

VARIANT CALLING TUTORIAL: GATK

Presented by Assane MBODJ



INTRODUCTION

Genome-wide variant detection involves the identification of single nucleotide polymorphisms (SNPs) which are changes of one nucleotide in a region of the genome and small insertions and deletions called indels from next generation sequencing data.

This tutorial will focus on the different parts of SNP detection from the mapping step to variant calling and the statistical analyses needed for false positive filtering.

Principle of this tutorial

The goal of this work is to allow any user to perform variant detection analysis (SNP or INDELS) using GATK and picard tools without having any knowledge in bioinformatics.

TOOLS REQUIRED

- Bwa 0.7.8
- Picard tools
- Gatk 4
- Samtools
- vcftools
- IGV
- Bcftools

INSTALL TOOLS

The easiest way is to use the conda library to install the different tools.
Conda allows you to manage all the tools and avoid dependency problems.

Here are the command lines to execute to install all the necessary tools:

>conda install -c bioconda bwa

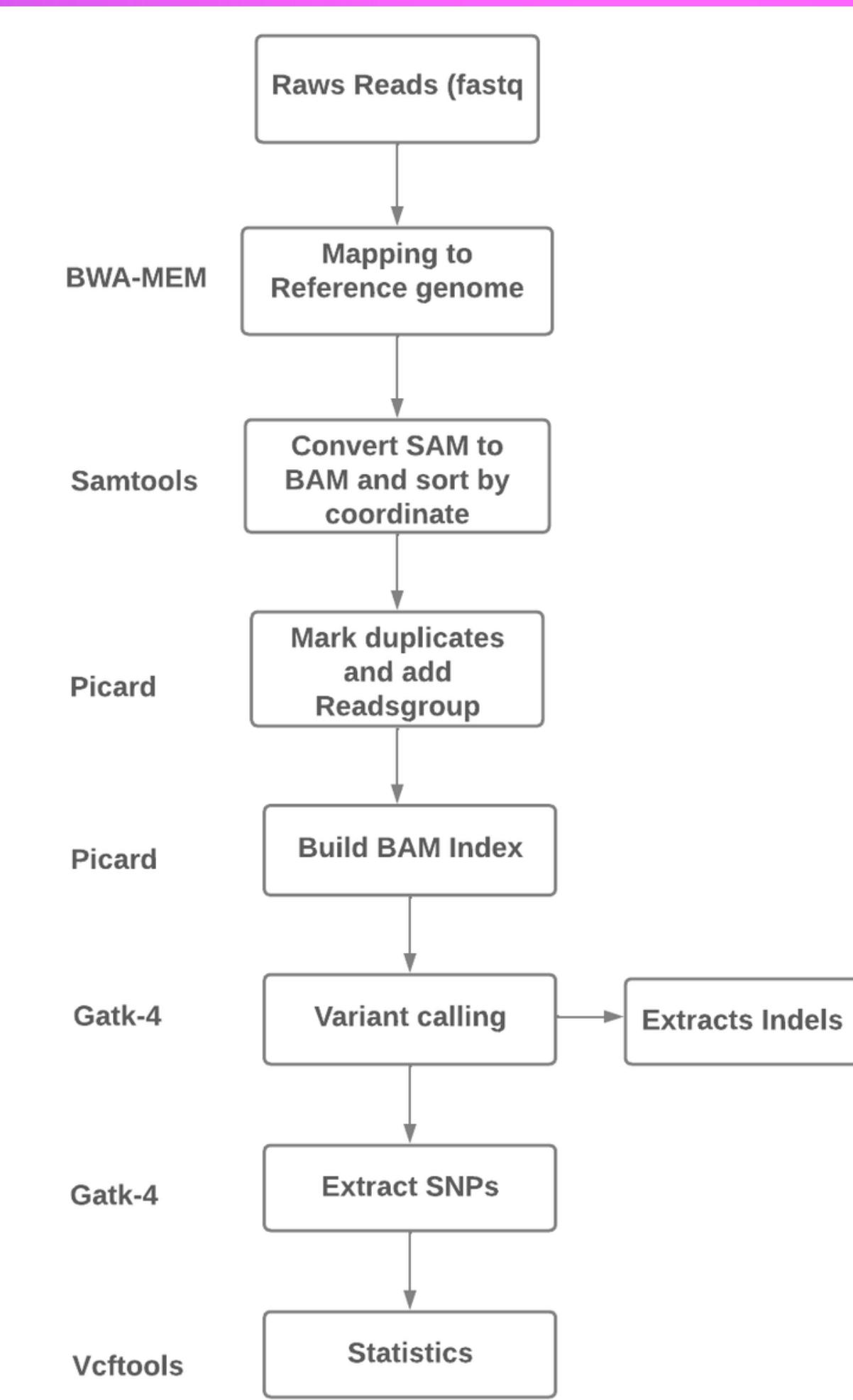
>conda install -c bioconda samtools

>conda install -c bioconda picard

>conda install -c bioconda gatk4

>conda install -c bioconda vcftools

Example of Workflow



DATA SETS

Here we use illumina sequencing data from the *Arabidopsis thaliana* plant with its reference genome to map paired short reads.



Pre-treatment

First you need to create all the folders to store the output data and declare all the file paths that you will need in the different stages of the analysis.

Creation of index file with reference genome and reference dictionary

With the help of the reference sequence, we can generate the index files and a dictionary sequence that are essential for the execution of GATK. They can be created with the following three command lines:

```
samtools faidx reference.fasta
```

```
bwa index reference.fasta
```

```
gatk CreateSequenceDictionary -R reference.fasta
```

Output of the commands

To know if the commands worked, we should expect the following output obtained with the `ls` command.

```
Arabidopsis_thaliana.TAIR10.dna.toplevel.fa
Arabidopsis_thaliana.TAIR10.dna.toplevel.fa.amb
Arabidopsis_thaliana.TAIR10.dna.toplevel.fa.ann
Arabidopsis_thaliana.TAIR10.dna.toplevel.fa.bwt
Arabidopsis_thaliana.TAIR10.dna.toplevel.fa.fai
Arabidopsis_thaliana.TAIR10.dna.toplevel.fa.pac
Arabidopsis_thaliana.TAIR10.dna.toplevel.fa.sa
```

Reads Alignment

We use in this step **BWA-MEM** to align the paired reads to a reference genome and **Samtools** to convert the **SAM** file to **BAM** format and index it.

