Andrés Camilo Zúñiga González - Assessment

1 Variant Calling

For this analysis, I would mainly rely on `DADA2` for quality filtering and `bowtie2` for mapping.

`DADA2` can be run from an R script or using R CLI, following these commands:

1. Fastq files are loaded in `R`:

- `fnFs <- list.files("sample1.R1.fastq.gz", full.names = TRUE)`
- `fnRs <- list.files("sample1.R1.fastq.gz", full.names = TRUE)`

2. Upon checking quality profile of each file, a truncation point (integer) can be decided for both forward and reverse reads:

- `truncF <- integer`
- `truncR <- integer2`

3. Afterwards, they can be trimmed and filtered using the following functions:

- `filtFs <- file.path(path, "filtered", paste0(sample.names, "_F_filt.fastq.gz"))`
- `filtRs <- file.path(path, "filtered", paste0(sample.names, "_R_filt.fastq.gz"))`
- `out <- filterAndTrim(fnFs, filtFs, fnRs, filtRs, truncLen = c(truncF, truncR), maxN = 0, maxEE = c(2,2), truncQ = 2, rm.phix = TRUE, compress = TRUE, multithread = TRUE)`

As a first measure after the quality filtering of the fastqc files from the sequencer, using bowtie, we are going to index the `reference.fas` file, which creates some `*.bt2` files for further use.

- `bowtie2-build reference.fas bowtie2/reference`

After building the index for the reference genome, now it is possible to map the paired-end sequences using the following command, which outputs a `.sam` file:

- `bowtie2 -x bowtie2/NC_012967.1 -1 sample1.R1_F_filt.fastq.gz -2 sample1.R2_R_filt.fastq.gz -S bowtie2/mapping.sam`

This processes can be run in parallel using the `-p` flag and setting the number of processors for the job.

Furthermore, for calling single nucleotide variants `samtools` is used, taking advantage of the output format of the mapping step. First, the reference file must indexed (different process than in `bowtie2`), which creates a `.fai` file

- `samtools faidx reference.fas`

Then, the output from the mapping must be converted to `.bam`. With the `.bam` file it is necessary to sort it and index it using the following functions:

- `samtools view -b -S -o mapping.bam mapping.sam`
- `samtools sort mapping.bam -o mapping.sorted.bam`
- `samtools index mapping.sorted.bam`

Afterwards, using the reference and the sorted mapping, the calling variants step is performed using the mpileup command, which creates a `.bcf` file

- `samtools mpileup -u -f reference.fas mapping.sorted.bam > mapping.bcf`

For better inspection of the result, this output can be transformed using `bcftools`:

- `bcftools call -v -c mapping.bcf > mapping.vcf`

Finally, if I wanted variants greater than a threshold (integer), for instance, greater than 20, I would run:

- `cat mapping.vcf | grep AF1>20 > mapping.filtered.vcf`

2 Data Parsing

Python3.8 was used for data parsing, alongside the pandas and regex libraries.

1. Both .csv files were loaded using pandas
2. A subset of the metadata file was created filtering where subclass is either CARBAPENEM or CEPHALOSPORIN and the Accession ID starts with NG_
3. A list of Accessions from the previous dataFrame was created for further use
4. A list of the loci in the ariba dataFrame was created
5. A function was defined in order to perform a double loop on the loci and the rows of the ariba dataFrame checking the desired conditions. If a row met the requirements it was added to a new dataframe.
6. The new dataFrame containing informatino about the sample, the antibiotic to which it is resistant and the Accession Number is saved as a csv file.

3 Nextflow Workflow

For this analysis, a new process was added to the `main.nf` file which used the `sistr` library on every `.fasta` file, just like the previous process did with `seqsero2`. The presentation is a available in this Google Slides link.