title

2.Paquete here, listas, familia Which y operadores lÃ³gicos

author

29 de octubre de 2018

Contents

Intro

He pensado que a partir de ahora, como un metodo para ir familiarizandonos con los datos con los que vamos a trabajar, la guía se va a desarrollar empleando los datos obtenidos por los sensores que nosotros hemos instalado.

Cuando quedamos el primer día, ya vimos que el método que se ha empleado para obtener los datos es una rayada, basada en *web scraping*[https://es.wikipedia.org/wiki/Web_scraping]. ¡¡Pero!!, no vamos ha entrar ahí. Vamos a suponer que ese código funciona y vamos a pasar a trabajar directamente con los datos.

De hecho he creado una carpeta llamada **data**(dentro del directorio de trabajo que creamos en github) donde están guardados estos datos. *PORFAVOR NO MODIFIQUEN ESTA CARPETA*,**SOLAMENTE IMPORTAR**.

Es importante remarcar que para que poder seguir esta guía correctamente debemos estar trabajando con el proyecto de App_anemometros abierto. Tal y como hicimos el primer dia. Creando un proyecto (version control) linkeado directamente con el repositorio de github, habiendo instalado Git previamente e iniciado sesión. SI AUN LES CUESTA REALIZAR ESTE PROCESO DE MANERA SENCILLA DECIDMELO Y HAGO UNA GUIA EXPLICATIVA TAMBIÉN, NO ME CUESTA NADA Y VA A AGILIZAR MUCHO LA DINÁMICA DE TRABAJO.

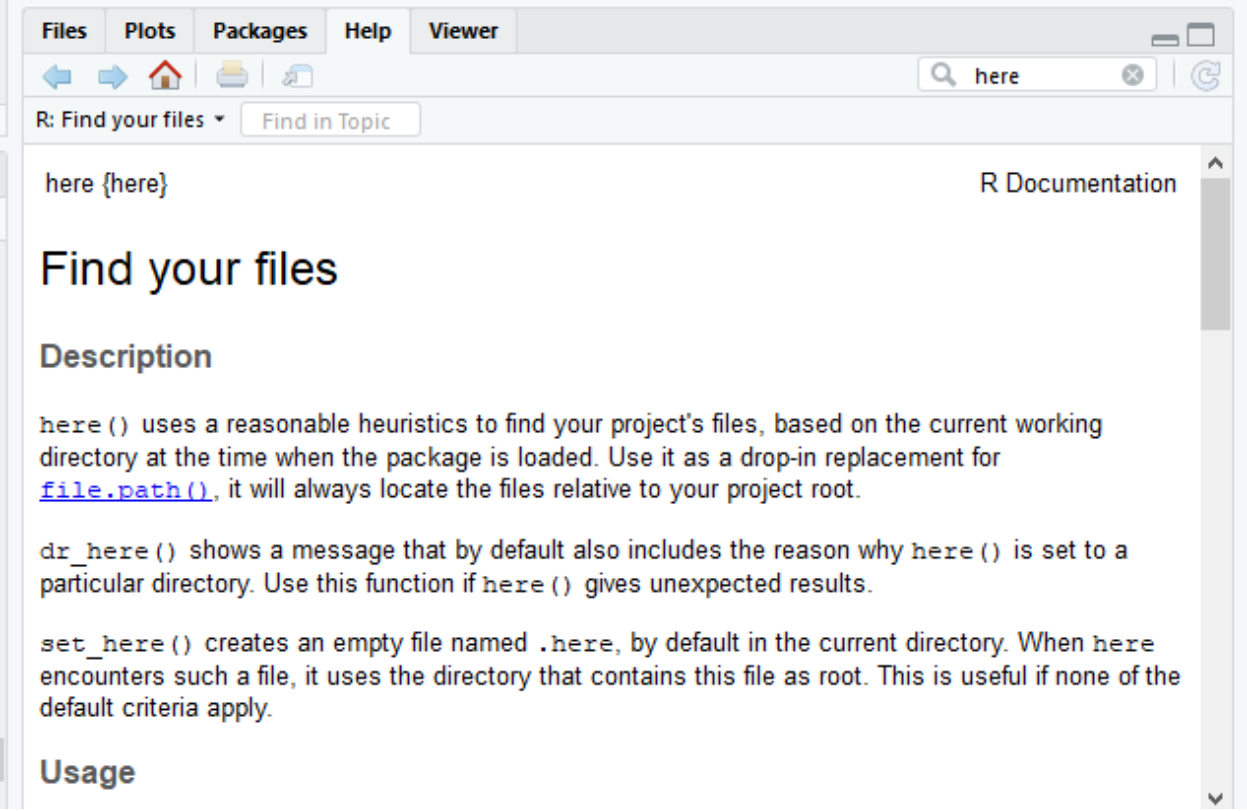
Paquete here()

