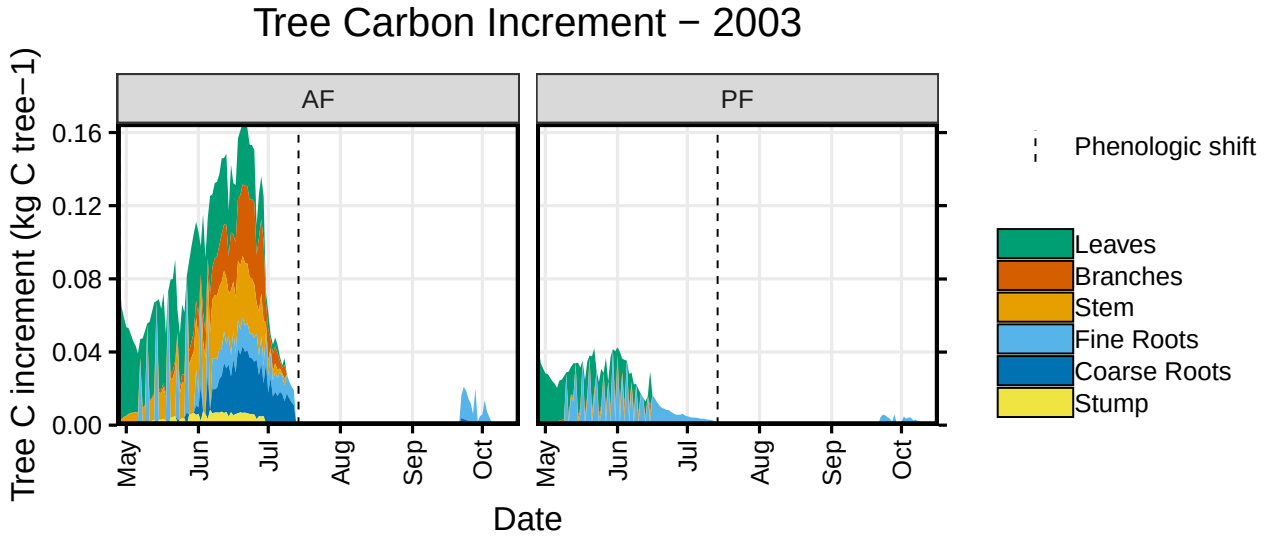

```
plot_hisafe_cycle_daily(hop = hop, cycle = "water", years = 2003)
```



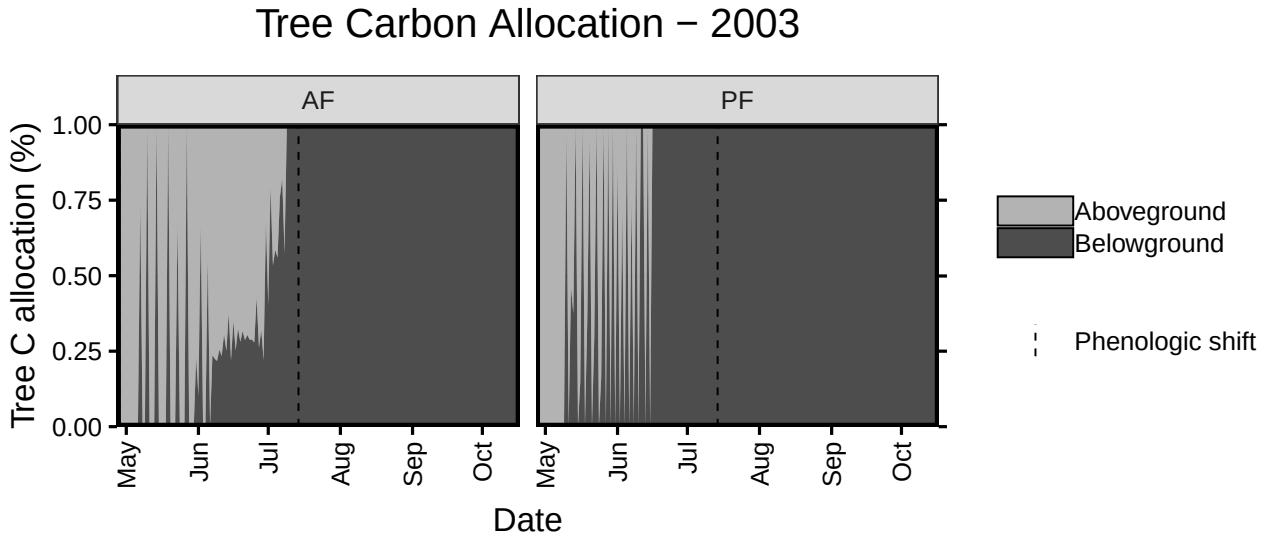

```
plot_hisafe_cycle_daily(hop = hop, cycle = "light", years = 2003)
```




```
plot_hisafe_cycle_daily(hop = hop, cycle = "carbon-increment", years = 2003)
```

