

## Pipeline Description

The pipeline structure follows the Snakemake modularization. Modules are set in a main snake file which will be called for the workflow execution (Figure 1). A module consists of one or more rules which decompose the workflow into small steps by specifying how to create sets of output files from sets of input files. A rule can not be run if the input files are not found, also if an error occurs during the execution the output files created are deleted to keep consistency in the workflow in case a corrupted output is created. A rule also could include configurable parameters such as params, shell, messages, among others (Figure 2). File names, paths and parameters can be set in specific configuration files for easy modifications and administration (Figure 3 and Figure 4).

```
# Config files containing hard coded variables
configfile: os.path.abspath(os.path.join(os.path.dirname(workflow.basedir), 'snakefiles/config.json'))

# Runtime variables
include: "variables.py"

# rule all
rule all:
    input:
        dupconcordant = config['data_dup_path'] + '/' + config['dup_concordant_file'],
        dupdiscordant = config['data_dup_path'] + '/' + config['dup_disconcordant_file']
        samplespassmerged = config['sample_merged_file']

# Include modules
include: "module1-data-conversion.snake"
include: "module2-data-calling.snake"
include: "module3-data-clean.snake"
```

Figure 1. Snakemake file. This file includes the main rule “all” (by default snakemake executes this rule), and all the modules which will be executed in the pipeline workflow. In this file you can define your pipeline adding or removing any module allowing a flexible workflow. The file config.json contains the paths names to access the files described in each rule. And, the variable.py file contains the path that should be created for the workflow structure results. This file also can define links to external tools, libraries and resources which could be called in any rule. And, it also can define global parameters and external parameters for external tools.

```

# Run PennCNV to detect CNVs
# 1. Input: PFB and GCmodel files
# 2. output: list of calls by sample in *.rawcn file
rule DETECT_CNVs:
    input:
        pbffile = config['pfb_file'],
        gcmodelfile = config['gcmodel_file']
    output:
        call_raw = expand(config['data_calling_path'] + '/' + '{prefix}' + '.rawcn', prefix=calling_prefix),
        call_log = expand(config['data_calling_path'] + '/' + '{prefix}' + '.log', prefix=calling_prefix)
    params:
        signalintensitydir = config['data_intensity_path'],
        outputdir = config['data_calling_path'],
        hmmfile = config['hmm_file'],
        listfile = config['list_signal_files_file'],
        logfile = config['log_path'] + '/' + 'data_calling.log',
    message: "Calling for CNV detection from {input} in {output}"

    shell:
        """
        #Calling from PennCNV command
        detect_cnv.pl \
            -test \
            -conf \
            -hmm {params.hmmfile} \
            -pfb {input.pbffile} \
            -directory {params.signalintensitydir} \
            -list {params.listfile} \
            -log {output.call_log} \
            -out {output.call_raw} \
            -gcmodel {input.gcmodelfile}

        #login
        samples_with_calls=`awk '{{print $5}}' {output.call_raw} | sort -u | wc -l`
        total_calls=`wc -l {output.call_raw} | cut -f1 -d' '`
        echo "Were found $samples_with_calls with $total_calls calls" >> {params.logfile}
        """

```

Figure 2. Module and rule structure. This figure shows a Module composed by one rule. Most commonly, rules consist of a name, input files, output files, and a shell command to generate the output from the input. This rule also contains the *params* specification which defines the alternative parameters used in this rule. Labels in red specify the path name for a file or directory. They are defined in the config.json file.

```

{
    "final_report_file": "/path_to/GSA2016_308_025_FinalReport.txt",
    "signal_intensity_file": "/path_to/signal_intensity.txt",
    "list_signal_files_file": "/path_to/data_conversion/list.txt",
    "map_file": "/path_to/data_conversion/sample_map.txt",
    "snp_file": "/path_to/data_conversion/SNPfile.txt",
    "pfb_file": "/path_to/data_conversion/addison.pfb",
    "gcmodel_file": "/path_to/data_conversion/hg19.gcmodel",
    "gc_content_file": "/path_to/resources/gc5Base.sorted.txt",
    "hmm_file": "/path_to/resources/hhall.hmm",

    "data_conversion_path": "/path_to/data_conversion",
    "data_intensity_path" : "/path_to/data_conversion/data_intensity",
    "data_calling_path": "/path_to/data_calling",
    "data_clean_path": "/path_to/data_clean",
    "graphic_path": "/path_to/graphic",
    "graphic_qc_path": "/path_to/graphic/qc",
    "log_path": "/path_to/logs",
    "data_dup_path": "/path_to/data_clean/DUP"
}

