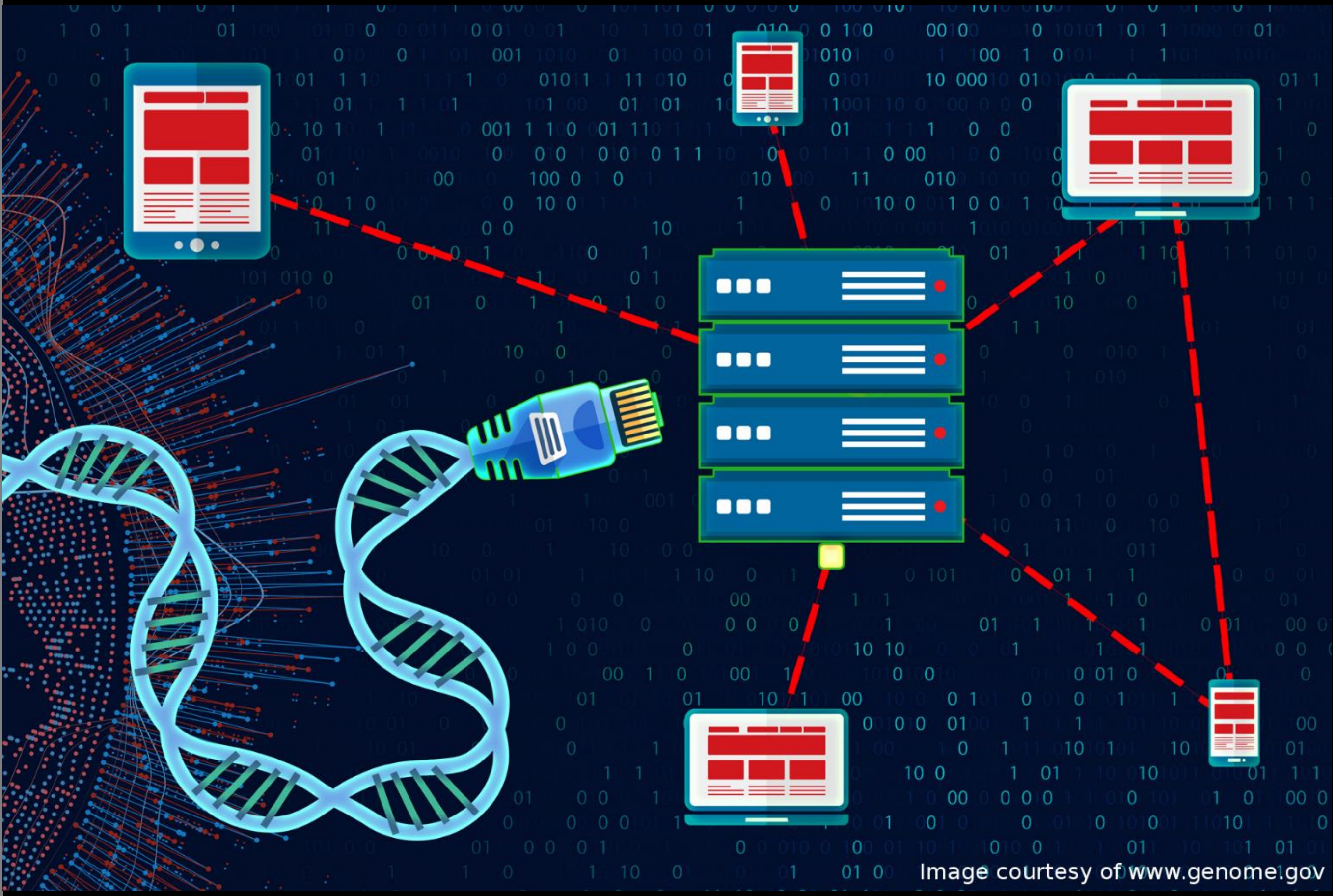


ResistNanome

Easy pipeline for resistome determination of metagenomic nanopore data

Stieva, A. (Ann-Britt)

14-01-2020

Universiteit Utrecht

Picture title page: https://cs.pitt.edu/degrees/bioinformatics/

ResistNanome

Name:	Ann-Britt Stieva
Student ID:	0893053
Solis ID:	Verwi006
Date:	14-01-2020
Internship at:	University of Utrecht
Faculty:	Veterinary
Department:	Infectious diseases and Immunology (I & I)
Section:	Clinical Infectiology (KLIF)
Building:	Androclus building
Address internship:	Yalelaan 1
	3584 CL Utrecht
Supervisor:	Dr A. L. Zomer
Internship period:	09-09-2019 – 24-01-2020
University:	Rotterdam University of Applied Science
Building:	Academieplein
Address:	G.J. de Jonghweg 4-6
	3015 GG Rotterdam
Institute:	Engineering and Applied Science (EAS)
Course:	Biology and Medical Laboratory researches (BML)
Differentiation:	Research
Minor:	Life Science
Supervisor:	Meike Hensmann

Contents

ResistNanome 1

Abstract 3

Samenvatting 3

List of terms and abbreviations 4

Introduction 5

Antibiotic resistance 5

Metagenomics 5

(Nanopore) sequencing 5

Bioinformatics 6

Python 7

Linux Bash 7

Goals 7

Materials and methods 9

Building a pipeline 9

Implemented Python modules 10

Implemented tools 11

Databases 12

GitHub 13

Results 14

Building the pipeline 14

Generated output 14

Example dataset 18

Conclusion and discussion 19

Recommendations 20

Acknowledgements 21

Bibliography 22

Appendix 27

1. ResistNanome wrapper script 27
2. GNU license 39
3. Database for Minimap2 based on NCBI RefSeq database 49
4. Porechop verbose output ..... **Error! Bookmark not defined.**
5. ResistNanome output of a run by A.B. with barcode 19 ..... **Error! Bookmark not defined.**

Abstract

Bioinformatics is part of the future of (laboratory) research. Big data needs to be processed and doing so manually would be impossible. Sequencing techniques keep improving and with one of the latest techniques, Nanopore sequencing, a vast number of long reads can be created. This technique makes it easier to gain new information, for example, it will work well for sequencing DNA from metagenomic samples. To process this data, new tools and bioinformatics pipelines are needed. ResistNanome is one such pipeline, created within this project. It was built to process long-read data to uncover the antibiotic resistance genes and the bacterial composition in a metagenomic sample. ResistNanome combines multiple tools to produce a readable output while providing a relatively simple command line interface. With a single command line in Unix, ResistNanome can be started. The output is based on what the user wants, given through simple commands by said user. It gives the top 10 most common bacteria and their antibiotic resistance genes to give a quick overview and all data in a tab-separated file.

Samenvatting

Bioinformatica is een groot onderdeel van de toekomst van (laboratorium) onderzoek. ‘Big data’ moet verwerkt worden, en met de hand is deze taak onmogelijk. Sequens-technieken blijven zich ontwikkelen en geven meer en meer data. Nanopore, een van de nieuwere technieken, heeft de mogelijkheid grote hoeveelheden lange reads te creëren. Op deze manier kan er een hoop nieuwe informatie bij komen. Een voorbeeld is metagenomische data. Er bestaan al tools voor het werken met metagenomische-/Nanopore data. Het gebruik hiervan, of wat eruit komt, is niet altijd duidelijk voor iemand met weinig kennis van informatica. Dit onderzoek had als doel een pipeline te creëren die de antibioticaresistentie, samen met een taxonomisatie, uit metagenomische data gegenereerd met Nanopore sequencing haalt. ResistNanome is geschreven in Python en maakt gebruik van een aantal bestaande tools. Deze worden gecombineerd zodat de gebruiker alleen basiskennis nodig heeft om het gebruik en interpretatie te begrijpen. Met één commando zin in Linux is de pipeline te starten, door het geven van argumenten is wat de pipeline exact doet aan te passen naar de wensen van de gebruiker. Het belangrijkste wat ResistNanome kan is een antibiotica resistentiebepaling van metagenomische data. Het resistoom wordt bepaald, naast een bacteriebepaling, deze worden samengevoegd in een tab-gescheiden file, te openen in Excel. Een top 10 van de meest voorkomende bacteriën wordt ook verwerkt tot een pdf-bestand.

List of terms and abbreviations

ASCII	American Standard Code for Information Interchange
Bash	Bourne Again Shell, a regularly used Unix shell
CLI	Command line interface, a terminal for processing commands to a computer, like Unix and Windows’ command prompt.
csv-file	Comma Separated Values file, the text in the file exist as columns, each column is separated by a specific character, like comma (;), or a tab. These files can be opened in Excel.
db	Database: A compilation of specific data. For example, all the genes that code for antibiotic resistance and their names.
(ss)DNA	(single strand) Deoxyribonucleic acid
Fast5	Format created by ONT, it gives the electric current fluctuation as created by the Nanopore.
FastA	A text-based format that represents the nucleotide sequence of one or more reads. The first line gives the read-name behind the character '>' the second line the sequence.
FastQ	Like FastA, but adds a third and fourth line per read. The third the character '+', possibly followed by the read name, again. Line four gives the quality of the sequence in ASCII.
Git	A revision control system which runs in command line interfaces.
GPL	general public license
gz	GNU zip. A form of data compression for Unix/Linux files.
HTML-file	HyperText Mark-up Language. A file containing the code for a web-page.
Illumina	A type of sequencing using PCR on a flow cell and fluorescently labelled nucleotides.
living (metagenomic) library	Fragments of metagenomic DNA cloned into the plasmids of laboratory bacteria.
module	A file containing one or more functions, to be used in an application.
(de)multiplex	Compiling things (like DNA or data) to undergo the same processing. (And taking it apart afterwards.)
NGS	Next Generation Sequencing
ONT	Oxford Nanopore Technologies. The company behind Nanopore Sequencing.
PCR	Polymerase Chain Reaction
PDF-file	Portable Document Format. A file format for text, in which the text can't be altered.
(informatics) pipeline	A combination of (informatics) processes, linked together to get to one (set of) output(s).
R9	The newest type of flow cell produced by ONT, it uses the CsgG protein nano pores.
taxonomy	The information about what specie, genus, family etc. a read is. Also called community screening.
txt-file	A text file. A simple way of storing information.

Introduction

Antibiotic resistance

Bacteria are becoming more and more antibiotic resistant globally [1] [2]. This is a problem because the use of antibiotics is a widely used and effective way to fight and prevent bacterial infections. Antibiotics can kill or suppress the growth of bacteria. The collection of genes responsible for antibiotic resistance in a metagenomics sample (see metagenomics, p.4) is called the resistome.

Antibiotics are mostly used as medicine. When a patient falls ill due to a bacterial infection, they get a prescription for antibiotics. Antibiotics can also be used to conduct tests in laboratories and, in lesser amounts, in feed given to livestock.

Antibiotic resistance can come about through random mutation or through acquisition of resistance genes through horizontal gene transfer [1] [2]. However, there are factors that facilitate the occurrence of antibiotic resistance. Misuse that accelerates this process includes overprescribing of antibiotics of health workers and veterinarians, over usage by the public, improper use by the public, using leftover antibiotics and overuse by livestock farmers.

When antibiotics aren't effective anymore, a simple infection could have disastrous consequences once more. Luckily there are ways to prevent overuse of them. One very important for healthcare professionals is to use the correct antibiotic, so stronger ones will not be overused. For this it is crucial to determine what bacteria and any possible resistances are present.

Metagenomics

Microbes are essential for all life on earth [3]. Metagenomics can be described as the study of the genetic makeup of these microbial communities as a whole. Working with data generated by one metagenomic community directly (or through a living library) has several benefits over pre-existing techniques. Because the DNA is taken of a sample directly, all the microbes are represented, even ones that won't be able to grow in clinical surroundings like an agar-plate [4]. Furthermore, because there is no need for amplification, the composition of the genomic diversity can be determined.

Though this is very useful, it also calls for different/new techniques for generating and processing this data. Some specific ways of sequencing, assembling and determination of the taxonomy are already created. The taxonomy tool Kraken 2 [5], for example, is created for community profiling of reads and can even generate the abundancy per species with help of a second tool (bracken [6]) (see Materials and methods for more information about the tools). The relatively new 'third generation' sequencing technique, Nanopore, developed by Oxford Nanopore Technologies (ONT) (see "(Nanopore) sequencing", p. 4) works well for generating metagenomic data.

(Nanopore) sequencing

DNA, the blueprints of all life. To translate a strand of DNA to a neat string of A, T, G and C (or RNA to A, U, G and C), the DNA is sequenced. The first form of sequencing, Sanger sequencing, was developed by Frederick Sanger in the mid-70s [7]. This technique is based on polymerase chain reaction (PCR) with, aside from normal nucleotides, dideoxynucleotides (stop) with fluorescent dye, one colour per nucleotide. After the PCR, the products (of different lengths) are separated using capillary electrophoresis. Because of the different colours being made visible and at different lengths, the order of nucleotides is made visible.

Later techniques are often called next generation sequencing [8]. There are multiple techniques called this, all of which quicker than Sanger sequencing. The common denominator in these techniques is that they all sequence millions of small DNA fragments at the same time. All these fragments are later pieced together with use of bioinformatic analyses, mapping against a reference genome. It is highly accurate and could sequence the human genome in a day. An example of next generation sequencing that is used a lot is Illumina sequencing [9]. For this, ssDNA with terminating sequencing, able to bind to a flow cell, are put on said flow cell. Afterwards, bridge amplification will happen to create clusters of the same strands, close together. The reverse strands are washed off, before a series of nucleotide washes. Each time the nucleotide will bind to the next nucleotide of the strand, the cluster will light up with a specific colour. This way, the sequence of each cluster can be determined.

The newest technique, often called third generation sequencing, is characterised by negating the requirement for DNA amplification [9]. While there are more techniques, the area most anticipated, must be Nanopore sequencing. This technique has been theorised, even before second generation. The first company to offer Nanopore sequencers, ONT, offers this on multiple platforms. The most well-known of which the MinION, a small USB device, not even the size of a small mobile phone. This makes the device very well-suited for sequencing, even at unconventional places. The device works by putting the sample on a flow cell [10] [11]. The flow cell has 4 x 512 sensing wells, each with their own nanopore. The pores currently used by ONT are a mutated lipoprotein from *E. coli*, with an inner diameter of 1nm (called version R9). The outside is outfitted with a motor protein that leads single strand DNA trough the pore at a fixed speed. The electric current flowing through the pore is measured and changes with everything going through the pore. Each different base gives a different current ‘squiggle’ as output. A schematic image of a nanopore can be seen in figure 1. There are multiple programs, produced by either ONT, other companies or independent creators, for base calling, translating the raw current to DNA sequences and a quality measurement in a format called FastQ. Because Nanopore sequencing can work with samples directly, it gives an accurate overview of what bacteria are in a sample. It is even possible to run multiple samples at once, by giving each sample a different barcode. The downside of Nanopore sequencing seems to be the accuracy of the reads (85%-95%) and the cost. The reads are, however, generated quite quickly and very long, compared to earlier sequence techniques.