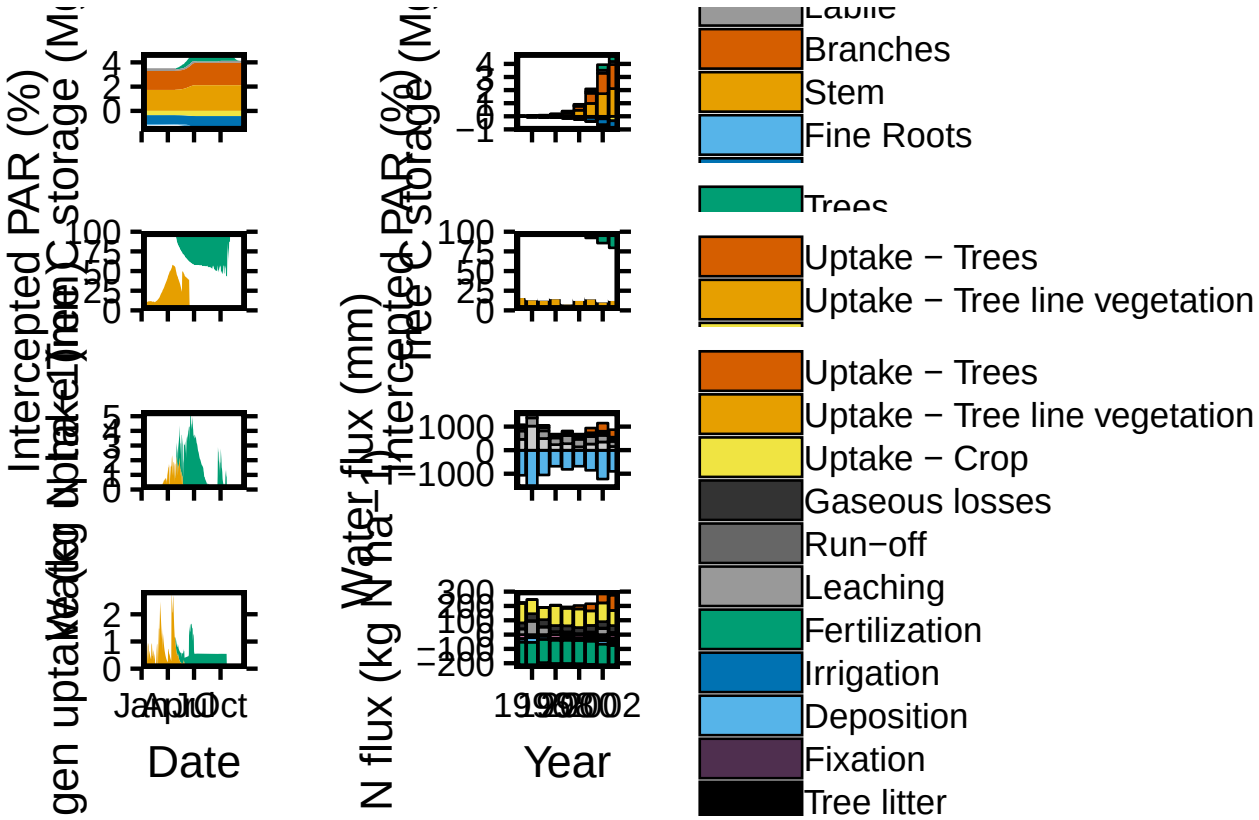



```
plot_hisafe_cycle_daily(hop = hop, cycle = "carbon-allocation", years = 2003)
```



You can also create a "dashboard" of the cycles of simulation via `cycle_summary()`:

```
summary.out <- cycle_summary(hop          = hop,
                             daily.year   = 2003,
                             cycles       = c("carbon", "light", "water", "nitrogen"),
                             simu.name    = "AF",
                             crop.names   = c("Crop", "Tree line vegetation"))
summary.out
```

