Groupe 3
Ludovic Pailloux, Bleuenn Rault
Léa Sauvage, Emma Wagner

université
PARIS-SACLAY

Septembre - Décembre 2021

PROJET HACKATHON

Étude des gènes impliqués dans les mélanomes uvéaux primaires

Table des matières

1 Introduction

1.1 Reproductibilité

Ces dernières années, beaucoup de débats ont émergé concernant la reproductibilité des travaux scientifiques. Des travaux de recherche ont tenté d'estimer le pourcentage des résultats de recherche reproductibles. Il s'est avéré que celui-ci était très faible. Par exemple, le statisticien Valen E. Johnson, dans son article scientifique *"Revised standards for statistical evidence"* publié en 2013 [1], avance qu'entre 17% et 25% des études scientifiques échoueraient à être répliquées. Un manque de reproductibilité des études scientifiques conduit à une perte de validité scientifique et donc à une perte de crédibilité de la science en général. Ainsi, un des enjeux actuels pour les chercheurs est de garantir des résultats reproductibles. Par exemple, ces derniers sont davantage invités à partager leurs données, leurs codes, et leurs méthodes d'analyse statistique. Les principes FAIR (*Findable Accessible Interoperable Reusable*, https://www.go-fair.org/fair-principles/) donnent des lignes directrices dans le but de faciliter la réutilisation des données de recherche par l'ensemble de la communauté scientifique.

L'irreproductibilité des résultats d'une publication scientifique peut être due à divers facteurs qui influent sur différents types de reproductibilité.

Reproductibilité expérimentale :
La reproductibilité expérimentale concerne l'expérience en elle-même. Des facteurs techniques peuvent nuire à celle-ci comme la variabilité naturelle des échantillons, la nature aléatoire des phénomènes observés, les changements de conditions expérimentales, le bruit dans les mesures, ou encore les changements dans la préparation des échantillons. Ensuite, il y a des facteurs humains qui peuvent jouer tels que le manque d'informations partagées sur les expériences ou le manque de traces écrites qui peuvent conduire à une irreproductibilité des résultats scientifiques.

Reproductibilité computationnelle :
La reproductibilité computationnelle concerne l'exploitation et l'analyse des résultats. Il peut exister une variabilité computationnelle empêchant la reproductibilité. Par exemple, les versions des systèmes d'opération et des outils utilisés peuvent être différentes. De plus, les logiciels et les algorithmes utilisés pour l'analyse des données peuvent varier.

On assiste depuis une vingtaine d'années à une "crise de la reproductibilité", et en particulier dans le domaine des sciences. Des solutions existent pour améliorer la reproductibilité notamment computationnelle comme l'utilisation de workflows reproductibles. Ces derniers permettent d'offrir un accès non seulement aux démarches, mais également aux données et à l'environnement sous lequel ont été exécutées les différentes instructions.

Il y a 3 niveaux de reproductibilité :

1. Répétition
2. Réplication
3. Reproduction

Enfin, il existe des environnements et des outils pour faciliter la reproductibilité des résultats. Par exemple, *the Virtual Imaging Platform* est un portail qui permet aux utilisateurs de traiter facilement leurs données grâce à des

logiciels pré-installés sur la plateforme.

La question de la reproductibilité intéresse donc de plus en plus le monde scientifique qui cherche des moyens pour faciliter et favoriser cela.

1.2 Outils utilisés

Pour ce projet, nous avons utilisé des outils pour le contrôle de version de données, des outils de gestion de workflow, ou encore un *operating system*. Ces outils permettent de structurer la raisonnement et la démarche scientifique, de façon à ce que n'importe qui puisse réutiliser l'intégralité de notre travail, du téléchargement des données jusqu'à l'analyse statistique.

1.2.1 Git

Git est un système de contrôle des versions avec de nombreux avantages. En effet, cela permet de conserver l'intégralité des fichiers à chaque version. L'historique est conservé. Chacun peut travailler localement sur son ordinateur sans altérer le fichier commun et quand même apporter ses modifications au projet. Cela permet à toute l'équipe de collaborer.
Nous avons travaillé en collaboration sur le répertoire Hackathon :
https://github.com/emma-wag/Projet_Hackathon

1.2.2 Singularity

Docker et Singularity permettent de regrouper des outils dans des conteneurs isolés qui peuvent être exécutés sur n'importe quel serveur. Docker est une plateforme *open-source* utilisant des images, qui sont des couches de fichiers empilés. Ces images sont les bases des conteneurs, ces derniers étant en fait des exécutions d'images. Singularity est le système *open-source* le plus largement utilisé. Il est simple d'utilisation et permet de créer des conteneurs ou d'utiliser des conteneurs existants. Singularity est un peu moins populaire que Docker, cependant c'est celui que nous avons utilisé puisque c'est celui que Snakemake prend en charge. Cependant, Singularity peut être utilisé avec des images Docker. Nous avons donc fait appel à plusieurs conteneurs Docker, mais nous avons exécuté le workflow avec Singularity.

1.2.3 Snakemake

Il existe plusieurs logiciels qui permettent de structurer les analyses. Nous avons choisi d'utiliser Snakemake, un gestionnaire de workflow. Cet outil de workflow permet la collecte et la préparation des données, l'analyse et la modélisation prédictive, qui aboutissent à de nouveaux produits de données. Il a pour avantage d'automatiser les processus et donc d'éviter les tâches répétitives, ce qui permet de gagner en productivité. Le workflow peut également s'adapter de manière transparente aux environnements des serveurs.

1.2.4 R

R est un langage de programmation pour traiter et visualiser les données. En effet, ce système permet de faire des analyses statistiques, mais aussi de représenter ces données graphiquement. Cela facilite l'interprétation des données.