NB: Other alternatives can be used such as Bowtie2.

Once we have the reference index, we can proceed to the alignment step. We run BWA as follows:

```
bwa mem -t 8 reference.fasta reads_1.fastq reads_2.fastq | samtools sort -@ 8  
-o output.sorted.bam
```

```
samtools index output.sorted.bam
```

Alignment statistics

This part will allow us to see the percentage of reads that have been mapped to the reference genome. We use Samtools flagstats for this.

```
samtools flagstat output.sorted.sam > alignment_metrics.txt
```

Output exemple :

```
3769644 + 0 in total (QC-passed reads + QC-failed reads)
3618048 + 0 primary
+ 0 secondary
51596 + 0 supplementary
+ 0 duplicates
+ 0 primary duplicates
2643954 + 0 mapped (95.26% : N/A)
2492358 + 0 primary mapped (95.23% : N/A)
3618048 + 0 paired in sequencing
1809024 + 0 read1
1809024 + 0 read2
1665798 + 0 properly paired (91.73% : N/A)
2408684 + 0 with itself and mate mapped
3674 + 0 singletons (0.35% : N/A)
46680 + 0 with mate mapped to a different chr
50634 + 0 with mate mapped to a different chr (mapQ>=5)
```

Marking and removing duplicates with Picard tools

This step consists in removing duplicates often produced during the construction of the libraries using PCR. We use Picard tools markduplicate to mark these duplicates and remove them in the BAM files.

```
java -jar picard.jar MarkDuplicates --INPUT output.sorted.bam --OUTPUT output.markdup.bam  
--METRICS_FILE output.metrics.txt --VALIDATION_STRINGENCY SILENT --CREATE_INDEX true --  
REMOVE_DUPLICATES true
```

Output with picard MarkDuplicates

If the command is executed correctly, we should see an output like this:

```
INFO    2022-08-22 11:00:01    MarkDuplicates    After generateDuplicateIndexes f
reeMemory: 6134522248; totalMemory: 8048345088; maxMemory: 8048345088
INFO    2022-08-22 11:00:01    MarkDuplicates    Marking 4101430 records as dupli
cates.
INFO    2022-08-22 11:00:01    MarkDuplicates    Found 121385 optical duplicate c
lusters.
INFO    2022-08-22 11:00:01    MarkDuplicates    Reads are assumed to be ordered
by: coordinate
INFO    2022-08-22 11:01:49    MarkDuplicates    Written 10 000 000 records.
Elapsed time: 00:01:47s. Time for last 10 000 000: 107s. Last read position:
Chr4:2 626 752
INFO    2022-08-22 11:03:28    MarkDuplicates    Writing complete. Closing input
iterator.
INFO    2022-08-22 11:03:28    MarkDuplicates    Duplicate Index cleanup.
INFO    2022-08-22 11:03:28    MarkDuplicates    Getting Memory Stats.
INFO    2022-08-22 11:03:28    MarkDuplicates    Before output close freeMemory:
8055760232; totalMemory: 8110735360; maxMemory: 8110735360
INFO    2022-08-22 11:03:28    MarkDuplicates    Closed outputs. Getting more Mem
ory Stats.
INFO    2022-08-22 11:03:28    MarkDuplicates    After output close freeMemory: 8
126539112; totalMemory: 8181514240; maxMemory: 8181514240
[Mon Aug 22 11:03:28 CEST 2022] picard.sam.markduplicates.MarkDuplicates done. E
lapsed time: 4,87 minutes.
Runtime.totalMemory()=8181514240
```

Add of read groups

The read groups are provided as a string. This information is essential for using GATK, as it does not work without a read group tag.

```
java -jar picard.jar AddOrReplaceReadGroups --INPUT output.sorted.markdup.bam  
--OUTPUT output.sorted.markdup.RG.bam --RGSM {NAME} --RGID {NAME} --RGLB  
lib1 --RGPL illumina --RGPU 2
```

Output with picad reads groups

We will have to have an exit like this if everything works out well :

```
INFO 2022-08-22 11:16:21 AddOrReplaceReadGroups Processed 10 000 000 records.
Elapsed time: 00:01:47s. Time for last 1 000 000: 10s. Last read position: Chr4:2 626
752
INFO 2022-08-22 11:16:32 AddOrReplaceReadGroups Processed 11 000 000 records.
Elapsed time: 00:01:59s. Time for last 1 000 000: 11s. Last read position: Chr4:8 328
169
INFO 2022-08-22 11:16:42 AddOrReplaceReadGroups Processed 12 000 000 records.
Elapsed time: 00:02:09s. Time for last 1 000 000: 10s. Last read position: Chr4:17 489
661
INFO 2022-08-22 11:16:52 AddOrReplaceReadGroups Processed 13 000 000 records.
Elapsed time: 00:02:19s. Time for last 1 000 000: 9s. Last read position: Chr5:7 946
489
INFO 2022-08-22 11:17:04 AddOrReplaceReadGroups Processed 14 000 000 records.
Elapsed time: 00:02:30s. Time for last 1 000 000: 11s. Last read position: Chr5:15 323
829
INFO 2022-08-22 11:17:14 AddOrReplaceReadGroups Processed 15 000 000 records.
Elapsed time: 00:02:41s. Time for last 1 000 000: 10s. Last read position: Chr5:24 585
944
INFO 2022-08-22 11:17:23 AddOrReplaceReadGroups Processed 16 000 000 records.
Elapsed time: 00:02:50s. Time for last 1 000 000: 9s. Last read position: ChrPt:20 03
1
INFO 2022-08-22 11:17:32 AddOrReplaceReadGroups Processed 17 000 000 records.
Elapsed time: 00:02:59s. Time for last 1 000 000: 8s. Last read position: ChrPt:73 31
4
INFO 2022-08-22 11:17:40 AddOrReplaceReadGroups Processed 18 000 000 records.
Elapsed time: 00:03:07s. Time for last 1 000 000: 8s. Last read position: ChrPt:123 8
96
INFO 2022-08-22 11:17:50 AddOrReplaceReadGroups Processed 19 000 000 records.
Elapsed time: 00:03:17s. Time for last 1 000 000: 9s. Last read position: /*/*
[Mon Aug 22 11:17:58 CEST 2022] picard.sam.AddOrReplaceReadGroups done. Elapsed time: 3,42
minutes.
Runtime.totalMemory()=397934592
```

