Projet ReproHackathon

Alexandre ABHAY, Agathénaïs ADIGUNA,

Soundous BELLABACI & Shanti PIJEAUD

https://github.com/Akooph/HackathonRepro2021

29 novembre 2021

Table des matières

1. Introduction

1.A. La reproductibilité

Pour toute publication scientifique, la reproductibilité des résultats prend une place de plus en plus importante dans la qualité des conclusions posées. Une étude faite par Baker M et publiée dans Nature en 2016 a montré que sur 1576 chercheurs interrogés, 70% d'entre eux ont au moins une fois été incapables de reproduire les résultats d'un article scientifique. Et plus de 50% des chercheurs interrogés ont admis avoir déjà été incapables de reproduire leurs propres résultats.

L'article explique que le manque d'investissement des chercheurs dans la reproductibilité de leurs résultats est dû au temps et aux coûts associés à la répétition des expériences. Sachant que les résultats de ces expériences supplémentaires n'apportent souvent aucun pouvoir statistique aux résultats initiaux. La pression académique est plus accentuée sur le fait de publier, plutôt que sur celui de publier des résultats reproductibles.

Par conséquent, de plus en plus de publications doivent être rétractées car l'impossibilité de reproduire leurs résultats indique souvent que ceux-ci sont erronés. L'investissement de temps et de coût dans la production de ces résultats est alors perdu. Ainsi que celui d'autres chercheurs essayant de reproduire ces résultats. Plusieurs articles traitant de ce problème l'ont défini comme celui de la « crise de la reproductibilité ».

Ceci souligne à la fois l'importance d'inclure une démarche de reproductibilité dans toute expérience, et l'importance de partager cette démarche pour pouvoir rendre ses résultats exploitables par toute la communauté scientifique. Bien que certains domaines soient plus affectés que d'autres comme les sciences sociales, la psychologie et la recherche clinique, il conviendrait d'établir des solutions permettant d'assurer la reproductibilité de tout résultat publié.

Les problèmes de reproductibilité peuvent être expliqués à différents niveaux : au niveau

technique en fonction des outils utilisés, au niveau humain en fonction de l'expertise et des biais de l'expérimentateur, au niveau du design de l'étude et des outils statistiques choisis, et au niveau des motivations et récompenses potentielles liées à la publication. Tous ces problèmes ne peuvent pas être corrigés par un protocole global. Mais les variabilités expérimentales, statistiques et computationnelles peuvent être éliminées en établissant un protocole reproductible.

Les expériences dites de « wet-lab » et le traitement informatique de données scientifiques ne vont pas avoir les mêmes protocoles de reproductibilité mais les mêmes concepts s'appliquent aux deux. Plusieurs niveaux de reproductibilité peuvent être ainsi définis (Cohen-Boulakia et al., FGCS, 2017) :

— La **répétition** : une expérience est répétée lorsqu'elle est à nouveau effectuée dans l'environnement initial. Cet environnement peut être le laboratoire où les manipulations sont faites, ou les paramètres computationnels utilisés. En s'assurant que la répétition d'une expérience est possible, tous les paramètres d'importance sont notés. Les résultats attendus à la suite d'une répétition sont identiques à ceux de l'expérience initiale.

— La **réplication** : une expérience est répliquée lorsqu'elle est effectuée dans un environnement différent. C'est un laboratoire différent ou un environnement computationnel différent. Les résultats peuvent différer tant que leurs interprétations ne changent pas. La réplication permet d'apprécier la robustesse des résultats ainsi que de la méthode.

— La **reproductibilité** : une expérience est reproduite lorsqu'elle peut être effectuée suivant un protocole différent et en vérifiant les mêmes hypothèses. Ce qui importe dans la reproductibilité est d'évaluer la qualité des résultats en les retrouvant avec des protocoles et des sets de données initiales différents.

En s'assurant de répondre à chacune des ces attentes lors de l'édification du protocole

expérimental, la reproductibilité des résultats devrait être assurée. Malgré le manque de consensus de la communauté scientifique sur ce qui cause la crise de la reproductibilité, suivre ces règles générales réduirait l'irreproductibilité de nombreux résultats.

Ainsi, prendre en compte la reproductibilité est une étape cruciale pour assurer la fiabilité de ses résultats. Cela permettrait à chaque publication scientifique d'avoir des conclusions plus robustes et une meilleure contribution à l'avancée de la recherche. Pour répondre aux besoins générés par la production de résultats reproductibles, l'utilisation d'outils et de workflows robustes est alors recommandée.

1.B. Objectifs du projet et choix des outils

L'objectif de ce projet est de reproduire une partie des analyses décrites dans les deux articles suivants :

— Harbour, J. W. *et al.* Recurrent mutations at codon 625 of the splicing factor SF3B1 in uveal melanoma. *Nat Genet* **45**, 133–135 (2013).

