

# **ONE ARM Metagenomics Pipeline**

Prepared for and by: Project 4

Last updated October 5, 2024

# Contents

<b>I</b>	<b>Project overview</b>	<b>4</b>
<b>1</b>	<b>Data collection</b>	<b>6</b>
1.1	Procurement . . . . .	6
<b>2</b>	<b>Bioinformatics Progress</b>	<b>7</b>
2.1	Kanban table . . . . .	7
<b>II</b>	<b>Bioinformatics work</b>	<b>11</b>
<b>3</b>	<b>Project 4 Scripts</b>	<b>12</b>
3.1	ARG-MGE.smk . . . . .	12
3.1.1	Pair merging . . . . .	12
3.1.2	Translation and Reverse Translation . . . . .	12
3.1.3	Iterative alignment of ARG-contigs . . . . .	13
3.1.4	Merging of paired reads . . . . .	13
3.1.5	Guided Metagenomic Assembly . . . . .	14
3.1.6	Confirmation of contigs with ARGs . . . . .	14
3.1.7	Standard Contig Quality Metrics . . . . .	14
3.1.8	Read-mapping . . . . .	14
	Read mapping parameters . . . . .	15
3.1.9	Taxonomic profiling of reads and contigs . . . . .	16
3.1.10	Detect structural variants (SVs) in contigs . . . . .	17
3.1.11	Sketching contigs followed by calculating Bray–Curtis diversity . . . . .	17
3.1.12	Determination of Transposons (TBA) . . . . .	17
3.1.13	Determination of putative plasmids . . . . .	17
3.1.14	Phage influence signatures . . . . .	18
3.1.15	Phage Signature Extraction and Phylogenetic Analysis . . . . .	19
3.1.16	Directory tree . . . . .	19
3.1.17	Future Considerations . . . . .	20
3.2	metagenomics_general.smk . . . . .	21
3.2.1	Quality control (Pre-processing) . . . . .	21
3.2.2	Trimming . . . . .	21
3.2.3	Metagenomic Taxonomy . . . . .	22

3.2.4	Diversity analysis . . . . .	22
3.2.5	Directory tree . . . . .	23
3.3	trim_randomizer.smk . . . . .	24
3.3.1	Random Parameter Generation . . . . .	24
3.3.2	Log Parameters to a TSV File . . . . .	24
3.3.3	Define Rules for Each Tool . . . . .	24
3.3.4	Interpretation and Analysis of Results (TBA) . . . . .	25
3.3.5	Annotated Directory Tree with File Temporary files and Prerequisites . . . . .	25
3.4	bootstrapping_rawreads.smk . . . . .	27
3.4.1	Directory setup of temporary files . . . . .	27
3.4.2	Sample Identification . . . . .	27
3.4.3	Bootstrapping . . . . .	27
3.4.4	Annotated Directory Tree with File Movement and Prerequisites . . . . .	28
3.4.5	Run kraken_pipeline.bash . . . . .	29
3.5	Binning.smk . . . . .	30
3.5.1	Universal Configurations . . . . .	30
3.5.2	FastUniq (Deduplication) . . . . .	30
3.5.3	Seqtk (FASTA Conversion) . . . . .	30
3.5.4	CD-HIT-EST (Clustering) at Identity: 90% . . . . .	31
3.5.5	MetaWRAP Binning and Reassembly . . . . .	31
3.5.6	DAS Tool (Bin Refinement) . . . . .	31
3.5.7	MAGpurify (Contamination Removal) . . . . .	31
3.5.8	MetaQUAST (Assembly Quality Assessment) . . . . .	32
3.5.9	dRep (Dereplication) . . . . .	33
3.5.10	CheckM2 . . . . .	33
3.5.11	Annotated Directory Tree with File Movements and Temporary Files . . . . .	33
3.6	krakenpipeline.bash . . . . .	35
3.6.1	Directory tree . . . . .	35
3.7	raw.bash; completely optional . . . . .	35
3.8	summary_stat.bash . . . . .	36
3.8.1	Directory tree . . . . .	36
3.9	plot_and_detect_intermediate_coverage.py . . . . .	37
3.10	Shortbred_ARG.bash . . . . .	37
3.11	refseq_bacteria.bash . . . . .	39
3.12	trimmomatic.bash; pipeline module . . . . .	39
3.13	renamingSIMS.bash . . . . .	39
3.13.1	Directory Tree with Notations . . . . .	40
3.14	prokka_ARG.bash . . . . .	40
3.14.1	Directory tree . . . . .	40
3.15	RefSeq.bash . . . . .	41
3.16	Pavian_analysis.R . . . . .	41
3.17	parseFastQC.py . . . . .	42

3.17.1	Directory tree . . . . .	42
3.18	minimum_length_CARD.py . . . . .	42
3.19	template modulars: megahit_binning.sh . . . . .	43
3.20	diversity_bootstrap.R . . . . .	43
3.21	calculate_plasmid_percentage.py . . . . .	44
3.22	calculate_diversity.py . . . . .	44
<b>4</b>	<b>Project Side Scripts</b>	<b>45</b>
4.1	kmer_contam.smk . . . . .	45
4.1.1	Contaminant Databases Download . . . . .	45
4.1.2	BUSCO Validation . . . . .	46
4.1.3	Marker K-mer Generation . . . . .	46
4.1.4	Mapping K-mers . . . . .	46
4.1.5	Normalization and Testing . . . . .	47
4.1.6	Filtering Ambiguous Sequences . . . . .	48
4.1.7	Re-validation of SCGs . . . . .	48
4.1.8	Directory tree . . . . .	48
<b>5</b>	<b>Project Main Scripts</b>	<b>50</b>
5.1	. . . . .	50
<b>III</b>	<b>Further investigations</b>	<b>51</b>
<b>IV</b>	<b>Daily entries</b>	<b>54</b>
1	September 26, 2024 . . . . .	55
2	September 25, 2024 . . . . .	56
3	September 23, 2024 . . . . .	57
4	September 22, 2024 . . . . .	58
5	September 21, 2024 . . . . .	59
6	September 20, 2024 . . . . .	60
7	September 19, 2024 . . . . .	61
8	September 18, 2024 . . . . .	62
9	September 17, 2024 . . . . .	63
<b>V</b>	<b>Supplementary data</b>	<b>64</b>
<b>VI</b>	<b>Information dump</b>	<b>66</b>
1	Biological OFF "Decay" . . . . .	67
2	Best Data science practices . . . . .	68

## **Part I**

# **Project overview**

Metagenomic analysis of Antibiotic/Antimicrobial Resistance Genes (ARGs) in NCR (Metro Manila) hospitals, wastewaters, and surface waters.

### Legend

✓ Done

□ Pending

🔄 Needs refinement

△ Unexpected issues

✎ Drafted

➡ Moved

• Essential

• Optional

• Robust

• Deprecated

• TBA - needs further investigation

# **Chapter 1**

## **Data collection**

### **1.1 Procurement**

## Chapter 2

# Bioinformatics Progress

### 2.1 Kanban table

Section	Task	<input type="checkbox"/>					
HPC Preparation	Confirmation of Required Robustness						X
	Agree upon the type of analyses					X	
	SLURM request management						
	Calculate using quotation from PGC						X
	Setup Docker containers for SLURM	X					
	SLURM container for simulations	X					
	Automation of SLURM requests	X					
File Management	Send email for HPC usage requests			X			
	BAM file parsing	X					
	Interconversion between SAM and BAM	X					
Raw Read Processing	Create the script						
	Raw reads QC						X
	raw.bash						
	Raw reads trimming						X
	trimmomatic.bash						
	fastp.bash						
	Looping mechanism for QC and trimming						X
	trimming-cleaning-checking.py						
	Data visualization	X					
	Determination of optimal tool/s	X					
	Parametric randomization						X
	trim_randomizer.smk						
	Parse QC of all metrics in parametric randomizer						X
	parseFastQC.py						
	Determination of optimal parameters	X					
	Tool combination randomization script	X					



Section	Task	<input type="checkbox"/>					<input checked="" type="checkbox"/>
	<b>Test on Datasets</b>						
	Raw reads QC						X
	Raw reads trimming						X
	trimmomatic.bash						
	fastp.bash						
	trimming-cleaning-checking.py						
	Aggregate quality metrics						X
	summary_stat.bash						
	Parametric randomization						X
	Parse QC of all metrics in parametric randomizer						X
	<b>Integrate to pipelines</b>						
	Pipeline integration of raw.bash	X					
	Raw reads trimming						X
Taxonomy	<b>Create the script</b>						
	Kraken2						X
	krakenpipeline.bash						
	Bracken						X
	krakenpipeline.bash						
	Diversity indices						X
	calculate_diversity.py						
	MetaPhlan4	X					
	<b>Test on Datasets</b>						
	Kraken2						X
	Bracken						X
	Diversity indices						X
	<b>Integrate to pipelines</b>						
	Kraken2						X
	Bracken						X
	Diversity indices						X
	Test whole pipeline						X
	metagenomics_general.smk						
	Bootstrapping script						X
	bootstrapping_rawreads.smk						
Assembly	<b>Create the script</b>						
	metaSPAdes	X					
	MEGAHIT						X
	MASURCA	X					
	PLASS	X					
	AbySS	X					
	<b>Test on Datasets</b>						
	metaSPAdes	X					

Section	Task	<input type="checkbox"/>					
	MEGAHIT						X
	MASURCA	X					
	PLASS	X					
	AbySS	X					
	KMA-iterative	X					
	Benchmarking between all of them	X					
	Contig quality checking		X				
	Integrate to pipelines						
	MetaWrap binning				X		
	Kraken (MEGAHIT)						X
Binning	Testing on datasets				X		
	Testing on higher depth datasets	X					
	Refinement of bins	X					
	Testing binning pipeline	X					
ARG Annotation	RGI			X			
	ShortBRED						X
	AMRFinder			X			
	Test on datasets						
	RGI	X					
	ShortBRED	X					
	AMRFinder	X					

## Script Descriptions

This section covers all the scripts that were created during the Project.

*Listen, I'm eternally curious and I don't just want to settle for any random journal. No offense, but I'm aiming for something high-impact!*

These were created during off hours or during work hours, so some scripts might seem irrelevant at first, *but trust me, there's a conscientiousness or meticulousness to this madness.*

I have separated them into folders/repositories (shameless plug here: <https://github.com/GABallena>) based on their relevance to the project.

- **Project4** - Essential scripts directly related to the core analyses of the project.
- **Side** - Scripts that can potentially be used to increase the robustness of the paper.
- **Main** - Scripts related to file organization and data management, crucial for handling large datasets.

## **Bioinformatics work**

# Chapter 3

## Project 4 Scripts

### 3.1 ARG-MGE.smk

#### Stage: Draft General Purpose

This pipeline is designed for **comprehensive metagenomic analysis of ARGs**, it also includes placeholders for analyses of **mobile genetic elements (MGEs)**, and **plasmid detection**. It integrates tools for **read quality control, assembly, annotation, taxonomic profiling, and structural variant (SV) detection** to provide a high-resolution view of the genetic components in metagenomic samples.

#### Preprocessing

---

#### 3.1.1 Pair merging

**Technical Notes:** PEAR (Paired-End reAd mergeR) compares paired-reads to correct infer the likely bases on its associated pair

**Rationale:** The main goal here is to ensure good read quality for downstream analyses and to maximize the amount of data by reducing gaps in the sequence and improving confidence of base calls within that region. seqtk then converts compressed FASTQ files to FASTA.

#### Rationale

**Note:** Another tool, PandaSeq which does the same thing is used later, this usage of PEAR here is because it is more optimized for larger datasets - which in this case are trimmed reads. **Note:** Conversion is necessary here as some tools cannot read FASTQ files, and instead rely on FASTA formats.

#### 3.1.2 Translation and Reverse Translation

**Technical Notes:** transeq from EMBOSS translates nucleotide sequences to protein sequences based on **standard genetic code**. backtranseq reverses this to allow iterative alignments with nucleotide sequences.

**Rationale:** This leverages conserved protein-level information, which is lost at the nucleotide level due to synonymous mutations - while also increasing the sensitivity to potential ARG proteins from k-mer alignment. See Nature Methods, doi: doi.org/10.1038/s41592-019-0437-4 (2019) for more details.

## Metagenomic Assembly

### 3.1.3 Iterative alignment of ARG-contigs

**Technical Notes:** KMA (K-mer Alignment) is used find the reverse-translated ARGs with the proximity filtering option to determine the surrounding regions of ARGs. This is done iteratively for each gene increasing the ARG-associated database.

**Rationale:** Firstly, KMA is used because, unlike Bowtie2 and BWA-MEM, which were created specifically for Human metagenomics, KMA does not suffer (or suffers less) from multi-allelic databases. Secondly, iterative alignment using this process allows us to contextualize the region wherein ARGs reside - thereby narrowing our focus onto these local regions instead of looking at the global genomic context. **Notes:** The script is designed to have a cap on the number of iterations KMA creates, increasing the database size.