BAM file index with Picard tools

This indexer file is important to do before using GATK, because it does not work without the indexed BAM file.

```
java -jar picard.jar BuildBamIndex --INPUT output.sort.bowtie.markdup.RG.bam
```

Here is the resulting output:

```
11:30:56.669 INFO NativeLibraryLoader - Loading libgkl_compression.so from jar:file:/home/lgdp/picard.jar!/com/intel/gkl/native/libgkl_compression.so
[Mon Aug 22 11:30:56 CEST 2022] BuildBamIndex --INPUT P010C-bwa.RG.bam --VERBOSITY INFO --QUIET false --VALIDATION_STRINGENCY STRICT --COMPRESSION_LEVEL 5 --MAX_RECORDS_IN_RAM 500000 --CREATE_INDEX false --CREATE_MD5_FILE false --GA4GH_CLIENT_SECRETS client_secrets.json --help false --version false --showHidden false --USE_JDK_DEFLATER false --USE_JDK_INFLATER false
[Mon Aug 22 11:30:56 CEST 2022] Executing as lgdp@gwas on Linux 5.4.0-124-generic amd64; OpenJDK 6 4-Bit Server VM 1.8.0_152-release-1056-b12; Deflater: Intel; Inflater: Intel; Provider GCS is not available; Picard version: Version:2.27.1
INFO 2022-08-22 11:31:36 BuildBamIndex Successfully wrote bam index file P010C-bwa.RG.bai
[Mon Aug 22 11:31:36 CEST 2022] picard.sam.BuildBamIndex done. Elapsed time: 0,66 minutes.
Runtime.totalMemory()=756023296
```

Variants Calling : SNP et Indels

We use GATK HaplotypeCaller to perform variant calling.

GATK HaplotypeCaller is able to call SNPs and Indels simultaneously using the reference sequence and the BAM file.

```
java -jar gatk-package-4.2.6.1-local.jar HaplotypeCaller -R reference.fasta -I  
output.sorted.markdup.RG.bam -O output_variants.vcf
```

Filtering SNPs AND Indels

This step will allow us to filter the SNPs from the Indels in order to process them independently using GATK SelectVariants. Here, we are only interested in SNPs.

SNP Filtering :

```
java -jar gatk-package-4.2.6.1-local.jar -T SelectVariants -R reference.fasta -V  
output_variants.vcf -selectType SNP -o output_snps.vcf
```

Indel Filtering :

```
java -jar gatk-package-4.2.6.1-local.jar -T SelectVariants -R reference.fasta -V  
output_variants.vcf -selectType indels -o output_snps.vcf
```

GATK script on Population

In the case of a population, the script remains the same for each individual. The difference is made after obtaining the vcf files of each individual which will be merged thereafter with the function CombineGVCFs of gatk.

```
#!/usr/bash

Ref=/path/reference.fasta
Bam_file=/path/Bam_file
Output=/path/outDir

#Index Reference Sequence
samtools faidx reference.fasta
bwa index reference.fasta
java -jar picard.jar CreateSequenceDictionary R=reference.fasta O=reference.fasta.dict

for fq1 in /path/file_R1.fq
do
fq2=$(echo $fq1 | sed -e "s/_R1.fq._/R2/g")
OUT=$(basename $fq1 | awk -F "_" '{print $1}')

bwa mem -t 8 $REF $fq1 $fq2 |samtools sort -@ 8 -o $Bam_file/$OUT.sorted.bam -
samtools flagstats $Bam_file/$OUT.sorted.bam > $output/$OUT.stats.txt

#Mark the duplicate with picard tools
java -jar picard.jar MarkDuplicates --INPUT $Bam_file/$OUT.sorted.bam --OUTPUT $Bam_file/$OUT.markdup.bam
--VALIDATION_STRINGENCY SILENT --CREATE_INDEX true --REMOVE_DUPLICATES true

#Add the read groups on the bam file
java -jar picard.jar AddOrReplaceReadGroups --INPUT $Bam_file/$OUT.sorted.markdup.bam
--OUTPUT $Bam_file/$OUT.markdup.RG.bam --RGSM $OUT --RGID $OUT --RGLB lib1 --RGPL illumina --RGPU 2

#Sort bam markdup with picard
java -jar picard.jar SortSam --INPUT $Bam_file/$OUT.markdup.RG.bam --OUTPUT $Bam_file/$OUT.sorted.markdup.RG.bam
--VALIDATION_STRINGENCY SILENT --SORT_ORDER coordinate

#index bam
java -jar picard.jar BuildBamIndex --INPUT $Bam_file/$OUT.sorted.markdup.RG.bam

#Called the Variants (indels and SNP)
java -jar gatk-package-4.2.6.1-local.jar HaplotypeCaller -R $REF -I $Bam_file/$OUT.sorted.markdup.RG.bam
-O $Output/$OUT.POP.g.vcf.gz -ERC GVCF

rm -r $Bam_file/$OUT.markdup.bam*
rm -r $Bam_file/$OUT.markdup.RG.bam*

done

#Create vcf file list
ls $vcf_file/*.g.vcf.gz > $vcf_file/vcfs.list

#Merged per samples gVCF files produced by HaplotypeCaller into a multi-sample gVCF file
java -jar gatk-package-4.2.6.1-local.jar CombineGVCFs -R $REF --variant $vcf_file/vcfs.list
-O $Output/$OUT.POP.merged.g.vcf

#GenotypeGVCFs the multi-sample gVCF file produced by CombineGVCFs
java -jar gatk-package-4.2.6.1-local.jar GenotypeGVCFs -R $Ref -V $Output/$OUT.POP.merged.g.vcf.gz
-O $Output/$OUT.POP.genotype.vcf

#Filter only SNP with gatk on multi-sample file
java -jar gatk-package-4.2.6.1-local.jar SelectVariants -R $Ref -V $Output/$OUT.POP.genotype.vcf
--select-type-to-include SNP -O $Output/$OUT.POP.SNP.vcf

#Filter only INDEL with gatk on multi-sample file
java -jar gatk-package-4.2.6.1-local.jar SelectVariants -R $Ref -V $Output/$OUT.POP.genotype.vcf
--select-type-to-include INDEL -O $Output/$OUT.POP.indels.vcf
```