— Furney, S. J. *et al. SF3B1* Mutations Are Associated with Alternative Splicing in Uveal Melanoma. *Cancer Discovery* **3**, 1122–1129 (2013).

Ces articles s'intéressent aux gènes responsables des mélanomes uvéaux. Le premier article décrit les mutations du gène SF3B1 dans des cellules cancéreuses encore correctement différenciées avec un bon taux de survie des patients. Le deuxième article essaie d'analyser tous les gènes impliqués dans le mélanome uvéal, en exploitant les résultats du premier article. Ils montrent que le gène SF3B1 code un élément du splicéosome, ainsi une mutation sur ce gène engendre un épissage alternatif.

Notre but est d'analyser les données utilisées par ces deux publications pour trouver quels gènes sont différentiellement exprimés. Nous travaillons avec 8 sets de données RNASeq provenants

de 8 échantillons de tumeurs de patients présentant un mélanome uvéal. Parmi ces échantillons, 4 sont des échantillons témoins dits *wild-types* pour le gène SF3B1 et 4 sont des échantillons mutés sur le gène SF3B1.

Pour réaliser ce projet de manière reproductible, nous avons choisi d'utiliser les outils suivants : Snakemake, Singularity, Docker, GitHub et Rstudio.

Snakemake nous a permis de créer un workflow contenant des étapes appelées `rules` pour chaque étape de notre analyse. Ces étapes renvoient à un container (Docker ou Singularity) particulier lorsque cela est nécessaire. Ainsi nous avons toujours accès à la version des outils qui nous ont permis de faire l'analyse. Cela nous permet de garantir la reproductibilité de notre étude. Ce projet était notre première introduction à Snakemake et aux containers. Nous sommes aidés de la documentation fournie avec le sujet et lors des TP pour les prendre en main. Puis, nous nous sommes reposés sur la documentation officielle de Snakemake pour en apprendre plus.

Github nous a permis de partager les avancées de chacun sur le projet et de pouvoir consulter les différentes versions et changements effectués au besoin. Enfin, nous avons effectué l'analyse statistique des résultats obtenus à l'aide d'un script R et du package DESeq2.

1.C. La répartition du travail

Pour l'écriture du snakefile, nous nous sommes retrouvés tous ensembles à l'université pour travailler dessus. Il était plus pratique pour nous de faire un travail collectif car nous étions initialement tous débutants. Ainsi nous avons pu être confrontés à tous les problèmes et nous avons appris à les résoudre ensemble. Cela nous a permis d'assurer que nous avions tous le même niveau de compréhension.

Avant chaque consultation avec notre professeur, nous nous retrouvions en visio pour revenir sur ce que nous voulions poser comme question et partager d'éventuelles avancées personnelles. A la suite des réunions, nous faisions à nouveau un appel visio pour discuter de la direction que devrait prendre notre travail pour la semaine suivante.

Pour l'écriture du script R, nous avons commencé par y travailler individuellement puis nous avons mis en commun nos avancées.

Enfin, l'écriture du rapport et de la documentation associée à nos scripts a été divisée en quatre parts égales et chacun a choisi d'écrire la partie avec laquelle il se sentait le plus à l'aise.

2. Matériels et méthodes

2.A. Gestionnaires de containers et de workflow

2.A.1. Singularity version 3.8.4

Singularity permet de créer et d'utiliser des containers. Un container peut contenir : une configuration système, des outils, des packages etc... Tout ce contenu est **versionné**. Cela signifie qu'en utilisant des containers dans un workflow, et plus généralement dans une analyse, l'analyse s'effectue dans le container avec les informations disponibles dans celui-ci et **uniquement** ces informations. C'est un point essentiel pour la reproductibilité de l'analyse.

En clair, quelle que soit la machine utilisée, puisque tout est versionné dans le container, l'analyse s'effectuera de la même manière et produira les mêmes résultats. Le seul changement sera dans le temps d'exécution.

2.A.2. Snakemake version 6.11.1

Snakemake est un outil de gestion de workflow dans le cadre d'analyses reproductibles de données. C'est un outil qui facilite également l'utilisation du workflow d'une infrastructure à l'autre en utilisant 100% des ressources allouées grâce à ces capacités de parallélisations, par exemple en passant d'une machine personnelle à un cluster de calcul.

