Bachelor's thesis

Automatise the process for metagenomics analysis of reads FAIR compliant

Tărnăuceanu Ștefan

Block 3 of Biotechnology Bachelor's Degree

HEH - Department of Science and Technology

HEPH – Condorcet

Academic year 2023-2024

Bachelor's thesis

Automatise the process for metagenomics analysis of reads FAIR compliant

Tărnăuceanu Ștefan

Block 3 of Biotechnology Bachelor's Degree

HEH - Department of Science and Technology

HEPH – Condorcet

Academic year 2023-2024

Dedication

To the memory of my beloved grandfather, Gheorghe Tărnăuceanu, whose wisdom, kindness, and unwavering support continue to inspire me every day. This thesis is dedicated to you, with gratitude for the values you instilled in me and the love you always shared. You are deeply missed and forever remembered.

Acknowledgements

I would like to express my heartfelt gratitude to my supervisor, Erik Bongcam-Rudloff, for the warm welcome and unwavering support throughout my internship at SLU Uppsala. Your guidance and mentorship have been pivotal to my professional growth, and I am deeply appreciative of the invaluable learning experience under your supervision. My sincere thanks also go to Renaud Van Damme, whose assistance and dedication were instrumental in making my internship both rewarding and enjoyable. Your support has significantly contributed to the success of my work during this period. Furthermore, I wish to acknowledge the entire bioinformatics team at the SLU department for their collaborative spirit and invaluable contributions. Your collective expertise and camaraderie made my experience all the more enriching.

I am profoundly grateful to my bioinformatics professor, David Coornaert, who introduced me to the fascinating world of bioinformatics and inspired my passion for this evolving field of science. Your influence has been a cornerstone of my academic journey. I would also like to extend my deepest thanks to Françoise Besanger for providing me with the opportunity to undertake this internship in Uppsala, Sweden. Your belief in my potential has been truly encouraging.

Lastly, I would like to express my deepest appreciation to my family for their constant support and for providing for me throughout this journey. A special thanks to my girlfriend, who has been my main source of support, always encouraging and standing by me during this challenging yet rewarding time.

Contents

List of Figures

List of Tables

Glossary

OTUs : Operational Taxonomic Units
DNA : Desoxyribonucleic Acid
PMA : Propidium Monoazide
NGS : Next-Generation Sequencing
HTS : High Througput Sequencing
PCR : Polymerase Chain Reaction
ddNTPs : Didesoxyribonucleotides Triphosphates
SBS : Sequencing by synthesis
dNTPs : Desoxynucleoside Triphosphates
BCBS : Boran Cattle Breeders Society
bp : base pairs
I : *Bos Indicus*
T : *Bos Taurus*
GTDB : Genome Taxonomy Database
cDNA : complementary DNA
RNA : Ribonucleic Acid
LCA : Lowest Common Ancestor
XML : eXtensible Markup Language
HTML5 : HyperText Markup Language version 5
CSS : Cascading Style Sheets
BAM : Binary Alignment/Map
K2B : Kraken2/Bracken

1 Introduction

1.1 Metagenomics

Metagenomics is a discipline that allows the genomic study of numerous uncultured microorganisms. Compared to genomics, where the data comes from a single organism and makes the sequence assembly and annotation easier to track, metagenomics has a slightly different approach. We know that in genomics we tend to focus on one particular individual and try to remove the possible impurities that may bias the final results. Metagenomics data come from heterogeneous microbial communities that sometimes contain more than 10.000 species, with the sequence data being noisy and partial. Bioinformaticians tend to face new demands in interpreting voluminous, noisy and partial sequence data, starting with the main steps of sampling, extracting, sampling, assembling, gene calling and function prediction[1].

Before proceeding with the shotgun metagenomic analysis assembly-free method, the type of study being processed for this bachelor's thesis, I will explain the different steps of the global metagenomics analysis and the particular steps of the shotgun metagenomics analysis assembly-free method.

1.1.1 Sampling

Sample Size and Number of Samples

A crucial step for a metagenomic study is to obtain enough samples to represent the population in which they are taken. The main issue with this step is that we aren't sure about the number of species that can be found in our environment. To address this issue, we mainly use refraction curves, which allow us to estimate the fraction of species to be sequenced[1].

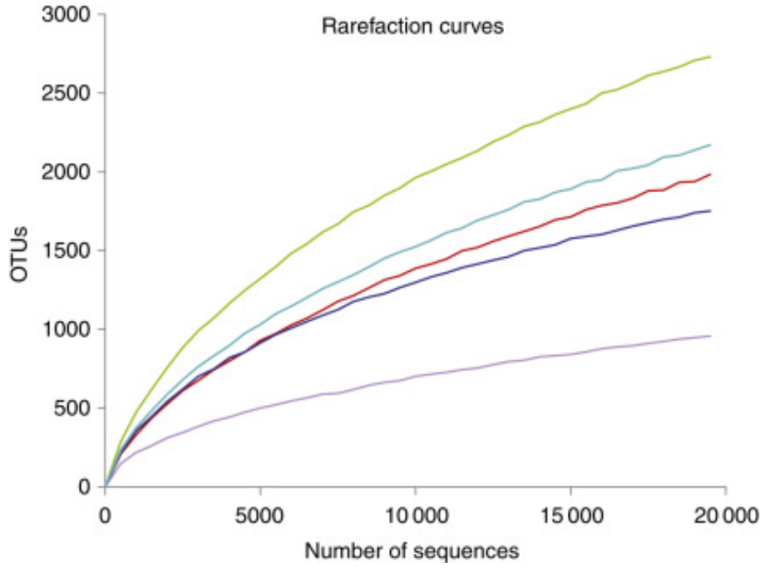


Fig. 1: Refraction curves[2]

Typically, this kind of curve tends to start with a steep slope and then flatten at some point, representing fewer species being discovered. The gentler the slope, the less additional sampling contribution is to the total number of taxonomic units / OTUs discovered. It becomes quite challenging to obtain a good-quality taxonomic classification. To circumvent this problem, researchers will estimate the species diversity through a pilot study or previous studies. They do this to get an idea of the number of samples necessary to get a comprehensive picture of the OTUs present in the environmental sample[1].

The pilot study cited in the previous paragraph is a small-sample and quantitative study carried out as a prelude to a larger-scale study / the study parent. The pilot and the parent study have similar methods and procedures that may reinforce the reasons for conducting the larger-scale study. This pilot study has various purposes, such as developing and testing the adequacy of research instruments, assessing the feasibility of a complete study, designing and testing the protocols for the more extensive study, establishing and testing the sampling and recruitment strategies, collecting preliminary data and obtaining effect size information[3].

Filtering

Once we can determine the maximum number of samples necessary to have a global image of the ecological community from the specific environment, we will pursue a filtering process. The main goal of this step is to isolate specific organisms, such as the host genome and possible contaminants from the environmental samples[1]. The filtering stage can be partially done during the prep of the sample to sequence, but the host removal and possible lab contamination should also be done during the bioinformatic part to completely eradicate any residual presence of the specific organisms.

The filtering stage becomes more important when the host's genome is large and may or may not cover the sequences of the microbial community[2].

Speaking of the fractionation process, we have multiple options when filtering/processing data such as physical fractionation and selective lysis. The physical fractionation is applicable when only a particular part of the community is the target of the analysis such as the viruses in the seawater samples. When we physically "break" the DNA from our samples we will obtain shorter DNA fractions that can be used to assemble the genomes later. This can be confirmed by different computational models that showed that separating one specific individual in multiple fractions, showed improvements in the assembly step of the individual's genome[4].

Meanwhile, selective lysis represents a process of host depletion or removal that uses a different approach from physical fractionation.

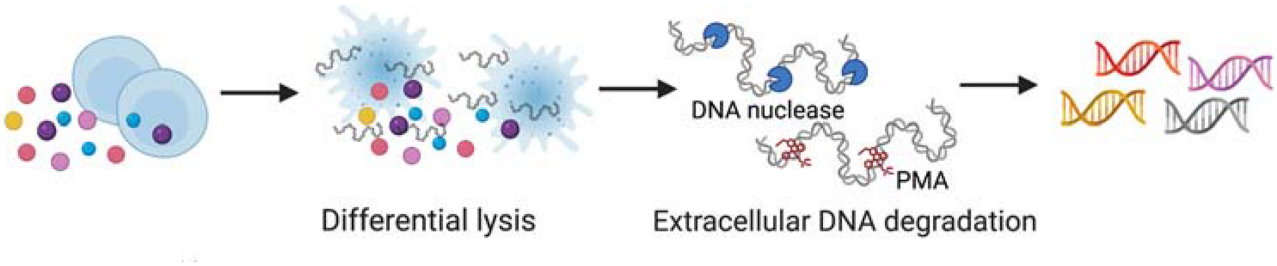


Fig. 2: Workflow of typical host DNA depletion approaches (selective lysis)[5]

When we try to remove the host's DNA via different DNA depletion approaches, we follow the same pattern (visible in Figure 2). We firstly break the cells from our samples with a selective lysis buffer, followed by a DNA nuclease or a PMA treatment.

For this analysis, no filtering step was included in the post-sequencing process.

Recording Metadata

Metadata represents primarily the data we obtain about the samples we gathered. In this data, we can include information such as the locations, sampling times and environmental conditions during sample collection. In other words, this data refers to the physical, chemical and other environmental characteristics of the sample's location. This metadata helps researchers establish correlations between the metagenomic dataset and the habitat-associated metadata, and having it in a specific format is vital to enable comparative studies. The sequences from the metagenomic dataset are more linked to the habitats rather than the species[1].

1.1.2 Sequencing

In the past years, in metagenomics, shotgun sequencing has gradually evolved from the classical Sanger sequencing technology to NGS and now HTS. The sequencing methods are evolving rapidly, and we have been able to go from old-generation sequencing methods to top-notch ones such as the fourth-generation sequencing. Among these techniques, we can distinguish the most used ones, which are Sanger sequencing (first-generation), Pyrosequencing, NextSeq (second-generation), MinION (third-generation) and PromethION (fourth-generation).

Concerning each next-generation sequencing method that I cited above, you can find below an advantages and disadvantages table (see Table 1).

Generation	Advantages	Disadvantages
Second-Generation	• High throughput • High accuracy • Cost-effective • Extensive application support	• Short read lengths • Complex and time-consuming library preparation • PCR amplification biases
Third-Generation	• Long read lengths • No amplification biases • Direct nucleotide detection • Improved *de novo* assembly	• High costs • High error rates in some technologies • Lower throughput • Specialized equipment required
Fourth-Generation	• Single-cell resolution • Detailed spatial gene expression profiling • Suitable for fixed tissues and cells	• Limited to specific applications • Standardization and integration challenges • Complexity in data analysis

Table 1: Advantages and Disadvantages of NGS Technologies[6]

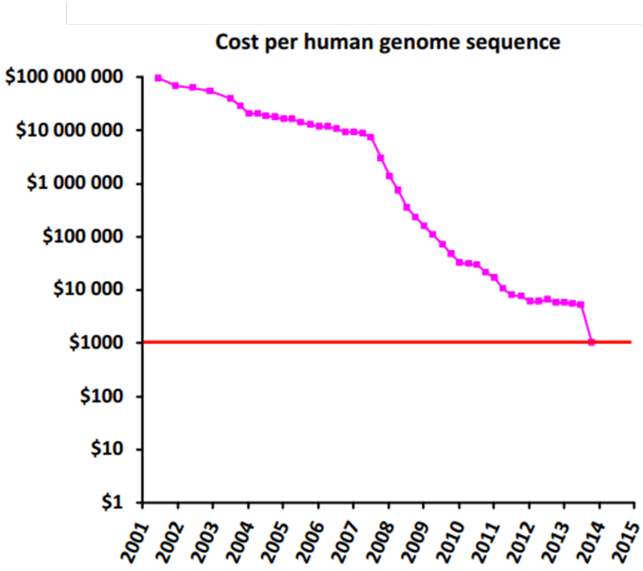


Fig. 3: Evolution of cost of sequencing a human genome from 2001 until today[7]

In Figure 3, we can observe that the general cost of human genome sequencing decreased constantly. This decrease is due to the evolution of the NGS technologies and the improvement of the techniques currently being used. Even though in Table 1 some techniques (Third-Generation) tend to have higher costs than other methods, this increase can be explained by the advantages of the method allowing different uses in different types of analysis.

Building upon the advancements in NGS, High-Throughput Sequencing (HTS) is a next-generation technology that simultaneously enables the rapid sequencing of large amounts of DNA or RNA. It allows for massive

parallelization, producing millions of sequences at once, and can generate both short and long reads. HTS is widely used in fields like microbial ecology and genomics, facilitating the study of community compositions and evolutionary relationships. The technology requires advanced bioinformatics tools for data analysis due to the large volume of generated data[8].

1.1.3 Shotgun sequencing data analysis methods

Regarding the assembly part, we can distinguish two different methods of processing the metagenomic data : the assembly-based method and the assembly-free method.

Assembly-based

The assembly-based method uses multiple approaches, but the most popular one is the de Bruijn graph approach. When we count single draft genome assemblies, the de Bruijn graph is generated by breaking each sequence into overlapping subsequences of a fixed k-size. The size of this k-mer will define the vertices and edges of the graph, leaving the most important task to the assembler. The assembler must find a path through the graph to reconstruct the genome. Unfortunately, this task becomes more complicated when the assembler faces sequencing errors, which may cause potential misassemblies and even the fragmentation of the assembly[6].

Once the sequences are assembled to form contigs, the researchers usually have two different approaches supervised and unsupervised methods. The supervised method of labelling contigs uses databases of already sequenced genomes. On the other hand, the unsupervised method relies more on similarity metrics and algorithms for the assignment. Unfortunately, the most used method is the unsupervised method label contigs due to the large number of unsequenced microbial species[6].

Another important aspect of the assembly-based process is functional annotation. Functional annotation is defined as the process of collecting information about a gene's biological identity, such as its molecular function, biological role(s), subcellular location, and expression domains within the species[9].

Assembly-free

On the other hand, the assembly-free method does not include steps such as binning or functional annotation.

Each one of the methods assembly-based and assembly-free, have their advantages and inconveniences (see Table 2) depending on the purpose of the analysis that's being processed.

The assembly-free analysis offers an overall image of community function and structure by aligning sequences. This method handles large communities efficiently if adequate sequencing depth and reference database coverage are available. Even though it can't resolve genomes of novel organisms without some close relatives, this method is cost-effective for large meta-analyses and doesn't require substantial expert intervention.

	Assembly-based analysis	Assembly-free analysis
Comprehensiveness	Whole genomes, coverage	Community function/structure, mapped sequences
Community complexity	Large communities, partial resolution	Large communities, sufficient depth, reference database
Novelty	Novel organisms, no sequenced relatives	No resolution, unknown relatives
Computational usage	High cost, assembly, mapping, binning	Low cost, large meta-analysis
Expert manual supervision	Requires assistance, binning, scaffolding	No major assistance

Table 2: Key differences between assembly-based and assembly-free analysis[6]

1.2 Multiplexing

Multiplexing and demultiplexing are crucial processes in NGS, allowing simultaneous analysis of multiple samples within a single sequencing run.

Multiplexing

Multiplexing involves combining various different samples into one sequencing run. This is achieved by adding unique molecular barcodes to each one of the samples during the library preparation. These barcodes are short DNA sequences (length of 6-9 nucleotides) that are appended to each sample's DNA fragments[10].

- Library Preparation: during the preparation of the sequencing library, each one of the samples is tagged with a specific barcode. This can be done using Y-adapters. These Y-adapters are double-stranded molecules that have complementary sequences on one side and non-complementary sequences on the other side. The barcode will allow the sequencing machine to easily identify which reads belong to which sample after sequencing
- Pooling Samples: once the samples are barcoded, they are pooled together and loaded onto a single flowcell for sequencing. This approach will maximise the sequencing capacity and reduce costs, as multiple samples can be sequenced in parallel.

Demultiplexing

After sequencing, the demultiplexing process separates the mixed reads back to their respective samples based on the barcodes[10].

- Reading Barcodes: the sequencing output includes both the sequence barcodes and cDNA reads. The demultiplexing software scans the sequencing data for the barcode sequences
- Assigning Reads: this process uses a mapping of barcodes to sample names to assign reads to its corresponding sample. This is done by matching the barcode in the read to the known barcode from the library preparation step
- Output Files: the demultiplexed reads are written in separate fastq files for each sample. In the case of paired-end sequencing (Illumina sequencing), two fastq files will be generated for the cDNA reads (forward and reverse) and one for the barcodes.
- Error Handling: to minimize the impact of sequencing errors, such as incorrect base calls in the barcode, the barcodes are designed to be sufficiently distinct from one another. This will allow for clear identification even with minor mismatches.

1.3 Nanopore reads

The Nanopore reads that are going to be analysed in this bachelor's thesis were provided by one of the first nanopore sequencers, MinION (third-generation sequencing).

This technology (see Figure 4) is using a nanoscale protein pore that serves as a biosensor. The biosensor in question is covered in an electrically resistant polymer membrane. In an electrolytic solution, a constant voltage is applied to this protein pore to produce an ionic current that allows the negatively charged single-stranded DNA or RNA molecules to drive through the nanopore. The translocation speed controlled by a motor protein, allowing the passage of the DNA or RNA molecules at a step-wise rate. The variation in the ionic current during the translocation corresponds to a specific nucleotide sequence present in the region, and this variation is being decoded by various computational algorithms, allowing real-time sequencing of these molecules[11].

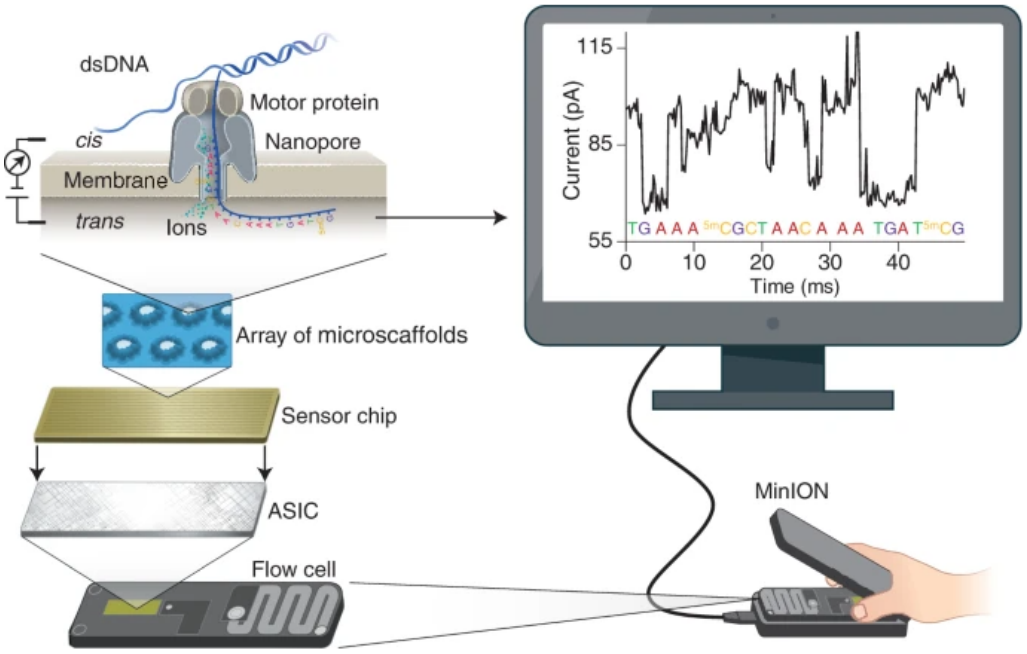


Fig. 4: Principle of Nanopore sequencing[11]

1.4 Illumina reads

Figure 5 illustrates the making process of the Illumina reads. Firstly, we will create a DNA library (A) where we break the genome DNA to form DNA fragments, add adapters at both ends and construct a single-stranded DNA library (ssDNA library). After that, we will add the library to the flow cell, and the DNA fragments will be attached to the surface of the flow cell when passing through it. Then, if we pay close attention to the schematic diagram of bridge PCR (B), we can observe that each DNA fragments is being clustered to its position, and after the amplification, each cluster contains multiple copies of the ssDNA template. The third step, which will allows us to make the connection between the amplification of the DNA fragments (B) and obtaining the final results (D), follows the principle of SBS. The DNA polymerase, primers and four different dNTPs (with specific fluorescence) are added to the reaction at the same time. The dNTPs are connected with an azide group (via their 3' end) that will block the incorporation of the next base, allowing only the extension of a base at a time. Once the dNTPs and the enzyme are washed off with water, photos are scanned, and a chemical reagent is added in order to remove the azide group and quenching fluorescence. Finally, we will visualise the results via a fluorescent signal reception[12].

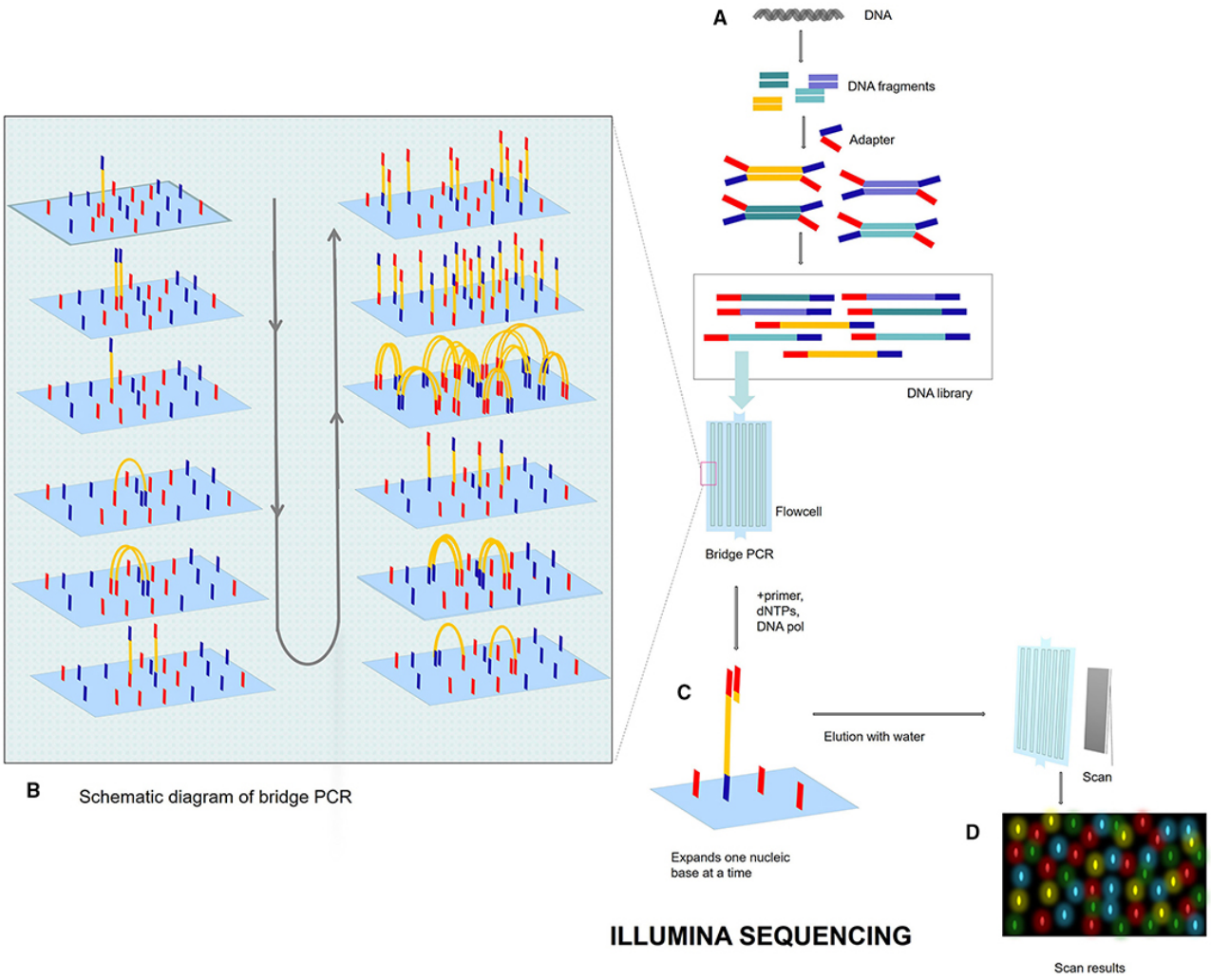


Fig. 5: Illumina sequencing[12]

1.5 Ethiopian Boran

The *Ethiopian Boran* is a boran developed in eastern Africa (Borana plateau in southern Ethiopia). This breed represents the majority in Africa and was the final result of Kenyan BCBS, in order to "improve" the boran[13]. The genetic studies of the International Livestock Research Institute showed that the *Ethiopian Boran* has a quite unique genetic background. The breed is predominantly Zebu, but it also contains a background from two different origins: a European-Near East taurine and an African indigenous taurine[14]. Considering these three influences, mainly Zebu and the two other backgrounds, we can conclude that the *Ethiopian Boran* is a hybrid between *Bos Indicus* (Zebu influences) and *Bos Taurus* (European influences).

To better understand the reasons behind this project subject, let's consider the advantages of the *Ethiopian Boran*. The arguments are the following: breed survival, walking ability, mothering ability, disease resistance and efficiency[14].

In order to survive the harsh conditions in Africa, the Boran cattle have developed a specific drought resistance. Due to their lower maintenance, their bodies can sustain hot, dry conditions. A trait strongly linked to their drought resistance is their capability to walk long distances. Being located in areas where long distances separate grass from water allowed the Boran to evolve under harsh conditions and selection pressure and obtain this specific trait[14].

The second reason cited is the mother instinct. The *Ethiopian Boran* exhibits exceptional mothering abilities, ensuring that her calf receives adequate nourishment, which leads to high weaning weights. Moreover, she fiercely protects her calf from predators, demonstrating her strong maternal instincts[14].

Furthermore, the *Ethiopian Boran* has an efficient disease resistance with a smooth coat and motile skin that will protect them from any buffalo fly infestation and a faster recovering time for other diseases such as : Foot and Mouth Disease[14].

Lastly, this species has a high efficiency when speaking of their longevity. The Boran cows tend to live up to fifteen years and breed regularly healthy calves almost all their life. Speaking of the calves they can gain 50% of their weight in the first nine months of their life. All these arguments will allow the constant sustainability of the herd[14].

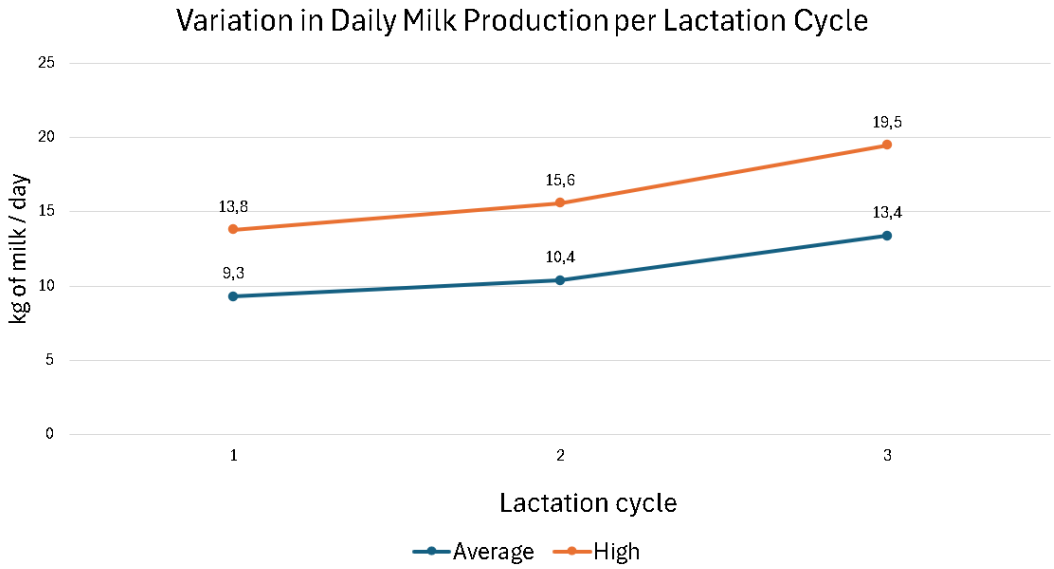


Fig. 6: Variation of the milk production based on lactation cycles[14]

Another aspect of their efficiency is linked to their milk and meat production. Regarding the milk production of the *Ethiopian Boran*, you can find it in Figure 6. We can see that milk production increases on both average and high levels as we collect the product from the cows.

The many advantages of this breed make it a pivotal contributor to meat production in Ethiopia. My research is part of a broader project exploring the correlation between the host genome and microbial composition. The Ethiopian Boran was chosen for this study due to its exceptional traits, genetic diversity, and the unique environmental conditions it thrives in.

2 Methods

2.1 Data

The data I'm analysing comes from two types of sequences: Nanopore (see 1.3) and Illumina (see 1.4). In order to better understand what we are analysing, we are going to understand the metadata (see 1.1.1). The samples we are processing can be mainly divided into two categories based on their extraction period: February and July. The main reason for having samples from two different periods of the year is to observe if there is any variation in the microbiome of the cow's rumen based on their diet. The weather and the available resources strongly influence the cow's diet. February is considered a dry season in Ethiopia. The temperature is gradually rising to its maximum, and the precipitation is mostly missing (33mm/inch); meanwhile, in July, it is quite the opposite with lower temperatures, and the precipitation values demonstrate an increase in intensity (249mm/inch). Based on the weather of these two periods, we can deduce that the diet in February is mainly based on dry vegetation (harder for the cows to digest it) and in the meantime, in July the diet is based on fresher vegetation (easier for the digestion of the cows)[15]. The data for our study was initially collected from a group of 20 cows in the arid southeastern region of Ethiopia. However, the sample size was reduced due to challenging weather conditions, and we ultimately continued the study with seven cows. Importantly, these samples were obtained from 7 different individuals, ensuring a diverse representation despite the reduction in sample size.

Before going much more in depth with our sequences (for further information consult the section A of Appendix) we will give out the names of the sequences for the sample `July0350` :

- Nanopore reads:

  - ONT ID Run_1: `Run_1-July0350`
  - ONT ID Run_2: `Run_2-July0350`
- Illumina reads:
  - Illumina ID Forward: `July0350_S2_L001_R1_001`
  - Illumina ID Reverse: `July0350_S2_L001_R2_001`

2.1.1 Nanopore

The Nanopore dataset is divided into 34 files with a total size of 178.4 Gb uncompressed. The file sequences aren't distributed evenly based on the periods (February and July). This means that we have less information for some periods, due to the inconvenience of the Nanopore sequencing method (see 1.3). Below you can find multiple Nanopore file examples :

- Run_1-July0476.fastq
- Run_1-July0667.fastq
- Run_2-Feb0350.fastq
- Run_2-Feb0407.fastq
- Unclassified_Run_1-Feb0350_July0350.fastq
- Unclassified_Run_2-Feb0350_July0350.fastq

Based on these examples, I will explain the information we can extract from these file names. Firstly, we can identify three categories: `Run_1`, `Run_2`, and `Unclassified`. The `Run_1` category represents the first run for the Nanopore sequences, while `Run_2` is the second run of the same samples using the same flow cell we used to obtain more data. The last category, `Unclassified`, is particular in that it assembles the reads that couldn't be distributed in their respective group during the demultiplexing.

Aside from the run number, we can also distinguish, two crucial pieces of information: when the samples were collected an the individual from whom the sample came. We can distinguish the two periods that I mentioned in the previous section (see 2.1): February and July. Meanwhile, the individuals that I was talking about were classified based on different code numbers, such as `0476`, `0667` and `0350`.

2.1.2 Illumina

The Illumina dataset is divided into 28 files with a total size of 430 Gb compressed. The file sequences are evenly distributed based on the periods. We may also find the forward and the reverse reads for this type of data. The Illumina reads share more or less the same information as the Nanopore reads, which are the periods and the individuals. Below, you can find multiple Illumina file examples :

- WD-3658-Feb0199_S8_L001_R1_001.fastq.gz
- WD-3658-Feb0199_S8_L001_R2_001.fastq.gz
- WD-3658-July0199_S1_L001_R1_001.fastq.gz
- WD-3658-July0199_S1_L001_R2_001.fastq.gz

The first part of the file name, `WD-3658`, represents the Illumina sequencer ID. The second part shows the period and the individual from whom the samples was collected: `Feb0199`. The last piece of information that we can extract is the type of Illumina reads. That last part can be observed by `R1`/`R2`, where `R1` represents the forward reads, while `R2` represents the reverse reads.

2.2 Quality Control

To verify the quality of both Nanopore and Illumina reads, I will employ two quality control tools: `FastQC` and `MultiQC`. This step is necessary to obtain good-quality taxonomic classification results. Impurities such as adapter content, overrepresented sequences, low-quality reads, duplicated reads, and others may introduce biases into the final result and give false information about the samples.

FastQC

FastQC is a popular bioinformatic tool, that provides a report with an overview of basic Quality Control metrics by spotting different issues. This tool is available on GitHub and can even be installed using a conda package : `conda install fastqc`. The reports generated by `FastQC` contain the following information: basic statistics, per-base sequence quality, per-base sequence content, duplicated sequences and over-represented sequences[16].

The first piece of information we get from the `FastQC` report is the basic statistics, which provide a simple overview of the input file, such as file name, total sequences, filtered sequences and sequence length.

The per-base sequence quality report generates a BoxWhisker plot that shows an overview of the range of quality values across all bases at each position of the input fastq/fasta file.

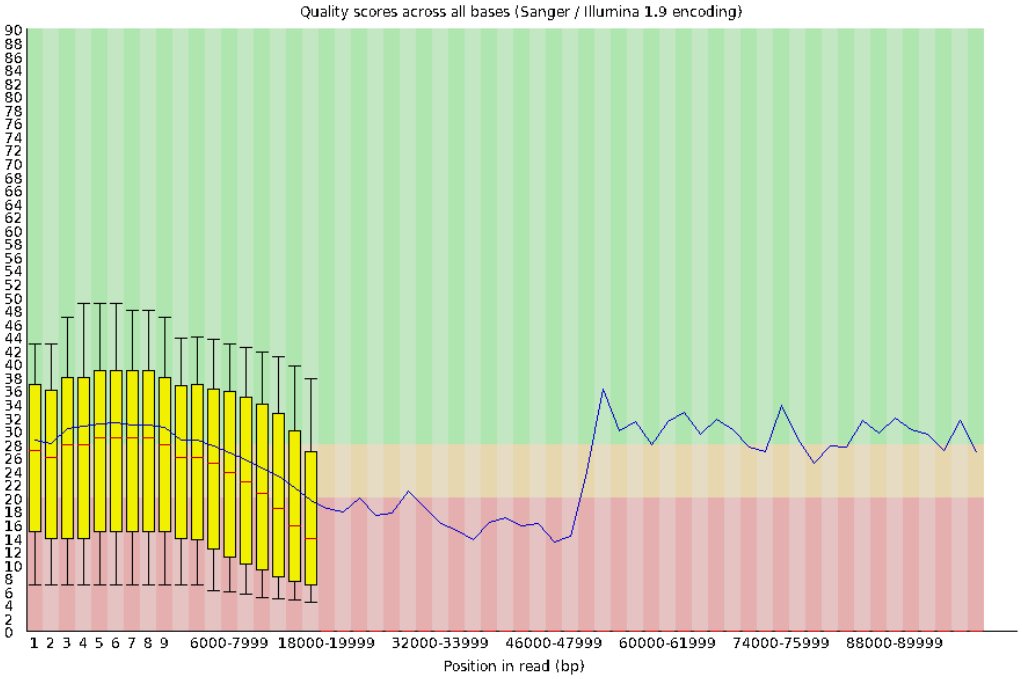


Fig. 7: Per Base Sequence Quality report for Run_1-Feb0199 (Nanopore)

The Y axis in Figure 7 represents the quality score values or the Phred score values. The Phred score indicates the measure of base quality in sequencing. This can be better observed via Table 3. Further explanation concerning the Phred score will be in section 2.3.1 (subsection `Chopper`).

Phred quality score	Probability of incorrect base call	Base call accuracy
10	1 in 10	90%
20	1 in 100	99%
30	1 in 1.000	99,9%
40	1 in 10.000	99,99%
50	1 in 100.000	99,999%
60	1 in 1.000.000	99,9999%

Table 3: The probability values of a Phred quality score and the interpretation of their base-calling accuracy[16]

This table helps us understand that the higher the value of the Phred score, the greater the accuracy of our base calling.

The per-base sequence content module displays a linear graph plot showing the proportion of each base position form the input file for each of the DNA bases (A, T, C, G) that have been called. If there are any remarkable variations between these DNA bases, the issue may come from the sequencer, and one of the best options is to remove the biased bases[16].

The level of duplicated reads is proportional to the quality of the coverage of targeted sequences. A high level of duplication can indicate the enrichment of biases[16].

MultiQC

Compared to FastQC, MultiQC is a tool that mainly collects multiple quality control reports within a single one. This will provide users with a fast way to scan key statistics quickly and easily, obtaining a more accurate comparison between samples and allowing them to detect subtle differences between the samples[17].

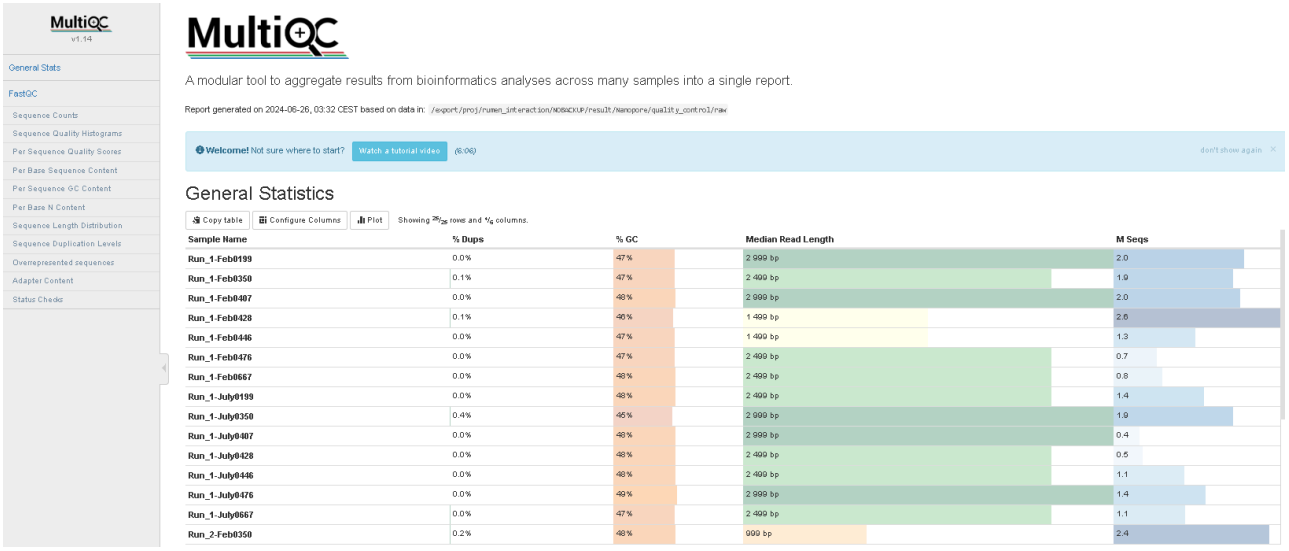


Fig. 8: MultiQC report for the initial Nanopore dataset

2.3 Trimming

Based on the results from the FastQC and MultiQC reports, the users has multiple choices concerning the usage of trimming tools. For this analysis, we used `chopper` for the Nanopore reads and `seqkit` for the Illumina reads.

2.3.1 Nanopore

Chopper

`Chopper` is a Rust implementation of two other softwares : `NanoFilt` and `NanoLyse`. The tool that will be implemented for the trimming step of the Nanopore reads is intended for long read sequencing (PacBio or ONT) and trimming. We can filter / trim by using an average read quality and a minimal/maximal read length and even applying a headcrop/tailcrop[18].

In order to trim/filter the Nanopore reads, I will pursue the analysis exclusively on this specific parameter, the quality score / the Phred score. We chose this parameter

Also, the typical threshold for most people nowadays is between 15 and 20 for Nanopore and 25 and 30 for Illumina, but nothing is written in stone. It's just what most people use to avoid losing too much data while maintaining a "standard." However, in some cases, you have no choice but to go lower (e.g., due to poor data quality or no more possible sequencing). At the same time some people may be stricter because they have the resources or the abundance of data to allow for that.

This step is necessary because when we try to trim the reads based on the quality score, `chopper` tends to remove a certain percentage of the initial reads, keeping only the reads above the chosen threshold. Further details will be developed in the sections: 3.1.1 and 3.1.2. Intending to select the most optimum Phred score, I conducted several tests by varying the quality score to choose the most optimal for the analysis and our available data. To accomplish this step, I ran these tests on one raw sample from Nanopore (July0407) including both runs (Run_1-July0407 and Run_2-July0407).

Before proceeding with the analysis of various Phred scores, I will briefly explain of what a Phred score is. The Phred score is the most used as a standard quality score and can be defined as the following[19] :

$$Q = -10 \times \log_{10}(P) \tag{1}$$

where P represents the sequencing error rate.

During my internship, I conducted multiple tests to evaluate different Phred scores. After thorough experimentation, I concluded that a Phred score of 10 was the most optimal. This value provided a balance between maintaining sufficient data for analysis and minimizing the risk of sequencing errors.

After a brainstorming session with the PhD student Renaud Van Damme, we both agreed to arbitrarily choose the Phred score value of 10. We made this decision because it seemed to be the most optimum value that allowed us to keep sufficient data for the analysis despite the higher risk of error in the sequencing. The quality is less than optimal for thorough analysis. Still, Illumina sequencing and multiple downstream analysis methods will circumvent most, if not all, of the misclassification due to sequencing error.

2.3.2 Illumina

Seqkit

Seqkit operates as a command-line toolkit designed for efficient manipulation of fasta and fastq files. Below, you can find a detailed breakdown of how the tool works[20].

- Command Structure: this software uses a simple command structure of "command subcommand". This concept is based on the usage of the main command `seqkit` and specifies the desired function to use
- Subcommands: it includes 19 subcommands that cover a wide range of string manipulation of fasta and fastq files.
  - Basic operations: validating sequences, getting subsequences and even generating basic statistics
  - Format Conversion: converting between fasta and fastq formats
  - Searching and Locating: finding sequences by patterns or IDs
  - Set Operations: removing possible duplicates, finding sequences and even splitting files
  - Editing and Ordering: editing sequence names and shuffling/sorting sequences
- File Handling: `seqkit` is able to handle both plain and compressed (.gzip) files, allowing for efficient input and output operations
- Performance Optimization: this tool is mainly created in the `Go` programming language (Go is a fast, statically typed, compiled language with clean syntax and powerful concurrency features. It combines efficient performance with the ease of garbage collection and run-time reflection, making it feel as simple as a dynamically typed language[21]) that allows the usage of multiple CPU cores to speed up the process
- Automatic Detection: the toolkit detects the type and sequence type based on the content of the files, simplifying the user experience
- User-Friendly: it is designed to be lightweight and user-friendly, requiring no dependencies or complex configurations for installation

2.4 Host Filtering

In order to ensure the best quality for our dataset, I decided to run multiple test checks and see if any host DNA was left in the dataset. Unfortunately, as we explained in the previous theoretical paragraph (see 1.5) about the *Ethiopian Boran*, the host is a hybrid between *Bos Indicus* and *Bos Taurus*. After rigorous internet research, we couldn't find a high-quality genome of the *Ethiopian Boran*. To ensure the efficiency of this pipeline and the quality of the final results, we decided to work with the host's genomic parents and take into account four possible cases :

- mapping *Bos Taurus* only
- mapping *Bos Indicus* only
- mapping *Bos Taurus*, extracting the unmapped reads and mapping them against *Bos Indicus*
- mapping *Bos Indicus*, extracting the unmapped reads and mapping them against *Bos Taurus*

We ran the following test on two specific cases, `July0407` and `July0428`, for both sequencing types, Nanopore and Illumina to gain time and obtain the maximum information to help us optimize our final pipeline. In the end, we can conclude that for both types of data (Nanopore and Illumina), only using one reference genome (*Bos Taurus*) will allow us to remove most of the host's DNA using less computational resources and gain more time. More information is available in the `Appendix` section (see B).

2.4.1 Nanopore

Minimap2

Minimap2 is a versatile mapping tool and pairwise aligner for nucleotide sequences. It is compatible with short reads ($\geq$ 100 bp in length), assembly contigs ($\geq$ 1 kb genomic reads at an error rate of 15%) , and long noisy DNA and RNA-seq reads[22].

Minimap2 performs a split-read alignment. It employs a concave gap cost for long mutations, such as long insertions and deletions, and inserts new heuristic to reduce spurious alignments. The software is mainly faster than most of the mappers, more precisely, it is $\geq$ 30 times faster than the long-read genomic or cDNA mappers at a higher accuracy[22].

This tool operates by using a seed-chain-align method and you can find below its working mechanism[22]:

- Minimizer Collection: the tool collects minimizers from the reference sequences. These minimizers are short, fixed-length subsequences that are used as seeds for alignment
- Hash Table Indexing: these minimizers are indexed in a hash table. The key in this table is the hash of a minimizer, while the value is a list of locations where that minimizer appears in the reference sequence
- Query Processing: for each query sequence, `minimap2` extracts its minimizers and uses them as seeds to find exact matches/anchors in the reference sequence
- Chaining Anchors: the algorithm identifies sets of collinear anchors or exact matches and forms chains. This chaining process helps in organizing the anchors into a coherent alignment structure
- Base-Level Alignment: if base-level alignment is required. `Minimap2` applies dynamic programming (DP) to extend from the ends of the chains and to close gaps between adjacent anchors. This step refines the alignment by considering the actual sequence data
- Handling Different Data Types: `minimap2` is versatile and can handle various types of data, including short reads, long noisy genomic reads, and even spliced RNA-seq reads
- Performance Optimization: the algorithm is designed to be efficient, using a fast base-level alignment algorithm and an accurate chaining algorithm. It employs a hash table for indexing, allowing quick access and reducing the computational burden compared to full-text indexing methods.

