**Semana 3**

**12. Funciones \*apply**

En esta lección aprenderemos a usar la familia de funciones *apply*, que es una familia vectorizada que nos ayudaran a aplicar algún tipo de función sobre los vectores o sobre los objetos para las que están diseñadas.

Las funciones vectorizadas son grupos de funciones especiales en donde implícitamente se piensa que la operación se hace en paralelo, por lo que tnemos una ganancia en el proceso de la función. Este tipo de funciones te permiten cruzar los datos de tal manera que evites el uso explícito de bucles, como por ejemplo **for, while** y **repeat**. Las funciones apply actúan sobre algún tipo de entrada como listas, matrices o vectores, aplicando la función como un argumento opcional.

La función más básica dentro de la familia *apply* es precisamente la función *apply()*. Ésta opera sobre rebanadas de los datos, es decir, sobre subconjuntos de datos que ya tenemos. Hay 3 tipos de funciones que pueden entrar como argumento:

* Funciones de agregación: Aquellas que operan sobre un conjunto de datos y regresan un valor numérico único, como la suma (**sum**) y la media (**mean**).
* Función de transformación o extracción de subconjuntos: Para acceder a índices de un vector o matriz.
* Funciones vectorizadas: Como
  + Listas, vectores, matrices y arreglos.

Las funciones apply pueden expresar complejidad muy grande en solo unas cuantas líneas de código y operan de manera paralela cuando es posible y son más fáciles de leer.

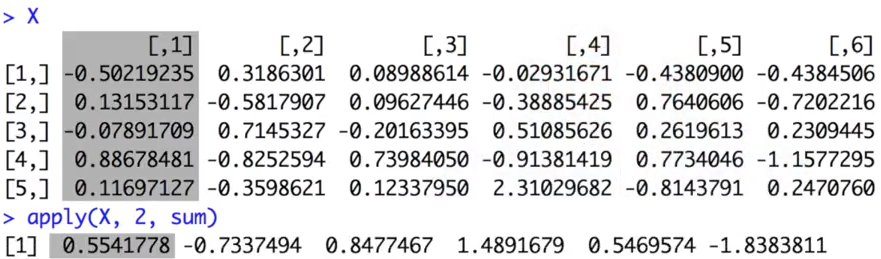
Las 3 funciones más importantes dentro de la familia son *apply(), sapply() y lapply()*, entre otras, que son variantes de las otras.

1. **Función *apply()***

Opera sobre arreglos como las matrices. Los 3 argumentos más importantes dentro de la función son:

* **X:** Vector o un arreglo donde se opera.
* **MARGIN:** Variable que nos va a definir si operamos sobre renglones y columnas. Un valor de MARGIN de 1 indica que se aplica sobre filas, y un valor de 2 que actúa sobre columnas. El parámetro **c(1,2)** indica que se actúa tanto sobre filas como en las columnas.
* **FUN:** La función

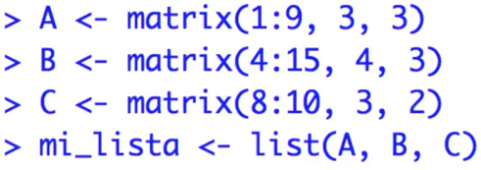
La función *set.seed()* lo que hace es que si se establece un valor inicial o semilla, todos los números aleatorios producidos después de ésta serán los mismos en todas las ordenadores, y si no se hace estaríamos generando todos valores distintos en los ordenadores, ya que las semillas con las que se generan los números aleatorios serían distintas para cada ordenador. Se combina con la función *rnorm()*, que genera números aleatorios de una normal 0,1. Todos estos números nos salen iguales a todos, pudiendo reproducir los ejemplos. Una vez que construyamos la matriz, podemos encontrar la suma de columna por columna si se usa la función *apply()*, indicando primero la matriz, luego con MARGIN la operación sobra la que la queremos realizar, y con el tercer parámetro la operación.



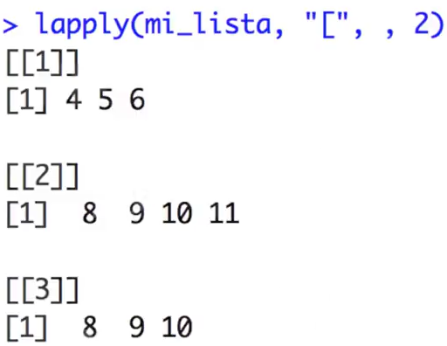
Este gráfico es solo para recordar que al usar 2 operamos sobre las columnas. Si quisiéramos hacer eso paso por paso habría que hacerlo con un ciclo **for**, lo que sería muy complicado.

