Msc Internship Intermittent Project Proposal:

Variant Calling in R

Dymphe Remarque,
S1838768

Supervisor:
Dr. Klaas Vrieling

Abstract

Linkage and association studies have gained popularity with the rise of fast and cheap whole genome sequencing. These studies rely on the use of genetic markers - short variable DNA sequences with a known location in the genome - to identify haplotypes and phenotypes related to specific variant alleles. The discovery and analysis of genetic markers within a genome, also known as variant calling, currently requires the use of multiple specialised software controlled via terminal commands. Therefore, the process of variant calling is laborious for research groups without access to a bio-informatician. The goal of this research project was to lower the threshold by creating an easy-to-use variant calling pipeline in R, with which most scientists already have ample experience. The final product is the R package `VariantCallinginR`, where each step of variant calling is compiled into an R function. Now, the entire variant discovery pipeline from raw data to variant call file can be executed in ten lines of code or less, including regular quality controls. In order to test the functionality of the package, compound variant discovery has been performed for nineteen previously sequenced, high interest *Cannabis sativa* strains aligned to the CBDRx reference strain. In total, over 11.5 million single nucleotide polymorphisms and 0.8 million microsatellites were discovered within the *Cannabis* genome.

Introduction

Genetics have become more and more accessible over the years with improved and cheaper sequencing possibilities. As a result, linkage and association studies are widely applied from forensic DNA profiling[1] to diagnosis of inheritable diseases[2] to crop breeding[3]. Such studies strongly rely on the identification of genetic markers -short variable DNA sequences with a known location in the genome- to differentiate between individuals of a population. Each individual has a unique combination of variant alleles. The inheritance patterns observed in a population reveal the linkage distance of variants relative to each other, such that linked variants tend to be inherited together. Genetic maps represent this linkage distance in centimorgan and form the foundation for all linkage and association studies. To make the genetic maps, however, genetic markers first need to be determined for the species or population of interest.

Discovery of genetic markers is performed using a method called variant calling. Here, the sequenced reads of the individual of interest are aligned to a reference genome. The reference can be selected based on the default genome published in online databases like NCBI and Ensembl, or on another haplotype that is representative of either a specific population or ecotype of interest.

Once the alignment is made and checked for its quality, each nucleotide of the individual of interest is compared to that of the reference. In general, there are two types of nucleotide, two alleles, possible: the nucleotide is the same as the reference (REF) or the nucleotide is different from the reference (ALT). This means that a diploid organism can be homozygote REF, homozygote ALT or heterozygote ALT:REF for a nucleotide. The latter two genotypes are the variants that have to be called. The length of a variant is not necessarily limited to a single nucleotide. Multiple nucleotide polymorphisms (MNP) like microsatellites are detectable as well.

Information of the found variants is stored in a Variant Call Format (VCF) file, designed specifically for this purpose in the 1000 genomes project (1KGP)[4]. This information includes general parameters like the genotype and allele location, frequency, and sequence. In addition, quality parameters such as coverage, variant positions within reads, strand bias and sequence alignment and genotype confidence are also stored in VCF files. With those parameters, variants are then filtered according to the requirements of the respective researcher. Some may aim for many microsatellites of average quality whilst others may require a few single nucleotide polymorphism (SNP) markers of high quality. In general, variants are used for two purposes: (1) to analyse variant frequencies within a population or (2) to genotype individuals using variants for development of allele specific PCR primers. In order to achieve the first aim mentioned, multiple genomes are aligned to a reference for comparisons ,whilst the latter aim is achieved by analysing variants of an individual of interest relative to only the reference genome. Here, subjects of interest may vary from the parent of a segregating population to an individual with specific traits like an inheritable disease.

To summarise, the genome of one or more individual(s) of interest is sequenced and stored in FASTA or FASTQ files. Then, the raw sequence data are aligned to a reference genome and stored in a BAM file. This BAM file is scanned for deviations from the reference and those variants are stored in a VCF file. Finally, the VCF file is filtered based on the research requirements to obtain the genetic markers.

In practice, alignment, variant calling and quality analysis are most often performed by using specialised programs such as GATK Haplotypecaller[5], FreeBayes[6], Samtools[7], VarScan[8,9] or any other of the open-access software available[10,11]. However, these programs do have limitations; some are not available for certain operating systems, others may have a tendency towards false positives. Moreover, and all before-mentioned programs require a background of bioinformatics in order to operate them. Although most currently available software is flexible and can be used for a wide array of variant calling types, a high amount of user input is essential to manage the data. As a result, research groups without access to a bio-informatician are unable to perform reliable variant calling. This means that whilst sequencing is accessible, the construction of genetic markers is not.

The aim of this project was to remedy this by constructing an easy to use variant calling pipeline package in R, a program with which most scientists already have ample experience. The goal was to write the package functions in such a way that any researcher will be able to use it in their own research for variant discovery, regardless of their operating system or background. The R pipeline will cover all steps of variant calling as described above, as well as various quality controls, in such a way that each step corresponds to an individual function.
To establish the effectiveness of the constructed VarianCallinginR package, the full analysis pipeline was performed for nineteen previously sequenced, high interest *Cannabis sativa* strains, mapped to the cs10 CBDRx reference strain. These data will not only provide a real-life example to what the R package is capable of, but it will also serve as a step-by-step introduction to handling the functions. In addition, the discovered high-quality variants within the *Cannabis* genome may also form the foundation for future research projects.

Materials and Methods

General Method

To construct a package in which both the entire variant discovery pipeline, as well as select subsections of the analysis can be performed according to research requirements with only a few lines of code, each step of variant calling was contained within a separate function. Depending on the nature of the function, a varying combination of currently available R packages from both Bioconductor and CRAN databases, system calls to external software such as GATK, samtools and BWA, and custom code were used to set R as the main platform for the analysis from start to finish. In Figure 1, the entire variant calling pipeline has been summarised and divided into three main components: Alignment of the alternative to the reference genome, variant discovery itself and the post-hoc quality analysis of the acquired variants. For both the alignment to the reference genome and for variant calling, two separate options were included to provide the user with more flexibility in performing the analysis. For the alignment, either Bowtie2 wrapped in R or external Barrows Wheels Alignment (BWA-MEM) can be used and controlled from within R. In a similar fashion, VariantTools provides a Samtools based variant calling algorithm wrapped within R itself, whilst the external GATK software can be controlled from R for more extensive variant calling. Furthermore, there are two available paths during the variant discovery, depending on the number of samples used. For a single sample a raw VCF file is immediately made, whilst multiple samples require intermittent .g.vcf files before being genotyped and combined into a compound VCF file.
At the end of the analysis, the variants are stored into an excel file, since .xlsx files are universally accessible across all platforms and the majority of researchers are very familiar with this file format. This eliminates the need for additional software or R coding after acquiring the desired high-quality variants.

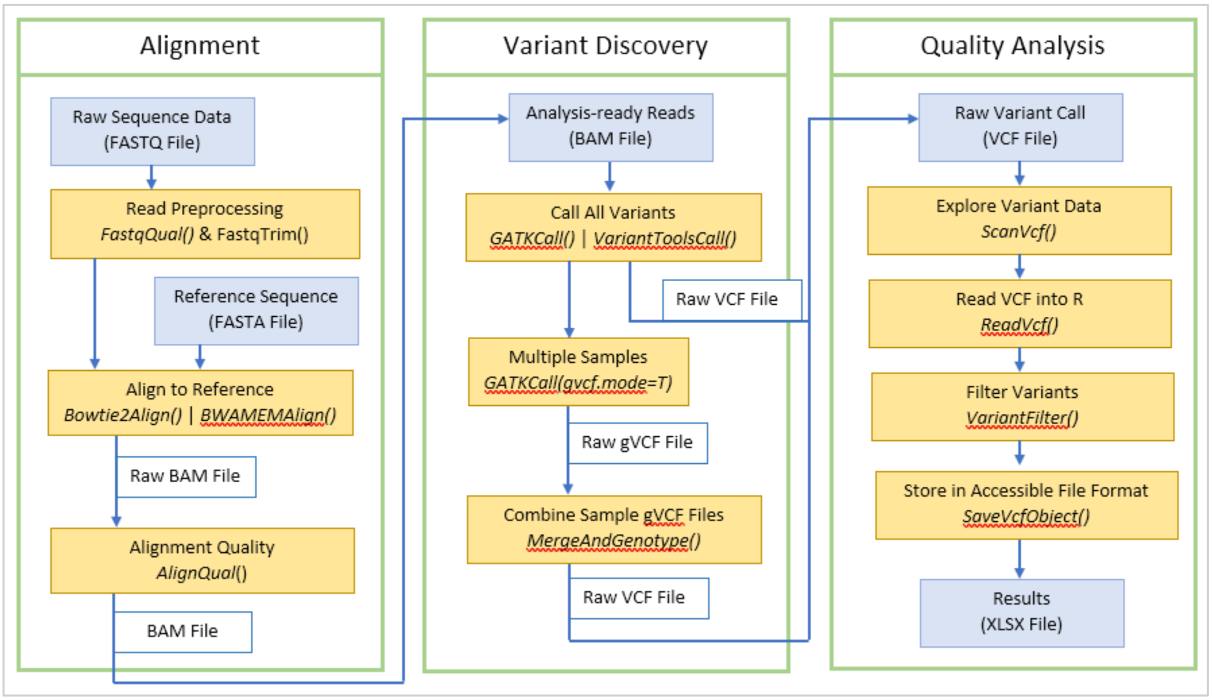

Alignment
Raw Sequence Data (FASTQ File)
Read Preprocessing FastqQual() & FastqTrim()
Reference Sequence (FASTA File)
Align to Reference Bowtie2Align() | BWAMEMAlign()
Raw BAM File
Alignment Quality AlignQual()
BAM File
Variant Discovery
Analysis-ready Reads (BAM File)
Call All Variants GATKCall() | VariantToolsCall()
Raw VCF File
Multiple Samples GATKCall(gvcf.mode=T)
Raw gVCF File
Combine Sample gVCF Files MergeAndGenotype()
Raw VCF File
Quality Analysis
Raw Variant Call (VCF File)
Explore Variant Data ScanVcf()
Read VCF into R ReadVcf()
Filter Variants VariantFilter()
Store in Accessible File Format SaveVcfObject()
Results (XLSX File)

Figure 1: An overview of variant discovery analysis steps. Enclosed by green are the three main components, divided into smaller steps, shown in orange. Along with the sub steps are the corresponding package function names. Shown in blue blocks are the input data and corresponding file extension name. In some blocks more than one function is listed. These functions are either combined (&) to perform a step, or two different approaches (|).