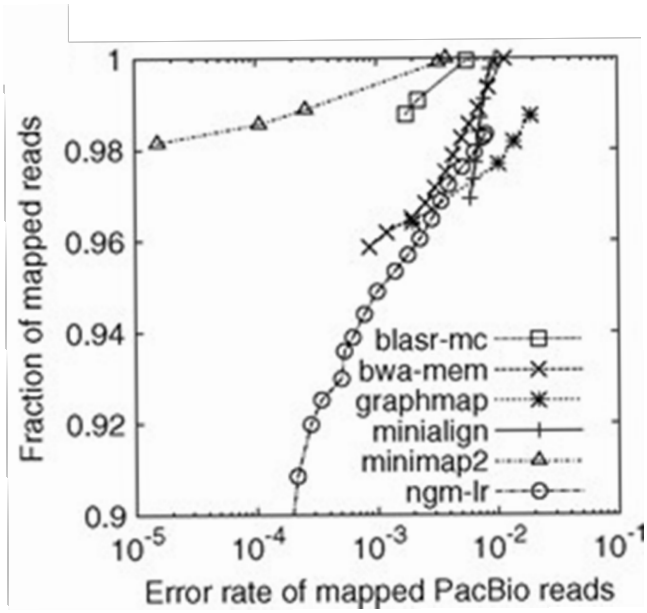


Fig. 9: Error rate of simulated mapped PacBio reads for multiple aligners[22]

Figure 9 represents a solid proof of this choice of mapper for the Nanopore reads. We can observe from the figure above that `minimap2` has high efficiency and sensitivity, error tolerance, and application suitability. Compared to the other tools present in Figure 9, our tool of choice (`minimap2`) is best suited for applications

requiring high precision and accuracy in reads mapping and is able to handle minimal errors while maintaining accuracy.

2.4.2 Illumina

Bowtie2

`Bowtie2` operates through four steps to efficiently align the reads against a reference genome. Here's a breakdown of the process[23]:

- Seed Extraction : this tool begins by extracting a so-called 'seed' substrings from the reads. These seeds serve as initial points for the alignment
- Ungapped Alignment : the seeds are aligned to the reference genome in an ungapped fashion, being assisted by a full-text minute index that allows a quick search
- Prioritization of Seed Alignments : `bowtie2` calculates the positions of the seeds in the reference genome based on the index
- Extension to Full Alignments : the alignment is being extended using SIMD-accelerating programming language. This step allows `bowtie2` to handle gaps and achieve more accurate alignments

2.5 Taxonomic Profiling

As we move forward, we arrive at the taxonomic profiling step, which is the most crucial part of our pipeline. To identify most of the species in our metagenomic samples from both Nanopore and Illumina reads, we will choose a k-mer-based matching strategy. Both `sourmash` and `Kraken2` employ this matching strategy, while `Bracken` uses more of a Bayesian refinement that will be developed later in the thesis[24].

Sourmash

`Sourmash` is a tool for creating, comparing and manipulating MinHash sketches of different genomic data[25].To be more precise, `sourmash` uses FracMinHash sketches for a faster and more lightweight sequence comparison than using only the MinHash sketches[26]. One of the main reasons is that by using MinHash sketches, we will only estimate the Jaccard similarity, while with the FracMinHash sketches, we will ensure both the Jaccard similarity and containment[27]. The FracMinHash sketches, as well as the MinHash sketches, are built to support both Jaccard similarity and containment analyses with members. This new approach with FracMinHash sketches expands the variety of operations that can be done quickly and with low memory usage[26].

The primary purposes of this taxonomic tool are the following[26] :

- genomic and metagenomic analyses should leverage all available reference genomes.
- metagenomic analyses should include assembly-independent methods to avoid biases caused by low coverage or high strain variability.
- both private and public databases should receive equal support.
- the tools should integrate seamlessly with the bioinformatics tool ecosystem, using common installation methods and standard formats
- the tools should be thoroughly tested, well-documented, and adequately supported.

FracMinHash is a bottom-sketch version that supports an accurate estimation of overlap and containment between two sequencing sets. This version is a lossy compression technique used to represent efficiently data sets[28].

The notion of lossy compression means that the techniques being used do not retain all original data but a compressed version of it, allowing multiple operations such as overlap estimation, bidirectional containment and Jaccard similarity[28].

This sketching technique has its advantages and inconveniences[28] :

- Advantages
  - compatible with data sets of different sizes, which is crucial for metagenomic data

  - does not require the original dataset

- Disadvantages
  - the size of the sketches is proportional to the size of the initial dataset and can tend to become quite large

As I developed in the previous paragraphs, FracMinHash uses both Jaccard similarity and containment.

The Jaccard similarity quantifies the similarity between two sets of data to observe which elements are shared and distinct. This similarity is calculated by dividing the number of observations in both sets by the number of elements that are in common in both sets. In other words, the Jaccard similarity can be explained by the following equation[29] :

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{2}$$

where $|A \cap B|$ represents the number of shared elements between the two datasets, A and B, and $|A \cup B|$ is the total number of elements from both datasets. The values for the Jaccard similarity vary between 0 and 1, where 0 signifies that the sets don't share any element, and 1 indicates that the sets are identical. Below is a numerical example that illustrates this concept better[29].

A = {0, 1, 2, 3, 4, 5, 6, 7, 8}

B = {0, 2, 4, 6, 8, 10}

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{\left|\{0,2,4,6,8\}\right|}{\left|\{0,1,2,3,4,5,6,7,8,10\}\right|} = \frac{5}{10} = 0.5 \tag{3}$$

For our numerical example, the two dataset, A and B, have a Jaccard similarity of 0,33, which means that they have 33% of elements in common.

On the other hand, the Jaccard containment is a fraction between the number of common the two sets and the total elements from one of the sets. This value can be calculated by the following formula[30] :

$$C(A,B) = \frac{|A \cap B|}{|A|} \tag{4}$$

where $|A \cap B|$ represents the number of common elements between the two sets, A and B, while A represents the total number of elements from set A.

To better understand the Jaccard containment, we will give an example similar to the one for the Jaccard similarity. The particularity of this notion is the asymmetrical measure that we are obtaining depending on the order of the sets $C(A,B) = C(B,A)$. Below, you can find a numeric example that will better explain the Jaccard containment.

A = {0, 1, 2, 3, 4, 5, 6, 7, 8}

B = {0, 2, 4, 6, 8, 10}

$$C(A,B) = \frac{|A \cap B|}{|A|} = \frac{\left|\{0,2,4,6,8\}\right|}{\left|\{0,1,2,3,4,5,6,7,8\}\right|} = \frac{5}{9} = 0.55 \tag{5}$$

$$C(B,A) = \frac{|B \cap A|}{|B|} = \frac{\left|\{0,2,4,6,8\}\right|}{\left|\{0,2,4,6,8,10\}\right|} = \frac{5}{6} = 0.83 \tag{6}$$

We can observe that both equations 5 and 6 illustrate the percentage of elements from one set that are contained in another. For example, in equation 5, we cam find only 55% of the elements from A in set B, while in equation 6, we can find 83% of elements from series B in series A.

For the next part of this sub- point, we will explain the parameters that were chosen for each one of the sequences. These parameters were tested during the internship and were best chosen to optimize the pipeline that was developed for this bachelor's thesis. For further information, please consult my internship report. The only parameter that was submitted to multiple tests was the scaling factor value from `sourmash`.

`Sourmash` uses the resemblance (Jaccard similarity) to assess the overall similarity between the sequence and the database and containment to assess how much one sequence is within the database. Setting appropriate thresholds decides when two sequences should be considered a match.

The scaling value of `sourmash` implements a method that is inspired by modulo sketches to dynamically scale hash subset retention size. When we use the scaled signatures, we choose a scaling factor that divides the hash spaces into bands equal to the scaling factor value. These bands are retained within the minimum band as the sketch and can be converted into standard bottom-hash signatures only if the subset retention's size is equal or inferior to the number of hashes in the scaled signatures[31].

The scaling factor in the context of the FracMinHash sketch is a changeable parameter that modifies the size of the sketch. It plays a crucial role in determining the resolution and accuracy of the metagenomic analysis. FracMinHash estimates the containment index with the help of the scaling factor. This parameter influences the number of k-mers that are included in the sketch, which will directly affect the performance of estimating the taxonomic composition of the metagenomic samples. Increasing this value will result in a larger sketch that includes more information about the metagenome, potentially leading to more accurate estimates at the cost of more computational requirements. There should be a balance between the desired accuracy and the computational resources available[32].

The database was the second parameter that was mandatory for the taxonomic profiling step. The respective databases are :

- GenBank
- GTDB

During my internship, after a quick debriefing with the PhD student Renaud Van Damme, we both agreed to check the databases and choose the most optimum one for this analysis. At first, GenBank's database was the initial answer due to its sizes and variety, but unfortunately, this database contains wrongly taxonomic assigned sequences (>80%) with an important degree of contamination and even submission errors that are submitted to GenBank as 'UNVERIFIED'[33]. Compared to GenBank, GTDB seems to be a more suited option for this metagenomics analysis because the database is regularly submitted to quality controls with CheckM (an automated method used to estimate the completeness and contamination of a genome using the marker genes[34]) and that is mainly composed of draft genomes from metagenomes and singles cells which respects better the conditions of a metagenomic sample[35]. Once we confirmed that GTDB was the better option, we will now explain what type of GTDB we employed for this bachelor's thesis: `GTDB R07-RS207`[36] that is containing 318.000 genomes in total with everything compressed in a 9,4 Gb (see G.23). Based on the name of the database, it specifies the fact that this database is the seventh release of GTDB and the 207^{th} reference set within the release.

One of the last crucial parameters linked to the database is the k-mer length. Regarding this parameter, based on the website of `sourmash`, we had the choice between 3 k-mer size: 21, 31 and 51. Choosing a low k-mer length, such as 21, would increase the chance of false positives and result in poor taxonomic profiling. Meanwhile, the other two values, 31 and 51, are giving more stringent matches with fewer errors. The only thing that separates these two values is the usage of computational resources, and the option that requires fewer resources is the k-mer with a size of 31[37].

In the following paragraphs I will briefly explain the reasons of why choosing specific values for the scaling score, before concluding this sub-section and enumerate the key parameters to use with `sourmash`.

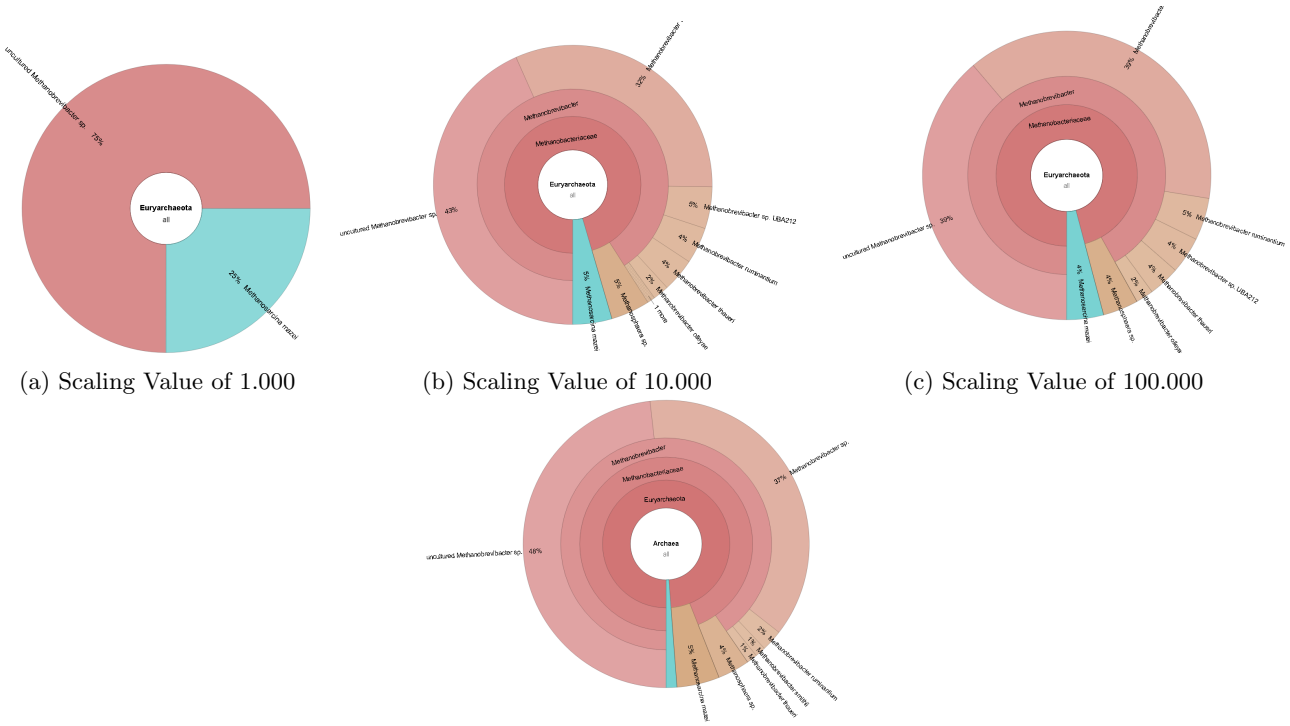
Nanopore

During the internship, we investigated the influence of the scaling factor on taxonomic classification for the Nanopore dataset. We followed the hypothesis that increasing the scaling value would enhance the quality of taxonomic profiling results. We tested various scaling values, including 1.000, 10.000, 100.000 and 1.000.000. The corresponding taxonomic classification results are displayed in Figure 10.

Based on Figure 10, we concluded that increasing the scaling value results in better and more complex taxonomic profiling. We also considered the percentage of unclassified reads at each scaling value to determine the optimal scaling value for the Nanopore dataset. As shown in Table 4, the percentage of unclassified reads decreased with increasing scaling values but reached a threshold of approximately 99.35% between scaling values of 100.000 and 1.000.000.

Scaling Value	Percentage of Unclassified Reads
1.000	99,5%
10.000	99,45%
100.000	99,34%
1.000.000	99,36%

Table 4: Percentage of unclassified Nanopore reads based on the scaling value



(a) Scaling Value of 1.000 (b) Scaling Value of 10.000 (c) Scaling Value of 100.000

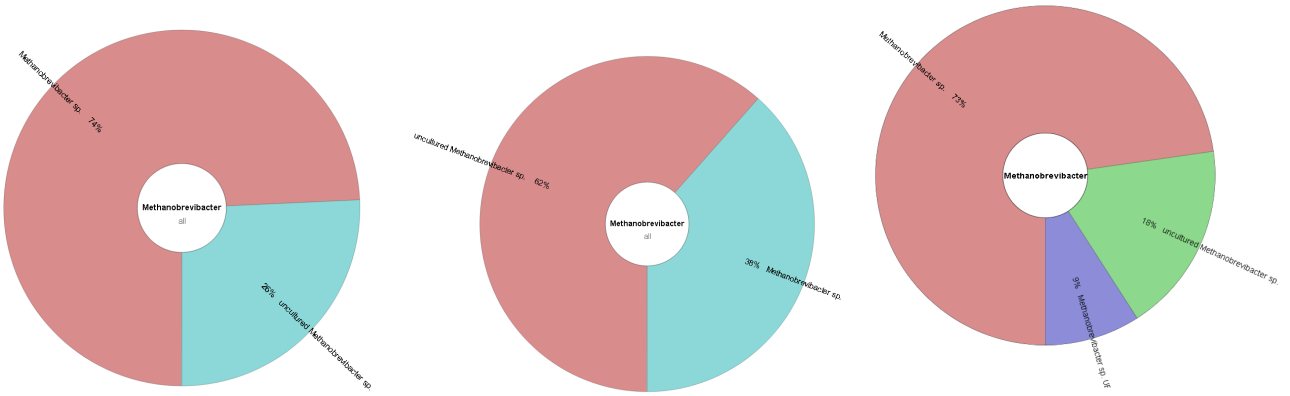
(d) Scaling Value of 1.000.000

Fig. 10: Krona reports for different scaling values on the Nanopore dataset

Additional test were conducted using a specific database (GTDB) to find the final value, which confirmed that `sourmash` has a threshold scaling factor value of 1.000.000. However, based on these observations, the optimal scaling value for the Nanopore dataset was determined to be 100.000, balancing both accuracy and computational efficiency.

Illumina

For the Illumina dataset, multiple tests were conducted using simulated data containing 10 *Archaea* species, including *Methanobrevibacter smithii ATCC 35061*, *Methanothermobacter thermautotrophicus*, and *Methanococcus maripaludis*. The scaling values tested were 1.000, 10.000, and 100.000 and the corresponding taxonomic classification results are shown in Figure 10.



(a) Scaling Value of 1.000 (b) Scaling Value 10.000 (c) Scaling Value 100.000

Fig. 11: Krona reports for different scaling values on the simulated Illumina dataset

Based on the analysis of these tests, the optimal scaling value for the Illumina dataset was determined to be 30.000. This value was found to offer the best trade-off between accuracy and computational efficiency for taxonomic classification.

In order to conclude this subsection here are the elements that are going to be used for the taxonomic profiling with `sourmash` :

- scaling value for :
    1. Nanopore reads: 100.000
    2. Illumina reads: 30.000
- database of usage: GTDB
- k-mer length: 31

Kraken2

`Kraken2` operates as a metagenomic classification tool that assigns taxonomic labels to sequence reads using a k-mer-based approach, enhanced by multiple innovations. Below, you can find a more detailed breakdown of the logic behind this tool[38]:

- Data Structure: `Kraken2` utilizes a probabilistic, compact hash table to store minimizers (short genomic pieces) and their corresponding LCA taxa. This structure is more memory-efficient than the traditional sorted list method used by `Kraken1`, allowing for a significant reduction in memory usage (about 85%).
- Minimizers: instead of sorting all k-mers from the reference sequence library, `Kraken2` only stores the minimizers which will reduce the size of the database and speed up the classification process
- Classification Process: when we classify a sequencing read, `Kraken2` extracts the minimizers from the read and compares them against the stored minimizers in the database. Each one of them is then mapped to its corresponding LCA taxon, allowing `Kraken2` to determine the taxonomic classification of the reads based on the minimizers
- Translated Search Mode: this tool introduces a new translated search mode that uses a reduced amino acid alphabet, increasing viral dataset sensitivity. By using this newly made mode it will only enhance `Kraken2`'s applicability in viral metagenomics
- Thread Scaling and Performance: `Kraken2` implements block and batch-based parsing techniques to improve thread scaling and allow it to utilize multiple CPU threads during classification. This improvement can be proved by a substantial increase in the processing speed and classifying millions of reads per minute.
- Integration with `Bracken`: `Kraken2` can be paired up with `Bracken` once we obtain the classification results. This tool employs a Bayesian algorithm to estimate species and genus-level sequence abundances based on the classification results of `Kraken2`. This integration enhances the accuracy of abundance estimates in metagenomic studies

Once we developed the functionality of `Kraken2`, we will explain the choice of the database that is going to be used by both `Kraken2` and `Bracken`. Concerning the database[39], we will employ a standard version (released on the 5^{th} of June 2024) of the Kraken database. This database includes a comprehensive collection of reference sequences, containing RefSeq data for *Archaea*, *Bacteria*, *Viruses*, and *Plasmids*, as well as human genome sequences and the UniVec Core database (helps identify and remove vector contamination). The database is highly compressed, with a total size of 78 Gb, making it an efficient tool for accurately and rapidly classifying metagenomic sequences.

Bracken

`Bracken` (Bayesian Re-estimation of Abundance after Classification with Kraken) works by estimating the species abundances in diverse metagenomic samples through a probabilistic redistribution of reads within a taxonomic tree. Below you can find an explanation of how `Bracken` operates[40] :

- Initial Classification with `Kraken` or `Kraken2`: this tool relies on the taxonomic assignments made by `Kraken2` (in our case) classifies the metagenomic dataset into various taxonomic levels
- Probabilistic Redistribution: after finishing the classification with `Kraken2`, `Bracken` redistributes the reads assigned to higher taxonomic nodes down to species level. Behind this process, a probabilistic model estimates how many reads should be allocated to each species based on the overall distribution of reads across the taxonomic tree.

- Handling Ambiguities: for example, if we encounter an issue where an individual is classified at a higher taxonomic rank, `Bracken` uses the probabilities derived from the `Kraken` database to allocate these reads to a more specific taxonomic rank. This process works the other way: if an individual is wrongly assigned at a lower level, it will be reassigned to its parent rank.
- Thresholding: in order to avoid possible false positives, `Bracken` allows users to set a threshold for the minimum number of reads required for a species to be considered in the final abundance estimates
- Output: the final result of this bioinformatic tool is a set of abundance estimates for the specified taxonomic level. This estimation can be more accurate than the initial classification done by `Kraken2`

2.6 Data Visualization

Krona

`Krona` is a visualization tool that works by utilizing a combination of XML data structures and web technologies to create an interactive output for metagenomic data. Here's a quick breakdown of how it functions[41] :

- Data Structure: `Krona` uses XML to store the hierarchical data. Each node of this hierarchy represents a taxonomic classification and attributes a specific magnitude and color for each one of the nodes. Following this method will allow the user to have a clear representation of relationships and abundances within the data.
- Web Technologies: the visualisation is created using HTML5, JavaScript, and CSS, making it platform-independent and easily accessible on web browsers. This means that the user can freely use `Krona` without any additional software to install
- Radial Space-Filling Display: the visualization with `Krona` reassembles a pie chart that incorporates an embedded hierarchy. Each sector represents a taxonomic group, while the angle of each sector represents the magnitude of the data, showing the relative abundance at a glance.
- Interactive Features :
  - Zooming: users can zoom in and out easily on specific nodes in order to perform a more in-depth analysis of the visualization report
  - Filtering and Searching: a search function enables the users to filter nodes by name and obtain more specific information about their search
  - Color Coding: the nodes can be colored based on quantitative or categorical variables
- Integration and Sharing: each `Krona` chart is contained in a single file, which is easy to share and does not require any internet connection

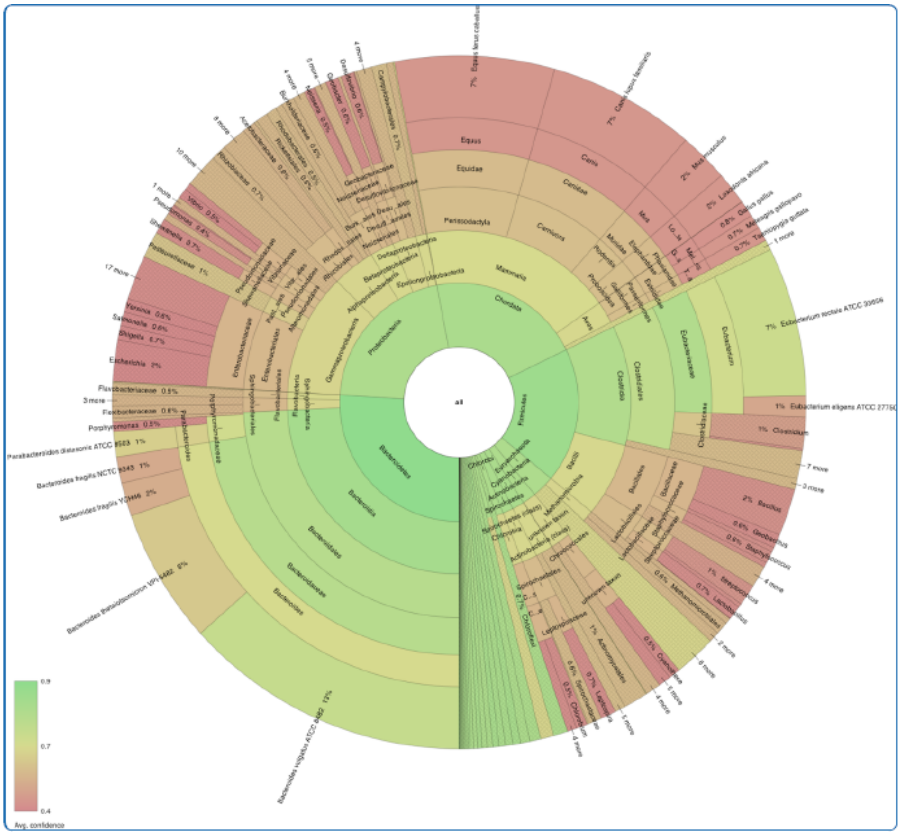


Fig. 12: Example of Krona output[42]

Pavian

`Pavian` is a web application that was mainly developed for the visualization of metagenomic data. Let's break down this tool and understand the mechanisms behind it :

- Data Input: `Pavian` can take in as input results from various metagenomic classifiers such as `Kraken`, `KrakenUniq` and `Centrifuge`.
- Visualization :
  - Sankey Diagrams: the default visualization is a Sankey diagram that illustrates the flow of read from the main taxonomy level to more specific ones. The width of those flows represents the number of reads specific to each of the lower ranks.
  - Sample Comparison Tables: the new identification results from multiple samples can be viewed in a tabular format, which will allow users to easily compare them.
- Alignment Viewer: this feature of `Pavian` uses BAM files that can show the distribution of reads across genomes. It is beneficial to employ this feature because it can reveal potential contamination.
- Interactive Features: this application is mainly built using `R` (a scripting language for statistical data manipulation and analysis[43]) and `Shiny` (a web application framework dependant on several `R` packages[44]) with some JavaScript incorporated in it, which will allow the users a more interactive data analysis. They can filter, sort or/and manipulate their data in order to gain insights into their metagenomic dataset.
- Accessibility: this program is accessible through a web browser, making it easier for researchers to employ it. It can also be used on a server or locally through an `R` environment.

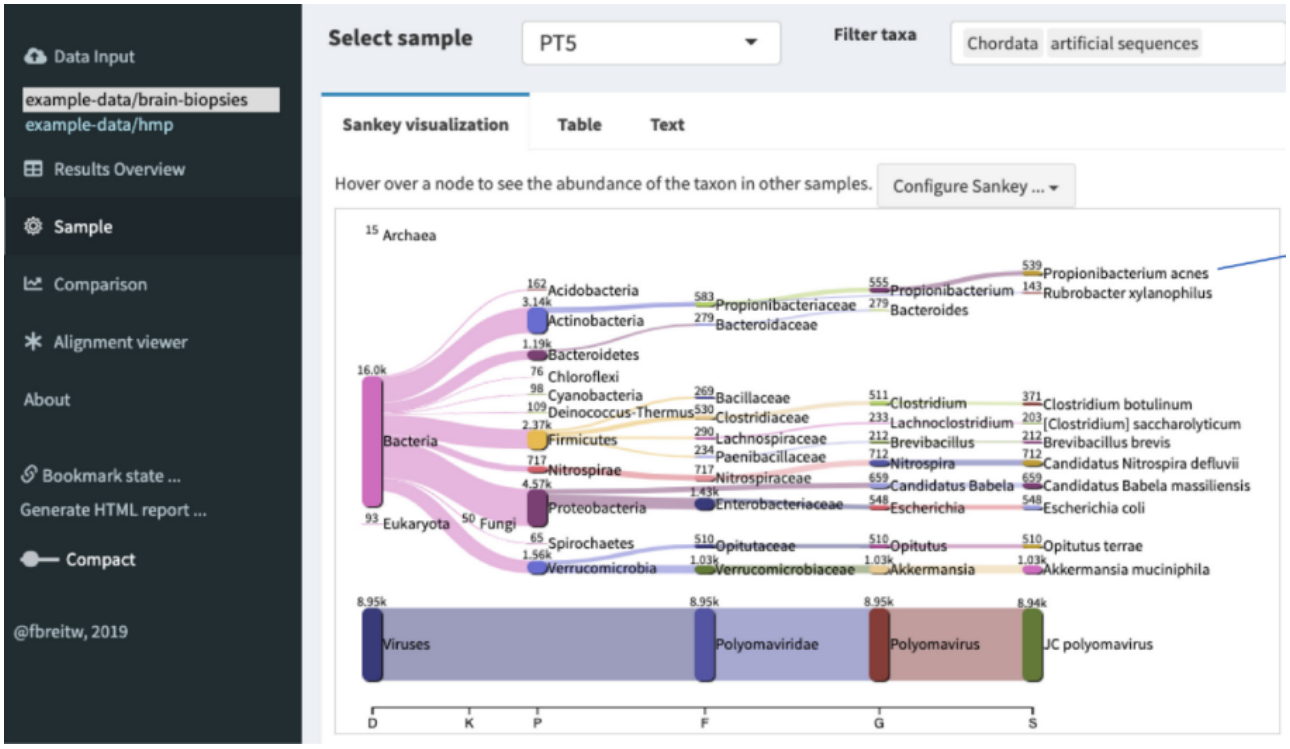


Fig. 13: Example of Pavian output[45]

2.7 Scripting

For my bachelor's thesis, I utilized Python alongside several Bash scripts to obtain the results of this shotgun metagenomic analysis. Multiple Bash scripts were necessary to verify the results at various stages of the process, ensuring accuracy and consistency throughout the project.

2.7.1 Bash Scripting

Before explaining Bash scripts, I will briefly explain Bash (Bourne Again Shell). Bash is a command interpreter that allows users to give commands to their operating systems interactively or to execute commands in batches (scripts)[46].

Bash scripting is a powerful and flexible tool for automating system administration tasks in Unix/Linux systems. This type of scripting includes multiple advantages such as[47] :

- Automation: these scripts will allow the automatizing of repetitive tasks and reduce the time consumption and the risk of errors
- Portability: the bash scripts can be run on various platforms and operating systems (Linux, macOS and Windows)
- Flexibility: they are highly customizable and can be adapted to specific requirements. These scripts can be combined with multiple programming languages, enhancing their efficiency
- Accessibility: these scripts don't require specific tools and can be edited with any text editor
- Integration: as they can be integrated with multiple programming languages, they can also be integrated with other tools such as, databases, web and cloud servers
- Debugging: the Bash scripts are easy to debug and have error-reporting tools that can help identify and fix potential issues quickly

2.7.2 Python Scripting

Python is a high-level, general-purpose, and dynamically typed programming language. It supports multiple programming paradigms, including structured, object-oriented, and functional programming[48]. For this bachelor's thesis, I used version `3.10.13` of Python alongside multiple libraries :

- sys: this module provides various functions and variables that are being used to manipulate different elements of the Python runtime environment[49]
- getopt: is a library that can be used as a parser for command line options whose API is designed to be similar to the users of the C `getopt()` function[50]
- matplotlib: this library plays a crucial role in data visualization by providing a wide range of tools and functions that are able to create static, animated or interactive plots and charts[51]. The tool used from this library is : `pyplot`
- os: is a module that provides a portable way of using operating system–dependent functionality[52] like creating files and directories, environment variables and process management[53]
- pandas: a fundamental Python library well-known for its capabilities in data manipulation and analysis, especially in data cleaning, transformation, and aggregation[51]
- seaborn: represents a better visualization library compared to matplotlib and includes a more extensive variety of plot types[51]
- argparse: this module of Python helps create a program in a command-line environment, that improves interaction and can handle the errors made by the users[54]
- re: a library that provides regular expression matching operations similar to those found in `Perl`[55]
- numpy: critical Python library widely used for numerical computing by providing massive support for large, multi-dimensional arrays and matrices alongside a vast collection of high-level mathematical functions to operate on these arrays[51]

3 Results

For the most important part of this bachelor's thesis, I will divide this section, into two categories: the automated analysis of the Nanopore data and the Illumina data. Seeing that we had 2 types of data, we did the same general analysis utilizing specialized software for each data type. Firstly, we will run through the Nanopore dataset and continue the Illumina reads.

3.1 Nanopore

3.1.1 Quality Control

To start the analysis of our Nanopore dataset, we proceeded with a standard quality check to ensure the quality of our sequences for later steps. The outputs include:

- FastQC generated reports for each sample.
- A MultiQC generated report of the entire dataset.
- A count file, where I compiled the number of reads of each sample for use in further steps.

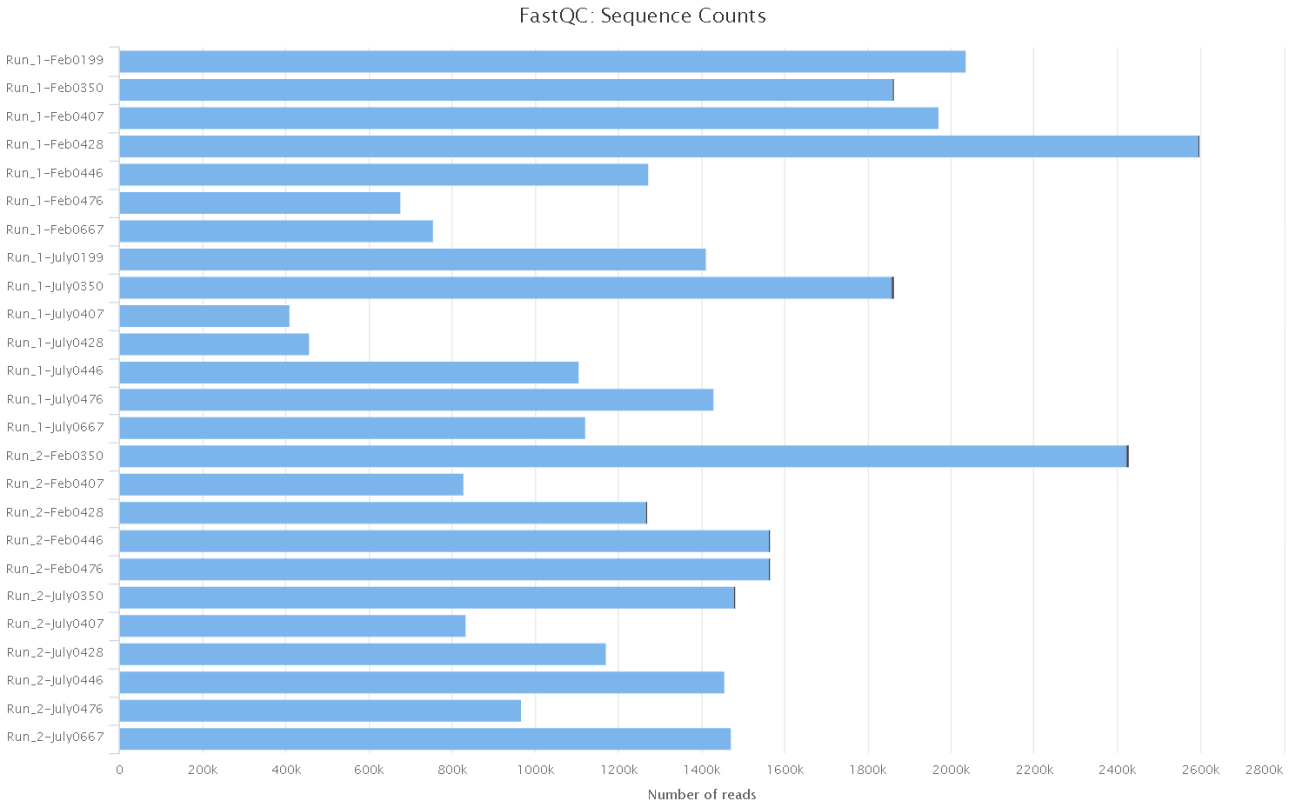


Fig. 14: MultiQC report of the duplicated reads in the raw Nanopore dataset

In the Figure 14, we can see that the number of reads is in blue, and in some samples, a black line represents the duplicated reads; Table 5 highlights duplicated reads with their percentage. We can also see a considerable variation in the number of reads per sample and run. Different factors can explain this; the first is that Nanopore sequences reads of various lengths. Thus, a higher number of reads does not always represent more DNA sequenced, as each read could be between a few hundred and a couple of hundred thousand bases. Second, the sequencing was done with multiplexing (see 1.2 for further information), where 2 or 3 samples were pooled together for each sequencing. This could have caused some samples to be more sequenced than others. Third, each sample had the same volume of DNA library used but their purity and concentration were not perfectly identical. Those 3 points explain the variation in number of reads.

Sample name	**Nbr of total reads**	**Percentage of duplicated reads**
Run_1-July0350	1.858.282	0,4%
Run_1-Feb0350	1.862.657	0,1%
Run_1-Feb0428	2.596.083	0,1%
Run_2-Feb0350	2.424.276	0,2%
Run_2-July0350	1.479.127	0,2%

Table 5: Summary of duplicate read levels across raw Nanopore samples

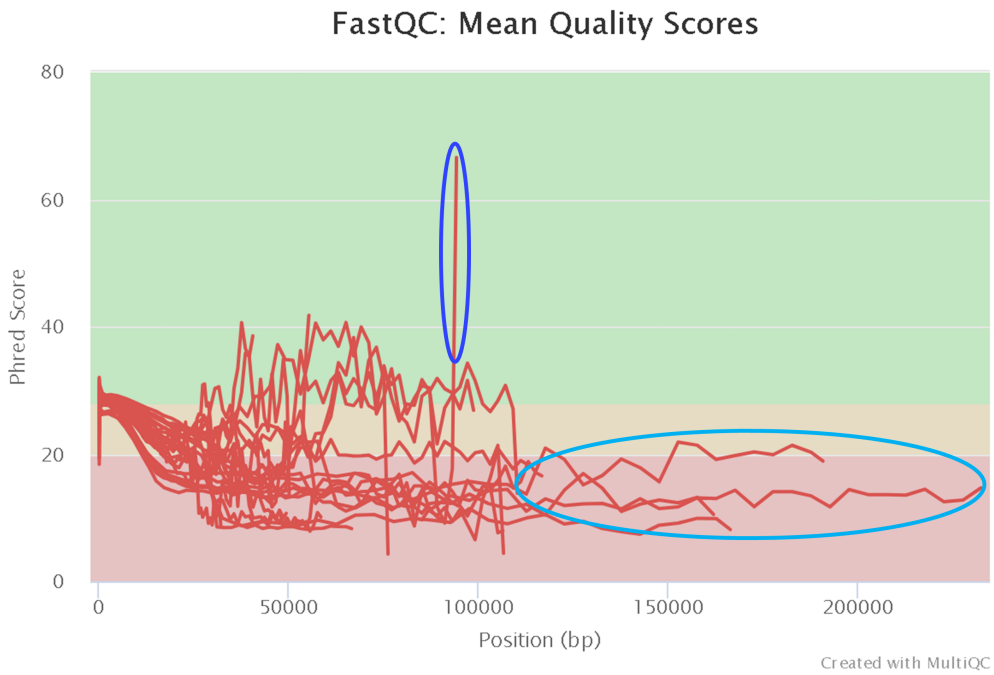


Fig. 15: MultiQC report of sequence quality in the raw Nanopore dataset

In Figure 15, we can see one sample with and odd behaviour in dark blue where the end of the read jumps severely in quality. This is probably a sequencing error and should be corrected during the trimming. In lighter blue we can see a few samples having a poor quality (below 20) but of great length (over 150.000 to 200.000 bases).

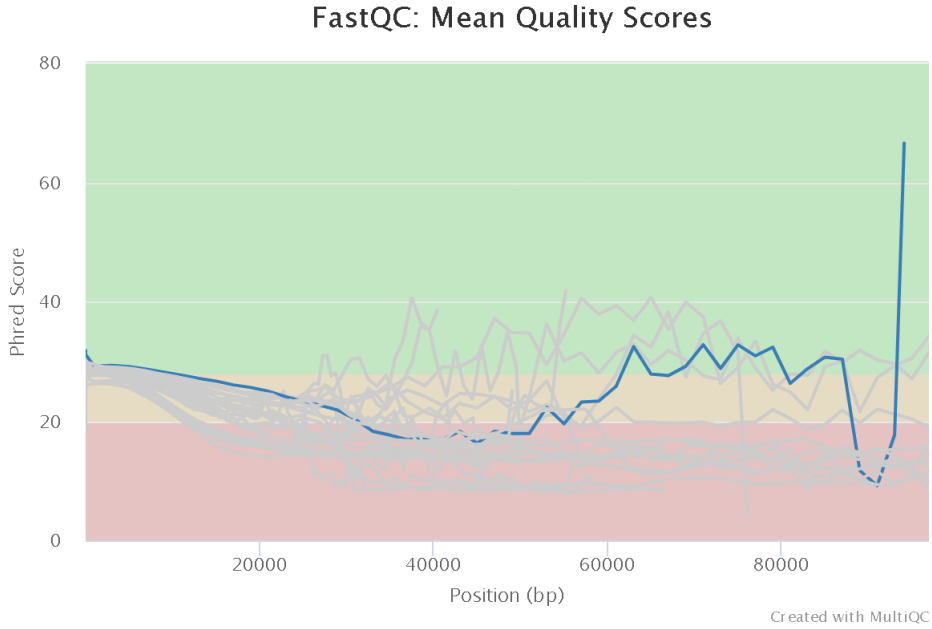


Fig. 16: Quality score of Run_1-July0407 (Nanopore)

Figure 16 isolates in darker blue the quality score of the reads from the sample `Run_1-July0407`. We can see that the overall quality score is around 30. Towards the end we can observe a sudden drop and then an important increase up to a quality score of 70.

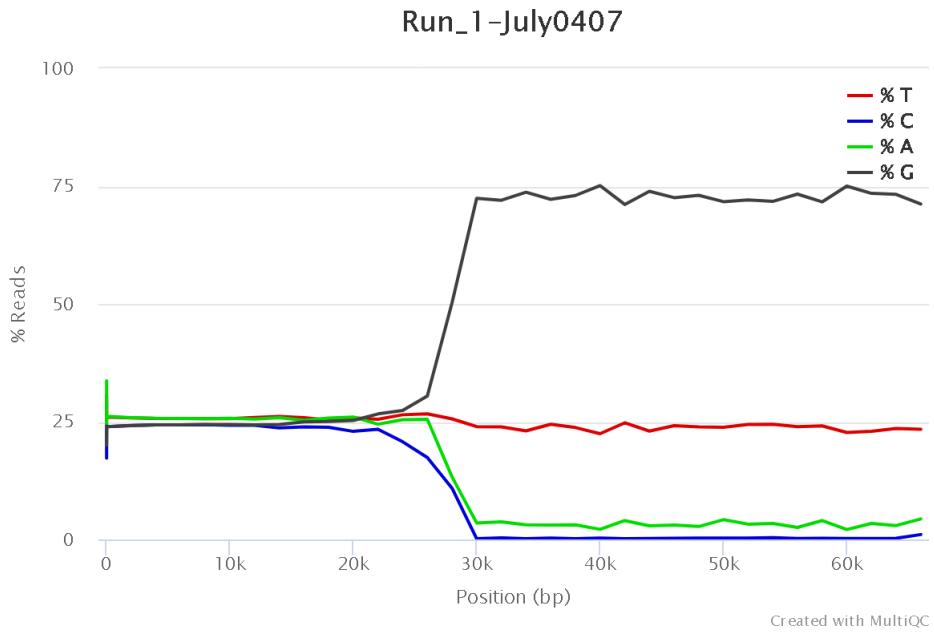


Fig. 17: Per base sequence content for Run_1-July0407

When we inspect the samples with the quality spike in the end we can see that G represents almost 75% of the bases and T 25% of the bases on the reads with a length superior to 25.000 bases; this most probably means that some reads got clogged in the pores and were read for a long time.

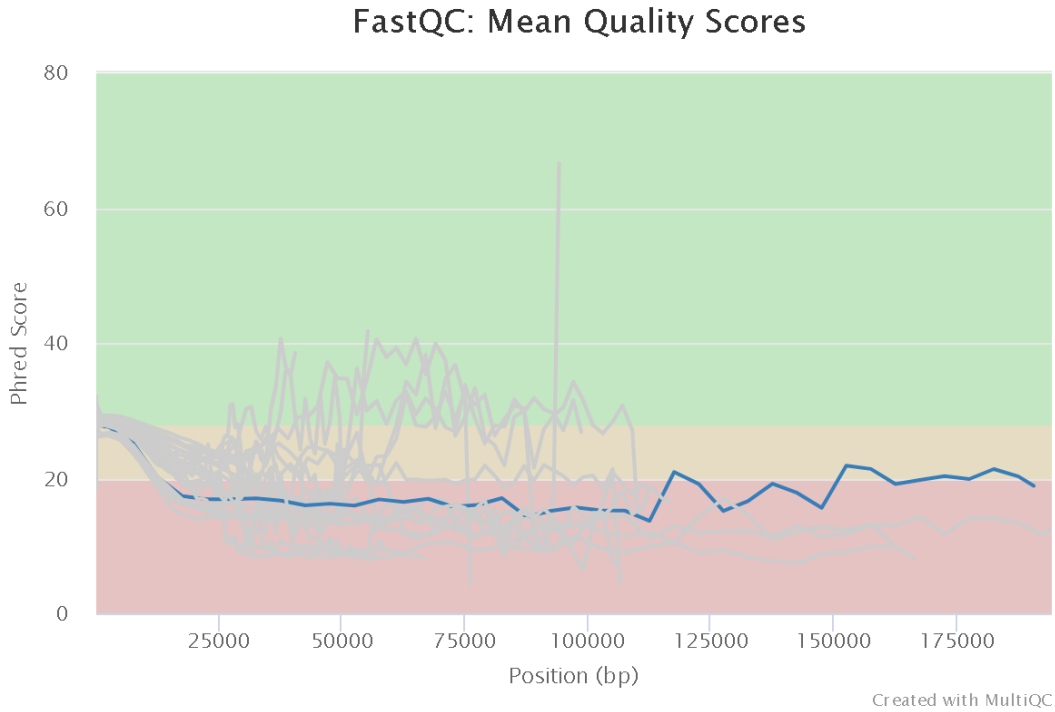


Fig. 18: Quality score of Run_2-Feb0428

As we explained for the previous sample, the Figure 18 illustrates the overall quality of the reads from the sample `Run_2-Feb0428`. This quality seem to be more constant with an average value below 20.

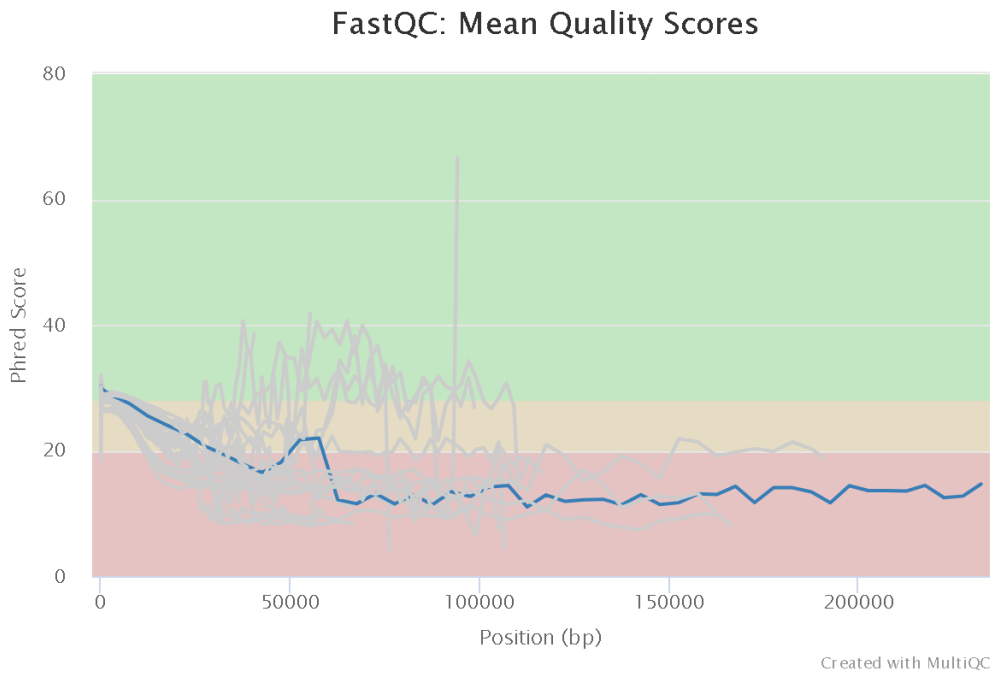


Fig. 19: Quality score of Run_1-July0667

We can observe the same tendency, from Figure 18 on Figure 19, where the overall quality score seem to be constant. Unfortunately, there is a drop around 50.000-60.000 bp and then the score remains the same under 15.