#### Sep 22 2024 Update

This is now deprecated as there is apparently another wrapper tool called ARG Profiler that has a module called ARG Extender that literally does the same thing but with an extra filtering step that applies filters on:

1. % query ID
2. % global consensus ID
3. Mean read depth

and is thus more robust. So I will instead be using their module and cite them as such

```
"This study utilized the ARGextender  
module from ARGProfiler (Martiny, et al., 2024)  
to extend genomic flanking regions around ARGs."
```

**Personal Note:** *Feels bad a bit on my part that someone already published the idea before I did but hey, at least I know this sort of idea is publishable haha.* Moreover, it does save us time because I only have to copy-paste their module (and associated scripts) into the existing pipeline.

### 3.1.4 Merging of paired reads

**Technical Notes:** PANDASeq is then used to further refine the paired-reads collated in the ARG-related genes database.

**Rationale:** PANDASeq was chosen as, while being slower than PEAR, it is more accurate. This mergins step is included to ensure that only high-confidence reads are assembled. **Note:** We leverage the fact that the ARG-related genes database is smaller compared to the raw metagenomic reads database.

### 3.1.5 Guided Metagenomic Assembly

**Technical Notes:** metaSPAdes is then used to create contiguous sequences from these local regions by extending them using reads from the whole metagenomic pool. Additionally, Contigs are filtered by length here to remove possible artefacts.

**Rationale:** metaSPAdes was chosen as, while being slower than MEGAHIT, it is optimized in handling highly diverse and mixed microbial populations. CARD is used here because it is manually curated and updated regularly - in order to be included in the database, there must be clinical data (e.g. ASTs(Antimicrobial susceptibility testing)) involved in the study.

**Notably, this would decrease the sensitivity of our ARGs - and would mostly be biased towards those reported in the clinical setting.** To counter this, we could also incorporate other tools such as ResFinder, the NCBI AMR Database, and ARG-ANNOT

**Notes:** Filtering of contig length is handled by a python script called `minimum_length_CARD.py`.

**Notes:** Contigs are further extended using `contigextender` to form scaffolds

**Notes:** Might add other contig extending programs like `GapFiller` - which leverages mate-pair information

### Contig Quality Checks

---

### 3.1.6 Confirmation of contigs with ARGs

**Technical Notes:** RGI scan the contigs and check whether which contigs created by metaSPAdes have ARGs in them.

**Rationale:** metaSPAdes may have created contigs that DO NOT contain ARGs, and have instead assembled them into a more matching contig (a false-positive misassembly) - this can happen because of the different databases being used; also parallelization of methods like this increases robustness because it has been confirmed independently from different starting points (bottom-up vs top-down approach). This allows us to filter ARG-containing contigs.

**Notes:** RGI is the official scanner of CARD.

### 3.1.7 Standard Contig Quality Metrics

**Technical Notes:** A custom Python script `calculate_contig_quality.py` is created to do another round of checking contig quality for downstream analysis, R scripts (TBA) are used to visualize the data.

**Notes:** The Python script will measure standard contig quality metrics: N50, L50, GC-content, and coverage, as well as more robust metrics: N90 and L90.

### 3.1.8 Read-mapping

**Technical Notes:** Samtools is used here to map the raw reads from the larger database back to the assembled contigs and then calculates the coverage over the entire contig.

**Rationale:** Read-mapping is a quality control protocol used in metagenomics, to determine the quality of the assembly. High-coverage means that many of the k-mers align well with that region of the contig, while low coverage is evidence of inconsistent mapping and that the contigs should be refined, split, or

discarded.

**Note** If there are persistent (after further refinement and reassemblies) sudden differences in coverage across a contig, that contig could be chimeric, meaning, it could be from two different populations.

**Extra note** A Python script `plot_and_detect_intermediate_coverage.py` is included in the pipeline that is determine visualize and check how the coverage changes over contig regions. In general, they could be interpreted as the following:

1. **Smooth, Uniform Coverage:** Typically shown by well-assembled contigs.
2. **Sharp Coverage Drop:** May need to be split or flagged for reassembly. May also be a misassembly point (chimeric contig) or caused by a structural variant.
3. **Coverage Gaps:** Regions with little to no read support; a strong indicator of misassembly.
4. **Gradual Drops:** Overlapping reads, repetitive or duplicate regions, partial HGT, sequence heterogeneity, or coverage differences due to a mixed population. Repeats and duplications can be filtered out using tools like RepeatMasker or BLAST (TBA).
5. **Sharp Increase:** May be due to repetitive or duplicated regions, amplification bias from PCR, HGT, SV, chimeras.

## Read mapping parameters

**Technical Notes:** Four (4) Tools will be used in parallel to do the read mapping process BWA Bowtie KMA, and minimap2, their parameters have been adjusted to map reads at 95 % identity to the contigs.

Mapper	K-mer Length	Mismatch Penalty	Gap Opening Penalty	Gap Extension Penalty
BWA	21	5	7	2
BWA	31	4	6	2
BWA	51	3	5	1
Bowtie2	-	4,2	5,2	5,2
KMA	Default	95% identity	Automatic	Automatic
Minimap2	-	5	7,2	4,1

Table 3.1: Example table of mapper configurations without command example

**Rationale:** 95 % identity is used to increase sensitivity - as is standard for determining homologous sequences. This adjustment was made because k-mers are either attach or don't. Parallelization is used to increase robustness.

**Note** K-mer extension is used to increase the accuracy of mapping. BWA k-mer lengths can be adjusted, while KMA does it by default. The others, cannot be adjusted.

### Note

I chose to change the script to not allow gaps during this phase as we already used reverse translation earlier to correct for synonymous codons, and protein sequences are more important when it comes to ARG function, the new values are below. I also added protein-based read mapping.



Tool	K-mer Length	Mismatch Penalty	Gap Opening Penalty	Gap Extension Penalty
BWA	21, 31, 51	5, 4, 3	1000	1000
Bowtie2	-	4,2	1000,1000	1000,1000
KMA	-	95% identity	Automatic	Automatic
Minimap2	-	5	1000,1000	1000,1000

Table 3.2: Alignment parameters for ungapped alignments across BWA, Bowtie2, KMA, and Minimap2

Tool	Input Files	Key Parameters
<b>tblastn</b>	<ul style="list-style-type: none"> <li>Protein sequences: Translated protein sequences contigs</li> <li>Nucleotide database: Cleaned sequence database</li> </ul>	<ul style="list-style-type: none"> <li>-outfmt 6</li> <li>-evalue 1e-5</li> <li>-gapopen 5</li> <li>-gapextend 2</li> <li>-matrix BLOSUM62</li> </ul>
<b>blastp</b>	<ul style="list-style-type: none"> <li>Protein sequences: Translated protein sequences contigs</li> <li>Protein database: Translated cleaned sequence database</li> </ul>	<ul style="list-style-type: none"> <li>-outfmt 6</li> <li>-evalue 1e-5</li> <li>-gapopen 5</li> <li>-gapextend 2</li> <li>-matrix BLOSUM62</li> </ul>

Table 3.3: Protein read-mapping parameters for tblastn and blastp

**Rationale:** The main rationale for adding a protein-based read mapping protocol is because ARGs are primarily about their *protein-protein interactions* (biological relevance). This method also accounts for frameshifts with higher specificity to homologous regions.

**On a personal note:** This approach may also require further exploration into whether protein-protein interactions are altered—perhaps by investigating changes in binding sites. Which is a story for another day (*why do I do this to myself?*)

### 3.1.9 Taxonomic profiling of reads and contigs

**Technical Notes:** Kraken2 uses k-mer-based classification to assign taxonomy based on raw-reads. While SprayNPray complements this by assigning taxonomy at the contig level.

**Rationale:** By comparing their respective databases with our ARG-related databases, we will be able to

connect our reads and/or contigs to their corresponding taxa, uncovering the microbial hosts responsible for carrying and potentially spreading ARGs in the environment.

### 3.1.10 Detect structural variants (SVs) in contigs

**Technical Notes:** Manta identifies large genomic rearrangements such as insertions, deletions, and duplications.

**Rationale:** Chimeric contigs may be due to systematic error or real biological signals. These chimeric contigs can be detected by Kraken2 and SprayNPray (i.e., when a portion of a contig is being assigned to different taxa). The rationale behind this step is to investigate whether structural variations are present — which may be evidence of horizontal gene transfer (HGT) events.

### 3.1.11 Sketching contigs followed by calculating Bray-Curtis diversity

**Technical Notes:** Mash Sketch uses a MinHash approach to generate a presence/absence profile of ARGs across contigs. This gives us a quick snapshot of what the contigs “look like” in terms of ARG content. For Bray-Curtis diversity, we calculate a dissimilarity matrix from the abundance data of ARGs, followed by a PCoA plot to visualize similarities between contigs based on their ARG profiles.

**Rationale:** The Mash Sketch helps rapidly identify the genetic makeup of contigs in terms of ARGs, which provides a foundation for further investigation. By applying Bray-Curtis diversity and using PCoA, we can group contigs based on their ARG similarity. If contigs with the same sketch group together, we can trace them back to their taxonomic IDs to identify the microbial hosts. However, if contigs have similar ARG profiles but belong to different taxa, this could serve as **evidence for Horizontal Gene Transfer (HGT)**. This dual approach allows us to trace ARG spread and potential HGT events in a metagenomic context.

**Note:** Bray-Curtis (dis-)similarity is most often used as a presence or absence diversity metric.

### Transposition

---

### 3.1.12 Determination of Transposons (TBA)

**Technical Notes:** Tools such as the following can be used:

- HMMER3 suite
- Tnppred - a transposon predictor tool

**Rationale:**

### Plasmids

---

### 3.1.13 Determination of putative plasmids

**Technical Notes:** The tools listed below will be used in parallel. Plasmids are considered valid when all 4 tools predict plasmid signatures in the contig.

- PlasPredict pipeline
- Recycler
- PlasmidFinder
- MOBsuite plasmid marker annotator

If plasmid signatures are present, plasmidSPAdes along with GapFiller will be used to check if the contig can circularize. oriTfinder will then be applied to contigs with fewer than 4 fragments. A Python script (TBA) will calculate GC skews of the chimeric contig and compare it to its taxonomic counterparts. Another Python script (TBA) will normalize the data according to 16S rRNA from trimmed reads. Lastly, plasmid percentage will be calculated based on reads mapped to putative plasmids over the total reads. A code snippet is present called `calculate_plasmid_percentage.py`.

**Equation:**

$$\text{Plasmid Percentage} = \left( \frac{\text{Plasmid Reads}}{\text{Total Reads}} \right) \times 100$$

**Rationale:** oriTfinder looks for origin of transfer sites (oriT), which are characteristics of conjugative plasmid. This whole sub-pipeline is to look for evidence of conjugative plasmid transfer as the cause of these chimeric contigs. Normalization and percentage counts are used here to further check whether these "plasmids" align with our understanding of the average plasmid copy number.

**Note:** Will also be drafting a script (TBA) to do sliding window analysis of GC-skews - as different characteristics of this curve can be interpreted in different ways.

## Phages

---

### 3.1.14 Phage influence signatures

**Technical Notes:** They will be determined using a variety of tools:

- VirSorter: Identifies viral signatures within microbial genomes and separates prophages from bacterial sequences.
- PHASTER: A web-based tool for phage search and annotation, identifying integrated prophages.
- VIBRANT: A tool that combines several approaches to identify and annotate phage elements in metagenomic sequences.

**Rationale:** This analysis aims to detect potential phage signatures in the chimeric contigs. Since phages are mobile genetic elements, their involvement in transferring ARGs through transduction is highly relevant. Phages, especially temperate phages, can integrate into bacterial genomes and excise themselves, sometimes carrying host genetic material, such as ARGs, with them. The integration and excision signatures detected in contigs will provide evidence of possible transduction events in our datasets, supporting the hypothesis of ARG dissemination via phages.

### 3.1.15 Phage Signature Extraction and Phylogenetic Analysis

**Technical Notes:** Phage-associated genes will be extracted from the chimeric contigs, followed by phylogenetic analysis to uncover evolutionary relationships. FastTree will be used to build a phylogenetic tree based on the extracted phage genes. For visualization, tools like iTOL or FigTree can be used to generate an interpretable phylogenetic tree.

**Rationale:** Phage genes embedded in chimeric contigs (their taxonomy) may serve as strong evidence of horizontal gene transfer (HGT) events. The aim here is to check the evolutionary origins of the phage genes found in our dataset and their potential involvement in the dissemination of ARGs.