Lors de l'analyse de données et ici plus particulièrement dans le cadre d'analyse de données RNASeq, il convient de créer un protocole d'analyse. Lors de l'analyse on utilise des données en entrée et on s'attend à certains résultats en sortie. Les données transiteront par plusieurs étapes de traitement. Snakemake utilise un script Snakefile principalement en Python pour décrire les étapes de ce traitement. Ainsi, dans une section `rule all`, on explicite les données entrantes et les données sortantes du workflow d'analyse. Les différentes étapes de traitement sont quant-à-elles des modules appelés `rules` qui se structurent généralement de la manière suivante :

— `input` : Indique les fichiers qui vont être traités (non obligatoire)

— `output` : Indique les fichiers attendus en sortie de `rule`

— `shell` : Indique quel traitement en ligne de commandes va être effectué

L'`input` peut avoir plusieurs syntaxes :

— `expand function` : Génère une liste Python. Elle permet d'indiquer les fichiers pour une analyse et de les formater correctement et rapidement en une ligne.

— `wildcards` : Indique que le processus est à effectuer sur tous les fichiers contenus dans un répertoire. Indique également que le processus peut être effectué dès qu'un nouveau fichier apparaît dans le répertoire même si on y écrit d'autres fichiers. Permet donc de l'optimisation supplémentaire grâce au multithreading notamment.

Dans le cadre du projet ReproHackathon, d'autres options de Snakemake que celles ci-dessus ont été utilisées :

— `singularity` : Indique dans quel container on effectue le traitement

— `threads` : Indique le nombre de threads disponibles pour l'exécution de la `rule`.

— `resources` : Indique aux processus une limite de ressources à ne pas dépasser lors de l'exécution.

L'option `threads` est essentielle dans le cadre de ce type d'analyse car les clusters de calculs sont généralement multithreadés avec des fréquences processeurs assez faibles, ce qui est aussi de plus en plus le cas pour les machines personnelles. Ainsi, dans la théorie, le temps de traitement est divisé par le nombre de threads. L'option `resources` quant-à-elle permet de limiter les ressources qui vont être utilisées par certains processus. En effet, lors de l'utilisation des machines virtuelles mises à disposition, certains processus très optimisés étaient trop voraces en mémoire cache et excédaient les capacités des machines. Il a donc fallu indiquer une limite au détriment de la rapidité des processus mais garantissant le bon déroulement, sans bugs, de l'analyse quelle que soit la machine utilisée. Enfin `singularity` permet de spécifier l'environnement à utiliser, ici on utilise des images `Docker` uniquement.

2.B. Données

On utilise plusieurs types de données dans cette analyse :

— Données de RNASeq via leurs numéros d'accession : `SRR628582`, `SRR628583`, `SRR628584`, `SRR628585`, `SRR628586`, `SRR628587`, `SRR628588` et `SRR628589` au format `.SRA`

— Génome humain de référence par chromosome (tous) au format `.FASTA` sur Ensembl

— Annotation du génome humain de référence par chromosome (tous) au format `.GTF` également sur Ensembl

2.C. Pipeline

On résume le pipeline d'analyse dans la Figure 1. Pour plus de détails sur les `input` et `output`, se référer au fichier `Snakefile` directement disponible sur GitHub.

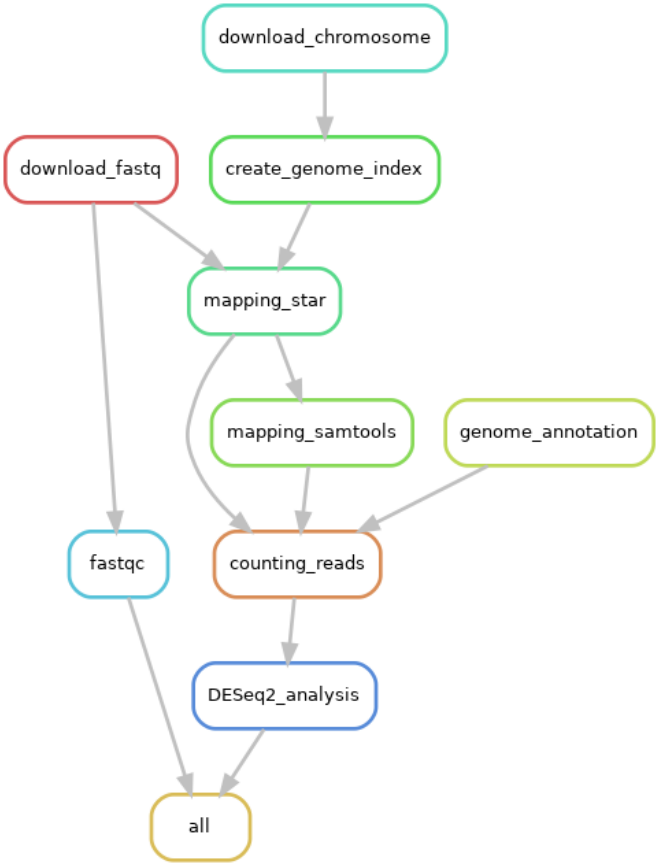

FIGURE 1 – *Pipeline issu de Snakemake*

2.D. Détail des rules

2.D.1. `download_fastq` : Téléchargement des fichiers `.fastq`

L'image `Docker` de `SRAtoolkit version 2.10.0` est utilisée, elle contient la fonction `fasterq-dump` qui permet de télécharger les données de RNASeq des 8 échantillons rapidement via leurs numéros d'accession. Cet outil produit les `output SRRXXXXX1_1.fastq` et `SRRXXXXX1_2.fastq`.