Use GATK on cluster

The execution of a script on a cluster is not always the same when you are on a personal machine. The difference can be noticed in the writing of the script or in the execution. For the case of variant calls like gatk, it is necessary to create two scripts, a run which contains the code to be executed and a launch which contains the path of the target files and the script to be executed. In our case, we write a script for an OpenPBS cluster with the qsub function which allows to execute the scripts.

run_gatk.sh Script on cluster

- Here, we use SGE cluster

```
#PBS -N name_your_run
#PBS -l nodes=1:ppn=6
#PBS -q lmem
#PBS -M email.fr
#PBS -m e

#samtools v1.13
module load samtools/1.15
module load bowtie2/2.4.4
module load bwa/0.7.17
module load gatk/4.2.5.0
module load picard/

Ref=/path/reference.fasta
Bam_file=/path/Bam_file
Output=/path/outDir

FQ1=$file
FQ2=$(echo $file | sed -e "s/_R1.fq.*/_R2*/g")
OUT=$(basename $file | awk -F "_" '{print $1}')

#file=$file
#OUT=$(basename $file | awk -F "." '{print $1}')

#bowtie2 --very-sensitive-local -p 4 -x $REF -1 $fq1 -2 $fq2 -S $Bam_file/$OUT-bowtie.sam
#samtools reheader -P $Bam_file/$OUT-bowtie.sam $Bam_file/$OUT-bowtie.bam

bwa mem -t 8 $REF $fq1 $fq2 | samtools sort -@ 8 -o $Bam_file/$OUT.sorted.bam -
samtools flagstats $Bam_file/$OUT.sorted.bam > $output/$OUT.stats.txt

#Mark the duplicate with picard tools
java -jar picard.jar MarkDuplicates --INPUT $Bam_file/$OUT.sorted.bam --OUTPUT $Bam_file/$OUT.markdup.bam
--VALIDATION_STRINGENCY SILENT --CREATE_INDEX true --REMOVE_DUPLICATES true

#Add the read groups on the bam file
java -jar picard.jar AddOrReplaceReadGroups --INPUT $Bam_file/$OUT.sorted.markdup.bam
--OUTPUT $Bam_file/$OUT.markdup.RG.bam --RGSM $OUT --RGID $OUT --RGLB lib1 --RGPL illumina --RGPU 2

#Sort bam markdup with picard
java -jar picard.jar SortSam --INPUT $Bam_file/$OUT.markdup.RG.bam --OUTPUT $Bam_file/$OUT.sorted.markdup.RG.bam
--VALIDATION_STRINGENCY SILENT --SORT_ORDER coordinate

#index bam
java -jar picard.jar BuildBamIndex --INPUT $Bam_file/$OUT.sorted.markdup.RG.bam

#Called the Variants (indels and SNP)
java -jar gatk-package-4.2.6.1-local.jar HaplotypeCaller -R $REF -I $Bam_file/$OUT.sorted.markdup.RG.bam
-O $Output/$OUT.POP.g.vcf.gz -ERC GVCF

rm -r $Bam_file/$OUT.markdup.bam*
rm -r $Bam_file/$OUT.markdup.RG.bam*
```

launch_gatk.sh Script on a cluster

```
#!/bin/bash

#path your fastq file
list_file=$(ls /path/fastq_file)

#Loupe on fastq file
for file in $list_file
do

#qsub allows to run the script
qsub -v file=$file /path/run_gatk.sh
done
```

Script gatk_merge.sh on cluster

This script merges all the vcf files obtained with gatk HaplotypeCaller into one file, then genotypes and filters them to separate SNPs from Indels.

```
#PBS -N gatk_merge
#PBS -l nodes=1:ppn=6
#PBS -q lmem
#PBS -M email@.fr
#PBS -m e

module load gatk/4.2.5.0

Ref=/path/reference.fasta
vcf_file=/path/vcf_file
Output=/path/outDir

#Create list vcf_file
ls $vcf_file/*.vcf.gz > $vcf_file/vcfs.list

#Merged per samples gVCF files produced by HaplotypeCaller into a multi-sample gVCF file
java -jar gatk-package-4.2.6.1-local.jar CombineGVCFs -R $REF --variant $vcf_file/vcfs.list
-O $Output/$OUT.POP.merged.g.vcf.gz

#GenotypeGVCFs the multi-sample gVCF file produced by CombineGVCFs
java -jar gatk-package-4.2.6.1-local.jar GenotypeGVCFs -R $Ref -V $Output/$OUT.POP.merged.g.vcf.gz
-O $Output/$OUT.POP.genotype.vcf

#Filter only SNP with gatk on multi-sample file
java -jar gatk-package-4.2.6.1-local.jar SelectVariants -R $Ref -V $Output/$OUT.POP.genotype.vcf
--select-type-to-include SNP -O $Output/$OUT.POP.SNP.vcf

#Filter only INDEL with gatk on multi-sample file
java -jar gatk-package-4.2.6.1-local.jar SelectVariants -R $Ref -V $Output/$OUT.POP.genotype.vcf
--select-type-to-include INDEL -O $Output/$OUT.POP.indels.vcf
```

