Introducción y Conceptos Básicos R

Consideraciones de R

Sintaxis básica

Las variables en R funcionan como elemntos para almacenar valores, en lo posible reutilizables

las funciones son elementos, también reutilizables, pero adicional pueden establecerse con características propias en su interior para realizar cualquier tipo de operación o proceso. Se encapsulan en un mismo lugar para que escribir código sea más eficiente.

```
#Números enteros
entero <- 3
# Validar tipo
print(paste0('Tipo de variable entero:', class(entero)))
```

```
[1] "Tipo de variable entero:numeric"
```

```
# Números flotantes (decimales=)
flotante <- 3.6
print(paste0('Tipo de variable flotante:', class(flotante)))
```

```
[1] "Tipo de variable flotante:numeric"
```

```
# Texto
texto <- 'Hola mundo'
print(paste0('Tipo de variable texto:', class(texto)))
```

```
[1] "Tipo de variable texto:character"
```

```r
# Lógicos
print(7 == 8)
```

```
[1] FALSE
```

```r
print(8 == 8)
```

```
[1] TRUE
```

```r
#Crear vectores
mixto <- c(2, 4, 'manzanas', 'bananas')
print('Contenido de mixto')
```

```
[1] "Contenido de mixto"
```

```r
print(mixto)
```

```
[1] "2"        "4"        "manzanas" "bananas"
```

```r
print('Segundo elemento del vector mixto:')
```

```
[1] "Segundo elemento del vector mixto:"
```

```r
print(mixto[2])
```

```
[1] "4"
```

```r
# Creas matrices
matriz_random <- matrix( mixto, nrow = 2, ncol = 6)
print('Contenido de mmatriz_random')
```

```
[1] "Contenido de mmatriz_random"
```

```r
print(matriz_random)
```

```
     [,1] [,2]       [,3] [,4]       [,5] [,6]
[1,] "2"  "manzanas" "2"  "manzanas" "2"  "manzanas"
[2,] "4"  "bananas"  "4"  "bananas"  "4"  "bananas"
```

```r
# Funciones + for + condicionales
valores <- c(20, 18, 25, 40, 49, 68)

# Una función se crea a través de = y asignación de parámetros
es_par <- function(valores) {
    # Ciclo for
    for (i in 1:length(valores)) {
        # Condicional
        if(valores[i]%%2 == 0){
            print(paste0('El valor ', valores[i], ' es par'))
        } else {
            print(paste0('El valor ', valores[i], ' es inparpar'))
        }
    }
}

# Llamar y usar función
es_par(valores)
```

```
[1] "El valor 20 es par"
[1] "El valor 18 es par"
[1] "El valor 25 es inparpar"
[1] "El valor 40 es par"
[1] "El valor 49 es inparpar"
[1] "El valor 68 es par"
```

Base vs tidyverse

En R, existen dos enfoques principales para la manipulación y análisis de datos: Base R y el Tidyverse. Ambos son poderosos, pero difieren significativamente en su sintaxis, filosofía y estilo de programación. Comprender estas diferencias es clave para escribir código R eficiente y legible.

¿Qué es Base R?

Base R se refiere a las funciones y paquetes que vienen preinstalados con R. Es el conjunto fundamental de herramientas que ha existido desde los inicios de R. Sus funciones a menudo operan directamente sobre vectores, matrices y data frames utilizando corchetes ([]) y el operador $ para acceder a elementos o columnas.

¿Qué es Tidyverse?

El Tidyverse es una colección de paquetes de R (como dplyr, ggplot2, tidyr, readr, entre otros) que comparten una filosofía de diseño común: hacer que la manipulación de datos sea más intuitiva, consistente y legible. Se enfoca en el concepto de "datos ordenados" (tidy data) y utiliza el operador `%>% (pipe) o |> (en R 4.1+)` para encadenar múltiples operaciones de forma secuencial.

