Metagenomics analysis ACFC Wastewater

Floris Menninga

2025-04-03

Contents

Introduction **1**

Pipeline **2**
Snakemake . . . 2
Library / Tool versions: . . . 2
1. Sample preparation . . . 3
1. Quality control . . . 4
1.1 Results . . . 4
2. Taxonomic classification . . . 7
Tool Comparison . . . 7
Visualization of Kraken2 results . . . 8
Bracken . . . 10
DESeq2. . . . 10
Faprotax . . . 11

Discussion: **15**
Citations . . . 16

```r
library(tidyverse)
```

15-03-2025:

Introduction

The wastewater of a Kenyan water treatment plant contains many organisms whose presence influences the water quality by their production of toxins like those produced by algal blooms from Cyanobacteria. (Hart et al. (2025)) These algal blooms are a global threat to freshwater systems like the Nyando River. Using more traditional chemical analyses, an abundance of nitrogen, phosphorus and potassium was determined.

It is imperative to measure those toxins or in this case genetic material from the micro-organisms that produce them. This wastewater is directly exhausted into the Nyando River, a basin covering about 3,590 square kilometers. The population density at the basin is higher than the national average, this indicates it's importance. The life expectancy is very low with an average of 37.7 years for males and 42.9 years for females. Enhancing water quality could contribute to a higher lifespan for the people and other organisms that make use of these wetlands.

This water provides food, stores energy and is crucial for biodiversity. (Obiero et al. (2012)) These resources are threatened by wastewater from factories and treatment plants.

In the wastewater treatment facility is among other process steps a digester, with a lagoon where the wastewater is expelled into. The influence of this digester on the microbial diversity and number will be determined.

There are a total of three samples, the first one was taken before the water enters the digester, the second inside of the digester and the last one in the lagoon before entering the Nyando River. We will make a comparison between these samples and we will determine if the amount of bacteria or other micro-organisms exceeds the limits for reclaimed water before and after entering the digester. Toxin producing algae can be present in the sample that exited the digester only if this was also the case before entering it. The scope of this article is limited to determining the influence of the digester in the wastewater treatment facility. So even if there already are contaminants, as long as their numbers are not increasing, the treatment process is not at fault.

Metagenomics is DNA based and can provide information about what organisms are present in the sample, this can be taxonomic and phylogenetic information. (Hong, Mantilla-Calderon, and Wang (2020)) Metagenomics differs from whole genome sequencing which refers to the sequencing of a single genomes.

To do this, a bio-informatics pipeline was constructed to compare these samples in which a taxonomic classification will be applied on the samples using the Kraken2 tool to check the presence of known algal bloom causing Cyanobacteria. (Hart et al. (2025)) Given the results from incubating the samples on agar plates, we hypothesize that there are toxin producing Cyanobacteria in the samples but that the digester doesn't make a difference for the diversity and number of those organisms.

In addition to the phylogenetic classification, there will also be an anti-biotic resistance test to determine to what antibiotic resistance genes the micro-organisms have. This will provide an overview of means to combat the bacteria found in the sample more effectively. (Lal Gupta, Kumar Tiwari, and Cytryn (2020))

To first assemble the microbial genomes from the metagenomic dataset, several tools can be used like Metabat, Concoct and Maxbin4. Identifying potentially up to 2000 micro-organisms in the sample can be challenging given that their average genome size would be 4 Mpb 8 Gpb of reads would have to be obtained to get an average coverage of 1x (al2015removal). A functional analysis can be done by comparing the microorganisms found using Kraken2 against a database like Faprotax (Terlouw et al. (2023)) This database has for every species/genus the pathways that are know to occur in them.

To-do: 16S DNA explanation.

The diversity of the microbiome will be quantified using the alpha diversity measure. This is a formula that can be applied to calculate this measure for diversity. **6**.

Pipeline

The different steps of the pipeline are explained below.

Overview: 1. Quality control with Fastplong 2. Trimming of adapter sequences with Fastplong 3. Taxonomic classification with Kraken2 4. Visualization of Kraken2 results using Kurona / Pavian 5. Functional analysis of the micro-organisms found

Snakemake

To execute this pipeline Snakemake is used. Snakemake is a workflow management system created to be reproducible and scalable. Workflows are described via a human readable Python based language. Snakemake can also enatail a description of required software, which can be automatically be deplayed to the execution environment.

Library / Tool versions:

Tool	Version	Description
R	4.4.1	Statistical computing and graphics environment
Fastplong	0.2.2	Tool for long-read sequence analysis
Kraken	2.1.2	Taxonomic sequence classification system

Tool	Version	Description
Bracken	3.0.1	Bayesian reestimation of abundance after classification with Kraken
Krona	2.8.1	Interactive metagenomic visualization tool
Pavian	1.0	Interactive browser application for analyzing metagenomics data
kraken-biom	1.2.0	Creates BIOM-format tables from Kraken output
FAPROTAX	1.2.10	Functional annotation of prokaryotic taxa

1. Sample preparation

Samples:

#	Condition
1	In lag, + glyc, - schud
2	In lag, + glyc, + schud
3	In lag, - glyc, - schud
4	In lag, - glyc, + schud
5	Out lag, + glyc, -schud
6	Out lag, + glyc, + schud
7	Out lag, - glyc, - schud
8	Out lag, - glyc, + schud
9	Dig, + glyc, + schud
10	Dig, + gly, + schud
11	Dig, -gly, -schud
12	Dig, - gly, +schud

The three groups: In the lagoon, Outside of the lagoon and in the digester. Some samples have been treated with glycerol in order to better preserve the sample in the -80 freezer.

As per usual with Nanopore results, the samples were distributed over many seperate .fastq files. These were combined using the following bash script:

First, to combine all fastq files for every barcode.

```
cat barcode01/*.fastq.gz > /students/2024-2025/Thema07/metagenomics/wastewater/data/barcode01.fastq
cat barcode02/*.fastq.gz > /students/2024-2025/Thema07/metagenomics/wastewater/data/barcode02.fastq
cat barcode03/*.fastq.gz > /students/2024-2025/Thema07/metagenomics/wastewater/data/barcode03.fastq
cat barcode04/*.fastq.gz > /students/2024-2025/Thema07/metagenomics/wastewater/data/barcode04.fastq
cat barcode05/*.fastq.gz > /students/2024-2025/Thema07/metagenomics/wastewater/data/barcode05.fastq
cat barcode06/*.fastq.gz > /students/2024-2025/Thema07/metagenomics/wastewater/data/barcode06.fastq
cat barcode07/*.fastq.gz > /students/2024-2025/Thema07/metagenomics/wastewater/data/barcode07.fastq
cat barcode08/*.fastq.gz > /students/2024-2025/Thema07/metagenomics/wastewater/data/barcode08.fastq
cat barcode09/*.fastq.gz > /students/2024-2025/Thema07/metagenomics/wastewater/data/barcode09.fastq
cat barcode10/*.fastq.gz > /students/2024-2025/Thema07/metagenomics/wastewater/data/barcode10.fastq
cat barcode11/*.fastq.gz > /students/2024-2025/Thema07/metagenomics/wastewater/data/barcode11.fastq
```

Combining multiple barcodes into three samples according to the group they are in (In lagoon, out lagoon and digester)

```
# Combining all barcodes for the "In lagoon" sample:
cat barcode01.fastq barcode02.fastq barcode03.fastq barcode04.fastq  > /students/2024-2025/Thema07/meta
```

```
# The same was done for the "Out lagoon" sample:
cat barcode05.fastq barcode06.fastq barcode07.fastq barcode08.fastq  > /students/2024-2025/Thema07/meta

# And the "Digester" sample:
cat barcode09.fastq barcode10.fastq barcode11.fastq barcode12.fastq  > /students/2024-2025/Thema07/meta
```

Later, I consolidated these bash commands into two snakemake rules:

```
rule barcode_concatenation:
    input:
        lambda wildcards: glob.glob(config["BARCODE_DIRECTORIES"][wildcards.barcode] + "/*")
    output:
        f"{OUT_DIR}/fastq/{{barcode}}.fastq.gz"
    log:
        f"{LOG_DIR}/cat/{{barcode}}.log"
    shell:
        """
        cat {input} > {output} 2> {log}
        """

rule combining_samples:
    input:
        lambda wildcards: expand("data/fastq/{barcode}.fastq.gz", barcode=config["sample_barcodes"][wil
    output:
        f"{OUT_DIR}/combined_fastq/{{sample}}.fastq"
    log:
        f"{LOG_DIR}/combining_samples/{{sample}}.log"
    shell:
        """
        cat {input} > {output} 2> {log}
        """
```

The first rule concatenates all the fastq files in the barcode directories into one file for every barcode. According to the sample table, all samples taken in the same place where put together ignoring the use of glycerol that was used for some of the samples. It should have no effect on the genetic material in the samples.

1. Quality control

To check the quality of the raw sequence data, Fastplong was used. (Chen (2023)) At first we planned to use Trimmomatic and FastQC like we had done in an earlier analysis but they are made to work with Illumina data while we are working with Nanopore data. Fastplong functionally combines Trimmomatic and FastQC with its adaptertrimming and reports with the quality distribution of the reads, both before and after filtering.

1.1 Results

The sample taken from the digester only contained 2 reads before filtering with Fastplong and 1 after. We decided that it was unusable and all downstream analyses where done without it. The table below shows this in the "Total Reads" row.

Statistic	Before Filtering	After Filtering
Total Reads	2	1
Total Bases	1.873000 K	507
Minimum Length	507	507
Maximum Length	1.366000 K	507
Median Length	1.366000 K	507

Statistic	Before Filtering	After Filtering
Mean Length	936	507
N50 Length	1.366000 K	507
GC Content	54.458089%	51.676529%
Q5 Bases	1.796000 K (95.888948%)	490 (96.646943%)
Q7 Bases	1.631000 K (87.079552%)	457 (90.138067%)
Q10 Bases	1.358000 K (72.504004%)	378 (74.556213%)
Q15 Bases	1.107000 K (59.103043%)	309 (60.946746%)
Q20 Bases	874 (46.663107%)	234 (46.153846%)
Q30 Bases	248 (13.240790%)	62 (12.228797%)
Q40 Bases	0 (0.000000%)	0 (0.000000%)

The sample taken from inside the lagoon has ~18K reads after filtering.