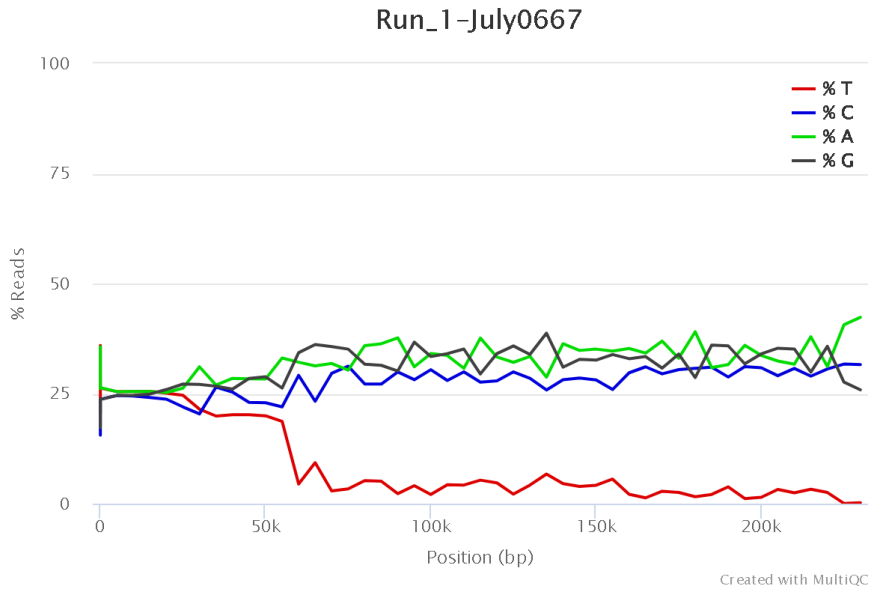


Fig. 20: Per base sequence content for Run_1-July0667

The Figure 20 shows that after 50.000 bp the percentage of T drops below 5%, while the other 3 nucleotides remain at a constant level of 25-35%.

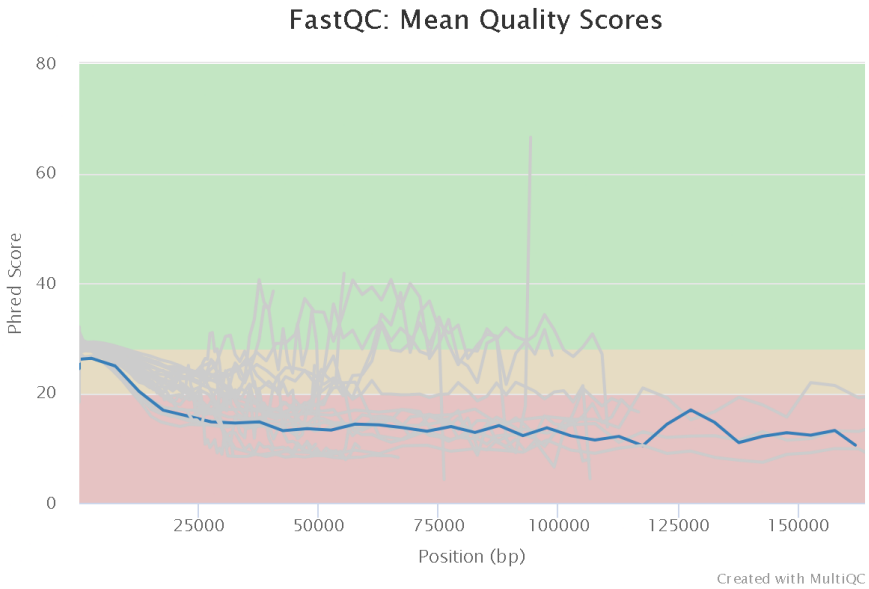


Fig. 21: Quality control of Run_2-July0428

The last sample that exhibits a great length is `Run_2-July0428` and the Figure 21 show us that the quality score is quite linear with the overall score under 20.

3.1.2 Trimming

Once potential issues with our Nanopore data were identified, we proceeded with the trimming step. As detailed in the `Chopper` section (see 2.3.1), we applied trimming based on quality scores using a Bash script (see G.2). This script trimmed the initial Nanopore dataset, counted the number of reads per trimmed sample, and generated fastqc reports and a global quality report of the trimmed dataset (see G.3).

Following the trimming, we observed the percentage of reads before and after the process. Figure 22 shows the plot of the trimmed Nanopore dataset. Table 6 provides details on the lost reads and their percentage after trimming. On average, each sample lost approximately 10% of its initial data, with some samples experiencing losses of over 12,5% to 14%. The number of lost reads fluctuated between 75.000 and 270.000, with some samples showing greater loss than others.

Based on the information from Table 6, we distinguish that the same samples lose more data than others and vice versa, based on the percentage of lost data.

- Highest loss :
  - `Run_2-July0428` with 14,68% (out of 1.170.337 reads)
  - `Run_2-Feb0428` with 14,14% (out of 1.270.335 reads)
- Lowest loss :
  - `Run_1-July0407` with 9,03% (out of 409.214 reads)
  - `Run_1-Feb0476` with 9,28% (out of 676.316 reads)

Sample name	Lost reads	Total reads before trimming	Percentage of lost reads
Run_1-Feb0199	211.604	2.036.152	10,39%
Run_1-Feb0350	220.094	1.864.980	11,80%
Run_1-Feb0407	208.239	1.971.160	10,56%
Run_1-Feb0428	277.091	2.598.263	10,67%
Run_1-Feb0446	134.921	1.273.906	10,59%
Run_1-Feb0476	62.778	676.316	9,28%
Run_1-Feb0667	76.930	756.381	10,17%
Run_1-July0199	147.656	1.412.131	10,45%
Run_1-July0350	197.467	1.865.333	10,59%
Run_1-July0407	36.967	409.214	9,03%
Run_1-July0428	46.539	457.212	10,18%
Run_1-July0446	119.264	1.104.809	10,79%
Run_1-July0476	153.352	1.430.544	10,72%
Run_1-July0667	114.417	1.122.004	10,20%
Run_2-Feb0350	307.370	2.429.145	12,66%
Run_2-Feb0407	109.276	828.680	13,18%
Run_2-Feb0428	179.624	1.270.335	14,14%
Run_2-Feb0446	207.577	1.564.784	13,27%
Run_2-Feb0476	201.689	1.565.013	12,89%
Run_2-July0350	170.890	1.482.434	11,53%
Run_2-July0407	106.042	834.166	12,71%
Run_2-July0428	171.867	1.170.337	14,68%
Run_2-July0446	152.502	1.456.434	10,47%
Run_2-July0476	103.104	968.201	10,65%
Run_2-July0667	144.049	1.472.822	9,78%

Table 6: Lost reads and percentage of lost reads after trimming of the Nanopore dataset

The Table 6 is showing us the percentage of lost reads after trimming across the samples from the Nanopore data. The percentage of lost reads varies between 9 and 14,68%. This lost seem to be more important in the second nanopore than the first one. Across these samples we have :

- `Run_2-July0428` with the highest percentage of lost reads: 14,68%
- `Run_1-July0407` with the lowest percentage of lost reads: 9,03%

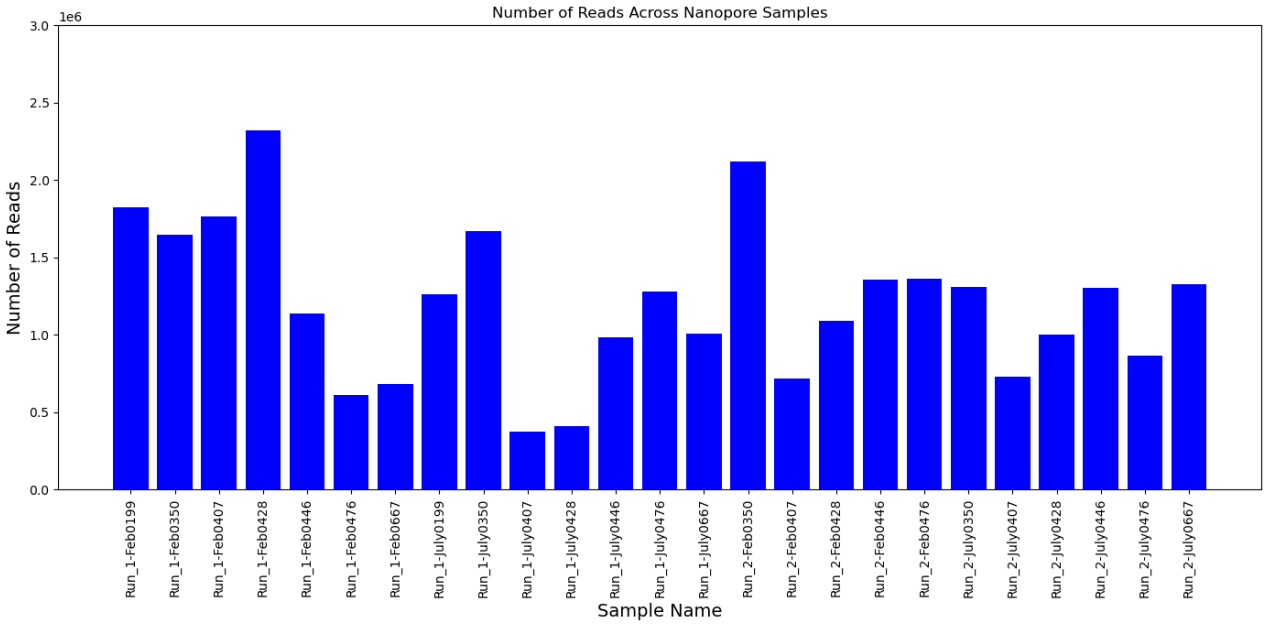


Fig. 22: Plot of the trimmed Nanopore dataset

Figure 22 illustrates the number of reads after trimming. We can see that this number varies for each sample and it's in the range of 500.000 up to 2.500.000 reads.

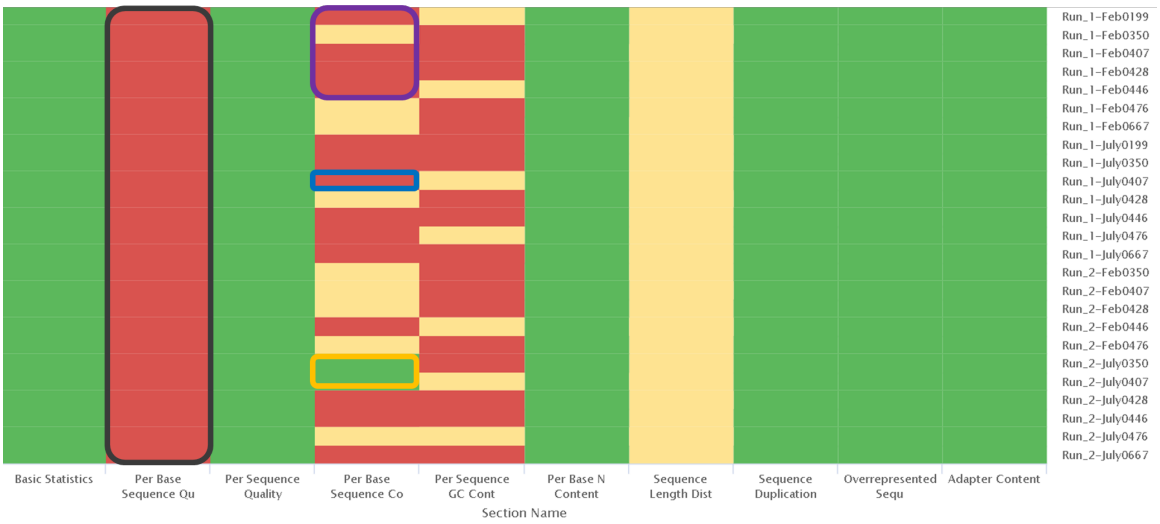


Fig. 23: Heatmap of the Nanopore dataset quality (raw)

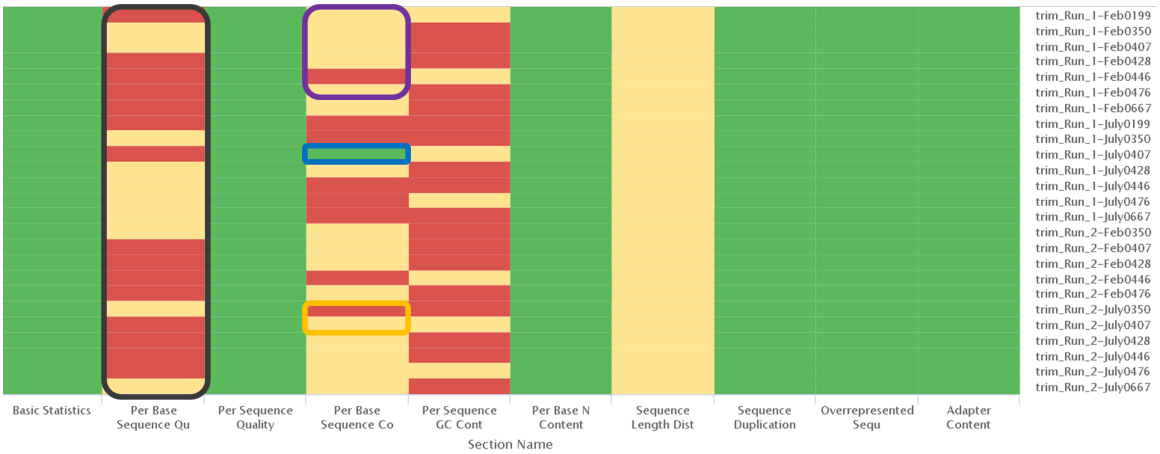


Fig. 24: Heatmap of the Nanopore dataset quality (trimmed)

Figure 23 and 24 represent the general quality report generated by `MultiQC`. The regions that were highlighted with different colors represent the variation in quality before and after trimming. The main changes occurred in the per-base sequence quality and the per-base sequence content quality.

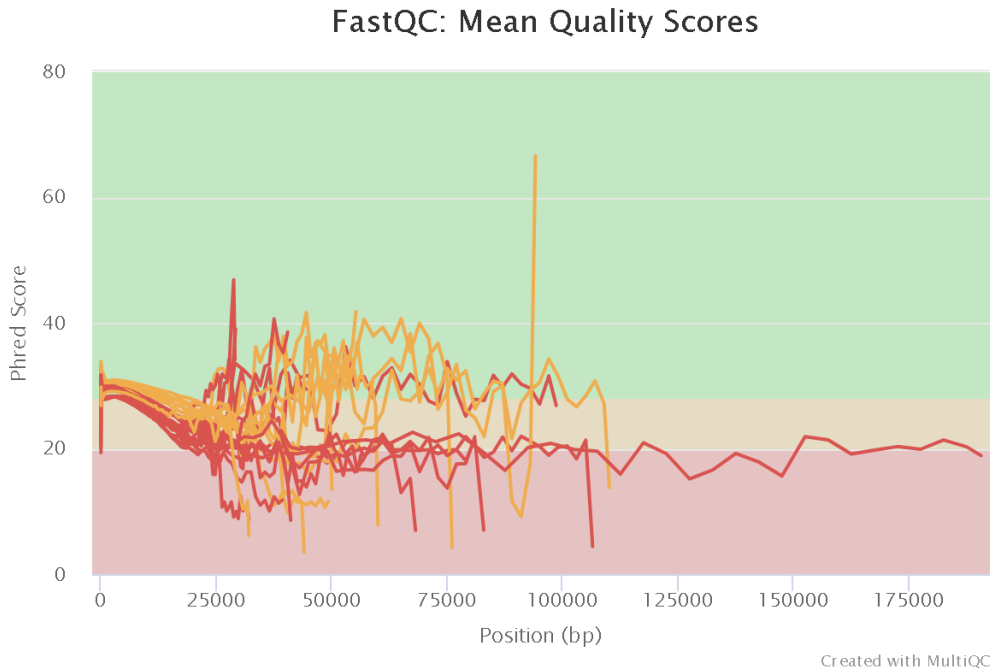


Fig. 25: MultiQC report of the sequence quality on the trimmed Nanopore dataset

In Figure 25 we can observe the overall quality of the Nanopore samples after trimming. We can see that sample with the huge spike at the end stay unchanged, as well as most of the samples. The only thing that changed is the maximum read length that surpasses 175.000 bp.

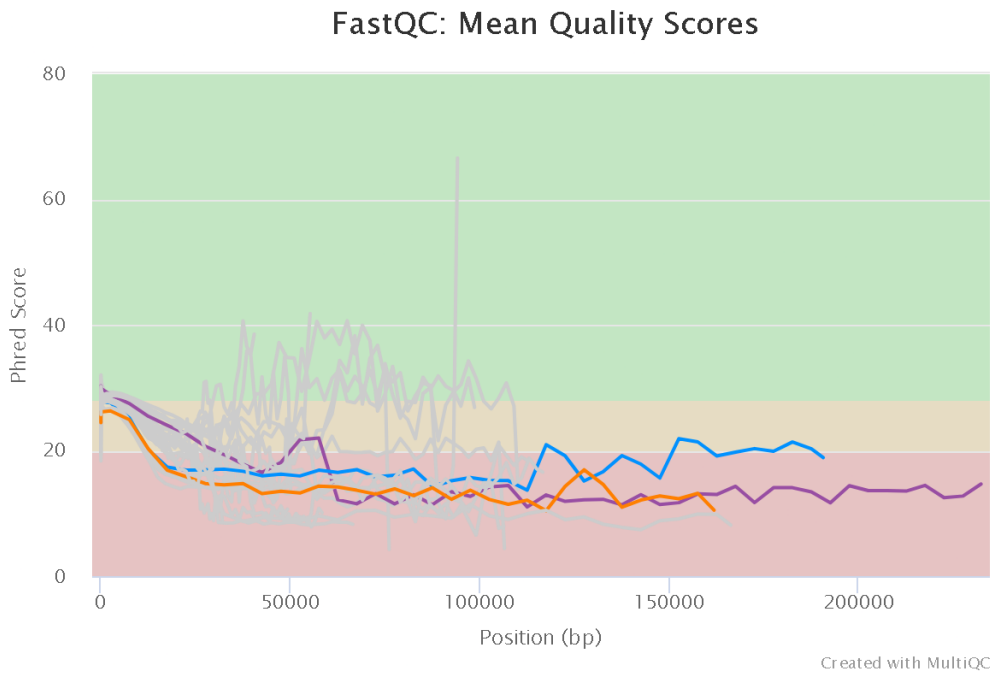


Fig. 26: Quality control of the special cases from the raw Nanopore dataset

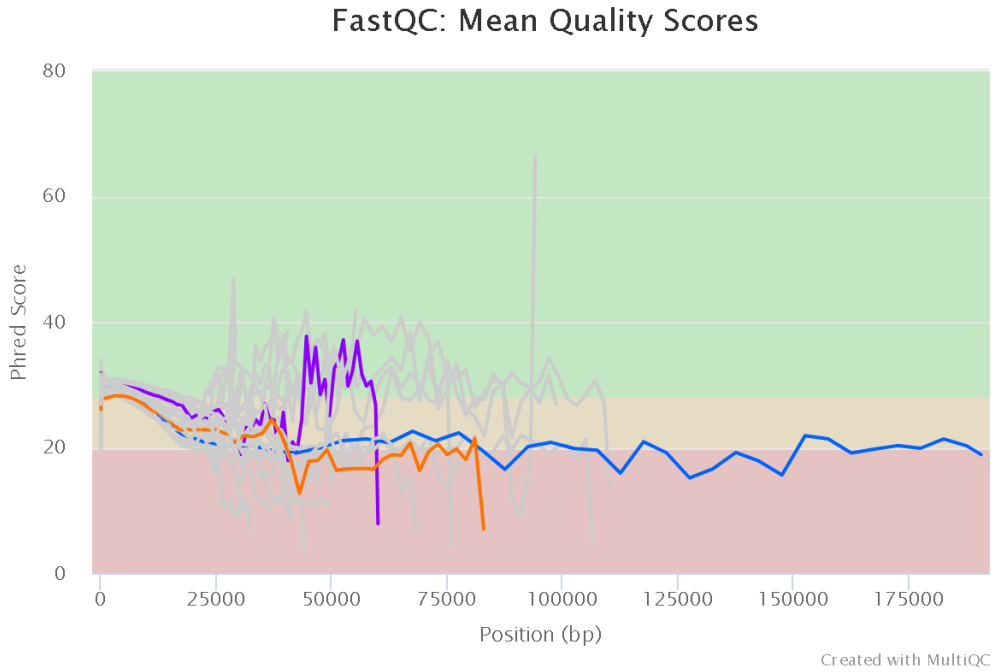


Fig. 27: Quality control of the special cases from the trimmed Nanopore dataset

The Figures 26 and 27 are showing the evolution of longest reads before and after trimming. The samples in purple and orange seem to have better quality scores in the detriment of shortening their length. On the other hand the sample in dark blue stays unchanged.

3.1.3 Host Filtering

The following step of our analysis consists of removing the host. As we explained previously (see 1.5), *Ethiopian Boran* had a particular genome background. To find out which reference genome to use between *Bos Taurus*

and *Bos Indicus*, we ran multiple tests explained in section 2.4 and concluded that only using *Bos Taurus* would be the best option for our analysis.

With this information in mind, we will apply two Bash scripts: one used to remove the DNA of *Bos Taurus* (see G.4) and a second one used in order to verify the quality of this newly generated dataset (see G.5). The first script will generate the newly unmapped reads and a small counting file that will help us for the visualization step. In contrast, the second script will only give out fastqc reports and a global quality report via `MultiQC` to help us understand the unmapped trimmed Nanopore dataset.

Once we obtained the results from these scripts, we observed a slight improvement in the quality for a certain number of samples: `Run_1-July0199`, which had its score of the base sequence content improved (from 0,3 to 0,5).

Sample name	**Lost reads**	**Total reads after filtering and host removal**	**Percentage of lost reads**
Run_1-Feb0199	219.932	1.816.220	10,80%
Run_1-Feb0350	227.372	1.637.608	12,19%
Run_1-Feb0407	217.746	1.753.414	11,05%
Run_1-Feb0428	323.516	2.274.747	12,45%
Run_1-Feb0446	139.155	1.113.451	10,92%
Run_1-Feb0476	66.300	610.016	9,80%
Run_1-Feb0667	82.843	673.538	10,95%
Run_1-July0199	149.217	1.262.914	10,57%
Run_1-July0350	209.391	1.655.942	11,22%
Run_1-July0407	37.520	371.694	9,17%
Run_1-July0428	48.272	408.940	10,55%
Run_1-July0446	120.762	984.047	10,93%
Run_1-July0476	155.452	1.275.092	10,86%
Run_1-July0667	118.541	1.003.463	10,57%
Run_2-Feb0350	321.841	2.107.304	13,25%
Run_2-Feb0407	114.064	714.616	13,76%
Run_2-Feb0428	203.382	1.066.953	16,01%
Run_2-Feb0446	212.893	1.351.891	13,61%
Run_2-Feb0476	210.123	1.354.890	13,43%
Run_2-July0350	181.876	1.300.558	12,27%
Run_2-July0407	107.160	727.006	12,85%
Run_2-July0428	175.873	994.464	15,03%
Run_2-July0446	154.525	1.301.909	10,61%
Run_2-July0476	104.647	863.554	10,81%
Run_2-July0667	151.211	1.321.611	10,27%

Table 7: Lost reads and percentage of lost reads after mapping out of the Nanopore dataset

Based on the information from Table 7, we distinguish that the same samples lose more data than others and vice-versa, based on the percentage of lost data. For most samples the host genome reads represented 9 to 16% with most of them between 10 and 13% and the cow `0428` in the second sequencing run with 15 to 16%.

- `Run_2-Feb0428` with 16,01% (out of 1.270.335 reads)
- `Run_2-July0428` with 15,03% (out of 1.170.337 reads)

3.1.4 Taxonomic Classification

sourmash

For the part where we only employ `sourmash` we will use a Bash script (see G.6) that will give out the following results :

- a krona output
- a kreport output

These two reports will be more than necessary and help us analyze the final result of the taxonomic classification. Based on the kreport output we can find out the percentage of unclassified reads from each one of the samples. Based on Table 8 we can deduce that there is a low percentage of classified reads when we employ `sourmash`. Due to these huge values, we only identified a small part of our Nanopore dataset that was already pre-processed (removed the DNA of *Bos Taurus* and trimmed them).

Sample Name	**% Classified Run_1**	**% Classified Run_2**
Feb0199	7,32%	**N/A**
Feb0350	6,05%	6,33%
Feb0407	6,90%	8,16%
Feb0428	6,69%	7,19%
Feb0446	6,88%	7,24%
Feb0476	7,33%	6,59%
Feb0667	8,13%	**N/A**
July0199	7,90%	**N/A**
July0350	6,80%	7,53%
July0407	8,05%	7,02%
July0428	7,72%	7,08%
July0446	7,01%	6,62%
July0476	7,34%	7,84%
July0667	6,47%	5,99%

Table 8: Percentage of Classified Reads after Taxonomic Profiling with `sourmash` on Nanopore

When we analyze Table 8, we can see that the overall percentage of classified sequences is around 7-9%, and the pre-processing steps may explain some behaviours. We can clearly see that the percentage of unclassified reads for both runs is concordant and that overall, we obtain more or less the same values. The variation is not significant enough to be symbolic of anything meaningful.

Sample Name	**Nbr species Run_1**	**Nbr species Run_2**
Feb0199	1.197	**N/A**
Feb0350	1.112	1.152
Feb0407	1.077	620
Feb0428	1.071	577
Feb0446	789	823
Feb0476	667	1.012
Feb0667	765	**N/A**
July0199	996	**N/A**
July0350	1.311	1.093
July0407	603	776
July0428	463	700
July0446	853	890
July0476	1.041	735
July0667	857	839

Table 9: Number of Species after Taxonomic Profiling with `sourmash` on Nanopore

The number of species found varies greatly between samples and runs. The lowest is `July0428`, with 463 species, and the highest is `July0350`, with 1.311 species found.

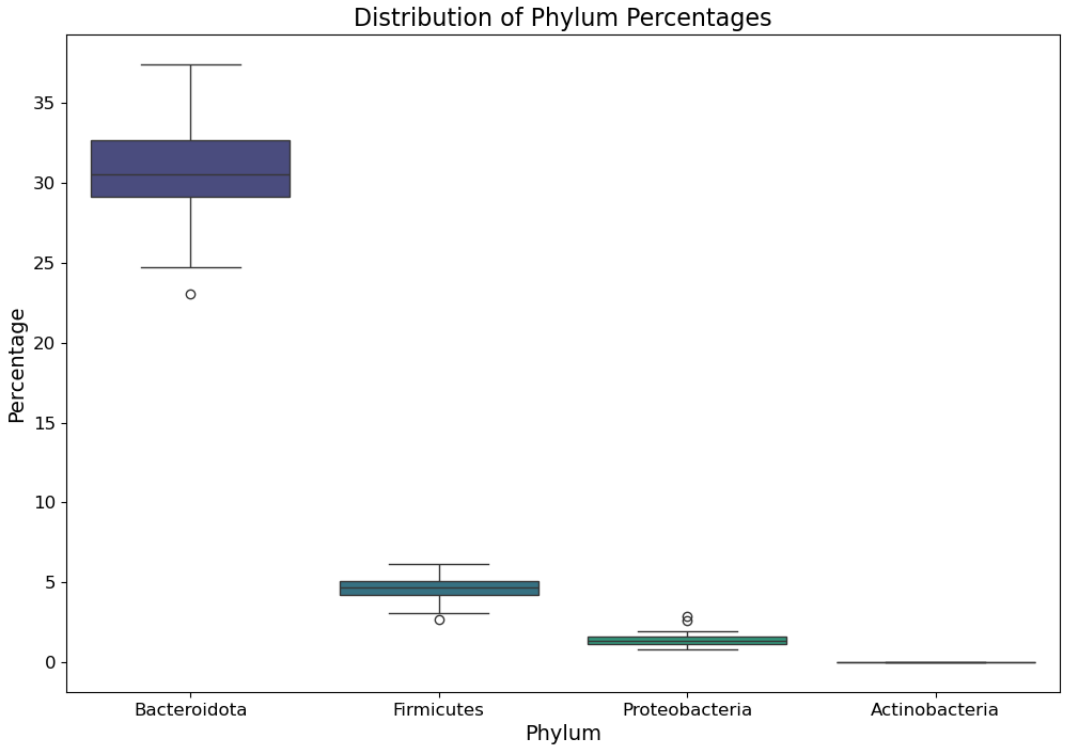


Fig. 28: Phylum Abundance Distribution Across Different Samples

The boxplot present in Figure 28 is showing the percentage of different phylum across the Nanopore data. The most represented phylum is the *Bacteroidota* with almost 33%, while the other: *Firmicutes*, *Proteobacteria* and *Actinobacteria* are under 5%.

Kraken2

Sample Name	**% Classified Run_1**	**% Classified Run_2**
Feb0199	75,47%	**N/A**
Feb0350	73,70%	66,51%
Feb0407	73,37%	69,48%
Feb0428	65,99%	61,40%
Feb0446	69,38%	64,28%
Feb0476	79,76%	76,92%
Feb0667	77,14%	**N/A**
July0199	73,92%	**N/A**
July0350	83,73%	80,61%
July0407	84,78%	81,77%
July0428	70,37%	62,63%
July0446	77,68%	72,75%
July0476	79,82%	73,56%
July0667	80,44%	74,08%

Table 10: Percentage of Classified Reads after Taxonomic Profiling with `Kraken2` on Nanopore

Table 10 is providing us important information concerning the usage of `Kraken2`. The concentration of classified reads varies between 61,4% and 84,78%. The highest percentage being visible for the sample `July0407` for both runs, while the lowest percentage is visible for the sample `July0428` for both runs. The percentage of

classification seem to be quite similar between the runs, with small variations being present. The samples from February seem to have a lower percentage of classification compared to the ones in July.

Sample Name	Nbr species Run_1	Nbr species Run_2
Feb0199	11.533	**N/A**
Feb0350	11.415	11.575
Feb0407	11.374	10.401
Feb0428	11.659	10.791
Feb0446	10.924	11.047
Feb0476	10.364	11.208
Feb0667	10.467	**N/A**
July0199	10.950	**N/A**
July0350	11.319	10.937
July0407	9.909	10.566
July0428	9.925	10.801
July0446	10.834	11.155
July0476	11.042	10.643
July0667	10.883	11.185

Table 11: Number of Species after Taxonomic Profiling with `Kraken2` on Nanopore

As we observed for the number of species found with `sourmash`, on Table 11 we can see the number of species for each sample based on the run. These values range from 9.909 to 11.659. We can see also that `sourmash` seem to find more species in the second run than the first one.

Bracken VS sourmash

Sample Name	% Bracken Classified Run_1	% Bracken Classified Run_2
Feb0199	29,37%	**N/A**
Feb0350	30,95%	31,4%
Feb0407	30,09%	29,29%
Feb0428	27,82%	27,85%
Feb0446	29,02%	29,66%
Feb0476	30,57%	30,06%
Feb0667	30,59%	**N/A**
July0199	30,06%	**N/A**
July0350	32,0%	32,12%
July0407	31,73%	31,69%
July0428	30,47%	29,82%
July0446	29,9%	29,64%
July0476	31,3%	30,63%
July0667	30,58%	29,77%

Table 12: Percentage of Classified Reads with Bracken

Sample Name	Newly Estimated Reads Run_1	Newly Estimated Reads Run_2
Feb0199	400.987	**N/A**
Feb0350	371.609	438.961
Feb0407	385.335	144.041
Feb0428	416.361	180.255
Feb0446	227.038	256.317
Feb0476	147.185	311.825
Feb0667	157.425	**N/A**
July0199	278.928	**N/A**
July0350	442.861	335.115
July0407	98.594	186.850
July0428	86.379	184.189
July0446	227.038	279.354
July0476	316.860	192.950
July0667	245.648	289.863

Table 13: Newly Estimated Reads Based on Bracken Percentages

Table 12 represents the percentage of classified reads by `Bracken` based on the newly estimated reads from Table 13. We can see via Table 12 that `Bracken` is re-estimating the 1/3 of the newly estimated reads.

Sample Name	% Classified Run_1	% Classified Run_2
Feb0199	5,54%	**N/A**
Feb0350	4,99%	5,07%
Feb0407	5,35%	6,09%
Feb0428	5,91%	5,62%
Feb0446	5,97%	5,17%
Feb0476	5,72%	4,72%
Feb0667	5,29%	**N/A**
July0199	6,60%	**N/A**
July0350	6,27%	6,60%
July0407	6,61%	5,33%
July0428	6,43%	5,12%
July0446	5,42%	4,62%
July0476	6,33%	6,44%
July0667	4,70%	4,61%

Table 14: Percentage of Classified Reads after Taxonomic Profiling with `Kraken2` / `sourmash` on Nanopore

Table 14 exhibits the percentage of classification done with `sourmash` on the unclassified sequences from `Kraken2`. These values are in the range from 4,72% to 6,61% and highlights the efficiency of taxonomic classification done with `sourmash`.

Sample Name	Adjusted % Classified Run_1	Adjusted % Classified Run_2
Feb0199	+ 4.18%	**N/A**
Feb0350	+ 3.68%	+ 3.37%
Feb0407	+ 3.92%	+ 4.23%
Feb0428	+ 3.91%	+ 3.45%
Feb0446	+ 4.14%	+ 3.32%
Feb0476	+ 4.56%	+ 3.63%
Feb0667	+ 4.08%	**N/A**
July0199	+ 4.95%	**N/A**
July0350	+ 5.25%	+ 5.32%
July0407	+ 5.61%	+ 4.37%
July0428	+ 4.52%	+ 3.21%
July0446	+ 4.21%	+ 3.36%
July0476	+ 5.05%	+ 4.74%
July0667	+ 3.79%	+ 3.41%

Table 15: Adjusted Percentage of Classified Reads after Taxonomic Profiling with `Kraken2` / `sourmash` on Nanopore

Sample Name	Adjusted nbr species Run_1	Adjusted nbr species Run_2
Feb0199	+ 281	**N/A**
Feb0350	+ 208	+ 255
Feb0407	+ 238	+ 138
Feb0428	+ 277	+ 132
Feb0446	+ 193	+ 198
Feb0476	+ 107	+ 187
Feb0667	+ 117	**N/A**
July0199	+ 199	**N/A**
July0350	+ 238	+ 207
July0407	+ 78	+ 117
July0428	+ 79	+ 137
July0446	+ 157	+ 176
July0476	+ 202	+ 147
July0667	+ 129	+ 176

Table 16: Adjusted Number of Species after Taxonomic Profiling with `Kraken2` / `sourmash` on Nanopore

Tables 15 and 16 are showing values adjusted to the percentage of classification and number of reads from only `Kraken2`. This will show that `sourmash` will classify an additional of 3,32 to 5,61% of reads and find 79 to 281 new species. The variation in the percentage of classified reads seem really low, for the number of species there are various samples who display important variation between the runs:

- `Feb0407` with 100 more species in `Run_1` than `Run_2`
- `Feb0428` with 145 more species in `Run_1` than `Run_2`

3.1.5 Data Visualisation

Krona on sourmash

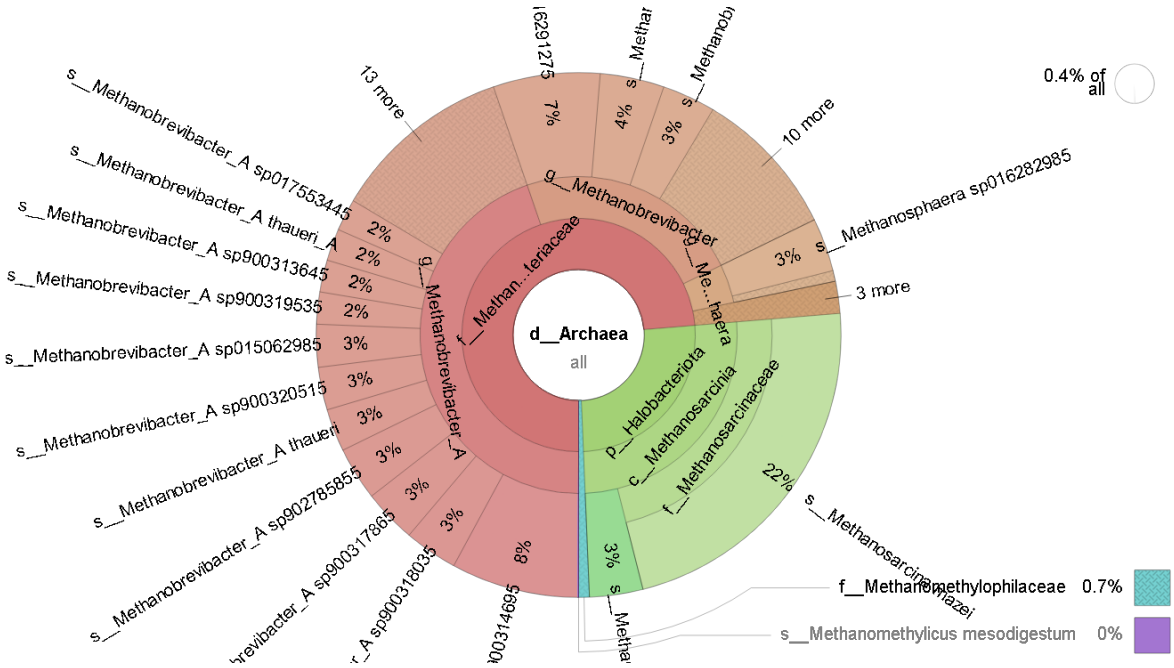


Fig. 29: Krona report for Run_1-Feb0199 Nanopore (sourmash)

In Figure 29, we see the classification of the reads in `Feb0199` in the Nanopore dataset using `sourmash`. The focus is made on the*Archaea* domain, which represents 0,4% of the results as indicated in the top right corner, and out of those 0,4%, roughly 75% are part of the *Methanobacteriaceae* family, and 25% are from the *Halobacteriota* phylum (22% is the *Methanosarcinaceae* family and 3% another family). For a report of organisms present in the rumen of cattle, a well-known animal for its production of methane through its microbiome and the degradation of its feed. It's quite normal to see that many *Archaea* involved in methanogenesis.

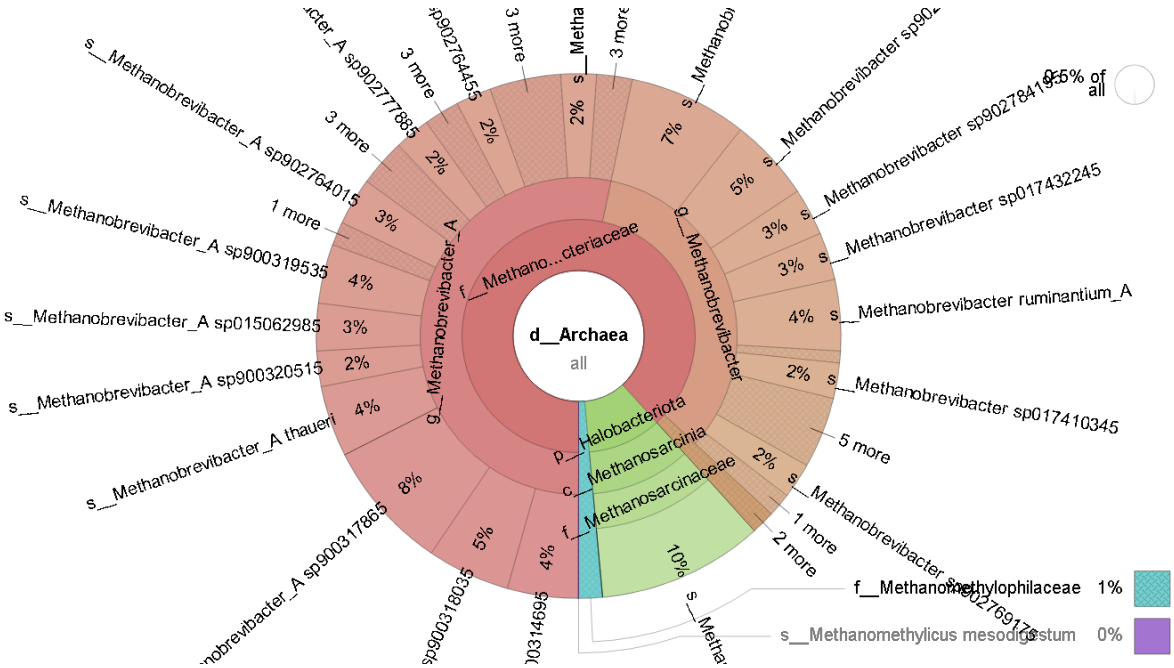


Fig. 30: Krona report for Run_1-July0199 Nanopore (sourmash)

In Figure 30, we can see that the proportions of the 2 main families changed, with *Methanobacteriaceae*

increasing to almost 90% of the population, while *Methanosarcinaceae* diminished to only 10%. This variation in the population for the same host under 2 different periods warrants further analyses by the collaboration in projects.

Krona on Kraken2/Bracken

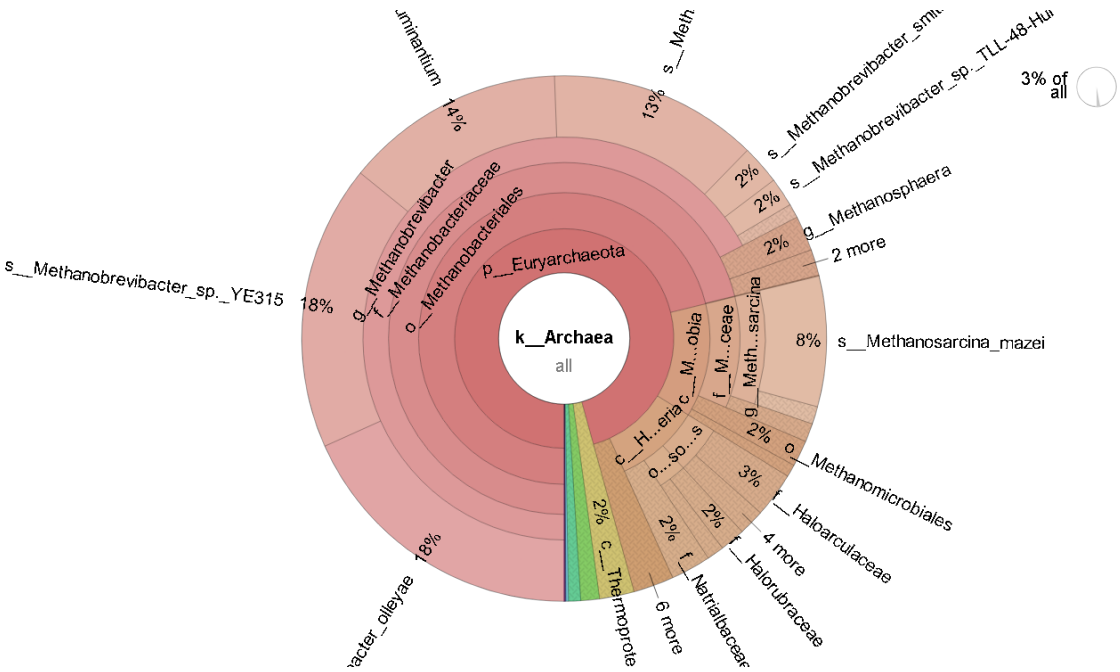


Fig. 31: Krona report for Run_1-Feb0199 Nanopore (K2B)

We can observe similar changes in the population distribution in the analyses with the `Kraken2/Bracken` method. The numbers are different because the method differ, but the behaviour remains. I have communicated this to the person in charge of the study.