Package Usage

As stated before, a number of previously built packages from the CRAN and Bioconductor database have been used. As a result, the designed package `VariantCallinginR` requires installation of additional packages. This is automatically done by R when installing `VariantCallinginR` by listing the package dependencies within the code itself. In addition, a function used within the source code of a function is always precluded with *`package_name::function()`*. This makes it easier to discern where and when a dependency is used. The complete code underlying all functions and their explanations can be found in Appendix 3 for further details.
To provide transparency and a quick overview of what components originate from previous work, the package dependencies are listed below in Table 1 as well.

***Table 1**: List of packages the general purpose for which they were implemented.*

Package Name	Usage
`ChIPsim`[12]	Write .fastq files for example data generation
`dada2`[13]	FilterandTrim() fastq files based on quality
`ggplot2`[14]	Plotting of quality graphs
`ggpubr`	Arrangement of quality graphs into a single image
`GenomicRanges`[15]	Selection of target regions for variant calling with VariantTools
`optparse`[16]	Python-like command line parsing for R server scripts
`Rbowtie2`[17]	Alignment and adapter removal with Bowtie2
`Rsamtools`[18]	Pileup of variant data from the alignment file
`seqinr`[19]	Write FASTA files for example data generation
`ShortRead`[20]	Quality plotting and adapter detection of single end reads
`systemPipeR`[21]	Quality plotting of paired end reads
`VariantTools`[22]	Variant calling within R based on samtools
`VariantAnnotation`[23]	Collection of VCF header information
`vcfR`[24,25]	Reading of unprocessed VCF data into coded R objects

External software calling

For more complete variant calling, additional software was used outside of R. This was the case for BWA, Samtools and GATK, which all run natively on Linux systems. To keep the analysis accessible on all operating systems, software images running in a virtual Linux environment, Docker Toolbox[26], were tested and implemented for Windows systems. All external software is controlled by the respective R functions by sending the Unix commands to the system either directly for Mac and Linux (*`system()`*), or indirectly via Git Bash (*`shell()`*).
Due to time constraints the external software was not converted in a wrapped shell, similar to how Bowtie2 was wrapped into R by Lerch and Stadler[17]. Nevertheless, no additional user input is required beyond installing the base software such as Docker Toolbox to access specific functions. For users unable to use external software, native R-only functions for each of the software dependant functions were created as alternative.

Selection and approach of case study data

*F*rom the previously published available data, a selection of 19 available *Cannabis* strain sequence data was made to test the package functionality with real-life data (table 2). This selection was based on the documented potential for genetic variation. The sequence data originates predominantly from female plant samples and all sequence data were collected with Illumina Novaseq 3000. The data therefore consist of 150 base pair reads with attached adapter sequences from NEBNext® Multiplex Oligos for Illumina®(Unique Dual Index UMI Adaptors DNA Set 1)[27].

Table 2: The selected Cannabis strain names, SRR codes, gender and chemotype. The strains are predominantly chemotype I female samples supplemented with chemotypes II and III, and key male and monoecious samples. The SRR codes correspond to the database number under which the raw sequence data can be found on the NCBI SRA.

SRR Code	Name	Gender	Chemotype
10578252	Master Kush	Female	1
10578250	Domnesia	Female	1
10578281	Sour Diesel	Female	1
10578287	Chem 91	Female	1
10578288	Arcata Trainwreck	Female	1
10578274	Jamaican Lion^3 Mother	Female	2
10578269	Carmagnola_3	Female	3
10578275	Jamaican Lion^3 Father	Male	2
10578268	Fedora17_6_1_CSU	n.a. (Monoecious)	3
10578270	BlueBerry Cheesecake X JL Male	Female	3
10578266	Harlox	Female	1
10578260	Tiborszallasi	Male	2
10578280	Headcheese	Female	1
10578285	Sour Tsunami	Female	2
10578283	Tahoe OG	Male	1
10578286	Mothers Milk #5	Female	1
10578277	Citrix	Female	1
10578253	Herijuana	Female	1
10578254	Grape Stomper	Female	1

Data access

All genomic data used are available on the NCBI. For the reference genome, the CBDRx cs10 genome assembly GCF_900626175.1 was used. This assembly was originally made by Grassa *et al.* in 2018[28] and was last updated in April 2020. This assembly spans 736 million base pairs over 10 chromosomes, with an additional 139 million base pairs in unplaced contigs. For variant calling, these unplaced contigs were excluded to ensure that all variants have reliable genomic location and annotation data. For the alternative *Cannabis* strains, raw sequence data were extracted from the NCBI Single Read Archive (SRA). These data were originally collected by McKernan *et al.,* in 2020[27], where they were used for mapping against the Jamaican Lion genome in search for pathogen resistance and cannabinoid synthesis gene variants.

Server and software specifications

The pipeline was built and tested in R[29] version 3.6.2 64-bit. The latest versions of all R packages were installed in April 2020 and are available for all operating systems.
The analysis of *Cannabis* strains was performed using resources provided by the Academic Leiden Interdisciplinary Cluster Environment (ALICE). This server is a hybrid cluster consisting of a total of 604 TFlops, 816 cores (1632 threads) and 14.4 TB RAM. For the analysis on the ALICE, specialised server versions with Python-like command flags were used of the designed R functions to enable simple input and output modifications within the SLURM job scheduler script itself, rather than in the R code. These additional server scripts are functionally the same as the regular functions and are merely

optimised for user comfort when working from command lines. These scripts were also implemented into the package for future use under the directory R_Server_Scripts.
Pipeline testing with randomly generated example data was performed locally on an Intel i5-5200U 2.20GHz CPU with 8 GB RAM machine running Windows 10 Home. Linux commands and software were executed on this device with Git Bash v2.9.0 and Docker Toolbox v19.03.1, respectively. For Linux-based alignment and variant calling, BWA version 0.7.17 and GATK version 4.1.3.0 were used and tested both within Docker Toolbox and on the ALICE server.
In addition, the finalised VariantCallinginR package and its installation instruction have been made available on Github (https://github.com/DRemarque/VariantCallinginR) for public access.

Results:

General:

In order to demonstrate the potential of the `VariantCallinginR` package, the full pipeline was executed for sequence data of nineteen *Cannabis* strains. The results for these data followed similar patterns in quality, input parameter requirements and overall behaviour during the analysis. Presenting the results for all strains separately would therefore be redundant. Thus, the general behaviour of sequence data during the analysis, and the utilisation of the package functions are demonstrated with the data of only *Cannabis sativa var Citrix*.
For additional detailing on the underlying code of the produced functions, please view Appendix 3.

Sequence Quality Assessment:

As shown in Figure 1, the first step of the pipeline is to assess the quality of the raw sequence data obtained from the NCBI SRA database. These data were unfiltered and still contained the adapter sequences from library preparation. Furthermore, sequencing was performed using paired ends, which requires additional controls to ensure that both forward and reverse reads are of equal quality. To generate the quality plots and to search for the exact adapter sequences to remove, the code shown in chunk 1 was used. Here, the two FASTQ files containing the raw sequence data of the forward (_fw) and reverse (_rv) reads are provided by the first two arguments in the *`FastqQual()`* function, along with the confirmation that these data are indeed paired-end (*`Paired_End_Data=TRUE`*). The desired file base name for the output is defined after *`Output_Base_Name`*, which the code automatically adds the appropriate file extension to. Finally, the last three arguments dictate whether an adapter search, graph creation and server specific graph saving should be executed. Since the complete quality analysis is performed on the ALICE server, these arguments are set to TRUE.

Chunk 1

```
FastqQual(Forward_Fastq = "Citrix_fw.fastq",
          Reverse_Fastq = "Citrix_rv.fastq",
          Output_Base_Name = "Citrix_Quality",
          Paired_End_Data = TRUE, Adapter_Check = TRUE, Create_Graph = TRUE,
          server = TRUE )
```

Output:

```
## Starting up...
## Collecting data. This may take a while
##
## Generating Quality Plots. This may take a while
##
## Quality plots have been saved in /data as Citrix_Quality.pdf
##
## Searching for potential adapter sequences
## arguments 'show.output.on.console', 'minimized' and 'invisible' are for Windows
## only
## Opening FASTQ file 'Citrix_fw.fastq'
## Opening FASTQ file 'Citrix_rv.fastq '
##
## Adapter sequences have been saved in Citrix_Adapters.txt
```

