Documentando Codigo Claudia

Carlos E Martinez R

2023-10-18

Explicación detallada del siguiente codigo:

**ids = c(“cdc5\_1\_S9”,“cdc5\_2\_S10”,“CmasM\_1\_S15”,“CmasM\_2\_S16”,“EhMyb10\_1\_S11”,“EhMyb10\_2\_S12”,“pEhEx\_2\_S8”, “pEhEx1\_S7”)**

**type = c(“EhCDC5-like”,“EhCDC5-like”,“EhCDC5-like+EhMyb10”,“EhCDC5-like+EhMyb10”,“EhMyb10”,“EhMyb10”,“Control”,“Control”)**

**results = “~/ballgown\_r2”**

**path = paste(results,ids,sep=“/”)**

**pheno\_data = data.frame(ids,type,path)**

**pheno\_data**

Este código en R se utiliza para crear un DataFrame llamado pheno\_data que almacena información sobre muestras experimentales en un estudio. Vamos a analizar el código paso a paso:

Se define un vector ids que contiene los identificadores de las muestras. Cada identificador corresponde a una muestra en el estudio. Los identificadores son cadenas de texto y representan las diferentes muestras en el estudio.

**ids = c(“cdc5\_1\_S9”, “cdc5\_2\_S10”, “CmasM\_1\_S15”, “CmasM\_2\_S16”, “EhMyb10\_1\_S11”, “EhMyb10\_2\_S12”, “pEhEx\_2\_S8”, “pEhEx1\_S7”)**

Se define un vector type que especifica el tipo de cada muestra. Cada elemento del vector type corresponde a una muestra en el mismo orden que los identificadores en el vector ids. Los tipos son también cadenas de texto y representan las categorías a las que pertenecen las muestras.

**type = c(“EhCDC5-like”, “EhCDC5-like”, “EhCDC5-like+EhMyb10”, “EhCDC5-like+EhMyb10”, “EhMyb10”, “EhMyb10”, “Control”, “Control”)**

Se define una variable results que almacena la ruta a un directorio o carpeta llamado “ballgown\_r2”. Esta variable se utilizará para construir las rutas completas a los archivos relacionados con las muestras.

**results = “~/ballgown\_r2”**

Se utiliza la función paste para combinar el contenido de las variables results y ids, separados por “/”, y almacenar los resultados en el vector path. Esto crea rutas completas para cada muestra.

**path = paste(results, ids, sep = “/”)**

Finalmente, se crea un DataFrame llamado pheno\_data que combina las columnas ids, type, y path. Esto se hace utilizando la función data.frame, que crea un DataFrame con estas tres columnas. Cada fila del DataFrame corresponde a una muestra, y las columnas almacenan información sobre el identificador, el tipo y la ruta de cada muestra.

pheno\_data = data.frame(ids, type, path) En resumen, este código se utiliza para organizar información sobre las muestras de un estudio en un DataFrame llamado pheno\_data. Cada muestra está asociada con un identificador, un tipo y una ruta de acceso a los archivos relacionados con la muestra.

**bg2 = ballgown(dataDir = “~/ballgown\_r2/”, samplePattern = “\_S”, pData=pheno\_data)**

El código que proporcionaste se utiliza para crear un objeto ballgown en R, que es una estructura de datos utilizada para analizar datos de expresión génica de alto rendimiento, como los datos de RNA-seq. A continuación, explicaré los diferentes argumentos y lo que hace este código:

a. bg2: Se está creando un nuevo objeto llamado bg2, que contendrá la estructura de datos ballgown para analizar los datos de expresión génica.  
  
b. ballgown(dataDir = "~/ballgown\_r2/", samplePattern = "\_S", pData = pheno\_data): Esto es una llamada a la función ballgown con varios argumentos:  
  
c. dataDir: Es el directorio donde se encuentran los archivos relacionados con los datos de RNA-seq. En este caso, el directorio se especifica como "~/ballgown\_r2/", que es el directorio que mencionaste anteriormente. Esto es donde se esperan encontrar los datos de expresión génica para las muestras.  
  
d. samplePattern: Es un patrón que se utilizará para identificar los archivos relacionados con las muestras en el directorio dataDir. En este caso, se utiliza "\_S" como patrón. Esto significa que se buscarán archivos cuyos nombres contengan "\_S". Esto se utiliza para asociar los archivos de datos con las muestras en el DataFrame pheno\_data.  
  
e. pData: Aquí se especifica el DataFrame pheno\_data que creaste anteriormente. Este DataFrame contiene información sobre las muestras, incluidos los identificadores, tipos y rutas de acceso. El ballgown utilizará esta información para asociar los archivos de datos con las muestras y sus características.

En resumen, este código crea un objeto ballgown llamado bg2 que se utiliza para analizar datos de expresión génica almacenados en el directorio “~/ballgown\_r2/”. Se asocian los archivos de datos con las muestras utilizando el patrón “\_S”, y se utiliza el DataFrame pheno\_data para mantener información sobre las muestras. El objeto bg2 será utilizado para llevar a cabo análisis posteriores de los datos de expresión génica.

