shinemas2R

An R package to visualize outputs from the data base Seed History and Network Management System (SHiNeMaS)

version 0.11

October 2, 2016

Pierre Rivière[1,2] Yannick de Oliveira[3]

[1] Réseau Semences Paysannes, 3 avenue de la gare, F-47190 Aiguillon, France
[2] INRA, UMR 0320, Génétique Quantitative et Evolution, DEAP team, Ferme du Moulon F-91190 Gif sur Yvette, France
[3] INRA, UMR 0320, Génétique Quantitative et Evolution, ABI team, Ferme du Moulon F-91190 Gif sur Yvette, France
Contact: pierre@semencespaysannes.org
Contributions: P. Rivière worte the R codes and the vignette, Y. de Oliveira wrote the SQL queries.

Le Réseau Semences Paysannes (the French Farmers' Seeds Network (RSP)), created in 2003, brings together a great diversity of collectives and people who preserve farmers' seeds in fields, orchards, vineyards and gardens. They are involved in supporting the consolidation of local initiatives to maintain and renew cultivated biodiversity through Community Seeds Systems. Over 80 organizations have come together to promote and develop farmers' seeds, and to protect farmers' rights over their seeds.
www.semencespaysannes.org (in french).

The Diversity, Evolution and Adaptation of Populations (DEAP) team led by Isabelle Goldringer is part of INRA UMR 0320 Quantitative Genetic and Evolution. Its work is based on the analysis of the genetic and evolutionary mechanisms underlying evolution and adaptation of crop populations. DEAP develops strategies for on farm management of crop genetic diversity and for plant breeding (evolutionary and/or participatory) adated to organic and low input agriculture. Assessing the benefits of in-field genetic diversity (variety mixtures, populations) and designing / breeding optimized mixtures adapted to local conditions are also key research objectives.
http://moulon.inra.fr/index.php/en/team/deap
The bioinformatics and informatics facility (ABI, Atelier Bioinformatique et Informatique) provides bioinformatics expertise and IT support. The staff includes 6 experts in system administration, software development or bio-analysis, and develops databases and softwares for proteomics, genetics and genomics. ABI offers hardware resources, scientific programming and consulting for DNA, RNA and protein sequence analysis up to genome-wide scale. ABI works in tight collaboration with the Bioinformatics facilities of University Paris-Sud and INRA, and contributes to the future French Bioinformatics Institute.
http://moulon.inra.fr/index.php/en/tranverse-team/atelier-de-bioinformatique

Contents

Wheat trials on farm within our participatory plant breeding programme, summer 2012, Auvergne, France. CC-BY-NC-SA. Pierre Rivière.

1 Philosophy of `shinemas2R`

1.1 What is SHiNeMaS?

Started in 2009, DEAP and ABI teams from INRA Le Moulon developped a data base to store data in a reliable way during research programs dedicated to the study of dynamic management of crop diversity within experimental stations and farmer networks (Thomas, 2011).

From 2011 to 2013, this data base was migrated to Seed History and Network Management System (SHiNeMaS) and has evolved in a participatory plant breeding program involving INRA Le Moulon and the Réseau Semences Paysannes (RSP) on bread wheat (Rivière, 2014).

From 2014 to today, farmers organisations' facilitators informed on their specific needs for their data management, involving other species than bread wheat such as maize, tomatos or trees. Specific developments were performed to fit the farmers organisations' needs.

SHiNeMaS stores information related to (Figure 1):

- network relations between seed-lots
- data linked to the seed-lots and to the relations between seed-lots

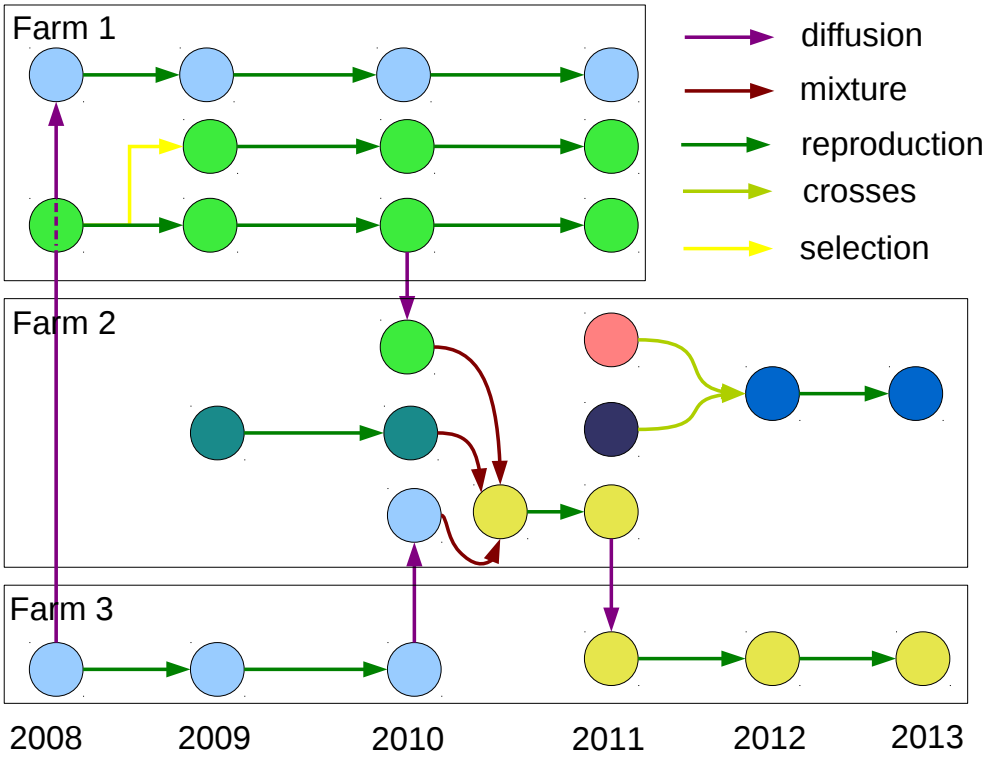


Figure 1: Seed-lots relations in SHiNeMaS. The seed-lots are represented by a circle. The color represents a germplasm (i.e. a variety). The arrows represent the relations between seed-lots (diffusion, mixture, reproduction, crosses and selection). Data are linked to the seed-lots and to the relations between seed-lots.

SHiNeMaS can be downloaded here. More information on the history of SHiNeMaS and its uses with tutorials can be found on the website (De Oliveira et al., 2015).

`shinemas2R` is an `R` package that analyses outputs from SHiNeMaS. It does descriptive analysis, formats data for existing `R` packages to perform statistical analysis as well as compiling results in pdf.

1.2 Function relations in `shinemas2R`

`shinemas2R` is divided into three steps (Figure 2):

1. Get the data from SHiNeMaS with `get.data`. It is possible to use data in the same format than `get.data` outputs. This can be done with `is.get.data.output`. This is useful when the user do not have SHiNeMaS but wish to use the other functions of the package.

   You may encrypt the data with `encrypt.data` or translate it with `translate.data`. There are two types of data:

   - network relations between seed-lots (section 3)
   - data linked to the seed-lots and to the relations between seed-lots (section 4)

2. From the data,

   (a) Get descriptive outputs from the data: plots with `get.ggplot` or tables with `get.table`

   (b) Get statistical outputs by formating the data with `format.data` in order to use an existing `R` package (the following package can be used: `PPBstats`).

3. Get a pdf that compiles the outputs in a pdf document with `get.pdf` (section 6)

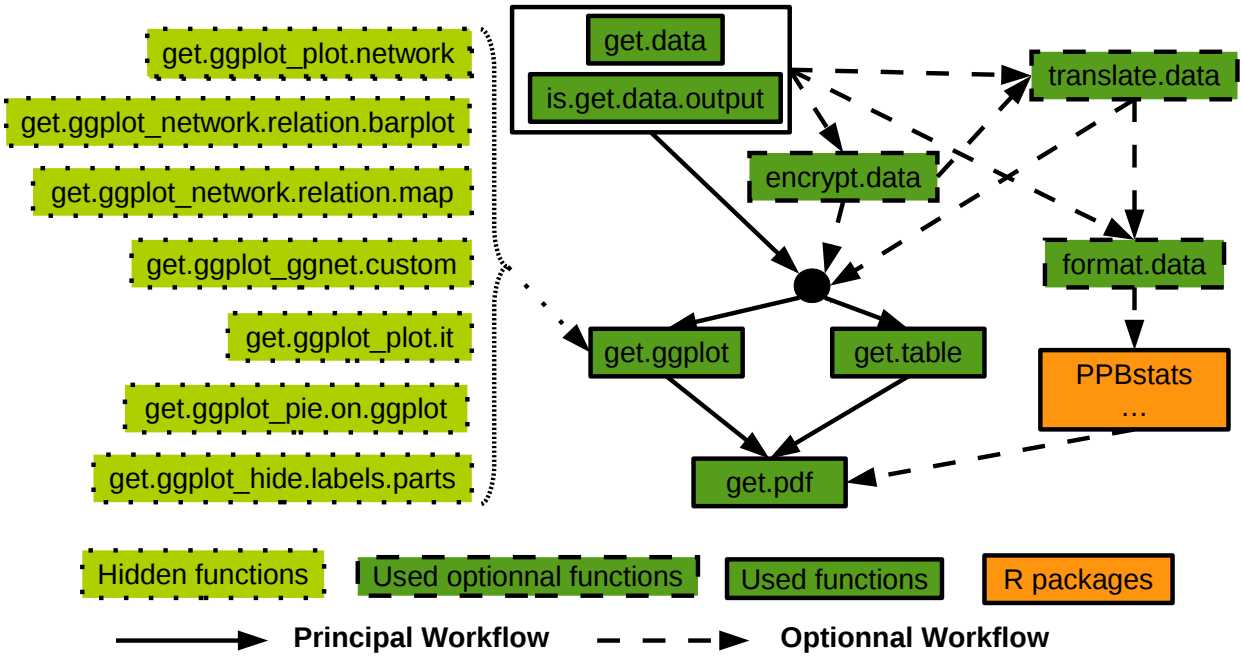


Figure 2: Relations between functions in `shinemas2R`

`get.ggplot` is based on `ggplot2` package. It is therefore easy to customize a plot generated by `get.ggplot` by adding layers. More details can be found on the `ggplot2` documentation website: http://ggplot2.org.

1.3 Let's go!

To continue, load the package:

```
library(shinemas2R)
```

For an easier visualization in the vignette, all warning messages are not represented (NAs introduced, rows deleted, etc.)

1.3.1 Test with SHiNeMaS installed

It may be useful to test the `get.data` function on SHiNeMaS directly. Two options:

- SHiNeMaS is installed on a server you can have access to. You need to ask the admin the following information: `db_user`, `db_host`, `db_name`, `db_password`. Add these information into `info_db`, which is needed in the next steps in `get.data`. For example:

```
info_db = list(
        db_user = "pierre",
        db_host = "127.0.0.1", # localhost
        db_name = "shinemas_dev",
        db_password = "toto"
)
```

- install a demo of SHiNeMaS on your computer. The procedure to do so is explained in appendix A.

As you will use `get.data`, you should set:

```
use.get.data = TRUE
```

1.3.2 Test without SHiNeMaS installed

In this vignette, you can test `shinemas2R` without installing SHiNeMaS. You do not use the `get.data` function:

```
use.get.data = FALSE
```

And you can download the data-sets here : https://www.dropbox.com/sh/u1wrwttbn4mq5l4/AACXqZNtzI6TPUeZ_15eyHz_a?dl=0 and put it in the folder data.
The following code is specific to my computer.

```
system("mkdir data")
system("cp -r /home/pierre/Documents/geek/R-stats/Rdev/R_package_shinemas2R/shinemas2R_data/*RData ./
```

and then `load()` it.

In this vignette, there are no examples with `is.get.data.output`. Refer to appendix D to know the format that must have the data to be used in the other functions of the package (i.e. `get.ggplot` and `get.table` followed by `get.pdf`).

2 Raw information on levels and variables

Several information are stored in SHiNeMaS. To get information on raw information, levels and variable, you must use the argument `query.type` in `get.data`. The following table summarize the different possibilities:

`query.type`	type of information present in SHiNeMaS
"species"	species
"variable"	variable
"person"	person
"year"	year
"project"	project
"seed.lot"	seed-lots
"selection.person"	persons that did intra-varietal mass selection
"reproduction.type"	reproduction type
"germplasm.type"	germplasm type
"germplasm"	germplasm

For example,

```
if(use.get.data){
        person = get.data(
                db_user = info_db$db_user, db_host = info_db$db_host,# db infos
                db_name = info_db$db_name, db_password = info_db$db_password, # db infos
                query.type = "person" # information on person present in SHiNeMaS
        )
} else {
                load("data/person.RData")
        }

person$data
```

```
## [1] "JFB" "JSG" "MLN" "OLR"
## attr(,"shinemas2R.object")
## [1] "person"
```

3 Network relations between seed-lots

3.1 Get the data set

To get the data, use the function `get.data` with `query.type = "network"`.

You can set filters to the query with the following argument :

- `filter.in` to choose nothing except `filter.in`
- `filter.out` to choose everything except `filter.out`

values of filters can be `germplasm`, `germplasm.type`, `year`, `person`, `project`, `seed-lots`, `relation` or `reproduction.type`.

It is important to choose whether to apply the filter to the `father` or to the `son` of a relation. This can be done with the argument `filter.on`. Possibles values are `"father"`, `"son"` or `"father-son"`. It is set by default to `"father-son"`.

It is possible to get the `Mdist` square matrix with the number of reproductions that separate two seed-lots since their last common diffusion (argument `Mdist = TRUE`). Note that a value of 1 means a distance between

- a seed-lot that has been diffused to location A without beeing sown and harvested on location B and
- a seed-lot that has been diffused to location B and beeing sown and harvested on location B

This square matrix can be compared to a differentiation distance. It can be put in relation with genetic F_{st} for example (Nei, 1973)[1].

```
if(use.get.data){
        data_network = get.data(
                db_user = info_db$db_user, db_host = info_db$db_host, # db infos
                db_name = info_db$db_name, db_password = info_db$db_password, # db infos
                query.type = "network", # network query
                filter.on = "father-son", # filter on father AND son
                Mdist = TRUE
                )
} else {
        # 1. Query SHiNeMaS ...
        # Mixtures from replication have been deleted and replaced by reproductions.
        # 2. Create network matrix ...
        # 3. Link information to vertex and edges ...
        # 4. Get network information on seed-lots ...
        # 5. Get Mdist square matrix ...

        load("./data/data_network.RData")
        }
```

The function returns a list with:

- the netwok object

```
n = data_network$data$network
n


##  Network attributes:
##   vertices = 98
##   directed = TRUE
##   hyper = FALSE
##   loops = FALSE
##   multiple = FALSE
```

[1] The `R` package `adegenet` can be used for that.

```
##   bipartite = FALSE
##   total edges= 102
##     missing edges= 0
##     non-missing edges= 102
##
##  Vertex attribute names:
##     germplasm germplasm.type person sex vertex.names year
##
##  Edge attribute names:
##     generation_local generation_total relation
```

Note you can convert this object to an `igraph` object. This may be useful if you like to use the `igraph` package. See http://mbojan.github.io/intergraph/ for more information.

```
n_igraph = intergraph::asIgraph(n)
n_igraph
```

```
## IGRAPH D--- 98 102 --
## + attr: germplasm (v/c), germplasm.type (v/c), na (v/l),
## | person (v/c), sex (v/c), vertex.names (v/c), year (v/c),
## | generation_local (e/c), generation_total (e/c), na (e/l),
## | relation (e/c)
## + edges:
##  [1]  7-> 1  1-> 2  7-> 2  2-> 3  1-> 4  5-> 7  6-> 7 12-> 8  1-> 9
## [10]  9->10  1->11 11->12  6->14 13->14 14->15 15->16 16->17 15->18
## [19] 18->19 25->20 20->21 20->22 22->23 13->25 24->25 20->26 30->27
## [28] 20->28 28->29 22->30 37->31 31->32 31->33 33->34 33->35 24->37
## [37] 36->37 31->38 35->39 31->40 40->41 46->42 42->43 13->45 44->45
## + ... omitted several edges
```

