**TIDYVERSE**

**Introducción por CHAPGPT:**

La librería "tidyverse" es un conjunto de paquetes de R diseñados para trabajar juntos de manera coherente y facilitar el análisis de datos. Fue desarrollado por Hadley Wickham y su equipo en RStudio, y se ha convertido en una de las librerías más populares y útiles en el mundo del análisis de datos con R.

El "tidyverse" incluye una serie de paquetes como "ggplot2", "dplyr", "tidyr", "readr", "purrr", "stringr" y otros más. Cada uno de estos paquetes aborda un aspecto específico del análisis de datos, desde la manipulación y limpieza de datos hasta la visualización y modelado.

Lo que hace que el "tidyverse" sea tan útil es que los paquetes se integran bien entre sí y utilizan una sintaxis coherente y fácil de entender. También promueve la filosofía "tidy data", que se refiere a la organización de los datos en tablas bien estructuradas y fáciles de analizar.

En resumen, el "tidyverse" es una herramienta poderosa y útil para el análisis de datos en R, y su popularidad sigue en aumento debido a su facilidad de uso y eficacia.

**Introducción:**

Como vemos en la introducción anterior, la IA de chatGPT nos da una idea bastante acertada de lo que es el universo ordenado o el “tidyverse”, aquí es donde R realmente se desprende de otros lenguajes que se utilizan para el análisis de datos, los lenguajes como Julia y Python han estado copiando estos esquemas y funciones desde hace ya algunos años, por ejemplo los famosos Pandas de Python se copiaron literalmente del dplyr, al ser código libre, se pueden hacer este tipo de copias hacia otros lenguajes, tengo entendido que Julia actualmente esta trabajando en la adaptación de la misma librería para su código.

Hasta ahora hemos manipulado algunos vectores, reordenándolos y creando conjuntos mediante índices. Sin embargo, cuando entramos a análisis mucho mas avanzados, la unidad de medida para almacenar los datos no son los vectores sino los dataframes, o en su forma avanzada el tibble.

**¿Qué vamos a aprender?**

Aprenderemos a trabajar directamente con dataframes y tibbles, lo que nos va a facilitar enormemente la organización de nuestra información. Utilizaremos dataframes en la gran mayoría de casos. Nos vamos a centrar en un formato de datos especifico llamado “tidy” además de una colección especifica de paquetes que serán particularmente útiles para trabajar con datos ordenados.

**Iniciando**

Si aun no lo hemos realizado procederemos a instalar nuestra librería de tidyverse además de dslabs con el siguiente comando:

install.packages**(**"tidyverse"**)**

install.packages**(**"dslabs"**)**

Aprenderemos como implementar el enfoque tidyverse a lo largo del curso, antes de profundizar en los detalles veremos a continuación algunas de las funcionalidades mas utilizadas en el tidyverse.

Vamos a comenzar con el paquete dplyr para manipular datos y el paquete purr para trabajar con funciones. Tengamos en cuenta que tidyverse incluye varios paquetes mas que iremos viendo poco a poco, lo primero es presentar el concepto y luego demostraremos y practicaremos como trabajar los datos en estos formatos.

**Tidy Data**

Podemos decir que el concepto de “tidy data”, se refiere a unas tablas que se encuentran en un formato ordenado si cada fila representa una observación y las columnas representan las diferentes variables disponibles para cada una de estas observaciones. En este caso veremos un conjunto de datos de la librería dslabs, llamado murders que es un dataframe en formato tidy.

library**(**tidyverse**)**

library**(**dslabs**)**

data**(**murders**)**

head**(**murders**)**

Veremos algo parecido a esto:

**>** head**(**murders**)**

state abb region population total

1 Alabama AL South 4779736 135

2 Alaska AK West 710231 19

3 Arizona AZ West 6392017 232

4 Arkansas AR South 2915918 93

5 California CA West 37253956 1257

6 Colorado CO West 5029196 65

**>**

Vemos como cada fila representa un estado y cada una de las 5 columnas proporcionan variables diferentes en relación con los mismos: nombre, abreviatura, región, población y total que corresponde al No. De asesinatos por estado.

Vamos a considerar el siguiente ejemplo para poder ver información en otros formatos:

gapminder %>% filter**(**country **==** c**(**"Germany", "South Korea"**))** %>% select**(**c**(**country, year, fertility**))** %>% head**()**

Obtenemos algo parecido a esto:

**>** gapminder %>% filter**(**country **==** c**(**"Germany", "South Korea"**))** %>% select**(**c**(**country, year, fertility**))** %>% head**()**

country year fertility

1 Germany 1960 2.41

2 South Korea 1960 6.16

3 Germany 1962 2.47

4 South Korea 1962 5.79

5 Germany 1964 2.49

6 South Korea 1964 5.36

Este conjunto de datos ordenado proporciona tasas de fertilidad para dos países a lo largo de los años. Este es un conjunto de datos ordenado porque cada fila presenta una observación con las tres variables: país, año y tasa de fertilidad.