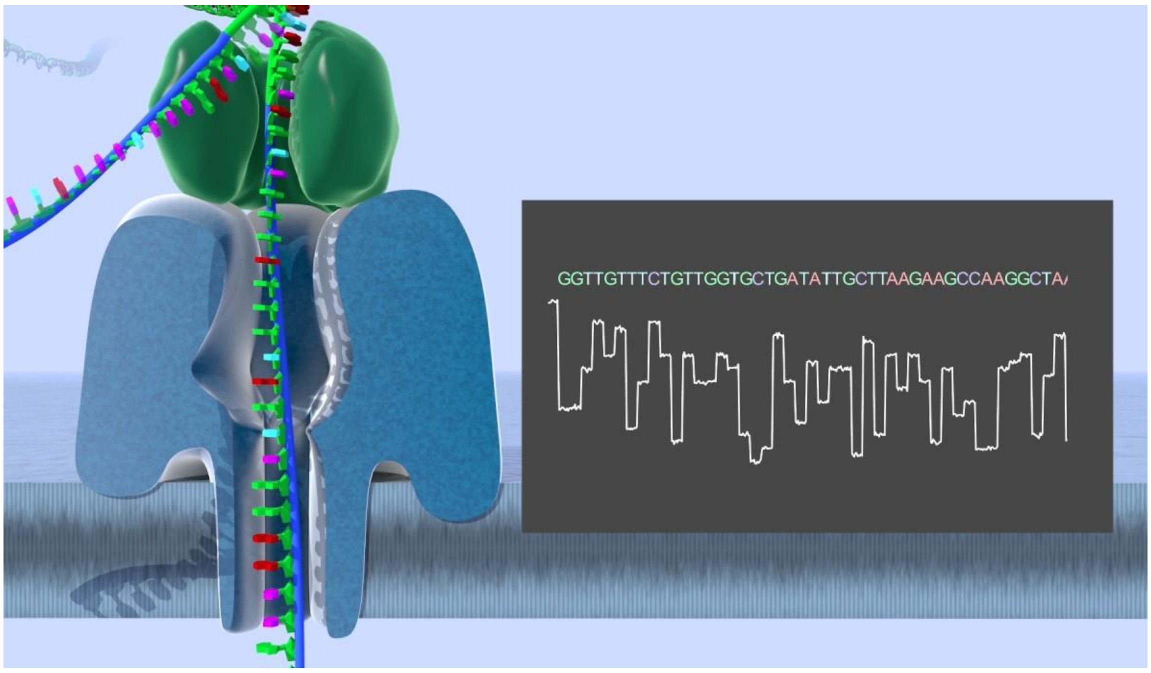


Figure 1. Nanopore reading a DNA strand, including output [55]

Bioinformatics

Bioinformatics combines the organic molecular biology with statistics and informatics. Biological research can generate large amounts of data and using informatics to process this can bring new things to the table, helpful to study life. Not to mention the time saved with automating processes that could take ages by hand. Sequencing is a good example of how bioinformatics can help automating a process and make it faster.

Python

Manipulating data is what informatics is about. To do that, you'll need a programming language and one of the most widely used is Python. Python is an extremely versatile language and is classified as high-level, both of which makes it suitable for beginning programmers [12] [13]. High-level means that Python is a very readable/understandable language for humans.

Python can be used for close to everything, from calculations to creating moving images to building games. Python is supplemented by functions, a single string without spaces, that takes inputs and is tasked to do something specific with it. Functions can be built in or build by the user themselves. They can also be imported as a module, Python is a modular programming language [14]. A module is based on scripts and can carry multiple functions or tools, they don't have to be in the same script as they are used in. Modules that provide additional functions, say reading DNA sequence reads, can easily be imported in your program by using the code "import [module]".

Certain modules collect multiple functions of a specific type. One example is the module specifically for bioinformatics: Biopython [15]. With Biopython, it is possible to read, adjust or write FastA/FastQ files produced through sequencing, make a sequence out of a string, transcribe or translate sequences or create a reverse and/or complement.

Linux Bash

Linux is an operating system used with powerful computers. Linux works with command line interpreter called a Unix shell. Bash is one such interpreter [16]. Bash, or Bourne Again Shell as a word play on its predecessor the Bourne shell, works differently than Python, talked about before. Through the shell, other languages can be interpreted as well. So a script can be written in Python, but with the right commands, can be executed through Bash as well. This can be helpful when using a tool, written in one type of code, within a script in another code. Unix is made easier by the added use of autofill, using the tab-key.

Goals

Using bioinformatics, all types of data processing can be automated. The ultimate goal of this research is to use this automatization to take metagenomics data, produced by the ONT MinION, and determine the antibiotic resistance genes and what bacteria they belong to. The pipeline built for this is called ResistNanome and a schematic representation of it is visible in figure 2 (p.7). This pipeline should ultimately be faster and more specific than the traditional way of seeing if cultures will grow on selective medium agar plates. This tool is designed for veterinarians to use on MinION data of animal faeces, after sequencing for around six hours.

Tools to get the results one wants out of the Nanopore data already exist but aren't always easy to find or use for users with little bioinformatics background. ResistNanome implements multiple tools without asking too much information or knowledge about informatics of the user. A so-called plug-and-play pipeline. The tools combined for ResistNanome can be found in the Materials and Methods (p.9). These tools were picked and implemented with specific parts of the pipeline in mind: quality control and improvement; removing of reads that contain host (vertebrate) DNA; taxonomy determination and antibiotic resistance determination. To make the pipeline more user-friendly, the output will be given in a format that any user with knowledge about bacteria and antibiotic resistance should be able to use. The implementation of each tool and how to use the output is set using Python coding.

A pipeline like this has been built at Universiteit Utrecht before [17]. This one, the FastDeMe, was built for Illumina data, so it won't work optimal for Nanopore data. However, the knowledge gained during this project is very valuable for the ResistNanome.

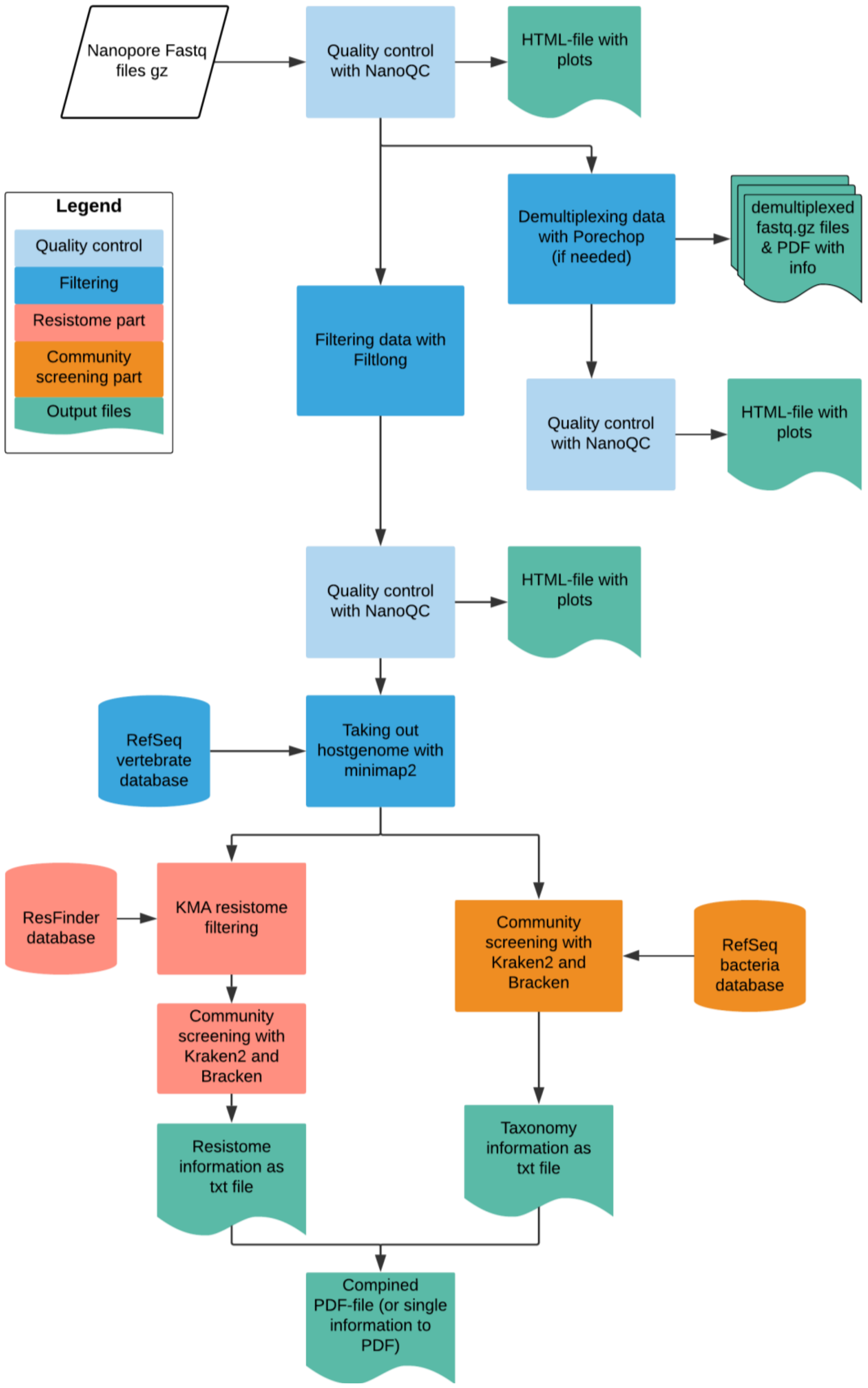

Nanopore Fastq files gz
Quality control with NanoQC
HTML-file with plots
Legend
Quality control
Filtering
Resistome part
Community screening part
Output files
Demultiplexing data with Porechop (if needed)
demultiplexed fastq.gz files & PDF with info
Filtering data with Filtlong
Quality control with NanoQC
HTML-file with plots
Quality control with NanoQC
HTML-file with plots
RefSeq vertebrate database
Taking out hostgenome with minimap2
ResFinder database
KMA resistome filtering
Community screening with Kraken2 and Bracken
RefSeq bacteria database
Community screening with Kraken2 and Bracken
Resistome information as txt file
Taxonomy information as txt file
Compined PDF-file (or single information to PDF)

Figure 2. Schematic representation of the ResistNanome pipeline

Materials and methods

Building a pipeline

Building a pipeline consist of a lot of trial and error. The start is knowing what you want the pipeline to do, in this case: improve the quality of Nanopore reads; take out the host genome of the metagenomic reads; determine the resistome; and determine the taxonomy. This is chopped up in smaller, easier sub-projects. Each part is the same, write code you think will work, perform a test-run, look at what went wrong, re-write the code and repeat (see figure 3). In between and at the end, it is important to make sure all parts work together correctly. Some coding is needed to take the output from one part and use that as input, this should be tested too, between every combination of parts. For this, a wrapper is built that combines/implements all the sub-parts and makes the pipeline a single pipeline(see appendix 1).

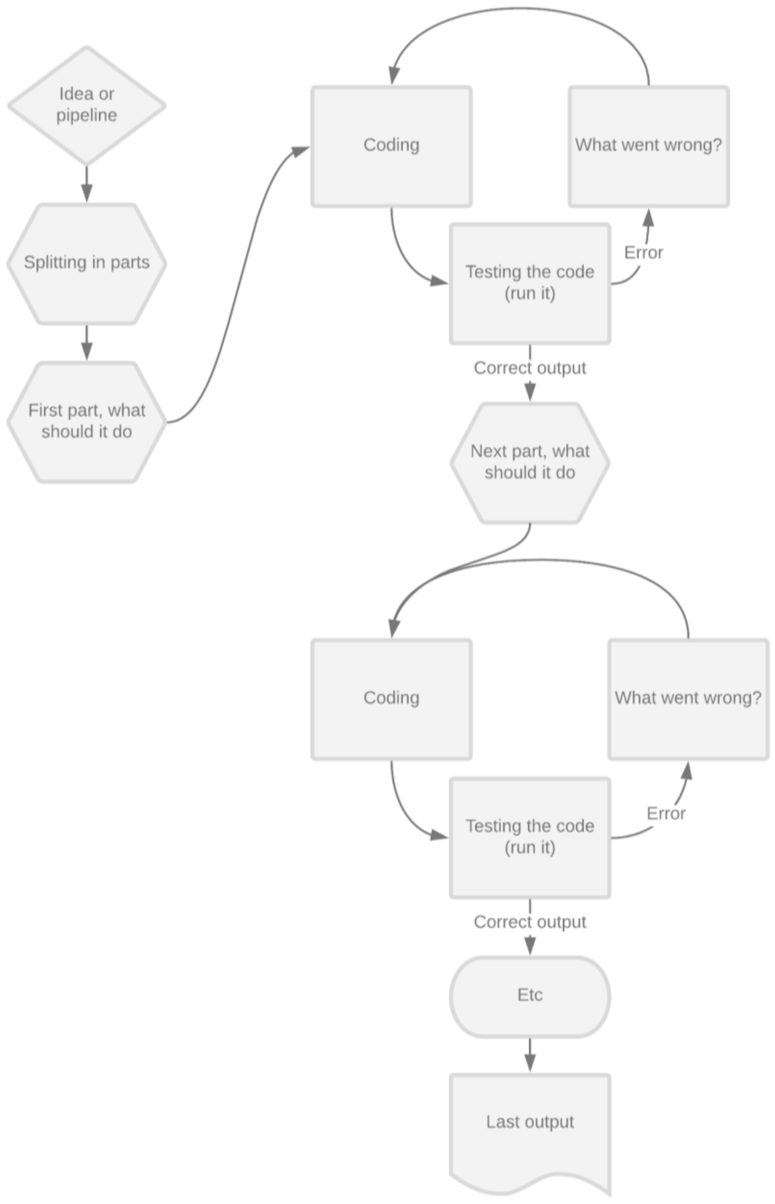
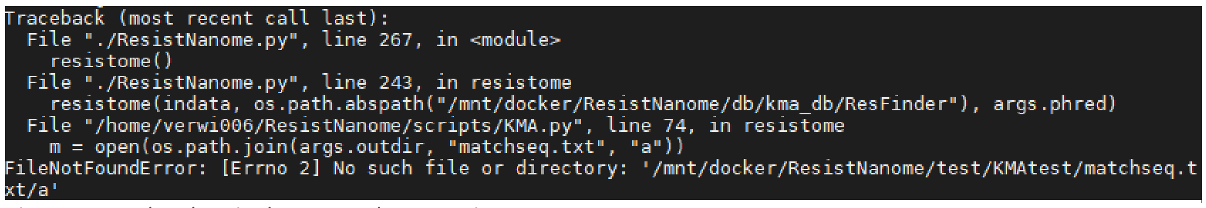


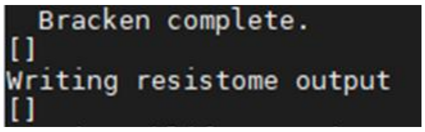
Figure 3. How building a pipeline works

Before having a working product, there will be errors. In the case of Python, the error will tell you the type of mistake, the line number of the line where the mistake has effect and will give this line as well. For example in figure 4: a line in the resistome coding, `m = open(os.path.join(args.outdir, "matchseq.txt", "a"))`, has a bracket in the wrong place. It is solved by moving the before last bracket to in front of the first comma and should be: `m = open(os.path.join(args.outdir, "matchseq.txt"), "a")`. When the error is not clear, the best way of solving it is looking it up on the internet. Small parts of coding could also be looked up on the internet, for example when it is unclear how to make it work. During this project, the site of Stack Overflow [18] was used a lot, whether for solving an error or for finding out how to make something work the way it's supposed to. While errors can be very unfortunate, they are also useful. Some coding that won't work will not give an error right away, though. To check whether a code works, checks are temporarily put in. In this project, those checks where most often the printing of a list. When the code didn't work correctly an empty list, like in figure 5, was printed.



```
Traceback (most recent call last):
  File "./ResistNanome.py", line 267, in <module>
    resistome()
  File "./ResistNanome.py", line 243, in resistome
    resistome(indata, os.path.abspath("/mnt/docker/ResistNanome/db/kma_db/ResFinder"), args.phred)
  File "/home/verwi006/ResistNanome/scripts/KMA.py", line 74, in resistome
    m = open(os.path.join(args.outdir, "matchseq.txt", "a"))
FileNotFoundError: [Errno 2] No such file or directory: '/mnt/docker/ResistNanome/test/KMAtest/matchseq.t
xt/a'
```

Figure 4. How brackets in the wrong place can give an error



```
 Bracken complete.
[]
Writing resistome output
[]
```

Figure 5. Example of printing empty lists as control

The whole ResistNanome pipeline can be found as figure 2 (introduction, p. 7). In this, the different aspects, or sub-projects, are made clear by different blocks/colours. The parts are: quality control; demultiplexing; Filtlong filtering; filtering to take out the host genome; filtering to keep only the reads with resistance genes; community screening and the wrapper (part of the pipeline that combines the other parts). The pipeline utilised existing tools, executable in Unix, through the Python script.