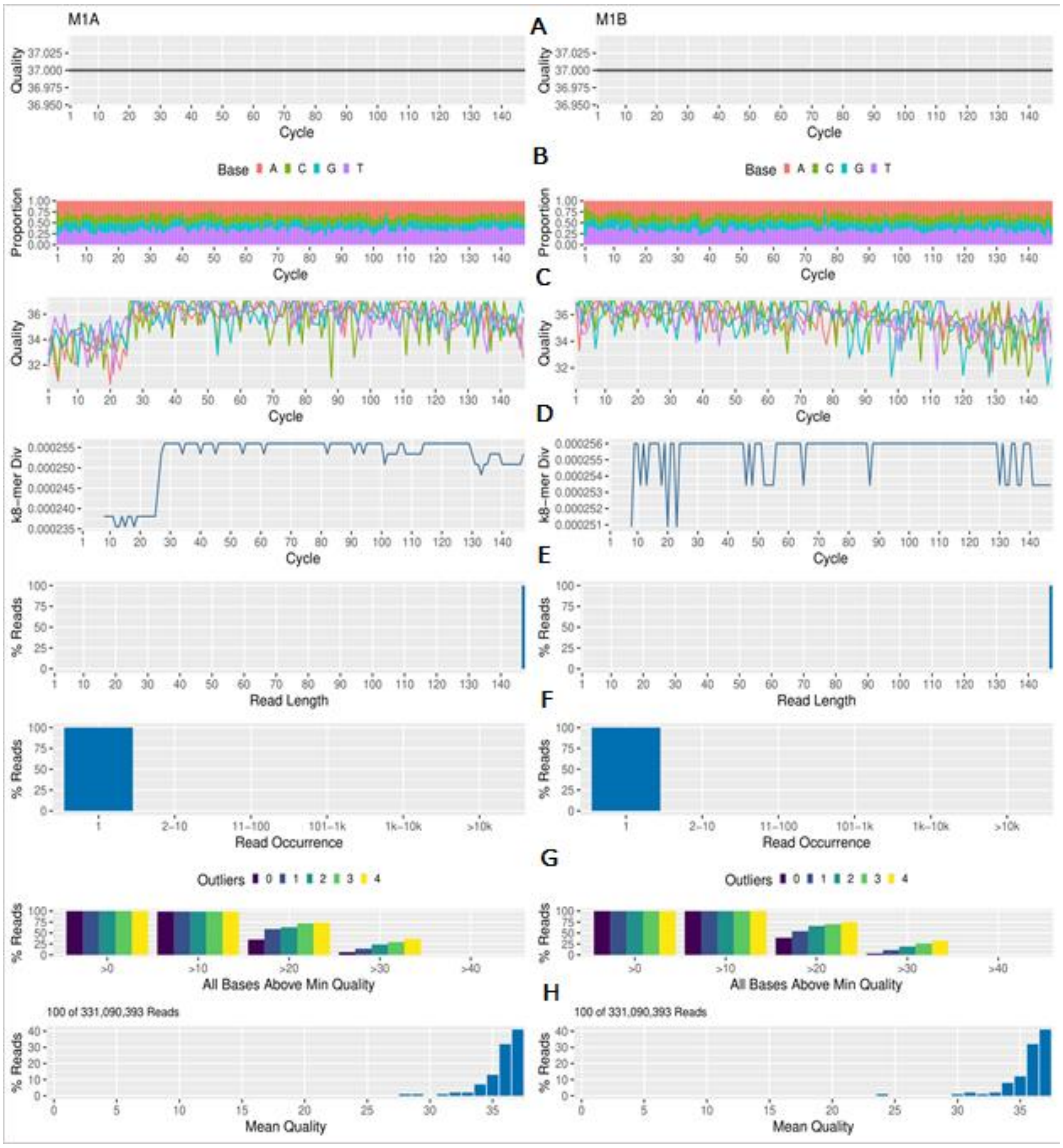


Figure 2: A set of quality graphs for both forward (left column) and reverse (right column) reads of Citrix. The code generating the graph relies on SystempipeR's seefastq function, which samples a number of reads from the dataset to give an indication of the general quality.

In A, the average read quality per cycle is shown to be a score of 37 out of 42, corresponding with the expectation that the sequence data overall is of high quality.

In addition, the division of nucleotides (A,T, C or G) is shown to be balanced with a slight tendency towards A-T base pairs (plot B). As an extension of A, plot C displays the average nucleotide quality per cycle as well as per nucleotide type. An exceptionally high or low quality for merely one nucleotide or cycle could indicate sequencing bias towards or against a base, whereas a low quality after a certain cycle may suggest a sequencing error after a certain read length. Here however, neither is prominently present with the exception of a small quality decrease at the end of the reverse reads and start of the forward reads, which is not unusual for Illumina sequencing.

A measure for read complexity is given in plots D via the relative k8-mer diversity. For a read of 150 base pairs, the expected k8-mer diversity is (Read Length-k+1)/48 = (150-8+1)/48 ≈ 0.00218 and, as can be seen from the plots, all k8-mers lie above this expected threshold. This indicates that the raw reads contain more complexity than expected and are therefore well suited for reliable alignment.

Plots E and F are very straightforward. The read lengths (E) all show the expected 150 bp in length and all sampled reads occur once; none of the reads were sampled twice (F).
The most relevant graph for setting filtering parameters based on individual nucleotide quality is plot G. These plots display the percentage of reads where all nucleotides have a quality score above a certain value. It is therefore a predictor of how many reads will be discarded when setting high quality thresholds. Alternatively, it may show that the desired low to medium quality criteria are already met by all sequenced nucleotides, making additional filtering based on those critera unnecessary.
Finally, plot H shows the percentages of mean read qualities. These plots provide an overview of the whole read quality distribution and are, similar to plots G, a good indication on how many data will be discarded when filtering based on high quality thresholds. For Citrix, none of the mean qualities are below 26 and the vast majority of the nucleotides have a quality of 10 or higher.

The outputted quality graphs, shown in Figure 2, were created based on `systemPipeR's` *`seefastq()`* plotting function. The quality graph consists of 8 individual plots for each of the forward (M1A) and reverse (M1B) reads, of which plots C and D are most notable in terms of quality indicators.
Plot C displays the average nucleotide quality in relation to its position within the read (cycle) and its type (A,T,G or C). The majority of the nucleotides show a quality score of approximately 36 out of 42, with a few downward peaks at the start of the forward, and end of the reverse reads. Generally, quality scores between 20 and 37 out of 42 are accepted as high quality, with scores from 10 to 20 being classified as medium quality.
Plot D shows the relative k8-mer diversity, calculated by dividing the number of detected k8-mers by the total number of possible k8-mers (=4^8). The expected relative k8-mer for 150 bp reads is (Read Length-k+1)/4^8 = (150-8+1)/4^8 ≈ 0.00218. Any value higher than the expected baseline indicates higher k-mer diversity, whereas a lower number indicates lower k-mer diversity. Low complexity reads such as repeats and abundant small transposon sequences are therefore detected when plotting. For this analysis, a k-mer of 8 bp was chosen to be able to detect problematic reads for alignment, whilst reducing false positives by not detecting short repeating sequences such as individual amino-acid codons (k=3). As can be seen in the graph, the k8-mer diversity before sequence filtering is higher than the expected diversity, indicating that repeats and transposons will not be problematic during alignment. Overall, the sequence data match the expectations of high-quality data, eliminating the need for strict filtering.

Sequence data filtering and adapter trimming:

After the quality control, the detected adapter sequences and a small fraction of low-quality reads are filtered out. This was done with the *`FastqTrim()`* function shown in chunk 2. The filtering criteria were the same for all *Cannabis* strains, as they are all of equal quality.
Due to the high-quality nature of the data, strict filtering was not required. The main focus was to remove reads containing one or more nucleotides with a quality score of 10 out of 42 or less. For Illumina sequencing, such a score indicates a 10% chance of incorrectly sequencing the nucleotideThis error potential decreases to a 1% chance for nucleotides with a score of 20 out of 42. The quality assessment (Figure 2) shows a considerable amount of data that would be lost under a stricter threshold of 20 and as a result the minimum nucleotide quality (*`minQ`*) was set to 10.
To ensure that all remaining unreliable reads, that were possibly missed during the quality assessment, are filtered out, a maximum number of unknown nucleotides (*`maxN`*) was set to 30 and suspected contamination sequences were detected and removed by *`rm.phix=TRUE`*. The remainder of the filtering parameters were left at the default settings made explicit in chunk 2. For further information on these parameters, view Appendix 3E. For the quality assessment of the filtered data and the corresponding *`FastqQual()`* graph, view Appendix 1 Figure 1.
In the output of chunk 2, the removal of the adapter sequences, stored in the Citrix.txt file, is reported in the first paragraph. The filtering based on the quality thresholds is reported in the second

paragraph along with the percentage of remaining data, here 99.6%. As mentioned before, no hard filtering was required and therefore only few reads were removed from the datasets. The final paragraph of the output reports the generated files and their location, signalling the end of the function.

Chunk 2

```
FastqTrim(Forward_Fastq="Citrix_fw.fastq",
          Reverse_Fastq="Citrix_rv.fastq",
          Paired_End_Data=TRUE,
          Adapter_Removal=TRUE,
          Adapters="Citrix.txt",
          Output_Base_Name="Citrix_Trim",
          minLen=10, maxLen=Inf, maxN=30, minQ=10,
          rm.phix=TRUE, truncQ=2, trimLeft=0,
          trimRight=0, maxEE=Inf, rm.lowcomplex=0)
```

Output:

```
## Starting up...
## Removing Adapters specified in .txt file
## arguments 'show.output.on.console', 'minimized' and 'invisible' are for Windows
## only
## Trimming paired end reads ...
## Opening FASTQ file 'Citrix_fw.fastq '
## Opening FASTQ file 'Citrix_rv.fastq '
##
## Filtering FastQ Files. This may take a while
## Overwriting file:/data/Citrix_Trim_fw.fastq
## Overwriting file:/data/Citrix_Trim_rv.fastq.fastq
## Read in 324889439 paired-sequences, output 323720761 (99.6%) filtered paired-se
quences.
##
## The Filtered FASTQ Files have been saved in /data as Citrix_Trim_fw.fastq and
## Citrix_Trim_rv.fastq
```

Sequence mapping to the CBDRx reference genome:

The next step of variant calling is to map the filtered sequence data to the reference genome of CBDRx. In the package, there are two different functions to for this purpose: *`BWAMEM_Align()`* utilises the external Barrows-Wheeler Aligner software, whereas *`Bowtie2Align()`* makes use of the Bowtie2 algorithm wrapped within R by the `RBowtie` package. For the *Cannabis* data, Bowtie2 was used in the default end-to-end alignment mode, rather than the less strict local mode.
The code used for the alignment and its output are given in Chunk 3. The first two paragraphs report the run time as well as the output file name and location of the unprocessed alignment. The second paragraph indicates which post processing steps were performed, namely converting, sorting, and indexing the raw file format to deliver a ready-to-use .bam alignment file. The final paragraph displays the alignment statistics stepwise, such that the first distinction is made between paired and unpaired reads. Since the data was filtered pairwise, all reads have one mate, with which they can align either concordantly – within expected base pair location distances – or disconcordantly – unexpectedly far away from each other -. In the report, both the percentage and the total number of reads for each subgroup are stated. In the case of Citrix, all data were paired end and 77.25% of all reads were aligned either concordantly (53.07%), disconcordantly (3.15%) or with only one mate (21.03%). The remaining 22.75% could not be aligned at all.