1.3 Objectifs du projet

Le mélanome uvéal est le cancer primaire de l'œil le plus fréquent et il entraîne souvent des métastases fatales. Ce dernier peut être guéri par la chirurgie ou la radiothérapie, mais la maladie à l'étape de métastase est réfractaire au

traitement. Des premiers travaux de recherches ont montré qu'il existait notamment des mutations récurrentes au niveau du codon 625 du facteur d'épissage SF3B1 dans le mélanome uvéal. Ce facteur d'épissage code la sous-unité 1 du facteur d'épissage 3B.

Ces résultats sont présentés dans une première publication dans *Nature Genetics* en 2013. Pour aboutir à ces résultats, les chercheurs ont réalisé le séquençage d'ARN de 18 mélanomes uvéaux primaires. Cela a permis de mettre en évidence que les mutations de SF3B1 étaient associées à un épissage alternatif différentiel de gènes codant certaines protéines.

Ensuite, une deuxième publication a repris les expériences précédentes en séquençant aléatoirement des échantillons d'ARN de 12 patients atteints de mélanomes uvéaux primaires. Les chercheurs ont évalué l'épissage alternatif de 8 gènes qui ont fourni les preuves les plus solides d'épissage par qRT-PCR. Il s'agit des gènes GUSBP11, UQCC, ANKHD1, GAS8, F8, ADAM12, CRNDE et ABCC5. Leur analyse a confirmé que ces 8 gènes étaient alternativement épissés dans les tumeurs SF3B1 mutantes (3 échantillons mutés) par rapport aux tumeurs SF3B1 de type sauvage (9 échantillons wild types). Ainsi, le mélanome uvéal ferait partie d'un petit groupe de cancers associés aux mutations du gène SF3B1. Ces mutations seraient associées à un épissage alternatif aberrant.

Nos travaux de reproductibilité se sont fondés sur les données de ces deux publications :

— Harbour JW, Roberson ED, Anbunathan H, Onken MD, Worley LA, Bowcock AM. Recurrent mutations at codon 625 of the splicing factor SF3B1 in uveal melanoma. Nat Genet. 2013 ;45(2) :133-135. doi :10.1038/ng.2523

— Furney SJ, Pedersen M, Gentien D, et al. SF3B1 mutations are associated with alternative splicing in uveal melanoma. Cancer Discov. 2013 ;3(10) :1122-1129. doi :10.1158/2159-8290.CD-13-0330

L'objectif de ce projet d'Hackathon reproductible est de créer un workflow permettant de retrouver les résultats de ces publications en repartant des données utilisées par les chercheurs.

2 Organisation du travail

Après un premier temps de lecture et de compréhension des deux articles scientifiques à notre disposition, nous avons réfléchi, en groupe, à l'organisation du projet et aux étapes nécessaires pour y arriver. Une des premières étapes a été de bien comprendre et d'identifier les objectifs du projet. Ensuite, nous avons commencé le workflow. Pour cela, dans un premier temps, nous avons beaucoup travaillé en binôme. Un premier binôme essayait de dégrossir le code et un second plus à l'aise en bash était chargé de corriger les erreurs. À chaque fin de séance, les codes étaient mis sur Git pour que chacun puisse y revenir de son côté. Ensuite, concernant les réunions hebdomadaires du vendredi, nous avons utilisé l'outil Google slide pour que chacun puisse ajouter ses questions et ses avancées. Durant la réunion, une ou deux personnes étaient chargées de prendre des notes.
À partir de début novembre, ayant beaucoup avancé sur le workflow, nous avons commencé à rédiger le rapport. En parallèle, nous avons commencé l'analyse statistique des résultats de comptage. La dernière étape a été la finalisation du rapport. Nous nous sommes répartis le travail de rédaction. Grâce à l'utilisation de *Overleaf*, nous avons pu travailler en simultané et en collaboration lors de nos réunions. Chacun a pu apporter des modifications à toutes les parties.

3 Résultats

3.1 Construction du workflow

Dans cette partie, nous expliquerons les différentes étapes de construction du workflow et les conteneurs utilisés pour chacune de ces étapes. Vous pourrez trouver une explication plus détaillée des options de Snakemake (en particulier les fonctions wildcards et expand) dans la partie 4.2.

Au début du workflow, nous avons créé des variables :

— gene_names : liste de nombres allant de 2 à 9 que nous allons utiliser pour télécharger les gènes. En effet, chaque nom de gène diffère par le dernier chiffre de son intitulé.

— chromosome_nb : une liste avec les 22 chromosomes composés d'ADN nucléaire et le chromosome "MT" composé d'ADN mitochondrial.

La règle générale au début du workflow (rule all) prend en input la sortie finale du workflow, c'est-à-dire le dossier qui contient les sorties statistiques obtenues avec R. Ensuite, viennent les règles spécifiques que nous allons détailler ci-dessous. Chaque règle peut prendre en input la sortie d'une des règles précédentes, ainsi tout le workflow est exécutable en une seule fois.

3.1.1 Téléchargement des données et dézipage des fichiers

La première étape est le **téléchargement des gènes** au format FastQ à partir de la base de données ENA (*European Nucleotide Archive*). Nous avons téléchargé des paires de gènes. En effet, le même gène a été séquencé à ses deux extrémités pour générer des données de séquençage de haute qualité. Le séquençage en paires (*paired-end*) facilite la détection des éléments répétitifs et des réarrangements génomiques.

Pour télécharger les données, nous avons donc utilisé deux règles dans le workflow :

— download_gene 1 : téléchargement des fichiers fastQ en shell avec la commande wget. Nous allons chercher les adresses de chaque gène dans la base de données en modifiant le dernier numéro de l'adresse pour avoir les différents gènes. Nous faisons donc appel ici à la liste gene_names. Puis, ces fichiers téléchargés sont stockés dans le dossier *genes*.

— download_gene2 : idem pour la deuxième paire du gène.

