Rapport de Reprohackathon

Gabriel CLEREMPUY, Maëlle CORNEC
Mathieu GUIMONT, Suzanne Guilteaux

Master IODAA - AMI2B

Septembre 2023 - Décembre 2023

Référents :
Frédéric LEMOINE
Thomas COKELAER

Résumé

Ce rapport présente notre expérience de reproductibilité visant à reproduire les résultats d'un article de la revue *Nature*, publié en mai 2020, intitulé *Intracellular Staphylococcus aureus persisters upon antibiotic exposure* et écrit par Frédéric Peyrusson, Hugo Varet, Tiep Khac Nguyen, Rachel Legendre, Odile Sismeiro, Jean-Yves Coppée, Christiane Wolz, Tanel Tenson et Françoise Van Bambeke. Cet article évoque les potentiels facteurs qui conduisent à la transition de *Staphylococcus Aureus* vers un phénotype persistant face à l'exposition aux antibiotiques.

L'objectif de ces quatre derniers mois était de se familiariser avec les différentes méthodes et outils tels que VS Code, Github, Dockerhub et Snakemake, détaillés plus tard, afin de reproduire un graphique de la variation d'expression génique entre les échantillons du groupe contrôle et les populations de bactéries du phénotype persistant. Cela permet de mettre en évidence les gènes dont l'expression est significativement différente par rapport aux groupes contrôle et ainsi d'identifier les gènes qui jouent un rôle clé pour le phénotype persistant.

Les résultats de notre expérience sont comparés à ceux de l'article d'origine et nous discutons des différences, des défis rencontrés tout au long du projet, et évoquons quelques recommandations pour améliorer la reproductibilité de telles études bioinformatiques. Cette expérience permet finalement de mettre en lumière l'importance de la reproductibilité dans la recherche en bioinformatique et ses implications pour la validité des résultats.

Table des matières

Table des figures

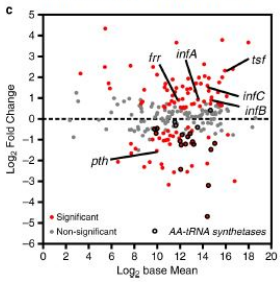
1 Introduction

Ce projet de Reprohackathon a été motivé par la crise de reproductibilité qui touche le monde de la recherche académique et notamment le domaine de la bioinformatique. En effet, cette crise a mis en évidence la nécessité d'améliorer la façon dont les analyses de données en bioinformatique sont mises en œuvre, exécutées et partagées. Par conséquent, ont été mis en place divers outils tels que des systèmes de gestion de versions comme Git, des outils de gestion de flux de travail (workflow) tels que Snakemake et Nextflow et des outils de gestion d'environnement logiciel à l'image de Docker et Singularity.

Ainsi, ce projet vise à se familiariser avec ces différents outils afin de reproduire une partie d'un article scientifique, publié dans la revue *Nature* et portant sur les populations de *Staphylococcus aureus* (abrégé en *S.aureus*) intracellulaires persistants après exposition aux antibiotiques. Ces cellules persistantes bactériennes sont des variants phénotypiques qui présentent un état de non-croissance transitoire et une tolérance aux antibiotiques. La présence de variants persistants est généralement mise en évidence par des courbes de destruction biphasique : une grande partie de la population bactérienne est sensible à l'antibiotique utilisé et est rapidement tuée tandis qu'une sous-population avec un "killing rate" plus faible émerge et persiste alors pendant une période beaucoup plus longue. Ce sont les *persisters*. Ils sont capables de survivre aux effets des antibiotiques et ont les caractéristiques d'être métaboliquement actifs et de présenter un profil transcriptomique modifié qui correspond à la répression de gènes liés à la prolifération et à l'activation de ceux responsables des mécanismes de défense et de réponse au stress induit par l'environnement antibiotique. Ce phénotype persistant est stable mais réversible lors de l'élimination de l'antibiotique. Par ailleurs, ces changements sont associés à la tolérance à plusieurs médicaments malgré une exposition à un seul antibiotique. Ces persistants sont problématiques car ils pourraient constituer un réservoir pour les infections récidivantes et ainsi contribuer à l'échec des thérapies.

Ici l'objectif est de reproduire deux graphiques (Figure 1) d'une expérience de séquençage d'ARN permettant de visualiser l'expression différentielle de gènes entre les échantillons du groupe contrôle et ceux des populations du phénotype persistant de *S.aureus*. Pour cela, nous avons utilisé les outils mentionnés ci-dessus (Github, Snakemake et Docker) afin de tenter de reproduire les résultats de l'article pour cette analyse des variations de l'expression génique.

Fig. 3.c.: MA-plot of genes related to translation



Supp. Fig. 3.: MA-plot of complete RNA-seq dataset

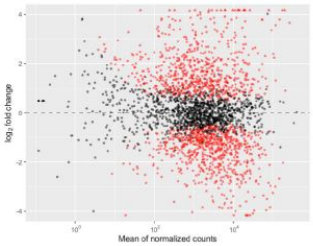


FIGURE 1 – Graphiques de l'article à reproduire
A gauche, le graphique affiche le log2 Fold Change en fonction du log2 Base Mean (signal d'expression moyen sur l'ensemble des échantillons). Les gènes impliqués dans la traduction sont pointés et les aminoacyl-ARNt synthétases sont entourés en noir. La ligne en pointillés indique le niveau d'expression de base dans les échantillons de contrôle. La signification statistique est basée sur la valeur P-ajustée.