Statistic	Before Filtering	After Filtering
Total Reads	20.242000 K	18.042000 K
Total Bases	28.870993 M	23.485654 M
Minimum Length	3	17
Maximum Length	13.903000 K	13.903000 K
Median Length	1.452000 K	1.320000 K
Mean Length	1.426000 K	1.301000 K
N50 Length	1.453000 K	1.322000 K
GC Content	53.655196%	53.160078%
Q5 Bases	28.257982 M (97.876724%)	23.065575 M (98.211338%)
Q7 Bases	26.944736 M (93.328054%)	22.133635 M (94.243213%)
Q10 Bases	24.666336 M (85.436396%)	20.436812 M (87.018279%)
Q15 Bases	21.538507 M (74.602585%)	18.037896 M (76.803891%)
Q20 Bases	18.308551 M (63.415037%)	15.468003 M (65.861496%)
Q30 Bases	5.359630 M (18.564065%)	4.511358 M (19.208995%)
Q40 Bases	12.277000 K (0.042524%)	10.494000 K (0.044683%)

And the last sample, taken when the water had exited the lagoon. There are only 2.16K reads in this sample after filtering, almost a factor of ten less compared to the “in lagoon” sample. This could have an significant impact on species diversity because there are simply less reads so less chance for a more diverse microbiome.

Statistic	Before Filtering	After Filtering
Total Reads	2.410000 K	2.164000 K
Total Bases	3.400336 M	2.855180 M
Minimum Length	3	17
Maximum Length	2.075000 K	2.075000 K
Median Length	1.435000 K	1.298000 K
Mean Length	1.410000 K	1.319000 K
N50 Length	1.436000 K	1.309000 K
GC Content	54.223112%	53.883748%
Q5 Bases	3.327345 M (97.853418%)	2.802639 M (98.159801%)
Q7 Bases	3.172289 M (93.293398%)	2.687310 M (94.120511%)
Q10 Bases	2.901834 M (85.339625%)	2.477674 M (86.778207%)
Q15 Bases	2.530093 M (74.407147%)	2.181687 M (76.411540%)
Q20 Bases	2.145847 M (63.106911%)	1.865486 M (65.336896%)
Q30 Bases	592.594000 K (17.427513%)	519.403000 K (18.191603%)

Statistic	Before Filtering	After Filtering
Q40 Bases	1.333000 K (0.039202%)	1.168000 K (0.040908%)

In the tables above, the N50 length is the largest lenght L such that 50% of all nucleotides are contained in contigs of size at least L. (source: https://www.nature.com/articles/35057062)

Q5, Q7, Q10 etc. bases is the value of the number of bases at each quality level (Phred score).

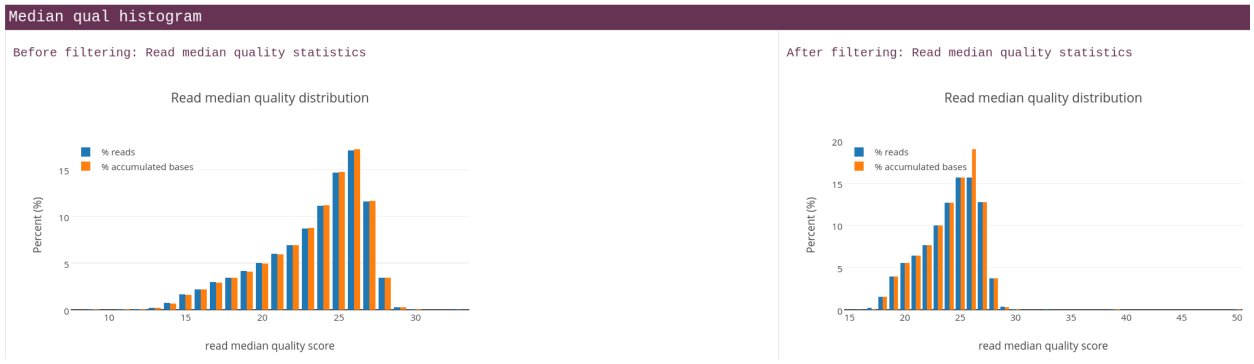


Figure 1: image

The figure above displays the difference in read quality before and after filtering with Fastplong. The amount of reads was reduced but the quality increased. Perhaps the reads with a lower phred score would suffice but I chose to remove them in order to have more statistical power in the downstream analyses like in Kraken2.

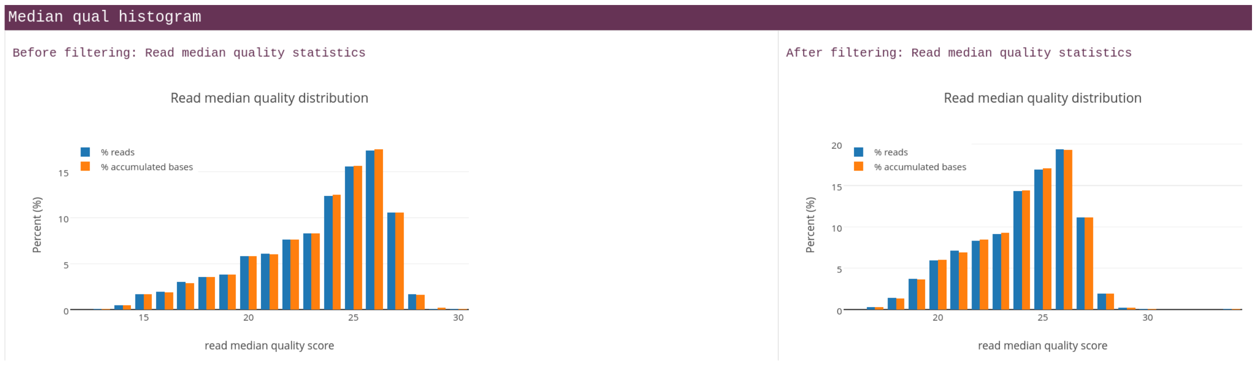


Figure 2: image

The same was done with the sample from the water that had exited the lagoon. The digester sample contained only one read after filtering so displaying this graph would be meaningless.

The rule in snakemake takes all the samples from the config.yaml used for all the rules. FastPlong generates both a .html report for the quality control and a fastq file that has been trimmed. (adapter sequences removed and low quality reads removed) The standard error is redirected to the logs/QC/ directory for every individual sample.

```
rule fastq_qc_plong:
    input:
        lambda wildcards: config["samples"][wildcards.sample]
    output:
        fastq="trimmed/{sample}.fastq",
        html="QC/{sample}_fastplong_QC.html"
    log:
        "logs/QC/{sample}.log"
    shell:
        """
        tools/fastplong \
        -i {input} \
        -o {output.fastq} \
        -h {output.html} \
        2> {log}
        """
```

2. Taxonomic classification

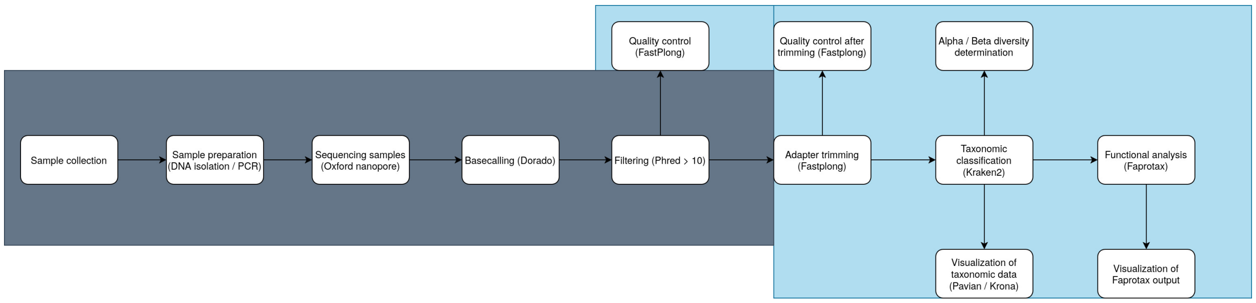
To classify all the reads as belonging to a species or less specific taxon, Kraken2 was used.

```
rule kraken2_taxonomic_classification:
    input:
        reads="trimmed/{sample}.fastq"
    output:
        kraken_report="kraken2/reports/{sample}_report.txt",
        output="kraken2/output/{sample}_output.txt"

    log:
        "logs/kraken2/{sample}.log"
    params:
        db="/data/datasets/KRAKEN2_INDEX/16S_Greengenes",
        confidence="1",
        threads=64
    conda:
        "envs/kraken2.yaml"
    shell:
        """
        kraken2 --db {params.db} \
                --threads {params.threads} \
                --confidence {params.confidence} \
                --output {output.output} \
                --report {output.kraken_report} \
                {input.reads} 2> {log}
        """
```

In the snakemake rule above, kraken2 is run with the parameters: database, confidence and threads. The database used is Greengenes, this is a database for use with 16S reads. Confidence is kept at its default value of 0 and 128 threads are used for this operation. The 16S Greengenes database was used for this. DeSantis et al. (2006)