2.D.2. `download_chromosome` : Téléchargement du génome de référence

Elle permet d'effectuer un contrôle qualité des séquences ARN qui ont été téléchargées dans la `rule download_fastq`. L'image `Docker FastQC version 0.11.9` est utilisée pour cette `rule`. Les fichier `output` résultants sont placés dans le dossier `fastqc_files`.

2.D.3. `create_genome_index` : Indexation du génome de référence

Ici, on utilise le `RunMode genomeGenerate` de l'outil `STAR version 2.7.6a` pour indexer le génome téléchargé dans la `rule download_genome`. Cette indexation prépare le génome de référence à l'alignement par STAR ce qui optimisera la suite du traitement des données. On obtient donc en sortie un dossier contenant le génome indexé.

2.D.4. `genome_annotation` : Téléchargement des annotations

Dans cette `rule`, on utilise simplement la fonction `wget` pour télécharger le génome annoté sur Ensembl par le protocole ftp à l'adresse suivante: ftp://ftp.ensembl.org/pub/release-101/gtf/homo_sapiens/Homo_sapiens.GRCh38.101.chr.gtf.gz. On obtient donc en sortie un fichier `annotated_genome.gtf` contenant l'annotation qui sera utilisé plus tard dans la `rule counting_reads`.

2.D.5. `mapping_star` : Mapping des `reads` sur le génome de référence

On utilise la fonction d'alignement de `STAR version 2.7.6a`. Elle prend en entrée le génome de référence indexé issu de la `rule create_genome_index` ainsi que tous les fichiers

SRRXXXXX1_1.fastq et SRRXXXXX1_2.fastq pour chaque échantillon, téléchargés à la première règle, la rule download_fastq et aligne tous les reads avec le génome de référence indexé contenu dans le répertoire SAindex obtenu à la rule create_genome_index. On obtient en sortie un fichier .bam par échantillon qui est un fichier contenant ces alignements, trié par coordonnées des indexes.

2.D.6. mapping_samtools : Indexation de l'alignement des reads sur le génome de référence

On utilise la fontion index de samtools version 1.11. Dans cette rule, on prend le fichier d'alignement des reads sur le génome de référence .bam issu de la rule mapping_star pour l'indexer. On obtient ainsi un fichier .bam.bai d'alignement indexé en sortie.

2.D.7. counting_reads : Comptage des reads en fonction de l'alignement

On utilise la fonction featureCounts de SubRead version 2.0.1. Dans cette rule, on prend le fichier d'alignement .bam sortant de la rule mapping_star, le fichier d'alignement indexé .bam.bai sortant de la rule mapping_samtools et enfin le génome annoté .gtf téléchargé dans la rule download_chromosome afin d'attribuer et de compter chaque read à un gène. On obtient en sortie un fichier .counts contenant ces informations pour l'analyse différentielle.

2.D.8. DESeq2_analysis : Analyse différentielle des données de comptage

Dans cette rule, on utilise DESeq2 version 1.28.1 et r-extended version 4.0.2 pour intégrer le script R script.R dans le workflow d'analyse. Il prend en input les fichiers .counts issus de la rule counting_reads et donne en sortie un fichier image.RData contenant les résultats d'analyse.

2.E. Analyse statistiques des résultats avec DESeq2

Parmi les 8 échantillons disponibles de tumeurs uvéales, dans la publication de Harbour et al, 3 ont été identifiés comme présentant une mutation du gène SF3B1, ce sont les échantillons *mutated* dont les identifiants sont : SRR628582, SRR628583 et SRR628584. Les 5 autres ne présentent pas

cette mutation, ce sont les *wildtype*.

L'équipe de Furney et al a démontré que l'échantillon d'identifiant `SRR628589`, précédemment identifié comme *wildtype*, présente également une mutation du gène SF3B1. Il a donc été classifié comme un échantillons *mutated* et non *wildtype*, et notre étude a pris en compte ce changement. Nous avons donc été en présence de 4 échantillons *mutated* et 4 échantillons *wildtype*.

Les données de comptage ont été rassemblées en un `data frame`, et transformées en matrice de comptage. Cette matrice contient les `reads` de 60 612 gènes, pour les 8 échantillons. Les gènes ne présentant aucun `read` dans toute la matrice de comptage ont été retirés, les dimensions finales de la matrice de comptage sont alors 30 029 gènes pour 8 échantillons, dont 4 sont classés comme *wildtype* et 4 comme *mutated*.

