SNPHeatMap

Generated by Doxygen 1.9.1

Chapter 1

Snp HeatMap

The goal of this project is to create a number of chart related to snp (simple nucleotide polymorphism) analysis

Those charts are :

- `Quantitative chart` (-q) : Show the number of gene (y) per number of snp (x)
- `Cumulative chart` (-c) : Show the number of gene (y) that have at least n snp (x)
- `Cumulative chart` (-u) : Cumulative bar chart but it's a heatmap
- `Global Heatmap` (-g) : concatenation of all Cumulative chart

1.1 Installation :

- Download
  1. Download Repository ("<> Code" button top right --> "Download zip")
  2. Unzip the Downloaded files
- Using linux terminal
  1. `git clone https://github.com/F-Marchal/SnpHeatMap.git`
  2. `cd SnpHeatMap`

see `Quick usage` and `Quick usage` in order to run main.py

1.2 File format :

Files used by this script are expected to have the following pattern :

Gene name / id	Other	Snp counter
Gene1	Other info1	x
Gene2	Other info2	y

- All files have to have the same headers
- When a gene is present multiple times, only the last occurrence is used
- Snp counter should be integer greater or equal to 0. No dot ('.') or comma (',') are allowed in this cell.

1.2.0.0.1 Path to your files

By default, the program assumes that all your files are in the `Data/` folder.

Path toward your files can be give using two methods :

- Using folder path
  - All files inside the folder are used.
  - Names in the graph are file's names
  - Files that start with "." are ignored.
  - Is the default option
- Using a json's path
  - This file should have the following format `{path_to_a_file: common_name}`. You can open `test/TargetedFiles.json` if you need an example
  - Names in the graph are `common_name`
  - If multiple path have the same `common_name` they will be considered as the same file.

All files represent a species or a group of related individuals.

Can also be provided using parameters see `path`

1.3 Usage :

Please read `File format` before using main.py and `Path to your files.`

1.3.1 Quick usage :

- Put your files inside the `data` folder
- Open a terminal in this folder or use `cd` to change terminal's current directory
- In Linux :
  - run `python3 main.py [Gene name Column] [Snp column]`
- If you use windows :
  - run `python3 main.py [Gene name Column] [Snp column]`

1.3.2 Complete usage :

- linux : `python3 main.py [Gene name Column] [Snp column] [Path to your files] [Options]`
- Windows : `python3 main.py [Gene name Column] [Snp column] [Path to your files] [Options]` This command will make a "global heatmap" for all your files in data

1.3.2.1 Arguments

- **Warning:** Arguments should always be before the parameters

1.3.2.1.1 Gene name Column (First argument) Name of the column that contain gene's names / ids in files. e.g. "Gene name / id" in `File format`

Can also be provided using parameters see `name_column`

1.3.2.1.2 Snp column (Second argument) Name of the column that contain snp in files. e.g. "Snp counter" in `File format`

Can also be provided using parameters see `snp_column`

1.3.2.1.3 File path (Third argument) See `Path to your files`

Can also be provided using parameters See `path`

1.3.2.2 Parameters

Warning: Parameters should always be afters the arguments

1.3.2.2.1 <tt>**--name_column**</tt> **or** <tt>**-n**</tt> This parameter must be followed by a string. See `gene name Column`

1.3.2.2.2 <tt>**--snp_column**</tt> **or** <tt>**-s**</tt> This parameter must be followed by a string. See `snp column`

1.3.2.2.3 <tt>**--path**</tt> **or** <tt>**-p**</tt> This parameter must be followed by a string that represent a path. See `path to your files`

1.3.2.2.4 <tt>**--output_path**</tt> **or** <tt>**-o**</tt> This parameter must be followed by a string that represent a path. This parameter modify where charts and tsv are saved.

Default = `./output`

1.3.2.2.5 <tt>**--job_name**</tt> **or** <tt>**-j**</tt> This parameter must be followed by a string that can be used as folder name. This added a prefix to all files generated by this script.

Default = "Unnamed"

1.3.2.2.6 <tt>**--max_length**</tt> **or** <tt>**-m**</tt> This parameter must be followed by an integer. If this inger is lower than 0, max_length is ignored. Limit the number of snp shown inside each graph.

Default = 20

1.3.2.2.7 <tt>**--help**</tt> **or** <tt>**-h**</tt> Display a help message.

Others parameters are ignored.

1.3.2.2.8 <tt>**--output_warning**</tt> **or** <tt>**-w**</tt> Disable the warning when you are about to generate files inside a non-empty folder.

1.3.2.2.9 <tt>**--sort_by_name**</tt> **or** <tt>**-r**</tt> Disable sort species by names in global heatmap.

1.3.2.2.10 <tt>**--simplified**</tt> **or** <tt>**-i**</tt> Ignore snp number represented by 0 genes. THIS MAY CREATE A DISCONTINUOUS X-AXIS

1.3.2.2.11 <tt>**--global_heatmap**</tt> **or** <tt>**-g**</tt> Generate a heatmap that represent all species.

1.3.2.2.12 <tt>**--quantitative_barchart**</tt> **or** <tt>**-q**</tt> Generate a barchart that represent snp distribution for each file (Number of gene that have n snp)

1.3.2.2.13 <tt>**--cumulative_barchart**</tt> **or** <tt>**-c**</tt> Generate a barchart that represent snp distribution for each file (Number of gene that **at least** n snp)

1.3.2.2.14 <tt>**--cumulative_heatmap**</tt> **or** <tt>**-u**</tt> Generate a heatmap for each file. This heatmap contain only one line.

1.3.2.2.15 <tt>**--tsv**</tt> **or** <tt>**-t**</tt> Generate a tsv for all generated charts.

1.3.2.2.16 <tt>**--png**</tt> **or** <tt>**-k**</tt> Generate a png for all generated charts.

1.3.2.2.17 <tt>**--show**</tt> **or** <tt>**-d**</tt> Show all generated charts during the execution. ALL CHARTS WILL STOP THE EXECUTION UNTIL IT IS CLOSED.

1.3.2.2.18 <tt>**--svg**</tt> **or** <tt>**-v**</tt> Generate a svg for all generated charts.

1.3.2.2.19 <tt>**--show_values**</tt> **or** <tt>**-e**</tt> An integer (positive or negative)

If greater or equal to 0, all heatmap's cells will contain theirs values. if lower than 0, text size in cell is automatically determined (can be ugly in the windows displayed by -d, but assure that the text is well sized in png and svg). If unspecified, cells are empty.

1.3.2.2.20 <tt>**--uniform_y**</tt> **or** <tt>**--uniform -y**</tt> Uniformize all y-axis. All Quantitative chart and all Cumulative chart will have the same upper limit and the same scale on their y-axis. (Quantitative and Cumulative charts still have disjoint y-axis)

1.4 Example chart

Charts generated using `python3 Contig_name BiAllelic_SNP tests/TargetedFiles.json -m 10 -kgqcuw -j Examples --show_values -1`

The data used in these graphs comes from a random generator

1.4.1 Quantitative chart :

A barchart that show the proportion of gene that have a certain number of snp.

