# 1 Antimicrobial resistance identification using ARIBA

## 1.1 Introduction

Antimicrobial resistance (AMR) is one of the major threats to human and animal health worldwide. An important component of any strategy to tackle antimicrobial resistance (AMR) is having rapid and accurate methods for identifying markers of resistance. ARIBA is one such method or tool that can be used to detect and identify antibiotic resistance genes from whole genome sequencing data.

ARIBA can be used to identify the prescence/abscence of a set of genes and their variants from sequence data of a sample. In particular, it was designed to detect and identify antibiotic resistance genes. This tutorial will walk you through the analysis of the *Neisseria gonorrhoeae* data set used in the ARIBA paper:

> **ARIBA: rapid antimicrobial resistance genotyping directly from sequencing reads**
> Hunt M, Mather AE, Sánchez-Busó L, Page AJ, Parkhill J, Keane JA, Harris SR.
> *Microbial Genomics 2017. doi: 110.1099/mgen.0.000131*
> PMID: 29177089

A copy of the paper can be found at

```
~/course_data/amr/ariba_paper.pdf
```

## 1.2 Learning outcomes

By the end of this tutorial you can expect to be able to:

- Use ARIBA to detect the prescence/abscence of a set of genes
- Download and prepare the standard AMR databases for use with ARIBA
- Run ARIBA on several samples to identify antibiotic resistance
- Prepare and use your own database for use with ARIBA
- Summarise ARIBA results for several samples
- Query the AMR results produced by ARIBA
- Use Phandango to visualise ARIBA results

## 1.3 Tutorial sections

This tutorial comprises the following sections:

1. Detect prescence/abscence of a set of genes with ARIBA
2. Use a standard AMR database with ARIBA
3. Prepare a custom reference database for ARIBA
4. Run ARIBA using a custom reference database
5. View summarized results using Phandango
6. Investigate MIC data in relation to variants in the samples

## 1.4 Authors and License

This tutorial was created by Jacqui Keane and Martin Hunt.

### 1.4.1 Running commands in this tutorial

You can follow this tutorial by typing all the commands you see in a terminal window on your computer. This is similar to the “Command Prompt” window on MS Windows systems, which allows the user to type DOS commands to manage files.

To get started, open a terminal window and type the command below followed by the `Enter` key:

```
cd ~/course_data/amr/data
```

Now you can follow the instructions in the tutorial from here.

## 1.5 Prerequisites

This tutorial assumes that you have the `ARIBA` software application installed on your computer. It has already been installed on the computer you are using for this training course. To activate the ARIBA conda environment and check that the software is installed correctly, run the following command:

```
conda activate ariba-2.14.6
ariba --help
```

This should return the following help message:

```
usage: ariba <command> <options>

ARIBA: Antibiotic Resistance Identification By Assembly

optional arguments:
  -h, --help      show this help message and exit

Available commands:

    aln2meta      Converts multi-aln fasta and SNPs to metadata
    expandflag    Expands flag column of report file
    flag          Translate the meaning of a flag
    getref        Download reference data
    micplot       Make violin/dot plots using MIC data
    prepareref    Prepare reference data for input to "run"
    pubmlstget    Download species from PubMLST and make db
    pubmlstspecies
                  Get list of available species from PubMLST
    refquery      Get cluster or sequence info from prepareref output
    run           Run the local assembly pipeline
    summary       Summarise multiple reports made by "run"
    test          Run small built-in test dataset
    version       Get versions and exit
```

To get started with the tutorial, head to the first section: Detect prescence/abscence of genes with ARIBA

## 2 Detect prescence/abscence of genes with ARIBA

One of the most basic tasks ARIBA can be used for is to detect the presence or abscence of a set genes. Let us look at an example. But first, let's check that you're in the right place. Type the command below in the terminal window followed by the Enter key:

```
pwd
```

It should display something like:

`/home/manager/course_data/amr/data`

Ok, back to the example! We have sequence data for 3 **Neisseria gonorrhoeae** samples and want to determine if the genes `fitA` and `fbpA` are present or absent in these samples. Let us use ARIBA to determine this from the raw sequence data.

First list the fastq files for the samples.

```
ls fastq
```

For the purpose of this exercise, we have created a fasta file containing the sequences of the genes `fitA` and `fbpA`, take a look:

```
cd basic
ls
cat genes.fasta
```

Running ARIBA is generally a three stage process:

- prepare a reference database
- run ariba on each sample
- summmarise the ariba results for all samples

First prepare the reference database:

```
ariba prepareref -f genes.fasta --all_coding yes out
```

Next, run ariba for all 3 samples

```
ariba run out ../fastq/ERR1067813_1.fastq.gz ../fastq/ERR1067813_2.fastq.gz \
ariba_results_ERR1067813
```

```
ariba run out ../fastq/ERR1067814_1.fastq.gz ../fastq/ERR1067814_2.fastq.gz \
ariba_results_ERR1067814
```

```
ariba run out ../fastq/ERR1067815_1.fastq.gz ../fastq/ERR1067815_2.fastq.gz \
ariba_results_ERR1067815
```

Next, summarise the results (the `--no_tree` option tells ARIBA not to generate a tree):

```
ariba summary --no_tree all_samples ariba_results*/report.tsv
```

Now inspect the results:

```
cat all_samples.csv
```

Notice that the original gene names are not present in the summary file, instead we have the cluster names cluster, cluster_1 etc. This is because ARIBA gathers together genes that are similar into clusters.

To determine which genes are in which cluster run:

```
ariba refquery out cluster cluster
ariba refquery out cluster cluster_1
```

## 2.1 Exercises

1. How many samples have the fitA gene?
2. How many samples have the fbpA gene?

Now go to the next part of the tutorial where we use a standard database with ARIBA to determine AMR.

# 3 Use a standard AMR database with ARIBA

This sections shows you how to run ARIBA on several samples to determine the presence of AMR genes using standard public AMR databases.

Make a new directory `standard` and move into that directory:

```
cd ..
mkdir standard
cd standard
```

## 3.1 Reference database

We are going to download and prepare a reference database for use with ARIBA. You can use any one of the standard reference databases that ARIBA supports

- ARG-ANNOT. PMID: 24145532
- CARD. PMID: 23650175
- MEGARes PMID: 27899569
- NCBI BioProject: PRJNA313047
- plasmidfinder PMID: 24777092
- resfinder. PMID: 22782487
- VFDB. PMID: 26578559
- SRST2's version of ARG-ANNOT. PMID: 25422674.
- VirulenceFinder PMID: 24574290.

Let's use the CARD database. The CARD (Comprehensive Antibiotic Resistance Database) is a biological database that collects and organizes reference information on antimicrobial resistance genes, proteins and phenotypes. Type the following commands to make an ARIBA database directory called `ariba_card_db`:

```
ariba getref card out.card
ariba prepareref -f out.card.fa -m out.card.tsv ariba_card_db
```

## 3.2 Run ARIBA on one sample

To run ARIBA on one of the *Neisseria gonorrhoeae* samples using the CARD database,type:

```
ariba run ariba_card_db ../fastq/ERR1067813_1.fastq.gz \
../fastq/ERR1067813_2.fastq.gz ERR1067813
```

This may take a few minutes to run and any WARNING messages can be ignored. In the above command we have supplied the database directory we prepared earlier (which we have called `ariba_card_db`) and two sequencing reads files `fastq/ERR1067813_1.fastq.gz`, `fastq/ERR1067813_2.fastq.gz` along with the name of the output directory `ERR1067813`.

When complete, check what files and directories were created by ARIBA:

```
ls
```

You should see a new directory called ERR1067813 that will contain the ARIBA results.

## 3.3 Run ARIBA on all samples

But what if we want to run ARIBA on all 3 samples? This can be done with a *for loop*. We assume that the reads files are named like this:

```
ERR1067813_1.fastq.gz ERR1067813_2.fastq.gz
ERR1067814_1.fastq.gz ERR1067814_2.fastq.gz
ERR1067815_1.fastq.gz ERR1067815_2.fastq.gz
```

To run ARIBA on all samples type:

```
for s in `ls ../fastq/*1.f*.gz | sed 's/..\/fastq\///' | sed 's/\_1.fastq.gz//'`
do
    ariba run ariba_card_db ../fastq/${s}_1.fastq.gz ../fastq/${s}_2.fastq.gz \
    ariba.${s}
done
```

The above command may take a few mintes to run. While you are waiting for it to finish read the explanation of the for loop below.

The loop iterates through each item returned by the command `ls ../fastq/*1.f*.gz | sed 's/..\/fastq\///' | sed 's/\_1.fastq.gz//'` and assigns the value of the item to the variable `$s`.

The body of the loop runs the command `ariba run`. The arguments passed to `ariba run` are the fastq files for the sample specified as `../fastq/${s}_1.fastq.gz` and `../fastq/${s}_2.fastq.gz` and the output directory for the sample specified as `ariba.$s`. For example, for sample ERR1067813, $s is ERR1067813 and the fastq files are ../fastq/ERR1067813_1.fastq.gz and ../fastq/ERR1067813_2.fastq.gz and ariba.ERR1067813 is the output directory.