La deuxième étape est le **téléchargement des 23 chromosomes** à partir de la base de données *Ensembl*. Pour cela, nous utilisons aussi la commande wget. L'exécution de cette règle nous donne un dossier *chromosome* comportant les 23 chromosomes au format Fasta.

Ensuite, les trois prochaines règles du workflow permettent de **déziper** les fichiers créés précédemment. Pour cela, nous utilisons la commande gunzip en shell. Il y a trois règles différentes pour les trois dossiers précédents. Chaque règle prend en input un des dossiers créés et retourne en output le même mais avec les fichiers dézipés.

3.1.2 Concaténation, indexation et annotation du génome

La **concaténation** du génome consiste à récupérer tous les chromosomes un par un en entrée pour les rassembler dans un même fichier : genome.fa.

Ensuite, vient l'étape de l'**indexation** afin de pouvoir gérer plus facilement le génome humain qui est volumineux. Pour l'indexation du génome, nous avons fait appel à un conteneur Docker qui a nous permis d'utiliser la commande

STAR. L'input nécessaire est le fichier genome.fa. STAR va nous permettre de générer des index génomiques. Pour cela, nous précisons les paramètres suivants :

— runThreadN : nombre de threads
— runMode : génère le génome
— genomeDir : direction de la sortie
— genomeFastaFiles : chemin des entrées

Cette règle nous fournit un ensemble de fichiers en sortie (plusieurs en format .txt notamment avec des informations sur les paramètres). Cependant, celui qui nous intéresse le plus (et qui est le dernier à être créé) est le fichier SAindex. Afin de s'assurer que le programme ne s'arrête pas de tourner avant sa création, nous avons choisi de le mettre en output.

Au sujet du paramètre *thread*, il faut savoir que ce dernier correspond au nombre de CPU (*Central Processing Unit*), c'est-à-dire au nombre de ressources allouées à la réalisation de cette tâche. Nous savons que si nous mettons une valeur élevée, le code tournera plus rapidement mais demandera plus de puissance électrique. Il faut donc faire un compromis entre le temps de calcul et la puissance allouée. Par exemple, cela nécessitera un système de refroidissement de l'ordinateur plus efficace et une batterie plus puissante. De plus, il est important de noter qu'un processeur à n noyaux n'est pas égal à n processeurs distincts. La puissance additionnelle gagnée lors de l'augmentation d'un CPU diminue à chaque ajout. Il existe bien une valeur optimale de noyaux. Nous avons donc fait plusieurs tests pour définir la valeur qui nous semblait optimale et nous avons opté pour 8.

Puis, pour l'**annotation** du génome, nous avons téléchargé et dézipé des fichiers en une seule et même règle. Cela nous renvoie un dossier *genome_annot* dézipé. Le script s'est vu davantage optimisé qu'en début de projet. En effet, au fur et à mesure de ce dernier, nous avons de mieux en mieux maîtrisé les entrées et sorties du workflow. Malgré des difficultés initiales de compréhension des commandes du workflow, nous avons amélioré notre maîtrise de ces outils.

3.1.3 Mapping

Dans cette étape, nous avons effectué le **mapping des gènes**. Cette règle prend en entrée les paires des gènes et le génome indexé pour ressortir en output des fichiers .bam (*Binary Alignment Map*). Les fichiers au format .bam sont composés de données brutes complètes codées en binaire sur le séquençage génomique. Dans cette étape du workflow, nous avons utilisé à nouveau un conteneur Docker pour exploiter la fonction STAR et effectuer ce mapping. La plupart des paramètres utilisés pour cette commande nous ont été fournis. Nous avons simplement précisé ceux pour runThreadN et genomeDir (définis plus haut), readFilesIn (chemins des fichiers contenant les séquences dont on veut faire le mapping) et limitBAMsortRAM (RAM maximale disponible pour trier les fichiers BAM).

3.1.4 Comptage des reads

L'étape suivante a été le **comptage des reads**. Avant cela, rappelons la méthode expérimentale utilisée pour obtenir les séquences FastQ. Tout d'abord, les ARN ont été récoltés dans plusieurs cellules cancéreuses à la fois. Ces macromolécules ont été découpées en fragments, eux-même amplifiés par PCR. Ensuite, un séquençage a été effectué et les chercheurs ont obtenu des *reads* qui sont des "petits bouts" de séquences génomiques. Il est important de rappeler aussi que ces ARN obtenus avec la méthode de RNA-seq sont des ARN déjà épissés. L'hypothèse est que le nombre de *reads* venant d'un certain gène est proportionnel à l'abondance de l'ARN de ce gène dans la cellule. Cette méthode de comptage de *reads* permet donc de calculer le niveau d'expression des gènes.

Cela correspond à la règle *count*. Les inputs sont les fichiers .bam obtenus après le mapping et le fichier correspondant au génome annoté. Nous faisons appel à un conteneur Docker nous permettant d'utiliser FeatureCounts qui compte les *reads*.

Ci-dessous les options spécifiées :

— T : nombre de threads

— t : séquences que l'on recherche : gène

— g : gene_id à stocker dans l'output

— s : indique si le comptage est "strand-specific" (associé au brin). Il y a 3 valeurs possibles (0, 1 ou 2). Par défault, nous avons pris 0.

— a : chemin vers le fichier d'annotation

— o : chemin de l'output

On obtient en output les fichiers de comptage. C'est-à-dire que l'on renvoie, pour les 8 ARN séquencés, le nombre de *reads* trouvés pour chaque partie du génome annoté.