**bg\_table2 = texpr(bg2, meas=“all”)**

El código que proporcionaste se utiliza para crear una tabla de expresión de genes a partir de un objeto ballgown llamado bg2. La función texpr se utiliza para extraer los datos de expresión de genes de bg2. A continuación, desglosaré el código:

a. bg\_table2: Se está creando un nuevo objeto llamado bg\_table2 que almacenará la tabla de expresión de genes. Este objeto contendrá los datos de expresión de genes extraídos de bg2.  
  
b. texpr(bg2, meas = "all"): Esto es una llamada a la función texpr. Los argumentos son los siguientes:  
  
c. bg2: Es el objeto ballgown del cual se extraerán los datos de expresión de genes.  
  
d. meas = "all": El argumento meas especifica qué tipo de medidas de expresión se deben extraer. En este caso, "all" se utiliza para extraer todas las medidas de expresión disponibles, lo que incluye medidas como recuento de fragmentos, recuento de transcripción, FPKM (Fragmentos Por Kilobase de millón) y otras medidas comunes utilizadas en análisis de expresión génica.

El resultado de esta operación es una tabla de expresión de genes que contiene información sobre la expresión de genes para cada muestra en el objeto ballgown. Esta tabla se almacena en el objeto bg\_table2 y se puede utilizar para realizar análisis posteriores, como identificar genes diferencialmente expresados o visualizar patrones de expresión génica en las muestras del estudio.

**bg\_table2 %>% head()**

El código que proporcionaste utiliza el operador %>% de la biblioteca dplyr (o de otra biblioteca que admita este operador) para aplicar la función head() a la tabla de expresión de genes bg\_table2. Aquí está lo que hace el código:

a. bg\_table2: Este es el objeto que contiene la tabla de expresión de genes que obtuviste en el paso anterior.  
  
b. %>%: Este es el operador de tubería (pipe) utilizado en R, que permite encadenar funciones o manipulaciones de datos en un flujo secuencial. Básicamente, toma el resultado de la operación a la izquierda del operador y lo pasa como entrada a la operación a la derecha del operador.  
  
c. head(): La función head() se utiliza comúnmente para mostrar las primeras filas de un DataFrame o una tabla de datos. En este caso, se está utilizando para mostrar las primeras filas de la tabla de expresión de genes bg\_table2.

Entonces, el código toma bg\_table2, que es una tabla de expresión de genes, y luego muestra las primeras filas de esa tabla, lo que proporciona una vista previa de los primeros genes y sus valores de expresión para las muestras del estudio. Esta operación es útil para inspeccionar los datos y entender cómo se ven antes de realizar análisis o visualizaciones más avanzados.

**pData(bg2)**

El código pData(bg2) se utiliza para extraer la información de los datos fenotípicos asociados con un objeto ballgown llamado bg2. Aquí se explica lo que hace este código:

a. pData(bg2): Esta expresión llama a la función pData() en el objeto ballgown bg2. La función pData() se utiliza para acceder a la información fenotípica o de metadatos asociada con las muestras en el objeto ballgown. En el contexto de análisis de expresión génica, los datos fenotípicos pueden incluir información sobre las condiciones experimentales, grupos de tratamiento, identificadores de muestra, tipos de muestra y cualquier otro metadato relevante.

La salida de pData(bg2) será un DataFrame que contiene la información fenotípica de las muestras asociadas con el objeto ballgown. Cada fila del DataFrame corresponderá a una muestra y las columnas representarán diferentes atributos o metadatos de esas muestras. Esta información fenotípica es útil para realizar análisis estadísticos y visualizaciones que relacionen los datos de expresión génica con las características de las muestras en el estudio.

**bg\_table2 %>% select(c(t\_name, num\_exons, length), starts\_with(“FPKM”)) %>% mutate\_at(vars(starts\_with(“FPKM”)), ~ log2(.x+1)) %>% pivot\_longer(cols=starts\_with(“FPKM”), values\_to = “FPKM”, names\_to = “Sample”) %>% mutate(Sample=str\_replace(Sample, “FPKM.”, ““)) %>% ggplot(aes(x=Sample, y=FPKM, fill=Sample)) + geom\_boxplot() + theme(axis.text.x = element\_text(angle = 45, vjust = 0.5, hjust=1)) -> p ggplotly(p)**

El código que proporcionaste realiza una serie de operaciones para visualizar datos de expresión génica utilizando el paquete ggplot2 y la función ggplotly del paquete plotly para obtener una gráfica interactiva. A continuación, se detallan las operaciones paso a paso:

a. bg\_table2 %>% ...: El código comienza con el objeto bg\_table2, que contiene datos de expresión génica. Luego, utiliza el operador %>% (pipe) para encadenar una serie de operaciones juntas. Esto permite realizar transformaciones sucesivas en los datos.  
  
b. select(c(t\_name, num\_exons, length), starts\_with("FPKM")): La función select se utiliza para seleccionar columnas específicas del DataFrame bg\_table2. En este caso, se seleccionan las columnas t\_name, num\_exons y length, además de todas las columnas que comienzan con "FPKM". Estas últimas contienen valores de expresión génica FPKM.  
  
