

---

# **GC NGS PIPELINE**

***Release 1.0a1***

**Bernat del Olmo, Xavier Sumpsi, Carles Ferrer-Costa**

**Jul 24, 2022**



## CONTENTS:

<b>1</b>	<b>GC NGS PIPELINE</b>	<b>3</b>
1.1	Introduction . . . . .	3
1.2	Scope . . . . .	3
1.3	Lab integration . . . . .	4
1.4	Resource configuration: . . . . .	4
1.5	Sub-commands: . . . . .	5
1.6	Pipeline description: . . . . .	6
<b>2</b>	<b>modules</b>	<b>11</b>
2.1	annotate module . . . . .	11
2.2	bam module . . . . .	13
2.3	civic module . . . . .	16
2.4	cna module . . . . .	17
2.5	cnv_plot module . . . . .	17
2.6	fastq module . . . . .	18
2.7	gene_panel module . . . . .	19
2.8	genomic_plots module . . . . .	20
2.9	igv module . . . . .	21
2.10	lowpass module . . . . .	21
2.11	map module . . . . .	22
2.12	params module . . . . .	24
2.13	report module . . . . .	27
2.14	report2 module . . . . .	27
2.15	sample module . . . . .	28
2.16	sqlite module . . . . .	28
2.17	trimming module . . . . .	43
2.18	utils module . . . . .	44
2.19	var_call module . . . . .	45
2.20	vep_vcf module . . . . .	46
<b>3</b>	<b>Indices and tables</b>	<b>49</b>
	<b>Python Module Index</b>	<b>51</b>
	<b>Index</b>	<b>53</b>



**Navigator**

- *GC NGS PIPELINE*
  - *Introduction*
  - *Scope*
  - *Lab integration*
  - *Resource configuration:*
  - *Sub-commands:*
  - *Pipeline description:*
    - \* *Pre-processing*
    - \* *Mapping short reads to a genome reference*
    - \* *Removing PCR/Optical read duplicates*
    - \* *Germline SNV/Indel variant calling*
    - \* *Somatic SNV/Indel variant calling*
    - \* *Germline CNV detection on targeted sequencing*
    - \* *Somatic CNA (SCNA) detection on targeted sequencing*
    - \* *Structural Variant (SV) analysis*
    - \* *Annotation resources*
    - \* *NGS QC reports*



## GC NGS PIPELINE

### 1.1 Introduction

GC NGS PIPELINE is a command-line tool to handle DNA-seq analysis from Gencardio Diagnostics (GenDx).

This software should follow these non-functional requirements:

1. **Modular:** each step must be able to be executed independently through sub-commands.
2. **Performant:** the system must be able to process sequencing data and store all relevant information fast.
3. **Lab interaction:** the system must interact with other applications and databases from the lab.
4. **Data integrity:** the system should check and validate the user inputs.
5. **Maturity:** the system has to run tests every time that a change is made, and new tests have to be built for new features. Both unit and inter-module tests should be done.
6. **Changeability:** everything is very likely to change, so the system must be able to handle any kind of changes in features. Configuration is key here.

Note: This software has been developed and tested with Python3.6 under a Linux Mint 20 server.

### 1.2 Scope

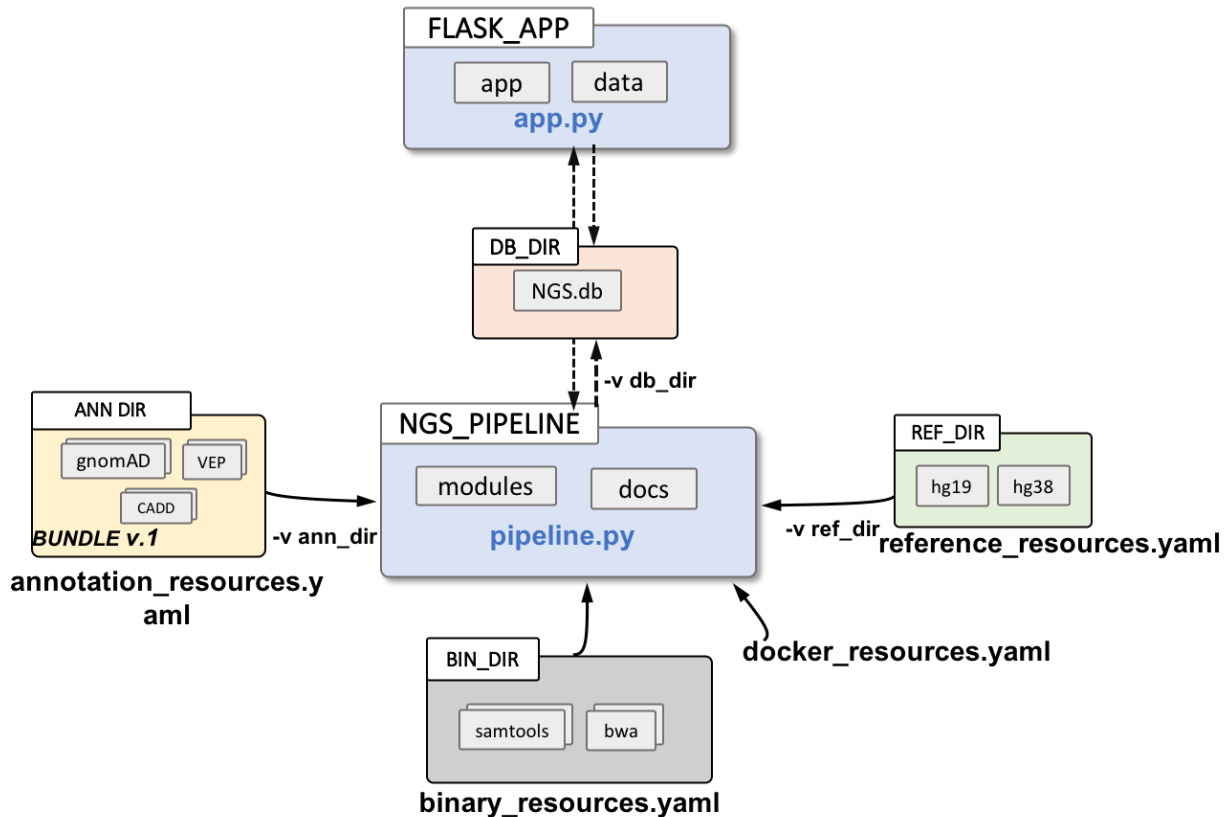
This software is intended to cover these NGS applications:

- Gene panel analysis:
  - Germline variants
  - Somatic variants
- Low-pass CNV detection.
- Exome sequencing analysis.
- Genome sequencing analysis.

The main steps of this pipeline can be executed independently (e.g. annotate an input VCF) or in its full extent (see all option)

## 1.3 Lab integration

NGS\_PIPELINE is being developed to interact with other services from our lab. This picture illustrates a first integration scheme with a flask app by sharing a common database structure that is used and fed bidirectionally.



## 1.4 Resource configuration:

Long-term development and maintenance of genomic pipelines is non-trivial since software and genomic annotation resources are evolving constantly. Therefore, we need a system that allows to rapidly configure new resources or modify existing versions. One way to keep track of these changes is by using YAML config files.

There are currently four external configuration files:

- `binary_resources.yaml` : software binaries/scripts and its versions.

Here are a few lines:

```
version: 1.0
bin_dir: /path/to/bin_directory
samtools:
  binary: samtools
  version: 1.7
```

- `reference_resources.yaml`: genome reference files (fasta, dicts, gene lists, chrom sizes).

Here are a few lines:



```

version: 1.0
ref_dir: /path/to/ref_directory
hg19:
  dirname: ""
  fasta: ucsc.hg19.fasta
  dict: ucsc.hg19.dict
  gene_bed: genelist.hg19.bed.gz
  chrom_sizes: hg19.txt

```

- `annotation_resources.yaml`: genomic annotations (bed, vcf, bigWig)

Here a few lines:

```

version: 1.0
ann_dir: /path/to/ann_directory
resources:
  gnomAD:
    version: "2.1.1"
    resource_name: gnomAD
  hg19:
    items:
      -1:
        key: GNOMAD_FILE
        file: gnomad.genomes.r2.1.1.sites.only_af.vcf.gz
      -2:
        key: GNOMAD_ONLY_AF_FILE
        file: somatic-b37_af-only-gnomad.raw.sites.chr.vcf.gz

```

- `docker_resources.yaml`: docker images

Here a few lines:

```

gatk:
  image: broadinstitute/gatk:4.1.3.0
  version: 4.1.3.0
picard:
  image: broadinstitute/picard
  version: ""

```

## 1.5 Sub-commands:

By prompting `python3 gc_nginx_pipeline.py` you will have access to the following sub-commands:

The usage information from each subcommand can be shown by with `-h` and `--help` options.

**all** Run all steps (map, call, annotate, report) successively

```

# Start from raw FASTQ files, run all NGS pipeline (map, call, annotate, report)
python3 gc_nginx_pipeline all -s <targeted,wgs,lowpass> \
  -r <hg19/hg38> -t <num_cpus> \
  -i input_fastq_dir/ -o output_dir/ \
  --lab_data <lab_xlsx> \
  --db_dir database_dir/ \

```

(continues on next page)

(continued from previous page)

```
--user_id <user_id> \  
--ann_yaml annotation_resources.yaml \  
--docker_yaml docker_resources.yaml \  
--ref_yaml reference_resources.yaml \  
--bin_yaml binary_resources.yaml
```

**map** Preprocess and map fastq files

```
# Map FASTQ files  
python3 gc_nginx_pipeline map -s <targeted,wgs,lowpass> \  
-r <hg19/hg38> -t <num_cpus> \  
-i input_fastq_dir/ -o output_dir/ \  
--lab_data <lab_xlsx> \  
--db_dir database_dir/ \  
--user_id <user_id> \  
--ann_yaml annotation_resources.yaml \  
--docker_yaml docker_resources.yaml \  
--ref_yaml reference_resources.yaml \  
--bin_yaml binary_resources.yaml
```

**call** Call somatic/germline variants

**annotate**

Enrich variants using various genome annotations

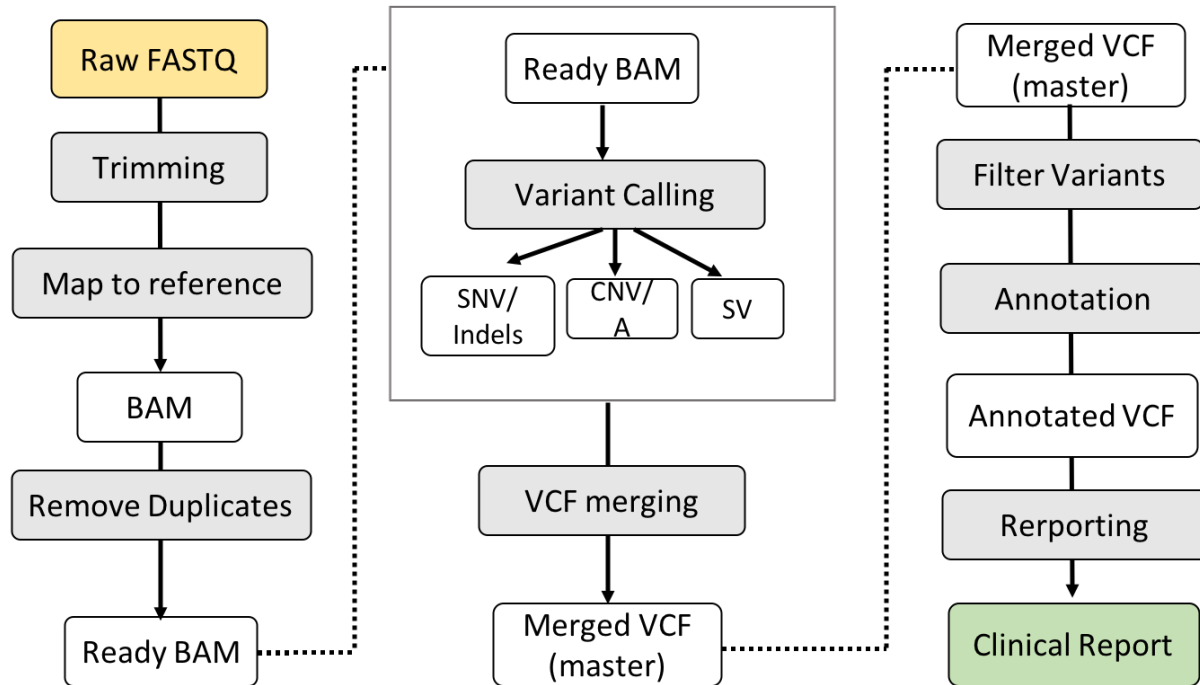
**report** NGS QC reports

TODO

**sarscov2** DEPRECATED. Sars-cov-2 analysis was added as an *ad-hoc* sub-command for convenience. Complete analysis starting from raw fastqs and ending with lineage classification

## 1.6 Pipeline description:

This part describes the main workflow for analyzing short-read sequences.



### 1.6.1 Pre-processing

- **Trimming:**

- **Low quality bases** towards 5' ends are often found on 2nd Gen NGS sequencers. Trimming them is needed specially when reads get longer than 76bp (google this: “300bp MiSeq” and check FastQC base-quality histogram)
- **Sequencing adapters** are short oligos that enable multiple samples to be sequenced together (using index sequences) and are also identified for the sequencer as valid fragments (P5, P7 sequences). These sequencers may be present at the raw FASTQ files whe the insert is smaller than the read length itself. Trimming them may also be convenient for downstream analysis

- **UMI processing:** Unique Molecular Identifiers (UMIs) allow to identify barcodes are short sequences used to uniquely tag each molecule in a sample library. UMIs are located within the adapter sequence.

The current software version uses [fastp](#) since it is widely used and it provides a solution for all the aforementioned requirements.

### 1.6.2 Mapping short reads to a genome reference

Short-read mapping/alignment is the core process of any NGS pipeline. This is because the sequencer produces FASTQ files without genomic context. Thus, we first need to locate them and assign a confidence (mapping quality) accordingly.