Chunk 3

```
Bowtie2Align(Reference_Genome = "CBDRx.fna",
             Forward_Fastq = "Citrix_Trim_fw.fastq",
             Reverse_Fastq = "Citrix_Trim_rv.fastq",
             Paired_End_Data = TRUE,
             Index_File_Name = "CBDRx",
             Output_Base_Name = "Citrix_CBDRx_Align",
             Threads = 24,
             flag="--rg-id 1 --rg SM:Citrix --rg LB:1 --rg PI:400 --rg PL:ILLUMINA
")
```

Output:

```
## Starting up...
## Previously made Reference Genome index located
##
## Aligning Reads. This will take a while
## arguments 'show.output.on.console', 'minimized' and 'invisible' are for Windows
## only
## Time loading reference: 00:00:01
## Time loading forward index: 00:00:02
## Time loading mirror index: 00:00:02
## Multiseed full-index search: 05:32:33
## Time searching: 05:32:38
## Overall time: 05:32:38
##
## The alignment has finished and has been saved as Citrix_CBDRx_Align.sam
## Storing alignment rates
## Alignment rates have been stored as Citrix_CBDRx_Align_Alignment_Rates.txt
##
## Converting SAM to BAM File
## [bam_sort_core] merging from 474 files and 1 in-memory blocks...
## Sorting BAM File
## [bam_sort_core] merging from 474 files and 1 in-memory blocks...
## [1] "Citrix_CBDRx_Align.bam"
## Indexing BAM File
## Citrix_CBDRx_Align.bam
## "Citrix_CBDRx_Align.bam.bai"
##
## The Alignment Rates Are:
 [1] "329898002 reads; of these:"
 [2] "  329898002 (100.00%) were paired; of these:"
 [3] "    154814900 (46.93%) aligned concordantly 0 times"
 [4] "    85844752 (26.02%) aligned concordantly exactly 1 time"
 [5] "    89238350 (27.05%) aligned concordantly >1 times"
 [6] "    ----"
 [7] "    154814900 pairs aligned concordantly 0 times; of these:"
 [8] "      10378276 (6.70%) aligned discordantly 1 time"
 [9] "    ----"
[10] "    144436624 pairs aligned 0 times concordantly or discordantly; of these:"
[11] "      288873248 mates make up the pairs; of these:"
[12] "        150112046 (51.96%) aligned 0 times"
[13] "        43732877 (15.14%) aligned exactly 1 time"
[14] "        95028325 (32.90%) aligned >1 times"
[15] "77.25% overall alignment rate"

## The entire alignment has been performed successfully!
## The output file Citrix_CBDRx_Align.sam and the corresponding .BAM files can be
## found at /data
```

Visualisation of the mapping quality

In order to further analyse the alignment, the corresponding mapping qualities can be plotted with the `AlignQual()` function as shown in chunk 4. The only input required for the plotting function is the alignment file name. However, to calculate the Lander/Waterman based coverage (C = (number of aligned reads · read length) / genome length), the reference genome length and the read length can be given as well. The calculated coverage is reported in the final paragraph, after the status and outputted file location reports. If the reference length and read lengths are unknown, the parameters can be left at the defaults, in which case no coverage calculation is attempted.

The resulting mapping quality visualisation is shown in Figure 3. For Bowtie2 alignment, the quality scores always follow this distinct pattern with four peaks. Generally, reads are divided into three groups: reads with two or more mapped positions of equal confidence (true multireads), reads with two or more mapped positions where the first location is of higher confidence (semi multireads) and reads that mapped to only one location (unireads). These groups, along with the whole read sequence quality and the number of mismatches in the alignment are used by Bowtie 2 to align a mapping score. The graph of Figure 3A is therefore more complex to analyse. A low mapping quality does not directly correspond to a highly unreliable mapped read that has to be removed from the analysis.
For example, all true multireads automatically receive a minimum score of 0 when there are 5 more mismatches, or they receive a maximum score of 1 when the number of mismatches is less than five. Stating that reads with MAPQ below 1 are of low-quality would inherently mean that all true multi-mapped reads are bad no matter the true alignment score or read quality. In the case of *Cannabis*, such a view would lead to a loss of information, as its genome is known to possess duplicated cannabinoid synthesis genes, due to selection by plant breeders. The presence of such genes in the reference genome could be a cause of true multi-mapping by providing two or more correct mapping locations, depending on the number of gene duplicates.

The suspected influence of duplicated genes is further motivated by the mapping quality patterns as observed for Citrix in Figure 3A. Here, the MAPQ shows four distinct peaks with the largest ranging from MAPQ 0 to 3 consisting mainly of true multi-reads. The second largest peak ranges from 22 to 26 and contains semi multireads with two to three mismatches. Finally, the remaining two peaks are smaller and range from 7 to 10 and 40 to 42, consisting of semi multireads with two to four mismatches and unireads, respectively. In summary, a large portion of the Citrix sequence data consist of true or semi multireads, with a smaller portion consisting of unireads.
An overview of the read requirements for key mapping quality scores as assigned by Bowtie2 is given in Figure 3B for additional clarification.

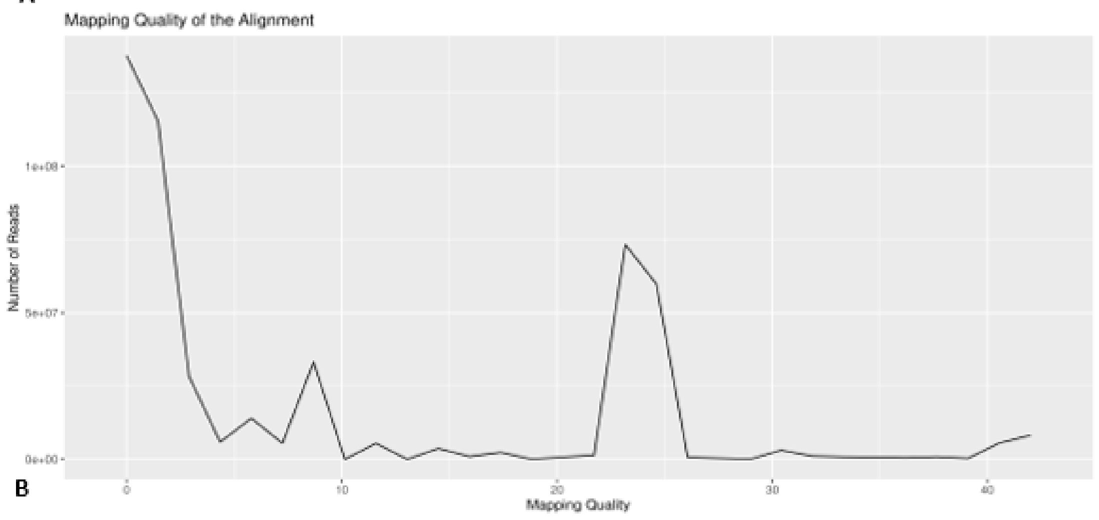
Overall, Figure 3A confirms the findings of alignment statistics of Citrix as shown in chunk 4. The alignment is of high quality with little unexpected behaviour and can therefore be used for variant calling. When the graph of Figure 3A had shown only a single peak at 0 or 42, then the alignment quality would be either unreliable (0) or improbably perfect (42).

Chunk 4

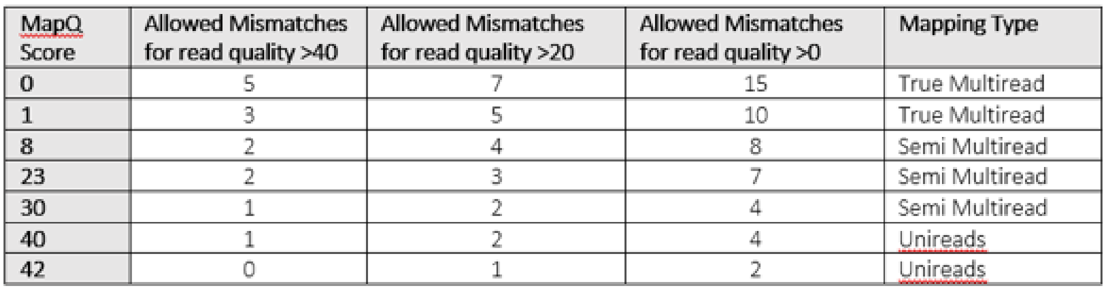
```
AlignQual(Alignment_File_Name="Citrix_CBDRx_Align",
          Reference_Length=736579359,
          Read_Length=150,
          server=T)

Output:
## Starting up...
## Data is being collected
## Plotting quality…
## Warning message:
## Removed 173517755 rows containing non-finite values (stat_bin).
## null device
##           1
## The Mapping Quality Graph has been saved in /data/s1838768 as
## Citrix_CBDRx_Align_Qual
##
## Collecting BAM Statistics
## The General_Alignment_Statistics have been saved as Citrix_CBDRx_Align_Statisti
## cs.txt in /data/
## The coverage has been estimated to be 103 with an alignment rate of 77%
```

A



B



MapQ Score	Allowed Mismatches for read quality >40	Allowed Mismatches for read quality >20	Allowed Mismatches for read quality >0	Mapping Type
0	5	7	15	True Multiread
1	3	5	10	True Multiread
8	2	4	8	Semi Multiread
23	2	3	7	Semi Multiread
30	1	2	4	Semi Multiread
40	1	2	4	Unireads
42	0	1	2	Unireads

***Figure 3**:*
A. The distribution of mapping quality per read where the quality ranges from 0 to 42. All analysed *Cannabis* data follow the same pattern of four major peaks with the largest being at 0-3 and the second largest ranging from 22 to 26.
B. A detailed description of what mapping quality score a certain mapped read can achieve, based on the number of mapped locations, the confidence thereof and the average sequence quality of the read[30]. More mismatches are allowed as the read quality decreases, based on the higher chance of a mismatch being caused by a sequencing error. Because of this, it is important to filter out such error prone sequences before alignment. The mapping quality alone does not provide sufficient information to filter out incorrectly mapped reads and is better suited as an indicator of overall alignment performance.