1.4.2 Cumulative chart :

A barchart that show the proportion of gene that have at least certain number of snp

1.4.3 Cumulative Heatmap :

A heatmap that show the proportion of gene that have at least certain number of snp (variation of Cumulative chart)

1.4.4 Global Heatmap :

A heatmap that show the proportion of gene that have at least certain number of snp for all species.

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 main Namespace Reference

Run scripts/snp_analyser.py.

6.1.1 Detailed Description

Run scripts/snp_analyser.py.

or any information refer to README.md or scripts/snp_analyser.py.

6.2 scripts Namespace Reference

Namespaces

- getopts_parser
- snp_analyser
- test_snp_analyser
- utilities

6.3 scripts.getopts_parser Namespace Reference

Namespaces

- getopts_parser
- test_getopts_parser

6.4 scripts.getopts_parser.getopts_parser Namespace Reference

Classes

- class GetoptsDigestionError

  Error raised during the digestion of the dictionary that contain options.
- class GetoptsOptionError

  Error raised when there is an error with an option during the getopts.
- class GetoptsParsingError

  Error raised when there is an error during the parsing of a command-line.

Functions

- any getopts_parser_boolean_option (tuple[any, any] value_restriction=None, use_default=False)

  Internal function used by getopts_parser.
- any getopts_parser_complex_option (value_restriction, value, use_default=False)

  Internal function used by getopts_parser.
- bool or None getopts_digester_check_item_endings (list[str] list_of_values, str last_char=":")

  Check if the last char of each string in `list_of_values` is equal to last_char.
- tuple[dict[str, tuple[any, any]], dict[str, str], dict[str, str], str, list[str]] getopts_digest_available_options (dict[str, tuple[any, any]] getopts_options, dict[str, any] dict_of_default_value=None)

  Transform `getopts_options` into usable instruction for getopts_retrieve_options and assure that option's aliases are coherent.
- list[tuple[str, str]] getopts_retrieve_options (list[str] argv, str short_string, list[str] long_list, list[str] main_↩ options)

  Use a list of string from a command-line (e.g.
- dict getopts_parser (list[str] or str argv, dict[str, tuple[any, any]] getopts_options, ∗str mandatory, fill_with_↩ default_values=True)

  Internal version of getopts.
- dict[str, any] or int getopts (list[str] or str argv, dict[str or tuple, None or tuple[any, any] or any] getopts_options, ∗str mandatory, str help_message=None, fill_with_default_values=True, raise_errors=False, str or tuple[str] help_options=("help",))

  Directly extract options from a command line into a dict.

Variables

- string __author__ = "Marchal Florent"
- list __credits__ = ["Marchal Florent"]

6.4.1 Function Documentation

6.4.1.1 getopts()

```
dict[str, any] or int scripts.getopts_parser.getopts_parser.getopts (
            list[str] or str argv,
            dict[str or tuple, None or tuple[any, any] or any] getopts_options,
            *str mandatory,
            str  help_message = None,
             fill_with_default_values = True,
             raise_errors = False,
            str or tuple[str]  help_options = ("help", ) )
```

Directly extract options from a command line into a dict.

All option's values are cast according to a dictionary @getopts_options which specifies option's short name (-h), option's long names (–help), option's aliases (–HELP), option's type (str, int ...) and option's default value.

This function use getopt.getopt.

You can find some usage example in ./test_getopts_parser.py

Parameters

argv	: list[str] or str =>
	• A command line e.g. :
	`"Argument1 Argument2 --Option1 value --Option2 -abc"`
	• list of string (such as sys.argv[1:]) e.g. :
	`["Argument1", "Argument2", "--Option1", "value", "--Option2", "-abc"]`

getopts_options	: dict[str, tuple[any, any]] =>
	• Structure for this dict :
	`{name: None} {tuple_of_name: None}`
	`{name: (short_name, None)}`
	`{tuple_of_name: (tuple_of_short_name, None)}`
	`{name: (tuple_of_short_name, default_value)}`
	`{name: (short_name, default_value)}`
	`{tuple_of_name: (tuple_of_short_name, default_value)}`
	`{tuple_of_name: (short_name, default_value)}`
	`{name: (tuple_of_short_name, (default_value, cast))}`
	`{name: (short_name, (default_value, cast))}`
	`{tuple_of_name: (tuple_of_short_name, (default_value, cast))}`
	`{tuple_of_name: (short_name, (default_value, cast))}`
	• Option that accept values should have a "=" at the end of each name and a ":" at the end of each short_name.
	• Option that require a value : (e.g. –file path)
	– default_value can be of any type, if callable the result of the function is used.
	– The cast can be any function. if None, nothing happen.
	• Option that does not require any value : (e.g. –help) :
	– The default value is False
	– The "on" value is True
	– If you specify a "cast" it will be used as the "on" value

Note

When an option has multiple long names, the first one in the tuple is selected as the "main_name". The "main_name" is the key used inside the returned dict.

short_names can be composed by only one character (plus ":" if needed)

Warning

do not use "-" in your options names

Parameters

∗mandatory	: str => a list of options whose value must be entered.
help_message	: str = None => A message displayed when an error is encountered or when "help" is triggered.
fill_with_default_values	= True => When True, the returned dict contains all options
raise_errors	= False => Do caught errors are raised ?
help_options	: str or tuple[str] = ("help",) => A tuple of option name that trigger `help_message`. Those options are always removed from the returned dictionary.

Returns

dict[str, any] or int =>

- dict[str, any] A dictionary that contain values related to options triggered by the command-line.
- Only when an error occur and `raise_errors` is False or when help message is displayed.
  - 1 = An error has been caught
  - 2 = help message was required.

Example :

```
  options_ = {
    'Alpha=': ("a:", (None, int)),
    'Beta=': ("b:", (None, list)),
    'Gamma': ("g", (True, False)),
    'Delta=': None,
    'Epsilon=': (None, ("Star", None)),
    'Zeta': ("Z", None),
    'Eta': None,
    'Theta': None,
    'Iota': None,
    'Kappa': None,
    'Lambda': None,
    'Mu': None,
}
val = getopts("1 --Eta --Iota --Beta Test2 -z", options_, fill_with_default_values=True)
# We have :
# val["Alpha"] == 1
# val["Beta"] == ["T", "e", "s", "t", "2"]
# val["Gamma"] is True
# val["Epsilon"] == "Star"
# val["Zeta"] is True"
# val["Mu"] is False"
```

6.4.1.2 getopts_digest_available_options()

```
tuple[dict[str, tuple[any, any]], dict[str, str], dict[str, str], str, list[str]] scripts.↩
getopts_parser.getopts_parser.getopts_digest_available_options (
```

```
            dict[str, tuple[any, any]] getopts_options,
            dict[str, any]  dict_of_default_value = None )
```