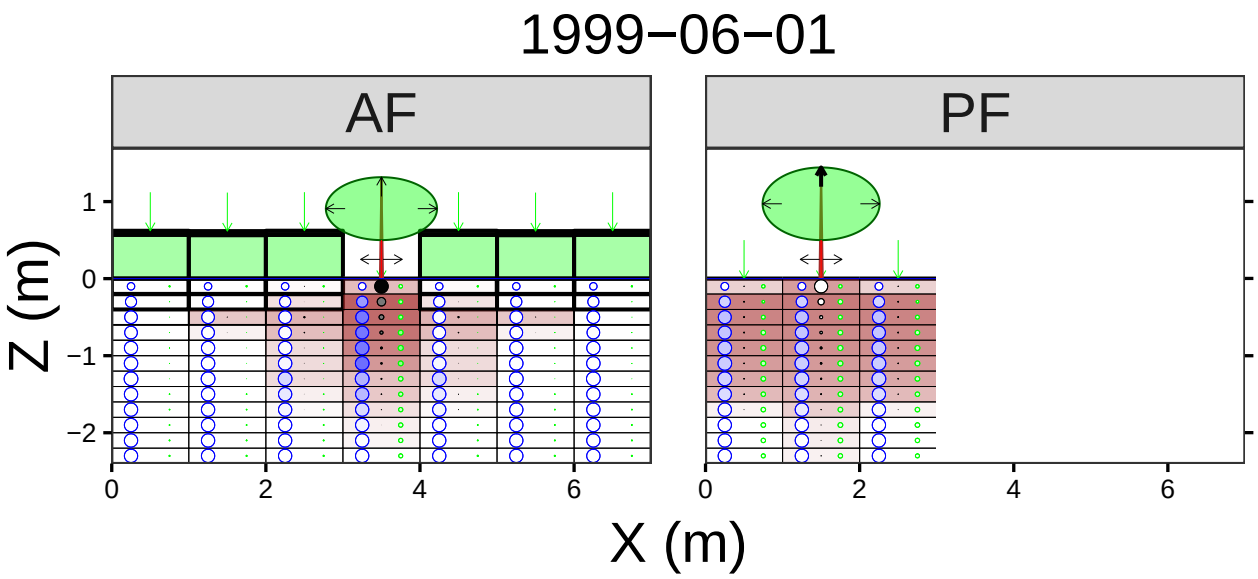


You can also easily apply the various analyze functions and write the output plots to the disk using the shortcut function `analyze_hisafe()`.