Variant discovery and joint genotyping

The alignment data can be used to detect small differences between the alternative and reference genome, known as variants. As with the alignment, `VariantCallinginR` offers two different functions to perform this detection. The native R *`VariantToolsCall()`* function is able to collect basic variant data of single samples. The *`GATKCall()`* sends the Unix commands required to perform

variant calling to the external software of GATK and there requires external software in order to collect more extensive variant data. Beyond the additional information, the main advantage of GATK is the g.vcf mode for variant discovery in multiple samples aligned to the same reference genome, making it the preferred function for analysis of the *Cannabis* data.
The execution and output of the variant call is shown in chunk 5. Here, the first three arguments request the file names of the reference genome, alignment data and desired output base name. The arguments *`BAM_Validation`* and `BAM_Repair` can be enabled to ensure that the alignment data has not been corrupted and to potentially repair the corruption by adding new data line flags. In the *Cannabis* case study, the entire analysis was overseen closely and validated externally when required. The validation options have therefore been turned off by inputting '*`FALSE`*'. Furthermore, the file path to the GATK software is defined under *`pathtoGATK`*. The analysis was performed on the ALICE server, where software is installed at non-standard locations. Therefore, the directory in which the *`GATKCall()`* function has to look for the required program is different from the directory on a local machine and has to be made explicit. Were the function executed on a windows operating system, then instead of defining the *`pathtoGATK`*, the *`pathtobash`* would have been made explicit to once again instruct the function on where to send the software commands to. Last is the option to turn on the g.vcf mode. When this option is enabled, GATK will generate an intermediate specialised Variant Call Format (VCF) file for the genome of the inputted sample (G.VCF), which is Citrix in this case.

To obtain the final VCF product all separate G.VCF files have to be merged and joint genotyped, which is performed by the function *`MergeandGenotype()`*, shown in chunk 6. Like *`GATKCall()`*, this function requires the reference genome, output base name and path to GATK software. In addition, all G.VCF files to be merged are defined in a combined character string after the *`GVCF_Files`* argument.

As can be seen in the output for both chunks 5 and 6, GATK provides the user with a lot of information and progress reports. The main messages have been left in the output, such as the GATK version, Java version, the interpretation of the inputted command, run time, overall memory usage and warnings when low quality variants were masked or assigned with missing values. These messages are part of standard GATK reporting and are displayed here as a model for expected output.

Chunk 5

```
GATKCall(Reference_Genome = "CBDRx.fna",
         Alignment_File_Name = "Citrix_CBDRx_Align.bam",
         Output_Base_Name = "Citrix_CBDRx",
         BAM_Validation = FALSE,
         BAM_Repair = FALSE,
         pathtoGATK = "/cm/shared/easybuild/software/GATK/4.1.3.0-GCCcore-8.3.0-Ja
va-1.8/gatk",
         gvcf.mode = TRUE)
```

Output:

```
## Starting up...
## Building BAM Index
 Using GATK jar /cm/shared/easybuild/software/GATK/4.1.3.0-GCCcore-8.3.0-Java-1.
 8/gatk-package-4.1.3.0-local.jar
 Running:
 java -Dsamjdk.use_async_io_read_samtools=false -Dsamjdk.use_async_io_write_samtoo
 ls=true -Dsamjdk.use_async_io_write_tribble=false -Dsamjdk.compression_level=2 -X
 mx100g -jar /cm/shared/easybuild/software/GATK/4.1.3.0-GCCcore-8.3.0-Java-1.8/gat
 k-package-4.1.3.0-local.jar BuildBamIndex -I Citrix_CBDRx_Align.bam
 (...)
 INFO  2020-06-12 18:34:12 BuildBamIndexSuccessfully wrote bam index file /data/Ci
 trix_CBDRx_Align.bai

 [Fri Jun 12 18:34:12 CEST 2020] picard.sam.BuildBamIndex done. Elapsed time: 24.1
 7 minutes.
 Runtime.totalMemory()=2239758336

## Performing gVCF variant calling
 Using GATK jar /cm/shared/easybuild/software/GATK/4.1.3.0-GCCcore-8.3.0-Java-1.8/
 gatk-package-4.1.3.0-local.jar

 Running:
 java -Dsamjdk.use_async_io_read_samtools=false -Dsamjdk.use_async_io_write_samtoo
 ls=true -Dsamjdk.use_async_io_write_tribble=false -Dsamjdk.compression_level=2 -X
 mx100g -jar /cm/shared/easybuild/software/GATK/4.1.3.0-GCCcore-8.3.0-Java-1.8/gat
 k-package-4.1.3.0-local.jar HaplotypeCaller -R CS_CBDRx.fna -I Citrix_CBDRx_Align
 .bam -O Citrix_CBDRx_Align.g.vcf -ERC GVCF --native-pair-hmm-threads 24 –verbosi
 ty=WARNING

 WARN  InbreedingCoeff - InbreedingCoeff will not be calculated; at least 10 sampl
                        es must have called genotypes
 WARN  DepthPerSampleHC - Annotation will not be calculated, genotype is not calle
                         d or alleleLikelihoodMap is null
 WARN  StrandBiasBySample - Annotation will not be calculated, genotype is not cal
                           led or alleleLikelihoodMap is null
 (…)

 [June 14, 2020 6:40:46 PM CEST] org.broadinstitute.hellbender.tools.walkers.haplo
 typecaller.HaplotypeCaller done. Elapsed time: 2,886.53 minutes.
 Runtime.totalMemory()=39886258176
##
## The variant call has been saved in data as Citrix_CBDRx.g.vcf
```

Chunk 6

```
MergeAndGenotype(Reference_Genome="CBDRx.fna",
                 GVCF_Files=c("Arcata_Trainwreck_CBDRx_Align.g.vcf",
                              "Blueberry_CBDRx_Align.g.vcf",
                              "Carmagnola_CBDRx_Align.g.vcf",
                              "Chem91_CBDRx_Align.g.vcf",
                              "Citrix_CBDRx_Align.g.vcf",
                              "Domnesia_CBDRx_Align.g.vcf",
                              "Fedora_CBDRx_Align.g.vcf",
                              "Grape_Stomper_CBDRx_Align.g.vcf",
                              "Harlox_CBDRx_Align.g.vcf",
                              "HeadCheese_CBDRx_Align.g.vcf",
                              "Herijuana_CBDRx_Align.g.vcf",
                              "Tiborszallasi_CBDRx_Align.g.vcf",
                              "Mothers_Milk_CBDRx_Align.g.vcf",
                              "Tahoe_CBDRx_Align.g.vcf",
                              "Master_Kush_CBDRx_Align.g.vcf",
                              "Sour_Diesel_CBDRx_Align.g.vcf",
                              "Sour_Tsunami_CBDRx_Align.g.vcf",
                              "Jamaican_Lion_Female_CBDRx_Align.g.vcf",
                              "Jamaican_LionLMale_CBDRx_Align.g.vcf"),
                 Output_Base_Name="CS",
                 pathtoGATK="/cm/shared/easybuild/software/GATK/4.1.3.0-GCCcore-8.
3.0-Java-1.8/gatk")
```

```
Output:
 Using GATK jar /cm/shared/easybuild/software/GATK/4.1.3.0-GCCcore-8.3.0-Java-1. 8
 /gatk-package-4.1.3.0-local.jar
 Running:
 java -Dsamjdk.use_async_io_read_samtools=false
      -Dsamjdk.use_async_io_write_samtools=true
      -Dsamjdk.use_async_io_write_tribble=false
      -Dsamjdk.compression_level=2
 -Xmx100g -jar /cm/shared/easybuild/software/GATK/4.1.3.0-GCCcore-8.3.0-Java-1.8/g
 atk-package-4.1.3.0-local.jar CombineGVCFs -R CS_CBDRx.fna --variant Arcata_CBDRx
 _Align.g.vcf --variant Blueberry_CBDRx_Align.g.vcf --variant Carma_CBDRx_Align.g.
 vcf --variant Chem91_CBDRx_Align.g.vcf --variant Citrix_CBDRx_Align.g.vcf –varia
 nt Domnesia_CBDRx_Align.g.vcf --variant Fedora_CBDRx_Align.g.vcf --variant Grape_
 CBDRx_Align.g.vcf --variant Harlox_CBDRx_Align.g.vcf --variant HeadCheese_CBDRx_A
 lign.g.vcf --variant Herijuana_CBDRx_Align.g.vcf --variant JLFema_CBDRx_Align.g.v
 cf --variant JLMale_CBDRx_Align.g.vcf --variant Milk_CBDRx_Align.g.vcf --variant
 MK_CBDRx_Align.g.vcf --variant SD_CBDRx_Align.g.vcf --variant ST_CBDRx_Align.g.vc
 f --variant Tahoe_CBDRx_Align.g.vcf --variant Tibor_CBDRx_Align.g.vcf -O CS_temp.
 g.vcf

 12:26:52.747 INFO  NativeLibraryLoader - Loading libgkl_compression.so from jar:f
 ile:/cm/shared/easybuild/software/GATK/4.1.3.0-GCCcore-8.3.0-Java-1.8/gatk-packag
 e-4.1.3.0-local.jar!/com/intel/gkl/native/libgkl_compression.so

 Jun 21, 2020 12:26:54 PM shaded.cloud_nio.com.google.auth.oauth2.ComputeEngineCre
 dentials runningOnComputeEngine
 (…)
 00:07:54.135 INFO  ProgressMeter - Traversal complete. Processed 496378455 total
                                    variants in 300.2 minutes.
 00:07:54.304 INFO  GenotypeGVCFs - Shutting down engine

[June 22, 2020 12:07:54 AM CEST] org.broadinstitute.hellbender.tools.walkers.Genot
ypeGVCFs done. Elapsed time: 300.69 minutes.
Runtime.totalMemory()=5739380736
```