Transform `getopts_options` into usable instruction for getopts_retrieve_options and assure that option's aliases are coherent.

```
 @param getopts_options : dict[str, tuple[any, any]] =>  A dictionary that follows one of the following struct
@code
{name: None}
{tuple_of_name: None}
{name: (short_name, None)}
{tuple_of_name: (tuple_of_short_name, None)}

{name: (tuple_of_short_name, default_value)}
{name: (short_name, default_value)}
{tuple_of_name: (tuple_of_short_name, default_value)}
{tuple_of_name: (short_name, default_value)}

{name: (tuple_of_short_name, (default_value, cast))}
{name: (short_name, (default_value, cast))}
{tuple_of_name: (tuple_of_short_name, (default_value, cast))}
{tuple_of_name: (short_name, (default_value, cast))}
@endcode
@param dict_of_default_value : dict[str, any] = None => Facultative, A dictionary that will be filled using op
default values.


 @return tuple[dict[str, tuple[any, any]], dict[str, str], dict[str, str], str, list[str]] =>
- option_dict : dict[str, tuple[any, any] => Contain each option's "main_name" associated with theirs  default
    cast options. The "main_name" is the named that will be used into the dictionary
    returned by @ref getopts_parser.
- boolean_keys : dict[str, str] => Contain all options aliases related to options that can handle only two sta
- complex_keys : dict[str, str] => Contain all options aliases related to options that can handle more than tw
- short_string : str => string that correspond to "shortopts" in  @ref getopt.getopt
- long_list : list[str] => list of string that correspond to "longopts" in @ref getopt.getopt
```

6.4.1.3 getopts_digester_check_item_endings()

```
bool or None scripts.getopts_parser.getopts_parser.getopts_digester_check_item_endings (
            list[str] list_of_values,
            str  last_char = ":" )
```

Check if the last char of each string in `list_of_values` is equal to last_char.

if some are equals and others are not, an error is raised.

Parameters

list_of_values	: list[str] => A list of string
last_char	: str = ":" => A character

Returns

bool or None =>

- True : All items in `list_of_values` end by `last_char`
- False : No item in `list_of_values` end by `last_char`

- None : `list_of_values` is empty
- ValueError : Some value(s) end by `last_char` but other(s) do(es) not."

6.4.1.4 getopts_parser()

```
dict scripts.getopts_parser.getopts_parser.getopts_parser (
             list[str] or str argv,
             dict[str, tuple[any, any]] getopts_options,
             *str mandatory,
              fill_with_default_values = True )
```

Internal version of getopts.

You can use this function if you don't need getopts ' overcoat. (help message display and error handling)

This docstring is a lighten version of getopts 's docstring.

See also

getopts

Parameters

argv	: list[str] or str => List of argument (strings). Usually sys.argv[1:] or str command line.
getopts_options	: dict[str, tuple[any, any]] => A dictionary that contain options.
**mandatory*	: str =>a list of options whose value must be entered
fill_with_default_values	= True => Every unused options are added to the final dict using defaults values

Returns

dict => A dictionary that contain options.

6.4.1.5 getopts_parser_boolean_option()

```
any scripts.getopts_parser.getopts_parser.getopts_parser_boolean_option (
             tuple[any, any]  value_restriction = None,
              use_default = False )
```

Internal function used by getopts_parser.

Parameters

value_restriction	: tuple[any, any] = None => A tuple of two value. (Default value, normal value). (False, True) when None
use_default	= False => Do the default value is returned

Returns

any =>

- True if value_restriction is None
- Default value if `use_default` is True (`value_restriction` [1])
- Normal value if `use_default` is False (`value_restriction` [0])

See also

getopts_parser

6.4.1.6 getopts_parser_complex_option()

```
any scripts.getopts_parser.getopts_parser.getopts_parser_complex_option (
              value_restriction,
              value,
              use_default = False )
```

Internal function used by getopts_parser.

Apply a number of restriction materialised by `value_restriction` on a value.

```
@param value_restriction : tuple[any, callable] => A tuple of two value :
- Default value => Any, if callable, the function is called each time the default value is requested.
- restrictions => None or a callable.
     (False, True) when None
     - if value_restriction[0] is callable AND default value is requested : value_restriction [0] ()
     - if value_restriction[0] is not callable and default value is requested : value_restriction [0]
     - if value_restriction[1] is callable : value_restriction is  called using @p value as argument, the resu
           used as return value. If the result is None, directly return the @p value.
@param value => Value on which @p value_restriction is applied.
@param use_default = False => If True, return the default value.

@return any => @p value transformed by  @p value_restriction [1]

@see getopts_parser
```

6.4.1.7 getopts_retrieve_options()

```
list[tuple[str, str]] scripts.getopts_parser.getopts_parser.getopts_retrieve_options (
             list[str] argv,
             str short_string,
             list[str] long_list,
             list[str] main_options )
```

Use a list of string from a command-line (e.g.

sys.argv[1:]) to extract selected options and theirs associated values.

Parameters

argv	: list[str] => A list of string (Same format as sys.argv[1:])
short_string	: str => string that correspond to shortopts in getopt.getopt
long_list	: list[str] => list of string that correspond to longopts in getopt.getopt
main_options	: list[str] => An ordered list of all options. Is used to determine the correspondence between arguments and parameters.

Returns

list[Tuple[str, str]] => A list composed of tuple that contain all selected options and associated with their values.

6.4.2 Variable Documentation

6.4.2.1 __author__

```
string scripts.getopts_parser.getopts_parser.__author__ = "Marchal Florent"  [private]
```

6.4.2.2 __credits__

```
list scripts.getopts_parser.getopts_parser.__credits__ = ["Marchal Florent"]  [private]
```

6.5 scripts.getopts_parser.test_getopts_parser Namespace Reference

Functions

- def test_simple_option ()

  Test simple functionality of getopts_parser.
- def test_aliases ()

  Test aliases functionality.
- def test_defaults_values ()

  Test default value and cast.
- def test_all ()

  Call each test functions.

Variables

- string __author__ = "Marchal Florent"
- list __credits__ = ["Marchal Florent"]

6.5.1 Function Documentation

6.5.1.1 test_aliases()

```
def scripts.getopts_parser.test_getopts_parser.test_aliases ( )
```

Test aliases functionality.

6.5.1.2 test_all()

```
def scripts.getopts_parser.test_getopts_parser.test_all ( )
```

Call each test functions.

6.5.1.3 test_defaults_values()

```
def scripts.getopts_parser.test_getopts_parser.test_defaults_values ( )
```

Test default value and cast.

6.5.1.4 test_simple_option()

```
def scripts.getopts_parser.test_getopts_parser.test_simple_option ( )
```

Test simple functionality of getopts_parser.