c. mutate\_at(vars(starts\_with("FPKM")), ~ log2(.x+1)): La función mutate\_at se usa para aplicar una transformación a las columnas seleccionadas previamente (las que comienzan con "FPKM"). En este caso, se toma el logaritmo en base 2 (log2) de los valores FPKM, con la adición de 1 para evitar problemas cuando los valores son cero. Esto es común en análisis de expresión génica para estabilizar las varianzas.  
  
d. pivot\_longer(cols = starts\_with("FPKM"), values\_to = "FPKM", names\_to = "Sample"): La función pivot\_longer se utiliza para reorganizar los datos de manera que las columnas de expresión FPKM se transformen en una columna "FPKM" y una columna "Sample". Esto facilita la creación de una gráfica de cajas.  
  
e. mutate(Sample = str\_replace(Sample, "FPKM.", "")): La función mutate se utiliza para eliminar el prefijo "FPKM." de los nombres de las muestras en la columna "Sample". Esto ayuda a limpiar los nombres de las muestras para la visualización.  
  
f. ggplot(aes(x = Sample, y = FPKM, fill = Sample)) + ...: Se crea un objeto ggplot que se utiliza para generar una gráfica de cajas. Se especifica que las muestras se muestren en el eje x, los valores FPKM en el eje y y que el relleno de las cajas sea según la muestra. Luego, se agrega una capa de cajas utilizando geom\_boxplot(). Se ajusta el formato de las etiquetas en el eje x utilizando theme().  
  
g. -> p: El resultado del objeto ggplot se asigna a la variable p.  
  
h. ggplotly(p): Finalmente, se utiliza la función ggplotly del paquete plotly para convertir la gráfica generada con ggplot en una gráfica interactiva que se puede explorar y personalizar en un entorno interactivo.

En resumen, este código realiza una serie de transformaciones en los datos de expresión génica y crea una gráfica de cajas interactiva para visualizar la distribución de los valores FPKM en diferentes muestras del estudio.

**stat\_results = stattest(bg2, feature=‘transcript’, meas=‘FPKM’, covariate=‘type’) stat\_results %>% head()**

El código que proporcionaste se utiliza para realizar pruebas estadísticas en los datos de expresión génica contenidos en el objeto ballgown llamado bg2. A continuación, se detallan las operaciones realizadas:

a. stattest(bg2, feature = 'transcript', meas = 'FPKM', covariate = 'type'): Se llama a la función stattest para realizar pruebas estadísticas en los datos contenidos en bg2. Los argumentos son los siguientes:  
  
b. bg2: Es el objeto ballgown que contiene los datos de expresión génica y metadatos.  
  
c. feature = 'transcript': Indica que se realizarán pruebas en el nivel de transcripción.  
  
d. meas = 'FPKM': Especifica la medida de expresión que se utilizará para las pruebas, en este caso, FPKM.  
e. covariate = 'type': Indica que se realizarán pruebas utilizando la variable categórica 'type' contenida en los metadatos de las muestras. Esta variable puede representar diferentes condiciones experimentales o grupos de tratamiento.  
stat\_results %>% head(): Después de realizar las pruebas estadísticas, se utiliza el operador %>% para mostrar las primeras filas de los resultados. La función head() se utiliza para mostrar las primeras filas del conjunto de resultados estadísticos.

Los resultados de stattest pueden incluir estadísticas de prueba, valores p, y otros valores estadísticos que indican si hay diferencias significativas en la expresión de transcripciones entre los grupos definidos por la variable categórica ‘type’.

En resumen, este código realiza pruebas estadísticas en los datos de expresión génica en el nivel de transcripción para analizar si hay diferencias significativas en la expresión de genes entre diferentes grupos definidos por la variable ‘type’. Luego, muestra las primeras filas de los resultados estadísticos para su inspección.

**stat\_results %>% filter(qval<=0.06)**

El código que proporcionaste filtra los resultados de las pruebas estadísticas contenidos en el objeto stat\_results para seleccionar las filas en las que el valor de q-valor (qval) sea menor o igual a 0.06. A continuación, se explica lo que hace este código:

a. stat\_results: Este es el objeto que contiene los resultados de las pruebas estadísticas realizadas anteriormente. Estos resultados pueden incluir estadísticas de prueba, valores p, q-valor (ajustado para controlar la tasa de errores tipo I), y otros valores estadísticos relacionados con la expresión génica.  
  
b. %>%: Se utiliza el operador %>% para encadenar una operación de filtrado a los resultados estadísticos. Esto permite aplicar la operación filter() directamente a stat\_results.  
  
c. filter(qval <= 0.06): La función filter() se utiliza para seleccionar las filas en las que el valor de q-valor (qval) sea menor o igual a 0.06. Esto significa que se retendrán solo las filas en las que las diferencias en la expresión génica sean estadísticamente significativas con un nivel de ajuste de p-valor del 0.06 o inferior.

La operación de filtrado se utiliza para identificar las transcripciones o genes cuyas diferencias en la expresión entre grupos son consideradas significativas en el contexto del análisis. Esto puede ser importante para identificar genes de interés o procesos biológicos relevantes en función de la variable ‘type’ o cualquier otra variable de interés utilizada en el análisis estadístico.