The command to run for this script is as follows:

qsub gatk_merge.sh

Variant Statistics

We use here tools able to filter variants from a vcf file to make distributions in the form of plot with R namely vcftools and RStudio.

The objective of these statistics is to bring a decision support on the choice of the filters to be applied on the vcf brût.

We can make analyses on several parameters such as the average depth of coverage of the variants, on the qualities of variants, on the frequency of minor alleles (MAF) or on the missing data. And in order to do this, we use R scripts to make plots.

These statistics can be applied on a population, here we use the whole At66 collection.

Average depth of coverage per individual

This part will allow us to calculate the average depth of coverage of each individual in the collection.

This calculation is done with vcftools thanks to its --depth function.

The command used is the following:

```
vcftools --vcf output_snps.vcf --depth --out output_snp
```

Here we obtain a file with an extension .idepht.mean containing the name of the individual, the number of variants and the average depth of coverage.

example of average coverage output per individual

INDV	N_SITES	MEAN_DEPTH
P001A-vs-TAIR10	4455580	15.4357
P001B-vs-TAIR10	4455531	16.3596
P001C-vs-TAIR10	4455667	13.3344
P002A-vs-TAIR10	4455334	15.1015
P002B-vs-TAIR10	4455660	14.8374
P002C-vs-TAIR10	4455594	16.8962
P002D-vs-TAIR10	4455727	15.0376
P002E-vs-TAIR10	4455458	19.4515
P002F-vs-TAIR10	4455534	18.8342
P002G-vs-TAIR10	4455162	15.4907
P002H-vs-TAIR10	4455254	13.8889
P002I-vs-TAIR10	4455497	16.0763
P002J-vs-TAIR10	4455689	15.7405
P003A-vs-TAIR10	4455692	16.537
P003B-vs-TAIR10	4455737	14.4205
P003C-vs-TAIR10	4456088	19.1175
P003D-vs-TAIR10	4455609	16.3083
P003E-vs-TAIR10	4455757	16.7391
P003F-vs-TAIR10	4456027	17.673
P003G-vs-TAIR10	4455884	16.918
P003H-vs-TAIR10	4455203	14.4419
P003I-vs-TAIR10	4455729	15.3194
P004A-vs-TAIR10	4455818	17.6617

Average depth of coverage per variants

The filter on the average depth of the variants will allow us to eliminate the false positive calls, i.e. the variants with a low or very high coverage.

This calculation is done with **vcftools** thanks to its **--site-mean-depth** function.

The command used is the following:

```
vcftools --vcf output_snps.vcf --site-mean-depth --out output_snps
```

Here we obtain a file with an extension .ldepth.mean containing the position of each variant, its average coverage depth and the variation of the coverage depth within the individual.

example of average coverage output by variant

CHROM	POS	MEAN_DEPTH	VAR_DEPTH
Chr1	55	2.24277	14.4989
Chr1	56	2.16763	12.7915
Chr1	60	2.46821	12.4597
Chr1	73	2.23121	11.5509
Chr1	115	7.78035	31.0678
Chr1	138	12.0173	43.1102
Chr1	194	3.75145	32.4553
Chr1	201	4.12139	40.0259
Chr1	214	4.55491	43.6438
Chr1	216	4.61272	42.9829
Chr1	237	6.74566	68.8768
Chr1	253	8.31214	86.588
Chr1	346	18.237	29.3679
Chr1	353	25.4798	188.053
Chr1	425	24.5145	190.391
Chr1	429	24.3988	193.125
Chr1	437	24.1387	190.597
Chr1	471	23.3988	175.055
Chr1	502	16.3699	19.1996
Chr1	508	16.3179	19.439
Chr1	524	16.3526	29.6249
Chr1	551	16.0751	30.1629
Chr1	553	16.0809	30.5632
Chr1	560	16.0058	31.3081
Chr1	564	16.0636	31.8157

What do you think is due to the huge difference in coverage in some sites?

Variants quality

The calculation of the quality of the variants gives us information on the quality score of each variant. This will then allow us to apply filters to eliminate poor quality variants.

The calculation is done with **vcftools** with its **--site-quality** function using the following command:

```
vcftools --vcf output_snps.vcf --site-quality --out output_snps
```

Here we get a file with a .lqual extension containing the position of each variant and the quality value.

example of output on the quality variants

CHROM	POS	QUAL
Chr1	55	134.57
Chr1	56	164.29
Chr1	60	133.01
Chr1	73	111.84
Chr1	115	390.63
Chr1	138	5123.24
Chr1	194	919.43
Chr1	201	2333.5
Chr1	214	2184.99
Chr1	216	1339.75
Chr1	237	2277.6
Chr1	253	2076.1
Chr1	346	109125
Chr1	353	4916.78
Chr1	425	4957.5
Chr1	429	1142.1
Chr1	437	2543.19
Chr1	471	4722.46
Chr1	502	111344

Minors allele frequencies

We calculate here the frequency of minor alleles of each variant with **vcftools** with the function **--freq2**.

The command used is the following:

```
vcftools --vcf output_snps.vcf --freq2 --out output_snps
```

We obtain here a file with a .freq extension containing the position of each variant and the frequency of minor alleles.