Implemented Python modules

Certain modules were needed to build the pipeline, these functions were added to standard Python by using the command "`import` …". What modules to use, were decided with use of tips from internet. How to use them could be found using the Python (3.8) documentation [19] on each module, some with extra information from developers of the modules. The modules used are: `Argparse`, `gzip`, `sys`, `os`, `multiprocessing`, `re`, `shutil`, `pysam`, `csv`, `datetime`, `threading`, `numpy` and `Biopython`.

With `argparse` [20], the user of the pipeline gets to call for arguments to be true, give them a value or give files as arguments. When the pipeline is downloaded, the help-argument (--help or -h) called after calling the pipeline (./ResistNanomeWrapper.py) will give all the arguments, how to call them and what they do. This improves the user-friendliness of the pipeline.

`Gzip` [21] is a module that is needed to either gunzip or gzip files. FastQ-files are often gzipped, because the files are very large. By gzipping files, they take up less space and can be unpacked later. Gzipped files have ".gz" at the end of the file (so example.fastq becomes example.fastq.gz).

The interpreter makes sure python is running, but to gain access to this, the module `sys` [22] is needed. With sys, it is, for example, possible to exit them or to get access to the directory another needed script is in.

Os stands for operating system. The `os` module [23] is used as a way of using the dependent functionality (the relation between two objects) of the operating system and contain functions to interact with the file system. This interaction is especially clear within the os.path sub-module, which can be used to create files and directories and can call for them. os.system is a specific function that gives Python the option to write and execute a line of code as if in the operating system, Unix in the case of ResistNanome.

`Multiprocessing` [24] is the module that allows for full usage of the multiple processors on the machine in use. This is used to set a maximum thread count the pipeline can use (16).

Regular expression, also called RegEx or `re` [25] [26] is used within multiple languages to check for specific patterns within a string. In Python it's known as the module re. Re opens the world of wildcards for Python, from random character to just letters at the start, end or anywhere in a string.

Low-level editing of files can be done with Python using `open()`, but high-level operations on (collections of) files needs `shutil` [27]. This includes coping and removal of files. Shutil will also work with .gz files.

`Pysam` [28] is a module for manipulating SAM/BAM format files in Python. SAM/BAM format is an efficient way to store alignments. Pysam needs to be installed before it can be imported. For ResistNanome, this has already been done.

`Csv` [29] or comma separated value is a type of file in which information is divided in columns, each separated by a comma, tab or a semicolon. To read and write such files using Python, the module csv is needed. Certain tools used in ResistNanome give their output in csv format, so the csv module is used to take the needed information.

The `datetime` [30] module gives the date and time when called. It is often used to create a log, likewise for ResistNanome.

`Threading` [31] is a module that gives Python the option to multithread. Multithreading gives the computer the option to run two things at the same time, without them interfering with each other. It is used to optionally run taxonomy- and resistome determination at the same time.

Python already has a range of options for mathematics, but not for advanced mathematics or scientific computing. For this, the `numpy` module was developed. The user won't have to calculate medians or average for example but calls a function to do so.

For bioinformatics in Python, the `Bio` [32] module is essential. This module enables the user to read and write FastQ and FastA files or take out specific information from each. Besides this, the complement/reverse/reverse-complement of a sequence can be created within seconds.

Implemented tools

When creating a pipeline, it's often not needed to create every part yourself. For a lot of the functions, tools already exist. (Bio-)Informatics tools/software are often made publicly available together with a general public license (GPL) that states that the original author(s) of the tool need to be credited and new software based on the existing software also needs to be available under the same license. The tools used for ResistNanome are: NanoQC (v0.5.0); Filtlong (v0.2.0); Porechop (v0.2.4); Minimap2 (v2.17-954-dirty); KMA (v1.2.16); Kraken 2 (v2.0.8-beta); Bracken; and PyFPDF (v1.7.2). These are all publicly available under a GPL and are created for a Unix operating system and are implemented in Python using `os.system`.

NanoQC is a quality check developed especially for Nanopore data [33] [34]. It makes use of the tool bokeh [35] for visualisation in interactive graphs, a total of five. The first gives the length of the reads on the x-axis and the amount of reads on y. Beneath this one, four graphs with the position (250 from start or end) on x, two give the frequency per nucleotide in in reads and the other give the mean quality. When called for QC in ResistNanome, the tool is called twice, before and after filtering with

Filtlong or Porechop. The tool (and other tools related to this one) is developed by Wouter de Coster, with help from the community [34].

Filtlong filters sets of long reads on read length and read identity [36]. Because Filtlong is developed for very long reads, it is used often to filter Nanopore data. Filtlong allows for input of thresholds, in the ResistNanome pipeline, some could be set as well. It is developed by Ryan Wick with help from his colleagues at the Holt lab.

Ryan Wick also developed Porechop [37] [38]. Porechop was originally developed for removing adapters from reads created with ONT's Nanopore sequencing. Because Porechop looks for the adapters anyway, demultiplexing was added. In ResistNanome it is used for the demultiplexing abilities. It is easy to use as a demultiplexer, because it works on FastQ data and doesn't need extra information about specific barcoding kits.

Minimap2 is an alignment tool created by Heng Li [39] [40]. It has special options for aligning Nanopore sequencing reads against large reference databases or for finding overlaps between the reads. The aligner is used to compare the reads against a vertebrate database (see appendix 3), containing DNA of the vertebrates that are most likely to have a cross-contamination within the metagenomic data. Every read that aligns with this database is taken out to keep as close to just bacteria/pathogens as possible.

KMA, created by Philip Clausen and colleagues at the Technical University of Denmark, is also an alignment tool [41] [42]. While it has the same basic function as Minimap2, KMA has an advantage; it has a unique way of calculating the degree of alignment. Each read will only get one, or at most two, alignment hits. This takes a little longer but is necessary when determining the antibiotic resistance gene. KMA is derived from k-mer alignment, the way of aligning.

Kraken 2 is a taxonomic sequence classifier for DNA sequences based on k-mers [43] [44]. Kraken 2's approaches works well for metagenomic datasets and is fast, two important features. While the memory usage is already improved with respect to its predecessor, it is important to note that Kraken 2 requires a large memory, which could be limiting. It is also important to build a database for Kraken 2, but this takes a long time and could be puzzling for someone unfamiliar with bioinformatics. For ResistNanome, the bacteria part of this database is already built and can be downloaded when downloading the pipeline. In ResistNanome Kraken 2 gives both the scientific name of the bacteria and the taxonomy ID.

As an extension of Kraken 2, Bracken is created by one of the developers who also helped create Kraken [6] [45]. Bracken takes Kraken 2 output to calculate an estimation of abundance. It can do this at different levels, which can be specified. In the pipeline, it can be specified as well, otherwise it will automatically calculate this for all levels. If all levels are calculated, the lowest level (specie) will be used when giving the output in ResistNanome. The top 10 most occurring bacteria will be put in a PDF-file, all information will be available in a csv file.

FPDF is a PHP class which allows users to create PDF files with programming language PHP [46] [47] [48]. Based on this, PyFPDF is created to do the same for Python. It is used to make a PDF from the verbose output from Porechop and to give a top 10 of most occurring bacteria based on the Bracken output.

Databases

Minimap2, Kraken 2/Bracken and KMA need a database to perform their function. These databases can be downloaded for ResistNanome with a script included in the initial download.

For Minimap2, a smaller version of two NCBI RefSeq databases, vertebrate_mammalian and vertebrate_other. To keep the filtering and download time down, only the most common host species are included. The full list of species can be found in table 2, appendix 3. This db is from December 2018.

The database for Kraken 2 can be built through Kraken 2 itself. Based on this, combined with a k-mer argument (the size of the k-mer), a database for Bracken can be built using a script provided by Bracken. This database is pre-built and can just be downloaded.

KMA makes use of the ResFinder database [42]. KMA itself has a ResFinder script for installation and use. For this as well goes that this one can be downloaded from the same place as the other databases for ResistNanome.

GitHub

To improve the accessibility of ResistNanome, it is put on GitHub. GitHub is a web-based, open-source, code project hosting service [49]. Projects on GitHub can be of a variety of programming languages and keeps track of the changes made through Git. Communication with users is another reason why a developer would want to put their (open source) project on GitHub. Communication is made very simple, so the developer can directly implement problems/suggestions from users.

Results

Building the pipeline

The pipeline went through several changes before reaching its final form, specifically determination of the resistome changed a lot, but changes in other parts worked similarly. Each time a piece of code seemed correct, a dataset was used to see if the code gave any results and if those results were as expected. If an error occurred, this was fixed within Unix, when something was either fully wrong or needed a lot of change, it had to go back to the drawing board.

The first version of resistome determination utilised Minimap 2, like used for host filtering, but the other way around (keep matches with the resistome db instead of getting rid of matches with the host db). This, however, made it hard to determine one single resistance gene as the resistance genes share a lot of DNA and substrate specificity sometimes depends only one nucleotide difference in a gene [39]. Naïve DNA read mappers therefore produce many spurious hits because they detect the high homology to the part of the resistance gene that is shared by different resistance genes of the same family.

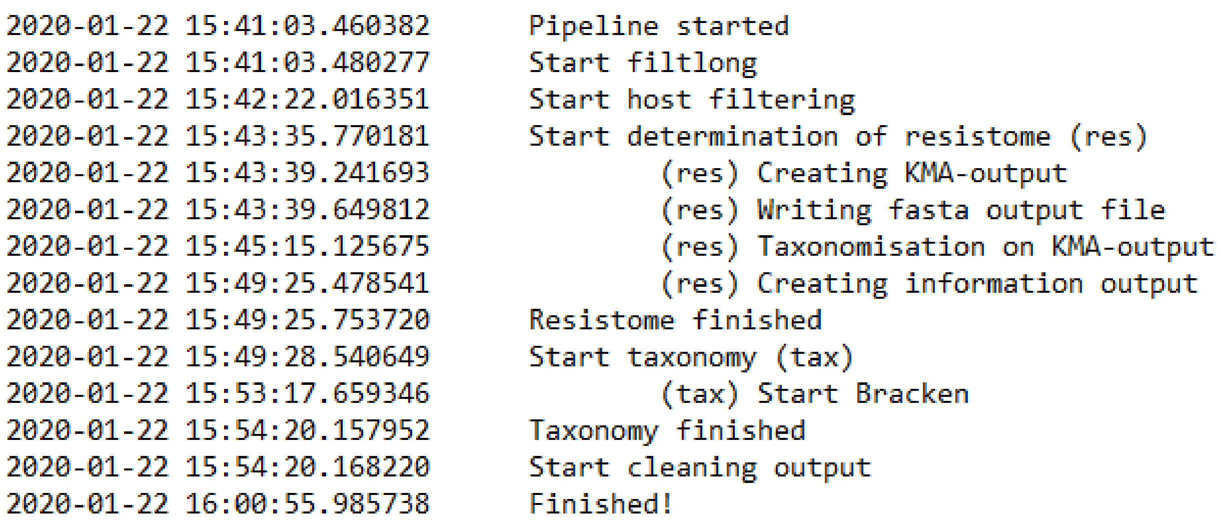
To combat this, the tool KMA was implemented, which has a different way of calculating which resistant gene fits best, finally one single gene is chosen based on best match results and local assembly of the reads. The output of this are several files with data in tab-separated format with the read-names, but unfortunately none of them is a FastQ or FastA file or has the full sequence.

Creating a FastQ/FastA file had to be done 'by hand', searching the input FastQ for all the reads that occur in the KMA output and combining only these in a new FastQ file. The first version of this took a long time, because each read in the KMA output was compared to the full input file, twice. This was first made faster by only having it loop once and building in a break* after each find. This reduced the time by a lot, but it could still take hours to write the FastQ output. This was made even faster by sorting both the input and the KMA-output on read name (the common factor). The search for the common factor works the same as before, but now it breaks after each find, notes the place of the break in the list and continues with the next read, starting at the noted breakpoint in the file. This greatly reduced the runtime as, instead of looping through the file 1000s of times (equal to the number of hits), only one loop is needed.

Generated output

ResistNanome is made publicly available through GitHub (github.com/AStieva/ResistNanome). The code of the wrapper is also available in appendix 1. The pipeline relies on (a combination of) functions to be called. No matter the functions, the pipeline will always give a log with the start- and in certain cases end-, times of the implementations of the pipeline in a .txt format (figure 6, p.15). Through this, the length of time for each part of ResistNanome can be calculated like in table 1 (p.15). The dataset this output came from is much larger than what will be used in practice, since the MinION was left to sequence for two to three times as long as the six hours adviced for veterinarians. The time ResistNanome will take could differ based on the size of the file and the computing power of the computer used. When using data generated over six hours with the same computer used in figure 6, the total time will be less. The taxonomy level used was just specie, when using the full list, this time will go up.