3. Résultats

3.A. Workflow : Exécution et temps d'exécution

Le workflow s'exécute en 3h avec une machine virtuelle de 16 cœurs et 64Go de RAM, et en 2h20 avec 24 cœurs et 96Go de RAM.

Le but de l'étude est de tenter de reproduire l'analyse des publications de Harbour et al et Furney et al, donc le workflow n'est fait que pour ces données. Il n'a pas été rendu généralisable à d'autres données RNASeq, renseignées par l'utilisateur.

A l'issu de l'exécution du workflow, 3 dossiers sont produits :

1. `fastqc_files` : contenant les liens html de l'analyse qualitative des fichiers `.fastq` téléchargés en première étape du workflow. Cette étape n'a pas d'incidence sur le reste du pipeline, elle permet uniquement de vérifier la qualité, a posteriori, des fichiers utilisés.
2. `counts_files` : contenant les fichiers de décompte des `reads` par gène pour chaque échantillon
3. `plots` : les graphiques générés lors de l'analyse de l'expression différentielle des gènes depuis les fichiers counts obtenus avec l'outil `subread`

Un fichier `image.RData` est également généré dans le dossier de travail. Il permet de récupérer les différentes données utiles à l'analyse statistique telles que la matrice de comptage `matrix_df` et l'objet `res`, résultats de `DESeq2`.

3.B. Résultats biologiques

3.B.1. Analyse en composantes principales

Nous avons d'abord normalisé les données de comptage, par la fonction `varianceStabilizing Transformation()` de `DESeq2`, pour les rendre les échantillons comparables. Ensuite, nous réalisons une Analyse en Composantes Principales avec la fonction `plotPCA()` de `DESeq2` dont voici le graphique :

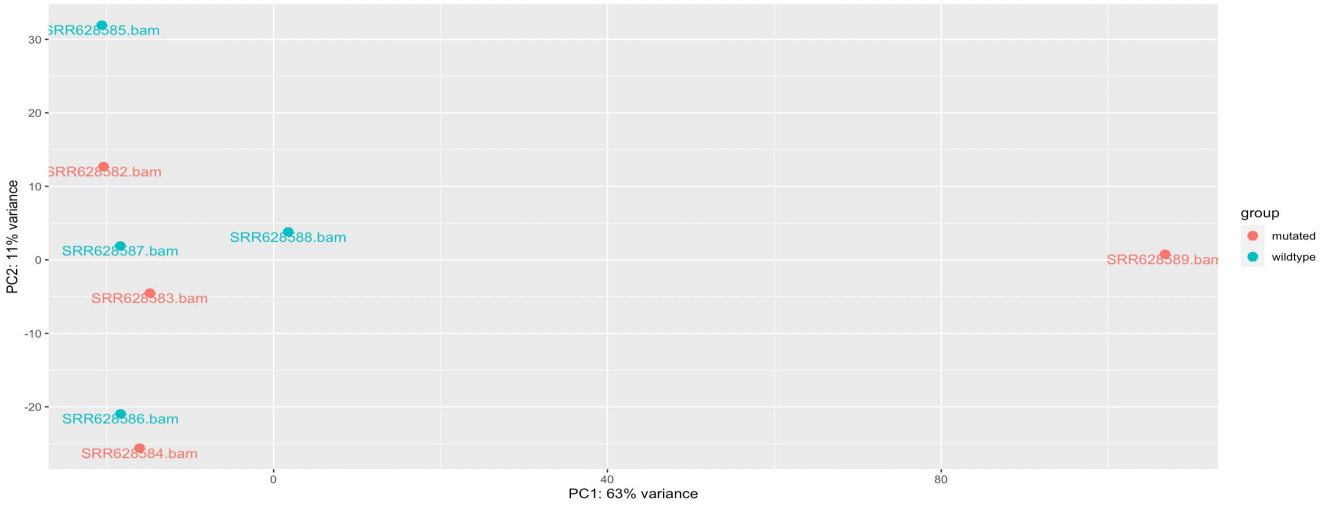

FIGURE 2 – *ACP des données de comptage post-normalisation*

Dans les deux composantes principales, les échantillons ne sont pas regroupés selon qu'ils soient *mutated* ou *wildtype*. On remarque également que l'échantillon `SRR628589`, qui a été ajouté par l'équipe Furney et al dans les échantillons *mutated* peut être considéré comme étant éloigné de tous les autres échantillons. Un tel graphe d'Analyse en Composantes Principales, ne dévoilant pas de cluster peut indiquer que les différences entre groupes sont légères.