Example of output on the Minors frequencies

CHROM	POS	N_ALLELES	N_CHR	{FREQ}
Chr1	55	2	70	0.971429 0.0285714
Chr1	56	2	38	0.947368 0.0526316
Chr1	60	2	88	0.977273 0.0227273
Chr1	73	2	18	0.888889 0.111111
Chr1	115	2	40	0.85 0.15
Chr1	138	2	78	0.435897 0.564103
Chr1	194	2	34	0.941176 0.0588235
Chr1	201	2	34	0.882353 0.117647
Chr1	214	2	70	0.942857 0.0571429
Chr1	216	2	40	0.95 0.05
Chr1	237	2	64	0.9375 0.0625
Chr1	253	2	222	0.981982 0.018018
Chr1	346	2	346	0.147399 0.852601
Chr1	353	2	346	0.959538 0.0404624
Chr1	425	2	346	0.959538 0.0404624
Chr1	429	2	346	0.988439 0.0115607
Chr1	437	2	346	0.976879 0.0231214
Chr1	471	2	342	0.959064 0.0409357
Chr1	502	2	330	0.1 0.9
Chr1	524	2	344	0.994186 0.00581395
Chr1	560	2	344	0.994186 0.00581395
Chr1	564	2	342	0.994152 0.00584795
Chr1	572	2	324	0.987654 0.0123457
Chr1	573	2	344	0.997093 0.00290698
Chr1	581	2	344	0.994186 0.00581395
Chr1	583	2	328	0.981707 0.0182927
Chr1	584	2	344	0.997093 0.00290698
..	..	-	-	-

Missing data

There are two ways to calculate missing data. One can calculate the proportion of missing data per sample with the **--missing-ind** function of **vcftools** or per variant with **--missing-site**.

The commands used are the following:

```
vcftools --vcf output_snps.vcf --missing-ind --out output_snps
```

```
vcftools --vcf output_snps.vcf --missing-site --out output_snps
```

We obtain here two files, the first with an .imiss extension (for the individuals) and another with an lmiss extension (for the variants) containing the position of each variant or each individual as well as the percentage of missing data.

Example of output on missing data

CHR	POS	N_DATA	N_GENOTYPE_FILTERED	N_MISS	F_MISS
Chr1	55	346	0	276	0.797688
Chr1	56	346	0	308	0.890173
Chr1	60	346	0	258	0.745665
Chr1	73	346	0	328	0.947977
Chr1	115	346	0	306	0.884393
Chr1	138	346	0	268	0.774566
Chr1	194	346	0	312	0.901734
Chr1	201	346	0	312	0.901734
Chr1	214	346	0	276	0.797688
Chr1	216	346	0	306	0.884393
Chr1	237	346	0	282	0.815029
Chr1	253	346	0	124	0.358382
Chr1	346	346	0	0	0
Chr1	353	346	0	0	0
Chr1	425	346	0	0	0
Chr1	429	346	0	0	0
Chr1	437	346	0	0	0
Chr1	471	346	0	4	0.0115607
Chr1	502	346	0	16	0.0462428
Chr1	508	346	0	16	0.0462428
Chr1	524	346	0	2	0.00578035
Chr1	551	346	0	22	0.0635838
Chr1	553	346	0	22	0.0635838
Chr1	560	346	0	2	0.00578035
Chr1	564	346	0	4	0.0115607
Chr1	572	346	0	22	0.0635838
Chr1	573	346	0	2	0.00578035
Chr1	581	346	0	2	0.00578035
Chr1	583	346	0	18	0.0520231
Chr1	584	346	0	2	0.00578035
Chr1	586	346	0	0	0
Chr1	611	346	0	18	0.0520231
Chr1	614	346	0	20	0.0578035
Chr1	615	346	0	20	0.0578035
Chr1	631	346	0	18	0.0520231

Plot on statistical data

To create diagrams, we use the **RStudio** programming language with the **tidyverse library**. To do this, we will load the package on **RStudio** with the following command:

```
library(tidyverse)
```

Statistics on the average depth of individuals

Here we examine the average depth of each variant. This is the number of reads mapped to that position.

To do this, we must first read the file on R by positioning on the data directory.

```
stat_depth <- read_delim("./output_snps.idepht", delim = "\t",
                           col_names = c("chr", "pos", "mean_depth", "var_depth"), skip = 1)
```

We then look at the median, mean and standard deviation with the summary command

```
summary(stat_depth$mean_depth)
```