**bg\_table2 %>% select(c(t\_name, num\_exons, length), starts\_with(“FPKM”)) %>% mutate\_at(vars(starts\_with(“FPKM”)), ~ log2(.x+1)) %>% pivot\_longer(cols=starts\_with(“FPKM”), values\_to = “FPKM”, names\_to = “Sample”) %>% mutate(Sample=str\_replace(Sample, “FPKM.”, ““)) %>%  
mutate(Type = str\_replace(Sample,”\_\*(1|2)\_S\d+“,”“)) %>% mutate(Type = str\_replace(Type,”CmasM”, “EhCDC5-like+EhMyb10”)) %>% mutate(Type = str\_replace(Type, “cdc5”, “EhCDC5-like”)) %>% mutate(Type = str\_replace(Type, “pEhEx”, “Control”)) %>% group\_by(t\_name, Type) %>% summarise(FPKM=mean(FPKM))-> tmp\_data**

El código que proporcionaste realiza una serie de transformaciones en los datos de expresión génica contenidos en el objeto bg\_table2 y luego crea una nueva tabla resumida llamada tmp\_data. Aquí se detallan las operaciones paso a paso:

a. bg\_table2 %>% ...: El código comienza con el objeto bg\_table2, que contiene datos de expresión génica. Luego, utiliza el operador %>% para encadenar una serie de operaciones juntas. Esto permite realizar transformaciones sucesivas en los datos.  
  
b. select(c(t\_name, num\_exons, length), starts\_with("FPKM")): La función select se utiliza para seleccionar columnas específicas del DataFrame bg\_table2. Se seleccionan las columnas t\_name, num\_exons y length, además de todas las columnas que comienzan con "FPKM".  
  
c. mutate\_at(vars(starts\_with("FPKM")), ~ log2(.x+1)): La función mutate\_at se usa para aplicar una transformación a las columnas seleccionadas previamente (las que comienzan con "FPKM"). En este caso, se toma el logaritmo en base 2 (log2) de los valores FPKM con la adición de 1 para evitar problemas cuando los valores son cero.  
  
d. pivot\_longer(cols = starts\_with("FPKM"), values\_to = "FPKM", names\_to = "Sample"): La función pivot\_longer se utiliza para reorganizar los datos de manera que las columnas de expresión FPKM se transformen en una columna "FPKM" y una columna "Sample". Esto facilita la creación de una tabla resumida.  
  
e. mutate(Sample = str\_replace(Sample, "FPKM.", "")): La función mutate se utiliza para eliminar el prefijo "FPKM." de los nombres de las muestras en la columna "Sample".  
  
f. mutate(Type = str\_replace(Sample, "\_\*(1|2)\_S\\d+", "")): Se crea una nueva columna "Type" en la que se extrae la información de tipo a partir de los nombres de las muestras. Esto se logra utilizando expresiones regulares para extraer el tipo de muestra de los nombres de las muestras, que parece estar codificado en el formato "\_\*(1|2)\_S\d+". Esto se utiliza para agrupar las muestras por tipo.  
  
Luego, se utilizan múltiples llamadas a mutate para asignar etiquetas de tipo más legibles a los tipos extraídos. Por ejemplo, si el tipo contiene "CmasM", se cambia a "EhCDC5-like+EhMyb10". Si el tipo contiene "cdc5", se cambia a "EhCDC5-like". Si el tipo contiene "pEhEx", se cambia a "Control".  
  
g. group\_by(t\_name, Type): Se utiliza la función group\_by para agrupar los datos por los valores en las columnas "t\_name" (nombre de la transcripción) y "Type" (tipo de muestra).  
  
h. summarise(FPKM = mean(FPKM)): La función summarise se utiliza para calcular la media de los valores de FPKM dentro de cada grupo (por "t\_name" y "Type").  
  
i. -> tmp\_data: Los resultados se almacenan en un nuevo DataFrame llamado tmp\_data.

En resumen, este código realiza una serie de transformaciones en los datos de expresión génica para crear una tabla resumida llamada tmp\_data. En esta tabla, los valores de FPKM se han logaritmado, las muestras se han agrupado por tipo, y se ha calculado la media de los valores de FPKM para cada transcripción en función de su tipo de muestra. Esto puede ser útil para analizar patrones de expresión genética por tipo de muestra.