6.5.2 Variable Documentation

6.5.2.1 __author__

```
string scripts.getopts_parser.test_getopts_parser.__author__ = "Marchal Florent"  [private]
```

6.5.2.2 __credits__

```
list scripts.getopts_parser.test_getopts_parser.__credits__ = ["Marchal Florent"]  [private]
```

6.6 scripts.snp_analyser Namespace Reference

Functions

- def help_usage ()
- bool filter_integer_greater_or_equal_to_0_ignore_0 (str or int key, dict dictionary=None)

  Test if the value in front of the key `key` inside `dictionary` can be cast into an integer bigger than 0.
- dict[int, dict[str, int]] compile_gene_snp (dict[str, any] genes_snp, dict[int, dict[str, int]] dict_of_number=None, str group="None")

  Extract the number of snp of all genes contained in `genes_snp` (snp = `genes_snp` 's values).
- (list[list[int]], list[int]) make_data_matrix (dict[int, dict[str, int]] compiled_dict, str group, *str groups, bool simplified=True, int max_length=None)

  Use a compiled dict from compile_gene_snp to create a matrix of value.
- list[int] generate_cumulative_list (list[int] or list[float] list_of_numbers, reversed_=False)

  Take a list of number and sum all values.
- int main (str path, str name_column, str snp_column, str file_separator="\t", bool simplified=True, int max_length=None, str output_path="output", bool output_warning=True, str job_name=None, bool global_heatmap=True, bool quantitative_barchart=False, bool cumulative_barchart=False, bool cumulative_heatmap=False, bool tsv=False, bool png=False, bool show=False, bool svg=True, bool sort_by_name=True, bool uniform_y=True, int show_values=-1)

  Create a number of chart related to snp analysis.
- def main_using_getopts (list[str] or str argv)

Variables

- string __author__ = "Marchal Florent"
- list __credits__ = ["Florent Marchal", "Vetea Jacot", "Concetta Burgarella", "Vincent Ranwez", "Nathalie Chantret"]
- dictionary __getopts__

6.6.1 Function Documentation

6.6.1.1 compile_gene_snp()

```
dict[int, dict[str, int]] scripts.snp_analyser.compile_gene_snp (
            dict[str, any] genes_snp,
            dict[int, dict[str, int]]  dict_of_number = None,
            str  group = "None" )
```

Extract the number of snp of all genes contained in `genes_snp` (snp = `genes_snp` 's values).

Each number of snp is stored inside a new dictionary (`dict_of_number` 's keys). A dict is created in front of all keys (i.e. snp number). This dict contain the `group` (key) and the number of occurrences of this snp number for this key.

Parameters

genes_snp	: dict[str,any] => A dictionary from extract_data_from_table. e.g. {gene_1: number_of_snp_in_gene_1} => `{"gene1": 3}`

Note

Values (number of snp) inside this dict are trans typed into integers.

Parameters

dict_of_number	: dict[int, dict[str,int]] = None. A dict with the same structure as dictionaries returned by this function.
group	: str = "None" => Each occurrence of a number of snp increment the counter related to this group.

Returns

dict[int, dict[str, int]] => A dictionary that store all number of snp found along with the number of occurrences
`{number_of_snp_1 : {group1: number_of_occurrences_of_number_of_snp_1_in_this_group}`

Warning

values `genes_snp` are cast into integer. Also, there is no verification made to see if the values are positive. We assume that data has been filtered using filter_integer_greater_or_equal_to_0_ignore_0 in extract_data↩ _from_table

6.6.1.2 filter_integer_greater_or_equal_to_0_ignore_0()

```
bool scripts.snp_analyser.filter_integer_greater_or_equal_to_0_ignore_0 (
            str or int key,
            dict  dictionary = None )
```

Test if the value in front of the key `key` inside `dictionary` can be cast into an integer bigger than 0.

Meant to be used inside extract_data_from_table as a "filter_" using a lambda.

- value == 0 : Line is ignored
- value > 0 : Line is kept
- Value < 0 : Error is raised

Warning

does not support "1.0 nor "1,0" format

Parameters

key	: str => A key contained by `dictionary`.
dictionary	: dict = None => A dictionary that contain `key`. If None, the function will assume that `dictionary = {key: key}`

Returns

bool => True : Yes False : value equal to 0 @raise ValueError when the value is an integer lower than 0

6.6.1.3 generate_cumulative_list()

```
list[int] scripts.snp_analyser.generate_cumulative_list (
             list[int] or list[float] list_of_numbers,
              reversed_ = False )
```

Take a list of number and sum all values.
`[0, 5, 6, 1] => [0+5+6+1, 5+6+1, 6+1, 1] == [12, 12, 7, 1]`

Parameters

list_of_numbers	: list[int] or list[float] => A list that contain numbers.
*reversed_*	= False => Do the accumulation start at the end and end at the beginning.

Returns

list[int] => A list of number

6.6.1.4 help_usage()

```
def scripts.snp_analyser.help_usage ( )
```

6.6.1.5 main()

```
int scripts.snp_analyser.main (
             str path,
             str name_column,
             str snp_column,
             str  file_separator = "\t",
             bool  simplified = True,
             int  max_length = None,
             str  output_path = "output",
```

```
bool  output_warning = True,
str  job_name = None,
bool  global_heatmap = True,
bool  quantitative_barchart = False,
bool  cumulative_barchart = False,
bool  cumulative_heatmap = False,
bool  tsv = False,
bool  png = False,
bool  show = False,
bool  svg = True,
bool  sort_by_name = True,
bool  uniform_y = True,
int  show_values = -1 )
```

Create a number of chart related to snp analysis.

As an example we will consider the following flatFile :

GeneName	GeneID	NumberOfSnp	GeneSize
Gene1	123	5	1000
Gene2	456	10	2000

Parameters

path	: str => Path that lead to a number of flatfile : • .json : {complete file path : Species name} • folder : Use file inside the folder (does not scan the folder recursively)
name_column	: str => Name of the column that contain a primary key e.g. GeneName, GeneID. If two line have the same "primary key", the last one will be used.
snp_column	: str => Name of the column that contain a count of snp e.g. NumberOfSnp
file_separator	: str = "\t" => The separator used in all flat file considered.
simplified	: bool = True => Do number of snp represented by 0 gene are deleted from the result
max_length	: int = None => Maximum length of each graph. keep the nth first result. Should be greater or equal to 1 otherwise, it would be ignored.
output_path	: str = "output" => Where graphs are saved.
output_warning	: bool = True => Ask confirmation when at least one file can be erased by this program.
job_name	: str = None => A name for this execution. (This creates a separated folder in `output_path`). Default = "unnamed"
global_heatmap	: bool = True => Do a heatmap that is the combination of all cumulative_heatmap is created
quantitative_barchart	: bool = False => Do this program create a barchart of snp distribution for each file (Number of gene that have n snp)
cumulative_barchart	: bool = False => Do this program create a barchart of snp distribution for each file ? (Number of gene that have AT LEAST n snp)
cumulative_heatmap	: bool = False => Do this program create a heatmap of snp distribution for each file ? (Number of gene that have AT LEAST n snp)
tsv	: bool = False => Do values used for chart are saved in a flatfile (.tsv)
png	: bool = False => Do created charts are saved as png
show	: bool = False => Do created charts are saved are shown

Warning

Each time a chart is shown, the program stop. It will resume when the chart is closed.

Parameters

svg	: bool = True => Do created charts are saved as svg (vectorize image)
sort_by_name	: bool = True => Do species are sorted in lexicographic order ?
uniform_y	: bool = True => Do all barchart share the same y-axis ?
show_values	: int = None => If greater or equal to 0, all cells will contain theirs values. if lower than 0, text in cell in automatically determined (can be ugly when show is True, but assure that the text is good in png and svg). If None, nothing happen.