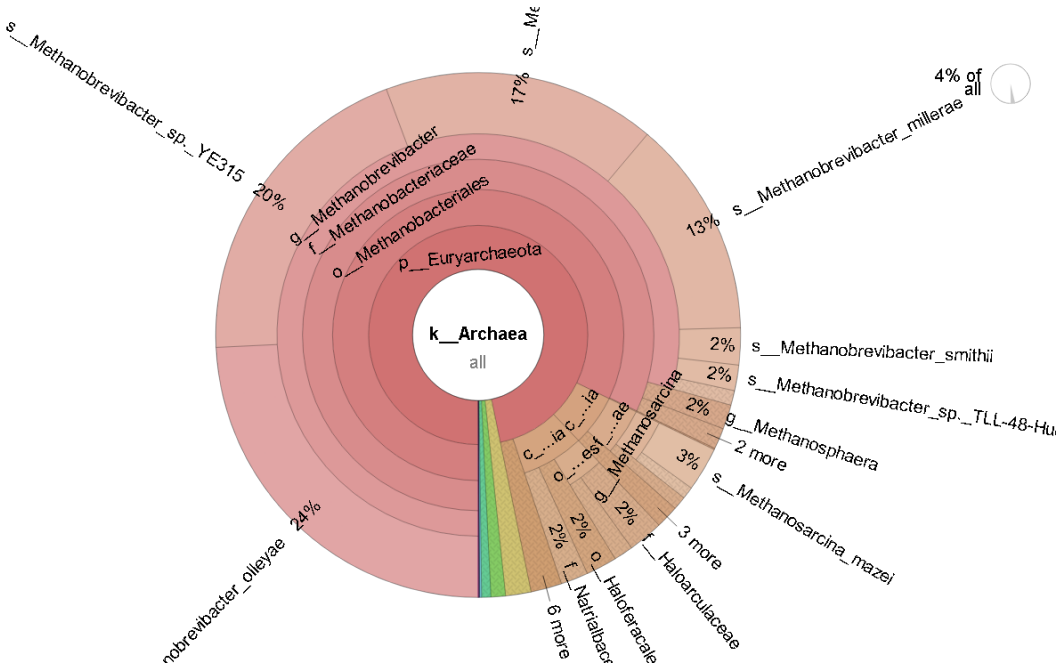


Fig. 32: Krona report for Run_1-July0199 Nanopore (K2B)

Pavian on Kraken2/Bracken

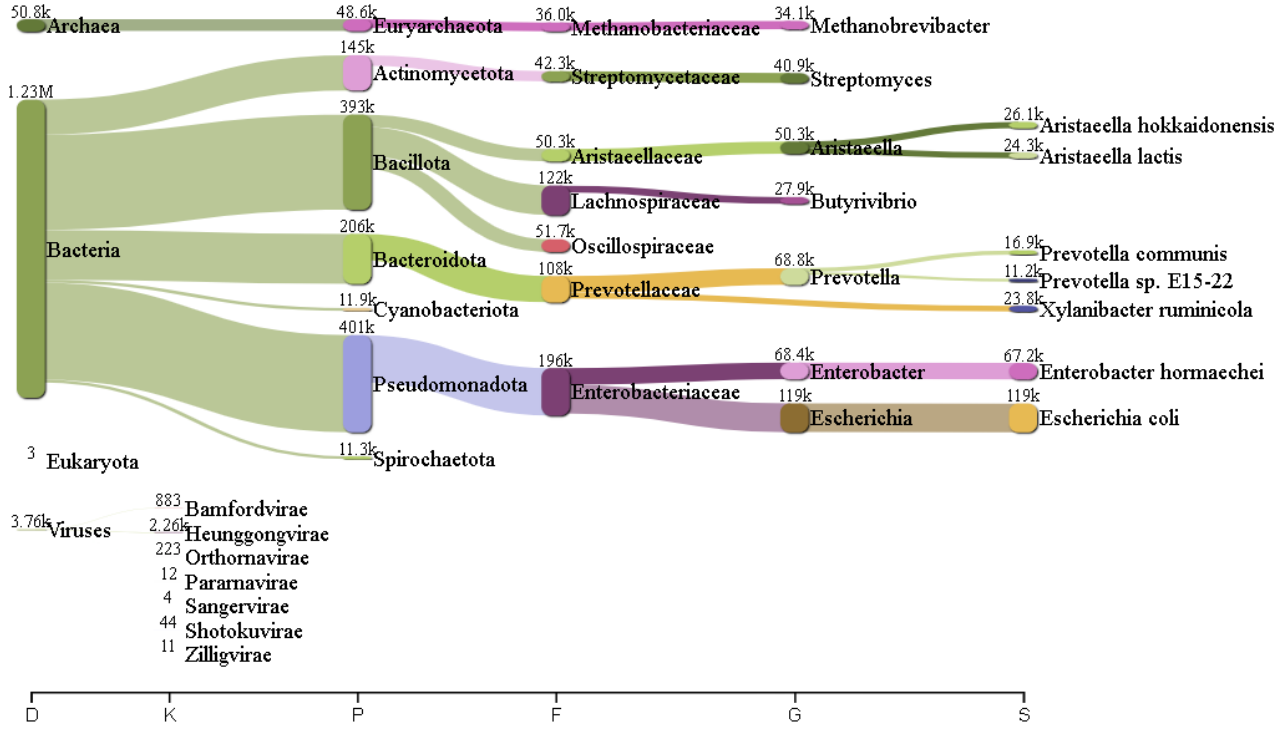


Fig. 33: Pavian report for Run_1-Feb0199 Nanopore (K2B)

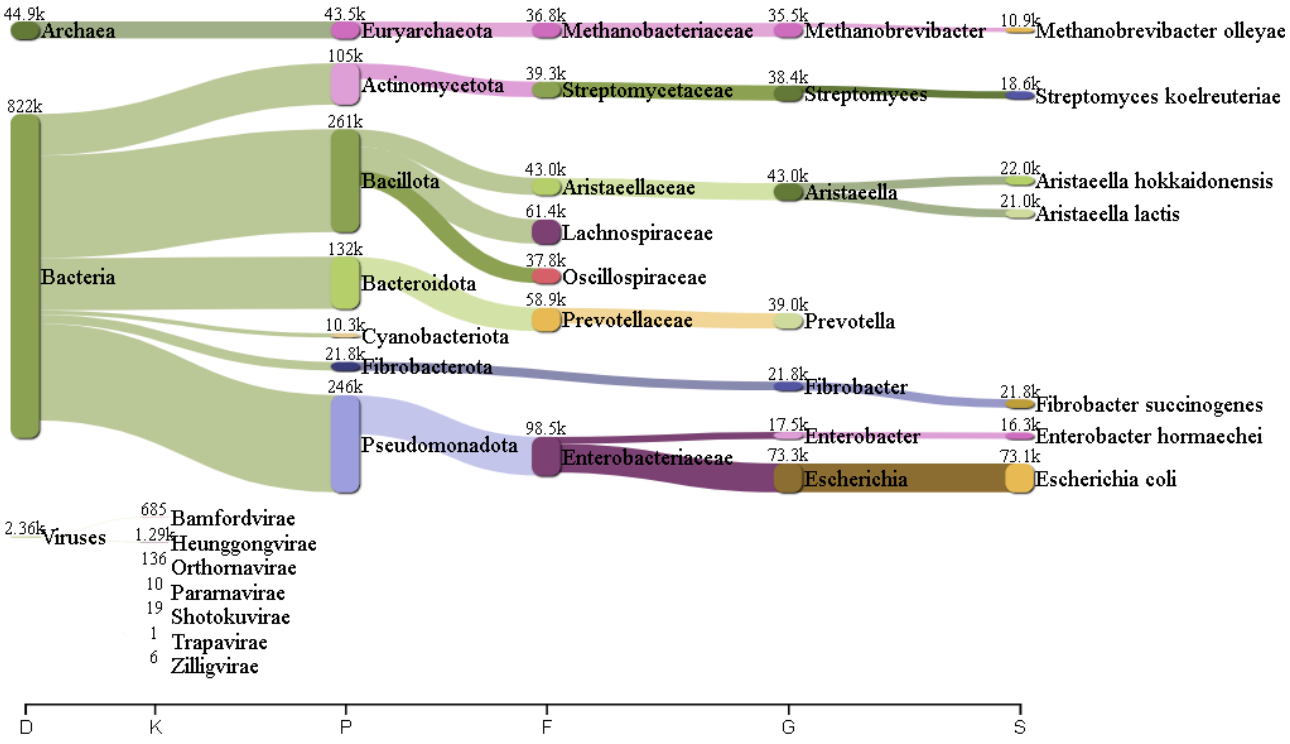


Fig. 34: Pavian report for Run_1-July0199 Nanopore (K2B)

3.2 Illumina

We will proceed with the analysis of the Illumina dataset to gain more information about our samples.

3.2.1 Quality Control

As we explained in subsection 3.1.1, we will begin the taxonomic classification of our dataset by performing a quick quality control in order to ensure the quality of our sequences that the following tools will later process. In order to obtain this quality report we will employ a Bash script (see G.9) that will provide the following results :

- fastqc reports for each one of the samples
- a multiqc report of the entire dataset
- a counting file containing the number of reads of each one of the samples

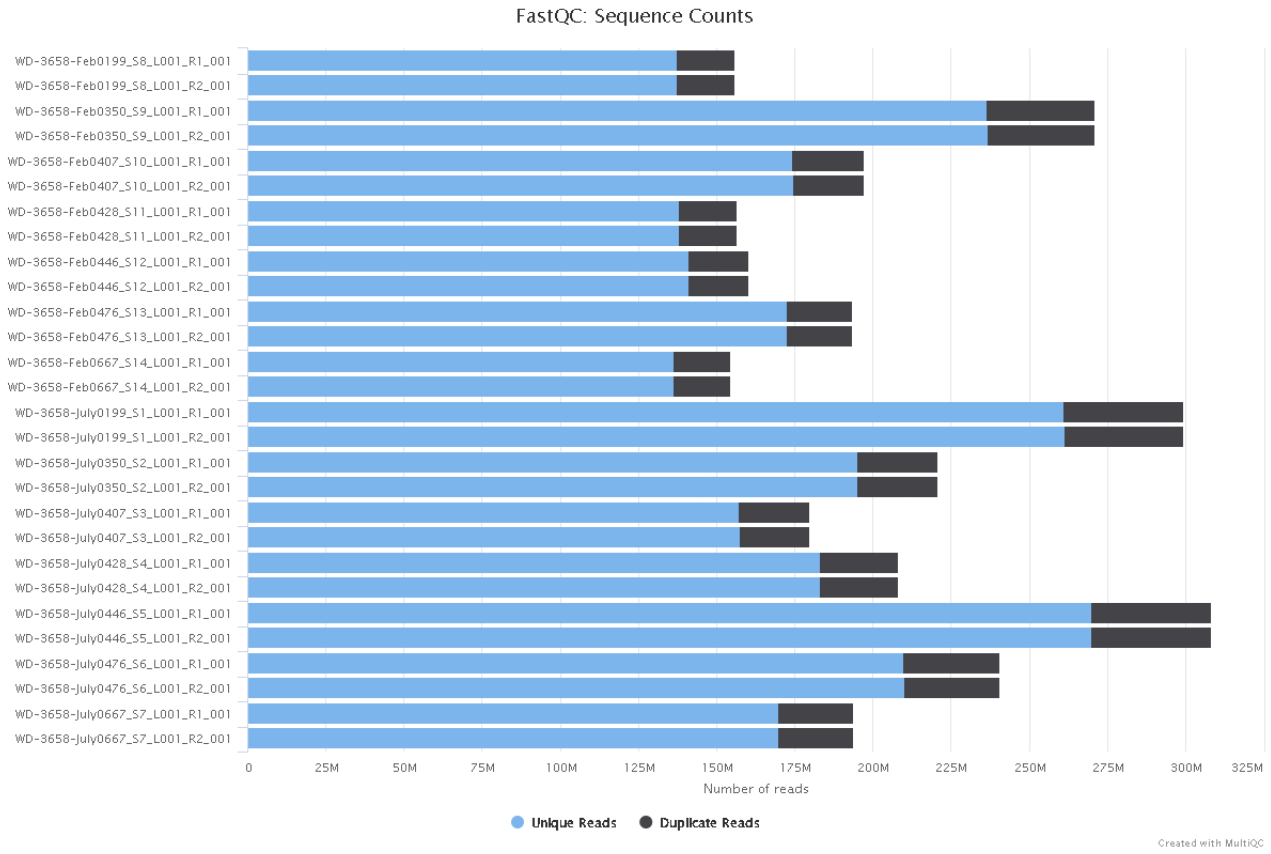


Fig. 35: MultiQC report for the duplicated reads on the raw Illumina dataset

The Figure 35 exhibits two crucial elements for the analysis of the Illumina data: the number of total reads (light blue) and the number of duplicated reads (black). We can see that the number of total reads varies depending on the sample, while the number of duplicated reads seem to be more constant.

Sample name	Unique reads	Duplicated reads
Feb0199_R1	137.316.058	18.558.491
Feb0199_R2	137.446.281	18.428.268
Feb0350_R1	236.591.144	34.577.563
Feb0350_R2	236.878.120	34.290.587
Feb0407_R1	174.323.308	22.738.429
Feb0407_R2	174.466.777	22.594.960
Feb0428_R1	138.107.987	18.468.511
Feb0428_R2	137.999.466	18.577.032
Feb0446_R1	140.993.035	19.159.594
Feb0446_R2	141.167.487	18.985.142
Feb0476_R1	172.602.630	20.669.349
Feb0476_R2	172.686.101	20.585.878
Feb0667_R1	136.362.138	18.025.240
Feb0667_R2	136.389.356	17.998.022
July0199_R1	261.205.385	38.004.560
July0199_R2	261.299.706	37.910.238
July0350_R1	195.122.371	25.539.001
July0350_R2	195.201.307	25.460.065
July0407_R1	157.239.306	22.353.556
July0407_R2	157.584.857	22.008.005
July0428_R1	183.235.640	24.720.204
July0428_R2	183.240.308	24.715.536
July0446_R1	269.926.906	38.179.420
July0446_R2	270.002.742	38.103.584
July0476_R1	209.791.479	30.840.515
July0476_R2	210.241.082	30.390.912
July0667_R1	169.743.449	24.190.412
July0667_R2	169.865.989	24.067.872

Table 17: Summary of Unique and Duplicated Reads per Sample for Illumina

Based on the Table 17 we can distinguish 3 samples that contain the highest number of reads (unique and duplicated) across the Illumina dataset :

- `WD-3658-Feb0350` with over 271.200.000 reads
- `WD-3658-July0199` with over 299.200.000 reads
- `WD-3658-July0446` with over 308.100.000 reads

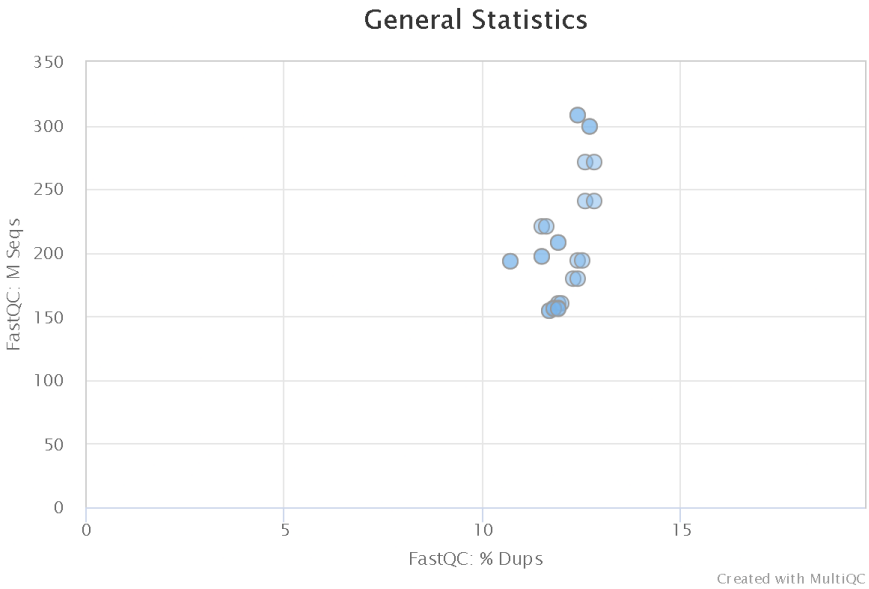


Fig. 36: Relationship between total reads and percentage of duplicated reads

Figure 36 represents an additional argument that helps us understand the high percentage of duplicated reads that are present in our Nanopore dataset. This feature was already visible in Figure 35, but with this scatter plot, we can better visualize and conclude that the overall percentage of duplicated reads is around 11-12%. This behaviour is explained by how Illumina sequencing works, due to being sequencing by synthesis; sometimes, a read will be "cloned" on 2 recording wells and thus will be sequenced twice. This information is crucial for us to understand because we will lose: 11-12% of data in the trimming step, eventually making us come closer to it by correcting a potential sequencing bias.

3.2.2 Host Filtering

Concerning the next step of our Illumina data set analysis, we will remove the DNA of *Bos Taurus*. To accomplish this phase we will utilize a Bash script (see G.10) that will give out the following information :

- the newly unmapped sequences compressed
- a counting file (.txt) that will help us better visualize the variation of the number of reads between each one of the steps of the analysis

Sample Name	**Nbr of Unmapped Reads**	**Percentage of Loss (%)**
Feb0199	155.567.272	0,1978%
Feb0350	270.577.247	0,2182%
Feb0407	196.772.406	0,1469%
Feb0428	155.635.097	0,6015%
Feb0446	159.957.898	0,1216%
Feb0476	192.818.297	0,2345%
Feb0667	153.974.267	0,2676%
July0199	299.176.159	0,0113%
July0350	219.997.033	0,3012%
July0407	179.494.406	0,0548%
July0428	207.812.037	0,0692%
July0446	308.038.278	0,0221%
July0476	240.573.529	0,0243%
July0667	193.699.555	0,1208%

Table 18: Unmapped Reads and Percentage of Loss for Each Sample

Table 18 shows us the loss of reads after host filtering(for further information about the values, please see D). We can clearly discern that we tend to lose more data from the sample in February than those in July. The values from the February subset range from 0,1469% to 0,6015%, with most of them being around 0,15-0,25%. On the other hand, in July, the values ranged from 0,0113% to 0,3012%. The highest loss in both subsets is present in :

- `Feb0428` with 0,6015%
- `July0350` with 0,3012%

3.2.3 Trimming

Compared to the workflow of the analysis of Nanopore sequences, we modified the order of the steps for the analysis of the Illumina sequences. This adjustment is necessary because `Bowtie2`, the tool used for mapping, requires an equal number of reads in both the forward and reverse read pairs for proper alignment. By performing the mapping step before trimming, we ensure that this requirement is met, thereby maintaining the integrity of the analysis. The subsequent trimming step is applied after mapping to remove any remaining low-quality sequences or adapters, followed by taxonomic profiling.

As we move forward with our analysis, we arrive at the trimming step of the Illumina dataset. This step is mandatory for our data, because of the high percentage of duplicated reads in our samples (see Figure 35). To remove the duplicated reads we ran a Bash script (see G.10) that would give out the following information:

- the trimmed unmapped Illumina dataset
- a counting file (.txt) containing the number of reads per sample

We are first going to check the level of duplicated reads in our dataset to observe. If any changes occurred and the eventual impact that they had on the entire dataset.

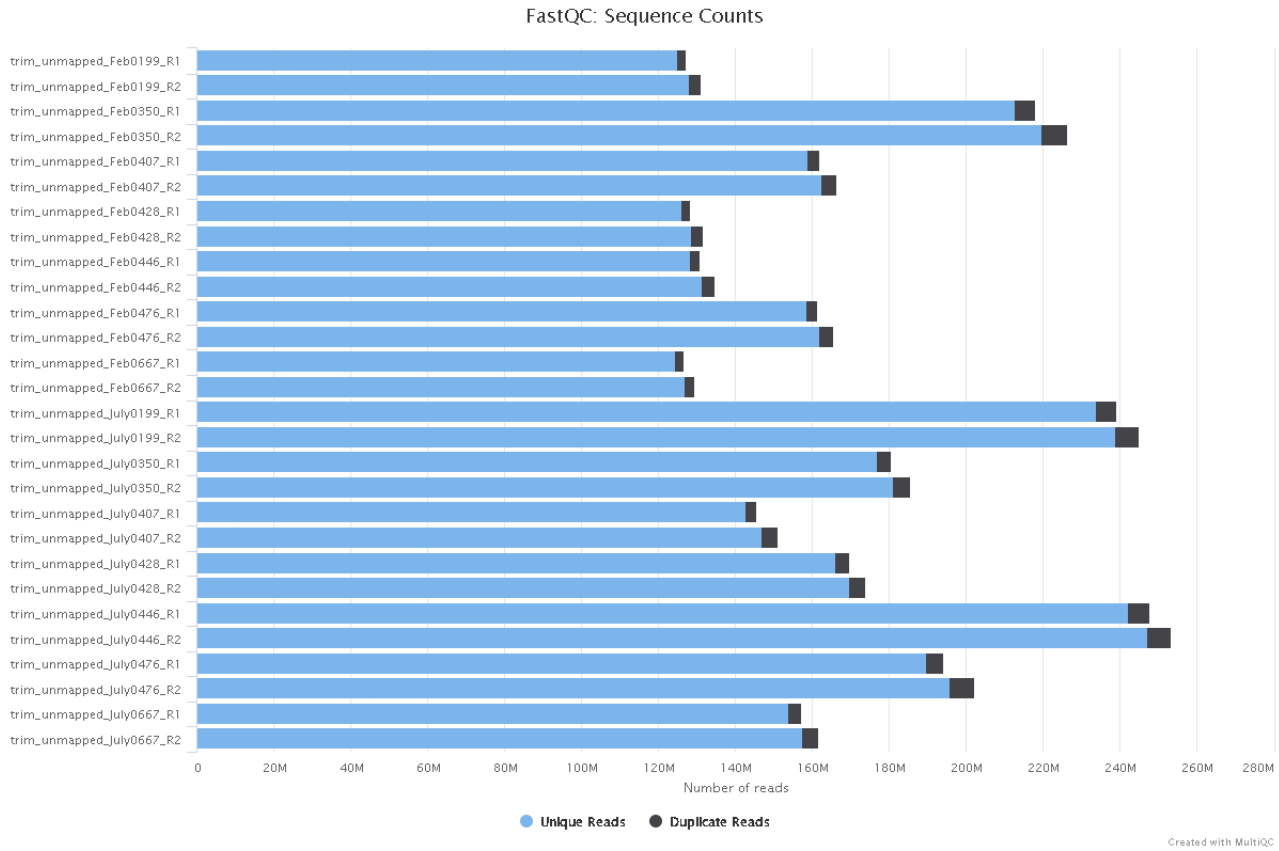
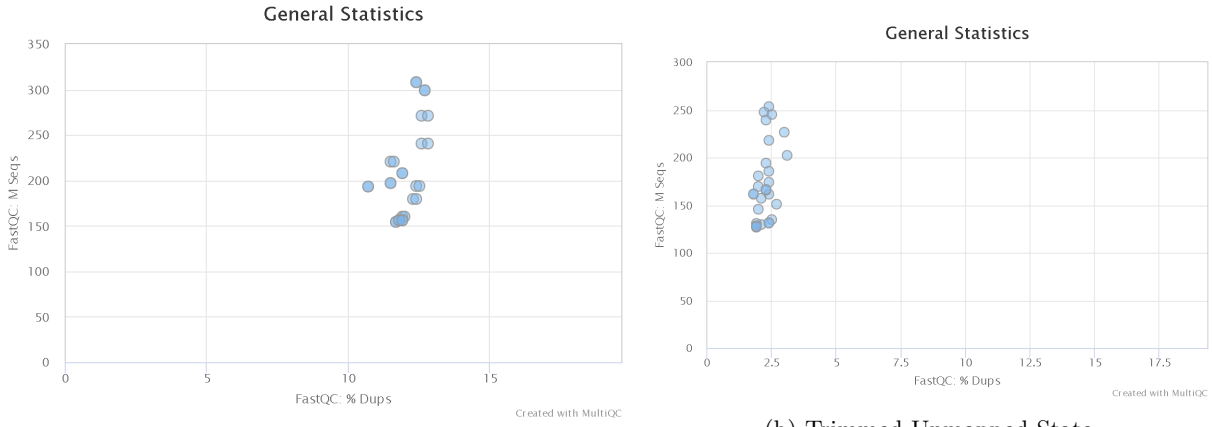


Fig. 37: MultiQC report for the duplicated reads on the trimmed unmapped Illumina dataset

If we observe Figure 38 we tend to see a visible improvement in the number of duplicated reads in the dataset. This proves that the trimming step removed most of the duplicated reads and drastically improved the overall quality of the dataset.



(a) Raw State

(b) Trimmed Unmapped State

Fig. 38: Level of Duplicated Reads for the Illumina Dataset

Sample	Lost Reads	Raw Reads after Host Removal and Trimming	Percentage of Lost Reads
Feb0199_R1	28.544.467	127.330.082	18,31%
Feb0199_R2	24.816.782	131.057.767	15,92%
Feb0350_R1	53.137.014	218.031.693	19,60%
Feb0350_R2	44.674.424	226.494.283	16,47%
Feb0407_R1	35.253.778	161.807.959	17,89%
Feb0407_R2	30.611.526	166.450.211	15,53%
Feb0428_R1	28.110.603	128.465.895	17,95%
Feb0428_R2	24.960.658	131.615.840	15,94%
Feb0446_R1	29.389.896	130.762.733	18,35%
Feb0446_R2	25.372.876	134.779.753	15,84%
Feb0476_R1	31.886.649	161.385.330	16,50%
Feb0476_R2	27.646.790	165.625.189	14,30%
Feb0667_R1	27.634.868	126.752.510	17,90%
Feb0667_R2	24.821.309	129.566.069	16,08%
July0199_R1	59.931.645	239.278.299	20,03%
July0199_R2	54.094.552	245.115.392	18,08%
July0350_R1	39.984.681	180.676.691	18,12%
July0350_R2	35.103.371	185.558.001	15,91%
July0407_R1	33.902.791	145.690.071	18,88%
July0407_R2	28.577.649	151.015.213	15,91%
July0428_R1	38.371.576	169.584.268	18,45%
July0428_R2	33.998.086	173.957.758	16,35%
July0446_R1	60.466.379	247.639.947	19,63%
July0446_R2	54.713.703	253.392.623	17,76%
July0476_R1	46.521.609	194.110.385	19,33%
July0476_R2	38.421.502	202.210.492	15,97%
July0667_R1	36.773.248	157.160.613	18,96%
July0667_R2	32.388.385	161.545.476	16,70%

Table 19: Lost Reads, Raw Reads, and Percentage of Lost Reads for each sample.

As we did for the Nanopore dataset, we will make a summary that will help us understand the percentage of lost data from each previous step : trimming and mapping out. This information should be taken into account when we will perform the taxonomic profiling.

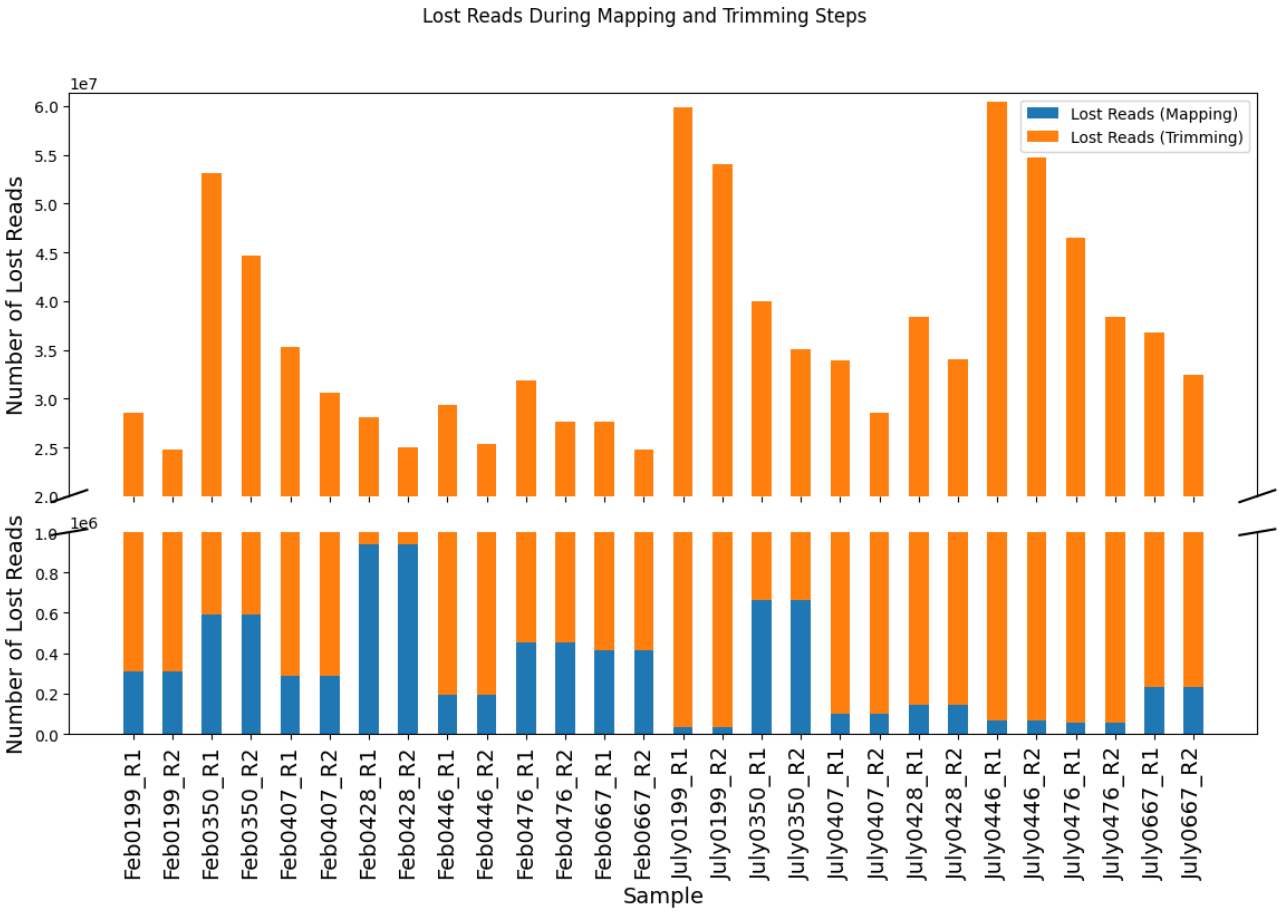


Fig. 39: Lost Data Trends for the Pre-Processed Steps for Illumina

Figure 39 (see G.19 for code source) allows us to better understand the data loss across the Illumina dataset. For the `Mapping Out` step, we could say that the following samples could influence the taxonomic classification results :

- `Feb0428`
- `Feb0350`
- `July0350`

Meanwhile, for the `Trimming` step, we identified other samples that didn't follow the pattern of the other samples:

- `Feb0350`
- `July0199`
- `July0446`

3.2.4 Taxonomic Classification

Concerning the taxonomic classification, we will take into account 3 different cases :

1. perform the taxonomic profiling only using `sourmash`
2. perform the taxonomic profiling using `Kraken2` and then run the results from `Kraken2` with `Bracken`
3. perform the taxonomic profiling using `Kraken2` and then run the results from `Kraken2` with `sourmash`

We will proceed with these 3 cases to obtain more information about the metagenomic content present in our trimmed and unmapped Illumina dataset.

Sourmash

For the part where we only employ `sourmash`, we will use a Bash script (see G.14) that will give out the following results :

- a krona output
- a kreport output

These two reports will be more than necessary and will help us analyze the final result of the taxonomic classification. Based on the kreport output we can find the percentage of unclassified reads from each one of the samples.

Sample Name	Percentage of classified reads	Number of Species
Feb0199	8,53%	2.572
Feb0350	6,33%	3.057
Feb0407	7,9%	2.797
Feb0428	8,23%	2.541
Feb0446	7,76%	2.528
Feb0476	7,96%	2.867
Feb0667	7,81%	2.610
July0199	7,38%	2.882
July0350	7,91%	2.822
July0407	7,53%	2.569
July0428	7,66%	2.573
July0446	6,79%	2.888
July0476	7,76%	2.396
July0667	6,88%	2.383

Table 20: Percentage of Classified Reads and Number of Classified Species after Taxonomic Profiling with `sourmash` on Illumina

Table 20 exhibits the percentage of classified reads done by `sourmash` and the number of species found across those classified reads. The percentage ranges from 6,33% to 8,53% for an average number of species of 2.700. There seem to be little to no correlation between the percentage of classified reads and the number of species found. A good example would be the following samples:

- `Feb0199` with 2.572 species found for 8,53% of classification
- `Feb0350` with 3.057 species found for 6,33% of classification

Kraken2

Concerning the taxonomic classification with `Kraken2`, before moving forward with the visualization results generated by both `Krona` and `Pavian`, we will analyze shortly different plots in order to understand our unmapped, trimmed Illumina dataset.

Sample Name	Percentage of classified reads	Number of Species
Feb0199	63,52%	16.947
Feb0350	64,43%	18.595
Feb0407	63,81%	17.408
Feb0428	62,46%	17.332
Feb0446	64,06%	16.934
Feb0476	63,69%	17.640
Feb0667	63,34%	16.870
July0199	63,13%	17.864
July0350	63,55%	17.995
July0407	63,98%	17.186
July0428	63,62%	17.107
July0446	63,76%	17.855
July0476	64,03%	16.464
July0667	63,26%	16.989

Table 21: Percentage of Unclassified Reads and Number of Classified Species after Taxonomic Profiling with `Kraken2` on Illumina

Unfortunately, Table 21, compared to Table 20, doesn't show visible variations concerning the percentage of unclassified reads for each one of the samples. Compared to the taxonomic profiling done with `sourmash`, a visible improvement is the number of classified reads, where we pass from an overall percentage value of 8% with `sourmash` to almost 36,5% with `Kraken2`. This fact alone proves that `Kraken2` is a better taxonomic classification tool than `sourmash`, but further inspection is required before closing this case.

Bracken VS sourmash

For this subsection, we will mainly compare `Bracken` and `sourmash` by running them on the unclassified reads from the taxonomic profiling with `Kraken2`.

As we pursue our analysis, we will dive into `Bracken` and `sourmash` and see what do we gain as information at what supplementary cost.

Sample Name	Classified reads Kraken2	Classified reads Bracken
Feb0199	63,52%	63,52%
Feb0350	64,43%	64,43%
Feb0407	63,81%	63,81%
Feb0428	62,46%	62,46%
Feb0446	64,06%	64,06%
Feb0476	63,69%	63,69%
Feb0667	63,34%	63,34%
July0199	63,13%	63,13%
July0350	63,55%	63,55%
July0407	63,98%	63,98%
July0428	63,62%	63,62%
July0446	63,76%	63,76%
July0476	64,03%	64,03%
July0667	63,26%	63,26%

Table 22: Percentage of Classified Reads by Kraken2 and Bracken for Different Samples

Table 22 is showing us the classifying the reads with `Kraken2` or with the suite Kraken2: `Kraken2/Bracken` will result in the same percentage of classification.

Sample Name	Nbr of species Kraken2	Nbr of species Bracken
Feb0199	16.947	16.895
Feb0350	18.595	18.538
Feb0407	17.408	17.354
Feb0428	17.332	17.282
Feb0446	16.934	16.882
Feb0476	17.640	17.589
Feb0667	16.870	16.819
July0199	17.864	17.807
July0350	17.995	17.939
July0407	17.186	17.129
July0428	17.107	17.055
July0446	17.855	17.799
July0476	16.646	16.596
July0667	16.989	16.937

Table 23: Number of Species Identified by Kraken2 and Bracken for Different Samples

Table 23 is showing the number of species found with `Kraken2` and then with both programs: `Kraken2` and `Bracken`. The values between these columns seem to be similar with a slight difference. There are fewer species found by `Kraken2/Bracken` then by only `Kraken2`.

3.2.5 Data Visualisation

As we approach the end of this pipeline, we approach the visualization part of the taxonomic classification results. For this step we will employ `Krona` and `Pavian`. Regrettably, based on the theory (see subsection `Pavian` of 2.5) and multiple tests ran during the internship, `sourmash`'s kreport isn't compatible with `Pavian`. This is the main reason why we are employing both visualization tools on the output of `Kraken2` and only `Krona` on the output of `sourmash`.

To be able to compare the samples between them by employing `sourmash` and be able to simplify the visualization, we will highlight *Archaea* due to their relatively low abundance. By emphasizing less abundant groups, it will help to reduce the visual clutter and make the data easier to interpret.

Krona on sourmash

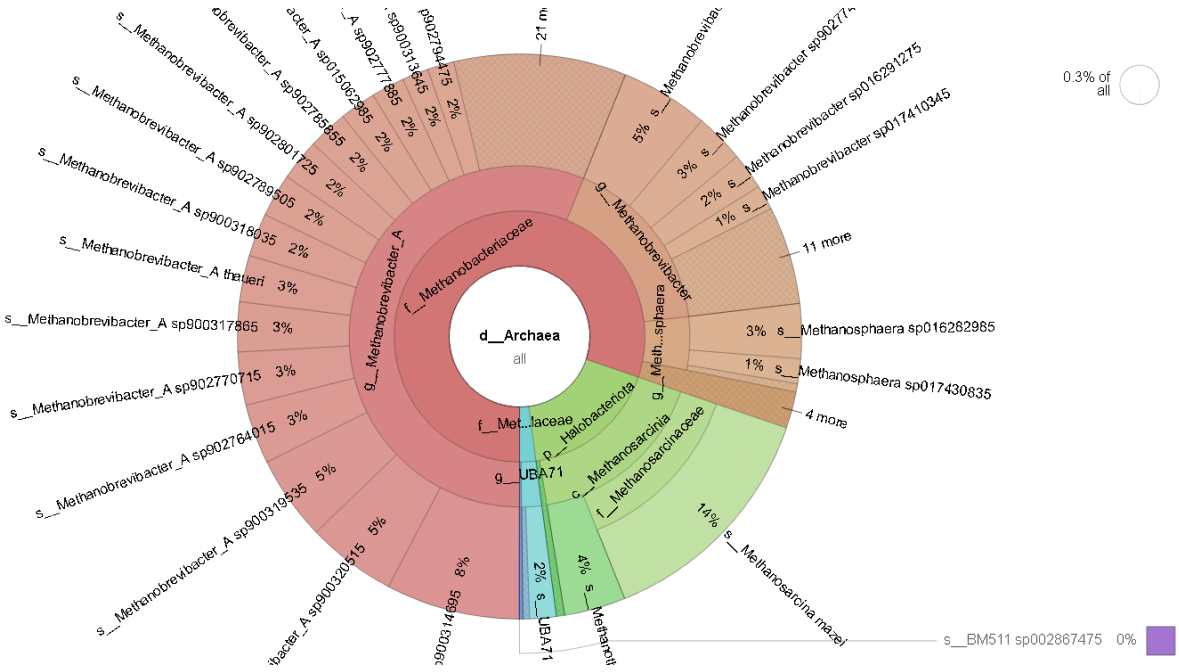


Fig. 40: Krona report for Feb0199 Illumina (sourmash)

In Figure 40, we see the classification of `Feb0199` from the Illumina dataset using `sourmash`. The main focus is shifted on the domain of *Archaea*, which represent 0,3% of the sample. Out of those 0,3%, we distinguish 60% from the family *Methanobacteriaceae*, 18% from the phylum *Halobacteriota* and the 2% from other family.

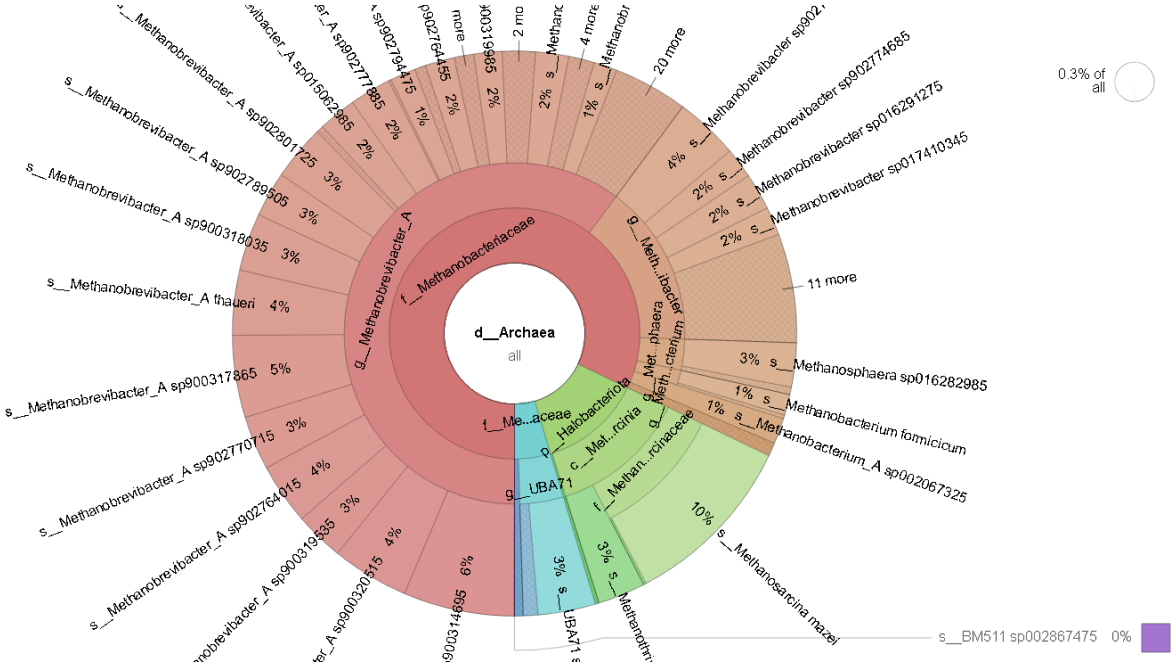


Fig. 41: Krona report for July0199 Illumina (sourmash)

Compared to Figure 40, Figure 41 displays the same phyla and families, but with slightly different percentages. Sample `July0199` contains around 60% of *Methanobacteriaceae* family, 13% of *Halobacteriota* phylum and the rest from another family.

As we clarified in the previous paragraph, we will highlight *Archaea* to compare the samples between them. The rest of the `Krona` output can be found in the Appendix (see F.2.1).

Krona on Kraken2/Bracken

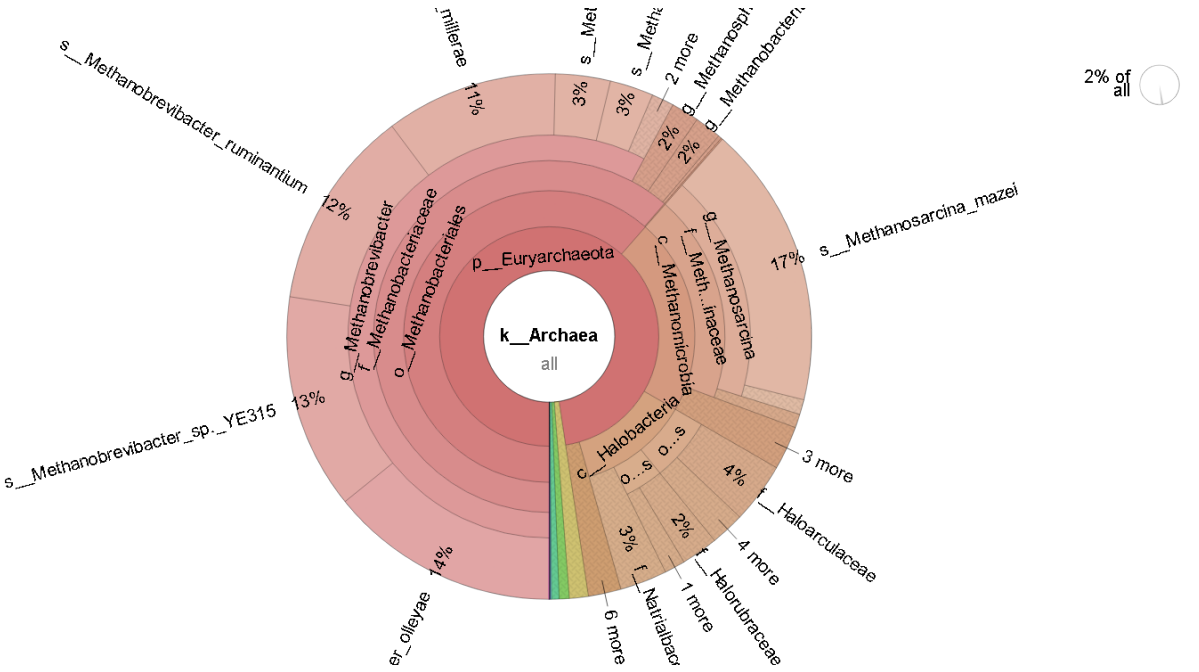


Fig. 42: Krona report for Feb0199 Illumina (Kraken2/Bracken)

Figure 42 exhibits a more complex taxonomic profiling results, by showing every taxa level. Using `Kraken2/Bracken` will give out a different `Krona` report compared to the previous one. We can distinguish that almost 95% of the sample is *Euryarchaeota* phylum. Out of this 95% there are 3 main branches : the class of *Methanomicrobia* with 22%, the class of *Halobacteria* with 13%, the order of *Methanobacteriales* with 63% and other organisms.

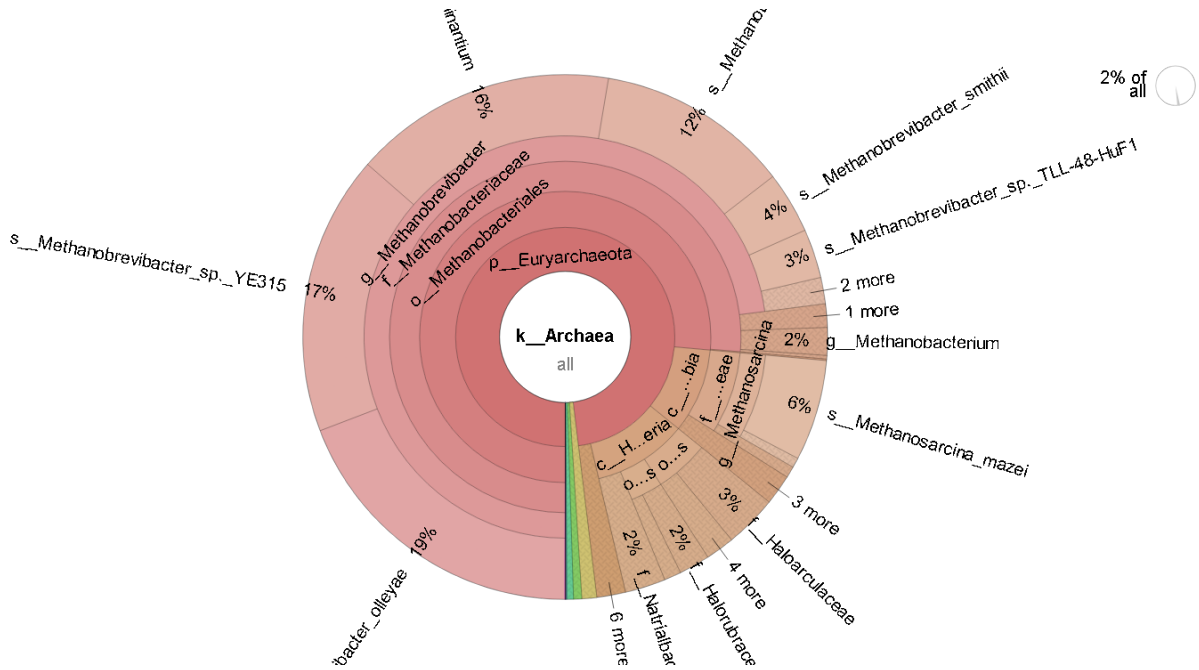


Fig. 43: Krona report for July0199 Illumina (Kraken2/Bracken)

The previous figure (Figure 43) displays a quite different behaviour compared to the sample in `Feb0199`. For this sample, phylum *Euryarchaeota* represents 98% of *Archaea* and contains the same elements: *Methanobacteriales* order (78%), *Halobacteria* class (11%), *Methanomicrobia* class (10%) and other organisms.

As we did for the previous paragraph, the rest of the Krona plots can be observed in the Appendix (see F.2.2).