**tmp\_dataType, levels=c(“Control”,“EhMyb10”,“EhCDC5-like”,“EhCDC5-like+EhMyb10”)) tmp\_data %>% ggplot(aes(x=Type, y=FPKM, fill=Type)) + geom\_boxplot() + theme(axis.text.x = element\_text(angle = 45, vjust = 0.5, hjust=1)) -> p ggplotly(p)**

El código que proporcionaste se utiliza para crear una gráfica de cajas (boxplot) interactiva que muestra la distribución de los valores de expresión FPKM (Fragmentos Por Kilobase de millón) para diferentes tipos de muestras. Aquí se explica lo que hace el código:

a. tmp\_data$Type = factor(tmp\_data$Type, levels=c("Control","EhMyb10","EhCDC5-like","EhCDC5-like+EhMyb10")): En esta línea, se convierte la columna "Type" de tmp\_data en un factor. Se establece el orden de los niveles del factor para asegurar que las categorías aparezcan en el orden deseado en la gráfica. El orden especificado es "Control," "EhMyb10," "EhCDC5-like," y "EhCDC5-like+EhMyb10."  
  
b. %>%: Se utiliza el operador %>% para encadenar una serie de operaciones en los datos.  
  
c. ggplot(aes(x=Type, y=FPKM, fill=Type)): Se crea un objeto ggplot que se utilizará para generar la gráfica de cajas. Se especifica que "Type" se muestre en el eje x, "FPKM" en el eje y y que el relleno (fill) de las cajas sea según la variable "Type."  
  
d. geom\_boxplot(): Se agrega una capa de gráficos de cajas a la gráfica utilizando geom\_boxplot(). Esto mostrará la distribución de los valores de FPKM para cada tipo de muestra.  
  
e. theme(axis.text.x = element\_text(angle = 45, vjust = 0.5, hjust = 1)): Se utiliza theme() para personalizar el aspecto de la gráfica. En este caso, se rota el texto en el eje x en un ángulo de 45 grados y se ajusta la posición vertical (vjust) y horizontal (hjust) de las etiquetas del eje x.  
  
f. -> p: El resultado de la gráfica se asigna a la variable p.  
  
g. ggplotly(p): Finalmente, se utiliza la función ggplotly del paquete plotly para convertir la gráfica de ggplot en una gráfica interactiva que se puede explorar y personalizar en un entorno interactivo.

En resumen, este código crea una gráfica de cajas interactiva que muestra la distribución de los valores de expresión FPKM para diferentes tipos de muestras, con las categorías de muestra en el orden especificado. La gráfica interactiva permite explorar y analizar los patrones de expresión genética de manera dinámica.

**bg\_table2 %>% select(c(t\_name, num\_exons, length), starts\_with(“FPKM”)) %>% mutate(mean\_FPKM\_pEhEx=(FPKM.pEhEx\_2\_S8+FPKM.pEhEx1\_S7)/2) %>% mutate\_at(vars(starts\_with(“FPKM”)), ~ .x/mean\_FPKM\_pEhEx) %>% head()**

El código que proporcionaste realiza una serie de operaciones en los datos contenidos en el objeto bg\_table2. A continuación, se detallan las operaciones paso a paso:

a. bg\_table2 %>% ...: El código comienza con el objeto bg\_table2, que contiene datos de expresión génica. Luego, utiliza el operador %>% para encadenar una serie de operaciones juntas.  
  
b. select(c(t\_name, num\_exons, length), starts\_with("FPKM")): La función select se utiliza para seleccionar columnas específicas del DataFrame bg\_table2. Se seleccionan las columnas t\_name, num\_exons y length, además de todas las columnas que comienzan con "FPKM".  
  
c. mutate(mean\_FPKM\_pEhEx = (FPKM.pEhEx\_2\_S8 + FPKM.pEhEx1\_S7) / 2): La función mutate se utiliza para crear una nueva columna llamada mean\_FPKM\_pEhEx. En esta columna se calcula la media de los valores FPKM de las muestras "pEhEx\_2\_S8" y "pEhEx1\_S7". Esto se hace dividiendo la suma de los valores FPKM entre 2.  
  
d. mutate\_at(vars(starts\_with("FPKM")), ~ .x / mean\_FPKM\_pEhEx): La función mutate\_at se utiliza para aplicar una transformación a todas las columnas que comienzan con "FPKM". En este caso, se divide cada valor FPKM por el valor promedio de FPKM calculado en la columna mean\_FPKM\_pEhEx.

El resultado de estas operaciones es que los valores FPKM en las columnas que comienzan con “FPKM” se han normalizado dividiendo cada valor por el valor promedio de FPKM de las muestras “pEhEx\_2\_S8” y “pEhEx1\_S7”. Esto puede ser útil para comparar la expresión relativa de genes entre diferentes muestras y normalizar los datos en función de una muestra de referencia.

**bg\_table2 %>% select(c(t\_name, num\_exons, length), starts\_with(“FPKM”)) %>% mutate(exprpEhEx=(FPKM.pEhEx\_2\_S8+FPKM.pEhEx1\_S7)/2) %>% mutate\_at(vars(starts\_with(“FPKM”)), ~ .x/exprpEhEx) %>% mutate\_at(vars(starts\_with(“FPKM”)), ~ log2(.x)) %>% pivot\_longer(cols=starts\_with(“FPKM”), values\_to = “log2FC”, names\_to = “Sample”) %>% mutate(Sample=str\_replace(Sample, “FPKM.”, ““)) %>% mutate(Type = str\_replace(Sample,”\_\*(1|2)\_S\d+“,”“)) %>% mutate(Type = str\_replace(Type,”CmasM”, “EhCDC5-like+EhMyb10”)) %>% mutate(Type = str\_replace(Type, “cdc5”, “EhCDC5-like”)) %>% mutate(Type = str\_replace(Type, “pEhEx”, “Control”)) %>% group\_by(t\_name, Type) %>% summarise(log2FC=mean(log2FC)) -> tmp\_data**