Returns

int => if greater than 0, an error occurred.

- 1 job stopped by user
- 2 no species found

6.6.1.6 main_using_getopts()

```
def scripts.snp_analyser.main_using_getopts (
             list[str] or str argv )
```

```
@brief start @ref snp_charts.main using a string or sys.argv[1:]
Example :
    - main_using_getopts(sys.argv[1:])
    - main_using_getopts("name_column snp_column tests/TargetedFiles.json -m 20 -gv -w -j Tests -e -1")

@param argv : list[str] or str => List of argument (strings). Usually sys.argv[1:].
@note If the first values are not known option (option in @p getopts_options), the function will consider that
those values correspond to the nth first option that require an argument in @p getopts_options.
In below example, you can give two arguments. The first will be matched with "Alpha" and the last with "Beta".
@code
argv = ["5", "6"]
getopts_options = {
    "Alpha=":               ("a:", None),
    "Beta=:                 ('b', (0, int)),
    "Gamma":                ("g", ("data/", str)),
    "Delta=":               ("d:", (0, lambda integer : int(integer) - 1)),
}
@endcode
@return int => exit code
```

6.6.1.7 make_data_matrix()

```
(list[list[int]], list[int]) scripts.snp_analyser.make_data_matrix (
            dict[int, dict[str, int]] compiled_dict,
            str group,
            *str groups,
            bool  simplified = True,
            int  max_length = None )
```

Use a compiled dict from compile_gene_snp to create a matrix of value.

Each lines represent one group (`groups` & `group`). Each line contain the number of genes that contain at least n genes

Note

Only groups specified in `groups` and `group` are extracted from `compiled_dict`

Note

For below example we will consider that :

- `compiled_dict = {1: {"E. coli": 5},`
  `2: {"E. coli": 3, "HIV": 4},`
  `3: {"HIV": 2}`
  `4: {"E. coli": 1, "HIV": 1}`
  `}`

Parameters

compiled_dict	: dict[int,dict[str,int]] => a compiled dict from compile_gene_snp
group	: str => A group name (e.g. Species name : "E. coli")
**groups*	: str => Same as `group`. Additional species name.
simplified	: bool = True => Do number of snp represented by 0 gene are deleted from the result ?

Note

This the presence / absence of a number is affected by other groups

- with group="E. coli":
  - If True:
    `([[5, 3, 1]], [1, 2, 4])`
  - If False:
    `([[5, 3, 0, 1]], [1, 2, 3, 4])`
- with group="E. coli" and groups= ("HIV",):
  - If True:
    `([[5, 3, 0, 1], [0, 4, 2, 1]], [1, 2, 4])`

Parameters

max_length	: int = None => Limit the length of each lines of the matrix. should be greater than 0. If not, max_length is ignored.

Returns

(list[list[int]], list[int]) => Return a matrix and a list. Each lines of the matrix represent the number of genes of a species (group) that contain n snp : If the position 1 of a line is equal to 4, there is 4 genes in the selected species that contain list[i] snp. The y axes can be labeled using `group` and `groups` (order in conserved) and the x-axis is labeled by the list[int].

Note

For some return example, see `simplified` for return example

6.6.2 Variable Documentation

6.6.2.1 __author__

```
string scripts.snp_analyser.__author__ = "Marchal Florent"  [private]
```

6.6.2.2 __credits__

```
list scripts.snp_analyser.__credits__ = ["Florent Marchal", "Vetea Jacot", "Concetta Burgarella",
"Vincent Ranwez", "Nathalie Chantret"]  [private]
```

6.6.2.3 __getopts__

```
dictionary scripts.snp_analyser.__getopts__  [private]
```

6.7 scripts.test_snp_analyser Namespace Reference

Functions

- def test_greater_than_0_int_filter ()
- def test_compile_gene_snp ()
- def test_make_data_matrix ()
- def test_generate_cumulative_list ()
- def test_all ()

6.7.1 Function Documentation

6.7.1.1 test_all()

```
def scripts.test_snp_analyser.test_all ( )
```

```
@brief Call all test functions
```

6.7.1.2 test_compile_gene_snp()

```
def scripts.test_snp_analyser.test_compile_gene_snp ( )
```

```
@brief Test compile_gene_snp
```

6.7.1.3 test_generate_cumulative_list()

```
def scripts.test_snp_analyser.test_generate_cumulative_list ( )
```

6.7.1.4 test_greater_than_0_int_filter()

```
def scripts.test_snp_analyser.test_greater_than_0_int_filter ( )
```

```
@brief Test greater_than_0_int_filter
```

6.7.1.5 test_make_data_matrix()

```
def scripts.test_snp_analyser.test_make_data_matrix ( )
```

```
@brief Test make_data_matrix
```

6.8 scripts.utilities Namespace Reference

Namespaces

- utilities

6.9 scripts.utilities.utilities Namespace Reference

Classes

- class FilterError

Functions

- def export_list_in_tsv_as_rows (str path, *rows, file_mode="w", encoding="UTF-8", list y_legend=None, list x_legend=None)

  Accept a number of list that represent rows of a tab and turn it into a tsv (flat file).
- def chart_export (list[list[int]] data, bool show=False, str png=None, str tsv=None, str svg=None, list x_↩ legend=None, list y_legend=None)

  Export the current chart.
- dict[str, str] parse_line (list[str] legend, str line, str separator="\t")

  Turn a line form a flat File with its legend and turn it into a dictionary.
- dict[str, any] extract_data_from_table (str path, str key, str value, str separator="\t", list legend=None, callable filter_=None)

  Read a table contained inside a flatFile (e.g.
- def make_bar_char (list[int] data, list x_legend=None, bool x_legend_is_int=True, bool y_legend_is_int=True, str chart_name=None, str title=None, str xlabel=None, str ylabel=None, bool show=False, str png=None, str tsv=None, str svg=None, bool erase_last_plt=True, int y_max_value=None)

  Create a plt.bar using a bunch of argument.
- def make_heatmap (list[list[int]] data, list x_legend=None, list y_legend=None, str title=None, str xlabel=None, str ylabel=None, bool show=False, str png=None, str tsv=None, str svg=None, bool erase_last_plt=True, int contain_number=None, str test_color="#a0a0a0", str cmap="jet")

  Create a heatmap using a bunch of argument.

6.9.1 Function Documentation

6.9.1.1 chart_export()

```
def scripts.utilities.utilities.chart_export (
            list[list[int]] data,
            bool  show = False,
            str  png = None,
            str  tsv = None,
            str  svg = None,
            list  x_legend = None,
            list  y_legend = None )
```

Export the current chart.

Note

if data is the only argument, nothing will happen.

Parameters

data	: list[list[int]] => A matrix of values
show	: bool = False => Do current plot will be displayed in a pop-up ?

Warning

this will stop program execution until the pop-up is closed.