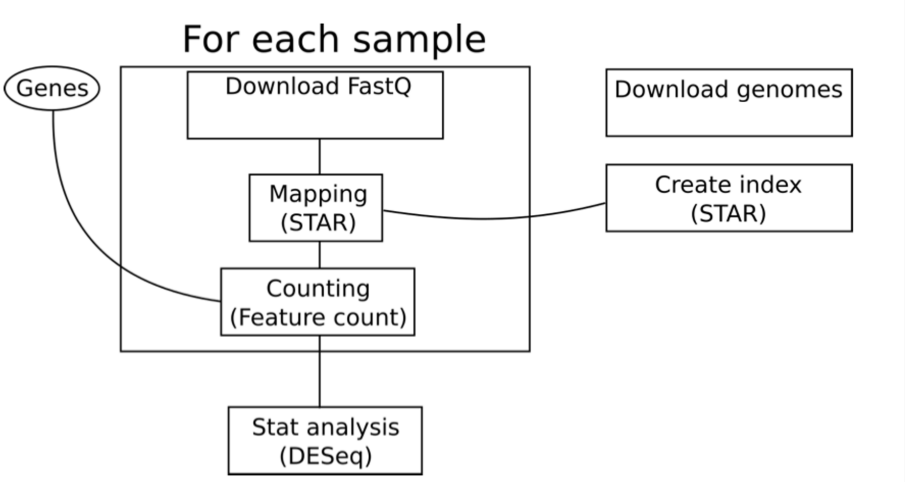

Figure 1 – Résumé des étapes du workflow

3.1.5 Analyse statistique

Nous avons intégré l'exécution de l'analyse statistique que nous avons fait via R dans le workflow. Cette règle prend en input les fichiers de sortie précédents sur le comptage des *reads*. Nous faisons appel au conteneur Docker DESeq2 et nous exécutons le script Hackathon.R. Tout cela nous permet d'obtenir l'output stats.Rdata (qui correspond bien à l'input que nous avons mis au début dans la règle principale). Celui-ci est constitué de toutes les informations dont nous aurons besoin pour réaliser les analyses statistiques. En particulier, on y retrouve quelques fichiers créés dans le script R (essai et mutation) et les résultats des fonctions de DESeq2 (dds, dds2 et resultats).

Dans la table "resultats", on y trouve plusieurs informations intéressantes :

— les identifiants des gènes
— la moyenne d'expression sur l'ensemble des 8 échantillons pour un gène donné
— log2fold change (log2 du ratio entre les 2 conditions étudiées. Cela rend les données symétriques pour mieux visualiser)
— la p-valeur pour un gène donné

3.2 Exécution du workflow

L'éxécution du workflow se fait grâce à la commande bash suivante : snakemake –use-singularity –cores 12 -s /home/ubuntu/Projet_Hackathon/Workflow/snakefile.

Il faut au préalable activer conda, télécharger singularity et récupérer le workflow du git. Cela est expliqué plus précisément sur le git dans le fichier Guide.txt qui se situe lui-même dans le dossier workflow. Pour exécuter tout le script, il faut compter entre 1h30 et 2h. De plus, il faut faire attention à avoir suffisamment d'espace sur la machine virtuelle. Après plusieurs essais, nous avons trouvé qu'il fallait au moins 32Go de mémoire.

3.3 Résultats biologiques

Nous allons procéder à une analyse statistique des données obtenues en sortie du workflow. Dans cette étape, nous avons utilisé le conteneur R/DESeq : evolbioinfo/deseq2 :v1.28.1

3.3.1 Analyses en Composantes Principales (ACP)

Dans un premier temps, nous avons cherché à réaliser une ACP sur les données d'expression des gènes. En effet, une ACP permet d'identifier des groupes et de quantifier des corrélations entre les variables quantitatives. Ici, nous nous intéressons à l'ACP pour savoir si l'on peut distinguer les échantillons WT des échantillons mutés.

Tout d'abord, nous avons étudié la part de variance expliquée par chaque dimension principale dans le but de déterminer le nombre d'axes à choisir pour l'ACP.

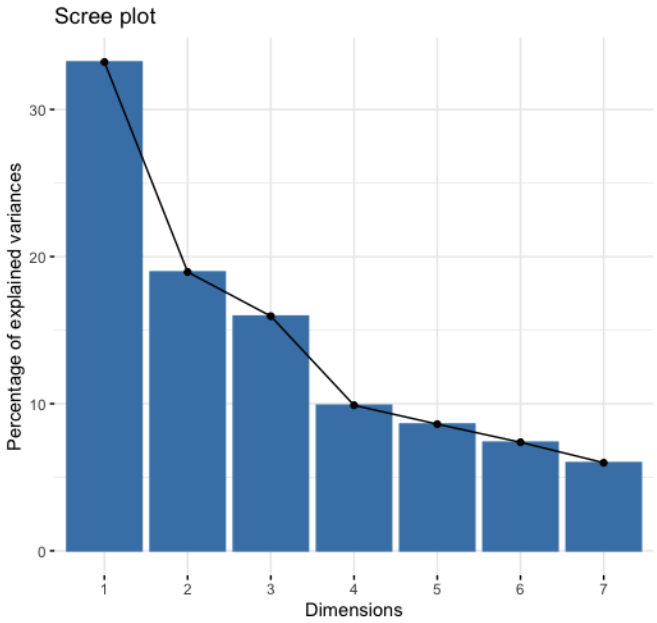

Figure 2 – Pourcentage de la variance expliquée par les composantes (scree plot)

D'après ce graphique, les trois premières dimensions expriment 68,1% de la variance, ce qui est un pourcentage intéressant. De plus, au-delà de la dimension 3, nous pouvons observer un "coude" avec, à partir de la dimension 4, une pente bien plus faible. Ces dimensions nous intéressent moins car elles expliquent moins de variance. Nous choisissons alors les deux premières dimensions pour réaliser une première ACP puisque ces deux axes concentrent à eux deux une grande partie de la variabilité (52,2%).