Sin embargo, este conjunto de datos originalmente vino en otro formato y se reformó para el paquete dslabs. Originalmente, los datos estaban en el siguiente formato:



Aquí el código para acceder desde R:

dat **<-** read\_csv**(**"data/sp\_dyn\_tfrt\_in.csv"**)**

dat %>% filter**(**country **==** "Germany"**)** %>% head**()**

Les pasaré el archivo o lo colgare en OneDrive para su descarga.

Se proporciona la misma información, pero hay dos diferencias importantes en el formato:

1. Cada fila incluye varias observaciones.
2. Una de las variables, año, se almacena en el encabezado.

Para que los paquetes tidyverse se utilicen de manera óptima, los datos deben volver a formarse en un formato ordenado (tidy), que aprenderá a hacer en la parte de domado de datos. Hasta entonces, utilizaremos conjuntos de datos de ejemplo que ya están en formato ordenado (tidy).

Aunque no es inmediatamente obvio, a medida que avance en el curso comenzará a apreciar las ventajas de trabajar en un marco en el que las funciones utilizan formatos ordenados tanto para las entradas como para las salidas. Verá cómo esto permite que el analista de datos se centre en aspectos más importantes del análisis en lugar del formato de los datos.

**DPLYR**

El paquete dplyr del tidyverse presenta funciones que realizan algunas de las operaciones más comunes cuando se trabaja con dataframes y usa nombres para estas funciones que son relativamente fáciles de recordar.

**Por ejemplo:**

* Para cambiar la tabla de datos agregando una nueva columna, usamos mutate().
* Para filtrar la tabla de datos a un subconjunto de filas, usamos filter().
* Finalmente, para subdividir los datos seleccionando columnas específicas, usamos select().

En este enlace podemos bajar la hoja de ayuda tanto para dplyr combinada con tidyr que se llama domado de datos o data Wrangling.

<https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf>

Función mutate()

Vamos a aprender a usar esta función de la librería dplyr, esta función nos permite agregar una columna extra con distintos cálculos, o llamando otras funciones.

Como ejemplo, vamos a querer que dentro del propio dataframe exista una columna extra en donde agregaremos la tasa de asesinatos por estado.

La función mutate() toma el dataframe como primer argumento y el nombre y los valores de la variable como segundo argumento usando la convención nombre = valores.

Entonces, para agregar tasas de asesinatos por cada 100,00o habitantes, usamos:

library**(**tidyverse**)**

library**(**dslabs**)**

data**(**"murders"**)**

murders **<-** mutate**(**murders, rate **=** total**/**population **\*** 10**^**5**)**

Aquí asignamos al objeto murders la opción de mutate, el primer argumento es el propio dataframe, el segundo es la operación de tasa que será igual al total de asesinatos dividido la población, la cual multiplicamos por 10 elevado a la 5 que es igual a 100,000.

Observe que aquí usamos total y población dentro de la función, que son objetos que no están definidos en nuestro espacio de trabajo. Pero ¿por qué no recibimos un error?

Esta es una de las principales características de dplyr. Las funciones en este paquete, como mutate, saben buscar variables en el dataframe proporcionado en el primer argumento. En la llamada a mutate() anterior, total tendrá los valores en murders$total. Este enfoque hace que el código sea mucho más legible.

Ahora observamos los cambios con la función head() y obtendremos algo como esto:

head**(**murders**)**

state abb region population total rate

1 Alabama AL South 4779736 135 2.824424

2 Alaska AK West 710231 19 2.675186

3 Arizona AZ West 6392017 232 3.629527

4 Arkansas AR South 2915918 93 3.189390

5 California CA West 37253956 1257 3.374138

6 Colorado CO West 5029196 65 1.292453

Observación:

Aunque hemos sobrescrito el objeto de asesinatos original, esto no cambia el objeto que se cargó con datos (murders). Si volvemos a cargar los datos murders, el original sobrescribirá nuestra versión mutada.

Función filter()

Ahora suponga que queremos filtrar la tabla de datos para mostrar solo las entradas para las cuales la tasa de homicidios es inferior a 0,71.