This process takes FASTQ files as input and produces Sequence Alignment Map (**SAM**) or its binary counterpart (BAM). The SAM/BAM format was established as an standard to handle alignment meta data.

The current software version uses **BWA-MEM** (Maximum-Exact Matches) as its main mapper due to its widespread usage and capabilities.

### 1.6.3 Removing PCR/Optical read duplicates

When preparing a DNA library for being sequenced, it is often desired to perform a PCR amplification once adapters have been ligated to ensure that enough DNA molecules are available. Thus, multiple copies of the original genomic DNA molecule are intentionally created producing some degree of PCR duplicates.

PCR polymerases have a low probability to introduce errors. However, sequencing errors in the first cycles of amplification will be magnified at the next steps. So, after sequencing these errors may be confounded as true variants. It is recommended to remove PCR duplicates on non-amplicon sequencing.

The current software version uses `Picard MarkDuplicates` for this purpose.

### 1.6.4 Germline SNV/Indel variant calling

While SNV calling is generally a solved problem (only on accessible regions), detecting short INDELs (1-20 bp) imposes additional challenges. INDELs tend to be accumulated on repeats, which are often enriched in sequencing errors. Some studies have revealed that erroneous INDELs are common in homopolymer A/T regions. Additionally, INDEL detection is more difficult in WES due to inherent capture biases.

Calling variants in germline mutations is quite different than calling somatic mutations. Germline mutations are expected to have a 50% or 100% allele frequencies; hence, variant calling is mainly aimed to choose the most likely genotype between AA, AB or BB.

Latest improvements on the field have been the incorporation of local assembly ([GATK HaplotypeCaller](#)), and the use of bayesian haplotyping strategies ([Freebayes](#), [Octopus](#)).

The current software version uses a combination of `GATK HaplotypeCaller` and `Octopus`.

### 1.6.5 Somatic SNV/Indel variant calling

Analyzing variants from tumor specimens has many challenges compared to germline variant analysis:

- Over time, FFPE-derived DNA may be downgraded in quality by artifacts such as deamination (C>T transitions).
- Tumour purity (proportion of tumoral cells with respect to the total cells) heavily influences the observed variant allele frequency (VAF).
- Subclonality, clones at different proportions.
- Copy number aberrations and Loss of Heterozygosity (LOH).
- A low DNA input that is translated in a high degree of PCR duplicates.

Some sophisticated approaches have been developed to overcome these limitations:

- [Mutect2](#) uses a probabilistic method to determine whether a variant is somatic, by comparing a tumor sample with a Panel of Normals (PoN)
- [Lancet](#) uses an enhanced de Bruijn local assembly procedure to improve indel detection.

UMIs can be used to reduce DNA polymerase errors by consensus base calling. The current software version uses a combination of `GATK Mutect2` for SNVs and `Lancet` for INDELs.

### 1.6.6 Germline CNV detection on targeted sequencing

CNV detection in targeted sequencing data (WES, gene panels) is challenging as the sparse nature of its sequencing usually is translated in the absence of breakpoint signatures. Consequently, most callers have been designed to detect CNVs using only RD signals, with few exceptions. Even when depth of coverage is higher in WES/targeted sequencing applications, there are much more fluctuations due to inconsistent capture efficiency among targeted regions.

- Detecting CNVs on clinical gene panels:

compared to Exomes, gene panel sequencing offers a higher read depth. This is translated in a higher resolution (up to a single-exon) for CNV analysis. There are currently a few tools that can be used for that purpose: [DECoN](#), [GRAPES](#) (from my thesis), [CoNVaDiNG](#), [panelcn.MOPS](#) are good examples.

However, CNV analysis on gene panels is in general difficult (lots of false-positives), It is recommended to filter calls using visual inspection and have an updated record of previous results to discard possible false positives.

- Detecting CNVs on whole-exome:

Whole-exome sequencing offers the possibility to detect CNVs typically larger than 10kb (multi-exon CNVs). Trying to detect single-exon CNVs may be not possible, since a lower coverage may introduce more fluctuations.

### 1.6.7 Somatic CNA (SCNA) detection on targeted sequencing

Similarly than with germline CNV detection, the available algorithm use relative read depth to derive CN status from a tumoral sample. Nonetheless, the cancer genome is more complex, and read depth alone is not enough to have an accurate genome picture. Other signals, such as copy-neutral loss of heterozygosity (CN-LOH), tumour admixture and potential contamination must be considered.

Ideally, with no budget limitations, the best experimental design would consider the sequencing of tumour-normal pairs. Alternatively, it is recommended to sequence a panel of normal samples (PoN).

Available tools that support PoN are: [GATK4](#) and [CNVKit](#).

The current software version uses a combination of [CNVkit](#) and a PoN of 10 FFPE samples.

### 1.6.8 Structural Variant (SV) analysis

In the past few years, the widespread usage of paired-end sequencing data has enabled the exploration of genomic SVs at a much finer extent than before. A key contribution can be attributed to the continuous development of more specialized algorithms, capable of taking advantage of sequencing improvements (e.g. read length). As a consequence, now we have a much accurate picture of SV size distribution and SV subtypes.

Best results are obtained from integrative callers that can use multiple signals (Read-Pairs, Split-Read, Read Depth and Assembly) Such a stepwise approach can result in an increase of specificity, but it cannot improve the number of true positives. Some integrative callers such as [Lumpy](#), [GRIDSS](#), and [Manta](#) can call SVs without requiring a pre-specified signal, so they are able to increase the overall sensitivity rate.

In Cancer, SVs are of spencial interest due to the existatnce of targeted therapies for some gene fusions such as ALK-EML4. Gene fusions are created by SVs.

The current software version uses a combination of [Manta](#) and [Lumpy](#).

### 1.6.9 Annotation resources

The Variant Call Format (VCF) has become the de facto format for describing variants. This format was originally created by the 1000 Genome Project because it was needed to establish a common format to annotate genomic locus. For a comprehensive description check: [Annovar description](#)

Integrating rich genomic annotations within VCFs is essential for downstream biological/clinical analysis. Some existing annotation engines such as Variant Effect Predictor (VEP), SnpEff or ANNOVAR provide gene-based annotation that can be leveraged afterwards.

This pipeline uses VEP extensively by either using its pre-defined annotation fields or by setting custom annotations (vcf, bed, bigWig) The available annotation resources are specified in the configuration file: `annotation_resources.yaml`

Find here a resource list that come with the `annotation bundle`:

Resource	Version
VEP	101
gnomAD	2.1.1
1000Genomes	20130502
dbNSFP	4.1
dbSNV	1.1
spliceAI	.
Blacklist	.
Phastcons	100way
PhyloP	100way
mappability	100mer
CGI	1
ncER	.
chimerKB	4

### 1.6.10 NGS QC reports

NGS analysis on the clinical field requires the delivery of custom-made genetic reports. We used historically JaspeReports (v6.15) as a background reporter. Today we make use of high quality html to pdf renderers such as WeasyPrint that enables to keep developing within the same python environment.

## 2.1 annotate module

`annotate.annotate_cancer_hotspots(sample_list, ann_conf)`

Annotate cancer hotspots: Single residue and in-frame indel mutation hotspots identified in 24,592 tumor samples by the algorithm described in [Chang et al. 2017] and [Chang et al. 2016]

**Parameters**

- **sample\_list** (*list*) – list of sample objects
- **bin\_conf** – BinaryConfig object
- **ann\_conf** – AnnotationConfig object

**Returns** list of sample objects

**Return type** list

`annotate.annotate_flanking_genes_to_sv(sample_list, bin_conf, ref_conf)`

This function appends flanking genes from SV calls.

**Parameters**

- **sample\_list** (*list*) – list of sample objects
- **bin\_conf** – BinaryConfig object
- **ref\_conf** – GenomeConfig object

**Returns** list of sample objects

**Return type** list

`annotate.annotate_known_fusions(sample_list, bin_conf, ann_conf)`

Annotate SVs using chimerKB known fusions. Create a new annotated VCF and json

**Parameters**

- **sample\_list** (*list*) – list of sample objects
- **bin\_conf** – BinaryConfig object
- **ann\_conf** – AnnotationConfig object

**Returns** list of sample objects

**Return type** list

`annotate.annotate_overlapping_genes_over_cnas(sample_list)`

`annotate.annotate_sv_with_vep(sample_list, analysis_conf, bin_conf, ref_conf, ann_conf, docker_conf)`

**Annotate raw VCF files with Ensembl's Variant Effect Predictor (VEP).** Create a new annotated vcf and json.

**Parameters**

- **sample\_list** (*list*) – list of sample objects
- **analysis\_conf** – AnalysisConfig object
- **bin\_conf** – BinaryConfig object
- **ref\_conf** – GenomeConfig object
- **ann\_conf** – AnnotationConfig object
- **docker\_conf** – DockerConfig object

**Returns** list of sample objects

**Return type** list

`annotate.annotate_with_vep(sample_list, analysis_conf, bin_conf, ref_conf, ann_conf, docker_conf)`

**Annotate raw VCF files with Ensembl's Variant Effect Predictor (VEP).** Create a new annotated vcf and json.

**Parameters**

- **sample\_list** (*list*) – list of sample objects
- **analysis\_conf** – AnalysisConfig object
- **bin\_conf** – BinaryConfig object
- **ref\_conf** – GenomeConfig object
- **ann\_conf** – AnnotationConfig object
- **docker\_conf** – DockerConfig object

**Returns** list of sample objects

**Return type** list

`annotate.classify_in_somatic_tiers(sample_list, bin_conf, ann_conf)`

Classification based on CIViC of somatic variants in three categories: 1. Drug sensitive variants 2. High impact variants with no CIViC evidence records (stop codons,

frameshifts, missense variants with high REVEL scores, splicing variants)

3. Other rare variants (intronic, intergenic, synonymous)

**Parameters**

- **sample\_list** (*list*) – list of sample objects
- **bin\_conf** – BinaryConfig object
- **ann\_conf** – AnnotationConfig object

**Returns** list of sample objects

**Return type** list



`annotate.launch_annotation(args, sample_list)`

Main annotation function

**Parameters**

- **args** – pipeline command line arguments
- **sample\_list** (*list*) – list of sample objects

**Returns** list of sample objects

**Return type** list

`annotate.merge_cnv_sv(sample_list, bin_conf)`

`annotate.merge_vcfs(sample_list, bin_conf)`

Merge VCFs (snv, CNV and SV) with bcftools. Create a new annotated vcf and json. :param list sample\_list: list of sample objects :param bin\_conf: BinaryConfig object :returns: list of sample objects :rtype: list

`annotate.select_analysis_genes_and_isoforms(sample_list, analysis_conf)`

**Output a new VCF with a single isoform as follows:**

- Dump a configured isoform if available through Gene Panel API
- Otherwise return canonical transcript from Ensembl

**Parameters** **sample\_list** (*list*) – list of sample objects

**Returns** list of sample objects

**Return type** list

## 2.2 bam module

`class bam.Bam(bam, bin_config, docker_config)`

Bases: object

Class to handle Bam operations such as bam indexing, pcr duplicate removal, and qc metrics extraction.

Class instantiation automatically checks for bam consistency, header and read group availability

**Parameters**

- **bam** (*str*) – bam file
- **bin\_config** – binary configuration object
- **docker\_config** – docker configuration object

`quick_check()` `index()` `remove_duplicates()` `get_perc_duplicates()` `get_total_reads()`  
`get_read_length()` `get_mean_isize()` `get_sd_isize()` `get_coverage_metrics()`

**property contigs:** list

**Getter** returns a list of contigs

**Variables** **\_header** (*str*) – bam header

**Return type** list(dict)

**get\_coverage\_metrics**(*panel\_name=None, bed=None, output\_dir=None*)

Calculate coverage metrics with Mosdepth

**Parameters**

- **bed** (*str*) – optional bed file when dealing with targeted sequencing applications
- **ouput\_dir** (*str*) – output directory to write coverage files

**Returns** mosdepth coverage file, coverage dictionary

**Return type** str

**Return type** dict

**get\_mapping\_metrics**()

**get\_mean\_isize**(*N=5000*)

Get the mean insert size

**Parameters** **N** (*int*) – use up to N alignments to compute the mean insert size in basepairs

**Variables** **\_bin\_config.samtools.binary** (*str*) – samtools binary

**Returns** mean insert size

**Return type** float

**get\_perc\_duplicates**()

Return the percentage of PCR/Optical duplicates :var str \_sample\_name: sample name :var str \_picard\_metrics\_file: metrics file created using picard :raises FileNotFoundError: when picard\_metrics\_file is not found :rtype: float

**get\_read\_length**(*N=5000*)

Get the mean read length

**Parameters** **N** (*int*) – use up to N alignments to compute the mena read length

**Variables** **\_bin\_config.samtools.binary** (*str*) – samtools binary

**Returns** mean read length

**Return type** int

**get\_sd\_isize**(*N=5000*)

Get the insert size standard deviation

**Parameters** **N** (*int*) – use up to N alignments to compute the std. deviation insert size. Default N = 5000

**Variables** **\_bin\_config.samtools.binary** (*str*) – samtools binary

**Raises** **ValueError** – N must be an integer value > 0

**Returns** insert size std. deviation

**Return type** float

**get\_total\_reads**(*bed=None*)

Get the total number of reads

**Parameters** **bed** (*str*) – optional bed file to restrict calculation over ROIs

**Variables** **\_sample\_name** (*str*) – sample name

**Returns** total number of reads

**Return type** int

**property header:** str

**Getter** return header from bam file

**Variables**

- **\_bam** (str) – input bam file
- **\_bin\_config.samtools.binary** (str) – samtools binary

**Return type** str

**index**(input\_bam=None) → str

Create a bam index

**Variables**

- **\_bam** (str) – input bam file
- **\_bin\_config.samtools.binary** (str) – samtools binary

**Returns** bai file

**Return type** str

**quick\_check**() → bool

Checks bam integrity

**Variables**

- **\_bam** (str) – input bam file
- **\_bin\_config.samtools.binary** (str) – samtools binary

**Returns** True if bam is correct, otherwise return False

**Return type** bool

**remove\_duplicates**(output\_dir=None)

Remove PCR/Optical duplicates

**Variables**

- **\_bam** (str) – input bam file
- **\_docker\_config.docker** (str) – gatk docker image

**Returns** rmdup bam file, pct\_duplicates,

**Return type** str

**Return type**

**property sample\_name:** str

**Getter** sample name from bam RG (Read group). If not found return bam prefix as sample name

**Variables**

- **\_bam** (str) – input bam file
- **\_bin\_config.samtools.binary** (str) – samtools binary

**Raises**

- **ValueError** – when bam header is not found

- **ValueError** – when RG is not available

**Return type** str

## 2.3 civic module

**class** civic.Civic(*civic\_dir*)

Bases: object

**download\_latest\_release()**

**property evidences**

**property genes**

**get\_evidence()**

**get\_variants()** → {}

**property latest\_release**

**query\_cnv**(*gene*, *cnvtype*)

**query\_fusion**(*gene1*, *gene2*)

**query\_variant**(*gene*, *variant*, *vartype*, *exon*, *conseq*)

**property variants**

**class** civic.CivicRestAPI

Bases: object

CIViC class This needs a big refactoring

**property accessed\_on**

This function returns the date in which CIViC was accessed

**check\_rest\_service()**

**load\_civic()**

load civic data in memory

**query\_variant**(*gene*, *variant*, *vartype*, *exon*, *conseq*)

return list of evidences given a gene, variant, exon and conseq

**civic.annotate\_cna\_civic**(*sample\_list*, *ann\_conf*, *analysis\_conf*, *bin\_conf*)

Annotate somatic CNAs using CIViC. Create a new annotated vcf and json.