Voici la carte factorielle que nous avons obtenu suite à l'exécution du script R :

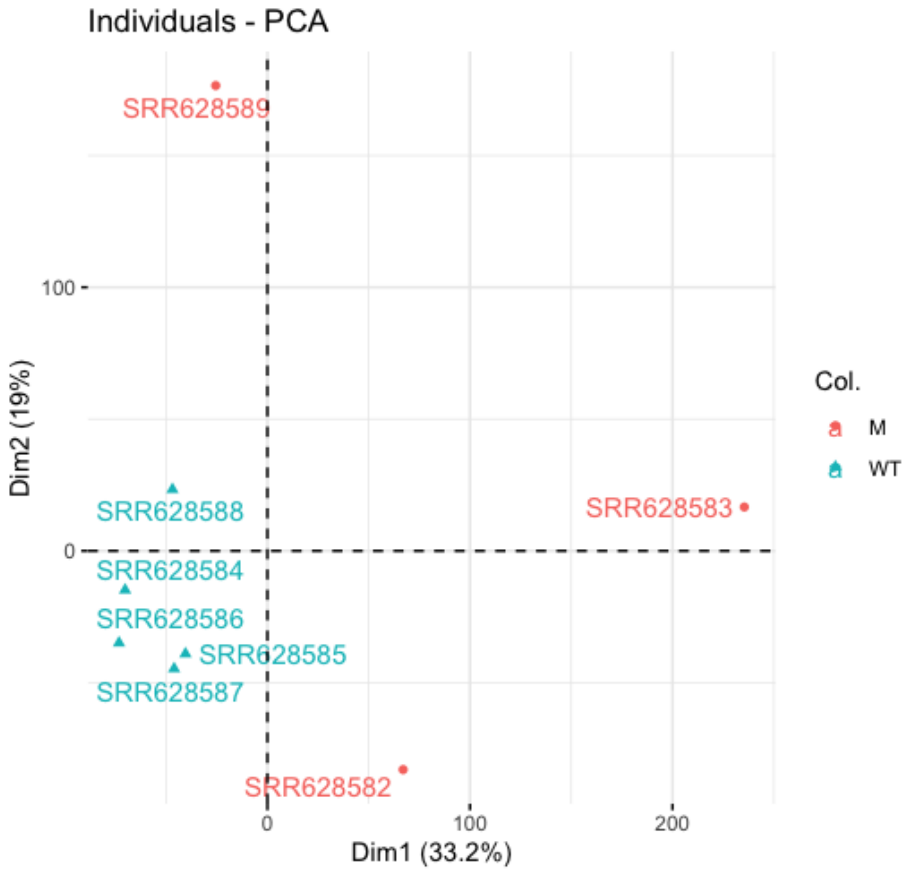

Figure 3 – Analyse en Composantes Principales dimension 1 et dimension 2

On constate que les données d'expression des gènes issus de phénotype WT sont regroupées (représentés en bleu sur la figure 3). Ce n'est pas le cas des individus mutés qui sont éloignés du groupe précédent selon les deux axes de projection.

L'axe 1 exprime 33,2% de l'inertie totale et l'axe 2 en exprime 19%. Ces valeurs sont assez élevées mais nous avons tout de même décidé de voir ce qu'il se passait avec la dimension 3 qui explique 15,9% de la variabilité. Nous procédons donc à une deuxième projection, ici selon les dimensions 2 et 3 de l'ACP.

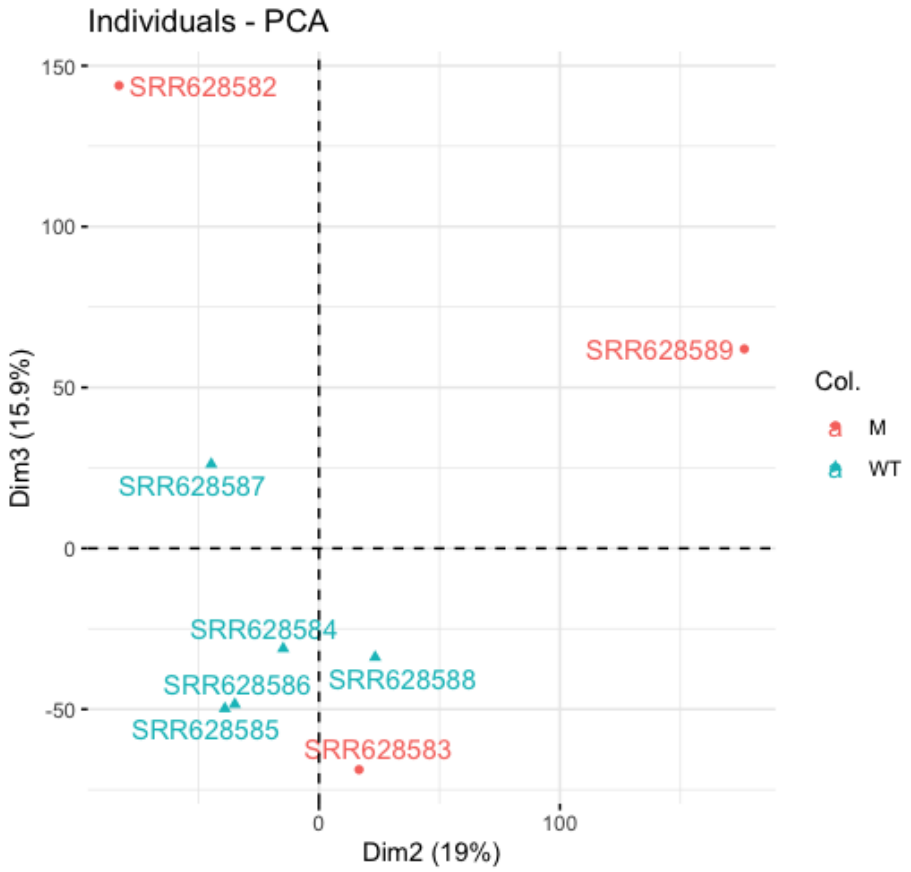

Figure 4 – Analyse en Composantes Principales dimension 2 et dimension 3

Dans cette deuxième projection, nous pouvons observer que les phénotypes WT forment à nouveau un groupe, comme vu précédemment, bien que ce groupe soit légèrement plus dispersé. Quant aux échantillons mutés, ils sont éloignés du groupe sauf pour un point qui est plus proche du groupe (SRR628583).
Cela semble cohérent car les individus mutés peuvent présenter des mutations différentes entre eux et donc l'expression des gènes peut varier.