### 3.1.16 Directory tree

**Notes:** The final directory tree should look like this to help you visualize. **Notice the multiple path/to/ here as this is still (WIP):**

```

/project_root/
├── path/to/cleaned_reads/ ..... Prerequisite
├── path/to/merged_reads/
├── path/to/CARD_db/ ..... Prerequisite - Database
├── path/to/output/
│   ├── final_annotated_plasmids.fasta
│   ├── fpkm_normalized_plasmids.txt
│   ├── categorized_MGEs.txt
│   └── final_filtered_contigs.fasta
├── path/to/logs/
│   ├── predict_plasmids.log
│   ├── run_mob_suite.log
│   ├── check_plasmidfinder.log
│   └── calculate_gc_skew.log
├── path/to/plaspredict_output/
├── path/to/PFAM_db/ ..... Prerequisite - Database
├── path/to/TnpPred_db/ ..... Prerequisite - Database
├── path/to/plasmid_prediction/
│   ├── predicted_plasmids.fasta
│   ├── plasmidfinder_report.txt
│   ├── oritfinder_report.txt
│   └── tnpred_report.txt
├── path/to/plasmidspades_output/
│   └── plasmid_assembly.fasta
├── path/to/mob_suite_output/
│   └── mob_suite_summary.txt
├── path/to/gc_skew_analysis/
│   └── gc_skew_plot.png
└── path/to/read_mapping/

```

```
|_ reads_mapped_to_plasmids.bam  
|_ plasmid_read_count.txt  
|_ total_read_count.txt
```

200

201 **Homework?**202 **3.1.17 Future Considerations**

203 Will continue improving this section (everything regarding HGT) by evaluating the results from these tools,  
204 aligning them with ARG presence, and refining the approach for identifying conjugation, transposon, and  
205 transduction events within chimeric contigs. This may also involve validating phage activity—*again, why*  
206 *do I do this to myself?*

## 3.2 metagenomics\_general.smk

**Stage: Done**

**General Purpose** taxo-metagenomics

**Specifics:** This pipeline is designed for **The essentials in metagenomics**, which includes **quality checking, filtering, and trimming of raw reads to clean reads**. As well as the usual **taxo-metagenomic analysis**.

### 3.2.1 Quality control (Pre-processing)

**Raw trimming of raw metagenomic data**

**Technical Notes:** FastQC is used on **raw reads**

**Rationale:** This is mainly used as a point of comparison - determine whether the next step (trimming) was effective. This pre-processing step is the starting point in any and all metagenomics pipelines.

**Notes:** This whole quality control steps are interconnected with each other.

### 3.2.2 Trimming

**Technical Notes:** Trimmomatic is used on **raw reads**

**Rationale:** Trimming involves removing low quality bases (often depending on something called the Phred Score - which is just a measure of how "confident" we are that the base on that site was accurate), adapters, and filtering reads that go below a specific length threshold.

**Notes:** Journals often report the parameters on Trimmomatic (or any trimmer they decide to use); this is often so that the study is reproducible, should one decide to actually reproduce the study starting from scratch (raw reads).

**Notes:** There are many different trimmers each with their own strengths and weaknesses, Trimmomatic is just the most popular trimmer and is thus used here, though studies differ in the parameters they used for trimming - which often dictates how strict they are with what they define as "good enough".

**Perspective:** Why different people choose different trimmers depend on the **strengths and weaknesses** of the trimmer e.g. trimmers like "fastp" is used because it's fast making it suitable for very large datasets like deep sequencing. While some trimmers like Sickle has automatic adjustment over the entire sequence - which makes it useful for very ancient datasets where DNA is often highly-degraded. Other times, it's for **convenience** like Trim-Galore which combines FastQC and Cutadapt trimmer in a single command. Another good example is BBDuk which is part of a larger package called BBtools, BBDuk also has an built-in contamination detection - so it's particularly good at filtering out usual contaminants like sequences known to be from the human genome. so you can simply just use all the modules in that package for all-in-one processing. Other times, it's just **familiarity**.

**Quality checks of post-trimmed data**

**Technical Notes:** FastQC is used on **trimmed reads** to determine how effective the trimming process was.

**Rationale:** If the trimming process was effective, we should notice a better quality reads here, otherwise, we might have to adjust the trimming parameters.

**Notes:** Determination of whether the trimmed reads are "clean enough" is more of an art rather than

actual science, though thresholds exist like  $\text{Phred} > 20$  or  $\text{Phred} > 30$  depending on how strict you are as a researcher.

## Processing cleaned reads

---

### 3.2.3 Metagenomic Taxonomy

#### Taxonomy from Cleaned Reads

**Technical Notes:** Kraken2 is used on **raw reads** to determine which species they came from.

**Rationale:** Taxonomy based on reads is standard on metagenomics instead of using assembled contigs because information is lost during the assembly process. By using cleaned reads we make sure that:

1. We are maximizing the amount of data
2. We are not being biased by low quality reads

**Notes:** There are many ways in which taxonomy is assigned to raw reads the Kraken-based packages use a curated database that links k-mers from your database to k-mers generated from their database (usually manually curated).

**Notes:** Others like MetaPhlan first create "markers" that are based off of known sequences, and then scan your raw reads for these markers, which it then checks for under what taxon/taxa that marker fell under.

### 3.2.4 Diversity analysis

#### Diversity analysis per site or sample

**Technical Notes:** Here Bracken - an extension of the Kraken packages or Qiime2 are often used to calculate diversity. Instead I opted to create my own Python script `calculate_diversity.py` because:

1. I find that the diversity indices in either tools are limited to only the most often used i.e. the most popular indices - so it is not comprehensive
2. I've had problems integrating them into the pipeline because some dependency limitations, un-updated scripts, and a pre-processing step that requires converting all of Kraken2, Bracken files, then importing them to Qiime2 which is too time-consuming and inefficient - my script just automatically calculates from Bracken outputs and just puts out all the possible (non-phylogenetic-based-which you would need a phylogenetic tree to build first) indices out there.

**Rationale:** Taxonomy based on reads is standard on metagenomics instead of using assembled contigs because information is lost during the assembly process. By using cleaned reads we make sure that:

1. We are maximizing the amount of data
2. We are not being biased by low quality reads

75 **Notes:** There are many ways in which taxonomy is assigned to raw reads the Kraken-based packages  
 76 use a curated database that links k-mers from your database to k-mers generated from their database  
 77 (usually manually curated).

78 **Notes:** Others like MetaPhlan4 first create "markers" that are based off of known sequences, and then  
 79 scan your raw reads for these markers, which it then checks for under what taxon/taxa that marker fell  
 80 under.

### 81 3.2.5 Directory tree

```

/project_root/
├── configs/
│   └── config.yaml
├── path/to/raw_reads_dir/ ..... Prerequisite - Raw Data
│   ├── sample1_R1.fastq.gz
│   ├── sample1_R2.fastq.gz
│   ├── sample2_R1.fastq.gz
│   └── sample2_R2.fastq.gz
├── path/to/trimmed_reads_dir/ ..... Prerequisite - Trimmed Data
│   ├── sample1_R1_paired.fastq.gz
│   ├── sample1_R2_paired.fastq.gz
│   ├── sample2_R1_paired.fastq.gz
│   └── sample2_R2_paired.fastq.gz
├── path/to/fastqc_output_dir/
├── path/to/kraken_output_dir/
│   ├── sample1.k2report
│   ├── sample1.kraken2
│   ├── sample2.k2report
│   └── sample2.kraken2
├── path/to/bracken_output_dir/
│   ├── sample1.bracken
│   ├── sample1.breport
│   ├── sample2.bracken
│   └── sample2.breport
├── path/to/cleaning_results_dir/
│   └── summary_report.txt
├── logs/
│   └── calculate_diversity.log
└── path/to/output/
    └── diversity_matrices.tsv
  
```



### 3.3 trim\_randomizer.smk

**Stage: Done**

**Purpose: Randomization of trimming parameters**

This is a module that will be part of a bigger pipeline. The idea here is to randomize parameters in a variety of trimmers in this case Trimmomatic, fastp, CutAdapt, BBDuk, and Sickle - famous bioinformatics trimming tools. Trimmomatic and Sickle in particular are widely used in Illumina-based data.

#### 3.3.1 Random Parameter Generation

**Technical Notes:** Random parameters are generated for each trimming tool (Trimmomatic, Fastp, Cutadapt, BBDuk, Sickle). This is done using the `random` module, which creates random values for parameters such as quality scores, read length, adapter sequences, and error rates. These parameters are stored in the `generated_parameters` dictionary to ensure consistency across iterations for each sample.

**Rationale:** By randomizing parameters, the workflow allows for testing different parameter sets across multiple iterations to find optimal settings for trimming and quality control.

**Notes:** This approach helps with parameter exploration, particularly when you are unsure which trimming settings will give the best results. The randomness provides variability, which can highlight which parameters consistently lead to good results.

#### 3.3.2 Log Parameters to a TSV File

**Technical Notes:** Each set of generated parameters is logged into a separate TSV file (e.g., `trimmomatic_params.tsv`, `fastp_params.tsv`). The file headers are written only once, and parameters are appended as the trimming steps proceed. This is done in a structured way so that you can track the exact parameters used for each sample and iteration.

**Rationale:** Logging ensures reproducibility and transparency in bioinformatics workflows. Having a record of all the parameter values used in each iteration is crucial for comparing results and for future reference.

**Notes:** This practice is a standard in scientific workflows where random parameter generation is involved. It helps maintain a clear audit trail of the steps performed and aids in troubleshooting or refining workflows later.

#### 3.3.3 Define Rules for Each Tool

**Technical Notes:** The script uses Snakemake rules to define separate rules for each tool which include

1. Input folder (as the script is designed to go through all the FASTQ samples within the folder)
2. Output folder, where the processed files will be saved (e.g., trimmed paired and unpaired reads).  
The script is designed to keep all the `trimmed` reads in separate files.
3. Params: Fetches the parameters to be randomized.

**Rationale:** Using Snakemake here allows for parallel execution of the workflow. This parallelization is very important as the generation of random parameters is created using a random seed. Using a random seed like this allows us to replicate what the parameters that had the optimal results were by tracking down what seed was assigned as dictated in the TSV file.

**Note** keep in mind that this will create a large number of folders if you decide to iterate many times - as each iteration, per tool, will have its own folder full of trimmed reads, per sample/site.

### 3.3.4 Interpretation and Analysis of Results (TBA)

**Technical Notes:** Once all iterations have completed, the trimmed files can be analyzed to determine which parameters led to the best results in terms of quality and length distribution of reads. There are many different metrics that can be used to interpret the results, including (but not limited to)

1. Quality Scores - significant differences in quality among sites and across entire sequences (Phred score, Contamination, Adapter Removal, N Content, Length Distribution, etc.). Can be done using tools like FastQC
2. Visualization can be done using Rstudio or Python's matplotlib to visually look for differences in abnormalities.

**Rationale:** Evaluating read quality and assessing key metrics post-trimming helps to ensure that the data is suitable for downstream analyses. Optimal trimming should maximize the number of high-quality, usable reads while eliminating low-quality bases and adapter contamination.

**Notes** See Box on Biological Information for more possible details on this.

### 3.3.5 Annotated Directory Tree with File Temporary files and Prerequisites

```

/project_root/
├── raw_reads/ ..... (permanent) (prerequisite)
│   ├── sample1_R1.fastq.gz ..... (permanent) (prerequisite)
│   ├── sample1_R2.fastq.gz ..... (permanent) (prerequisite)
│   ├── sample2_R1.fastq.gz ..... (permanent) (prerequisite)
│   └── sample2_R2.fastq.gz ..... (permanent) (prerequisite)
├── output_dir/
│   ├── fastp_output/ ..... (temporary) (to be deleted)
│   │   ├── iteration_1/
│   │   │   ├── sample1_R1_fastp_trimmed.fastq.gz ..... (temporary)
│   │   │   ├── sample1_R2_fastp_trimmed.fastq.gz ..... (temporary)
│   │   │   ├── sample2_R1_fastp_trimmed.fastq.gz ..... (temporary)
│   │   │   └── sample2_R2_fastp_trimmed.fastq.gz ..... (temporary)
│   │   ├── iteration_2/
│   │   │   └── (same as iteration_1 but with iteration_2 files) ..... (temporary)
│   │   └── iteration_3/
│   │       └── (same as iteration_1 but with iteration_3 files) ..... (temporary)
│   └── trimmomatic_output/ ..... (temporary) (to be deleted)

```

```

├─ iteration_1/
│   ├── sample1_R1_paired.fastq.gz .....(temporary)
│   ├── sample1_R1_unpaired.fastq.gz ..... (temporary)
│   ├── sample1_R2_paired.fastq.gz .....(temporary)
│   ├── sample1_R2_unpaired.fastq.gz ..... (temporary)
│   └─ (same for sample2) .....(temporary)
├─ iteration_2/
│   └─ (same structure as iteration_1) .....(temporary)
├─ iteration_3/
│   └─ (same structure as iteration_1) .....(temporary)
├─ cutadapt_output/ .....(temporary) (to be deleted)
│   ├── iteration_1/
│   │   ├── sample1_R1_cutadapt_trimmed.fastq.gz ..... (temporary)
│   │   └─ sample1_R2_cutadapt_trimmed.fastq.gz ..... (temporary)
│   ├── iteration_2/
│   │   └─ (same structure as iteration_1) .....(temporary)
│   └─ iteration_3/
│       └─ (same structure as iteration_1) .....(temporary)
├─ bbduk_output/ .....(temporary) (to be deleted)
│   ├── iteration_1/
│   │   ├── sample1_R1_bbdduk_trimmed.fastq.gz .....(temporary)
│   │   └─ sample1_R2_bbdduk_trimmed.fastq.gz .....(temporary)
│   ├── iteration_2/
│   │   └─ (same structure as iteration_1) .....(temporary)
│   └─ iteration_3/
│       └─ (same structure as iteration_1) .....(temporary)
├─ sickle_output/ .....(temporary) (to be deleted)
│   ├── iteration_1/
│   │   ├── sample1_R1_sickle_trimmed.fastq.gz .....(temporary)
│   │   ├── sample1_R2_sickle_trimmed.fastq.gz .....(temporary)
│   │   └─ sample1_singles_sickle_trimmed.fastq.gz .....(temporary)
│   ├── iteration_2/
│   │   └─ (same structure as iteration_1) .....(temporary)
│   └─ iteration_3/
│       └─ (same structure as iteration_1) .....(temporary)
├─ logs/ ..... (permanent) (prerequisite)
├─ fastp_params.tsv ..... (permanent) (prerequisite)
├─ trimmomatic_params.tsv ..... (permanent) (prerequisite)
├─ cutadapt_params.tsv ..... (permanent) (prerequisite)
├─ bbduk_params.tsv ..... (permanent) (prerequisite)
└─ sickle_params.tsv ..... (permanent) (prerequisite)

```

## 3.4 bootstrapping\_rawreads.smk

**Stage:** Further refinement

**Purpose:** Bootstrapping the taxo\_metagenomic pipeline itself

Bootstrapping is the process of randomly selecting from a pool of samples (with replacement) and using that in a specific process you want to bootstrap. This is a standard method used in molecular phylogenetics to determine the robustness of trees where a  $> 70$  support from bootstrapped data is considered robust enough.