Sería importante que entendieran el funcionamiento y la aplicación del paquete **here**. Agiliza mucho el trabajo en carpetas. Muchas veces, para importar o guardar datos hay que poner el directorio donde queremos que Rstudio busque los archivos. El comando here() nos permite realizar esto de manera rápida por que al usarlo directamente referencia al directorio de trabajo principal. Con la ventaja de que cuando se ejectura en diferentes ordenadores y o se ha cambiado el directorio de trabajo cumple su función perfectamente sin necesidad de cambiar a mano el directorio de trabajo.

```
here()
```

```
## [1] "C:/Users/Oscar/Documents/App_Anemometros"
```

Si lo ejecutan uds, verán que el directorio es diferente. Esto me sirve en parte para incitaros tambien a indagar en la potente herramienta que es Rstudio a nivel de información, ejemplos y foros. Podemos buscar información del paquete en la ayuda del programa, aun estando off-line.



Además el caracter open-source de R, fomenta que tenga una comunidad muy activa. Generando mogollon de contenido en forma de ejemplos, foros muy activos, paquetes, incluso cursos interactivos on-line. El foro por excelencia en materia de programación: Stackoverflow Cursos on-line gratuitos en Datacamp A fin de cuentas, en mi opinion, la programación necesita de curiosidad y buenas fuentes de información para favorecer el **auto-aprendizaje**. Es buena praxis, cuando vallamos a trabajar en R, preguntar constantemente al señor/a "guugel", "ecosia", "opera"... (cualquiera menos "bing", eso es pa' psicopatas). Y **buscar en inglés aumenta nuestras probabilidades de éxito.**

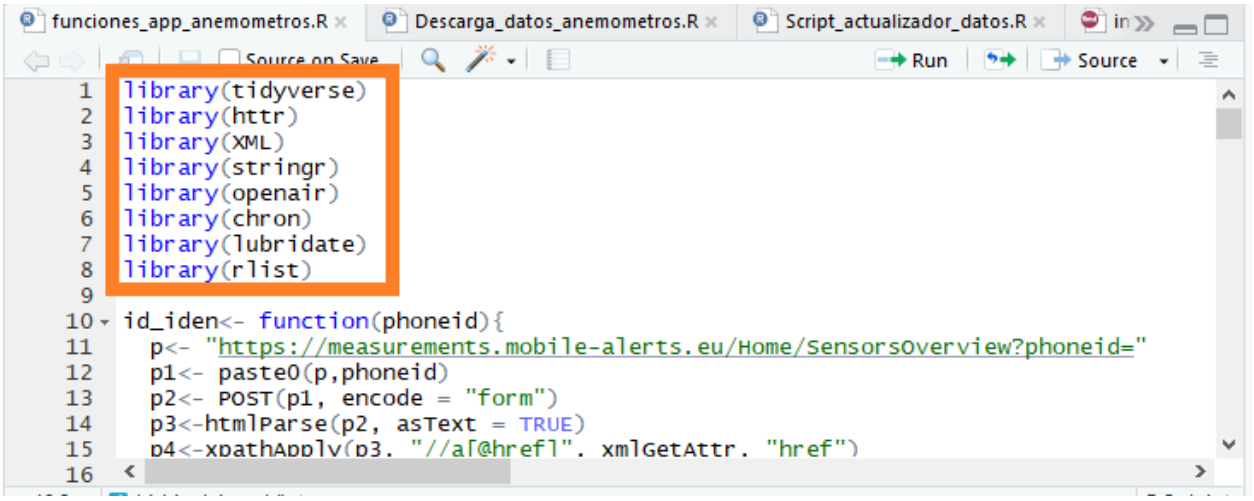
Instalar e importar paquetes

Para importar los datos vamos a necesitar dos paquetes. Uno relacionado con el tratamiento de los datos en formato lista (rlist) y el paquete comentado anteriormente (here). Se pueden importar datos de muchas maneras y formatos diferentes, nosotros en este ejemplo usamos esta manera por sencillez. Esto me sirve para explicar rapidamente como se instalan y se importan paquetes en Rstudio.

```
#Comando para instalar paquetes
install.packages("rlist", "here")

#Comandos para "cargar paquetes"
library(rlist)
library(here)
```

En Rstudio, cuando cerramos sesion se resetean los paquetes que usamos. Es decir, siguen instalados, pero hay que "llamarlos", "cargarlos", "importarlos" usando el comando **library()**. A modo de ejemplo podemos mirar como está organizado el *script: funciones_app_anemometros.R*.