3.3.2 Volcano plot : Wild types vs Mutés

Pour compléter notre analyse, nous avons effectué un *Volcano Plot*. Il s'agit d'un diagramme de dispersion permettant d'identifier rapidement les changements significatifs dans un très grand ensemble de données. Ici nous avons 20851 variables qui correspondent à l'expression des 20851 gènes de l'ensemble du génome humain. Ce diagramme représente la p-valeur (-log10 p) sur l'axe des ordonnées en fonction de l'ampleur du changement sur l'axe des abscisses (log2 fold-change). Le fold-change correspond au ratio du niveau d'expression du gène chez les sujets mutés sur le niveau d'expression du gène chez les sujets wild types.

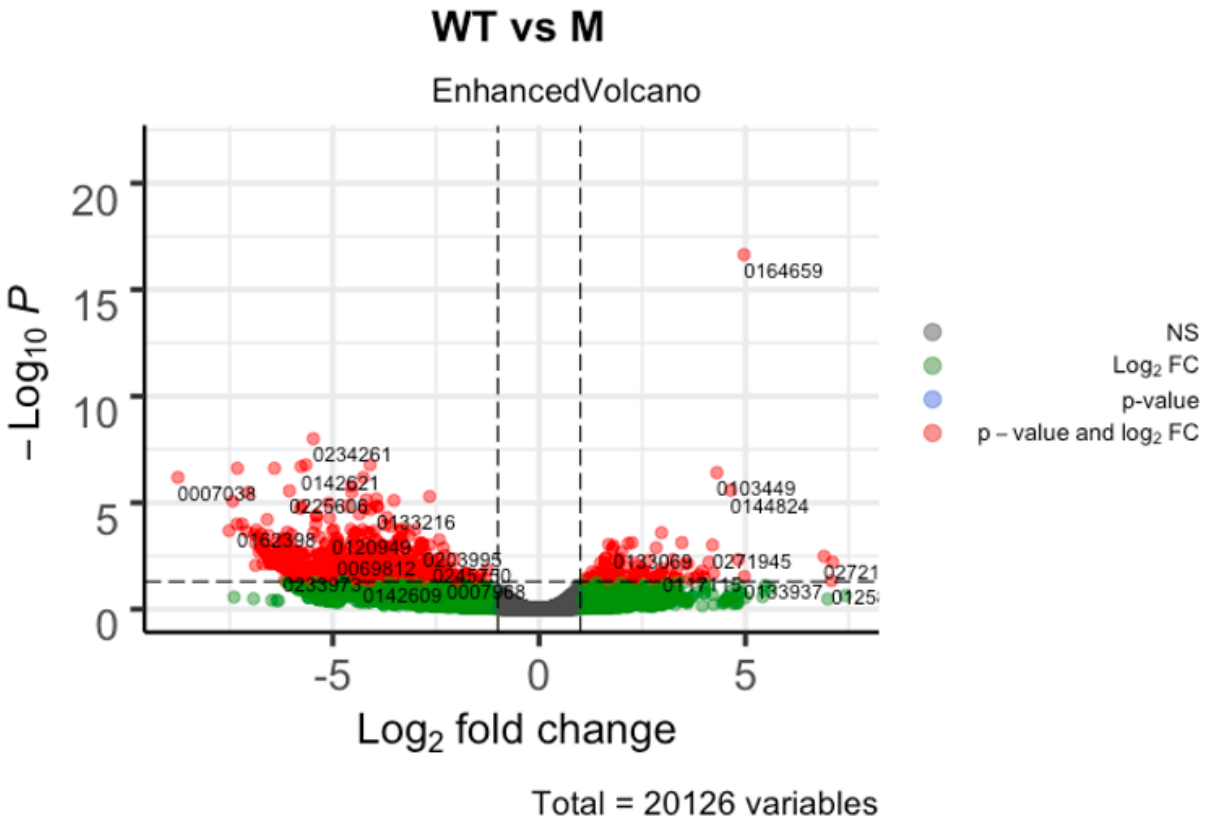

Figure 5 – Volcano plot : Wild type vs Muté

Sur cette figure, nous avons fixé deux seuils : un pour l'axe des ordonnées et un pour l'axe des abscisses. Concernant l'axe des ordonnées, le seuil a été fixé à 1.3, correspondant à une p-valeur de 0,05. Les points en rouge sont ceux dont la p-valeur est inférieure à 0,05 et correspondent donc à des résultats significatifs. Pour l'axe des abscisses, nous avons choisi de délimiter à -1 et 1. Quand on se déplace vers la droite sur le graphique, on trouve des gènes plus exprimés chez les mutés. Inversement, quand on se déplace vers la gauche, les gènes sont plus exprimés chez les WT. Les gènes se trouvant à gauche de -1 correspondent à des gènes deux fois plus exprimés chez les WT. De même, les gènes se trouvant à droite de 1 correspondent à des gènes deux fois plus exprimés chez les individus mutés.

Parmi nos 20851 gènes, il y en a 809 qui ont une p-valeur ajustée inférieure à 0.05. Cela veut dire que nous avons 809 gènes dont l'expression est significativement différente entre les mutés et les WT. On peut alors comparer ce résultat à ceux obtenus dans les deux publications avec lesquelles nous travaillons. Dans l'article de *Furney et al.*, les auteurs indiquent avoir trouvé 325 gènes à expression différentielle entre muté et WT, et la différence se creuse encore plus avec l'article de *Harbour et al.* qui dit n'en avoir que 10.