**Parameters:**

**analysis\_env** [dict] Global dictionary that stores analysis parameters

**sample\_env** [dict] Global dictionary that stores sample data

**Returns:** Nothing

**civic.annotate\_snv\_civic**(*sample\_list*, *ann\_conf*, *dump\_json=True*)  
 Annotate somatic SNV/Indels using CIViC. Create a new annotated vcf and json.

**Parameters:**

**analysis\_env** [dict] Global dictionary that stores analysis parameters  
**sample\_env** [dict] Global dictionary that stores sample data

**Returns:** *sample\_list*

## 2.4 cna module

**class cna.CnvKit**(*docker\_conf*, *sample\_name*, *tumour\_bam*, *normals\_cnn*, *tumour\_purity*, *bed*, *genome*, *threads*, *output\_dir*)

Bases: object

**batch()**

Do all steps.

**call\_cnv()**

**property cns\_calls\_file**

**create\_access\_regions()**

**create\_target\_regions**(*do\_target=False*, *do\_antitarget=False*)

**export\_bed()**

**export\_seg()**

Export a SEG file (which is compatible with DNACopy, PureCN and others)

**export\_vcf()**

**extract\_coverage()**

**fix()**

**property ratio\_file**

**segment()**

CNV segmentation using CSB as default

## 2.5 cnv\_plot module

**class cnv\_plot.CnvPlot**(*cnr\_file*, *cns\_file*, *calls*, *sample*, *output\_dir*, *ref\_conf*)

Bases: object

Class for plotting cnvs

**plot\_cnv**(*chr*, *start*, *end*, *gene\_list=None*, *add\_genes=False*, *offset=None*)

Plot a CNV call, add gene labels (optionally)

**plot\_genomewide**(*genomewide*, *by\_chr*)

**cnv\_plot.remove\_bed\_header**(*file*, *pattern*)

## 2.6 fastq module

**class** fastq.**Fastq**(fq, expect\_paired=True, force\_naming\_convention=True)

Bases: object

Fastq class. This class automatically validates fastq nomenclature and file consistency

**Parameters** **fq** (str) – fastq file

[check\\_consistency\(\)](#) [mean\\_read\\_length\(\)](#) [check\\_nomenclature\(\)](#)

**check\_consistency()** → bool

Check that fastq1 and fastq2 are consistent:

- Check that SEQ and QUAL have equal lengths
- Check that fastq1 and fastq2 have equal read number

**Parameters**

- **fastq1** (str) – fastq1 file to be checked
- **fastq2** (str) – fastq2 file to be checked

**Returns** returns True if no issues were detected, otherwise False

**Return type** bool

**check\_nomenclature()** → bool

Validate a fastq file:

- Check FASTQ valid file extension (fq, fasta, fq.gz, fasta.gz)
- Check that names are equal between fq1 and fq2,
- Check illumina's nomenclature (\_S[0-9]+\_L[0-9]+\_R[12]\_[0-9]+)

If paired:

**Parameters**

- **self.fq1** – fastq1 to be checked
- **self.fq2** – fastq2 to be checked

**Type** str

**Type** str

If single-ended:

**Parameters** **self.fq** – fastq to be checked

**Type** str

**Returns** True if fastq(s) follow the Illumina's nomenclature

**Return type** bool

**property fq1:** str

**Getter** Returns fastq1

```

property fq1_basename: str
    Getter Returns fastq1 basename or prefix
property fq2: str
    Getter Returns fastq2
property fq2_basename: str
    Getter Returns fastq2 basename or prefix
property mean_read_length: int
    Calculate the mean read length in base pairs
    Parameters fastq – fastq file to be checked
    Type str
    Returns returns the mean read length
    Return type int
property sample_name: str
    Getter sample name from fastq prefix
    Variables fq (str) – input fastq file
    Raises ValueError – if fastq name cannot be splitted by “_”
    Return type str
exception fastq.FastqNotFound
    Bases: Exception
exception fastq.InvalidFastqFile(fq)
    Bases: Exception
    Custom exception to track FASTQ invalid files :param str fq: fastq file
exception fastq.InvalidFastqNomenclature(fq)
    Bases: Exception
    Custom exception to track FASTQ invalid nomenclature :param str fq: fastq file
exception fastq.MissingFastqPair
    Bases: Exception

```

## 2.7 gene\_panel module

```

class gene_panel.GenePanel(panel_name)
    Bases: object
    disclaimer_db
        alias of modules.sqlite.Disclaimer
    property disclaimers
    get_roi_info(chr, start, end)

```

**is\_registered()** → bool

Function that checks if the panel is registered on the system Returns True if the panel is registered, otherwise returns False

**property is\_subpanel**

**panel\_content\_db**

alias of modules.sqlite.PanelContent

**panel\_db**

alias of modules.sqlite.Panel

**panel\_isoforms**

alias of modules.sqlite.PanelIsoforms

**property transcripts**

Get gene-transcript relationship for the specified panel using a dict where the primary key refers to a gene id and the secondary key to the selected transcript ID.

**property version**

**class** gene\_panel.**GenePanelAPI**(*panel\_name, panel\_version=None*)

Bases: object

**property analysis\_genes:** dict

**property biomarkers:** dict

**property cna:** dict

**property disclaimers:** dict

**is\_registered()** → bool

**property latest\_version:** str

**property qc\_criteria:** dict

## 2.8 genomic\_plots module

**class** genomic\_plots.**IGV**(*igv\_exe, sample\_name, bam, bed, snapshot\_dir*)

Bases: object

IGV class to handle variant imaging

### Parameters

- **igv\_exe** (*str*) – igv executable
- **sample\_name** (*str*) – sample name
- **bam** (*str*) – input bam file
- **bed** (*str*) – ROI bed file
- **snapshot\_dir** (*str*) – output directory where snapshots will be created

*create\_batch\_script()* *create\_snapshots()*



**create\_batch\_script**(*img, squish=True, collapse=False*)

Create a batch script (.bat) needed to create multiple images.

**Parameters**

- **img** (*str*) – picture format
- **squish** (*bool*) – squish tracks
- **collapse** (*bool*) – collapse alignments to fit vertically

**create\_snapshots**()

Create IGV snapshots using a configured batch script

**Variables**

- **\_igv\_exe** (*str*) – picture format
- **\_batch\_file** (*str*) – picture format

**class** genomic\_plots.**SimpleBamSnap**(*bam, fasta, docker\_conf, chr, start, end, output\_dir, output\_name*)

Bases: object

**generate\_view**()

**load\_bam**()

## 2.9 igv module

### 2.10 lowpass module

**lowpass.annotate\_gc**(*input\_bed, analysis\_conf, bin\_conf, ref\_conf, ann\_conf*)

Add gc content, return a dict with gc content

**lowpass.annotate\_mappability**(*input\_bed, analysis\_conf, bin\_conf, ref\_conf, ann\_conf*)

**lowpass.assign\_genotype\_based\_on\_cn**(*cn*)

Simple genotype assignment based on copy number

**lowpass.bed\_to\_vcf**(*sample\_list, analysis\_conf, bin\_conf, ref\_conf, docker\_conf*)

Export \*acgm.bed to vcf

**lowpass.calculate\_ratios**(*sample\_list, merged\_df*)

Calculate bin ratio

**lowpass.call\_cnvs**(*sample\_list, del\_threshold, dup\_threshold, z\_score*)

Calling CNVs

**lowpass.classify\_cnv**(*sample\_list, analysis\_conf, bin\_conf*)

**lowpass.create\_bins**(*analysis\_conf, bin\_conf, ann\_conf, ref\_conf, bin\_size*)

Create a BED file with genomewide bins

**lowpass.do\_lowpass**(*args*)

**lowpass.do\_ratio\_ref**(*row, baseline, sample*)

`lowpass.do_ratio_same(row, median_sample, sample)`

`lowpass.extract_coverage(sample_list, lowpass_bins, analysis_conf, bin_conf, gc_dict, map_dict)`  
Coverage extraction with megadept

`lowpass.merge_samples_coverage(sample_list, analysis_conf, lowpass_bins)`  
Plotting coverage normalization

`lowpass.normalize(df, sample, median_cov, fields)`

`lowpass.plot_normalization(df, sample, output_dir)`  
Plotting coverage normalization

`lowpass.ratios_to_json(ratio_file, sample_name, output_folder)`  
Convert a ratio file (.bed) to json

`lowpass.remove_bed_header(file, pattern)`

`lowpass.segment_coverage(sample_list, n_segments, alpha)`  
segment with CBS

`lowpass.update_lowpass_db(sample_list, analysis_conf)`

`lowpass.write_vcf_header_template_for_cnv(sample_name, bam, analysis_conf, bin_conf, ref_conf, docker_conf)`  
Create a vcf template. This is required for cyvcf2

## 2.11 map module

`class map.BWA(sample_name, fq1, fq2, ref, outdir, docker_config)`

Bases: `map.BaseMapper`

BWA-MEM mapper class

### Parameters

- **sample\_name** (*str*) – sample name extracted from fastq
- **fq1** (*str*) – raw fastq1 (R1)
- **fq2** (*str*) – raw fastq2 (R2)
- **ref** (*str*) – reference genome in fasta format
- **outdir** (*str*) – output directory
- **bin\_config** – configuration object for binaries

`align(num_cpu=4, sort=True)`

Align with BWA-MEM

### Parameters

- **num\_cpu** (*int*) – number of CPU's
- **sort** (*bool*) – sort output BAM by coordinates

**class** map.**BaseMapper**(*sample\_name, fq1, fq2, ref, outdir, docker\_config*)

Bases: object

Parent class to handle read alignment processes. This class checks that input fastq files are valid

#### Parameters

- **sample\_name** (*str*) – sample name extracted from fastq
- **fq1** (*str*) – raw fastq1 (R1)
- **fq2** (*str*) – raw fastq2 (R2)
- **ref** (*str*) – reference genome in fasta format
- **outdir** (*str*) – output directory
- **bin\_config** – configuration object for binaries

map.**create\_list\_file**()

DEPRECATED Generates a GATK-compatible list file from a BED file. TODO documentation

# :param args: pipeline command line arguments # :param list sample\_list: list of sample objects # :returns: list of sample objects # :rtype: list

map.**create\_summary\_qc\_xlsx**(*sample\_list, analysis\_conf*)

Writes an xlsx summarizing QC metrics. This file outlines the following NGS metrics:

- Total number of reads (in Millions)
- Mean coverage
- Call rate (%) at different coverage thresholds (1X, 10X, 20X, 30X, 100X, 200X)
- Lost exons at different coverage thresholds (1X, 10X, 20X, 30X, 100X, 200X)
- Enrichment (%)
- PCR/Optical duplicates (%)
- Mean and std. deviation insert size values (bp)

#### Parameters

- **sample\_list** (*list*) – list of sample objects
- **analysis\_conf** ([AnalysisConfig](#)) – list of sample objects

**Return str** a summary file in TSV format

map.**launch\_mapping**(*args, sample\_list*)

Main function for mapping, duplicate removal and qc metric extraction

#### Parameters

- **args** – pipeline command line arguments
- **sample\_list** (*list*) – list of sample objects

**Returns** list of sample objects

**Return type** list

`map.read_summary_qc_xlsx(sample_list, analysis_config)`

Read from summary\_qc tsv file to get coverage and alignment metrics

**Parameters**

- **sample\_list** (*list*) – list of sample objects
- **analysis\_conf** (*AnalysisConfig*) – list of sample objects

**Return list sample\_list** list of sample objects

`map.summarize_sample_qc(sample, qc_folder, raw_bam_file, rmdup_bam_file, roi_bed, bin_conf, docker_conf)`

From a bam, calculate coverage metrics using Mosdepth and return a summary dict and a bed file :param Sample sample: Sample object :param str qc\_folder: list of sample objects :param str bam\_file: list of sample objects :param str roi\_bed: list of sample objects :param BinaryConfig bin\_conf: configuration object for binaries :param DockerConfig docker\_conf: configuration object for docker images