Para hacer esto, utilizamos la función filter(), que toma la tabla de datos como el primer argumento y luego la declaración condicional como el segundo. Al igual que mutate(), podemos usar los nombres de variables sin comillas de murders dentro de la función y sabrá que nos referimos a las columnas y no a los objetos en el espacio de trabajo.

filter**(**murders, rate **<=** 0.71**)**

state abb region population total rate

1 Hawaii HI West 1360301 7 0.5145920

2 Iowa IA North Central 3046355 21 0.6893484

3 New Hampshire NH Northeast 1316470 5 0.3798036

4 North Dakota ND North Central 672591 4 0.5947151

5 Vermont VT Northeast 625741 2 0.3196211

Función select()

Aunque nuestra tabla de datos solo tiene seis columnas, algunas tablas de datos incluyen cientos. Si queremos ver algunas, podemos usar la función dplyr select().

En el siguiente código seleccionamos tres columnas, asignamos esto a un nuevo objeto y luego filtramos el nuevo objeto:

new\_table **<-** select**(**murders, state, region, rate**)**

filter**(**new\_table, rate **<=** 0.71**)**

En la llamada a select(), el primer argumento de murders es un objeto en este caso un dataframe, pero estado, región y tasa son nombres de variables.

Nuestro resultado será muy similar a este:

**>** new\_table **<-** select**(**murders, state, region, rate**)**

**>** filter**(**new\_table, rate **<=** 0.71**)**

state region rate

1 Hawaii West 0.5145920

2 Iowa North Central 0.6893484

3 New Hampshire Northeast 0.3798036

4 North Dakota North Central 0.5947151

5 Vermont Northeast 0.3196211

Operadores de tubería (pipes)

Con dplyr podemos realizar una serie de operaciones, por ejemplo, seleccionar y luego filtrar, enviando los resultados de una función a otra y así sucesivamente utilizando lo que se llama operador de tubería o pipes en inglés: **%>%**.

Escribimos el código anterior para mostrar tres variables (estado, región, tasa) para los estados que tienen tasas de homicidio por debajo de 0,71. Para hacer esto, definimos el objeto intermedio new\_table. En dplyr podemos escribir código que se parezca más a una descripción de lo que queremos hacer sin objetos intermedios:

* original data → select → filter

Para tal operación, podemos usar la tubería%>% (pipes). El código se ve así:

library**(**tidyverse**)**

library**(**dslabs**)**

data**(**"murders"**)**

murders **<-** mutate**(**murders, rate **=** total**/**population **\*** 10**^**5**)**

murders %>% select**(**state, region, rate**)** %>% filter**(**rate **<=** 0.71**)**

Algunos se preguntarán si se puede hacer todo de un solo plumazo, o de corrido, por supuesto que, si se puede, sin embargo, aquí separe en dos líneas de código para hacerlo más comprensible.

Esta línea de código es equivalente a las dos líneas de código anteriores.

- ¿Que está pasando aqui?

En general, la tubería envía el resultado del lado izquierdo de la tubería como primer argumento de la función en el lado derecho de la tubería. Aquí hay un ejemplo muy simple:

16 %>% sqrt**()** %>% log**(**base **=** 2**)**

Aquí el 16 lo pasamos a su raíz cuadrada que será 4 y este resultado al log base 2 que es igual a dos, lo podemos partir para mejor comprensión así:

**>** 16 %>% sqrt**()**

**[**1**]** 4

**>** 4 %>% log**(**base **=** 2**)**

**[**1**]** 2

**>**

Por lo tanto, al usar la tubería con dataframe y dplyr, ya no necesitamos especificar el primer argumento requerido ya que las funciones de dplyr que hemos descrito toman todos los datos como el primer argumento. En el código que escribimos:

murders %>% select**(**state, region, rate**)** %>% filter**(**rate **<=** 0.71**)**

Tenga en cuenta que la tubería funciona bien con funciones donde el primer argumento son los datos de entrada. Las funciones en paquetes tidyverse como dplyr tienen este formato y se pueden usar fácilmente con la tubería.

**Resumiendo Datos**

Una parte importante del análisis exploratorio de datos es resumir los datos.

El promedio y la desviación estándar son dos ejemplos de estadísticas de resumen ampliamente utilizadas.

A menudo se pueden obtener resúmenes más informativos dividiendo primero los datos en grupos.

En esta sección, cubrimos dos nuevas funciones dplyr que facilitan estos cálculos:

* summarize()
* group\_by().

Aprendemos a acceder a los valores resultantes utilizando la función pull().

Función summarize()

La función summarize() en dplyr proporciona una forma de calcular estadísticas de resumen con código intuitivo y legible. Comenzamos con un ejemplo simple basado en alturas. El conjunto de datos de alturas incluye las alturas y el sexo reportados por los estudiantes en una encuesta en clase.