- the network.query dataframe coming from the query

```
head(data_network$data$network.query)
```

```
##   son_species son_project                son son_germplasm
## 1  ble-tendre        DEMO C1#a_JFB_2013_0001            C1
## 2  ble-tendre        DEMO C1#a_JFB_2014_0001            C1
## 3  ble-tendre        DEMO C1#a_JSG_2014_0001            C1
## 4  ble-tendre        DEMO C1#c_JFB_2014_0001            C1
## 5  ble-tendre        DEMO   C1_JFB_2011_0001            C1
## 6  ble-tendre        DEMO   C1_JFB_2011_0001            C1
##   son_person son_year son_germplasm_type son_alt son_long  son_lat
## 1        JFB     2013               <NA>      NA 0.616363 44.20314
## 2        JFB     2014               <NA>      NA 0.616363 44.20314
## 3        JSG     2014               <NA>      NA 3.087025 45.77722
## 4        JFB     2014               <NA>      NA 0.616363 44.20314
## 5        JFB     2011               <NA>      NA 0.616363 44.20314
## 6        JFB     2011               <NA>      NA 0.616363 44.20314
##   son_total_generation_nb son_local_generation_nb
## 1                       6                       5
## 2                       6                       3
## 3                       7                       2
## 4                       1                       2
## 5                       6                       2
## 6                       5                       1
```

```
##   son_generation_confidence son_comments father_species
## 1                         0    blablabla     ble-tendre
## 2                         1    blablabla     ble-tendre
## 3                         0    blablabla     ble-tendre
## 4                         0    blablabla     ble-tendre
## 5                         0    blablabla     ble-tendre
## 6                         0    blablabla     ble-tendre
##   father_project                    father father_germplasm
## 1          DEMO           C1_JFB_2012_0001               C1
## 2          DEMO         C1#a_JFB_2013_0001               C1
## 3          DEMO         C1#a_JFB_2014_0001               C1
## 4          DEMO           C1_JFB_2013_0001               C1
## 5          DEMO   Blé-du-Lot_JFB_2010_0001       Blé-du-Lot
## 6          DEMO Rouge-du-Roc_JFB_2010_0001     Rouge-du-Roc
##   father_person father_year father_germplasm_type father_alt
## 1           JFB        2012                  <NA>         NA
## 2           JFB        2013                  <NA>         NA
## 3           JFB        2014                  <NA>         NA
## 4           JFB        2013                  <NA>         NA
## 5           JFB        2010                  <NA>         NA
## 6           JFB        2010                  <NA>         NA
##   father_long father_lat father_total_generation_nb
## 1   0.616363   44.20314                          4
## 2   0.616363   44.20314                          6
## 3   0.616363   44.20314                          1
## 4   0.616363   44.20314                          6
## 5   0.616363   44.20314                          5
## 6   0.616363   44.20314                          8
##   father_local_generation_nb father_generation_confidence
## 1                          2                            1
## 2                          8                            0
## 3                          5                            1
## 4                          1                            1
## 5                          6                            1
## 6                          4                            1
##   father_comments reproduction_id reproduction_method_name is_male
## 1       blablabla             467                      <NA>       X
## 2       blablabla             484                      <NA>       X
## 3       blablabla            <NA>                      <NA>       X
## 4       blablabla             483                      <NA>       X
## 5       blablabla             454                     cross       M
## 6       blablabla             454                     cross       F
##   block selection_id selection_person mixture_id diffusion_id
## 1     1           40              JFB       <NA>         <NA>
## 2     1         <NA>             <NA>       <NA>         <NA>
## 3  <NA>         <NA>             <NA>       <NA>          266
## 4     1           48              JFB       <NA>         <NA>
## 5     1         <NA>             <NA>       <NA>         <NA>
## 6     1         <NA>             <NA>       <NA>         <NA>
##   relation_year_start relation_year_end
## 1                2012              2013
## 2                2013              2014
## 3                <NA>              <NA>
## 4                2013              2014
## 5                2010              2011
## 6                2010              2011
```

- the network.info matrix with information on relations in the network.

```
head(data_network$data$network.info)
```

```
##                   sl germplasm germplasm_type person year alt
## 1 C1#a_JFB_2013_0001        C1           <NA>    JFB 2013  NA
## 2 C1#a_JFB_2014_0001        C1           <NA>    JFB 2014  NA
## 3 C1#a_JSG_2014_0001        C1           <NA>    JSG 2014  NA
## 4 C1#c_JFB_2014_0001        C1           <NA>    JFB 2014  NA
## 5   C1_JFB_2011_0001        C1           <NA>    JFB 2011  NA
## 6   C1_JFB_2011_0001        C1           <NA>    JFB 2011  NA
##       long      lat diffusion id.diff reproduction mixture
## 1 0.616363 44.20314      <NA>    <NA>          <NA>    <NA>
## 2 0.616363 44.20314      <NA>    <NA>       harvest    <NA>
## 3 3.087025 45.77722   receive      19          <NA>    <NA>
## 4 0.616363 44.20314      <NA>    <NA>          <NA>    <NA>
## 5 0.616363 44.20314      <NA>    <NA>  harvest-sow    <NA>
## 6 0.616363 44.20314      <NA>    <NA>  harvest-sow    <NA>
##   selection cross.info
## 1 selection       <NA>
## 2      <NA>       <NA>
## 3      <NA>       <NA>
## 4 selection       <NA>
## 5      <NA>       <NA>
## 6      <NA>       <NA>
```

- the `Mdist` square matrix

```
dim(data_network$data$Mdist)
```

```
## [1] 98 98
```

You may want to fill gaps into your network data set regarding diffusion events (if the information is stored in SHiNeMaS!). This may be usefull, regarding a network on one person, to know where seed-lots come from. To do so, use `fill.diffusion.gap = TRUE`.

```
if(use.get.data){
        data_network_OLR = get.data(
                db_user = info_db$db_user, db_host = info_db$db_host, # db infos
                db_name = info_db$db_name, db_password = info_db$db_password,  # db infos
                query.type = "network", # network query
                person.in = "OLR", # person to keep
                filter.on = "father-son" # filter on father AND son
                )
} else {
        # 1. Query SHiNeMaS ...
        # Mixtures from replication have been deleted and replaced by reproductions.
        # 2. Create network matrix ...
        # 3. Link information to vertex and edges ...
        # 4. Get network information on seed-lots ...

        load("./data/data_network_OLR.RData")
        }
```

```
if(use.get.data){
        data_network_OLR_fill_gap = get.data(
                db_user = info_db$db_user, db_host = info_db$db_host, # db infos
                db_name = info_db$db_name, db_password = info_db$db_password, # db infos
                query.type = "network", # network query
                person.in = "OLR", # person to keep
                filter.on = "father-son", # filter on father AND son
                fill.diffusion.gap = TRUE
                )
} else {
        # 1. Query SHiNeMaS ...
        # Mixtures from replication have been deleted and replaced by reproductions.
        # 2. Create network matrix ...
        # 2.1. Fill diffusion gaps ...
        # 3. Link information to vertex and edges ...
        # 4. Get network information on seed-lots ...

        load("./data/data_network_OLR_fill_gap.RData")
        }
```

Sometimes, seed-lots are replicated and then mixted. This can be seen as 'false' mixture compared to 'real' mixture where the germplasm father and son are different. In order to take this into account in the analysis of the network, it can be useful to delete the 'false' mixture and declare it as a reproduction. This can be done with the argument `mixrep_to_repro` which tranform the mixtures of replications into reproductions, i.e. the network keeps only 'real' mixtures (i.e. different germplasm for father and son) and correct the information regarding what have been harvested and sown after as the mixture disappear. The value of `mixrep_to_repro` is `TRUE` by default.

3.2 Get the ggplots

Once you have acquired the data, you can create plots with the function `get.ggplot`.

```
default_network_ggplot = get.ggplot(data_network)

## As ggplot.type is NULL, ggplot.type is set to network-network, network-reproduction-sown,
network-reproduction-harvested, network-reproduction-positive-inter-selected, network-reproduction-po
network-reproduction-negative-inter-selected, network-reproduction-crossed, network-diffusion-sent,
network-diffusion-received, network-diffusion-relation, network-mixture-real, network-mixture-rep,
network-mixture-all, network-positive-intra-selected
## As ggplot.display is NULL, ggplot.display is set to c("barplot", "map").  Note that for
ggplot.type == "network-diffusion-relation", ggplot.display is set to "map"
## As x.axis and in.col are NULL, all the combinaisons of x.axis and in.col are done for barplot.

## [1] "A METTRE A JOUR QUAND ON AURA EVENT YEAR"
## [1] "A METTRE A JOUR QUAND ON AURA EVENT YEAR"
```

By default, all the possible plots or maps are done.

```
names(default_network_ggplot)

##  [1] "network-network"
##  [2] "network-reproduction-sown"
##  [3] "network-reproduction-harvested"
##  [4] "network-reproduction-positive-inter-selected"
##  [5] "network-reproduction-negative-inter-selected"
##  [6] "network-diffusion-sent"
##  [7] "network-diffusion-received"
```

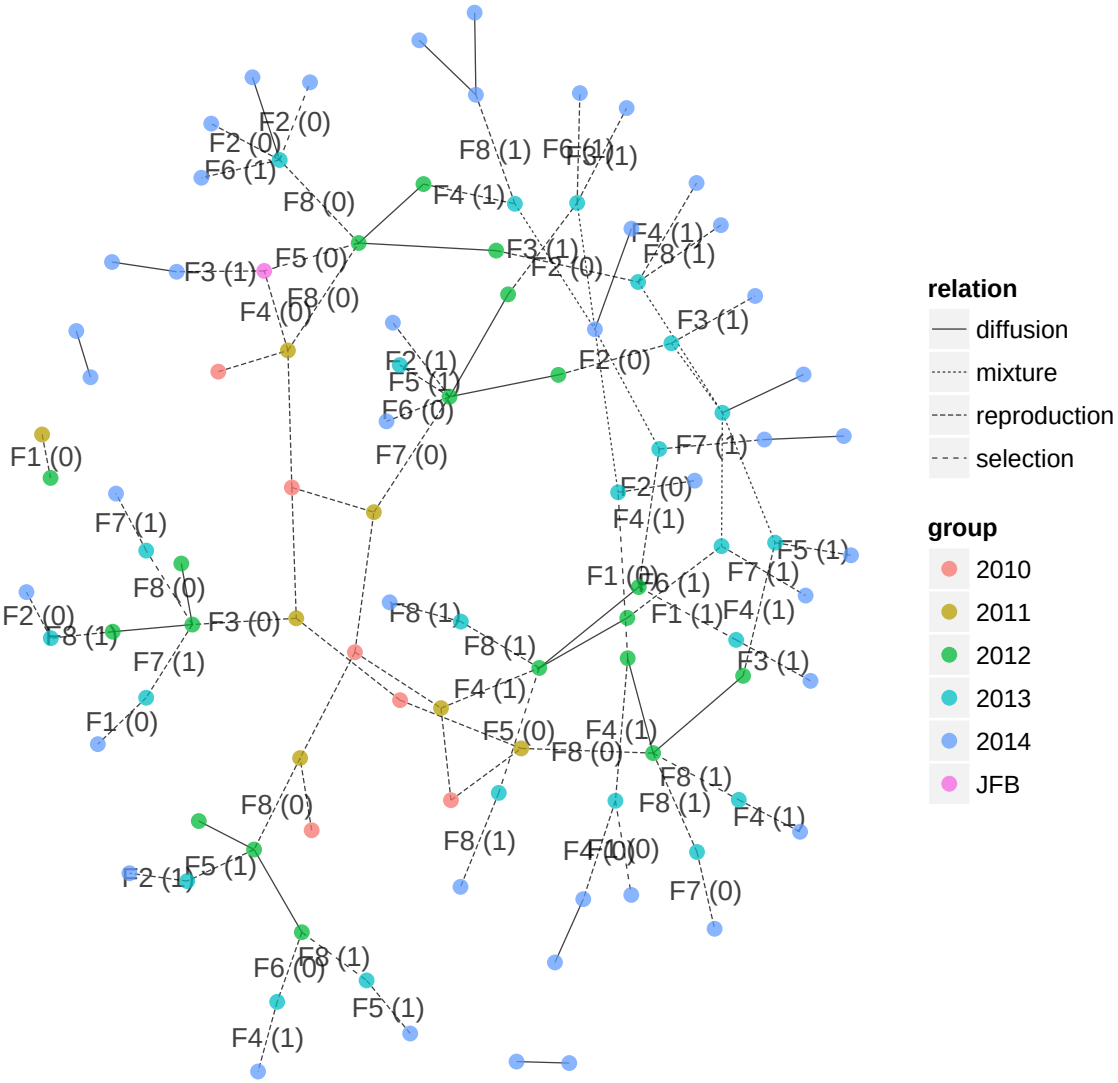
```
##  [8] "network-diffusion-relation"
##  [9] "network-mixture-real"
## [10] "network-mixture-rep"
## [11] "network-mixture-all"
## [12] "network-positive-intra-selected"
## [13] "network-reproduction-crossed"
```

It is possible to get only one plot by choosing one of the above names with the argument `ggplot.type`. Each `ggplot.type` with default and customized arguments are explained in the following sub-sections.

3.2.1 `ggplot.type = "network-network"`

Default arguments By default, the following network is displayed:

```
p_net_C1 = default_network_ggplot$"network-network"
p_net_C1
```

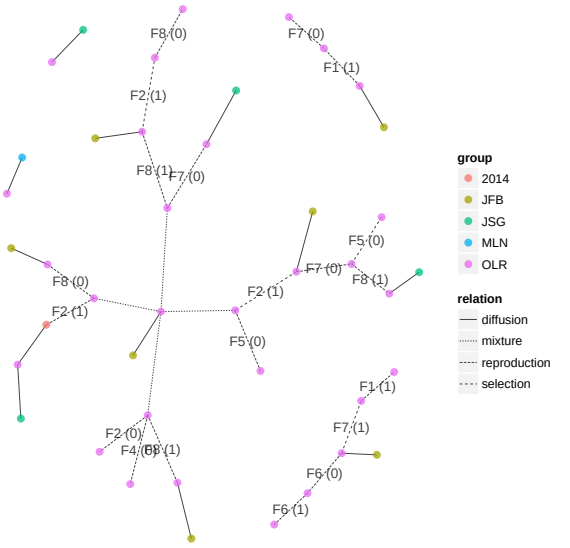


Regarding the argument `fill.diffusion.gap` in the function `get.data`:

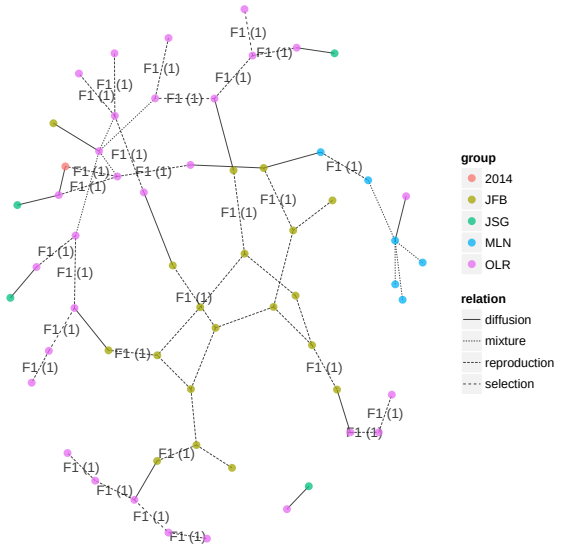
```
p_OLR = get.ggplot(
        data_network_OLR,
        ggplot.type = "network-network",
        vertex.color = "person"
        )
p_OLR = p_OLR$`network-network`

p_OLR_fill_gap = get.ggplot(
        data_network_OLR_fill_gap,
        ggplot.type = "network-network",
        vertex.color = "person"
        )
p_OLR_fill_gap = p_OLR_fill_gap$`network-network`
```

We can see that extra information is added to know where the seed-lots come from.