El código que proporcionaste realiza una serie de transformaciones en los datos de expresión génica contenidos en el objeto bg\_table2 y crea una nueva tabla resumida llamada tmp\_data. A continuación, se detallan las operaciones paso a paso:

a. bg\_table2 %>% ...: El código comienza con el objeto bg\_table2, que contiene datos de expresión génica. Luego, utiliza el operador %>% para encadenar una serie de operaciones juntas. Esto permite realizar transformaciones sucesivas en los datos.  
  
b. select(c(t\_name, num\_exons, length), starts\_with("FPKM")): La función select se utiliza para seleccionar columnas específicas del DataFrame bg\_table2. Se seleccionan las columnas t\_name, num\_exons y length, además de todas las columnas que comienzan con "FPKM".  
  
c. mutate(exprpEhEx = (FPKM.pEhEx\_2\_S8 + FPKM.pEhEx1\_S7) / 2): La función mutate se utiliza para crear una nueva columna llamada exprpEhEx. En esta columna, se calcula la media de los valores FPKM de las muestras "pEhEx\_2\_S8" y "pEhEx1\_S7" dividiendo la suma de estos valores por 2.  
  
d. mutate\_at(vars(starts\_with("FPKM")), ~ .x/exprpEhEx): La función mutate\_at se utiliza para aplicar una transformación a todas las columnas que comienzan con "FPKM". En este caso, se divide cada valor FPKM por el valor calculado en la columna exprpEhEx.  
  
e. mutate\_at(vars(starts\_with("FPKM")), ~ log2(.x)): Se aplica una nueva transformación a todas las columnas que comienzan con "FPKM". En este caso, se toma el logaritmo en base 2 de cada valor FPKM. Esto es común en análisis de expresión génica para estabilizar las varianzas.  
  
f. pivot\_longer(cols = starts\_with("FPKM"), values\_to = "log2FC", names\_to = "Sample"): La función pivot\_longer se utiliza para reorganizar los datos de manera que las columnas de expresión FPKM se transformen en una columna "log2FC" y una columna "Sample". Esto facilita la creación de una tabla resumida.  
  
g. mutate(Sample = str\_replace(Sample, "FPKM.", "")): La función mutate se utiliza para eliminar el prefijo "FPKM." de los nombres de las muestras en la columna "Sample".  
  
h. mutate(Type = str\_replace(Sample, "\_\*(1|2)\_S\\d+", "")): Se crea una nueva columna "Type" en la que se extrae la información de tipo a partir de los nombres de las muestras utilizando expresiones regulares.

Luego, se utilizan múltiples llamadas a mutate para asignar etiquetas de tipo más legibles a los tipos extraídos, similar a lo que se hizo en el código anterior.

i. group\_by(t\_name, Type): Se utiliza la función group\_by para agrupar los datos por los valores en las columnas "t\_name" (nombre de la transcripción) y "Type" (tipo de muestra).  
  
j. summarise(log2FC = mean(log2FC)): La función summarise se utiliza para calcular la media de los valores de "log2FC" (log2 del fold change) dentro de cada grupo (por "t\_name" y "Type").  
  
k. -> tmp\_data: Los resultados se almacenan en un nuevo DataFrame llamado tmp\_data.

En resumen, este código realiza una serie de transformaciones en los datos de expresión génica para crear una tabla resumida llamada tmp\_data. En esta tabla, los valores FPKM se normalizan, se toma el logaritmo en base 2 de los valores y se calcula la media del fold change (log2FC) de los genes por tipo de muestra. Esto puede ser útil para analizar patrones de expresión genética y comparar la expresión entre diferentes tipos de muestras.

**tmp\_dataType, levels=c(“Control”,“EhMyb10”,“EhCDC5-like”,“EhCDC5-like+EhMyb10”)) tmp\_data %>% ggplot(aes(x=Type, y=log2FC, fill=Type)) + geom\_boxplot() + theme(axis.text.x = element\_text(angle = 20, vjust = 0.5, hjust=1)) -> p ggplotly(p)**

Este código crea una gráfica de cajas (boxplot) interactiva que muestra la distribución de los valores del logaritmo en base 2 del fold change (log2FC) para diferentes tipos de muestras. A continuación, se explica el funcionamiento del código:

a. tmp\_data$Type = factor(tmp\_data$Type, levels=c("Control","EhMyb10","EhCDC5-like","EhCDC5-like+EhMyb10")): En esta línea, la columna "Type" de tmp\_data se convierte en un factor y se especifica el orden de los niveles del factor. Esto asegura que las categorías de tipo se muestren en el orden deseado en la gráfica. El orden especificado es "Control," "EhMyb10," "EhCDC5-like" y "EhCDC5-like+EhMyb10."  
  
b. %>%: Se utiliza el operador %>% para encadenar una serie de operaciones en los datos.  
  