3.B.2. Analyse différentielle de l'expression des gènes

La fonction `plotMA()` de `DESeq2` permet d'afficher les `log2foldchange` des différents gènes à l'issue de l'analyse différentielle, par rapport à la moyenne des comptes normalisés pour tous les échantillons. Sur ce graphique, on retrouve en bleu les 260 gènes différentiellement exprimés et en gris, les autres gènes. On peut observer que les gènes différentiellement exprimés sont plus nombreux à être surexprimés dans les échantillons *mutated* (`log2foldchange` négatif).

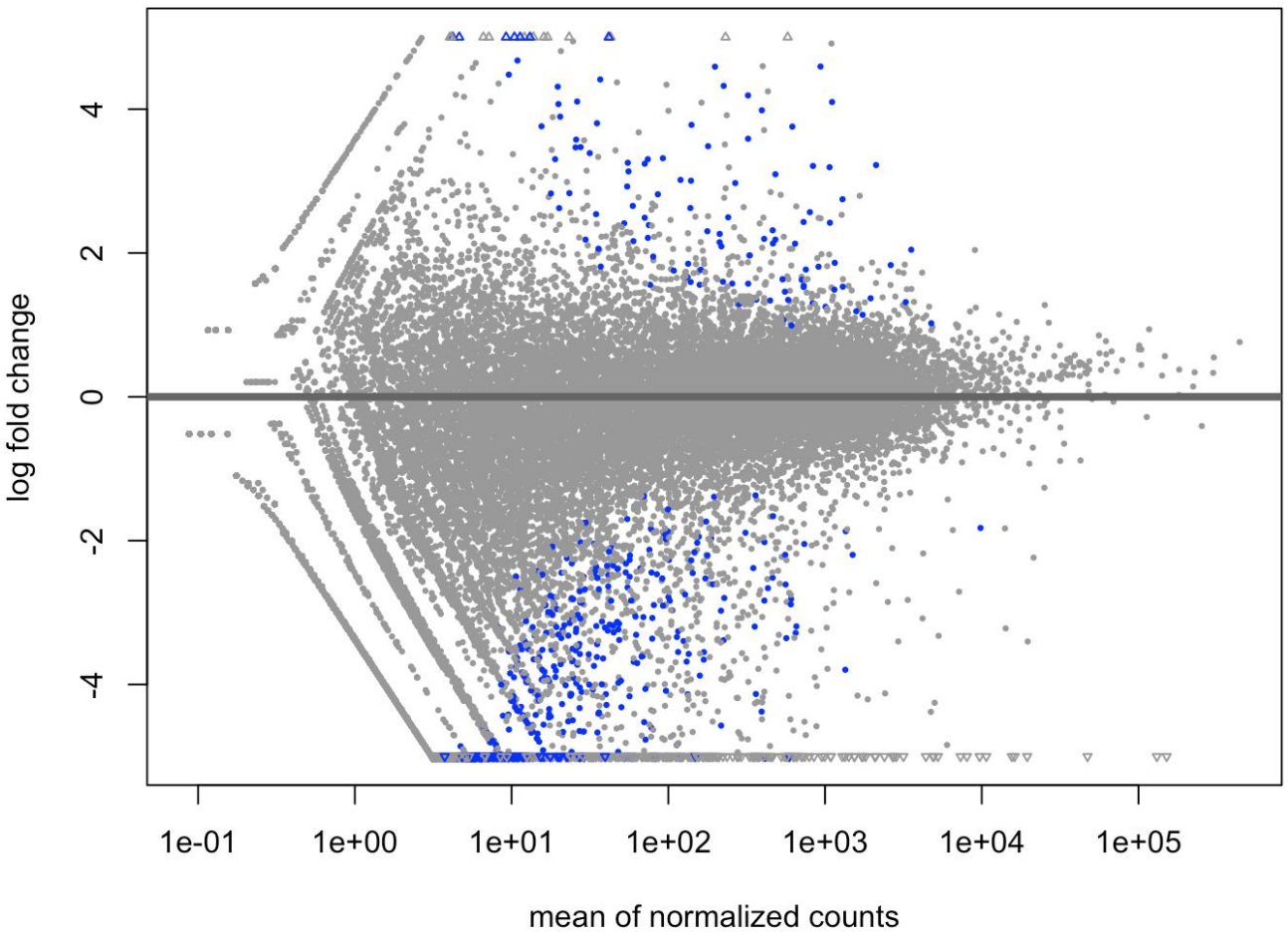

FIGURE 3 – *log2foldchange des gènes par rapport à la moyenne des comptes normalisés pour tous les échantillons. Bleu: Gènes différentiellement exprimés. Gris: Gènes non différentiellement exprimés*

Le gène le plus significativement différentiellement exprimé, c'est-à-dire présentant le taux minimum de fausse découverte (FDR), correspondant à la p-value ajustée (padj minimum) a pour identifiant Ensembl `ENSG00000144824`, qui correspond au gène PHLDB2 (Fig. 4). `DESeq2` permet de représenter le nombre de `reads` normalisé pour un seul gène à travers les différents groupes, par la fonction `plotCounts()`. Ce graphique nous permet d'observer visuellement que ce gène présente une différence d'expression entre les échantillons *mutated* et les *wildtype*, avec une sous-expression dans les échantillons *mutated*.