* Break in Python breaks the inner loop. In this case, it stopped searching for the read after it was found and continued with the next.



```
2020-01-22 15:41:03.460382      Pipeline started
2020-01-22 15:41:03.480277      Start filtlong
2020-01-22 15:42:22.016351      Start host filtering
2020-01-22 15:43:35.770181      Start determination of resistome (res)
2020-01-22 15:43:39.241693              (res) Creating KMA-output
2020-01-22 15:43:39.649812              (res) Writing fasta output file
2020-01-22 15:45:15.125675              (res) Taxonomisation on KMA-output
2020-01-22 15:49:25.478541              (res) Creating information output
2020-01-22 15:49:25.753720      Resistome finished
2020-01-22 15:49:28.540649      Start taxonomy (tax)
2020-01-22 15:53:17.659346              (tax) Start Bracken
2020-01-22 15:54:20.157952      Taxonomy finished
2020-01-22 15:54:20.168220      Start cleaning output
2020-01-22 16:00:55.985738      Finished!
```

Figure 4. ResistNanome log of a Nanopore run by A.B. with barcode 19, gzipped fastq file of ±2,4 GB

Table 1. Time ResistNanome takes to perform most of the possible tasks based on figure 6

Part	Time (m:s)
Filtlong	1:19
Host filter	1:13
Resistome determination	5:50
• **KMA**	• 00:01
• **Creating FastA/FastQ**	• 1:36
• **Taxonomisation**	• 4:10
• **Writing output**	• 00:01
Taxonomy	4:51
• **Kraken**	• 3:48
• **Bracken**	• 1:03
Rest	6:36
Total	**19:53**

Though not in this example, the time NanoQC uses will be put in as a whole. NanoQC gives its own log, but this one is not necessarily kept, except when keeping all data.

As output, NanoQC creates five graphs shown in an html-file (figure 7, p.14). These files are renamed based on the point of execution (QC1/QC2 for before/after Filtlong) and, if needed, the barcode given by Porechop. The graphs give the current quality of the data in a visually comprehensive way and won't change it.

Of the graphs in the output, one (A) gives the read length compared to the number of reads per length on the X and Y axis respectively. The next two (B) give the 200 positions from start/end on the X-axis and for each of these the frequency in percentage of each nucleotide on the Y-axis. The last two (C) got the same values on the X-axis with on the Y-axis the mean quality of the base call/Phred score.

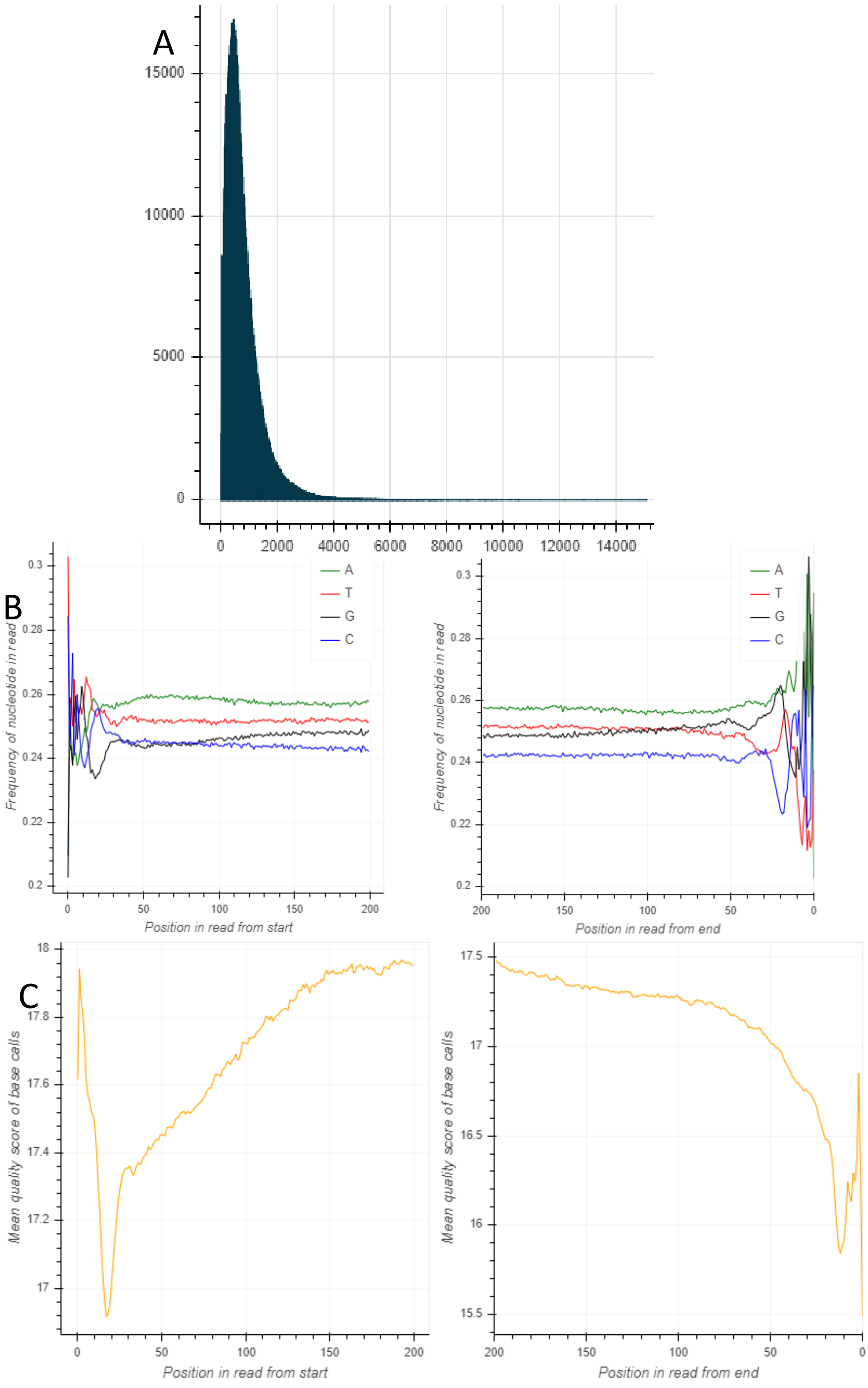
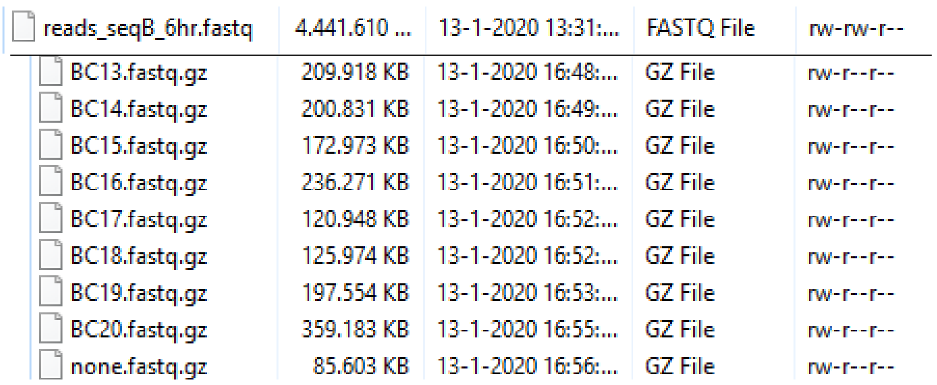


Figure 5. NanoQC output before Filtlong. A) The length of the read (x) to the amount of reads (y) B) Nucleotide frequency of the reads (first/last 200bp) C) Mean quality of the reads (first/last 200bp)

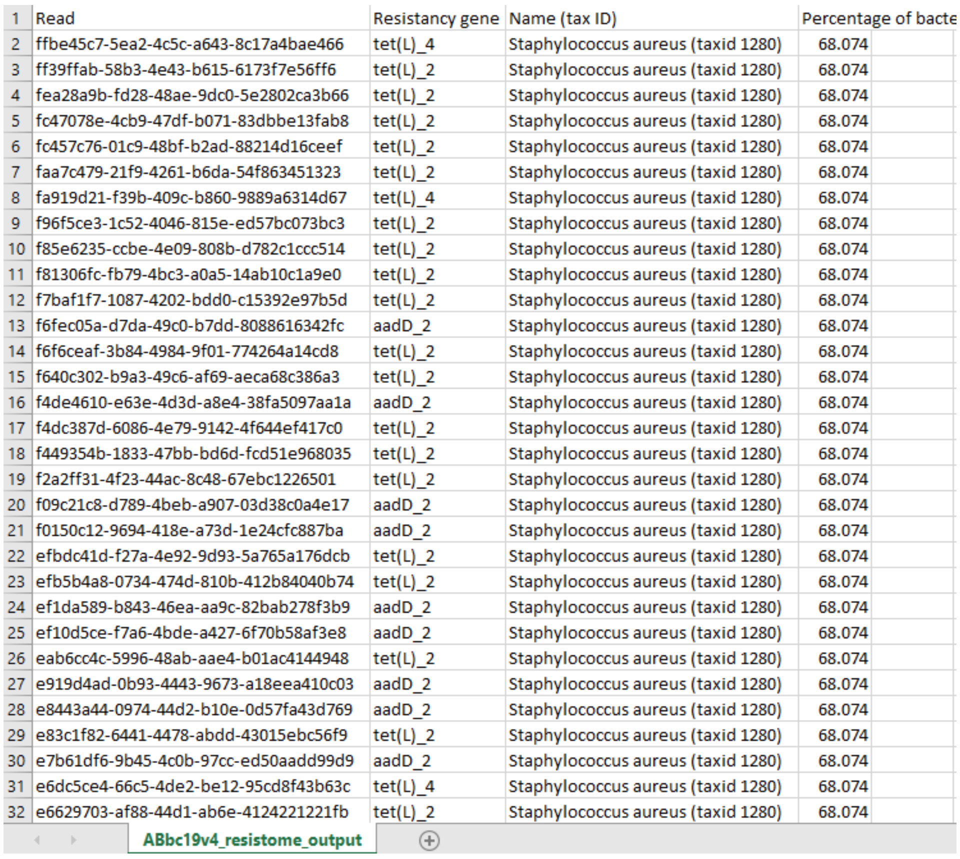
When running Porechop, it gives an output like the one in appendix 4 in PDF format. Porechop will, most importantly, divide the reads based on barcode, like in figure 8. This divided output will be in a sub-folder within the output folder. Each of these output files can be ran through the rest of the pipeline.



reads_seqB_6hr.fastq	4.441.610 ...	13-1-2020 13:31:...	FASTQ File	rw-rw-r--
BC13.fastq.gz	209.918 KB	13-1-2020 16:48:...	GZ File	rw-r--r--
BC14.fastq.gz	200.831 KB	13-1-2020 16:49:...	GZ File	rw-r--r--
BC15.fastq.gz	172.973 KB	13-1-2020 16:50:...	GZ File	rw-r--r--
BC16.fastq.gz	236.271 KB	13-1-2020 16:51:...	GZ File	rw-r--r--
BC17.fastq.gz	120.948 KB	13-1-2020 16:52:...	GZ File	rw-r--r--
BC18.fastq.gz	125.974 KB	13-1-2020 16:52:...	GZ File	rw-r--r--
BC19.fastq.gz	197.554 KB	13-1-2020 16:53:...	GZ File	rw-r--r--
BC20.fastq.gz	359.183 KB	13-1-2020 16:55:...	GZ File	rw-r--r--
none.fastq.gz	85.603 KB	13-1-2020 16:56:...	GZ File	rw-r--r--

Figure 6. Example of demultiplexing as done by Porechop. Top is the input, bottom the output

Determining the resistome is the most important function of ResistNanome. For each read, this part of the tool will give the read-ID, the resistance gene, the name and ID of the determined bacterium and the percentage of this bacterium with respect to the combination of reads. All the information is stored in tab seperated format text file. Only one, or at most two, resistance genes are given per read (see figure 9). The top 10 of these, based on bacteria occurance, are put in a PDF file. The same goes for the taxonomy output of all the (filtered) reads, par the resistancy gene. The two top 10 lists are combined to one PDF file (see appendix 5, same dataset), if both are determined. These top 10 lists and the full list are created based on multiple outputs to show the most important data. The user does not have to go through multiple files to find what resistancy genes corrispond to what bacteria.