Ejemplos Prácticos

Para ilustrar las diferencias, usaremos un pequeño data.frame de ejemplo:

```
# Crear un data frame de ejemplo
datos <- data.frame(
  ID = 1:5,
  Producto = c("Manzana", "Banana", "Naranja", "Manzana", "Kiwi"),
  Precio = c(1.5, 0.75, 1.2, 1.5, 2.0),
  Cantidad = c(10, 20, 15, 5, 8)
)
print(datos)
```

```
  ID Producto Precio Cantidad
1  1  Manzana   1.50       10
2  2   Banana   0.75       20
3  3  Naranja   1.20       15
4  4  Manzana   1.50        5
5  5     Kiwi   2.00        8
```

1. Filtrar Filas y Seleccionar Columnas

Objetivo: Obtener el ID y Producto de los ítems con Cantidad mayor a 10.

Con Base R:

```
# Filtrar filas y seleccionar columnas en Base R
resultados_baseR <- datos[datos$Cantidad > 10, c("ID", "Producto")]
print(resultados_baseR)
```

```
  ID Producto
2  2   Banana
3  3  Naranja
```

Con Tidyverse (dplyr):

```
# Cargar la librería dplyr
library(dplyr)

# Filtrar filas y seleccionar columnas con Tidyverse (dplyr)
resultados_tidyverse <- datos %>%
  filter(Cantidad > 10) %>%
  select(ID, Producto)
print(resultados_tidyverse)
```

```
  ID Producto
1  2   Banana
2  3  Naranja
```

2. Agrupar y Resumir Datos

Objetivo: Calcular el Precio total por Producto.

Con Base R:

```
# Agrupar y resumir en Base R
precios_totales_baseR <- aggregate(Precio ~ Producto, data = datos, FUN = sum)
print(precios_totales_baseR)
```

```
  Producto Precio
1   Banana   0.75
2     Kiwi   2.00
3  Manzana   3.00
4  Naranja   1.20
```

Explicación: group_by() define los grupos para la operación. summarise() luego aplica la función de agregación (sum) a la columna Precio dentro de cada grupo y crea una nueva columna PrecioTotal. Nuevamente, el flujo es claro y secuencial.

Comparación: Pros y Contras

Característica	Base R	Tidyverse
Sintaxis	Flexible, a menudo concisa pero puede ser menos intuitiva para cadenas complejas. Usa `[]`, `$`.	Consistente, verbos claros (`filter`, `select`, `mutate`). Usa `%>%` o `|>`. Muy legible.
Filosofía	Operaciones directas sobre estructuras de datos.	Enfoque en "tidy data" (datos ordenados), facilitando la manipulación secuencial.
Curva de Aprendizaje	Puede ser empinada inicialmente para manipulación de datos, pero fundamental para entender R.	Generalmente más suave para tareas de manipulación de datos debido a su consistencia.
Legibilidad	Puede volverse difícil de leer con muchas anidaciones o llamadas a funciones.	Altamente legible debido al encadenamiento de operaciones con el pipe.
Dependencias	No requiere instalar paquetes adicionales.	Requiere instalar y cargar paquetes específicos (ej. `dplyr`).
Rendimiento	A menudo muy eficiente para operaciones vectorizadas.	Generalmente muy bueno, con implementaciones optimizadas para muchas funciones comunes.
Flexibilidad	Muy alta, ya que permite un control granular sobre todos los aspectos.	Alta, pero guiada por una gramática de datos específica.

¿Cuándo usar cada uno?

Usa Base R cuando:

- Estás realizando operaciones muy simples y directas (ej. calcular la media de un vector).
- Quieres un control muy granular y entender profundamente cómo R maneja los objetos y las operaciones.
- Estás trabajando en entornos donde la instalación de paquetes adicionales es restringida o indeseable.
- Necesitas escribir código para funciones muy específicas de R que aún no tienen un "equivalente tidy" directo o eficiente.

Usa Tidyverse cuando:

- Estás realizando tareas de manipulación de datos complejas o que involucran múltiples pasos.
- La legibilidad y la reproducibilidad del código son prioritarias.
- Estás trabajando con otros analistas de datos o científicos de datos que usan el Tidyverse (lo cual es muy común).
- Necesitas hacer visualizaciones de datos avanzadas (ggplot2).

- Buscas un flujo de trabajo consistente y eficiente para la limpieza, transformación y análisis de datos.