### The principle of bootstrapping in phylogenetics

Phylogenetics uses this statistical technique because (in principle) it effectively means that removing parts of the entire sequence does not alter the topology of the tree.

Here I'm adapting this method with Kraken2's taxonomic profiling to see whether the taxonomic support of its k-mer assignment is also consistent even with changes in the sampling sites.

This is still WIP because I plan to go step further and start bootstrapping the raw reads themselves to see if changes in reads changes the topology of taxonomic assignment.

### 3.4.1 Directory setup of temporary files

**Technical Notes:** Snakemake starts by ensuring the existence of necessary directories. Most notably, the temporary bootstrap directories.

**Rationale:** Bootstrapping is sometimes done iteratively thousands of times, so making this a temporary directory helps manage space.

**Notes:**

### 3.4.2 Sample Identification

**Technical Notes:** The workflow identifies sample names by parsing filenames in the raw reads directory.

**Rationale:** Inclusion of these in the script allows the user to flexibly configure the naming convention and the directory in which they want to bootstrap.

**Notes:** Presently it looks for `_R1.fastq.gz` and its associated pair, `_R2.fastq.gz` in the `raw_reads` directory.

### 3.4.3 Bootstrapping

**Technical Notes:** This executes `bootstrap_reads.py` in the `scripts` directory to start bootstrapping the paired-end reads and outputs them in the temporary folders I mentioned earlier.

**Rationale:** Moving bootstrapping logic into a Python script leverages its ability to create randomizations from its libraries. Also it allows us to define a `seed` so it is reproducible (if you want to reproduce) the results anyway.

**Note:** The number of bootstraps and the fraction of samples you want to retain can be controlled in the `config.yaml` file in the configuration folder.

**Update Sep 19 2024**

**On a personal note:** Before writing this part of the document, I decided to change the bootstrapping rule. It used to rely on a simplified approximation. The probability of not being selected after  $N$  independent draws from a sample of size  $N$  is given by:

$$P = \left(1 - \frac{1}{N}\right)^N$$

This is a well-known mathematical equation describing the probability of not selecting a sample at least once in  $N$  draws. As  $N \rightarrow \infty$ , this probability approaches:

$$\frac{1}{e}$$

Previously, I used the probability of a sample *being selected*, which is:

$$1 - \frac{1}{e}$$

This was used as an approximation for bootstrapping by shuffling and adjusting the sample size. However, the updated script now performs **actual bootstrapping**, sampling with replacement, which is a more accurate statistical method for resampling.

29

30 **3.4.4 Annotated Directory Tree with File Movement and Prerequisites**

```

/project_root/
├── configs/
│   └── config.yaml .....(prerequisite)
├── path/to/raw_reads_dir/ .....(permanent) (prerequisite)
│   ├── sample1_R1.fastq.gz .....(permanent) (prerequisite)
│   ├── sample1_R2.fastq.gz .....(permanent) (prerequisite)
│   ├── sample2_R1.fastq.gz .....(permanent) (prerequisite)
│   └── sample2_R2.fastq.gz .....(permanent) (prerequisite)
├── path/to/temp_bootstrap_dir/ .....(temporary) (to be deleted after pipeline resolves)
│   ├── sample1/ ..... (temporary)
│   │   ├── rep_1_R1.fastq.gz .....(temporary)
│   │   ├── rep_1_R2.fastq.gz .....(temporary)
│   │   ├── rep_2_R1.fastq.gz .....(temporary)
│   │   ├── rep_2_R2.fastq.gz .....(temporary)
│   │   └── total_reads.txt .....(temporary)
│   └── sample2/ ..... (temporary)
│       ├── rep_1_R1.fastq.gz .....(temporary)
│       ├── rep_1_R2.fastq.gz .....(temporary)
│       ├── rep_2_R1.fastq.gz .....(temporary)
│       ├── rep_2_R2.fastq.gz .....(temporary)
│       └── total_reads.txt .....(temporary)

```

```

├── rep_1/ ..... (temporary)
│   ├── diversity_matrices_sample1_rep_1.tsv .(temporary) (moved to diversity_results/)
│   └── diversity_matrices_sample2_rep_1.tsv .(temporary) (moved to diversity_results/)
├── rep_2/ ..... (temporary)
│   ├── diversity_matrices_sample1_rep_2.tsv .(temporary) (moved to diversity_results/)
│   └── diversity_matrices_sample2_rep_2.tsv .(temporary) (moved to diversity_results/)
├── logs/ ..... (permanent) (prerequisite)
│   ├── metagenomics_pipeline_sample1_rep_1.log ..... (permanent) (prerequisite)
│   ├── metagenomics_pipeline_sample1_rep_2.log ..... (permanent) (prerequisite)
│   ├── metagenomics_pipeline_sample2_rep_1.log ..... (permanent) (prerequisite)
│   └── metagenomics_pipeline_sample2_rep_2.log ..... (permanent) (prerequisite)
├── diversity_results/ ..... (permanent) (contains moved files) (prerequisite)
│   ├── diversity_rep_1.tsv ..... (moved from temp_bootstrap_dir)
│   └── diversity_rep_2.tsv ..... (moved from temp_bootstrap_dir)
└── aggregated_results/ ..... (permanent) (prerequisite)
    └── diversity_aggregate.tsv ..... (permanent)

```

### 3.4.5 Run kraken\_pipeline.bash

**Technical Notes:** This Shellsript (or bash file) is used to automate the processing of all files from the bootstrapping. It runs them under Kraken2 then Bracken to generate taxonomy profiles for all of them. It also creates a log file for each replicate to provide traceability and error checking, helping diagnose any issues with specific replicates or samples.

**Rationale:** Read why I'm bootstrapping from the textbox above. The reason why I also included Bracken and measurements diversity metrics in the analysis per sampling replicate is so we can analyze how the topology of diversity also changes - similiar to how we look at topology of phylogenetic trees. The final process consolidates all the diversity metrics (alpha diversity) and matrices (beta diversity) into a TSV file.

**Note:** The decision to use Snakemake and Shellscripts here is so that the bootstrapping comes first before the pipeline is introduced. Otherwise Snakemake will run the entire thing in parallel, taking up so much memory because it runs Kraken2 for every single sample instead of doing it in batches - which takes up so much unnecessary time.

## 3.5 Binning.smk

**Stage:** To test on higher coverage data

**Binning pipeline to create high quality (Metagenome Assembled Genomes) MAGs**

**Specifics:** This pipeline passes through multiple quality checks during binning of using a variety of tools (both wrappers and modules) including MetaWrap, DasTool, CheckM2, MagPurify etc.

### 3.5.1 Universal Configurations

**Technical Notes:** The pipeline allows the user to configure settings they want for the binning process. By default, the settings are Minimum Contig Length (2500bp), Completeness (50%), Contamination (10%).

**Rationale:** The default settings were curated by me and the reason I chose them is because of the following

1. Longer contigs tend to represent more complete genomic fragments. Setting a threshold of 2500bp ensures better assembly quality. Others prefer a lower threshold for more sensitivity like 2000 bp.
2. Completeness 50% and Contamination 10% are actually based from a standard called the MIMAG standards.

**Notes:** Making configurations universal this way creates consistency across the script, i.e. when specifically asked by a specific tool, this returns a universal parameter.

**Notes:** Additionally, the user can specify the memory usage and number of threads they want to allocate per tool as well as other tool-specific parameters at the top of the script for ease of use. *I plan to add this to the configuration file soon once I have tested the file to be working at higher coverage - since you can't make high quality bins with low read counts - and my PC can't practically handle that sorry.*

### 3.5.2 FastUniq (Deduplication)

**Technical Notes:** FastUniq used to remove duplicate reads.

**Rationale:** Since we are focused on creating MAGs or genomes based on populations of genomes, removing duplicates is less risky during binning and is thus included. It also allows us to completely remove amplification bias from PCR reactions. Moreover, deduplication reduces redundancy and thereby memory usage downstream.

### 3.5.3 Seqtk (FASTA Conversion)

**Technical Notes:** Seqtk converts FASTQ to FASTA.

**Rationale:** This conversion prepares sequences for tools that require FASTA inputs, such as CD-HIT-EST.

**Notes:** I used to include a decompress-then-compress mechanism in the script to minimize memory storage but according to my calculations from the sequencing facility quotations, re-compressing may actually be more costly when done throughout the pipeline. Hence, it should be more cost-efficient to start compressing files once the bins are done.

### 3.5.4 CD-HIT-EST (Clustering) at Identity: 90%

**Technical Notes:** CD-HIT-EST clusters similar sequences at 90% identity.

**Rationale:** Clustering reduces redundancy in the contig data while maintaining closely related sequences.

**Notes:** I chose 90% ID because that's what I often see in published journals that is all. Perhaps, the 90% identity threshold balances removing duplicates while preserving diversity and that higher thresholds would result in fewer clusters but might oversimplify the data.

*Fine, I'll make it my homework assignment why this specific threshold is used (TBA).*

### Bin Refinement

---

### 3.5.5 MetaWRAP Binning and Reassembly

**Technical Notes:** MetaWRAP is what is known as a wrapper program - meaning it makes use of other tools as its modules. For the binning process in particular it uses 3: MetaBAT2, MaxBin2, and CONCOCT. Each binning process goes through internal quality control checks and the one with the best bin-qualities are selected. It also has a reassembly feature wherein it reassembles the contigs again to try and further refine the bins.

**Rationale:** Using 3 bidders in parallel and choosing the best bins, quality checking, and then reassembling (not-so-good) bins make the binning process very robust, creating very refined bins *not sponsored by the way, talking as a fellow researcher.*

**Notes:** No moving forward, the process of further refinement of bins seems redundant. But do note that I have checked and validated that the process used in refining and checking by the tools used here cover different metrics - and therefore can be seen as parallel processes.

### 3.5.6 DAS Tool (Bin Refinement)

**Technical Notes:** DAS\_Tool is very similar to MetaWrap in that they both choose the best bins from a pool of bins from different bidders (in this pipeline DAS Tool is designed to **ALSO** use information from MetaBAT2, MaxBin2, and CONCOCT outputs to improve binning)

**Rationale:** However, as rationale for including it, is that it focuses more on single-copy gene (SCG) analysis. In contrast, in MetaWrap, bins are evaluated using completeness and contamination thresholds.

**Notes:** Notably, in the checking phase of this pipeline we will not be using DAS\_Tool for SCG analysis. It is optimized for refining bins not quality checking bins. Instead we will use a more updated and optimized software for the latter called BUSCO.

### 3.5.7 MAGpurify (Contamination Removal)

**Technical Notes:** MAGpurify is also a bin refiner, in a sense that it uses several modules to detect and **prune** contamination in genome bins. It also uses other metrics to define bin quality specifically it looks for differences in

#### 1. Phylo-markers,



72 2. **Clade-markers**,

73 3. **Tetranucleotide-frequencies**,

74 4. **GC-content**,

75 5. and then removes **known-contaminants** from it's manually curated database (created back in  
76 2013)

77 **Rationale:** This step improves genome quality by removing low-confidence contigs or contamination  
78 from other taxa. Each module targets different contamination types (phylogenetic, clade, etc.).

79 **Notes:** Similar to DAS\_tool, MAGPurify is relatively old (in the bioinformatics world where new tools  
80 are being published every day). So in checking our bins we will be using more recently updated tools.

## 83 Quality checking

---

### 85 3.5.8 MetaQUAST (Assembly Quality Assessment)

86 **Technical Notes:** MetaQUAST assesses the quality of genome assemblies via the following:

- 87 1. N50 and L50 to determine contiguity
- 88 2. Number of contigs to determine fragmentation
- 89 3. GC content - since a single MAG should have a constant GC content across its entirety (usually)
- 90 4. Alignment to a reference sequence
- 91 **Additionally, it also detects** other metrics using modular tools
- 92 5. structural variations (requires GRIDSS)
- 93 6. presence of rRNA (requires SILVA)
- 94 7. Conserved gene sets (requires BUSCO)

95 **Rationale:** This step quantifies the completeness and accuracy of the assembled genomes, and is updated  
96 frequently.