Tool Comparison

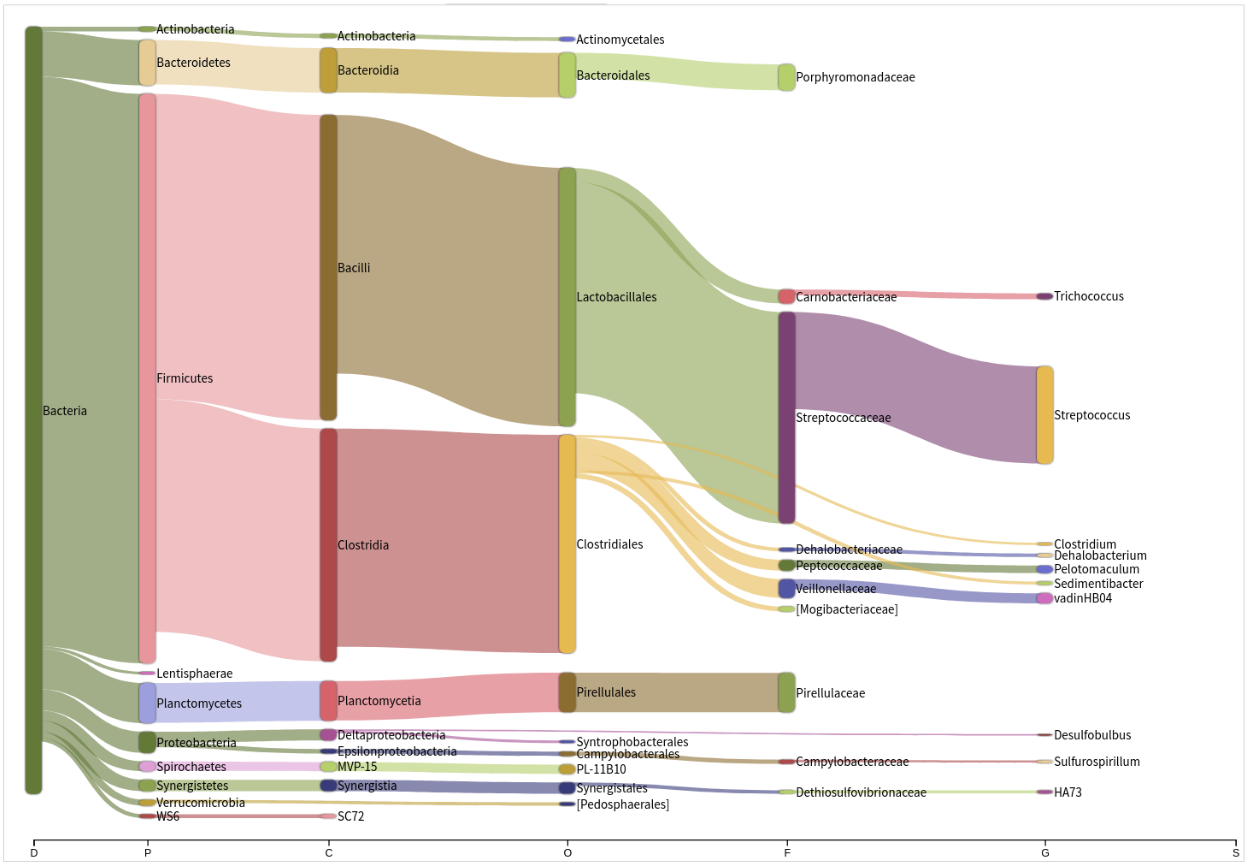
This tools was chosen based on a comparison made in the following article, (Ye et al. (2019)) Kraken2 performed comparable to similar tools like MEGAN-N, Minimap2 and Kaiju. The reason we choose this tool over the other ones is that there are more tools that can use Kraken2's output compared to many of the others. A close contender was Humann3 for taxonomic classification and metaphlan4 for functional analysis but since the nessasary databases for Kraken2 were already on the system, we setteled on Kraken2 and Faprotax for the functional analysis.


Sample collection
Sample preparation (DNA isolation / PCR)
Sequencing samples (Oxford nanopore)
Basecalling (Dorado)
Filtering (Phred > 10)
Quality control (FastPlong)
Adapter trimming (Fastplong)
Quality control after trimming (Fastplong)
Taxonomic classification (Kraken2)
Alpha / Beta diversity determination
Functional analysis (Faprotax)
Visualization of taxonomic data (Pavian / Krona)
Visualization of Faprotax output

The workflow diagram displays the analyses that where not done by us in gray and those that where done by us in light blue.

Visualization of Kraken2 results

Pavian was used to generate a Sankey graph based on the report from Kraken2.



There are many more different microorganisms found in the “In lagoon” sample compared to the “Out lagoon” sample.

Both charts show that most microorganisms are bacteria, specifically Firmicutes. This is a phylum of mostly gram-positive bacteria.

Bracken was used to compute the abundance of species. When using 16S reads, this could not be done on a species level so the genus level was used.

When comparing the output of Kraken2 with Bracken, many taxa where lost after running Bracken, this is why its output was discarded and the raw Kraken data was used for the downstream analyses.

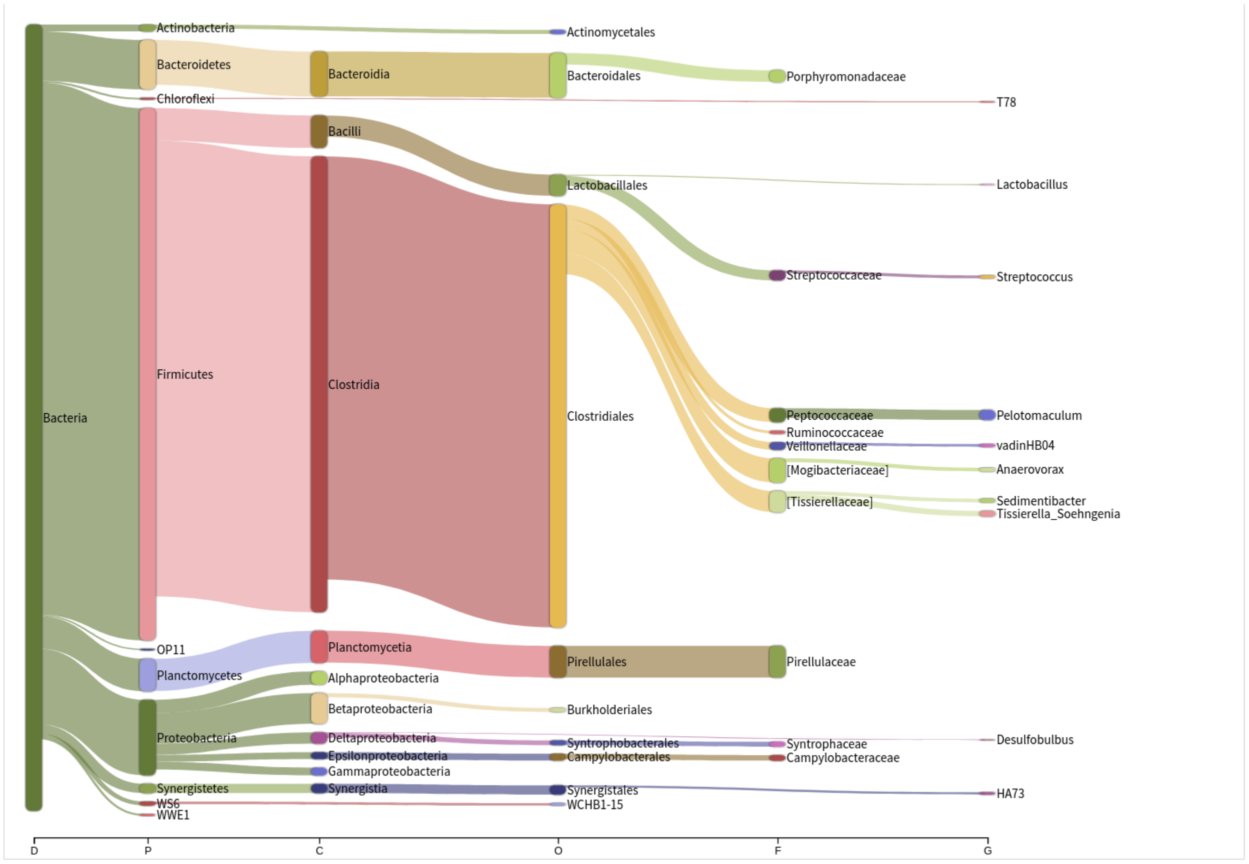


Figure 3: Out Lagoon

Bracken

Bracken (Bayesian Reestimation of Abundance with KrakEN) is a statistical method that computes the abundance of species in DNA sequences from a metagenomics sample. Braken uses taxonomy labels that are assigned by Kraken2 to estimate the number of reads that originated from each species in the sample. Even though Kraken classifies reads to the best matching location in the taxonomic tree, it does not estimate the abundances of species. Bracken uses the Kraken database to derive probabilities that describe how much sequence from each genome is the same to other genomes in the database. This data is combined with with the assignments for the sample to estimate the abundance at the species level or in our case, the genus level.

The tool did not work when “S” for species was given as argument. After reading about researchers having the same problem and figuring out that species level estimation is not possible with 16S data, we used “G” to let bracken estimate it to the genus level instead.

After using Bracken, it became apparent that many species got filtered out that were there before and since species level estimation is not possible, we decided to omit Bracken from the pipeline. Yamila has a comparison of the bracken/ no bracken outputs in her Notebook.

```
rule bracken:
    input:
        kraken_report=rules.kraken2_taxonomic_classification.output.kraken_report,
        kraken_database=f"{config['kraken2_db_dir']}/16S_Greengenes"
    output:
        bracken_report=f"{OUT_DIR}/bracken/reports/{{sample}}.txt",
        bracken_output=f"{OUT_DIR}/bracken/output/{{sample}}.out"
    params:
        read_length=150,
        threshold=10,
        level="G"
    log:
        f"{LOG_DIR}/bracken/{{sample}}.log"
    threads:
        32
    conda:
        workflow.source_path("../envs/bracken.yaml")
    shell:
        """
        bracken \
            -d {input.kraken_database} \
            -i {input.kraken_report} \
            -o {output.bracken_output} \
            -w {output.bracken_report} \
            -r {params.read_length} \
            -t {params.threshold} \
            -l {params.level} \
        2> {log}
        """
```

The Snakemake rule uses bracken with a read length of 150, a phred threshold of 10 and estimation at genus level. The threshold is the default value of 10. The read length should require some experimentation because the average read length of the samples varies a lot. But there was not enough time to do this and almost all reads conform to this length.

DESeq2. . . .

At first, I thought that DESeq2 could be insightful to compare the count data but after looking into it more, the statistical method used would not be usable for metagenomics data since the distribution is different than what DESeq2 expects.

This means that the results would be statistical insignificant and not worth the time investment to adapt our data for use with it. The code below is my fruitless attempt at using Deseq2.

```
colnames <- c("Group")

coldata <- data.frame(c("In_Lagoon", "Out_Lagoon"), c("In_Lagoon","Out_Lagoon"), row.names = 1)
colnames(coldata) <- colnames

coldata$Group <- factor(coldata$Group)
```

```
dds <- DESeqDataSetFromMatrix(countData = counts_df,
                              colData = coldata,
                              design= ~ Group)

dds <- DESeq(dds)

resultsNames(dds)

res <- results(dds, name="In_vs_Out_lagoon")

res <- lfcShrink(dds, coef="In_vs_Out_lagoon", type="apeglm")

summary(res)
```