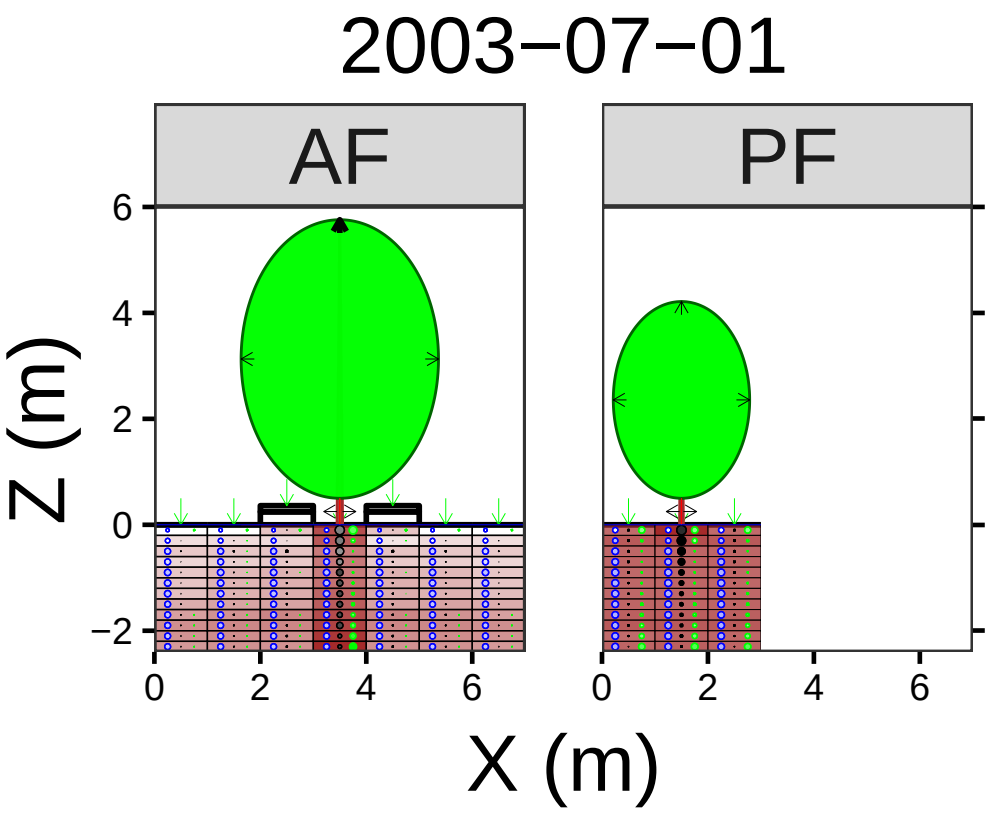
6.3 Visualizations

The most important visualization tool in hisafer is `hisafe_slice()` which "slices" through the 3D Hi-sAFe scene to create a 2D projection.

```
hisafe_slice(hop        = hop,
             simu.names = c("AF", "PF"),
             date       = "1999-06-01")
```



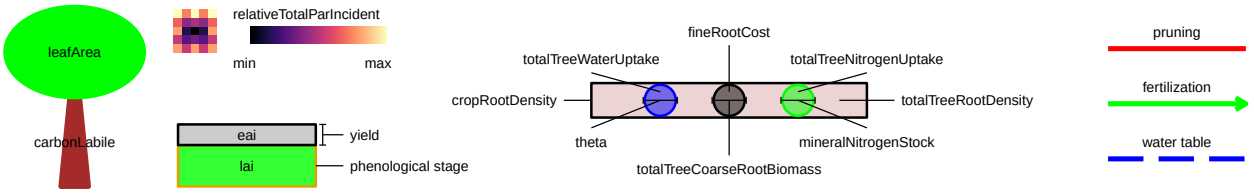

```
hisafe_slice(hop         = hop,
             simu.names = c("AF", "PF"),
             date        = "2003-07-01")
```



You can use `hisafe_slice()` even if you do not have the voxels profile within your `hop` object by setting `voxels = FALSE`. You can also customize what you want each aspect of the plot to represent via the `vars` argument. Furthermore, you can switch the direction in which the scene is "sliced" by setting `plot.x = "y"`. This gives you the opportunity to view the 3D scene from both 2D directions.

To see a legend of what everything represents:

```
visual_legend(hop = hop)
```



These plots from `hisafe_slice()` can be combined with plots from `plot_hisafe_cells()` using `hisafe_snapshot()` to export daily "snapshot" images of the Hi-sAFe simulations. These images can then be combined into a video/gif to visualize the simulations in real time!

To visualize tree root topology, you can use `hiasfe_root3D()`. You must have the `rgl` library installed and functional to do so!

```
hisafe_root3D(hop      = hop,
              simu.name = "AF",
              date      = "2003-06-01")
```

6.4 FACE experiments

One very special case of analyzing Hi-sAFe simulations occurs when you want to compare one or more agroforestry simulations with a forestry control simulation and a monocrop control simulation. hisafer refers to this type of experiment as a "FACE" (Forestry-Agroforestry-Crop Experiment). The first step in this special analysis is to create a `face` object, which is a modified `hop` objects that merges three individual `hop` objects of the respective systems:

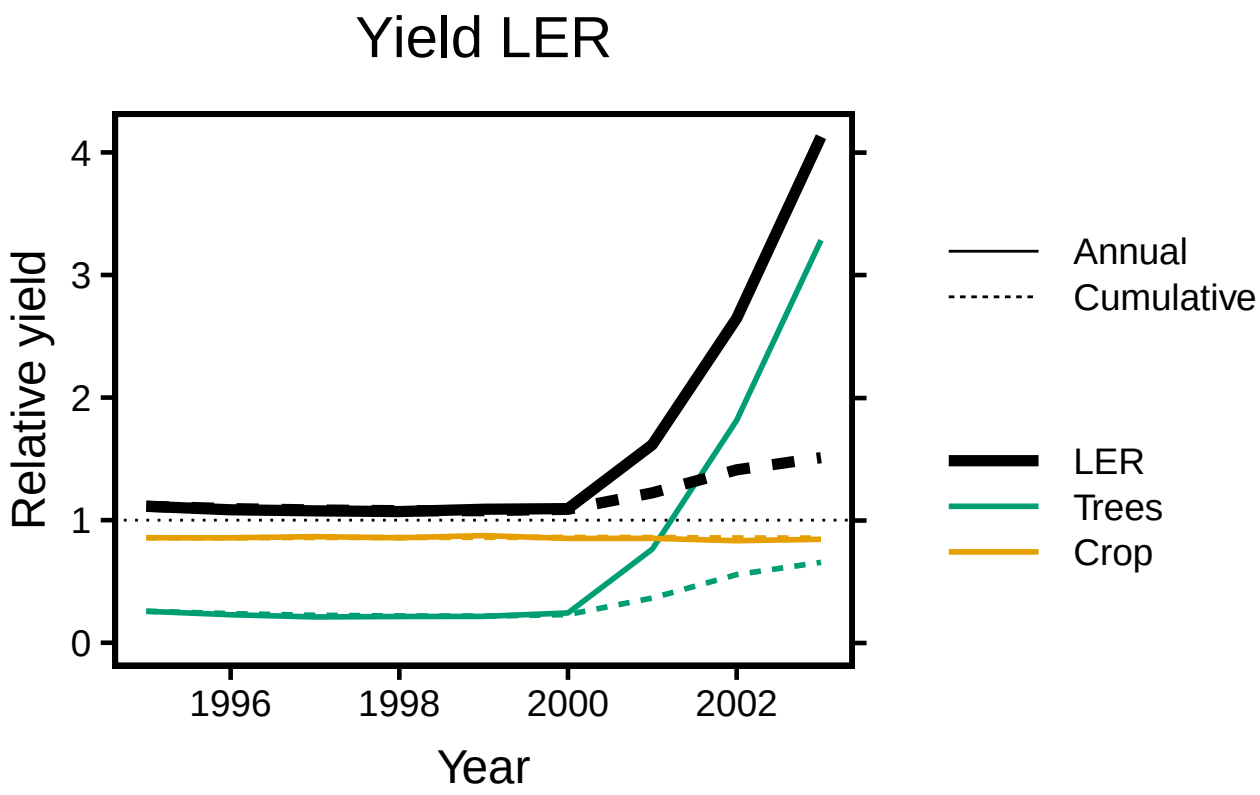
- agroforestry
- forestry
- monocrop

To create a `face` object, use `create_face()`and supply the three required `hop` objects along with a path to a diretory where you want all FACE analyses to be saved. The agroforestry `hop` can contain any number of agroforestry simulations, but the forestry `hop` and monocrop `hop` can only contain a single simulation each. Since we already have those three systems within one hop, we will have to use hop_filter().

```
face <- create_face(agroforestry = hop_filter(hop, "AF"),
                    forestry     = hop_filter(hop, "PF"),
                    monocrop     = hop_filter(hop, "MC"),
                    face.path    = ".")
```