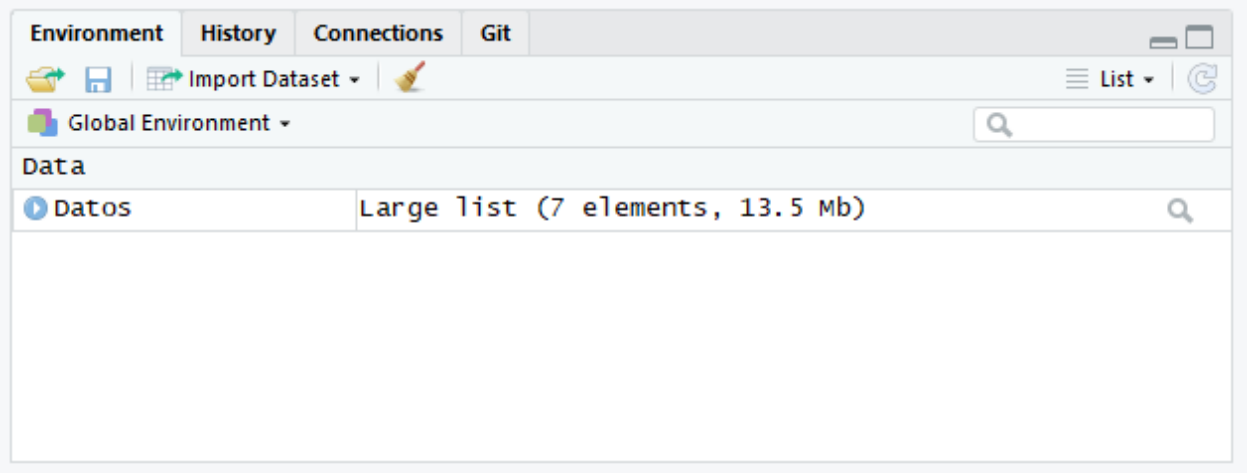
Se puede ver como las primeras líneas del código están dirigidas a "importar" todos los paquetes necesarios para que R pueda interpretar nuestro *script*.

Importar los datos de los sensores

Una vez hemos cargado los paquetes, importamos los datos.

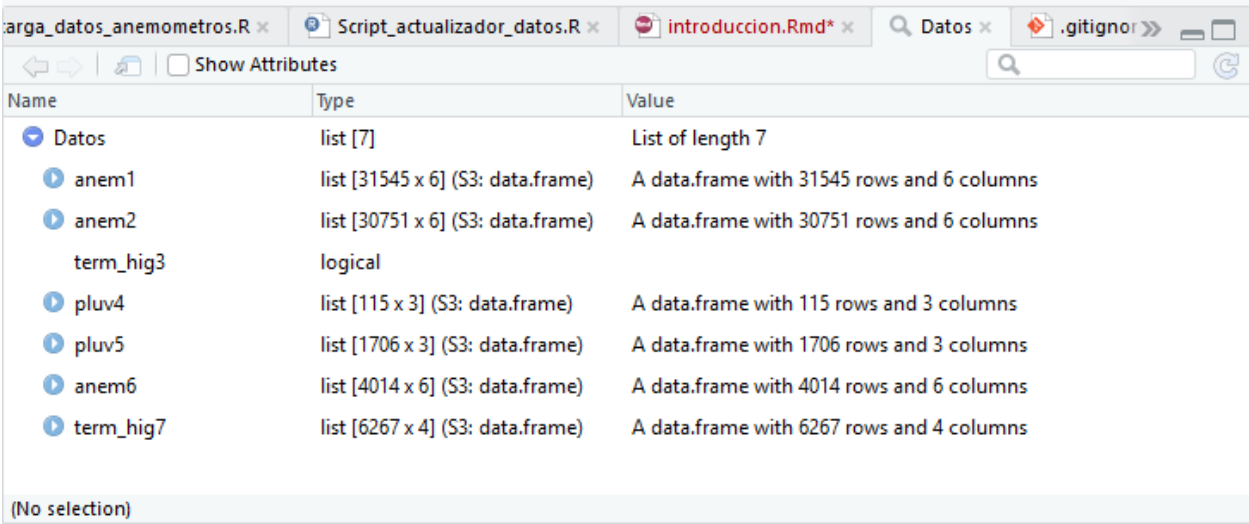
```
#Función para importar los datos.
Datos<- list.load(here::here("data/Datos_anemometros.rdata"))
```