```

Figure 3. Config.json file. This file contains file and directory paths for input and output data. Any change in the file location just has to be modified in this file. Similarly, the output directory can be changed just by modifying this file. Changes are broadcast to all modules and rules.

```

### snakemake_workflows initialization #####
libdir = os.path.abspath(os.path.join(os.path.dirname(workflow.basedir), '../lib'))
resourcesdir = os.path.abspath(os.path.join(os.path.dirname(workflow.basedir), '../resources'))

### programs
plink = "plink"
shapeit = "path_to/shapeit.v2.904.3.10.0-693.11.6.el7.x86_64/bin/shapeit"
vcf_sort = "/path_to/vcftools_0.1.13/perl/vcf-sort"
bcftools = "/path_to/bcftools/bcftools-1.8/bin/bcftools"
### prefix
### module 1,2 and 3
signal_prefix = "split"
calling_prefix = "sampleall"
plink_prefix = 'study_raw'

### Workflow parameters #####
PLINK_EXT = ['.bed', '.bim', '.fam']
TPLINK_EXT = ['.tped', '.tfam']

### variables #####
### PennCNV #####
qcnumcnv = "50"
wf = "0.05"
qcbafdrift = "0.01"
qclrrsd = "0.3"

### Create paths if don't exist #####
### data_conversion and data_preparation modules ###
if not os.path.exists(config['log_path']):
    os.makedirs(config['log_path'])
if not os.path.exists(config['data_conversion_path']):
    os.makedirs(config['data_conversion_path'])
if not os.path.exists(config['data_intensity_path']):
    os.makedirs(config['data_intensity_path'])
if not os.path.exists(config['data_calling_path']):
    os.makedirs(config['data_calling_path'])
if not os.path.exists(config['data_clean_path']):
    os.makedirs(config['data_clean_path'])
if not os.path.exists(config['data_dup_path']):
    os.makedirs(config['data_dup_path'])
if not os.path.exists(config['graphic_path']):
    os.makedirs(config['graphic_path'])
if not os.path.exists(config['graphic_qc_path']):
    os.makedirs(config['graphic_qc_path'])

```

Figure 4. The variable.py file is a configuration file which contains the path structure created for the workflow to store results. This file also can define links to external tools, libraries and resources which could be called in any rule. And, it also can define global parameters and external parameters for external tools.

## Calling and quality control pipeline

### 1. Module Data Conversion

#### Rule CREATE\_INTENSITY\_FILES

**Description:** Create the individual signal intensity files, the list of the signals name per individual, and the list of SNPs for PennCNV calling. Also create the map file containing the original ID sample with the new name assigned by PennCNV (split\* when using the –numeric option to hide the original ID for sensitive data).

**Input:** signal intensity and Illumina report (Figure 5 and Figure 6).

**Output:** a file with the list of names for each intensity file created, the SNPs list file (name, Chr, Position), and the individual signal intensity (Figure 7).

Name	Chr	Position	Patient1.GType	Patient1.B Allele Freq	Patient1.Log R Ratio
1:100292476	1	100292476	AA	0,00	-0,3792592
1:101064936	1	101064936	AA	0,01713777	-0,1650909
1:103380393	1	103380393	BB	0,95952	-0,02254537
1:104303716	1	104303716	BB	0,9974257	-0,09440669
1:104864464	1	104864464	BB	1	-0,08369645

A log file is generated inside the logs directory with the module name (logs/data\_conversion.log) showing useful information for future reports.

Figure 5. Signal intensity file format. This figure shows the extraction of one patient intensity

```

GSGT Version      2.0.4
Processing Date    1/17/2020 11:48 AM
Content           GSAMD-24v1-0_20011747_A4.bpm
Num SNPs          700078
Total SNPs        700078
Num Samples       6112
Total Samples     6112
[Data]
SNP Name          Sample ID      B Allele Freq    Log R Ratio
1:100292476       Patient1      0.0000          -0.3793
1:101064936       Patien1      0.0171          -0.1651
1:103380393       Patient1      0.9595          -0.0225
1:104303716       Patient1      0.9974          -0.0944
1:104864464       Patient1      1.0000          -0.0837

```

signal for the first four markers.

Figure 6. Final report from Genome studio format

Name	ID.Log R Ratio	ID.B Allele Freq
1:100292476	-0.3793	0.0000
1:101064936	-0.1651	0.0171
1:103380393	-0.0225	0.9595
1:104303716	-0.0944	0.9974
1:104864464	-0.0837	1.0000

Figure 7. Individual signal intensity obtained by PennCNV

#### Rule CREATE PennCNV INPUT FILES

**Description:** Generate the Population frequency of B allele (PFB) file and the GCModel file used by PennCNV in the CNV detection.