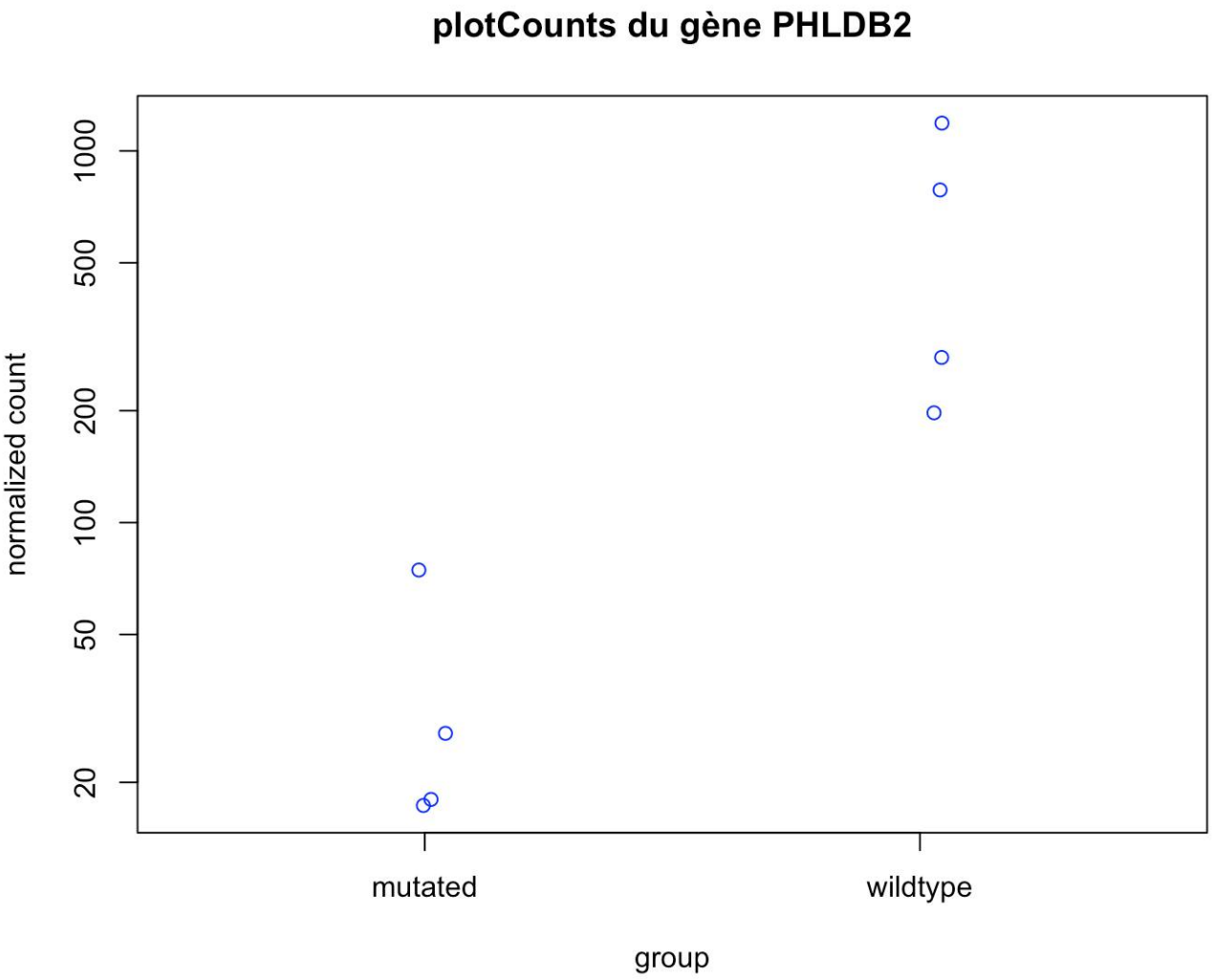

FIGURE 4 – *Nombre de reads normalisé pour PHLDB2*

3.B.3. Comparaison des gènes différentiellement exprimés avec la publication Harbour et al

Après analyse par `DESeq2`, il y a 260 gènes significativement différentiellement exprimés.. Ce nombre diffère par rapport à celui rapporté par Harbour et al, qui n'avaient, en effet, rapporté que 10 gènes différentiellement exprimés par la méthode `SAM - Significance Analysis of MicroArrays`, avec également un seuil de FDR à 0.05 (sur différents échantillons car il s'agissait de 5 SF2B1-mutant et de 6 SF3B1-wildtype, ce qui ne correspond pas aux échantillons RNAseq

rendus disponibles).