1. **Funciones *sapply()* y *lapply()***

La función *lapply()* sirve para aplicar una función a cada uno de los elementos de una lista, devolviéndonos asimismo una lista. La sintaxis de la función *lapply()* es similiar a *apply()*, solo que en lugar del parámetro MARGIN, como tercer parámetro recibe los parámetros de la función que se usan como argumento. Otra diferencia es que los objetos sobre los que pueden actuar son diferentes, como dataframes, listas o vectores, y el objeto de salida que devuelve es siempre una lista. En el ejemplo vamos a construir 3 matrices:

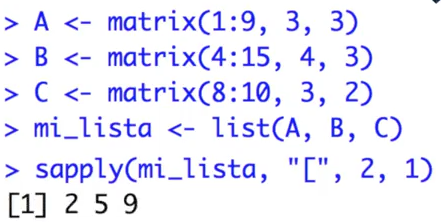
****

Para extraer el segundo valor de cada uno, usamos la función *lapply()*, con el primer parámetro el origen sobre el que se va a hacer la operación, el segundo parámetro la operación que queremos realizar (el operador “[“ (corchete), que significa extracción) y un tercer parámetro que son los renglones y las columnas (como queremos todos los renglones, el primero va vacío).



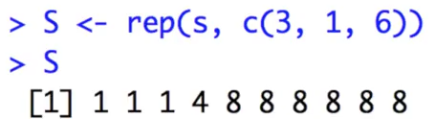
Si queremos extraer un elemento de cada una de las matrices, lo que hay que hacer es no dejar parámetros vacíos en la parte del corchete.

La función *sapply()* intenta simplificar la salida de *lapply()*, es decir, que nos regresaba una lista mientras que ésta nos regresa un vector. Es decir, *sapply()* es *lapply()* pero con ciertos parámetros por omisión. Como ejemplo, vamos a comparar la salida de *lapply()* con *sapply()*. Si usamos la función *sapply()* con los mismos parámetros de *lapply()* nos devuelve un vector, es decir, nos simplifica la salida.

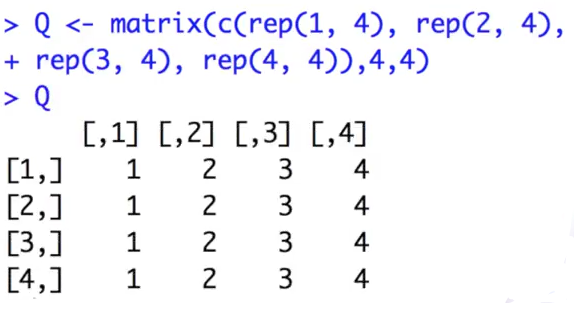


1. **Las funciones *mapply()* y *rep***

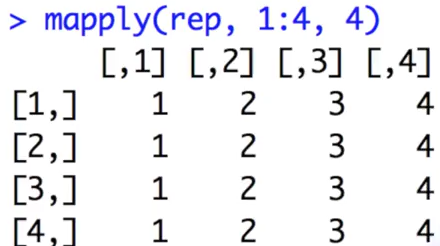
La función *mapply()* viene de “*multi-varied apply”* y su propósito es vectorizar argumentos para una función que normalmente no acepta vectores como argumentos. La función *rep()* es usada en conjunto con las funciones de esta familia, y genera un vector con las repeticiones de un vector que le pasemos, es decir, recibe 2 vectores, el primero que se repite y un segundo que indica cuantas veces se repite el primero. Si usamos la función *rep()* sobre un vector *s*  y añadimos un segundo término con los elementos 3,1,6, lo que indica es que el primer término se repetirá 3 veces, el segundo 1 y el tercero 6 veces.



Podemos construir una matriz de la siguiente forma:



Una alternativa es usar la función *mapply()* que como primer parámetro es la función *rep()*, el segundo son los parámetros, y el tercer elemento es cuántas veces queremos repetir cada elemento del parámetro anterior.



Si ponemos un vector en el tercer parámetro dentro de la función *mapply()*, lo que va a hacer es indicando 1 por 1 los elementos de los vectores (como el valor 4:1, va restando uno de los valores a la matriz, quedando filas de 4, filas de 3, filas de 2 y una fila de 1 con los valores 1, 2, 3 y 4 respectivamente).

* **Tarea RStudio**

Si dentro de la variable *apply()* incluimos el parámetro *FUN=min*, se nos devuelve el valor más pequeño de la variable que tuviéramos indicada dentro de la función. Se puede combinar con MARGIN para indicar si queremos el valor más pequeño de las filas o las columnas.

Si deseamos aplicar una función a una lista o un vector, tenemos que tener en cuenta que *apply()* solo opera sobre arreglos, por lo que para aplicar una función a cada elemento de un vector o una lista y regresar una lista, puedes usar la función *lapply()*. Para crear una lista, hay que usar la función *list()*. Para ver los argumentos de una variable, usamos la función *formals()*.

La función *toupper()* regresa la cadena que reciba como entrada en mayúsculas. *Lapply* siempre regresa una lista (lista + apply = lapply).

Las funciones de la familia *read.\*()* regresan un data frame. Si introducimos el nombre de la variable en la que está almacenado un archivo .csv y le añadimos el símbolo $ seguido del nombre de una de las columnas, nos devuelve todos los elementos de la misma. Una manera de poder ver todos los niveles aún cuando sean muchos es usando la función *unique()*. En el ejemplo sería *unique(asa\_datos$Descripcion)*. Con la función *sum()* puedes sumar todos los elementos que recibe como argumentos.

La función *sapply()* te permite simplificar este proceso funcionando como *lapply()*, pero intenta simplificar la salida de la estructura de datos más elemental que sea posible. De ahí su nombre: simple + apply = sapply.