Let's also look more closely at the command `ls ../fastq/*1.f*.gz | sed 's/..\/fastq\///' | sed 's/\_1.fastq.gz//'`.

First try:

```
ls ../fastq/*1.f*.gz
```

This lists all the files in the `../fastq` directory that end with '1.fastq.gz' which should give us one fastq file for each sample.

However all the files are prefixed with the path or location of the file (`../fastq/`). To remove this we will use the `sed` command to replace all occurences of `../fastq/` with an empty string ''. Note the additional backslashes (\) used to escape the forward slashes (/) in the path.

Try it:

```
ls ../fastq/*1.f*.gz | sed 's/..\/fastq\///'
```

The above command now lists just the fastq files which include the _1.fastq.gz at the end. Our aim is to extract a sample name or unique identifier for each of our samples (so everything before the

_1.fastq.gz). To acieve this we use an additional `sed` to replace the occurence of `_1.fastq.gz` with an empty string “.

Try it:

```
ls ../fastq/*1.fastq.gz | sed 's/..\/fastq\///' | sed 's/\_1.fastq.gz//'
```

**Note**: If you run ariba on your own data then you may need to edit the above command depending on how your own files are named.

## 3.4 ARIBA output

While you are waiting for ARIBA to run on all 3 samples, go to the ARIBA wiki (https://github.com/sanger-pathogens/ariba/wiki/Task:-run and read about the ARIBA output and what each of the columns in the report.tsv output file mean.

## 3.5 Summarising the results

Now gather together the results

```
ariba summary all_results ariba.*/report.tsv
```

Look at the files produced by ariba summary:

```
ls
```

You should see 3 files with the prefix `all_results`:

`all_results.csv all_results.phandango.csv all_results.phandango.tre`

Now look at the file `all_results.csv` and answer the questions below.

## 3.6 Exercises

1. Which AMR genes are present in all 3 samples?
2. Which AMR genes are absent in sample ERR1067813 but present in the other two samples?
3. Which AMR genes are absent in sample ERR1067814 but present in the other two samples?
4. Which AMR genes are absent in sample ERR1067815 but present in the other two samples?

Now go to the next part of the tutorial where we prepare a custom reference database for ARIBA.

# 4 Prepare a custom reference database for ARIBA

For the *N. gonorrhoeae* data from the ARIBA manuscript, we were also interested in a looking for a specific set of sequences and SNPs not found in any of the standard databases. This means using custom reference data as opposed to one of the public reference sets like in the previous section.

We were interested in several reference sequences (coding and non-coding sequences) and particular variants in those sequences. The idea is to generate input files to ARIBA that contain all the sequences and variants of interest, using the ARIBA function aln2meta.

First, let's change to the directory with the reference data.

```
cd ../Ref/
```

## 4.1 Using one reference sequence

We start by describing *folP*, but the method is (nearly) the same for all other sequences.

There are several alleles of *folP* and we want to include them all in the ARIBA database. We have a SNP of interest R228S in the sequence in the allele called "folP.WHO_F_01537c", which confers resistance to sulphonamides. When we run ARIBA, a particular sample may have a different allele from folP.WHO_F_01537c, but we would still like to know whether it has the SNP R228S. However, the position may not be 228 because of insertion or deletions. So we use a multiple alignment of all the reference alleles, and just supply the SNP to ARIBA in one of the alleles, in this case folP.WHO_F_01537c.

The "aln2meta" function of ARIBA needs two input files: a multiple alignment file of the alleles, and a tab-delimited file of the SNPs of interest. In this case, the SNPs file simply contains one line:

```
cat aln2meta_input/folP_in.tsv
```

There are four columns:

1. Sequence name
2. SNP (an amino acid change at position 228, where R is the wild type and S is the variant)
3. A "group name" for this SNP. This is optional and a dot "." means no group name. Putting SNPs into the same group allows ARIBA to report them together later on.
4. A description of the SNP. This will appear in ARIBA's output files to save looking up the reason the SNP is of interest.

This file is used together with the mulitple alignment file to generate input files to ARIBA when making the database. This is the command to run:

```
ariba aln2meta --variant_only aln2meta_input/folP.aln \
aln2meta_input/folP_in.tsv coding aln2meta_output/folP
```

A few things to note about the above command:

1. The option `--variant_only` was used, which affects how ARIBA reports later on when summarizing across all samples. We are only interested in this gene being present if it has a variant that causes resistance.

2. `aln2meta_input/folP.aln` is the name of the multiple alignment file.
3. The sequence is "coding", which makes ARIBA treat it as such, in particular it will interpret the variant R228S as an amino acid change at position 228 in the translated amino acid sequence. The input sequence is still in nucleotides, not amino acids.
4. The command outputs three files, which can be used as input to the command `ariba prepareref` (see later).

```
ls aln2meta_output/folP*
```

Although we have many more reference sequences of interest to deal with for the complete analysis, for illustrative purposes here we can use the three files `aln2meta_output/folP*` to make an ARIBA reference database:

```
ariba prepareref -f aln2meta_output/folP.fa \
    -m aln2meta_output/folP.tsv --cdhit_clusters \
    aln2meta_output/folP.cluster test.aribadb
```

This made an ARIBA database of just those sequences and the SNP R228S in a new directory called `test.aribadb`. Let's check that it was made:

```
ls test.aribadb
```

You do not need to worry about the contents of the new directory `test.aribadb`, just know that it can be used as input to run ARIBA. However, this was for just one of the many reference sequences of interest, so we will delete it.

```
rm -r test.aribadb
```

## 4.2 Using all reference sequences

We need to deal with each of the reference sequences in turn by running `ariba aln2meta` on each, like in the above example with *folP*. The only difference is that some of them are non-coding sequences, which means that the command must have `noncoding` instead of `coding`. For example:

```
ariba aln2meta --variant_only aln2meta_input/16S.aln \
    aln2meta_input/16S_in.tsv noncoding aln2meta_output/16S
```

There are 10 coding sequences and two non-coding sequences. Instead of writing 12 commands, we will use two 'for loops'. First, the coding sequences:

```
for x in folP gyrA mtrR parC parE penA ponA porB1b rpoB rpsJ
do
    ariba aln2meta --variant_only aln2meta_input/$x.aln \
    aln2meta_input/$x\_in.tsv coding aln2meta_output/$x
done
```

And now the two non-coding sequences:

```
for x in 16S 23S
do
    ariba aln2meta --variant_only aln2meta_input/$x.aln \
    aln2meta_input/$x\_in.tsv noncoding aln2meta_output/$x
done
```

This has generated three files for each sequence. We will combine these to make input files for running `ariba prepareref`.

```
cat aln2meta_output/*.fa presence_absence/*.fa > Ngo_ARIBA.fa
```

```
cat aln2meta_output/*.tsv presence_absence/presence_absence.tsv \
    > Ngo_ARIBA.tsv
```

```
cat aln2meta_output/*.cluster \
    presence_absence/presence_absence.clusters \
    > Ngo_ARIBA.clusters
```

Finally, we have the three input files needed to make a single ARIBA database that has information on all the sequences and SNPs of interest. In case the directory is already there, we delete it first, then generate the database:

```
rm -rf Ngo_ARIBAdb
```

```
ariba prepareref -f Ngo_ARIBA.fa -m Ngo_ARIBA.tsv \
    --cdhit_clusters Ngo_ARIBA.clusters Ngo_ARIBAdb
```

```
ls
```

```
cd ..
```

We now have a directory `Ngo_ARIBAdb` that can be used as the reference database when running `ariba run` on each sample.

Now move on to the next part of the tutorial where we run ARIBA using the custom reference data.

# 5 Run ARIBA using a custom reference database

This section illustrates how to run ARIBA on a large number of samples using a custom reference database. To save time, you will not actually run any of the commands in this section (each run of ARIBA takes a few minutes).

## 5.1 Reference database

First you need a reference database. You will already have one if you followed the instructions in the previous part of this tutorial (Prepare a custom reference data for ARIBA).

## 5.2 How to run on one sample

ARIBA needs the database directory, which we have called `Ngo_ARIBAdb` in the previous section of the tutorial, and two sequencing reads files `reads.1.fastq.gz`, `reads.2.fastq.gz`. The command to run ARIBA would look like (do not run this here!):

```
ariba run Ngo_ARIBAdb reads.1.fastq.gz reads.2.fastq.gz outdir
```

The above command will make a new directory called `outdir` that contains the results of matching the data for your sample (reads.1.fastq.gz and reads.2.fastq.gz) against the database Ngo_ARIBAdb.

## 5.3 Run on all samples

The *N. gonorrhoeae* dataset consists of 1517 samples, and we need to run ARIBA on each sample, which can be done with a "for" loop. Assuming that the reads files are named like this:

```
ERR1067813.1.fq.gz ERR1067813.2.fq.gz
ERR1067814.1.fq.gz ERR1067814.2.fq.gz
ERR1067815.1.fq.gz ERR1067815.2.fq.gz
```

Then we could run ARIBA on all samples like this (do not run this command here!):

```
for sample in `ls *.1.fq.gz | sed 's/\.1.fq.gz//'`
do
    ariba run Ngo_ARIBAdb $sample.1.fq.gz $sample.2.fq.gz $sample.ariba
done
```

The output directory of each sample is called `$sample.ariba`, for example ERR1067813.ariba is the output directory for sample ERR1067813.