Aquí el código:

library**(**dplyr**)**

library**(**dslabs**)**

data**(**heights**)**

El siguiente código calcula el promedio y la desviación estándar para las mujeres:

s **<-** heights %>%

filter**(**sex **==** "Female"**)** %>%

summarize**(**promedio **=** mean**(**height**)**, desviacion\_standard **=** sd**(**height**))**

s

Esto toma nuestra tabla de datos original como entrada, la filtra para mantener solo a las mujeres y luego produce una nueva tabla resumida con solo el promedio y la desviación estándar de las alturas. Podemos elegir los nombres de las columnas de la tabla resultante. Por ejemplo, arriba decidimos usar el promedio y la desviación estándar, pero podríamos haber usado otros nombres de la misma manera.

Debido a que la tabla resultante almacenada en s es un dataframe, podemos acceder a los componentes con el descriptor de acceso $:

s**$**promedio

s**$**desviacion\_standard

Al igual que con la mayoría de las otras funciones de dplyr, summarize() es consciente de los nombres de las variables y podemos usarlas directamente.

Entonces, dentro de la llamada a la función de resumen, escribimos media (altura), la función accede a la columna con el nombre "altura" y luego calcula el promedio del vector numérico resultante. Podemos calcular cualquier otro resumen que opera en vectores y devuelve un solo valor.

Por ejemplo, podemos agregar las alturas mediana, mínima y máxima de esta manera:

heights %>%

filter**(**sex **==** "Female"**)** %>%

summarize**(**median **=** median**(**height**)**, minimum **=** min**(**height**)**, maximum **=** max**(**height**))**

Nuestro resultado será algo parecido a esto:

**>** heights %>%

**+** filter**(**sex **==** "Female"**)** %>%

**+** summarize**(**median **=** median**(**height**)**, minimum **=** min**(**height**)**, maximum **=** max**(**height**))**

median minimum maximum

1 64.98031 51 79

**>**

Podemos obtener estos tres valores con una sola línea utilizando la función de cuantiles:

Por ejemplo, cuantil (x, c (0,0.5,1)) devuelve el mínimo (percentil 0), la mediana (percentil 50) y el máximo (percentil 100 ) del vector x.

Anteriormente, si intentamos usar una función como esta que devuelve dos o más valores dentro de summarize() obteníamos un error, hoy eso ha cambiado:

heights %>%

filter**(**sex **==** "Female"**)** %>%

summarize**(**range **=** quantile**(**height, c**(**0, 0.5, 1**)))**

Nuestro resultado será parecido a esto:

**>** heights %>%

**+** filter**(**sex **==** "Female"**)** %>%

**+** summarize**(**range **=** quantile**(**height, c**(**0, 0.5, 1**)))**

range

1 51.00000

2 64.98031

3 79.00000

**>**

Para otro ejemplo de cómo podemos usar la función summarize(), calculemos la tasa promedio de asesinatos en los Estados Unidos. Recuerde que nuestra tabla de datos incluye asesinatos totales y el tamaño de la población para cada estado y ya hemos utilizado dplyr para agregar una columna de índice de asesinatos:

data**(**murders**)**

murders **<-** murders %>% mutate**(**rate **=** total**/**population**\***10**^**5**)**

Recuerde que la tasa de asesinatos en los Estados Unidos no es el promedio de las tasas de asesinatos estatales y el siguiente código estaría errado:

summarize**(**murders, mean**(**rate**))**

Esto se debe a que, en el cálculo anterior, los estados pequeños tienen el mismo peso que los grandes. La tasa de homicidios en los Estados Unidos es el número total de asesinatos en los Estados Unidos dividido por la población total de los Estados Unidos, entonces el cálculo correcto es:

us\_murder\_rate **<-** murders %>%

summarize**(**rate **=** sum**(**total**)** **/** sum**(**population**)** **\*** 10**^**5**)**

us\_murder\_rate

Este cálculo cuenta estados más grandes proporcionalmente a su tamaño, lo que da como resultado un valor mayor.

Función pull()

El objeto us\_murder\_rate definido anteriormente representa solo un número. Sin embargo, lo estamos almacenando en un dataframe:

class**(**us\_murder\_rate**)**

**>** class**(**us\_murder\_rate**)**

**[**1**]** "data.frame"

Dado que, como la mayoría de las funciones de dplyr, summarize siempre devolverá un dataframe.

Esto podría ser problemático si queremos usar este resultado con funciones que requieren un valor numérico. Aquí mostramos un truco útil para acceder a los valores almacenados en los datos cuando se usan tuberías:

Cuando un objeto de datos se canaliza a ese objeto y se puede acceder a sus columnas usando la función pull().