p_OLR



p_OLR_fill_OLR



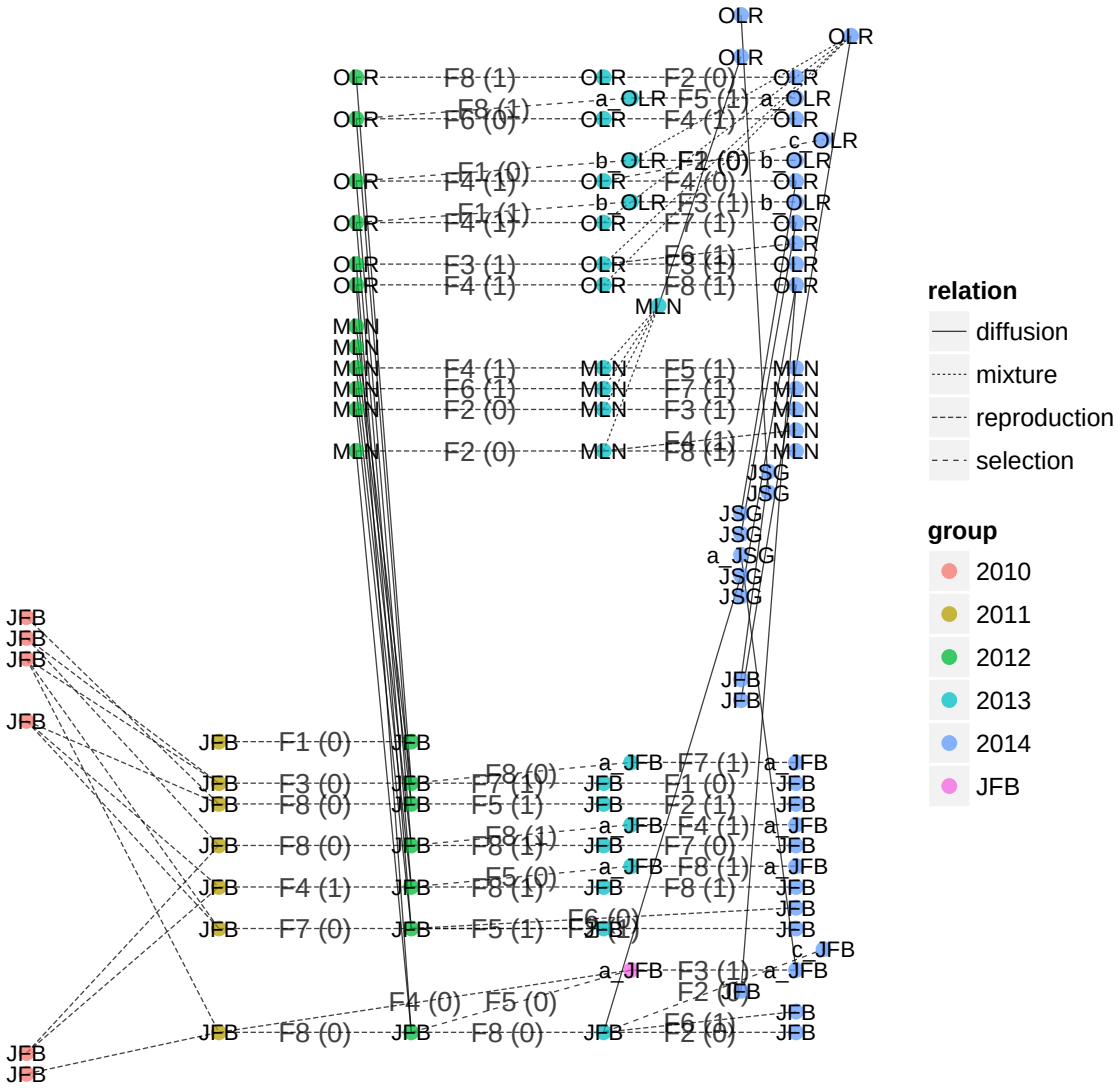
Custom arguments It is possible to tune the following arguments (Table 1):

argument	default value	description
vertex.size	3	size of the vertex
vertex.color	"year"	color of the vertex. It can be chosen according to "person", "germplasm" or "year". If NULL, it is in black.
organise.sl	FALSE	organize seed-lots for an easier visualization
hide.labels.parts	"all"	parts of the label hidden: "germplasm", "person", "year", "person:germplasm", "year:germplasm", "person:year", "all". "all" means that no label is dispayed. If NULL labels are displayed. Labels are based on seed-lots names under the form germplasm_year_person_digit. For easier visualization, digit is never displayed unless you choose NULL.
display.labels.sex	TRUE	if TRUE, displays the sex of the seed-lot if it has been used in a cross. Nothing is displayed if hide.labels.parts = "all".
labels.generation	TRUE	if TRUE, displays generation for each reproduction.
labels.size	3	size of the labels

Table 1: Possible arguments to customize regarding network.

For example,

```
get.ggplot(
        data_network,
        ggplot.type = "network-network",
        organise.sl = TRUE,
        hide.labels.parts = "year:germplasm",
        vertex.color = "year"
        )

## $`network-network`
```



3.2.2 `ggplot.type = "network-reproduction-harvested"`

`ggplot.type = "network-reproduction-harvested"` corresponds to seed-lots that have been harvested after a reproduction.

Two plots are possible regarding `ggplot.display`.

argument	default value	description
`ggplot.display`	`NULL`	`"barplot"` or `"map"`. It can be a vector of several elements i.e. `c("barplot", "map")`. `NULL` by default: both are done.

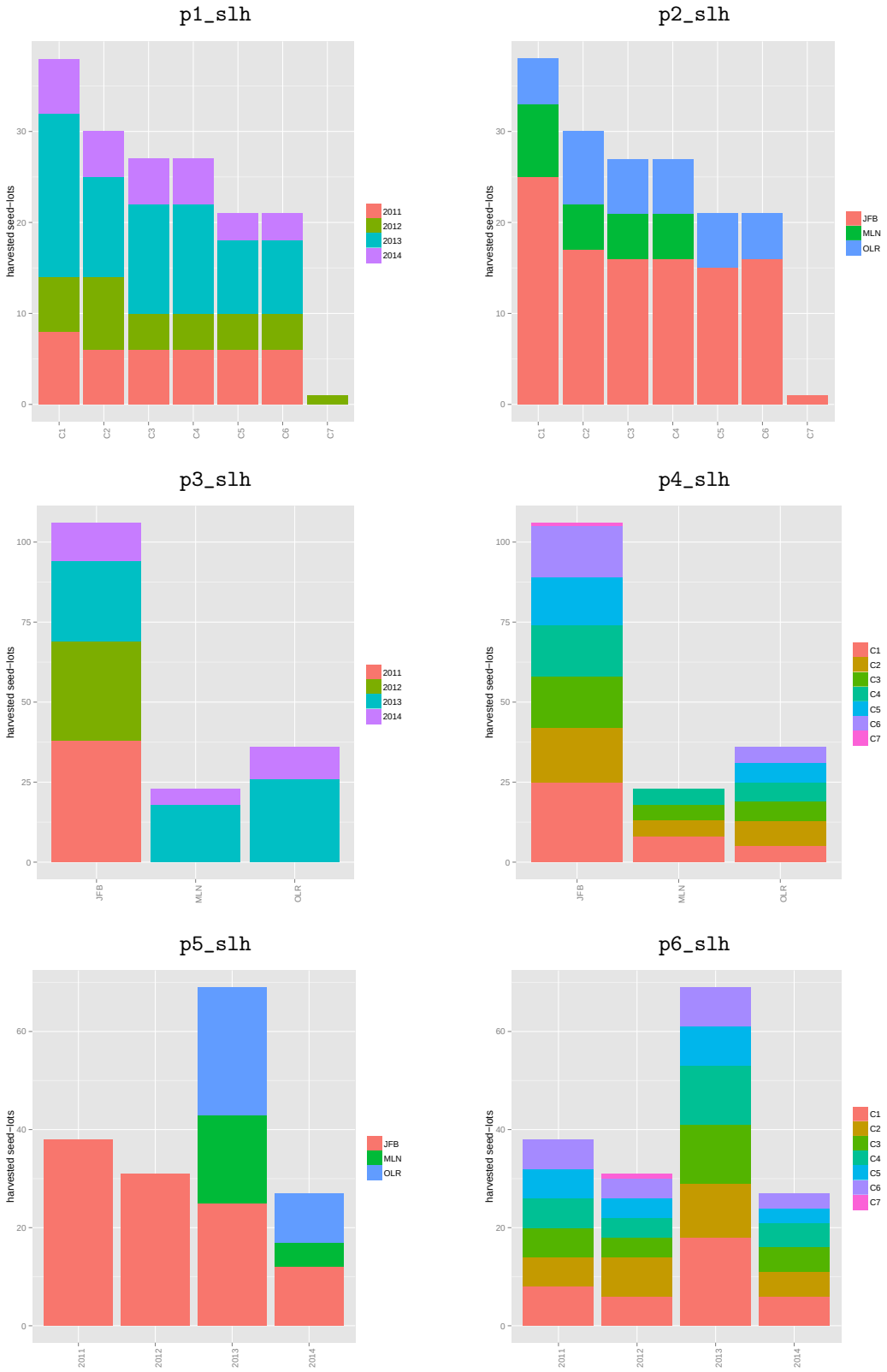
`ggplot.display = barplot` For `ggplot.display = "barplot"`, you may choose what you want in the x axis (`x.axis` argument) and in color (`in.col` argument). The possible values for `x.axis` and `in.col` are: `germplasm`, `year`, `person`. By default all combinaisons of `x.axis` and `in.col` are done with default argument settings. The name of the plot is under the form `x.axis-in.col`.

- Default arguments

```
p_slh = default_network_ggplot$`network-reproduction-harvested`$barplot
p1_slh = p_slh$`germplasm-year`$`x.axis-1|in.col-1`
p2_slh = p_slh$`germplasm-person`$`x.axis-1|in.col-1`
p3_slh = p_slh$`person-year`$`x.axis-1|in.col-1`
p4_slh = p_slh$`person-germplasm`$`x.axis-1|in.col-1`
p5_slh = p_slh$`year-person`$`x.axis-1|in.col-1`
p6_slh = p_slh$`year-germplasm`$`x.axis-1|in.col-1`
```

with

plot	x.axis	in.col
p1_slh	"germplasm"	"year"
p2_slh	"germplasm"	"person"
p3_slh	"person"	"year"
p4_slh	"person"	"germplasm"
p5_slh	"year"	"person"
p6_slh	"year"	"germplasm"



- Custom arguments

  It is possible to tune the following arguments:

argument	default value	description
`x.axis`	`NULL`	factor displayed on the x.axis of a plot: `"germplasm"`, `"year"` or `"person"` referring to the attributes of a seed-lots. If NULL, all the combinaison are done for `x.axis` and `in.col`.
`in.col`	`NULL`	display in color of a plot: `"germplasm"`, `"year"` or `"person"` referring to the attributes of a seed-lots. If `NULL`, `in.col` is not displayed.
`nb_parameters_per_plot_x.axis`	`NULL`	the number of parameters per plot on `x.axis` argument
`nb_parameters_per_plot_in.col`	`NULL`	the number of parameters per plot for `in.col` argument

Table 2: Possible arguments to custom arguments regarding barplot.

For example, for a given `x.axis` and a given `in.col`:

```
p_slh = get.ggplot(
        data_network,
        ggplot.type = "network-reproduction-harvested",
        ggplot.display = "barplot",
        x.axis = "person",
        in.col = "year"
        )
```

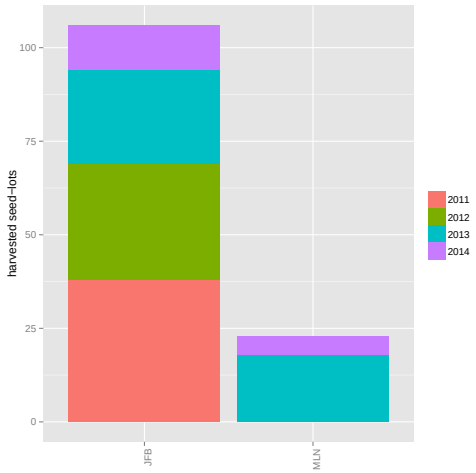
It can be useful to separate the plot in several plots regarding `x.axis`:

```
p1_slh = get.ggplot(
        data_network,
        ggplot.type = "network-reproduction-harvested",
        ggplot.display = "barplot",
        x.axis = "person",
        in.col = "year",
        nb_parameters_per_plot_x.axis = 2
        )

p11_slh = p1_slh$`network-reproduction-harvested`$barplot$`person-year`$`x.axis-1|in.col-1`
p12_slh = p1_slh$`network-reproduction-harvested`$barplot$`person-year`$`x.axis-2|in.col-1`
```

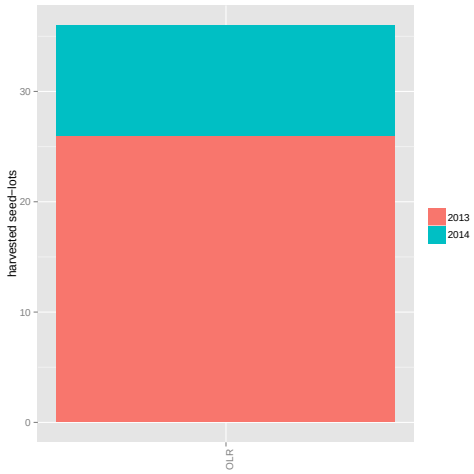
Note that you can easily change settings of the plot as it is a ggplot object, for example

```
p12_slh_bw = p12_slh + theme_bw()
```

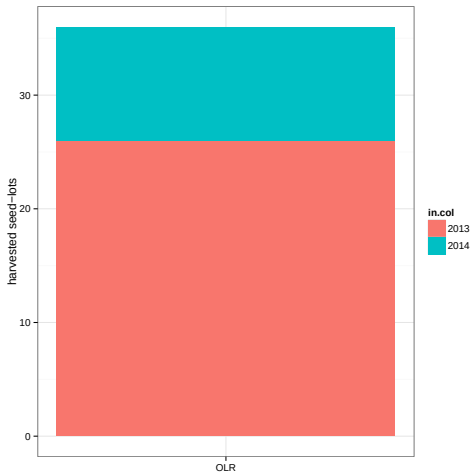
p11_slh



p12_slh



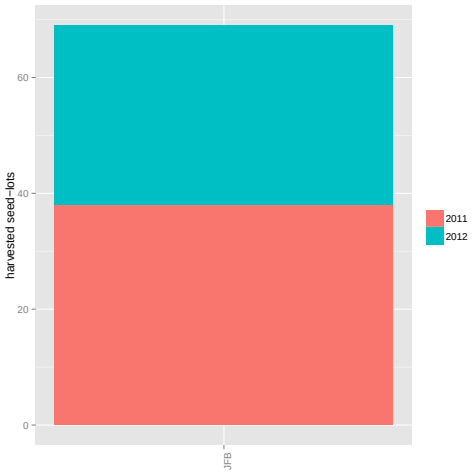
p12_slh_bw



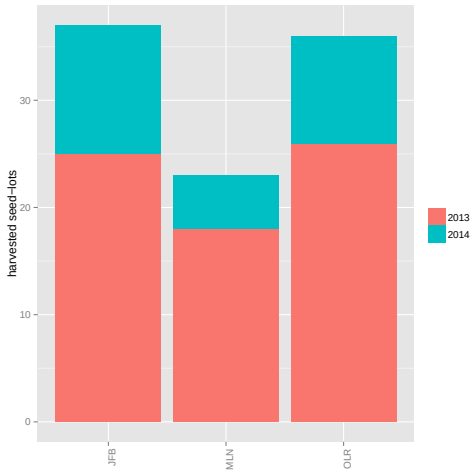
Or for `in.col`:

```
p2_slh = get.ggplot(
        data_network,
        ggplot.type = "network-reproduction-harvested",
        ggplot.display = "barplot",
        x.axis = "person",
        in.col = "year",
        nb_parameters_per_plot_in.col = 2
        )

p21_slh = p2_slh$`network-reproduction-harvested`$barplot$`person-year`$`x.axis-1|in.col-1`
p22_slh = p2_slh$`network-reproduction-harvested`$barplot$`person-year`$`x.axis-1|in.col-2`
```

p21_slh



p22_slh



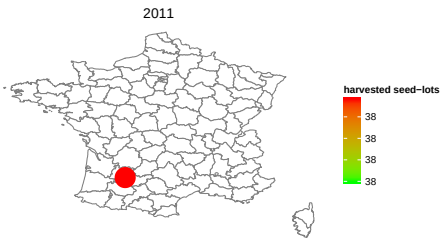
Or for both, for each subset of `x.axis`, you have the set of `in.col`.

`ggplot.display = map` There are as many maps as years and a map with all the years mixed. The scale represent the number of seed-lots harvested.

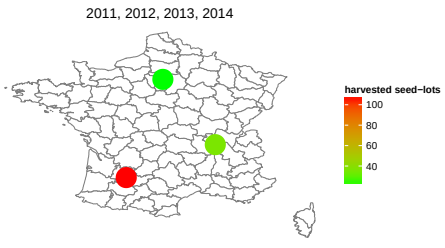
- Default arguments

```
m_slh = default_network_ggplot$`network-reproduction-harvested`$map
names(m_slh)

## [1] "map-[2011]"                    "map-[2012]"
## [3] "map-[2013]"                    "map-[2014]"
## [5] "map-[2011, 2012, 2013, 2014]"

m1_slh = m_slh$`map-[2011]`
m5_slh = m_slh$`map-[2011, 2012, 2013, 2014]`
```

m1_slh



m5_slh



- Custom arguments

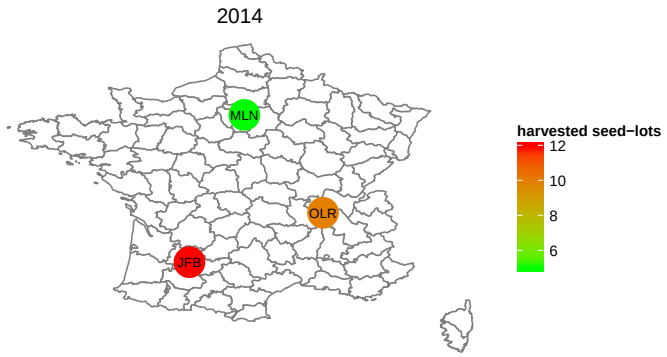
  It is possible to tune the following arguments (Table 3):

argument	default value	description
`hide.labels.parts`	`NULL`	can be NULL or "all" as only person can be display. `NULL` by default
`labels.size`	`3`	size of the labels
`location.map`	`"france"`	location of the map see `?map` for more details
`pie.size`	`.5`	size of the pie when using pies

Table 3: Possible arguments to custom arguments regarding maps.