2 Plan Expériemental

Pour reproduire l'expérience, nous avons suivi les différentes étapes suggérées dans le sujet du projet et indiquées ci-dessous (Figure 2) :

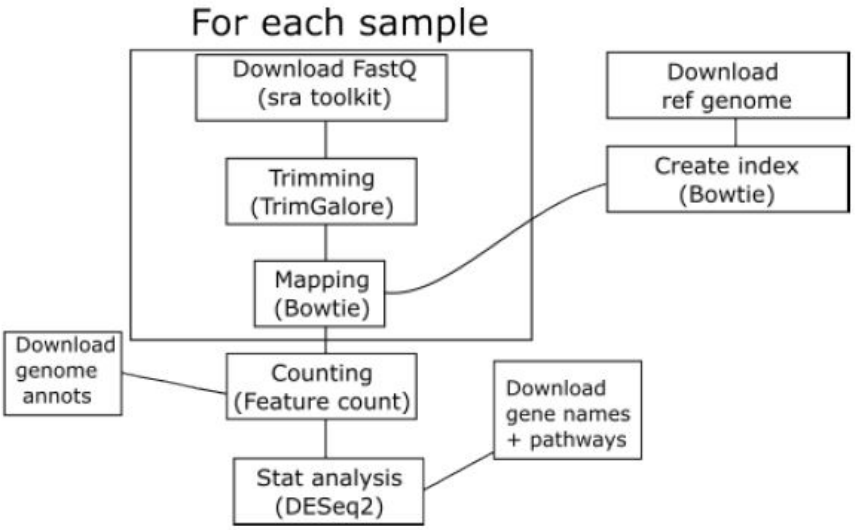


FIGURE 2 – Workflow à implémenter et à exécuter
Différentes étapes du workflow nécessaires à la reproduction de l'expérience du téléchargement des données à l'analyse statistique, en passant par le prétraitement et la génération du fichier de comptage des gènes.

2.1 Acquisition des données

Pour reproduire l'étude, nous avons commencé par télécharger les données brutes utilisées pour l'étude et mentionnées dans la section *Methods* de l'article. Six fichiers *.fastq* ont été téléchargés à l'aide de la commande fastq dump de SRA Toolkit et du numéro SRR des fichiers dans la base de données NCBI. Parmi ces six fichiers, trois concernent les échantillons du groupe contrôle et les trois autres portent sur des populations de persistants.

Le génome de référence, indiqué dans la section *Data Availability*, a également été téléchargé, ainsi que ses annotations au format *.gff*.

2.2 Environnement logiciel

Pour la mise en place de l'environnement de travail dédié à l'exécution du workflow, chacun des outils nécessaires à la reproduction de l'expérience ont été placés dans un container Docker. Pour garantir une meilleure reproductibilité, nous avons porté une attention particulière aux versions des outils utilisés par l'étude. Ces versions sont notamment mentionnées dans la sous-section *RNA-Sequencing* de la partie *Methods* de l'article. Une fois les images des outils construites (SRA Toolkit version 2.10.0, TrimGalore 0.6.4, Bowtie

0.12.7 et featureCounts 1.4.6-p3), nous les avons poussées sur Dockerhub pour en faciliter l'utilisation. La version de SRA Toolkit n'étant pas spécifiée dans l'article de *Nature*, nous avons opté pour une version cohérente avec la date de publication de l'article (mai 2020), de même pour TrimGalore. Dans le container de ce dernier, nous avons également dû installer cutadapt mais nous ne sommes pas parvenus à télécharger sa version 1.11, *a priori* pour des raisons de compatibilité. Nous avons donc opté pour une version par défaut, 2.8. L'énoncé du projet mentionnait l'utilisation de TrimGalore pour effectuer le nettoyage des séquences, tandis que l'article mentionnait l'utilisation de cutadapt pour cette étape. Il aurait alors été intéressant de comparer les résultats obtenus avec TrimGalore et ceux engendrés par cutadapt et mesurer l'impact potentiel que ce changement d'outils cause sur la reproductibilité de l'expérience.

Concernant le container de Bowtie 0.12.7, nous n'avons pas inclus directement l'outil *samtools* dedans car nous avons privilégié d'une part, l'utilisation d'un container Bowtie construit par nous-mêmes pour exécuter la commande *bowtie build* et d'autre part, l'utilisation d'une image déjà construite de *samtools* et disponible depuis Dockerhub pour exécuter la commande *samtools*.

Par ailleurs, nous avons également rencontré quelques difficultés sur la construction de l'image de R 3.4.1 et de l'installation de BiocManager et DESeq2 1.16. Pour résoudre les problèmes survenus (échec de la décompression, compilateur absent, versions non compatibles), notamment ceux de comptabilité, nous avons tenté d'utiliser une image de base ubuntu plus ancienne pour pallier les problèmes de la construction de l'image avec les versions mentionnées des outils, en vain. Nous avons donc décidé de poursuivre l'expérience en utilisant une image de R et DESeq2 déjà disponible depuis Dockerhub, bien que ce ne soit pas les bonnes versions. De plus, notre script R original nécessitant DESeq2 et factoextra pour l'analyse statistique et l'affichage du graphe d'ACP et comme nous n'avons pas trouvé d'images fonctionnelles ayant à la fois les outils DESeq2 et factoextra (pour l'ACP), nous avons divisé le script R en deux parties. La première partie du script contient l'analyse statistique avec DESeq2, pour laquelle nous avons repris l'image de pegi3s r_deseq2 (*docker pull pegi3s/r_deseq2 :1.32.0_v2*). La seconde partie du script R comporte les lignes de code nécessaires à l'ACP et repose sur une autre image disponible sur Dockerhub et contenant factoextra *(kamirab/r_deseq_pca :latest)*.

