Estudio piloto de la expresión alélica específica a partir de datos de RNA-seq y genotipado de SNPs en trastorno bipolar

AUTHOR
Susana Inés Rodríguez Sanz

PUBLISHED
December 12, 2024

Introducción

Esta viñeta tiene como finalidad describir y explicar los scripts desarrollados es este trabajo de fin de máster, en el que se analizan datos de RNA-seq y genotipado de individuos afectados y no afectados por el trastorno bipolar, con el objetivo de estudiar la expresión alélica específica (ASE).

La **Figura 1** resume los diferentes tipos de expresión alélica, y también incluye una representación de cómo se cuantifica esta expresión alélica a partir de los datos empleados en este TFM, RNA-seq y genotipado.

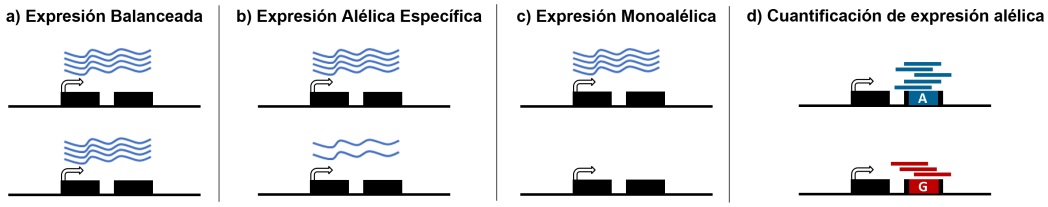


Figura 1. Patrones de expresión alélica específica. a) Expresión Balanceada: ambos alelos del gen se expresan por igual; b) Expresión Alélica Específica: un alelo muestra una mayor expresión en comparación con el otro; c) Expresión monoalélica: cuando solo se expresa una forma alélica. d) Cuantificación de la expresión alélica: se utilizan los datos de RNA-seq y los datos de SNPs heterocigotos para contar las lecturas alineadas de cada uno de los dos alelos.

El flujo de trabajo que se ha seguido se resume en la **Figura 2**:

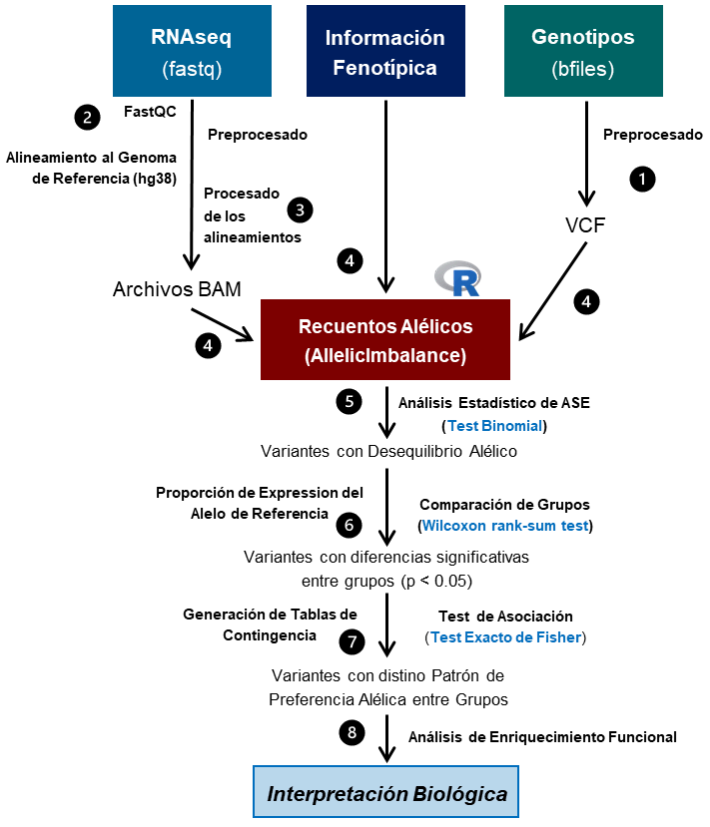


Figura 2. Flujo de trabajo empleado en el TFM. (1) Generación del VCF multimuestra con los datos de genotipado. (2) Control de calidad (FastQC) de los datos de RNA-seq, procesamiento y alineamiento al genoma de referencia. (3) Eliminación de las lecturas duplicadas y reducción del sesgo de mapeo. (4) Cálculo de los recuentos alélicos en sitios heterocigotos con AllelicImbalance. (5) Test binomial para detectar desequilibrio alélico en cada variante por individuo. (6) Cálculo de la proporción de expresión del alelo de referencia para cada individuo. Comparación de la distribución de proporciones entre los grupos mediante una prueba de suma de rangos de Wilcoxon. (7) Clasificación cualitativa de las variantes significativas ($p < 0.05$) según el alelo que se expresa preferentemente (referencia o alternativo). Generación de tablas de contingencia con los patrones de preferencia alélica entre los grupos. Evaluación de la asociación entre el estado clínico y los patrones de preferencia alélica con el test exacto de Fisher. (8) Los genes con ASE diferencial pasan a un análisis de enriquecimiento funcional para revelar posibles vías implicadas.

Bloque 1: datos de partida

En este estudio se ha partido de los datos de genotipado y RNA-seq de 10 individuos afectados por trastorno bipolar y 10 individuos sanos, manteniendo una proporción equitativa de hombres y mujeres en ambos grupos.

Tabla 1. Información de los individuos incluidos en el estudio. Se incluyen los identificadores anonimizados, el sexo y la condición: A) individuos sanos y B) individuos afectados por trastorno bipolar.

A)

Muestra	Condición	Sexo
HSPR_0042	Sano	MUJER
UAMR_0001	Sano	HOMBRE
UAMR_0002	Sano	HOMBRE
UAMR_0013	Sano	MUJER
UAMR_0038	Sano	HOMBRE
UAMR_0039	Sano	HOMBRE
UAMR_0042	Sano	MUJER
UAMR_0045	Sano	MUJER
UAMR_0048	Sano	MUJER
UAMR_0045	Sano	MUJER

B)

Muestra	Condición	Sexo
PNTR_0001	Enfermo	MUJER
PNTR_0002	Enfermo	HOMBRE
PNTR_0003	Enfermo	MUJER
PNTR_0006	Enfermo	MUJER
PNTR_0008	Enfermo	MUJER
PNTR_0011	Enfermo	HOMBRE
PNTR_0023	Enfermo	HOMBRE
PNTR_0030	Enfermo	HOMBRE
PNTR_0054	Enfermo	HOMBRE
PNTR_0238	Enfermo	MUJER

Bloque 2: procesamiento de los datos (bash)

1. Datos de genotipado

1.1. De bfiles a VCF (plink)

El código siguiente utiliza la herramienta PLINK para convertir archivos iniciales del genotipado, que están en formato .bim/.bed/.fam (bfiles), a formato .vcf (Variant Call Format, VCF), que ha sido necesario para los pasos siguientes.

```
# Cargar el módulo de PLINK
module load plink
# Convertir archivos bfiles a formato VCF
plink --bfile bfiles --recode vcf --out vcf_multi20
```

1.2. VCFs de los cromosomas 1 a 22 (bcftools)

El archivo VCF resultante se comprime con bgzip, se indexa con tabix y luego se generan varios archivos VCF individuales, uno por cromosoma (1 al 22) usando bcftools.

```
#!/bin/bash

# Comprimir el archivo VCF inicial
bgzip vcf_multi20_NamesOk.vcf
```

```
# Generar el índice para el archivo VCF comprimido
tabix -p vcf vcf_multi20_NamesOk.vcf.gz

# Generar un vcf para cada cromosoma (del 1 al 22) a partir del vcf.gz
for i in {1..22}; do
    bcftools view -O v -o chr${i}.vcf vcf_multi20_NamesOk.vcf.gz -r ${i}
done
```

2. Datos de RNA-seq

2.1. Control de calidad (FastQC)

Se ha realizado un control de calidad inicial de las lecturas de la secuenciación de RNA con FastQC.

```
#!/bin/bash
module load fastqc/0.11.9
fastqc */*.fq.gz -o FASTQC
```

La calidad de las lecturas obtenidas ha sido alta, con valores de calidad de las bases muy superior a 30, sin secuencias sobrerrepresentadas ni un condenido de adaptadores significativo que requieran pasos adicionales para su eliminación.

2.2. Alineamiento al genoma de referencia (hisat2 y Samtools)

Después, se alinean los datos de RNA-seq al genoma de referencia (hg38) utilizando hisat2, con la opción -k 1 para mantener solo la mejor alineación de cada lectura analizada. Esto asegura que hisat2 seleccione la alineación con el menor puntaje de error, descartando posibles alineaciones alternativas. Luego, el archivo de salida SAM se convierte a BAM usando Samtools, se ordena, se indexa y se eliminan los archivos intermedios SAM.

```
#!/bin/bash
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=4
#SBATCH --cpus-per-task=4
module load hisat2/2.1.0
module load samtools/1.9
index=$1
forward=$2
reverse=$3

#Align each sample
echo Starting with sample ${index}
hisat2 --rna-strandness RF -k 1 -p 4 -x /home/irs16/genpsych_acc_directo/BIOBANK/RNASEQ/TFM_In

echo Getting into samtools, sample ${index}
samtools view -@ 4 -bo ${index}_notsorted.bam ${index}.sam

echo Removing ${index}.sam
rm ${index}.sam

echo Sorting ${index}_notsorted.bam
samtools sort -@ 4 -o ${index}.bam ${index}_notsorted.bam

echo Indexing
```

```
samtools index ${index}.bam

echo Finishing with sample ${index}
```

El código siguiente permite mandar varios trabajos al centro de computación científica de la UAM, para ejecutar el código anterior de forma automatizada para procesar los datos de las múltiples muestras de estudio. Para cada carpeta de muestra, busca los archivos forward y reverse, y si ambos existen, envía el trabajo de alineamiento a la cola del CC para ejecutar el script de alineamiento al genoma de referencia.

```
#!/bin/bash

# Ejecutar el script de alineación para cada carpeta
for carpeta in */; do
    # Obtener el nombre de la carpeta (sin la barra diagonal final)
    sample_name=${carpeta%/}

    # Buscar el archivo forward que empiece por el nombre de la muestra y termine en _1.fq.gz
    forward=$(find "$carpeta" -name "${sample_name}*_1.fq.gz" -print -quit)
    # Buscar el archivo reverse que empiece por el nombre de la muestra y termine en _2.fq.gz
    reverse=$(find "$carpeta" -name "${sample_name}*_2.fq.gz" -print -quit)

    # Verificar si los archivos forward y reverse existen
    if [[ -f "$forward" && -f "$reverse" ]]; then
        # Enviar el trabajo a la cola con los argumentos
        sbatch -A genpsych_serv -p bioinfo 1_alignment.sh "$sample_name" "$forward" "$reverse"
    else
        echo "Archivos no encontrados para la muestra: $sample_name"
    fi
done
```

2.3. Eliminación de duplicados (Samtools)

El fragmento siguiente de texto utiliza Samtools para procesar los archivos .bam crudos, eliminando los duplicados de forma aleatoria y generando un resumen de sus estadísticas asociadas. Los pasos principales que se realizan en este código son:

1. Ordenar y procesar BAM: Los archivos .bam se ordenan por nombre de consulta (queryname), se utiliza fixmate ajustar la información de las lecturas pareadas, y luego se vuelven a ordenar por coordenadas.

2. Eliminar duplicados: Utilizando markdup con la opción -r para eliminar duplicados aleatorios, generando un nuevo archivo BAM sin duplicados.

3. Generar índices y estadísticas: Se genera el índice para cada archivos .bam sin duplicados y se obtienen estadísticas con flagstat para los archivos originales y los procesados.

4. Limpieza: se eliminan los archivos intermedios no necesarios para conservar solo el archivo BAM final.

```
#!/bin/bash

module load samtools/1.16.1
```

```
# Directorios de entrada y salida
input_dir="/home/irs16/genpsych_acc_directo/BIOBANK/RNASEQ/TFM_InesSus_RNAseq/bams_1_iniciales
output_dir="/home/irs16/genpsych_acc_directo/BIOBANK/RNASEQ/TFM_InesSus_RNAseq/bams_2_nodup"

# Crear el directorio de salida si no lo hemos creado previamente
mkdir -p "$output_dir"

# Iterar sobre cada archivo .bam en el directorio de entrada
for bam_file in "$input_dir"/*.bam; do
    # Obtener el nombre base del archivo BAM sin la extensión .bam
    base_name=$(basename "$bam_file" .bam)

    # Definir los nombres de los archivos intermedios y de salida
    queryname_sorted_bam="$output_dir/${base_name}_queryname_sorted.bam"
    fixmate_bam="$output_dir/${base_name}_fixmate.bam"
    sorted_bam="$output_dir/${base_name}_sorted.bam"
    final_bam="$output_dir/${base_name}_nodup.bam"

    # Paso 1: Ordenar por nombre de consulta (queryname)
    echo "Ordenando por nombre de consulta $bam_file"
    samtools sort -n -o "$queryname_sorted_bam" "$bam_file" 2>> "$output_dir/sort_queryname_er

    # Paso 2: Ejecutar samtools fixmate y registrar errores
    echo "Ejecutando samtools fixmate para $queryname_sorted_bam"
    samtools fixmate -m "$queryname_sorted_bam" "$fixmate_bam" 2>> "$output_dir/fixmate_errors

    # Paso 3: Ejecutar samtools sort por coordenadas y registrar errores
    echo "Ejecutando samtools sort por coordenadas para $fixmate_bam"
    samtools sort -o "$sorted_bam" "$fixmate_bam" 2>> "$output_dir/sort_errors.log"

    # Paso 4: Ejecutar samtools markdup (con -r para "remove" aleatoriamente) y registrar erro
    echo "Ejecutando samtools markdup para $sorted_bam"
    samtools markdup -r "$sorted_bam" "$final_bam" 2>> "$output_dir/markdup_errors.log"

    # Paso 5: Generar el índice .bam.bai para el nuevo BAM sin duplicados
    echo "Generando índice para $final_bam"
    samtools index "$final_bam"

    # Paso 6: Generar estadísticas rápidas con flagstat para el BAM inicial
    echo "Generando estadísticas iniciales con flagstat para $bam_file"
    samtools flagstat "$bam_file" > "$output_dir/${base_name}_initial_flagstat.txt"

    # Paso 7: Generar estadísticas rápidas con flagstat para el BAM sin duplicados
    echo "Generando estadísticas con flagstat para $final_bam"
    samtools flagstat "$final_bam" > "$output_dir/${base_name}_nodup_flagstat.txt"

    # Limpiar archivos intermedios si no son necesarios
    rm "$queryname_sorted_bam"
    rm "$fixmate_bam"
    rm "$sorted_bam"

done

echo "Proceso completado. Se pueden revisar los logs de errores y las estadísticas en la carpe
```