For example,

```
p_slh = get.ggplot(
        data_network,
        ggplot.type = "network-reproduction-harvested",
        ggplot.display = "map",
        hide.labels.parts = NULL
        )

p_slh$`network-reproduction-harvested`$map$`map-[2014]`
```



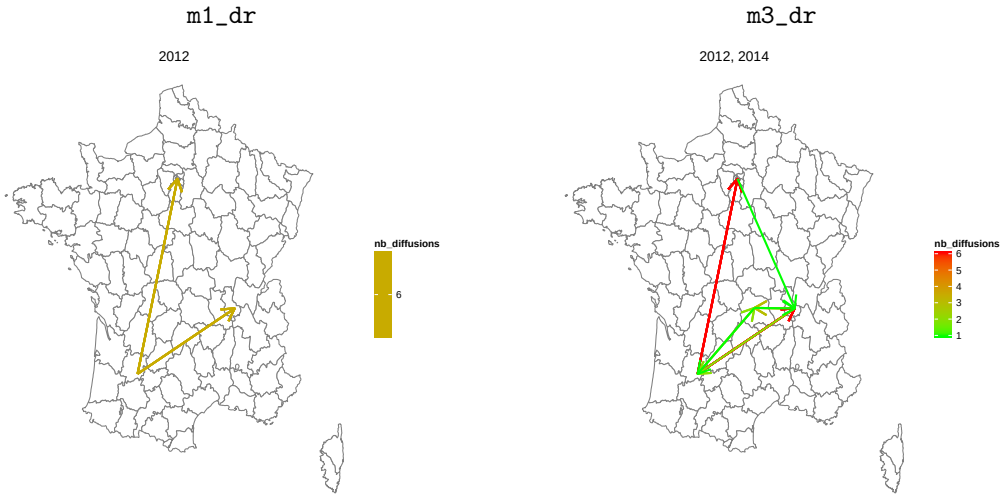
3.2.3 `ggplot.type = "network-diffusion-relation"`

`ggplot.type = "network-diffusion-relation"` corresponds to diffusion of seed-lots on a map. Note that `ggplot.display` is useless here.

There are as many maps as years and a map with all the years mixed. The size of the arrows is proprotionnal to the number of diffusions (`nb_diffusions`).

- Default arguments

```
m_dr = default_network_ggplot$`network-diffusion-relation`$map
names(m_dr)

## [1] "map-[2012]"       "map-[2014]"       "map-[2012, 2014]"

m1_dr = m_dr$`map-[2012]`
m3_dr = m_dr$`map-[2012, 2014]`
```



- Custom arguments

  Arguments can be customized related to maps. See Table 3.

3.2.4 `ggplot.type = "network-reproduction-crossed"`

Information on the crosses, their parents (mother and father) and grandparents (grandmother and grandfather). It may be useful to know information on grandparents, especially location, if the cross have been done on another location.

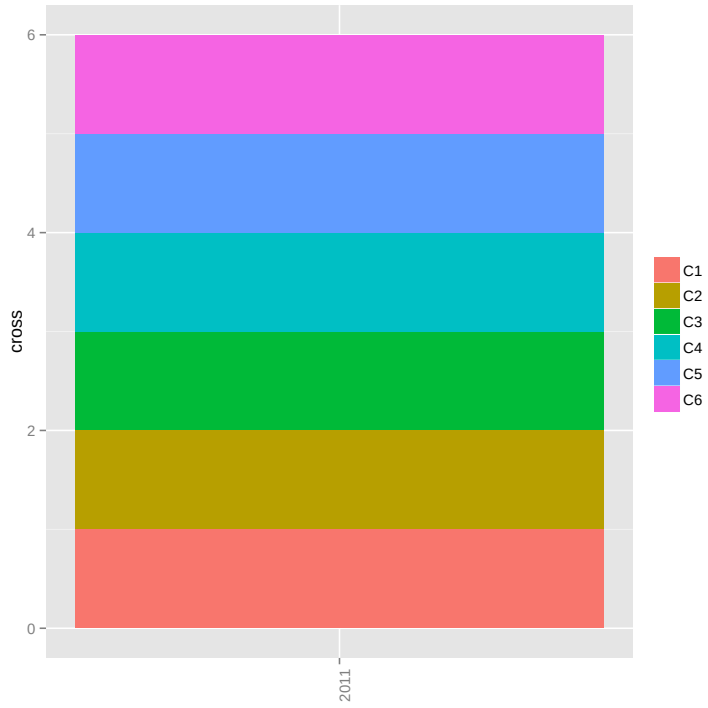
Note that in this example, it represents all the cross where `Rouge-du-Roc` is involved as `"father"` or `"son"` in relation. This is because in `get.data`, the following arguments were set: `germplasm.in = Rouge-du-Roc` and `filter.on = "father-son"`.

- Default arguments

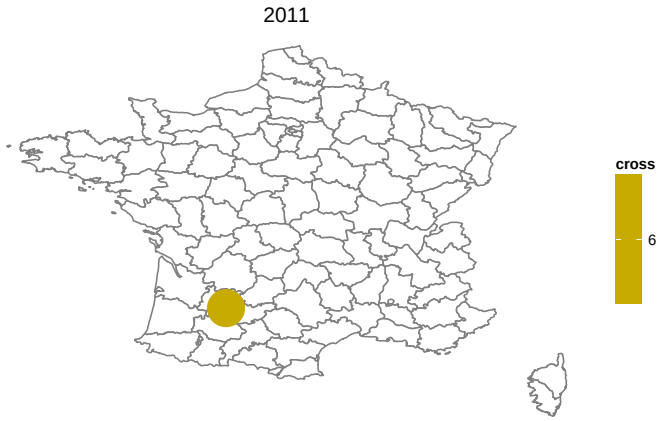
```
cb = default_network_ggplot$`network-reproduction-crossed`$barplot
names(cb)

## [1] "cross"       "father"      "grandfather" "mother"
## [5] "grandmother"

cm = default_network_ggplot$`network-reproduction-crossed`$map
names(cm)

## [1] "cross"       "father"      "grandfather" "mother"
## [5] "grandmother"
```

For example:

```
cb$cross$barplot$`year-germplasm`$`x.axis-1|in.col-1`
```




```
cm$cross$map$`map-[2011]`
```



- Custom arguments

  Arguments can be customed related to barplots (Table 2) or maps (Table 3).

3.2.5 Others `ggplot.type`

For the other `ggplot.type`, as it is exactly the same type of plots than for `ggplot.type = "network-reproduction-harve` no example is displayed in this vignette.

The following table describe the different `ggplot.type`:

ggplot.type	description
`"network-reproduction-sown"`	seed-lots that have been sown.
`"network-reproduction-positive-inter-selected"`	seed-lots that have been harvested after a reproduction and have been sown the next season. It corresponds to positive inter selection.
`"network-reproduction-negative-inter-selected"`	seed-lots that have been harvested after a reproduction and have NOT been sown the next season. It corresponds to negative inter selection.
`"network-diffusion-sent"`	seed-lots that have been sent to another location.
`"network-diffusion-received"`	seed-lots that have been received in a given location.
`"network-mixture-real"`	seed-lots that have been 'really' mixed into a mixture.
`"network-mixture-rep"`	seed-lots coming from replication that have been mixed into a mixture. Note that it is `NULL` when the argument `mixrep_to_repro` in `get.data` is `TRUE` (default argument).
`"network-mixture-all"`	seed-lots that have been mixed into a mixture (i.e. `"network-mixture-real"` + `"network-mixture-rep"`.
`"network-positive-intra-selected"`	seed-lots where intra germplasm selection have been done. It corresponds to mass selection.

3.3 Get the tables

It is possible to get the table from the data used for the plot. For example,

```
p = default_network_ggplot$`network-reproduction-crossed`$barplot$cross$barplot
p = p$`year-germplasm`$`x.axis-1|in.col-1`
tab_p = get.table(p)
tab_p

##   mean germplasm year
## 1    1        C1 2011
## 2    1        C2 2011
## 3    1        C3 2011
## 4    1        C4 2011
## 5    1        C5 2011
## 6    1        C6 2011
```

4 Data linked to the seed-lots and to the relations between seed-lots

4.1 Get the data set

Three types of queries are possible:

- `query.type = "data-classic"` to study classic variables for each seed-lot or relation between seed-lots (subsection 4.1.1)
- `query.type = "data-S"` to study selection differential (subsection 4.1.2)
- `query.type = "data-SR"` to study response to selection (subsection 4.1.2)

You can set filters to the query with the following argument :

- `filter.in` to choose nothing except `filter.in`
- `filter.out` to choose everything except `filter.out`

values of filters can be `germplasm`, `germplasm.type`, `year`, `person`, `project`, `seed-lots`, `relation`, `reproduction.type` or `variable`.

It is important to choose if you apply the filter on the `father` or the `son` of a relation. This can be done with the argument `filter.on`. Possible values are `"father"`, `"son"` or `"father-son"`.

You can either query information on relation between seed-lots or on seed-lots. This is set with the argument `data.type`. Possibles values are

- `"relation"` for data linked to relation between seed lots and
- `"seed-lots"` for data linked to seed lots

The function returns a list with

- a data frame with the data set
- a list with data set with individuals that are correlated for a set of variables
- the description of methods used for each variable with its description and units

Note that for data linked to seed-lots, all the data are correlated as there is one measure for a given seed-lot. Therefore the element of the list for correlated data is always NULL. For data linked to relations, as it can be linked to individual within a seed-lot, data may be correlated (data taken on the same individual) or not.

A data set with only correlated variables for each individual is useful when doing multivariate analysis such as PCA.

The names of the variables in the output of the function are under the form `variable_name--methods`.

In the following examples, three variables are studied:

```
vec_variables = c("hauteur", "pmg", "poids_epis")
```

4.1.1 `query.type = "data-classic"`

```
if(use.get.data){
        data_classic = get.data(
                db_user = info_db$db_user, db_host = info_db$db_host, # db infos
                db_name = info_db$db_name, db_password = info_db$db_password, # db infos
                query.type = "data-classic", # data-classic query
                person.in = "JFB", # person to keep
                filter.on = "father-son", # filters on father AND son
```

```
                data.type = "relation", # data linked to relation between seed-lots
                variable = vec_variables, # the variables to display
                project.in = "DEMO" # the project
                )
} else {
        # 1. Query SHiNeMaS ...
        # 2. Set up data set ...
        # |============================================================| 100%

        load("./data/data_classic.RData")
}

names(data_classic$data)

## [1] "data"
## [2] "data.with.correlated.variables"
## [3] "methods"

colnames(data_classic$data$data)

##  [1] "son_species"                  "son_project"
##  [3] "son"                          "son_ind"
##  [5] "son_year"                     "son_germplasm"
##  [7] "son_germplasm_type"           "son_person"
##  [9] "son_alt"                      "son_long"
## [11] "son_lat"                      "son_total_generation_nb"
## [13] "son_local_generation_nb"      "son_generation_confidence"
## [15] "son_comments"                 "father_species"
## [17] "father_project"               "father"
## [19] "father_year"                  "father_germplasm"
## [21] "father_germplasm_type"        "father_person"
## [23] "father_alt"                   "father_long"
## [25] "father_lat"                   "father_total_generation_nb"
## [27] "father_local_generation_nb"   "father_generation_confidence"
## [29] "father_comments"              "reproduction_id"
## [31] "reproduction_method_name"     "selection_id"
## [33] "selection_person"             "mixture_id"
## [35] "diffusion_id"                 "relation_year_start"
## [37] "relation_year_end"            "X"
## [39] "Y"                            "block"
## [41] "hauteur---hauteur_tige"       "hauteur---hauteur_tige$date"
## [43] "pmg---pmg"                    "pmg---pmg-2"
## [45] "pmg---pmg-2$date"             "pmg---pmg$date"
## [47] "poids_epis---poids_epis"      "poids_epis---poids_epis$date"

head(data_classic$data$methods)

##    method_name variable_name method_description unit
## 1 hauteur_tige       hauteur
## 2   poids_epis    poids_epis
## 3          pmg           pmg
## 4        pmg-2           pmg
##        variable---methods
## 1  hauteur---hauteur_tige
## 2 poids_epis---poids_epis
## 3               pmg---pmg
## 4             pmg---pmg-2
```

The data with correlated variables :

```
names(data_classic$data$data.with.correlated.variables)

## [1] "A_2013" "B_2013" "A_2014" "B_2014"
```

The methods related to each variable :

```
data_classic$data$methods

##    method_name variable_name method_description unit
## 1 hauteur_tige       hauteur
## 2   poids_epis    poids_epis
## 3          pmg           pmg
## 4        pmg-2           pmg
##        variable---methods
## 1  hauteur---hauteur_tige
## 2 poids_epis---poids_epis
## 3               pmg---pmg
## 4             pmg---pmg-2
```

4.1.2 Selection differential (`query.type = "data-S"`), response to selection (`query.type = "data-SR"`) and heritability

For a given trait, selection differential corresponds to the difference between mean of the selected spikes and mean of the bulk (i.e. spikes that have not been selected). Response to selection correponds to the difference between mean of spikes coming from the selected spikes and the spikes coming from the bulk (Figure 3).

Selection differential (S) and response to selection (R) are linked with the realized heritability (h_r^2):

$$R = h_r^2 \times S$$

See appendix B for more details on the theory behind this.

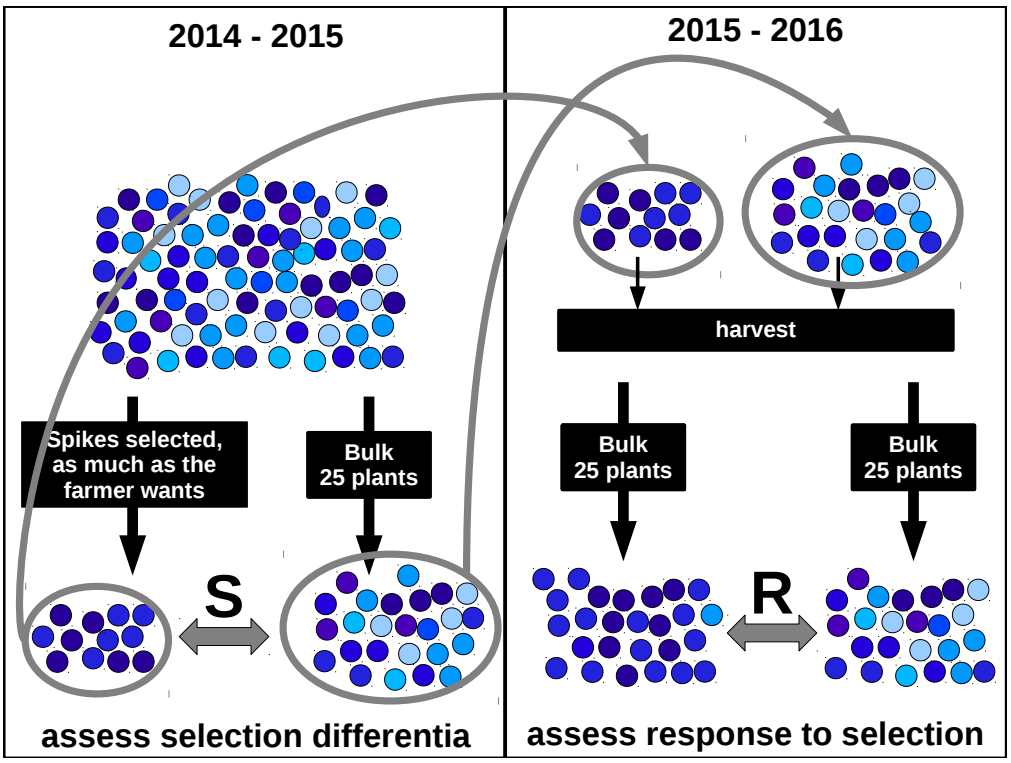


Figure 3: Seletion differential (S) in 2014-2015 and response to selection (R) in 2015-2016. Circles and arrows in gray represent the seed-lots that have been sown in 2015 after harvest in 2015.