De ces premiers obstacles, nous constatons que la reproduction des conditions de travail, notamment lorsqu'elles mettent en jeu des anciennes versions logicielles, constitue l'un des premiers défis de la bonne reproductibilité de l'expérience.

2.3 Prétraitement des données

2.3.1 Nettoyage des séquences

Nous avons effectué un prétraitement des données conformément aux méthodes décrites dans l'article original, en appliquant d'abord Trimgalore, comme suggéré par l'énoncé du projet, pour nettoyer les données de séquençage d'ARN des six échantillons *.fastq*. L'étude mentionne, toujours dans sa section *Methods*, que les séquences ont été débarrassées des adaptateurs, des séquences de faible qualité et que seules celles d'une longueur d'au moins 25 nucléotides ont été conservées pour la suite de l'analyse. Par conséquent, nous avons utilisé la commande suivante :

trim_ galore -q 20 –phred33 –length 25 <FASTQ FILE>.

— "-q 20" permet de spécifier le seuil de qualité minimum. Les bases ayant une qualité inférieure à 20 seront trimmées, ce qui signifie que TrimGalore éliminera donc les bases de faible qualité.
— "–phred33" permet d'indiquer que les scores de qualité dans les fichiers fastq sont au format Phred+33, l'un des formats les plus couramment utilisés pour représenter la qualité des séquences dans ces fichiers.
— Enfin, "–length 25" permet d'indiquer la longueur minimale requise pour une séquence après le trimming. Conformément à l'expérience de l'article, les séquences de moins de 25 nucléotides sont alors éliminées.

Après cette étape de Trimming dans laquelle nous mettons nos six fichiers *fastq* en entrée, nous obtenons également en sortie six fichiers trimmés *_ trimmed.fq*.

2.3.2 Alignement des séquences et construction de l'index du génome

Ensuite, nous avons effectué l'alignement des séquences d'ARN à l'aide de la version 0.12.7 de l'outil Bowtie. D'abord, nous avons construit l'index qui va permettre d'aligner les séquences sur le génome indexé grâce à la commande :

bowtie-build genome.fasta output

On utilise la fonction bowtie-build qui crée six fichiers index (au format ebwt) à partir du génome de référence (au format fasta). On utilise la commande :

bowtie -p 4 -S -x index/indexation input | samtools sort -@ 4 -o output samtools index output

— "-p 4" définit le nombre de cœurs que le processeur doit utiliser. Le nombre 4 a été choisi de manière arbitraire car il s'agit d'un nombre de cœurs standard

pour un processeur.

— "-S" indique à bowtie de générer une sortie au format SAM (Sequence Alignment/Map).

— "-x index" spécifie l'index (généré avec bowtie-build) utilisé pour l'alignement. Il indique le chemin vers l'indexation du génome.

— "input" fait référence aux 6 fichiers fastq modifiés par trim_galore à l'étape précédente.

— "samtools sort" est une commande de samtools qui prend en entrée le fichier SAM et le trie pour produire un fichier au format BAM.

— "-@ 4" spécifie le nombre de threads que samtools sort peut utiliser pour trier les données.

— "-o output" spécifie le nom du fichier de sortie trié, le fichier BAM.

— samtools index output crée un fichier index (.bami) pour le fichier BAM. Celà permettra d'extraire les informations plus rapidement.

Dans le pipeline, cette étape fait l'objet de deux règles séparées, l'une utilisant comme mentionné ci-dessus l'image de bowtie pour l'exécution de la commande *bowtie* et l'autre l'image déjà construite de samtools pour exécuter la commande *samtools*.

2.3.3 Comptage des gènes

Enfin, avant de passer à l'analyse statistique, nous avons utilisé l'outil featureCounts pour compter le nombre de lectures de séquençage alignées sur les caractéristiques du génome de référence, à l'aide de la commande suivante :

featureCounts -t gene -g ID -F GTF -T 4 -a input.ref -o output input.ech

— "-t gene" spécifie le type de fonctionnalité que featureCounts va compter. Ici, il s'agit de gènes.

— "-g ID" spécifie l'attribut GTF utilisé pour identifier les fonctionnalités. Ici, l'identifiant est "ID".

— "-F GTF" indique que le format du fichier d'annotation fourni est au format GTF.

— "-T 4" : spécifie le nombre de cœurs de processeur à utiliser pour l'opération.

— "-a input.ref" fournit le fichier GTF contenant les annotations à utiliser pour le comptage.

— "-o output" indique le nom du fichier de sortie (counts.txt) où featureCounts va stocker les comptages générés.

— "input.ech" est le fichier BAM pour effectuer le comptage des lectures alignées sur les fonctionnalités géniques spécifiées.

A l'issue de cette étape, nous avons obtenu un fichier de comptage .txt exploitable pour l'analyse statistique.