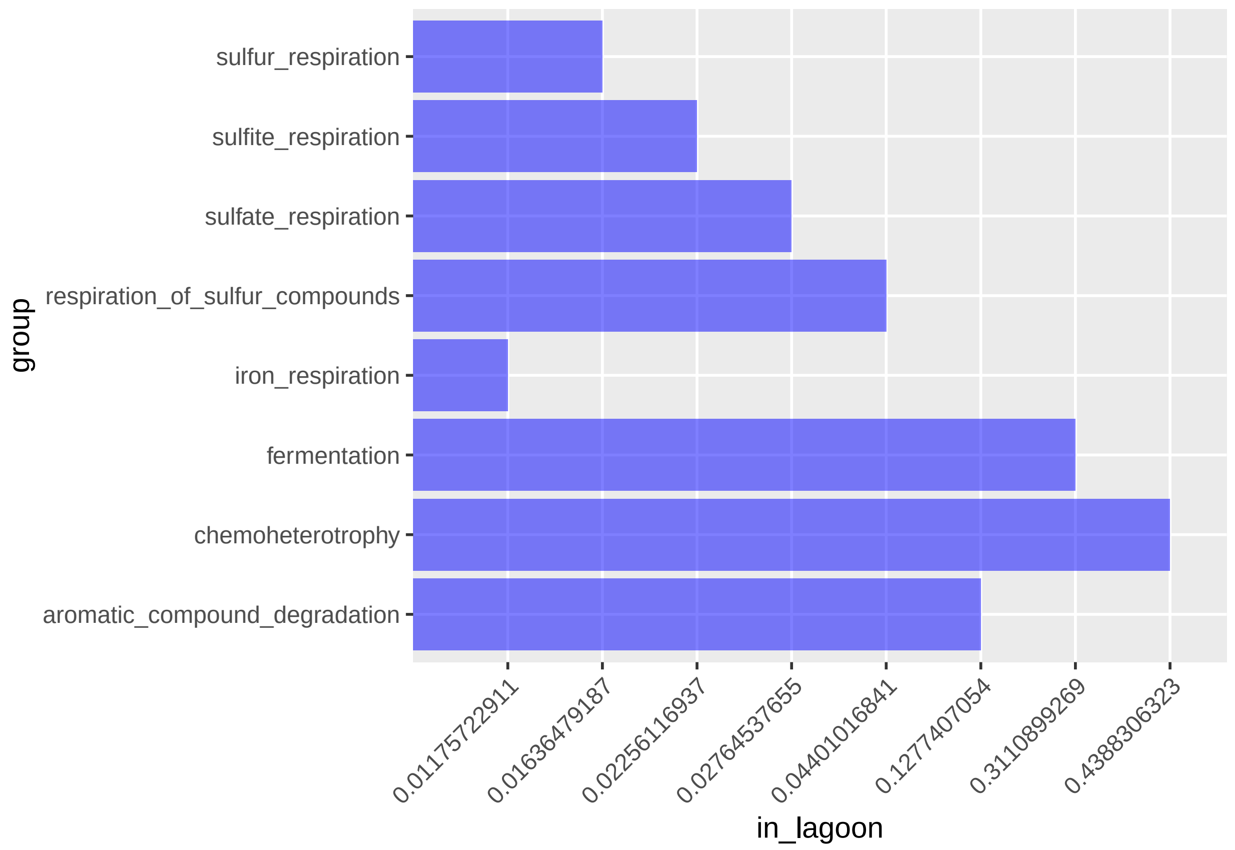
Faprotax

After Kraken2 had classified the reads and the content of the sample became known, Faprotax was used to look up what known pathways are used by the microorganisms. The Faprotax (Functional Annotation of Prokaryotic Taxa) database is a database that maps prokaryotic clades, for example genera and species. This is done in order to find out what ecologically relevant metabolic functions the organism possesses.

FAPROTAX maintains a database of Over 7600 functional annotations covering over 4600 taxa. (Louca, Parfrey, and Doebeli (2016)) Faprotax includes a tool that functions as a loop up table for the organisms from the OTU table gathered from Kraken2 and compares this to known pathways.

There are other databases that could be used that have a broader scope compared to the mostly ecological pathways from Faprotax. I have found MinPath and Tax4fun2 which are functional profilers for 16S rRNA sequences like we have. In the end, due to time constrains only Faprotax was used.

```
faprotax_in_lagoon <- read.csv2("~/Documenten/Data_set/ACFC_Wastewater_Metagenomics/faprotax/in_lagoon/

faprotax_in_lagoon <- faprotax_in_lagoon %>% filter(in_lagoon > 0)

ggplot(data = faprotax_in_lagoon, mapping = aes(x = group, y = in_lagoon)) +
    geom_bar(stat="identity", fill = "blue", alpha=0.5) +
    theme(axis.text.x=element_text(angle=45, hjust=1)) +
    coord_flip()
```

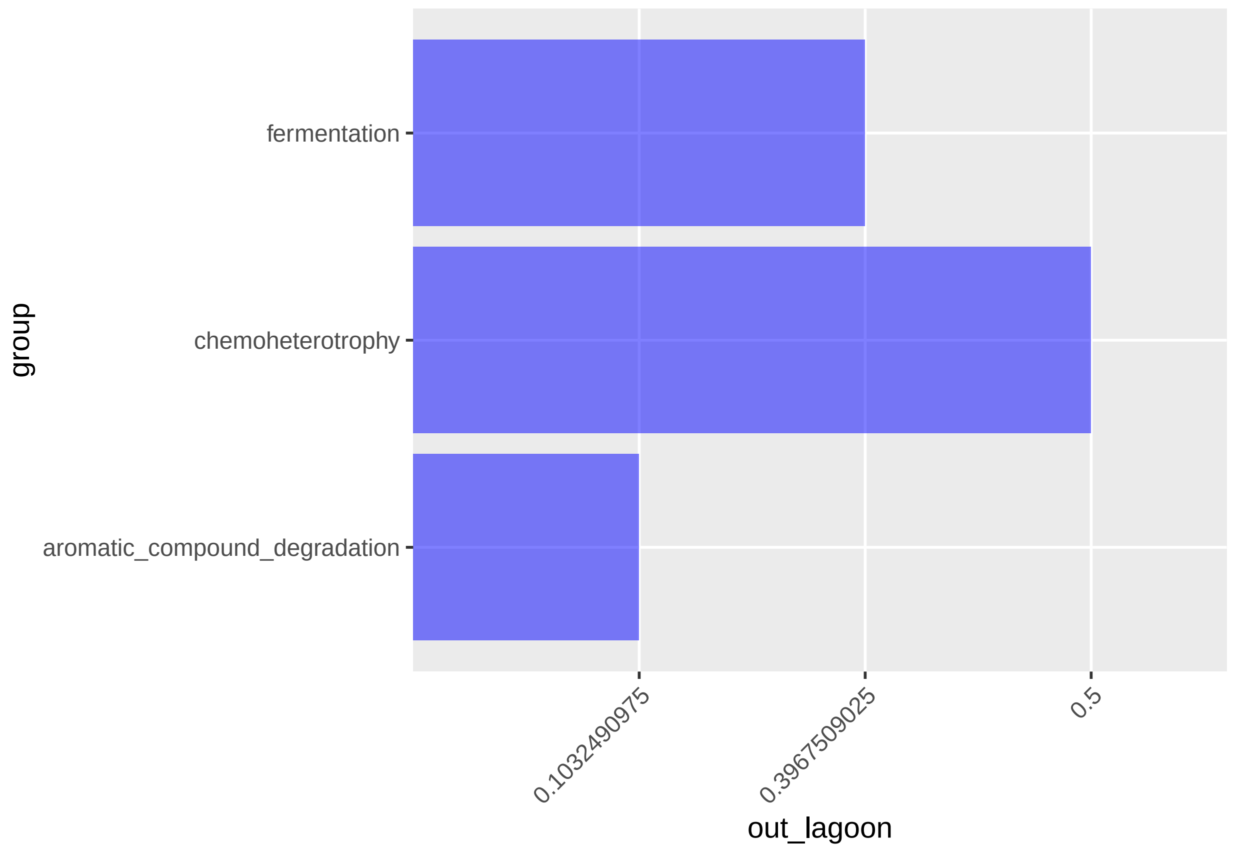


The graph above displays the results of the FAPROTAX analysis.

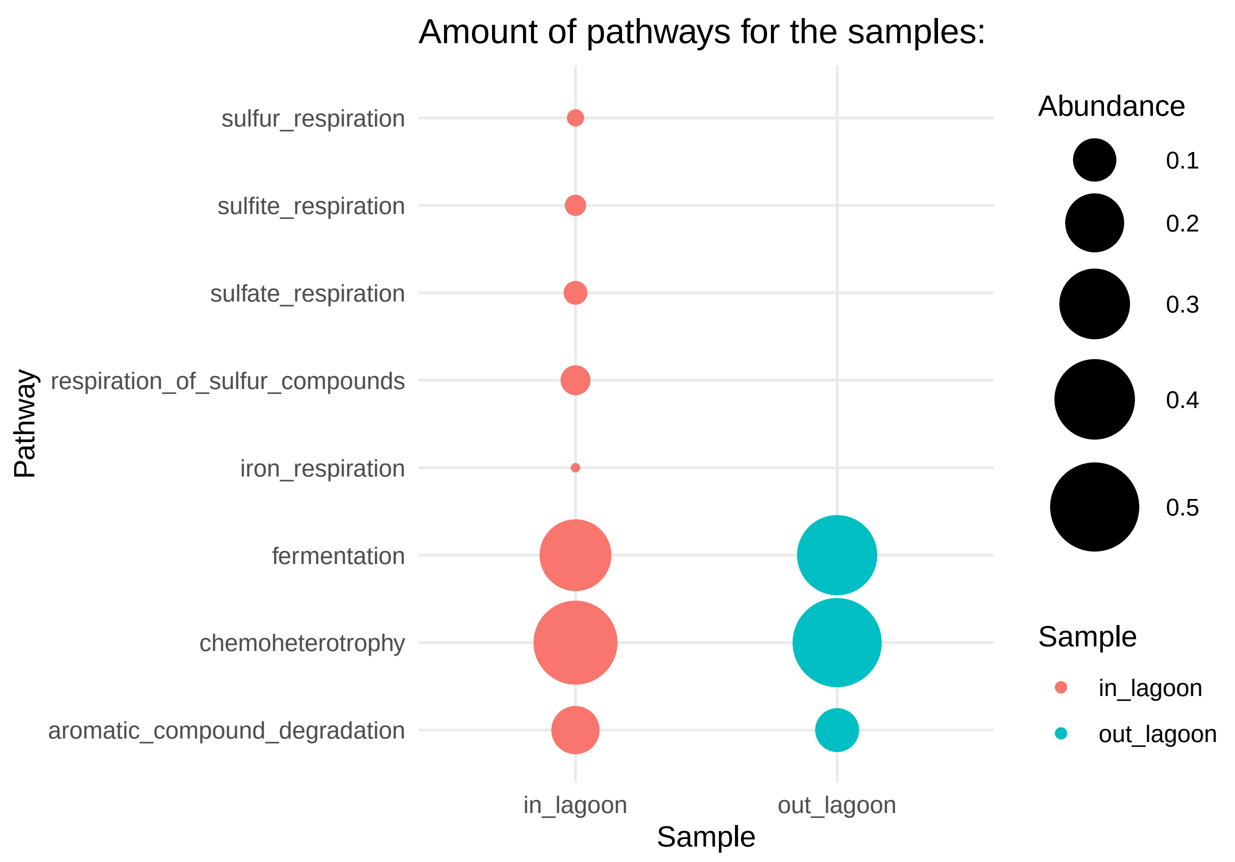
One of FAPROTAX's applications is the ecological interpretation of 16S marker gene data like what is done for this study. FAPROTAX is non-exhaustive, that means it is likely that many organisms known to perform certain functions may be missing or may only be partially included in the database. This could explain the few pathways found in both samples but especially in the "out lagoon" sample below.