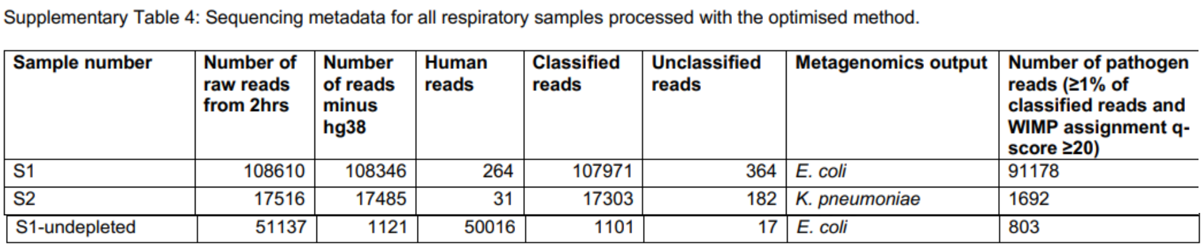


1	Read	Resistancy gene	Name (tax ID)	Percentage of bacte
2	ffbe45c7-5ea2-4c5c-a643-8c17a4bae466	tet(L)_4	Staphylococcus aureus (taxid 1280)	68.074
3	ff39ffab-58b3-4e43-b615-6173f7e56ff6	tet(L)_2	Staphylococcus aureus (taxid 1280)	68.074
4	fea28a9b-fd28-48ae-9dc0-5e2802ca3b66	tet(L)_2	Staphylococcus aureus (taxid 1280)	68.074
5	fc47078e-4cb9-47df-b071-83dbbe13fab8	tet(L)_2	Staphylococcus aureus (taxid 1280)	68.074
6	fc457c76-01c9-48bf-b2ad-88214d16ceef	tet(L)_2	Staphylococcus aureus (taxid 1280)	68.074
7	faa7c479-21f9-4261-b6da-54f863451323	tet(L)_2	Staphylococcus aureus (taxid 1280)	68.074
8	fa919d21-f39b-409c-b860-9889a6314d67	tet(L)_4	Staphylococcus aureus (taxid 1280)	68.074
9	f96f5ce3-1c52-4046-815e-ed57bc073bc3	tet(L)_2	Staphylococcus aureus (taxid 1280)	68.074
10	f85e6235-ccbe-4e09-808b-d782c1ccc514	tet(L)_2	Staphylococcus aureus (taxid 1280)	68.074
11	f81306fc-fb79-4bc3-a0a5-14ab10c1a9e0	tet(L)_2	Staphylococcus aureus (taxid 1280)	68.074
12	f7baf1f7-1087-4202-bdd0-c15392e97b5d	tet(L)_2	Staphylococcus aureus (taxid 1280)	68.074
13	f6fec05a-d7da-49c0-b7dd-8088616342fc	aadD_2	Staphylococcus aureus (taxid 1280)	68.074
14	f6f6ceaf-3b84-4984-9f01-774264a14cd8	tet(L)_2	Staphylococcus aureus (taxid 1280)	68.074
15	f640c302-b9a3-49c6-af69-aeca68c386a3	tet(L)_2	Staphylococcus aureus (taxid 1280)	68.074
16	f4de4610-e63e-4d3d-a8e4-38fa5097aa1a	aadD_2	Staphylococcus aureus (taxid 1280)	68.074
17	f4dc387d-6086-4e79-9142-4f644ef417c0	tet(L)_2	Staphylococcus aureus (taxid 1280)	68.074
18	f449354b-1833-47bb-bd6d-fcd51e968035	tet(L)_2	Staphylococcus aureus (taxid 1280)	68.074
19	f2a2ff31-4f23-44ac-8c48-67ebc1226501	tet(L)_2	Staphylococcus aureus (taxid 1280)	68.074
20	f09c21c8-d789-4beb-a907-03d38c0a4e17	aadD_2	Staphylococcus aureus (taxid 1280)	68.074
21	f0150c12-9694-418e-a73d-1e24cfc887ba	aadD_2	Staphylococcus aureus (taxid 1280)	68.074
22	efbdc41d-f27a-4e92-9d93-5a765a176dcb	tet(L)_2	Staphylococcus aureus (taxid 1280)	68.074
23	efb5b4a8-0734-474d-810b-412b84040b74	tet(L)_2	Staphylococcus aureus (taxid 1280)	68.074
24	ef1da589-b843-46ea-aa9c-82bab278f3b9	aadD_2	Staphylococcus aureus (taxid 1280)	68.074
25	ef10d5ce-f7a6-4bde-a427-6f70b58af3e8	aadD_2	Staphylococcus aureus (taxid 1280)	68.074
26	eab6cc4c-5996-48ab-aae4-b01ac4144948	tet(L)_2	Staphylococcus aureus (taxid 1280)	68.074
27	e919d4ad-0b93-4443-9673-a18eea410c03	aadD_2	Staphylococcus aureus (taxid 1280)	68.074
28	e8443a44-0974-44d2-b10e-0d57fa43d769	aadD_2	Staphylococcus aureus (taxid 1280)	68.074
29	e83c1f82-6441-4478-abdd-43015ebc56f9	tet(L)_2	Staphylococcus aureus (taxid 1280)	68.074
30	e7b61df6-9b45-4c0b-97cc-ed50aadd99d9	aadD_2	Staphylococcus aureus (taxid 1280)	68.074
31	e6dc5ce4-66c5-4de2-be12-95cd8f43b63c	tet(L)_4	Staphylococcus aureus (taxid 1280)	68.074
32	e6629703-af88-44d1-ab6e-4124221221fb	tet(L)_2	Staphylococcus aureus (taxid 1280)	68.074

Figure 7. Part of the reads, opened in excel. The same input as the input from figure 6

Example dataset

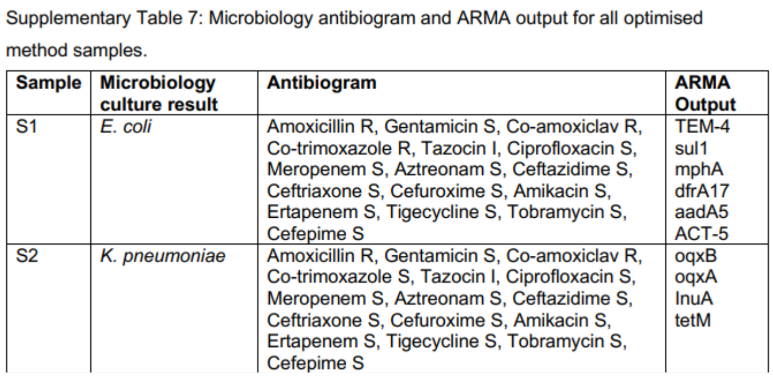
To test how well the pipeline works, a few specific data sets were put through it. The datasets came from the European Nucleotide Archive [50] and were generated by and used in a research paper; “Nanopore metagenomics enables rapid clinical diagnosis of bacterial lower respiratory infection” [51]. Datasets S1, S2 and S1-undepleted (without filtering out the host-genome), when ran through ResistNanome do give the same culture result as most common bacterium, *E. coli*, *K. pneumoniae* and *Escherichia* respectively, as found in aforementioned research. Both S1 and S2 seem to also give the same antibiotic resistance genes as the control, though not all are visible in the top 10. The full output from ResistNanome was compared to the information stored in supplementary table 4 and 7 (figure 10 and 11 respectively) in the research paper. The top 10 output from ResistNanome (resistome and taxonomy for the undepleted) are visible in figures 12 – 14.



Supplementary Table 4: Sequencing metadata for all respiratory samples processed with the optimised method.

Sample number	Number of raw reads from 2hrs	Number of reads minus hg38	Human reads	Classified reads	Unclassified reads	Metagenomics output	Number of pathogen reads (≥1% of classified reads and WIMP assignment q-score ≥20)
S1	108610	108346	264	107971	364	*E. coli*	91178
S2	17516	17485	31	17303	182	*K. pneumoniae*	1692
S1-undepleted	51137	1121	50016	1101	17	*E. coli*	803

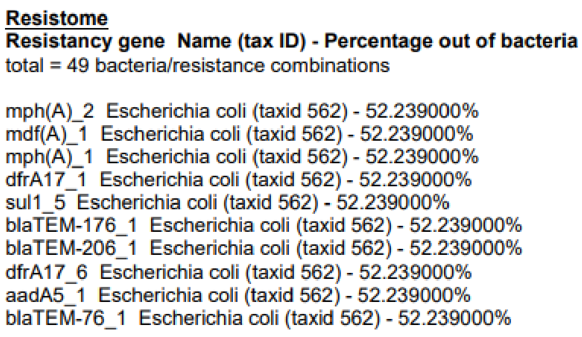
Figure 10. Part of supplementary table 4 from the article



Supplementary Table 7: Microbiology antibiogram and ARMA output for all optimised method samples.

Sample	Microbiology culture result	Antibiogram	ARMA Output
S1	*E. coli*	Amoxicillin R, Gentamicin S, Co-amoxiclav R, Co-trimoxazole R, Tazocin I, Ciprofloxacin S, Meropenem S, Aztreonam S, Ceftazidime S, Ceftriaxone S, Cefuroxime S, Amikacin S, Ertapenem S, Tigecycline S, Tobramycin S, Cefepime S	TEM-4 sul1 mphA dfrA17 aadA5 ACT-5
S2	*K. pneumoniae*	Amoxicillin R, Gentamicin S, Co-amoxiclav R, Co-trimoxazole S, Tazocin I, Ciprofloxacin S, Meropenem S, Aztreonam S, Ceftazidime S, Ceftriaxone S, Cefuroxime S, Amikacin S, Ertapenem S, Tigecycline S, Tobramycin S, Cefepime S	oqxB oqxA InuA tetM

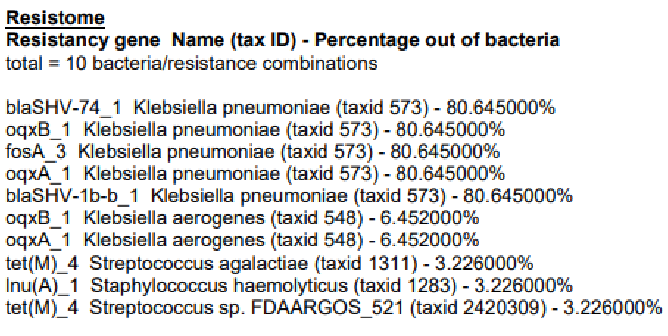
Figure 11. Part of supplementary table 7 from the article



```
Resistome
Resistancy gene  Name (tax ID) - Percentage out of bacteria
total = 49 bacteria/resistance combinations

mph(A)_2  Escherichia coli (taxid 562) - 52.239000%
mdf(A)_1  Escherichia coli (taxid 562) - 52.239000%
mph(A)_1  Escherichia coli (taxid 562) - 52.239000%
dfrA17_1  Escherichia coli (taxid 562) - 52.239000%
sul1_5  Escherichia coli (taxid 562) - 52.239000%
blaTEM-176_1  Escherichia coli (taxid 562) - 52.239000%
blaTEM-206_1  Escherichia coli (taxid 562) - 52.239000%
dfrA17_6  Escherichia coli (taxid 562) - 52.239000%
aadA5_1  Escherichia coli (taxid 562) - 52.239000%
blaTEM-76_1  Escherichia coli (taxid 562) - 52.239000%
```

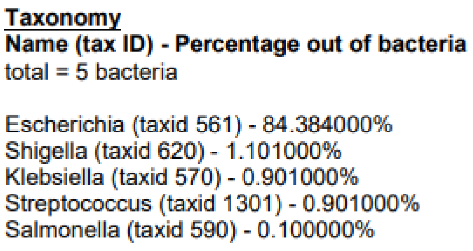
Figure 12. ResistNanome PDF-output of S1



```
Resistome
Resistancy gene  Name (tax ID) - Percentage out of bacteria
total = 10 bacteria/resistance combinations

blaSHV-74_1  Klebsiella pneumoniae (taxid 573) - 80.645000%
oqxB_1  Klebsiella pneumoniae (taxid 573) - 80.645000%
fosA_3  Klebsiella pneumoniae (taxid 573) - 80.645000%
oqxA_1  Klebsiella pneumoniae (taxid 573) - 80.645000%
blaSHV-1b-b_1  Klebsiella pneumoniae (taxid 573) - 80.645000%
oqxB_1  Klebsiella aerogenes (taxid 548) - 6.452000%
oqxA_1  Klebsiella aerogenes (taxid 548) - 6.452000%
tet(M)_4  Streptococcus agalactiae (taxid 1311) - 3.226000%
lnu(A)_1  Staphylococcus haemolyticus (taxid 1283) - 3.226000%
tet(M)_4  Streptococcus sp. FDAARGOS_521 (taxid 2420309) - 3.226000%
```

Figure 13. ResistNanome PDF-output of S2



```
Taxonomy
Name (tax ID) - Percentage out of bacteria
total = 5 bacteria

Escherichia (taxid 561) - 84.384000%
Shigella (taxid 620) - 1.101000%
Klebsiella (taxid 570) - 0.901000%
Streptococcus (taxid 1301) - 0.901000%
Salmonella (taxid 590) - 0.100000%
```

Figure 1. ResistNanome PDF-output of S1-undepleted

Conclusion and discussion

ResistNanome is a working pipeline by now. To make ResistNanome accessible for a wider audience, it is put online on GitHub. The full code, tools, etc. can be looked at and/or downloaded here: https://github.com/AStieva/ResistNanome. The GNU license (appendix 2) is added to open it up to the public with a form of protection.

ResistNanome can determine the antibiotic resistant genes in metagenomic data with help of KMA [42], so the most basic goal was reached. The output from ResistNanome is slightly different from the reference sets. The undepleted set was on another level (genus in ResistNanome, specie in test) and not all resistance genes were as well-represented in the top 10 compared to the reference.

The output is generated in less than 20 minutes, together with the six hours of sequencing this is still quicker than the three days it takes to grow the bacteria in a laboratory culture and within a day. The output has been altered from the standard output of the tools, to make it easy to get the most important information right away. Besides this, how to implement the tool, when connected to Unix, is explained in detail on the GitHub page/in the README that is provided when downloading ResistNanome.

The reads containing the genes are classified based on a bacteria database with help of Kraken 2 [44] and Bracken [45]. This classification can also be done on all the data, either filtered or not. Filtering can be carried out before taxonomisation and/or resistome determination. Filtering could be based on read quality/length with Filtlong [36] and/or on which reads are perceived as vertebrate DNA, with help of Minimap2 [40] and a vertebrate database.

There are multiple extra options to improve the quality of the output. However, these make for a longer run-time. NanoQC [34], which gives the quality of the reads, takes about 20 minutes extra per time it runs (two times when called together with Filtlong or the number of found barcodes + two times withPorechop). ResistNanome gives the option of masking the gene in the read. Masking the gene will make sure the community screening won't flag reads unfairly as the origin bacteria of the gene. However, this takes a really long time: Up to 2 and a half day for a gzipped FastQ file of a little over 3 GB in size (this was only tested with a larger test-dataset). This makes the masking unusable for the purposes, since the improvement of the quality doesn't weigh up against the loss of time.

Recommendations

The ResistNanome pipeline works for everything, but there are some possible improvements that can be added to the pipeline, or to make it work better.

Firstly, it would be easier for users to not have to install bokeh themselves to use NanoQC. The two possible solutions for this would be either to make bokeh into an executable, like the other tools used, and rewrite a bit of NanoQC to refer to this. The other option is to replace NanoQC with another tool or piece of code that doesn't use any extras.

A small change that could be implemented for clarity reasons, is in the top 10 lists. Right now, they are just strings, one after the other, but it might be more legible when the stings are cut up and put in a table instead. Like the tab separated file when put in excel.

Another change in the top 10 list could be to add how often a combination of bacteria and antibiotic resistance gene occurs. Right now, the top 10 of antibiotic resistance only gives the top 10 based on bacteria.

The information output for demultiplexing, as done by Porechop, is clear, with the different colours and underlining. These colours are determined by the output but are implemented by a form of coding which works for Unix output. The way of rewriting it to PDF makes that it is pure coding, without the colours. The colours are added later, but the coding around it is still visible and it would be prettier and less confusing if the coding was to be taken away.

The pipeline might also benefit from some added functions. Right now, the pipeline only takes FastQ files. It could be useful in the future to have Fast5 or FastA files as input directly. Fast5 is impossible right now because almost no tool can work with this format. FastA might be possible, though some tools won't take it, due to not giving additional data. Being able to take these formats will make the pipeline more flexible and could make it so ResistNanome could be used on existing datasets.

Porechop, now only used for demultiplexing, was originally created for trimming the ends. It might be interesting to add the trimming part of Porechop besides/instead of Filtlong. The option of demultiplexing might be redundant in a small wile, since the interpreter, minIT, makes use of Guppy. Guppy has recently gotten the option to demultiplex the output in real time.

The last addition might be adding an assembly. The suggested assembler for this would be Flye [52]. This has options specific for metagenomic datasets as well as being created to work well with Nanopore data. When using an assembly prior to taxonomy determination, the last can be done more effective.

Acknowledgements

I would like to thank everyone that made contributions to my thesis. I particularly want to thank my mentors, Dr. A. L. Zomer and M. Hensmann.

Dr. A. L. Zomer, Aldert, I would like to thank you for giving me the opportunity of joining you at KLIF for this internship. I felt very welcome and respected by you and the rest of the staff, a very good overall atmosphere. I would also like to thank you for the guidance you provided during the project. You were very patient when I couldn’t understand something right away and trusted in my competences, even when I didn’t always myself.