`query.type = "data-S"` The data frame returned has a column `"expe"` which corresponds to an id of one selection differential.

```
if(use.get.data){
        data_S = get.data(
                db_user = info_db$db_user, db_host = info_db$db_host,
                db_name = info_db$db_name, db_password = info_db$db_password,
                query.type = "data-S",
                person.in = "JFB",
                filter.on = "father-son",
                data.type = "relation",
                variable = vec_variables,
                project.in = "DEMO"
                )
} else {
        # 1. Query SHiNeMaS ...
        # 2. Set up data set ...
        # |============================================================| 100%

        load("./data/data_S.RData")
}

colnames(data_S$data$data)

##  [1] "son"                           "expe"
##  [3] "sl_statut"                     "expe_name"
##  [5] "expe_name_2"                   "son_species"
##  [7] "son_project"                   "son_ind"
```

```
##  [9] "son_year"                      "son_germplasm"
## [11] "son_germplasm_type"            "son_person"
## [13] "son_alt"                       "son_long"
## [15] "son_lat"                       "son_total_generation_nb"
## [17] "son_local_generation_nb"       "son_generation_confidence"
## [19] "son_comments"                  "father_species"
## [21] "father_project"                "father"
## [23] "father_year"                   "father_germplasm"
## [25] "father_germplasm_type"         "father_person"
## [27] "father_alt"                    "father_long"
## [29] "father_lat"                    "father_total_generation_nb"
## [31] "father_local_generation_nb"    "father_generation_confidence"
## [33] "father_comments"               "reproduction_id"
## [35] "reproduction_method_name"      "selection_id"
## [37] "selection_person"              "mixture_id"
## [39] "diffusion_id"                  "relation_year_start"
## [41] "relation_year_end"             "X"
## [43] "Y"                             "block"
## [45] "hauteur---hauteur_tige"        "hauteur---hauteur_tige$date"
## [47] "pmg---pmg"                     "pmg---pmg-2"
## [49] "pmg---pmg-2$date"              "pmg---pmg$date"
## [51] "poids_epis---poids_epis"       "poids_epis---poids_epis$date"
```

`query.type = "data-SR"` The data frame returned has a column `"expe"` which corresponds to an id of one selection differential and the corresponding response to selection

The query takes into account when selection have been done in a seed lot, that this seed lot have been merged and then have been sown. It is the case when selection have been carried out in a replication that have been merge afterwards. Even if this case should not arise, it might happen.

```
if(use.get.data){
        data_SR = get.data(
                db_user = info_db$db_user, db_host = info_db$db_host,
                db_name = info_db$db_name, db_password = info_db$db_password,
                query.type = "data-SR",
                person.in = "JFB",
                filter.on = "father-son",
                data.type = "relation",
                variable = vec_variables,
                project.in = "DEMO"
                )
} else {
        # 1. Query SHiNeMaS ...
        # 2. Set up data set ...
        # |============================================================| 100%

        load("./data/data_SR.RData")
}

colnames(data_SR$data$data)
```

```
##  [1] "son"                           "expe"
##  [3] "sl_statut"                     "g"
##  [5] "expe_name"                     "expe_name_2"
##  [7] "son_species"                   "son_project"
##  [9] "son_ind"                       "son_year"
```

```
## [11] "son_germplasm"                "son_germplasm_type"
## [13] "son_person"                   "son_alt"
## [15] "son_long"                     "son_lat"
## [17] "son_total_generation_nb"      "son_local_generation_nb"
## [19] "son_generation_confidence"    "son_comments"
## [21] "father_species"               "father_project"
## [23] "father"                       "father_year"
## [25] "father_germplasm"             "father_germplasm_type"
## [27] "father_person"                "father_alt"
## [29] "father_long"                  "father_lat"
## [31] "father_total_generation_nb"   "father_local_generation_nb"
## [33] "father_generation_confidence" "father_comments"
## [35] "reproduction_id"              "reproduction_method_name"
## [37] "selection_id"                 "selection_person"
## [39] "mixture_id"                   "diffusion_id"
## [41] "relation_year_start"          "relation_year_end"
## [43] "X"                            "Y"
## [45] "block"                        "hauteur---hauteur_tige"
## [47] "hauteur---hauteur_tige$date"  "pmg---pmg"
## [49] "pmg---pmg-2"                  "pmg---pmg-2$date"
## [51] "pmg---pmg$date"               "poids_epis---poids_epis"
## [53] "poids_epis---poids_epis$date"
```

4.2 Get the ggplots

To get the plots, use the function `get.ggplot`.
First of all ,you need to choose

- on which data set you get the ggplots. This can be done on `correlated_group` having the following argument:
  - `NULL`, the default argument (`shinemas2R::get.data()$data$data`)
  - the name of a given `correlation_group` (`shinemas2R::get.data()$data$data.with.correlated.variables`
- if you want to merge germplasm name and selection name with the argument `merge_g_and_s` which is `TRUE` by default

The names of the variables are under the form `variable_name--methods`:

```
plot_vec_variables = c(
        "hauteur---hauteur_tige",
        "pmg---pmg",
        "poids_epis---poids_epis"
        )
```

4.2.1 ggplots for `data-classic`

- barplot, boxplot, interaction, radar and biplot
  - Default arguments

```
default_data_classic_ggplot = get.ggplot(
        data_classic,
        vec_variables = plot_vec_variables
        )
```

```
## As ggplot.type is NULL, ggplot.Type is set to data-barplot, data-boxplot, data-interactio
data-radar, data-biplot
## As x.axis and in.col are NULL, all the combinaisons of x.axis and in.col are done
for data-barplot, data-boxplot and data-interaction.
## As in.col is NULL, each in.col are done for data-radar and data-biplot.

## [1] "A Virer quand les données seront propres dans get.data"
## [1] "A Virer quand les données seront propres dans get.data"
## [1] "A Virer quand les données seront propres dans get.data"

## For data-biplot, hide.labels.parts has been set to NULL instead of "all".
```

By default, the following plots are done:

```
names(default_data_classic_ggplot)

## [1] "data-barplot"     "data-boxplot"     "data-interaction"
## [4] "data-radar"       "data-biplot"
```

which correpond to `ggplot.type` arguments:

`ggplot.type`	description
`"data-barplot"`	barplot, there is one barplot per variable
`"data-boxplot"`	boxplot, there is one boxplot per variable
`"data-interaction"`	interaction, there is one interation plot per variable
`"data-radar"`	radar, there is one radar per set of variables (i.e. the whole `vec_variables`). It is the mean that is represented.
`"data-biplot"`	biplot, there is one biplot per pairs of variables. Note that no plots are possible for between raw data linked to individuals and raw data linked to relation.

For `ggplot.type = data-barplot`, `data-boxplot` and `data-interaction`, you may chose what you want in the x axis (`x.axis` argument) and in color (`in.col` argument). The possible values for `x.axis` and `in.col` are: `germplasm`, `year`, `person`. By default all combinaisons of `x.axis` and `in.col` are done with default argument settings. The name of the plot is under the form `x.axis-in.col`.

```
names(default_data_classic_ggplot$`data-barplot`)

## [1] "germplasm-year"   "germplasm-person" "person-year"
## [4] "person-germplasm" "year-person"      "year-germplasm"
```

Knowing a combinaison of `x.axis` and `in.col`, choose the variable you want to get. For example:

* For barplot:

```
p_cal_ph_bar_all = default_data_classic_ggplot$`data-barplot`$`germplasm-year`
p_cal_ph_bar = p_cal_ph_bar_all$`hauteur---hauteur_tige`$`x.axis-1|in.col-1`
```

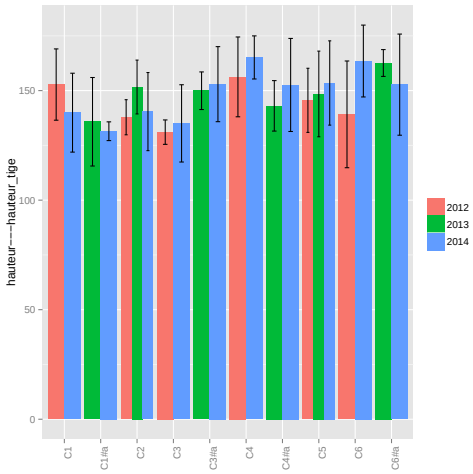
* For boxplot:

```
p_cal_ph_box_all = default_data_classic_ggplot$`data-boxplot`$`germplasm-year`
p_cal_ph_box = p_cal_ph_box_all$`hauteur---hauteur_tige`$`x.axis-1|in.col-1`
```

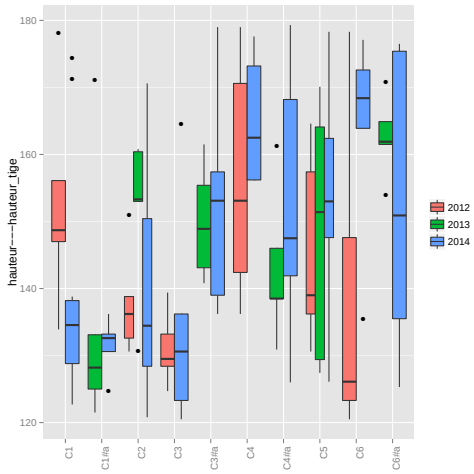
* For interaction plot:

```
p_cal_ph_int_all = default_data_classic_ggplot$`data-interaction`$`year-germplasm`
p_cal_ph_int = p_cal_ph_int_all$`hauteur---hauteur_tige`$`x.axis-1|in.col-1`
```

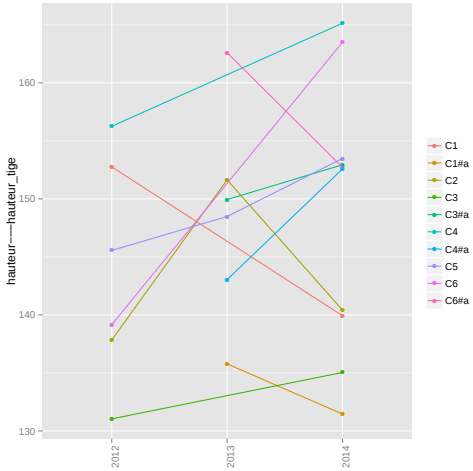
p_cal_ph_bar



p_cal_ph_box



p_cal_ph_int



For `ggplot.type = data-radar`, `data-biplot`, you may choose what you want in color (`in.col` argument). The possible values for `in.col` are: `germplasm`, `year` and `person`.

```
names(default_data_classic_ggplot$`data-radar`)

## [1] "NA-year"      "NA-person"    "NA-germplasm"
```

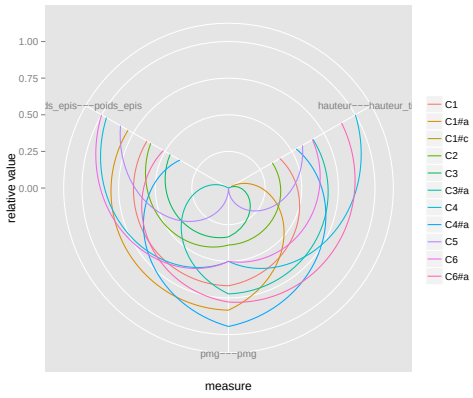
For example,

```
p_cal_rad_all = default_data_classic_ggplot$`data-radar`
p_cal_rad = p_cal_rad_all$`NA-germplasm`$`x.axis-1|in.col-1`
```

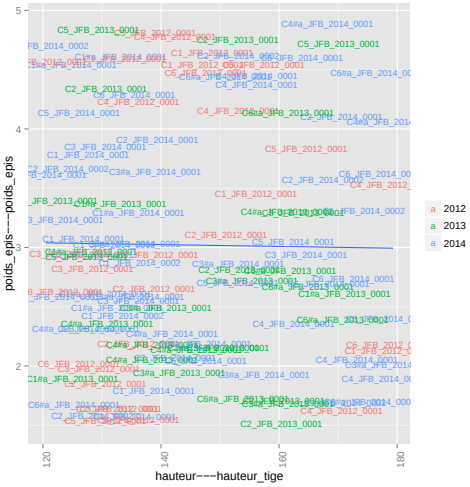
For biplot, you must choose the pair of variables you wish, for example,

```
p_cal_ph_sl_bip_all = default_data_classic_ggplot$`data-biplot`
p_cal_ph_sl_bip = p_cal_ph_sl_bip_all$`NA-year`
p_cal_ph_sl_bip = p_cal_ph_sl_bip$`hauteur---hauteur_tige - poids_epis---poids_epis`
p_cal_ph_sl_bip = p_cal_ph_sl_bip$`x.axis-1|in.col-1`
```

`p_cal_rad`



`p_cal_ph_sl_bip`



– Custom arguments

  According to `ggplot.type`, arguments can be customized:

argument	default value	description	data-barplot	data-boxplot	data-interaction	data-radar	data-biplot
ggplot.on	"son"	"father" or "son" depending on which seed-lot you want to plot..	X	X	X	X	X
x.axis	NULL	factor displayed on the x.axis of a plot: "germplasm", "year" or "person" referring to the attributes of a seed-lots. If NULL, all the combinaison are done for x.axis and in.col.	X	X	X		
in.col	NULL	display in color of a plot: "germplasm", "year" or "person" referring to the attributes of a seed-lots. If NULL, in.col is not displayed. Note it is compulsory for data-biplot and data-radar as in these cases x.axis is not used.	X	X	X	X	X
nb_parameters_per_plot_x.axis	NULL	the number of parameters per plot on x.axis argument	X	X	X		
nb_parameters_per_plot_in.col	NULL	the number of parameters per plot for in.col argument	X	X	X	X	X
hide.labels.parts	"all"	parts of the label hidden: "germplasm", "person", "year", "person:germplasm", "year:germplasm", "person:year", "all". "all" means that no label is dispayed. If NULL labels are displayed. Labels are based on seed-lots names under the form germplasm_year_person_digit. For "data-biplot", the default value is NULL. For easier visualisation, digit is never displayed unless you choose NULL.					X

Table 4: Possible arguments regarding ggplot.type. A cross (X) means that for a given ggplot.type, a given argument can be used.

- pies on network
  - Default arguments

    It is possible to add pies on seed-lots represented on a network. The pie represent the distribution of a given variable for a given seed-lot.

    In the following example, as we're working with encrypted data, get.ggplot reverse the transcription to query SHiNeMaS and encrypt again.

    Note that this example is done on a little data set. Indeed, the pies are drawn with a polygon which needs lots of points that take memory.

```
if(use.get.data){

        data_classic_bis = get.data(
        db_user = info_db$db_user, db_host = info_db$db_host, # db infos
        db_name = info_db$db_name, db_password = info_db$db_password, # db infos
```

```
        query.type = "data-classic", # data-classic query
        person.in = "OLR", # person to keep
        filter.on = "father-son", # filters on father AND son
        data.type = "relation", # data linked to relation between seed-lots
        variable = "poids_epis", # the variables to display
        project.in = "DEMO" # the project
)

vec_sl = unique(as.character(data_classic_bis$data$data$son))

data_classic_bis_network = get.data(
        db_user = info_db$db_user,
        db_host = info_db$db_host,
        db_name = info_db$db_name,
        db_password = info_db$db_password,
        query.type = "network",
        seed.lot.in = vec_sl,
        filter.on = "father-son",
        network.info = FALSE
        )

pn = get.ggplot(
        data_classic_bis,
        data_classic_bis_network,
        ggplot.type = "data-pie.on.network",
        vec_variables = "poids_epis---poids_epis",
        pie.size = .25,
        hide.labels.parts = "person",
        vertex.color = "person", organise.sl = TRUE
)

} else {
        # 1. Query SHiNeMaS ...
        # 2. Set up data set ...
        # |============================================================| 100%
        load("./data/data_classic_bis.RData")

        # 1. Query SHiNeMaS ...
        # Mixtures from replication have been deleted and replaced by reproductions.
        # 2. Create network matrix ...
        # 3. Link information to vertex and edges ...
        load("./data/data_classic_bis_network.RData")

        # 1. Query SHiNeMaS ...
        # 2. Create network matrix ...
        # 3. Link information to vertex and edges ...
        load("./data/pn.RData")
        }
```

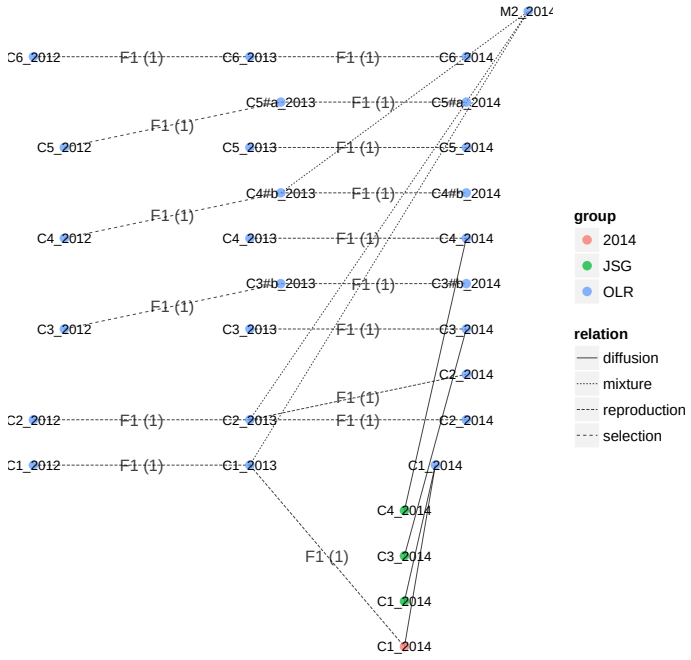
As we're working on one person, we set `hide.labels.parts = "person"`.
The result is divided into two plots: the empty network and the network with the pies.

```
names(pn$`data-pie.on.network`)
```