(Source: https://pages.uoregon.edu/slouca/LoucaLab/archive/FAPROTAX/lib/php/index.php)

```
faprotax_out_lagoon <- read.csv2("~/Documenten/Data_set/ACFC_Wastewater_Metagenomics/faprotax/out_lagoo

faprotax_out_lagoon <- faprotax_out_lagoon %>% filter(out_lagoon > 0)

ggplot(data = faprotax_out_lagoon, mapping = aes(x = group, y = out_lagoon)) +
    geom_bar(stat="identity", fill = "blue", alpha=0.5) +
    theme(axis.text.x=element_text(angle=45, hjust=1)) +
    coord_flip()
```

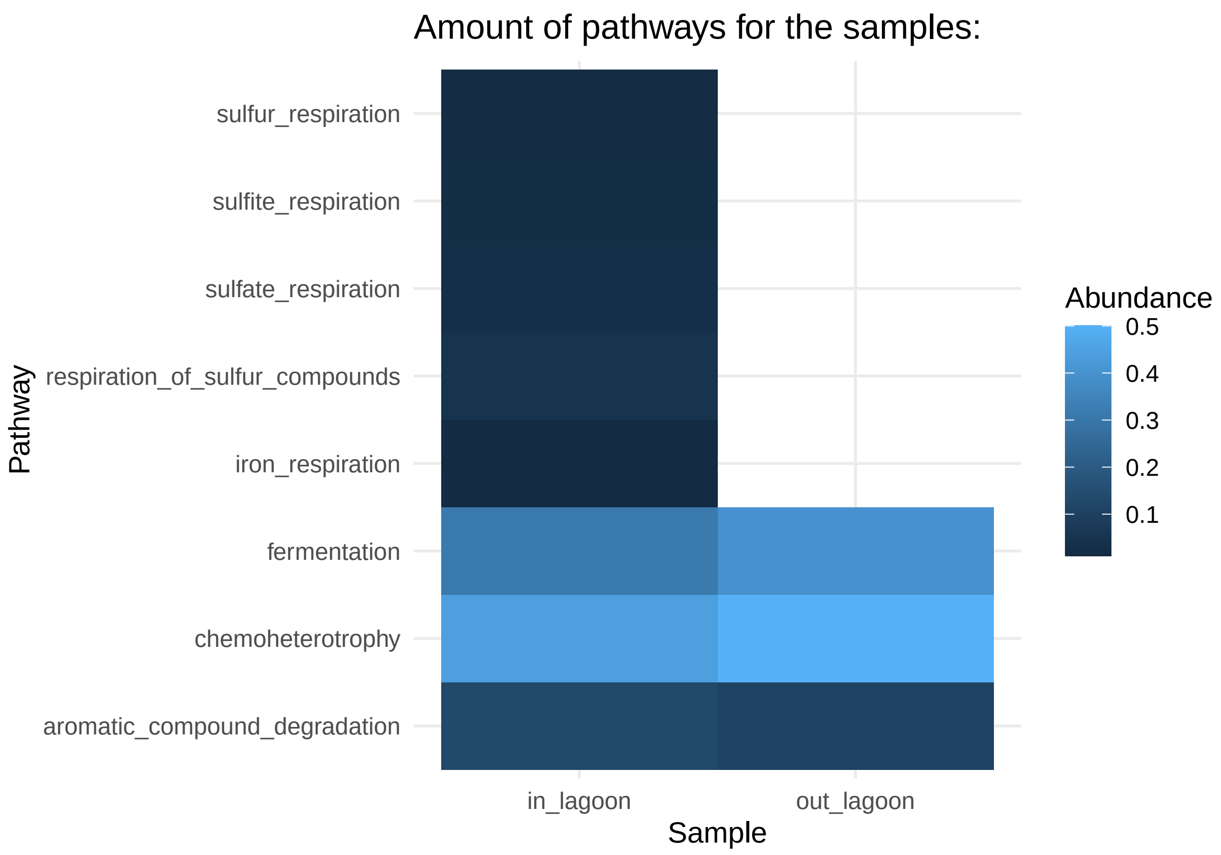


```r
faprotax_out_lagoon <- read.table("~/Documenten/Data_set/ACFC_Wastewater_Metagenomics/faprotax/out_lago
faprotax_in_lagoon <- read.table("~/Documenten/Data_set/ACFC_Wastewater_Metagenomics/faprotax/in_lagoon/

combination_in_out_lagoon <- data.frame(cbind(faprotax_out_lagoon, faprotax_in_lagoon))

data_long <- combination_in_out_lagoon %>% # Make it into long format for the plot.
    rownames_to_column("Pathway") %>%
    pivot_longer(cols = -Pathway,
                 names_to = "Sample",
                 values_to = "Abundance")

data_long <- data_long %>% filter(Abundance > 0) # Filter the 0 values.

ggplot(data_long, aes(x = Sample, y = Pathway, size = Abundance)) +
    geom_point(aes(color=Sample)) +
    scale_size(range = c(1, 15), name = "Abundance") + # Looks better with scaling...
    theme_minimal() +
    labs(title = "Amount of pathways for the samples:",x = "Sample",y = "Pathway")
```



The bubble chart above looks anemic with only a few pathways. I had to filter the data_long to only include pathways with an abundance of more than 0.

```r
ggplot(data_long, aes(x = Sample, y = Pathway, fill = Abundance)) +
    geom_tile() +
    theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
    labs(title = "Amount of pathways for the samples:", x = "Sample", y = "Pathway") +
    theme_minimal()
```



The same data visualized with a heatmap. There is only overlap between the samples for three pathways so it might not be the most informative graph.

1-04-2025:

Discussion:

The main objective of this study was to identify the microorganisms present in the digester, lagoon in and lagoon out samples . Due to sequencing problems, the digester sample only contained two reads, one of which was remove during the trimming with Fastplong. This sample was unusable for further analyses.

Our results demonstrate that the sample taken in the lagoon has the most diverse microbial community with an alpha diversity of … and a beta diversity between the two samples of … This difference can be partially attributed to a large difference in reads between the two samples. The sample taken from inside the lagoon has ~18K reads after filtering while the sample from the water that had exited the lagoon only contains ~2.2K reads.

Kraken2 showed that most microorganisms in both samples are bacteria, specifically Firmicutes. This is a phylum of mostly gram-positive bacteria. Bracken was used to compute the abundance of species. When using 16S reads, this could not be done on a species level so the genus level was used. When comparing the output of Kraken2 with Bracken, many taxa where lost after running Bracken, this is why its output was discarded and the raw Kraken data was used for the downstream analyses.

The data gathered from Kraken depended heavily on what database was used.

Faprotax was used to determine some of the pathways these microorganisms use. For the "out lagoon" sample, there where only three pathways found: fermentation, chemotherotropy and aromatic compound degradation. The "in lagoon" sample had in addition to these pathways the following respiration pathways: sulfur, sufite, sulfate and iron respiration. This larger amount of pathways could also be attributed to the difference in reads between the samples. Perhaps it should have been normalized. None of the pathways indicate the production of toxins but the database used Faprotax has a limited scope and mostly features enviromental pathways and not

Unit tests where generated for all the snakemake rules: "snakemake –generate-unit-tests"

Citations

1. https://www.zotero.org/groups/5870913/acfc_afvalwater/items/WLVUDDXJ/attachment/RHY85NTI/reader
2. https://www.zotero.org/groups/5870913/acfc_afvalwater/items/DW2JTKTX/reader
3. biosynthetic gene clusters. Nucleic Acids Res 51:D603–D610. https:// doi.org/10.1093/nar/gkac1049
4. https://www.zotero.org/groups/5870913/acfc_afvalwater/items/F3R5L7PV/attachment/6YZYPRXC/reader
5. is the treated wastewater safe to reuse for agricultural irrigation? Water Res 73:277–290. https://doi.org/10.1016/
6. Best practices for analysing microbiomes. Nat Rev Microbiol 16:410 – 422. https://doi.org/10.1038/s41579-018-0029-9.
7. Platforms for elucidating antibiotic resistance in single genomes and complex metagenomes. Environ Int 138:105667. https://doi.org/10.1016/j.envint.2020.105667.

Fastplong Shifu Chen. 2023. Ultrafast one-pass FASTQ data preprocessing, quality control, and deduplication using fastp. iMeta 2: e107. https://doi.org/10.1002/imt2.107