Parmi les 10 gènes différentiellement exprimés présentant les p-adjust les plus faibles dans notre analyse, que nous avons présentés dans la Fig. 5, aucun n'appartient aux 10 gènes différentiellement exprimés identifiés par Harbour et al. L'analyse lancée par `DESeq2` étant *wildtype vs mutated*, un `log2foldchange` positif correspond à une surexpression dans les échantillons *wildtype*, par rapport aux échantillons *mutated*. Nos gènes les plus différentiellement exprimés sont pour la plupart des proto-oncogènes. Ils sont soit surexprimés dans les échantillons *wildtype* (`log2foldchange` positif), donc sous-exprimés dans les échantillons *mutated*. Ou inversement pour les gènes présentant un `log2foldchange` négatif.

Identifiant Ensembl	Nom du gène	Description du gène	Fonction moléculaire (UniProt)	Log2foldchange
ENSG00000144824	PHLDB2	Pleckstrin Homology Like Domain Family B Member 2	enables protein binding	4.19087
ENSG00000101977	2 MCF	MCF.2 cell line derived transforming sequence	Guanine-nucleotide releasing factor	4.59235
ENSG00000198795	ZNF521	Zinc finger protein 521	DNA-binding / Transcription, Transcription regulation	3.58823
ENSG00000169548	ZNF280A	Zinc finger protein 280A	DNA-binding / Transcription, Transcription regulation	-8.66861
ENSG00000163017	ACTG2	Actin gamma 2, smooth muscle	Muscle protein	-4.76764
ENSG00000250337	PURPL	P53 Upregulated Regulator Of P53 Levels	N/A	-4.13612
ENSG00000284377	chrXq27	N/A	N/A	-5.62210
ENSG00000182632	CCNYL2	Cyclin Y Like 2 (Pseudogene)	enables cyclin-dependent protein serine/threonine kinase regulator activity	-5.78413
ENSG00000226145	KRT16P	Keratin 16 Pseudogene 6	N/A	-4.52341
ENSG00000125816	NKX2-4	NK2 homeobox 4	DNA-binding / probable facteur de transcription	7.57969

FIGURE 5 – *Tableau présentant les log2foldchange des 10 gènes différentiellement exprimés identifiés par Harbour et al.*

4. Conclusion

Le projet ReproHackathon consistait en la reproduction de certaines parties de l'analyse décrite dans deux articles scientifiques. Il s'agissait alors de mettre en place un workflow d'analyse RNAseq afin de mettre en évidence les gènes différentiellement exprimés c'est-à-dire les gènes qui sont plus ou moins exprimés dans la condition où SF3B1 est muté comparé la condition où SF3B1 n'est pas muté. Ce workflow comprend différentes étapes : le téléchargement des données, la transformation des données au format fastQ, le mapping des données sur un génome humain de référence téléchargé et indexé, le calcul du niveau d'expression des gènes à l'aide des données de mapping et enfin l'analyse statistique du niveau d'expression afin d'identifier les gènes différentiellement exprimés entre les sujets mutés et sauvages.

Le workflow qui a été produit avec Snakemake et Singularity a été utilisé pour appeler des containers existants et comportants les dépendances requises pour le pipeline. Un répertoire GitHub a également été créé. Ce dernier contient le pipeline principal annoté, mais également les ressources et les informations nécessaires à une bonne compréhension et réutilisation du workflow, rendant toute cette analyse reproductible.

Notre étude a montré l'expression différentielle de 260 gènes entre des tumeurs uvéales non mutées pour SF3B1 et des tumeurs uvéales mutées. Dans les tumeurs mutées, il y a une majorité de gènes surexprimés.

Le gène SF3B1 code un élément du splicéosome, donc une mutation de ce gène peut entraîner des modifications de l'épissage des gènes dépendants de ce facteur. Il serait donc intéressant de s'intéresser à l'épissage alternatif des gènes surexprimés dans les tumeurs pour mieux comprendre les mécanismes responsables d'une pathologie au pronostic plus grave que les mélanomes uvéaux non mutés pour SF3B1.

Mais le fait que nous n'ayons pas trouvé les mêmes gènes différentiellement exprimés que les

publications indique qu'il y aurait peut-être un défaut dans le protocole. Bien que nous avons vérifié la répétabilité et la réplicabilité, nous ne savons pas si le design de notre protocole apporte les bons résultats. Pour cela il faudrait reproduire les résultats en suivant un protocole différent. Aussi, nous avons pris des décisions différentes de celles prises par Furney et al, ce qui rend moins probable l'obtention de résultats similaires.