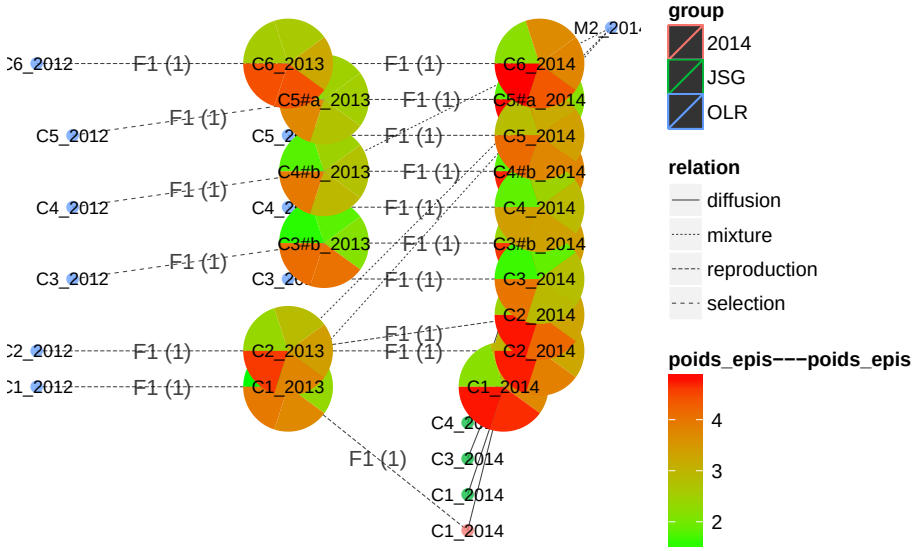
```
## [1] "poids_epis---poids_epis"
```

```
p1 = pn$`data-pie.on.network`$`poids_epis---poids_epis`$network
p2 = pn$`data-pie.on.network`$`poids_epis---poids_epis`$pie.on.network
```

p1


M2_2014
C6_2012
F1 (1)
C6_2013
F1 (1)
C6_2014
C5#a_2013
F1 (1)
C5#a_2014
F1 (1)
C5_2012
C5_2013
F1 (1)
C5_2014
C4#b_2013
F1 (1)
C4#b_2014
F1 (1)
C4_2012
C4_2013
F1 (1)
C4_2014
C3#b_2013
F1 (1)
C3#b_2014
F1 (1)
C3_2012
C3_2013
F1 (1)
C3_2014
C2_2014
F1 (1)
C2_2012
F1 (1)
C2_2013
F1 (1)
C2_2014
C1_2012
F1 (1)
C1_2013
C1_2014
C4_2014
F1 (1)
C3_2014
C1_2014
C1_2014
group
2014
JSG
OLR
relation
diffusion
mixture
reproduction
selection

p2


M2_201
C6_2012
F1 (1)
C6_2013
F1 (1)
C6_2014
C5#a_2013
F1 (1)
C5#a_2014
F1 (1)
C5_2012
C5_2
F1 (1)
C5_2014
C4#b_2013
F1 (1)
C4#b_2014
F1 (1)
C4_2012
C4_2
F1 (1)
C4_2014
C3#b_2013
F1 (1)
C3#b_2014
F1 (1)
C3_2012
C3_20
F1 (1)
C3_2014
C2_2014
F1 (1)
C2_2012
F1 (1)
C2_2013
F1 (1)
C2_2014
C1_2012
F1 (1)
C1_2013
C1_2014
C4_20
F1 (1)
C3_2014
C1_2014
C1_2014
group
2014
JSG
OLR
relation
diffusion
mixture
reproduction
selection
poids_epis−−−poids_epis
4
3
2

– Custom arguments
Arguments can be customized related to network plots (Table 1).

- pies on map

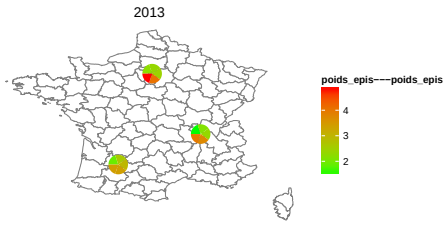
– Default arguments
`get.data` is called to get the coordinates data.

```
if(use.get.data){
        data_classic_ter = get.data(
                db_user = info_db$db_user, db_host = info_db$db_host, # db infos
                db_name = info_db$db_name, db_password = info_db$db_password, # db infos
                query.type = "data-classic", # data-classic query
                germplasm.in = "C1", # germplasm to keep
                person.in = c("OLR", "MLN", "JFB"), # person to keep
                filter.on = "father-son", # filters on father AND son
                data.type = "relation", # data linked to relation between seed-lots
                variable = "poids_epis", # the variables to display
                project.in = "DEMO" # the project
                )

        pm = get.ggplot(
                data_classic_ter,
                ggplot.type = "data-pie.on.map",
                vec_variables = "poids_epis---poids_epis"
                )
} else {
        load("./data/data_classic_ter.RData")
        load("./data/pm.RData")
        }

names(pm$`data-pie.on.map`)

## [1] "poids_epis---poids_epis"
```

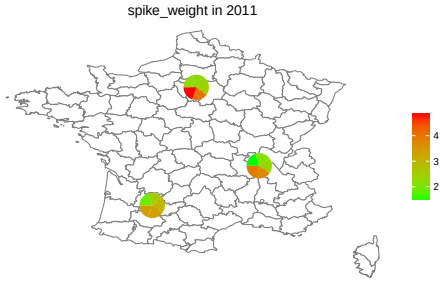
You can get the default ggplot (`p`) or customized it (`p_bis`):

```
p = pm$`data-pie.on.map`$`poids_epis---poids_epis`$`map-[2013]`

p_bis = p +
        ggtitle("spike_weight in 2011") + # change the title
        theme(legend.title=element_blank()) # delete the name of the legend
```

p



p_bis



– Custom arguments

Arguments can be customed related to map plots (Table 3).

For `data-pie.on.map`, it is also possible to choose `x.axis` in order to get a map for each `year` or each `germplasm`.

In the example, as `data_classic` is done on one germplasm (C1), `x.axis = year` has been chosen (default argument).

4.2.2 ggplots for `data-S`

```
default_data_S_ggplot = get.ggplot(
        data_S,
        vec_variables = plot_vec_variables,
        nb_parameters_per_plot_x.axis = 5
        )

## As ggplot.type is NULL, ggplot.Type is set to data-barplot, data-boxplot, data-interaction,
data-radar, data-biplot
## As ggplot.type is NULL and data come from "data-S" or "data-SR", ggplot.type is set to "data-barpl
"data-boxplot", "data-interaction".
## With "data-S" and "data-SR", in.col and x.axis are set automaticaly.

## [1] "A Virer quand les données seront propres dans get.data"
## [1] "A Virer quand les données seront propres dans get.data"
## [1] "A Virer quand les données seront propres dans get.data"
```

By default, the following plots are done with `x.axis` and `in.col` (you can not change it). Note that `NA` are put when there are no data for `bouquet` AND `vrac` (i.e. nothing is displayed).

```
names(default_data_S_ggplot)

## [1] "data-barplot"     "data-boxplot"     "data-interaction"
```

which correpond to ggplot.type arguments as explained in the previous section.

- For barplot:

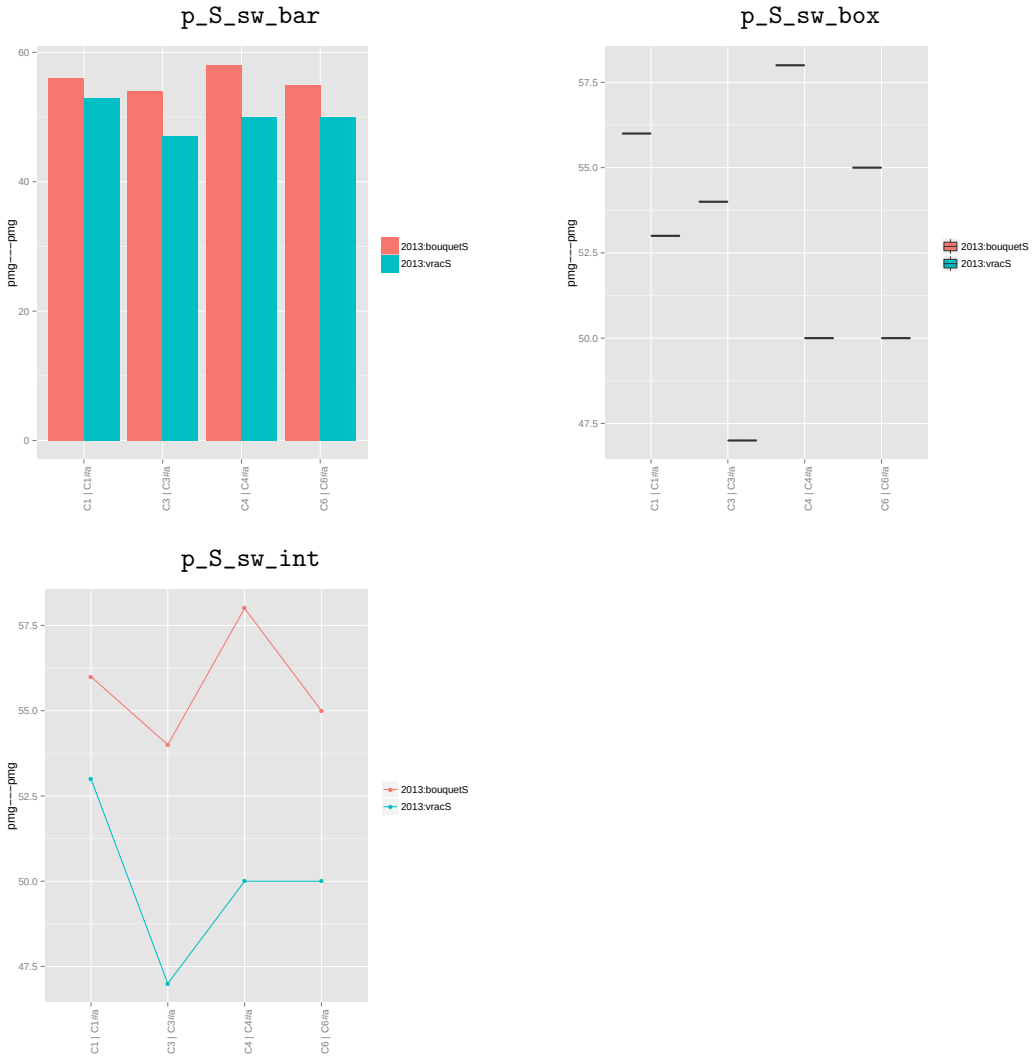
```
p_S_sw_bar_all = default_data_S_ggplot$`data-barplot`
p_S_sw_bar = p_S_sw_bar_all$`pmg---pmg`$`x.axis-1|in.col-1`
```

- For boxplot:

```
p_S_sw_box_all = default_data_S_ggplot$`data-boxplot`
p_S_sw_box = p_S_sw_box_all$`pmg---pmg`$`x.axis-1|in.col-1`
```

- For interaction plot:

```
p_S_sw_int_all = default_data_S_ggplot$`data-interaction`
p_S_sw_int = p_S_sw_int_all$`pmg---pmg`$`x.axis-1|in.col-1`
```



It is possible to customize the plots as presented in Table 4.

4.2.3 ggplots for `data-SR`

```
default_data_SR_ggplot = get.ggplot(
        data_SR,
        vec_variables = plot_vec_variables,
        nb_parameters_per_plot_x.axis = 5
        )
```

```
## As ggplot.type is NULL, ggplot.Type is set to data-barplot, data-boxplot, data-interaction,
data-radar, data-biplot
## As ggplot.type is NULL and data come from "data-S" or "data-SR", ggplot.type is set to "data-barpl
"data-boxplot", "data-interaction".
## With "data-S" and "data-SR", in.col and x.axis are set automaticaly.

## [1] "A Virer quand les données seront propres dans get.data"
## [1] "A Virer quand les données seront propres dans get.data"
## [1] "A Virer quand les données seront propres dans get.data"
```

By default, the following plots are done with `x.axis` and `in.col` (you can not change it). Note that `NA` are put when there are no data for `bouquet` AND `vrac` (i.e. nothing is displayed).

```
names(default_data_SR_ggplot)

## [1] "data-barplot"     "data-boxplot"     "data-interaction"
```

which correpond to ggplot.type arguments as explained in the previsou section.

- For barplot:

```
p_SR_sw_bar_all = default_data_SR_ggplot$`data-barplot`
p_SR_sw_bar = p_SR_sw_bar_all$`poids_epis---poids_epis`$`x.axis-1|in.col-1`
```

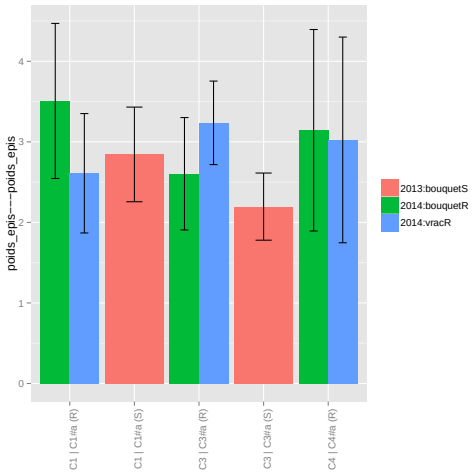
- For boxplot:

```
p_SR_sw_box_all = default_data_SR_ggplot$`data-boxplot`
p_SR_sw_box = p_SR_sw_box_all$`poids_epis---poids_epis`$`x.axis-1|in.col-1`
```

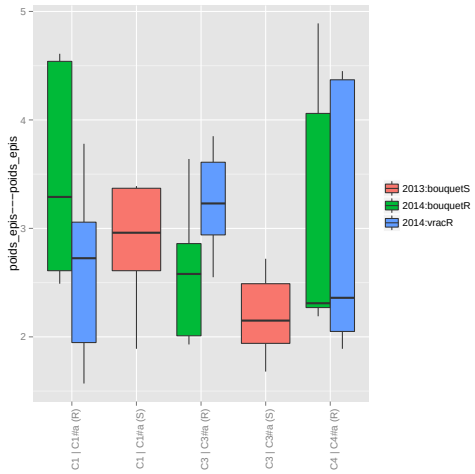
- For interaction plot:

```
p_SR_sw_int_all = default_data_SR_ggplot$`data-interaction`
p_SR_sw_int = p_SR_sw_int_all$`poids_epis---poids_epis`$`x.axis-1|in.col-1`
```

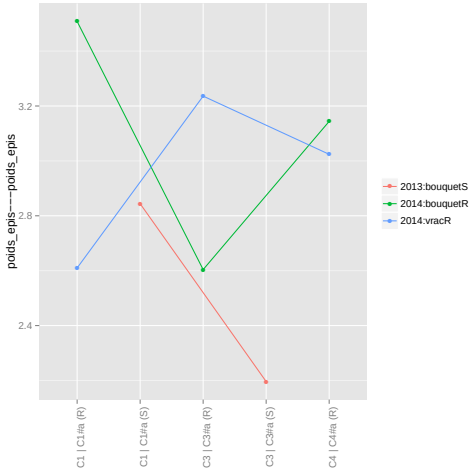
p_SR_sw_bar



p_SR_sw_box



p_SR_sw_int



It is possible to customize the plots as presented in Table 4.

4.3 Get the tables

From the data, to get the tables, use the function `get.table`.

First of all ,you need to choose

- on which data set you get the ggplots. This can be done with `correlated_group` having the following argument:
  - `NULL`, the default argument (`shinemas2R::get.data()$data$data`)
  - the name of a given `correlation_group` (`shinemas2R::get.data()$data$data.with.correlated.variables`
- if you want to merge germplasm name and selection name with the argument `merge_g_and_s` which is `TRUE` by default

Five types of tables can be displayed according to `table.type` argument :

`table.type`	description
`"raw"`	display raw data. Useful with text for example
`"mean"`	displayed for each variable columns with mean
`"mean.sd"`	displayed for each variable columns with mean and standard deviation
`"mean.sd.cv"`	displayed for each variable columns with mean, standard deviation and coefficient of variation
`"summary"`	display `"Min."`, `"1st Qu."`, `"Median"`, `"3rd Qu."`, `"Max."` of the data

The table always display information with seed-lots on the left side. For information on relations, you must choose if the seed-lots displayed in the table are the son or the father with the argument `table.on`. For information on seed-lots, there is no problem.