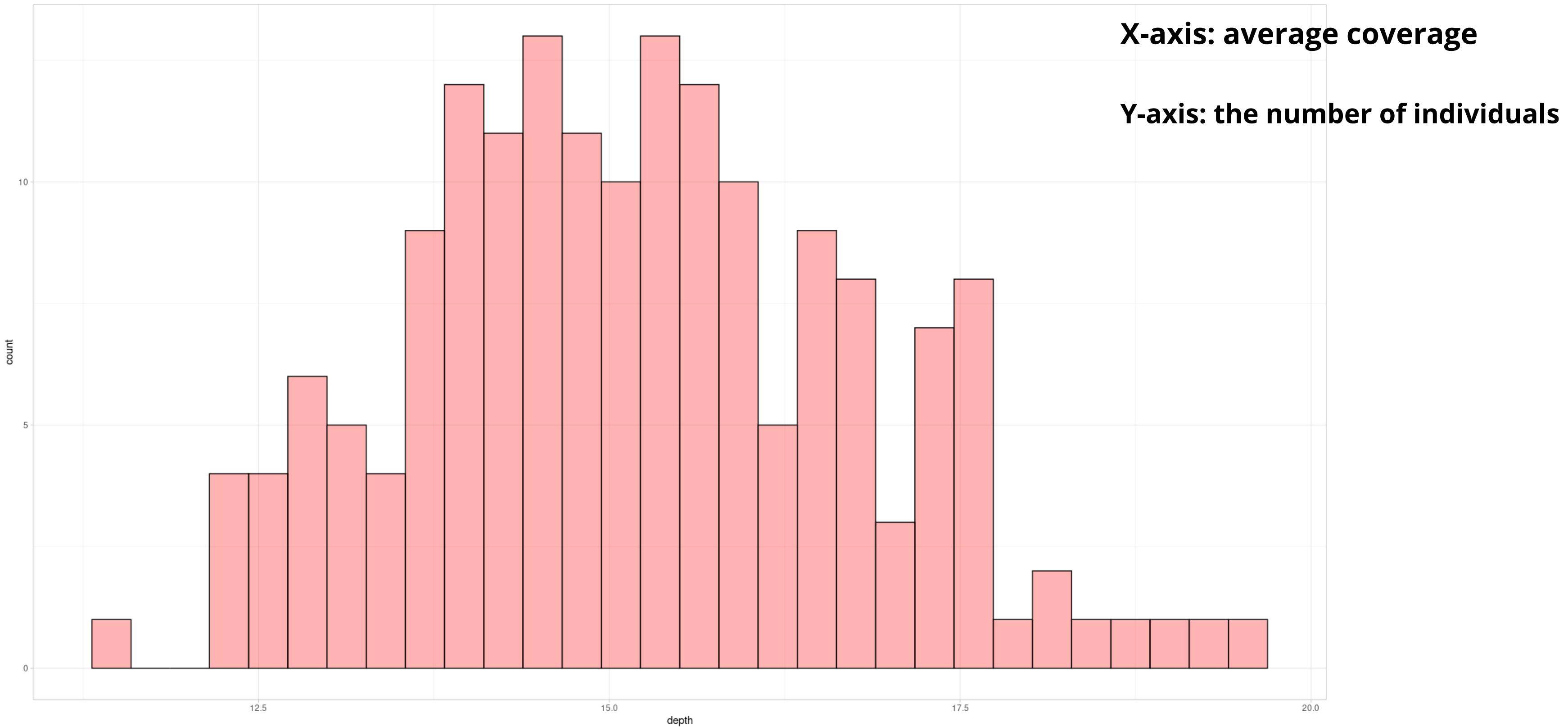
Statistics on the average depth of individuals

We use the **ggplot** package to build graphs on R.

```
plot_depth<- ggplot(stat_depth, aes(mean_depth)) +  
geom_density(fill = "dodgerblue1", colour = "black", alpha = 0.3)
```

```
plot_depth + theme_light() + xlim(0, 100)
```

Histogramm on depth coverage per individual



Statistics on variant quality

Variant quality refers to the measure of confidence we have in variant calls.

First, we read from the site quality report that we generated using the following command:

```
stats_qual <- read_delim("./output_snps.lqual", delim = "\t",  
  col_names = c("chr", "pos", "qual"), skip = 1)
```

We then look at the median, mean and standard deviation with the summary command :

```
summary(stats_qual$qual)
```