**Returns** summary\_qc\_dict

**Return type** dict

## 2.12 params module

`class params.AnalysisConfig(args)`

Bases: *params.BaseConfig*

Analysis configuration class that inherits a basic configuration object

**property analysis\_date:** str

**property ann\_dir:** str

**property ann\_yaml:** str

**property bin\_yaml:** str

Returns from arg param or from defaults

**property cgi:** bool

**property civic:** bool

**property cnvkit:** bool

**property command:** str

**property conservation:** str

**property create\_snapshots:** bool

**property date\_time:** str

**property db\_dir:** str

**property docker\_yaml:** str

**property freebayes:** bool

**property gatk:** bool

```

property genome_version: str
property gnomad: bool
property input_dir: str
property lab_data: str
property lancet: bool
property manta: bool
property output_dir: str
property output_name: str
property panel_dirname: str
property panel_full_path: str
property panel_list_full_path: str
property panel_list_name: str
property panel_name: str
property ref_dir: str
property ref_yaml: str
property report_language: str
property rm_dup: bool
property seq_analysis: str
    Seq analysis can be: targeted, wgs, lowpass, sarscov2
property thousand_genomes: bool
property threads: int
property user_id: str
validate() → bool
property variant_analysis: str

class params.AnnotationConfig(data)
    Bases: params.BaseConfig
    Configuration for annotation resources
    validate(dump_messages=True)

class params.BaseConfig
    Bases: object
    Base config class that sets default parameters

```

```
class params.BinaryConfig(data)
```

Bases: [params.BaseConfig](#)

Configuration for binary utilities.

```
static get_bin_path(self, program)
```

Get the PATH of a program

```
get_software_versions(as_dict=True) → bool
```

```
validate(dump_messages=True)
```

```
class params.DockerConfig(data, args)
```

Bases: [params.BaseConfig](#)

Class that sets docker image

```
property docker
```

```
get_bin_path(program)
```

Get the PATH of a program

```
validate(dump_messages=True)
```

simple check for docker images installed

```
class params.GenomeConfig(data, args)
```

Bases: [params.BaseConfig](#)

Class that sets genome configuration

```
property genome_dir
```

```
property genome_fasta
```

```
validate(dump_messages=True)
```

```
params.get_panel_configuration(analysis_conf)
```

Get panel configuration :param analysis\_conf: Analysis configuration object

```
params.load_configuration(args, dump_messages=False)
```

Setting analysis parameters: analysis mode, IO directories, number of cpus, analysis date, database directory, annotation resource directory, reference sequence directory, among others. Input parameters are validated here.

```
params.set_labdata_env(analysis_conf)
```

TEMPORARY, we need a petition manager to integrate all data from lab and external Lab data (docx) and sample\_data (xlsx) loading. These two files are required to get code relationship (Lab, AP, HC codes) and others like tumour purity estimates,

```
params.set_logging(args)
```

## 2.13 report module

`report.add_clinical_variant(session, tables_dict, clinical_class, sample, record)`

Adds a new clinical variant into the main database

`report.annotate_biomarkers(sample_list, bin_conf)`

Annotate depth information to biomarker positions

`report.igv_snapshots(sample_list, analysis_conf, bin_conf)`

Create a report for somatic variants

`report.launch_report(args, sample_list)`

Main reporting function

### Parameters

- **args** – pipeline command line arguments
- **sample\_list** (*list*) – list of sample objects

**Returns** list of sample objects

**Return type** list

`report.somatic_report(sample_list, analysis_conf, bin_conf, ann_conf, docker_conf)`

Create a report for somatic variants

`report.update_cnas(sample, analysis_conf, cna_table, session, cna_bed)`

`report.update_global_biomarkers(sample, analysis_conf)`

`report.update_global_cnas(sample, analysis_conf, cna_bed)`

`report.update_global_fusions(sample, analysis_conf)`

Record all Fusion calls from MANTA

`report.update_global_lost_exons(sample, analysis_conf)`

`report.update_global_somatic_db(sample_list, analysis_conf, bin_conf, ann_conf, docker_conf)`

`report.update_global_summary_qc(sample, analysis_conf)`

## 2.14 report2 module

`report2.annotate_biomarkers(sample_list, bin_conf)`

Annotate depth information to biomarker positions

`report2.beautify_info(value: str) → str`

**Parameters** **str** (*info*) – input value to be beautified

**Return info** **str** beautified string

`report2.generate_genomic_snapshots(sample_list, analysis_conf, ref_conf, docker_conf)`

Create a report for somatic variants

`report2.germline_report(sample_list, analysis_conf)`

Create a report for germline variants

`report2.launch_report2(args, sample_list)`

Main reporting function

**Parameters**

- **args** – pipeline command line arguments
- **sample\_list** (*list*) – list of sample objects

**Returns** list of sample objects

**Return type** list

`report2.somatic_report(sample_list, analysis_conf)`

Create a report for somatic variants (March '22)

`report2.somatic_report2(sample_list, analysis_conf)`

New version adding tier classifications (July '22)

`report2.spliceai_effect(csq: dict) → str`

`report2.update_global_biomarkers(sample, analysis_conf)`

`report2.update_global_cnas(sample, analysis_conf, cna_bed)`

`report2.update_global_fusions(sample, analysis_conf)`

Record all Fusion calls from MANTA

`report2.update_global_lost_exons(sample, analysis_conf)`

`report2.update_global_somatic_db(sample_list, analysis_conf, bin_conf, ann_conf, docker_conf)`

`report2.update_global_summary_qc(sample, analysis_conf)`

## 2.15 sample module

`class sample.Sample(name)`

Bases: object

**add**(key, value)

Add a new attribute dinamically

**property name**

## 2.16 sqlite module

`class sqlite.AllCnas(user_id, lab_id, ext1_id, ext2_id, run_id, chromosome, start, end, genes, svtype, ratio, qual, cn)`

Bases: sqlalchemy.orm.decl\_api.Model

**chromosome**

**cn**

**end**



ext1\_id

ext2\_id

genes

id

lab\_id

qual

ratio

run\_id

start

svtype

user\_id

```
class sqlite.AllFusions(user_id, lab_id, ext1_id, ext2_id, run_id, chromosome, start, end, qual, svtype,  
                        read_pairs, split_reads, vaf, depth, fusion_partners, fusion_source, fusion_diseases,  
                        flanking_genes)
```

Bases: sqlalchemy.orm.decl\_api.Model

chromosome

depth

end

ext1\_id

ext2\_id

flanking\_genes

fusion\_diseases

fusion\_partners

fusion\_source

id

lab\_id

qual

read\_pairs

run\_id

split\_reads

start

svtype

```
    user_id
    vaf
class sqlite.Biomarker(ID, gene, variant, exon, chr, pos, end, panel)
    Bases: sqlalchemy.orm.decl_api.Model
    chr
    end
    exon
    gene
    id
    panel
    pos
    variant
class sqlite.BiomarkerTable(**kwargs)
    Bases: sqlalchemy.orm.decl_api.Model
    chr
    depth
    end
    exon
    ext1_id
    ext2_id
    gene
    id
    lab_id
    panel
    pos
    run_id
    user_id
    vaf
    variant
class sqlite.Cna(chromosome, start, end, gene, genome_version, panel_name, panel_version,
                 dump_therapeutic, min_cn)
    Bases: sqlalchemy.orm.decl_api.Model
```

```
chromosome
dump_therapeutic
end
gene
genome_version
id
min_cn
panel_name
panel_version
start

class sqlite.Disclaimer(id, panel, genes, methodology, analysis, lab_confirmation, technical_limitations,
                        legal_provisions, language)
    Bases: sqlalchemy.orm.decl_api.Model
    analysis
    genes
    id
    lab_confirmation
    language
    legal_provisions
    methodology
    panel
    technical_limitations

class sqlite.Job(User_id, Job_id, Queue_id, Date, Analysis, Panel, Samples, Status)
    Bases: sqlalchemy.orm.decl_api.Model
    Analysis
    Date
    Id
    Job_id
    Panel
    Queue_id
    Samples
    Status
```

```
User_id
class sqlite.LostExonsTable(**kwargs)
    Bases: sqlalchemy.orm.decl_api.Model
    call_rate_100X
    call_rate_10X
    call_rate_1X
    call_rate_200X
    call_rate_20X
    call_rate_30X
    chromosome
    end
    ensg_id
    enst_id
    exon_number
    ext1_id
    ext2_id
    gene
    id
    lab_id
    probe_group
    run_id
    start
    user_id

class sqlite.LowpassCnv(**kwargs)
    Bases: sqlalchemy.orm.decl_api.Model
    acmg_classification
    acmg_keywords
    acmg_score
    acmg_version
    chromosome
    cn
    dosage_sensitive_genes
```

```
end
ext1_id
ext2_id
fold_change
fold_change_zscore
genotype
id
lab_classification
lab_classification_date
lab_confirmation
lab_confirmation_technique
lab_id
log_fold_change
protein_coding_genes
run_id
start
svlen
svtype
user_id
vcf_json
class sqlite.OtherVariantsTable(**kwargs)
    Bases: sqlalchemy.orm.decl_api.Model
    allele_frequency
    blacklist
    classification
    clinical_trials
    consequence
    db_detected_freq
    db_detected_number
    db_sample_count
    depth
```

```
    enst_id
    exon
    ext1_id
    ext2_id
    gene
    hgvc
    hgvcg
    hgvsp
    id
    intron
    lab_id
    max_af
    max_af_pop
    petition_id
    read_support
    run_id
    therapies
    tier_catsalut
    tumor_type
    user_id
    validated_assessor
    validated_bioinfo
    var_json
    variant_type
class sqlite.Panel(**kwargs)
    Bases: sqlalchemy.orm.decl_api.Model
    Id
    call_rate_filter
    call_rate_perc
    enrichment_perc_filter
    genome_version
```

```
language
last_modified
lost_exons_filter
lost_exons_perc
panel
panel_bed
read_num_filter
size
total_genes
total_rois
variant_call
version

class sqlite.PanelContent(**kwargs)
    Bases: sqlalchemy.orm.decl_api.Model
    chromosome
    end
    ense_id
    ensg_id
    ensp_id
    enst_id
    exon_end
    exon_number
    exon_start
    feature
    gene_name
    genome_version
    id
    mane_select
    panel
    panel_version
    probe_group
```

```
    refseq_id
    start
    strand

class sqlite.PanelIsoforms(ID, CHROMOSOME, START, END, ENSG_ID, ENST_ID, GENE_NAME,
                           GENOME_VERSION, PANEL, PANEL_VERSION)
    Bases: sqlalchemy.orm.decl_api.Model
    chromosome
    end
    ensg_id
    enst_id
    gene_name
    genome_version
    id
    panel
    panel_version
    start

class sqlite.Petition(Petition_id, User_id, Date, AP_code, HC_code, CIP_code, Tumour_pct, Volume,
                       Conc_nanodrop, Ratio_nanodrop, Tape_postevaluation, Medical_doctor, Billing_unit,
                       Medical_indication, Date_original_biopsy)
    Bases: sqlalchemy.orm.decl_api.Model
    AP_code
    Billing_unit
    CIP_code
    Conc_nanodrop
    Date
    Date_original_biopsy
    HC_code
    Id
    Medical_doctor
    Medical_indication
    Petition_id
    Ratio_nanodrop
    Tape_postevaluation
```



```
Tumour_pct
User_id
Volume
class sqlite.PipelineDetailsTable(**kwargs)
    Bases: sqlalchemy.orm.decl_api.Model
    bwa_version
    cgi_version
    chimerkb_version
    civic_version
    cnvkit_version
    dbnsfp_version
    dbscsnv_version
    fastp_version
    gatk_version
    genome_version
    gnomad_version
    id
    manta_version
    pipeline_version
    run_id
    samtools_version
    thousand_genomes_version
    vep_version
class sqlite.RareVariantsTable(**kwargs)
    Bases: sqlalchemy.orm.decl_api.Model
    allele_frequency
    blacklist
    classification
    clinical_trials
    consequence
    db_detected_freq
    db_detected_number
```

```
db_sample_count
depth
enst_id
exon
ext1_id
ext2_id
gene
hgvsc
hgvs_g
hgvsp
id
intron
lab_id
max_af
max_af_pop
petition_id
read_support
run_id
therapies
tier_catsalut
tumor_type
user_id
validated_assessor
validated_bioinfo
var_json
variant_type

class sqlite.SampleTable(**kwargs)
    Bases: sqlalchemy.orm.decl_api.Model
    analysis_date
    bam
    cnv_json
```

```
date_original_biopsy
diagnosis
ext1_id
ext2_id
ext3_id
extraction_date
id
lab_id
latest_report_pdf
medical_address
medical_center
merged_vcf
panel
petition_id
physician_name
report_db
report_pdf
roi_bed
run_id
sample_db_dir
sample_type
sex
software
software_version
subpanel
tumour_purity
user_id

class sqlite.SampleVariants(**kwargs)
    Bases: sqlalchemy.orm.decl_api.Model
    ann_id
    ann_json
```

```
ann_key
classification
confirmation_technique
id
lab_confirmation
sample_id
var_id
class sqlite.SummaryQcTable(**kwargs)
    Bases: sqlalchemy.orm.decl_api.Model
    ext1_id
    ext2_id
    fastp_json
    id
    lab_id
    petition_id
    run_id
    summary_json
    user_id
class sqlite.TherapeuticTable(**kwargs)
    Bases: sqlalchemy.orm.decl_api.Model
    allele_frequency
    blacklist
    classification
    clinical_trials
    consequence
    db_detected_freq
    db_detected_number
    db_sample_count
    depth
    enst_id
    exon
    ext1_id
```

```
    ext2_id
    gene
    hgvs_c
    hgvs_g
    hgvs_p
    id
    intron
    lab_id
    max_af
    max_af_pop
    petition_id
    read_support
    run_id
    therapies
    tier_catsalut
    tumor_type
    user_id
    validated_assessor
    validated_bioinfo
    var_json
    variant_type

class sqlite.VarAnnotation(**kwargs)
    Bases: sqlalchemy.orm.decl_api.Model
    ann_json
    ann_key
    id
    var_id

class sqlite.Variants(**kwargs)
    Bases: sqlalchemy.orm.decl_api.Model
    alt
    blacklist
    chromosome
```

**count**  
**gene**  
**genome\_version**  
**hgvsc**  
**hgvsg**  
**hgvsp**  
**id**  
**isoform**  
**pos**  
**ref**  
**var\_type**

`sqlite.init(analysis_conf)`

`sqlite.init_ngs_database(database: str)`

`sqlite.load_cna(pname)`

This function requests the available configured CNAs from the CNA database Parameters:

Panel name (str)

**Returns:** dict with the requested CNAs

`sqlite.load_panel_biomarkers(pname)`

This function requests the available configured biomarkers from the biomarker database Parameters:

Panel name (str)