Then you should set `vec_variables` with the variables displayed in the table.

4.3.1 tables for `data-classic`

- Default arguments

```
tab_class = get.table(
        data_classic,
        table.type = "raw",
        vec_variables = plot_vec_variables
        )
```

the function returns a list with two elements:

```
names(tab_class)

## [1] "duplicated_infos"     "not_duplicated_infos"
```

  - `"duplicated_infos"`: lists of two elements with seed-lots involved and variable values

```
tab_class$duplicated_infos$`set-1`

## NULL
```

It is `NULL` as there is no duplicated information here.

An example with duplicated information:

```
if(use.get.data){
        data_classic_4 = get.data(
                db_user = info_db$db_user, db_host = info_db$db_host, # db infos
                db_name = info_db$db_name, db_password = info_db$db_password, # db infos
                query.type = "data-classic", # data-classic query
                person.in = "JFB", # person to keep
                filter.on = "father-son", # filters on father AND son
                data.type = "relation", # data linked to relation between seed-lots
                variable = "densite_semis", # the variables to display
                project.in = "DEMO" # the project
                )

} else {
        load("./data/data_classic_4.RData")
        }

tab_class_4 = get.table(
        data_classic_4,
        table.type = "raw",
        vec_variables = "densite_semis---densite_semis"
        )
```

The information is divided in two:

```
names(tab_class_4$duplicated_infos$`set-1`)

## [1] "duplicated_infos_seed-lots" "duplicated_infos_variables"
```

The seed-lots that are concerned. By default `col_to_display = c("person", "germplasm", "year", "block", "X", "Y")`, therefore, the information is under the form : `person-germplasm-year-block-`

```
tab_class_4$duplicated_infos$`set-1`$`duplicated_infos_seed-lots`

##
## 1 JFB-C1-2014-1-A-2, JFB-C1-2012-1-A-1, JFB-C1-2013-1-A-1, JFB-C1-2014-1-A-1, JFB-C1-2014
```

And the data linked to these seed lots:

```
tab_class_4$duplicated_infos$`set-1`$`duplicated_infos_variables`

##   densite_semis---densite_semis
## 1                           200
```

– `"not_duplicated_infos"`: a list with the table of non duplicated information

```
dim(tab_class$not_duplicated_infos$`set-1`)

## [1] 137   9
```

Note that the threshold up to which the information are duplicated or not is tuned with the argument `nb_duplicated_rows` (see Custom arguments).

Instead of raw information, some statistics may be useful, for example,

```
tab_class = get.table(
        data_classic,
        table.type = "mean.sd.cv",
        vec_variables = plot_vec_variables
        )
names(tab_class$not_duplicated_infos)

## [1] "set-1"

colnames(tab_class$not_duplicated_infos$`set-1`)

##  [1] "person"                        "germplasm"
##  [3] "year"                          "block"
##  [5] "X"                             "Y"
##  [7] "hauteur---hauteur_tige mean"   "hauteur---hauteur_tige sd"
##  [9] "hauteur---hauteur_tige cv"     "pmg---pmg mean"
## [11] "pmg---pmg sd"                  "pmg---pmg cv"
## [13] "poids_epis---poids_epis mean" "poids_epis---poids_epis sd"
## [15] "poids_epis---poids_epis cv"

dim(tab_class$not_duplicated_infos$`set-1`)

## [1] 24 15
```

- Custom arguments

  The following arguments can be customized:

argument	default value	description
`nb_row`	`NULL`	the number of rows in the table
`nb_col`	`NULL`	the number of columns for variables in the table. `col_to_display` remains fixed.
`nb_duplicated_rows`	`NULL`	minimum number of duplicated rows for each variable of a table up to which the information is put in only one row.
`col_to_display`	`c("person", "germplasm", "year", "block", "X", "Y")`	columns to display in the table. It can be a vector with `"person"`, `"year"`, `"germplasm"`, `"block"`, `"X"` and `"Y"`. If `NULL`, none of these columns is displayed. The variables follow these columns. For `"data-S"` and `"data-SR"` type, the column `"expe"` and `"sl_statut"` are added by default.
`invert_row_col`	`FALSE`	if `TRUE`, invert row and col in the table. This is possible only for `col_to_display = NULL`.

Table 5: Possible arguments to customize `get.table`.

For example,

```
tab_class = get.table(
        data_classic,
        table.type = "mean.sd.cv",
        vec_variables = plot_vec_variables,
        nb_row = 10
        )
names(tab_class$not_duplicated_infos)

## [1] "set-1" "set-2" "set-3"
```

The information is split into several tables with at least 10 rows.

```
dim(tab_class$not_duplicated_infos$`set-1`)

## [1] 10 15
```

4.3.2 tables for `data-S` and `data-SR`

Tables are the same than for `classic-data`, there is just the column `"expe"` and `"sl_statut"` added by default.

For example,

```
tab_S_msc = get.table(
        data_S,
        table.type = "mean.sd.cv",
        vec_variables = plot_vec_variables,
        nb_col = 3 # Otherwise, the table is too large
        )

colnames(tab_S_msc$not_duplicated_infos$`set-1`)

##  [1] "person"                    "germplasm"
##  [3] "year"                      "block"
##  [5] "X"                         "Y"
```

```
##  [7] "expe"                          "sl_statut"
##  [9] "hauteur---hauteur_tige mean" "hauteur---hauteur_tige sd"
## [11] "hauteur---hauteur_tige cv"
```

It is possible to customize the tables as presented in Table 5.

4.4 Format the data for existing R packages

It is often useful to go beyond descriptive analysis and apply statistical tests to your data. To do so, you need to use existing R packages to perform such analysis.

You need to format your data in order to use these packages. This is done with `format.data` which format your data for the following packages: `PPBstats`.

For example, if you want to use the `PPBstats` package:

```
data_for_PPBstats = format.data(
        data_classic,
        format = "PPBstats"
)

head(data_for_PPBstats)
```

```
##    year location germplasm block X Y hauteur---hauteur_tige
## 1  2013      JFB        C1     1 A 1                  171.1
## 5  2013      JFB        C1     1 A 1                  128.2
## 9  2013      JFB        C1     1 A 1                  121.5
## 13 2013      JFB        C1     1 A 1                  125.0
## 17 2013      JFB        C1     1 A 1                  133.1
## 21 2013      JFB        C1     1 A 1                   <NA>
##    hauteur---hauteur_tige$date pmg---pmg pmg---pmg-2
## 1                   2013-07-04      <NA>        <NA>
## 5                   2013-07-04      <NA>        <NA>
## 9                   2013-07-04      <NA>        <NA>
## 13                  2013-07-04      <NA>        <NA>
## 17                  2013-07-04      <NA>        <NA>
## 21                        <NA>        56        <NA>
##    pmg---pmg-2$date pmg---pmg$date poids_epis---poids_epis
## 1              <NA>           <NA>                    2.61
## 5              <NA>           <NA>                    2.96
## 9              <NA>           <NA>                    3.39
## 13             <NA>           <NA>                    1.89
## 17             <NA>           <NA>                    3.37
## 21             <NA>     2013-07-15                    <NA>
##    poids_epis---poids_epis$date
## 1                    2013-07-23
## 5                    2013-07-23
## 9                    2013-07-23
## 13                   2013-07-23
## 17                   2013-07-23
## 21                         <NA>
```

Or to use data coming from `data-S` type:

```
data_version_for_PPBstats = format.data(
        data_S,
        format = "PPBstats"
)
```

```
## With data coming from "data-S" or "data-SR", fuse_g_and_s is set to TRUE by default.

head(data_version_for_PPBstats)

##    year location germplasm     group    version
## 1  2013      JFB        C1 C1 | C1#a 2013:vracS
## 2  2013      JFB        C3 C3 | C3#a 2013:vracS
## 3  2013      JFB        C4 C4 | C4#a 2013:vracS
## 4  2013      JFB        C6 C6 | C6#a 2013:vracS
## 8  2014      JFB        C1 C1 | C1#c 2014:vracS
## 12 2014      JFB        C1 C1 | C1#c 2014:vracS
```

5 Other information in SHiNeMaS

For the following `query.type`, you can set filters to the query with the following argument :

- `filter.in` to choose nothing except `filter.in`
- `filter.out` to choose everything except `filter.out`

Values of filter depends on the `query.type`.

5.1 `query.type = "cross"`

values of filters can be `germplasm`, `germplasm.type`, `year`, `person` or `project`.

5.2 `query.type = "method"`

values of filters can be `variable`.

5.3 `query.type = "person-info"`

values of filters can be `person`.

5.4 `query.type = "grandfather"`

values of filters can be `germplasm`, `germplasm.type`, `year`, `person`, `project`, `seed-lots` or `reproduction.type`.

6 pdf compilation with LaTeX

`get.pdf` aggregates texts and outputs from `get.ggplot` and `get.table`. This is particularly useful in participatory research to give feedback to stackholders such as farmers.

`get.pdf` creates a `.tex` file that is compiled into a `.pdf` file. You need to install LaTeX and the following packages : `longtable`, `lscape`, `graphicx`, `pdfpages`, `float`, `hyperref` and `fancyhdr`. To download LaTeX, go to latex-project.org/ftp.html.

6.1 Philosophy of `get.pdf`

The idea of `get.pdf` is to create a list (`LaTeX_body` argument) with several elements that can be:

- `"titlepage"` : a list containing the following elements :
  - `"title"` : the title of the document. Nothing by default.
  - `"authors"` : the authors of the document. Nothing by default.
  - `"email"` : the corresponding email. Nothing by default.
  - `"date"` : the date of the document. Nothing by default.
- `"tableofcontents"` : if `TRUE`, a table of content is displayed.
- `"chapter"` : name of a chapter
- `"section"` : name of a section
- `"subsection"` : name of a subsection
- `"subsubsection"` : name of a subsubsection
- `"text"` : text to add
- `"includepdf"` : path of a .pdf to include. It includes entire document.
- `"includeimage"`: a list containing the following elements:
  - `"content"`: path of image to include. It can be a pdf, png, jpeg ...
  - `"caption"`: caption of the image
  - `"width"` : width of the picture in textwidth unit. 1 means the same width as the text. It is 1 by default.
- `"input"` : path of a .tex to insert
- `"table"` : a list containing the following elements :
  - `"content"` : output from `get.table`
  - `"caption"` : caption of the table
  - `"landscape"` : If `TRUE`, the table is in landscape. `FALSE` by default.
  - `"display.rownames"` : If `TRUE`, display the row names of the table. `FALSE` by default.
- `"figure"` : a list containing the following elements :
  - `"content"` : output from `get.ggplot`
  - `"caption"` : caption of the figure
  - `"layout"` : the layout of the plots. It is a matrix under the form `layout = matrix(c(1:4), ncol = 2, nrow = 2)`. It is `layout = matrix(1, ncol = 1, nrow= 1)` by default.
  - `"width"` : width of the figure in textwidth unit. 1 means as width as the text. It is 1 by default.
  - `"landscape"` : If `TRUE`, the figure is in landscape. `FALSE` by default.

The elements of this list will be in the same order in the final pdf.

You do not have to know LaTeX. `get.pdf` uses it for you. Nevertheless, if you use LaTeX you'll be able to customize the `.pdf` document by creating the tex structure of the document (`LaTeX_head` argument). It is also possible to add specific tex files (`input` element of `LaTeX_body` argument). If you use LaTeX macro, do not forget to use the escape mode (Table 6),

macro	description
\\textbf{blod}	**blod**
\\textit{italique}	*italique*
\\underline{underline}	underline
\\texttt{type writer}	`type writer`

Table 6: Basic maros in LaTeX to use in `get.pdf` with escape mode.

Tutorials on LaTeX can be found here: en.wikibooks.org/wiki/LaTeX.

See appendix C for some examples.

6.2 Examples

6.2.1 First Information

First, set the directory where you want the pdf to be stored:

```
dir = "/home/pierre/"
```

Then, choose a name for your pdf (as well as a folder that contain all the raw information to create the pdf)

```
form.name = "my_first_pdf"
```

6.2.2 LaTeX body

Then, create the list with the different elements. The names of plots and tables are coming from sections 3 and 4.

```
OUT = NULL

###################################
# 1. Title page
###################################
out = list(
        "titlepage" = list(
                "title" = "Example of \\texttt{shinemas2R::get.pdf}",
                "authors" = "P. Riviere",
                "date" = "\\today",
                "email" = "pierre@semencespaysannes.org")
        ); OUT = c(OUT, out)


###################################
# 2. Table of contents
###################################
out = list("tableofcontents" = TRUE); OUT = c(OUT, out)
```

```
##################################
# 3. Network relations between seed-lots
##################################
out = list(
        "chapter" = "Network relations between seed-lots"
        ); OUT = c(OUT, out)

out = list(
        "section" = "Network for Rouge du Roc"
        ); OUT = c(OUT, out)

out = list(
        "text" = "The following figure display the network for Rouge du Roc"
        ); OUT = c(OUT, out)

out = list(
        "figure" = list(
                "caption" = "Network for C1",
                "content" = p_net_C1,
                "width" = 1)
        ); OUT = c(OUT, out)

out = list(
        "section" = "Seed-lots harvested for Rouge du Roc after a reproduction"
        ); OUT = c(OUT, out)

out = list(
        "figure" = list(
                "caption" = "Seed-lots harvested for each person by year for Rouge du Roc",
                "content" = p3_slh,
                "width" = 1)
        ); OUT = c(OUT, out)

out = list(
        "figure" = list(
                "caption" = "Repartition of seed-lots harvested
since the beginning of the project.",
                "content" = m5_slh,
                "width" = 1)
        ); OUT = c(OUT, out)


##################################
# 4. Data linked to the relations between seed-lots
##################################
out = list(
        "chapter" = "Data linked to the relations between seed-lots"
        ); OUT = c(OUT, out)


out = list(
        "section" = "Data related to sowing practices"
        ); OUT = c(OUT, out)

out = list(
        "table" = list(
```

```
                "caption" = "Sowing practices for seed-lots at RAB",
                "content" = tab_class_4,
                "landscape" = FALSE
                )
        ); OUT = c(OUT, out)


out = list(
        "section" = "Data on selection differentia"
        ); OUT = c(OUT, out)

out = list(
        "figure" = list(
                "caption" = paste("Selection differentia for spike weight on RAB farm."),
                "content" = p_S_sw_bar,
                "width" = 1)
        ); OUT = c(OUT, out)

# Here, all the plots in the list are displayed:
# Set "layout" argument is useful as there are more than one plot
out = list(
        "figure" = list(
                "caption" = paste("Selection differentia for spike weight on RAB farm."),
                "content" = p_S_sw_box_all,
                "layout" = matrix(c(1,2), ncol = 1),
                "width" = 1)
        ); OUT = c(OUT, out)


out = list(
        "table" = list(
                "caption" = "Mean, standard deviation and coefficient of variation
for differentia to selection",
                "content" = tab_S_msc,
                "landscape" = TRUE
                )
        ); OUT = c(OUT, out)

out = list(
        "section" = "Data on response to selection"
        ); OUT = c(OUT, out)

out = list(
        "figure" = list(
                "caption" = paste("Selection differentia for spike weight on RAB farm."),
                "content" = p_SR_sw_bar,
                "width" = .5)
        ); OUT = c(OUT, out)
```

6.2.3 Compile the pdf

It is possible to custom the color of the rows of tables with `color1` (color of the head of the table) or `color2` (color of the row, the color of the rows is an alternation of white and `color2`). `compile.tex = TRUE` compiles the pdf.

```
get.pdf(
        dir = dir,
        form.name = form.name,
        LaTeX_head = NULL,
        LaTeX_body = OUT,
        compile.tex = TRUE
        )

## 1.  Get LaTeX head ...
## 2.  Get LaTeX body ...
## 3.  Compilation of .tex ...
## /home/pierre/my_first_pdf.pdf has been compiled.
```

7 Common use rules

The use of a tool such as a data base rises questions in collaborative research. The Réseau Semences Paysannes is a network that brings together a great diversity of collectives. Some collectives are known as Community Seed Houses (CSH) (Réseau Semences Paysannes, 2014).

Within RSP, biodiversity is seen as a common. Therefore, common use rules are needed to manage this common, especilly on data management.

Common use rules related to data management within CSHs can be seen at two levels :

1. Internal organization rules of the group and their relations with juridic, social, agronomic and economic environments:
    - Up to which level dematerialze the information ?
    - Which data to store ?
    - Which access to data ?
2. Biopiracy risks:
    - Which data to store ?
    - Which access to data ?
    - Link of CSHs with research : which protections of CSHs'work, ressources and knowledge ?

More information on this topic can be found in Réseau Semences Paysannes (2015).