2.4. Corrección del sesgo de mapeo (WASP)

Para poder corregir el posible sesgo de mapeo en nuestros archivos .bams necesitamos preparar dos archivos .txt que necesitaremos a continuación. Por una parte, un archivo con el identificador de las muestras de estudio, y por otra parte, un archivo con información de los cromosomas a analizar, que se puede obtener del *USCS Genome Browser*.

```
ls *.bam | sed 's/\.bam//g' > my_samples.txt
```

```
# Descargar del archivo chromInfo.txt de UCSC:
wget http://hgdownload.soe.ucsc.edu/goldenPath/hg38/database/chromInfo.txt.gz
gunzip chromInfo.txt.gz
```

El siguiente paso es generar los archivos HDF5 a partir del los VCF de los 22 cromosomas. Los archivos HDF5 son tres (haplotypes.h5, snp_index.h5, y snp_tab.h5), y contienen la información de los haplotipos y los SNPs analizados. Para generar estos archivos se utilizan los siguientes parámetros:

- –chrom: apunta al archivo que contiene la información de los cromosomas (chr y longitud, descargado de USCS)
- –format vcf: indica que el formato del archivo de entrada es VCF
- –haplotype haplotypes.h5: archivo de salida que contendrá la información de haplotipos
- –snp_index snp_index.h5: archivo de salida que contendrá el índice de SNPs
- –snp_tab snp_tab.h5: nombre del archivo de salida que contendrá la tabla de SNPs
- vcfs_samples/chr*.vcf: archivos VCF de entrada, uno por cada cromosoma, generados a partir del VCF.gz general

```
# Generación del archivo HDF5 a partir de los VCF de los 22 cromosomas y el chromInfo.txt:

/usr/local/wasp/0.3.4/snp2h5/snp2h5 --chrom chromInfo_v2.txt \
                --format vcf \
                --haplotype haplotypes.h5 \
                --snp_index snp_index.h5 \
                --snp_tab snp_tab.h5 \
                vcfs_samples/chr*.vcf
```

A continuación, se utiliza el script *find_intersecting_snps.py* de WASP para identificar lecturas que pueden tener sesgos de mapeo y que necesitan ser remapeadas, así como las lecturas que han alineado bien y no necesitan mapearse de nuevo. Este script se ejecuta para cada muestra de estudio y tiene los siguientes parámetros:

- –is_paired_end: indica que los datos de entrada son de lecturas emparejadas (paired-end)
- –is_sorted: indica que los archivos .bam de entrada están ordenados por coordenadas
- –output_dir: especifica el directorio de salida donde se guardan los resultados

- –snp_tab, –snp_index, –haplotype: son los archivos HDF5 que contienen información sobre los SNPs y los haplotipos
- –samples: el nombre de las muestras que se están procesando
- BAM_DIR/${SAMPLE_NAME}_nodup.bam: El archivo .bam correspondiente a la muestra, donde ya se han eliminado los duplicados en los pasos anteriores

```
#!/bin/bash

#SBATCH --nodes=4
#SBATCH --ntasks-per-node=4
#SBATCH --cpus-per-task=4

# Cargar módulos necesarios
module load python/3.7.7
module load samtools/1.9
module load hisat2/2.1.0
module load miniconda/3.7

# Variables de entrada
SNP_TAB="/home/irs16/genpsych_acc_directo/BIOBANK/RNASEQ/TFM_InesSus_RNAseq/bfiles_y_vcf/HDF5_
SNP_INDEX="/home/irs16/genpsych_acc_directo/BIOBANK/RNASEQ/TFM_InesSus_RNAseq/bfiles_y_vcf/HDF
HAPLOTYPE="/home/irs16/genpsych_acc_directo/BIOBANK/RNASEQ/TFM_InesSus_RNAseq/bfiles_y_vcf/HDF
OUTPUT_DIR="/home/irs16/genpsych_acc_directo/BIOBANK/RNASEQ/TFM_InesSus_RNAseq/bams_3_wasp"

# Crear el directorio de salida si no existe
mkdir -p ${OUTPUT_DIR}

# Archivo que contiene la lista de muestras
SAMPLES_FILE="my_samples.txt"

# Directorio donde están los BAMs
BAM_DIR="/home/irs16/genpsych_acc_directo/BIOBANK/RNASEQ/TFM_InesSus_RNAseq/bams_2_nodup"

##############################################################################################
# Bloque 1:
echo "Iniciando Bloque 1"

# Iterar sobre cada muestra
while read SAMPLE_NAME; do
    echo "Identificando lecturas que pueden tener sesgos de mapeo para la muestra ${SAMPLE_NAM

    # Ejecutar el script find_intersecting_snps.py para cada muestra
    python /usr/local/wasp/0.3.4/mapping/find_intersecting_snps.py \
           --is_paired_end \
           --is_sorted \
           --output_dir ${OUTPUT_DIR} \
           --snp_tab ${SNP_TAB} \
           --snp_index ${SNP_INDEX} \
           --haplotype ${HAPLOTYPE} \
           --samples ${SAMPLE_NAME} \
           ${BAM_DIR}/${SAMPLE_NAME}_nodup.bam  # El archivo BAM de entrada
```

```
    echo "Finalizando identificación de SNPs para la muestra ${SAMPLE_NAME}."

done < ${SAMPLES_FILE}
```

Una vez idenficadas las lecturas que necesitan mapearse de nuevo, se utiliza el remapeo de estas lecturas con la misma herramienta que utilizamos anteriormente, hitsat2. De nuevo, los alineamientos se convierten a formato .bam, y luego se ordenan e indexan con Samtools, obteniendo un archivo .bam final con las lecturas que se han remapeado.

```
#!/bin/bash

# Cargar módulos necesarios
module load samtools/1.9
module load hisat2/2.1.0

# Variables de entrada
SNP_TAB="/home/irs16/genpsych_acc_directo/BIOBANK/RNASEQ/TFM_InesSus_RNAseq/bfiles_y_vcf/HDF5_
SNP_INDEX="/home/irs16/genpsych_acc_directo/BIOBANK/RNASEQ/TFM_InesSus_RNAseq/bfiles_y_vcf/HDF
HAPLOTYPE="/home/irs16/genpsych_acc_directo/BIOBANK/RNASEQ/TFM_InesSus_RNAseq/bfiles_y_vcf/HDF
OUTPUT_DIR="/home/irs16/genpsych_acc_directo/BIOBANK/RNASEQ/TFM_InesSus_RNAseq/bams_3_wasp"

# Remapeo de lecturas con potencial sesgo de mapeo
for remap_fq1 in ${OUTPUT_DIR}/*.remap.fq1.gz; do
    # Extraer el nombre de la muestra del nombre del archivo
    SAMPLE_NAME=$(basename ${remap_fq1} .remap.fq1.gz)
    echo "Nombre de la muestra ${SAMPLE_NAME} extraído con éxito."

    # Especificar el archivo de lectura emparejada
    remap_fq2=${OUTPUT_DIR}/${SAMPLE_NAME}.remap.fq2.gz

    echo "Remapeando lecturas para la muestra ${SAMPLE_NAME}..."

    # Ejecutar hisat2 para remapear las lecturas emparejadas
    hisat2 --rna-strandness RF -k 1 -p 4 -x /home/irs16/genpsych_acc_directo/BIOBANK/RNASEQ/TF
           -1 ${remap_fq1} \
           -2 ${remap_fq2} \
           -S ${OUTPUT_DIR}/${SAMPLE_NAME}_remap.sam 2>> summary_remapping_${SAMPLE_NAME}.txt

    echo "Convirtiendo ${SAMPLE_NAME}_remap.sam a BAM..."

    # Convertir a BAM
    samtools view -@ 4 -bo ${OUTPUT_DIR}/${SAMPLE_NAME}_remap_notsorted.bam ${OUTPUT_DIR}/${SA

    echo "Eliminando ${SAMPLE_NAME}_remap.sam"
    rm ${OUTPUT_DIR}/${SAMPLE_NAME}_remap.sam

    echo "Ordenando ${SAMPLE_NAME}_remap_notsorted.bam..."
    samtools sort -@ 4 -o ${OUTPUT_DIR}/${SAMPLE_NAME}_remap.bam ${OUTPUT_DIR}/${SAMPLE_NAME}_

    echo "Indexando ${SAMPLE_NAME}_remap.bam..."
    samtools index ${OUTPUT_DIR}/${SAMPLE_NAME}_remap.bam
```

```
    echo "Finalizando remapeo de lecturas para la muestra ${SAMPLE_NAME}."
done
```

A partir de los archivos .bam con las lecturas remapeadas, se realiza el filtrado de estas lecturas, de forma que si han alineado en el mismo sitio, se consevan, y si no, se descartan. Para ello se ejecuta el script de WASP *filter_remapped_reads.py*. Este script necesita tres archivos .bam de entrada: el archivo con lecturas que deben ser remapeadas (to_remap_bam), el archivo con las lecturas remapeadas (remap_bam), y el archivo donde se almacenarán las lecturas que se deben mantener después del filtrado (keep_bam). El filtrado se realiza para cada muestra de forma secuencial.

```
#!/bin/bash

#SBATCH --nodes=4
#SBATCH --ntasks-per-node=4
#SBATCH --cpus-per-task=4

# Cargar módulos necesarios
module load samtools/1.9
module load hisat2/2.1.0
module load miniconda/3.7

# Variables de entrada
SNP_TAB="/home/irs16/genpsych_acc_directo/BIOBANK/RNASEQ/TFM_InesSus_RNAseq/bfiles_y_vcf/HDF5_
SNP_INDEX="/home/irs16/genpsych_acc_directo/BIOBANK/RNASEQ/TFM_InesSus_RNAseq/bfiles_y_vcf/HDF
HAPLOTYPE="/home/irs16/genpsych_acc_directo/BIOBANK/RNASEQ/TFM_InesSus_RNAseq/bfiles_y_vcf/HDF
OUTPUT_DIR="/home/irs16/genpsych_acc_directo/BIOBANK/RNASEQ/TFM_InesSus_RNAseq/bams_3_wasp"

# Archivo que contiene la lista de muestras
SAMPLES_FILE="my_samples.txt"

# Filtrar lecturas remapeadas
while read SAMPLE_NAME; do
    echo "Filtrando lecturas remapeadas para la muestra ${SAMPLE_NAME}..."

    # Especificar los archivos de entrada y salida
    to_remap_bam=${OUTPUT_DIR}/${SAMPLE_NAME}_nodup.to.remap.bam
    remap_bam=${OUTPUT_DIR}/${SAMPLE_NAME}_nodup_remap.bam
    keep_bam=${OUTPUT_DIR}/${SAMPLE_NAME}_nodup.remap.keep.bam

    # Ejecutar el script de filtrado
    python /usr/local/wasp/0.3.4/mapping/filter_remapped_reads.py \
           ${to_remap_bam} \
           ${remap_bam} \
           ${keep_bam}

    echo "Finalizando filtrado de lecturas remapeadas para la muestra ${SAMPLE_NAME}."

done < ${SAMPLES_FILE}
```

Finalmente, se utiliza samtools merge para fusionar los archivos .bam que contienen las lecturas que se deben conservar de cada muestra, es decir, el archivo de lecturas originales y el archivo de lecturas

remapeadas en un único archivo .bam para cada muestra analizada (_keep.merge.bam).

Además, se ordena el archivo fusionado utilizando samtools sort, generando un archivo ordenado (_keep.merge.sort.bam); y se elimina el archivo .bam fusionado no ordenado para ahorrar espacio en disco.

Finalmente, se indexa el archivo .bam ordenado con samtools index para facilitar el acceso rápido a las posiciones de las lecturas en los pasos siguientes del estudio.

```
#!/bin/bash

#SBATCH --nodes=4
#SBATCH --ntasks-per-node=4
#SBATCH --cpus-per-task=4

# Cargar módulos necesarios
module load samtools/1.9
module load hisat2/2.1.0
module load miniconda/3.9

# Variables de entrada
SNP_TAB="/home/irs16/genpsych_acc_directo/BIOBANK/RNASEQ/TFM_InesSus_RNAseq/bfiles_y_vcf/HDF5_
SNP_INDEX="/home/irs16/genpsych_acc_directo/BIOBANK/RNASEQ/TFM_InesSus_RNAseq/bfiles_y_vcf/HDF
HAPLOTYPE="/home/irs16/genpsych_acc_directo/BIOBANK/RNASEQ/TFM_InesSus_RNAseq/bfiles_y_vcf/HDF
OUTPUT_DIR="/home/irs16/genpsych_acc_directo/BIOBANK/RNASEQ/TFM_InesSus_RNAseq/bams_3_wasp"

# Archivo que contiene la lista de muestras
SAMPLES_FILE="my_samples.txt"

# Paso final: Fusionar, ordenar e indexar los BAMs filtrados
for SAMPLE_NAME in $(cat ${SAMPLES_FILE}); do
    echo "Fusionando BAMs para la muestra ${SAMPLE_NAME}..."

    # Fusionar los archivos BAM filtrados
    samtools merge ${OUTPUT_DIR}/${SAMPLE_NAME}_nodup.keep.merge.bam \
                   ${OUTPUT_DIR}/${SAMPLE_NAME}_nodup.keep.bam \
                   ${OUTPUT_DIR}/${SAMPLE_NAME}_nodup.remap.keep.bam

    echo "Ordenando el archivo fusionado ${SAMPLE_NAME}_nodup.keep.merge.bam..."

    # Ordenar el archivo BAM fusionado
    samtools sort -o ${OUTPUT_DIR}/${SAMPLE_NAME}_nodup.keep.merge.sort.bam \
                  ${OUTPUT_DIR}/${SAMPLE_NAME}_nodup.keep.merge.bam

    # Eliminar el archivo intermedio no ordenado
    rm ${OUTPUT_DIR}/${SAMPLE_NAME}_nodup.keep.merge.bam

    echo "Indexando el archivo ordenado ${SAMPLE_NAME}_nodup.keep.merge.sort.bam..."

    # Indexar el archivo BAM ordenado
    samtools index ${OUTPUT_DIR}/${SAMPLE_NAME}_nodup.keep.merge.sort.bam

    echo "Finalizado el proceso para la muestra ${SAMPLE_NAME}."
```

```
done

echo "Finalizado con éxito."
```

Una vez realizados estos pasos de procesado de los archivos .bam se pueden sacar tablas y gráficos que resuman el número de lecturas que se utilizan en cada paso de filtrado, para así poder evaluar el rendimiento de estas estrategias. En este TFM hemos representado estos resultados en boxplots, como se ve en la siguiente figura:

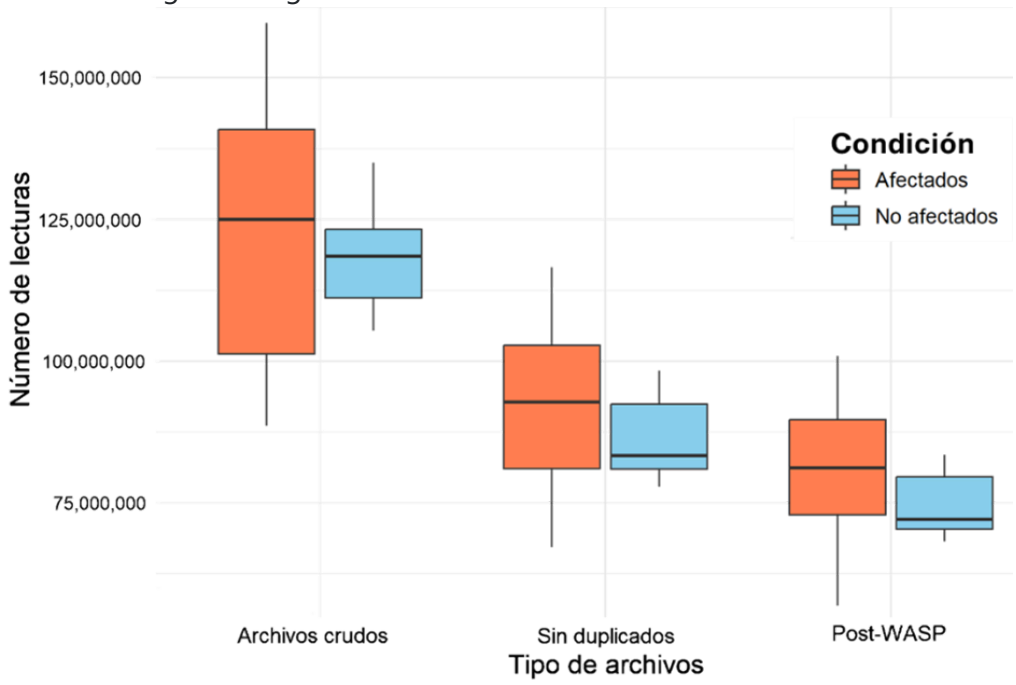


Figura 3. Distribución del número de lecturas de RNA-seq por paso de procesamiento y condición clínica. Se muestran tres etapas: lecturas crudas, tras la eliminación de duplicados y tras la corrección del sesgo de mapeo.