Para entender lo que queremos decir, eche un vistazo a esta línea de código:

us\_murder\_rate %>% pull**(**rate**)**

Esto devuelve el valor en la columna de tasa de us\_murder\_rate, por lo que es equivalente a us\_murder\_rate$rate.

Para obtener un número de la tabla de datos original con una línea de código, podemos escribir:

us\_murder\_rate **<-** murders %>%

summarize**(**rate **=** sum**(**total**)** **/** sum**(**population**)** **\*** 10**^**5**)** %>%

pull**(**rate**)**

us\_murder\_rate

Que ahora es numérico:

**>** class**(**us\_murder\_rate**)**

**[**1**]** "numeric"

**>**

Función group\_by()

Una operación común en la exploración de datos es dividir primero los datos en grupos y luego calcular resúmenes para cada grupo. Por ejemplo, es posible que queramos calcular la desviación promedio y estándar para las alturas de hombres y mujeres por separado. La función group\_by nos ayuda a hacer esto.

Si escribimos esto:

heights %>% group\_by**(**sex**)** %>% head**()**

Obtendremos esto:

**>** heights %>% group\_by**(**sex**)** %>% head**()**

# A tibble: 6 × 2

# Groups: sex [2]

sex height

**<**fct**>** **<**dbl**>**

1 Male 75

2 Male 70

3 Male 68

4 Male 74

5 Male 61

6 Female 65

El resultado no se ve muy diferente de las alturas, excepto que vemos Grupos: sexo cuando imprimimos el objeto. Aunque no es inmediatamente obvio por su apariencia, ahora es un dataframe especial llamado dataframe agrupado y las funciones de dplyr, en particular, se comportarán de manera diferente cuando actúen sobre este objeto.

Conceptualmente, puede pensar en esta tabla como muchas tablas, con las mismas columnas, pero no necesariamente el mismo número de filas, apiladas juntas en un objeto. Cuando resumimos los datos después de la agrupación, esto es lo que sucede:

heights %>%

group\_by**(**sex**)** %>%

summarize**(**average **=** mean**(**height**)**, standard\_deviation **=** sd**(**height**))**

**>** heights %>%

**+** group\_by**(**sex**)** %>%

**+** summarize**(**average **=** mean**(**height**)**, standard\_deviation **=** sd**(**height**))**

# A tibble: 2 × 3

sex average standard\_deviation

**<**fct**>** **<**dbl**>** **<**dbl**>**

1 Female 64.9 3.76

2 Male 69.3 3.61

La función summarize() aplica el resumen a cada grupo por separado.

Para otro ejemplo, calculemos la tasa media de asesinatos en las cuatro regiones del país:

murders %>%

group\_by**(**region**)** %>%

summarize**(**median\_rate **=** median**(**rate**))**

Pueden explorar ustedes mismos el resultado, que está vez no lo he colocado por acá.

**Ordenando Data Frames**

Al examinar un conjunto de datos, a menudo es conveniente ordenar la tabla por las diferentes columnas. Sabemos acerca de la función order() y sort(), pero para ordenar tablas enteras, la función de dplyr arrange() es útil. Por ejemplo, aquí ordenamos los estados por tamaño de población:

Función arrange()

library**(**tidyverse**)**

library**(**dslabs**)**

data**(**murders**)**

murders %>%

arrange**(**population**)** %>%

head**()**

Este es el resultado:

**>** murders %>%

**+** arrange**(**population**)** %>%

**+** head**()**

state abb region population total

1 Wyoming WY West 563626 5

2 District of Columbia DC South 601723 99

3 Vermont VT Northeast 625741 2

4 North Dakota ND North Central 672591 4

5 Alaska AK West 710231 19

6 South Dakota SD North Central 814180 8

Como vemos lo ha ordenado por población en orden ascendente.

Con la funcion arrange() podemos decidir por qué columna ordenar. Para ver los estados por población, de menor a mayor, organizamos por rate:

murders **<-** mutate**(**murders, rate **=** total**/**population **\*** 10**^**5**)**

murders %>%

arrange**(**rate**)** %>%

head**()**

Tenga en cuenta que el comportamiento predeterminado es ordenar en orden ascendente. En dplyr, la función desc() transforma un vector para que esté en orden descendente. Para ordenar la tabla en orden descendente, podemos escribir:

murders %>% arrange**(**desc**(**rate**))** %>% head**()**

En caso les de error es porque posiblemente llamamos a la data murders nuevamente, podemos incluir la tasa nuevamente así:

murders **<-** mutate**(**murders, rate **=** total**/**population **\*** 10**^**5**)**