Manipulación de datos

Clase 6

Joaquin Cavieres G.

Estudiante doctorado

j.cavieres.g@gmail.com

18 de diciembre de 2019

Notas breves

- La mayoría de las funciones vistas en clases sirven como herramientas en los procesos de análisis.
- Existen test lógicos que permiten eficiencia en la manipulación de los datos.
- Las funciones `any()`, `all()` o `which()` permiten realizar estos test lógicos.

Test lógicos

Creamos un vector **númerico**

```
num <- c(12, 9, 8, 14, 7, 16, 3, 2, 9)
```

- ¿Existe cualquier número mayor a 10?

```
> any(num > 10)
[1] TRUE
```

- ¿Todos los números son mayores a 10?

```
> all(num > 10)
[1] FALSE
```

- ¿Que números son menores que 10?

```
> which(num < 10)
[1] 1 4 6
```

Observaciones duplicadas

- Uso de formas lógicas ayuda en forma eficiente en la manipulación de datos.

```
data <- matrix(rep(c(1,1,2,3), each=3), ncol=3, byrow=T)

# Repetimos el 1,2 y 3 cada 3 veces, en 3 columnas y que
la matriz se llene por filas
```

- `duplicated()` retorna un vector lógico indicando que elementos del vector o matriz es duplicado (no es único).

```
> duplicated(data)
[1] FALSE TRUE FALSE FALSE
```

Observaciones duplicadas

- `unique()` remueve los elementos duplicados.

```
> unique(data)
```

```
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    2    2    2
[3,]    3    3    3
```

Creación de nuevos factores

- Algunas observaciones que provienen de una variable continua, tal vez puede ser mejor tratarla en grupos o categoricamente como un factor, por ejemplo:

  La función `cut()` ayuda a crear grupos en forma de factor.

  ```
  cut(x, breaks=3, labels=NULL, rigth=T)
  ```

Donde:

- x: variable observada.
- breaks: Número de grupos a crear.
- labels: Niveles del factor.
- rigth: Control sobre los limites de los valores altos y bajos (incluidos los del breaks).

Creación de nuevos factores

```
edad <- c(24, 20, 35, 40, 70, 35)
```

- Usar 3 breaks igualmente espaciados (intervalos)

```
> cut(edad, breaks=3)

[1] (19.9,36.7] (19.9,36.7] (19.9,36.7]
(36.7,53.3] (53.3,70] (19.9,36.7]
(30,40] (40,50] (30,40]

[1] Levels: (19.9,36.7] (36.7,53.3] (53.3,70]
```

Creación de nuevos factores

```
edad <- c(24, 20, 35, 40, 70, 35)
```

- Defino los breaks con aplicación de niveles

```
> cut(edad, breaks=c(0,18,65,Inf),
labels=c(‘‘Joven’’, ‘‘Adulto’’, ‘‘Viejo’’))
```

```
[1] Adulto Adulto Adulto Adulto Viejo Adulto
```

```
[1] Levels: Joven Adulto Viejo
```

Creación de nuevos factores

Ahora, si queremos cambiar la paleta de colores, especificamos `colors()` en la venatana de comandos, de ahí podemos elegir los que queramos.

```
> colors()
```

Por ejemplo, nueva paleta de colores:

```
points.colors <- c(‘‘red’’, ‘‘orange’’, ‘‘green’’,
‘‘blue’’, ‘‘magenta’’)

plot(altura peso, data=relacion,
xlab= ‘‘Peso (grs)’’, ylab= ‘‘Altura (cm)’’,
col=points.colors)
```

VER CÓDIGO PARA MAS EJEMPLOS....

Sub-conjuntos de datos

Vamos a construir un sub-conjunto de datos pero que contenga NA.

```
>x <- data.frame(a=c(5,9,12,15,17,11),
b=c(8,NA,12,10,NA,15))
```

```
>x[x$b > 10,]
```

	a	b
NA	NA	NA
3	12	12
NA.1	NA	NA
6	11	15

Función `subset()`

La función `subset()` entrega un objeto que cumple las condiciones realizadas por el analista, además de generar un manejo sobre los NA.

```
subset(objeto, expresión lógica, variable seleccionada)
```

Ejemplo:

```
> subset(x,b > 10)
```

	a	b
3	12	12
6	11	15

Función `subset()`

También se pueden seleccionar columnas especificas.

```
> subset(x, b > 10, b)
```

```
   b
3 12
6 15
```

Función `apply()`

La función `apply()` es muy flexible y ayuda para realizar diversos cálculos. Es mas rápida en terminos de costos computacionales que un ciclo `for`, por ejemplo.

```
apply(x, MARGIN, FUN,...
```

- x= matriz.
- MARGIN= 1=rows (filas), 2=columns (columnas).
- FUN= una función de R.
- .... = Argumentos adicionales

```
> m<- matrix(1:4, ncol=2)
```

```
> apply(m, 2, mean)
```

Significa: aplicar a las columnas (**2**) de la matriz m (**m**) la media (**mean**).

```
[1] 1.5 3.5
```

Función `apply()`

- La función `apply()` permite reducir los datos en valores que interesan al investigador en forma eficiente.
- Creamos una matriz con variables aleatorias, donde cada columna es un sujeto y las filas son observaciones de ese sujeto.

```
sujetos <- matrix(rnorm(50, mean=1:5), nrow=10, byrow=T)
```

esto significa que simulamos 10 variables observadas para 5 sujetos distintos.

```
       [,1]      [,2]     [,3]     [,4]     [,5]
 [1,] 0.8060285 1.614561 3.049718 3.553686 3.452866
 [2,] 1.6393570 1.201174 3.237046 4.427272 3.621400
 [3,] 1.5078694 3.033200 2.200431 4.796346 3.723389
 [4,] 0.6451399 1.521813 5.619839 3.846313 6.148770
 .... ..... .... ..... ..... .....
```

Función `apply()`

- Cálculo de las medias `mean` dependiendo de la organización de los datos.

```
> apply(sujetos, 2, mean)
```

esto significa: aplicar la media en las columnas (2) de la matriz "sujetos".

```
[1]  1.004257  1.791198  3.379779  3.973856  4.885046
```

```
> apply(sujetos, 1, mean)
```

esto significa: aplicar la media en las filas (1) de la matriz "sujetos".

```
[1]  3.777220  2.443275  2.696850  3.596633  3.560976...
```

Función `apply()`

- `sapply()` y `lapply()` son funciones que se aplican sólo en listas.
- La función `tapply()` se puede aplicar sobre vectores por factores categoricos.
  - `tapply(x, indice, fun)`

```
> observaciones <- data.frame(especie=rep(1:5,5),
talla=rnorm(25, mean=1:5))
```

```
> tapply(observaciones$talla, observaciones$especie,
mean)
```

esto quiere decir que: a la talla (datos = x) de las categorías observadas (especies) le aplicamos la media (son 5 categorías).

Ordenando elementos

- Generalmente necesitamos ordenar algunas variables en orden correlativo o por un "índice" que lo identifique. Esta orden se puede hacer con las funciones `sort()` y `order()`. Por ejemplo:
- Creamos un vector

```
> y <- sample(1:10).
[1] 1 3 9 4 7 2 10 5 6 8
> sort(y)
[1] 1 2 3 4 5 6 7 8 9 10
> order(y)
[1] 1 3 6 8 10 7 5 4 2 9
```

GRACIASSSSSS!!!