Bloque 3: análisis de expresión alélica específica (AllelicImbalance, R)

1. Obtención de recuentos y análisis de ASE

En el siguiente script se utiliza el paquete *AllelicImbalance* para realizar el análisis de expresión alélica específica (ASE). A partir de los archivos .bams y el vcf multimuestra generados previamente, se realiza un filtrado de las variantes, para descartar la información de aquellas que son heterocigotas o tienen pocas lecturas. También se aplican pruebas estadísticas para identificar posibles desequilibrios alélicos asociados con el trastorno bipolar. El análisis se enfoca en comparar individuos afectados y no afectados (casos *vs* controles).

Primero, instalamos y cargamos los paquetes necesarios en R. Entre los paquetes utilizados se encuentran *AllelicImbalance*, *VariantAnnotation* y *GenomicAlignments*, fundamentales para procesar y analizar datos de secuenciación genómica y de RNA-Seq. Se definen también los directorios de trabajo y las rutas de acceso a los archivos de entrada, que incluyen archivos .vcf (con información de variantes genéticas), archivos .bam (con lecturas alineadas de RNA-Seq) y un excel (.xlxs) con datos fenotípicos sobre las muestras, que en nuestro caso es el estado afectado o no afectado por trastorno bipolar de los individuos.

```
setwd("C:/irs_tfm")
getwd()
```

```
##############################################################################
### 1. Instalar los paquetes necesarios (AllelicImbalance y VariantAnnotation)
if (!requireNamespace("BiocManager", quietly = TRUE))
  install.packages("BiocManager")

BiocManager::install(c("AllelicImbalance", "VariantAnnotation",
                       "GenomicAlignments", "SNPlocs.Hsapiens.dbSNP144.GRCh38"))

install.packages("readxl")

if (!requireNamespace("biomaRt", quietly = TRUE)) {
  install.packages("biomaRt")
}

# Instalar dplyr si aún no lo tienes
if (!requireNamespace("dplyr", quietly = TRUE)) {
  install.packages("dplyr")
}

# Cargar AllelicImbalance, VariantAnnotation y otros paquetes necesarios
library(AllelicImbalance)
library(VariantAnnotation)
library(GenomicAlignments)
library(readxl)
library(SNPlocs.Hsapiens.dbSNP144.GRCh38)
library(biomaRt)
library(dplyr)
library(tidyr)

##############################################################################
### 2. Rutas a los archivos necesarios:
# Definir la ruta al archivo VCF (VCF multisample)
pathToVcf <- "C:/irs_tfm/vcf_multi.vcf"

# Definir el directorio donde se encuentran los BAMs
pathToFiles <- "X:/ASE_irs/bams_ase"

# Se carga el excel con los fenotipos de los individuos (affected vs unaffected)
fenotipo <- read_excel("C:/irs_tfm/fenotipo_cond.xlsx")
```

En el siguiente paso, se lee el archivo .vcf multimuestra con la función *readVcf()* del paquete *VariantAnnotation*. Una vez cargadas las variantes, se aplica un filtrado para conservar únicamente aquellas que son bialélicas, es decir, que presentan un alelo de referencia y un solo alelo alternativo, lo que facilita el análisis de expresión alélica específica. Después, el objeto VCF se convierte en un objeto *GRanges*, necesario para trabajar con el paquete **AllelicImbalance** de *Bioconductor*, que permite acceder de manera eficiente a la información de las variantes analizadas.

```
##############################################################################
### 3. Leer el archivo VCF (VCF: info de todos los individuos) y lo filtramos
VCF_raw <- readVcf(pathToVcf, "hg38")
# Eliminamos los posibles duplicados y el archivo con duplicados para liberar espacio
VCF <- unique(VCF_raw)
rm(VCF_raw)
cat("Archivo VCF cargado exitosamente.\n")
```

```r
# Convertir el VCF en un objeto GRanges
gr <- granges(VCF)

# Filtrado: quedarnos solo con las variantes bialélicas (Alelos: 1 REF y 1 ALT)
gr.filt <- gr[width(mcols(gr)[,"REF"]) == 1 &
                sapply(mcols(gr)[,"ALT"], function(alt) length(alt) == 1 &&
                         !is.na(alt) && nchar(alt) > 0)]
save(gr.filt, file = "gr.filt.RData")

cat("Filtrado completado: se han retenido solo variantes bialélicas.\n")

# Filtrar directamente los primeros 22 niveles de gr.filt_chr (chr 1 al 22)
gr.filt <- keepSeqlevels(gr.filt, seqlevels(gr.filt)[1:22], pruning.mode = "coarse")

# Filtrar variantes bialélicas: REF de longitud 1 y ALT con exactamente un alelo
# Comprobar si todas las variantes son bialélicas (en todas las posiciones)
biallelic_check <- sapply(mcols(gr.filt)$ALT, function(alt) length(alt) == 1)
result_bialelic_check <- sum(biallelic_check) == length(gr.filt)
cat("Verificación de variantes bialélicas completada: ", result_bialelic_check, "\n")
# Debería salir TRUE. Sí, sale TRUE

# Guardar el resultado en un txt
save(result_bialelic_check, file = "result_bialelic_check.Rdata")

# Función readGT: permite leer la info de fase del archivo VCF de partida
geno_fullVCF <- geno(VCF)$GT
save(geno_fullVCF, file = "geno_fullVCF.Rdata")
cat("Información genotípica del VCF leida correctamente. \n")

################################################################################
```

A continuación, se genera un bucle iterativo de análisis, en el que se analiza secuencialmente la información de los 22 cromosomas autosómicos, permitiendo optimizar el uso de memoria y organizar los resultados de manera estructurada. Los pasos principales de este apartado son:

1) Definición de tamaños cromosómicos: Primero, se definen las longitudes de los cromosomas autosómicos (hg38), necesarias para establecer rangos de búsqueda en las lecturas de RNA-seq.

2) Iteración sobre cromosomas: El bucle *for* itera a través de cada cromosoma, creando un entorno específico para el análisis individual:

- **Filtrado de variantes:** se seleccionan las variantes del cromosoma actual (*gr.filt_chr*) a partir de un conjunto de datos filtrado previamente (*gr.filt*). Se eliminan duplicados y se mantiene solo la información de los cromosomas 1 a 22.
- **Carga de lecturas de RNA-seq:** utilizando la función *impBamGAL*, se extraen las lecturas correspondientes al cromosoma actual. Se ajustan las longitudes cromosómicas para garantizar la coherencia entre .bams y variantes.
- **Recuento de alelos:** la función *getAlleleCounts* calcula los conteos alélicos para las posiciones de interés. Se guardan estos datos y se eliminan las lecturas .bams el ambiente de R para liberar memoria.

3) Filtrado de recuentos alélicos: Se aplica un proceso de filtrado para cada posición. Se descartan los recuentos totales inferiores a 10 y se eliminan los recuentos de los individuos homocigotos para cada variante. Posteriormente, se genera un objeto *ASEset* que incorpora la información de fase de las variantes.