Meike Hensmann, thank you very much for your help during this internship and the way leading to it. You listened to me and to where my passion lies within this field of profession. I would also like to thank you, together with the BML internship coordinator for allowing me to do a bioinformatics internship.

Other than this, I’d like to thank my co-workers and fellow interns for the great work environment. Especially Connor Twomey and Alejandro Baars, who provided sets of data for me to work with.

Bibliography

[1] WHO - World Health Organization, “Antibiotic resistance,” 5 February 2018. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/antibiotic-resistance. [Accessed 9 12 2019].

[2] RIVM - RijksInstituut voor Volksgezondheid en Millieu, “RIVM - Antimicrobial resistance,” 28 11 2019. [Online]. Available: https://www.rivm.nl/en/antimicrobial-resistance. [Accessed 9 12 2019].

[3] National Research Council (US) Committee on Metagenomics: Challenges and Functional Applications., The New Science of Metagenomics: Revealing the Secrets of Our Microbial Planet., Washington (DC): National Academies Press (US), 2007.

[4] E. J. Stewart, “Growing Unculturable Bacteria,” *Journal of bacteriology,* vol. 194, no. 16, pp. 4151 - 4160, 8 2012.

[5] D. E. L. J. L. B. Wood, “bioRxiv 762302;Improved metagenomic analysis with Kraken 2,” 01 01 2019. [Online]. Available: https://www.biorxiv.org/content/10.1101/762302v1. [Accessed 7 1 2020].

[6] J. Lu, F. P. Breitwieser, P. Thielen and S. L. Salzberg, “Bracken: estimating species abundance in metagenomics data,” *PeerJ Computer Science,* vol. 3, no. 104, 2 1 2017.

[7] J. Schoales, “How Does Sanger Sequencing Work?,” ThermoFisher scientific, 17 6 2015. [Online]. Available: https://www.thermofisher.com/blog/behindthebench/how-does-sanger-sequencing-work/. [Accessed 11 12 2019].

[8] S. Behjati and P. S. Tarpey, “NCBI PMC - What is next generation sequenching?,” 28 August 2013. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3841808/. [Accessed 11 12 2019].

[9] J. M. Heather and B. Chain, “The sequence of sequencers: The history of sequencing DNA,” *Genomics,* vol. 107, no. 1, pp. 1-8, January 2016.

[10] Oxford Nanopore Technologies, “MinION and flow cells,” Oxford Nanopore Technologies, 9 10 2019. [Online]. Available: https://community.nanoporetech.com/technical_documents/hardware/v/hwtd_5000_v1_revk_03may2016/the-minion-mk-i. [Accessed 17 12 2019].

[11] Oxford Nanopore Technologies, “Nanopore sensing - how it works,” Oxford Nanopore Technologies, 7 June 2018. [Online]. Available: https://community.nanoporetech.com/technical_documents/nanopore-sensing/v/nstd_5000_v1_revg_04apr2016. [Accessed 17 12 2019].

[12] J. McKellar, “A hands-on introduction to Python for beginning programmers,” Santa Clara, 2013.

[13] Python Software Foundation, “What is Python? Executive Summary,” Python Software Foundation, [Online]. Available: https://www.python.org/doc/essays/blurb/. [Accessed 18 12 2019].

[14] Python Software Foundation, “Python docs on Modules,” 30 4 2019. [Online]. Available: https://docs.python.org/3/tutorial/modules.html. [Accessed 18 12 2019].

[15] P. J. A. Cock, T. Antao, J. T. Chang, B. A. Chapman, C. J. Cox, A. Dalke, I. Friedberg, T. Hamelryck, F. Kauff, B. Wilczynski and M. J. L. de Hoon, “Biopython: freely available Python tools for computational molecular biology and bioinformatics,” *Bioinformatics,* vol. 25, no. 11, pp. 1422 - 1423, 20 3 2009.

[16] C. Ramey, Interviewee, *The A-Z of Programming Languages: BASH/Bourne-Again Shell.* [Interview]. 30 5 2008.

[17] S. Vermeulen, “Fast and easy metagenomic data analysis with FastDeMe,” 2019.

[18] m. developers, “Stack Overflow,” Stack Exchange Inc, [Online]. Available: https://stackoverflow.com/. [Accessed 6 1 2020].

[19] Python Software Foundation, “Python 3.8.1 documentation,” Python Software Foundation, [Online]. Available: https://docs.python.org/3/. [Accessed 6 1 2020].

[20] Python Software Foundation, “argparse — Parser for command-line options, arguments and sub-commands,” Python Software Foundation, [Online]. Available: https://docs.python.org/3/library/argparse.html. [Accessed 7 1 2020].

[21] Python Software Foundation, “gzip — Support for gzip files,” Python Software Foundation, [Online]. Available: https://docs.python.org/3/library/gzip.html. [Accessed 7 1 2020].

[22] Python Software Foundation, “sys — System-specific parameters and functions,” Python Software Foundation, [Online]. Available: https://docs.python.org/3/library/sys.html#module-sys. [Accessed 7 1 2020].

[23] Python Software Foundation, “os — Miscellaneous operating system interfaces,” Python Software Foundation, [Online]. Available: https://docs.python.org/3/library/os.html. [Accessed 7 1 2020].

[24] Python Software Foundation, “multiprocessing — Process-based parallelism,” Python Software Foundation, [Online]. Available: https://docs.python.org/3.8/library/multiprocessing.html. [Accessed 7 1 2020].

[25] Python Software Foundation, “re — Regular expression operations,” Python Software Foundation, [Online]. Available: https://docs.python.org/3/library/re.html. [Accessed 7 1 2020].

[26] w3schools, “Python RegEx,” w3schools, [Online]. Available: https://www.w3schools.com/python/python_regex.asp. [Accessed 7 1 2020].

[27] Python Software Foundation, “shutil — High-level file operations,” Python Software Foundation, [Online]. Available: https://docs.python.org/3/library/shutil.html#module-shutil. [Accessed 7 1 2020].

[28] A. Heger, K. Jacobs and contributors, “pysam: htslib interface for python,” pysam developers, 27 7 2018. [Online]. Available: https://pysam.readthedocs.io/en/latest/index.html. [Accessed 7 1 2020].

[29] Python Software Foundation, “csv — CSV File Reading and Writing,” Python Software Foundation, [Online]. Available: https://docs.python.org/3/library/csv.html. [Accessed 7 1 2020].

[30] Python Software Foundation, “datetime — Basic date and time types,” Python Software Foundation, [Online]. Available: https://docs.python.org/3/library/datetime.html. [Accessed 7 1 2020].

[31] J. Anderson, “An Intro to Threading in Python,” Real Python, 25 3 2019. [Online]. Available: https://realpython.com/intro-to-python-threading/. [Accessed 7 1 2020].

[32] Biopython, “Biopython,” Biopython, 20 12 2019. [Online]. Available: https://biopython.org/. [Accessed 7 1 2020].

[33] W. de Coster, S. D'Hert, D. T. Schultz, M. Cruts and C. van Broeckhoven, “NanoPack: visualizing and processing long-read sequencing data,” *Bioinformatics,* vol. 34, no. 15, pp. 2666-2669, 1 8 2018.

[34] wdecoster, “nanoQC github page,” 11 9 2019. [Online]. Available: https://github.com/wdecoster/nanoQC. [Accessed 7 1 2020].

[35] bokeh, “bokeh github page,” 24 12 2019. [Online]. Available: https://github.com/bokeh/bokeh. [Accessed 7 1 2020].

[36] rrwick, “Filtlong github page,” 15 5 2019. [Online]. Available: https://github.com/rrwick/Filtlong. [Accessed 7 1 2020].

[37] R. R. Wick, L. M. Judd, C. L. Gorrie and K. E. Holt, “Completing bacterial genome assemblies with multiplex MinION sequencing,” *Microbal Genomics,* vol. 3, no. 10, 14 9 2017.

[38] rrwick, “Porechop github page,” 19 10 2018. [Online]. Available: https://github.com/rrwick/Porechop. [Accessed 7 1 2020].

[39] L. Heng, “Minimap2: pairwise alignment for nucleotide sequences,” *Bioinformatics,* vol. 34, no. 18, pp. 3094 - 3100, 15 9 2018.

[40] lh3, “minimap2 github page,” 6 2019. [Online]. Available: https://github.com/lh3/minimap2. [Accessed 7 1 2020].

[41] P. T. L. C. Clausen, F. M. Aarestrup and O. Lund, “Rapid and precise alignment of raw reads against redundant databases with KMA,” *BMC Bioiformatics,* vol. 19, no. 307, 29 8 2018.

[42] "KMA bitbucket page," 3 1 2020. [Online]. Available: https://bitbucket.org/genomicepidemiology/kma/src/master/. [Accessed 7 1 2020].

[43] D. E. Wood, J. Lu and B. Langmead, "Improved metagenomic analysis with Kraken 2," *Genome Biology,* vol. 20, no. 257, 28 11 2019.

[44] DerrickWood, "Kraken2 github page," 12 2019. [Online]. Available: https://github.com/DerrickWood/kraken2. [Accessed 7 1 2020].

[45] jenniferlu717, "Bracken github page," 8 2019. [Online]. Available: https://github.com/jenniferlu717/Bracken. [Accessed 7 1 2020].

[46] R. Kharin, "FPDF for Python," 15 8 2012. [Online]. Available: https://pyfpdf.readthedocs.io/en/latest/. [Accessed 7 1 2020].

[47] reingart, "pyfpdf github page," 26 1 2018. [Online]. Available: https://github.com/reingart/pyfpdf. [Accessed 7 1 2020].

[48] Setasign, "FPDF github page," 7 12 2019. [Online]. Available: https://github.com/Setasign/FPDF. [Accessed 7 1 2020].

[49] L. Bradford, "What Is GitHub, and Why Should I Use It?," the balance careers, 30 5 2019. [Online]. Available: https://www.thebalancecareers.com/what-is-github-and-why-should-i-use-it-2071946. [Accessed 21 1 2020].

[50] ENA, "ENA - study: PRJEB30781," European Nucleotide Archive (ENA), 14 01 2019. [Online]. Available: https://www.ebi.ac.uk/ena/data/view/PRJEB30781. [Accessed 22 1 2020].

[51] T. Charalampous, G. L. Kay, H. Richardson, A. Aydin, R. Baldan, C. Jeanes, D. Rae, S. Grundy, D. J. Turner, J. Wain, R. M. Leggett, D. M. Livermore and J. O'Grady, "Nanopore metagenomics enables rapid clinical diagnosis of bacterial lower respiratory infection," *Nature biotechnology,* vol. 37, pp. 783-792, 24 6 2019.

[52] M. Kolmogorov, M. Rayko, J. Yuan, E. Polevikov and P. Pevzner, "metaFlye: scalable long-read metagenome assembly using repeat graphs," *bioRxiv,* 15 5 2019.

[53] pysam-developers, "pysam github page," 27 3 2019. [Online]. Available: https://github.com/pysam-developers/pysam. [Accessed 7 1 2020].

[54] SURFsara, "SURFsara algemene voorwaarden," 1 May 2016. [Online]. Available: https://www.surf.nl/files/2019-03/surfsara_algemene_voorwaarden.pdf.

[55] Oxford Nanopore Technologies, *Sequencing DNA (or RNA) | Real-time, Ultra Long-Reads, Scalable Technology from Oxford Nanopore,* 2017.

[56] Python Software Foundation, "About the Python Software Foundation," [Online]. Available: https://www.python.org/psf/about/.

[57] H. Krehenwinkel, A. Pomerantz, J. B. Henderson, S. R. Kennedy, J. Y. Lim, V. Swamy, J. D. Shoobridge, N. Graham, N. H. Patel, R. G. Gillespie and S. Prost, "Nanopore sequencing of long ribosomal DNA amplicons enables portable and simple biodiversity assessments with

high phylogenetic resolution across broad taxonomic scale," *GigaScience,* vol. 8, no. 5, 5 3 2019.