Pavian on Kraken2 / Bracken

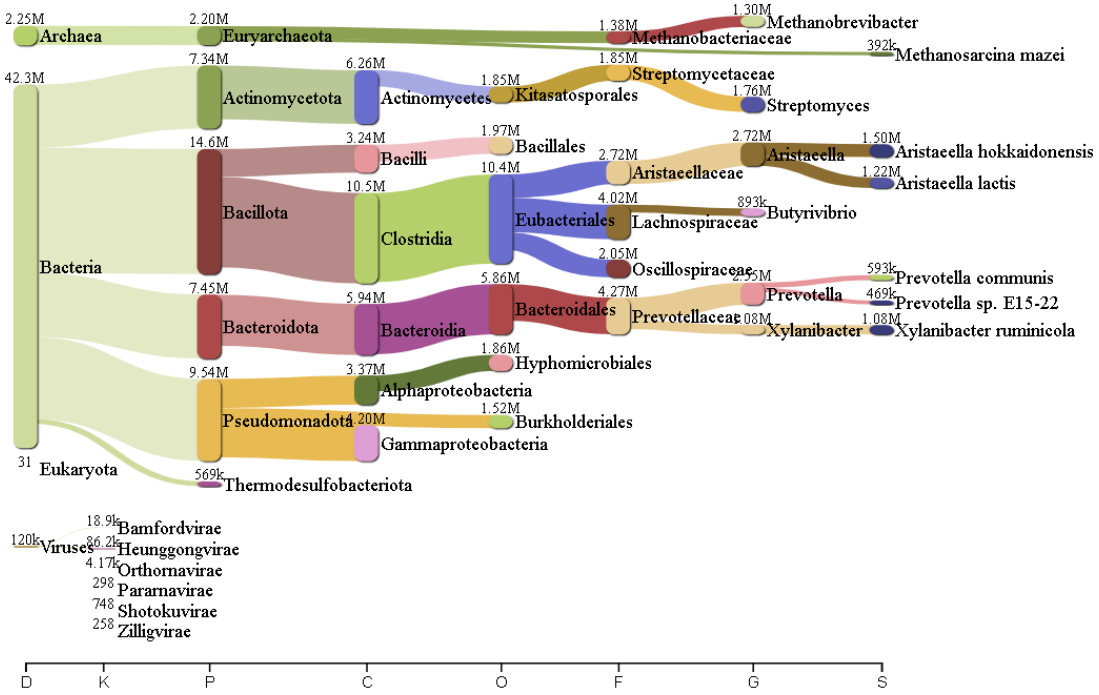


Fig. 44: Pavian report for Feb0199 Illumina (Kraken2/Bracken)

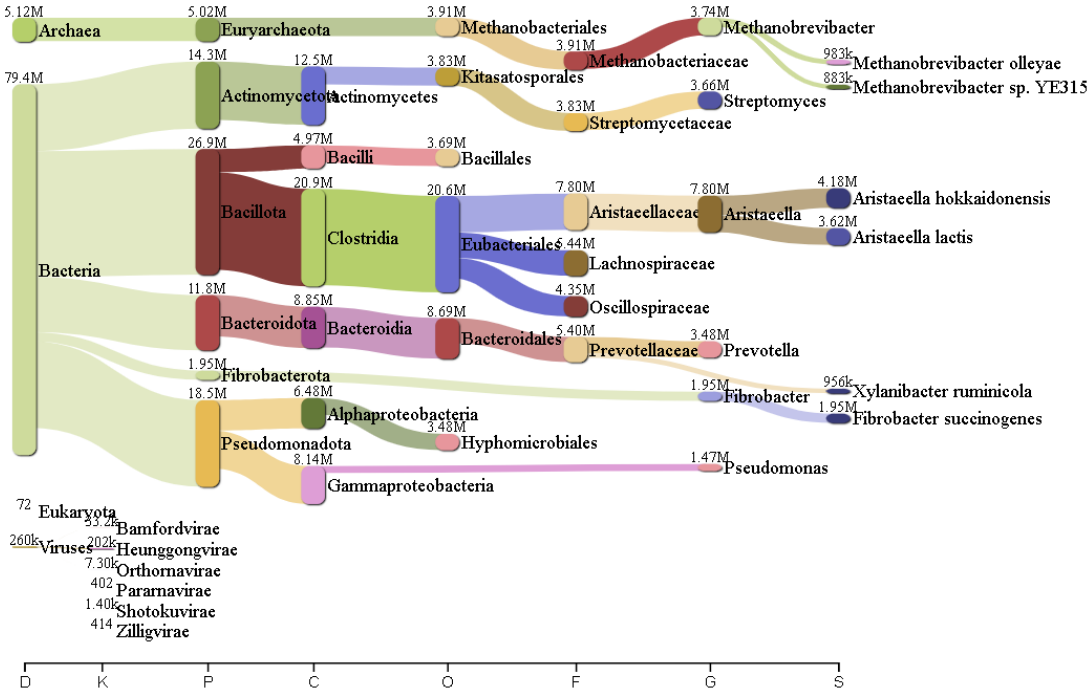


Fig. 45: Pavian report for July0199 Illumina (Kraken2/Bracken)

Lastly, for this section of data visualisation the rest of the Pavian plot can be found also in the Appendix section (see F.2.3).

4 Discussions

4.1 Nanopore

4.1.1 Quality Control

Figure 14 provides us with the number of reads across multiple samples. A significant difference is visible, with fewer reads in July compared to February. Additionally, one sample (Run_2-Feb0667) is missing in the February dataset, potentially due to failed or missing sequencing. There are multiple reasons that can explain this anomaly:

- A first element is that samples were multiplexed, meaning that different samples were sequenced together. This can explain the differences that we encountered in the quality control, especially since we pooled the samples from the same individual
- Lastly, the particularity of the nanopore reads is that we can have better sample while having fewer reads simply because they are longer (a better way to compare these samples is the compare them based on the number of bases in each sample than the number of reads)

Figure 14 shows the level of duplication across the dataset. Although Nanopore sequencing typically doesn't exhibit duplicated reads, certain samples do show low but noticeable duplication levels. This variation can be observed by the black bar at the end of each row from Figure 14. Table 5 summarizes this, revealing that samples from individual `0350` tend to have higher duplication rates. Despite the percentage being below 1%, this pattern is consistent and may reflect unique sample characteristics.

Figure 15 presents the sequence quality. While early reads exhibit good quality (Phred scores 15-30), variations arise as sequencing progresses. Some notable deviations are highlighted, particularly in samples like Run_1-July0476 and those with longer sequences (Run_2-Feb0428, Run_1-July0667, Run_2-July0428).

The quality anomalies in Run_1-July0476 (Figure 16) may be explained by the base content (Figure 17), where cytosine and thymine levels drop significantly, leaving only adenine and guanine. This phenomenon can be explained by the fact that some nucleotides (in this case, adenine and guanine) could have been blocked in the pore of the flow cell, giving out reads containing those two as long tail reads when, in fact, the whole reads was sequenced.

Run_2-Feb0428 (Figure 18) shows a more stable quality score, typical of Nanopore reads, while Run_1-July0667 (Figure 19) exhibits a drop in quality early on, which can be linked to the base content variation in Figure 20. These weird patterns can be explained by possible sequencing errors and should be discarded for the further steps of the analysis.

Lastly, Run_2-July0428 (Figure 21) also shows longer reads, which may offer additional insights during further analysis.

In conclusion, the quality check reveals that individual `0350` has a higher duplication rate and that some samples with longer reads may contain more information for analysis, notably:

1. Run_2-Feb0428
2. Run_1-July0667
3. Run_2-July0428

4.1.2 Trimming

The trimming step aimed to improve the overall quality of the Nanopore dataset. The heatmaps in Figures 23 and 24 highlight a clear improvement in quality after trimming, particularly in the per-base sequence quality (black box). However, some samples lost quality (purple, blue, and yellow boxes) due to the trimming based on a Phred score of 10. This "drop" in quality can be explained by the fact that we removed the wrong part of our sequences and that we lost some data. This loss of data/shortening of the reads can influence this drop in the overall quality of some samples. Notable the affected samples include:

- Run_2-July0350 (per base sequence content quality dropped from 1 to 0,3)
- Run_2-July0407 (per base sequence content quality dropped from 1 to 0,5)

The MultiQC reports further illustrate this improvement. Figure 15 shows that the entire dataset (25/25) failed the quality check before trimming, whereas Figure 25 reveals a significant improvement, with only 15 samples failing and the rest showing warnings. We can see that the filtering did not remove the "tower" visible

in Figure 25 (sample Run_1-July0476); this might be an issue in further analysis. Unfortunately the lack of removal of the reads/reads causing this was overlooked, and thus, the subsequent analysis was done with it remaining in the sample. The X-axis (Position (bp)) decreasing from over 220.000 to around 180.000 confirms that trimming with chopper based on the Phred score works effectively, resulting in a higher quality dataset despite the reduction in data volume.

Further analysis of the specific samples identified initially (Run_2-Feb0428, Run_1-July0667, Run_2-July0428) shows obvious improvements. However, these improvements also come with potential inconveniences, as illustrated in Figures 26 and 27. The samples Run_1-July0667 and Run_2-July0428 experienced significant data loss but gained in Phred score until a sudden quality drop occurred at the end. Conversely, Run_2-Feb0428, which had better overall quality, remained largely unchanged.

These findings highlight several important points. The overall reduction in data volume due to trimming is a trade-off for improved sequence quality. While the trimming step has succeeded in enhancing the dataset's overall quality, as evidenced by the MultiQC and heatmap analyses, the loss of reads in specific samples presents a potential concern. Future analyses may need to consider whether the improvements in quality justify the extent of data loss, or if alternative trimming strategies could achieve a better balance.

In addition, the observed data loss in samples such as Run_2-Feb0428, which initially had a high read count, emphasizes the need for careful examination of the trade-offs involved. This sample's length was notably larger than others, and despite its good initial quality, the trimming process reduced its length significantly, which might affect downstream analyses.

The reduction in per base sequence content score for samples like Run_2-July0350 and Run_2-July0407 further suggests that while trimming can improve overall data quality, it can also have selective impacts on certain samples. This highlights the importance of tailoring trimming parameters to the specific characteristics of each dataset to optimize results.

Overall, the trimming process has effectively filtered out lower-quality data, leading to a more reliable dataset for subsequent analyses. However, researchers should remain cautious about the impact of such pre-processing steps on the data and carefully consider how they may affect the interpretation of results. Future work might explore alternative methods or adjustments to the trimming parameters to better balance data quality and quantity, aiming to minimize data loss while maintaining high-quality sequencing results.

4.1.3 Host Removal

Interestingly, the samples that experienced the highest and lowest read losses during the trimming step (see Table 6) also exhibited similar patterns in the subsequent mapping step (see Table 7). This consistency suggests that the reads affected by the trimming process may also be more prone to loss during the mapping out process. Specifically, the samples with the most significant data reduction post-trimming continued to show high read loss after mapping, while those with minimal loss during trimming maintained relatively low losses in the mapping step.

Based on Table 19, we can deduce that the percentage of lost reads varies from one sample to another and generally falls in the range of 14,3% and 20,03%. If we compare the seasons between them, we can notice that there is a higher loss in July with two samples that show the highest percentage of lost reads : July0199_R1 and July0446_R1. Speaking of extremities we have on the side of the highest percentage loss : July0199_R1 (20,03%) and July0446_R1 (19,63%), while on the other side we have Feb0476_R2 with the lowest fraction of lost reads (14,3%). This variation shown in Table 19 implies that some samples have less usable data for the analysis, potentially impacting the overall results. The higher loss in some samples from July could be linked to various reasons, such as seasonal effects on the host, environment, extraction methods, or even sequencing-specific factors.

If we add together the percentage of lost reads for each sample from both the trimming and mapping out step, we deduce that the highest percentage is around 30%, while the lowest is around 20%. With this information in mind, we can say that between the raw data and the data that is going to be classified, we lose between 20 and 30% of reads in general. These values fluctuate from one sample to another.

4.1.4 Taxonomic Profiling

sourmash

Analyzing Table 8 reveals a low percentage of classified reads. This indicates that the classification rates are consistent between the two runs (despite some samples missing in the second run, as detailed in Section 1.2). This consistency suggests that the method is reproducible, even though the overall classification rates are low.

The low percentage of classified reads can be attributed to several factors, such as an incomplete reference database, `sourmash` limitations, and the sequencing data quality.

Secondly, Table 9 is illustrating different values for the number of species discovered by `sourmash` in each sample ranging from 463 to 1.311 with some cases indicating high variation between the runs. This can be explained by the way multiplexing works (see 1.2).

Once we are able to explain at least this anomaly concerning the percentage of classification, we will pursue a small comparison between the samples based on the 4 phyla that we cited previously (see 2.5). We used these 4 phyla to compare the samples and understand their variations: *Bactoreidetes* (also known as *Bacteroidota*), *Firmicutes*, *Proteobacteria* and *Actinobacteria*.

In order to visualize the phyla comparison for the trimmed and unmapped Nanopore dataset, we will utilize Figure 28 (see G.18 for code source). This figure shows that generally 33% of the phyla from the samples are *Bacteroidota*, while the *Firmicutes* and *Proteobacteria* are showing values almost inferior to 5% and the *Actinobacteria* which are nonexistent in our sample.

Kraken2

When we try to classify the Nanopore samples with `Kraken2`, there is a noticeable decrease in the percentage of classified reads from Run_1 to Run_2 (see Table 10), indicating that the second run may have different or more challenging conditions for classification (phenomena that can be explained via Section 1.2). Comparing `Kraken2` with `sourmash` illustrates clear differences concerning the percentage of classification (see Tables 10 and 8) and the number of reads discovered by each program (see Tables 9 and 11). These values can be explained by the following:

- Algorithm Difference: based on the way `sourmash` works (see `sourmash` section in 2.5), we will obtain a fast and approximate taxonomic result but not as comprehensive in terms of species detection, while `Kraken2` (see `Kraken2` section in 2.5) is more an exhaustive algorithm designed to be highly sensitive and more comprehensive in taxonomic classification
- Database Size and Composition: the difference in the databases represents an important factor in the classification rate and species count. The database employed by `Kraken2` is more curated with comprehensive species representation, which will allow the tool to detect a broader range of species. On the other hand `sourmash` is using an estimating sequence similarity that might not leverage as detailed a reference
- Sensitivity VS Specificity: the results from `Kraken2` (see Tables 10 and 11) indicate that this tool has higher sensitivity, meaning it detects a larger number of sequences belonging to known species. However, this increased sensitivity comes with a trade-off, as the classification might be less precise. In contrast, `sourmash` (see Tables 8 and 9) exhibits lower sensitivity than `Kraken2` but is likely more conservative, focusing on more certain matches. This conservatism may result in missing some species present in the samples.
- Practical Implications: `Kraken2` might be more suitable for detailed taxonomic profiling, capturing as much diversity as possible, while `sourmash` is preferable for rapid, approximate comparisons for large-scale studies even if it comes at the cost of reduced sensitivity

Bracken VS sourmash

For this section, we will compare the performance of `Bracken` and `sourmash`, and we will run on the results from `Kraken2`. Table 14 shows that we obtain low classification percentages after running `sourmash` on the unclassified reads of `Kraken2`. These low values can be explained by `sourmash` functionality, which allows us to obtain a low percentage of classification. On the other hand, Table 15 represents the adjusted percentage of the classification done with both tools : `Kraken2` and `sourmash`. This alone means that using both tools, may allow us to obtain a better classification but with a slight improvement that ranges from 3,32% to 5,61%. Another element that is visible when we compare the adjusted percentage between the runs is that the values seem to be quite close to each other.

Another element that is linked to a small improvement of the taxonomic classification when `sourmash` is run after `Kraken2` is the number of species that were discovered. These numbers vary from 78 to 281 species. Unfortunately, compared to the number of species classified by `Kraken2`, we were able to add 1,8-2% of newly classified species to the initial number of species profiled by `Kraken2`.

`sourmash` was able to add more valuable information to our analysis and help us to increase the percentage of classification and profile of new species that `Kraken2` may have missed out on in the beginning.

4.1.5 Data Visualisation

Figure 29 and 30 display the microbial community from the samples `Run_1-Feb0199` and `Run_1-July0199` under the kingdom of *Archaea*. We can see clearly that the plots exhibits a dominant phylum, *Euryarchaeota*, containing a high proportion of *Methanobrevibacter*. The proportion of these samples suggests that there are no visible differences in the microbial community at least for this kingdom.

On the other hand when we visualize the taxonomic results done with `Kraken2/Bracken` (see Figures 31 and 32) it becomes clear that we may obtain a similar taxonomic classification. Additionally, to `sourmash`, the Kraken suite seems to be giving out a more proper taxonomic classification, by displaying all the taxonomic levels.

By using `Pavian` to visualize the taxonomic results (see Figures 33 and 34), it seems that we are able to better visualize two important factors:

- the fact that we obtain an incomplete taxonomic classification for some species (e.g. : *Methanobrevibacter* being found in `Run_1-Feb0199`)
- the number of reads associated with these organisms seem to have quite large differences between them :
  - *Lachnospiraceae* with 122.000 and 61.400 reads for `Run_1-Feb0199` and `Run_1-July0199`, respectively
  - *Prevotellaceae* with 108.000 and 58.900 reads for `Run_1-Feb0199` and `Run_1-July0199`, respectively

To make easier to understand the hypothesis from above, below you can find the `Pavian` reports for `Run_1-Feb0199` and `Run_1-July0199`.

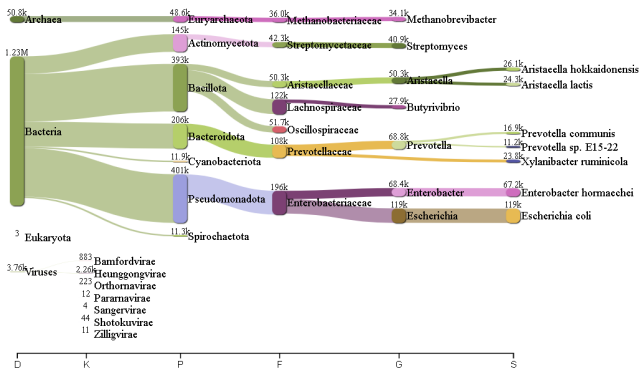


Fig. 46: Pavian report Run_1-Feb0199

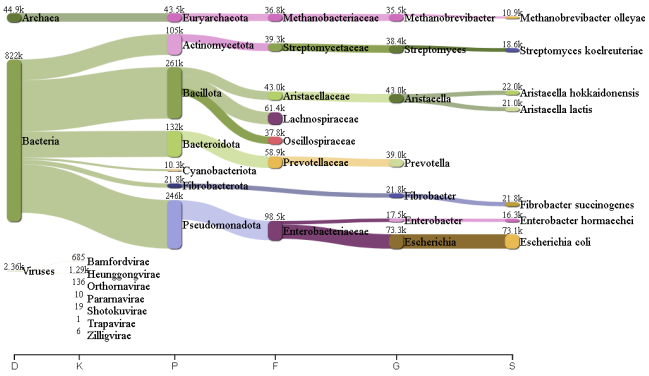


Fig. 47: Pavian report Run_1-July0199

4.2 Illumina

4.2.1 Quality Control

When checking the quality control generated by `MultiQC`, one of the first elements that catches our attention is the duplication level, which can be visualized in Figure 35. This factor can be explained by the simple theory behind the Illumina sequencing techniques (see 1.4), where by having shorter sequences, the chances of obtaining an identical copy are higher than the one for the Nanopore sequences. Figure 35 shows similar levels of read duplication for each one of the samples, but to be sure of this hypothesis, we will plot the number of reads and the percentage of duplicated reads.

Another element that can be visible in Figure 35 is that almost every sample contains a similar number of reads. However, only three samples seem to have more reads than most of the dataset, and it should be interesting to pay closer attention to these samples. The reasoning is, quite simple: by having more reads, there is a higher chance of finding out more individuals. Unfortunately, when we take this hypothesis into account, we do not need to neglect the possible individuals that can be identified in the other samples. The samples in question that contain a higher number of reads, compared to the rest of the dataset, are `WD-3658-Feb0350`, `WD-3658-July0199` and `WD-3658-July0446`.

Figure 36 represents an additional argument that helps us understand the high percentage of duplicated reads that are present in our Nanopore dataset. This feature was already visible in Figure 35, but with this scatter plot, we can better visualize and conclude that the overall percentage of duplicated reads is around 11-12%. This behaviour is solid proof of how Illumina sequencing works. This information is crucial for us to

understand because, in the trimming step, we will lose 11-12% of data, eventually making us come closer to it by correcting a potential sequencing bias.

We can observe via Figure 35 the same pattern more or less identified on the Nanopore dataset. These two figures show visible variations, which can be explained by various reasons such as the individual's diet (see 2.1), sampling, extracting, quality and concentration sequencing. Across the dataset there are multiple individuals that are not following the same pattern. The respective individuals are :

- `0199` going from 155.874.549 (in February) to 299.209.944 reads (in July)
- `0446` going from 160.152.629 (in February) to 308.106.326 reads (in July)

It becomes clear that individuals `0199` and `0446` need further inspection for the next steps of the analysis due to their huge differences between the periods (we have almost double the reads from February to July).

The analysis of quality control metrics, especially duplication levels, supports the understanding that Illumina sequencing tends to produce more duplicated reads due to shorter sequences. The presence of a similar number of reads across most samples, with exceptions such as `WD-3658-Feb0350`, `WD-3658-July0199`, and `WD-3658-July0446`, highlights the need for careful interpretation. Samples `0199` and `0446` particularly stand out for further investigation due to their substantial increase in reads from February to July, which may affect downstream analysis and conclusions.

4.2.2 Host Removal

For the July dataset, there is only one sample in which we lose a significant amount of data after removing the DNA of *Bos Taurus*: `July0350`, with 664.339 reads lost (representing 0,3% of the total number of reads).

If we take into account the number of lost reads for both periods for individuals `0199` and `0446`, which we identified in the previous step as 'special' due to their large amount of reads, we observe that both follow the pattern explained in the previous paragraph. Specifically, we lose more data in February than in July for both individuals, and they tend to follow the curve for each of the periods.

This discrepancy might be attributed to variations in host DNA contamination, which can occur when sampling by inserting a tube into cow's throat to pump the stomach contents. In this process, host cells are inevitably collected, and the amount of host DNA may vary depending on the cow, the time of sampling, and how effective and well executed the steps preventing the contamination (e.g. discarding the first part of the pump content to avoid host cells). Furthermore, only a subset of the entire sample is taken during DNA extraction, which can result in differences in host DNA content between extractions, potentially explaining the variation between the Nanopore and Illumina datasets.

In conclusion, while the overall quality of our sequences did not show visible changes according to the MultiQC reports, the number of reads lost after removing the DNA of Bos Taurus did vary. The samples most affected by this step were `Feb0428` and `Feb0350` from the February sub-dataset, and `July0350` from the July sub-dataset.

4.2.3 Trimming

The level of duplicated reads decreases drastically from an average percentage of 11-12% (see Figure 38a) to less than 3% (see Figure 38b), demonstrating a significant improvement in the quality of our Illumina dataset. However, during the trimming step, we encountered an issue with the removal of duplicated reads using `seqkit`. This software tends to remove unique sequences along with duplicate ones, which could impact our taxonomic classification results by potentially losing specific or smaller individuals, leading to a taxonomic outcome that might not accurately represent the initial dataset. Despite this loss, we aim to maintain the quality of taxonomic classification for the species that remain.

To better understand the impact of `seqkit` on unique reads, we will analyze the ratio of lost reads between the `Mapping Out` and `Trimming` steps for each period: February and July.

In Table 19, we observe that the number of lost reads is generally below 30.000000 across most samples, with a few exceptions, such as `Feb0350_R1` and `Feb0350_R2`, which do not follow this pattern.

In contrast, the July dataset shows more variation in the number of lost reads per sample. The median value for lost reads is around 40.000.000, with some samples, such as `July0199_R1`, `July0199_R2`, `July0446_R1`, `July0446_R2`, and `July0476_R1` stand out from the others.

The drastic reduction in duplicated reads from 11-12% to less than 3% highlights the improvement in data quality after trimming, although the removal of unique sequences by `seqkit` poses a challenge. This issue could affect the accuracy of taxonomic classification, but it is a necessary step to ensure high-quality results for the species that remain.

Our analysis shows varying patterns of data loss between February and July, with some samples, particularly in July, losing significantly more reads. Understanding this loss, both in absolute numbers and percentages, is crucial for evaluating the impact on downstream analyses, particularly in the `Taxonomic Classification` step.

4.2.4 Taxonomic Profiling

Sourmash

Once we are able to explain at least this anomaly concerning the percentage of classification, we will pursue a small comparison between the samples based on the 4 phyla that we cited previously (see 2.5). We used these 4 phyla to compare the samples and understand their variations: *Bacteroidetes*, *Firmicutes*, *Proteobacteria* and *Actinobacteria*.

Tables 20 and **??** tend to display an unexpected behavior compared to taxonomic profiling done with `sourmash` on the Nanopore data. An obvious inverse relationship exists between the percentage of classification and the number of reads discovered. Generally, the samples with fewer unclassified reads tend to discover more species for example :

- `Feb0350` with 3.057 found species for a percentage of classification of 6,33%
- `July0446` with 2.888 found species for a percentage of classification of 6,79%

The tables display a positive trend: samples with higher percentage of classification generally have better taxonomic results. However, the number of species may not directly correlate with this percentage of classified reads.

For the usage of `sourmash`, this tool demonstrated consistent but low percentages of classification across samples, indicating some possible limitations linked to the classification efficiency. This can be explained by multiple factors such as an incomplete reference database and inherent limitations of `sourmash`. The variation in the number of detected species suggests that the multiplexing and the sequencing quality can be the cause.

Kraken2

The results observed in Tables 21 and **??** indicate that `Kraken2` significantly outperforms `sourmash` in terms of the percentage of read classification and diversity of species identified. `Kraken2` reaches a percentage of classification of around 63,5% on average which represents an increase of 8% observed with `sourmash`. This comparative analysis solidifies the fact that `Kraken2` is a more effective tool for taxonomic profiling in the context of metagenomics.

Compared to `sourmash`, `Kraken2` shows a higher percentage of classified reads and a broader detection of species. The higher sensitivity and comprehensive database contribute to its ability to classify a significant number of species accurately. Unfortunately, `Kraken2`'s sensitivity comes with some precision trade-offs that can impact classification accuracy.

Bracken VS sourmash

Firstly, before comparing `Bracken` and `sourmash`, we will try to observe the possible improvements that these tools can have on the taxonomic classification with `Kraken2`. Table 22 shows that the proportion of classified reads by using `Kraken2` is around 22% while using `Bracken` also will add up to a value of 7,8% of classified reads. Even though the values don't fluctuate too much, we can distinguish the fact that we tend to classify more reads in February than in July.

A second table (Table 23) will help us understand the effects of `Bracken` on the results from `Kraken2`. Sadly, we obtain the same number of species across the samples of the Illumina dataset for both `Kraken2` and `Bracken`. This can be explained by the way `Bracken` works. These tools will get the results from `Kraken2` and estimate the species abundance (see the subsection `Bracken` from 2.5).

`Bracken`'s performance, when combined with `Kraken2` is showing visible improvements in classification percentages. The added value of `Bracken` in refining those of `Kraken2` was limited (1,8-2%). Even though the changes are small, it becomes clear that `Bracken` is enhancing the results from its predecessor.

4.2.5 Data Visualization

As we can observe in Figures 40 and 41, `sourmash` provides a quite complex classification mainly containing *Methanobacteriaceae* and *Halobacteriota*. Unfortunately the classification seems to miss multiple taxonomic

ranks as the classification goes directly from the domain of *Archaea* to the family of *Methanobacteriaceae*. Even though we may judge this result as complex, by using `sourmash` we can see that *Archaea* represent 0,3% of the whole sample.

Compared to `sourmash` the suite `Kraken2/Bracken` proves again that they are better suited for this kind of analysis, via Figures 42 and 43. As we can observe, the taxonomy seems to be more complete than the one done with `sourmash` and the domain of *Archaea* represents 2% of the entire sample.

Lastly Figures 44 and 45 illustrate the main organisms found in the individual `0199` for both periods. The main structure seems similar, with only the number of reads and certain species varying between the samples. Here are the differences concerning the genus *Methanobrevibacter* :

- both samples contain this genus but in different quantities:
  - 1.3 million of reads for `Feb0199`
  - 3.74 million of reads for `July0199`
- we are able to go further with the classification from this genus with `July` and classify two species : *Methanobrevibacter olleyae* and *Methanobrevibacter sp. YE315*

5 Conclusion

The automation of metagenomic analysis, with a strong emphasis on FAIR (Findable, Accessible, Interoperable, and Reusable) principles, represents a significant leap forward in the field of bioinformatics. Throughout this thesis, I have demonstrated how the integration of automated processes can enhance the efficiency of shotgun metagenomic analysis. By automating key steps such as quality control, trimming, host removal, and taxonomic classification, the time and resources required for high-throughput data analysis are significantly reduced, ensuring consistent and reproducible results. Even though this workflow was developed during this bachelor's thesis, there are elements that can be improved on different steps in order to enhance the quality of the results for further analysis.

The results of my internship and thesis are that we now have automated scripts for the different steps of analysis for both Illumina and Nanopore data that can be used in future projects, scripts that have been crucial in calculating the best parameters and host genome for my sample set but that can be used also for future research. Furthermore, I generated the initial results that will be used in the research project, and the different intermediary data results can be reused for novel analysis if wanted/needed.

My experience in Sweden, particularly with my supervisor's guidance and the research team's support, was instrumental in the successful completion of this project. It was both professionally enriching and personally fulfilling, as I contributed an automated analysis that hopefully will be used for further research.

In conclusion, this thesis underscores the critical role of automation in modern bioinformatics and highlights the importance of FAIR principles in ensuring that scientific data can be leveraged to its full potential. The skills and experiences I gained during this project, especially in Sweden, have laid a strong foundation for my future career in bioinformatics, and I am deeply grateful to my supervisor, the entire team and my university professors for their constant support and mentorship.

References

1. Wooley, J. C., Godzik, A. & Friedberg, I. A Primer on Metagenomics. PLOS Computational Biology **6,** 1–13. `https://doi.org/10.1371/journal.pcbi.1000667` (**february** 2010).
2. Haynes, M. **in** Encyclopedia of Ecology Additional DOI: 10.1002/abio.370040210, 153–156 (Elsevier, 2008). `https://doi.org/10.1016/B978-0-444-63768-0.00933-1`.
3. Connelly, L. M. Pilot studies. Medsurg nursing **17,** 411 (2008).
4. Brum, J. R., Culley, A. I. & Steward, G. F. Assembly of a Marine Viral Metagenome after Physical Fractionation. PLOS ONE **8,** 1–10. `https://doi.org/10.1371/journal.pone.0060604` (**april** 2013).
5. Shi, Y., Wang, G., Lau, H. C.-H. & Yu, J. Metagenomic Sequencing for Microbial DNA in Human Samples: Emerging Technological Advances. International Journal of Molecular Sciences **23.** ISSN: 1422-0067. `https://www.mdpi.com/1422-0067/23/4/2181` (2022).
6. Quince, C., Walker, A. W., Simpson, J. T., Loman, N. J. & Segata, N. Shotgun metagenomics, from sampling to analysis. Nature Biotechnology **35,** 833–844. ISSN: 1546-1696. `https://doi.org/10.1038/nbt.3935` (2017).

7. Dijk, E. L. V., Auger, H., Jaszczyszyn, Y. & Thermes, C. Ten years of next-generation sequencing technology. Trends in Genetics **30,** 418–426 (2014).
8. Tedersoo, L., Albertsen, M., Anslan, S. & Callahan, B. Perspectives and Benefits of High-Throughput Long-Read Sequencing in Microbial Ecology. Applied and Environmental Microbiology **87,** e00626–21. eprint: https://journals.asm.org/doi/pdf/10.1128/aem.00626-21. https://journals.asm.org/doi/abs/10.1128/aem.00626-21 (2021).
9. Berardini, T. Z. **andothers**. Functional annotation of the Arabidopsis genome using controlled vocabularies. Plant Physiology **135,** 745–755. https://doi.org/10.1104/pp.104.040071 (**june** 2004).
10. Muzzey, D. **in** Noninvasive Prenatal Testing (NIPT) (**editors** Page-Christiaens, L. & Klein, H.-G.) 7–24 (Academic Press, 2018). ISBN: 978-0-12-814189-2. https://www.sciencedirect.com/science/article/pii/B9780128141892000025.
11. Wang, Y., Zhao, Y., Bollas, A., Wang, Y. & Au, K. F. Nanopore sequencing technology, bioinformatics and applications. Nature Biotechnology **39,** 1348–1365. ISSN: 1546-1696. https://doi.org/10.1038/s41587-021-01108-x (**november** 2021).
12. Zhang, L. **andothers**. Advances in Metagenomics and Its Application in Environmental Microorganisms. Frontiers in Microbiology **12,** 766364. https://doi.org/10.3389/fmicb.2021.766364 (2021).
13. Boran Cattle https://www.thecattlesite.com/breeds/beef/94/boran/. Accessed: 2024-06-28.
14. Boran: The Breed https://borankenya.org/?page_id=6063. Accessed: 2024-06-28.
15. Ethiopian Seasons https://seasonsyear.com/Ethiopia#content-4.
16. FutureLearn. Making Sense of Genomic Data for COVID-19: A Web-Based Bioinformatics Course https://www.futurelearn.com/info/courses/making-sense-of-genomic-data-covid-19-web-based-bioinformatics/0/steps/319526.
17. Ewels, P., Magnusson, M., Lundin, S. & Käller, M. MultiQC: summarize analysis results for multiple tools and samples in a single report. Bioinformatics **32,** 3047–3048. ISSN: 1367-4803. eprint: https://academic.oup.com/bioinformatics/article-pdf/32/19/3047/49021359/bioinformatics_32_19_3047.pdf. https://doi.org/10.1093/bioinformatics/btw354 (**june** 2016).
18. Decoster, W. Chopper: A tool for clustering and visualization of sequencing reads https://github.com/wdecoster/chopper. Accessed: YYYY-MM-DD. 2024.
19. Delahaye, C. & Nicolas, J. Sequencing DNA with nanopores: Troubles and biases. PLOS ONE **16** (**editor** Andre's-Leo'n, E.) e0257521. https://doi.org/10.1371/journal.pone.0257521 (**october** 2021).
20. Shen, W., Le, S., Li, Y. & Hu, F. SeqKit: A Cross-Platform and Ultrafast Toolkit for FASTA/Q File Manipulation. PLOS ONE **11,** 1–10. https://doi.org/10.1371/journal.pone.0163962 (**october** 2016).
21. The Go Authors. Go Documentation Accessed: 2024-08-09. 2024. https://go.dev/doc/.
22. Li, H. Minimap2: pairwise alignment for nucleotide sequences. Bioinformatics **34,** 3094–3100. ISSN: 1367-4803. eprint: https://academic.oup.com/bioinformatics/article-pdf/34/18/3094/48919122/bioinformatics_34_18_3094.pdf. https://doi.org/10.1093/bioinformatics/bty191 (**may** 2018).
23. Langmead, B. & Salzberg, S. L. Fast gapped-read alignment with Bowtie 2. Nature Methods **9,** 357–359. ISSN: 1548-7105. https://doi.org/10.1038/nmeth.1923 (2012).
24. Portik, D. M., Brown, C. T. & Pierce-Ward, N. T. Evaluation of taxonomic classification and profiling methods for long-read shotgun metagenomic sequencing datasets. BMC Bioinformatics **23,** 541. ISSN: 1471-2105. https://doi.org/10.1186/s12859-022-05103-0 (**december** 2022).
25. Brown, C. T. & Irber, L. sourmash: a library for MinHash sketching of DNA. Journal of open source software **1,** 27 (2016).
26. Developers, S. Sourmash Documentation Accessed: 2024-07-10 (2024). https://sourmash.readthedocs.io/en/latest/index.html.
27. Zhu, E. K. MinHash: A Probabilistic Data Structure for Estimating Similarity Accessed: 2024-07-10. 2024. https://ekzhu.com/datasketch/minhash.html.
28. Irber, L., Pierce-Ward, N. T. & Brown, C. T. Sourmash Branchwater Enables Lightweight Petabyte-Scale Sequence Search. bioRxiv. https://www.biorxiv.org/content/early/2022/11/03/2022.11.02.514947 (2022).

29. LearnDataSci. Jaccard Similarity: Definition and Explanation Accessed: 2024-07-10. 2024. https://www.learndatasci.com/glossary/jaccard-similarity/#:~:text=The%20Jaccard%20similarity%20measures%20the,of%20observations%20in%20either%20set..

30. Informatics, G. Mash Screen Accessed: 2024-07-10. 2024. https://genomeinformatics.github.io/mash-screen/.

31. Pierce, N., Irber, L., Reiter, T., Brooks, P. & Brown, C. Large-scale sequence comparisons with sourmash. F1000Research **8,** 1006 (**july** 2019).

32. Brown, C. T., Irber, L. & Crusoe, M. Sourmash: a library for comparing DNA and other sequences https://dib-lab.github.io/2020-paper-sourmash-gather/v/b2b79b86dc15762eb9e4567983b5c0b155809c89/. Accessed: 2024-08-09. 2020.

33. Van den Burg, M. P., Herrando-Pérez, S. & Vieites, D. R. ACDC, a global database of amphibian cytochrome-b sequences using reproducible curation for GenBank records. Scientific Data **7,** 268 (**august** 2020).

34. Parks, D. H., Imelfort, M., Skennerton, C. T., Hugenholtz, P. & Tyson, G. W. CheckM: assessing the quality of microbial genomes recovered from isolates, single cells, and metagenomes. Genome Research **25.** Epub 2015 May 14, 1043–1055. https://genome.cshlp.org/content/25/7/1043 (**july** 2015).

35. Parks, D. H. **andothers**. GTDB: Genome Taxonomy Database Accessed: 2024-08-05. 2024. https://gtdb.ecogenomic.org/about.

36. Sourmash Development Team. GTDB-R07-RS207 Genomic Representatives https://sourmash.readthedocs.io/en/latest/databases.html#gtdb-r07-rs207-genomic-representatives-66k. Accessed: 2024-08-08. 2024.

37. Sourmash Development Team. Sourmash Documentation https://sourmash.readthedocs.io/en/latest/index.html. Accessed: 2024-05-10.

38. Wood, D. E., Lu, J. & Langmead, B. Improved metagenomic analysis with Kraken 2. Genome Biology **20,** 257. ISSN: 1474-760X. https://doi.org/10.1186/s13059-019-1891-0 (2019).

39. Langmead, B. Kraken 2 AWS Indexes Accessed: 2024-08-22. 2024. https://benlangmead.github.io/aws-indexes/k2.

40. Lu, J., Breitwieser, F. P., Thielen, P. & Salzberg, S. L. Bracken: estimating species abundance in metagenomics data. PeerJ Computer Science **3,** e104 (2017).

41. Ondov, B. D., Bergman, N. H. & Phillippy, A. M. Interactive metagenomic visualization in a Web browser. BMC bioinformatics **12,** 1–10 (2011).

42. Ondov, B. D., Bergman, N. H. & Phillippy, A. M. Interactive metagenomic visualization in a Web browser. BMC bioinformatics **12,** 1–10 (2011).

43. Matloff, N. The art of R programming: A tour of statistical software design (No Starch Press, 2011).

44. Nieuwenhuijse, D., Oude Munnink, B. & Koopmans, M. viromeBrowser: A Shiny App for Browsing Virome Sequencing 2021.

45. Breitwieser, F. P. & Salzberg, S. L. Pavian: interactive analysis of metagenomics data for microbiome studies and pathogen identification. Bioinformatics **36,** 1303–1304. ISSN: 1367-4803. eprint: https://academic.oup.com/bioinformatics/article-pdf/36/4/1303/48981880/btz715.pdf. https://doi.org/10.1093/bioinformatics/btz715 (**september** 2019).

46. Wiki, W. BashGuide Accessed: 2024-08-23. 2024. https://mywiki.wooledge.org/BashGuide.

47. FreeCodeCamp. Bash Scripting Tutorial - Linux Shell Script and Command Line for Beginners Accessed: 2024-08-23. 2024. https://www.freecodecamp.org/news/bash-scripting-tutorial-linux-shell-script-and-command-line-for-beginners/.

48. contributors, W. Python (Programming Language) Accessed: 2024-08-23. 2024. https://en.wikipedia.org/wiki/Python_(programming_language).

49. GeeksforGeeks. Python sys Module - A Detailed Guide Accessed: 2024-08-23. 2024. https://geeksforgeeks.org/python-sys-module/.

50. Foundation, P. S. getopt — C-style parser for command line options Accessed: 2024-08-23. 2024. https://docs.python.org/3/library/getopt.html#module-getopt.

51. Flexiple. Top Python Libraries - A Comprehensive Guide Accessed: 2024-08-23. 2024. https://flexiple.com/python/python-libraries.

52. Foundation, P. S. os — Miscellaneous operating system interfaces Accessed: 2024-08-23. 2024. https://docs.python.org/3/library/os.html.

53. W3Schools. Python os Module Accessed: 2024-08-23. 2024. https://www.w3schools.com/python/module_os.asp?ref=escape.tech.

54. GeeksforGeeks. Command-Line Option and Argument Parsing using argparse in Python Accessed: 2024-08-23. 2024. https://www.geeksforgeeks.org/command-line-option-and-argument-parsing-using-argparse-in-python/.

55. Foundation, P. S. re — Regular expression operations Accessed: 2024-08-23. 2024. https://docs.python.org/3/library/re.html.

A Sample ID Mapping Across Illumina and Nanopore Sequencing Platforms by Season

Cow ID	Season	Illumina ID Forward	Illumina ID Reverse	ONT ID Run_1	ONT ID Run_2
0199	Feb	Feb0199_S8_L001_R1_001	Feb0199_S8_L001_R2_001	Run_1-Feb0199	**N/A**
	July	July0199_S1_L001_R1_001	July0199_S1_L001_R2_001	Run_1-July0199	**N/A**
0350	Feb	Feb0350_S9_L001_R1_001	Feb0350_S9_L001_R2_001	Run_1-Feb0350	Run_2-Feb0350
	July	July0350_S2_L001_R1_001	July0350_S2_L001_R2_001	Run_1-July0350	Run_2-July0350
0407	Feb	Feb0407_S10_L001_R1_001	Feb0407_S10_L001_R2_001	Run_1-Feb0407	Run_2-Feb0407
	July	July0407_S3_L001_R1_001	July0407_S3_L001_R2_001	Run_1-July0407	Run_2-July0407
0428	Feb	Feb0428_S11_L001_R1_001	Feb0428_S11_L001_R2_001	Run_1-Feb0428	Run_2-Feb0428
	July	July0428_S4_L001_R1_001	July0428_S4_L001_R2_001	Run_1-July0428	Run_2-July0428
0446	Feb	Feb0446_S12_L001_R1_001	Feb0446_S12_L001_R2_001	Run_1-Feb0446	Run_2-Feb0446
	July	July0446_S5_L001_R1_001	July0446_S5_L001_R2_001	Run_1-July0446	Run_2-July0446
0476	Feb	Feb0476_S13_L001_R1_001	Feb0476_S13_L001_R2_001	Run_1-Feb0476	Run_2-Feb0476
	July	July0476_S6_L001_R1_001	July0476_S6_L001_R2_001	Run_1-July0476	Run_2-July0476
0667	Feb	Feb0667_S14_L001_R1_001	Feb0667_S14_L001_R2_001	Run_1-Feb0667	**N/A**
	July	July0667_S7_L001_R1_001	July0667_S7_L001_R2_001	Run_1-July0667	Run_2-July0667

Table 24: Comparison of Sample Names Between Illumina and Nanopore Sequencing Platforms

B Summary of Read Mapping Results for the Samples from Both Nanopore and Illumina Reads

B.1 Nanopore

File name	Run_1-July0407				Run_2-July0407			
Genomes	I / T	T / I	I	T	I / T	T / I	I	T
Total reads	409.215				834.167			
Unmapped reads	408.573	408.573	408.811	408.585	832.775	832.775	833.245	832.807
Mapped reads	642	642	404	630	1.392	1.392	922	1.360

Table 25: Summary of read mapping results for July0407 (Nanopore)

File name	Run_1-July0428				Run_2-July0428			
Genome	I / T	T / I	I	T	I / T	T / I	I	T
Total reads	457.212				1.170.337			
Unmapped reads	408.914	408.914	409.176	408.940	994.381	994.381	995.039	994.464
Mapped reads	48.298	48.298	48.036	48.272	175.956	175.956	175.298	175.873

Table 26: Summary of read mapping results for July0428 (Nanopore)

We can observe a consistent pattern in the distribution of mapped reads and unmapped reads across the samples. Notably, when we align the reads against both reference genomes we observe no difference between these two cases. Another element that is pretty clear when we take a closer look at the Tables 25 and 26 is that when we align against both reference genomes : *Bos Taurus* and *Bos Indicus* and only against *Bos Taurus* we obtain similar values for the number of mapped and unmapped reads.

B.2 Illumina

File name	WD-3658-July0407_S3_L001_R1_001 WD-3658-July0407_S3_L001_R2_001			
Genome's order	I / T	T / I	I	T
Total reads	179.592.862			
Unmapped reads	179.494.134	179.494.134	179.504.138	179.494.406
Mapped reads	98.728	98.728	88.724	98.456

Table 27: Summary of read mapping results for July0407 (Illumina)

File name	WD-3658-July0428_S4_L001_R1_001 WD-3658-July0428_S4_L001_R2_001			
Genome's order	I / T	T / I	I	T
Total reads	207.955.844			
Unmapped reads	207.811.683	207.811.683	207.828.490	207.812.037
Mapped reads	144.161	144.161	127.354	143.807

Table 28: Summary of read mapping results for July0428 (Illumina)

A consistent pattern in the distribution of mapped and unmapped reads is also observed across Illumina samples. Aligning against both reference genomes (*Bos Taurus* and *Bos Indicus*) yields similar results to aligning only against *Bos Taurus*, with no significant differences noted between the cases.