En resumen: Para la mayoría de las tareas de ciencia de datos y análisis de datos en R, el Tidyverse se ha convertido en el estándar de facto debido a su legibilidad, consistencia y eficiencia para la manipulación de datos. Sin embargo, comprender los fundamentos de Base R es esencial para cualquier usuario serio de R, ya que muchas funciones del Tidyverse se construyen sobre o interactúan con conceptos de Base R. La mejor práctica a menudo implica el uso de ambos, aprovechando las fortalezas de cada uno según la necesidad.

Coerciones

En R, los datos pueden ser coercionados, es decir, forzados, para transformarlos de un tipo a otro.

La coerción es muy importante. Cuando pedimos a R ejecutar una operación, intentará coercionar de manera implícita, sin avisarnos, los datos de su tipo original al tipo correcto que permita realizarla. Habrá ocasiones en las que R tenga éxito y la operación ocurra sin problemas, y otras en las que falle y obtengamos un error.

Es decir si exiten vectores con contenido númerico y luego se le incluye un *string* este se convertirá en *character*

```
vector <- c(10, 20, 30)

class(vector)
```

```
[1] "numeric"
```

Conversión:

```
vector2 <- c(10, 20, 30, 'b')

class(vector2)
```

```
[1] "character"
```

Incluso, es posible convertir booleano en números a través de *as.numeric()*

```
bool <- c(2, as.numeric(FALSE), 5)

bool
```

```
[1] 2 0 5
```

Uso ggplot2

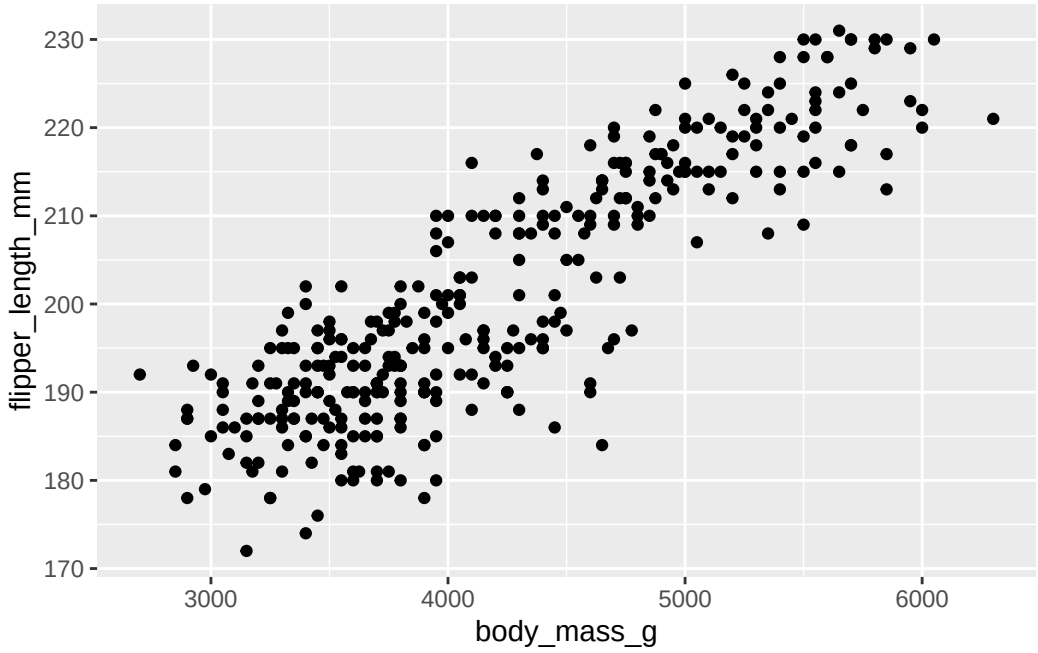
ggplot2 es una librería para visualización de datos, una de las más populares por su gran variedad de tipos de gráficos.