2.3.4 Analyse statistique

Pour reproduire le graphique de l'étude, nous avons utilisé le package de R DESeq2 pour analyser les données obtenues dans le fichier de comptage des gènes, afin d'identifier les gènes différentiellement exprimés entre les échantillons du groupe contrôle et ceux des populations persistantes. L'objectif était de construire le graphique représentant la variation de l'expression génique par rapport à l'état de référence (Log2 Fold Change) en fonction de la moyenne de l'expression génique à travers les échantillons (Log2 Base Mean). Ces deux valeurs sont exprimées en échelle logarithmique (Log2). Chaque point représente alors un gène spécifique, avec en abscisse son niveau d'expression de base et en ordonnée à quel point son expression a changé. La coloration des points indique si la variation est statistiquement significative. Pour l'expression différentielle des gènes, la p-valeur retenue est de 5%, conformément à la méthode de l'article de *Nature*.

Pour ce qui est du script R, nous avons commencé par appeler la librairie DESeq2 qui apparatient au package BiocManager. On utilise ensuite les fonctions read.csv, et na.omit pour charger les données et omettre les lignes contenant des valeurs manquantes sur une sélection de colonnes bien précise.

Nous avons utilisé la fonction *DESeqDataSetFromMatrix* du package DESeq2 afin qu'elle soit mise en forme pour être utilisée par le package. On normalise ensuite les données avec DESeq comme stipulé dans l'article de référence et on obtient les résultats par la commande *results*. Nous les avons affichés, comme voulu, en logarithme de base 2.

Puis, les gènes impliqués dans la traduction ont également été récupérés pour pouvoir être pointés sur le graphique de résultats. Pour cela, nous nous sommes référés au lien directement mentionné dans l'article menant au site de KEGG . Nous avons alors récupéré les noms des gènes impliqués dans la traduction, notamment ceux en lien avec le ribosome et la biosynthèse de l'aminoacyl ARNt. Nous avons ensuite créé un nouveau tableau de données ne contenant que les données des gènes liés à la traduction (Gene_Names_1col_-true.csv). On passe de 2 975 gènes étudiés à l'étude de 154 gènes seulement.

Parmi ces 154 gènes, il y en a 6 qui sont pointés dans la figure de l'article. Ils sont identifiés par des noms de code : infA, infB, infC, pth, frr, tsf. La page web du génome de *S.aureus NCTC8325* indique que NCTC8325 est la sous-espèce de *S.aureus* utilisée dans l'article. A partir de cette page, nous avons pu récupérer les noms des gènes associés à ces codes. Ces gènes semblaient impliqués dans la traduction (notamment des facteurs d'initiation de la traduction) mais ne faisaient pas partie des gènes présents dans le fichier des gènes impliqués dans la traduction déjà téléchargé. Nous les avons donc ajoutés à la liste des noms des gènes.

2.3.5 Implémentation du workflow Snakemake

S'assurer qu'une expérience est reproductible est une des premières étapes pour prouver sa validité scientifique. Ce principe naturel lorsqu'on pense à des expériences de biologie, s'applique également à l'informatique. En effet, un processus informatique n'est pas si facilement reproductible : les versions des données, le protocole d'analyse, l'environnement (outils, librairies, systèmes d'exploitation aux versions plus ou moins récentes), le support d'exécution (cluster, cloud, local), sont autant d'obstacles à la reproductibilité d'un protocole informatique. Pour répondre à ce défi, des systèmes de gestion de workflow ont été développés. Ceux-ci permettent de décrire de manière explicite les étapes et les outils utilisés dans l'analyse. Dans notre projet, nous avons utilisé le système de gestion de workflow Snakemake. Bien qu'il en existe d'autres, celui-ci présente l'avantage d'être moderne et relativement facile à prendre en main. Son principe est simple : des règles exprimant des étapes de l'analyse sont définies dans un fichier Snakefile. Ces règles sont définies à l'aide de mots clés, tels que 'input', 'output', 'message', 'container', 'shell', 'script', 'log', etc. Ces mots clés confèrent une certaine reproductibilité au workflow :

- les 'input' indiquent le ou les fichiers d'entrée à la règle ;
- les 'output' indiquent naturellement les fichiers de sortie ;
- 'container' indique le container à utiliser pour la règle. Comme discuté précédemment, c'est un élément clé à la reproductibilité du workflow.
- le 'shell' ou le 'script' indique la ou les commandes qui vont transformer les fichiers d'entrées en fichiers de sortie. Le 'shell' permet d'entrer une commande Unix tandis que le 'script' fait appel à un fichier script (Python, R) extérieur.
- D'autres mots clés, à l'instar de 'message' ou 'log' permettent une meilleure lisibilité pour l'utilisateur.

L'exécution du fichier Snakefile lancera l'exécution des règles nécessaires à la création des fichiers définis. Un autre avantage des systèmes de gestion de workflow est leur utilisation dans la parallélisation des tâches en utilisant la capacité de nombreux outils de bioinformatique à utiliser simultanément plusieurs CPU. Cette parallélisation est permise grâce à la définition de wildcards (voir ci-dessous).

Enfin, il est possible de généraliser un worflow de différentes manières.