97 **Notes:** Using reference genomes improves the accuracy of the assessment, but it's optional if references  
98 are unavailable.

#### A Personal Note

**Personal Note:** As of this writing, BUSCO has updated beyond the version required by QUAST (BUSCO od9). Unfortunately, this version is not available in the archives (you'll encounter a 404 error). Likewise, SILVA and GRIDSS frequently update. I recommend downloading each separately and manually linking their databases to avoid potential issues with QUAST's download management. *Good luck and have fun!*

### 3.5.9 dRep (Dereplication)

**Technical Notes:** dRep dereplicates genomes by clustering them based on similarity.

**Rationale:** Dereplication reduces redundancy in the assembled genome data, ensuring unique genome representations. Basically CD-HIT but for whole genomes.

**Notes:** FastANI is utilized for quick, precise clustering, and is part of the dRep package. It defaults to a 95% ANI (Average Nucleotide Identity) threshold, a common yet somewhat subjective metric used to determine microbial species boundaries. This is often sufficient for microbial genomes due to their high gene density, frequently organized in operons. However, it may not be as suitable for eukaryotic genomes, which are laden with repetitive elements.

#### A Personal Note

**Personal Note:** I cannot find a newer version of either dRep or FastANI (both dating back to 2013, basically ancient in bioinformatics terms). There's a newer version called `pyani`, which is available in Bioconda, that might be a good replacement. However, I still need to reverse-engineer its source code to fully understand how it operates. Might be worth trying!

### 3.5.10 CheckM2

**Technical Notes:** CheckM2 is used to predict genome completeness and contamination using low-memory mode (essential for resource-limited systems like mine; remember to adjust this on HPC systems).

**Rationale:** CheckM2 is an updated version of CheckM, but many tools in this pipeline haven't been updated to recognize it. I haven't tested whether aliasing CheckM2 as CheckM would work, so it's added here as a final step to ensure the results meet MIMAG standards for completeness and contamination.

### 3.5.11 Annotated Directory Tree with File Movements and Temporary Files

```

/project_root/
├── trimmed_reads/ .....(prerequisite)
│   ├── site1_R1_paired.fastq.gz .....(permanent)
│   ├── site1_R2_paired.fastq.gz .....(permanent)
│   ├── site2_R1_paired.fastq.gz .....(permanent)
│   └── site2_R2_paired.fastq.gz .....(permanent)
├── fastuniq/ ..... (temporary) (to be deleted)
│   ├── site1_R1_uniq.fastq .....(temporary)
│   ├── site1_R2_uniq.fastq .....(temporary)
│   ├── site2_R1_uniq.fastq .....(temporary)
│   └── site2_R2_uniq.fastq .....(temporary)
├── fasta/ ..... (permanent)
│   ├── site1_R1_clean.fasta ..... (permanent)
│   ├── site1_R2_clean.fasta ..... (permanent)
│   ├── site2_R1_clean.fasta ..... (permanent)
│   └── site2_R2_clean.fasta ..... (permanent)
└── megahit_output/ .....(permanent)

```

```

├─ site1_assembly/
│   └─ site1_filtered_contigs.fa ..... (permanent)
├─ site2_assembly/
│   └─ site2_filtered_contigs.fa ..... (permanent)
├─ cdhit_contigs/ ..... (permanent)
│   └─ site1_cdhit_contigs.fasta ..... (permanent)
│   └─ site2_cdhit_contigs.fasta ..... (permanent)
├─ CLEAN_READS/ ..... (permanent) (moved)
│   └─ site1/
│       └─ site1_1.fastq ..... (moved from fastuniqu)
│       └─ site1_2.fastq ..... (moved from fastuniqu)
│   └─ site2/
│       └─ site2_1.fastq ..... (moved from fastuniqu)
│       └─ site2_2.fastq ..... (moved from fastuniqu)
├─ binning_output/ ..... (permanent)
│   └─ site1_binning_output/
│       └─ concoct_bins.scaffolds2bin.tsv ..... (permanent)
│       └─ metabat2_bins.scaffolds2bin.tsv ..... (permanent)
│       └─ maxbin2_bins.scaffolds2bin.tsv ..... (permanent)
│   └─ site2_binning_output/
│       └─ (same as site1) ..... (permanent)
├─ dastool_output/ ..... (permanent)
│   └─ DAS_Tool_bins_site1 ..... (permanent)
│   └─ DAS_Tool_bins_site2 ..... (permanent)
├─ reassembly_output/ ..... (permanent)
│   └─ site1_reassembly_output/
│       └─ assembly.fasta ..... (permanent)
│   └─ site2_reassembly_output/
│       └─ assembly.fasta ..... (permanent)
├─ magpurify_output/ ..... (permanent)
│   └─ site1_cleaned.fasta ..... (permanent)
│   └─ site2_cleaned.fasta ..... (permanent)
├─ metaquast_output/ ..... (permanent)
│   └─ site1_metaquast_output/
│       └─ (MetaQUAST results) ..... (permanent)
│   └─ site2_metaquast_output/
│       └─ (MetaQUAST results) ..... (permanent)
├─ dereplication_output/ ..... (permanent)
│   └─ dereplicated_genomes/
│       └─ dereplicated_genomes.fasta ..... (permanent)
│       └─ checkm2_output/ ..... (permanent)
│           └─ quality_report.tsv ..... (permanent)

```

```

├─ logs/ .....(permanent)
│   └─ run_megahit_site1.log ..... (permanent)
│   └─ run_cdhit_site1.log ..... (permanent)
│   └─ (logs for each step) .....(permanent)

```

## 3.6 krakenpipeline.bash

**Stage: Done**

**The entire basic taxo-metagenomic pipeline with the usual tools**

**Specifics:** This pipeline passes through a loop between FastQC and Trimmomatic trimming-cleaning-checking.py before progressing to Kraken2 then Bracken then calculate\_diversity.py.

**Notes:** This does not include FastQC-ing of raw reads (yet), you have to run raw.bash for that. Additionally, it also currently does not automate the production of aggregated summaries of the FastQC files for you (yet), you have to run summary\_stat.bash for that as well.

### 3.6.1 Directory tree

```

/project_root/
├─ trimmed_reads/ .....Prerequisite - Trimmed reads directory
│   └─ sample1_R1_paired.fastq.gz
│   └─ sample1_R2_paired.fastq.gz
│   └─ sample2_R1_paired.fastq.gz
│   └─ sample2_R2_paired.fastq.gz
├─ kraken_db/ .....Prerequisite - Kraken2 database directory
│   └─ database files (.k2d, etc.)
├─ kraken_output/ ..... Output directory for Kraken2 reports
│   └─ sample1.k2report
│   └─ sample1.kraken2
│   └─ sample2.k2report
│   └─ sample2.kraken2
├─ bracken_output/ ..... Output directory for Bracken reports
│   └─ sample1.bracken
│   └─ sample1.breport
│   └─ sample2.bracken
│   └─ sample2.breport

```

## 3.7 raw.bash; completely optional

**Stage: Done**

**FastQC Automation script to run FastQC on raw reads**

**Specifics:** This script takes raw reads and creates a summary report for each of them in FasQC\_raw/

**Notes:** summary\_stat.bash can already do that for you, so it's a bit redundant. **But if you're only interested in seeing QC from the raw reads, then have fun!**

7 **Personal notes:** *The only reason I am keeping this alive is because I am not quite sure yet if some*  
 8 *pipelines rely on this and it's corresponding output files; I have to check.*

## 1 3.8 summary\_stat.bash

### 2 Stage: Done

#### 3 FastQC Automation script

4 **Specifics:** This script takes reads from the raw reads directory and the trimmed reads directory, then  
 5 creates a summary directory (if not already existent) (you can manually set this as user). The summary  
 6 directory will contain summary statistics extracted from FastQC sub-directories, it also creates. It has  
 7 two notable functions:

- 8 1. Counts the number of reads in raw and trimmed directories.
- 9 2. Extract quality metrics from FastQC subdirectories specifically: per base sequence quality,  
 10 and per sequence quality scores.

11 It then saves them into a file the summary subfolder.

12 **Notes:** The way it counts reads from FASTQ file is by reading the number of lines and dividing it by 4  
 13 (since each read in a FASTQ file consists of 4 lines: sequence identifier, sequence, plus sign, and quality  
 14 score).

15 **Notes:** This does not include config file integration yet (TBA).

16 **Notes:** Not integrated into any pipeline yet (TBA).

### 17 3.8.1 Directory tree

```

/project_root/
├── raw_reads/ .....(Prerequisite - Raw Data)
│   ├── sample1.fastq.gz
│   ├── sample2.fastq.gz
│   └── ...
├── trimmed_reads/ .....(Prerequisite - Trimmed Data)
│   ├── sample1_trimmed.fastq.gz
│   ├── sample2_trimmed.fastq.gz
│   └── ...
├── summary_statistics/
│   ├── fastqc_raw/
│   │   ├── sample1_fastqc.zip
│   │   ├── sample2_fastqc.zip
│   │   └── ...
│   ├── fastqc_trimmed/
│   │   ├── sample1_trimmed_fastqc.zip
│   │   ├── sample2_trimmed_fastqc.zip
│   │   └── ...
│   └── extracted_metrics/
  
```

```
├── sample1_base_quality.txt
├── sample1_sequence_quality.txt
├── ...
├── raw_read_counts.txt
└── trimmed_read_counts.txt
```

### 3.9 plot\_and\_detect\_intermediate\_coverage.py

#### Stage: Done

#### Processing of coverage data

**Specifics:** This script takes the coverage data from a coverage file, which is expected to have 3 columns: chrom(chromosome/contig name), pos(position along the contig), cov(coverage value at that specific position). It then takes the latter two columns and uses matplotlib to create a line plot of the coverage values - and saves it as an image.

**Notes:** Presently, it determines, intermediate coverage by looking first looking for the minimum and maximum value of coverage data.

#### Sept 20 2024 Update

I updated this one here to be a bit more robust in detection of chimeric contigs. Instead of visual inspection (which is a bit too subjective for my taste), I added 4 new features to this

1. Sliding window approach to detect changes coverage (rather than visualization alone) other than that we apply sliding window to detect changes in:
  - (a) Codon usage bias
  - (b) GC content
  - (c) Tetranucleotide frequencies
2. Each of these "windows" are then subjected to ANOVA, to determine statistical significance and
3. PCA, that combines all the (3) features to generate a 2D plot which to see whether different sections of the contig cluster differently - which would indicate likely chimerism.

### 3.10 Shortbred\_ARG.bash

#### Stage: Back to drawing board and to test

**Specifics:** Marks trimmed reads with ARGs **Technical Details** This script takes cleaned reads (dictated by the user), unzips them (to FASTQ), then starts finding ARG-markers on them. Markers are determined by the function shortbred\_identify.py (intrinsic to the tool), taking first the CARD database and aligning it with the RefSeq. After that, it clusters the RefSeq database at 95% similarity with those on the CARD database, to create markers. After that, it uses another function shortbred\_quantify.py

**Notes:** I included an additional step here that filters out specific keywords from the FASTQ files from a reference presently. It called `filter_keywords.txt` which you can make yourself should you want to filter out specific ARGs that you consider "low-confidence", or just create the file and leave it blank (up to you).

### Sept 20 2024 Update

I am revisiting this process because:

- It turns out that ShortBRED is typically used on **contigs**, not raw reads. This makes sense because marker-based analyses are more accurate on contigs, as they reduce false positives and provide a clearer view of the genetic background.

**Note:** ShortBRED is used to quantify **genes**, and raw reads are fragmented sequences it is the **CONTIGS THAT CONTAIN GENES**.

- The current method of filtering low-confidence ARGs is inefficient: it creates a temporary file full of ARGs with the keywords, and then filters the marker database using those. There must be a more efficient way to skip this unnecessary step of generating temporary FASTQ files.

I have also streamlined the script to:

- Unzip FASTQ files in parallel.
- Remove the temporary file afterwards.
- Focus only on high-confidence ARGs. Previously, it also produced markers on the RefSeq database, but checking marker percentages turned out to be inefficient and pointless.
- Add checks when directory creation or unzipping fails.
- Use a consistent naming scheme.
- and finally, it now **includes MEGAHIT assembly** and a filtering step that removes contigs below 200 bp before ShortBRED processing.

The most important addition is the conversion from FASTQ to FASTA using `seqtk`. I had overlooked this step, as some tools cannot process FASTQ format.

**Personal Note:** Dear Reader, please be aware that ShortBRED is quite particular about FASTA headers. It requires the headers to be numeric identifiers, which adds an extra step of renaming all sequences to numbers. You'll also need to create an index to trace the original sequence headers back to their numeric counterparts.

While troubleshooting, I encountered persistent naming errors that were confusing at first. After diving into the source code, I eventually discovered that ShortBRED requires headers to be strictly numeric. This unexpected requirement added to the complexity of the process and was not immediately obvious.