L'article de *Furney et al.*. donne cependant une autre précision : 15% des 325 gènes à expression différentielle sont sur-exprimés chez les mutants, alors que 85% sont sous-exprimés chez les mutants. À l'aide de notre work-

flow, nous trouvons que 717 gènes sur 809 ont un log2(fold-change) inférieur à -1, ce qui veut dire que 88.6% des gènes sont sous-exprimés chez les mutants. En revanche, on trouve 92 gènes avec un log2(fold-change) supérieur à 1, ce qui nous donne 11.4% de gènes sur-exprimés chez les mutants. Nos résultats sont donc similaires à ceux de l'article.

Pour aller plus loin, nous avons cherché les gènes qui ont une expression différentielle très élevée (les points qui sortent du lot sur le graphique), afin de voir si nous retrouvons les mêmes dans les articles. Nous avons pris quelques exemples, en récupérant quelques ID de gènes sur le volcano plot, puis leur identifiants à partir de la nomenclature Ensembl.

- ENSG00000164659 → KIAA1324L
- ENSG00000103449 → SALL1
- ENSG00000144824 → PHLDB2
- ENSG00000234261 → AL138720.1
- ENSG00000007038 → PRSS21

L'article de *Furney et al.* indique alors qu'on peut trouver dans le tableau complémentaire 8 la liste des 325 gènes identifiés comme exprimés de manière différentielle. Cependant, aucun des gènes cités plus haut n'est mentionné dans ce tableau, ce qui est plutôt curieux.

4 Matériel et méthodes

4.1 Données utilisées

Tout d'abord, les séquences ARN que nous avons utilisées ont été téléchargées via le site ENA Browser. Il s'agit de 8 séquences : "SRR628582", "SRR628583", "SRR628584", "SRR628585", "SRR628586", "SRR628587", "SRR628588" et "SRR628589". Le site nous permet également de trouver l'état de mutation de chaque gène (muté ou wild-type), dans la section "Sample Accession". Il y a cependant une erreur sur le site, puisque le gène "SRR628584" est noté comme étant muté, mais qu'il apparaît dans le groupe des WT sur nos ACP.

Ensuite, nous avons téléchargé le génome humain sous forme de 23 chromosomes que nous avons concaténé par la suite. Ces chromosomes ont été téléchargés sur le site *Ensembl* au format FASTA. Sur ce site, nous avons aussi téléchargé l'annotation du génome en format GTF (*General Feature Format*). Ce format est utilisé pour décrire les séquences d'ADN, d'ARN et de protéines. Ce sont des fichiers tabulaires avec 9 champs par lignes, séparés par tabulation. La structure générale est la suivante : séquence - source - élément - début - fin - score - brin - phase - attributs.

Le workflow permet de ne pas télécharger toutes ces données manuellement mais via le snakefile directement.

4.2 Choix des outils et de leurs options

Tout d'abord, nous avons utilisé des commandes bash sur nos propres machines virtuelles pour activer le package conda, télécharger singularity, puis pour cloner le fichier créé sur le git sur nos machines, afin de travailler tous simultanément.

Snakemake a été choisi pour faire le workflow car nous voulions écrire nos commandes en langage Python. Cependant, nous avons quand même utilisé "shell" pour pouvoir écrire certaines commandes en Bash afin de nous faciliter la tâche. Dans le workflow, nous avons utilisé des Dockers pour appeler des fonctions spécifiques comme *STAR* et *featureCounts*.

De plus, nous avons utilisé la fonction **wildcards**. Cela nous permet de réduire le nombre de lignes de code nécessaires et de généraliser la pipeline. Plus précisément, cette fonction nous permet de définir des variables dans les input et output et les appeler dans la commande shell. Par exemple, si dans notre workflow, nous avions mis en input sample = 2, la commande
wget -O output "ftp ://ftp.sra.ebi.ac.uk/vol1/fastq/SRR628/SRR62858{wildcards.sample}/SRR62858 {wildcards.sample}_1.fastq.gz"
remplacerait tous les wildcards.sample par 2, ce qui nous donnerait le lien
"ftp ://ftp.sra.ebi.ac.uk/vol1/fastq/SRR628/SRR628582/SRR628582_1.fastq.gz"
En fait, la fonction wildcards est remplacée par une variable définie au préalable.

La combinaison de la fonction wildcards à la fonction **expand** nous permet de gagner en temps et en espace. En effet, la fonction expand permet la sélection et l'agrégation de plusieurs fichiers. Ici, au lieu d'écrire 23 fois le lien à télécharger pour chaque chromosome, nous avons remarqué qu'ils se différenciaient les uns des autres par un unique chiffre. Grâce à la fonction expand, nous avons pu garder un tronc commun et faire varier ce chiffre de 1 à 22 (+ le chromosome "MT"). La commande :
input : expand("chromosome/Chrchr_list/Homo_sapiens.GRCh38.dna.chromosome.chr_list.fa", chr_list = chromosome_nb)
permet de créer un fichier par chromosome, en gardant un tronc commun ("Homo_sapiens.GRCh38.dna.chromosome."). Il n'y a que la fin qui varie en prenant un par un chaque élément de la liste chromosome_nb définie précédemment.

Finalement, l'utilisation de **wildcards** et **expand** nous permet d'utiliser une même syntaxe pour désigner plusieurs fichiers qui s'exécutent de la même manière. L'exécution de tous ces fichiers se fait dans une règle commune.