Premièrement, l'utilisation de wildcards, des variables définies avec des accolades, permettent de ne pas avoir à préciser un à un les fichiers nécessaires à la création du fichier final souhaité. Le système « remontera » les règles pour définir les fichiers dont la création est nécessaire. Toutefois, ce point représente une difficulté dans la compréhension du fonctionnement d'un worflow Snakemake : Snakemake fonctionne « à l'envers ». Si rien n'est spécifié lors de son exécution, il choisit par défaut d'obtenir les fichiers demandés dans la première règle. C'est celle qu'on appelle en général 'rule all' et qui représente les fichiers cibles à obtenir. De cette règle, il va fouiller les règles pour connaître lesquelles sont

à lancer pour obtenir ces cibles, puis itérativement pour obtenir les fichiers intermédiaires jusqu'à arriver aux premiers fichiers de données d'entrée.

Deuxièmement, l'utilisation d'un fichier de configuration va permettre d'épurer le workflow au maximum de tout ce qui pourrait être spécifique d'une expérience. Un fichier 'config.yaml' permettra par exemple de renseigner les codes SRA des séquences à télécharger. Celui-ci pourra ensuite être appelé par la commande configfile : « config.yaml ». Ainsi, si l'on souhaite réaliser la même expérience à l'aide de différentes séquences, il suffira de changer le fichier de configuration sans modifier le workflow.

Dans notre workflow (Figure 3), nous avons commencé par définir les règles dans l'ordre d'exécution :

- La première règle, la rule downloading, permet le téléchargement des six fichiers échantillons d'intérêt (les 3 échantillons du groupe contrôle et les 3 persistants). Ceux-ci sont placés dans un dossier data.
- La deuxième règle, rule trimming, réalise une première étape de nettoyage sur les échantillons. En utilisant les fichiers du dossier data, elle enlève les séquences trop courtes ou de mauvaise qualité, et place les nouveaux fichiers nettoyés dans un dossier data_trim.
- L'étape suivante de l'analyse consiste à cartographier les échantillons. Pour cela, il faut d'abord disposer du génome de référence et l'indexer. La troisième règle, rule genome, sert alors au téléchargement du génome de référence à utiliser lorsqu'il faudra aligner les séquences d'échantillons. Le lien de téléchargement a été informé dans le fichier de configuration pour ne pas "polluer" le workflow.
- La quatrième règle, rule indexing, sert à indexer le génome de référence. Cette indexation conduit à la création de 6 fichiers ebwt rangés dans un dossier index.
- La règle de cartographie arrive enfin mais se heurte à un problème technique : nous ne disposons pas d'un unique container contenant les outils *bowtie* et *samtools* mais de deux containers ayant chacun un de ces outils, comme mentionné précédemment. Nous avons donc divisé la rule mapping en 3 rules :
  - Samming qui utilise l'outil *bowtie* et crée des fichiers sam dans un répertoire data_sam à partir des fichiers trim grâce à la commande : *bowtie -x -s index/indexation input.ech -S output*
  - Bamming qui utilise *samtools* et crée des fichiers bam à partir des fichiers sam dans un répertoire data_bam grâce à la commande : *samtools sort -O bam -o output input*
  - Mapping qui utilise samtools et crée les fichiers bam.bai

- Ensuite, pour le comptage des motifs sur les 6 échantillons à notre disposition, il nous faut d'abord disposer d'un fichier annotant le génome de référence. Comme pour la règle génome, la règle annotation_gen va nous fournir ce fichier

d'annotation en le téléchargeant à partir d'un lien (indiqué dans le fichier de configuration). La règle counting va agréger les connaissances précédemment acquises en un fichier counts.txt dans le répertoire scripts/genes qui utilise le container feature_counts.

— A présent, nous pouvons utiliser ces données pour les représenter graphiquement. L'objectif était de produire des MA-plots de ces données, notamment pour les gènes de la traduction. Pour cela, nous avons donc besoin de connaitre ces gènes impliqués dans la traduction. On utilise alors le fichier htext disponible sur le site KEGG. Toutefois, celui – ci contient tous les gènes de *S.aureus*. Pour sélectionner les gènes de la transcription, nous avons rédigé un script download_translation_genes.sh qui effectue plusieurs opérations sur les lignes du fichier jusqu'à obtenir une unique colonne intitulée *Name* contenant le nom SAOUHSC_* des gènes impliqués dans la transcription. Ce script est appelé au travers de la règle download_genes et permet la création du fichier scripts/-genes/translation_genes.csv.

— Enfin, pour l'analyse statistique et la visualisation des résultats, nous avons rencontré et surmonté quelques difficultés dues aux containers, comme mentionné précédemment :
  — Compatibilité des versions entre R et DESeq2
  — Utilisation d'images déjà construites et disponibles sur Dockerhub, une pour DESeq et une autre pour factoextra. Par conséquent, nous avons divisé la règle en deux :
    — Une première règle pour les MA-plots et les volcano-plots utilisant l'image de *pegi3s/r_deseq2 :1.32.0_v2*.
    — Une seconde règle permettant la réalisation de l'ACP à partir de l'image de *kamirab/r_deseq_pca :latest*.

  Ces deux rules appellent respectivement leur script, nommés script.R et script_ACP.R, et produisent en sortie nos figures dans le répertoire reports.