**Returns:** dict with the requested biomarkers

`sqlite.load_panel_disclaimers(pname, lang)`

This function requests the available disclaimers Parameters:

Panel name (str) language (str)

**Returns:** dict with the requested disclaimers

`sqlite.load_panel_transcripts(pname)`

This function requests all transcripts from the ROI database Parameters:

Panel name (str)

**Returns:** dict with the requested transcripts

`sqlite.load_petitions()`

This function requests all petitions from the petition database

**Parameters:** Petition class

**Returns:** list of all petitions

`sqlite.update_summary_db(sample_list, analysis_conf)`

This function adds sample-wise qc metrics into the summary\_qc database

**Parameters:**

**analysis\_env** [dict] Global dictionary that stores analysis parameters

**lab\_data** [dict] Global dictionary that stores lab sample data

**sample\_env** [dict] Global dictionary that stores sample data

**Returns:** Nothing

## 2.17 trimming module

`trimming.fastp(sample_name, output_dir, fq1, fq2, threads, fastp_exe)`

Trim raw FASTQ files using fastp

**Parameters**

- **sample\_name** (*str*) – sample name extracted from fastq
- **output\_dir** (*str*) – output directory
- **fq1** (*str*) – raw fastq1 (R1)
- **fq2** (*str*) – raw fastq2 (R2)
- **threads** (*str*) – number of cpu cores
- **fastp\_exe** (*str*) – fastp binary location

**Returns** trimmed\_fq1, trimmed\_fq2

**Return type** tuple

`trimming.launch_trimming(analysis_conf, bin_conf)`

Trim raw FASTQ files

**Parameters**

- **analysis\_conf** – analysis configuration object
- **bin\_conf** – binary (third-party software) configuration object

**Type** *AnalysisConfig*

**Type** *BinaryConfig*

**Returns** list of sample objects

**Return type** list

## 2.18 utils module

`utils.check_docker_images(command, image)`  
simple check for docker images installed

`utils.compress_vcf(input_vcf, bgzip_exe)`  
Compress a plain text vcf

`utils.convert_vcf_2_json(vcf)`

`utils.convert_vcf_to_dict(vcf)`

`utils.create_vep_dict(info)`

`utils.decompress_vcf(input_vcf, gunzip_exe)`  
Decompress a bgzipped vcf file

`utils.get_bin_path(program)`  
Get the PATH of a program

`utils.get_fastq_files(input_dir, avoid_trimmed=False)`  
Get fastq files from input directory

`utils.get_input_files(input_dir, file_type)`  
Get all files (fastq, bam, vcf) from an input directory

`utils.index_vcf(input_vcf: str, tabix_exe: str) → None`  
Index a bgzipped vcf

`utils.log2_fold_change(num) → float`  
Convert a numeric value to log2

`utils.long_aminoacid_2_short(long_aa: str) → str`

`utils.long_hgvsp_2_short(long_hgvsp: str) → str`

`utils.mean_depth_coordinate(bam, coordinate, samtools_exe)`

`utils.num_to_human(num)`  
Convert huge numbers to human readable. Return in Million units

`utils.prepare_samples_from_bams(input_dir, output_dir) → []`  
Create sample objects from an input directory with bam files

`utils.prepare_samples_from_vcfs(input_dir, output_dir) → []`  
Create sample objects from an input directory with vcf files

`utils.vcf_2_bed(vcf)`

`utils.yaml_to_dict(yaml_file) → str`  
Take a valid yaml file and return a dict. Sanity check



## 2.19 var\_call module

`var_call.append_manta_small_indels(sample_list: List[modules.sample.Sample], bin_conf: modules.params.BinaryConfig) → List[modules.sample.Sample]`

`var_call.cnvkit_log2_json(cnr_file, cnv_vcf, sample, outdir)`

`var_call.combine_mutect2_lancet(sample_list, only_lancet_indels=True)`

Take a list of VCF files and a list of caller names (e.g Mutect2, Lancet) and merges variants according to the following conditions: - SNVs are combined using all callers (Union). Mutect2 info is reported. - Only Indels from Lancet are considered if lancet is used.

`var_call.filter_by_orientation_bias(sample_list, analysis_conf, bin_conf, ref_conf, docker_conf)`

Filter VCF by orientation bias. This kind of bias is typically observed on low quality degraded FFPE samples.

`var_call.get_mutect2_indels_bed(sample_list, offset=50)`

Create a BED file with Indel coordinates to be refined.

`var_call.launch_variant_calling(args, sample_list)`

Main function for deploying variant calling steps :param args: pipeline command line arguments :param list sample\_list: list of sample objects :returns: list of sample objects :rtype: list

`var_call.run_cnvkit(sample_list, analysis_conf, bin_conf, ref_conf, docker_conf)`

Run CNV analysis with CNVkit :param list sample\_list: list of sample objects :param AnalysisConfig analysis\_conf: analysis configuration object :param BinaryConfig bin\_conf: binary configuration object :param GenomeConfig ref\_conf: genome configuration object :param DockerConfig docker\_conf: docker configuration object :returns: list of sample objects :rtype: list

`var_call.run_decon(sample_list, analysis_conf, bin_conf, ref_conf, docker_conf)`

Run germline CNV analysis with DECoN :param list sample\_list: list of sample objects :param AnalysisConfig analysis\_conf: analysis configuration object :param BinaryConfig bin\_conf: binary configuration object :param GenomeConfig ref\_conf: genome configuration object :param DockerConfig docker\_conf: docker configuration object :returns: list of sample objects :rtype: list

`var_call.run_freebayes(sample_list, bin_conf, ref_conf, ann_conf)`

`var_call.run_gatk_hc(sample_list, analysis_conf, bin_conf, ref_conf, ann_conf, docker_conf)`

Call variants with GATK's HaplotypeCaller

`var_call.run_grapes(sample_list, analysis_conf, bin_conf, ref_conf, docker_conf)`

`var_call.run_lancet(sample_list, analysis_conf, bin_conf, docker_conf, ref_conf, ann_conf)`

Refine indels using Lancet. We observed that complex indels are not well resolved by Mutect2.

`var_call.run_manta(sample_list, analysis_conf, bin_conf, ref_conf, germline=True, somatic=False)`

`var_call.run_mutect2(sample_list, analysis_conf, bin_conf, ref_conf, ann_conf, docker_conf)`

`var_call.run_octopus(sample_list, analysis_conf, bin_conf, ref_conf)`

Call variants with Octopus

## 2.20 vep\_vcf module

```
class vep_vcf.CSQ
    Bases: object
class vep_vcf.Record(variant_str: str, header=None)
    Bases: object
    Variant record representation from a vcf
        Parameters variant_str – VCF entry as string
        Type str
    property ALT: str
    property CHROM: str
    property END
    property FILTER: str
    property FORMAT: []
    property GT: list
    property ID: str
    property INFO: dict
    property POS: str
    property QUAL: str
    property REF: str
    property SAMPLE: str
    add_info_field(key: str, value: str, custom: bool)
    as_dict() → {}
    as_string() → str
    property civic_items
    property clinical_directions
    property clinical_significance
    property clinical_trials
    property clinical_trials_str
    property copy_number
    property depth
    property discordant_reads
```

```

property diseases
property diseases_str
property fold_change
property genotype: str
get_clinical_trials()
get_diseases()
get_therapeutic_drugs()
get_tier_classification(source)
is_cna() → bool
    Return True if the variant is an insertion
is_deletion() → bool
    Return True if the variant is a deletion (not SVTYPE)
is_insertion() → bool
    Return True if the variant is an insertion
is_mnv() → bool
    Return True if the variant is an SNV
property is_rare: bool
is_snv() → bool
    Return True if the variant is an SNV
is_sv() → bool
    Return True if the variant is an insertion
property len_alt: int
property len_ref: int
read_info() → dict
property read_support: int
set_info_subfield_from_list(key, value_list)
set_info_subfield_from_str(key, value_str)
property split_reads
property svend
property svlen
property therapeutic_drugs
property therapeutic_drugs_str
property vaf

```

```
property vartype: str
    Return variant type (SNV, Deletion, Insertion, MNV, SV, CNA)

class vep_vcf.VCFreader(vcf_in: str)
    Bases: object

    add_info_to_header(info_dict: dict) → str

    fetch(chr: str, pos: Optional[int] = None, end: Optional[int] = None) → vep_vcf.Record
        Fetch variants from a chr/region

    property header: str

    static is_gz(vcf: str) → bool

class vep_vcf.VCFwriter(vcf_out, template=None, compress=False, sample_name=None)
    Bases: object
    VCF writer class

    add_info_to_header(info_dict)

    close()

    property header: str

    static is_gz(vcf) → bool

    write(record)
```

## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### a

[annotate](#), 11

### b

[bam](#), 13

### c

[civic](#), 16

[cna](#), 17

[cnv\\_plot](#), 17

### f

[fastq](#), 18

### g

[gene\\_panel](#), 19

[genomic\\_plots](#), 20

### l

[lowpass](#), 21

### m

[map](#), 22

### p

[params](#), 24

### r

[report](#), 27

[report2](#), 27

### s

[sample](#), 28

[sqlite](#), 28

### t

[trimming](#), 43

### u

[utils](#), 44

### v

[var\\_call](#), 45

[vep\\_vcf](#), 46





## A

[accessed\\_on](#) (*civic.CivicRestAPI* property), 16  
[acmg\\_classification](#) (*sqlite.LowpassCnv* attribute), 32  
[acmg\\_keywords](#) (*sqlite.LowpassCnv* attribute), 32  
[acmg\\_score](#) (*sqlite.LowpassCnv* attribute), 32  
[acmg\\_version](#) (*sqlite.LowpassCnv* attribute), 32  
[add\(\)](#) (*sample.Sample* method), 28  
[add\\_clinical\\_variant\(\)](#) (*in module report*), 27  
[add\\_info\\_field\(\)](#) (*vep\_vcf.Record* method), 46  
[add\\_info\\_to\\_header\(\)](#) (*vep\_vcf.VCFreader* method), 48  
[add\\_info\\_to\\_header\(\)](#) (*vep\_vcf.VCFwriter* method), 48  
[align\(\)](#) (*map.BWA* method), 22  
[AllCnas](#) (*class in sqlite*), 28  
[allele\\_frequency](#) (*sqlite.OtherVariantsTable* attribute), 33  
[allele\\_frequency](#) (*sqlite.RareVariantsTable* attribute), 37  
[allele\\_frequency](#) (*sqlite.TherapeuticTable* attribute), 40  
[AllFusions](#) (*class in sqlite*), 29  
[alt](#) (*sqlite.Variants* attribute), 41  
[ALT](#) (*vep\_vcf.Record* property), 46  
[analysis](#) (*sqlite.Disclaimer* attribute), 31  
[Analysis](#) (*sqlite.Job* attribute), 31  
[analysis\\_date](#) (*params.AnalysisConfig* property), 24  
[analysis\\_date](#) (*sqlite.SampleTable* attribute), 38  
[analysis\\_genes](#) (*gene\_panel.GenePanelAPI* property), 20  
[AnalysisConfig](#) (*class in params*), 24  
[ann\\_dir](#) (*params.AnalysisConfig* property), 24  
[ann\\_id](#) (*sqlite.SampleVariants* attribute), 39  
[ann\\_json](#) (*sqlite.SampleVariants* attribute), 39  
[ann\\_json](#) (*sqlite.VarAnnotation* attribute), 41  
[ann\\_key](#) (*sqlite.SampleVariants* attribute), 39  
[ann\\_key](#) (*sqlite.VarAnnotation* attribute), 41  
[ann\\_yaml](#) (*params.AnalysisConfig* property), 24  
[annotate](#)  
     *module*, 11  
[annotate\\_biomarkers\(\)](#) (*in module report*), 27

[annotate\\_biomarkers\(\)](#) (*in module report2*), 27  
[annotate\\_cancer\\_hotspots\(\)](#) (*in module annotate*), 11  
[annotate\\_cna\\_civic\(\)](#) (*in module civic*), 16  
[annotate\\_flanking\\_genes\\_to\\_sv\(\)](#) (*in module annotate*), 11  
[annotate\\_gc\(\)](#) (*in module lowpass*), 21  
[annotate\\_known\\_fusions\(\)](#) (*in module annotate*), 11  
[annotate\\_mappability\(\)](#) (*in module lowpass*), 21  
[annotate\\_overlapping\\_genes\\_over\\_cnas\(\)](#) (*in module annotate*), 11  
[annotate\\_snv\\_civic\(\)](#) (*in module civic*), 16  
[annotate\\_sv\\_with\\_vep\(\)](#) (*in module annotate*), 11  
[annotate\\_with\\_vep\(\)](#) (*in module annotate*), 12  
[AnnotationConfig](#) (*class in params*), 25  
[AP\\_code](#) (*sqlite.Petition* attribute), 36  
[append\\_manta\\_small\\_indels\(\)](#) (*in module var\_call*), 45  
[as\\_dict\(\)](#) (*vep\_vcf.Record* method), 46  
[as\\_string\(\)](#) (*vep\_vcf.Record* method), 46  
[assign\\_genotype\\_based\\_on\\_cn\(\)](#) (*in module lowpass*), 21

## B

[bam](#)  
     *module*, 13  
[Bam](#) (*class in bam*), 13  
[bam](#) (*sqlite.SampleTable* attribute), 38  
[BaseConfig](#) (*class in params*), 25  
[BaseMapper](#) (*class in map*), 22  
[batch\(\)](#) (*cna.CnvKit* method), 17  
[beautify\\_info\(\)](#) (*in module report2*), 27  
[bed\\_to\\_vcf\(\)](#) (*in module lowpass*), 21  
[Billing\\_unit](#) (*sqlite.Petition* attribute), 36  
[bin\\_yaml](#) (*params.AnalysisConfig* property), 24  
[BinaryConfig](#) (*class in params*), 25  
[Biomarker](#) (*class in sqlite*), 30  
[biomarkers](#) (*gene\_panel.GenePanelAPI* property), 20  
[BiomarkerTable](#) (*class in sqlite*), 30  
[blacklist](#) (*sqlite.OtherVariantsTable* attribute), 33  
[blacklist](#) (*sqlite.RareVariantsTable* attribute), 37  
[blacklist](#) (*sqlite.TherapeuticTable* attribute), 40

blacklist (*sqlite.Variants* attribute), 41  
BWA (class in *map*), 22  
bwa\_version (*sqlite.PipelineDetailsTable* attribute), 37