Note if you are analysing your own data you may need to edit the command depending on how your own fastq files are named.

## 5.4 ARIBA output

There is no need to run the commands in this section, to save time they have already been run and the results are found in the directory `ARIBA_reports`. Take a look:

```
ls ARIBA_reports
```

Now go to the next part of the tutorial where we use Phandango to view the results.

# 6 Viewing ARIBA results in Phandango

This section describes how to use Phandango to view a summary of ARIBA results from many samples.

The most important output file from ARIBA is the report called `report.tsv`. For this tutorial, we have all 1517 reports in the directory `ARIBA_reports/`.

```
ls ARIBA_reports | wc -l
```

As we have seen before, ARIBA has a functon called "summary" that can summarise presence/absence of sequences and/or SNPs across samples. It takes at least two ariba reports as input, and makes a CSV file that can be opened in your favourite spreadsheet program, and also makes input files for Phandango. The two Phandango files (a tree and a CSV file) can be dropped straight into the Phandango page for viewing.

The tree that ARIBA makes is based on the CSV file, which contains results of presence/absence of sequence and SNPs, and other information such as percent identity bewteen contigs and reference sequences. This means that it does not necessarily represent the real phylogeny of the samples. It is more accurate to provide a tree built from the sequencing data. For this reason, we will use a pre-computed tree file `tree_for_phandango.tre`.

## 6.1 Basic usage of ariba summary

First, let's run `ariba summary` using the default settings, except we will skip making the tree (option `--no_tree`):

```
ariba summary --no_tree out ARIBA_reports/*.tsv
```

We can see that this made two files:

```
ls out.*
```

They are the same except for the first line, which has Phandango-specific information. ARIBA uses the filenames as sample names in the output:

```
head -n 2 out.phandango.csv
```

The first name is "ARIBA_reports/ERR1067709.tsv", and the rest are named similarly. This is not ideal, as it will look ugly in Phandango. Further, the names must exactly match the names in the tree file for Phandango to work (have a look in the tree file `tree_for_phandango.tre`). You could do a little hacking here using the Linux command `sed` on the CSV file. Or alternatively, we can supply ARIBA with a file of filenames that also tells ARIBA what to call the samples in the output CSV files it produces. Instead of "ARIBA_reports/ERR1067709.tsv", we would like to simply use "ERR1067709", which is cleaner and matches the tree file. It also means we can (and will) repeatedly run `ariba summary` with different options, and get output files that can be loaded straight into Phandango. This is one way to make the file with the naming information:

```
ls ARIBA_reports/* | awk -F/ '{print $0,$NF}' | \
    sed 's/.tsv$//' > filenames.fofn
```

The file is quite simple. Column 1 is the filename, and column 2 is the name we would like to use in the output.

```
head filenames.fofn
```

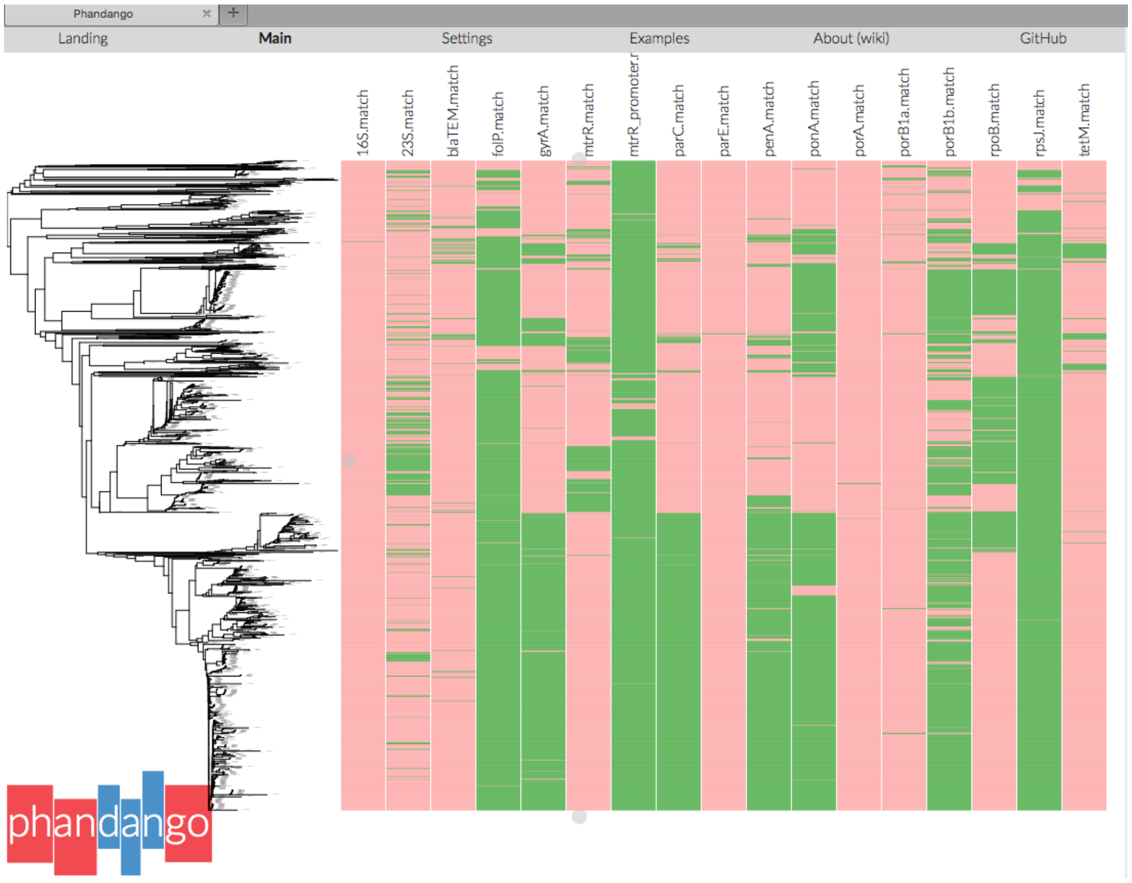
Now we can rerun summary using this input file. Note the use of the new option `--fofn`.

```
ariba summary --no_tree --fofn filenames.fofn \
    out ARIBA_reports/*.tsv
```

Check that the renaming worked:

```
head -n 2 out.phandango.csv
```

Now go to Phandango http://phandango.net and drag and drop the files `out.phandango.csv` and `tree_for_phandango.tre` into the window. The result should like this:



This a very high-level summary of the data. For each cluster, it is simply saying whether or not each sample has a 'match'. Green means a match, and pink means not a match. For presence/absence genes, this means that the gene must simply be there to count as a match. If it is a "variant only" gene, then the gene must be there along with one of the variants that we told ARIBA about earlier when generating the custom ARIBA database.