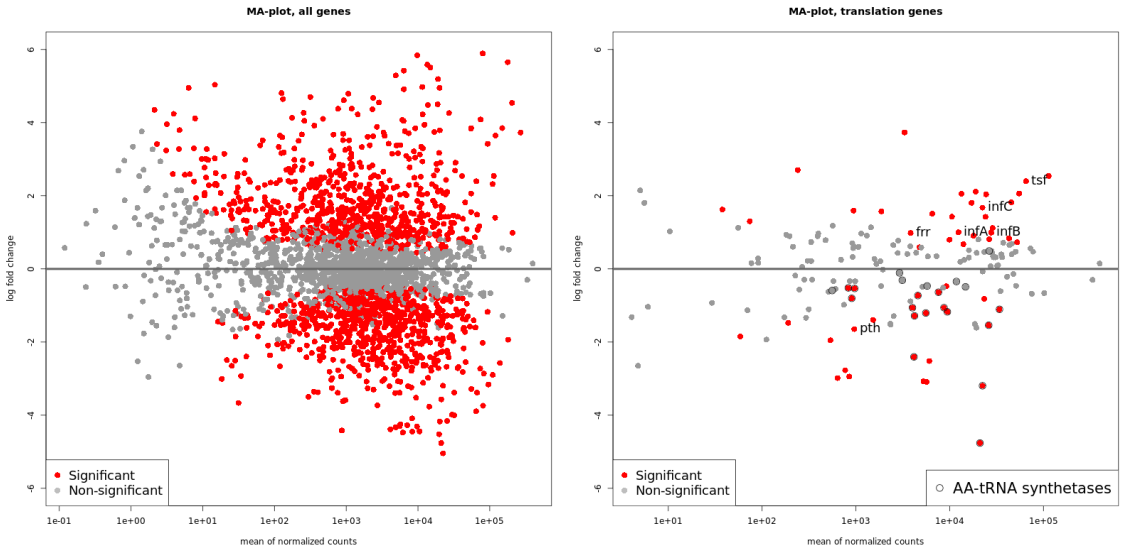
— La dernière règle, la règle all, indique ces fichiers en entrée et va lancer tout le script (selon l'explication précédente).

FIGURE 3 – Représentation de l'enchaînement des règles du Snakefile

3 Résultats

3.1 Présentation de nos résultats

A la suite de l'analyse statistique, nous avons obtenu les deux graphiques suivants :



(a) MA-plot, tous les gènes (b) MA-plot des gènes de la traduction

FIGURE 4 – MA-plots

3.2 Comparaison avec les résultats de l'article

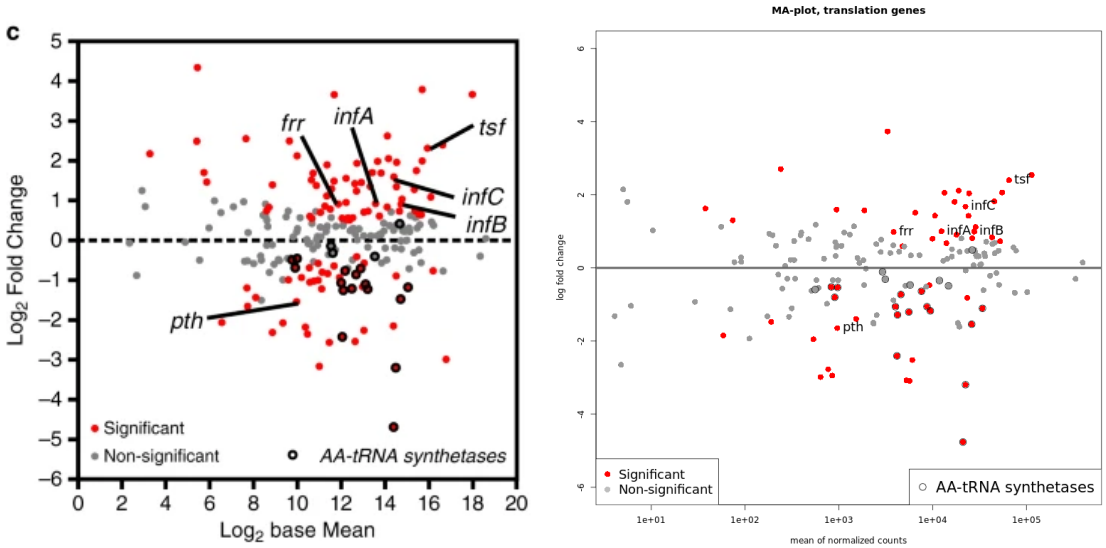
En comparant nos résultats et ceux de l'étude (Figure 5), on remarque d'abord que l'on retrouve bien les positions des six gènes annotés. Inf A, B et C se situent en haut à droite de manière significative en dessous de tsf. frr se situe bien à gauche de Inf A et phr est bien le seul des six gènes à être le moins exprimé par rapport à la situation contrôle. Par ailleurs, on retrouve également bien le même ordre de grandeur de gènes liés à l'aminoacyl ARNt synthétase que ceux de l'article.

Bien que les deux figures présentent des similarités, elles possèdent aussi des différences :

— Notre figure semble contenir plus de gènes avec de faibles expressions (à gauche sur le graphique).

— Certains gènes possèdent certes des valeurs (log2(BaseMean),log2FoldChange) dans les mêmes ordres de grandeur que dans l'article mais d'autres gènes ont des valeurs différentes ou sont absents (cas des deux gènes colorés en rouge au dessus de tsf). Par exemple, on observe que les gènes possédant un Log2FoldChange supérieur à 3 dans l'article ont, ici, ont une valeur de Log2FoldChange plus faible ou ne sont pas représentés.

Finalement, nos résultats sont, dans l'aspect, relativement proches de ceux de l'article, malgré les différences mentionnées ci-dessus.