Kraken2 visualization script: https://github.com/lorenzgerber/krakenSankey

```r
build_sankey_network <- function(my_report, taxRanks =  c("U","D","K","P","C","O","F","G","S"), maxn=10
                 zoom = F, title = NULL,
                 ...) {
    stopifnot("taxRank" %in% colnames(my_report))
    if (!any(taxRanks %in% my_report$taxRank)) {
        warning("report does not contain any of the taxRanks - skipping it")
        return()
    }
    my_report <- subset(my_report, taxRank %in% taxRanks)
    my_report <- plyr::ddply(my_report, "taxRank", function(x) x[utils::tail(order(x$cladeReads,-x$deptl

    my_report <- my_report[, c("name","taxLineage","taxonReads", "cladeReads","depth", "taxRank")]

    my_report <- my_report[!my_report$name %in% c('-_root'), ]
    #my_report$name <- sub("^-_root.", "", my_report$name)

    splits <- strsplit(my_report$taxLineage, "\\|")

    ## for the root nodes, we'll have to add an 'other' link to account for all cladeReads
    root_nodes <- sapply(splits[sapply(splits, length) ==2], function(x) x[2])

    sel <- sapply(splits, length) >= 3
    splits <- splits[sel]

    links <- data.frame(do.call(rbind,
                                lapply(splits, function(x) utils::tail(x[x %in% my_report$name], n=2)))
    colnames(links) <- c("source","target")
```

```r
links$value <- my_report[sel,"cladeReads"]

my_taxRanks <- taxRanks[taxRanks %in% my_report$taxRank]
taxRank_to_depth <- stats::setNames(seq_along(my_taxRanks)-1, my_taxRanks)

nodes <- data.frame(name=my_report$name,
                    depth=taxRank_to_depth[my_report$taxRank],
                    value=my_report$cladeReads,
                    stringsAsFactors=FALSE)

for (node_name in root_nodes) {
  diff_sum_vs_all <- my_report[my_report$name == node_name, "cladeReads"] - sum(links$value[links$so
  if (diff_sum_vs_all > 0) {
    nname <- paste("other", sub("^._","",node_name))
    #nname <- node_name
    #links <- rbind(links, data.frame(source=node_name, target=nname, value=diff_sum_vs_all, string
    #nodes <- rbind(nodes, nname)
  }
}

names_id = stats::setNames(seq_len(nrow(nodes)) - 1, nodes[,1])
links$source <- names_id[links$source]
links$target <- names_id[links$target]
links <- links[links$source != links$target, ]

nodes$name <- sub("^._","", nodes$name)
links$source_name <- nodes$name[links$source + 1]

if (!is.null(links))
  sankeyD3::sankeyNetwork(
    Links = links,
    Nodes = nodes,
    doubleclickTogglesChildren = TRUE,
    Source = "source",
    Target = "target",
    Value = "value",
    NodeID = "name",
    NodeGroup = "name",
    NodePosX = "depth",
    NodeValue = "value",
    dragY = TRUE,
    xAxisDomain = my_taxRanks,
    numberFormat = "pavian",
    title = title,
    nodeWidth = 15,
    linkGradient = TRUE,
    nodeShadow = TRUE,
    nodeCornerRadius = 5,
    units = "cladeReads",
    fontSize = 12,
    iterations = maxn * 100,
    align = "none",
    highlightChildLinks = TRUE,
        orderByPath = TRUE,
        scaleNodeBreadthsByString = TRUE,
        zoom = zoom,
    ...
      )
}

# vim: noai:ts=2:sw=2

## recursive function that transforms the kraken dataframe into a cascading list
build_kraken_tree <- function(report) {
  if (nrow(report) == 0 || nrow(report) == 1) {
    # this should only happen if the original input to the function has a size <= 1
    return(list(report))
  }

  ## select the current depth as the one of the topmost data.frame row
  sel_depth <- report[,'depth'] == report[1,'depth']

  ## partition the data.frame into parts with that depth
  depth_partitions <- cumsum(sel_depth)

  ## for each depth partition
  res <- lapply(unique(depth_partitions),
              function(my_depth_partition) {
                sel <- depth_partitions == my_depth_partition

                ## return the data.frame row if it is only one row (leaf node, ends recursion)
                if (sum(sel) == 1)
                  return(report[sel,,drop=F])

                ## otherwise: take first row as partition descriptor ..
                first_row <- which(sel)[1]
                ##  and recurse deeper into the depths with the remaining rows
                dres <- build_kraken_tree(report[which(sel)[-1],,drop=F])

                attr(dres,"row") <- report[first_row,,drop=F]
                dres
              })
  names(res) <- report$name[sel_depth]
  res
}


## Collapse taxonomic taxRanks to only those mentioned in keep_taxRanks
collapse.taxRanks <- function(krakenlist,keep_taxRanks=LETTERS,filter_taxon=NULL) {
  ## input: a list, in which each element is either a
  ##            a list or a data.frame (for the leafs)
  ##   the input has an attribute row that gives details on the current taxRank

  ## columns whose values are added to the next taxRank when
  ##  a taxRank is deleted
  cols <- c("taxonReads","n_unique_kmers","n_kmers")
```

```r
if (length(krakenlist) == 0 || is.data.frame(krakenlist)) {
  return(krakenlist)
}

parent_row <- attr(krakenlist,"row")
all.child_rows <- c()

if (is.null(parent_row)) {
  return(do.call(rbind,lapply(krakenlist,collapse.taxRanks,keep_taxRanks=keep_taxRanks,filter_taxon=fi
}

## rm.cladeReads captures the number of cladeReads that are deleted.
##  this has to be propagated to the higher taxRank
rm.cladeReads <- 0

for (kl in krakenlist) {
  if (is.data.frame(kl)) {  ## is a leaf node?
    child_rows <- kl
  } else {                 ## recurse deeper into tree
    child_rows <- collapse.taxRanks(kl,keep_taxRanks,filter_taxon=filter_taxon)
    if ('rm.cladeReads' %in% names(attributes(child_rows))) {
      rm.cladeReads <- rm.cladeReads + attr(child_rows,'rm.cladeReads')
    }
  }

  ## check if this taxRank and the taxRanks below should be removed
  delete.taxon <- child_rows[1,'name'] %in% filter_taxon
  if (delete.taxon) {
    rm.cladeReads <- rm.cladeReads + child_rows[1,'cladeReads']
    dmessage(sprintf("removed %7s cladeReads, including %s childs, for %s",child_rows[1,'"cladeReads"

    ## remove all children
    child_rows <- NULL

  } else {

    ## check if the last (top-most) row should be kept
    keep_last.child <- child_rows[1,'taxRank'] %in% keep_taxRanks

    if (!keep_last.child) {
      cols <- cols[cols %in% colnames(parent_row)]

      ## save the specified colum information to the parent
      parent_row[,cols] <- parent_row[,cols] + child_rows[1,cols]

      ## remove row
      child_rows <- child_rows[-1,,drop=FALSE]

      ## decrease depths of rows below child row
      if (nrow(child_rows) > 0)
        child_rows[,'depth'] <- child_rows[,'depth'] - 1

    }
    }
    all.child_rows <- rbind(all.child_rows,child_rows)
  }

  ## subtract deleted read count from parent row
  parent_row[,'cladeReads'] <- parent_row[,'cladeReads'] - rm.cladeReads
  res <- rbind(parent_row,all.child_rows)

  if (parent_row[,'cladeReads'] < 0)
    stop("mistake made in removing cladeReads")
  #if (parent_row[,'"cladeReads"'] == 0)
  #  res <- c()

  if (rm.cladeReads > 0)
    attr(res,'rm.cladeReads') <- rm.cladeReads
  return(res)
}

delete_taxRanks_below <- function(report,taxRank="S") {
  del_taxRank <- 0
  do_del <- FALSE
  del_row <- 0

  cols <- c("taxonReads","n_unique_kmers","n_kmers")
  sub.sums <- c(0,0,0)

  rows_to_delete <- c()
  for (i in seq_len(nrow(report))) {
    if (report[i,'taxRank'] %in% taxRank) {
      del_depth <- report[i,'depth']
      do_del <- TRUE
      del_row <- i
      sub.sums <- c(0,0,0)
    } else {
      if (do_del) {
        if (report[i,'depth'] > del_taxRank) {
          rows_to_delete <- c(rows_to_delete,i)
          sub.sums <- sub.sums + report[i,cols]
        } else {
          report[del_row,cols] <- report[del_row,cols]+sub.sums
          sub.sums <- c(0,0,0)
          do_del <- FALSE
        }
      }
    }
  }
  report[-rows_to_delete,]
}

#' Read kraken or centrifuge-style report
#'
#' @param myfile kraken report file
#' @param collapse  should the results be collapsed to only those taxRanks specified in keep_taxRanks?
#' @param keep_taxRanks taxRanks to keep when collapse is TRUE
#' @param min.depth minimum depth
#' @param filter_taxon filter certain taxon names
#' @param has_header if the kraken report has a header or not
#' @param add_taxRank_columns if TRUE, for each taxRank columns are added
#'
#' @return report data.frame
#' @export
#'
read_report2 <- function(myfile,collapse=TRUE,keep_taxRanks=c("D","K","P","C","O","F","G","S"),min.dept
                         has_header=NULL,add_taxRank_columns=FALSE) {

  first.line <- readLines(myfile,n=1)
  isASCII <-  function(txt) all(charToRaw(txt) <= as.raw(127))
  if (!isASCII(first.line)) {
    dmessage(myfile," is no valid report - not all characters are ASCII")
    return(NULL)
  }
  if (is.null(has_header)) {
    has_header <- grepl("^[a-zA-Z]",first.line)
  }

  if (has_header) {
    report <- utils::read.table(myfile,sep="\t",header = T,
                                quote = "",stringsAsFactors=FALSE, comment.char="#")
    #colnames(report) <- c("percentage","cladeReads","taxonReads","taxRank","taxID","n_unique_kmers","n

    ## harmonize column names. TODO: Harmonize them in the scripts!
    colnames(report)[colnames(report)=="clade_perc"] <- "percentage"
    colnames(report)[colnames(report)=="perc"] <- "percentage"

    colnames(report)[colnames(report)=="n_reads_clade"] <- "cladeReads"
    colnames(report)[colnames(report)=="n.clade"] <- "cladeReads"

    colnames(report)[colnames(report)=="n_reads_taxo"] <- "taxonReads"
    colnames(report)[colnames(report)=="n.stay"] <- "taxonReads"

    colnames(report)[colnames(report)=="rank"] <- "taxRank"
    colnames(report)[colnames(report)=="tax_rank"] <- "taxRank"

    colnames(report)[colnames(report)=="taxonid"] <- "taxID"
    colnames(report)[colnames(report)=="tax"] <- "taxID"

  } else {
    report <- utils::read.table(myfile,sep="\t",header = F,
                                col.names = c("percentage","cladeReads","taxonReads","taxRank","taxID",
                                quote = "",stringsAsFactors=FALSE, comment.char="#")
  }

  report$depth <- nchar(gsub("\\S.*","",report$name))/2
  report$name <- gsub("^ *","",report$name)
  report$name <- paste(tolower(report$taxRank),report$name,sep="_")
  ## Only stop at certain taxRanks
  ## filter taxon and further up the tree if 'filter_taxon' is defined
  kraken.tree <- build_kraken_tree(report)
  report <- collapse.taxRanks(kraken.tree,keep_taxRanks=keep_taxRanks,filter_taxon=filter_taxon)

  ## Add a metaphlan-style taxon string
  if (add_taxRank_columns) {
    report[,keep_taxRanks] <- NA
  }
  report$taxLineage = report$name
  rows_to_consider <- rep(FALSE,nrow(report))

  for (i in seq_len(nrow(report))) {
    ## depth > 2 correspond to taxRanks below 'D'
    if (i > 1 && report[i,"depth"] > min.depth) {
      ## find the maximal index of a row below the current depth
      idx <- report$depth < report[i,"depth"] & rows_to_consider
      if (!any(idx)) { next() }

      current.taxRank <- report[i,'taxRank']
      my_row <- max(which(idx))
      report[i,'taxLineage'] <- paste(report[my_row,'taxLineage'],report[i,'taxLineage'],sep="|")

      if (add_taxRank_columns) {
        if (report[my_row,'taxRank'] %in% keep_taxRanks) {
          taxRanks.cp <- keep_taxRanks[seq(from=1,to=which(keep_taxRanks == report[my_row,'taxRank']))]
          report[i,taxRanks.cp] <- report[my_row,taxRanks.cp]
        }

        report[i,report[i,'taxRank']] <- report[i,'name']
      }
    }
    rows_to_consider[i] <- TRUE
  }

  report <- report[report$depth >= min.depth,]

  report$percentage <- round(report$cladeReads/sum(report$taxonReads),6) * 100

    for (column in c("taxonReads", "cladeReads"))
    if (all(floor(report[[column]]) == report[[column]]))
        report[[column]] <- as.integer(report[[column]])

  if ('n_unique_kmers'  %in% colnames(report))
    report$kmerpercentage <- round(report$n_unique_kmers/sum(report$n_unique_kmers,na.rm=T),6) * 100
  #report$taxRankperc <- 100/taxRank(report$cladeReads)

  rownames(report) <- NULL

  report
}
```

```
#' Filter lines from a kraken report result based on the taxonomy name
#'
#' It updates the read_stay counts, and removes any children below the
#' entry, and any parent entries that have no "cladeReads" that stay
#'
#' @param report Report \code{data.frame}.
#' @param filter_taxon Name of entry to remove.
#' @param rm_clade If \code{TRUE}, remove all cladeReads at and below clade, otherwise just set the num
#' @param do_message If \code{TRUE}, report how many rows and cladeReads were deleted.
#'
#' @return filtered report
#' @export
filter_taxon <- function(report, filter_taxon, rm_clade = TRUE, do_message=FALSE) {
  taxon_depth <- NULL
  taxonReads <- 0

  pos.taxons <- which(sub("._","",report$name) %in% filter_taxon)
  #pos.taxon <- which(report$name := filter_taxon)
  if (length(pos.taxons) == 0) {
    return(report)
  }

  row_seq <- seq_len(nrow(report))
  rows_to_delete <- rep(FALSE,nrow(report))

  taxon_depths <- report[pos.taxons,"depth"]
  if (isTRUE(rm_clade)) {
    taxonReads <- report[pos.taxons,"cladeReads"]
  } else {
    taxonReads <- report[pos.taxons,"taxonReads"]
    report[pos.taxons,"taxonReads"] <- 0
  }


  for (i in seq_along(pos.taxons)) {
    pos.taxon <- pos.taxons[i]
    if (pos.taxon == 1) {