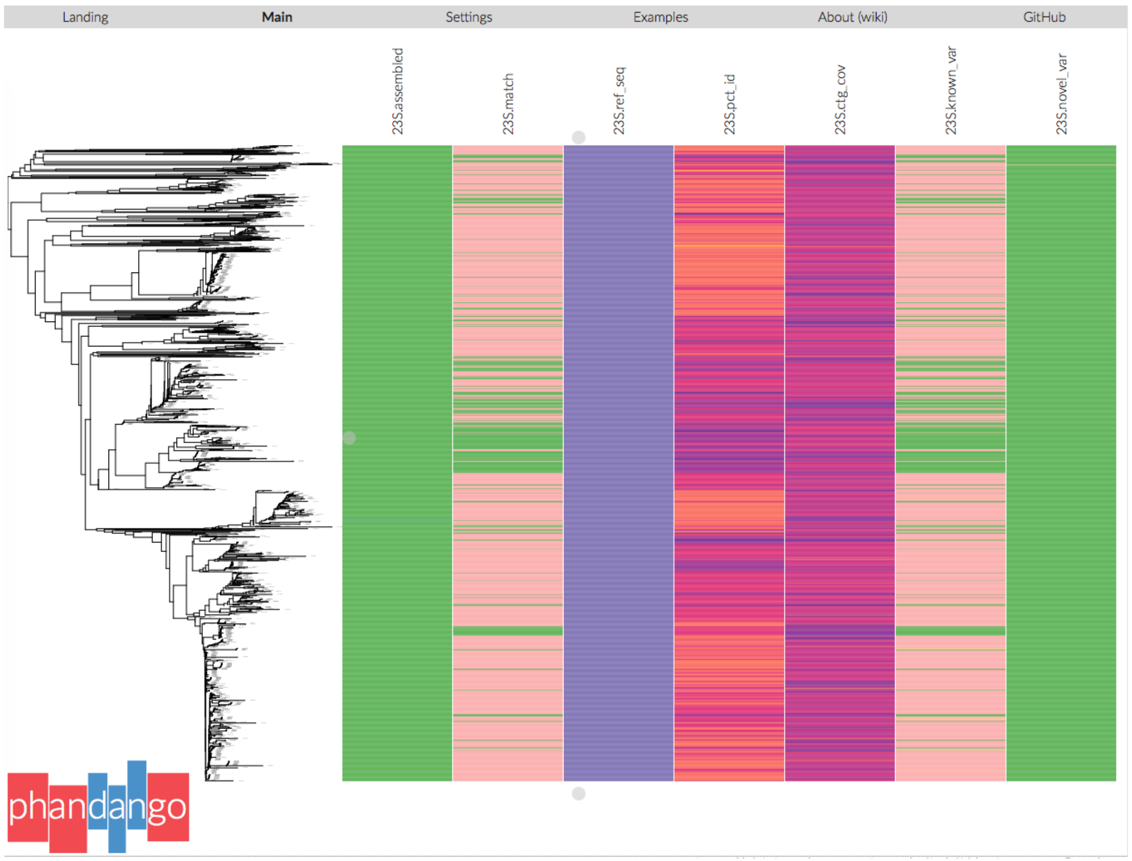
## 6.2 More information per cluster

In addition to a simple "yes" or "no" as to whether a sample "matches" a given cluster (as explained above), additional columns can be output for each cluster. See the ARIBA summary wiki page for a full description of the options.

Adding more columns can result in a very wide plot, so we will just look at 23S for now, using the option `--only_clusters 23S`. Adding the option `--preset all` will show all available columns for the 23S cluster:

```
ariba summary --only_clusters 23S --preset cluster_all --no_tree \
  --fofn filenames.fofn out ARIBA_reports/*.tsv
```

As before, drag and drop the files `out.phandango.csv` and `tree_for_phandango.tre` into the window. This time the result should look like this:



Now there are seven columns, showing various findings from ARIBA for 23S. These columns are described in the ARIBA summary wiki page.

You can control which of the seven cluster columns are output using the option --cluster_cols instead of --preset. For example, this will show just the “match” and “pct_id” columns:

```
ariba summary --only_clusters 23S --cluster_cols match,pct_id \
    --no_tree --fofn filenames.fofn out \
    ARIBA_reports/*.tsv
```

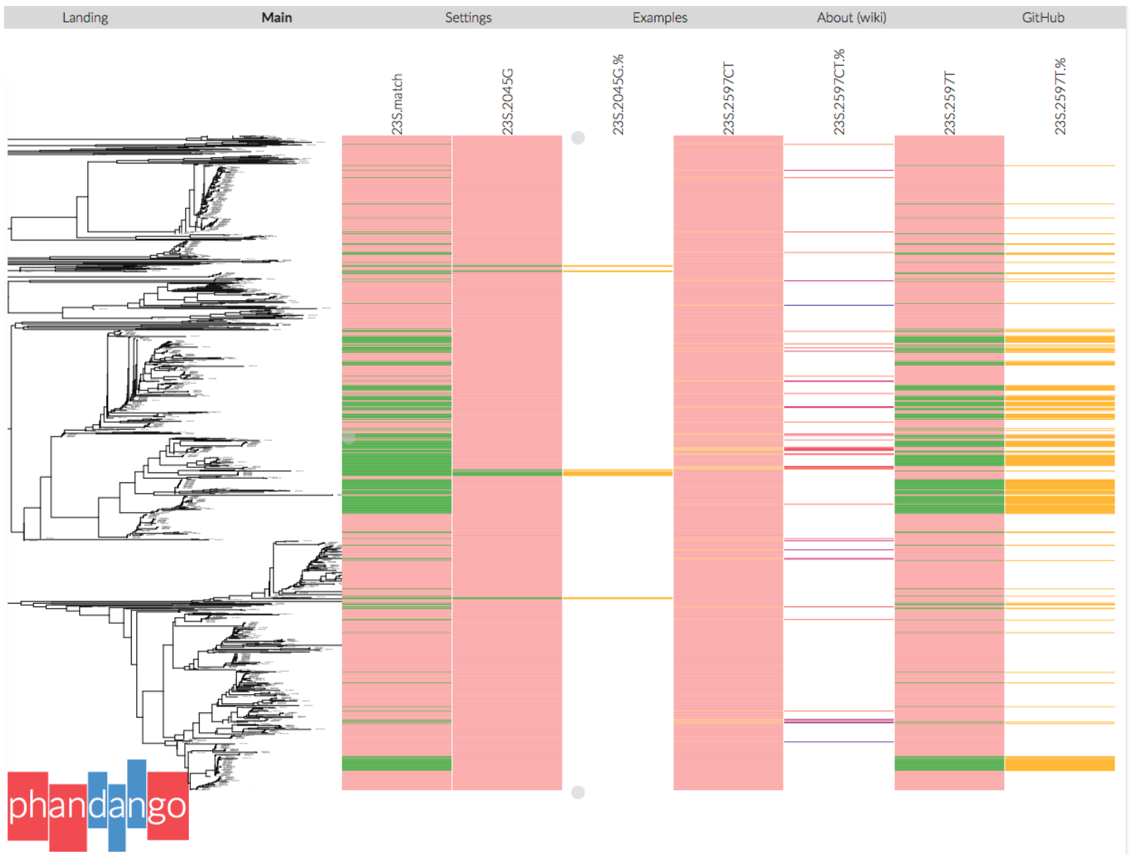
## 6.3 Variants

In the previous screenshot, where the option --preset cluster_all, there are two variant columns: “known_var” and “novel_var”. Green means “yes” and pink means “no”. Here, we are considering a variant to be a difference bewteen the reference sequence and the assembly contig from the reads. The known_var column indicates whether each sample has any variant that is known to ARIBA, which means it was included when the original ARIBA database was generated. The novel_var column indicates whether or not a sample has any variant that is not already known to ARIBA.

We can view the calls for all the known variants by adding the --known_variants option:

```
ariba summary --only_clusters 23S --known_variants --no_tree \
  --fofn filenames.fofn out ARIBA_reports/*.tsv
```

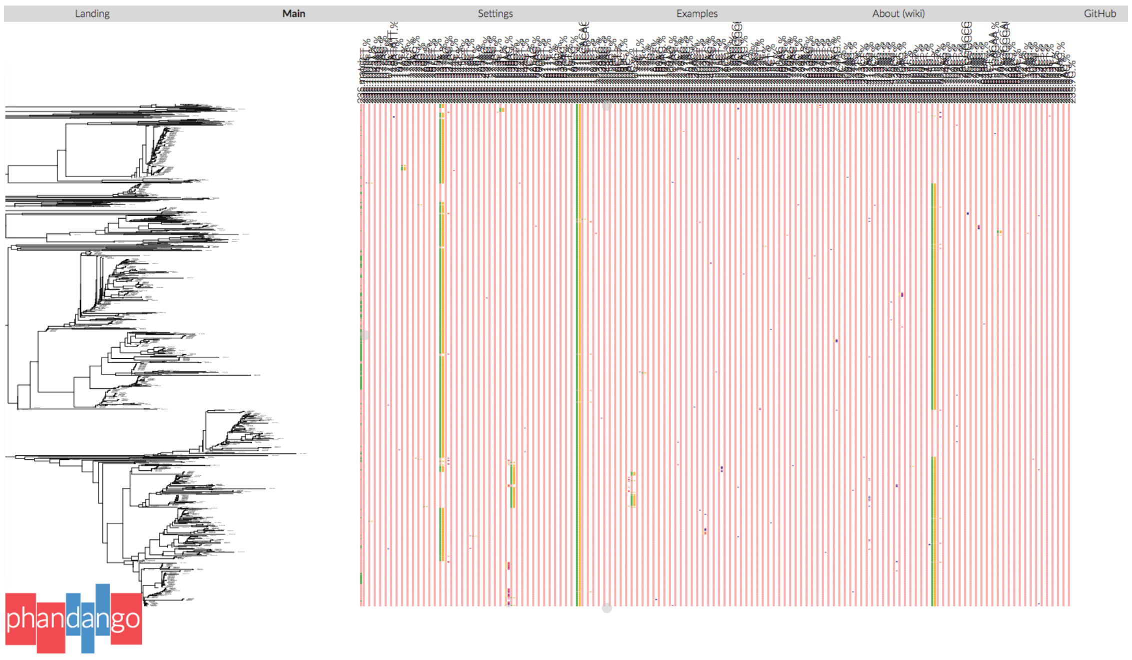
As before, drag and drop the files out.phandango.csv and tree_for_phandango.tre into the window. This time the result should like this:



Each known SNP and whether or not it is present is shown, but you may have noticed there are three colours. Green means "yes", orange means "heterozygous", and pink means "no". *N. gonorrhoeae* has four copies of 23S. A SNP could be present in none, some, or all copies. Where it is present in 1,2, or 3 copies, it is called "heterozygous" by ARIBA. Consider the SNP 2597CT in column 4, and column 5 to the right "2597CT.%". Samples either do not have this SNP, or have heterozygous calls. The 2597CT.% shows the percent of reads that have the SNP, when mapped to the contig. Hover over the coloured blocks in Phandango to see the percentage.

```
ariba summary --only_clusters 23S --novel_variants --no_tree \
  --fofn filenames.fofn out ARIBA_reports/*.tsv
```

Now Phandango shows all variants found in any of the samples (even if the variant is unique to one sample). This results in a lot of columns!