Git nous a ensuite permis de mettre tous les travaux en commun grâce à la commande *push*. Au préalable, il faut utiliser la commande : git add "nom_fichier" puis, git commit -m "commit message" qui enregistre les changements apportés sur notre propre machine virtuelle. La commande *push* va aller copier ces fichiers sur Git.
De plus, nous avons inclu R dans le workflow. Nous utilisons le conteneur DESeq2 et faisons appel à un script Hackathon.R. Ensuite, le script AnalyseStat.R utilise les sorties du script précédent pour nous fournir les graphiques présentés précédemment.

4.3 Choix des machines virtuelles

Afin de faire tourner notre workflow nous avons utilisé la machine virtuelle BioPipes sur le site de *Biosphere*. Pour faire le lien entre GitHub et la machine virtuelle, ceux d'entre nous qui étaient sous environnement Windows, ont utilisé le serveur MobaXTerm. Le reste de l'équipe, travaillant sur MacOS, a directement fait le lien entre la machine virtuelle et GitHub via le terminal. Dans les deux cas, une clef SSH a été générée.

Au cours du projet, lors de l'exécution du workflow, nous avons rencontré quelques difficultés. En effet, plusieurs messages d'erreurs apparaissaient ne permettant pas l'exécution du script pour l'étape d'indexation du génome. En essayant d'exécuter la commande seule ou dans le worflow complet, il y avait toujours une erreur. Nous avons compris après qu'il s'agissait d'un problème de mémoire. Notre machine virtuelle n'était que de 8Gb. En augmentant la mémoire de la machine virtuelle à 32Gb sur *Biosphere*, le script a finalement pu être exécuté.

5 Conclusion

Pour conclure, ce projet avait pour objectif de créer un workflow reproductible afin de retrouver les résultats présentés dans des publications scientifiques.

En ce qui concerne la partie reproductible, notre projet est exécutable sur n'importe quelle machine virtuelle du moment que l'on respecte les informations explicitées dans le READ.ME accompagnant le workflow. Cependant, ce projet est dépendant des bases de données utilisées. En effet, il ne faut pas de changement dans les adresses de téléchargement des données, auquel cas nous aurions à apporter quelques modifications au workflow pour s'adapter au potentiel changement. De même, des mises à jour des outils et logiciels utilisés pourraient créer des changements dans les résultats attendus. Nous rappelons aussi que nous sommes très dépendants des conditions expérimentales. Les difficultés liées aux reproductibilités expérimentale et computationnelle ne sont donc pas négligeables.

À propos des résultats obtenus, nous observons des différences par rapport à ce que nous attendions. En effet, nous pensions retrouver les gènes mutés présentés dans les articles comme étant les gènes représentant la variabilité dans nos résultats. Avec l'ACP, nous avons bien constaté que les organismes non mutés formaient un groupe d'individus alors que les organismes mutés représentaient des individus isolés et éloignés. Toutefois, d'après les résultats du Volcano Plot, les gènes avec une expression différente dans les deux conditions muté et WT ne correspondent pas aux gènes présentés dans l'article. Nous avons dû faire des erreurs lors de l'analyse de nos résultats ou lors de la construction du workflow.

Nous nous sommes alors demandés d'où pouvait provenir cette erreur. Pour cela, nous nous sommes intéressés à d'autres articles scientifiques traitant aussi des mutations au codon 625 du facteur d'épissage SF3B1 dans le mélanome uvéal. Comme le mentionnent différents articles, des mutations sur les gènes GNAQ ou GNA11 seraient considérés comme les mutations activatrices majeures de la carcinogénèse du mélanome uvéal et sont souvent citées ([4],[5]). Il est donc étrange de ne pas retrouver ces gènes par exemple dans nos résultats. Cependant, les résultats des articles divergent eux aussi. Ces résultats ne sont pas absolus. De nombreuses facteurs interviennent comme les conditions expérimentales.

Ici encore, l'intérêt d'utiliser un workflow, rendant notre étude reproductible, est évident. Si notre travail était publié et qu'une erreur était détectée dans le script, une simple modification permettrait de réutiliser notre travail sans grande modification.

Pour finir au sujet du workflow, nous aurions pu optimiser certaines étapes comme télécharger les fichiers et les déziper dans une seule et même règle. Le temps d'exécution du workflow, quant à lui, ne semble pas aberrant. De plus, le workflow est commenté pour permettre à d'autres personnes de le réutiliser tout en comprenant l'enchaînement des différentes étapes.

6 ANNEXE

Bibliographie :

[1] Johnson VE. Revised standards for statistical evidence. Proc Natl Acad Sci U S A. 2013 ;110(48) :19313-19317. doi :10.1073/pnas.1313476110

[2] Harbour JW, Roberson ED, Anbunathan H, Onken MD, Worley LA, Bowcock AM. Recurrent mutations at codon 625 of the splicing factor SF3B1 in uveal melanoma. Nat Genet. 2013 ;45(2) :133-135. doi :10.1038/ng.2523

[3] Furney SJ, Pedersen M, Gentien D, et al. SF3B1 mutations are associated with alternative splicing in uveal melanoma. Cancer Discov. 2013 ;3(10) :1122-1129. doi :10.1158/2159-8290.CD-13-0330

[4] Martin M, Maßhöfer L, Temming P, et al. Exome sequencing identifies recurrent somatic mutations in EIF1AX and SF3B1 in uveal melanoma with disomy 3. Nat Genet. 2013 ;45(8) :933-936. doi :10.1038/ng.2674

[5] Canbezdi C, Tarin M, Houy A, et al. Functional and conformational impact of cancer-associated SF3B1 mutations depends on the position and the charge of amino acid substitution. Comput Struct Biotechnol J. 2021 ;19 :1361-1370. Published 2021 Feb 27. doi :10.1016/j.csbj.2021.02.012

[6] Weidmann C. Caractérisation des mécanismes moléculaires à potentiel thérapeutique dans la progression métastatique du mélanome uvéal. https ://corpus.ulaval.ca/jspui/bitstream/20.500.11794/35053/1/33378.pdf. 2017