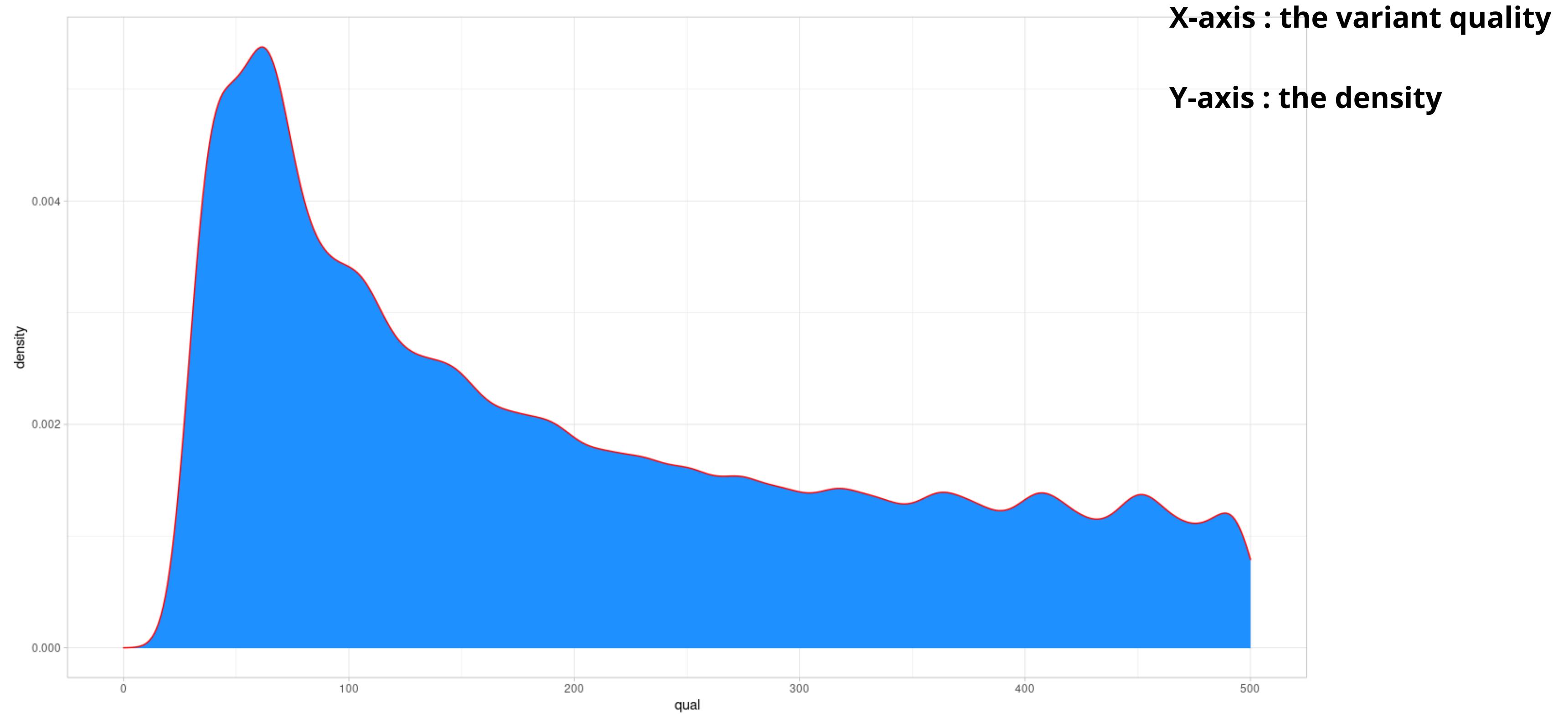
Statistics on variant quality

We use the ggplot package to build graphs on R.

```
plot_qual <- ggplot(stats_qual, aes(qual)) + geom_density(fill = "dodgerblue1", colour = "black", alpha = 0.3)
```

```
plot_qual + theme_light() + xlim(0, 500) #xlim to be defined according to sample size
```

Graph on variants quality



Statistics on Minors allele frequencies

We will examine the distribution of allele frequencies to set the minimum MAF threshold.

```
stat_maf <- read_delim("./output_snps.frq", delim = "\t",
                         col_names = c("chr", "pos", "nalleles", "nchr", "f1", "f2"), skip = 1)

# find minor allele frequency
stat_maf$maf <- stat_maf %>% select(f1, f2) %>% apply(1, function(z) min(z))
```

We then look at the median, mean and standard deviation with the summary command.

```
summary(stat_maf$maf)
```

Statistics on Minors allele frequencies

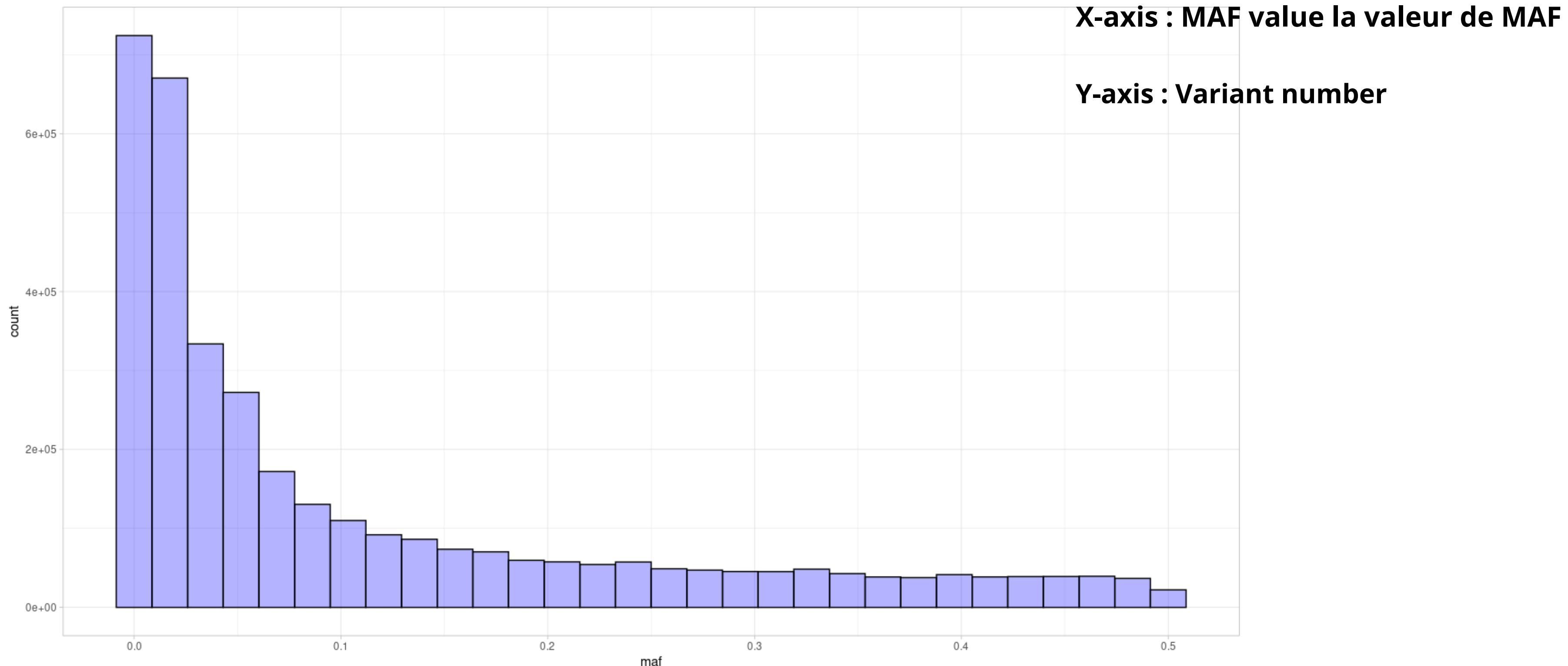
Here we plot the distribution of minor allelic frequencies using ggplot.

```
plot_maf <- ggplot(stat_maf, aes(maf)) + geom_density(fill = "dodgerblue1", colour =  
"black", alpha = 0.3)
```

```
plot_qual + theme_light()
```

Graph on Minors allele frequencies

MAF



Filtering of the vcf with the parameters defined with the stats

After the obtained statistics, we will be able to define the values of the minimum and maximum average depth of each variant, the quality of the variants and the MAF defined by the respective terms **--min-meanDP**/**--max-meanDP**, **--minQ**, **--maf**. Don't forget also to define the minimum value of missing data which is interesting for the **GWAS** step.

The command used is the following:

```
vcftools --vcf output_snps.vcf --max-missing INT --min-meanDP/--max-meanDP INT --minQ INT --maf INT --recode --out output_snps
```



Thank
you