Now go to the next part of the tutorial where we Investigate MIC data in relation to variants in the samples.

## 7 Investigating MIC data

We will use the *N. gonorrhoeae* dataset again. This tutorial includes pre-computed output of running ARIBA on all the samples, and the ARIBA database that was made in the earlier section.

ARIBA has a function called "micplot" that generates plots showing the distribution of MICs across samples with different combinations of genotypes. To use it, a file is required of MIC data for each sample and at least one drug. It looks like this:

```
head mic_data.tsv
```

The first column must be named "Sample" and have names that exactly match those in ARIBA summary files used as input to micplot (we will see this shortly). The remaining columns should contain drug names and MIC scores, however, note that the first two columns contain other data that will be ignored by ARIBA. When ARIBA loads the file, it tries to convert everything in columns 2 onwards to numbers and assign a value of "NA" when this is not possible.

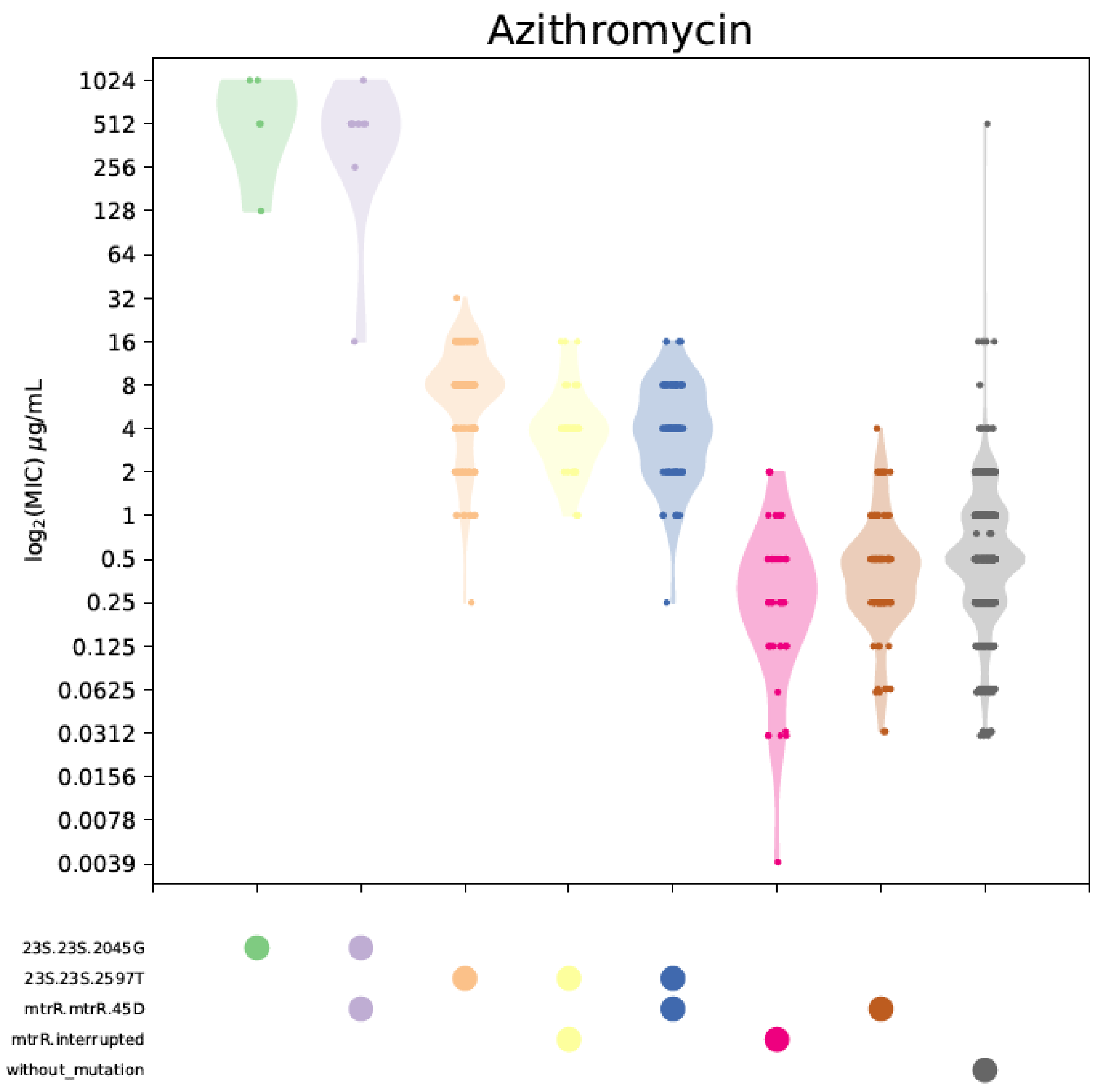
To run micplot, we need an MIC file, like the one above, and an ARIBA summary file (as described in the previous section). This generates a summary of known 23S and mtrR mutations and includes the "assembled" cluster column, so that interrupted mtrR can be identified:

```
ariba summary --row_filter n --cluster_cols assembled,known_var \
    --only_clusters 23S,mtrR --v_groups --no_tree \
    --fofn filenames.fofn summary.AZMknowngroups
```

Now we can run micplot using the new file `summary.AZMknowngroups.csv` and the MIC file `mic_data.tsv`, showing the MIC data for azithromicin compared with the different combinations of sequences and known mutations in 23S and mtrR:

```
ariba micplot Ref/Ngo_ARIBAdb/ --interrupted Azithromycin \
    mic_data.tsv summary.AZMknowngroups.csv \
    micplot.AZMknowngroups
```

This produced a pdf file `micplot.AZMknowngroups.pdf` that looks like this:



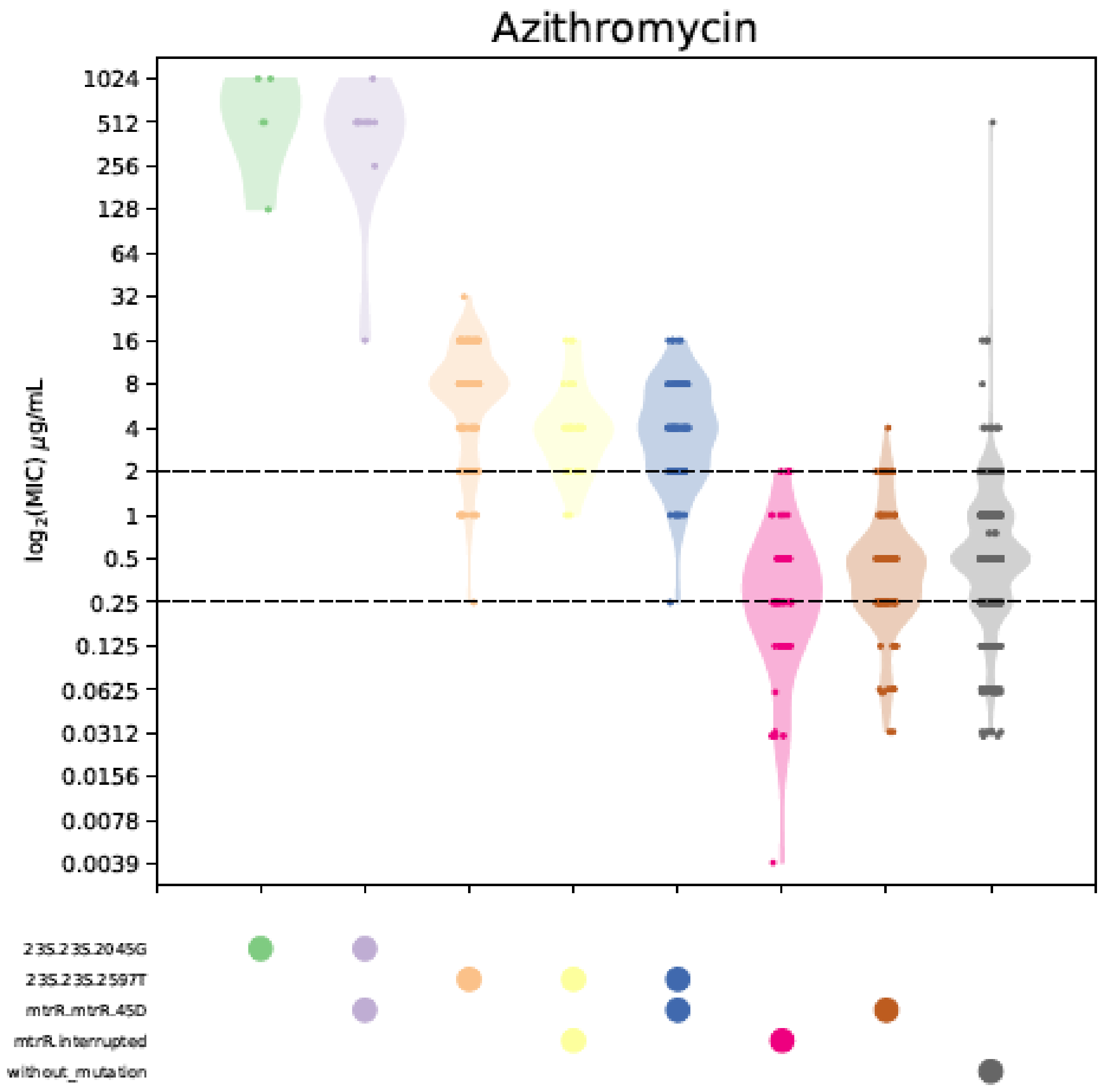
There are various options that can be changed. We will show some of them here, but try running `ariba micplot --help` to see all the options.