c. ggplot(aes(x=Type, y=log2FC, fill=Type)): Se crea un objeto ggplot que se utilizará para generar la gráfica de cajas. Se especifica que "Type" se muestre en el eje x, "log2FC" en el eje y y que el relleno (fill) de las cajas sea según la variable "Type."  
  
d. geom\_boxplot(): Se agrega una capa de gráficos de cajas a la gráfica utilizando geom\_boxplot(). Esto mostrará la distribución de los valores de log2FC para cada tipo de muestra.  
  
e. theme(axis.text.x = element\_text(angle = 20, vjust = 0.5, hjust = 1): Se utiliza theme() para personalizar el aspecto de la gráfica. En este caso, se rota el texto en el eje x en un ángulo de 20 grados y se ajusta la posición vertical (vjust) y horizontal (hjust) de las etiquetas del eje x.  
  
f. -> p: El resultado de la gráfica se asigna a la variable p.  
  
g. ggplotly(p): Finalmente, se utiliza la función ggplotly del paquete plotly para convertir la gráfica de ggplot en una gráfica de cajas interactiva. Esto permite explorar y personalizar la gráfica en un entorno interactivo.

En resumen, este código crea una gráfica de cajas interactiva que muestra la distribución de los valores del logaritmo en base 2 del fold change (log2FC) para diferentes tipos de muestras, con las categorías de muestra en el orden especificado. La gráfica interactiva permite explorar y analizar los patrones de expresión genética de manera dinámica.

**bg\_table2 %>% select(c(t\_name, num\_exons, length), starts\_with(“FPKM”)) %>% mutate(exprpEhEx=(FPKM.pEhEx\_2\_S8+FPKM.pEhEx1\_S7)/2) %>% mutate\_at(vars(starts\_with(“FPKM”)), ~ .x/exprpEhEx) %>% mutate\_at(vars(starts\_with(“FPKM”)), ~ log2(.x)) %>% pivot\_longer(cols=starts\_with(“FPKM”), values\_to = “log2FC”, names\_to = “Sample”) %>% mutate(Sample=str\_replace(Sample, “FPKM.”, ““)) %>% filter(!grepl(”^U”,Sample)) %>% ggplot(aes(x=exprpEhEx, y=log2FC, color=Sample)) + geom\_point() + scale\_x\_log10() + scale\_color\_brewer(palette = “Set1”)**

El código que proporcionaste realiza una serie de operaciones en los datos de expresión génica contenidos en el objeto bg\_table2 y crea un gráfico de dispersión (scatter plot) utilizando ggplot2. A continuación, se detallan las operaciones paso a paso:

a. bg\_table2 %>% ...: El código comienza con el objeto bg\_table2, que contiene datos de expresión génica. Luego, utiliza el operador %>% para encadenar una serie de operaciones juntas.  
  
b. select(c(t\_name, num\_exons, length), starts\_with("FPKM")): La función select se utiliza para seleccionar columnas específicas del DataFrame bg\_table2. Se seleccionan las columnas t\_name, num\_exons y length, además de todas las columnas que comienzan con "FPKM".  
  
c. mutate(exprpEhEx = (FPKM.pEhEx\_2\_S8 + FPKM.pEhEx1\_S7) / 2): La función mutate se utiliza para crear una nueva columna llamada exprpEhEx. En esta columna, se calcula la media de los valores FPKM de las muestras "pEhEx\_2\_S8" y "pEhEx1\_S7" dividiendo la suma de estos valores por 2.  
  
d. mutate\_at(vars(starts\_with("FPKM")), ~ .x/exprpEhEx): La función mutate\_at se utiliza para aplicar una transformación a todas las columnas que comienzan con "FPKM". En este caso, se divide cada valor FPKM por el valor calculado en la columna exprpEhEx.  
  
e. mutate\_at(vars(starts\_with("FPKM")), ~ log2(.x)): Se aplica una nueva transformación a todas las columnas que comienzan con "FPKM". En este caso, se toma el logaritmo en base 2 de cada valor FPKM.  
  
f. pivot\_longer(cols = starts\_with("FPKM"), values\_to = "log2FC", names\_to = "Sample"): La función pivot\_longer se utiliza para reorganizar los datos de manera que las columnas de expresión FPKM se transformen en una columna "log2FC" y una columna "Sample". Esto facilita la creación de un gráfico de dispersión.  
  
g. mutate(Sample = str\_replace(Sample, "FPKM.", "")): La función mutate se utiliza para eliminar el prefijo "FPKM." de los nombres de las muestras en la columna "Sample".  
  
h. filter(!grepl("^U", Sample)): La función filter se utiliza para eliminar las muestras que comienzan con "U" en la columna "Sample" utilizando una expresión regular. Esto permite excluir muestras que cumplan con este patrón.  
  
i. ggplot(aes(x = exprpEhEx, y = log2FC, color = Sample)): Se crea un objeto ggplot que se utilizará para generar el gráfico de dispersión. Se especifica que "exprpEhEx" se muestre en el eje x, "log2FC" en el eje y, y que el color de los puntos sea según la variable "Sample".  
  
j. geom\_point(): Se agrega una capa de puntos al gráfico utilizando geom\_point(). Esto crea un gráfico de dispersión donde cada punto representa una muestra y su posición en los ejes x e y está determinada por "exprpEhEx" y "log2FC" respectivamente.  
  
k. scale\_x\_log10(): Se escala el eje x utilizando una escala logarítmica en base 10 para "exprpEhEx". Esto puede ser útil si los valores en este eje abarcan un rango amplio y se desean visualizar en una escala logarítmica.  
  
l. scale\_color\_brewer(palette = "Set1"): Se ajusta la paleta de colores utilizada en el gráfico. En este caso, se utiliza una paleta de colores de Brewer llamada "Set1" para colorear los puntos según la variable "Sample".