## C

calculate\_ratios() (in module *lowpass*), 21  
call\_cnv() (*cna.CnvKit* method), 17  
call\_cnvs() (in module *lowpass*), 21  
call\_rate\_100X (*sqlite.LostExonsTable* attribute), 32  
call\_rate\_10X (*sqlite.LostExonsTable* attribute), 32  
call\_rate\_1X (*sqlite.LostExonsTable* attribute), 32  
call\_rate\_200X (*sqlite.LostExonsTable* attribute), 32  
call\_rate\_20X (*sqlite.LostExonsTable* attribute), 32  
call\_rate\_30X (*sqlite.LostExonsTable* attribute), 32  
call\_rate\_filter (*sqlite.Panel* attribute), 34  
call\_rate\_perc (*sqlite.Panel* attribute), 34  
cgi (*params.AnalysisConfig* property), 24  
cgi\_version (*sqlite.PipelineDetailsTable* attribute), 37  
check\_consistency() (*fastq.Fastq* method), 18  
check\_docker\_images() (in module *utils*), 44  
check\_nomenclature() (*fastq.Fastq* method), 18  
check\_rest\_service() (*civic.CivicRestAPI* method), 16  
chimerkb\_version (*sqlite.PipelineDetailsTable* attribute), 37  
chr (*sqlite.Biomarker* attribute), 30  
chr (*sqlite.BiomarkerTable* attribute), 30  
CHROM (*vep\_vcf.Record* property), 46  
chromosome (*sqlite.AllCnas* attribute), 28  
chromosome (*sqlite.AllFusions* attribute), 29  
chromosome (*sqlite.Cna* attribute), 30  
chromosome (*sqlite.LostExonsTable* attribute), 32  
chromosome (*sqlite.LowpassCnv* attribute), 32  
chromosome (*sqlite.PanelContent* attribute), 35  
chromosome (*sqlite.PanelIsoforms* attribute), 36  
chromosome (*sqlite.Variants* attribute), 41  
CIP\_code (*sqlite.Petition* attribute), 36  
civic  
    module, 16  
Civic (class in *civic*), 16  
civic (*params.AnalysisConfig* property), 24  
civic\_items (*vep\_vcf.Record* property), 46  
civic\_version (*sqlite.PipelineDetailsTable* attribute), 37  
CivicRestAPI (class in *civic*), 16  
classification (*sqlite.OtherVariantsTable* attribute), 33  
classification (*sqlite.RareVariantsTable* attribute), 37  
classification (*sqlite.SampleVariants* attribute), 40  
classification (*sqlite.TherapeuticTable* attribute), 40  
classify\_cnv() (in module *lowpass*), 21  
classify\_in\_somatic\_tiers() (in module *annotate*), 12

clinical\_directions (*vep\_vcf.Record* property), 46  
clinical\_significance (*vep\_vcf.Record* property), 46  
clinical\_trials (*sqlite.OtherVariantsTable* attribute), 33  
clinical\_trials (*sqlite.RareVariantsTable* attribute), 37  
clinical\_trials (*sqlite.TherapeuticTable* attribute), 40  
clinical\_trials (*vep\_vcf.Record* property), 46  
clinical\_trials\_str (*vep\_vcf.Record* property), 46  
close() (*vep\_vcf.VCFwriter* method), 48  
cn (*sqlite.AllCnas* attribute), 28  
cn (*sqlite.LowpassCnv* attribute), 32  
cna  
    module, 17  
Cna (class in *sqlite*), 30  
cna (*gene\_panel.GenePanelAPI* property), 20  
cns\_calls\_file (*cna.CnvKit* property), 17  
cnv\_json (*sqlite.SampleTable* attribute), 38  
cnv\_plot  
    module, 17  
CnvKit (class in *cna*), 17  
cnvkit (*params.AnalysisConfig* property), 24  
cnvkit\_log2\_json() (in module *var\_call*), 45  
cnvkit\_version (*sqlite.PipelineDetailsTable* attribute), 37  
CnvPlot (class in *cnv\_plot*), 17  
combine\_mutect2\_lancet() (in module *var\_call*), 45  
command (*params.AnalysisConfig* property), 24  
compress\_vcf() (in module *utils*), 44  
Conc\_nanodrop (*sqlite.Petition* attribute), 36  
confirmation\_technique (*sqlite.SampleVariants* attribute), 40  
consequence (*sqlite.OtherVariantsTable* attribute), 33  
consequence (*sqlite.RareVariantsTable* attribute), 37  
consequence (*sqlite.TherapeuticTable* attribute), 40  
conservation (*params.AnalysisConfig* property), 24  
contigs (*bam.Bam* property), 13  
convert\_vcf\_2\_json() (in module *utils*), 44  
convert\_vcf\_to\_dict() (in module *utils*), 44  
copy\_number (*vep\_vcf.Record* property), 46  
count (*sqlite.Variants* attribute), 41  
create\_access\_regions() (*cna.CnvKit* method), 17  
create\_batch\_script() (*genomic\_plots.IGV* method), 20  
create\_bins() (in module *lowpass*), 21  
create\_list\_file() (in module *map*), 23  
create\_snapshots (*params.AnalysisConfig* property), 24  
create\_snapshots() (*genomic\_plots.IGV* method), 21  
create\_summary\_qc\_xlsx() (in module *map*), 23  
create\_target\_regions() (*cna.CnvKit* method), 17  
create\_vep\_dict() (in module *utils*), 44

CSQ (class in *vep\_vcf*), 46

## D

Date (*sqlite.Job* attribute), 31

Date (*sqlite.Petition* attribute), 36

Date\_original\_biopsy (*sqlite.Petition* attribute), 36

date\_original\_biopsy (*sqlite.SampleTable* attribute), 38

date\_time (*params.AnalysisConfig* property), 24

db\_detected\_freq (*sqlite.OtherVariantsTable* attribute), 33

db\_detected\_freq (*sqlite.RareVariantsTable* attribute), 37

db\_detected\_freq (*sqlite.TherapeuticTable* attribute), 40

db\_detected\_number (*sqlite.OtherVariantsTable* attribute), 33

db\_detected\_number (*sqlite.RareVariantsTable* attribute), 37

db\_detected\_number (*sqlite.TherapeuticTable* attribute), 40

db\_dir (*params.AnalysisConfig* property), 24

db\_sample\_count (*sqlite.OtherVariantsTable* attribute), 33

db\_sample\_count (*sqlite.RareVariantsTable* attribute), 37

db\_sample\_count (*sqlite.TherapeuticTable* attribute), 40

dbnsfp\_version (*sqlite.PipelineDetailsTable* attribute), 37

dbscsnv\_version (*sqlite.PipelineDetailsTable* attribute), 37

decompress\_vcf() (in module *utils*), 44

depth (*sqlite.AllFusions* attribute), 29

depth (*sqlite.BiomarkerTable* attribute), 30

depth (*sqlite.OtherVariantsTable* attribute), 33

depth (*sqlite.RareVariantsTable* attribute), 38

depth (*sqlite.TherapeuticTable* attribute), 40

depth (*vep\_vcf.Record* property), 46

diagnosis (*sqlite.SampleTable* attribute), 39

Disclaimer (class in *sqlite*), 31

disclaimer\_db (*gene\_panel.GenePanel* attribute), 19

disclaimers (*gene\_panel.GenePanel* property), 19

disclaimers (*gene\_panel.GenePanelAPI* property), 20

discordant\_reads (*vep\_vcf.Record* property), 46

diseases (*vep\_vcf.Record* property), 46

diseases\_str (*vep\_vcf.Record* property), 47

do\_lowpass() (in module *lowpass*), 21

do\_ratio\_ref() (in module *lowpass*), 21

do\_ratio\_same() (in module *lowpass*), 21

docker (*params.DockerConfig* property), 26

docker\_yaml (*params.AnalysisConfig* property), 24

DockerConfig (class in *params*), 26

dosage\_sensitive\_genes (*sqlite.LowpassCnv* attribute), 32

download\_latest\_release() (*civic.Civic* method), 16

dump\_therapeutic (*sqlite.Cna* attribute), 31

## E

end (*sqlite.AllCnas* attribute), 28

end (*sqlite.AllFusions* attribute), 29

end (*sqlite.Biomarker* attribute), 30

end (*sqlite.BiomarkerTable* attribute), 30

end (*sqlite.Cna* attribute), 31

end (*sqlite.LostExonsTable* attribute), 32

end (*sqlite.LowpassCnv* attribute), 32

end (*sqlite.PanelContent* attribute), 35

end (*sqlite.PanelIsoforms* attribute), 36

END (*vep\_vcf.Record* property), 46

enrichment\_perc\_filter (*sqlite.Panel* attribute), 34

ense\_id (*sqlite.PanelContent* attribute), 35

ensg\_id (*sqlite.LostExonsTable* attribute), 32

ensg\_id (*sqlite.PanelContent* attribute), 35

ensg\_id (*sqlite.PanelIsoforms* attribute), 36

ensp\_id (*sqlite.PanelContent* attribute), 35

enst\_id (*sqlite.LostExonsTable* attribute), 32

enst\_id (*sqlite.OtherVariantsTable* attribute), 33

enst\_id (*sqlite.PanelContent* attribute), 35

enst\_id (*sqlite.PanelIsoforms* attribute), 36

enst\_id (*sqlite.RareVariantsTable* attribute), 38

enst\_id (*sqlite.TherapeuticTable* attribute), 40

evidences (*civic.Civic* property), 16

exon (*sqlite.Biomarker* attribute), 30

exon (*sqlite.BiomarkerTable* attribute), 30

exon (*sqlite.OtherVariantsTable* attribute), 34

exon (*sqlite.RareVariantsTable* attribute), 38

exon (*sqlite.TherapeuticTable* attribute), 40

exon\_end (*sqlite.PanelContent* attribute), 35

exon\_number (*sqlite.LostExonsTable* attribute), 32

exon\_number (*sqlite.PanelContent* attribute), 35

exon\_start (*sqlite.PanelContent* attribute), 35

export\_bed() (*cna.CnvKit* method), 17

export\_seg() (*cna.CnvKit* method), 17

export\_vcf() (*cna.CnvKit* method), 17

ext1\_id (*sqlite.AllCnas* attribute), 28

ext1\_id (*sqlite.AllFusions* attribute), 29

ext1\_id (*sqlite.BiomarkerTable* attribute), 30

ext1\_id (*sqlite.LostExonsTable* attribute), 32

ext1\_id (*sqlite.LowpassCnv* attribute), 33

ext1\_id (*sqlite.OtherVariantsTable* attribute), 34

ext1\_id (*sqlite.RareVariantsTable* attribute), 38

ext1\_id (*sqlite.SampleTable* attribute), 39

ext1\_id (*sqlite.SummaryQcTable* attribute), 40

ext1\_id (*sqlite.TherapeuticTable* attribute), 40

ext2\_id (*sqlite.AllCnas* attribute), 29

ext2\_id (*sqlite.AllFusions* attribute), 29

ext2\_id (*sqlite.BiomarkerTable* attribute), 30

`ext2_id` (*sqlite.LostExonsTable* attribute), 32  
`ext2_id` (*sqlite.LowpassCnv* attribute), 33  
`ext2_id` (*sqlite.OtherVariantsTable* attribute), 34  
`ext2_id` (*sqlite.RareVariantsTable* attribute), 38  
`ext2_id` (*sqlite.SampleTable* attribute), 39  
`ext2_id` (*sqlite.SummaryQcTable* attribute), 40  
`ext2_id` (*sqlite.TherapeuticTable* attribute), 40  
`ext3_id` (*sqlite.SampleTable* attribute), 39  
`extract_coverage()` (*cna.CnvKit* method), 17  
`extract_coverage()` (in module *lowpass*), 22  
`extraction_date` (*sqlite.SampleTable* attribute), 39

## F

`fastp()` (in module *trimming*), 43  
`fastp_json` (*sqlite.SummaryQcTable* attribute), 40  
`fastp_version` (*sqlite.PipelineDetailsTable* attribute), 37  
`fastq`  
    module, 18  
*Fastq* (class in *fastq*), 18  
*FastqNotFound*, 19  
`feature` (*sqlite.PanelContent* attribute), 35  
`fetch()` (*vep\_vcf.VCFreader* method), 48  
*FILTER* (*vep\_vcf.Record* property), 46  
`filter_by_orientation_bias()` (in module *var\_call*), 45  
`fix()` (*cna.CnvKit* method), 17  
`flanking_genes` (*sqlite.AllFusions* attribute), 29  
`fold_change` (*sqlite.LowpassCnv* attribute), 33  
`fold_change` (*vep\_vcf.Record* property), 47  
`fold_change_zscore` (*sqlite.LowpassCnv* attribute), 33  
*FORMAT* (*vep\_vcf.Record* property), 46  
`fq1` (*fastq.Fastq* property), 18  
`fq1_basename` (*fastq.Fastq* property), 18  
`fq2` (*fastq.Fastq* property), 19  
`fq2_basename` (*fastq.Fastq* property), 19  
*freebayes* (*params.AnalysisConfig* property), 24  
`fusion_diseases` (*sqlite.AllFusions* attribute), 29  
`fusion_partners` (*sqlite.AllFusions* attribute), 29  
`fusion_source` (*sqlite.AllFusions* attribute), 29

## G

`gatk` (*params.AnalysisConfig* property), 24  
`gatk_version` (*sqlite.PipelineDetailsTable* attribute), 37  
`gene` (*sqlite.Biomarker* attribute), 30  
`gene` (*sqlite.BiomarkerTable* attribute), 30  
`gene` (*sqlite.Cna* attribute), 31  
`gene` (*sqlite.LostExonsTable* attribute), 32  
`gene` (*sqlite.OtherVariantsTable* attribute), 34  
`gene` (*sqlite.RareVariantsTable* attribute), 38  
`gene` (*sqlite.TherapeuticTable* attribute), 41  
`gene` (*sqlite.Variants* attribute), 42  
`gene_name` (*sqlite.PanelContent* attribute), 35  
`gene_name` (*sqlite.PanelIsoforms* attribute), 36