**Input:** a file with the list of names for each intensity file created, and the SNPs list file (name, Chr, Position).

**Output:** the pfb file and the gc-model file.

A log file is generated inside the logs directory with the module name (logs/data\_conversion.log) showing useful information for future reports.

## 2. Module Data Calling

### Rule DETECT\_CNVS

**Description:** Run the PennCNV for CNVs detection

**Input:** the pfb file and the gc-model file.

**Output:** list of calls per individual in PennCNV \*.rawcn (Figure 8) and log files. This file shows CNV chromosome coordinates, number of SNPs contained in the CNV, length, type of variation (CN=0 or 1 means there is a deletion and CN>=3 means there is a duplication), the starting marker identifier and the ending marker identifier in the CNV, respectively.

A log file is generated inside the logs directory with the module name (logs/data\_calling.log) showing useful information for future reports.

```
chr1:149095346-149143879      numsn=4      length=48,534      state2,cn=1 split1 startsn=GSA-1:149095346 endsnp=GSA-rs79760750 conf=18.072
chr4:156963692-156966696      numsn=3      length=3,005      state2,cn=1 split1 startsn=rs4691238 endsnp=GSA-rs112975832 conf=12.029
chr5:97066021-97096042       numsn=5      length=30,022      state2,cn=1 split1 startsn=GSA-rs76728709 endsnp=rs4380692 conf=20.359
chr6:76261562-76265642       numsn=3      length=4,081      state2,cn=1 split1 startsn=rs61234544 endsnp=rs9360921 conf=16.661
chr11:95979334-95980389      numsn=3      length=1,056      state2,cn=1 split1 startsn=rs79686966 endsnp=rs535912 conf=14.694
chr14:106208082-106232585    numsn=3      length=24,504      state2,cn=1 split1 startsn=GSA-rs11621259 endsnp=B0T2-rs10136766 conf=8.561
chr22:42527793-42528976      numsn=4      length=1,184      state2,cn=1 split1 startsn=rs1080989 endsnp=rs28360521 conf=17.310
chr3:155481492-155505991     numsn=6      length=24,500      state5,cn=3 split1 startsn=chr3-155481492 endsnp=rs112074828 conf=16.732
chr7:141763387-141775383     numsn=10     length=11,997      state5,cn=3 split1 startsn=chr7-141763387 endsnp=GSA-rs77848363 conf=28.198
chr8:12009572-12009597       numsn=3      length=26          state5,cn=3 split1 startsn=GSA-rs9773610 endsnp=GSA-rs75619199 conf=10.326
```

Figure 8. PennCNV calling file rawcn format.

## 3. Module Data Clean

### Rule FILTER\_LOW\_QUALITY\_SAMPLES

**Description:** filter low quality samples based mainly on LRR and BAF statistics.

**Input:** rawcn and log file from PennCNV.

**Output:** rawcn file with good quality samples and their CNVs, file with the list of samples passed the QC, and summary file with all samples and their calls (passed and not passed the QC) include LRR and BAF statistics.

A plot showing the statistics LRR, BAF and WF distribution is generated. Also a log file is created inside the logs directory with the module name (logs/data\_clean.log) showing useful information for future reports.

### Rule REMOVE\_SPURIOUS\_CNVS

**Description:** Remove CNVs from spurious regions such as HLA, centromere and telomere.

**Input:** rawcn file with good quality samples and their CNVs.

**Output:** rawcn file with good quality samples and their CNVs not located on spurious regions  
A log file is generated inside the logs directory with the module name (logs/data\_clean.log) showing useful information for future reports.

### Rule MERGING\_ADJACENT\_CNVS

**Description:** Merge adjacent CNVcalls, in one single one, that could have been split from a large CNVs (>500kb) into smaller parts.

**Input:** rawcn file with good quality samples and their CNVs not located on spurious regions.

**Output:** merged CNVs in rawcn format. Violin plot showing the calls number distribution before and after the merging process is also generated.

A log file is generated inside the logs directory with the module name (logs/data\_clean.log) showing useful information for future reports.

## **Rare copy number variants (CNV) pipeline**

### **1. Module Data Conversion**

#### Rule CONVERT\_PENNCNV\_TO\_PLINK\_FORMAT

**Description:** convert rawcn files from PennCNV to Plink cnv, fam and map files. File with **all** CNVs previously detected at Module Data Calling in QC pipeline is converted in plink format. Information about number of calls, number of cases, controls, males and females can be extracted from these files. External function (function.sh) is referenced in this rule to execute external bash code.