Parameters

png	: str = None => Give a path to export the current plot as png
tsv	: str = None => Give a path to export `data` into a tsv.
svg	: str = None => Give a path to export the current plot as svg
y_legend	: list = None => When `tsv` is not none: A list of item to be display in the first column (export_list_in_tsv_as_rows)
x_legend	: list = None => When `tsv` is not none: A list of item to be display in the first line (export_list_in_tsv_as_rows)

6.9.1.2 export_list_in_tsv_as_rows()

```
def scripts.utilities.utilities.export_list_in_tsv_as_rows (
              str path,
              * rows,
               file_mode = "w",
               encoding = "UTF-8",
              list  y_legend = None,
              list  x_legend = None )
```

Accept a number of list that represent rows of a tab and turn it into a tsv (flat file).

Warning

Any "\t" or "\n" in rows' values will disrupt lines and / or columns

- each "\t" will generate additional columns
- each "\n" will break the current line and start a new one.

Parameters

path	: str => path (and name) of the file that will be written.
∗rows	=> A number of list
file_mode	= "w" => "w" or "a" • "w": if a file with the same path exist, old file is erased • "a": if a file with the same path exist, old file append new values
encoding	= "UTF-8" => File encoding
y_legend	: list = None => A list of item to be display in the first column
x_legend	: list = None => A list of item to be display in the first line

6.9.1.3 extract_data_from_table()

```
dict[str, any] scripts.utilities.utilities.extract_data_from_table (
            str path,
            str key,
            str value,
            str  separator = "\t",
            list  legend = None,
            callable  filter_ = None )
```

Read a table contained inside a flatFile (e.g.

tsv, csv, ...)

Warning

If the column `key` contains the same value multiple times, only the last one is kept.

Parameters

path	: Path to a flatFile.
key	: A column name that can be found in the legend. This will be used as a key in the returned dict. Example: "Column3"
value	: A column name that can be found in the legend. This will be used as a value in the returned dict WHEN `filter` returns None or True. Example: "Column2"
separator	: The symbol that splits line's values. Example: "|", "\n", "\t" ...
legend	: If None: The first non-empty line in the file split using `separator`. Else: A list of column names. Example: [Column1, Column2, Column3]
*filter_*	: A function that accepts 3 arguments: `key`, `value`, and the parsed line (dict). It selects/generates the value present next to each key. • If it returns True or None: value in the column `value`. • If it returns False: this line is ignored. • Else: The returned value is used (instead of the content of the column `value`).

Note

filter_ is called one time per line.

Returns

A dictionary: {values in the column `key` (values that do not pass `filter_` are ignored): values in the column `value` OR value returned by `filter_`}

6.9.1.4 make_bar_char()

```
def scripts.utilities.utilities.make_bar_char (
              list[int] data,
              list  x_legend = None,
              bool  x_legend_is_int = True,
              bool  y_legend_is_int = True,
              str  chart_name = None,
              str  title = None,
              str  xlabel = None,
              str  ylabel = None,
              bool  show = False,
              str  png = None,
              str  tsv = None,
              str  svg = None,
              bool  erase_last_plt = True,
              int  y_max_value = None )
```

Create a plt.bar using a bunch of argument.

This function is made to assure a correct looking legend when used for snp.

Parameters

data	: list[int] => A list of integer
x_legend	: list = None => Values used to legend the x-axis.
x_legend_is_int	: bool = True => Do x-axis represent oly integer
y_legend_is_int	: bool = True => Do y-axis represent oly integer
chart_name	: str = None => A name that will be used if tsv is not None to name a line.
title	: str = None => A title for this chart
xlabel	: str = None => A title for the x-axis
ylabel	: str = None => A title for the y-axis
show	: bool = False => Do current plot will be displayed ?

Warning

this will stop program execution until the pop-up is closed.

Parameters

png	: str = None => Give a path to export the current plot as png
tsv	: str = None => Give a path to export `data` into a tsv.
svg	: str = None => Give a path to export the current plot as svg
y_max_value	: int = None => The y-axis will stop at this value
erase_last_plt	: bool = True => If True, last plot is removed from matplotlib.pyplot display

6.9.1.5 make_heatmap()

```
def scripts.utilities.utilities.make_heatmap (
              list[list[int]] data,
```

```
list  x_legend = None,
list  y_legend = None,
str  title = None,
str  xlabel = None,
str  ylabel = None,
bool  show = False,
str  png = None,
str  tsv = None,
str  svg = None,
bool  erase_last_plt = True,
int  contain_number = None,
str  test_color = "#a0a0a0",
str  cmap = "jet" )
```

Create a heatmap using a bunch of argument.

This function is made to assure a correct looking legend when used for snp.

Parameters

data	: list[int] => A list of integer
x_legend	: list = None => Values used to label the x-axis.
y_legend	: list = None => Values used to label the y-axis.
title	: str = None => A title for this chart
xlabel	: str = None => A title for the x-axis
ylabel	: str = None => A title for the y-axis
show	: bool = False => Do current plot will be displayed ?

Warning

this will stop program execution until the pop-up is closed.

Parameters

png	: str = None => Give a path to export the current plot as png
tsv	: str = None => Give a path to export `data` into a tsv.
svg	: str = None => Give a path to export the current plot as svg
erase_last_plt	: bool = True => If True, last plot is removed from matplotlib.pyplot memory
contain_number	: int = None => If greater or equal to 0, all cells will contain theirs values. if lower than 0, text in cell in automatically determined. If None, nothing happen.
test_color	: str = #a0a0a0 => HTML color code for text inside cells

Note

Only when contain_number is True

Parameters

cmap	: str = jet => Color mod. supported values are 'Accent', 'Accent_r', 'Blues', 'Blues_r', 'BrBG', 'BrBG_r', 'BuGn', 'BuGn_r', 'BuPu', 'BuPu_r', 'CMRmap', 'CMRmap_r', 'Dark2', 'Dark2_r', 'GnBu', 'GnBu_r', 'Grays', 'Greens', 'Greens_r', 'Greys', 'Greys_r', 'OrRd', 'OrRd_r', 'Oranges', 'Oranges_r', 'PRGn', 'PRGn_r', 'Paired', 'Paired_r', 'Pastel1', 'Pastel1_r', 'Pastel2', 'Pastel2_r', 'PiYG', 'PiYG_r', 'PuBu', 'PuBuGn', 'PuBuGn_r', 'PuBu_r', 'PuOr', 'PuOr_r', 'PuRd', 'PuRd_r', 'Purples', 'Purples_r', 'RdBu', 'RdBu_r', 'RdGy', 'RdGy_r', 'RdPu', 'RdPu_r', 'RdYlBu', 'RdYlBu_r', 'RdYlGn', 'RdYlGn_r', 'Reds', 'Reds_r', 'Set1', 'Set1_r', 'Set2', 'Set2_r', 'Set3', 'Set3_r', 'Spectral', 'Spectral_r', 'Wistia', 'Wistia_r', 'YlGn', 'YlGnBu', 'YlGnBu_r', ' YlGn_r', 'YlOrBr', 'YlOrBr_r', 'YlOrRd', 'YlOrRd_r', 'afmhot', 'afmhot_r', 'autumn', 'autumn_r', 'binary', 'binary_r', 'bone', 'bone_r', 'brg', 'brg_r', 'bwr', 'bwr_r', 'cividis', 'cividis_r', 'cool', 'cool_r', 'coolwarm', 'coolwarm_r', 'copper', 'copper_r', 'cubehelix', 'cubehelix_r', 'flag', 'flag_r', 'gist_earth', 'gist_earth_r', 'gist_gray', 'gist_gray_r', 'gist_grey', 'gist_heat', 'gist_heat_r', 'gist_ncar', 'gist_ncar_r', 'gist_rainbow', 'gist_rainbow_r', 'gist_stern', 'gist_stern_r', 'gist_yarg', 'gist_yarg_r', 'gist_yerg', 'gnuplot', 'gnuplot2', 'gnuplot2_r', 'gnuplot_r', 'gray', 'gray_r', 'grey', 'hot', 'hot_r', 'hsv', 'hsv_r', 'inferno', 'inferno_r', 'jet', 'jet_r', 'magma', 'magma_r', 'nipy_spectral', 'nipy_spectral_r', 'ocean', 'ocean_r', 'pink', 'pink_r', 'plasma', 'plasma_r', 'prism', 'prism_r', 'rainbow', 'rainbow_r', 'seismic', 'seismic_r', 'spring', 'spring_r', 'summer', 'summer_r', 'tab10', 'tab10_r', 'tab20', 'tab20_r', 'tab20b', 'tab20b_r', 'tab20c', 'tab20c_r', 'terrain', 'terrain_r', 'turbo', 'turbo_r', 'twilight', 'twilight_r', 'twilight_shifted', 'twilight_shifted_r', 'viridis', 'viridis_r', 'winter', 'winter_r'