      rows_to_delete[1] <- TRUE
      next
    }
    taxon_depth <- taxon_depths[i]
    taxonReads <- taxonReads[i]

    if (rm_clade) {
      tosum_below <-  row_seq >= pos.taxon & report$depth <= taxon_depth
      taxons_below <- cumsum(tosum_below) == 1
      rows_to_delete[taxons_below] <- TRUE
    }
    rows_to_update <- c(pos.taxon)

    taxons_above <- seq_len(nrow(report)) < pos.taxon & report$depth == taxon_depth

    any_stays <- FALSE
    prev_taxon_depth <- taxon_depth
    taxons_above <- c()
    for (i in seq(from=(pos.taxon-1),to=1)) {
      curr_taxon_depth <- report[i,"depth"]
      if (curr_taxon_depth < prev_taxon_depth) {
        if (!any_stays) {
          if (report[i,"cladeReads"] == taxonReads) {
            rows_to_delete[i] <- TRUE
            if (do_message)
              dmessage("Deleting ",report[i,"name"])
          } else {
            any_stays <- TRUE
          }
        }
        if (!rows_to_delete[i]) {
          rows_to_update <- c(rows_to_update, i)
          if (do_message)
            dmessage("Updating ",report[i,"name"])
        }
        prev_taxon_depth <- curr_taxon_depth
      } else {
        any_stays <- TRUE
      }
    }
    report[rows_to_update, "cladeReads"] <- report[rows_to_update, "cladeReads"] - taxonReads
  }

  #if (rm_clade)
  report[!rows_to_delete,]
  #else
  #  report
}


#' Read Kraken-style and MetaPhlAn reports
#'
#' @param myfile Kraken-style or MetaPhlAn report file.
#' @param has_header If the kraken report has a header or not.
#' @param check_file If TRUE, only the first 5 lines of the file are loaded.
#'
#' @return report data.frame
#' @export
#'
read_report <- function(myfile, has_header=NULL, check_file = FALSE) {

  # TODO: Support for gzipped files ..
  #myfile <- file(myfile)
  #file_class <- summary(myfile)$class
  #if (file_class == "gzfile")
  #  myfile <- gzcon(myfile)

  first.line <- tryCatch( readLines(myfile,n=1, warn=FALSE),
                          error = function(e) { warning("Error reading ",myfile); return() })
```

```r
isASCII <-  function(txt) {
  if (length(txt) == 0)
    return(FALSE)
  raw <- charToRaw(txt)
  all(raw <= as.raw(127) & (raw >= as.raw(32) | raw == as.raw(9)))
}
if (length(first.line) == 0) {
  dmessage("Could not read ", myfile, ".")
  return(NULL)
}
tryCatch({
if (nchar(first.line) == 0) {
  dmessage("First line of ", myfile, " is empty")
  return(NULL)
}
}, error = function(e) {
  dmessage(e)
  return(NULL)
})

if (!isTRUE(isASCII(first.line))) {
  dmessage(myfile," is not a ASCII file")
  return(NULL)
}

if (is.null(has_header)) {
  has_header <- grepl("^[a-zA-Z#%\"]",first.line)
}
is_metaphlan3_fmt <- grepl("^#mpa_v", first.line)
is_metaphlan2_fmt <- grepl("Metaphlan2_Analysis$", first.line)
is_krakenu_fmt <- grepl("^.?%\treads\ttaxReads\tkmers", first.line)
is_kaiju_fmt <- grepl("^  *%\t  *reads", first.line)

ntabs <- lengths(regmatches(first.line, gregexpr("\t", first.line)))

nrows <- ifelse(isTRUE(check_file), 5, -1)
if (!is_krakenu_fmt && is_kaiju_fmt) {
  cont <- readLines(myfile)
  cont <- cont[!grepl("^-", cont)]
  cont <- sub(".*\t  *","", cont)
  cont <- sub("; ?$","", cont)
  report <- utils::read.delim(textConnection(cont), stringsAsFactors = FALSE)
  colnames(report) <- c("taxonReads", "taxLineage")
  report$cladeReads <- report$taxonReads

  report$taxLineage <- gsub("^","-_",report$taxLineage)
  report$taxLineage <- gsub("; ","|-_",report$taxLineage)
  report$taxLineage <- gsub("-_Viruses", "d_Viruses", report$taxLineage, fixed=T)
  report$taxLineage <- gsub("-_cellular organisms|-_Bacteria", "-_cellular organisms|d_Bacteria", repo
  report$taxLineage <- gsub("-_cellular organisms|-_Eukaryota", "-_cellular organisms|d_Eukaryota", re
  report$taxLineage <- gsub("-_cellular organisms|-_Archaea", "-_cellular organisms|d_Archaea", repor
  report$taxLineage[1:(length(report$taxLineage)-1)] <- paste0("-_root|", report$taxLineage[1:(length
  report$taxLineage[report$taxLineage=="-_unclassified"] <- "u_unclassified"

  new_counts <- integer(length = 0)
  for (j in seq_len(nrow(report))) {
    count <- report$cladeReads[j]
    tl <- report$taxLineage[j]
    tl2 <- sub("\\|[^|]*$","", tl)
    while (tl2 != tl) {
      if (tl2 %in% names(new_counts)) {
        new_counts[tl2] <- new_counts[tl2] + count
      } else {
        new_counts[tl2] <- count
      }
      tl <- tl2
      tl2 <- sub("\\|[^|]*$","", tl)
    }
  }
  report <- rbind(report,
                  data.frame(taxonReads=0,taxLineage=names(new_counts),cladeReads=as.integer(new_coun
  tl_order <- order(report$taxLineage)
  tl_order <- c(tl_order[length(tl_order)],tl_order[-length(tl_order)])
  report <- report[tl_order, c("taxLineage", "taxonReads", "cladeReads")]
} else if (is_metaphlan3_fmt) {
  report <- tryCatch({
    ## TODO: Having comment_char here causes a problem w/ Metaphlan report!!

    utils::read.table(myfile,sep="\t",header = F,
                      quote = "",stringsAsFactors=FALSE,
                      comment.char = "#", nrows = nrows,
                      col.names = c("taxLineage", "taxID", "cladeReads", "additional_species"),
                      check.names=FALSE)
  }, error = function(x) NULL, warning = function(x) NULL)
  if (is.null(report)) { return(NULL); }

  report$taxID <- sub(".*\\|", "", report$taxID)

} else if (has_header) {
  report <- tryCatch({
    ## TODO: Having comment_char here causes a problem w/ Metaphlan report!!

    utils::read.table(myfile,sep="\t",header = T,
                      quote = "",stringsAsFactors=FALSE,
                      comment.char = ifelse(is_metaphlan2_fmt, "", "#"), nrows = nrows,
                      check.names=FALSE)
  }, error = function(x) NULL, warning = function(x) NULL)
  if (is.null(report)) { return(NULL); }
  #colnames(report) <- c("percentage","cladeReads","taxonReads","taxRank","taxID","n_unique_kmers","n_

  ## harmonize column names. TODO: Harmonize them in the scripts!
  colnames(report)[colnames(report) %in% c("#%","%","clade_perc","perc","percReadsClade")] <- "percen
  colnames(report)[colnames(report) %in% c("reads","numReadsClade","n_reads_clade","n.clade","n-clade
  colnames(report)[colnames(report) %in% c("taxReads","numReadsTaxon","n_reads_taxo","n.stay","n-stay
  colnames(report)[colnames(report) %in% c("rank","tax_taxRank","level")] <- "taxRank"
```

```
    colnames(report)[colnames(report) %in% c("tax","taxonid")] <- "taxID"
    colnames(report)[colnames(report) %in% c("indentedName","taxName")] <- "name"
    colnames(report)[colnames(report) %in% c("dup")] <- "kmerDuplicity"
    colnames(report)[colnames(report) %in% c("cov")] <- "kmerCoverage"
  } else {
    report <- NULL
    if (ntabs == 5) {
      col_names <-  c("percentage","cladeReads","taxonReads","taxRank","taxID","name")
    } else if (ntabs == 7) {
      col_names <- c("percentage","cladeReads","taxonReads", "n_unique_kmers","n_kmers", "taxRank","tax
    }
    report <- tryCatch({
      utils::read.table(myfile,sep="\t",header = F,
                        col.names = col_names,
                        quote = "",stringsAsFactors=FALSE,
                        nrows = nrows)
    }, error=function(x) NULL, warning=function(x) NULL)

    if (is.null(report)) {
      dmessage(paste("Warning: File",myfile,"does not have the required format"))
      return(NULL);
    }

  }

  if (ncol(report) < 2) {
    dmessage(paste("Warning: File",myfile,"does not have the required format"))
    return(NULL)
  }
  if (is_metaphlan2_fmt || is_metaphlan3_fmt) {
    ## Metaphlan report
    colnames(report)[1] <- "taxLineage"
    colnames(report)[colnames(report) == "Metaphlan2_Analysis"] <- "cladeReads"
    report <- report[order(report$taxLineage), ]
    report$taxLineage <- gsub("_"," ",report$taxLineage)
    report$taxLineage <- gsub("  ","_",report$taxLineage)
    report$taxLineage <- paste0("-_root|", report$taxLineage)
    root_lines <- data.frame(taxLineage=c("u_unclassified","-_root"),"cladeReads"=c(0,100), stringsAsFa

    if ("taxID" %in% colnames(report)) {
      root_lines <- cbind(root_lines, "taxID" = c(0, 1))
      report <- report[, c("taxLineage", "cladeReads", "taxID")]
    }

    report <- rbind(root_lines, report)
  }

  if (all(c("name","taxRank") %in% colnames(report)) && !"taxLineage" %in% colnames(report)) {
    ## Kraken report
    report$depth <- nchar(gsub("\\S.*","",report$name))/2
    if (!all(report$depth == floor(report$depth))) {
      warning("Depth doesn't work out!")
      return(NULL)
    }
    report$name <- gsub("^ *","",report$name)

    ## 'fix' taxRank
    table(report$taxRank)
    allowed_taxRanks <- c("U", "S", "G", "F", "C", "D", "O", "K", "P")
    report$taxRank[report$taxRank=="class"] <- "C"
    report$taxRank[report$taxRank=="family"] <- "F"
    report$taxRank[report$taxRank=="genus"] <- "G"
    report$taxRank[report$taxRank=="superkingdom"] <- "D"
    report$taxRank[report$taxRank=="kingdom"] <- "K"
    report$taxRank[report$taxRank=="order"] <- "O"
    report$taxRank[report$taxRank=="phylum"] <- "P"
    report$taxRank[report$taxRank=="species"] <- "S"
    report$taxRank[report$name=="unclassified"] <- "U"
    report$taxRank[!report$taxRank %in% allowed_taxRanks] <- "-"

    report$name <- paste(tolower(report$taxRank),report$name,sep="_")

    rownames(report) <- NULL

    ## make taxLineage path
    report$taxLineage <- report$name
    n <- nrow(report)
    depths <- report$depth
    taxLineages <- report$name
    taxLineages_p <- as.list(seq_along(report$name))

    depth_row_tmp <- c(1:25)

    for (current_row in seq(from=1, to=nrow(report))) {
      dcr <- depths[current_row]
      depth_row_tmp[dcr+1] <- current_row
      if (dcr >= 1) {
        prev_pos <- depth_row_tmp[[dcr]]
        taxLineages_p[[current_row]] <- c(taxLineages_p[[prev_pos]], current_row)
      }
    }
    report$taxLineage <- sapply(taxLineages_p, function(x) paste0(taxLineages[x], collapse="|"))
    #report$taxLineage <- taxLineages

  } else if ("taxLineage" %in% colnames(report)) {
    taxLineages <- strsplit(report$taxLineage, "|", fixed=TRUE)

    if (!"name" %in% colnames(report))
      report$name <- sapply(taxLineages, function(x) x[length(x)])

    if (!"depth" %in% colnames(report)) {
      report$depth <- sapply(taxLineages, length) - 1
    }
    if (!"taxRank" %in% colnames(report))
      report$taxRank <- toupper(substr(report$name, 0, 1))
  }
```

```
  if (!all(c("name","taxRank") %in% colnames(report)) ||
      nrow(report) < 2 ||
      !((report[1,"name"] == "u_unclassified" && report[2,"name"] == "-_root") || report[1,"name"] == "
    dmessage(paste("Warning: File",myfile,"does not have the required format"))
    str(report)
    return(NULL)
  }


  if (!"taxonReads" %in% colnames(report)) {
    parent <- sub("^\\(.*\\)\\|.*$", "\\1", report$taxLineage)
    taxLineages <- strsplit(report$taxLineage, "|", fixed=TRUE)
    ## fix taxonReads
    report$taxonReads <- report$cladeReads - sapply(report$name, function(x) sum(report$cladeReads[pare
    #report$taxonReads[sapply(report$taxonReads, function(x) isTRUE(all.equal(x, 0)))] <- 0
    report$taxonReads[report$taxonReads <= 0.00001] <- 0  # fix for rounding in percentages by MetaPhlA
  }

  report$percentage <- signif(report$cladeReads/sum(report$taxonReads),6) * 100
  if ('n_unique_kmers'  %in% colnames(report))
    report$kmerpercentage <- round(report$n_unique_kmers/sum(report$n_unique_kmers,na.rm=T),6) * 100
  #report$taxRankperc <- 100/taxRank(report$cladeReads)

  #report$depth <- NULL

  if ("taxID" %in% colnames(report)) {
    std_colnames <- c("percentage","cladeReads","taxonReads","taxRank", "taxID","name")
  } else {
    std_colnames <- c("percentage","cladeReads","taxonReads","taxRank","name")
  }
  stopifnot(all(std_colnames %in% colnames(report)))
  report[, c(std_colnames, setdiff(colnames(report), std_colnames))]
}

sankeyNetwork(
  Links,
  Nodes,
  Source,
  Target,
  Value,
  NodeID,
  NodeGroup = NULL,
  NodeColor = NULL,
  NodePosX = NULL,
  NodePosY = NULL,
  NodeValue = NULL,
  NodeFontColor = NULL,
  NodeFontSize = NULL,
  LinkGroup = NULL,
  linkColor = "#A0A0A0",
  units = "",
  colourScale = NULL,
```

```r
  fontSize = 12,
  fontFamily = "Arial",
  fontColor = NULL,
  nodeWidth = 15,
  nodePadding = NULL,
  nodeStrokeWidth = 1,
  nodeCornerRadius = 0,
  margin = NULL,
  title = NULL,
  numberFormat = ",.1f",
  orderByPath = FALSE,
  highlightChildLinks = FALSE,
  doubleclickTogglesChildren = FALSE,
  xAxisDomain = NULL,
  dragX = FALSE,
  dragY = TRUE,
  height = 500,
  width = 800,
  iterations = 32,
  zoom = FALSE,
  align = "left",
  showNodeValues = TRUE,
  linkType = "bezier",
  curvature = 0.5,
  nodeLabelMargin = 2,
  linkOpacity = 0.5,
  linkGradient = FALSE,
  nodeShadow = FALSE,
  scaleNodeBreadthsByString = FALSE,
  xScalingFactor = 1,
  yOrderComparator = NULL
)

test <- read_report("/home/floris/Documenten/Data_set/ACFC_Wastewater_Metagenomics/kraken2/reports/in_la

test2 <- build_sankey_network(test, taxRanks =  c("D","K","P","C","O","F","G","S"), maxn=10, zoom = F)
test2

saveNetwork(
  test2,
  folder_path = NULL,
  file_name = "temp",
  png = "create",
  html = "none",
  vwidth = network$width,
  vheight = network$height,
  zoom = 5,
  delay = 0.5,
  selfcontained = TRUE
)
ggsave("Sankey_test.png",plot = test2)
```