4) Optimización y filtrado final: Las variantes con recuentos completamente nulos)=0) se descartan, y se crea un objeto *ASEset* definitivo con las variantes y recuentos filtrados. La información de fase se añade nuevamente, garantizando que solo se consideren variantes heterocigotas.

```
################################################################################
### 4. Trabajo con un solo cromosoma en cada iteración:

# Definir los tamaños de los cromosomas humanos (autosómicos) (versión hg38)
chr_sizes <- c(248956422, 242193529, 198295559, 190214555, 181538259,
               170805979, 159345973, 145138636, 138394717, 133797422,
               135086622, 133275309, 114364328, 107043718, 101991189,
               90338345, 83257441, 80373285, 58617616, 64444167,
               46709983, 50818468) #Se sacan de SNPlocs.Hsapiens.dbSNP144.GRCh38

# Crear un bucle que itere a través de los cromosomas 1 al 22 (autosómicos)
for (chr in 5:1) {

  # Definir el nombre del cromosoma actual y la longitud del cromosoma
  chr_name <- paste0("chr", chr)
  chr_length <- chr_sizes[chr]

  # Crear un directorio específico para guardar los resultados del chr actual
  result_dir <- paste0("resultados_", chr_name) # Directorio para guardar resultados
  dir.create(result_dir, showWarnings = FALSE)

  # Informar de que el proceso ha comenzado para el cromosoma actual
  cat(paste0("Iniciando análisis para ", chr_name, "...\n"))

  ##########################################
  ### 4.1. Filtrar variantes del cromosoma estudiado del gr o vcf total
  gr.filt_chr <- gr.filt[seqnames(gr.filt) == c(chr)]

  # Eliminamos los posibles duplicados
  gr.filt_chr <- gr.filt_chr[!duplicated(gr.filt_chr)]

  ###### ###### ###### ###### ###### ###### ###### ###### ###### ###### ###### ######

  # Filtrar para conservar solo los cromosomas 1 al 22 en el objeto gr_filt_chr
  chromosomes_to_keep <- paste0("chr", 1:22)  # Lista de los cromosomas del 1 al 22

  # Crear un nuevo objeto con solo estos cromosomas
  gr.filt_chr <- keepSeqlevels(gr.filt_chr, 1:22, pruning.mode = "coarse")

  ###### ###### ###### ###### ###### ###### ###### ###### ###### ###### ###### ######

  # Comprobar cuántas variantes quedan tras el filtrado
  n_var_chr <- length(gr.filt_chr)
```

```r
# Guardar nº de variantes por chr
save(n_var_chr, file = paste0(result_dir, "/n_var_", chr_name, ".RData"))
# Información en consola:
cat(paste0("Número de variantes en ", chr_name, ": ", length(gr.filt_chr), "\n"))

##########################################
### 4.2. Filtrar los BAMs para sacar la información del cromosoma estudiado

# Establecer el área de búsqueda para el cromosoma concreto
searchArea_chr <- GRanges(seqnames = c(chr),
                          ranges = IRanges(start = 1, end = (chr_length)))

# Cargar la información del RNAseq del cromosoma estudiado (reads = lecturas de los BAMs)
reads_chr <- impBamGAL(pathToFiles, searchArea_chr, verbose = TRUE)
cat("Lecturas del cromosoma ", chr_name, " cargadas exitosamente.\n")

# Mantener solo los primeros 22 cromosomas en reads_chr
reads_chr <- keepSeqlevels(reads_chr, seqlevels(reads_chr)[1:22], pruning.mode = "coarse")
# ajustamos la longitud de los cromosomas para que coincidan entre bams y vcf
seqlengths(gr.filt_chr)[1:22] <- seqlengths(reads_chr)

##########################################
### 4.3. Recuento de los alelos que hay en las reads para las posiciones filtradas
countList_chr <- getAlleleCounts(reads_chr, gr.filt_chr, verbose = TRUE)
cat("Recuento de alelos realizado para el cromosoma ", chr_name, ".\n")

# Mostrar el resultado (solo una parte inicial con head)
print(head(countList_chr))

# Guardar los recuentos resultantes (iniciales o crudos)
save(countList_chr, file = paste0(result_dir, "/countList_", chr_name, ".RData"))
cat("Recuendos alélicos del cromosoma ", chr_name, " guardados con éxito .\n")

# Eliminamos las lecturas de los bams para reducir la carga en memoria
rm(reads_chr)
cat("Lecturas de los bams eliminadas del enviroment con éxito.")

##########################################
### 4.4. Filtrado de los recuentos a partir de la countList

# Inicializa una lista para almacenar los resultados filtrados
filtered_countList_chr <- list()

# Itera sobre cada posición en countList
for (pos in names(countList_chr)) {
  # Extrae los conteos de la posición actual
  counts <- countList_chr[[pos]]

  # Itera sobre cada fila (cada persona) para verificar los totales
  for (i in 1:nrow(counts)) {
    total_counts <- sum(counts[i, ])  # Suma de lecturas para esa persona

    # Si el total es menor que 10, establece toda la fila a 0
    if (total_counts < 10) {
```

```
      counts[i, ] <- 0
      cat("Fila ", i, " establecida a 0 para variante ", pos, "\n")
    }
  }

  filtered_countList_chr[[pos]] <- counts
  # }
}

# Mostrar el resultado del filtrado de los recuentos
cat("Filtrado de recuentos completado para el cromosoma ", chr, ".\n")
print(head(filtered_countList_chr))

##############################################
### 4.5. Crear el objeto ASEset inicial a partir de la lista de conteos filtrada

ase_chr <- ASEsetFromCountList(rowRanges = gr.filt_chr,
                               filtered_countList_chr)
cat("Objeto ASEset inicial creado para el cromosoma ", chr,".\n")

##############################################
### 4.6. Añadir información de fase (genotipo: homocigotos o heterocigotos)
      # Info de GT se ha extraido previamente para todas las posiciones del VCF con:
      # geno_fullVCF <- geno(VCF)$GT

# Extraer las posiciones de gr.filt_chr
positions_chr <- start(gr.filt_chr)

# Filtrar el VCF para las variantes del cromosoma usando las posiciones extraídas
filt_indices_chr <- which(seqnames(VCF) == chr &
                            start(granges(VCF)) %in% positions_chr)

# Extraer la información de genotipos solo para las variantes filtradas
geno_chr_filt <- geno_fullVCF[filt_indices_chr,]

# Ajustar la información de fase a las dimensiones de `ase_chr` automáticamente
required_rows <- nrow(ase_chr)  # Número de filas necesario basado en `ase_chr`
geno_chr_filt_adjusted <- geno_chr_filt[1:required_rows, ]  # Ajustar filas

# Confirmar dimensiones antes de asignar
cat("Dimensiones de geno_chr_filt_adjusted:", dim(geno_chr_filt_adjusted), "\n")
cat("Dimensiones de ase_chr:", dim(ase_chr), "\n")

# Asignar la información de fase al objeto ASEset inicial
phase(ase_chr) <- geno_chr_filt_adjusted

# Checkear el funcionamiento con:
cat("Información de fase añadida a ASEset inicial para el cromosoma ", chr, ".\n")
print(head(phase(ase_chr)))

##############################################
### 4.7. Filtrado de Heterocigotos y Recuentos = 0
        # Generación del objeto ASEset definitivo
```

```r
# Fase (phase information) y recuentos (allele counts)
phase_info_chr <- phase(ase_chr)

# 1. Primer bucle: Poner a 0 los recuentos si son homocigotos o NA
# Iterar sobre cada variante en la lista de recuentos
for (variante in names(filtered_countList_chr)) {

  # Recorrer cada individuo en la fase y recuentos
  for (individuo in colnames(phase_info_chr)) {

    # Si el individuo es homocigoto ("0/0", "1/1", "0|0", "1|1") o NA
    if (phase_info_chr[variante, individuo] %in% c("0/0", "1/1", "0|0", "1|1", "./.")
        || is.na(phase_info_chr[variante, individuo])) {
      # Poner a 0 todos los recuentos para esa variante y ese individuo
      filtered_countList_chr[[variante]][individuo, ] <- 0
    }

    # En caso contrario, es decir, si el individuo es heterocigoto
    # ("0/1", "1/0", "0|1", "1|0"), no hacemos nada (los recuentos se mantienen)
  }
cat(chr_name, "Variante:", variante, "Recuentos de homocigotos = 0.\n")
}

# 2. Segundo bucle: Verificar si todos los recuentos son 0 y eliminar variantes si es necesa

# Crear una lista para almacenar las variantes eliminadas y conservadas
var_delete_chr <- c()
filtered_countList_chr_def <- list()

for (variante in names(filtered_countList_chr)) {

  # Extraer los recuentos para la variante actual
  counts <- filtered_countList_chr[[variante]]

  # Verificamos si todos los valores son 0 para todas las personas
  if (all(counts == 0)) {
    # Si todos son ceros, añadimos esta variante a la lista de eliminadas
    var_delete_chr <- c(var_delete_chr, variante)
  } else {
    # Si no todos son ceros, añade esta posición a la lista filtrada
    filtered_countList_chr_def[[variante]] <- counts
  }
cat("Eliminada la variante ",variante,"por recuentos = 0 del", chr_name,".\n")
}

# Extraer solo las posiciones numéricas de las variantes eliminadas
# Asumiendo que los nombres son del formato "chrn_posicion"
pos_delete_chr <- as.numeric(sub(".*_", "", var_delete_chr))


# Filtrar el objeto GRanges eliminando las variantes en las posiciones indicadas
gr.filt_chr_def <- gr.filt_chr[!start(gr.filt_chr)
                               %in% pos_delete_chr]
```

```r
# Guardar los recuentos resutlantes
save(filtered_countList_chr, file = paste0(result_dir, "/filtered_countList_chr"
                                           , chr, ".RData"))
cat("Recuentos filtrados del cromosoma", chr, " guardados correctamente.\n")

##############################################
### 4.8. Crear el objeto ASEset definitivo
# (con los recuentos filtrados previamente, en dos pasos anteriores)

ase_chr_def <- ASEsetFromCountList(rowRanges = gr.filt_chr_def,
                                   filtered_countList_chr_def)
cat("Objeto ASEset creado para el ", chr_name, ".\n")

# Añadir la información de fase al objeto ASEset definitivo
# Solo miramos las variantes que han pasado los filtros previos
positions_def_chr <- start(gr.filt_chr_def)

# Filtrar el VCF para las variantes del cromosoma usando las posiciones extraídas
phase_indices_chr <- which(seqnames(VCF) == chr &
                             start(granges(VCF)) %in% positions_def_chr)

# Extraer la información de genotipos solo para las variantes filtradas
phase(ase_chr_def) <- geno_fullVCF[phase_indices_chr, ]

##############################################
### 4.9. INFORMACIÓN FENOTÍPICA:

# Cargar el archivo Excel en R y transformarlo a Data Frame
feno_data <- DataFrame(Condition = fenotipo$Condition)

# Añadir el identificador de las muestras al Data Frame anterior
rownames(feno_data) <- fenotipo$Sample

# Incluir el fenotipo en el objeto ASEset
colData(ase_chr_def) <- feno_data

# Para poder checkear que se ha cargado bien podemos utilizar:
cat("Información fenotípica añadida con éxito para el", chr_name,"\n")
print(head(colData(ase_chr_def)))

# O también para un individuo concreto:
# colData(ase_chr_def)["identificador_individuo.bam", "Condition"]

##############################################
### 4.10. INFO REF Y ALT ALLELES

# Extraer los alelos de referencia desde la columna REF del gr.filt correspondiente
ref_alleles_chr <- as.character(mcols(gr.filt_chr_def)[, "REF"])

# Extraer los alelos alternativos desde la columna ALT del gr.filt correspondiente
alt_alleles_chr <- sapply(mcols(gr.filt_chr_def)$ALT, function(x) as.character(x))

# Se extraen los alelos REF y ALT de forma distinta porque se almacenan de forma distinta en
```

```r
# DNAStringSet (REF): Se puede convertir directamente a caracteres.
# DNAStringSetList (ALT): Se debe extraer cada elemento antes de convertir.

# Verificar los alelos de referencia extraídos
print(head(ref_alleles_chr))
print(head(alt_alleles_chr))

# Asignar estos alelos de referencia y alternativo objeto ASEset
ref(ase_chr_def) <- ref_alleles_chr
alt(ase_chr_def) <- alt_alleles_chr
cat("Información Ref y Alt alleles añadida con éxito para el", chr_name, ".\n")

# Verificar la asignación
cat("Alelo de referencia y alternativo asignado en ASEset:\n")
print(head(ref(ase_chr_def)))
print(head(alt(ase_chr_def)))

##############################################
### 4.11. INFO GENOTYPE

# Inferir los genotipos y añadirlos en el campo correspondiente del objeto ASEset
genotype(ase_chr_def) <- inferGenotypes(ase_chr_def)
cat("Información genotípica añadida con éxito para el", chr_name, ".\n")

##############################################
### 4.12. Check Mapping Bias (average reference allele fraction)

# Sacar la fracción del alelo de referencia
refFrac_chr <- fraction(ase_chr_def, top.fraction.criteria="ref")

# obtener la media
refFrac_mean_chr <- mean(refFrac_chr, na.rm = TRUE)

# Imprimir el resultado:
cat("Fracción media del alelo de referencia:", refFrac_mean_chr, "\n")
save(refFrac_mean_chr, file = paste0(result_dir, "/refFrac_mean_", chr_name, ".RData"))

##############################################
### 4.13. Análisis estadístico y Filtrado por significación estadística

# guardar el objeto ASE set definitivo antes de continuar
save(ase_chr_def, file = paste0(result_dir, "/ase_chr_def_", chr, ".RData"))

######################     ######################     ######################
    ##### Test Binomial (AllelicImbalance-Bioconductor):
######################     ######################     ######################

binom_results_chr <- binom.test(ase_chr_def, n = '*')
cat("Test binomial realizado con éxito para el ", chr_name, ".\n")

# Creamos un objeto para almacenar los resutlados significativos
# Cambiamos NaN por NA para facilitar el manejo de esta información
binom_results_chr[binom_results_chr == "NaN"] <- NA
```

```r
# Convierte el resultado a un dataframe
binom_df_chr <- as.data.frame(binom_results_chr)

# Crea un dataframe vacío para almacenar los resultados significativos
signif_df_chr <- data.frame(matrix(ncol = ncol(binom_df_chr), nrow = 0))

colnames(signif_df_chr) <- colnames(binom_df_chr)

# Inicializar una lista para almacenar las variantes significativas
binom_sig_variants_chr <- c()

# Bucle para analizar las filas y columnas
for (i in 1:nrow(binom_df_chr)) {
  for (j in 1:ncol(binom_df_chr)) {
    if (!is.na(binom_df_chr[i, j]) && binom_df_chr[i, j] <= 0.05) {
      # Almacena el valor significativo en el nuevo dataframe
      signif_df_chr <- rbind(signif_df_chr, data.frame(Sample = rownames(binom_df_chr)[i],
                                                       Variant = colnames(binom_df_chr)[j]
                                                       Value = binom_df_chr[i, j],
                                                       Test = "Binomial"))
      # Añadir el nombre de la variante significativa a la lista
      binom_sig_variants_chr <- c(binom_sig_variants_chr, colnames(binom_df_chr)[j])
    }
  }
}
cat("Variantes significativas extraídas con éxito para el", chr_name, ".\n")

# Verifica si no se han encontrado resultados significativos
if (nrow(signif_df_chr) == 0) {
  # Si no hay resultados significativos, dejar el dataframe vacío
  signif_df_chr <- signif_df_chr(Sample = character(),
                                 Variant = character(),
                                 Value = numeric(),
                                 Test = character(),
                                 Strand = character(),
                                 stringsAsFactors = FALSE)

  cat("No hay variantes significativas en el test binomial del ", chr_name, ".\n")
}

# Imprimir el dataframe de resultados significativos
print(head(signif_df_chr))

# Guardar los resultados del análsis estadístico
save(signif_df_chr, file = paste0(result_dir, "/signif_df_", chr_name, ".RData"))

# Guardar el número de variantes significativas con este test estadístico
num_var_binom_chr <- nrow(signif_df_chr)
save(num_var_binom_chr, file = paste0(result_dir, "/num_var_binom_", chr_name, ".RData"))

# Guardar los nombres de las variantes únicas significativas del test estadístico
unique_binom_sig_var_chr <- unique(binom_sig_variants_chr)
save(unique_binom_sig_var_chr, file = paste0(result_dir, "/unique_binom_sig_var_", chr_name,
```

```
  cat("Información del test binomial del ", chr_name," guardado con éxito.\n")

  ##################################################################################################
  ##################################################################################################
  # 4.14. IDENTIFICADOR DE LAS VARIANTES CONOCIDAS (identificador rs conocido)

  ### SOLUCIÓN: solo con binom.test:

  # Filtrar el objeto rowRanges del ASEset para quedarnos solo con las variantes comunes signi
  filtered_gr_chr <- rowRanges(ase_chr_def)[names(rowRanges(ase_chr_def))
                                            %in% unique_binom_sig_var_chr]

  # Confirmamos que las longitudes de los cromosomas coincidan:
  seqlengths(filtered_gr_chr)[1:22] <- seqlengths(SNPlocs.Hsapiens.dbSNP144.GRCh38)[1:22]

  # Cambiamos las localizaciones por el rs
  rs_signif_var_chr <- getSnpIdFromLocation(filtered_gr_chr, SNPlocs.Hsapiens.dbSNP144.GRCh38)

  # Creo un nuevo objeto ASEset solo con la información de las posiciones significativas

    # Identificar las posiciones que están tanto en ase_chr_def como en filtered_gr_chr
  pos_to_keep_chr <- which((rownames(ase_chr_def)) %in% unique_binom_sig_var_chr)

  # Filtrar el ASEset original para quedarse solo con las posiciones de interés
  ase_chr_def_v2 <- ase_chr_def[pos_to_keep_chr, ]

  # Podemos añadir estos rs al objeto ASEset para utilziarlos en los gráficos posteriores
  rowRanges(ase_chr_def_v2) <- rs_signif_var_chr

  # Extraer los rs conocidos en una lista
  rs_known_list_chr <- names(rs_signif_var_chr[!grepl(chr_name,"_", names(rs_signif_var_chr))])

  # Guardamos el objeto resultante
  save(rs_known_list_chr, file = paste0(result_dir, "/rs_known_list_", chr_name,".RData"))
  cat("Identificadores rs del cromosoma", chr, "guardados con éxito.\n")

  ##################################################################################################
  ##################################################################################################

  cat(" ##########################################\n",
      "Análisis completado para ", chr_name,"\n",
      "##########################################\n")

  # Limpiar el entorno dejando solo los archivos necesarios para la siguiente iteración
  rm(list = setdiff(ls(), c("gr.filt", "geno_fullVCF", "pathToFiles",
                            "fenotipo", "chr_sizes", "VCF", "chr")))
  cat("Limpieza del enviroment del cromosoma", chr, " realizada con éxito.\n")

}
```

Una vez generados estos datos, podemos generar tablas que nos permitan reunir la distribución de estas variantes en los 22 cromosomas y en los distintos pasos de filtrado realizados, como por ejemplo:

Cromosoma	Variantes Iniciales	Variantes tras Filtrados	Variantes Signif. Test Binomial
1	56625	4750	1246
2	49408	4059	1076
3	41908	3661	913
4	39161	2504	615
5	37647	3005	789
6	46030	4592	1408
7	34299	2711	1007
8	31764	2134	953
9	28181	2174	838
10	35469	2854	910
11	35715	3012	1343
12	32826	2844	1181
13	23244	1246	417
14	22280	1955	684
15	22124	2275	1339
16	25291	2219	839
17	23733	2755	1061
18	20211	1100	313
19	20332	2297	1242
20	16238	1403	500
21	9167	717	379
22	10658	1095	494
TOTAL	662311	55362	19547

Tabla 2. Distribución de variantes por cromosoma en las diferentes etapas del filtrado. Se incluye información de las variantes iniciales, las variantes filtradas en los objetos ASEset y aquellas que han superado el test binomial.