6.9.1.6 parse_line()

```
dict[str, str] scripts.utilities.utilities.parse_line (
            list[str] legend,
            str line,
            str  separator = "\t" )
```

Turn a line form a flat File with its legend and turn it into a dictionary.

Parameters

legend	: Names all the line's columns. Example: ["A", "B", "C"]
line	: Contains all the line's values. Example: "1|2|3|", "1|2|3" ...
separator	: The symbol that splits the line's values. Example: "|", "\n", "\t" ...

Returns

A dictionary composed of legend's values and line's values. Example (using previous examples):
`{"A": "1", "B": "2", "C": "3"}`

Note

The returned dict always contain the same number of object than legend.

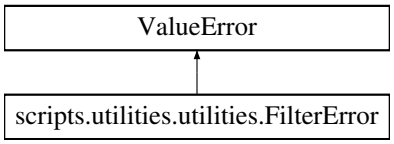
- If legend > line : part of the legend's values will point to an empty string
- If legend < line : part of the line will be ignored

Chapter 7

Class Documentation

7.1 scripts.utilities.utilities.FilterError Class Reference

Inheritance diagram for scripts.utilities.utilities.FilterError:



7.1.1 Detailed Description

```
An error raised by @ref extract_data_from_table when a filter_ raise an error
```

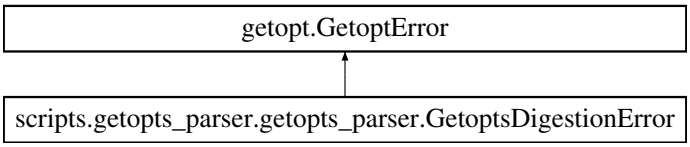
The documentation for this class was generated from the following file:

- utilities.py

7.2 scripts.getopts_parser.getopts_parser.GetoptsDigestionError Class Reference

Error raised during the digestion of the dictionary that contain options.

Inheritance diagram for scripts.getopts_parser.getopts_parser.GetoptsDigestionError:



7.2.1 Detailed Description

Error raised during the digestion of the dictionary that contain options.

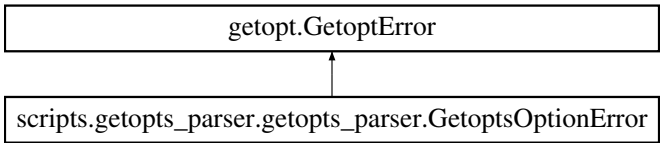
The documentation for this class was generated from the following file:

- getopts_parser.py

7.3 scripts.getopts_parser.getopts_parser.GetoptsOptionError Class Reference

Error raised when there is an error with an option during the getopts.

Inheritance diagram for scripts.getopts_parser.getopts_parser.GetoptsOptionError:



7.3.1 Detailed Description

Error raised when there is an error with an option during the getopts.

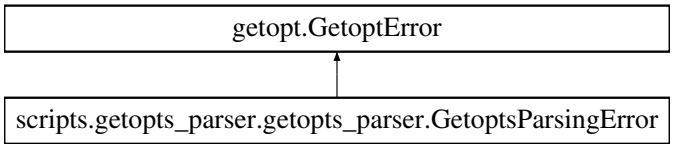
The documentation for this class was generated from the following file:

- getopts_parser.py

7.4 scripts.getopts_parser.getopts_parser.GetoptsParsingError Class Reference

Error raised when there is an error during the parsing of a command-line.

Inheritance diagram for scripts.getopts_parser.getopts_parser.GetoptsParsingError:



7.4.1 Detailed Description

Error raised when there is an error during the parsing of a command-line.

The documentation for this class was generated from the following file:

- getopts_parser.py

Chapter 8

File Documentation

8.1 __init__.py File Reference

Namespaces

- scripts

8.2 __init__.py File Reference

Namespaces

- scripts.getopts_parser

8.3 __init__.py File Reference

Namespaces

- scripts.utilities

8.4 getopts_parser.py File Reference

Classes

- class scripts.getopts_parser.getopts_parser.GetoptsDigestionError

  Error raised during the digestion of the dictionary that contain options.
- class scripts.getopts_parser.getopts_parser.GetoptsOptionError

  Error raised when there is an error with an option during the getopts.
- class scripts.getopts_parser.getopts_parser.GetoptsParsingError

  Error raised when there is an error during the parsing of a command-line.

Namespaces

- scripts.getopts_parser.getopts_parser

Functions

- any scripts.getopts_parser.getopts_parser.getopts_parser_boolean_option (tuple[any, any] value_↩ restriction=None, use_default=False)

  Internal function used by getopts_parser.
- any scripts.getopts_parser.getopts_parser.getopts_parser_complex_option (value_restriction, value, use_↩ default=False)

  Internal function used by getopts_parser.
- bool or None scripts.getopts_parser.getopts_parser.getopts_digester_check_item_endings (list[str] list_of_↩ values, str last_char=":")

  Check if the last char of each string in `list_of_values` is equal to last_char.
- tuple[dict[str, tuple[any, any]], dict[str, str], dict[str, str], str, list[str]] scripts.getopts_parser.getopts_parser.getopts_digest_availabl (dict[str, tuple[any, any]] getopts_options, dict[str, any] dict_of_default_value=None)

  Transform `getopts_options` into usable instruction for getopts_retrieve_options and assure that option's aliases are coherent.
- list[tuple[str, str]] scripts.getopts_parser.getopts_parser.getopts_retrieve_options (list[str] argv, str short_↩ string, list[str] long_list, list[str] main_options)

  Use a list of string from a command-line (e.g.
- dict scripts.getopts_parser.getopts_parser.getopts_parser (list[str] or str argv, dict[str, tuple[any, any]] getopts_options, ∗str mandatory, fill_with_default_values=True)

  Internal version of getopts.
- dict[str, any] or int scripts.getopts_parser.getopts_parser.getopts (list[str] or str argv, dict[str or tuple, None or tuple[any, any] or any] getopts_options, ∗str mandatory, str help_message=None, fill_with_default_↩ values=True, raise_errors=False, str or tuple[str] help_options=("help",))

  Directly extract options from a command line into a dict.