C Read Counts Across Different Steps of the Analysis for the Nanopore Dataset

Sample name	Nbr of total reads	Nbr of trimmed reads	Nbr of unmapped reads
Run_1-Feb0199	2.036.152	1.824.548	1.816.220
Run_1-Feb0350	1.864.980	1.644.886	1.637.608
Run_1-Feb0407	1.971.160	1.762.921	1.753.414
Run_1-Feb0428	2.598.263	2.321.173	2.274.747
Run_1-Feb0446	1.273.906	1.138.985	1.134.751
Run_1-Feb0476	676.316	613.538	610.016
Run_1-Feb0667	756.381	679.451	673.538
Run_1-July0199	1.412.131	1.264.475	1.262.914
Run_1-July0350	1.865.333	1.667.866	1.655.942
Run_1-July0407	409.214	372.247	371.694
Run_1-July0428	457.212	410.673	408.940
Run_1-July0446	1.104.809	985.545	984.047
Run_1-July0476	1.430.544	1.277.192	1.275.092
Run_1-July0667	1.122.004	1.007.587	1.003.463
Run_2-Feb0350	2.429.145	2.121.775	2.107.304
Run_2-Feb0407	828.680	719.404	714.616
Run_2-Feb0428	1.270.335	1.090.711	1.066.953
Run_2-Feb0446	1.564.784	1.357.207	1.351.891
Run_2-Feb0476	1.565.013	1.363.324	1.354.890
Run_2-July0350	1.482.434	1.311.544	1.300.558
Run_2-July0407	834.166	728.124	727.006
Run_2-July0428	1.170.337	999.470	994.464
Run_2-July0446	1.456.434	1.303.932	1.301.909
Run_2-July0476	968.201	865.097	863.554
Run_2-July0667	1.472.822	1.327.773	1.321.611

Table 29: Read Counts Across Different Step of the Analysis for Each Sample

D Read Counts Before and After Removing *Bos Taurus* from the Illumina Dataset

Sample name	Nbr of total reads	Nbr of unmapped reads
Feb0199	155.874.549	155.567.272
Feb0350	271.168.707	270.577.247
Feb0407	197.061.737	196.772.406
Feb0428	156.576.498	155.635.097
Feb0446	160.152.629	159.957.898
Feb0476	193.271.979	192.818.297
Feb0667	154.387.378	153.974.267
July0199	299.209.944	299.176.159
July0350	220.661.372	219.997.033
July0407	179.592.862	179.494.406
July0428	207.955.844	207.812.037
July0446	308.106.326	308.038.278
July0476	240.631.994	240.573.529
July0667	193.933.861	193.699.555

Table 30: Read Counts Before and After Removing *Bos Taurus* from the Illumina Dataset

E Read Counts Before and After the Trimming Step of the Illumina Dataset

Sample Name	Nbr of unmapped reads	Nbr of trimmed reads
Feb0199_R1	155.567.272	127.330.082
Feb0199_R2		131.057.767
Feb0350_R1	270.577.247	218.031.693
Feb0350_R2		226.494.283
Feb0407_R1	196.772.406	161.807.959
Feb0407_R2		166.450.211
Feb0428_R1	155.635.097	128.465.895
Feb0428_R2		131.615.840
Feb0446_R1	159.957.898	130.762.733
Feb0446_R2		134.779.753
Feb0476_R1	192.818.297	161.385.330
Feb0476_R2		165.625.189
Feb0667_R1	153.974.267	126.752.510
Feb0667_R2		129.566.069
July0199_R1	299.176.159	239.278.299
July0199_R2		245.115.392
July0350_R1	219.997.033	180.676.691
July0350_R2		185.558.001
July0407_R1	179.494.406	145.690.071
July0407_R2		151.015.213
July0428_R1	207.812.037	169.584.268
July0428_R2		173.957.758
July0446_R1	308.038.278	247.639.947
July0446_R2		253.392.623
July0476_R1	240.573.529	194.110.385
July0476_R2		202.210.492
July0667_R1	193.699.555	157.160.613
July0667_R2		161.545.476

Table 31: Read Counts Before and After the Trimming Step of the Illumina Dataset

F Data Visualisation

F.1 Nanopore

F.1.1 Krona on sourmash

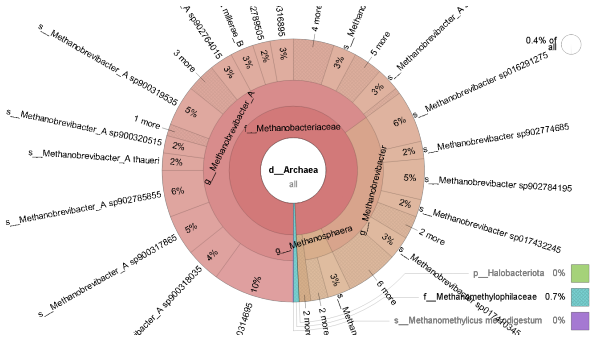


Fig. 48: Krona report Run_1-Feb0350 (sourmash)

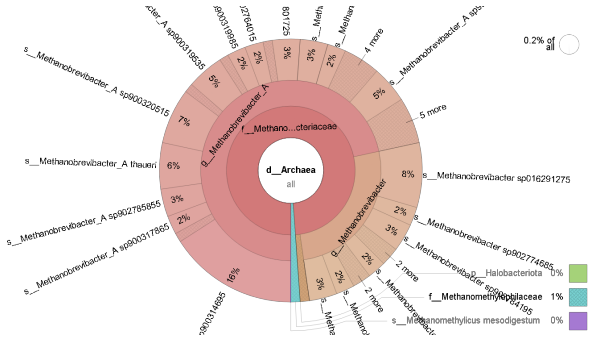


Fig. 49: Krona report Run_2-Feb0350 (sourmash)

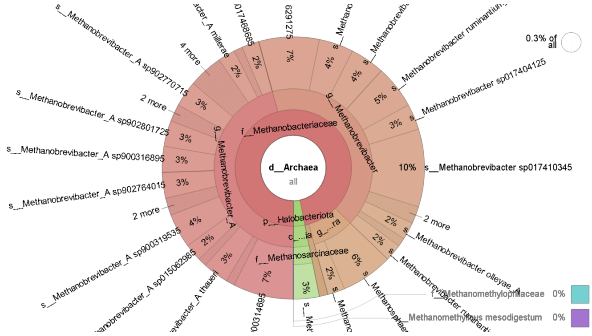


Fig. 50: Krona report Run_1-Feb0407 (sourmash)

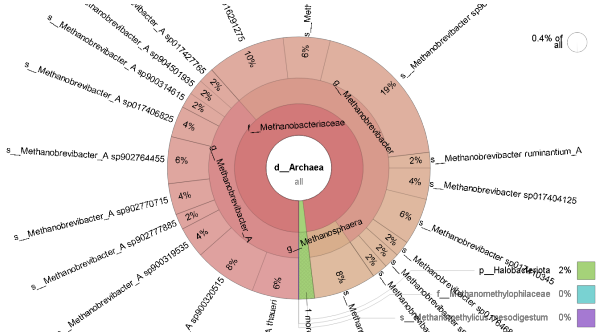


Fig. 51: Krona report Run_2-Feb0407 (sourmash)

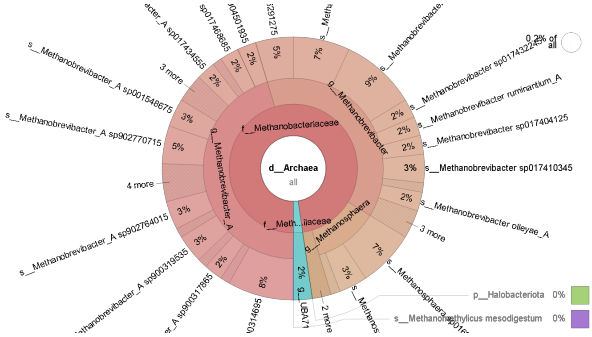


Fig. 52: Krona report Run_1-Feb0428 (sourmash)

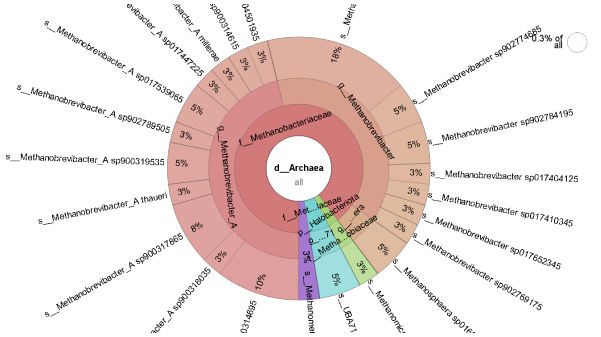


Fig. 53: Krona report Run_2-Feb0428 (sourmash)

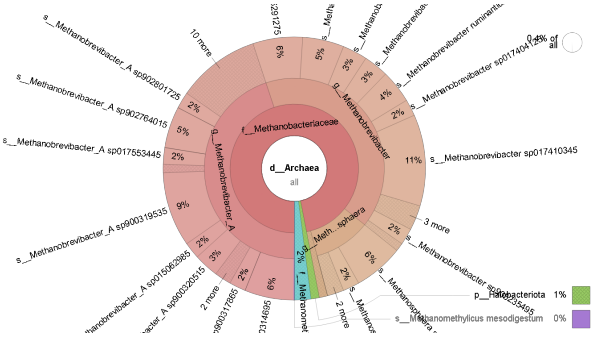


Fig. 54: Krona report Run_1-Feb0446 (sourmash)

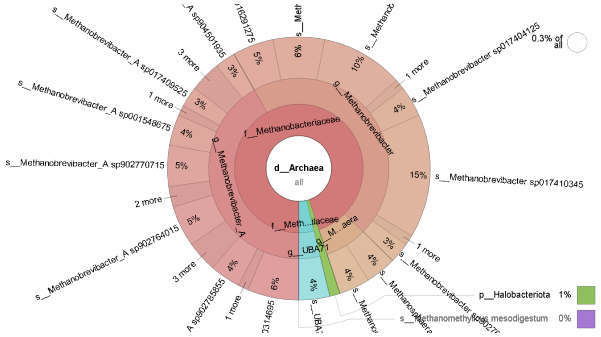


Fig. 55: Krona report Run_2-Feb0446 (sourmash)

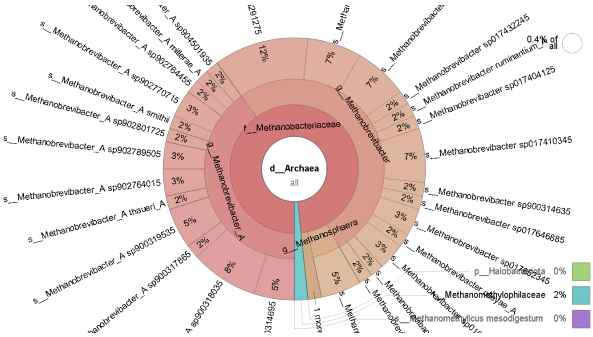


Fig. 56: Krona report Run_1-Feb0476 (sourmash)

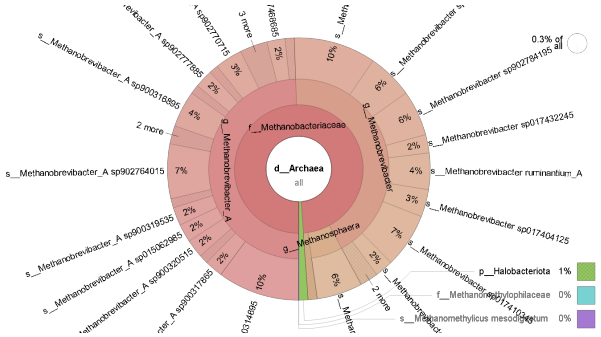


Fig. 57: Krona report Run_2-Feb0476 (sourmash)

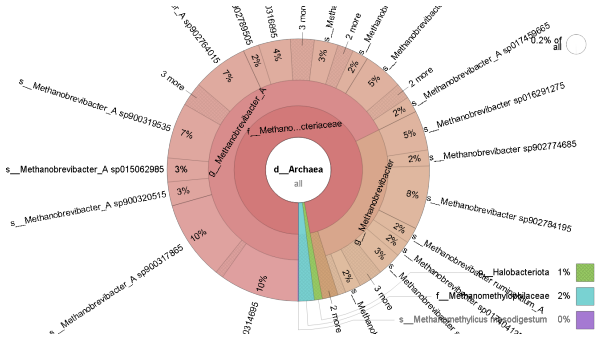


Fig. 58: Krona report Run_1-July0350 (sourmash)

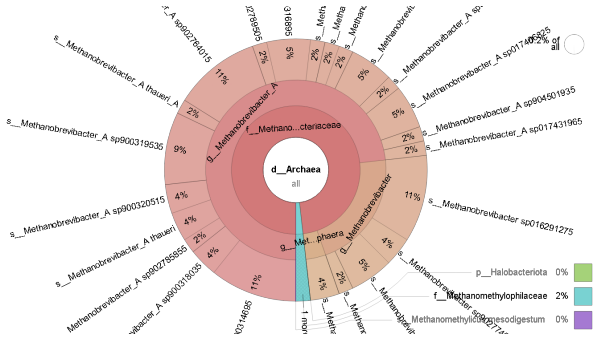


Fig. 59: Krona report Run_2-July0350 (sourmash)

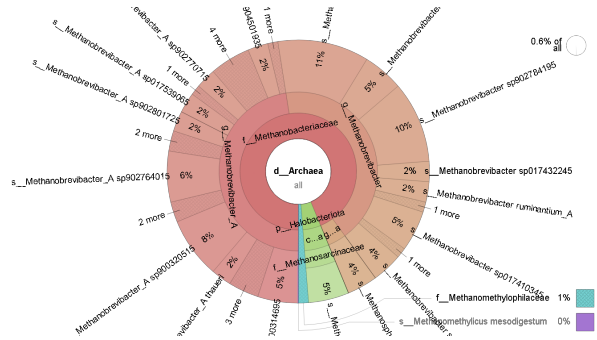


Fig. 60: Krona report Run_1-July0407 (sourmash)

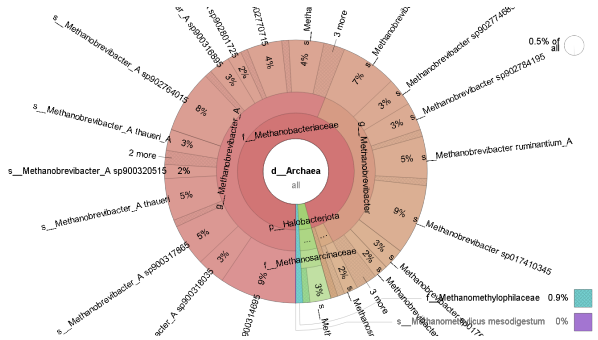


Fig. 61: Krona report Run_2-July0407 (sourmash)

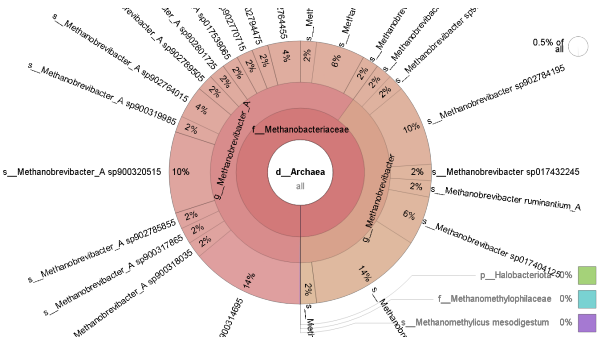


Fig. 62: Krona report Run_1-July0428 (sourmash)

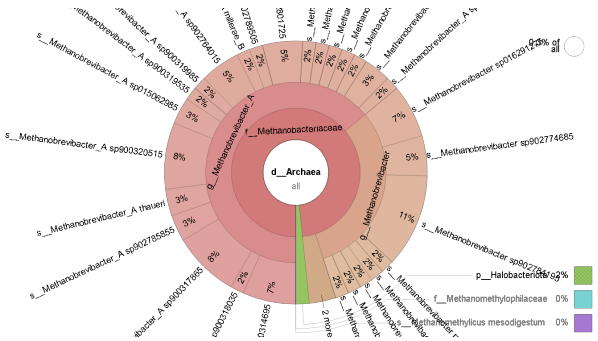


Fig. 63: Krona report Run_2-July0428 (sourmash)

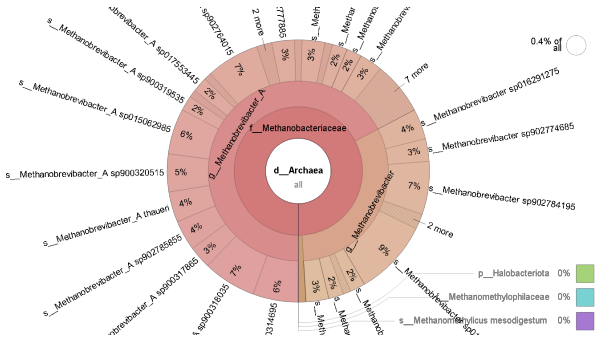


Fig. 64: Krona report Run_1-July0446 (sourmash)

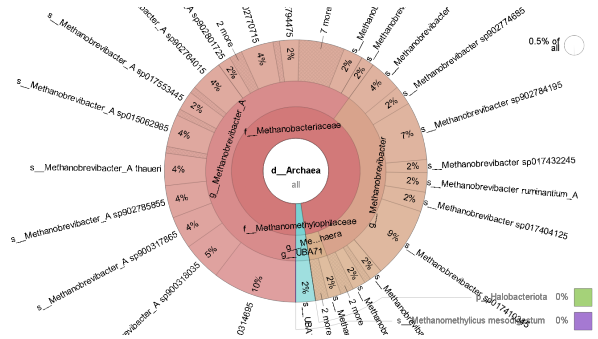


Fig. 65: Krona report Run_2-July0446 (sourmash)

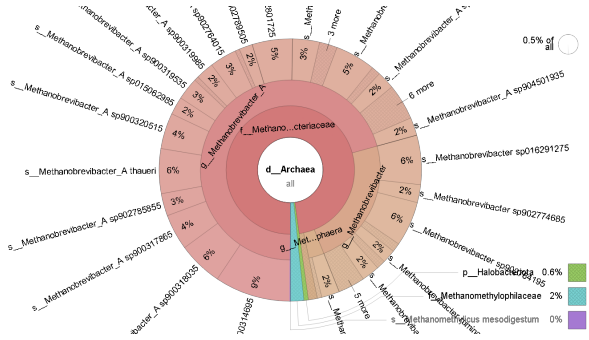


Fig. 66: Krona report Run_1-July0476 (sourmash)

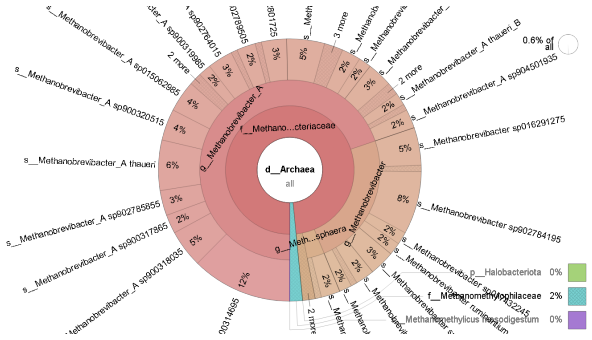


Fig. 67: Krona report Run_2-July0476 (sourmash)

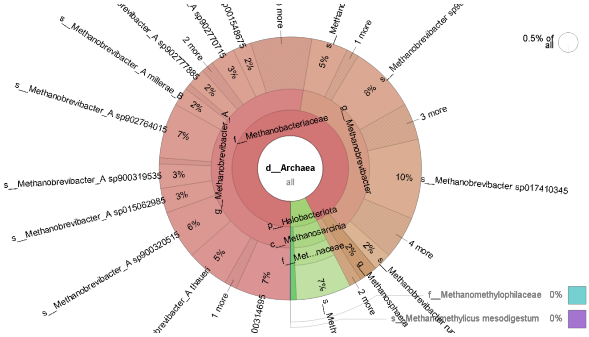


Fig. 68: Krona report Run_1-July0667 (sourmash)

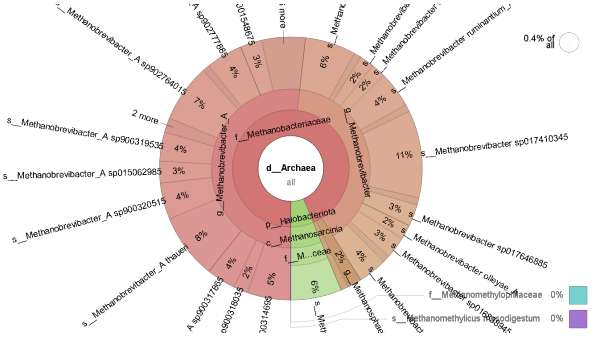


Fig. 69: Krona report Run_2-July0667 (sourmash)

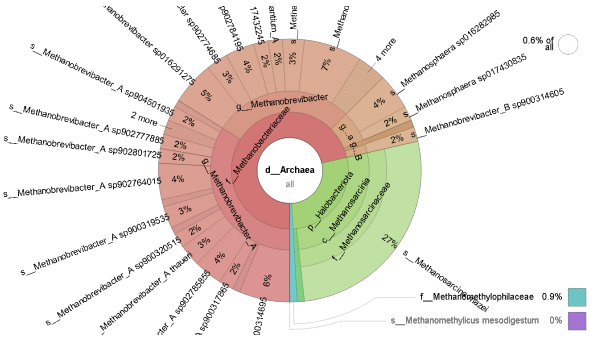


Fig. 70: Krona report Run_1-Feb0667 (sourmash)

F.1.2 Krona on Kraken2/Bracken

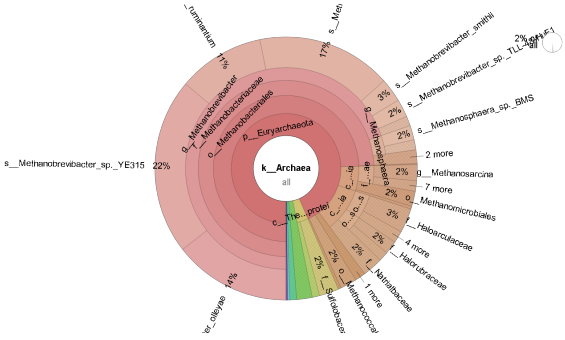


Fig. 71: Krona report Run_1-Feb0350 (K2B)

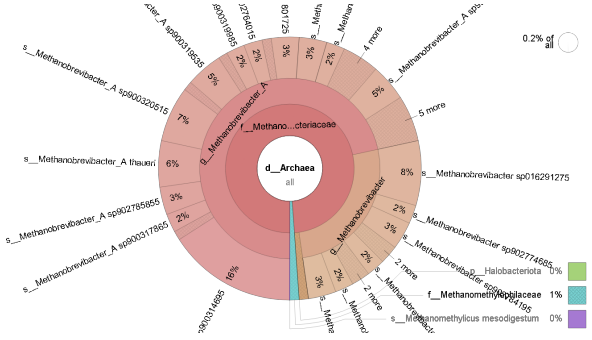


Fig. 72: Krona report Run_2-Feb0350 (K2B)

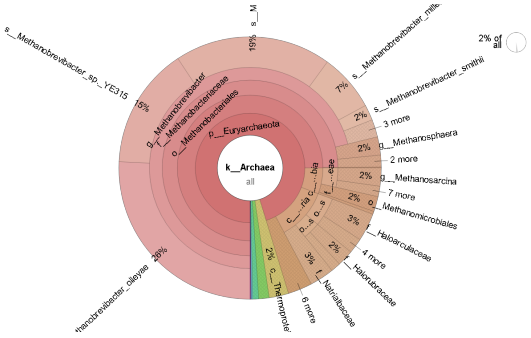


Fig. 73: Krona report Run_1-Feb0407 (K2B)

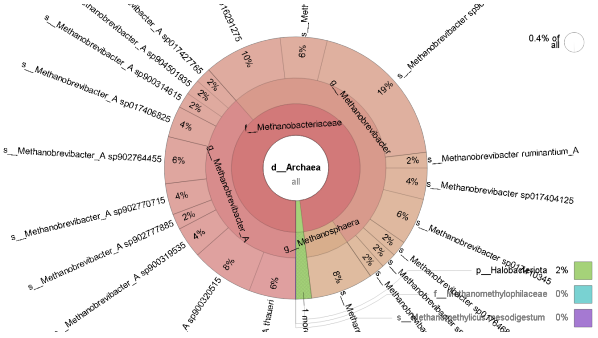


Fig. 74: Krona report Run_2-Feb0407 (K2B)

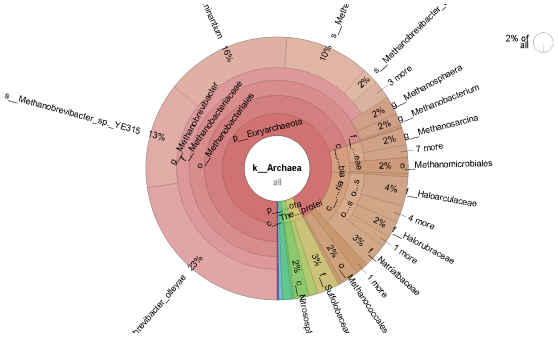


Fig. 75: Krona report Run_1-Feb0428 (K2B)

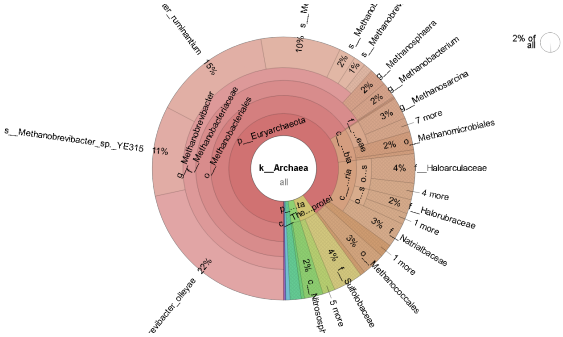


Fig. 76: Krona report Run_2-Feb0428 (K2B)

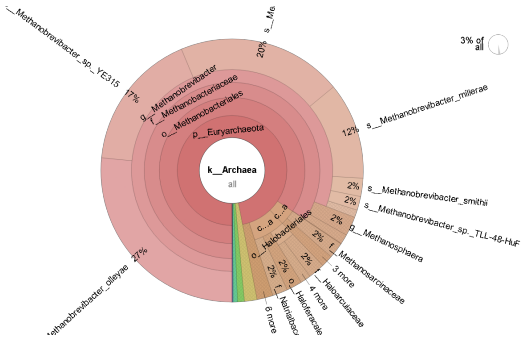


Fig. 77: Krona report Run_1-Feb0446 (K2B)

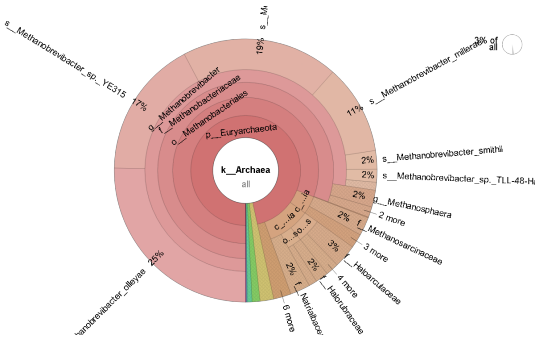


Fig. 78: Krona report Run_2-Feb0446 (K2B)

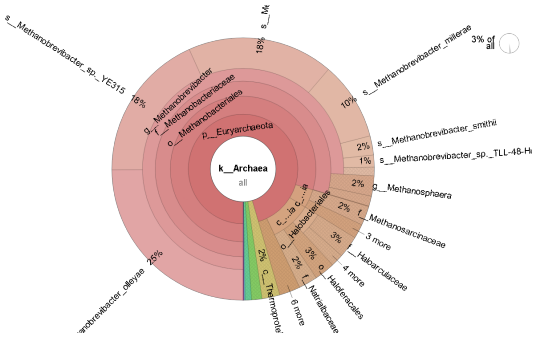


Fig. 79: Krona report Run_1-Feb0476 (K2B)

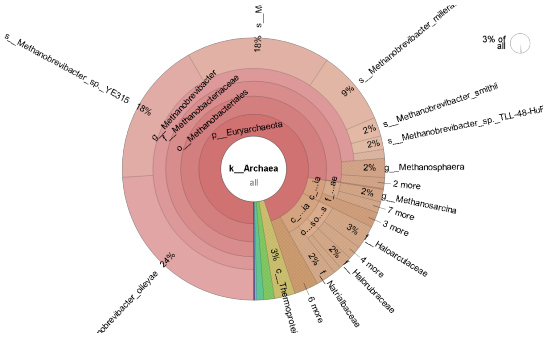


Fig. 80: Krona report Run_2-Feb0476 (K2B)

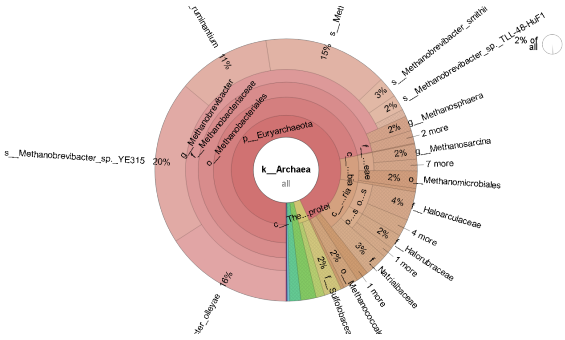


Fig. 81: Krona report Run_1-July0350 (K2B)

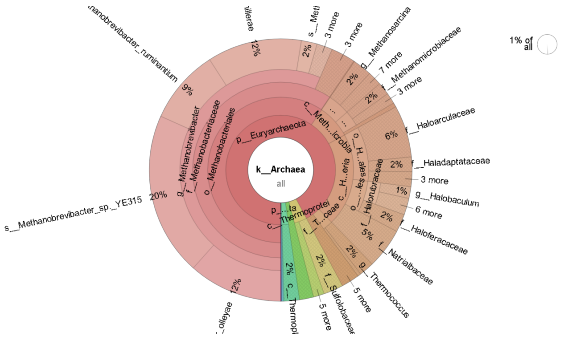


Fig. 82: Krona report Run_2-Feb0476 (K2B)

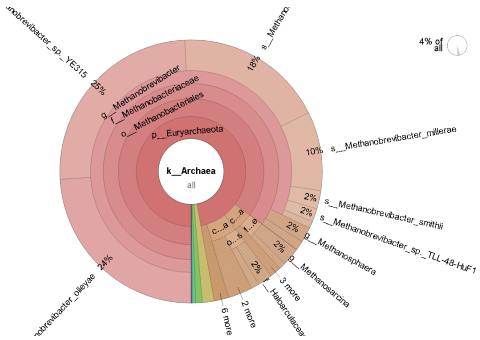


Fig. 83: Krona report Run_1-July0407 (K2B)

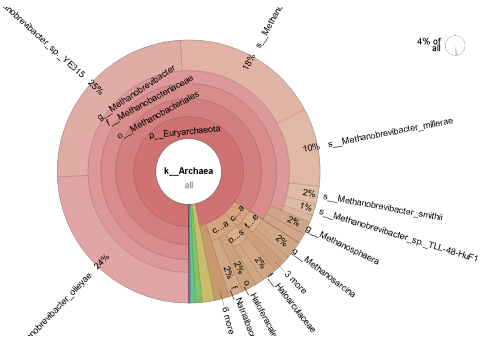


Fig. 84: Krona report Run_2-Feb0407 (K2B)

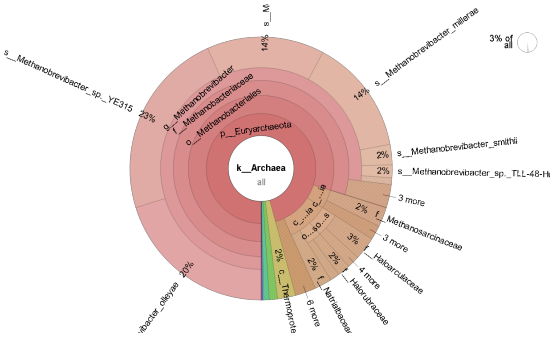


Fig. 85: Krona report Run_1-July0428 (K2B)

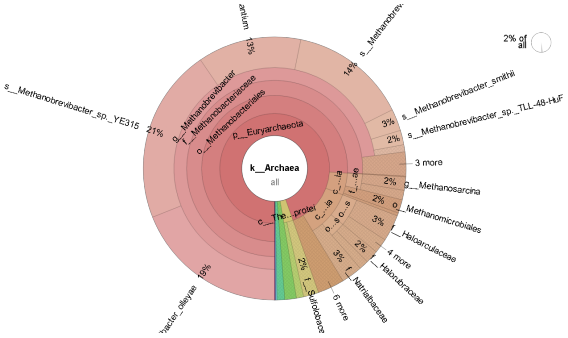


Fig. 86: Krona report Run_2-Feb0428 (K2B)

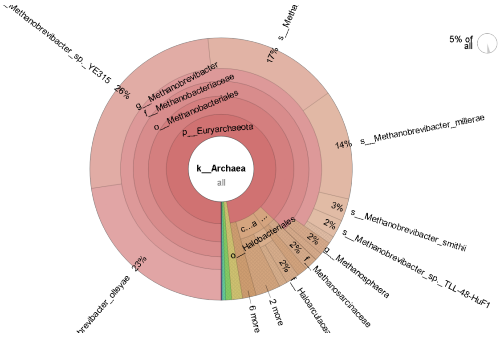


Fig. 87: Krona report Run_1-July0446 (K2B)

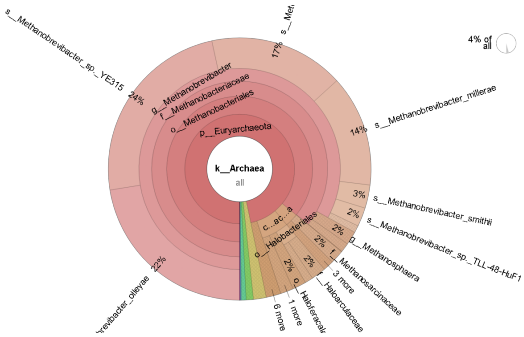


Fig. 88: Krona report Run_2-Feb0446 (K2B)

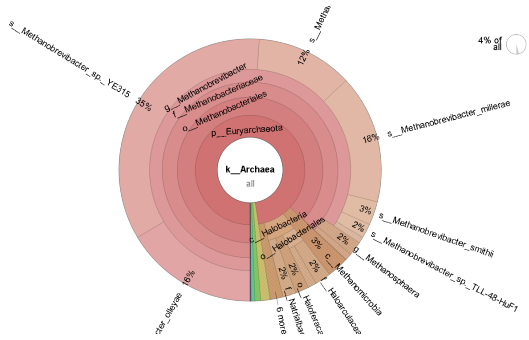


Fig. 89: Krona report Run_1-July0476 (K2B)

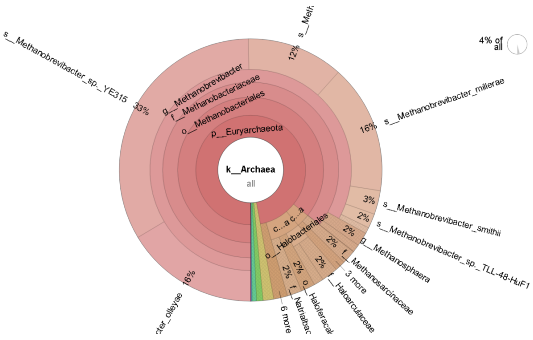


Fig. 90: Krona report Run_2-July0476 (K2B)

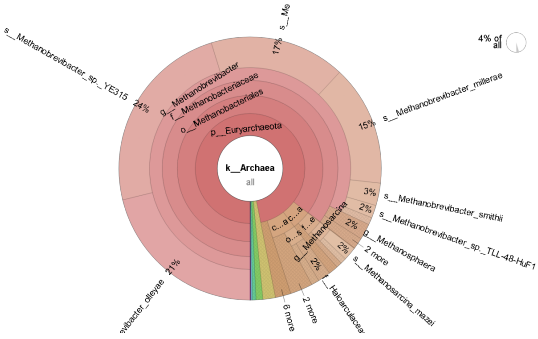


Fig. 91: Krona report Run_1-July0667 (K2B)

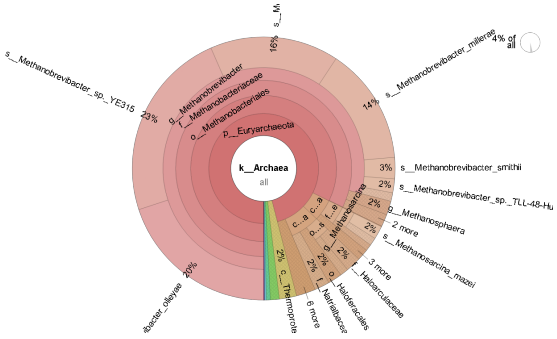


Fig. 92: Krona report Run_2-July0667 (K2B)

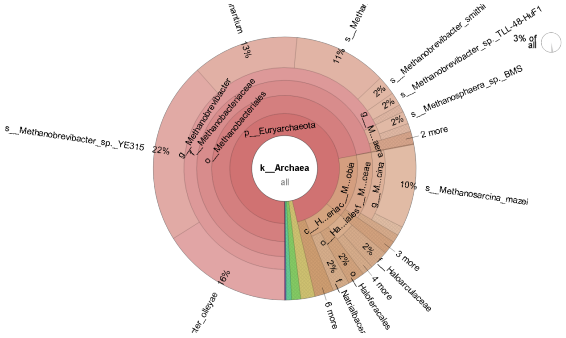


Fig. 93: Krona report Run_1-Feb0667 (K2B)

F.1.3 Pavian on Kraken2/Bracken

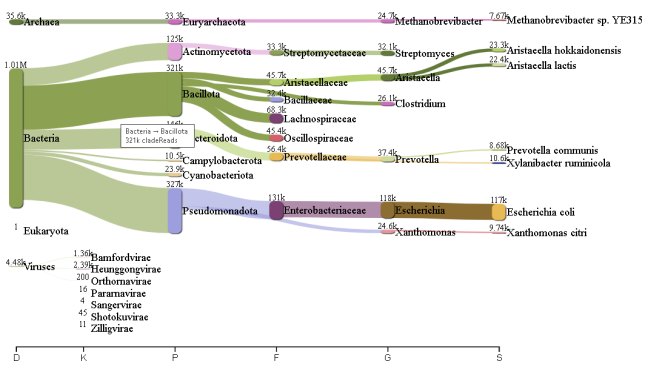


Fig. 94: Pavian report Run_1-Feb0350 (K2B)

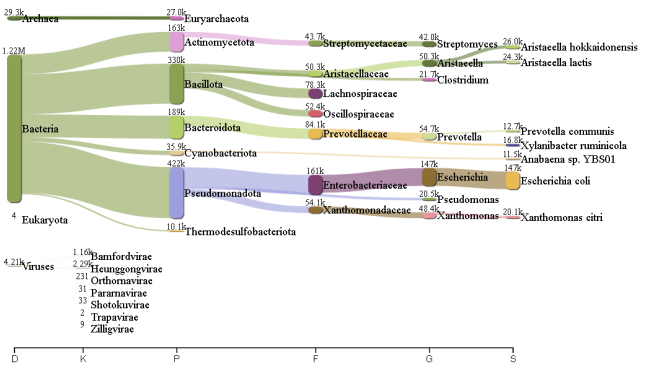


Fig. 95: Pavian report Run_2-Feb0350 (K2B)

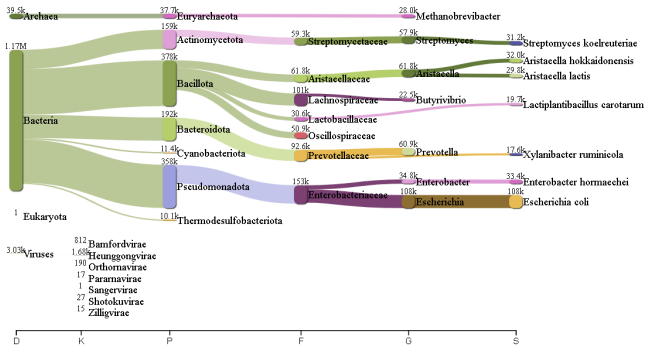


Fig. 96: Pavian report Run_1-Feb0407 (K2B)

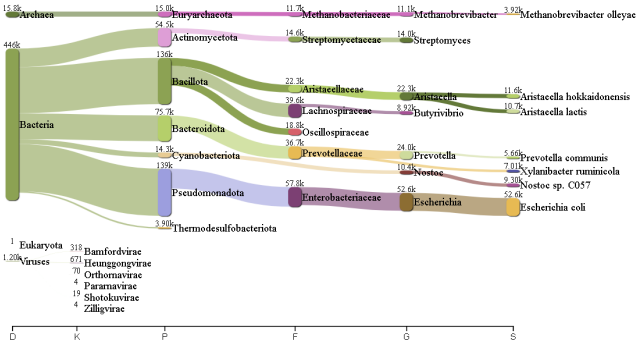


Fig. 97: Pavian report Run_2-Feb0407 (K2B)

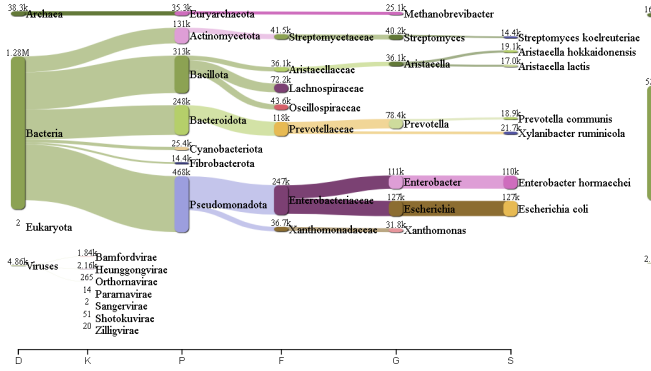


Fig. 98: Pavian report Run_1-Feb0428 (K2B)

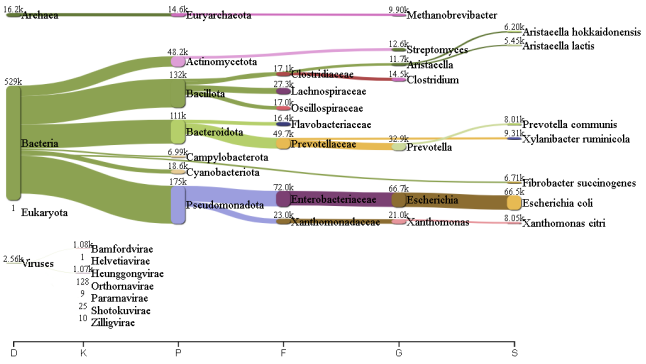


Fig. 99: Pavian report Run_2-Feb0428 (K2B)

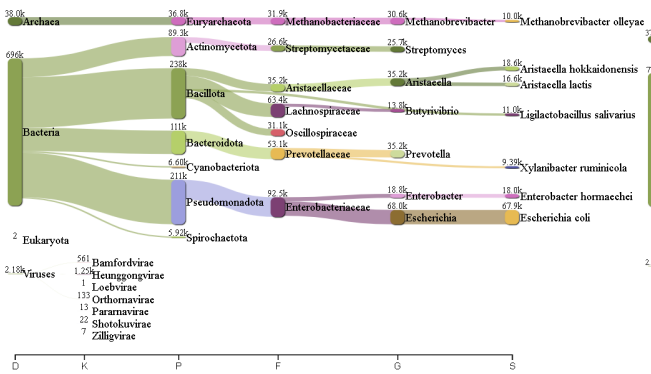


Fig. 100: Pavian report Run_1-Feb0446 (K2B)

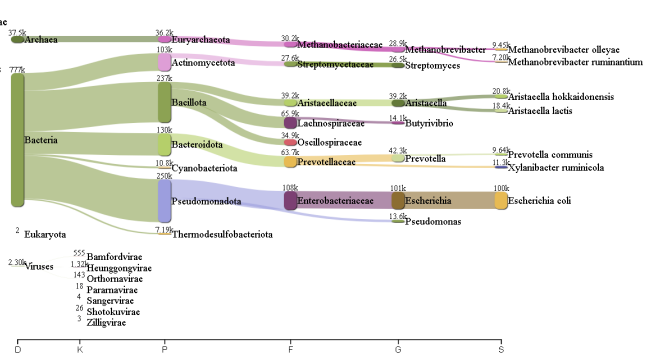


Fig. 101: Pavian report Run_2-Feb0446 (K2B)

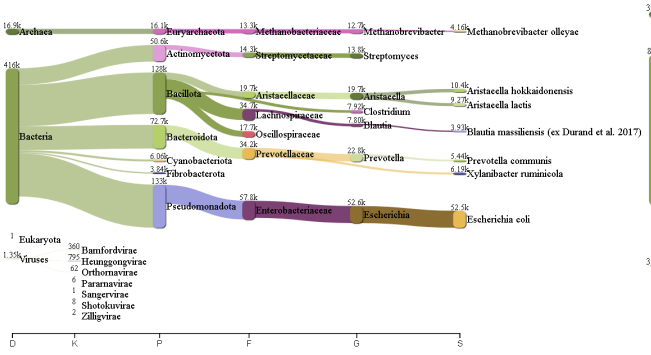


Fig. 102: Pavian report Run_1-Feb0476 (K2B)

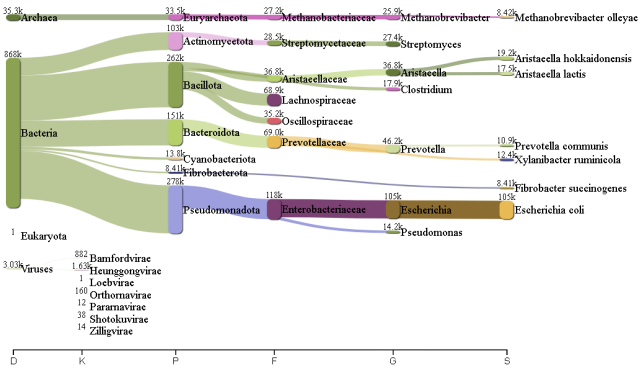


Fig. 103: Pavian report Run_2-Feb0476 (K2B)

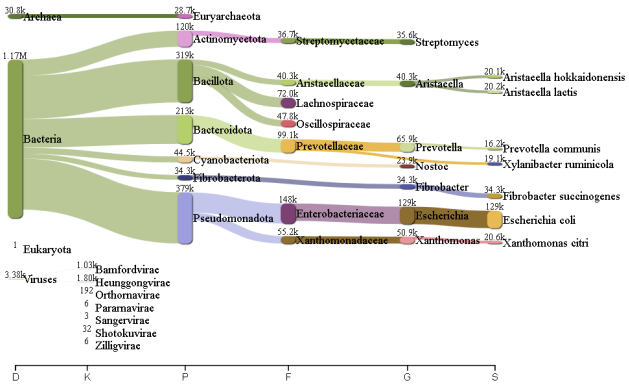


Fig. 104: Pavian report Run_1-July0350 (K2B)

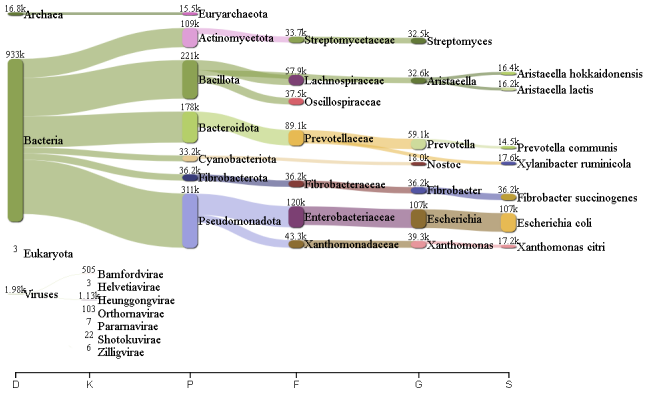


Fig. 105: Pavian report Run_2-July0350 (K2B)

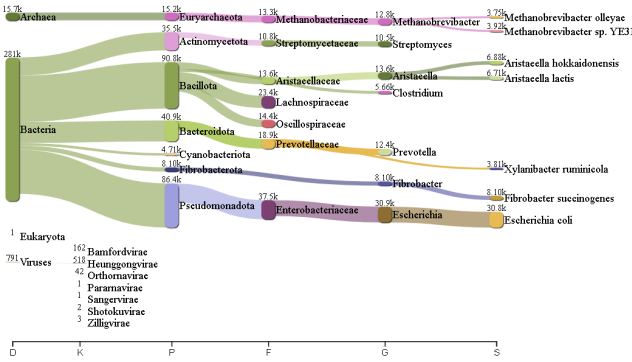


Fig. 106: Pavian report Run_1-July0407 (K2B)

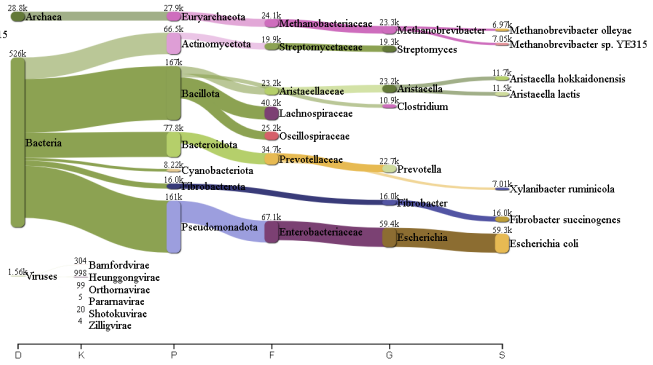


Fig. 107: Pavian report Run_2-July0407 (K2B)

