Documentando Codigo Claudia

Carlos E Martinez R

2023-10-18

Explicación detallada del siguiente codigo:

‘r, *eval=FALSE* ids = c(“cdc5_1_S9”,“cdc5_2_S10”,“CmasM_1_S15”,“CmasM_2_S16”,“EhMyb10_1_S11”,“EhMyb10_2_
“pEhEx1_S7”)

type = c(“EhCDC5-like”,“EhCDC5-like”,“EhCDC5-like+EhMyb10”,“EhCDC5-like+EhMyb10”,“EhMyb10”,“EhMyb10”,“Cont

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.

‘r, *eval=FALSE* 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.

‘r, *eval=FALSE* 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.

‘r, *eval=FALSE*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.

‘r, *eval=FALSE*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.

‘r, *eval=FALSE*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.

‘r, *eval=FALSE* 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

b. ballgown(dataDir = "~/ballgown_r2/", samplePattern = "_S", pData = pheno_data): Esto es una llamada a

c. dataDir: Es el directorio donde se encuentran los archivos relacionados con los datos de RNA-seq. En

d. samplePattern: Es un patrón que se utilizará para identificar los archivos relacionados con las muest

e. pData: Aquí se especifica el DataFrame pheno_data que creaste anteriormente. Este DataFrame contiene

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.

‘*r, eval=FALSE*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

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 est

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.

‘*r, eval=FALSE*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 an

b. %>%: Este es el operador de tubería (pipe) utilizado en R, que permite encadenar funciones o manipul

c. head(): La función head() se utiliza comúnmente para mostrar las primeras filas de un DataFrame o una

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.

‘*r, eval=FALSE*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() s

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.

'r, eval=FALSE 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

b. select(c(t_name, num_exons, length), starts_with("FPKM")): La función select se utiliza para seleccio

c. mutate_at(vars(starts_with("FPKM")), ~ log2(.x+1)): La función mutate_at se usa para aplicar una tra

d. pivot_longer(cols = starts_with("FPKM"), values_to = "FPKM", names_to = "Sample"): La función pivot_l

e. mutate(Sample = str_replace(Sample, "FPKM.", "")): La función mutate se utiliza para eliminar el pre

f. ggplot(aes(x = Sample, y = FPKM, fill = Sample)) + ...: Se crea un objeto ggplot que se utiliza para

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

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.

'r, eval=FALSE 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 statt

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, FPK
e. covariate = 'type': Indica que se realizarán pruebas utilizando la variable categórica 'type' conteni
stat_results %>% head(): Después de realizar las pruebas estadísticas, se utiliza el operador %>% para r

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.

'r, eval=FALSE 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 ar

b. %>%: Se utiliza el operador %>% para encadenar una operación de filtrado a los resultados estadístico

c. filter(qval <= 0.06): La función filter() se utiliza para seleccionar las filas en las que el valor d

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.

'r, eval=FALSE 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

b. select(c(t_name, num_exons, length), starts_with("FPKM")): La función select se utiliza para seleccio

c. mutate_at(vars(starts_with("FPKM")), ~ log2(.x+1)): La función mutate_at se usa para aplicar una tra

d. pivot_longer(cols = starts_with("FPKM"), values_to = "FPKM", names_to = "Sample"): La función pivot_l

e. mutate(Sample = str_replace(Sample, "FPKM.", "")): La función mutate se utiliza para eliminar el pre

f. mutate(Type = str_replace(Sample, "_*(1|2)_S\\d+", "")): Se crea una nueva columna "Type" en la que s

Luego, se utilizan múltiples llamadas a mutate para asignar etiquetas de tipo más legibles a los tipos d

g. group_by(t_name, Type): Se utiliza la función group_by para agrupar los datos por los valores en las

h. summarise(FPKM = mean(FPKM)): La función summarise se utiliza para calcular la media de los valores d

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.

‘r, *eval=FALSE* tmp_data$Type = factor(tmp_data$Type, 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+EhMyb1`

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 gra`

d. `geom_boxplot(): Se agrega una capa de gráficos de cajas a la gráfica utilizando geom_boxplot(). Esto`

e. `theme(axis.text.x = element_text(angle = 45, vjust = 0.5, hjust = 1)): Se utiliza theme() para perso`

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`

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.