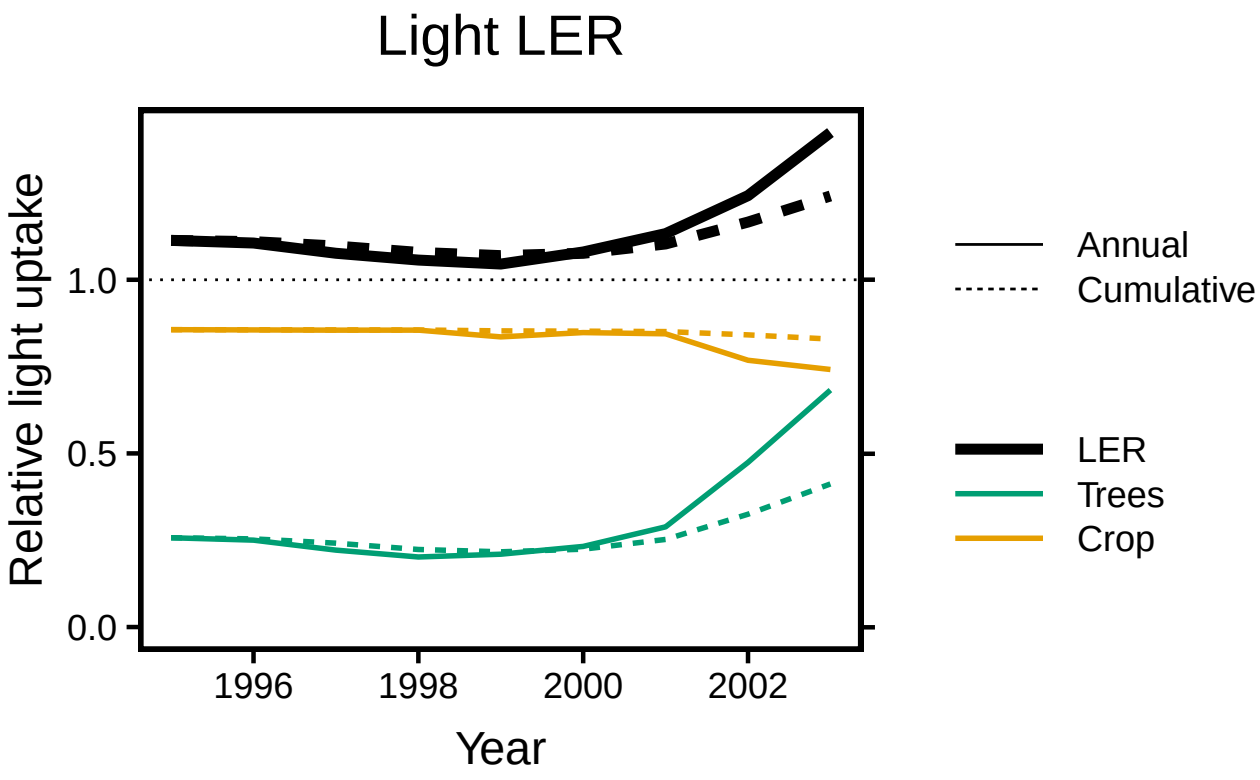
Once you have a `face` object, the most common analysis is to calculate the various LER metrics of the agroforestry systems. You can do this using `LER()`.

```
LER(face = face)
```



Beyond LER in terms of yield components, you can also calculate LER in terms of light, water, and nitrogen balances.

```
LER(face = face, cycle = "light")
```



Furthermore, just as for normal `hop` objects, you can also use `plot_hisafe_cycle_annual()` and `plot_hisafe_cycle_daily()` to compare the simulations in a `face` object.

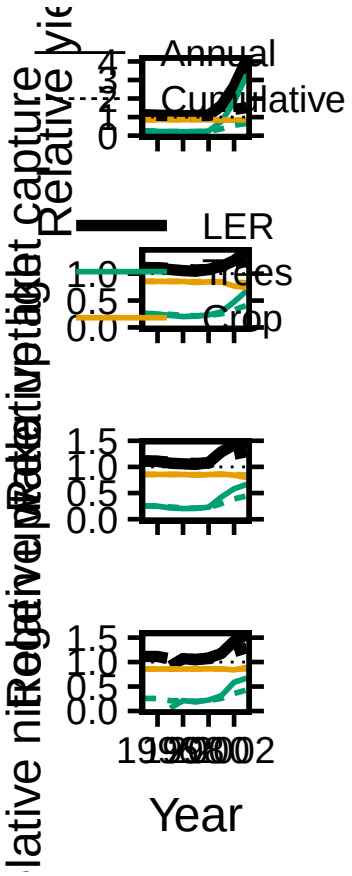
```
plot_hisafe_cycle_annual(hop = face, cycle = "carbon")
```

```
plot_hisafe_cycle_annual(hop = face, cycle = "light")
```

For a complete, publication-ready summary figure of LER, you can use `LER_summary()`.

```
ler.out <- LER_summary(face = face)

ler.out
```



6.5 Budgets

Ensuring that the water and nitrogen budgets are closed isn't necessary every time you run a simulation, but it is very important if you are pushing the limits of Hi-sAFe with new functionality or complex management interventions. To check the basic Hi-sAFe water and nitrogen budgets, use `hiasfe_budget()`. To compare the Hi-sAFe-calculated budget with the STICS-calculated budget, use `stics_budget_comp()`. projection.

Notes

hisafer utilizes the tidyverse approach to R programming and data manipulation. While most consequences of this approach are behind the scenes, one obvious example to users is that outputs from most hisafer functions are tibbles rather than simple data frames. This has little practical impact on your use of hisafer, but should improve your overall experience.