Appendix

1. ResistNanome wrapper script

```
#!/usr/bin/python3
# The big one, used for calling all the different parts and making sure everything runs smoothly

import argparse, gzip, sys, os, multiprocessing, re, shutil, pysam, csv, datetime, threading
from numpy import median
from Bio import SeqIO



# Setting a default maximum of threads, even if maximum is higher

def cpu_threads(max_threads):
  if multiprocessing.cpu_count() > max_threads:
    return max_threads
  else:
    return multiprocessing.cpu_count()



# Arguments to be put in by user. They may not be in the correct order, some may need to be split in different
options

parser = argparse.ArgumentParser(description = "A pipeline to make it easy to extract the resistome (and other
                                               information) from a set of metagenomic Nanopore data")
parser.add_argument("--inp", "-i",
                    action = "store",
                    required = True,
                    help = "Place here the input files, in fastq or fastq.gz format")
parser.add_argument("--outdir", "-o",
                    action = "store",
                    required = True,
                    help = "Set the output directory, please make sure to input the whole path")
parser.add_argument("--prefix", "-p",
                    action = "store",
                    help = "Set the prefix for all output files, default is the name of the input files. This
                           name can't start with 'temp_' as this is used to define temporary data")
parser.add_argument("--threads", "-t",
                    default = cpu_threads(16),
                    required = False,
                    type = int,
                    help = "Set the number of threads used, default is maximum available up to 16 threads")
parser.add_argument("--keep",
                    action = "store_true",
                    default = False,
                    help = "If called, all the intermediate data is also kept. Otherwise only outputs with new
```

```
                           information wil be kept")
parser.add_argument("--demux",
                    action = "store_true",
                    default = False,
                    help = "Execute demultiplexing with Porechop, the verbose will be saved to a pdf file")
parser.add_argument("--filtlong", "-fl",
                    action = "store_true",
                    default = False,
                    required = False,
                    help = "Execute filtlong, add --minlen [bp] to add a custom minimum length of reads,
                           default for this is 1000")
parser.add_argument("--minlen",
                    action = "store",
                    default = 1000,
                    required = False,
                    help = "The option to put in the minimum length of the reads saved in bp, default is 1000.
                           Needs filtlong to be used")
parser.add_argument("--QC", "-qc",
                    action = "store_true",
                    default = False,
                    help = "Execute the quality control")
parser.add_argument("--host",
                    action = "store_true",
                    default = False,
                    help = "Execute the host contamination screening, the database has a lot of vertebrate, but
                           if yours isn't in it, please feel free to add it")
parser.add_argument("--lvl",
                    action = "store",
                    default = ["S", "G", "F", "O", "C", "P", "K"],
                    required = False,
                    help = "The classification level for determining the abundance levels with Bracken. Default
                           is all."
                           "Options are: K (kingdom level), P (phylum), C (class), O (order), F (family),
                           G (genus), and S (species)")
parser.add_argument("--resistome", "-ar",
                    action = "store_true",
                    default = False,
                    help = "Execute the resistome analysis")
parser.add_argument("--repN", "-N",
                    action="store_true",
                    default=False,
                    required=False,
                    help="Replace the resistancy gene with a series of N. This can take a long time")
parser.add_argument("--phred",
                    action="store",
                    default=0,
                    required=False,
                    help="The option of giving a minimum phred score for resistome filtering")
parser.add_argument("--taxonomy", "-cs",
                    action = "store_true",
                    default = False,
                    help = "Execute the bacterial community screening, output set with --lvl")
parser.add_argument("--threading",
                    default = False,
                    required = False,
                    action = "store_true",
                    help = "Option to turn on threading. Both taxonomy and resistome will run at the same time,
                    it uses more memory but is faster")
parser.add_argument("--gz", "-gz",
                    action = "store_true",
                    default = False,
                    help = "gzip-ing the last input file")
args = parser.parse_args()




# If filtlong is not called, but the minlen is, it won't just run without filtlong


if args.minlen is not 1000 and args.filtlong == False:
    print("A minimum length has been passed, but the filtlong step is not called. Would you like to continue
          without filtering?")
    filt = input("(Y/N) ")
    if filt == "N" or filt == "n" or filt == "No" or filt == "no":
        print("To implement filtlong, please add --filtlong or -fl to your arguments")
        sys.exit()
    else:
        print("Continuing without filtlong")

# If KMA is not called, but the repN is, it won't just run without KMA


if args.repN is True and args.resistome is False:
    print("Called for replacing resistome genes, but resistome analysis isn't called. Continue without
          resistome analysis?")
    rep = input("(Y/N) ")
    if rep == "N" or rep == "n" or rep == "No" or rep == "no":
        print("To implement resistome analysis, please add --resistome or -ar to your arguments")
        sys.exit()
    else:
        print("Continuing without resistome analysis")



# Demux disclamer
```

```
if args.demux and (args.filtlong or args.host or args.resistome or args.taxonomy):
    print("You've selected demultilexing, only this and possibly QC will be ran, the output will be zipped.
          Please run the demultiplexed output files seperately")
    print("Would you like to continue?")
    dem = input("(Y/N) ")
    if dem == "N" or dem == "n" or dem == "No" or dem == "no":
        print("Please take the --demux argument out to run without demultiplexing")
        sys.exit()
    else:
        print("Continue with demultiplexing")
# Make sure the directory path exists, if not, create it

if not os.path.exists(args.outdir):
    os.mkdir(args.outdir)
# Setting the input as indata

indata = args.inp
# The prefix will be specified here, so it doesn't has to happen later

pre = os.path.basename(args.inp)
prefix = re.sub("\.gz|\.fastq|\.fq", "", pre)
if args.prefix is not None:
    prefix = args.prefix



# Creating a log of the pipeline

RNlog = os.path.join(args.outdir, "{}_ResistNanome-info.log".format(prefix))
f = open(RNlog, "a")
dt = datetime.datetime.now()
f.write(str(dt) + "\tPipeline started\n")
f.close()



# Creating a 'shortcut' to directories

directory = os.path.abspath(os.path.dirname(sys.argv[0]))
tool_dir = os.path.join(directory, "tools/")
lib_dir = os.path.join(directory, "database/")
script_dir = os.path.join(directory, "scripts/")



# Making sure the scripts are found

sys.path.append(script_dir)
sys.path.append(os.path.join(tool_dir, "PyFPDF", "fpdf", "fpdf.py"))
```

```python
from fpdf import FPDF


# Function for running kraken2 and bracken

def profiling(db, inp, part, id):
    print("Running Kraken2 for {}".format(part))
    os.system("{} --db {} --report {}_{}kreport.txt --threads {} --use-names --output {}_{}kraken.txt
              {}".format(os.path.join(tool_dir, "Kraken/kraken2"), db, os.path.join(args.outdir, "temp_" +
                         prefix), id, args.threads,os.path.join(args.outdir, "temp_" + prefix), id, inp))

def abundance(db, id, len, level):
    if isinstance(level, list):
        for lvl in level:
            os.system("{} -d {} -i {}_{}kreport.txt -o {}_{}bracken.txt -r {} -l {}".format(
                      os.path.join (tool_dir, "bracken"), db, os.path.join(args.outdir, "temp_" + prefix), lvl
                      + "_" + id, os.path.join(args.outdir, "temp_" + prefix), id, len, lvl))
    else:
        os.system("{} -d {} -i {}_{}kreport.txt -o {}_{}bracken.txt -r {} -l {}".format(
                  os.path.join(tool_dir, "bracken"), db, os.path.join(args.outdir, "temp_" + prefix), id,
                  os.path.join(args.outdir, "temp_" + prefix), id, len, level))


def med_round(data):
    len_read = []
    for seq in SeqIO.parse(data, "fastq"):
        len_read.append(int(len(seq)))
    med = round(median(len_read))
    smed = str(med)
    if len(smed) >= 4 and med > 2549:
        step = -3
    elif len(smed) == 3 or med <= 2549:
        step = -2
    rounding = round(med, step)
    return int(rounding)


# Running first QC, if asked for QC

if args.QC:
    f = open(RNlog, "a")
    dt = datetime.datetime.now()
    f.write(str(dt) + "\tStart NanoQC1\n")
    f.close()
    from QC import nanoqc
    nanoqc("QC1", "./tools/nanoQC/nanoQC.py", indata)
    print("Done, the html report can be found in the map 'QC' in the outdir")


# Running demultiplexing, if asked for

deminput = os.path.normpath(os.path.join(args.outdir, "{}_demulti/".format(prefix)))
if args.demux:
    f = open(RNlog, "a")
    dt = datetime.datetime.now()
    f.write(str(dt) + "\tStart Demultiplexing\n")
    f.close()
    from Demultiplexing import demux
    demux("./tools/porechop-runner.py", indata)
    indata = deminput



#Running Filtlong if asked for. Possibly with different min-length

if args.filtlong and not args.demux:
    f = open(RNlog, "a")
    dt = datetime.datetime.now()
    f.write(str(dt) + "\tStart filtlong\n")
    f.close()
    from Filtlong import filtlong
    filtlong(os.path.join(tool_dir, "filtlong"), indata, args.minlen)
    indata = os.path.join(args.outdir, "temp_{}_filtlong.fastq".format(prefix))



# Running second QC, if asked for QC and asked for demultiplexing or filtlong, for comparison

if args.QC and (args.demux or args.filtlong):
    from QC import nanoqc
    if os.path.isdir(indata):
        lst = sorted(os.listdir(indata))
        qclst = []
        for fl in lst:
            fl = os.path.join(indata, fl)
            qclst.append(fl)
        for qc in qclst:
            f = open(RNlog, "a")
            dt = datetime.datetime.now()
            f.write(str(dt) + "\tStart " + re.sub("\.fastq|\.gz", "", os.path.basename(qc)) + "\n")
            f.close()
            nanoqc("QC_" + re.sub("\.fastq|\.gz", "", os.path.basename(qc)), "./tools/nanoQC/nanoQC.py", qc)
        print("Done, the html reports can be found in the map 'QC' in the outdir")
    else:
        f = open(RNlog, "a")
        dt = datetime.datetime.now()
        f.write(str(dt) + "\tStart NanoQC2\n")
        f.close()
        nanoqc("QC2", "./tools/nanoQC/nanoQC.py", indata)
        print("Done, the html report can be found in the map 'QC' in the outdir")


# Running screening for/extracting of host DNA, if asked for

if args.host and not args.demux:
    f = open(RNlog, "a")
    dt = datetime.datetime.now()
    f.write(str(dt) + "\tStart host filtering\n")
    f.close()
    from Minifilter import unmapped
    unmapped(os.path.abspath(os.path.join(lib_dir, "/mash_db/")), indata)
    indata = os.path.join(args.outdir, "temp_{}_novert.fastq".format(prefix))


# Running resistome analysis and/or community profiling, in multithreading

def resistome():
    f = open(RNlog, "a")
    dt = datetime.datetime.now()
    f.write(str(dt) + "\tStart determination of resistome (res)\n")
    f.close()
    from KMA import resistome
    resistome(indata, os.path.abspath(os.path.join(lib_dir, "KMA_ResFinder")), args.phred)
    if args.repN:
        resist_indata = os.path.join(args.outdir, "temp_{}_resistome.fasta".format(prefix))
    else:
        resist_indata = os.path.join(args.outdir, "temp_{}_resistome.fastq".format(prefix))
    if not os.path.isfile(resist_indata):
        print("No any antibiotic resistance found!")


def taxonomy():
    f = open(RNlog, "a")
    dt = datetime.datetime.now()
    f.write(str(dt) + "\tStart taxonomy (tax)\n")
    f.close()
    profiling(os.path.abspath(os.path.join(lib_dir, "/Kraken2_Nanodb/")), indata, "taxonomy", "t")
    med = int(med_round(indata))
    abundance(os.path.abspath(os.path.join(lib_dir, "/Kraken2_Nanodb/")), "t", med, "G")
```

```
f = open(RNlog, "a")
dt = datetime.datetime.now()
f.write(str(dt) + "\t\t(tax) Start Bracken\n")
f.close()
com = []
kra = "{}_tkraken.txt".format(os.path.join(args.outdir, "temp_" + prefix))
with open(kra, "rt") as csvf:
    reader = csv.reader(csvf, delimiter="\t")
    for read in reader:
        if read[0] == "C":
            com.append(read[1] + ":" + read[2])


tax = []
taxduo = []
if type(args.lvl) is list:
    bra = "{}_S_tbracken.txt".format(os.path.join(args.outdir, "temp_" + prefix))
else:
    bra = "{}_tbracken.txt".format(os.path.join(args.outdir, "temp_" + prefix))
for b in com:
    tt = b.split(":")
    ID = tt[1].split(" ")
    lID = list(ID[-1])
    lID.pop()
    sID = "".join(lID)
    with open(bra, "rt") as csvfi:
        reader = csv.reader(csvfi, delimiter="\t")
        for col in reader:
            if col[1] == sID:
                taxduo.append("{}:{}".format(b, col[-1]))
for du in taxduo:
    if du not in tax:
        tax.append(du)


tax.sort(reverse = True)

print("Writing taxonomy output")
toutput = os.path.join(args.outdir, "{}_taxonomy_output.txt".format(prefix))
tt = open(toutput, "a")
tt.write("Read\tName (tax ID)\tPercentage of bacteria\n")
tt.close()
taxinfo = []
for inf in tax:
    o = inf.split(":")
```

```
        perc = float(o[0]) * 100
        peround = "{0:.6f}".format(perc)
        res = "{} - {}%\n".format(o[2], peround)
        if res not in taxinfo:
            taxinfo.append(res)
        tt = open(toutput, "a")
        tt.write("{}\t{}\t{}\n".format(o[1], o[2], peround))
        tt.close()


    tlog = os.path.join(args.outdir, "temp_{}_taxonomy_topout.txt".format(prefix))
    t = open(tlog, "a")
    t.write("Taxonomy\n")
    t.write("Name (tax ID) - Percentage out of bacteria\ntotal = {} bacteria\n\n".format(len(taxinfo)))
    for e, i in enumerate(taxinfo):
        if e <= 9:
            t.write(i)
    t.close()


    f = open(RNlog, "a")
    dt = datetime.datetime.now()
    f.write(str(dt) + "\tTaxonomy finished\n")
    f.close()


if args.threading and args.resistome and args.taxonomy and not args.demux:
    # Creating threads

    t1 = threading.Thread(target = resistome())
    t2 = threading.Thread(target = taxonomy())
    # Starting threads

    t1.start()
    t2.start()
    # Nothing else will start 'till both are done

    t1.join()
    t2.join()
elif args.resistome and args.taxonomy and not (args.demux and args. threading):
    resistome()
    taxonomy()
elif args.resistome and not args.demux and not args.taxonomy:
    resistome()
elif args.taxonomy and not args.demux and not args.resistome:
    taxonomy()
```

```
# Merging all output-files to one PDF

if not args.demux and (args.resistome or args.taxonomy):
    outlist = sorted(os.listdir(args.outdir))
    Pdf = []
    for i in outlist:
        if re.search("output", i):
            ii = os.path.join(args.outdir, i)
            Pdf.append(ii)


    pdf = FPDF()
    out = os.path.join(args.outdir, "{}_ResistNanome.pdf".format(prefix))
    if not os.path.exists(out):
        open(out, "x").close()
    textlist = []
    try:
        Pdf[1]
    except IndexError:
        with open(Pdf[0], "r") as f:
            pdf.add_page(orientation="P")
            for row in f.readlines():
                if row == f.readlines[0]:
                    pdf.set_font("Arial", "BU", size=10)
                    pdf.write(4, row)
                elif re.search("Read|Name", row):
                    pdf.set_font("Arial", "B", size=10)
                    pdf.write(4, row)
                else:
                    pdf.set_font("Arial", size=10)
                    pdf.write(4, row)
        pdf.output(out, "F")
    else:
        for path in Pdf:
            p = open(path, "r")
            pdf.add_page(orientation="P")
            lines = p.readlines()
            for row in lines:
                if row == lines[0]:
                    pdf.set_font("Arial", "BU", size=10)
                    pdf.write(4, row)
                elif re.search("Read|Name", row):
                    pdf.set_font("Arial", "B", size=10)
                    pdf.write(4, row)
                else:
                    pdf.set_font("Arial", size=10)
                    pdf.write(4, row)
            p.close()
        pdf.output(out, "F")


    if not args.keep:
        for path in Pdf:
            os.remove(path)
f = open(RNlog, "a")
dt = datetime.datetime.now()
f.write(str(dt) + "\tStart cleaning output\n")
f.close()
# The temporary output directory can be deleted, if keep argument is called, this argument will be false, so
all data will be kept

if not args.keep:
    outlist = sorted(os.listdir(args.outdir))
    rmlist = []
    for i in outlist:
        if re.search("temp_.+", i):
            ii = os.path.join(args.outdir, i)
            rmlist.append(ii)
    for r in rmlist:
        if os.path.isdir(r):
            rm = sorted(os.listdir(r))
            for f in rm:
                os.remove(os.path.join(r, f))
            os.rmdir(r)
        else:
            os.remove(r)


# Zipping fastq files to .gz

fq_files = []
if args.gz:
    print("Zipping files")
    outlist = sorted(os.listdir(args.outdir))
    for thing in outlist:
        if re.search(".+\.fastq$", str(thing)) or re.search(".+\.fq$", str(thing)):
            thing_path = os.path.join(args.outdir, thing)
            fq_files.append(thing_path)
# Zipping demux files to .gz, this will be done when demux is called
```

```
elif args.demux:
    outdir = sorted(os.listdir(deminput))
    for bc in outdir:
        if re.search(".+\.fastq", str(bc)) or re.search(".+\.fq", str(bc)):
            bc_path = os.path.join(deminput, bc)
            fq_files.append(bc_path)
# The actual zipping part

if args.gz or args.demux:
    for fq in fq_files:
        outp =  fq + ".gz"
        with open(fq, "rb") as full:
            with gzip.open(outp, "wb")  as output:
                shutil.copyfileobj(full, output)
        os.remove(fq)
f = open(RNlog, "a")
dt = datetime.datetime.now()
f.write(str(dt) + "\tFinished!\n")
f.close()
```

2. GNU license

GNU GENERAL PUBLIC LICENSE
Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <https://fsf.org/>
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license forsoftware and other kinds of works.