`gene_panel`  
    module, 19  
*GenePanel* (class in *gene\_panel*), 19  
*GenePanelAPI* (class in *gene\_panel*), 20  
`generate_genomic_snapshots()` (in module *report2*), 27  
`generate_view()` (*genomic\_plots.SimpleBamSnap* method), 21  
`genes` (*civic.Civic* property), 16  
`genes` (*sqlite.AllCnas* attribute), 29  
`genes` (*sqlite.Disclaimer* attribute), 31  
`genome_dir` (*params.GenomeConfig* property), 26  
`genome_fasta` (*params.GenomeConfig* property), 26  
`genome_version` (*params.AnalysisConfig* property), 24  
`genome_version` (*sqlite.Cna* attribute), 31  
`genome_version` (*sqlite.Panel* attribute), 34  
`genome_version` (*sqlite.PanelContent* attribute), 35  
`genome_version` (*sqlite.PanelIsoforms* attribute), 36  
`genome_version` (*sqlite.PipelineDetailsTable* attribute), 37  
`genome_version` (*sqlite.Variants* attribute), 42  
*GenomeConfig* (class in *params*), 26  
*genomic\_plots*  
    module, 20  
`genotype` (*sqlite.LowpassCnv* attribute), 33  
`genotype` (*vep\_vcf.Record* property), 47  
`germline_report()` (in module *report2*), 27  
`get_bin_path()` (in module *utils*), 44  
`get_bin_path()` (*params.BinaryConfig* static method), 26  
`get_bin_path()` (*params.DockerConfig* method), 26  
`get_clinical_trials()` (*vep\_vcf.Record* method), 47  
`get_coverage_metrics()` (*bam.Bam* method), 13  
`get_diseases()` (*vep\_vcf.Record* method), 47  
`get_evidence()` (*civic.Civic* method), 16  
`get_fastq_files()` (in module *utils*), 44  
`get_input_files()` (in module *utils*), 44  
`get_mapping_metrics()` (*bam.Bam* method), 14  
`get_mean_isize()` (*bam.Bam* method), 14  
`get_mutect2_indels_bed()` (in module *var\_call*), 45  
`get_panel_configuration()` (in module *params*), 26  
`get_perc_duplicates()` (*bam.Bam* method), 14  
`get_read_length()` (*bam.Bam* method), 14  
`get_roi_info()` (*gene\_panel.GenePanel* method), 19  
`get_sd_isize()` (*bam.Bam* method), 14  
`get_software_versions()` (*params.BinaryConfig* method), 26  
`get_therapeutic_drugs()` (*vep\_vcf.Record* method), 47  
`get_tier_classification()` (*vep\_vcf.Record* method), 47  
`get_total_reads()` (*bam.Bam* method), 14  
`get_variants()` (*civic.Civic* method), 16  
*gnomad* (*params.AnalysisConfig* property), 25



gnomad\_version (*sqlite.PipelineDetailsTable* attribute), 37

GT (*vep\_vcf.Record* property), 46

## H

HC\_code (*sqlite.Petition* attribute), 36

header (*bam.Bam* property), 15

header (*vep\_vcf.VCFreader* property), 48

header (*vep\_vcf.VCFwriter* property), 48

hgvs (*sqlite.OtherVariantsTable* attribute), 34

hgvs (*sqlite.RareVariantsTable* attribute), 38

hgvs (*sqlite.TherapeuticTable* attribute), 41

hgvs (*sqlite.Variants* attribute), 42

hgvs (*sqlite.OtherVariantsTable* attribute), 34

hgvs (*sqlite.RareVariantsTable* attribute), 38

hgvs (*sqlite.TherapeuticTable* attribute), 41

hgvs (*sqlite.Variants* attribute), 42

hgvs (*sqlite.OtherVariantsTable* attribute), 34

hgvs (*sqlite.RareVariantsTable* attribute), 38

hgvs (*sqlite.TherapeuticTable* attribute), 41

hgvs (*sqlite.Variants* attribute), 42

## I

id (*sqlite.AllCnas* attribute), 29

id (*sqlite.AllFusions* attribute), 29

id (*sqlite.Biomarker* attribute), 30

id (*sqlite.BiomarkerTable* attribute), 30

id (*sqlite.Cna* attribute), 31

id (*sqlite.Disclaimer* attribute), 31

id (*sqlite.Job* attribute), 31

id (*sqlite.LostExonsTable* attribute), 32

id (*sqlite.LowpassCnv* attribute), 33

id (*sqlite.OtherVariantsTable* attribute), 34

id (*sqlite.Panel* attribute), 34

id (*sqlite.PanelContent* attribute), 35

id (*sqlite.PanelIsoforms* attribute), 36

id (*sqlite.Petition* attribute), 36

id (*sqlite.PipelineDetailsTable* attribute), 37

id (*sqlite.RareVariantsTable* attribute), 38

id (*sqlite.SampleTable* attribute), 39

id (*sqlite.SampleVariants* attribute), 40

id (*sqlite.SummaryQcTable* attribute), 40

id (*sqlite.TherapeuticTable* attribute), 41

id (*sqlite.VarAnnotation* attribute), 41

id (*sqlite.Variants* attribute), 42

ID (*vep\_vcf.Record* property), 46

IGV (*class in genomic\_plots*), 20

igv\_snapshots() (*in module report*), 27

index() (*bam.Bam* method), 15

index\_vcf() (*in module utils*), 44

INFO (*vep\_vcf.Record* property), 46

init() (*in module sqlite*), 42

init\_ngs\_database() (*in module sqlite*), 42

input\_dir (*params.AnalysisConfig* property), 25

intron (*sqlite.OtherVariantsTable* attribute), 34

intron (*sqlite.RareVariantsTable* attribute), 38

intron (*sqlite.TherapeuticTable* attribute), 41

InvalidFastqFile, 19

InvalidFastqNomenclature, 19

is\_cna() (*vep\_vcf.Record* method), 47

is\_deletion() (*vep\_vcf.Record* method), 47

is\_gz() (*vep\_vcf.VCFreader* static method), 48

is\_gz() (*vep\_vcf.VCFwriter* static method), 48

is\_insertion() (*vep\_vcf.Record* method), 47

is\_mnv() (*vep\_vcf.Record* method), 47

is\_rare (*vep\_vcf.Record* property), 47

is\_registered() (*gene\_panel.GenePanel* method), 19

is\_registered() (*gene\_panel.GenePanelAPI* method), 20

is\_snv() (*vep\_vcf.Record* method), 47

is\_subpanel (*gene\_panel.GenePanel* property), 20

is\_sv() (*vep\_vcf.Record* method), 47

isoform (*sqlite.Variants* attribute), 42

## J

Job (*class in sqlite*), 31

Job\_id (*sqlite.Job* attribute), 31

## L

lab\_classification (*sqlite.LowpassCnv* attribute), 33

lab\_classification\_date (*sqlite.LowpassCnv* attribute), 33

lab\_confirmation (*sqlite.Disclaimer* attribute), 31

lab\_confirmation (*sqlite.LowpassCnv* attribute), 33

lab\_confirmation (*sqlite.SampleVariants* attribute), 40

lab\_confirmation\_technique (*sqlite.LowpassCnv* attribute), 33

lab\_data (*params.AnalysisConfig* property), 25

lab\_id (*sqlite.AllCnas* attribute), 29

lab\_id (*sqlite.AllFusions* attribute), 29

lab\_id (*sqlite.BiomarkerTable* attribute), 30

lab\_id (*sqlite.LostExonsTable* attribute), 32

lab\_id (*sqlite.LowpassCnv* attribute), 33

lab\_id (*sqlite.OtherVariantsTable* attribute), 34

lab\_id (*sqlite.RareVariantsTable* attribute), 38

lab\_id (*sqlite.SampleTable* attribute), 39

lab\_id (*sqlite.SummaryQcTable* attribute), 40

lab\_id (*sqlite.TherapeuticTable* attribute), 41

lancet (*params.AnalysisConfig* property), 25

language (*sqlite.Disclaimer* attribute), 31

language (*sqlite.Panel* attribute), 34

last\_modified (*sqlite.Panel* attribute), 35

latest\_release (*civic.Civic* property), 16

latest\_report\_pdf (*sqlite.SampleTable* attribute), 39

latest\_version (*gene\_panel.GenePanelAPI* property), 20

launch\_annotation() (*in module annotate*), 13

launch\_mapping() (*in module map*), 23

`launch_report()` (in module *report*), 27  
`launch_report2()` (in module *report2*), 27  
`launch_trimming()` (in module *trimming*), 43  
`launch_variant_calling()` (in module *var\_call*), 45  
`legal_provisions` (*sqlite.Disclaimer* attribute), 31  
`len_alt` (*vep\_vcf.Record* property), 47  
`len_ref` (*vep\_vcf.Record* property), 47  
`load_bam()` (*genomic\_plots.SimpleBamSnap* method), 21  
`load_civic()` (*civic.CivicRestAPI* method), 16  
`load_cna()` (in module *sqlite*), 42  
`load_configuration()` (in module *params*), 26  
`load_panel_biomarkers()` (in module *sqlite*), 42  
`load_panel_disclaimers()` (in module *sqlite*), 42  
`load_panel_transcripts()` (in module *sqlite*), 42  
`load_petitions()` (in module *sqlite*), 42  
`log2_fold_change()` (in module *utils*), 44  
`log_fold_change` (*sqlite.LowpassCnv* attribute), 33  
`long_aminoacid_2_short()` (in module *utils*), 44  
`long_hgvsp_2_short()` (in module *utils*), 44  
`lost_exons_filter` (*sqlite.Panel* attribute), 35  
`lost_exons_perc` (*sqlite.Panel* attribute), 35  
`LostExonsTable` (class in *sqlite*), 32  
`lowpass`  
    module, 21  
`LowpassCnv` (class in *sqlite*), 32

## M

`mane_select` (*sqlite.PanelContent* attribute), 35  
`manta` (*params.AnalysisConfig* property), 25  
`manta_version` (*sqlite.PipelineDetailsTable* attribute), 37  
`map`  
    module, 22  
`max_af` (*sqlite.OtherVariantsTable* attribute), 34  
`max_af` (*sqlite.RareVariantsTable* attribute), 38  
`max_af` (*sqlite.TherapeuticTable* attribute), 41  
`max_af_pop` (*sqlite.OtherVariantsTable* attribute), 34  
`max_af_pop` (*sqlite.RareVariantsTable* attribute), 38  
`max_af_pop` (*sqlite.TherapeuticTable* attribute), 41  
`mean_depth_coordinate()` (in module *utils*), 44  
`mean_read_length` (*fastq.Fastq* property), 19  
`medical_address` (*sqlite.SampleTable* attribute), 39  
`medical_center` (*sqlite.SampleTable* attribute), 39  
`Medical_doctor` (*sqlite.Petition* attribute), 36  
`Medical_indication` (*sqlite.Petition* attribute), 36  
`merge_cnv_sv()` (in module *annotate*), 13  
`merge_samples_coverage()` (in module *lowpass*), 22  
`merge_vcfs()` (in module *annotate*), 13  
`merged_vcf` (*sqlite.SampleTable* attribute), 39  
`methodology` (*sqlite.Disclaimer* attribute), 31  
`min_cn` (*sqlite.Cna* attribute), 31  
`MissingFastqPair`, 19  
module

`annotate`, 11  
`bam`, 13  
`civic`, 16  
`cna`, 17  
`cnv_plot`, 17  
`fastq`, 18  
`gene_panel`, 19  
`genomic_plots`, 20  
`lowpass`, 21  
`map`, 22  
`params`, 24  
`report`, 27  
`report2`, 27  
`sample`, 28  
`sqlite`, 28  
`trimming`, 43  
`utils`, 44  
`var_call`, 45  
`vep_vcf`, 46

## N

`name` (*sample.Sample* property), 28  
`normalize()` (in module *lowpass*), 22  
`num_to_human()` (in module *utils*), 44

## O

`OtherVariantsTable` (class in *sqlite*), 33  
`output_dir` (*params.AnalysisConfig* property), 25  
`output_name` (*params.AnalysisConfig* property), 25

## P

`Panel` (class in *sqlite*), 34  
`panel` (*sqlite.Biomarker* attribute), 30  
`panel` (*sqlite.BiomarkerTable* attribute), 30  
`panel` (*sqlite.Disclaimer* attribute), 31  
`Panel` (*sqlite.Job* attribute), 31  
`panel` (*sqlite.Panel* attribute), 35  
`panel` (*sqlite.PanelContent* attribute), 35  
`panel` (*sqlite.Panellsoforms* attribute), 36  
`panel` (*sqlite.SampleTable* attribute), 39  
`panel_bed` (*sqlite.Panel* attribute), 35  
`panel_content_db` (*gene\_panel.GenePanel* attribute), 20  
`panel_db` (*gene\_panel.GenePanel* attribute), 20  
`panel_dirname` (*params.AnalysisConfig* property), 25  
`panel_full_path` (*params.AnalysisConfig* property), 25  
`panel_isoforms` (*gene\_panel.GenePanel* attribute), 20  
`panel_list_full_path` (*params.AnalysisConfig* property), 25  
`panel_list_name` (*params.AnalysisConfig* property), 25  
`panel_name` (*params.AnalysisConfig* property), 25  
`panel_name` (*sqlite.Cna* attribute), 31