Su elaboración consiste en capas

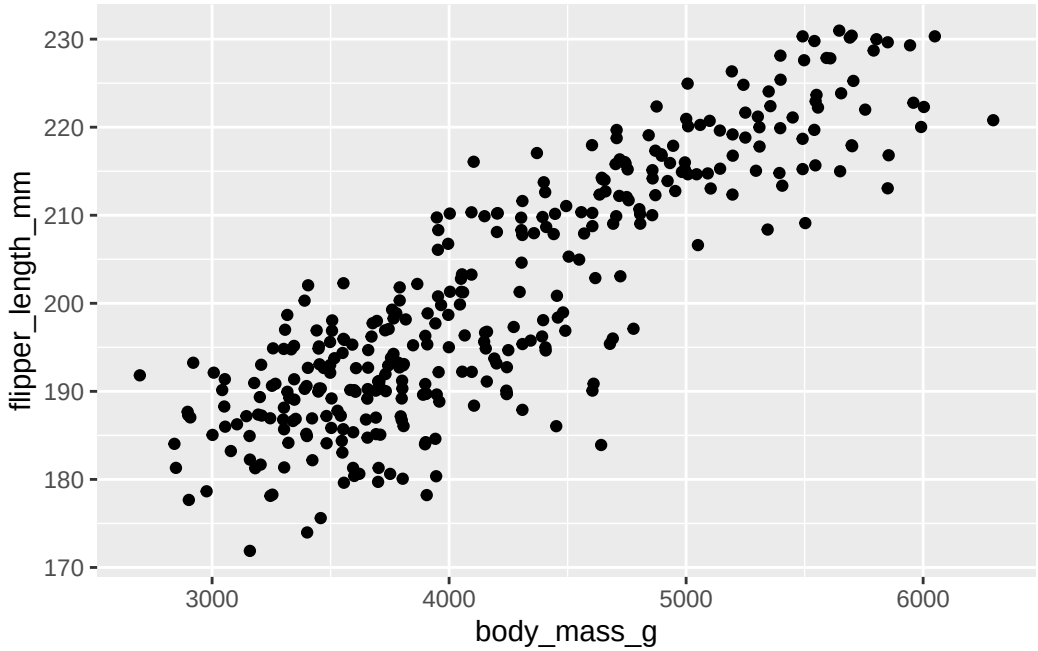
```
library(ggplot2)
library(palmerpenguins)

ch <- ggplot(data = penguins,
                aes(x = flipper_length_mm, y = body_mass_g)) +
  coord_flip()

ch + geom_point()    # Puntos básicos
```




```
ch + geom_jitter()   # Dispersión de puntos
```



Modificación del contenido y clasificación de las barras

```
plot <- ggplot(SICOP,
    aes(x = clasificacion_objeto, fill = tipo_modalidad))

plot +
    geom_bar(position = 'fill')

plot +
    geom_bar(position = 'stack')
```

Flujo de trabajo en R (Básico)

Desde R4DataScience se describe que el flujo de trabajo en R se basa en:

- Conceptos básicos de codificación (variables, vectores, operaciones)
- Comentarios; Con una claridad de "*Usa comentarios para explicar el porqué de tu código, no el cómo ni el qué . El qué y el cómo de tu código siempre se pueden entender, aunque sea tedioso, leyéndolo con atención.*"
- Convención de nombre de objeto

```
i_use_snake_case
otherPeopleUseCamelCase
some.people.use.periods
And_aFew.People_RENOUNCEconvention
```

- Llamado de funciones según los argumentos internos.

Flujo de trabajo en R (Estilo)

Desde R4DataScience, se recomienda para el uso desde RStudio la librería de "styler" para la estilización de código, he incluso la buena práctica de implementación de convenciones.

A nivel de estilo, (que debería ser regla en varios lenguajes), mantener espacios durante la escritura de código "*Coloque espacios a ambos lados de los operadores matemáticos excepto ^(es decir +, -, ==, <, …)*" y "*No deje espacios dentro ni fuera de los paréntesis en las llamadas a funciones regulares*"

La implementación de pipe |> para versiones más recientes de R y la agregación de comentarios de seccionamiento cuando el código se hace extenso y se hace necesario segmentar sesiones para brindar claridad durante su lectura.

```
# Load data ------------------------------------

# Plot data ------------------------------------
```

Flujo de trabajo en R (script y proyecto)

Al crearse antes de la existencia de Positron, este flujo de trabajo se centra en implementar el módulo de script de RStudio para el momento de crear código, incluso que permitirá guardar los archivos para posterior modificación e incluso uso desde el llamado en la terminal.

E incluso, como elemento extra, se habla de proyectos, que consiste en crear un Workspace de trabajo desde la configuración de RStudio para gestionar multiples análsis en "scripts" separados desde una única ubicación de la carpeta.