Para generar la tabla anterior, que combina la información de los 22 cromosomas en los distintos pasos del analisis, se ha utilizado el código siguiente:

```
# Cargar las librerías necesarias
library(dplyr)
library(biomaRt)

################################################################################
################################################################################

# TABLA 1: Número de variantes por cromosoma

################################################################################
### 1. Crear la tabla vacía donde se van a almacenar los resultados:

summary_table <- data.frame(
  Cromosoma = c("1_1", "1_2", "2_1", "2_2", 3:22),
  Variantes_totales = integer(24),
  Variantes_tras_filtrar = integer(24),
  Variantes_signif_binom_test = integer(24),
  RefFrac_mean = numeric(24)
)

################################################################################
### 2. Extraer la información relevante de los objetos correspondientes:

# Establecer el directorio principal donde están las carpetas de los cromosomas
main_dir <- "C:/Users/inesu/Desktop/TFM/RESULTADOS"

for (chr in c("1_1", "1_2", "2_1", "2_2", 3:22)) {
```

```r
  # Construir el nombre de la subcarpeta para el cromosoma actual
  subfolder <- file.path(main_dir, paste0("resultados_chr", chr))

  # Cambiar al directorio de la subcarpeta
  setwd(subfolder)

  # Determinar el número del cromosoma base (sin el sufijo de mitad)
  chr_num <- ifelse(chr %in% c("1_1", "1_2"), "1",
                    ifelse(chr %in% c("2_1", "2_2"), "2", as.character(chr)))

  # Generar los nombres de los archivos a cargar
  countList_file <- paste0("countList_chr", chr_num, ".RData")
  ase_chr_def_file <- paste0("ase_chr_def_", chr_num, ".RData")
  unique_binom_sig_var_file <- paste0("unique_binom_sig_var_chr", chr_num, ".RData")
  refFrac_mean_file <- paste0("refFrac_mean_chr", chr_num, ".RData")

  # Cargar los archivos necesarios
  load(countList_file)            # Carga countList
  load(ase_chr_def_file)          # Carga ase_chr_def
  load(unique_binom_sig_var_file) # Carga unique_binom_sig_var_chr
  load(refFrac_mean_file)         # Carga refFrac_mean_chr

  # Asignar valores a la tabla para el cromosoma actual
  summary_table$Variantes_totales[summary_table$Cromosoma == chr] <- length(countList_chr)
  summary_table$Variantes_tras_filtrar[summary_table$Cromosoma == chr] <- length(seqnames(rowR
  summary_table$Variantes_signif_binom_test[summary_table$Cromosoma == chr] <- length(unique_b
  summary_table$RefFrac_mean[summary_table$Cromosoma == chr] <- refFrac_mean_chr

  # Volver al directorio principal
  setwd(main_dir)
}
# Imprimir la tabla completa por mitades
print(summary_table)

##############################################################################
### 3. Generar una tabla con los resultados por cromosomas del 1 al 22:
# Crear una copia de la tabla y combinar las mitades para los cromosomas 1 y 2
summary_table_combined <- summary_table %>%
  mutate(Cromosoma = case_when(
    Cromosoma %in% c("1_1", "1_2") ~ "1",  # Combinar las filas de 1_1 y 1_2 en "1"
    Cromosoma %in% c("2_1", "2_2") ~ "2",  # Combinar las filas de 2_1 y 2_2 en "2"
    TRUE ~ as.character(Cromosoma)
  ))

# Agrupar por Cromosoma y calcular la suma y la media de las fracciones de alelo de referencia
summary_table_combined <- summary_table_combined %>%
  group_by(Cromosoma) %>%
  reframe(
    Variantes_totales = sum(Variantes_totales),
    Variantes_tras_filtrar = sum(Variantes_tras_filtrar),
    Variantes_signif_binom_test = sum(Variantes_signif_binom_test),
    RefFrac_mean = mean(RefFrac_mean),
  ) %>%
```

```
  arrange(as.numeric(Cromosoma))

# Imprimir la tabla combinada
print(summary_table_combined)


################################################################################
### 4. Guardar las tablas:

# Guardar la tabla completa como archivo CSV
write.csv(summary_table, "summary_table.csv", row.names = FALSE)

# Guardar la tabla combinada como archivo CSV
write.csv(summary_table_combined, "summary_table_def.csv", row.names = FALSE)
```

A partir de los objetos ASEset, que se guardan como archivos .RData, se puede acceder a toda la información que hemos cargado anteriormente (fenotipo, recuentos, identificador de las muestras, genotipo, etc.). Por ejemplo, podemos ver los recuentos iniciales, crudos para una variante concreta, y luego extraer fácilmente los recuentos filtrados donde algunos valores pasan a 0 para descartarse y no utilizarse en el análisis. Además, podríamos extraer la información solo de las personas para las que haya recuentos tras el filtrado. También podemos comprobar en el visualizador de IGV que estos recuentos se han obtenido bien, y los datos que extrae AllelicImbalance coinciden con los almacenados en los .bams de partida. A continuación se incluye un ejemplo de los distintos recuentos de una variante del cromosoma 6:

A. Recuentos iniciales (crudos).

```
> countList_chr[["chr6_32827728"]]
                                    A C   G T
HSPR_0042_nodup.keep.merge.sort.bam 0 0  67 0
PNTR_0001_nodup.keep.merge.sort.bam 0 0  34 0
PNTR_0002_nodup.keep.merge.sort.bam 0 0  27 0
PNTR_0003_nodup.keep.merge.sort.bam 0 0  76 0
PNTR_0006_nodup.keep.merge.sort.bam 0 0  84 0
PNTR_0008_nodup.keep.merge.sort.bam 0 0  32 0
PNTR_0011_nodup.keep.merge.sort.bam 0 0  29 0
PNTR_0023_nodup.keep.merge.sort.bam 0 0  52 0
PNTR_0030_nodup.keep.merge.sort.bam 1 0 130 0
PNTR_0054_nodup.keep.merge.sort.bam 0 0  20 0
PNTR_0238_nodup.keep.merge.sort.bam 0 0  30 0
UAMR_0001_nodup.keep.merge.sort.bam 0 0  41 0
UAMR_0002_nodup.keep.merge.sort.bam 0 0  81 0
UAMR_0013_nodup.keep.merge.sort.bam 0 0   0 0
UAMR_0038_nodup.keep.merge.sort.bam 1 0  99 0
UAMR_0039_nodup.keep.merge.sort.bam 0 0  90 0
UAMR_0042_nodup.keep.merge.sort.bam 1 0  68 0
UAMR_0045_nodup.keep.merge.sort.bam 0 0  88 0
UAMR_0048_nodup.keep.merge.sort.bam 0 0   0 0
UAMR_0051_nodup.keep.merge.sort.bam 0 0  35 0
```

B. Recuentos en el objeto ASEset (después de aplicar los filtrados).

```
> alleleCounts(ase_chr_def)[["chr6_32827728"]]
           A C  G T
HSPR_0042 0 0  0 0
PNTR_0001 0 0 34 0
PNTR_0002 0 0 27 0
PNTR_0003 0 0  0 0
PNTR_0006 0 0  0 0
PNTR_0008 0 0 32 0
PNTR_0011 0 0 29 0
PNTR_0023 0 0 52 0
PNTR_0030 0 0  0 0
PNTR_0054 0 0 20 0
PNTR_0238 0 0 30 0
UAMR_0001 0 0 41 0
UAMR_0002 0 0  0 0
UAMR_0013 0 0  0 0
UAMR_0038 0 0  0 0
UAMR_0039 0 0  0 0
UAMR_0042 0 0  0 0
UAMR_0045 0 0  0 0
UAMR_0048 0 0  0 0
UAMR_0051 0 0 35 0
```

C. Recuento de los individuos a analizar en los siguientes pasos.

```
> allele_counts_muestras
           A C  G T
PNTR_0001 0 0 34 0
PNTR_0002 0 0 27 0
PNTR_0008 0 0 32 0
PNTR_0011 0 0 29 0
PNTR_0023 0 0 52 0
PNTR_0054 0 0 20 0
PNTR_0238 0 0 30 0
UAMR_0001 0 0 41 0
UAMR_0051 0 0 35 0
```

D. Comprobación de los recuentos de 4 individuos en IGV (*Integrative genomics viewer*).

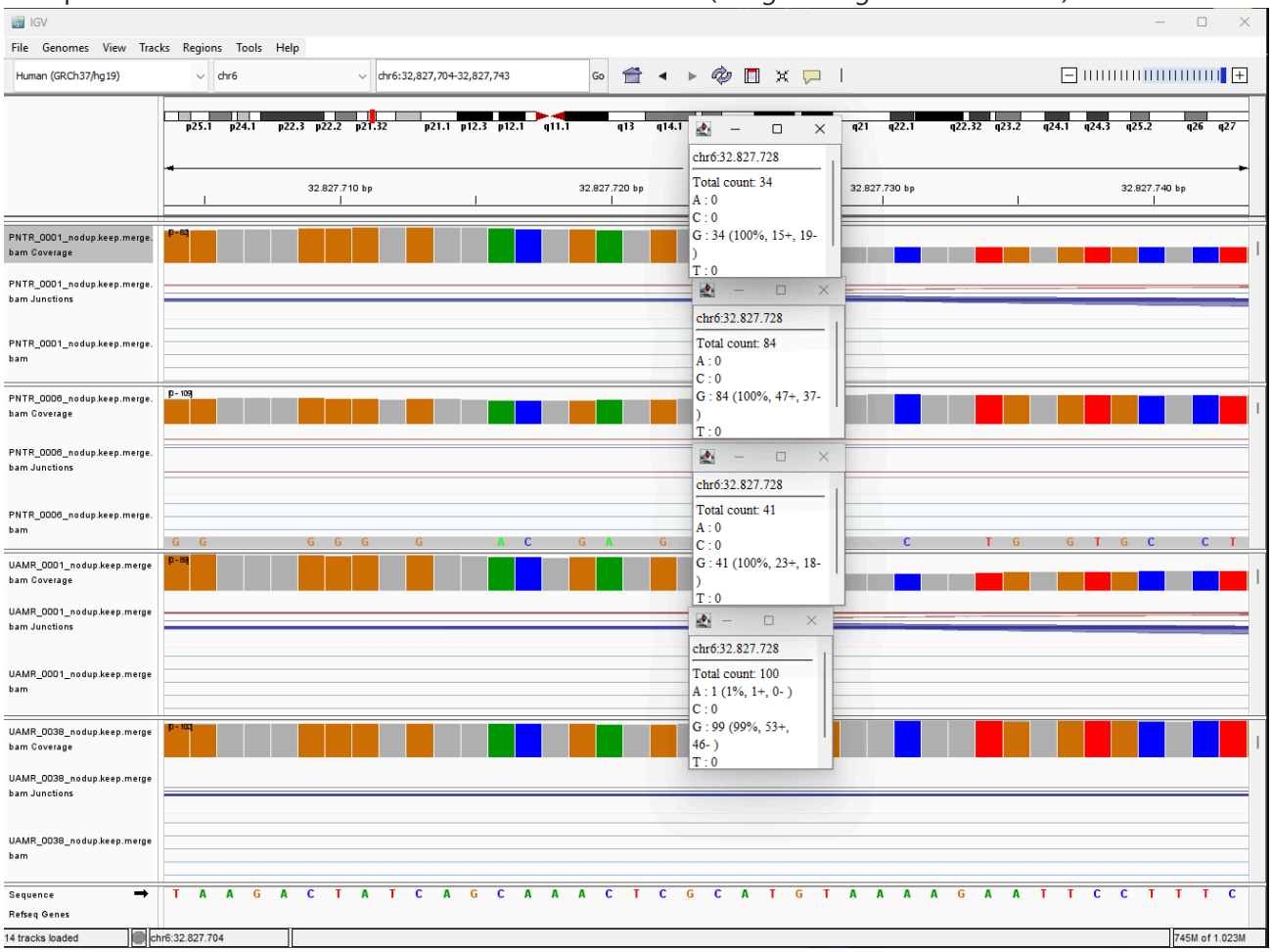


Figura 4. Recuentos alélicos en los distintos pasos. A. Recuentos crudos generados con la funcion *getAlleleCounts* de *AllelicImbalance* en R. B. Recuentos obtenidos tras los pasos de filtrado aplicados. C. Recuentos filtrados, conservando la información únicamente de las personas heterocigotas con recuentos superiores a 10 lecturas por muestra. D. Recuentos alélicos crudos obtenidos en IGV a partir de los archivos .bam procesados de cuatro personas aleatorias, dos sanas (UAMR_0001 y UAMR_0006) y dos enfermas (PNTR_0001 y PNTR_0038)

2. Análisis estadísticos

2.1. Prueba Wilcoxon Rank-Sum (R)

Una vez generados los recuentos y los objetos ASEset con toda la información necesaria de las variantes heterocigotas de interés y de los individuos de estudio, pasamos a aplicar la prueba de Wilcoxon Rank-Sum o U de Mann Whitney, para comparar la ASE entre individuos afectados y no afectados por TB. En este caso, utilizamos los paquetes *AllelicImbalance*, *VariantAnnotation*, *biomaRt* y *SNPlocs.Hsapiens.dbSNP144.GRCh38*. Los pasos principales de este código son:

- **Conexión con *Ensembl*:** e establece una conexión con *Ensembl* mediante la funcion *useMart* de *biomaRt* para más adelante en el código obtener información sobre los genes de interés.
- **Cálculo de proporciones de alelo de referencia para cada variante:** se verifica que los alelos sean las bases estándar (A, C, G, T), y se calcula la proporción del alelo de referencia como: Nº de lecturas con el alelo de referencia / Nº de lecturas totales, donde Nº de lecturas totales son Nº lecturas contienen el alelo de refernecia + Nº lecturas contienen el alelo alternativo.
- **Test de Wilcoxon:** se utiliza la función *wilcox.test* para comparar la proporción de alelos de referencia entre individuos afectados y no afectados.
- **Obtención de información adicional de las variantes:** si la prueba estadística enterior es significativa (p-valor < 0.05), se recupera información adicional de la variante desde *Ensembl* y *SNPlocs.Hsapiens.dbSNP144.GRCh38*.
- **Almacenamiento de los resultados:** los resultados del test se almacenan en una tabla que se guarda como archivo .txt. Se incluye información como el rsID de la variante, el Gen_Ensembl_ID, y el Gen_External_Name, entre otros.

```
################################################################################
# Cargar AllelicImbalance, VariantAnnotation y otros paquetes necesarios
library(AllelicImbalance)
library(SNPlocs.Hsapiens.dbSNP144.GRCh38)
library(biomaRt)
library(dplyr)

################################################################################
# Establecer la ruta donde vamos a guardar los resultados de este paso del TFM
result_dir <- getwd()
data_dir <- "C:/Users/inesu/Desktop/TFM/wilcoxon/data/"

# Conectar con Ensembl para obtener información de genes
ensembl_gen <- useMart("ENSEMBL_MART_ENSEMBL", dataset = "hsapiens_gene_ensembl",
                   host = "https://www.ensembl.org")

# Crear una tabla de fenotipos a partir de colData
fenotipos <- as.data.frame(colData(ase_chr_def))

# Crear una tabla para almacenar el conteo de variantes en cada cromosoma
tabla_conteo <- data.frame(Cromosoma = character(),
                           N_var_analizadas = integer(),
                           N_var_signif_wilcoxon = integer(),
                           stringsAsFactors = FALSE)
```

```r
# Bucle que recorre cada valor en el vector (son los cromosomas)
# En el caso del cromosoma 1 y 2, como eran muy grandes se han hecho por mitades
for (valor in c("2_1", "2_2", "1_2", "1_1", 3:22)) {

  # Notificación del inicio del proceso para cada cromosoma
  cat("Trabajando con el cromosoma:", valor, "\n")

  # Cargar el archivo .RData correspondiente
  load(file.path(data_dir, paste0("ase_chr_def_", valor, ".RData")))
  cat("Archivo ase_chr_def_", valor, " cargado correctamente.\n")

  # Cargar el archivo .RData correspondiente
  load(file.path(data_dir, paste0("signif_df_chr", valor, ".RData")))
  cat("Archivo signif_df_chr", valor, " cargado correctamente.\n")

  ################################################################################
  # Data frame para almacenar los resultados finales
  tabla_resultados <- data.frame(rsID = character(),
                                 Cromosoma = character(),
                                 Posicion = integer(),
                                 pvalor_wilcox = numeric(),
                                 pvalor_fdr = numeric(),
                                 Gen_Ensembl_ID = character(),
                                 Gen_External_Name = character(),
                                 Num_individuos = character(),
                                 stringsAsFactors = FALSE)

  # Reestablecer el contador de variantes analizadas en el test de Wilcoxon
  contador_variantes <- 0
  contador_variantes_signif <- 0

  # Iterar sobre las variantes significativas
  for (variant in unique(signif_df_chr$Variant)) {

    cat("Analizando variante:", variant, "\n")

    # Obtener el índice de la variante en ase_chr_def
    variant_idx <- match(variant, rownames(ase_chr_def))
    if (is.na(variant_idx)) {
      warning(paste("La variante", variant, "no se encontró en ase_chr_def."))
      next
    }

    # Extraer el cromosoma y la posición directamente del nombre de la variante
    split_variant <- strsplit(variant, "_")[[1]]
    cromosoma_var <- gsub("chr", "", split_variant[1])  # Parte antes del "_"
    posicion_var <- as.integer(split_variant[2])  # Parte después del "_"
    # Crear la región cromosómica en el formato correcto para BioMart
    region <- paste0(cromosoma_var, ":", posicion_var, "-", posicion_var)

    # Extraer los alelos de referencia y alternativos de rowData
    ref_allele <- rowData(ase_chr_def)$ref[variant_idx]
    alt_allele <- rowData(ase_chr_def)$alt[variant_idx]
```

```r
# Verifica que el alelo de referencia sea A, C, G o T
if (!(ref_allele %in% c("A", "C", "G", "T"))) {
  # Si no es A, C, G o T, pasa a la siguiente iteración o excluye la variante
  next # usa `next` si estás en un bucle o ajusta según tu código para omitir la variante
}

# Verifica que el alelo alternativo sea A, C, G o T
if (!(alt_allele %in% c("A", "C", "G", "T"))) {
  # Si no es A, C, G o T, pasa a la siguiente iteración o excluye la variante
  next # usa `next` si estás en un bucle o ajusta según tu código para omitir la variante
}

# Obtener los recuentos alélicos solo para las muestras relevantes
allele_counts <- alleleCounts(ase_chr_def)[[variant]]

# Sumar los recuentos por cada persona (fila)
suma_recuentos <- rowSums(allele_counts)

# Crear un vector con los identificadores de las personas que tienen al menos un recuento
muestras_para_variante <- rownames(allele_counts)[suma_recuentos > 0]

allele_counts_muestras <- allele_counts[muestras_para_variante, , drop = FALSE]

# Calcular proporciones de alelo de referencia, manejando errores
cat("Realizando cálculos de proporciones vara la variante:", variant, "\n")
total_counts <- allele_counts_muestras[, ref_allele] + allele_counts_muestras[, alt_allele
proporciones_ref <- ifelse(total_counts == 0, NA, allele_counts_muestras[, ref_allele] / t

# Si el alelo de referencia tiene recuentos 0, asignar proporción 0
proporciones_ref[allele_counts_muestras[, ref_allele] == 0] <- 0

# Extraer los fenotipos de las muestras relevantes a partir de la tabla de fenotipos
fenotipos_muestras <- fenotipos[muestras_para_variante, "Condition", drop = FALSE]

# Crear un dataframe con las proporciones de alelo de referencia y el fenotipo
datos_variante <- data.frame(Sample = rownames(fenotipos_muestras),
                             ProporcionRef = as.vector(proporciones_ref),
                             Condition = fenotipos_muestras$Condition)
print(datos_variante)

# Contar el número de muestras por condición
num_sanos <- sum(datos_variante$Condition == "Unaffected")
num_enfermos <- sum(datos_variante$Condition == "Affected")

# Solo realizar el test si hay al menos 2 muestras sanas y 2 enfermas
if (num_sanos >= 2 & num_enfermos >= 2) {
  cat("Realizando test de Wilcoxon para la variante:", variant, "\n")

  # Incrementar el contador de variantes
  contador_variantes <- contador_variantes + 1

  # Realizar el test de Wilcoxon Rank Sum o U de Mann Whitney
  wilcox_test <- wilcox.test(ProporcionRef ~ Condition, data = datos_variante)
```

```r
cat("Test de wilcoxon realizado con éxito.", "\n")

# Guardar siempre los resultados del test, no solo los significativos
tabla_resultados <- rbind(tabla_resultados, data.frame(
  rsID = NA, Cromosoma = cromosoma_var,
  Posicion = posicion_var,
  pvalor_wilcox = wilcox_test$p.value,
  pvalor_fdr = NA, Gen_Ensembl_ID = NA,
  Gen_External_Name = NA, Num_individuos = paste(num_sanos, num_enfermos, sep = " + ")
))

# Verificar si el test es significativo (p-valor < 0.05)
if (!is.na(wilcox_test$p.value) && wilcox_test$p.value < 0.05) {
  cat("WILCOXON TEST SIGNIFICATIVO para la variante:", variant, "\n")
  cat("Buscando información del SNP y el gen para la variante:", variant, "\n")

  contador_variantes_signif <- contador_variantes_signif + 1

  # Obtener información adicional sobre la variante desde Ensembl
  variant_info <- getBM(attributes = c("chromosome_name", "start_position",
                                       "ensembl_gene_id", "external_gene_name"),
                        filters = c("chromosome_name", "start", "end"),
                        values = list(cromosoma_var,posicion_var,posicion_var),
                        mart = ensembl_gen)

  # Genero un objeto gr para buscar los rsIDs con SNPlocs.Hsapiens.dbSNP144.GRCh38
  gr <- GRanges(seqnames = cromosoma_var, ranges = IRanges(start = posicion_var, end = p
  # Obtener los rsIDs de las variantes usando getSnpIdFromLocation
  gr_with_rs <- getSnpIdFromLocation(gr,SNPlocs.Hsapiens.dbSNP144.GRCh38)
  # Lo pasamos a un dataframe y el rs quedará como el rowname
  df_rs <- as.data.frame(gr_with_rs)

  # Si se encuentra información de la variante, añadir a la tabla de resultados
  if (nrow(variant_info) > 0) {

    cat("Añadiendo información del SNP y el gen para la variante:", variant, "\n")
    # Obtener los datos que necesitaremos a continuación
    rsID <- rownames(df_rs)
    cromosoma <- cromosoma_var
    posicion <- posicion_var
    gen_ensembl_id <- variant_info$ensembl_gene_id[1]
    gen_name <- variant_info$external_gene_name[1]

    # Formato del número de individuos como "sanos + enfermos"
    num_individuos <- paste(num_sanos, num_enfermos, sep = " + ")

    # Buscar la fila correspondiente en tabla_resultados usando la columna 'Posicion'
    fila_index <- which(tabla_resultados$Posicion == posicion_var & tabla_resultados$Cro

    # Comprobar si se ha encontrado la fila
    if (length(fila_index) == 1) {
      # Actualizar la fila encontrada con la nueva información
      tabla_resultados[fila_index, "rsID"] <- rsID
      tabla_resultados[fila_index, "Gen_Ensembl_ID"] <- gen_ensembl_id
```

```
          tabla_resultados[fila_index, "Gen_External_Name"] <- gen_name
          tabla_resultados[fila_index, "Num_individuos"] <- num_individuos
        } else {
          warning("No se ha encontrado la fila correspondiente en la tabla para la posición
        }
      }
    }
  } else {
    # Si no hay suficientes muestras, registrar la variante con solo el p-valor
    cat("No hay suficientes muestras para analizar la variante", variant,".\n")
    cat("Número de sanos =", num_sanos,". Número de enfermos=", num_enfermos,".")
  }
}

# Guardar el número de variantes analizadas para este cromosoma
tabla_conteo <- rbind(tabla_conteo,
                      data.frame(
                        Cromosoma = valor,
                        N_var_analizadas = contador_variantes,
                        N_var_signif_wilcoxon = contador_variantes_signif))
cat("Tabla de conteos actualizada con:", contador_variantes,"y",contador_variantes_signif, "

# Ajustar los p-valores mediante FDR
# (por el peq. tamaño muestral no se esperan resultados significativos)
tabla_resultados$pvalor_fdr <- p.adjust(tabla_resultados$pvalor_wilcox, method = "fdr")
archivo_restultados <- paste0("resultados_wilcox_chr", valor, ".txt")
write.table(tabla_resultados, file = archivo_restultados, sep = "\t", row.names = FALSE, quo
cat("Se han guardado los resultados obtenidos del cromosoma:", valor, "\n")

# Filtrar las variantes significativas
tabla_significativa <- tabla_resultados[!is.na(tabla_resultados$pvalor_wilcox) & tabla_resul
tabla_significativa_fdr <- tabla_resultados[!is.na(tabla_resultados$pvalor_fdr) & tabla_resu

# Guardar las tablas solo si contienen datos
if (nrow(tabla_significativa) > 0) {
  # Crear el nombre del archivo con la iteración actual (valor)
  archivo_significativa <- paste0("signif_var_wilcox_chr", valor, ".txt")
  write.table(tabla_significativa, file = archivo_significativa, sep = "\t", row.names = FAL
  cat("Se ha generado la tabla de variantes significativas para el cromosoma:", valor, "\n")
}

if (nrow(tabla_significativa_fdr) > 0) {
  # Crear el nombre del archivo con la iteración actual (valor)
  archivo_significativa_fdr <- paste0("signif_var_fdr_chr", valor, ".txt")
  write.table(tabla_significativa_fdr, file = archivo_significativa_fdr, sep = "\t", row.nam
  cat("Se ha generado la tabla de variantes SIGNIF-FDR para el cromosoma:", valor, "\n")
}

# Mostrar las tablas de resultados significativos
head(tabla_resultados)
cat("Tabla resultados con", dim(tabla_resultados)[1],"variantes analizadas y",
    dim(tabla_resultados)[2], "personas incluidas.", "\n")
print(tabla_significativa)
print(tabla_significativa_fdr)
```

```
  # Eliminamos todos los objetos que no sean necesarios
  rm(list = setdiff(ls(), c("fenotipos", "result_dir", "ensembl_gen", "valor", "data_dir", "ta
}

# Guardar la tabla de conteo en un archivo
write.table(tabla_conteo, file = "tabla_conteo_variantes.txt", sep = "\t", row.names = FALSE,
cat("Se ha generado la tabla de conteo de variantes analizadas.\n")
```

Una vez obtenidos todos los resultados del test de Wilcoxon Rank-Sum, podemos representar todas las variantes analizadas y su respectivo p-valor en un gráfico de tipo Manhattan Plot, generado con el paquete *ggplot2* en R. Este gráfico permite visualizar la distribución de estas variantes a lo largo de todo el genoma, así como su significación estadística (cuánto más arriba en el eje Y, más significativo). El código necesario para hacer este gráfico se incluye a continuación:

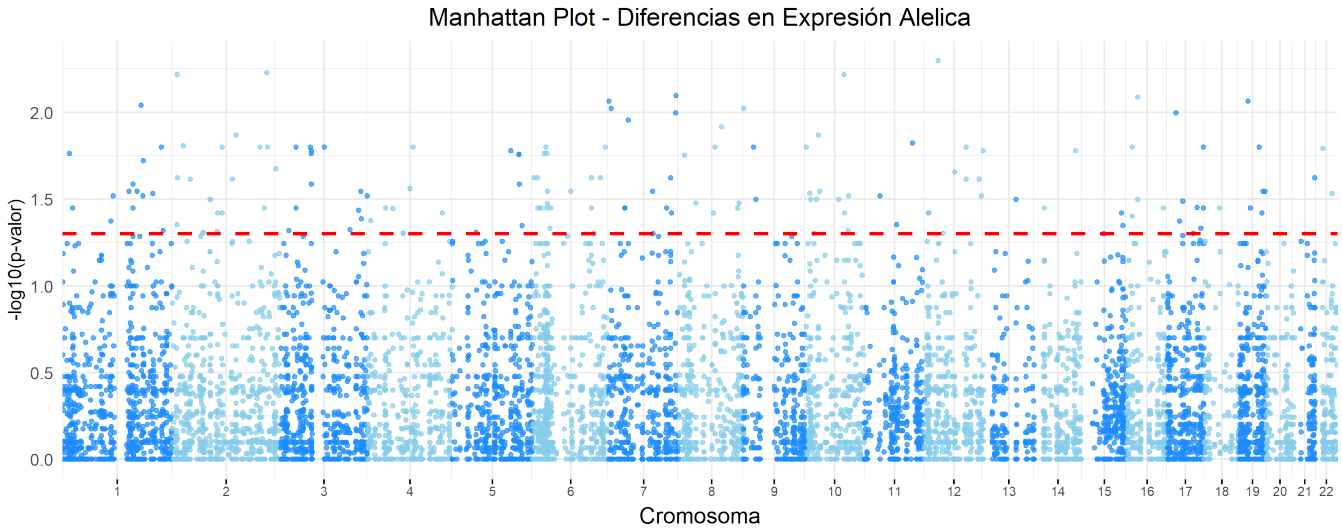


Figura 5. Distribución genómica de las variantes analizadas en el test de Wilcoxon Rank Sum. Se muestran los cromosomas en el eje X y los p-valores transformados en el eje Y. La línea de color rojo indica el umbral de significación estadística empleado (p-valor < 0.05).