- panel\_version (*sqlite.Cna* attribute), 31  
 panel\_version (*sqlite.PanelContent* attribute), 35  
 panel\_version (*sqlite.PanelIsoforms* attribute), 36  
 PanelContent (class in *sqlite*), 35  
 PanelIsoforms (class in *sqlite*), 36  
 params  
     module, 24  
 Petition (class in *sqlite*), 36  
 petition\_id (*sqlite.OtherVariantsTable* attribute), 34  
 Petition\_id (*sqlite.Petition* attribute), 36  
 petition\_id (*sqlite.RareVariantsTable* attribute), 38  
 petition\_id (*sqlite.SampleTable* attribute), 39  
 petition\_id (*sqlite.SummaryQcTable* attribute), 40  
 petition\_id (*sqlite.TherapeuticTable* attribute), 41  
 physician\_name (*sqlite.SampleTable* attribute), 39  
 pipeline\_version (*sqlite.PipelineDetailsTable* attribute), 37  
 PipelineDetailsTable (class in *sqlite*), 37  
 plot\_cnv() (*cnv\_plot.CnvPlot* method), 17  
 plot\_genomewide() (*cnv\_plot.CnvPlot* method), 17  
 plot\_normalization() (in module *lowpass*), 22  
 pos (*sqlite.Biomarker* attribute), 30  
 pos (*sqlite.BiomarkerTable* attribute), 30  
 pos (*sqlite.Variants* attribute), 42  
 POS (*vep\_vcf.Record* property), 46  
 prepare\_samples\_from\_bams() (in module *utils*), 44  
 prepare\_samples\_from\_vcfs() (in module *utils*), 44  
 probe\_group (*sqlite.LostExonsTable* attribute), 32  
 probe\_group (*sqlite.PanelContent* attribute), 35  
 protein\_coding\_genes (*sqlite.LowpassCnv* attribute), 33
- ## Q
- qc\_criteria (*gene\_panel.GenePanelAPI* property), 20  
 qual (*sqlite.AllCnas* attribute), 29  
 qual (*sqlite.AllFusions* attribute), 29  
 QUAL (*vep\_vcf.Record* property), 46  
 query\_cnv() (*civic.Civic* method), 16  
 query\_fusion() (*civic.Civic* method), 16  
 query\_variant() (*civic.Civic* method), 16  
 query\_variant() (*civic.CivicRestAPI* method), 16  
 Queue\_id (*sqlite.Job* attribute), 31  
 quick\_check() (*bam.Bam* method), 15
- ## R
- RareVariantsTable (class in *sqlite*), 37  
 ratio (*sqlite.AllCnas* attribute), 29  
 ratio\_file (*cna.CnvKit* property), 17  
 Ratio\_nanodrop (*sqlite.Petition* attribute), 36  
 ratios\_to\_json() (in module *lowpass*), 22  
 read\_info() (*vep\_vcf.Record* method), 47  
 read\_num\_filter (*sqlite.Panel* attribute), 35  
 read\_pairs (*sqlite.AllFusions* attribute), 29  
 read\_summary\_qc\_xlsx() (in module *map*), 23  
 read\_support (*sqlite.OtherVariantsTable* attribute), 34  
 read\_support (*sqlite.RareVariantsTable* attribute), 38  
 read\_support (*sqlite.TherapeuticTable* attribute), 41  
 read\_support (*vep\_vcf.Record* property), 47  
 Record (class in *vep\_vcf*), 46  
 ref (*sqlite.Variants* attribute), 42  
 REF (*vep\_vcf.Record* property), 46  
 ref\_dir (*params.AnalysisConfig* property), 25  
 ref\_yaml (*params.AnalysisConfig* property), 25  
 refseq\_id (*sqlite.PanelContent* attribute), 35  
 remove\_bed\_header() (in module *cnv\_plot*), 17  
 remove\_bed\_header() (in module *lowpass*), 22  
 remove\_duplicates() (*bam.Bam* method), 15  
 report  
     module, 27  
 report2  
     module, 27  
 report\_db (*sqlite.SampleTable* attribute), 39  
 report\_language (*params.AnalysisConfig* property), 25  
 report\_pdf (*sqlite.SampleTable* attribute), 39  
 rm\_dup (*params.AnalysisConfig* property), 25  
 roi\_bed (*sqlite.SampleTable* attribute), 39  
 run\_cnvkit() (in module *var\_call*), 45  
 run\_decon() (in module *var\_call*), 45  
 run\_freebayes() (in module *var\_call*), 45  
 run\_gatk\_hc() (in module *var\_call*), 45  
 run\_grapes() (in module *var\_call*), 45  
 run\_id (*sqlite.AllCnas* attribute), 29  
 run\_id (*sqlite.AllFusions* attribute), 29  
 run\_id (*sqlite.BiomarkerTable* attribute), 30  
 run\_id (*sqlite.LostExonsTable* attribute), 32  
 run\_id (*sqlite.LowpassCnv* attribute), 33  
 run\_id (*sqlite.OtherVariantsTable* attribute), 34  
 run\_id (*sqlite.PipelineDetailsTable* attribute), 37  
 run\_id (*sqlite.RareVariantsTable* attribute), 38  
 run\_id (*sqlite.SampleTable* attribute), 39  
 run\_id (*sqlite.SummaryQcTable* attribute), 40  
 run\_id (*sqlite.TherapeuticTable* attribute), 41  
 run\_lancet() (in module *var\_call*), 45  
 run\_manta() (in module *var\_call*), 45  
 run\_mutect2() (in module *var\_call*), 45  
 run\_octopus() (in module *var\_call*), 45
- ## S
- sample  
     module, 28  
 Sample (class in *sample*), 28  
 SAMPLE (*vep\_vcf.Record* property), 46  
 sample\_db\_dir (*sqlite.SampleTable* attribute), 39  
 sample\_id (*sqlite.SampleVariants* attribute), 40  
 sample\_name (*bam.Bam* property), 15  
 sample\_name (*fastq.Fastq* property), 19  
 sample\_type (*sqlite.SampleTable* attribute), 39

Samples (*sqlite.Job* attribute), 31  
 SampleTable (class in *sqlite*), 38  
 SampleVariants (class in *sqlite*), 39  
 samtools\_version (*sqlite.PipelineDetailsTable* attribute), 37  
 segment() (*cna.CnvKit* method), 17  
 segment\_coverage() (in module *lowpass*), 22  
 select\_analysis\_genes\_and\_isoforms() (in module *annotate*), 13  
 seq\_analysis (*params.AnalysisConfig* property), 25  
 set\_info\_subfield\_from\_list() (*vep\_vcf.Record* method), 47  
 set\_info\_subfield\_from\_str() (*vep\_vcf.Record* method), 47  
 set\_labdata\_env() (in module *params*), 26  
 set\_logging() (in module *params*), 26  
 sex (*sqlite.SampleTable* attribute), 39  
 SimpleBamSnap (class in *genomic\_plots*), 21  
 size (*sqlite.Panel* attribute), 35  
 software (*sqlite.SampleTable* attribute), 39  
 software\_version (*sqlite.SampleTable* attribute), 39  
 somatic\_report() (in module *report*), 27  
 somatic\_report() (in module *report2*), 28  
 somatic\_report2() (in module *report2*), 28  
 spliceai\_effect() (in module *report2*), 28  
 split\_reads (*sqlite.AllFusions* attribute), 29  
 split\_reads (*vep\_vcf.Record* property), 47  
 sqlite  
     module, 28  
 start (*sqlite.AllCnas* attribute), 29  
 start (*sqlite.AllFusions* attribute), 29  
 start (*sqlite.Cna* attribute), 31  
 start (*sqlite.LostExonsTable* attribute), 32  
 start (*sqlite.LowpassCnv* attribute), 33  
 start (*sqlite.PanelContent* attribute), 36  
 start (*sqlite.PanelIsoforms* attribute), 36  
 Status (*sqlite.Job* attribute), 31  
 strand (*sqlite.PanelContent* attribute), 36  
 subpanel (*sqlite.SampleTable* attribute), 39  
 summarize\_sample\_qc() (in module *map*), 24  
 summary\_json (*sqlite.SummaryQcTable* attribute), 40  
 SummaryQcTable (class in *sqlite*), 40  
 svend (*vep\_vcf.Record* property), 47  
 svlen (*sqlite.LowpassCnv* attribute), 33  
 svlen (*vep\_vcf.Record* property), 47  
 svtype (*sqlite.AllCnas* attribute), 29  
 svtype (*sqlite.AllFusions* attribute), 29  
 svtype (*sqlite.LowpassCnv* attribute), 33

## T

Tape\_postevaluation (*sqlite.Petition* attribute), 36  
 technical\_limitations (*sqlite.Disclaimer* attribute), 31  
 therapeutic\_drugs (*vep\_vcf.Record* property), 47

therapeutic\_drugs\_str (*vep\_vcf.Record* property), 47  
 TherapeuticTable (class in *sqlite*), 40  
 therapies (*sqlite.OtherVariantsTable* attribute), 34  
 therapies (*sqlite.RareVariantsTable* attribute), 38  
 therapies (*sqlite.TherapeuticTable* attribute), 41  
 thousand\_genomes (*params.AnalysisConfig* property), 25  
 thousand\_genomes\_version  
     (*sqlite.PipelineDetailsTable* attribute), 37  
 threads (*params.AnalysisConfig* property), 25  
 tier\_catsalut (*sqlite.OtherVariantsTable* attribute), 34  
 tier\_catsalut (*sqlite.RareVariantsTable* attribute), 38  
 tier\_catsalut (*sqlite.TherapeuticTable* attribute), 41  
 total\_genes (*sqlite.Panel* attribute), 35  
 total\_rois (*sqlite.Panel* attribute), 35  
 transcripts (*gene\_panel.GenePanel* property), 20  
 trimming  
     module, 43  
 tumor\_type (*sqlite.OtherVariantsTable* attribute), 34  
 tumor\_type (*sqlite.RareVariantsTable* attribute), 38  
 tumor\_type (*sqlite.TherapeuticTable* attribute), 41  
 Tumour\_pct (*sqlite.Petition* attribute), 36  
 tumour\_purity (*sqlite.SampleTable* attribute), 39

## U

update\_cnas() (in module *report*), 27  
 update\_global\_biomarkers() (in module *report*), 27  
 update\_global\_biomarkers() (in module *report2*), 28  
 update\_global\_cnas() (in module *report*), 27  
 update\_global\_cnas() (in module *report2*), 28  
 update\_global\_fusions() (in module *report*), 27  
 update\_global\_fusions() (in module *report2*), 28  
 update\_global\_lost\_exons() (in module *report*), 27  
 update\_global\_lost\_exons() (in module *report2*), 28  
 update\_global\_somatic\_db() (in module *report*), 27  
 update\_global\_somatic\_db() (in module *report2*), 28  
 update\_global\_summary\_qc() (in module *report*), 27  
 update\_global\_summary\_qc() (in module *report2*), 28  
 update\_lowpass\_db() (in module *lowpass*), 22  
 update\_summary\_db() (in module *sqlite*), 43  
 user\_id (*params.AnalysisConfig* property), 25  
 user\_id (*sqlite.AllCnas* attribute), 29  
 user\_id (*sqlite.AllFusions* attribute), 29  
 user\_id (*sqlite.BiomarkerTable* attribute), 30  
 User\_id (*sqlite.Job* attribute), 31  
 user\_id (*sqlite.LostExonsTable* attribute), 32  
 user\_id (*sqlite.LowpassCnv* attribute), 33  
 user\_id (*sqlite.OtherVariantsTable* attribute), 34  
 User\_id (*sqlite.Petition* attribute), 37  
 user\_id (*sqlite.RareVariantsTable* attribute), 38  
 user\_id (*sqlite.SampleTable* attribute), 39  
 user\_id (*sqlite.SummaryQcTable* attribute), 40



`user_id` (*sqlite.TherapeuticTable* attribute), 41  
`utils`  
     module, 44

## V

`vaf` (*sqlite.AllFusions* attribute), 30  
`vaf` (*sqlite.BiomarkerTable* attribute), 30  
`vaf` (*vep\_vcf.Record* property), 47  
`validate()` (*params.AnalysisConfig* method), 25  
`validate()` (*params.AnnotationConfig* method), 25  
`validate()` (*params.BinaryConfig* method), 26  
`validate()` (*params.DockerConfig* method), 26  
`validate()` (*params.GenomeConfig* method), 26  
`validated_assessor` (*sqlite.OtherVariantsTable* attribute), 34  
`validated_assessor` (*sqlite.RareVariantsTable* attribute), 38  
`validated_assessor` (*sqlite.TherapeuticTable* attribute), 41  
`validated_bioinfo` (*sqlite.OtherVariantsTable* attribute), 34  
`validated_bioinfo` (*sqlite.RareVariantsTable* attribute), 38  
`validated_bioinfo` (*sqlite.TherapeuticTable* attribute), 41  
`var_call`  
     module, 45  
`var_id` (*sqlite.SampleVariants* attribute), 40  
`var_id` (*sqlite.VarAnnotation* attribute), 41  
`var_json` (*sqlite.OtherVariantsTable* attribute), 34  
`var_json` (*sqlite.RareVariantsTable* attribute), 38  
`var_json` (*sqlite.TherapeuticTable* attribute), 41  
`var_type` (*sqlite.Variants* attribute), 42  
`VarAnnotation` (class in *sqlite*), 41  
`variant` (*sqlite.Biomarker* attribute), 30  
`variant` (*sqlite.BiomarkerTable* attribute), 30  
`variant_analysis` (*params.AnalysisConfig* property), 25  
`variant_call` (*sqlite.Panel* attribute), 35  
`variant_type` (*sqlite.OtherVariantsTable* attribute), 34  
`variant_type` (*sqlite.RareVariantsTable* attribute), 38  
`variant_type` (*sqlite.TherapeuticTable* attribute), 41  
`variants` (*civic.Civic* property), 16  
`Variants` (class in *sqlite*), 41  
`vartype` (*vep\_vcf.Record* property), 47  
`vcf_2_bed()` (in module *utils*), 44  
`vcf_json` (*sqlite.LowpassCnv* attribute), 33  
`VCFreader` (class in *vep\_vcf*), 48  
`VCFwriter` (class in *vep\_vcf*), 48  
`vep_vcf`  
     module, 46  
`vep_version` (*sqlite.PipelineDetailsTable* attribute), 37  
`version` (*gene\_panel.GenePanel* property), 20  
`version` (*sqlite.Panel* attribute), 35

`Volume` (*sqlite.Petition* attribute), 37

## W

`write()` (*vep\_vcf.VCFwriter* method), 48  
`write_vcf_header_template_for_cnv()` (in module *lowpass*), 22

## Y

`yaml_to_dict()` (in module *utils*), 44