To cite shinemas2R

To cite this package and or this vignette:

```
citation("shinemas2R")

##
## To cite package 'shinemas2R' in publications use:
##
##   Pierre Riviere (2016). shinemas2R: An R package to
##   visualize outputs from the data base Seed History and
##   Network Management System (SHiNeMaS). R package version
##   0.11. http://github.com/priviere/shinemas2R
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {shinemas2R: An R package to visualize outputs from the data base Seed History and Net
##     author = {Pierre Riviere},
##     year = {2016},
##     note = {R package version 0.11},
##     url = {http://github.com/priviere/shinemas2R},
##   }
```

Aknowledgement

This worked has been first funded by the European Community's Seventh Framework Programme (FP7/9 2007–2013) under the grant agreement n245058-Solibam (Strategies for Organic and Low-input Integrated Breeding and Management). It has been completed by funding from European Union's Horizon 2020 research and innovation programme under grant agreement No 633571 (DIVERSIFOOD project) and Fondation de France.

Thanks to Hadley Wickham for his web site r-pkgs.had.co.nz/ that help us a lot in the creation of this package.

Thanks to Gaelle Van Frank and Alice Beaugrand for their comments and remarks improving this vignette and the code of the package.

References

Y. De Oliveira, L. Burlot, M. Lefebvre, D. Madi, P. Rivière, and M. Thomas. SHiNeMaS : a database dedicated to seed lots genealogy, phenotyping and cultural practices. Poster at Jobim, Clermont-Ferrand, 2015.

M. Nei. Analysis of gene diversity in subdivided populations. *Proceedings of the National Academy of Sciences*, 70:3321–3323, 1973.

Réseau Semences Paysannes. *Les maisons des semences paysannes : Regards sur la gestion collective de la biodiversité cultivée en France.* Réseau Semences Paysannes, 2014.

Réseau Semences Paysannes. *Éléments de réflexion sur la gestion des données dans les Maisons des Semences Paysannes.* Réseau Semences Paysannes, 2015.

P. Rivière. *Méthodologie de la sélection décentralisée et participative: un exemple sur le blé tendre.* PhD thesis, Université Paris Sud, 2014.

M. Thomas. *Gestion dynamique à la ferme de l'agrobiodiversité : relation entre la structure des populations de blé tendre et les pratiques humaines.* PhD thesis, Université Paris Diderot, 2011.

A Install SHiNeMaS on localhost

A.1 Install PostgreSQL and SHiNeMaS

With Linux, firt install `postgresql` and create your account and password.

```
pierre@RSP:~$ sudo apt-get install postgresql
pierre@RSP:~$ sudo -i -u postgres
postgres@RSP:~$ psql
postgres=# ALTER USER postgres with password 'postgres';
postgres=# create user pierre;
postgres=# ALTER ROLE pierre WITH CREATEDB;
postgres=# ALTER USER pierre SUPERUSER;
postgres=# alter user pierre with encrypted password 'pierre';
```

Then, create the empty data base `shinemas_tuto`:

```
postgres=# create database shinemas_tuto owner pierre;
postgres=# GRANT ALL PRIVILEGES ON DATABASE shinemas_tuto TO pierre;
```

To exit `postgres=#`, press `ctrl + D`.
Once you've done this, download `shinemas_tuto.sql`. You can find it in the package:

```
system.file("extdata", "shinemas_tuto.sql", package = "shinemas2R")

## [1] "/home/pierre/Documents/geek/R-stats/Rdev/R_package_shinemas2R/shinemas2R/inst/extdata/shinema
```

Put it in your `/home/pierre`, and install it on postgresql.
Note that `shinemas_tuto.sql` must be accesible in writing and reading.

```
pierre@RSP:~$ psql -h localhost -d shinemas_tuto -U pierre -f /home/pierre/shinemas_tuto.sql
```

You can look at the data base you have:

```
sudo -i -u postgres
postgres@RSP:~$ psql -l
```

You can delete the database with

```
sudo -i -u postgres
postgres@RSP:~$ dropdb shinemas_tuto
```

More information on posgresql can be found here: http://help.ubuntu.com/community/PostgreSQL

A.2 Set up the information to connect to SHiNeMaS with `get.data`

```
info_db = list(
        db_user = "pierre",
        db_host = "127.0.0.1", # localhost
        db_name = "shinemas_tuto",
        db_password = "pierre"
        )
```

`db_info` is use in `get.data` function.

B Therory regarding selection differential, response to selection and heritability

to do !!!

C `get.pdf` examples

C.1 `LaTeX_head` examples

to do !

C.2 `.tex` examples for `input`

to do !

D Examples of `is.get.data.output`

In this appendix, you can find empty data-set that will fit in `is.get.data.output`.

D.1 `shinemas2R.object == "network"`

```
network = NULL

network.query = data.frame(
        son_species = factor(c("a", "b", "c")),
        son_project = factor(c("a", "b", "c")),
        son = factor(c("a", "b", "c")),
        son_germplasm = factor(c("a", "b", "c")),
        son_person = factor(c("a", "b", "c")),
        son_year = factor(c("a", "b", "c")),
        son_germplasm_type = factor(c("a", "b", "c")),
        son_alt = as.numeric(c(0, 0, 0)),
        son_long = as.numeric(c(0, 0, 0)),
        son_lat = as.numeric(c(0, 0, 0)),
        son_total_generation_nb = as.numeric(c(0, 0, 0)),
        son_local_generation_nb = as.numeric(c(0, 0, 0)),
        son_generation_confidence = c("a", "b", "c"),
        son_comments = c("a", "b", "c"),

        father_species = factor(c("a", "b", "c")),
        father_project = factor(c("a", "b", "c")),
        father = factor(c("a", "b", "c")),
        father_germplasm = factor(c("a", "b", "c")),
        father_person = factor(c("a", "b", "c")),
        father_year = factor(c("a", "b", "c")),
        father_germplasm_type = factor(c("a", "b", "c")),
        father_alt = as.numeric(c(0, 0, 0)),
        father_long = as.numeric(c(0, 0, 0)),
        father_lat = as.numeric(c(0, 0, 0)),
        father_total_generation_nb = as.numeric(c(0, 0, 0)),
        father_local_generation_nb = as.numeric(c(0, 0, 0)),
        father_generation_confidence = c("a", "b", "c"),
        father_comments = c("a", "b", "c"),

        reproduction_id = c("a", "b", "c"),
        reproduction_method_name = factor(c("a", "b", "c")),
        is_male = factor(c("a", "b", "c")),
        block = factor(c("a", "b", "c")),

        selection_id = c("a", "b", "c"),
        selection_person = factor(c("a", "b", "c")),
        mixture_id = c("a", "b", "c"),
        diffusion_id = c("a", "b", "c"),
        relation_year_start = factor(c("a", "b", "c")),
        relation_year_end = factor(c("a", "b", "c"))
)

network.query$son_generation_confidence = as.character(network.query$son_generation_confidence)
network.query$son_comments = as.character(network.query$son_comments)
network.query$father_generation_confidence = as.character(network.query$father_generation_confidence)
network.query$father_comments = as.character(network.query$father_comments)
```

```
network.query$reproduction_id = as.character(network.query$reproduction_id)
network.query$selection_id = as.character(network.query$selection_id)
network.query$mixture_id = as.character(network.query$mixture_id)
network.query$diffusion_id = as.character(network.query$diffusion_id)


network.info = data.frame(
        sl = factor(c("a", "b", "c")),
        alt = as.numeric(c(0, 0, 0)),
        long = as.numeric(c(0, 0, 0)),
        lat = as.numeric(c(0, 0, 0)),
        diffusion = factor(c("give", "receive", "give-receive")),
        id.diff = factor(c(1, 2, 3)),
        reproduction = factor(c("sow", "harvest", "harvest-sow")),
        mixture = factor(c("mixture", NA, NA)),
        selection = factor(c("selection", NA, NA)),
        cross.info = factor(c("a", "b", "c")),
        germplasm = factor(c("a", "b", "c")),
        person = factor(c("a", "b", "c")),
        year = factor(c("a", "b", "c"))
)

Mdist = NULL

n = list("network" = network, "network.query" = network.query, "network.info" = network.info, "Mdist"
n = is.get.data.output(n)

## The data has been sucessfully updated with shinemas2R.object = "network".
```

D.2 shinemas2R.object == "data..."

In all data, you need a methods format:

```
methods = data.frame(
        method_name = c("a", "b"),
        variable_name = c("a", "b"),
        method_description = c("a", "b"),
        unit = c("a", "b"),
        varmeth = c("a", "b")
)

methods$method_name = as.character(methods$method_name)
methods$variable_name = as.character(methods$variable_name)
methods$method_description = as.character(methods$method_description)
methods$unit = as.character(methods$unit)
methods$varmeth = as.character(methods$varmeth)

colnames(methods)[5] = "variable---methods"
```

D.2.1 shinemas2R.object == "data-classic-seed-lots"

```
data = data.frame(
        species = factor(c("a", "b", "c")),
        project = factor(c("a", "b", "c")),
```

```
        sl = factor(c("a", "b", "c")),
        germplasm = factor(c("a", "b", "c")),
        germplasm_type = factor(c("a", "b", "c")),
        person = factor(c("a", "b", "c")),
        year = factor(c("a", "b", "c")),
        lat = as.numeric(c(0, 0, 0)),
        long = as.numeric(c(0, 0, 0)),
        alt = as.numeric(c(0, 0, 0)),
        total_generation_nb = as.numeric(c(0, 0, 0)),
        local_generation_nb = as.numeric(c(0, 0, 0)),
        generation_confidence = as.numeric(c(0, 0, 0)),
        sl_comments = factor(c("a", "b", "c"))
        )

data.with.correlated.variables = list(data, data)

out = list("data" = data, "data.with.correlated.variables" = data.with.correlated.variables, "methods
out = is.get.data.output(out, shinemas2R.object = "data-classic-seed-lots")

## The data has been sucessfully updated with shinemas2R.object = "data-classic-seed-lots".
```

D.2.2 shinemas2R.object == "data-classic-relation"

```
data = data.frame(
        son_species = factor(c("a", "b", "c")),
        son_project = factor(c("a", "b", "c")),
        son = factor(c("a", "b", "c")),
        son_ind = factor(c("a", "b", "c")),
        son_year = factor(c("a", "b", "c")),
        son_germplasm = factor(c("a", "b", "c")),
        son_germplasm_type = factor(c("a", "b", "c")),
        son_person = factor(c("a", "b", "c")),
        son_alt = as.numeric(c(0, 0, 0)),
        son_long = as.numeric(c(0, 0, 0)),
        son_lat = as.numeric(c(0, 0, 0)),
        son_total_generation_nb = as.numeric(c(0, 0, 0)),
        son_local_generation_nb = as.numeric(c(0, 0, 0)),
        son_generation_confidence = as.numeric(c(0, 0, 0)),
        son_comments = factor(c("a", "b", "c")),

        father_species = factor(c("a", "b", "c")),
        father_project = factor(c("a", "b", "c")),
        father = factor(c("a", "b", "c")),
        father_year = factor(c("a", "b", "c")),
        father_germplasm = factor(c("a", "b", "c")),
        father_germplasm_type = factor(c("a", "b", "c")),
        father_person = factor(c("a", "b", "c")),
        father_alt = as.numeric(c(0, 0, 0)),
        father_long = as.numeric(c(0, 0, 0)),
        father_lat = as.numeric(c(0, 0, 0)),
        father_total_generation_nb = as.numeric(c(0, 0, 0)),
        father_local_generation_nb = as.numeric(c(0, 0, 0)),
        father_generation_confidence = as.numeric(c(0, 0, 0)),
        father_comments = factor(c("a", "b", "c")),
```

```
        reproduction_id = factor(c("a", "b", "c")),
        reproduction_method_name = factor(c("a", "b", "c")),
        selection_id = factor(c("a", "b", "c")),
        selection_person = factor(c("a", "b", "c")),
        mixture_id = factor(c("a", "b", "c")),
        diffusion_id = factor(c("a", "b", "c")),
        relation_year = factor(c("a", "b", "c")),
        X = factor(c("a", "b", "c")),
        Y = factor(c("a", "b", "c")),
        block = factor(c("a", "b", "c"))
        )

data.with.correlated.variables = list(data, data)

out = list("data" = data, "data.with.correlated.variables" = data.with.correlated.variables, "methods
out = is.get.data.output(out, shinemas2R.object = "data-classic-relation")

## The data has been sucessfully updated with shinemas2R.object = "data-classic-relation".
```

D.2.3 shinemas2R.object == "data-S-seed-lots" & shinemas2R.object == "data-SR-seed-lots"

```
data = data.frame(
        expe = factor(c("a", "b", "c")),
        sl_statut = factor(c("a", "b", "c")),
        expe_name = factor(c("a", "b", "c")),
        expe_name_2 = factor(c("a", "b", "c")),
        species = factor(c("a", "b", "c")),
        project = factor(c("a", "b", "c")),
        sl = factor(c("a", "b", "c")),
        germplasm = factor(c("a", "b", "c")),
        germplasm_type = factor(c("a", "b", "c")),
        person = factor(c("a", "b", "c")),
        year = factor(c("a", "b", "c")),
        lat = as.numeric(c(0, 0, 0)),
        long = as.numeric(c(0, 0, 0)),
        alt = as.numeric(c(0, 0, 0)),
        total_generation_nb = as.numeric(c(0, 0, 0)),
        local_generation_nb = as.numeric(c(0, 0, 0)),
        generation_confidence = as.numeric(c(0, 0, 0)),
        sl_comments = factor(c("a", "b", "c"))
        )

data.with.correlated.variables = list(data, data)

out = list("data" = data, "data.with.correlated.variables" = data.with.correlated.variables, "methods
out_S = is.get.data.output(out, shinemas2R.object = "data-S-seed-lots")

## The data has been sucessfully updated with shinemas2R.object = "data-S-seed-lots".

out_SR = is.get.data.output(out, shinemas2R.object = "data-SR-seed-lots")

## The data has been sucessfully updated with shinemas2R.object = "data-SR-seed-lots".
```

D.2.4 shinemas2R.object == "data-S-relation" & shinemas2R.object == "data-SR-relation"

```
data = data.frame(
        expe = factor(c("a", "b", "c")),
        sl_statut = factor(c("a", "b", "c")),
        expe_name = factor(c("a", "b", "c")),
        expe_name_2 = factor(c("a", "b", "c")),

        son_species = factor(c("a", "b", "c")),
        son_project = factor(c("a", "b", "c")),
        son = factor(c("a", "b", "c")),
        son_ind = factor(c("a", "b", "c")),
        son_year = factor(c("a", "b", "c")),
        son_germplasm = factor(c("a", "b", "c")),
        son_germplasm_type = factor(c("a", "b", "c")),
        son_person = factor(c("a", "b", "c")),
        son_alt = as.numeric(c(0, 0, 0)),
        son_long = as.numeric(c(0, 0, 0)),
        son_lat = as.numeric(c(0, 0, 0)),
        son_total_generation_nb = as.numeric(c(0, 0, 0)),
        son_local_generation_nb = as.numeric(c(0, 0, 0)),
        son_generation_confidence = as.numeric(c(0, 0, 0)),
        son_comments = factor(c("a", "b", "c")),

        father_species = factor(c("a", "b", "c")),
        father_project = factor(c("a", "b", "c")),
        father = factor(c("a", "b", "c")),
        father_year = factor(c("a", "b", "c")),
        father_germplasm = factor(c("a", "b", "c")),
        father_germplasm_type = factor(c("a", "b", "c")),
        father_person = factor(c("a", "b", "c")),
        father_alt = as.numeric(c(0, 0, 0)),
        father_long = as.numeric(c(0, 0, 0)),
        father_lat = as.numeric(c(0, 0, 0)),
        father_total_generation_nb = as.numeric(c(0, 0, 0)),
        father_local_generation_nb = as.numeric(c(0, 0, 0)),
        father_generation_confidence = as.numeric(c(0, 0, 0)),
        father_comments = factor(c("a", "b", "c")),

        reproduction_id = factor(c("a", "b", "c")),
        reproduction_method_name = factor(c("a", "b", "c")),
        selection_id = factor(c("a", "b", "c")),
        selection_person = factor(c("a", "b", "c")),
        mixture_id = factor(c("a", "b", "c")),
        diffusion_id = factor(c("a", "b", "c")),
        relation_year = factor(c("a", "b", "c")),
        X = factor(c("a", "b", "c")),
        Y = factor(c("a", "b", "c")),
        block = factor(c("a", "b", "c"))
        )

data.with.correlated.variables = list(data, data)

out = list("data" = data, "data.with.correlated.variables" = data.with.correlated.variables, "methods
out_S = is.get.data.output(out, shinemas2R.object = "data-S-relation")

## The data has been sucessfully updated with shinemas2R.object = "data-S-relation".

out_SR = is.get.data.output(out, shinemas2R.object = "data-SR-relation")
```

```
## The data has been sucessfully updated with shinemas2R.object = "data-SR-relation".
```