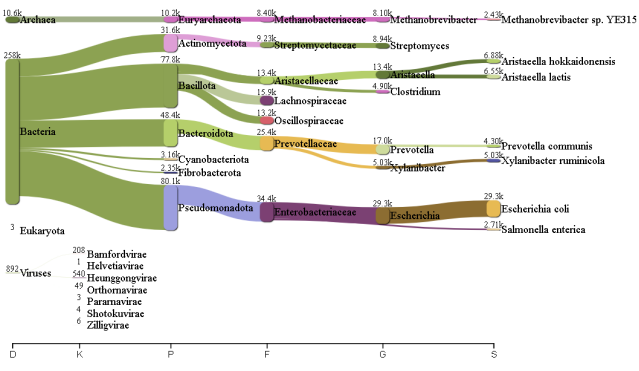


Fig. 108: Pavian report Run_1-July0428 (K2B)

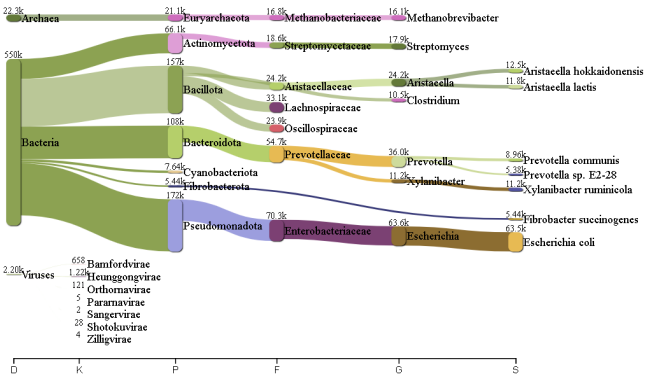


Fig. 109: Pavian report Run_2-July0428 (K2B)

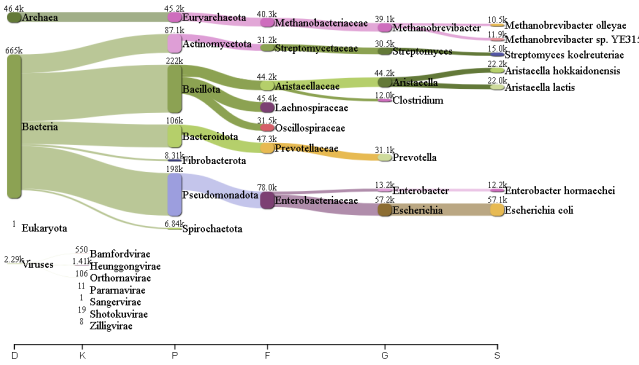


Fig. 110: Pavian report Run_1-July0446 (K2B)

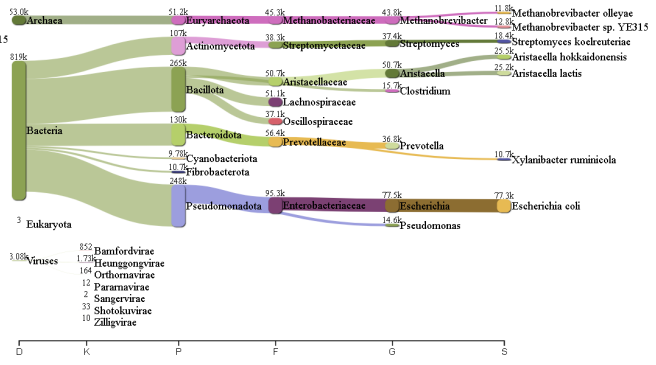


Fig. 111: Pavian report Run_2-July0446 (K2B)

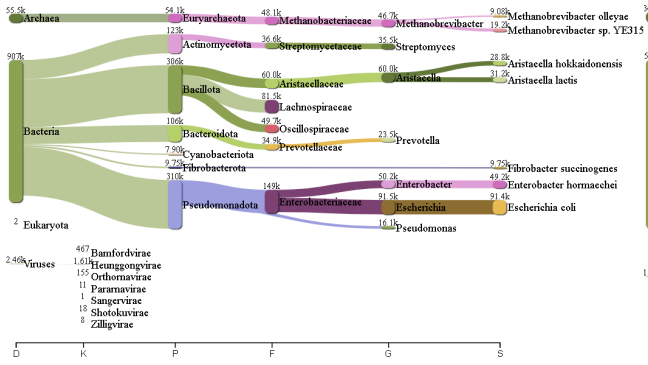


Fig. 112: Pavian report Run_1-July0476 (K2B)

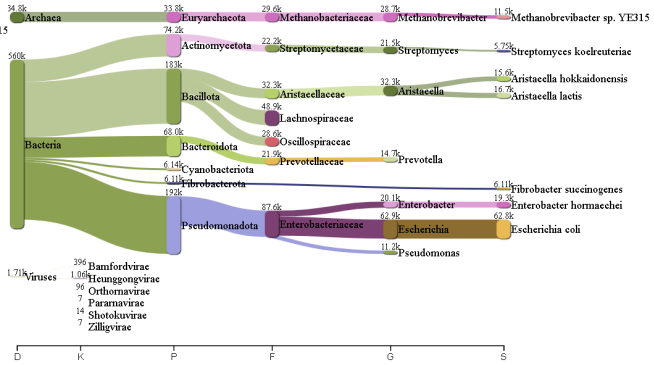


Fig. 113: Pavian report Run_2-July0476 (K2B)

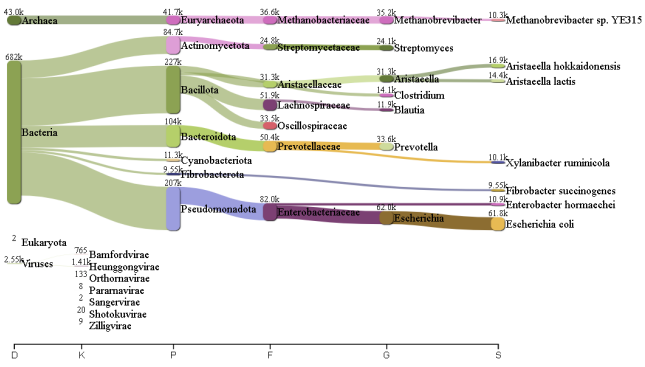


Fig. 114: Pavian report Run_1-July0667 (K2B)

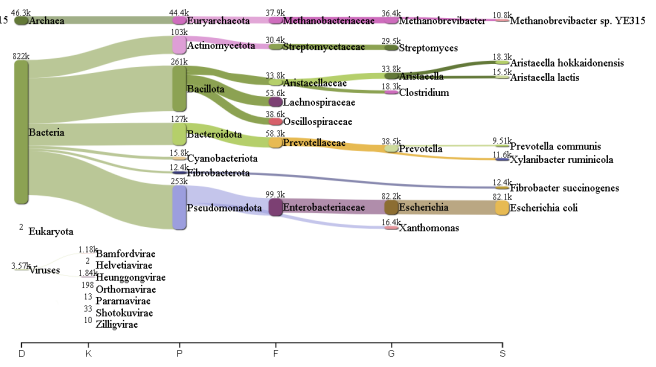


Fig. 115: Pavian report Run_2-July0667 (K2B)

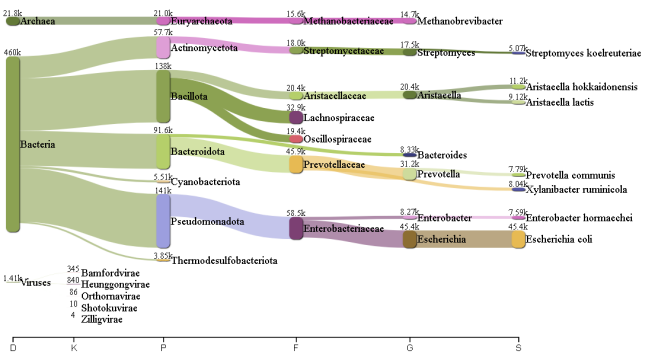


Fig. 116: Pavian report Run_1-Feb0667 (K2B)

F.2 Illumina

F.2.1 Krona on sourmash

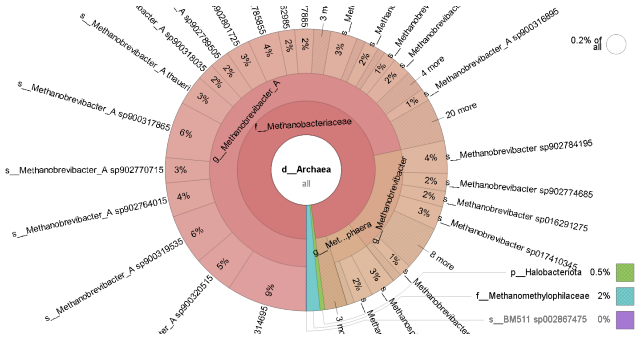


Fig. 117: Krona report Feb0350 Illumina (sourmash)

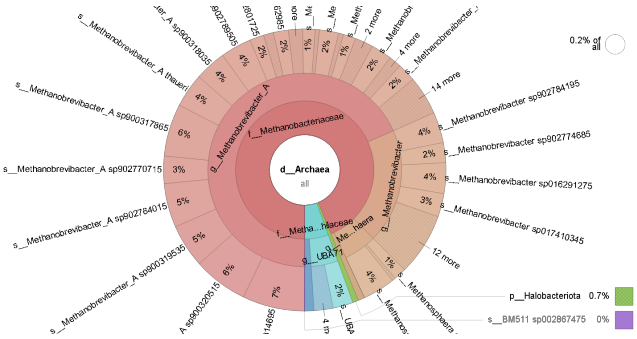


Fig. 118: Krona report July0350 Illumina (sourmash)

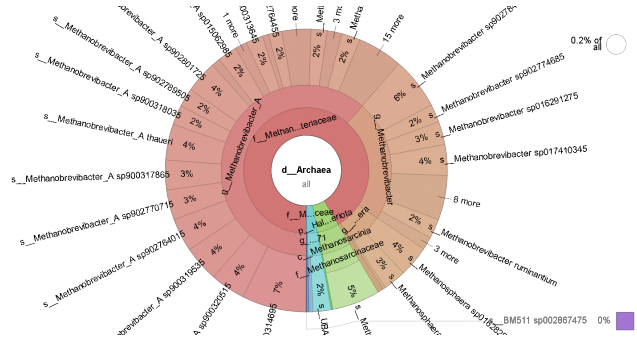


Fig. 119: Krona report Feb0407 Illumina (sourmash)

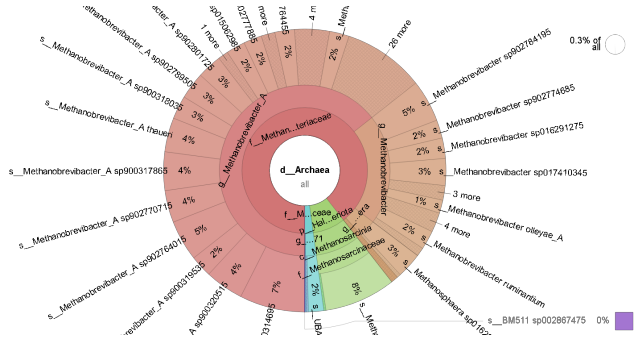


Fig. 120: Krona report July0407 Illumina (sourmash)

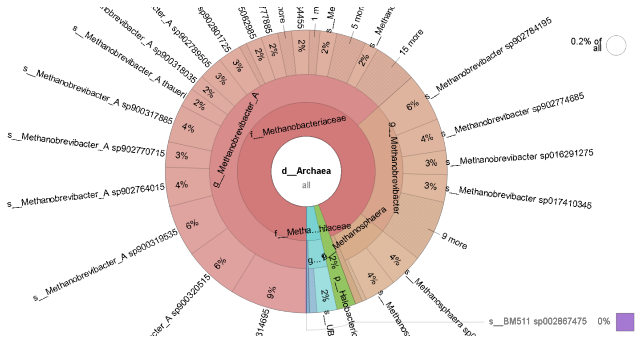


Fig. 121: Krona report Feb0428 Illumina (sourmash)

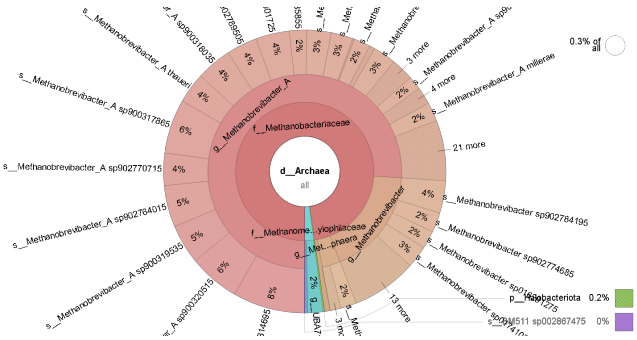


Fig. 122: Krona report July0428 Illumina (sourmash)

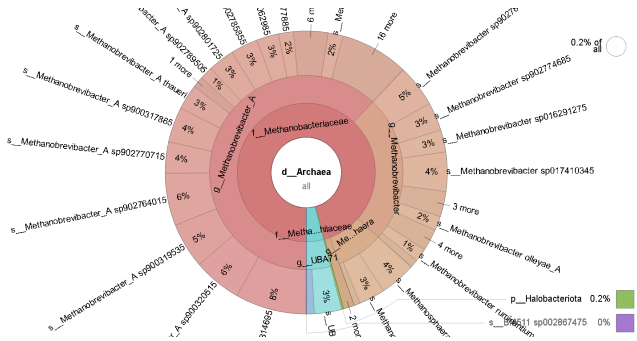


Fig. 123: Krona report Feb0446 Illumina (sourmash)

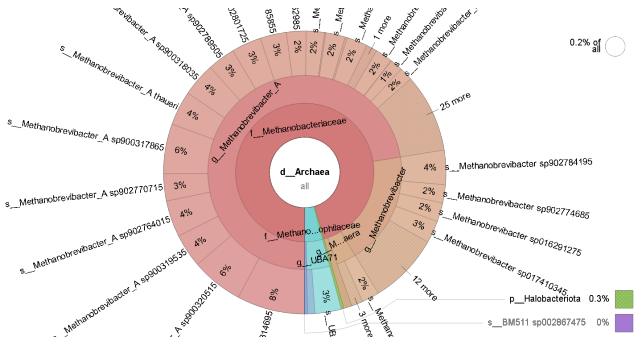


Fig. 124: Krona report July0446 Illumina (sourmash)

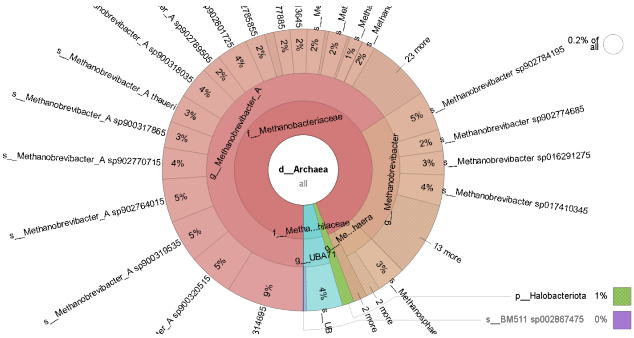


Fig. 125: Krona report Feb0476 Illumina (sourmash)

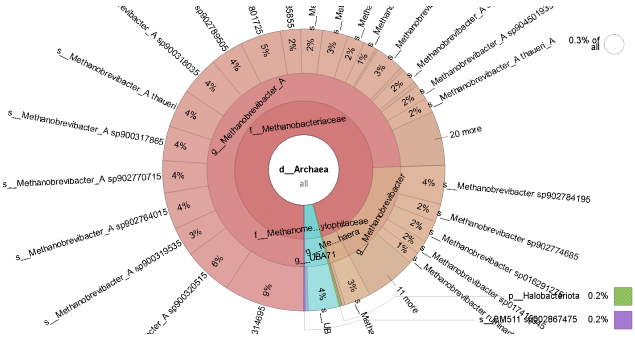


Fig. 126: Krona report July0476 Illumina (sourmash)

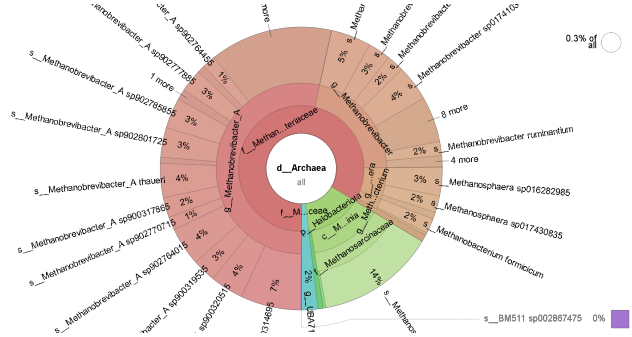


Fig. 127: Krona report Feb0667 Illumina (sourmash)

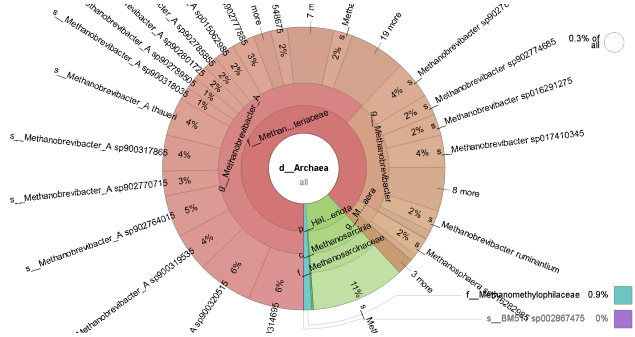


Fig. 128: Krona report July0667 Illumina (sourmash)

F.2.2 Krona on Kraken2/Bracken

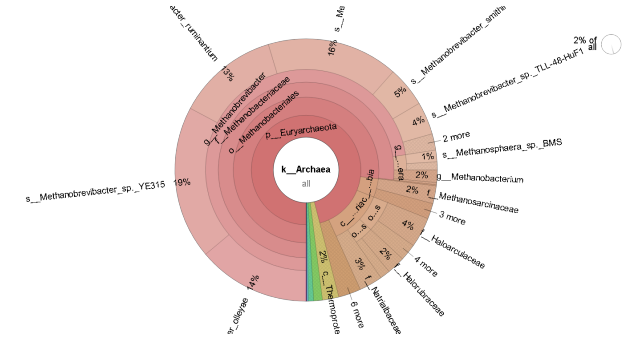


Fig. 129: Krona report Feb0350 Illumina (K2B)

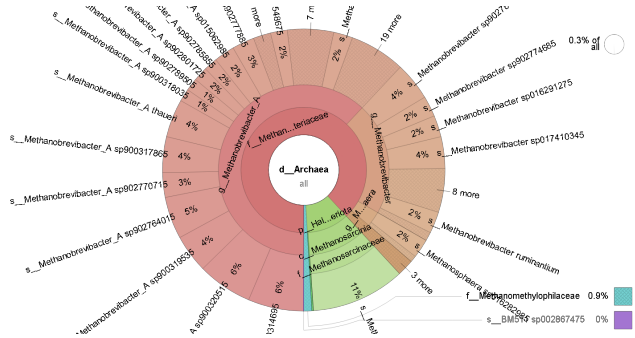


Fig. 130: Krona report July0350 Illumina (K2B)

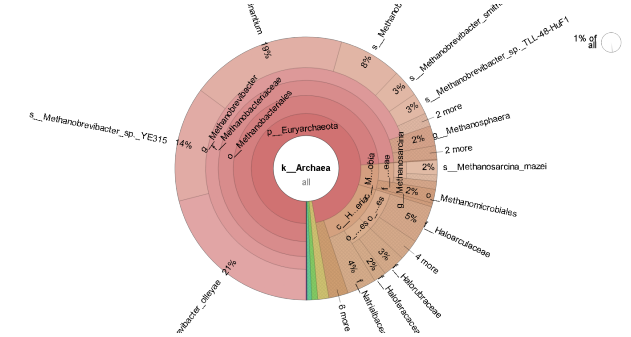


Fig. 131: Krona report Feb0407 Illumina (K2B)

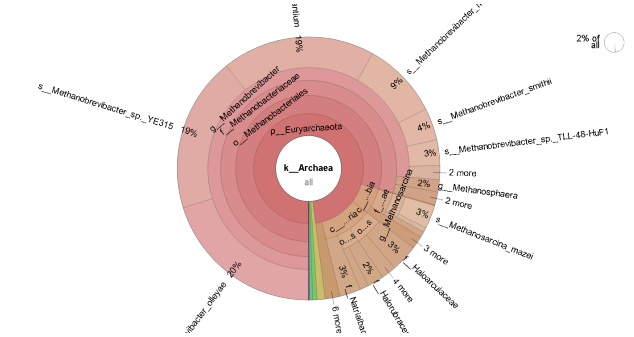


Fig. 132: Krona report July0407 Illumina (K2B)

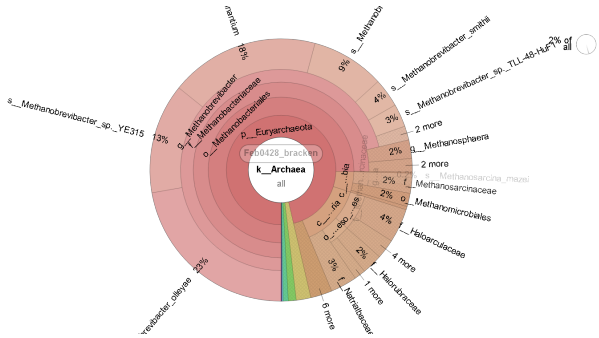


Fig. 133: Krona report Feb0428 Illumina (K2B)

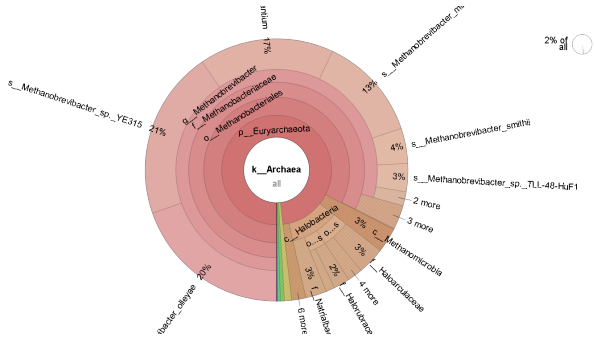


Fig. 134: Krona report July0428 Illumina (K2B)

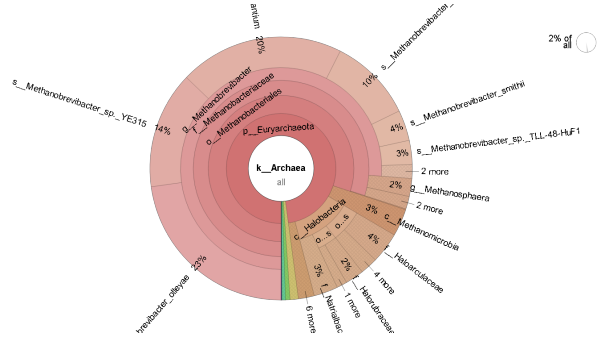


Fig. 135: Krona report Feb0446 Illumina (K2B)

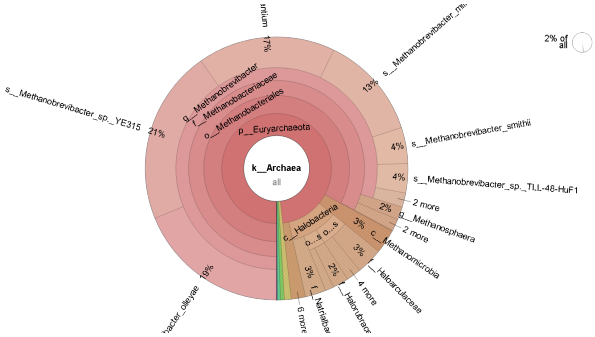


Fig. 136: Krona report July0446 Illumina (K2B)

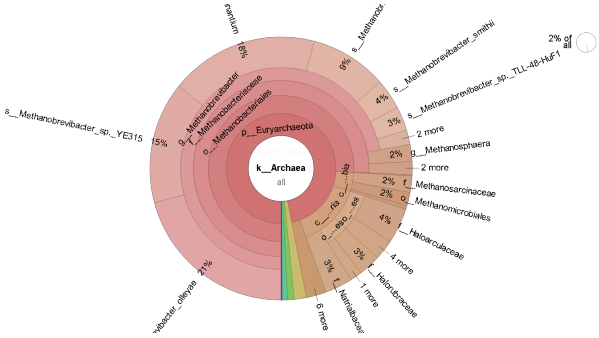


Fig. 137: Krona report Feb0476 Illumina (K2B)

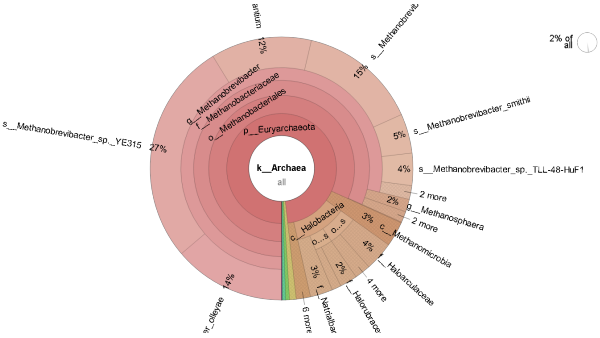


Fig. 138: Krona report July0476 Illumina (K2B)

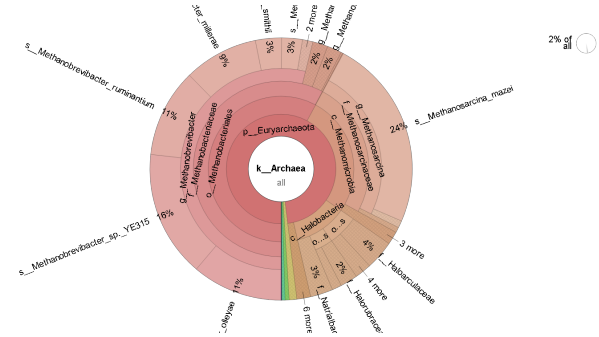


Fig. 139: Krona report Feb0667 Illumina (K2B)

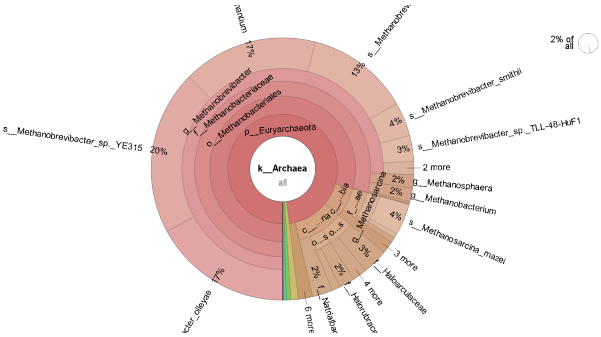


Fig. 140: Krona report July0667 Illumina (K2B)

F.2.3 Pavian on Kraken2/Bracken

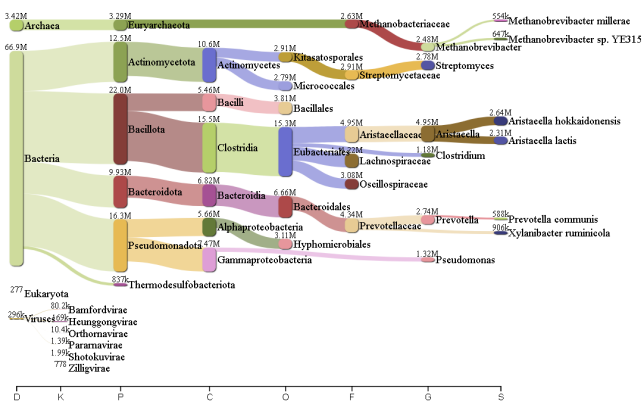


Fig. 141: Pavian report Feb0350 Illumina (K2B)

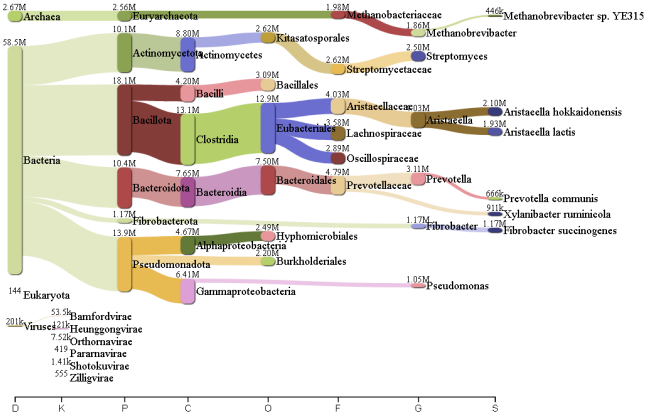


Fig. 142: Pavian report July0350 Illumina (K2B)

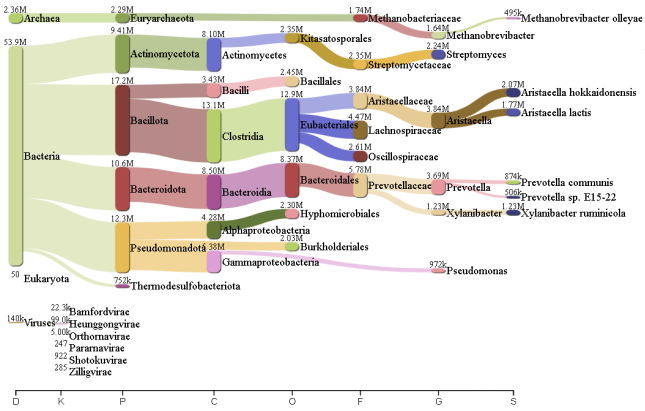


Fig. 143: Pavian report Feb0407 Illumina (K2B)

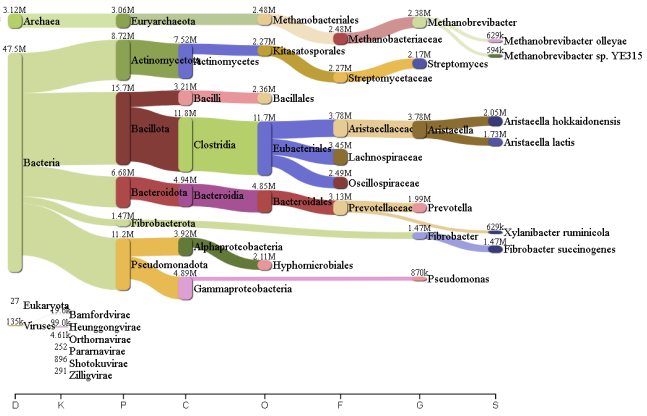


Fig. 144: Pavian report July0407 Illumina (K2B)

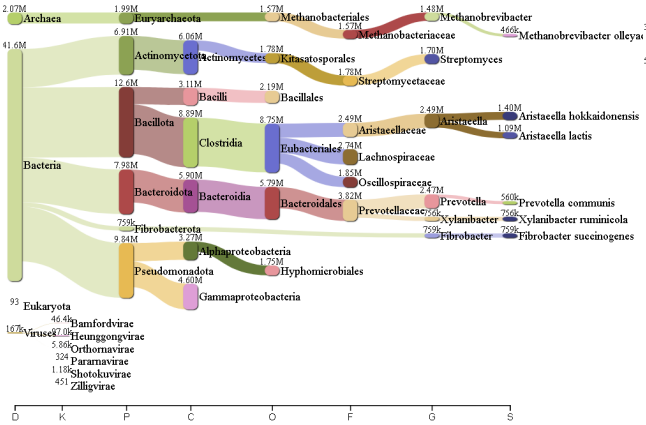


Fig. 145: Pavian report Feb0428 Illumina (K2B)

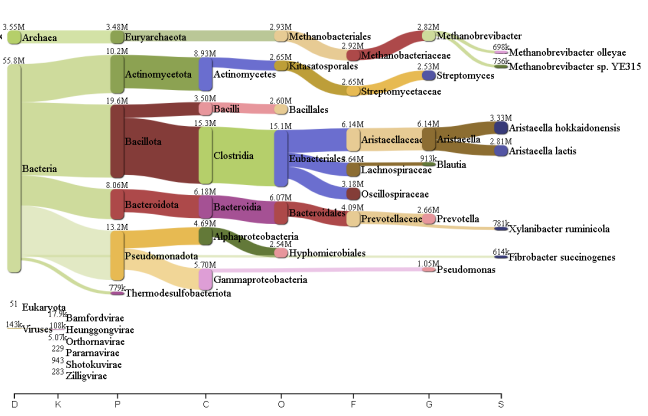


Fig. 146: Pavian report July0428 Illumina (K2B)

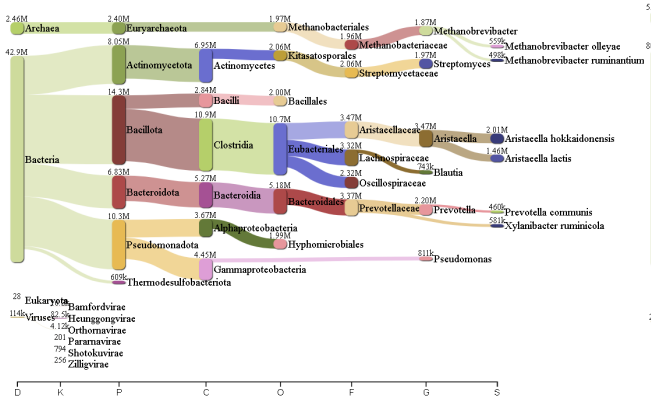


Fig. 147: Pavian report Feb0446 Illumina (K2B)

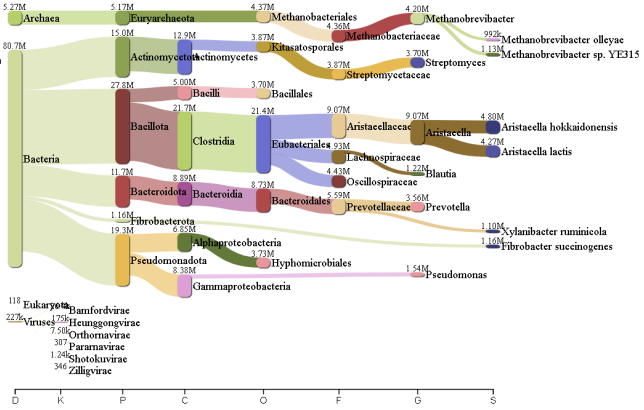


Fig. 148: Pavian report July0446 Illumina (K2B)

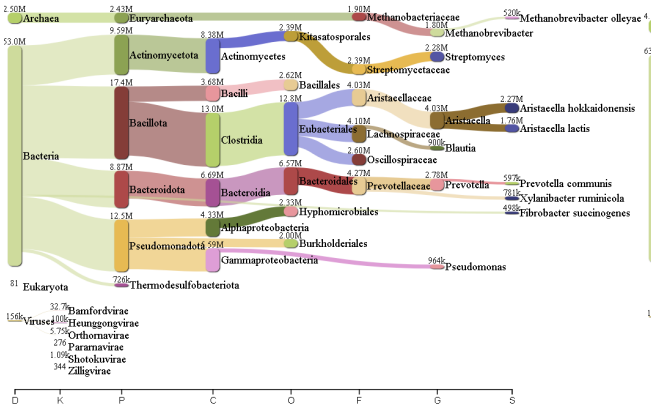


Fig. 149: Pavian report Feb0476 Illumina (K2B)

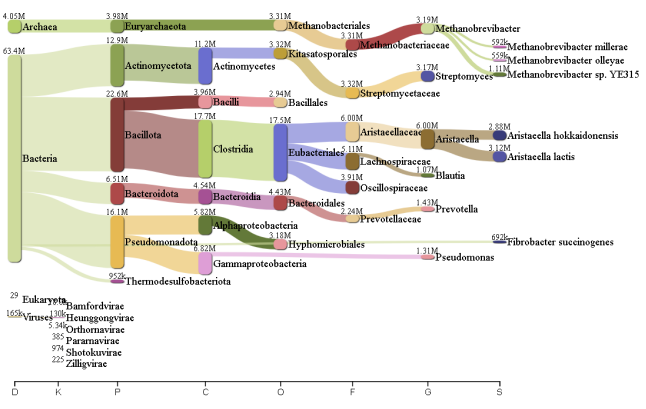


Fig. 150: Pavian report July0476 Illumina (K2B)

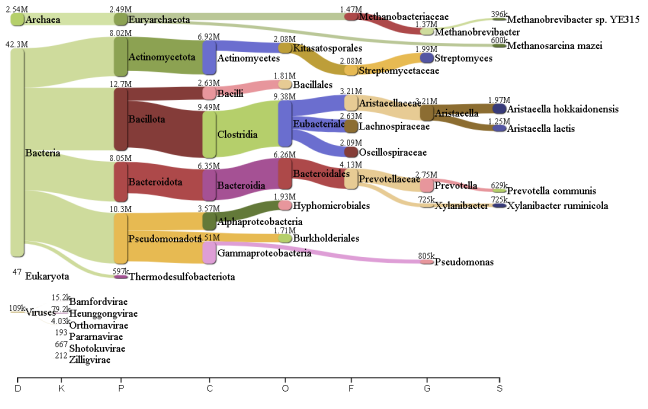


Fig. 151: Pavian report Feb0667 Illumina (K2B)

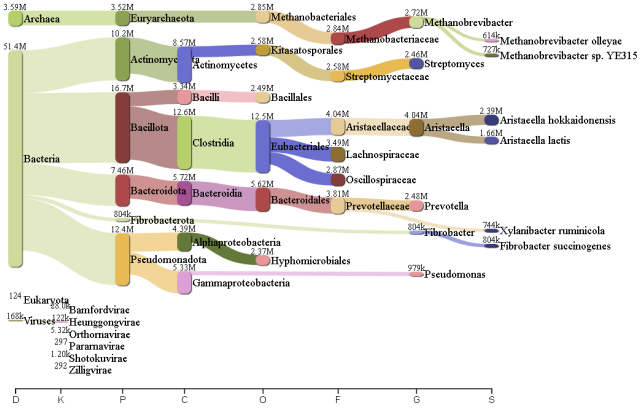


Fig. 152: Pavian report July0667 Illumina (K2B)

G Codes

G.1 Quality Control of the Raw Nanopore Dataset

```
echo "Loading FastQC..."
module load conda
module load fastqc
. /opt/sw/conda/3/etc/profile.d/conda.sh
echo "FastQC loaded !"

```

```
fastqc -t 12 /proj/rumen_interaction/data/boran_rumen/Nanopore/Run_*.fastq -o /proj/
    rumen_interaction/NOBACKUP/results/Nanopore/fastq_rep/

echo "Loading MultiQC..."
module load multiqc
echo "MultiQC loaded !"


echo "Assemblying all the reports with MultiQC..."
mkdir -p /proj/rumen_interaction/NOBACKUP/results/Nanopore/fastq_rep/multiqc/

multiqc /proj/rumen_interaction/NOBACKUP/results/Nanopore/fastq_trim/ -o /proj/
    rumen_interaction/NOBACKUP/results/Nanopore/fastq_rep/multiqc/

echo "Job done !"
```

G.2 Trimming of the Raw Nanopore Dataset

```
for file in $(ls ls /proj/rumen_interaction/data/boran_rumen/Nanopore/Run*);
do
file_name=$(echo $file | cut -d'/' -f7 | cut -d'.' -f1)

echo "Trimming the ${file} of Nanopore with a Phred score of 10..."
cat $file | chopper -t 12 -q 10 > "/proj/rumen_interaction/NOBACKUP/results/Nanopore/chopper/
    trimmed_${file_name}.fastq"
echo "Finished trimming !"


echo "Counting the number of reads for ${file_name}..."
echo $(infoseq "/proj/rumen_interaction/NOBACKUP/results/Nanopore/chopper/trimmed_${file_name
    }.fastq" 2>/dev/null | awk 'NR>1' | wc -l) : "trimmed_${file_name}" >> /proj/
    rumen_interaction/NOBACKUP/results/Nanopore/counting/count_qc_after_trim.txt
echo "Finished counting for ${file_name} !"
done
```

G.3 Quality Control of the Trimmed Nanopore Dataset

```
echo "Loading FastQC..."
module load conda
module load fastqc
. /opt/sw/conda/3/etc/profile.d/conda.sh
echo "FastQC loaded !"

fastqc -t 12 /proj/rumen_interaction/NOBACKUP/results/Nanopore/chopper/*.fastq -o /proj/
    rumen_interaction/NOBACKUP/results/Nanopore/fastq_rep_trim/

echo "Loading MultiQC..."
module load multiqc
echo "MultiQC loaded !"

echo "Assemblying all the reports with MultiQC..."
mkdir -p /proj/rumen_interaction/NOBACKUP/results/Nanopore/fastq_rep_trim/multiqc/
multiqc /proj/rumen_interaction/NOBACKUP/results/Nanopore/fastq_rep_trim/ -o /proj/
    rumen_interaction/NOBACKUP/results/Nanopore/fastq_rep_trim/multiqc/
echo "Job done !"
```

G.4 Mapping Out of the Trimmed Nanopore Dataset

```
echo "Loading the modules..."
module load minimap
module load samtools
module load seqtk
module load emboss
echo "Modules loaded !"

echo "Started the mapping against Bos Taurus..."
for reads in $(ls /proj/rumen_interaction/NOBACKUP/results/Nanopore/chopper/*.fastq)
do
```

```
reads_name=$(echo $reads | cut -d'/' -f8 | cut -d'.' -f1)

minimap2 -t 12 -ax map-ont /proj/rumen_interaction/NOBACKUP/results/Nanopore/minimap2/indexes/index_taurus.mmi $reads > "/proj/rumen_interaction/NOBACKUP/results/Nanopore/minimap2/results/${reads_name}_taurus.sam"
echo "Mapping finished !"

echo "Extracting the unmapped reads..."

echo "Creating the BAM file..."
samtools view -bS "/proj/rumen_interaction/NOBACKUP/results/Nanopore/minimap2/results/${reads_name}_taurus.sam" > "/proj/rumen_interaction/NOBACKUP/results/Nanopore/minimap2/results/${reads_name}_taurus.bam"
echo "BAM file created !"

echo "Removing the SAM file..."
rm "/proj/rumen_interaction/NOBACKUP/results/Nanopore/minimap2/results/${reads_name}_taurus.sam"
echo "SAM file removed !"

echo "Creating the second SAM file..."
samtools view -f4 "/proj/rumen_interaction/NOBACKUP/results/Nanopore/minimap2/results/${reads_name}_taurus.bam" > "/proj/rumen_interaction/NOBACKUP/results/Nanopore/minimap2/results/U_${reads_name}_taurus.sam"
echo "Second SAM file created !"

echo "Removing the BAM file..."
rm "/proj/rumen_interaction/NOBACKUP/results/Nanopore/minimap2/results/${reads_name}_taurus.bam"
echo "BAM file removed !"

echo "Creating the list of unmapped reads..."
cut -f1 "/proj/rumen_interaction/NOBACKUP/results/Nanopore/minimap2/results/U_${reads_name}_taurus.sam" | sort | uniq > "/proj/rumen_interaction/NOBACKUP/results/Nanopore/minimap2/results/U_${reads_name}_taurus.list"
echo "List created !"

echo "Removing the second SAM file..."
rm "/proj/rumen_interaction/NOBACKUP/results/Nanopore/minimap2/results/U_${reads_name}_taurus.sam"
echo "Seconda SAM file removed !"

echo "Creating the new fastq file..."
seqtk subseq $reads "/proj/rumen_interaction/NOBACKUP/results/Nanopore/minimap2/results/U_${reads_name}_taurus.list" > "/proj/rumen_interaction/NOBACKUP/results/Nanopore/minimap2/results/U_${reads_name}_taurus.fastq"
echo "New fastq file created !"

echo "Removing the list of unmapped reads..."
rm "/proj/rumen_interaction/NOBACKUP/results/Nanopore/minimap2/results/U_${reads_name}_taurus.list"
echo "List of unmapped reads removed !"

echo "Unmapped reads extracted !"
done

echo "Counting the reads after mapping against Bos Taurus..."
for data in $(ls /proj/rumen_interaction/NOBACKUP/results/Nanopore/minimap2/results/*.fastq)
do
data_name=$(echo $data | cut -d'/' -f9 | cut -d'.' -f1)
echo $(infoseq $data 2>/dev/null | awk 'NR>1' | wc -l) : $data_name >> /proj/rumen_interaction/NOBACKUP/results/Nanopore/counting/count_qc_after_mapping_out.txt
done
echo "Counting finished !"
```

G.5 Quality Control of the Unmapped Trimmed Nanopore Dataset

```
echo "Loading FastQC..."
module load conda
module load fastqc
. /opt/sw/conda/3/etc/profile.d/conda.sh
```

```
echo "FastQC loaded !"

mkdir -p /proj/rumen_interaction/NOBACKUP/results/Nanopore/fastq_rep_mapp/

fastqc -t 12 /proj/rumen_interaction/NOBACKUP/results/Nanopore/minimap2/results/*.fastq -o /
    proj/rumen_interaction/NOBACKUP/results/Nanopore/fastq_rep_mapp/

echo "Loading MultiQC..."
module load multiqc
echo "MultiQC loaded !"

echo "Assemblying all the reports with MultiQC..."
mkdir -p /proj/rumen_interaction/NOBACKUP/results/Nanopore/fastq_rep_mapp/multiqc/

multiqc /proj/rumen_interaction/NOBACKUP/results/Nanopore/fastq_rep_mapp/ -o /proj/
    rumen_interaction/NOBACKUP/results/Nanopore/fastq_rep_mapp/multiqc/
echo "Job done !"
```

G.6 Taxonomic Profiling of the Unmnapped Trimmed Nanopore Dataset using sourmash

```
echo "Loading the modules..."
module load conda
. /opt/sw/conda/3/etc/profile.d/conda.sh
module load sourmash
echo "Modules loaded !"

for file in $(ls /proj/rumen_interaction/NOBACKUP/results/Nanopore/minimap2/results/*.fastq)
do
f_name=$(echo $file | cut -d'/' -f9 | cut -d'.' -f1)
echo "Creating the sketches..."
sourmash sketch dna $file -p k=31,scaled=100000 --name "${f_name}" -o "/proj/rumen_interaction
    /NOBACKUP/results/Nanopore/sourmash/GTDB/pipeline/${f_name}.sig"
echo "Sketches created !"

echo "Creating a CSV file..."
sourmash gather -k 31 "/proj/rumen_interaction/NOBACKUP/results/Nanopore/sourmash/GTDB/
    pipeline/${f_name}.sig" /proj/rumen_interaction/NOBACKUP/genomes/GTDB/gtdb-rs207.genomic.
    k31.zip -o "/proj/rumen_interaction/NOBACKUP/results/Nanopore/sourmash/GTDB/pipeline/${
    f_name}.gather.k31.csv"
echo "CSV file created !"

echo "Removing the signature file..."
rm "/proj/rumen_interaction/NOBACKUP/results/Nanopore/sourmash/GTDB/pipeline/${f_name}.sig"
echo "Signature file removed !"

echo "Creating a KRONA output..."
sourmash tax metagenome --gather-csv "/proj/rumen_interaction/NOBACKUP/results/Nanopore/
    sourmash/GTDB/pipeline/${f_name}.gather.k31.csv" --taxonomy /proj/rumen_interaction/
    NOBACKUP/genomes/GTDB/gtdb-rs207.taxonomy.with-strain.csv.gz --output-format krona --rank
    species > "/proj/rumen_interaction/NOBACKUP/results/Nanopore/sourmash/GTDB/pipeline/${
    f_name}.krona"
echo "KRONA output created !"

echo "Creating a Kreport output..."
sourmash tax metagenome --gather-csv "/proj/rumen_interaction/NOBACKUP/results/Nanopore/
    sourmash/GTDB/pipeline/${f_name}.gather.k31.csv" --taxonomy /proj/rumen_interaction/
    NOBACKUP/genomes/GTDB/gtdb-rs207.taxonomy.with-strain.csv.gz --output-format kreport > "/
    proj/rumen_interaction/NOBACKUP/results/Nanopore/sourmash/GTDB/pipeline/${f_name}.kreport"
echo "Kreport output created !"

echo "Removing the CSV file..."
rm "/proj/rumen_interaction/NOBACKUP/results/Nanopore/sourmash/GTDB/pipeline/${f_name}.gather.
    k31.csv"
echo "CSV file removed !"
done
```

G.7 Taxonomic Profiling of the Unmapped Trimmed Nanopore Dataset using Kraken2 and Bracken

```
echo "Loading Krake2..."
module load conda
. /opt/sw/conda/3/etc/profile.d/conda.sh
module load kraken2
echo "Kraken2 loaded !"


echo "Performing the taxonomic classification with Kraken2..."
for file in $(ls /proj/rumen_interaction/NOBACKUP/results/Nanopore/minimap2/results/*.fastq)
do
f_name=$(echo $file | cut -d'/' -f9 | cut -d'.' -f1)
kraken2 --db /proj/rumen_interaction/NOBACKUP/genomes/Kraken2/ --threads 12 --output "/proj/
    rumen_interaction/NOBACKUP/results/Nanopore/kraken2/${f_name}.kraken" --report "/proj/
    rumen_interaction/NOBACKUP/results/Nanopore/kraken2/${f_name}.kreport" $file
done
echo "Finished the taxonomic classification with Kraken2 !"


echo "Loading Bracken..."
module load bracken
echo "Bracken loaded !"

echo "Continuing the taxonomic classification with Bracken..."
for file in $(ls /proj/rumen_interaction/NOBACKUP/results/Nanopore/kraken2/*.kreport)
do
f_name=$(echo $file | cut -d'/' -f8 | cut -d'.' -f1)
bracken -d /proj/rumen_interaction/NOBACKUP/genomes/Kraken2/ -i $file -o "/proj/
    rumen_interaction/NOBACKUP/results/Nanopore/kraken2/${f_name}.bracken" -l S
done
echo "Finished the taxonomic classification for Kraken2/Bracken !"
```

G.8 Taxonomic Profiling of the Unmapped Trimmed Nanopore Dataset using Kraken2 and sourmash

```
echo "Loading Kraken2..."
module load conda
. /opt/sw/conda/3/etc/profile.d/conda.sh
module load kraken2
echo "Kraken2 loaded !"

for file in $(ls /proj/rumen_interaction/NOBACKUP/results/Nanopore/minimap2/results/*.fastq)
do
f_name=$(echo $file | cut -d'/' -f9 | cut -d'.' -f1)
echo "Extracting the unclassified reads with Kraken2..."
kraken2 --db /proj/rumen_interaction/NOBACKUP/genomes/Kraken2/ --threads 12 --unclassified-out
     "/proj/rumen_interaction/NOBACKUP/results/Nanopore/k2_sourmash/uncl_${f_name}.fastq"
    $file
echo "Unclassified reads extracted !"
done


echo "Loading the sourmash..."
module load conda
. /opt/sw/conda/3/etc/profile.d/conda.sh
module load sourmash
echo "sourmash loaded !"

for file in $(ls /proj/rumen_interaction/NOBACKUP/results/Nanopore/k2_sourmash/*.fastq)
do
f_name=$(echo $file | cut -d'/' -f8 | cut -d'.' -f1)
echo "Creating the sketches..."
sourmash sketch dna $file -p k=31,scaled=100000 --name "${f_name}" -o "/proj/rumen_interaction
    /NOBACKUP/results/Nanopore/k2_sourmash/${f_name}.sig"
echo "Sketches created !"

echo "Creating a CSV file..."
sourmash gather -k 31 "/proj/rumen_interaction/NOBACKUP/results/Nanopore/k2_sourmash/${f_name
    }.sig" /proj/rumen_interaction/NOBACKUP/genomes/GTDB/gtdb-rs207.genomic.k31.zip -o "/proj/
    rumen_interaction/NOBACKUP/results/Nanopore/k2_sourmash/${f_name}.gather.k31.csv"
echo "CSV file created !"