Quality inspection of the acquired variants

The Variant Call Format (VCF) file is specialised to hold variant data efficiently and as a result, the data within is encoded in such a way that accessing the file requires special software and coding. To quickly overview the contents of a VCF file, the *`ScanVcf()`* function is used as shown in chunk 7.
The output of this function provides the contig and sample names, as well as their corresponding number. The first sample in a VCF file is always number 10 as numbers 1 to 9 are reserved for quality variables.
For *Cannabis*, this output is merely a confirmation that joint genotyping has been performed correctly. However, for other data where the samples are unknown or uncertain, this output allows a quick assessment of the VCF file contents.

Chunk 7

```
ScanVcf(vcf_file="CS.vcf")

Output:
## Starting up...
##
 $Chromosomes
      Chromosome
 [1,] "NC_044370.1"
 [2,] "NC_044371.1"
 [3,] "NC_044372.1"
 [4,] "NC_044373.1"
 [5,] "NC_044374.1"
 [6,] "NC_044375.1"
 [7,] "NC_044376.1"
 [8,] "NC_044377.1"
 [9,] "NC_044378.1"
[10,] "NC_044379.1"

$Samples
                                 Sample_Name Sample_Nr
1                                     Arcata        10
2  BlueBerry_Cheesecake_X_Jamaican_Lion_Male        11
3                                 Carmagnola        12
4                                    Chem_91        13
5                                    Citrix         14
6                                   Domnesia        15
7                                     Fedora        16
8                              Grape_Stomper        17
9                                     Harlox        18
10                                HeadCheese        19
11                                 Herijuana        20
12                      Jamaican_Lion_Female        21
13                        Jamaican_Lion_Male        22
14                               Master_Kush        23
15                              Mothers_Milk        24
16                               Sour_Diesel        25
17                              Sour_Tsunami        26
18                                     Tahoe        27
19                             Tiborszallasi        28
##
```

In order to further explore the collected variant data, quality parameters relevant for removal of false positives can be plotted in R with *VariantPlot()*. First, however, the VCF file has to be read into the R memory and decoded into regular numerical and character strings, contained within a list of dataframes. This step has to be executed only once as is shown in chunk 8, after which the R workspace can be saved into a .RData file usable in later sessions.

The first argument of *ReadVcf()* asks for the *vcf_file* name, after which additional arguments are available to read in a select portion of the variant data. Contig dictates whether all chromosomes should be read in (default) or if only one specific contig should be read in. *Nrows* determines how many rows of the VCF file have to be read in and corresponds with the *nrows* displayed by *ScanVcf()*. The default nrows setting is -1 to indicate that all rows are to be read in. In similar fashion, *skip* signifies the number of rows that have to be skipped before reading starts. For example, if only *Cannabis* chromosome 3 has to be read into the R memory, the *nrows = 94670641* and *skip = 206196560*. When only a single sample should be considered, the corresponding sample number can be inputted with *sample_nr*.
For example, the first five variants with only sample specific data from Citrix (*sample_nr=14*), the resulting R object would look like the final output paragraph displayed in Chunk 8. Further access the data within the object can be achieved by selecting the dataframe followed by a specific parameter. For example, *object_name$General_Parameters$CHROM* shows the chromosome positions of all variants.

The arguments for the *VariantPlot()* function requests the VCF R object name, or the RData file in which the object was stored when using the server script, as was done for *Cannabis*. For plotting only one parameter, rather than the entire plot arrangement, the desired parameter name can be inserted after the `plot` argument. Moreover, the generated image can be saved by entering any name after `save_name`. An alternative saving method can be activated when working on a server without graphical interface with the `server=TRUE` argument. To plot all quality parameters of the *Cannabis* dataset and to save the graph whilst on the ALICE server, the code shown in chunk 9 was used.

Chunk 8

```
ReadVcf(vcf_file="CS.vcf",contig="No Default" nrows=-1,skip=0,sample_nr=0,verbose=
TRUE)

Output:
## Starting up...
## Scanning file to determine attributes.
## Reading in data...
## gt matrix initialized.
##  Character matrix gt created.
##  Character matrix gt rows: 17059323
##  Character matrix gt cols: 28
##   skip: 0
##   nrows: 17059323
##  row_num: 0
## All variants processed
## Sorting Info columns
## Sorting Sample Specific columns
## Processed sample 1 out of 19
(…)
## Processed sample 19 out of 19
## Changing into R characters and numericals
##
 $General_Parameters

    CHROM        POS    ID    Type  REF ALT  QUAL     FILTER AC    AF       AN
1 NC_044370.1   1102   NA    SNP   C   G    2139.21    NA    3     0.083    36
2 NC_044370.1   1301   NA    SNP   C   T    1760.86    NA    10    0.263    38
3 NC_044370.1   1327   NA    SNP   G   C    9234.66    NA    19    0.500    38
4 NC_044370.1   1388   NA    SNP   A   T    7531.12    NA    13    0.342    38
5 NC_044370.1   1398   NA    SNP   A   C    23551.59   NA    19    0.594    32
   BaseQRankSum    DP       ExcessHet     FS       InbreedingCoeff MLEAC MLEAF
1   0.156          5392     3.3995        68.821  -0.3221          7     0.194
2   0.530          2123     9.9978        6.522   -0.3877          10    0.263
3  -0.293          2047     51.2979       16.396  -0.9544          19    0.500
4   0.470          1160     18.1852       18.694  -0.6504          15    0.395
5   2.140          911      4.9619        88.734  -0.1062          23    0.719
   MQ       MQRankSum    QD       ReadPosRankSum    SOR
1  27.98    -5.958       4.62     1.050             5.015
2  24.08    -1.126       1.30    -1.066             2.238
3  24.22    -1.627       4.64    -0.095             6.671
4  24.23    -0.704       8.97     1.510             2.694
5  24.49    -1.115       31.28    1.600             5.327

 $Citrix

   GT    AD       DP    GQ   PGT    PID          PL          PS
1 0/0    416,0    416   0    NA     NA           0,0,8514    NA
2 0/0    185,0    185   0    NA     NA           0,0,5865    NA
3 0/1    110,30   140   99   NA     NA           565,0,3335  NA
4 0/0    23,0     23    51   NA     NA           0,51,765    NA
5 0|0    11,0     13    45   0|1    1395_T_TAC   0,45,579    1395
```

Chunk 9

```
VariantPlot(vcf_list="CS_vcf.RData",plot="No Default",
           save_name="CS_vcf_Quality",server=T)

Output:
## Starting up...
## Loading vcf R object...
## Plotting General Variables
## Plotting quality normalised to depth (QD)
## Plotting Fisher Strand Bias (FS)
## Plotting Mapping Quality (MQ)
## Plotting MQRankSum
## Plotting Read Position Ranksum
## Plotting Strand Odds Ratios (SOR)
## Arranging plots
## The plot has been saved as CS_graph.pdf in /data
```