Chen, S. 2023. "Ultrafast One-pass FASTQ Data Preprocessing, Quality Control, and Deduplication Using Fastp." *iMeta* 2 (2). https://doi.org/10.1002/imt2.107.

DeSantis, T. Z., P. Hugenholtz, N. Larsen, M. Rojas, E. L. Brodie, K. Keller, T. Huber, D. Dalevi, P. Hu, and G. L. Andersen. 2006. "Greengenes, a Chimera-Checked 16S rRNA Gene Database and Workbench Compatible with ARB." *Applied and Environmental Microbiology* 72 (7): 5069–72. https://doi.org/10.1128/AEM.03006-05.

Hart, Lauren N, Brittany N Zepernick, Kaela E Natwora, Katelyn M Brown, Julia Akinyi Obuya, Davide Lomeo, Malcolm A Barnard, et al. 2025. "Metagenomics Reveals Spatial Variation in Cyanobacterial Composition, Function, and Biosynthetic Potential in the Winam Gulf, Lake Victoria, Kenya." *Applied and Environmental Microbiology*, e01507–24.

Hong, Pei-Ying, David Mantilla-Calderon, and Changzhi Wang. 2020. "Metagenomics as a Tool to Monitor Reclaimed-Water Quality." *Applied and Environmental Microbiology* 86 (16): e00724–20.

Lal Gupta, Chhedi, Rohit Kumar Tiwari, and Eddie Cytryn. 2020. "Platforms for Elucidating Antibiotic Resistance in Single Genomes and Complex Metagenomes." *Environment International* 138: 105667. https://doi.org/https://doi.org/10.1016/j.envint.2020.105667.

Louca, Stilianos, Laura Wegener Parfrey, and Michael Doebeli. 2016. "Decoupling Function and Taxonomy in the Global Ocean Microbiome." *Science* 353 (6305): 1272–77.

Obiero, KO, PO Wa'Munga, PO Raburu, and JB Okeyo-Owuor. 2012. "The People of Nyando Wetland: Socioeconomics, Gender and Cultural Issues." *Community Based Approach to the Management of Nyando Wetland, Lake Victoria Basin, Kenya* 1: 41–44.

Terlouw, Barbara R, Kai Blin, Jorge C Navarro-Munoz, Nicole E Avalon, Marc G Chevrette, Susan Egbert, Sanghoon Lee, et al. 2023. "MIBiG 3.0: A Community-Driven Effort to Annotate Experimentally Validated Biosynthetic Gene Clusters." *Nucleic Acids Research* 51 (D1): D603–10.

Ye, Simon H, Katherine J Siddle, Daniel J Park, and Pardis C Sabeti. 2019. "Benchmarking Metagenomics Tools for Taxonomic Classification." *Cell* 178 (4): 779–94. https://doi.org/10.1016/j.cell.2019.07.010.