(a) MA-plot de l'article (b) MA-plot, généré par le pipeline

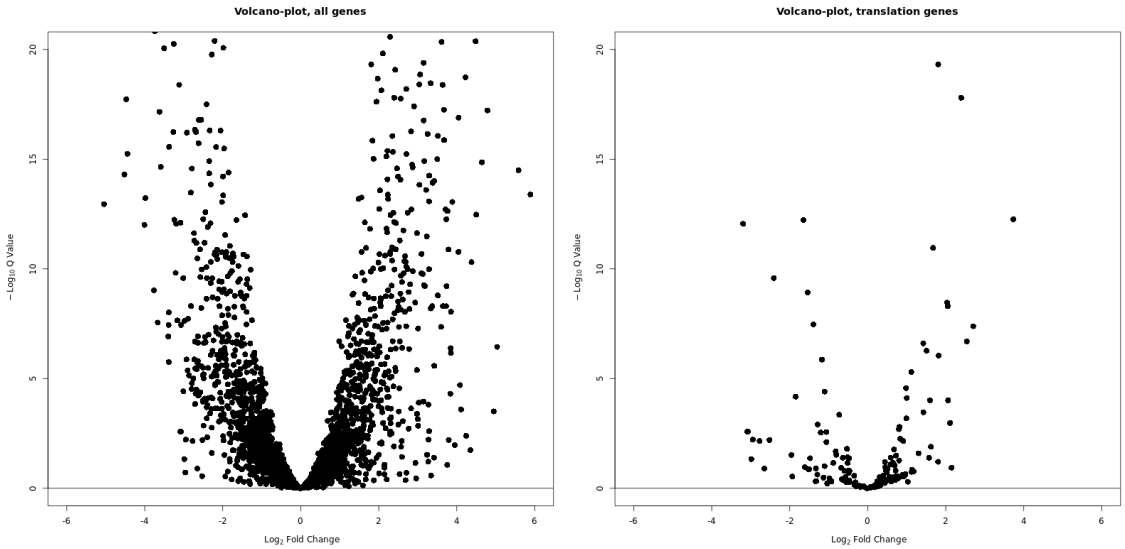
FIGURE 5 – Comparaison des résultats
Log2 Fold Change en fonction de la moyenne des expressions. Les gènes frr, pth, tsf, infA, infB et infC ont été mis en évidence par des étiquettes de données et sont encerclés en noir.

4 Discussion

4.1 Interprétation des résultats

Pour comprendre davantage nos résultats et en vérifier la cohérence, nous avons effectué un volcano plot (Figure 6) et une ACP (Figure 7).

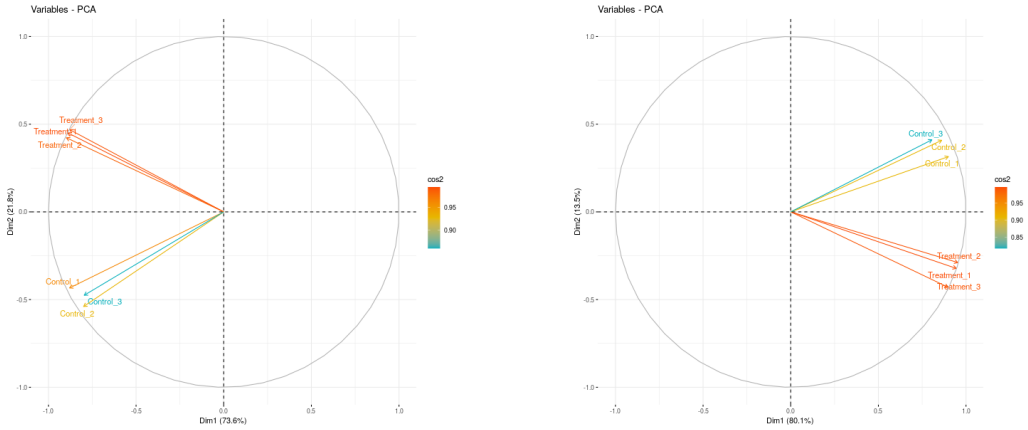
Le volcano-plot confirme que la majorité des points peu significatifs (inférieurs à -log(P-value)=1.3) sont regroupés autour de valeurs de Log2FoldChange proches de 0. On observe également une différence d'expression entre les échantillons persistants et les contrôles. En effet, on constate qu'un certain nombre de gènes sont exprimés 1.5 à 4 fois plus chez la forme persistante de *S.aureus* que dans la situation contrôle. Toutefois, certains gènes sont également moins exprimés. Nous avons aussi dû retirer un point car il avait une significativité de plus de 40 pour un log2foldchange de -4, ayant alors pour conséquence d'écraser le reste des données. Plus généralement, on observe bien que l'état persistant de *S.aureus* a une régulation de ses gènes très différente de celle de l'état contrôle.



(a) Volcano-plot, tous les gènes (b) Volcano-plot des gènes de la traduction

FIGURE 6 – Volcano-plots

Enfin, pour vérifier la cohérence de nos résultats, nous avons réalisé une ACP. On s'attend en effet à distinguer deux groupes distincts sur le graphique : l'un pour la population contrôle et l'autre pour la population de persitants de *S.aureus*. C'est d'ailleurs ce que l'on obtient sur la Figure 7 ci-dessous où l'on observe une différence selon l'état de *S.aureus* (groupe contrôle ou *persisters*).