**Input:** file with all calls (rawcn format) previously detected.

**Output:** cnv, fam and map files for all calls (Plink format).

A log file is created inside the logs directory with the module name (logs/data\_conversion.log) showing useful information for future reports.

#### Rule EXTRACT\_CORE\_SAMPLES

**Description:** extract CORE homogeneous unrelated samples, and their CNVs in Plink format files. Users should provide the ID list of unrelated samples (e.g. European descendant individuals). This rule could be modified according to the project requirements. External bash code, included in the function.sh file, is called in this rule.

**Input:** rawcn file with clean CNVs (passed QC and merged CNVs) and the list of unrelated European samples (IDs in a column).

**Output:** core samples and clean CNVs in cnv, fam and map plink files format.

A log file is created inside the logs directory with the module name (logs/data\_conversion.log) showing useful information for future reports.

#### Rule EXTRACT\_FINAL\_CNVS

**Description:** A set of clean big calls covered by a minimum of markers is extracted in this step. Both, length and number of markers, can be set at variables.py file (by default we use > 50 kb and > 5 markers). This rule could be adapted or removed according to the project requirements.

**Input:** rawcn file with core samples and clean calls (passed QC and merged CNVs)

**Output:** core samples and clean CNVs (passed QC and merged CNVs) bigger than 50 kb and covered by more than 5 markers, in cnv, fam and map plink files format. Individuals with 0 CNVs are also included in this final set.

A log file is created inside the logs directory with the module name (logs/data\_conversion.log) showing useful information for future reports.

## 2. Module Burden Analysis

### Rule BURDEN CNVs

**Description:** Perform the global burden test for CNVs in the core samples, cases vs. controls.

**Input:** CNVs for core samples in cnv, fam and cnv.map files.

**Output:** burden test result for CNVs in the core samples, showed at \*.summary.mperm and grp.summary files (Figure 9).

A log file is created inside the logs directory with the module name (logs/burden\_analysis.log) showing useful information for future reports.

Figure 9. Burden test output. This test reports four tests in both cases and controls, RATE (Number of segments), PROP (Proportion of sample with one or more segments), TOTKB (Total kb length spanned), and AVGKB (Average segment size). Tests are based (1-sided) on comparing these metrics in cases versus. For more detail see [https://zzz.bwh.harvard.edu/plink/cnv.shtml#write\\_cnvlist](https://zzz.bwh.harvard.edu/plink/cnv.shtml#write_cnvlist). In this example there is not apparent significant difference in any of the four test (EMP1 > 0.05)

```
*.cnv.grp.summary
TEST      GRP      AFF      UNAFF
N          ALL      2270     7728
RATE      ALL      1.92     1.927
PROP      ALL      0.8477   0.8536
TOTKB     ALL      362.6    356.8
AVGKB     ALL      160.5    157.2

*.cnv.summary.mperm
CHR      SNP      EMP1
S        RATE    0.562744
S        PROP    0.706829
S        KBTOT   0.341366
S        KBAVG   0.284572
S        GRATE    1
S        GPROP    1
S        GRICH    1
```

### Rule SPLIT CNVs DEL AND DUP

**Description:** Split CNVs in deletions and duplications and then obtain the frequency of deletions and duplications in cases vs. controls.

**Input:** CNVs for core samples in cnv, fam and cnv.map files.

**Output:** \*.cnv.indiv files for deletions and duplications in core samples. Also generate a tsv file containing the deletions and duplications ratio, split by length (by default 50kb, 100kb, 200kb, 500kb and 1Mb) which can be configured in variable.py file. Moreover, generate a tsv



file containing the total mean length (in KB) for deletions and duplications in cases and controls (Figure 10); and a plot showing the CNVs length-distribution in cases vs. controls (Figure 11).

A log file is created inside the logs directory with the module name (logs/burden\_analysis.log) showing useful information for future reports.

Deletion frequency TSV file			
pheno	length	numCNV	ratio
1	50KB	3502	0.873317
1	100KB	1667	0.415711
1	200KB	424	0.105736
1	500KB	83	0.0206983
1	1000KB	12	0.00299252
2	50KB	1027	0.868866
2	100KB	460	0.389171
2	200KB	132	0.111675
2	500KB	30	0.0253807
2	1000KB	13	0.0109983

Total mean length (in KB) TSV file			
Deletions			
pheno	tot_length_CNVs(Kb)	tot_CNVs	tot_mean_length_CNVs(Kb)
1	463500	3502	132.353
2	145182	1027	141.365

Duplications			
pheno	tot_length_CNVs(Kb)	tot_CNVs	tot_mean_length_CNVs(Kb)
1	757743	4226	179.305
2	218126	1243	175.484

Figure 10. TSV files generated in the module Burden Analysis. The first example shows the deletions frequency classified by length and phenotype (1:controls, 2:cases); and the second example shows the mean length for all deletions and duplications in controls and cases, 1 and 2 for pheno column respectively.