Variables

- string scripts.getopts_parser.getopts_parser.__author__ = "Marchal Florent"
- list scripts.getopts_parser.getopts_parser.__credits__ = ["Marchal Florent"]

8.5 main.py File Reference

Namespaces

- main

  Run scripts/snp_analyser.py.

8.6 README.md File Reference

8.7 snp_analyser.py File Reference

Create a number of chart related to snp (simple nucleotide polymorphism) analysis.

Namespaces

- scripts.snp_analyser

Functions

- def scripts.snp_analyser.help_usage ()
- bool scripts.snp_analyser.filter_integer_greater_or_equal_to_0_ignore_0 (str or int key, dict dictionary=None)

  Test if the value in front of the key `key` inside `dictionary` can be cast into an integer bigger than 0.
- dict[int, dict[str, int]] scripts.snp_analyser.compile_gene_snp (dict[str, any] genes_snp, dict[int, dict[str, int]] dict_of_number=None, str group="None")

  Extract the number of snp of all genes contained in `genes_snp` (snp = `genes_snp` 's values).
- (list[list[int]], list[int]) scripts.snp_analyser.make_data_matrix (dict[int, dict[str, int]] compiled_dict, str group, *str groups, bool simplified=True, int max_length=None)

  Use a compiled dict from compile_gene_snp to create a matrix of value.
- list[int] scripts.snp_analyser.generate_cumulative_list (list[int] or list[float] list_of_numbers, reversed_=False)

  Take a list of number and sum all values.
- int scripts.snp_analyser.main (str path, str name_column, str snp_column, str file_separator="\t", bool simplified=True, int max_length=None, str output_path="output", bool output_warning=True, str job_↩ name=None, bool global_heatmap=True, bool quantitative_barchart=False, bool cumulative_barchart=False, bool cumulative_heatmap=False, bool tsv=False, bool png=False, bool show=False, bool svg=True, bool sort_by_name=True, bool uniform_y=True, int show_values=-1)

  Create a number of chart related to snp analysis.
- def scripts.snp_analyser.main_using_getopts (list[str] or str argv)

Variables

- string scripts.snp_analyser.__author__ = "Marchal Florent"
- list scripts.snp_analyser.__credits__ = ["Florent Marchal", "Vetea Jacot", "Concetta Burgarella", "Vincent Ranwez", "Nathalie Chantret"]
- dictionary scripts.snp_analyser.__getopts__

8.7.1 Detailed Description

Create a number of chart related to snp (simple nucleotide polymorphism) analysis.

8.7.1.1 to use this file

Python3 [Column that contain gene's names] [Column tha contain the number of snp] [path to a folder that contain your files] [options]

See string returned by snp_charts.help_usage() for information

8.7.1.2 Charts

- Quantitative bar chart (-q) : Show the number of gene (y) per number of snp (x)
- Cumulative bar chart (-c): Show the number of gene (y) that have at least n snp (x)
- Monoheatmap (-u): Cumulative bar chart but it's a heatmap
- global heatmap (-g): concatenation of all Monoheatmap

8.7.1.3 Formats

- as tsv (-t)
- as png (-k)
- as svg (-b)
- as a pop-up (-d)

8.7.2 Author(s)

- Created by Marchal Florent on 06/05/2024. This module has been made in 2024 during an internship at the UMR Agap, GE2pop (France, Montpellier)

8.7.3 Librairies/Modules

- os
- json
- matplotlib.pyplot
- sys
- getopts_parser
- utilities

8.8 test_getopts_parser.py File Reference

Namespaces

- scripts.getopts_parser.test_getopts_parser

Functions

- def scripts.getopts_parser.test_getopts_parser.test_simple_option ()

  Test simple functionality of getopts_parser.
- def scripts.getopts_parser.test_getopts_parser.test_aliases ()

  Test aliases functionality.
- def scripts.getopts_parser.test_getopts_parser.test_defaults_values ()

  Test default value and cast.
- def scripts.getopts_parser.test_getopts_parser.test_all ()

  Call each test functions.

Variables

- string scripts.getopts_parser.test_getopts_parser.__author__ = "Marchal Florent"
- list scripts.getopts_parser.test_getopts_parser.__credits__ = ["Marchal Florent"]

8.9 test_snp_analyser.py File Reference

Namespaces

- scripts.test_snp_analyser

Functions

- def scripts.test_snp_analyser.test_greater_than_0_int_filter ()
- def scripts.test_snp_analyser.test_compile_gene_snp ()
- def scripts.test_snp_analyser.test_make_data_matrix ()
- def scripts.test_snp_analyser.test_generate_cumulative_list ()
- def scripts.test_snp_analyser.test_all ()

8.10 utilities.py File Reference

Contain a number of function related to file reading and generation of figure using matplotlib.

Classes

- class scripts.utilities.utilities.FilterError

Namespaces

- scripts.utilities.utilities

Functions

- def scripts.utilities.utilities.export_list_in_tsv_as_rows (str path, *rows, file_mode="w", encoding="UTF-8", list y_legend=None, list x_legend=None)

  Accept a number of list that represent rows of a tab and turn it into a tsv (flat file).
- def scripts.utilities.utilities.chart_export (list[list[int]] data, bool show=False, str png=None, str tsv=None, str svg=None, list x_legend=None, list y_legend=None)

  Export the current chart.
- dict[str, str] scripts.utilities.utilities.parse_line (list[str] legend, str line, str separator="\t")

  Turn a line form a flat File with its legend and turn it into a dictionary.
- dict[str, any] scripts.utilities.utilities.extract_data_from_table (str path, str key, str value, str separator="\t", list legend=None, callable filter_=None)

  Read a table contained inside a flatFile (e.g.
- def scripts.utilities.utilities.make_bar_char (list[int] data, list x_legend=None, bool x_legend_is_int=True, bool y_legend_is_int=True, str chart_name=None, str title=None, str xlabel=None, str ylabel=None, bool show=False, str png=None, str tsv=None, str svg=None, bool erase_last_plt=True, int y_max_value=None)

  Create a plt.bar using a bunch of argument.
- def scripts.utilities.utilities.make_heatmap (list[list[int]] data, list x_legend=None, list y_legend=None, str title=None, str xlabel=None, str ylabel=None, bool show=False, str png=None, str tsv=None, str svg=None, bool erase_last_plt=True, int contain_number=None, str test_color="#a0a0a0", str cmap="jet")

  Create a heatmap using a bunch of argument.

8.10.1 Detailed Description

Contain a number of function related to file reading and generation of figure using matplotlib.

It's a toolbox for scripts/snp_analyser.py

Index