```

```
echo "Removing the signature file..."
rm "/proj/rumen_interaction/NOBACKUP/results/Nanopore/k2_sourmash/${f_name}.sig"
echo "Signature file removed !"

echo "Creating a KRONA output..."
sourmash tax metagenome --gather-csv "/proj/rumen_interaction/NOBACKUP/results/Nanopore/
    k2_sourmash/${f_name}.gather.k31.csv" --taxonomy /proj/rumen_interaction/NOBACKUP/genomes/
    GTDB/gtdb-rs207.taxonomy.with-strain.csv.gz --output-format krona --rank species > "/proj/
    rumen_interaction/NOBACKUP/results/Nanopore/k2_sourmash/${f_name}.krona"
echo "KRONA output created !"

echo "Creating a Kreport output..."
sourmash tax metagenome --gather-csv "/proj/rumen_interaction/NOBACKUP/results/Nanopore/
    k2_sourmash/${f_name}.gather.k31.csv" --taxonomy /proj/rumen_interaction/NOBACKUP/genomes/
    GTDB/gtdb-rs207.taxonomy.with-strain.csv.gz --output-format kreport > "/proj/
    rumen_interaction/NOBACKUP/results/Nanopore/k2_sourmash/${f_name}.kreport"
echo "Kreport output created !"

echo "Removing the CSV file..."
rm "/proj/rumen_interaction/NOBACKUP/results/Nanopore/k2_sourmash/${f_name}.gather.k31.csv"
echo "CSV file removed !"
done
```

G.9 Quality Control for the Raw Illumina Dataset

```
for file in $(ls /proj/rumen_interaction/data/boran_rumen/Illumina/Sample_WD-3658-*/*)
do
file_name=$(echo $file | cut -d'/' -f8 | cut -d'.' -f1 | cut -d'-' -f3 | cut -d'_' -f1)
echo $(zcat $file | grep -c '^@') : $file_name >> /proj/rumen_interaction/NOBACKUP/results/
    Illumina/counting/raw_count.txt
done

echo "Loading FastQC..."
module load conda
. /opt/sw/conda/3/etc/profile.d/conda.sh
module load fastqc
echo "FastQC loaded !"

fastqc -t 12 /proj/rumen_interaction/data/boran_rumen/Illumina/Sample_WD-3658-*/*.gz -o /proj/
    rumen_interaction/NOBACKUP/results/Illumina/full_analysis/fastqc_raw_rep/

echo "Loading MultiQC..."
module load multiqc
echo "MultiQC loaded !"


echo "Assemblying all the reports with MultiQC..."
multiqc /proj/rumen_interaction/NOBACKUP/results/Illumina/full_analysis/fastqc_raw_rep/ -o /
    proj/rumen_interaction/NOBACKUP/results/Illumina/full_analysis/fastqc_raw_rep/multiqc/
echo "Job done !"
```

G.10 Mapping Out of the Raw Illumina Dataset

```
echo "Loading bowtie2..."
module load bowtie
echo "bowtie2 loaded !"

input_dir="/proj/rumen_interaction/data/boran_rumen/Illumina"
output_dir="/proj/rumen_interaction/NOBACKUP/results/Illumina/bowtie2/alignement"

for sample_dir in $input_dir/Sample_WD-3658-*;
do
        if [ -d "${sample_dir}" ]; then
                fastq_r1=$(echo "${sample_dir}/*_R1_*.fastq.gz")
                fastq_r2=$(echo "${sample_dir}/*_R2_*.fastq.gz")
                sample_date=$(echo $sample_dir | cut -d'/' -f7 | cut -d'_' -f2 | cut -d'-' -f3
    )
        fi

output_file="${output_dir}/${sample_date}_taurus.sam"
```

```
17
18 echo "Mapping Bos Taurus against July0407..."
19 bowtie2 -p 20 -x /proj/rumen_interaction/NOBACKUP/results/Illumina/bowtie2/index_taurus -1
       $fastq_r1 -2 $fastq_r2 > $output_file
20 echo "Mapping finished !"
21
22 echo "Extracting the unmapped reads..."
23
24 echo "Loading samtools..."
25 module load samtools
26 echo "samtools loaded !"
27
28 echo "Creating the BAM file..."
29 samtools view -bS $output_file > "${output_file%.sam}.bam"
30 echo "BAM file created !"
31
32 echo "Removing the SAM file..."
33 rm $output_file
34 echo "SAM file removed !"
35
36 echo "Creating the second SAM file..."
37 samtools view -f4 "${output_file%.sam}.bam" > "${output_dir}/unmapped_${sample_date}_taurus.
       sam"
38 echo "Second SAM file created !"
39
40 echo "Removing the BAM file..."
41 rm "${output_file%.sam}.bam"
42 echo "BAM file removed !"
43
44 echo "Creating the list of unmapped reads..."
45 cut -f1 "${output_dir}/unmapped_${sample_date}_taurus.sam" | sort | uniq > "${output_dir}/
       unmapped_${sample_date}_taurus.list"
46 echo "List created !"
47
48 echo "Removing the second SAM file..."
49 rm "${output_dir}/unmapped_${sample_date}_taurus.sam"
50 echo "Second SAM file removed !"
51
52
53 echo "Loading seqtk..."
54 module load seqtk
55 echo "seqtk loaded !"
56
57 echo "Creating the newly unmapped reads for ${sample_date}..."
58 seqtk subseq $fastq_r1 "${output_dir}/unmapped_${sample_date}_taurus.list" > "${output_dir}/
       unmapped_${sample_date}_R1.fastq"
59 seqtk subseq $fastq_r2 "${output_dir}/unmapped_${sample_date}_taurus.list" > "${output_dir}/
       unmapped_${sample_date}_R2.fastq"
60 echo "Reads created for ${sample_date} !"
61
62 echo "Counting the reads for unmapped_${sample_date}..."
63 echo $(infoseq "${output_dir}/unmapped_${sample_date}_R1.fastq" 2>/dev/null | awk 'NR>1' | wc
       -l) : "${sample_date}" >> /proj/rumen_interaction/NOBACKUP/results/Illumina/counting/
       unmap_count.txt
64 echo "Finished the counting for unmapped_${sample_date} !"
65
66 echo "Zipping the file unmapped_${sample_date}..."
67 gzip -c "${output_dir}/unmapped_${sample_date}_R1.fastq" > "${output_dir}/unmapped_${
       sample_date}_R1.fastq.gz"
68 rm "${output_dir}/unmapped_${sample_date}_R1.fastq"
69 gzip -c "${output_dir}/unmapped_${sample_date}_R2.fastq" > "${output_dir}/unmapped_${
       sample_date}_R2.fastq.gz"
70 rm "${output_dir}/unmapped_${sample_date}_R2.fastq"
71 echo "Unmapped_${sample_date} zipped !"
72
73 echo "Unmapped reads extracted !"
74
75 echo "Removing the list of unmapped reads..."
76 rm "${output_dir}/unmapped_${sample_date}_taurus.list"
77 echo "List removed !"
78
79 done
```

G.11 Quality Control of the Unmapped Illumina Dataset

```
for file in $(ls /proj/rumen_interaction/NOBACKUP/results/Illumina/bowtie2/alignement/*.gz)
do
file_name=$(echo $file | cut -d'/' -f9 | cut -d'.' -f1)
echo $(zcat $file | grep -c '^@') : $file_name >> /proj/rumen_interaction/NOBACKUP/results/
    Illumina/counting/count_qc_after_map.txt
done


echo "Loading fastqc..."
module load conda
module load fastqc
. /opt/sw/conda/3/etc/profile.d/conda.sh
echo "Fastqc loaded !"

fastqc -t 20 /proj/rumen_interaction/NOBACKUP/results/Illumina/bowtie2/alignement/*.gz -o /
    proj/rumen_interaction/NOBACKUP/results/Illumina/full_analysis/fastqc_unmap_rep/

echo "Loading multiqc..."
module load multiqc
echo "Multiqc loaded !"

multiqc /proj/rumen_interaction/NOBACKUP/results/Illumina/full_analysis/fastqc_unmap_rep/ -o /
    proj/rumen_interaction/NOBACKUP/results/Illumina/full_analysis/fastqc_unmap_rep/multiqc/
```

G.12 Trimming of the Unmapped Illumina Dataset

```
echo "Loading the modules..."
module load conda
. /opt/sw/conda/3/etc/profile.d/conda.sh

module load seqkit
module load emboss
echo "Modules loaded !"

for file in $(ls /proj/rumen_interaction/NOBACKUP/results/Illumina/bowtie2/alignement/*.gz)
do
f_name=$(echo $file | cut -d'/' -f9 | cut -d'.' -f1)

echo "Remove the duplicated reads for ${f_name}..."
seqkit rmdup --threads 20 --by-seq --ignore-case $file -o "/proj/rumen_interaction/NOBACKUP/
    results/Illumina/trim/trim_${f_name}.fastq.gz"
echo "Duplicated reads removed for ${f_name} !"

echo "Removing the unmapped reads, in order to gain more disk space..."
rm $file
echo "Unmapped reads removed !"

echo "Counting the reads for ${f_name}..."
echo $(zcat "/proj/rumen_interaction/NOBACKUP/results/Illumina/trim/trim_${f_name}.fastq.gz" |
     grep -c '^@') : "trim_${f_name}" >> /proj/rumen_interaction/NOBACKUP/results/Illumina/
    counting/count_qc_after_dupl.txt
echo "Counting finished !"
done
```

G.13 Quality Control of the Trimmed Unmapped Illumina Dataset

```
for file in $(ls /proj/rumen_interaction/NOBACKUP/results/Illumina/trim/*.gz)
do
file_name=$(echo $file | cut -d'/' -f8 | cut -d'.' -f1)
echo $(zcat $file | grep -c '^@') : $file_name >> /proj/rumen_interaction/NOBACKUP/results/
    Illumina/counting/count_qc_after_no_dupl.txt
done


echo "Loading fastqc..."
module load conda
module load fastqc
```

```
. /opt/sw/conda/3/etc/profile.d/conda.sh
echo "Fastqc loaded !"

fastqc -t 20 /proj/rumen_interaction/NOBACKUP/results/Illumina/trim/*.gz -o /proj/
    rumen_interaction/NOBACKUP/results/Illumina/full_analysis/fastqc_trim_rep/

echo "Loading multiqc..."
module load multiqc
echo "Multiqc loaded !"

multiqc /proj/rumen_interaction/NOBACKUP/results/Illumina/full_analysis/fastqc_trim_rep/ -o /
    proj/rumen_interaction/NOBACKUP/results/Illumina/full_analysis/fastqc_trim_rep/multiqc/
```

G.14 Taxonomic Profiling of the Trimmed Unmapped Illumina Dataset using sourmash

```
echo "Loading the modules..."
module load conda
. /opt/sw/conda/3/etc/profile.d/conda.sh
module load sourmash
echo "Modules loaded !"


input_dir="/proj/rumen_interaction/NOBACKUP/results/Illumina/trim"
output_dir="/proj/rumen_interaction/NOBACKUP/results/Illumina/taxa_res/sourmash/gtdb/pipeline"

for sample_file in $input_dir/*;
do
        trim_R1=$(echo "${sample_file}" | grep 'R1')
        trim_R2=$(echo "${sample_file}" | grep 'R2')
        output_file_templ=$(echo ${sample_file} | cut -d'_' -f4)

echo "Creating the sketches..."
sourmash sketch dna -p k=31,scaled=30000 --name $output_file_templ $trim_R1 $trim_R2 --merge
    $output_file_templ -o "/proj/rumen_interaction/NOBACKUP/results/Illumina/taxa_res/sourmash
    /gtdb/pipeline/${output_file_templ}.sig"
echo "Sketches created !"

echo "Creating a CSV file..."
sourmash gather -k 31 "/proj/rumen_interaction/NOBACKUP/results/Illumina/taxa_res/sourmash/
    gtdb/pipeline/${output_file_templ}.sig" /proj/rumen_interaction/NOBACKUP/genomes/GTDB/gtdb
    -rs207.genomic.k31.zip -o "/proj/rumen_interaction/NOBACKUP/results/Illumina/taxa_res/
    sourmash/gtdb/pipeline/${output_file_templ}.gather.k31.csv"
echo "CSV file created !"

echo "Removing the signature file..."
rm "/proj/rumen_interaction/NOBACKUP/results/Illumina/taxa_res/sourmash/gtdb/pipeline/${
    output_file_templ}.sig"
echo "Signature file removed !"

echo "Creating a KRONA output..."
sourmash tax metagenome --gather-csv "/proj/rumen_interaction/NOBACKUP/results/Illumina/
    taxa_res/sourmash/gtdb/pipeline/${output_file_templ}.gather.k31.csv" --taxonomy /proj/
    rumen_interaction/NOBACKUP/genomes/GTDB/gtdb-rs207.taxonomy.with-strain.csv.gz --output-
    format krona --rank species > "/proj/rumen_interaction/NOBACKUP/results/Illumina/taxa_res/
    sourmash/gtdb/pipeline/${output_file_templ}.krona"
echo "KRONA output created !"

echo "Creating a Kreport output..."
sourmash tax metagenome --gather-csv "/proj/rumen_interaction/NOBACKUP/results/Illumina/
    taxa_res/sourmash/gtdb/pipeline/${output_file_templ}.gather.k31.csv" --taxonomy /proj/
    rumen_interaction/NOBACKUP/genomes/GTDB/gtdb-rs207.taxonomy.with-strain.csv.gz --output-
    format kreport > "/proj/rumen_interaction/NOBACKUP/results/Illumina/taxa_res/sourmash/gtdb
    /pipeline/${output_file_templ}.kreport"
echo "Kreport output created !"

echo "Removing the CSV file..."
rm "/proj/rumen_interaction/NOBACKUP/results/Illumina/taxa_res/sourmash/gtdb/pipeline/${
    output_file_templ}.gather.k31.csv"
echo "CSV file removed !"
done
```

G.15 Taxonomic Profiling of the Trimmed Unmapped Illumina Dataset using Kraken2 and Bracken

```
echo "Loading Kraken2..."
module load kraken2
echo "Kraken2 loaded!"

input_dir="/proj/rumen_interaction/NOBACKUP/results/Illumina/trim"
output_dir="/proj/rumen_interaction/NOBACKUP/results/Illumina/taxa_res/k2b"

for trim_R1 in $input_dir/*_R1*.fastq.gz; do
    trim_R2="${trim_R1/_R1/_R2}"  # Substitute R1 with R2 to get the corresponding R2 file
    output_file_templ=$(basename "$trim_R1" | cut -d'_' -f3 | cut -d'.' -f1)

    echo "Performing the taxonomic classification with Kraken2..."
    kraken2 --db /proj/rumen_interaction/NOBACKUP/genomes/Kraken2/ --threads 20 --paired
    $trim_R1 $trim_R2 --output "${output_dir}/${output_file_templ}.kraken" --report "${
    output_dir}/${output_file_templ}.kreport"
    echo "Finished the taxonomic classification with Kraken2!"

    echo "Loading Bracken..."
    module load bracken
    echo "Bracken loaded!"

    bracken -d /proj/rumen_interaction/NOBACKUP/genomes/Kraken2 -i "${output_dir}/${
    output_file_templ}.kreport" -o "${output_dir}/${output_file_templ}.bracken" -l S
    echo "Finished Bracken for $output_file_templ!"
done
```

G.16 Taxonomic Profiling of the Trimmed Unmapped Illumina Dataset using Kraken2 and sourmash

```
echo "Loading Kraken2..."
module load kraken2
echo "Kraken2 loaded !"


input_dir="/proj/rumen_interaction/NOBACKUP/results/Illumina/trim"
output_dir="/proj/rumen_interaction/NOBACKUP/results/Illumina/taxa_res/kraken"


for trim_R1 in $input_dir/*_R1*.fastq.gz;
do
        trim_R2="${trim_R1/_R1/_R2}"
        output_file_templ=$(echo $trim_R1 | cut -d'/' -f8 | cut -d'.' -f1 | cut -d'_' -f1,2,3)
        echo "These are the forward reads : ${trim_R1}"
        echo "These are the reverse reads : ${trim_R2}"
        echo "This is the output file : ${output_file_templ}"


        echo "Extracting the unclassified reads with Kraken2..."
        kraken2 --db /proj/rumen_interaction/NOBACKUP/genomes/Kraken2/ --threads 20 --paired
    $trim_R1 $trim_R2 --unclassified-out "/proj/rumen_interaction/NOBACKUP/results/Illumina/
    taxa_res/kraken/uncl_${output_file_templ}_R#.fastq"
        echo "Reads extracted !"


        echo "Loading sourmash..."
        module load conda
        . /opt/sw/conda/3/etc/profile.d/conda.sh
        module load sourmash
        echo "Sourmash loaded !"

        sourmash sketch dna -p k=31,scaled=30000 --name ${output_file_templ} "/proj/
    rumen_interaction/NOBACKUP/results/Illumina/taxa_res/kraken/uncl_${output_file_templ}_R1.
    fastq" "/proj/rumen_interaction/NOBACKUP/results/Illumina/taxa_res/kraken/uncl_${
    output_file_templ}_R2.fastq" --merge ${output_file_templ} -o "/proj/rumen_interaction/
    NOBACKUP/results/Illumina/taxa_res/kraken/${output_file_templ}.sig"

        echo "Loading emboss..."
```

```
        module load emboss
        echo "Emboss loaded !"

        echo "Counting the unclassified reads from Kraken2..."
        f_name=$(echo /proj/rumen_interaction/NOBACKUP/results/Illumina/taxa_res/kraken/uncl_${output_file_templ}_R*.fastq | cut -d'/' -f9 | cut -d'.' -f1)
        echo $(infoseq "/proj/rumen_interaction/NOBACKUP/results/Illumina/taxa_res/kraken/uncl_${output_file_templ}_R*.fastq" 2>/dev/null | awk 'NR>1' | wc -l) : $f_name >> /proj/rumen_interaction/NOBACKUP/results/Illumina/couting/count_uncl_krake2.txt
        echo "Counting finished !"

done

echo "Loading FastQC..."
module load conda
module load fastqc
. /opt/sw/conda/3/etc/profile.d/conda.sh
echo "FastQC loaded !"

echo "Quality control over the unclassified reads from Kraken2..."
fastqc -t 20 /proj/rumen_interaction/NOBACKUP/results/Illumina/taxa_res/kraken/*.fastq -o /proj/rumen_interaction/NOBACKUP/results/Illumina/full_analysis/fastqc_uncl_rep/
echo "Quality check over !"

echo "Removing the unclassified sequences from Kraken2..."
rm /proj/rumen_interaction/NOBACKUP/results/Illumina/taxa_res/kraken/*.fastq
echo "Reads removed !"

echo "Loading MultiQC..."
module load multiqc
echo "MultiQC loaded !"

echo "Assemblying together the fastqc reports..."
multiqc /proj/rumen_interaction/NOBACKUP/results/Illumina/full_analysis/fastqc_uncl_rep/ -o /proj/rumen_interaction/NOBACKUP/results/Illumina/full_analysis/fastqc_uncl_rep/multiqc/
echo "MultiQC report created !"

echo "Loading sourmash..."
module load conda
. /opt/sw/conda/3/etc/profile.d/conda.sh
module load sourmash
echo "sourmash loaded !"


for sig in $(ls /proj/rumen_interaction/NOBACKUP/results/Illumina/taxa_res/kraken/*.sig);
do
sig_name=$(echo $sig | cut -d'/' -9 | cut -d'.' -f1)
echo "Creating a CSV file..."
sourmash gather -k 31 $sig /proj/rumen_interaction/NOBACKUP/genomes/GTDB/gtdb-rs207.genomic.k31.zip -o "/proj/rumen_interaction/NOBACKUP/results/Illumina/taxa_res/kraken/${sig_name}.gather.k31.csv"
echo "CSV file created !"

echo "Removing the signature file..."
rm $sig
echo "Signature file removed !"

echo "Creating a KRONA output..."
sourmash tax metagenome --gather-csv "/proj/rumen_interaction/NOBACKUP/results/Illumina/taxa_res/kraken/${sig_name}.gather.k31.csv" --taxonomy /proj/rumen_interaction/NOBACKUP/genomes/GTDB/gtdb-rs207.taxonomy.with-strain.csv.gz --output-format krona --rank species > "/proj/rumen_interaction/NOBACKUP/results/Illumina/taxa_res/kraken/${sig_name}.krona"
echo "KRONA output created !"

echo "Creating a Kreport output..."
sourmash tax metagenome --gather-csv "/proj/rumen_interaction/NOBACKUP/results/Illumina/taxa_res/kraken/${sig_name}.gather.k31.csv" --taxonomy /proj/rumen_interaction/NOBACKUP/genomes/GTDB/gtdb-rs207.taxonomy.with-strain.csv.gz --output-format kreport > "/proj/rumen_interaction/NOBACKUP/results/Illumina/taxa_res/kraken/${sig_name}.kreport"
echo "Kreport output created !"

echo "Removing the CSV file..."
rm "/proj/rumen_interaction/NOBACKUP/results/Illumina/taxa_res/kraken/${sig_name}.gather.k31.
```

```
    csv"
echo "CSV file removed !"
done
```

G.17 Plotting of Different Types of Data

```
import sys,getopt
import matplotlib.pyplot as plt
import os

def organise_data(file_path:str)->dict:
    data={}
    with open(file_path,'r') as file:
        for line in file:
            parts=line.strip().split(':')
            number=int(parts[0].strip())
            sample_name=str(parts[1].strip())
            data[sample_name]=number
    return data


def ensure_output_dir(output_dir):
    if output_dir and not os.path.exists(output_dir):
        os.makedirs(output_dir)
    else: None

def plot_initial_data(data:dict,output_dir:str=None):
    list_sample_name=list(data.keys())
    list_seq_nbr=list(data.values())
    fig=plt.figure(figsize=(14,7))
    plt.bar(list_sample_name,list_seq_nbr,color='blue',width=0.8)
    plt.xlabel("Sample Name",fontsize=14)
    plt.xticks(rotation=90,fontsize=10)
    plt.ylabel("Number of Reads",fontsize=14)
    plt.ylim(0,3000000)
    plt.title("Number of Reads Across Nanopore Samples")
    plt.tight_layout()
    if output_dir:
        ensure_output_dir(output_dir)
        plt.savefig(os.path.join(output_dir,"plot_initial_data.png"))
    else:
        plt.savefig("plot_initial_data.png")
    plt.close(fig)


def plot_run_data(data:dict,run_nbr:int,output_dir:str=None):
    list_sample_name=sorted([sample_name for sample_name in data.keys() if f'Run_{run_nbr}' in
     sample_name])
    list_seq_nbr=[data[sample_name] for sample_name in list_sample_name]
    fig=plt.figure(figsize=(14,7))
    plt.bar(list_sample_name,list_seq_nbr,color='blue',width=0.8)
    plt.xlabel('Sample Name',fontsize=14)
    plt.xticks(rotation=90,fontsize=10)
    plt.ylabel('Number of Reads',fontsize=14)
    plt.title(f'Number of Reads Across Nanopore Samples for Run_{run_nbr}')
    plt.ylim(0,3000000)
    plt.tight_layout()
    if output_dir:
        ensure_output_dir(output_dir)
        plt.savefig(os.path.join(output_dir,f'plot_run_{run_nbr}_data.png'))
    else:
        plt.savefig(f'plot_run_{run_nbr}_data.png')
    plt.close(fig)


def plot_period_data(data:dict,period:str,output_dir:str=None):
    list_sample_name = sorted([sample_name for sample_name in data.keys() if period in
    sample_name])
    list_seq_number = [data[sample_name] for sample_name in list_sample_name]
    fig = plt.figure(figsize=(14,7))
    plt.bar(list_sample_name,list_seq_number,color='blue',width=0.8)
    plt.xlabel("Sample Name",fontsize=14)
```

```
    plt.xticks(rotation=90,fontsize=10)
    plt.ylabel("Number of Reads",fontsize=14)
    plt.title(f"Number of Reads Across Nanopore Samples for {period}")
    plt.ylim(0,3000000)
    plt.tight_layout()
    if output_dir:
        ensure_output_dir(output_dir)
        plt.savefig(os.path.join(output_dir,f'plot_period_{period}_data.png'))
    else:
        plt.savefig(f'plot_period_{period}_data.png')
    plt.close(fig)


def plot_individual_data(data:dict,individual:str,output_dir:str=None):
    list_sample_name = sorted([sample_name for sample_name in data.keys() if individual in
    sample_name])
    list_seq_number = [data[sample_name] for sample_name in list_sample_name]
    fig = plt.figure(figsize=(14,7))
    plt.bar(list_sample_name,list_seq_number,color='blue',width=0.8)
    plt.xlabel("Sample Name",fontsize=14)
    plt.xticks(rotation=90,fontsize=10)
    plt.ylabel("Number of Reads",fontsize=14)
    plt.title(f"Number of Reads Across Nanopore Samples for {individual}")
    plt.ylim(0,3000000)
    plt.tight_layout()
    if output_dir:
        ensure_output_dir(output_dir)
        plt.savefig(os.path.join(output_dir,f'plot_individual_{individual}_data.png'))
    else:
        plt.savefig(f'plot_individual_{individual}_data.png')
    plt.close(fig)


def usage():
    print("Usage: plotting.py -o <outputfile> -v")
    print("Options:")
    print(" -i, --input     specify input file")
    print(" -h, --help      show this help message and exit")
    print(" -o, --output    specify output file")
    print(" -v, --verbose   enable verbose mode")




def main():
    try:
        opts,args=getopt.getopt(sys.argv[1:],"hi:o:v",["help","input=","output="])
    except getopt.GetoptError as error:
        print(error)
        usage()
        sys.exit(2)
    output=None
    verbose=False
    file_path=None
    for o,a in opts:
        if o in ("-i","--input"):
            file_path=a
        elif o=="-v":
            verbose=True
        elif o in ("-h","--help"):
            usage()
            sys.exit()
        elif o in ("-o","--output"):
            output_dir=a
        else:
            assert False, "unhandled option"
    if file_path:
        data = organise_data(file_path)
        plot_initial_data(data,output_dir)
        for run_nbr in range(1,3):
            plot_run_data(data,run_nbr,output_dir)
        for month in ['Feb','July']:
            plot_period_data(data,month,output_dir)
```

```
        for individual in ['0476','0446','0428','0407','0350','0667','0199']:
            plot_individual_data(data,individual,output_dir)

    else:
        usage()
        sys.exit(2)

if __name__ == "__main__":
    main()
```

G.18 Boxplot of various phyla

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import os
import argparse

def parse_args():
    parser = argparse.ArgumentParser(description='Generate boxplots for phylum percentages.')
    parser.add_argument('-d', '--directory', type=str, required=True, help='Directory
    containing .krona files')
    parser.add_argument('-o', '--output', type=str, required=True, help='Path to save the
    output plot')
    return parser.parse_args()

def show_directory_content(directory_path: str, extension: str) -> list:
    list_of_file_paths = []
    for file_name in os.listdir(directory_path):
        file_path = os.path.join(directory_path, file_name)
        if os.path.isfile(file_path) and file_name.endswith(extension):
            list_of_file_paths.append(file_path)
    return list_of_file_paths

def get_phylum_all_count(sample_path: str) -> dict:
    data = {}
    with open(sample_path, 'r') as file:
        sample_name = '_'.join(sample_path.split('/')[-1].split('.')[0].split('_')[2:4])
        line_count = len(file.readlines()[1:-1])
        data[sample_name] = line_count
    return data

def get_phylum_count(directory_path: str, phylum: str) -> dict:
    data = {}
    for sample in show_directory_content(directory_path, '.krona'):
        phylum_count = 0
        sample_name = '_'.join(sample.split('/')[-1].split('.')[0].split('_')[2:4])
        with open(sample, 'r') as file:
            lines = file.readlines()[1:-1]
            for line in lines:
                taxa_phylum = line.strip().split('\t')[2].split('__')[1]
                if taxa_phylum == phylum:
                    phylum_count += 1
        total_count = list(get_phylum_all_count(sample).values())[0]
        data[sample_name] = round(phylum_count / total_count * 100, 2)
    return data

def main():
    args = parse_args()

    data1 = get_phylum_count(args.directory, 'Bacteroidota')
    data2 = get_phylum_count(args.directory, 'Firmicutes')
    data3 = get_phylum_count(args.directory, 'Proteobacteria')
    data4 = get_phylum_count(args.directory, 'Actinobacteria')

    df = pd.DataFrame({
        'Bacteroidota': list(data1.values()),
        'Firmicutes': list(data2.values()),
        'Proteobacteria': list(data3.values()),
        'Actinobacteria': list(data4.values())
    })

```

```
    df_melted = df.melt(var_name='Phylum', value_name='Percentage')

    plt.figure(figsize=(12, 8))
    sns.boxplot(x='Phylum', y='Percentage', data=df_melted, palette='viridis',hue='Phylum',
    legend=False)

    plt.title('Distribution of Phylum Percentages', fontsize=16)
    plt.xlabel('Phylum', fontsize=14)
    plt.ylabel('Percentage', fontsize=14)

    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)

    plt.savefig(args.output)
    print(f"Plot saved to {args.output}")

if __name__ == '__main__':
    main()
```

G.19 Lost Reads Plot Analysis for Illumina dataset

```
import re
import matplotlib.pyplot as plt
import numpy as np
import argparse

def prepare_data(file_path: str) -> dict:
    data = {}
    with open(file_path, 'r') as file:
        for line in file:
            read_count = line.strip().split(' : ')[0]
            sample_name = line.strip().split(' : ')[1]
            data[sample_name] = read_count
    return data

def calculate_read_variation(first_dataset: dict, second_dataset: dict, forward_reverse: bool)
     -> dict:
    data_input = {}
    for sample_name in first_dataset.keys():
        if not forward_reverse:
            data_input[sample_name] = int(first_dataset[sample_name]) - int(second_dataset[
    sample_name])
        else:
            data_input[f'{sample_name}_R1'] = int(first_dataset[sample_name]) - int(
    second_dataset.get(f'{sample_name}_R1', 0))
            data_input[f'{sample_name}_R2'] = int(first_dataset[sample_name]) - int(
    second_dataset.get(f'{sample_name}_R2', 0))
    return data_input

def stacked_bar_plot(samples: list, lost_reads_mapping: list, lost_reads_trimming: list,
    output_file: str):
    fig, (ax1, ax2) = plt.subplots(2, 1, sharex=True, figsize=(14, 10), gridspec_kw={'
    height_ratios': [2, 1]})

    bar_width = 0.5
    index = np.arange(len(samples))

    ax1.bar(index, lost_reads_mapping, bar_width, label='Lost Reads (Mapping)')
    ax1.bar(index, lost_reads_trimming, bar_width, bottom=lost_reads_mapping, label='Lost
    Reads (Trimming)')

    ax1.set_ylim(20000000, max(max(lost_reads_mapping), max(lost_reads_trimming)) + 1000000)
    ax2.set_ylim(0, 1000000)

    ax2.bar(index, lost_reads_mapping, bar_width, label='Lost Reads (Mapping)')
    ax2.bar(index, lost_reads_trimming, bar_width, bottom=lost_reads_mapping, label='Lost
    Reads (Trimming)')

    ax1.spines['bottom'].set_visible(False)
    ax2.spines['top'].set_visible(False)
    ax1.tick_params(labeltop=False)
    ax2.xaxis.tick_bottom()
```

```
    d = .015
    kwargs = dict(transform=ax1.transAxes, color='k', clip_on=False)
    ax1.plot((-d, +d), (-d, +d), **kwargs)
    ax1.plot((1 - d, 1 + d), (-d, +d), **kwargs)

    kwargs.update(transform=ax2.transAxes)
    ax2.plot((-d, +d), (1 - d, 1 + d), **kwargs)
    ax2.plot((1 - d, 1 + d), (1 - d, 1 + d), **kwargs)

    ax2.set_xlabel('Sample', fontsize=14)
    ax1.set_ylabel('Number of Lost Reads', fontsize=14)
    ax2.set_ylabel('Number of Lost Reads', fontsize=14)
    fig.suptitle('Lost Reads During Mapping and Trimming Steps', fontsize=12)

    ax2.set_xticks(index)
    ax2.set_xticklabels(samples, rotation=90, fontsize=14)

    ax1.legend()

    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.savefig(output_file)

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Plot lost reads during mapping and trimming
    steps.")
    parser.add_argument('-r', '--raw', required=True, help='Path to raw reads file')
    parser.add_argument('-u', '--unmap', required=True, help='Path to unmapped reads file')
    parser.add_argument('-t', '--trim', required=True, help='Path to trimmed reads file')
    parser.add_argument('-o', '--output', required=True, help='Output file for the figure')

    args = parser.parse_args()

    raw_reads_dict = prepare_data(args.raw)
    unmapp_reads_dict = prepare_data(args.unmap)
    trim_reads_dict = prepare_data(args.trim)

    samples = list(raw_reads_dict.keys())
    raw_reads = list(raw_reads_dict.values())
    unmapp_reads = list(unmapp_reads_dict.values())
    trim_reads = list(trim_reads_dict.values())

    lost_mapping = [int(raw) - int(unmap) for raw, unmap in zip(raw_reads, unmapp_reads)]
    lost_trimming = [int(unmap) - int(trim) for unmap, trim in zip(unmapp_reads, trim_reads)]

    stacked_bar_plot(samples, lost_mapping, lost_trimming, args.output)
```

G.20 Scatter Plot Analysis of Unmapped Trimmed Illumina Reads vs. Species Count from Taxonomic Classification with Kraken2

```
import os
import argparse
import matplotlib.pyplot as plt

def show_directory_content(directory_path: str, extension: str) -> list:
    list_of_file_paths = []
    for file_name in os.listdir(directory_path):
        file_path = os.path.join(directory_path, file_name)
        if os.path.isfile(file_path) and file_name.endswith(extension):
            list_of_file_paths.append(file_path)
    return list_of_file_paths

def count_species(file_path: str) -> int:
    species_count = 0
    with open(file_path, 'r') as file:
        for line in file:
            taxa_level = line.strip().split('\t')[3]
            if taxa_level == 'S':
                species_count += 1
    return species_count

```

```
def prepare_count_data(file_input: str, R_value: str) -> dict:
    data = {}
    with open(file_input, 'r') as file:
        for line in file:
            read_count = int(line.strip().split(' : ')[0])
            sample_name = str(line.strip().split(' : ')[1])
            if sample_name.endswith(R_value):
                data[sample_name] = read_count
    return data

def get_sample_name(list_file_input: str) -> list:
    data = []
    for file in show_directory_content(list_file_input, '.bracken'):
        sample_name=file.split('\\')[-1].split('.')[0]
        if sample_name.startswith('k2b/'):
            sample_name=sample_name.replace('k2b/','')
        data.append(sample_name)
    return data

def scatter_plot_reads_vs_species(count_file_input: str, directory_kreport_kraken: str,
    output_plot: str):
    forward_reads_count = list(prepare_count_data(count_file_input, 'R1').values())
    reverse_reads_count = list(prepare_count_data(count_file_input, 'R2').values())
    sample_names = get_sample_name(directory_kreport_kraken)
    species_count = [count_species(file) for file in show_directory_content(
    directory_kreport_kraken, '.kreport') if '_bracken' not in file]

    plt.scatter(forward_reads_count, species_count, color='blue', label='Forward Reads')
    plt.scatter(reverse_reads_count, species_count, color='green', label='Reverse Reads')

    plt.xlabel('Number of Reads')
    plt.ylabel('Species Count')
    plt.title('Species Count VS Number of Reads per Sample')
    for _, sample in enumerate(sample_names):
        plt.annotate(sample + ' F', (forward_reads_count[_], species_count[_]), color='blue')
        plt.annotate(sample + ' R', (reverse_reads_count[_], species_count[_]), color='green')
    plt.legend()
    plt.savefig(output_plot)

def main():
    parser = argparse.ArgumentParser(description='Generate a scatter plot of species count
    versus number of reads per sample.')
    parser.add_argument('-r', '--read_file_input', required=True, help='Path to the read count
     input file')
    parser.add_argument('-d', '--directory_kraken', required=True, help='Path to the directory
     containing Kraken kreport files')
    parser.add_argument('-o', '--output_plot', required=True, help='Path to save the output
    plot')

    args = parser.parse_args()

    scatter_plot_reads_vs_species(args.read_file_input, args.directory_kraken, args.
    output_plot)

if __name__ == "__main__":
    main()
```

G.21 Scatter Plot Analysis of Classified Illumina Reads vs. Species Count from Taxonomic Classification with Kraken2 / Bracken

```
import os
import matplotlib.pyplot as plt
import argparse

def show_directory_content(directory_path: str, extension: str) -> list:
    list_of_file_paths = []
    for file_name in os.listdir(directory_path):
        file_path = os.path.join(directory_path, file_name)
        if os.path.isfile(file_path) and file_name.endswith(extension):
            list_of_file_paths.append(file_path)
    return list_of_file_paths
```

```
12
13 def get_read_count_bracken(file_path: str) -> int:
14     total_read_count = 0
15     with open(file_path, 'r') as file:
16         file.readline()
17         for line in file:
18             k2b_read_count = int(line.strip().split('\t')[4])
19             total_read_count += k2b_read_count
20     return total_read_count
21
22 def get_read_count_kraken2(file_path: str) -> int:
23     total_read_count = 0
24     with open(file_path, 'r') as file:
25         file.readline()
26         for line in file:
27             k2b_read_count = int(line.strip().split('\t')[3])
28             total_read_count += k2b_read_count
29     return total_read_count
30
31 def get_read_count_k2b(file_path: str) -> int:
32     total_read_count = 0
33     with open(file_path, 'r') as file:
34         file.readline()
35         for line in file:
36             k2b_read_count = int(line.strip().split('\t')[5])
37             total_read_count += k2b_read_count
38     return total_read_count
39
40 def get_sample_name(file_path: str) -> str:
41     sample_name=file_path.split('\\')[-1].split('.')[0]
42     if sample_name.startswith('k2b/'):
43         sample_name=sample_name.replace('k2b/','')
44     return sample_name
45
46 def get_species_count(file_path: str) -> int:
47     with open(file_path, 'r') as file:
48         file.readline()  # Skip header line
49         line_count = sum(1 for line in file)
50     return line_count
51
52 def scatter_plot_reads_vs_species(directory_path: str, output_pic_before: str,
    output_pic_after: str):
53     sample_names = [get_sample_name(file) for file in show_directory_content(directory_path, '
    .bracken')]
54     species_count = [get_species_count(file) for file in show_directory_content(directory_path
    , '.bracken')]
55
56     read_count_before = [get_read_count_kraken2(file) for file in show_directory_content(
    directory_path, '.bracken')]
57
58     plt.figure(figsize=(10, 6))
59     plt.scatter(read_count_before, species_count, color='blue', label='Samples')
60     for i, sample in enumerate(sample_names):
61         plt.annotate(sample, (read_count_before[i], species_count[i]), fontsize=12, ha='right'
    )
62     plt.title('Species Count VS Number of Reads Per Sample Before Bracken')
63     plt.xlabel('Number of Reads', fontsize=14)
64     plt.xticks(fontsize=12)
65     plt.ylabel('Number of Species', fontsize=14)
66     plt.yticks(fontsize=12)
67     plt.grid(True)
68     plt.savefig(output_pic_before, dpi=300)
69
70     read_count_after = [get_read_count_k2b(file) for file in show_directory_content(
    directory_path, '.bracken')]
71
72     plt.figure(figsize=(10, 6))
73     plt.scatter(read_count_after, species_count, color='green', label='Samples')
74     for i, sample in enumerate(sample_names):
75         plt.annotate(sample, (read_count_after[i], species_count[i]), fontsize=12, ha='right')
76     plt.title('Species Count VS Number of Reads Per Sample After Bracken')
77     plt.xlabel('Number of Reads', fontsize=14)
78     plt.xticks(fontsize=12)
```

```
    plt.ylabel('Number of Species', fontsize=14)
    plt.yticks(fontsize=12)
    plt.grid(True)
    plt.savefig(output_pic_after, dpi=300)

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Plot species count vs number of reads.')

    parser.add_argument('-d', '--directory', type=str, required=True, help='Directory path containing the .bracken files.')
    parser.add_argument('-b', '--output_before', type=str, required=True, help='Output PNG file to save the plot (before Bracken correction).')
    parser.add_argument('-a', '--output_after', type=str, required=True, help='Output PNG file to save the plot (after Bracken correction).')

    args = parser.parse_args()

    scatter_plot_reads_vs_species(args.directory, args.output_before, args.output_after)
```

G.22 Installation Guide for the Software Necessary for this Metagenomic Analysis

```
echo "Installing Conda"
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
bash Miniconda3-latest-Linux-x86_64.sh
conda init base

echo "Installing FastQC"
conda install fastqc

echo "Installing MultiQC"
conda install multiqc

echo "Samtools"
sudo apt install samtools

echo "Seqkit"
conda install -c bioconda seqkit

echo "Installing bowtie2"
sudo apt install bowtie2

echo "Installing minimap2"
sudo apt install minimap2

echo "Installing sourmash"
bash Miniforge3-Linux-x86_64.sh
~/minforge3/bin/mamba init
echo 'source~/.bashrc' > ~/.bash_profile
source ~/.bash_profile

echo "Installing Kraken2"
sudo apt install kraken2

echo "Installing Bracken"
conda install bioconda::bracken

echo "Installing Krona"
mamba install krona
mamba update krona

echo "Installing Pavian on R"
install.packages("BiocManager")
BiocManager::install("Rsamtools")
install.packages("remotes")
remotes::install_github("fbreitwieser/pavian")
```

G.23 Databases Download Script

```
echo "Download GTDB for sourmash"
wget https://farm.cse.ucdavis.edu/~ctbrown/sourmash-db/gtdb-rs207/gtdb-rs207.genomic.k31.zip
```

```
3
4 echo "Download the Kraken database for Kraken2 and Bracken"
5 wget https://genome-idx.s3.amazonaws.com/kraken/k2_standard_20240605.tar.gz
```