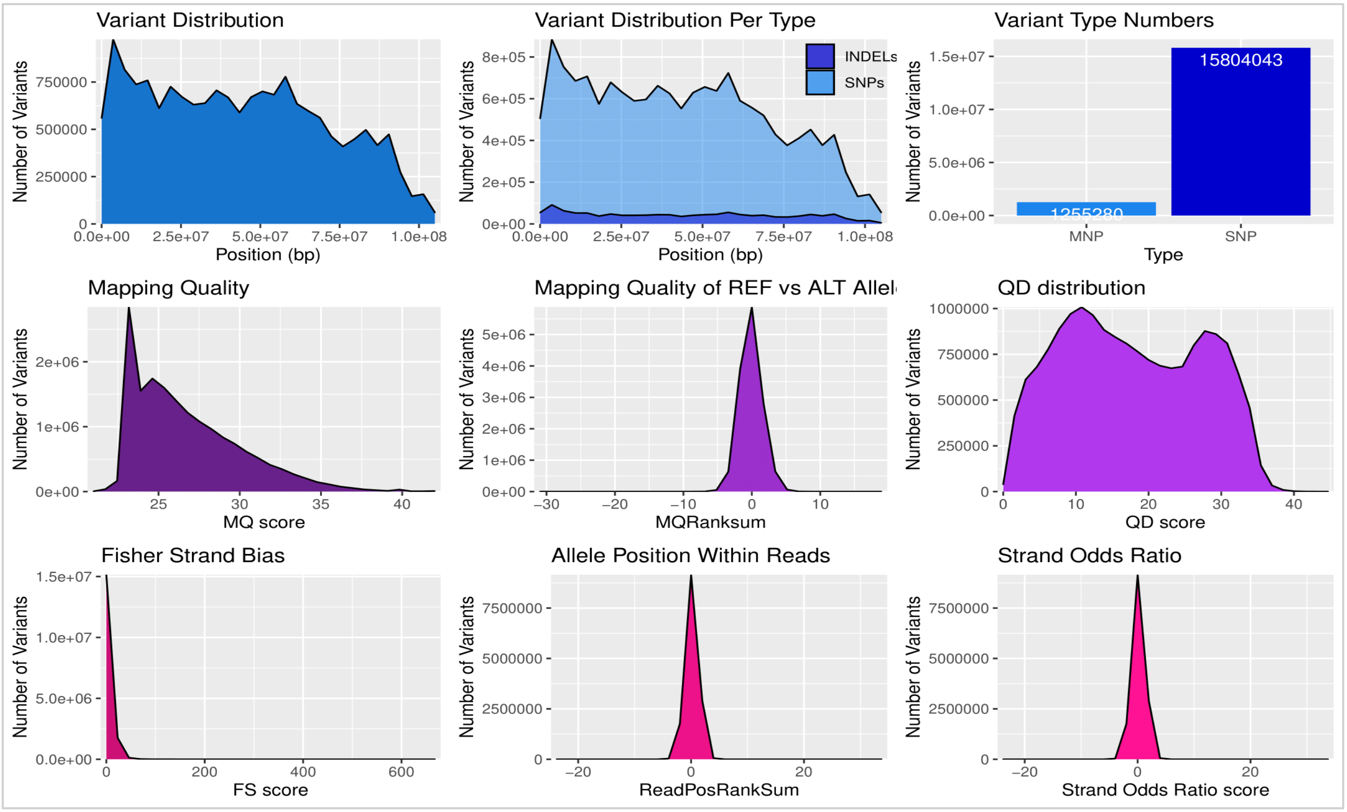


Figure 4: Graph of relevant variant quality variables created with the *VariantPlot()* function.
The first row displays the general variant distribution throughout the contigs, such that each contig starts at 0 bp and ends corresponding with their respective actual base pair length. The division of single nucleotide polymorphisms (SNPs) and microsatellites (INDELs) is shown both in a distribution throughout the contigs and in raw numbers in the bar plot. For *Cannabis*, the variant distribution is relatively even with fewer variants towards the end of contigs and with predominantly SNP variants.
The second row displays the influence of the alignment quality. The mapping quality scores (MQ) are shown in the first plot of the row and seem to differ strongly from the previously made mapping quality plot in Figure 3A. This difference is caused by pooling all samples into one dataset and GATK using stricter criteria for variants with a low MAPQ. As a result, the majority of the variants have a MAPQ of 23 or higher. Following this plot is the MAPQ relation between reference and alternative alleles expressed (MQRankSum). A negative value indicates a higher MAPQ for the REF allele and vice versa. If there is no bias and therefore equal MAPQ scores, the MQRankSum is zero. For *Cannabis*, the majority of variants show little bias, as can be seen by the high peak around zero. Finally, the MAPQ relative to the depth – the number of considered reads that mapped to the variant location- is summarised as the by depth normalised variant confidence (QD). The lower the quality-depth score, the less confidence the variant caller had as to whether a variant is an actual variant or a false positive caused by sequencing or alignment errors. For *Cannabis*, most reads are reliable with a score of five or higher.
The third row displays allele location bias such that heterozygous variants without bias receive a score of zero. The Fisher Strand Bias (FS) determines whether an allele was observed more often on one of the paired reads. For example, a variant found very frequently on the forward read and little on the reverse read will get a penalty that increases the score. The

larger the discrepancy, the higher the penalty and thus the score becomes. Next, the read position RankSum (ReadPosRankSum) looks at the allele location within a read and assigns a penalty when a variant is frequently found near the end of reads, where the chance of sequencing errors is larger. Negative values correspond with biased reference alleles and positive values signify biased alternative alleles. Lastly, the Strand Odds Ratio (SOR) once again detects read bias for alleles similar to the Fisher Strand Bias, but with an algorithm optimised for larger datasets with high coverage. In addition, the SOR score shows what allele is biased by being able to assign negative penalties for biased reference alleles. For *Cannabis*, all the bias plots show sharp peaks around zero, indicating little bias within the discovered variants.

Removal of low-quality variants

The final step is to remove potential false positive variants and to save the data into an accessible file format. The functions used for this goal are `VarianFilter()` and `SaveVcfObject()`, shown in chunks 10 and 11 respectively. Note that the *Cannabis* dataset is too large for xlsx files and the saving is merely shown as an example on how to do so.
The filtering criteria are submitted as a combined character string with the General_filter argument and optionally with the `Sample_Specific_Filter` argument (for genotype, depth, genotype quality and phasing variables). When selection for pure coverage is required, the minimum and maximum allelic depths for all samples can be calculated and considered by inserting the thresholds with the `min_AD` and `max_AD` arguments. Furthermore, variant specific primer development may require a variant-free primer binding location around the potential variants of interest. Thus, the minimal distance between variants can be set with `min_dist`, which will remove all variants that are located too close to each other. This function can therefore be utilised to select high quality variants at regular intervals for genetic mapping.

For the *Cannabis* dataset, basic filtering thresholds advised by the GATK developers were adapted to fit the quality patterns observed with `VariantPlot()` (Figure 4).
To select a minimum value for the mapping quality (MQ), the first high peak is taken as a landmark for the median MQ. Because the assignment of mapping quality differs per alignment algorithm, it is important to keep those rules in mind when filtering. GATK is hard coded to discard all reads with a MAPQ of 0 and thus all true multireads with more than 5 to 7 mismatches were disregarded. The removal of unreliable true multireads is now further enforced by setting MQ> 20 to remove the long sided left tail of the MQ peak. While this may very well exclude variants in duplicated genes, the confidence of these not being false positives is low.
Next is the MQRankSum, the threshold of which has been set as >-2.5. This prevents a bias towards reference, but not towards alternative alleles. The same has been done for ReadPosRankSum>-8. Only a lower boundary has been set remove heterozygous variants that are heavily biased towards the higher quality reference allele.
To further eliminate bias, the fisher strand bias (FS) and strand odds ratio (SOR) thresholds were set to <50 and <3 respectively to eliminate the long-ended tail of read position biased variants.
Finally, the quality relative to the depth (QD) threshold was determined based on the placement of the first major peak. This homozygous variant peak can be seen at QD=11, whilst the second, heterozygous variant peak is located at QD=28. The two distinct peaks are caused by the fact that homozygous variants have twice the number of reads resulting in a twice larger divider for QD=Quality/Depth. The first peak therefore consists of the lowered QD scores of homozygous variants whereas the second peak contains heterozygous variants that can achieve a higher QD score due to inherently smaller dividers. All variants with a QD score smaller than the first peak have a higher likelihood of being a false positive due to the low quality to depth ratio relative to the rest of the variants in the dataset. *Cannabis* has a relatively sharp drop-off left of the first QD peak, consequently a QD>2 criterium was set.

For the `VariantPlot()` graph of the high-quality variants, view Appendix 1 Figure 2.

Chunk 10

```
VariantFilter(vcf_list="CS_vcf.RData",
              General_filter= c("MQ>20",
                                "MQRankSum>-2.5",
                                "ReadPosRankSum>-8",
                                "FS<50",
                                "SOR<3",
                                "QD>2"),
              Sample_Specific_Filter="No Default",
              min_dist="No Default", min_AD=0,max_AD=Inf)
Output:
## Starting up...
## Scanning file to determine attributes.
## Reading in data...
## gt matrix initialized.
```

Chunk 11

```
SaveVcfObject(vcf_list="CS_vcf_filt.RData",
              Output_Base_Name="Cannabis_Sativa_Variants")

Output:
## Starting up...
## Saving General Parameters
## Saving Sample Specific Parameters
## The vcf R object has been saved as Cannabis_Sativa_Variants.xlsx in /data
```

Final results and function performance

In total, over 1000 computational hours were spend filtering, aligning and variant calling nineteen *Cannabis* sativa strains. After filtering, 11.5 million SNPs and 0.8 million INDELs of high quality were collected throughout the genome (table 3).

Table 3: An overview of the high-quality variant distribution throughout the *Cannabis* genome. Overall, the variants are divided equally over all chromosomes with no clear correlation with the chromosome length. There are always more single nucleotide polymorphisms than microsatellites and the type ratio per chromosome does not differ radically.

Chromosome	Contig Name	Number of SNPs	Number of microsatellites
1	*NC_044370.1*	1083046	94206
2	*NC_044371.1*	1088005	101972
3	*NC_044372.1*	1209279	84461
4	*NC_044373.1*	1319871	105177
5	*NC_044374.1*	1154776	83689
6	*NC_044375.1*	1363681	97706
7	*NC_044376.1*	926627	73053
8	*NC_044377.1*	1300957	93927
9	*NC_044378.1*	1104439	77086
10	*NC_044379.1*	1013098	86805
Total	-	11563779	898082

The obtained coverage and run times were found to be strongly linearly related to the compiled FASTQ file size. With the linear equation shown in Figure 5, computation time estimations could be made increasingly accurately beforehand, which were then used to optimised ALICE server use. Furthermore, the file sizes ranged from 3.9 Gb to 48.3 Gb, altering the coverage and run times, but not the alignment rates and general data quality depending on the size (table 4).