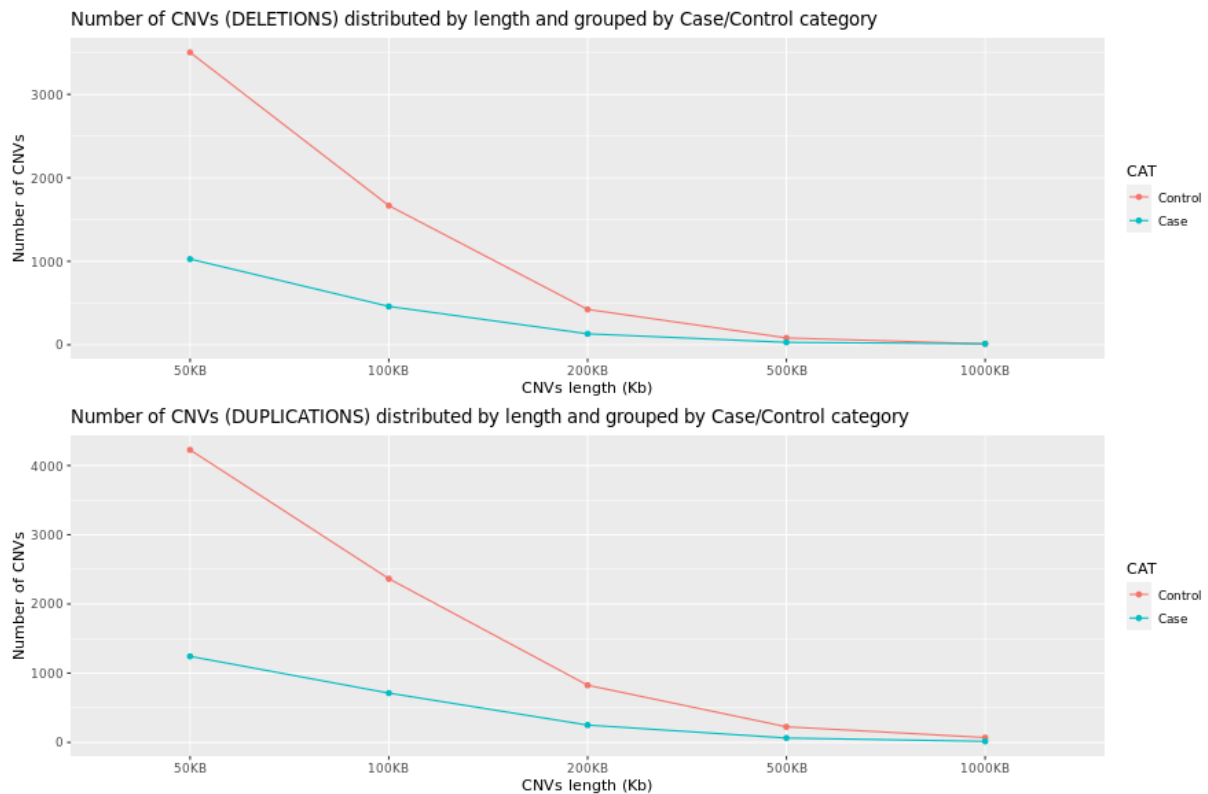


Figure 11. Length distribution in deletions and duplications. In this example CNVs are split in deletions and duplications and classified by cases and controls. Then they are distributed in 5 length intervals [50kb, 100kb, 200kb, 500kb, 1000kb]. This plot shows that bigger is the CNV length less is the number of CNVs in that interval, in cases and controls.

### 3. Module Rare CNVs Analysis

#### Rule GET\_REF\_CONTROLS\_COMMON\_CNVS

**Description:** Based on a subset of control samples, it obtains the **common** CNVs (deletions and duplications). The number of controls included in this subset can be defined in the config file `variables.py`. Common CNVs are defined as CNVs with high frequency, and this frequency also can be defined in `variables.py`.

**Input:** The ID list for the set of reference controls. These samples should be obtained randomly and including the same number of males and females (and any other population variable as nationality).

**Output:** \*.cnv, \*.fam and \*.cnv.map files for reference controls and **common variants** (high frequency) split by type: deletions and duplications

A log file is created inside the logs directory with the module name (`logs/rare_cnvs.log`) showing useful information for future reports.