### 3.11 refseq\_bacteria.bash

**Stage: Done**

**FastQC Automation script to run download from the bacteria RefSeq**

**Specifics:** This only downloads the ones that end in .faa as opposed to .fna - the latter are genomes, the former are just all the annotated proteins themselves, with a genome or not.

**Notes:** This script is kept here because it was difficult to find the link to the FTP site, because NCBI seemed to have had a recent restructuring.

#### Sept 20 2024 Update

Removed the hard-cap of 1-511.faa files, to add flexibility; now downloads without the need for the user to check how many to download. Also now uses aria2c, which has and intrinsic redownload unfinished files instead of wget function.

### 3.12 trimmomatic.bash; pipeline module

**Stage: Done**

**Trimmomatic automation script to run FastQC on reads**

**Specifics:** This script takes input reads (dictated by the pipeline) and trims them.

**Note:** This also contains the Trimmomatic parameters that I decided to use as placeholder at the beginning of this whole project.

**Note:** Will likely become deprecated soon (or simply altered) depending on the results of my optimization tests on trimmers (TBA). Code snippet is below.

```
TRIMMOMATIC_ADAPTERS="NexteraPE-PE.fa"
```

```
TRIMMOMATIC_SETTINGS="LEADING:10 TRAILING:10 SLIDINGWINDOW:4:20 MINLEN:60"
```

```
.
```

```
.
```

```
trimmomatic PE -phred33
```

### 3.13 renamingSIMS.bash

**Stage: Done**

**renaming output files from raw-reads simulators i.e. CAMISIM**

**Specifics:** This script takes the output folder of CAMISIM called the out directory and automates renaming them, and moving them.

**Note:** Directory structure is below to show you why automation is necessary and manually renaming is too time-consuming.



### 9 3.13.1 Directory Tree with Notations

```

CAMISIM/
├── out/
│   ├── sample1/ .....Renamed from "2024.09.05_11.25.28_sample_0" to "sample1"
│   │   ├── bam_1/ ..... Renamed from "bam"
│   │   ├── contigs_1/ ..... Renamed from "contigs"
│   │   ├── reads_1/ ..... Renamed from "reads"
│   │   │   ├── anonymous_reads.fq.gz ..... Original interleaved FASTQ file
│   │   │   ├── R1.fastq.gz ..... Created by seqtk
│   │   │   └── R2.fastq.gz ..... Created by seqtk
│   │   └── sample2/ .....Renamed from "2024.09.05_11.25.28_sample_1" to "sample2"
│   │       ├── bam_2/ ..... Renamed from "bam"
│   │       ├── contigs_2/ ..... Renamed from "contigs"
│   │       ├── reads_2/ ..... Renamed from "reads"
│   │       │   ├── anonymous_reads.fq.gz ..... Original interleaved FASTQ file
│   │       │   ├── R1.fastq.gz ..... Created by seqtk
│   │       │   └── R2.fastq.gz ..... Created by seqtk
│   └── raw_reads/
│       ├── sample1_R1.fastq.gz ..... Copied from "CAMISIM/out/sample1/reads_1/R1.fastq.gz"
│       ├── sample1_R2.fastq.gz ..... Copied from "CAMISIM/out/sample1/reads_1/R2.fastq.gz"
│       ├── sample2_R1.fastq.gz ..... Copied from "CAMISIM/out/sample2/reads_2/R1.fastq.gz"
│       └── sample2_R2.fastq.gz ..... Copied from "CAMISIM/out/sample2/reads_2/R2.fastq.gz"

```

## 1 3.14 prokka\_ARG.bash

### 2 Stage: Done

### 3 Uses the wrapper tool prokka to predict genes from contigs

4 **Specifics:** This script first concatenates the CARD and NCBI-AMR databases then using the `--protein`  
 5 option, which replaces its database with the one the user specifies, uses that concatenated database to  
 6 look for possible ARGs in contigs.

### 7 Note:

#### Sept 21 2024 Update

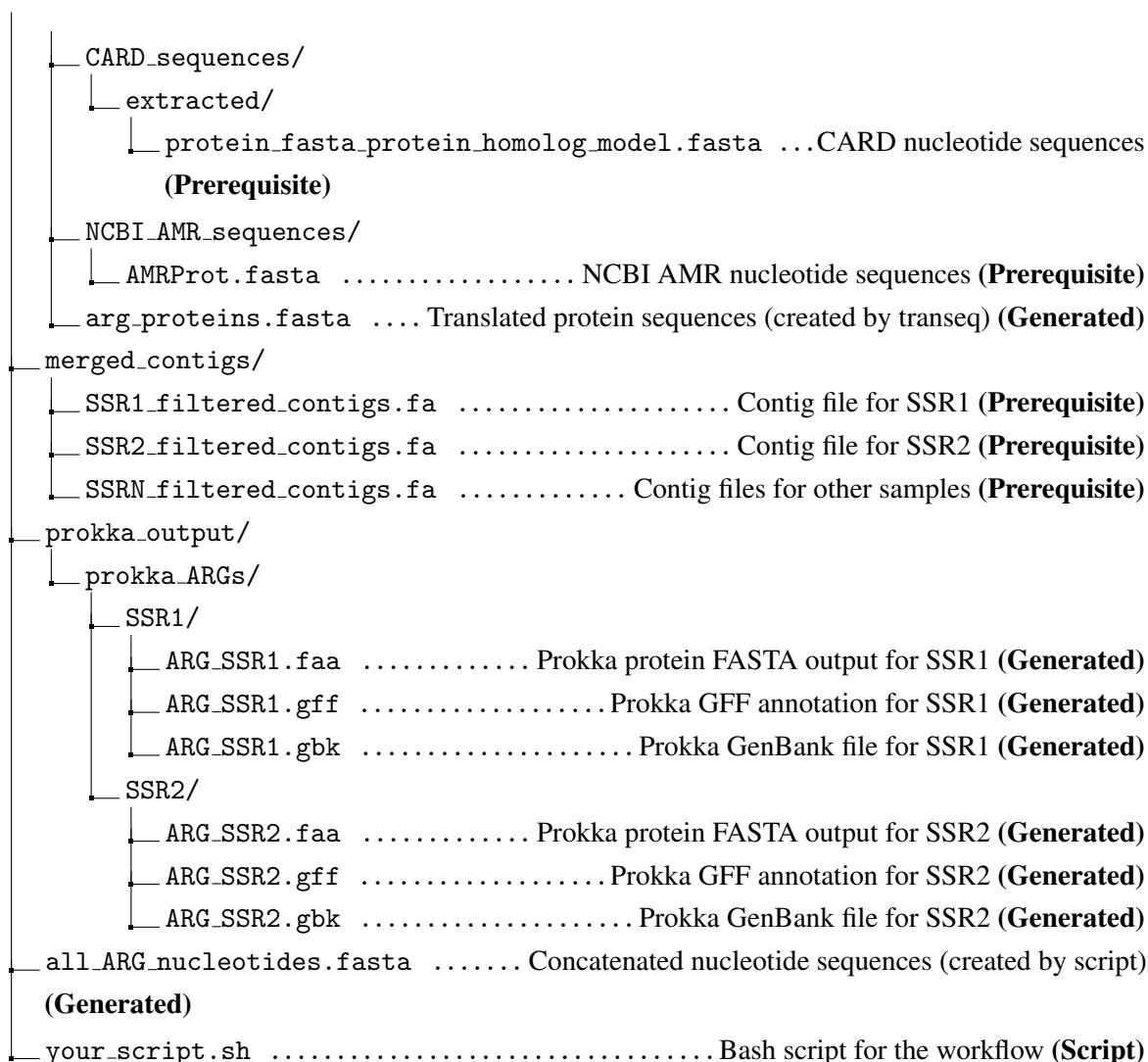
I removed it's companion script `prokka_Uniprot.bash` from the repo because it is redundant. Prokka already uses UniProt. I also updated it to include an `EMBOSS transeq` function to translate the fasta files to protein sequences, something that I overlooked and thought that prokka was able to translate by itself.

### 9 3.14.1 Directory tree

```

project_root/
├── databases/

```



### 3.15 RefSeq.bash

**Stage: Done**

**Downloads Bacterial RefSeq database**

**Specifics:** It uses aria2 a multi-connection download tool with `-x 16 -s -16` parameters meaning 16 connections. It then

- Verifies the downloads (confirming if the downloads are successful)
- Redownloads the unsuccessful downloads
- Extracts each individual file
- Concatenate or combine them into a single file called `refseq_bacteria.fasta`

**Note:** Used to be called `nr_download.bash`, renamed it at Sep 22, 2024.

### 3.16 Pavian analysis.R

**Stage: Done**

**Visualization of Kraken2 via Pavian**

- 4 **Specifics:** Uses Pavian to visualize specifically the `_kraken2_summary.txt` files. Additionally, it calls  
 5 on `ggplot2` to create a bar plot taxonomic classification counts (at different taxonomic levels). It then  
 6 redirects all output (tables and plots) into a `pavian_output.pdf`.  
 7 **Note:** You have to set the change the path to the correct Kraken2 output directory - where the `.kraken`  
 8 files are.

### 1 3.17 parseFastQC.py

2 **Stage: Done**

3 **Interprets all the FastQC summaries**

4 **Specifics:** It does a few things in sequence:

- 5 1. Uses the `summary.txt` files from FastQC.
- 6 2. Creates a Legend that lists the abbreviations used in the file and puts them in a separate `legend.tsv`  
 7 file.
- 8 3. Extracts each FASTQC ZIP file into a temporary directory and looks for the `summary.txt` file.
- 9 4. Generates a summary report called `output.tsv` for each sample (per direction R1 and R2).

10 **Note:** Make sure to set the correct path to the FastQC output directory where the `.zip` files are.

#### 11 3.17.1 Directory tree

```

project_root/
├── fastqc_output/
│   ├── sample1_R1_fastqc.zip .....FastQC zip file for Sample 1 (Read 1) (Prerequisite)
│   ├── sample1_R2_fastqc.zip .....FastQC zip file for Sample 1 (Read 2) (Prerequisite)
│   ├── sample2_R1_fastqc.zip .....FastQC zip file for Sample 2 (Read 1) (Prerequisite)
│   └── sample2_R2_fastqc.zip .....FastQC zip file for Sample 2 (Read 2) (Prerequisite)
├── output/
│   ├── fastqc_report.tsv .....Generated FastQC summary report (Generated)
│   └── legend.tsv .....Abbreviation legend for FastQC metrics (Generated)
└── scripts/
    └── check_fastqc.py ..... Python script for generating FastQC report (Script)
  
```

### 1 3.18 minimum\_length\_CARD.py

2 **Stage: Done**

3 **Description:** Parses the CARD protein homologue FASTA file and identifies the shortest sequence  
 4 length.

5 **Specifics:** This script uses the Biopython library to read each sequence in the FASTA file, comparing  
 6 lengths and returning the smallest sequence length in the dataset.

7 **Note:** The input FASTA file should contain the CARD protein homologue sequences—so the user  
 8 has to set it to where it is.

### 3.19 template modulars: megahit\_binning.sh

**Stage:** Done

**Description:** Template modular script that can be integrated into Snakefiles.

**Specifics:** This script loops through deduplicated paired reads (\*\_R1\_uniq.fastq) and runs MEGAHIT, then filters out contigs > 200 bp. It is written specifically for integration into Snakefiles, so its input and output directories are dictated by the Snakefile shell.

**Note:** MEGAHIT can read FASTQ files—I double-checked, haha.

The modified companion script, megahit\_binning.sh, is specifically designed for the Binning.smk pipeline, but they both perform the same task.

```
MEGAHIT_SETTINGS="--k-min 35 --k-max 141 --k-step 28"
```

**Note:** FastQC.bash is another template modular script that needs to be integrated into Snakefiles, but it instead uses FastQC.

### 3.20 diversity\_bootstrap.R

**Stage:** Draft

**Description:** Reads and plots diversity indices.

**Specifics:** Parses diversity files for individual metrics and generates the following plots:

- **Alpha-diversity plotting:**

- **Ridgeline plot:** Shows the distribution of diversity metrics across samples, per bootstrap replicate.
- **Violin plots:** Displays diversity index distributions with individual points plotted for each bootstrap.

- **Beta-diversity plotting:**

- **Heatmap:** Visualizes the consistency of beta-diversity across bootstraps.
- **NMDS (Non-Metric Multidimensional Scaling):** Plots ordination based on Bray-Curtis distances.

**Note:** This script needs updates to include: (TBA)

- Specific paths to diversity files.
- More customization options for the heatmap, such as clustering methods or color scales.
- Optional: Optimization for memory usage, as large datasets with many bootstraps can consume significant resources.

### 3.21 calculate\_plasmid\_percentage.py

**Stage: Done**

**Description:** This script calculates the percentage of reads that map to plasmids based on the total and plasmid read counts.