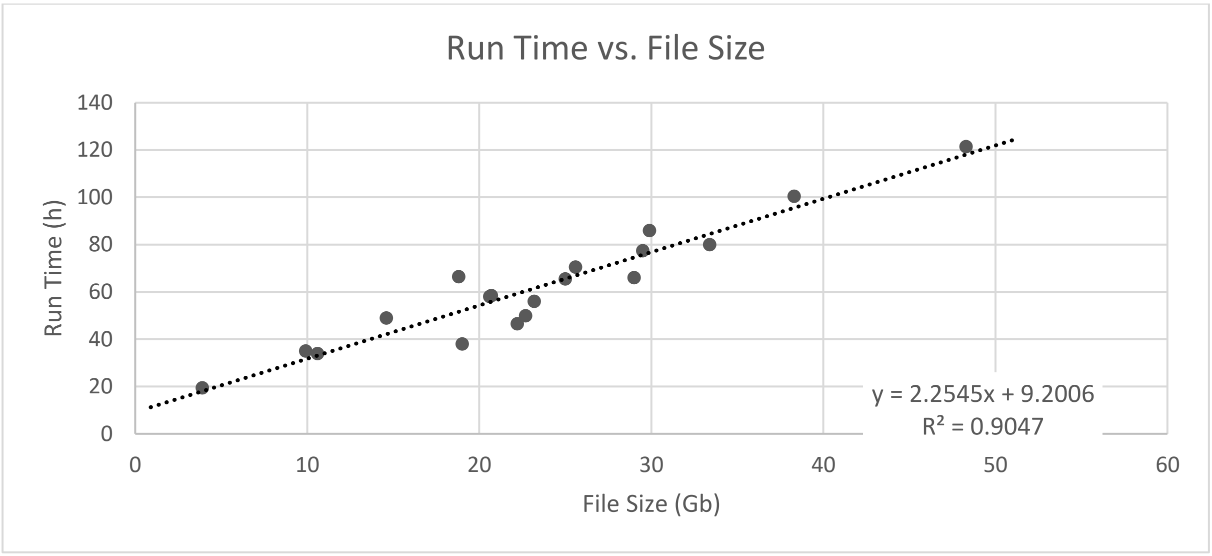


Figure 5 Graph of the linear relationship between total run time from start to finish (g.vcf) and the compiled file size of the raw sequence data per sample.

Table 4 Summary of expected statistics based on the compiled file size and the results from the 19 tested *Cannabis* strains. In addition, the most extreme values within the dataset are shown. The maximum file size, run time and coverage all originate van *Carmagnola*, whereas the three minimum file size, run time and coverage values originate from *Domnesia*. The highest alignment rate was achieved by *Jamaican Lion^3 Mother* and the lowest rate belongs to the *Blueberry Cheesecake x Jamaican Lion Male* outcross, which was also the sample data with the lowest quality.

	Relative to Compiled File size (g-zip)	Maximum	Minimum
Compiled File size	-	48.3 Gb	3.9 Gb
Complete Run Time	1 Gb : 11.5 hours	121.5 hours	19.5 hours
Coverage (Lander/Waterman equation)	1 Gb : 1.3 x	170 x	12 x
Overall alignment	1 Gb : 73%	80.5%	64.6%

Discussion and Conclusion:

The goal of this project was to make variant discovery accessible to researchers with little or no bio-informatics knowledge, in order to allow all to make use of genetic markers in their research. Many researchers are already familiar with R for statistical analyses and therefore know how to handle R functions. The `VariantCallinginR` package succeeds to encompass all major steps of variant discovery in ten simple functions within the R interface. This effectively changes the process of variant calling from specialised software terminal commands into a few understandable lines of code.
The efficacy of the package is supported by the *Cannabis sativa* case study, where nineteen strains were analysed and used for joint variant calling. These previously produced sequence data were successfully pre-processed, aligned and analysed with over 12 million high-quality variants as a result. These data may form the foundation for future research based on genetic mapping, quantitative trait analyses (QTLs), segregation cross selection, and comparisons of genes of interest between various strains.

Despite the implementation of all major variant discovery analyses steps, there are still future improvements to be made to the package. For example, one of the caveats of utilising external software from within the R interface, to perform BWA MEM alignment or variant calling with GATK, is the requirement of installing additional software either in the form of Docker Toolbox or the original BWA and GATK software. This requirement could be eliminated by wrapping the software into a R executable in similar fashion as the Bowtie2 wrapper.
In addition, there is no method in the `VariantCallinginR` package for saving the filtered variant data back into a VCF file or for saving very large datasets to excel files due to memory limitations. Finally, there is no small example dataset available to test the package with. Initially, a random data generation function was written (Appendix 4), but for unknown reasons these data could not align to the corresponding reference genome. Due to time constraints this issue could not be resolved, nor could a suitable subset the demonstrated *Cannabis* data be created in time.

Overall, the `VariantCallinginR` package serves well as a low threshold variant discovery method despite the inherent problems of handling big data. Access to a powerful local system or server such as the ALICE is advisable for working with whole genomes. Nevertheless, smaller analyses for a couple of genes or a small region of interest are perfectly doable on everyday computers.
In the end, bio-informaticians and experienced geneticists may still be better off using specialised software with more options and higher flexibility, but `VariantCallinginR` provides all other researchers with an accessible alternative for recruiting genetic markers for their own research and experiments.

References:

1. Liu, Y. Y. & Harbison, S. A. A review of bioinformatic methods for forensic DNA analyses. *Forensic Science International: Genetics* (2018). doi:10.1016/j.fsigen.2017.12.005

2. Müller, B. *et al.* Improved prediction of complex diseases by common genetic markers: state of the art and further perspectives. *Human Genetics* (2016). doi:10.1007/s00439-016-1636-z

3. Leonova, I. N. Molecular markers: Implementation in crop plant breeding for identification, introgression and gene pyramiding. *Russ. J. Genet. Appl. Res.* (2013). doi:10.1134/S2079059713060051

4. Auton, A. & Salcedo, T. The 1000 genomes project. in *Assessing Rare Variation in Complex Traits: Design and Analysis of Genetic Studies* (2015). doi:10.1007/978-1-4939-2824-8_6

5. Depristo, M. A. *et al.* A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nat. Genet.* (2011). doi:10.1038/ng.806

6. Garrison, E. & Marth, G. Haplotype-based variant detection from short-read sequencing -- Free bayes -- Variant Calling -- Longranger. *arXiv Prepr. arXiv1207.3907* (2012). doi:arXiv:1207.3907 [q-bio.GN]

7. Li, H. A statistical framework for SNP calling, mutation discovery, association mapping and population genetical parameter estimation from sequencing data. *Bioinformatics* (2011). doi:10.1093/bioinformatics/btr509

8. Koboldt, D. C. *et al.* VarScan: Variant detection in massively parallel sequencing of individual and pooled samples. *Bioinformatics* (2009). doi:10.1093/bioinformatics/btp373

9. Koboldt, D. C. *et al.* VarScan 2: Somatic mutation and copy number alteration discovery in cancer by exome sequencing. *Genome Res.* (2012). doi:10.1101/gr.129684.111

10. Liu, X., Han, S., Wang, Z., Gelernter, J. & Yang, B. Z. Variant Callers for Next-Generation Sequencing Data: A Comparison Study. *PLoS One* (2013). doi:10.1371/journal.pone.0075619

11. Sandmann, S. *et al.* Evaluating Variant Calling Tools for Non-Matched Next-Generation Sequencing Data. *Sci. Rep.* (2017). doi:10.1038/srep43169

12. Humburg, P. ChIPsim: Simulation of ChIP-seq experiments. R package version 1.42.0. (2020).

13. Callahan, B. J. *et al.* DADA2: High-resolution sample inference from Illumina amplicon data. *Nat. Methods* (2016). doi:10.1038/nmeth.3869

14. Ginestet, C. ggplot2: Elegant Graphics for Data Analysis. *J. R. Stat. Soc. Ser. A (Statistics Soc.* (2011). doi:10.1111/j.1467-985x.2010.00676_9.x

15. Lawrence, M. *et al.* Software for Computing and Annotating Genomic Ranges. *PLoS Comput. Biol.* (2013). doi:10.1371/journal.pcbi.1003118

16. Davis, T. L. optparse: Command Line Option Parser. R package version 1.6.6. *https://CRAN.R-project.org/package=optparse* (2020).

17. Hahne, F., Lerch, A. & Stadler, M. Rbowtie: An R wrapper for bowtie and SpliceMap short read aligners." R package version 1.28.0.

18. M, M., H, P., V, O. & N, H. Rsamtools: Binary alignment (BAM), FASTA, variant call (BCF), and tabix file import. (2020).

19. Charif, D. & Lobry, J. R. SeqinR 1.0-2: A Contributed Package to the R Project for Statistical Computing Devoted to Biological Sequences Retrieval and Analysis. in (2007).

doi:10.1007/978-3-540-35306-5_10

20. Morgan, M. *et al.* ShortRead: A bioconductor package for input, quality assessment and exploration of high-throughput sequence data. *Bioinformatics* (2009). doi:10.1093/bioinformatics/btp450

21. Tyler, T. W. & Girke, T. systemPipeR: NGS workflow and report generation environment. *BMC Bioinformatics* (2016). doi:10.1186/s12859-016-1241-0

22. Lawrence, M., Degenhardt, J. & Gentleman, R. VariantTools: Tools for Exploratory Analysis of Variant Calls. (2019).

23. Obenchain, V. *et al.* VariantAnnotation: A Bioconductor package for exploration and annotation of genetic variants. *Bioinformatics* (2014). doi:10.1093/bioinformatics/btu168

24. Knaus, B. & Grünwald, N. VcfR: a package to manipulate and visualize VCF format data in R. *vcfr a Packag. to Manip. Vis. Var. call format data R* (2016). doi:10.1101/041277

25. Knaus, B. J. & Grünwald, N. J. vcfr: a package to manipulate and visualize variant call format data in R. in *Molecular Ecology Resources* (2017). doi:10.1111/1755-0998.12549

26. Dirk Merkel. Docker Lightweight Linux Containers for Consistent Development and Deployment. *Linux J.* (2014).

27. McKernan, K. J. *et al.* Sequence and annotation of 42 cannabis genomes reveals extensive copy number variation in cannabinoid synthesis and pathogen resistance genes. *bioRxiv* (2020). doi:10.1101/2020.01.03.894428

28. Grassa, C. J. *et al.* A complete Cannabis chromosome assembly and adaptive admixture for elevated cannabidiol (CBD) content. *bioRxiv* (2018). doi:10.1101/458083

29. R Core Team. A Language and Environment for Statistical Computing. *R Foundation for Statistical Computing* (2018).

30. JohnUrbanGenome. How does bowtie2 assign MAPQ scores? (2014). Available at: http://biofinysics.blogspot.com/2014/05/how-does-bowtie2-assign-mapq-scores.html. (Accessed: 25th June 2020)