#### Rule FILTER\_REFERENCE\_CONTROLS

**Description:** Remove the reference control samples and their CNVs, from the original control cohort, since this dataset has been used to extract the common CNVs.

**Input:** Reference controls ID list, \*.cnv, \*.fam and \*.cnv.map for common variants and reference controls.

**Output:** \*.cnv, \*.fam and \*.cnv.map files for the new set of samples and CNVs which exclude the reference controls.

A log file is created inside the logs directory with the module name (logs/rare\_cnvs.log) showing useful information for future reports.

### Rule GET\_RARE\_CNVS

**Description:** Extract rare CNVs, deletions and duplications. All CNVs overlapping at least 50% with common variants will be removed to retain just low frequency CNVs.

**Input:** \*.cnv, \*.fam, \*.map files for common variants and new dataset excluding reference controls.

**Output:** \*.cnv, \*.fam and \*.cnv.map files for rare deletions and duplications. Also BED files were created for a graphic visualization in the Genome Browser and any genome viewer as IGV (Figure 12)

A log file is created inside the logs directory with the module name (logs/rare\_cnvs.log) showing useful information for future reports.

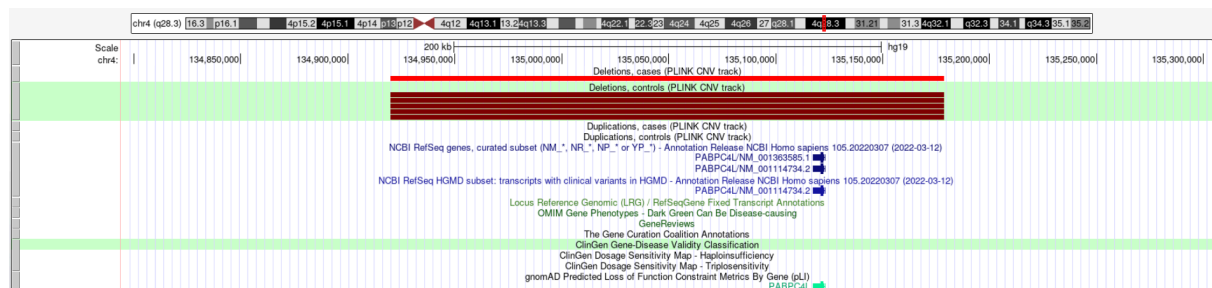


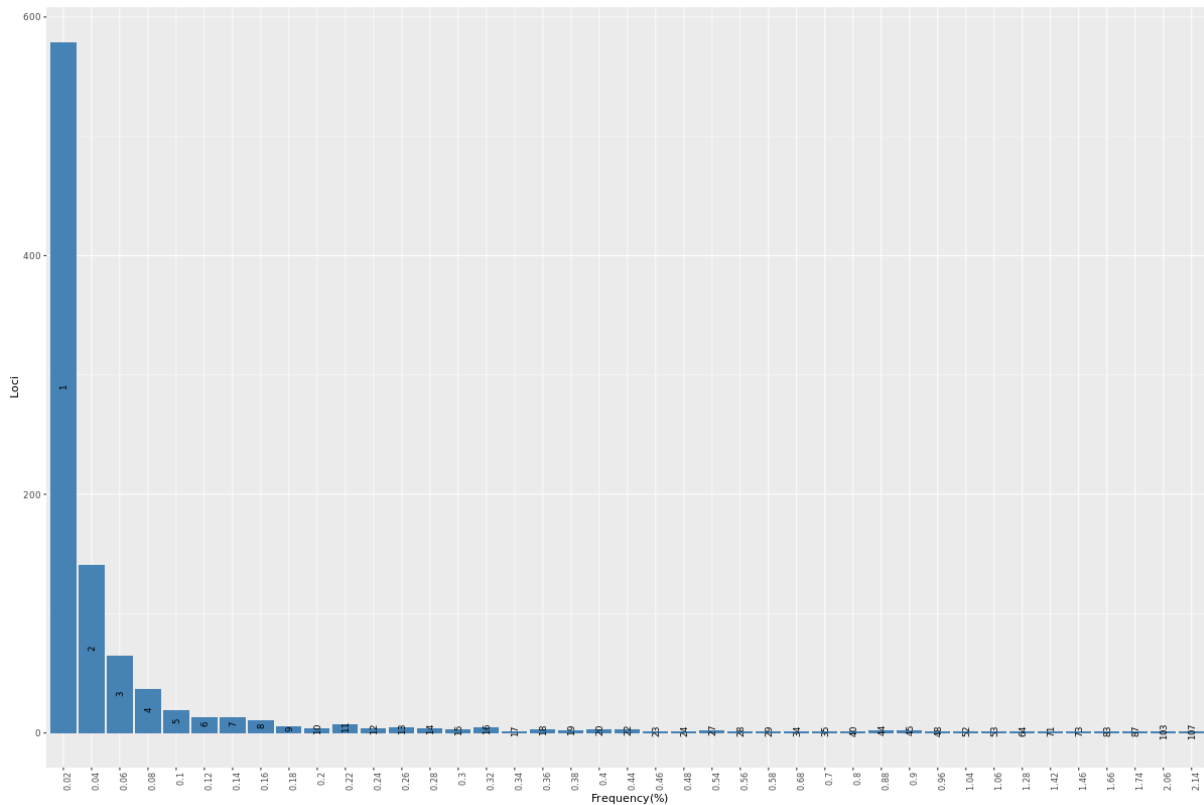
Figure 12. CNVs visualized in the Genome Browser. Deletions in cases and controls (light red versus dark red) visualized using the BED files.