## 7.1 Horizontal lines

Horizontal lines can be added to indicate import cutoffs for MIC data, in this case 0.25 and 2, using the option `--hlines`.

```
ariba micplot Ref/Ngo_ARIBAdb/ --interrupted --hlines 0.25,2 \
    Azithromycin mic_data.tsv summary.AZMknowngroups.csv \
    micplot.AZMknowngroups
```

Here is the result:

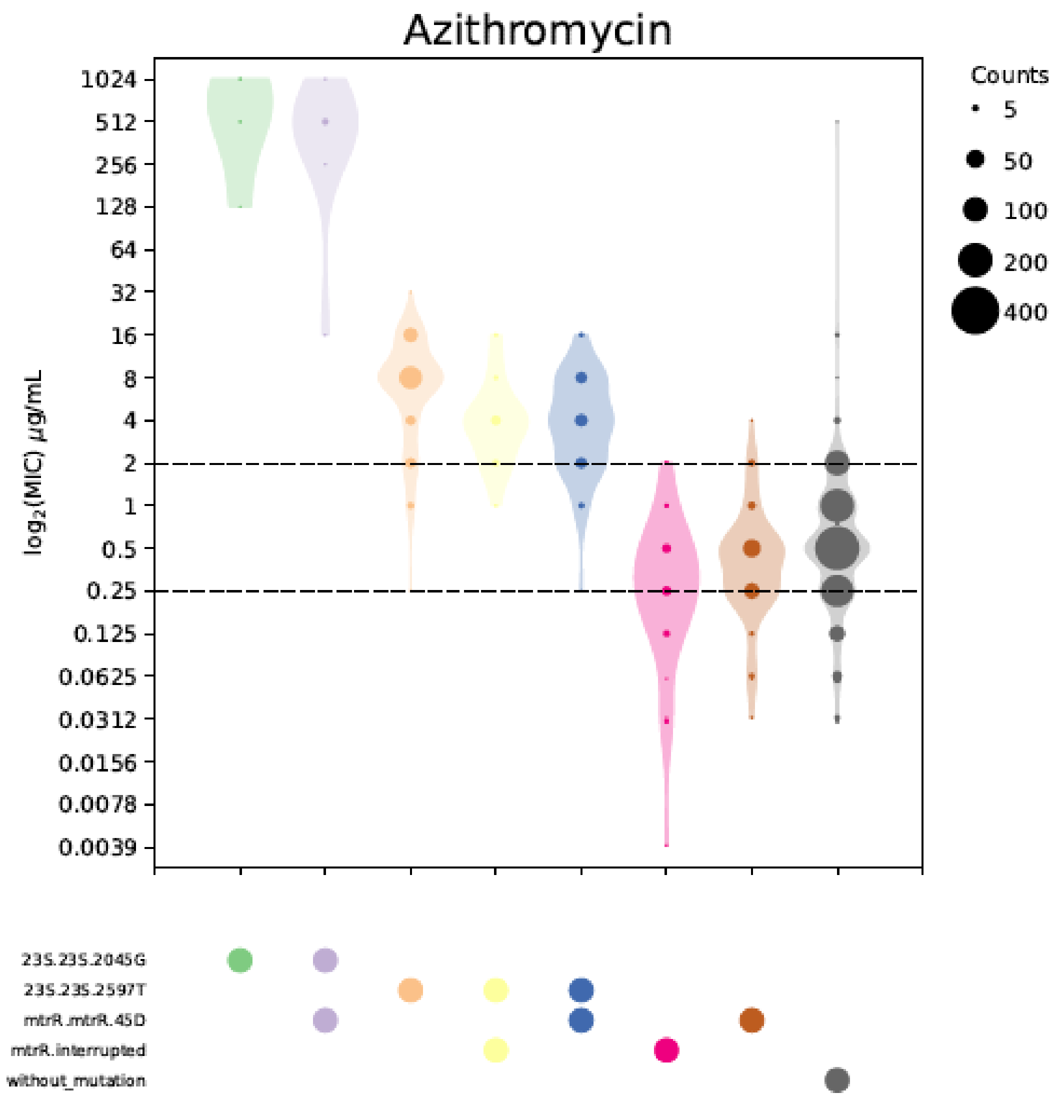


## 7.2 Plot styles

In the plots above, there is one point per sample. It can be hard to see how many points there are, despite there being jittering applied to the horizontal position. We can change the style to group the points together and plot circles of sizes proportional to the number of samples, using the option `--point_size`. This option determines the size of the points, but when set to zero if groups the points together.

```
ariba micplot Ref/Ngo_ARIBAdb/ --interrupted --hlines 0.25,2 \
    --point_size 0 Azithromycin mic_data.tsv \
    summary.AZMknowngroups.csv micplot.AZMknowngroups
```

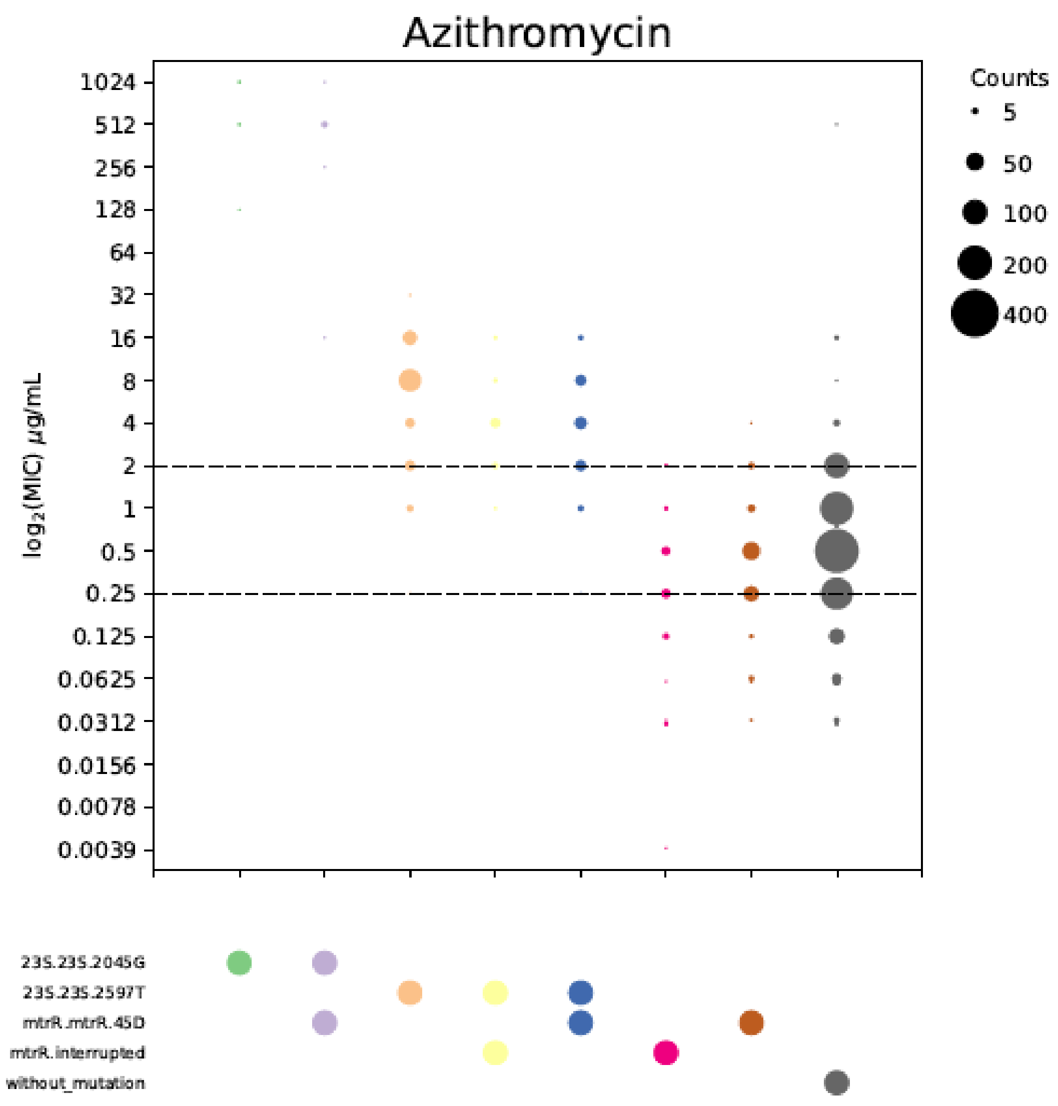
Here is the result:



You can choose to not show the violin plots or the dots in the upper plot, using the option `--plot_types`. The default is `violin,point`, which means show both. To only show the dots:

```
ariba micplot Ref/Ngo_ARIBAdb/ --interrupted --hlines 0.25,2 \
 --plot_types point --point_size 0 \
 Azithromycin mic_data.tsv summary.AZMknowngroups.csv \
 micplot.AZMknowngroups
```

Here is the result:

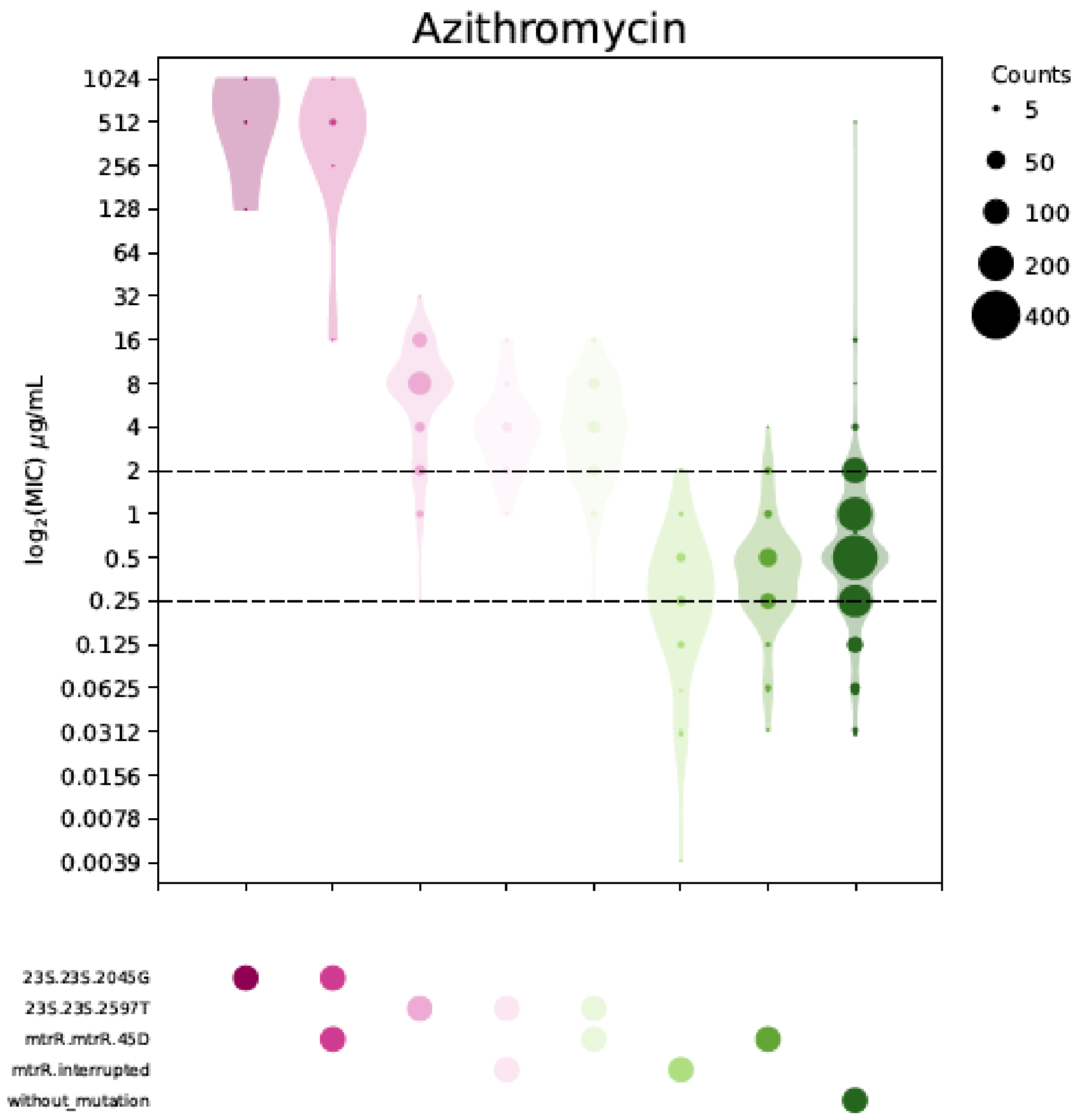


## 7.3 Colours

There are various colour options - see the matplotlib colourmaps page for all of the available colour palettes. The default is “Accent”, which has 8 colours. ARIBA will cycle through these, repeating colours if there are more than 8 columns in the plot. The palette can be changed using the option `--colourmap`.

```
ariba micplot Ref/Ngo_ARIBAdb/ --interrupted --hlines 0.25,2 \
 --colourmap PiYG --point_size 0 Azithromycin mic_data.tsv \
 summary.AZMknowngroups.csv micplot.AZMknowngroups
```

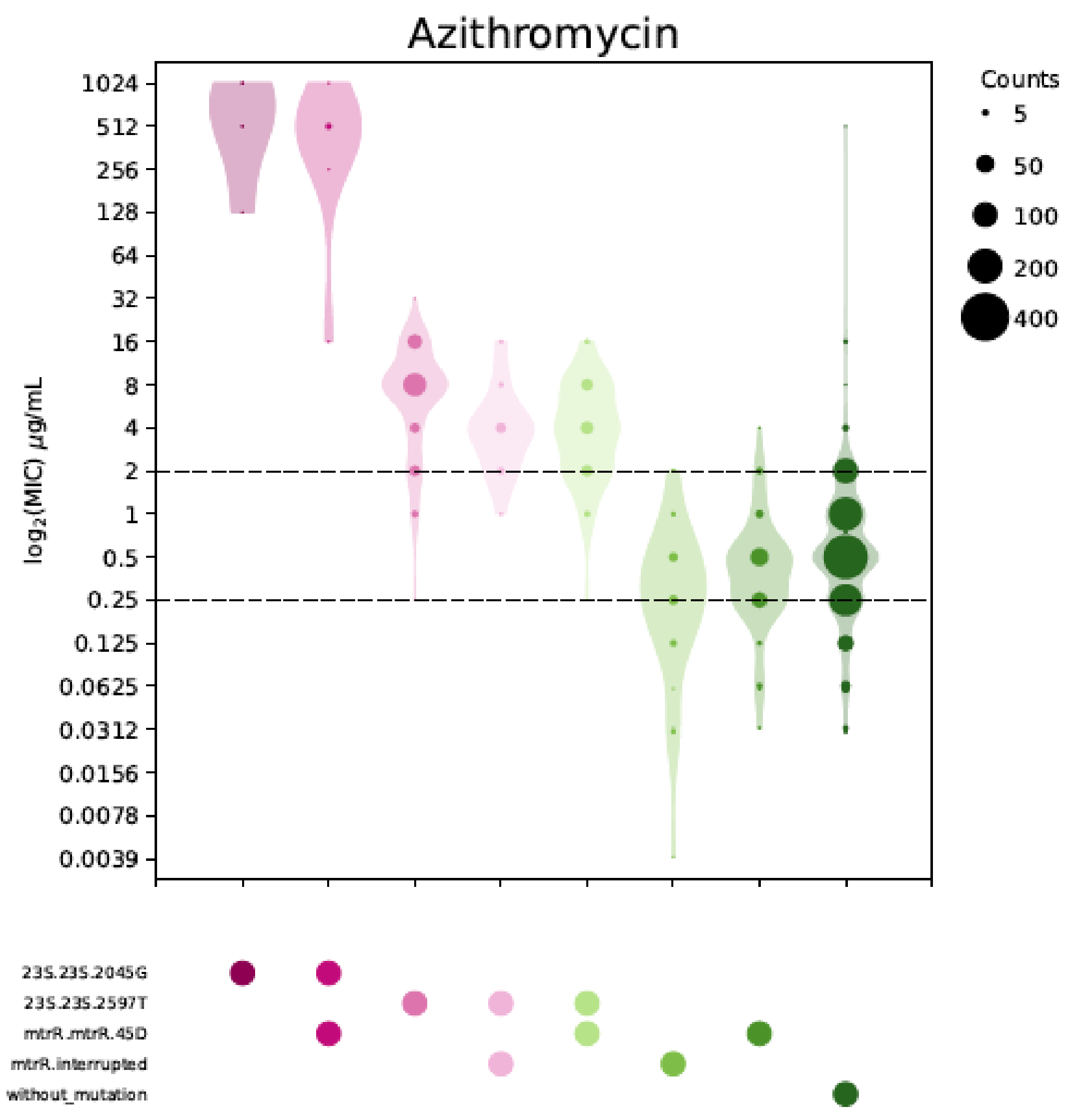
Here is the result:



The palette PiYG is continuous, and is almost white in the middle. This is not ideal. We can skip the range in the middle, specifically 40-60%, using the option `--colour_skip`:

```
ariba micplot Ref/Ngo_ARIBAdb/ --interrupted --hlines 0.25,2 \
 --colourmap PiYG --colour_skip 0.35,0.65 --point_size 0 \
 Azithromycin mic_data.tsv summary.AZMknowngroups.csv \
 micplot.AZMknowngroups
```

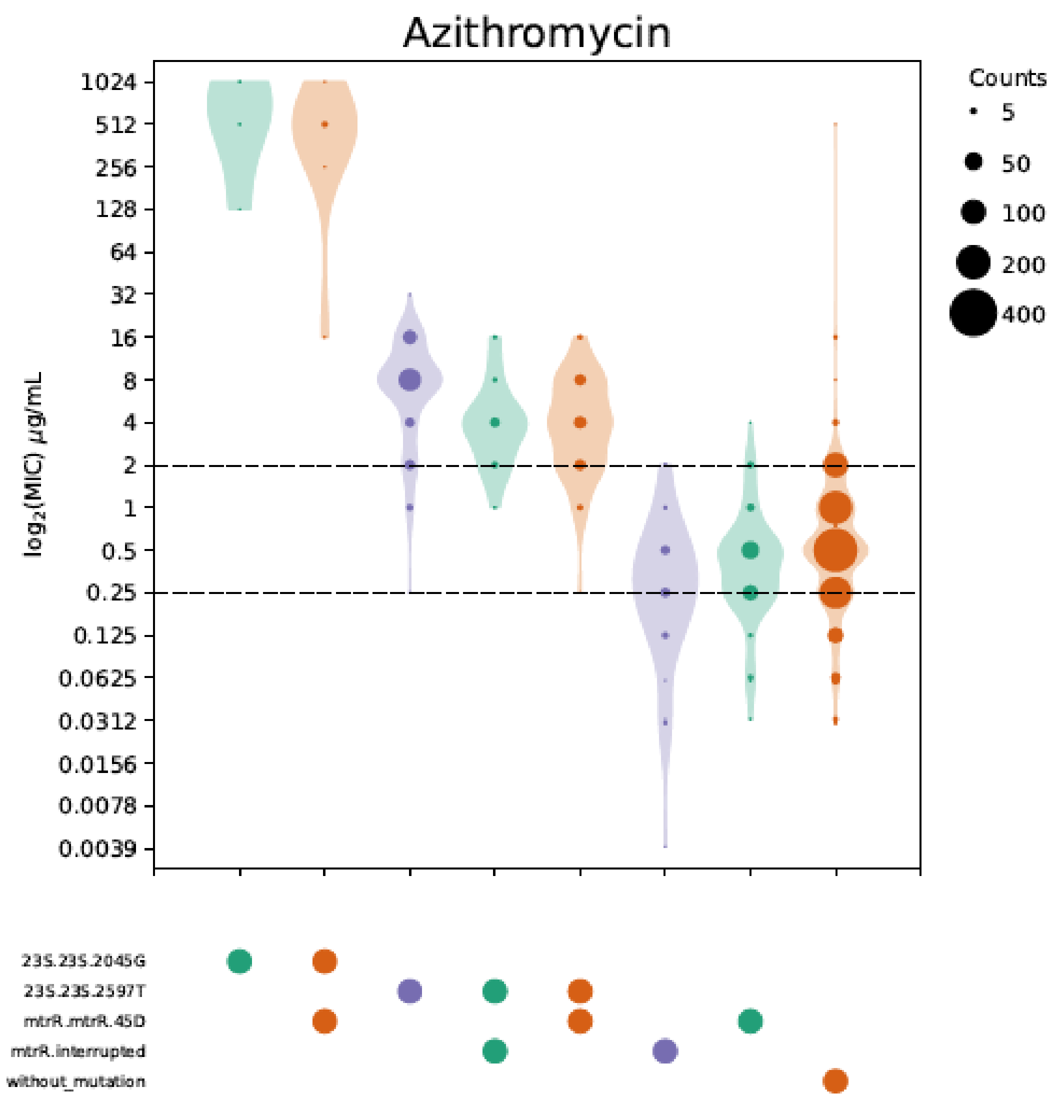
Here is the new plot:



The number of colours can be set to less than the number of columns using the option `--number_of_colours`. This makes ARIBA cycle the colours. Here is an example using the first three colours from the "Dark2" colour palette:

```
ariba micplot Ref/Ngo_ARIBAdb/ --interrupted --hlines 0.25,2 \
 --colourmap Dark2 --number_of_colours 3 --point_size 0 \
 Azithromycin mic_data.tsv summary.AZMknowngroups.csv \
 micplot.AZMknowngroups
```

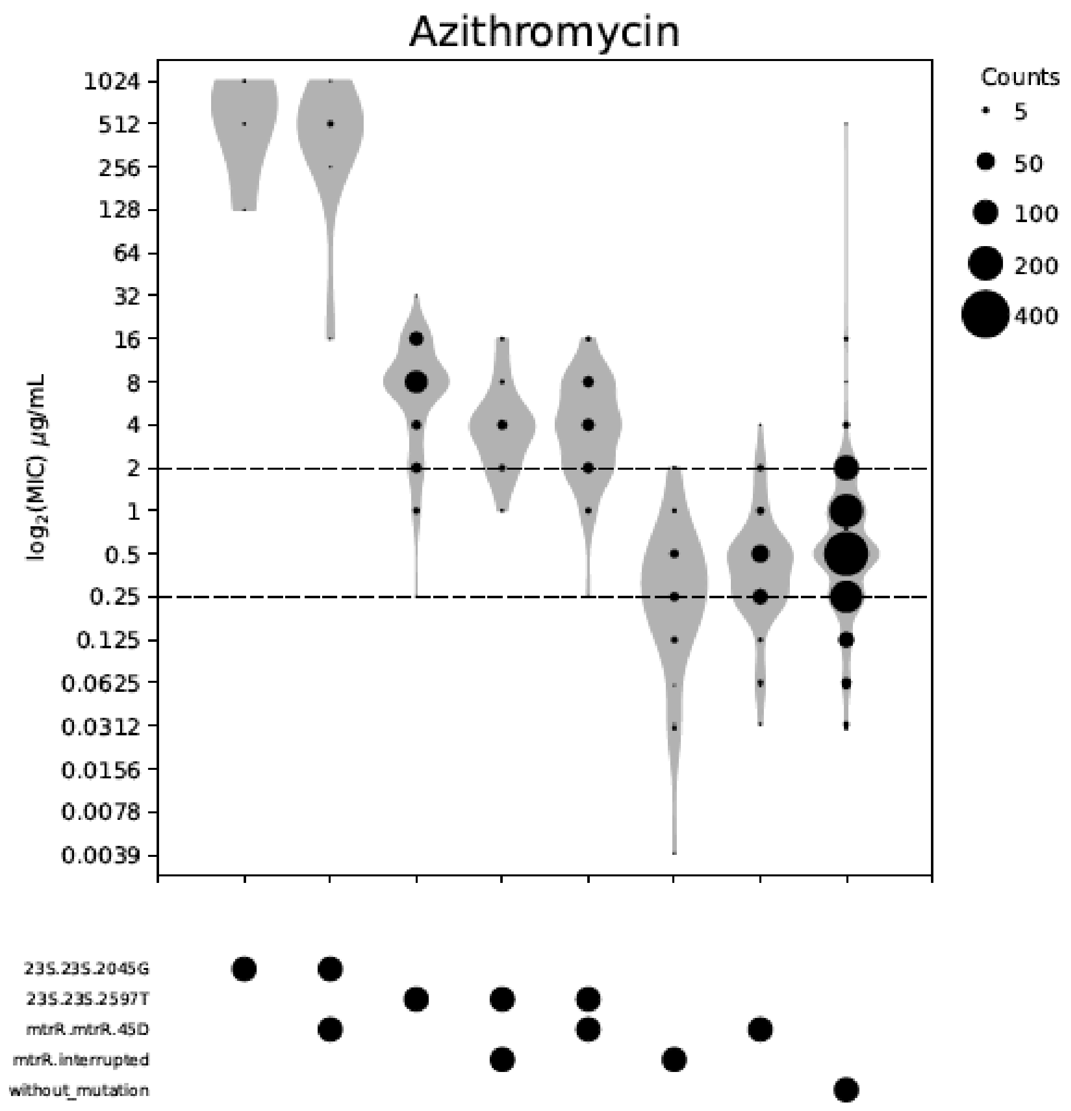
And we only have three colours:



Setting the number of colours to one results in a black and white figure.

```
ariba micplot Ref/Ngo_ARIBAdb/ --interrupted --hlines 0.25,2 \
 --number_of_colours 1 --point_size 0 Azithromycin \
 mic_data.tsv summary.AZMknowngroups.csv micplot.AZMknowngroups
```

Here is the black and white figure:



Congratultaions, you have reached the end of the tutorial!