En resumen, este código realiza una serie de transformaciones en los datos de expresión génica y crea un gráfico de dispersión para visualizar la relación entre “exprpEhEx” y “log2FC” para diferentes muestras. Los puntos se colorean según la variable “Sample” y el eje x se escala de manera logarítmica.

**Warning: Transformation introduced infinite values in continuous x-axis**

El mensaje de advertencia “Transformation introduced infinite values in continuous x-axis” indica que al aplicar una transformación logarítmica en base 10 a los valores del eje x (en este caso, “exprpEhEx”), se han generado valores infinitos o indefinidos en el eje x. Esto puede ocurrir cuando algunos de los valores originales en “exprpEhEx” son iguales a cero o negativos, ya que el logaritmo en base 10 de cero o un número negativo es indefinido.

Para solucionar este problema, puedes considerar las siguientes opciones:

a. Manejar los valores problemáticos: Identifica los valores problemáticos en la columna "exprpEhEx" que están causando el problema. Puedes tratar de ajustar estos valores de alguna manera que tenga sentido para tus datos. Por ejemplo, podrías sumar una pequeña constante positiva a todos los valores antes de aplicar el logaritmo para evitar el valor cero. Ten en cuenta que esta corrección debe ser coherente con el significado de tus datos.  
  
b. Filtrar valores no deseados: Si ciertos valores en "exprpEhEx" no son relevantes o no deberían estar en el gráfico, puedes filtrarlos antes de realizar la transformación logarítmica.  
  
c. Considerar otra escala: Si los valores en "exprpEhEx" abarcan un rango amplio y algunos de ellos son cero o negativos, es posible que una escala logarítmica no sea la más adecuada. En lugar de una escala logarítmica, puedes considerar usar una escala diferente, como una escala cuadrada o raíz cuadrada, según lo que sea más apropiado para tus datos.  
  
d. Imputar valores faltantes: Si algunos valores en "exprpEhEx" están ausentes o faltan, puedes considerar imputar valores para que no haya valores faltantes en los datos antes de aplicar la transformación logarítmica.

Recuerda que la elección de la transformación y la manipulación de los datos debe basarse en un entendimiento sólido del significado de los datos y los objetivos de tu análisis. Asegúrate de que cualquier transformación o ajuste que realices sea relevante y justificable en el contexto de tus datos y tu pregunta de investigación.

**Warning: Removed 8213 rows containing missing values (geom\_point()).**

El mensaje de advertencia “Removed 8213 rows containing missing values (geom\_point())” indica que se eliminaron 8,213 filas de datos de la gráfica de dispersión (geom\_point()) debido a la presencia de valores faltantes en una o ambas de las variables utilizadas en el gráfico (en este caso, “exprpEhEx” y “log2FC”). Los valores faltantes suelen representarse como “NA” en R.

Para abordar este problema, puedes considerar las siguientes opciones:

a. Identificar y gestionar los valores faltantes: Lo primero que debes hacer es identificar por qué existen valores faltantes en tus datos. Pueden deberse a datos incompletos, errores en la recopilación de datos o problemas con las transformaciones previas que realizaste. Una vez que identifiques la causa, puedes decidir cómo manejar los valores faltantes. Algunas opciones incluyen imputar valores, eliminar filas con valores faltantes o investigar más a fondo para obtener los datos faltantes.  
  
b. Imputación de valores faltantes: Si los valores faltantes son legítimos y no se pueden obtener más datos, puedes considerar la imputación de valores. La imputación implica estimar o reemplazar los valores faltantes por otros valores que tengan sentido en el contexto de tus datos. Las técnicas de imputación pueden incluir la sustitución por la media, la mediana, u otros métodos estadísticos.  
  
c. Revisar la calidad de los datos de origen: Asegúrate de que los datos de origen utilizados para crear la columna "exprpEhEx" y "log2FC" sean precisos y estén completos. Puedes revisar el proceso de recolección y transformación de datos para identificar posibles problemas.  
  
d. Filtrar o ajustar los datos problemáticos: Si tienes una comprensión clara de por qué algunos valores son faltantes o extremos, puedes considerar filtrarlos o ajustarlos de acuerdo con la lógica de tus datos.

En resumen, el mensaje de advertencia “Removed 8213 rows containing missing values (geom\_point())” indica que existen valores faltantes en tus datos, y es importante abordar este problema antes de continuar con el análisis o la representación gráfica. La gestión de valores faltantes es fundamental para garantizar la integridad y la validez de tus resultados.