‘r, *eval=FALSE* 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`

b. `select(c(t_name, num_exons, length), starts_with("FPKM")): La función select se utiliza para selecci`

c. `mutate(mean_FPKM_pEhEx = (FPKM.pEhEx_2_S8 + FPKM.pEhEx1_S7) / 2): La función mutate se utiliza para`

d. `mutate_at(vars(starts_with("FPKM")), ~ .x / mean_FPKM_pEhEx): La función mutate_at se utiliza para a`

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.

‘r, *eval=FALSE* 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

b. select(c(t_name, num_exons, length), starts_with("FPKM")): La función select se utiliza para seleccio

c. mutate(exprpEhEx = (FPKM.pEhEx_2_S8 + FPKM.pEhEx1_S7) / 2): La función mutate se utiliza para crear u

d. mutate_at(vars(starts_with("FPKM")), ~ .x/exprpEhEx): La función mutate_at se utiliza para aplicar un

e. mutate_at(vars(starts_with("FPKM")), ~ log2(.x)): Se aplica una nueva transformación a todas las colu

f. pivot_longer(cols = starts_with("FPKM"), values_to = "log2FC", names_to = "Sample"): La función pivot

g. mutate(Sample = str_replace(Sample, "FPKM.", "")): La función mutate se utiliza para eliminar el pref

h. mutate(Type = str_replace(Sample, "_*(1|2)_S\\d+", "")): Se crea una nueva columna "Type" en la que s

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

j. summarise(log2FC = mean(log2FC)): La función summarise se utiliza para calcular la media de los valor

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.

‘*r, eval=FALSE* tmp_data$Type = factor(tmp_data$Type, 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+EhMyb1

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 g

d. geom_boxplot(): Se agrega una capa de gráficos de cajas a la gráfica utilizando geom_boxplot(). Esto

e. theme(axis.text.x = element_text(angle = 20, vjust = 0.5, hjust = 1): Se utiliza theme() para persona

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

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.

‘*r, eval=FALSE* 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

b. `select(c(t_name, num_exons, length), starts_with("FPKM"))`: La función select se utiliza para seleccio

c. `mutate(exprpEhEx = (FPKM.pEhEx_2_S8 + FPKM.pEhEx1_S7) / 2)`: La función mutate se utiliza para crear u

d. `mutate_at(vars(starts_with("FPKM")), ~ .x/exprpEhEx)`: La función mutate_at se utiliza para aplicar un

e. `mutate_at(vars(starts_with("FPKM")), ~ log2(.x))`: Se aplica una nueva transformación a todas las colu

f. `pivot_longer(cols = starts_with("FPKM"), values_to = "log2FC", names_to = "Sample")`: La función pivot

g. `mutate(Sample = str_replace(Sample, "FPKM.", ""))`: La función mutate se utiliza para eliminar el pref

h. `filter(!grepl("^U", Sample))`: La función filter se utiliza para eliminar las muestras que comienzan c

i. `ggplot(aes(x = exprpEhEx, y = log2FC, color = Sample))`: Se crea un objeto ggplot que se utilizará par

j. `geom_point()`: Se agrega una capa de puntos al gráfico utilizando geom_point(). Esto crea un gráfico d

k. `scale_x_log10()`: Se escala el eje x utilizando una escala logarítmica en base 10 para "exprpEhEx". Es

l. `scale_color_brewer(palette = "Set1")`: Se ajusta la paleta de colores utilizada en el gráfico. En esto

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.

‘*r, eval=FALSE* 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

b. Filtrar valores no deseados: Si ciertos valores en "exprpEhEx" no son relevantes o no deberían estar

c. Considerar otra escala: Si los valores en "exprpEhEx" abarcan un rango amplio y algunos de ellos son

d. Imputar valores faltantes: Si algunos valores en "exprpEhEx" están ausentes o faltan, puedes consider

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.

‘r, eval=FALSE 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 exis`

`b. Imputación de valores faltantes: Si los valores faltantes son legítimos y no se pueden obtener más da`

`c. Revisar la calidad de los datos de origen: Asegúrate de que los datos de origen utilizados para crea`

`d. Filtrar o ajustar los datos problemáticos: Si tienes una comprensión clara de por qué algunos valores`

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.