### Rule FREQUENCY\_CNVS\_INSPECTION

**Description:** Inspect the frequency of rare CNVs, deletions and duplications. CNVs regions (CNVR) are defined to calculate the CNVs frequencies. After that, frequency is plotted to verify that frequencies are approximately below the frequency threshold set in the variable.py file (Figure 13).

**Input:** \*.cnv, \*.fam, \*.map files for rare deletions and duplications, and the fam file for all CNVs included in the study to calculate the frequency in each CNVR.

**Output:** \*.cnv, \*.fam and \*.cnv.map files for CNVR. Bar plot showing the CNVs frequency distribution.



## Rule FREQUENCY CNVs ASSESSMENT

**Input:** rare CNVs (deletions and duplications) \*.cnv, \*.fam and \*.cnv.map files.

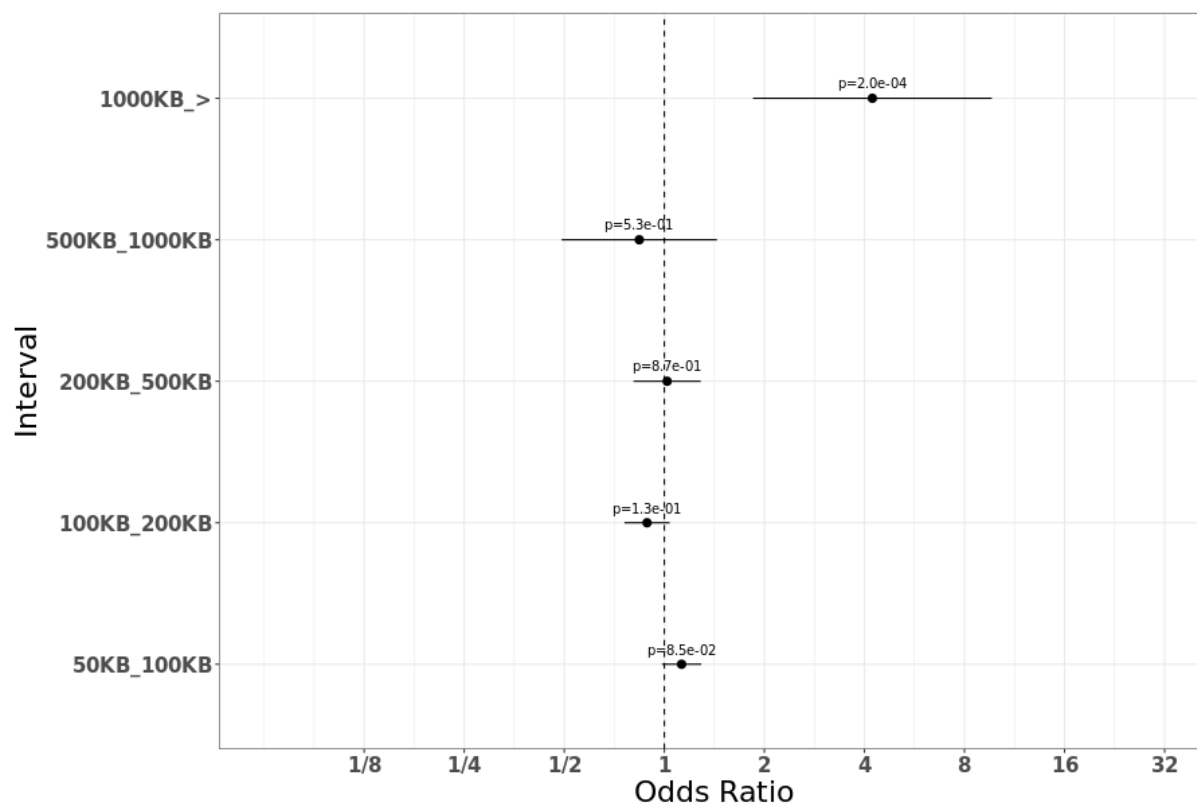


Figure 14. Deletions frequencies forest plot.