The licenses for most software and other practical works are designedto take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps:
(1) assert copyright on the software, and
(2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:
a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

3. Database for Minimap2 based on NCBI RefSeq database

Table 2. Information about the NCBI RefSeq sub-database

Species in standard database	English name	GCF ID
Bos indicus	Zebu cattle	GCF_000247795.1
Bos mutus	Wild yak	GCF_000298355.1
Bos taurus	Cattle	GCF_002263795.1
Canis lupus familiaris	Dog	GCF_000002285.3
Capra hircus	Goat	GCF_001704415.1
Chinchilla lanigera	Long-tailed chinchilla	GCF_000276665.1
Equus caballus	Horse	GCF_002863925.1
Felis catus	Domestic cat	GCF_000181335.3
Gorilla gorilla gorilla	Western lowland gorilla	GCF_000151905.2
Homo sapiens	Human	GCF_000001405.38
Mus musculus	House mouse	GCF_000001635.26
Ovis aries	Sheep	GCF_000298735.2
Ovis aries musimon	Mouflon	GCF_000765115.1
Pan troglodytes	Chimpanzee	GCF_002880755.1
Rattus norvegicus	Brown rat/Norway rat	GCF_000001895.5, GCF_000002265.2
Sus scrofa	Swine/pig	GCF_000003025.6
Danio rerio	Zebrafish	GCF_000002035.6
Gallus gallus	Red junglefowl/chicken	GCF_000002315.5
Meleagris gallopavo	Turkey	GCF_000146605.2

4. Porechop verbose output

demultiplex

```
 [1m [4mLoading reads [0m
/mnt/data/klif/ResistNanome/testdata/6hour_fastq/reads_seqB_6hr.fastq
1,784,000 reads loaded


 [1m [4mLooking for known adapter sets [0m


Barcodes determined to be in forward orientation
                                    Best            [0m
                                    read      Best    [0m
                                    start     read end [0m
  [4mSet                                  %ID        %ID       [0m
  [32mSQK-NSK007                          100.0       81.8 [0m
  Rapid                              66.7        0.0
  RBK004_upstream                            81.1        0.0
  [32mSQK-MAP006                           96.4       100.0 [0m
  [32mSQK-MAP006 short                       100.0       100.0 [0m
  [32mPCR adapters 1                      100.0       100.0 [0m
  [32mPCR adapters 2                      100.0       100.0 [0m
  [32mPCR adapters 3                      100.0       100.0 [0m
  1D^2 part 1                          80.0        75.0
  [32m1D^2 part 2                      97.0        90.3 [0m
  cDNA SSP                             86.8        84.6
  Barcode 1 (reverse)                    76.9        84.0
  Barcode 2 (reverse)                    76.9        80.0
  Barcode 3 (reverse)                    76.9        77.8
  Barcode 4 (reverse)                    75.0        76.9
  Barcode 5 (reverse)                    76.0        76.9
  Barcode 6 (reverse)                    80.0        80.0
  Barcode 7 (reverse)                    75.0        80.0
  Barcode 8 (reverse)                    75.0        79.2
  Barcode 9 (reverse)                    77.8        80.0
  Barcode 10 (reverse)                    80.0        80.0
  Barcode 11 (reverse)                    79.2        80.8
  Barcode 12 (reverse)                    76.9        79.2
  Barcode 1 (forward)                    76.0        76.0
  Barcode 2 (forward)                    79.2        80.0
  Barcode 3 (forward)                    76.0        76.9
  Barcode 4 (forward)                    77.8        76.9
  Barcode 5 (forward)                    77.8        77.8
  Barcode 6 (forward)                    77.8        79.2
Barcode 7 (forward)              79.3     80.0
Barcode 8 (forward)              75.0     76.9
Barcode 9 (forward)              76.9     79.2
Barcode 10 (forward)              76.9     76.9
Barcode 11 (forward)              79.2     79.2
Barcode 12 (forward)              77.8     80.8
 [32mBarcode 13 (forward)              100.0     100.0 [0m
 [32mBarcode 14 (forward)              100.0     100.0 [0m
 [32mBarcode 15 (forward)              100.0     100.0 [0m
 [32mBarcode 16 (forward)              100.0     100.0 [0m
 [32mBarcode 17 (forward)              100.0     100.0 [0m
 [32mBarcode 18 (forward)              100.0     100.0 [0m
 [32mBarcode 19 (forward)              100.0     100.0 [0m
 [32mBarcode 20 (forward)              100.0     100.0 [0m
Barcode 21 (forward)              76.0     76.9
Barcode 22 (forward)              80.8     80.0
Barcode 23 (forward)              76.9     76.0
Barcode 24 (forward)              79.2     76.9
Barcode 25 (forward)              76.9     75.0
Barcode 26 (forward)              76.9     76.0
Barcode 27 (forward)              76.0     77.8
Barcode 28 (forward)              80.0     76.0
Barcode 29 (forward)              77.8     76.9
Barcode 30 (forward)              80.0     76.9
Barcode 31 (forward)              80.0     80.0
Barcode 32 (forward)              79.2     76.9
Barcode 33 (forward)              79.2     76.0
Barcode 34 (forward)              76.9     76.9
Barcode 35 (forward)              75.0     76.9
Barcode 36 (forward)              76.0     76.9
Barcode 37 (forward)              76.9     79.2
Barcode 38 (forward)              76.9     79.2
Barcode 39 (forward)              76.0     76.9
Barcode 40 (forward)              76.0     76.0
Barcode 41 (forward)              76.0     76.9
Barcode 42 (forward)              76.9     80.0
Barcode 43 (forward)              76.0     80.0
Barcode 44 (forward)              74.1     76.0
Barcode 45 (forward)              77.8     76.9
Barcode 46 (forward)              79.2     79.2
Barcode 47 (forward)              79.2     76.9
Barcode 48 (forward)              76.9     76.9
Barcode 49 (forward)              80.0     76.9
Barcode 50 (forward)              76.0     79.2
```

Barcode 51 (forward)	76.9	79.2
Barcode 52 (forward)	76.0	76.9
Barcode 53 (forward)	75.0	76.9
Barcode 54 (forward)	76.0	80.0
Barcode 55 (forward)	76.9	76.9
Barcode 56 (forward)	76.9	79.2
Barcode 57 (forward)	76.0	76.0
Barcode 58 (forward)	80.0	77.8
Barcode 59 (forward)	76.0	76.9
Barcode 60 (forward)	80.0	79.2
Barcode 61 (forward)	75.0	76.9
Barcode 62 (forward)	76.0	76.0
Barcode 63 (forward)	76.9	80.8
Barcode 64 (forward)	75.0	76.9
Barcode 65 (forward)	75.0	76.9
Barcode 66 (forward)	76.9	80.0
Barcode 67 (forward)	80.0	80.0
Barcode 68 (forward)	77.8	76.0
Barcode 69 (forward)	75.9	77.8
Barcode 70 (forward)	76.0	80.0
Barcode 71 (forward)	77.8	80.0
Barcode 72 (forward)	76.9	84.0
Barcode 73 (forward)	74.1	80.8
Barcode 74 (forward)	76.0	76.0
Barcode 75 (forward)	76.0	77.8
Barcode 76 (forward)	76.0	79.2
Barcode 77 (forward)	76.9	76.9
Barcode 78 (forward)	76.9	80.0
Barcode 79 (forward)	76.0	79.2
Barcode 80 (forward)	76.9	76.0
Barcode 81 (forward)	79.2	76.0
Barcode 82 (forward)	75.0	79.2
Barcode 83 (forward)	79.2	79.2
Barcode 84 (forward)	76.9	76.9
Barcode 85 (forward)	80.0	76.0
Barcode 86 (forward)	75.0	76.0
Barcode 87 (forward)	76.0	77.8
Barcode 88 (forward)	76.9	77.8
Barcode 89 (forward)	79.2	79.2
Barcode 90 (forward)	76.0	76.0
Barcode 91 (forward)	76.0	77.8
Barcode 92 (forward)	76.9	76.9
Barcode 93 (forward)	80.0	79.2
Barcode 94 (forward)	75.0	79.2

```
Barcode 95 (forward)            80.0    76.9
Barcode 96 (forward)            79.2    76.9


 [1m [4mTrimming adapters from read ends [0m
             SQK-NSK007_Y_Top:  [31mAATGTACTTCGTTCAGTTACGTATTGCT [0m
          SQK-NSK007_Y_Bottom:  [31mGCAATACGTAACTGAACGAAGT [0m
        SQK-MAP006_Y_Top_SK63:  [31mGGTTGTTTCTGTTGGTGCTGATATTGCT [0m
     SQK-MAP006_Y_Bottom_SK64:  [31mGCAATATCAGCACCAACAGAAA [0m
  SQK-MAP006_Short_Y_Top_LI32:  [31mCGGCGTCTGCTTGGGTGTTTAACCT [0m
SQK-MAP006_Short_Y_Bottom_LI33:  [31mGGTTAAACACCCAAGCAGACGCCG [0m
                  PCR_1_start:  [31mACTTGCCTGTCGCTCTATCTTC [0m
                    PCR_1_end:  [31mGAAGATAGAGCGACAGGCAAGT [0m
                  PCR_2_start:  [31mTTTCTGTTGGTGCTGATATTGC [0m
                    PCR_2_end:  [31mGCAATATCAGCACCAACAGAAA [0m
                  PCR_3_start:  [31mTACTTGCCTGTCGCTCTATCTTC [0m
                    PCR_3_end:  [31mGAAGATAGAGCGACAGGCAAGTA [0m
             1D2_part_2_start:  [31mCTTCGTTCAGTTACGTATTGCTGGCGTCTGCTT [0m
               1D2_part_2_end:  [31mCACCCAAGCAGACGCCAGCAATACGTAACT [0m
                         BC13:  [31mAGAACGACTTCCATACTCGTGTGA [0m
                     BC13_rev:  [31mTCACACGAGTATGGAAGTCGTTCT [0m
                         BC14:  [31mAACGAGTCTCTTGGGACCCATAGA [0m
                     BC14_rev:  [31mTCTATGGGTCCCAAGAGACTCGTT [0m
                         BC15:  [31mAGGTCTACCTCGCTAACACCACTG [0m
                     BC15_rev:  [31mCAGTGGTGTTAGCGAGGTAGACCT [0m
                         BC16:  [31mCGTCAACTGACAGTGGTTCGTACT [0m
                     BC16_rev:  [31mAGTACGAACCACTGTCAGTTGACG [0m
                         BC17:  [31mACCCTCCAGGAAAGTACCTCTGAT [0m
                     BC17_rev:  [31mATCAGAGGTACTTTCCTGGAGGGT [0m
                         BC18:  [31mCCAAACCCAACAACCTAGATAGGC [0m
                     BC18_rev:  [31mGCCTATCTAGGTTGTTGGGTTTGG [0m
                         BC19:  [31mGTTCCTCGTGCAGTGTCAAGAGAT [0m
                     BC19_rev:  [31mATCTCTTGACACTGCACGAGGAAC [0m
                         BC20:  [31mTTGCGTCCTGTTACGAGAACTCAT [0m
                  BC20_rev:  [31mATGAGTTCTCGTAACAGGACGCAA [0m



1,775,766 / 1,784,000 reads had adapters trimmed from their start (177,542,763 bp removed)
1,674,045 / 1,784,000 reads had adapters trimmed from their end (126,814,581 bp removed)


 [1m [4mDiscarding reads containing middle adapters [0m


106,066 / 1,784,000 reads were discarded based on middle adapters


 [1m [4mSaving trimmed reads to barcode-specific files [0m

   [4mBarcode  Reads    Bases        File                                  [0m
  BC13     167,175  194,292,230  /mnt/docker/ResistNanome/test/resultRun3/reads_seqB_6hr_demu
                    lti/BC13.fastq
  BC14     196,925  184,080,582  /mnt/docker/ResistNanome/test/resultRun3/reads_seqB_6hr_demu
                    lti/BC14.fastq
  BC15     266,267  155,053,620  /mnt/docker/ResistNanome/test/resultRun3/reads_seqB_6hr_demu
                    lti/BC15.fastq
  BC16     228,493  217,889,538  /mnt/docker/ResistNanome/test/resultRun3/reads_seqB_6hr_demu
                    lti/BC16.fastq
  BC17     119,793  110,442,405  /mnt/docker/ResistNanome/test/resultRun3/reads_seqB_6hr_demu
                    lti/BC17.fastq
  BC18     124,180  115,536,767  /mnt/docker/ResistNanome/test/resultRun3/reads_seqB_6hr_demu
                    lti/BC18.fastq
  BC19     233,175  179,270,183  /mnt/docker/ResistNanome/test/resultRun3/reads_seqB_6hr_demu
                    lti/BC19.fastq
  BC20     254,697  333,047,244  /mnt/docker/ResistNanome/test/resultRun3/reads_seqB_6hr_demu
                    lti/BC20.fastq
  none     83,271   79,244,471  /mnt/docker/ResistNanome/test/resultRun3/reads_seqB_6hr_demu
                    lti/none.fastq
```

5. ResistNanome output of a run by A.B. with barcode 19

```
Resistome
Resistancy gene  Name (tax ID) - Percentage out of bacteria
total = 41 bacteria/resistance combinations

tet(L)_4  Staphylococcus aureus (taxid 1280) - 68.074000%
tet(L)_2  Staphylococcus aureus (taxid 1280) - 68.074000%
aadD_2  Staphylococcus aureus (taxid 1280) - 68.074000%
tet(L)_3  Staphylococcus aureus (taxid 1280) - 68.074000%
blaZ_16  Staphylococcus aureus (taxid 1280) - 68.074000%
blaZ_78  Staphylococcus aureus (taxid 1280) - 68.074000%
aadD_1  Staphylococcus aureus (taxid 1280) - 68.074000%
tet(L)_8  Staphylococcus aureus (taxid 1280) - 68.074000%
tet(L)_2  Enterococcus faecium (taxid 1352) - 7.388000%
tet(L)_3  Enterococcus faecium (taxid 1352) - 7.388000%
```

Taxonomy
Name (tax ID) - Percentage out of bacteria
total = 37 bacteria

Lactobacillus (taxid 1578) - 60.223000%
Bacillus (taxid 1386) - 8.019000%
Salmonella (taxid 590) - 6.634000%
Enterococcus (taxid 1350) - 6.539000%
Staphylococcus (taxid 1279) - 5.967000%
Listeria (taxid 1637) - 5.574000%
Escherichia (taxid 561) - 3.365000%
Pseudomonas (taxid 286) - 2.740000%
Shigella (taxid 620) - 0.224000%
Citrobacter (taxid 544) - 0.053000%