**Specifics:** The script performs the following steps:

1. Reads the plasmid and total read counts from input files provided by Snakemake.
2. Calculates the percentage of plasmid reads relative to the total number of reads using the formula:

$$\text{plasmid\_percentage} = \left( \frac{\text{plasmid\_reads}}{\text{total\_reads}} \right) \times 100$$

3. Writes the percentage of plasmid reads (formatted to two decimal places) to the specified output file.

**Note:** The input and output file paths are provided by Snakemake, ensuring integration into a larger pipeline.

### 3.22 calculate\_diversity.py

**Stage: Draft**

**Description:** This script calculates both alpha and beta diversity metrics.

**Specifics:** The script reads species abundance data from .bracken files, calculates various diversity metrics, and outputs the alpha and beta diversity results, both to the console and to a file called `diversity_matrices.tsv`.

#### Alpha Diversity Metrics:

- Fisher’s Alpha
- Chao1
- Berger-Parker Index
- Shannon
- Simpson
- Pielou’s Evenness
- Custom implementations

#### Beta Diversity Metrics (via scikit-bio):

- Bray-Curtis
- Jaccard

**Note:** There are two other beta diversity metrics not included here because they require phylogenetic trees (in Newick file format) beforehand, i.e., UniFrac and unweighted UniFrac (TBA). The Newick file format looks like so:

```
(A:0.1,B:0.2,(C:0.3,D:0.4):0.5);
```

where A, B, C, and D correspond to different taxa, the numbers represent branch lengths (distances), and the bracketing specifies the clades.

## Chapter 4

# Project Side Scripts

### 4.1 kmer\_contam.smk

#### Stage: To test General Purpose

This pipeline is designed for the **detection and removal of contaminant sequences** using k-mer-based filtering. It includes steps for **contaminant k-mer generation**, mapping k-mers to metagenomic reads, and **statistical testing for ambiguous sequences**. The pipeline incorporates tools for **read quality control, contamination filtering, and re-validation** using k-mers and BUSCO analysis.

#### Preprocessing

---

#### 4.1.1 Contaminant Databases Download

**Technical Notes:** The contaminant databases UniVec from NCBI, PhiX, and the 1000 Genomes Project are downloaded using `aria2c` to ensure a fast and reliable download process.

**Rationale:** These databases contain many known contaminants that may appear in metagenomic samples, which must be filtered out before further downstream analyses.

**Personal Notes**

Usually, in standard bioinformatics analyses:

1. The largely annotated human genome, GRCh38 (often referred to as HG38), is an upgraded version of the previous build, GRCh37 (sometimes called HG19), and is commonly used to filter out contaminants.
2. Tools like minimap2 or BBDuk are typically used to map reads to the human genome (or other contaminant references) and then remove contaminant sequences via k-mer matching or alignment.

**So why did I create this more elaborate workflow?**

- Tools like BBDuk and other k-mer-based mappers, such as Kraken2, often rely on **exact matches** to assign taxonomic profiles of contaminants or other sequences.
- While GRCh37 (or GRCh38) is a robust and well-annotated reference genome, it doesn't fully capture the breadth of human genetic diversity. It represents an amalgamation of human populations, but strict k-mer matching means it only identifies specific k-mers from that reference genome. The 1000 Genomes Project covers a broader range of human genetic diversity and is therefore included in this workflow to better account for this variability.

**4.1.2 BUSCO Validation**

**Technical Notes:** BUSCO is used to validate the presence of Single Copy Genes (SCGs) across the metagenomic data. This step ensures the retention of high-quality sequences that are biologically meaningful.

**Rationale:** SCG validation confirms the biological integrity of the metagenomic sample after contamination filtering. It uses all available BUSCO lineages for a comprehensive assessment.

**4.1.3 Marker K-mer Generation**

**Technical Notes:** K-mer counting is performed using Jellyfish for both SCGs and contaminants. This process generates k-mers that represent sequences of interest (SCGs) and contaminants (UniVec, PhiX, 1000 Genomes).

**Rationale:** K-mers allow rapid matching between known contaminants and sample reads, enabling efficient filtering and high-fidelity read retention.

**Contamination Filtering and Statistical Testing****4.1.4 Mapping K-mers**

**Technical Notes:** KMA is used to map k-mers from SCGs and contaminants to raw metagenomic reads.

**Rationale:** Mapping k-mers identifies sequences that match known contaminants, allowing for removal of ambiguous or low-quality sequences from the data.

**Personal Notes**

The use of KMA here specifically, instead of other mappers like Bowtie2 and BWA, is because:

- While Bowtie2, BWA, and KMA are not restricted to **exact matching** (i.e., they can tolerate mismatches, increasing sensitivity),
- KMA performs better with highly redundant databases. Although Bowtie2 and BWA were originally optimized for mapping reads to human genomes or other low-redundancy datasets, they are not restricted to such uses. Both are excellent mappers, but KMA is specifically optimized for working with highly redundant databases, such as those commonly encountered in metagenomic or microbial datasets. For more information, see the RGI documentation.

**4.1.5 Normalization and Testing**

**Technical Notes:** Contaminant and SCG k-mers are normalized using wavelet transformation, and a statistical test (e.g., t-test or Z-test) is performed to detect ambiguous regions.

**Rationale:** Normalization ensures that the k-mer counts are comparable across samples. Statistical testing allows detection of ambiguous areas that may require further analysis or removal. **Notes** (For more details on wavelets, see Box on Wavelets)

**Personal Notes**

The usage of wavelets here increases sensitivity and provides statistical robustness. Since we are no longer relying on exact matching but allowing for mismatches, we expect a distribution of k-mer bindings embedded within the coverage data. To analyze this distribution, we apply a wavelet transformation (similar to a Fourier transformation, but localized) to decompose the data into smaller components (frequencies) that contribute to the overall pattern.

Wavelet-based statistical testing is implemented in this step through a series of Python scripts:

- `normalize_scgs_wavelet.py` and `normalize_contam_wavelet.py` load the k-mer counts from SCGs (identified by BUSCO odb10) and from contaminant databases, respectively. These scripts apply continuous wavelet transformations (CWT) to create wavelet distributions and then calculate Z-scores for the wavelet coefficients to assess their statistical significance. Wavelet coefficients with Z-scores corresponding to a p-value of less than 0.05 ( $Z\text{-score} \geq 1.96$ ) are retained, while the others are filtered out.
- `compare_wavelets_stats.py` compares the wavelet distributions of SCGs and contaminants, testing whether they are statistically different (with the null hypothesis,  $H_0$ , being that they are not different). If the null hypothesis is accepted (i.e., the distributions are similar), the sequences are flagged as ambiguous and filtered out.

```
significant = np.where(p_value < 0.05, 'significant', 'ambiguous')
```

**Update Sep 27, 5:33 am** `compare_wavelets_stats.py` has also been now updated to handle:



- Decision making of using parametric or non-parametric wavelet tests
- Handle a battery of different tests
- Apply Bonferroni and FDR corrections for such tests

#### 4.1.6 Filtering Ambiguous Sequences

**Technical Notes:** Sequences identified as ambiguous are filtered out to retain only high-fidelity reads in the final metagenomic assembly.

**Rationale:** This step ensures that the final assembly is free of contamination and contains only high-quality sequences for downstream analyses.

#### 4.1.7 Re-validation of SCGs

**Technical Notes:** BUSCO is rerun on the final cleaned assembly to confirm that SCGs have been retained after contamination filtering.

**Rationale:** Re-validation with BUSCO ensures that the final dataset remains biologically meaningful after the filtering process.

#### 4.1.8 Directory tree

```

project_root/
├── databases/
│   ├── UniVec .....(Generated)
│   ├── PhiX .....(Generated)
│   └── 1000_genomes.fasta ..... (Generated)
├── results/
│   ├── cleaned_high_fidelity_spikes.fasta
│   ├── scg_kmer_stats.txt
│   ├── contam_kmer_stats.txt
│   ├── phix_kmer_stats.txt
│   ├── genome_1000_kmer_stats.txt
│   ├── wavelet_normalized_contam.txt
│   ├── wavelet_normalized_scgs.txt
│   ├── t_test_results.txt
│   └── ambiguous_sequences.txt
├── busco_outputs/
│   ├── dataset1/
│   ├── dataset2/
│   └── ...
├── busco_validation_outputs/
│   └── short_summary.txt
└── scripts/ .....(Prerequisite)

```

		normalize_contam_wavelet.py .....	(Prerequisite)
		normalize_scgs_wavelet.py .....	(Prerequisite)
		compare_wavelets_stats.py .....	(Prerequisite)
		env/	
		busco.env.yaml .....	(Prerequisite)
		jellyfish.env.yaml .....	(Prerequisite)
		raw_reads.fastq .....	(Prerequisite)

### Update Sep 27 3:57 am

Provided the limitations of the original Snakefile, which assumes normally distributed k-mer binding, I created another workflow that tests "normality" of the distribution. This determines whether a Z-test or t-test is valid. Here's a brief explanation of the scripts:

- `normality_test.py` uses the Shapiro-Wilk and Kolmogorov-Smirnov tests to determine normality. If normality fails, we proceed with:
- `fit_distributions.py` to check whether the data follow log-normal, exponential, or gamma distributions, which can be transformed using:
- `transform_data.py`, which attempts to normalize the data. If normalization succeeds, we proceed with parametric tests.
- If normalization fails, we use `check_nonparametric.py` to determine the type of non-parametric distribution the data likely follows.
- To increase confidence in the distribution type, I included two scripts that test goodness of fit:
  - `goodness_of_fit_parametric.py`, and
  - `goodness_of_fit_nonparametric.py`.

Using statistical tests rather than visual inspection removes human judgement bias and focuses purely on mathematical/statistical validation.

In line with this, researchers often settle on subjective p-values (e.g.,  $< 0.05$  or  $< 0.01$ ) as thresholds, known as alpha. I included scripts that determine the best alpha for whichever distribution has the best goodness of fit:

- `best_alpha_parametric.py` and `best_alpha_nonparametric.py`, either of which feed into:
- `bestfit_and_alpha.py`, which determines the best alpha for downstream analyses.

<sup>57</sup> **Chapter 5**

<sup>58</sup> **Project Main Scripts**

<sup>1</sup> **5.1**

3

## **Further investigations**

2

4 This section comprises of concepts or ideas that require further investigation. They are written in  
5 boxes to help categorize them cleanly. I will refer to this section often as no script is perfect - and can be  
6 further improved.

### Biological Information

I have long pondered upon the idea of creating a graph with biological information on the y-axis and read length on the x-axis. I hypothesize that we will find a "sweet spot" wherein we can optimize the amount of information/read using such trimmers. Unfortunately, it is extremely difficult to define what a biological sequence really is, because technically you can generate any random sequence - whether protein or nucleotide. That's the main reason I have been stuck on this problem for quite a while, I've been trying to find the answer first.

The core difficulty here lies in defining what constitutes biological information in a meaningful, quantifiable way. Since any random sequence of nucleotides or proteins could be technically "valid" (false-positives).

When people are asked this question they often give **DESCRIPTIONS** of what a biological sequence is but not what **DEFINES** a biological sequence. I understand that there are many characteristics that can help in determining the signal from the noise, but I am just not satisfied with whether this "thing" checks most (if not all) the boxes - I need a non-subjective answer to this problem.

### Box on Wavelets

#### What is a Wavelet?

A **wavelet** is a small, localized wave that is used to represent signals at different scales. Unlike sinusoids in the Fourier transform, which extend infinitely, wavelets are confined to a limited duration. This makes them highly useful for capturing both frequency and time (or space) information simultaneously.

Wavelets are particularly well-suited for analyzing signals with transient or localized features, such as sharp peaks, edges, or changes in behavior. Because of their ability to zoom in on fine details while also capturing broad trends, wavelets are widely used in signal processing, image compression, and data analysis.

#### How Do Wavelets Work?

Wavelets work by breaking down a signal into smaller, simpler components—each representing the signal at different scales. The signal is convolved with a family of wavelets, each scaled and shifted to analyze the data at various resolutions. The result is a multi-scale representation that provides insights into both high-frequency details (like sudden spikes) and low-frequency trends (like global patterns).

#### Continuous Wavelet Transform (CWT)

The **Continuous Wavelet Transform (CWT)** is a specific type of wavelet transform that decomposes a signal continuously over a range of scales. CWT produces a 2D representation of the signal in both time and frequency domains, making it ideal for detecting localized features that vary across scales.

#### Why CWT for k-mer Normalization?

In bioinformatics, CWT is employed to normalize k-mer counts by analyzing the data across multiple scales. This multi-scale analysis captures localized features in the k-mer distributions, such as regions affected by contaminants or biologically significant regions (e.g., single-copy genes, SCGs). By applying CWT, we can detect and highlight these variations while preserving the overall structure of the data.

### **Signal vs. Noise Differentiation with CWT**

One of the key strengths of wavelet transforms is their ability to differentiate signal from noise. By breaking down a signal across multiple scales, wavelets can identify high-frequency components often associated with noise, while preserving lower-frequency, biologically meaningful signals. This is especially important in this workflow, where tools that allow k-mer mismatches are used. Unlike tools like minimap2 or BBDuk, which prioritize specificity (via exact kmer matching), the wavelet-based approach prioritizes sensitivity, helping to detect real signals despite mismatches or noise in the data.