#### 4. Module Rare CNVs Enrichment

##### Rule GENIC\_ENRICHMENT\_ANALYSIS

**Description:** Perform the basic geneset-enrichment test to evaluate which genes are enriched for CNVs. This test is a permutation-based test with 10000 null permutations to generate empirical p-values. Also, Gene regions borders are set at 20 kb around each region. Both, permutations and regions border, can be adjusted inside the rule.

**Input:** rare deletions and duplications \*.cnv, \*.fam and \*.cnv.map files, and the gene annotation for the entire genome.

**Output:** \*.cnv.burden files with the Performing GLM-based CNV burden test results. Also, a text file containing the list of genes (chromosome, coordinates, number of CNVs in cases and controls) overlapped by any CNV is generated.

A log file is created inside the logs directory with the module name (logs/enrichment\_rare\_cnvs.log) showing useful information for future reports.

### Rule PATHWAY\_ENRICHMENT\_ANALYSIS

**Description:** Perform the basic geneset-enrichment test to evaluate whether a subset of genes are enriched, relative to the whole genome. This test is a permutation-based test with 10000 null permutations to generate empirical p-values. Also, Gene regions borders are set at 20 kb around each region. Both, permutations and regions border, can be adjusted inside the rule.

**Input:** rare deletions and duplications \*.cnv, \*.fam and \*.cnv.map files, and the gene annotation for the entire genome and file of gene names forming the pathway to be tested.

**Output:** \*.cnv.burden files with the Performing GLM-based CNV burden test results. Also, a text file containing the list of genes (chromosome, coordinates, number of CNVs in cases and controls) overlapped by any CNV is generated.

A log file is created inside the logs directory with the module name (logs/enrichment\_rare\_cnvs.log) showing useful information for future reports.

# Rare CNVs Analysis Pipeline

## Overview

This pipeline is a generic bioinformatic solution to identify rare CNVs in case-control based studies. Using SNPs-array genotyping data, this pipeline performs the CNV detection and quality control, followed by the burden analysis, the rare CNV frequency analysis and the enrichment CNV analysis (see workflow).

## Dependencies

Snakemake 5.22.1

Python 3.8.5

R 3.6.3

## Installation

### 1. Snakemake

Installation via Conda:

- conda install -n base -c conda-forge mamba
- mamba create -c conda-forge -c bioconda -n snakemake snakemake
- conda activate snakemake

Find other options in [https://snakemake.readthedocs.io/en/stable/getting\\_started/installation.html](https://snakemake.readthedocs.io/en/stable/getting_started/installation.html)

### 2. R dependencies

#Install all libraries used in this pipeline

# Package names

```
packages <- c("ggplot2", "fmsb", "gridExtra", "dplyr", "reshape", "introdaviz")
```

# Install packages not yet installed

```
installed_packages <- packages %in% rownames(installed.packages())
```

```
if (any(installed_packages == FALSE)) {  
  install.packages(packages[!installed_packages])  
}
```

## Pipeline execution

### • Detection calls and QC analysis:

```
$ snakemake -s qc-pipeline/snakefiles/qc.snake
```

### • Rare CNVs analysis:

```
$ snakemake -s association-pipeline/snakefiles/association.snake
```

### CNV detection and QC analysis output

```
./
../
data_calling/
data_clean/
data_conversion/
graphic/
logs/
```

### Rare CNVs analysis output

```
./
../
burden_analysis/
data_conversion/
enrichment_rare_cnvs/
graphics/
logs/
rare_cnvs/
```

## Pipeline structure

The pipeline consists of two major tasks: (1) quality control analysis, which uses the SNP-array genotyping data (green box) as an input to obtain good-quality samples and high-quality calls. (2) rare CNVs analysis, which takes samples and calls from the QC pipeline output, and after the data format conversion, performs the burden, rare CNVs and enrichment analysis. Black dotted lines split each analysis in their corresponding modules, purple boxes represent a specific task in each module, yellow boxes show representative outputs (files and/or plots), and the blue box represents external functions used by some modules. Dotted purple boxes are optional tasks which could be easily removed or changed to adapt the pipeline with the study requirements.