(a) ACP, tous les gènes (b) ACP, gènes de la traduction

FIGURE 7 – ACP
Sur chacun des deux graphiques, on distingue bien deux groupes : les trois échantillons du groupe contrôle et les trois échantillons du groupe de *persisters* de *S.aureus*.

4.2 Recommandations

Pour améliorer la reproductibilité d'une expérience, nous pouvons formuler les recommandations générales suivantes afin d'en renforcer la robustesse et d'offrir une base solide pour la collaboration et la validation scientifique :

- Adopter une stratégie de versionnement cohérente en utilisant des systèmes tels que Git et GitHub et documenter les changements apportés à chaque itération grâce à l'utilisation des messages au moment des commits. Cela garantira une plus grande traçabilité des modifications et facilitera la collaboration.
- Exploiter le système de containers Docker pour reproduire plus facilement l'environnement logiciel. Veillez à ce que toutes les dépendances nécessaires soient bien listées et que le Dockerfile soit bien commenté et documenté. Pour simplifier la distribution, on recommande d'envoyer les images Docker construites sur Dockerhub. Afin de s'assurer que les containers contiennent bien les outils nécessaires, on peut également utiliser la commande *docker run* après *docker build* et vérifier la présence de la commande en question dans le chemin indiqué dans le Dockerfile. On s'assurera aussi de respecter au maximum les versions utilisées par l'étude afin de reproduire le plus fidèlement possible l'environnement logiciel.
- Utiliser Snakemake de manière efficace pour gérer les dépendances entre les différentes étapes du pipeline et veiller à ce que le fichier Snakefile soit organisé de manière modulaire et compréhensible, afin de faciliter l'adaptation à de nouvelles analyses ou à des jeux de données différents.
- Archiver des données intermédiaires produites à chacune des étapes du pipeline, de sorte à ne pas avoir à l'exécuter à nouveau dans son entièreté, ce qui est très

souvent chronophage et énergivore. Cela nous a notamment permis de reprendre le Snakefile à partir de l'étape souhaitée sans avoir à tout relancer.

— Créer un fichier de configuration pour le pipeline, de sorte que s'il y a des modifications sur les données initiales (échantillons notamment), l'utilisateur n'ait pas à modifier directement le Snakefile mais plutôt le fichier de configuration. Cela permet également de réutiliser plus aisément le Snakefile si l'on veut, par exemple, reproduire la même étude mais avec des échantillons différents.

— Nous ne l'avons pas réalisé au cours de notre projet, mais on aurait également pu intégrer des tests automatisés au pipeline pour vérifier la validité des résultats à chaque étape. Cela peut être réalisé avec des outils comme pytest. Ces tests réguliers nous auraient en effet permis de vérifier que le pipeline fonctionne correctement avec les mises à jour et modifications apportées.

— Enfin, on aurait aussi pu intégrer des contrôles de qualité des données à différentes étapes du pipeline pour identifier et gérer les éventuelles anomalies dans les données brutes, améliorant ainsi la fiabilité des résultats finaux. Cela aurait aussi pu permettre d'identifier dès lors les différences de résultats entre ceux que nous avons obtenus et ceux de l'étude.

5 Conclusion

Ce projet de reprohackathon nous a permis de développer et de mettre en œuvre un large éventail de compétences, tant scientifiques, techniques que managériales. D'une part, la technicité du sujet a été l'un des premiers défis auxquels nous avons été confrontés, notamment en raison de la diversité de nos parcours desquels la bioinformatique, et plus généralement l'informatique, étaient quasi, voire totalement absents. Se familiariser rapidement avec les outils nécessaires fut un réel challenge que nous avons tout de même su relever. Ce fut aussi l'occasion de célébrer, avec humilité bien sûr, les petites victoires telles qu'un script fonctionnel, l'export des images des containers sur Dockerhub ou encore, la visualisation des résultats finaux après exécution du Snakefile...

D'autre part, ce projet a également mobilisé de nombreuses capacités managériales, sous-estimées au début du projet, à tort. En effet, nous avons dans un premier temps pâti d'un manque de communication et de planification claires, qui ont engendré une répartition inégale des tâches ainsi qu'un manque d'efficacité. Forts de ce constat, nous avons donc entrepris une gestion de projet plus appropriée et instauré des réunions hebdomadaires en présentiel pour pallier la communication inefficiente par messages. L'instauration d'un calendrier de suivi nous a également permis de remplir les objectifs et de répartir les rôles plus uniformément.

Enfin, ce projet a été l'opportunité pour nous de prendre pleinement conscience des enjeux de la reproductibilité des expériences, notamment en bioinformatique où les difficultés comme la reproduction de l'environnement logiciel, les outils alternatifs, le choix des paramètres, l'influence de l'environnement d'exécution sur l'orchestration des outils entre eux, peuvent être nombreuses et engendrer une variabilité des résultats.

Bien que nous n'ayons pas tous des perspectives de développement dans la recherche académique, ce projet porte en lui-même une leçon forte sur l'importance d'expliciter clairement et précisément sa démarche expérimentale et sa méthodologie, sans quoi les résultats obtenus perdent en admissibilité et en reconnaissance par la communauté scientifique et les pairs. Finalement, en tant que futurs ingénieurs, ce sont des compétences dont nous devons faire preuve au quotidien, peu importe le secteur ou le domaine d'application.