El código necesario para generar el gráfico de tipo Manhattan anterior es:

```
# Cargar las librerías necesarias
library(ggplot2)
library(dplyr)
library(readr)

data_dir = "C:/irs_tfm/wilcoxon/Resultados_2_sanos_vs_enf/"

# Crear una lista de archivos a cargar
archivos <- list.files(path = data_dir, pattern = "resultados_wilcox_chr.*\\.txt", full.names

# Leer y combinar todos los archivos
datos <- archivos %>%
  lapply(read_delim, delim = "\t") %>%  # Leer cada archivo
  bind_rows(.id = "Archivo") %>%  # Combinar en un solo dataframe
  mutate(Cromosoma = factor(Cromosoma, levels = c(1, "1_1", "1_2", 2, "2_1", "2_2", 3:22))) #

# Agrupar los cromosomas 1_1 y 1_2, así como 2_1 y 2_2, en un solo cromosoma
datos <- datos %>%
  mutate(Cromosoma = case_when(
```

```
    Cromosoma %in% c("1_1", "1_2") ~ "1",
    Cromosoma %in% c("2_1", "2_2") ~ "2",
    TRUE ~ as.character(Cromosoma)
  ))

# Suponiendo que tus datos están almacenados en un objeto llamado 'datos'
datos_seleccionados <- datos %>%
  select(rsID, Cromosoma, Posicion, pvalor_wilcox)

# Convertir las columnas a tipo numérico
datos_seleccionados_numeric <- datos_seleccionados %>%
  mutate(
    Cromosoma = as.numeric(Cromosoma),
    Posicion = as.numeric(Posicion),
    pvalor_wilcox = as.numeric(pvalor_wilcox)
  )

# Datos de tamaño de los cromosomas
chr_sizes <- c(248956422, 242193529, 198295559, 190214555, 181538259,
               170805979, 159345973, 145138636, 138394717, 133797422,
               135086622, 133275309, 114364328, 107043718, 101991189,
               90338345, 83257441, 80373285, 58617616, 64444167,
               46709983, 50818468)

# Filtrar datos eliminando filas con NA y valores de pvalor_wilcox fuera del rango [0,1]
datos_seleccionados_numeric <- datos_seleccionados_numeric %>%
  filter(!is.na(pvalor_wilcox) & pvalor_wilcox >= 0 & pvalor_wilcox <= 1) %>%  # Filtrar NA y
  mutate(
    Cromosoma = as.factor(Cromosoma),  # Asegurarse de que Cromosoma sea un factor
    color_cromosoma = ifelse(as.numeric(Cromosoma) %% 2 == 0, "skyblue", "dodgerblue"),
    Cromosoma_numerico = as.numeric(Cromosoma)  # Crear columna numérica para ordenar cromosom
  ) %>%
  # Ajustar las posiciones según el tamaño de los cromosomas
  mutate(
    pos_cromosoma = Posicion + cumsum(c(0, chr_sizes[-length(chr_sizes)]))[Cromosoma_numerico]
  )

ggplot(datos_seleccionados_numeric, aes(x = pos_cromosoma, y = -log10(pvalor_wilcox), color =
  geom_point(alpha = 0.7, size = 1) +  # Usar puntos con transparencia
  scale_color_identity() +  # Usar colores definidos por el usuario
  scale_x_continuous(
    breaks = cumsum(chr_sizes) - chr_sizes / 2,  # Poner un corte en el centro de cada cromoso
    labels = as.character(1:22),  # Etiquetas para los cromosomas, solo el número
    expand = c(0, 0)  # Eliminar espacio extra antes del primer cromosoma
  ) +
  geom_hline(yintercept = 1.3, linetype = "dashed", color = "red", linewidth = 1.0) +  # Línea
  theme_minimal() +
  theme(
    axis.text.x = element_text(size = 7, angle = 0, hjust = 0.5, vjust = 2),  # Reducir tamaño
    axis.title.x = element_text(size = 12),
    axis.title.y = element_text(size = 10),
    legend.position = "none",  # Eliminar leyenda
    axis.ticks.x = element_line(size = 0.5)  # Añadir algo de grosor a las marcas del eje
  ) +
```

```
  labs(
    y = "-log10(p-valor)",
    x = "Cromosoma",  # Título del eje X
    title = "Manhattan Plot - Diferencias en Expresión Alelica"
  ) +
  theme(
    plot.title = element_text(hjust = 0.5)  # Centrar título
  )

# Guardar el gráfico en un archivo PNG de alta calidad
ggsave("manhattan_plot2.png",
       plot = last_plot(),
       width = 10,
       height = 3,
       dpi = 300)  # Alta resolución (300 dpi)

ggsave("manhattan_plot.pdf", plot = last_plot(), width = 10, height = 4)*
```

2.2 Prueba Exacta de Fisher (R)

Una vez realizado el test de Wilcoxon Rank-Sum, las variantes candidatas (con p-valor < 0.05 en el test de Wilcoxon Rank-Sum), se clasifican cualitativamente según el alelo que se expresa de forma mayoritaria en cada individuo (referencia o alternativo).

Para ello, se trabaja con cada cromosoma de interés (del 1 al 22), y el código siguiente permite cargar los datos y calcular las proporciones de alelo de referencia. Además, aplica el test exacto de Fisher para evaluar si la distribución de los alelos que se expresan de forma mayoritaria varía significativamente entre las dos condiciones de estudio (sanos *vs* enfermos). Si se detecta un cambio significativo (p-valor ≤ 0.05), se guarda una tabla con los recuentos de alelos y las estadísticas correspondientes (odds ratio, p-valor de Fisher). Los resultados se generan por cromosoma y se exportan a archivos de texto. Las funciones principales que se utilizan en este fragmento de código son:

- ***alleleCounts()*:** Extrae los recuentos de alelos para cada variante.
- ***fisher.test()*:** Realiza el test exacto de Fisher para comparar la distribución de alelos en las condiciones afectadas y no afectadas.
- ***left_join()*:** Fusiona los resultados de los recuentos de alelos con las tablas de variantes significativas.
- ***write.table()*:** Guarda los resultados en archivos de texto.

```
################################################################################
# Cargar AllelicImbalance, VariantAnnotation y otros paquetes necesarios
library(AllelicImbalance)
library(SNPlocs.Hsapiens.dbSNP144.GRCh38)
library(biomaRt)
library(dplyr)

################################################################################
# Establecer la ruta donde queremos guardar los resultados
result_dir <- getwd()
data_dir <- "C:/Users/inesu/Desktop/TFM/wilcoxon/data/"
```

```r
subcarpeta <- "C:/Users/inesu/Desktop/TFM/wilcoxon/recuentos_concretos/"

# Crear una tabla de fenotipos a partir de colData
fenotipos <- as.data.frame(colData(ase_chr_def))

# Bucle que recorre cada valor en el vector (los cromosomas estudiados)
for (valor in c(22, 21, 19:3, "2_1", "2_2", "1_2", "1_1")) {

  # Notificación del inicio del proceso para cada cromosoma
  cat("Trabajando con el cromosoma:", valor, "\n")

  # Cargar el archivo .RData correspondiente
  load(file.path(data_dir, paste0("ase_chr_def_", valor, ".RData")))
  cat("Archivo ase_chr_def_",valor, "cargado correctamente.\n")

  setwd(data_dir)
  # Cargar el archivo .txt correspondiente con las variantes de interés
  # Generar el nombre del archivo dinámicamente
  file_name <- paste0("signif_var_wilcox_chr", valor, ".txt")
  signif_data <- read.table(file_name, header = TRUE, sep = "\t")
  var_list <- list()
  var_153 <- var_list[[paste0("chr", valor)]] <- paste0("chr", signif_data$Cromosoma, "_", sig
  cat("Archivo .txt del cromosoma", valor, " cargado correctamente.\n")
  setwd(result_dir)

  ##############################################################################################
  # Crear un data frame vacío donde se guardarán los resultados
  proporciones_medias <- data.frame(Cromosoma = character(),
                                    Posicion = integer(),
                                    Affected = numeric(),
                                    Unaffected = numeric(),
                                    cambio_de_alelo = character(),
                                    stringsAsFactors = FALSE)
  ##############################################################################################

  # Iterar sobre las variantes significativas
  for (variant in var_153) {

    cat("Analizando variante:", variant, "\n")

    # Obtener el índice de la variante en ase_chr_def
    variant_idx <- match(variant, rownames(ase_chr_def))
    if (is.na(variant_idx)) {
      warning(paste("La variante", variant, "no se encontró en ase_chr_def."))
      next
    }

    # Extraer el cromosoma y la posición directamente del nombre de la variante
    split_variant <- strsplit(variant, "_")[[1]]
    cromosoma_var <- gsub("chr", "", split_variant[1])  # Parte antes del "_"
    posicion_var <- as.integer(split_variant[2])  # Parte después del "_"

    # Extraer los alelos de referencia y alternativos de rowData
    ref_allele <- rowData(ase_chr_def)$ref[variant_idx]
```

```r
alt_allele <- rowData(ase_chr_def)$alt[variant_idx]

# Verifica que el alelo de referencia sea A, C, G o T
if (!(ref_allele %in% c("A", "C", "G", "T"))) {
  # Si no es A, C, G o T, pasa a la siguiente iteración o excluye la variante
  next # usa `next` si estás en un bucle o ajusta según tu código para omitir la variante
}

# Verifica que el alelo alternativo sea A, C, G o T
if (!(alt_allele %in% c("A", "C", "G", "T"))) {
  # Si no es A, C, G o T, pasa a la siguiente iteración o excluye la variante
  next # usa `next` si estás en un bucle o ajusta según tu código para omitir la variante
}

# Obtener los recuentos alélicos
allele_counts <- alleleCounts(ase_chr_def)[[variant]]

# Sumar los recuentos por cada persona (fila)
suma_recuentos <- rowSums(allele_counts)

# Crear un vector con los identificadores de las personas que tienen al menos un recuento
muestras_para_variante <- rownames(allele_counts)[suma_recuentos > 0]

allele_counts_muestras <- allele_counts[muestras_para_variante, , drop = FALSE]

# Calcular proporciones de alelo de referencia, manejando errores
cat("Realizando cálculos de proporciones vara la variante:", variant, "\n")
total_counts <- allele_counts_muestras[, ref_allele] + allele_counts_muestras[, alt_allele
proporciones_ref <- ifelse(total_counts == 0, NA, allele_counts_muestras[, ref_allele] / t

# Si el alelo de referencia tiene recuentos 0, asignar proporción 0
proporciones_ref[allele_counts_muestras[, ref_allele] == 0] <- 0

# Extraer los fenotipos de las muestras relevantes a partir de la tabla de fenotipos
fenotipos_muestras <- fenotipos[muestras_para_variante, "Condition", drop = FALSE]

# Crear un dataframe con las proporciones de alelo de referencia y el fenotipo
datos_variante <- data.frame(Sample = rownames(fenotipos_muestras),
                             ProporcionRef = as.vector(proporciones_ref),
                             Condition = fenotipos_muestras$Condition)
print(datos_variante)

################################################################################

# Crear una nueva columna para clasificar el alelo mayoritario
datos_variante$AleloMayoritario <- ifelse(datos_variante$ProporcionRef > 0.5,
                                          "Referencia", "Alternativo")
print(datos_variante)

# Crear la tabla de contingencia
tabla_contingencia <- table(datos_variante$Condition, datos_variante$AleloMayoritario)
print(tabla_contingencia)

# Verificar las dimensiones de la tabla
```

```r
dim_tabla <- dim(tabla_contingencia)

# Si la tabla no tiene 2 filas, añadir una fila de ceros
if (dim_tabla[1] < 2) {
  tabla_contingencia <- rbind(tabla_contingencia, c(0, 0))  # Añadir una fila de ceros
}

# Si la tabla no tiene 2 columnas, añadir una columna de ceros
if (dim_tabla[2] < 2) {
  tabla_contingencia <- cbind(tabla_contingencia, c(0, 0))  # Añadir una columna de ceros
}

write.table(
  tabla_contingencia,
  file = paste0("tabla_conting_", variant, ".txt"),
  sep = "\t",
  row.names = TRUE,
  col.names = NA,
  quote = FALSE
)

# Realizar el test exacto de fisher (es un tipo de test de asociación)
# Utilizamos Fisher porque es como Chi cuadrado, pero para nº datos reducido
fisher_result <- fisher.test(tabla_contingencia)
print(fisher_result)

# Extraer p-valor y odds ratio del test de Fisher
p_valor_fisher <- fisher_result$p.value
odds_ratio <- ifelse(is.null(fisher_result$estimate), NA, fisher_result$estimate)
##############################################################################

medias_por_grupo <- datos_variante %>%
  group_by(Condition) %>%
  summarize(media = mean(ProporcionRef, na.rm = TRUE))

media_df = as.data.frame(t(medias_por_grupo$media))

# Asignar nombres a las columnas para facilitar su comprensión
colnames(media_df) <- c("Affected", "Unaffected")

# Añadir la columna cromosoma, posición, p-val y odd ratio de la variante al resultado
media_df$Cromosoma <- cromosoma_var
media_df$Posicion <- posicion_var
media_df$Pvalor_Fisher <- p_valor_fisher
media_df$Odds_Ratio <- odds_ratio

# Verificar el cambio de la expresión predominante del alelo (SI O NO)
media_df$cambio_de_alelo <- ifelse(media_df$Pvalor_Fisher <= 0.05, "SI", "NO")


# Guardar allele_counts_muestras en un archivo .txt solo si cambio_de_alelo es "SI"
if (media_df$cambio_de_alelo == "SI") {

  setwd(subcarpeta)
```

```
      # Guardar el objeto allele_counts_muestras en un archivo .txt
      write.table(
        allele_counts_muestras,
        file = paste0("subset_counts_", variant, ".txt"),
        sep = "\t",
        row.names = TRUE,
        col.names = NA,
        quote = FALSE
      )
      setwd(result_dir)
    }

    # Reorganizar las columnas para que la tabla tenga la estructura correcta
    media_df <- media_df[, c("Cromosoma", "Posicion", "Affected", "Unaffected",
                             "Pvalor_Fisher", "cambio_de_alelo", "Odds_Ratio")]

    # Añadir el resultado de la variante a la tabla final
    proporciones_medias <- bind_rows(proporciones_medias, media_df)

  }

  # Guardar las tablas solo si contienen datos
  if (nrow(proporciones_medias) > 0) {
    # Crear el nombre del archivo con la iteración actual (valor)
    file_name <- paste0("data_solo_fisher_", valor, ".txt")
    write.table(proporciones_medias, file = file_name, sep = "\t", row.names = FALSE, quote =
    cat("Se ha generado la tabla de medias para el cromosoma:", valor, "\n")
  }

  head(proporciones_medias)

  # Realizar la unión de signif_data con proporciones_medias
  signif_data$Cromosoma <- as.character(signif_data$Cromosoma)
  proporciones_medias$Cromosoma <- as.character(proporciones_medias$Cromosoma)
  # Añadir columnas solo cuando las columnas "Cromosoma" y "Posicion" coinciden
  signif_data <- signif_data %>%
    left_join(proporciones_medias %>% select(Cromosoma, Posicion, Affected,
                                             Unaffected, Pvalor_Fisher,
                                             cambio_de_alelo, Odds_Ratio),
              by = c("Cromosoma", "Posicion"))

  tabla_final <- paste0("fisher_full_", valor, ".txt")
  write.table(signif_data, file = tabla_final, sep = "\t", row.names = FALSE, quote = FALSE)
  cat("Se ha generado la tabla de final para el cromosoma:", valor, "\n")

  # Eliminamos todos los objetos que no sean necesarios
  rm(list = setdiff(ls(), c("fenotipos", "result_dir", "data_dir",
                            "subcarpeta", "ase_chr_def", "valor", "proporciones_medias")))

}
```