### **Key Advantages of CWT:**

- **Multi-scale analysis:** CWT allows us to view the data at different scales, capturing both small localized features and broader trends.
- **Localized feature detection:** Unlike global methods (e.g., Fourier transform), CWT can detect localized anomalies in the data, such as contamination spikes that only affect certain regions.
- **Non-stationary signal analysis:** Many biological signals, including k-mer counts, are non-stationary (their statistical properties vary over time or space). CWT is well-suited for handling such data.
- **Noise filtering:** CWT can differentiate high-frequency noise from low-frequency signal, making it an excellent tool for extracting meaningful data even in noisy datasets.

### **Application in This Workflow:**

In this workflow, CWT is applied to normalize k-mer counts for both contaminants and SCGs. After normalization, statistical tests like t-tests or Z-tests are used to detect ambiguous regions that may require further analysis or removal. This process ensures that k-mer counts are comparable across samples and that biologically meaningful patterns are preserved.

## **Daily entries**

## 1 September 26, 2024


### To-do List

#### Note:

- ✓ distribution.yaml for kmer\_contam.smk

### Completed Tasks

- ✓ kmer\_contam.smk explanation (initial)

 what\_distribution.smk

### Scripts Worked On

- kmer\_contam.smk
- what\_distribution.smk
- best\_fit.py
- check\_nonparametric.py
- compare\_wavelets\_stats.py
- fit\_distributions.py
- goodness\_of\_fit.py (parametric)

- goodness\_of\_fit\_nonparametric.py
- normality\_test.py
- normalize\_contam\_wavelet.py
- normalize\_scgs\_wavelet.py
- transform\_data.py
- best\_alpha\_parametric.py
- best\_alpha\_nonparametric.py
- compare\_wavelets\_stats.py
- bestfit\_and\_alpha.py
- compare\_wavelets\_stats.py

### Issues Encountered

- 

### Need to Troubleshoot

-



## 2 September 25, 2024

### To-do List

#### Note:

✓

### Completed Tasks

- Did further research on repo Side to further research to improve wavelet transform-based contaminant removal
- Conceptualized the 'Exponential Decay and

Biological Gene Expression' Box found in Information Dump

### Scripts Worked On

- Drafted a biological\_halflife.smk
- onoroff.py

### Issues Encountered

•

### Need to Troubleshoot

•

### 3 September 23, 2024

#### To-do List

##### Note:

- ✓ Start group meetings regarding paperwork in private repo
- ✓ Draft the Paperwork.tex document
- ✓ Moved some daily entries tied to Confidential work to Paperwork.tex document

#### Completed Tasks

- 

#### Scripts Worked On

- 

#### Issues Encountered

- 

#### Need to Troubleshoot

-

## 4 September 22, 2024

### To-do List

#### To do list

- ✓ Project4 Modules
  - ✓ KMA-iterative now deprecated  
check own section as to why
  - ✓ NCBI\_taxonomy.bash now also deleted  
from the repo as deprecated (residual  
file used to be part of the original  
Kraken2 pipeline)
  - ✓ prokka\_ARG.bash
  - ✓ RefSeq.bash
  - ✓ Pavian\_analysis.R
  - ✓ parseFastQC.py
  - ✓ minimum\_length\_CARD.py

- ✓ megahit\_binning.sh
- ✓ megahit\_binning.sh
- ✓ FastQC.bash
- ✓ calculate\_plasmid\_percentage.py
- ✓ calculate\_diversity.py

### Completed Tasks

•

### Scripts Worked On

•

### Issues Encountered

•

### Need to Troubleshoot

- ✓ Ubuntu crashed (again) (resolved)

## 5 September 21, 2024

### To-do List

#### To do list

- ☐ Project4 Modules
  - ✓ `trimmomatic.bash`
  - ✓ `renamingSIMS.bash`

#### Completed Tasks

- ✓ Restructured directory trees so it's easier to understand.

- ✓ Restructuring of the checklist to become a Kanban board so it's easier to understand at a glance.

#### Scripts Worked On

- 

#### Issues Encountered

- 

#### Need to Troubleshoot

-

## 6 September 20, 2024

### To-do List

#### To do list

- ☐ Project4 Modules
  - ✓ krakenpipeline.bash
  - ✓ raw.bash (optional)
  - ✓ summary\_stat.bash
  - ✓ plot\_and\_detect...py
  - ✓ refseq\_bacteria.bash
- ✓ Send the signed AR, PCF, and COS documents as pdf

#### Completed Tasks

- ✓ Created directory trees template for each script (if relevant, which includes prerequisites and temporary folders)

- ✓ Restructuring of this Documentation to include Further Investigations, Supplementary Data, and Information section for more streamlined reading of the script portions.

#### Scripts Worked On

- ✓ Improved plot\_and\_detect...py for more robust analysis (Details on its own section)
- ✓ Drafted a fix to the Shortbred\_ARG.bash shell (Details on its own section)

#### Issues Encountered

- Landline-based to-do list moved to next week due to landline down and it's weekend tomorrow so offices will be closed.

#### Need to Troubleshoot

-

## 7 September 19, 2024

### To-do List

- ☐ Try to complete LaTeX documentation of scripts
  - ✓ Project 4 pipelines
    - ✓ ARG-MGE.smk
    - ✓ metagenomics\_pipeline.smk
    - ✓ Binning\_.smk
  - ➡ Project Main pipelines
  - ➡ Project Side pipelines
- ✓ Counter-check Project4 scripts with documentation to see redundancies

### Completed Tasks

- ✓ Deleted redundancies in Project 4 scripts
  - FastQC\_check.py
  - fastp.bash
  - README.md (since this document exists and is more comprehensive)

– fastp metrics from summary\_stat.bash

- ✓ Decided to move to Project Side K-mer\_diversity.smk as it is not essential (yet).
- ✓ Finally resolved an issue with wakatime not integrating properly with TeXStudio
- ✓ I realize now that the entry dates were one-day behind.
- ✓ Also added the names of the python scripts I refer for proper documentation.

### Scripts Worked On

- calculate\_contig\_quality.py now updated to calculate coverage statistics.

### Issues Encountered

- Landline-based to-do list moved due to landline down

### Need to Troubleshoot

-

## 8 September 18, 2024

### To-do List

- ➡ Call for a group meeting regarding logistics
  - ✓ Offer to use landline to minimize on-site visits and reduce transportation costs; as point-persons have not been replying via email as quickly lately
  - ✓ Discuss PGC calculations and talk about allocatable budget
  - ✓ On-site orientation with PGC engineering department (OETS) regarding sampling sites **taken care of by James and Roch**
- ➡ Raise concerns about data storage and management
- ➡ Inquire if there are scripts I can develop to streamline the bureaucratic process

### Completed Tasks

- Setup wakatime in VS Code and Ubuntu to track my coding productivity
- Helped in receiving and moving the autoclave and fridge needed by the Program

### Scripts Worked On

- `rawAPIformatter.sh`
- ✓ `checkingAPI.sh`
- `wakatime.sh`

### Issues Encountered

- ✓ Connection blockage from wakatime API
- ➡ raw API results is unreadable

### Need to Troubleshoot

- conky display of wakatime API for desktop

## 9 September 17, 2024

### To-do List

- ➡ Call for a group meeting regarding logistics
  - ➡ Offer to use landline to minimize on-site visits and reduce transportation costs; as point-persons have not been replying via email as quickly lately
  - ➡ Discuss PGC calculations and talk about allocatable budget
  - ➡ Raise concerns about data storage and management
  - ➡ Inquire if there are scripts I can develop to streamline the bureaucratic process

### Completed Tasks

- ✓ Updated GitHub repositories; created new repositories: Documentation and Confidential
- ✓ Created a new bash script to track progress across all repositories

### Issues Encountered

- ✓ Could not access GitHub, needed to reset the SSH keys and update my Git histories for version control

### Need to Troubleshoot

- None identified at this time

- ✓ Set up this research diary to document daily progress and tasks
- ✓ Python script that calculates contig quality metrics L50, N50, but also L90, N90, GC skew etc.
- ✓ Python script that detects and calculates changes in coverage over contigs for read mapping
- ✓ Typesetting: explaining the process and rationale inside the ARG-MGE.smk pipeline
- ✓ Integration of the two scripts to the ARG-MGE pipeline

### Scripts Worked On

- Project4/ARG\_MGE.smk - updated
- ✓ Project4/calculate\_contig\_quality.py
- ✓ Project4/plot\_and\_detect\_intermediate\_coverage.py



## **Supplementary data**

---

This will contain data on my investigations on standardized or simulated datasets. This is mainly a testing ground for the scripts, and their accompanying data visualizations.

## **Information dump**

This part is mainly created as an information dump outside of just bioinformatics, that perhaps we can one day apply to bioinformatic data analysis (mostly). This part could also serve as an index of papers that I've read regarding various topics that seem interesting to me and that I think can be applied in the wider field of computational or mathematical biology.

## 1 Biological OFF "Decay"

In statistical mechanics, microstates (positions, velocities, energies, etc.) are often unpredictable—this is related to Schrödinger's thought experiment and further elaborated by Werner Heisenberg's Uncertainty Principle. However, macrostates (large-scale properties) that emerge from the ensemble of microstates are model-able. For **radioactive decay**, while we can't predict when individual atoms will decay, we can model the **half-life** of the entire ensemble. The **exponential decay equation** models this process:

$$N(t) = N_0 e^{-kt}$$

Where:

- $N(t)$  is the amount of substance remaining after time  $t$ ,
- $N_0$  is the initial amount of the substance (at  $t = 0$ ),
- $e$  is the base of the natural logarithm ( $e \approx 2.718$ ),
- $k$  is the **decay constant**, determining the rate of decay,
- $t$  is the time elapsed.

This equation describes a process where the quantity decreases over time at a rate proportional to its current value, leading to exponential decay. The negative exponent indicates that the amount is **decreasing** over time.

### How does this connect to gene expression?

Individual cells in a tissue either have certain genes **ON** or turned **OFF**—this binary on/off behavior is analogous to the decay of atoms (binary:

decayed or not). Thus, we can use the same exponential decay model to predict the number of genes that turn **OFF** in a population of cells (or tissue or organism) over time.

In this model, we simply change the variables to

- $N(t)$  is the number of cells with genes still **ON** at time  $t$ ,
- $N_0$  is the initial number of **ON** genes (at  $t = 0$ ),
- $k$  is the **decay constant**, determining the rate at which genes are turned **OFF** (analogous to radioactive decay),
- $t$  represents biological time (e.g., the time it takes for environmental conditions to cause changes in gene expression).

### How can we determine the decay constant $k$ ?

In gene expression studies,  $k$  could represent the rate at which environmental or physiological conditions (such as cold temperatures or seasonal changes) cause genes to turn off. This decay constant could be derived from **empirical data** (e.g., using transcriptomics data like DeSeq2) by calculating how quickly gene repression occurs over a certain period of time.

The model can help us estimate when a certain percentage of genes will be repressed, akin to the half-life concept in radioactive decay. This framework allows us to mathematically predict e.g. when a plant or organism is preparing for overwintering, or to determine the rate of gene repression under specific environmental conditions.

## 2 Best Data science practices

Here are some of the best practices and principles in data science which can be adopted to any scientific discipline dealing with large amount of data.

1. Defining the problem clearly which includes
  - Objective/s
  - Scope of the Project
  - Key metrics for success
2. Knowing data data which including
  - Data structure
  - Data type
  - Data distribution
  - Anomalies e.g. missing values
  - Imbalances in data e.g. oversampling, undersampling, synthetic data generation etc.
3. Ensure reproducibility
  - Documentation: cleaning, analysis, modeling
  - Scripts and Notebooks
  - Makes your code easier to understand and facilitates collabs and updates.
4. Validation
  - Split data into training and testing sets to ensure models generalize well to unseen data.
  - Prototype first before implementation
  - Feature Engineering
  - Optimize trade-off between complexity and interpret-ability
  - Implement Data Augmentation techniques to increase diversity and quantity of training data, improving robustness
5. Consider ethical practices and biases; ensure fairness on treatment of data
  - Data privacy and Security is top priority
    - Encryption
    - Access controls
    - Anonymization techniques
6. Automate where possible to reduce the likelihood of human error - this includes:
  - Pipelines
  - Feature engineering updates
  - Feedback
7. Stay updated with the latest tools
8. Scalability: ensure models
  - Can handle increasing amounts of data (data storage scalability) without
  - Significant performance degradation
  - Significant increases in computational resources
9. Impact: avoid "Analysis Paralysis" by focusing on insights that are actionable
10. Implement feedback loops based on
  - New data
  - Performance metrics
  - Stakeholder and end-user feedback
  - Alignment with broader objectives (be relevant and actionable)
  - Explore other models
  - Data leakage checks
  - Note that bias can arise from data**
  - Appropriate metrics are used
  - **Always be skeptical of results**
  - Feedback from domain expertise
  - Auditing

11. Mind the end users; models must be

- Accessible
- Interpretable
- Usable
- Continuous Integration and Deployment (CI/CD)

12. Communication:

- Uncertainties

- Assumptions

- Model explainability techniques: how complex models make decisions

13. Understand the data lifecycle

- (a) Collection to preprocessing
- (b) Analysis
- (c) Modeling
- (d) Eventual archival and deletion