Después, se pueden combinar los archivos resultantes para guardarlos en formatos específicos como .xlxs o .txt, que permitan reunir todos los datos en un *data frame* para inspeccionar los resultados o para realizar los pasos posteriores del TFM.

```
library(dplyr)
library(writexl)

# Definir el directorio donde están los archivos .txt de partida
dir_path <- "C:/Users/inesu/Desktop/TFM/wilcoxon/Resultados/full_data"

# Obtener una lista de todos los archivos que coinciden con el patrón
files <- list.files(path = dir_path, pattern = "^medias_wilcox_.*\\.txt$", full.names = TRUE)

# Leer el primer archivo con encabezado
first_data <- read.table(files[1], header = TRUE, sep = "\t")

# Leer los archivos restantes sin encabezado, asignando los nombres de columna del primer arch
other_data <- lapply(files[-1], function(file) {
  data <- read.table(file, header = FALSE, sep = "\t")
  colnames(data) <- colnames(first_data)
  return(data)
})

# Combinar el primer archivo con los demás archivos en un solo data frame
will_wilcoxon <- do.call(rbind, c(list(first_data), other_data))

# Eliminar las filas adicionales del encabezado, si existen
will_wilcoxon <- will_wilcoxon[will_wilcoxon$rsID != "rsID", ]


# Guardar el resultado en un archivo Excel y en otro RData
write_xlsx(will_wilcoxon, "will_wilcoxon.xlsx")
save(will_wilcoxon, file = "will_wilcoxon.RData")
```

3. Anotación y enriquecimiento funcional

3.1. Anotación funcional (VEP)

Una vez realizado el análisis de ASE, se puede pasar a interpretar los resultados. En este caso se ha utilizado la herramienta web de *VEP* (*Variant Effect Predictor*) de *Ensembl* para extraer información sobre la anotación de las variantes canididatas, en concreto, se ha obtenido información de los valores CADD (*Combined Annotation Dependent Depletion*) y de las frecuencias alélicas de la base de datos *genomÁD genomes* correspondientes a la población europea no finlandesa. El enlace para aceder a esta herramienta es el siguiente: VEP-Ensembl.

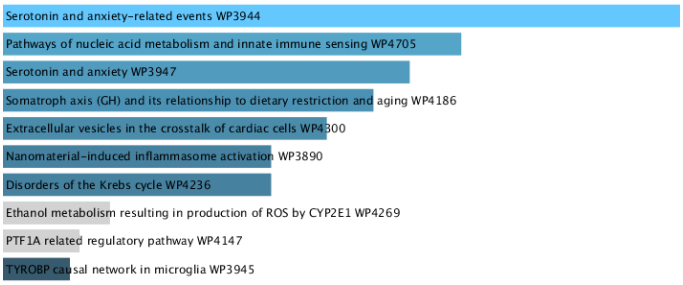
3.2. Enriquecimiento funcional (Enrichr)

También se ha realizado un análisis de enriquecimiento funcional de las variantes candidatas obtenidas, mediante la herramienta Enrichr, que está disponible en el siguiente enlace: Enrichr.

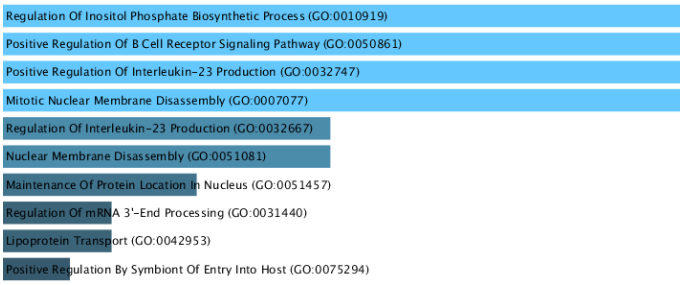
Esta herramienta nos permite obtener información de múltiples bases de datos. En nuestro caso, nos hemos centrado en los resultados de tres bases de datos complementarias: *WikiPathways*, *Gene Ontology (GO) Biological Process* y *Reactome*. Como *output*, esta herramienta nos puede dar las tablas

incluidas en el **Anexo 4** del TFM, y también algunos gráficos como los que se incluyen a continuación, donde se indican los procesos con mayor enriquecimiento:

A. *WikiPathways*



B. *GO Biological Process*



C. *Reactome*

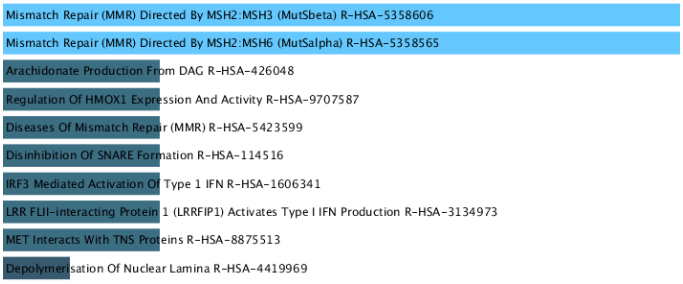


Figura 6. Gráficos de barras obtenidos en el análisis de enriquecimiento funcional realizado con Enrichr. Se incluye información de tres bases de datos: A) *WikiPathways*, B) *GO Biological Process*; y C) *Reactome*.

3.3. Validación con resultados de un análisis *gene-based* previo

Finalmente, los resultados obtenidos se pueden comparar con los resultados de otros estudios, como es el caso de un análisis de tipo *gene-based* realizado previamente por el grupo de investigación a partir de los datos de la mayor cohorte de trastorno bipolar que hay hasta la fecha, generada por el consorcio PGC (*Psychiatric Genomics Consortium*).

Una vez comparados estos resultados, podemos representar las variantes de los genes que hayan obtenido p-valores significativos en ambos análisis, para así, visualizar la distribución de estas variantes y su significación estadística. Los p-valores significativos del análisis *gene-based* indican que dichos genes están fuertemente asociación con la enfermedad, lo que sugiere que las variantes dentro de esos genes podrían tener un papel relevante en la patogénesis del trastorno bipolar. Este tipo de gráficos se puede conseguir con la herramienta LocusZoom, disponible en: LocusZoom.

A continuación se incluye un ejemplo de un gráfico generado con la herramienta LocusZoom, manteniendo los parámetros por defecto salvo el *Flankg Size* que se ha establecido en 5 Kb.

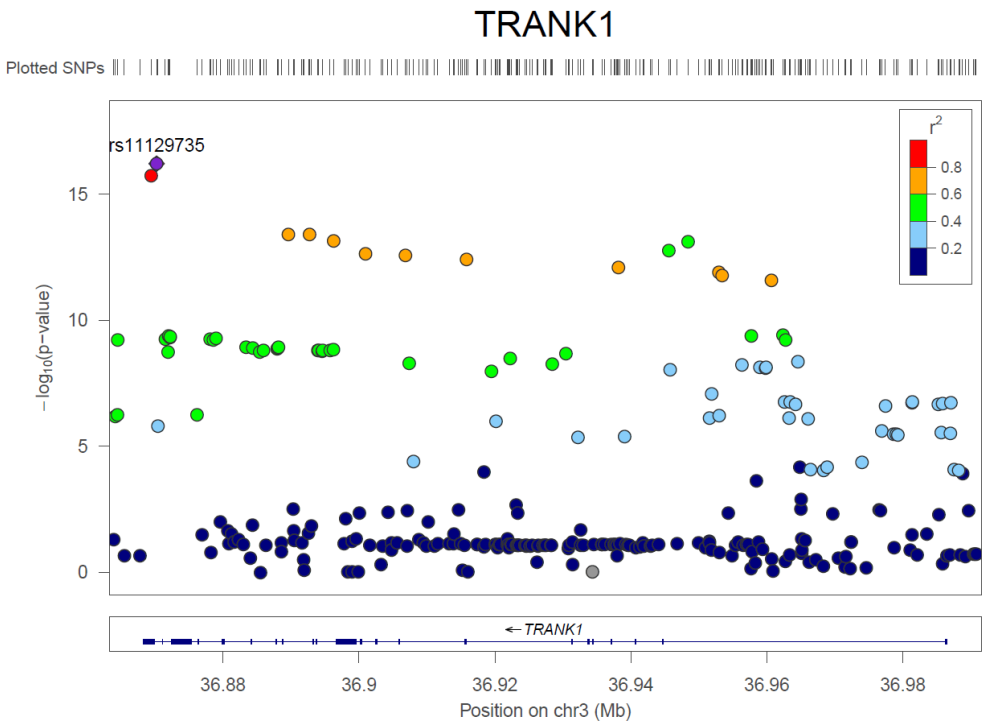


Figura 7. Gráfico generado con la herramienta *LocusZoom* donde se representan las variantes localizadas en el gen *TRANK1*. Se trata del gen con mayor significación estadística en el análisis *gene-based*, que ha obtenido también un p-valor significativo en el análisis de ASE realizado en este TFM.

Referencias

Chen, E. Y., Tan, C. M., Kou, Y., Duan, Q., Wang, Z., Meirelles, G. V., Clark, N. R., & Ma'ayan, A. (2013). Enrichr: interactive and collaborative HTML5 gene list enrichment analysis tool. BMC bioinformatics, 14, 128. https://doi.org/10.1186/1471-2105-14-128

de Leeuw, C. A., Mooij, J. M., Heskes, T., & Posthuma, D. (2015). MAGMA: generalized gene-set analysis of GWAS data. PLoS computational biology, 11(4), e1004219. https://doi.org/10.1371/journal.pcbi.1004219

Deng, Y., He, T., Fang, R., Li, S., Cao, H., & Cui, Y. (2020). Genome-Wide Gene-Based Multi-Trait Analysis. Frontiers in genetics, 11, 437. https://doi.org/10.3389/fgene.2020.00437

Dexter F. (2013). Wilcoxon-Mann-Whitney test used for data that are not normally distributed. Anesthesia and analgesia, 117(3), 537–538. https://doi.org/10.1213/ANE.0b013e31829ed28f

Gådin, J. R., van't Hooft, F. M., Eriksson, P., & Folkersen, L. (2015). AllelicImbalance: an R/bioconductor package for detecting, managing, and visualizing allele expression imbalance data from RNA sequencing. BMC bioinformatics, 16(1), 194. https://doi.org/10.1186/s12859-015-0620-2

Gene Ontology Consortium (2015). Gene Ontology Consortium: going forward. Nucleic acids research, 43(Database issue), D1049–D1056. https://doi.org/10.1093/nar/gku1179

Hunt, S. E., Moore, B., Amode, R. M., Armean, I. M., Lemos, D., Mushtaq, A., Parton, A., et al. (2022). Annotating and prioritizing genomic variants using the Ensembl Variant Effect Predictor-A tutorial. Human mutation, 43(8), 986–997. https://doi.org/10.1002/humu.24298

Kim, D., Langmead, B., & Salzberg, S. L. (2015). HISAT: a fast spliced aligner with low memory requirements. Nature methods, 12(4), 357–360. https://doi.org/10.1038/nmeth.3317

Lee, J., Lee, S., Jang, J. Y., & Park, T. (2018). Exact association test for small size sequencing data. BMC medical genomics, 11(Suppl 2), 30. https://doi.org/10.1186/s12920-018-0344-z

Leggett, R. M., Ramirez-Gonzalez, R. H., Clavijo, B. J., Waite, D., & Davey, R. P. (2013). Sequencing quality assessment tools to enable data-driven informatics for high throughput genomics. Frontiers in genetics, 4, 288. https://doi.org/10.3389/fgene.2013.00288

Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., Durbin, R., & 1000 Genome Project Data Processing Subgroup (2009). The Sequence Alignment/Map format and SAMtools. Bioinformatics (Oxford, England), 25(16), 2078–2079. https://doi.org/10.1093/bioinformatics/btp352

Martens, M., Ammar, A., Riutta, A., Waagmeester, A., Slenter, D. N., Hanspers, K., A Miller, R., Digles, D., Lopes, E. N., Ehrhart, F., Dupuis, L. J., Winckers, L. A., Coort, S. L., Willighagen, E. L., Evelo, C. T., Pico, A. R., & Kutmon, M. (2021). WikiPathways: connecting communities. Nucleic acids research, 49(D1), D613–D621. https://doi.org/10.1093/nar/gkaa1024

Milacic, M., Beavers, D., Conley, P., Gong, C., Gillespie, M., Griss, J., Haw, R., Jassal, B., Matthews, L., May, B., Petryszak, R., Ragueneau, E., Rothfels, K., Sevilla, C., Shamovsky, V., Stephan, R., Tiwari, K., Varusai, T., Weiser, J., Wright, A., et al. (2024). The Reactome Pathway Knowledgebase 2024. Nucleic acids research, 52(D1), D672–D678. https://doi.org/10.1093/nar/gkad1025

Mullins, N., Forstner, A. J., O'Connell, K. S., Coombes, B., Coleman, J. R. I., Qiao, Z., et al. (2021). Genome-wide association study of more than 40,000 bipolar disorder cases provides new insights into the underlying biology. Nature genetics, 53(6), 817–829. https://doi.org/10.1038/s41588-021-00857-4

Pruim, R. J., Welch, R. P., Sanna, S., Teslovich, T. M., Chines, P. S., Gliedt, T. P., Boehnke, M., Abecasis, G. R., & Willer, C. J. (2010). LocusZoom: regional visualization of genome-wide association scan results. Bioinformatics (Oxford, England), 26(18), 2336–2337. https://doi.org/10.1093/bioinformatics/btq419

Purcell, S., Neale, B., Todd-Brown, K., Thomas, L., Ferreira, M. A., Bender, D., et al. (2007). PLINK: a tool set for whole-genome association and population-based linkage analyses. American journal of human genetics, 81(3), 559–575. https://doi.org/10.1086/519795

Rentzsch, P., Witten, D., Cooper, G. M., Shendure, J., & Kircher, M. (2019). CADD: predicting the deleteriousness of variants throughout the human genome. Nucleic acids research, 47(D1), D886–D894. https://doi.org/10.1093/nar/gky1016

Robinson, J. T., Thorvaldsdóttir, H., Winckler, W., Guttman, M., Lander, E. S., Getz, G., & Mesirov, J. P. (2011). Integrative genomics viewer (IGV). Nature biotechnology, 29(1), 24–26. https://doi.org/10.1038/nbt.1754

Slifer S. H. (2018). PLINK: Key Functions for Data Analysis. Current protocols in human genetics, 97(1), e59. https://doi.org/10.1002/cphg.59

van de Geijn, B., McVicker, G., Gilad, Y., & Pritchard, J. K. (2015). WASP: allele-specific software for robust molecular quantitative trait locus discovery. Nature methods, 12(11), 1061–1063. https://doi.org/10.1038/nmeth.3582