Los datos se cargarán en una variable que nosotros en este caso hemos llamado *datos*. Si todo ha ido bien aparecerá en lo que se suele llamar *Enviroment*. Ahí podemos ver una resumida información: *lista de 7 elementos y 13.5 Mb*



Si pinchamos encima de *Datos* podemos acceder a algo más de información o, también podemos ir a dicha información ejecutando el siguiente comando.

```
View(Datos)
```



Accedemos a poder ver el nombre de cada elemento de la lista y la forma. En nuestro caso: **data.frames** (tabla). Nosotros queremos trabajar con *data.frames*.

Objetos tipo lista

Como ya vimos en el capitulo 1, usamos el comando class para ver de que tipo de objeto es datos.

```
class(Datos)
```

```
## [1] "list"
```

Vemos que se trata de una **lista**, es un formato de objeto muy utilizado en R por su versatilidad. En una lista se pueden agrupar muchos objetos de diferente tipo, lo cual ofrece una oportunidad para organizar y simplificar nuestro enviroment. Además ofrece muchas posibilidades a la hora de generar datos con loops (bucle for), pero eso lo veremos más adelante.

Refiriendonos a nuestro ejemplo, hemos agrupado la informacion de todos los sensores en una sola "variable"

llamada Datos (tipo lista).

Ya vimos en el capitulo 1 de la guía como obteniamos valores de un vector y una tabla. A continuación veremos como se obtiene la informacion contenida en una lista.

Ya vimos que nuestra lista se compone de 7 elementos (una tabla por cada sensor que se ha instalado), para acceder, por ejemplo, a los datos del anemometro 1 (anem1). Como ya vimos en el capitulo 1, el simbolo "$" nos puede ayudar a buscar esta información.

```
Datos$anem1
```

Comando head()

Al ejecutar el anterior comando saldrá por consola todos los elementos de *anem1*. 31000 filas por lo menos... **¡Demasiado!**. Muchas veces nos interesa ver por consola la información contenida en una variable, pero no necesitamos 30000 lineas para comprobar que aspecto tiene, con un par de filas nos vale. R tiene un comando interesante para esto.

Este comando es **head()**. Nos permite ver la "cabeza de la variable" es decir, saca por consola solo las 6 primeras líneas. Suficiente para nosotros, porque muchas veces lo que necesitamos es ver que variables contiene la tabla en cada columna. De esta manera no "petamos" la consola y nosotros encontramos la información que andabamos buscando.

```
head(Datos$anem1)
```

```
##   s.since_1.1.1970    date_string_hour Mean Max    Dir_ch Dir_deg
## 1       1540303537 23/10/2018 16:05:37  1.1 2.6 Northeast      45
## 2       1540303115 23/10/2018 15:58:35  2.1 4.8 Northeast      45
## 3       1540302694 23/10/2018 15:51:34  1.6 7.2 Northeast      45
## 4       1540302271 23/10/2018 15:44:31  0.9 6.5     North       0
## 5       1540301850 23/10/2018 15:37:30  1.1 4.8 Northeast      45
## 6       1540301428 23/10/2018 15:30:28  2.5 5.7 Northeast      45
```

Analizando el formato de los datos de los anemómetros

Ana vez hemos ejecutado el head() ya podemos analizar que información contiene nuestra tabla.

- **s.since_1.1.1970**: esta columna contiene información de tiempo, expresada en "Segundos desde el 1 de enero de 1970". Este formato a la hora de representar una fecha puede resultar raro, pero es muy utilizado en el análisis estadístico de datos meteorológicos.
- **date_string_hour**: información de tiempo expresado en formato "string" o palabra, una manera de ver las fechas de manera más legible para nosotros.
- **Mean**: velocidad del viento media.
- **Max**: Velocidad del viento máxima.
- **Dir_ch**: Dirección del viento en formato palabra.
- **Dir_deg**: Dirección del viento en formato grados. Ahora me gustaría explicar porqué los datos están organizados de este modo. Principalmente los datos tiene este aspecto por la manera en la que los sensores envían la información. Los sensores que usamos MA10660 suben datos a la nube cada 7 minutos (se puede comprobar en la tabla) y además envían información de velocidad máxima durante esos 7 minutos y velocidad media. En cuanto a la dirección del viento. La información contenida en la web está en formato palabra (*Dir_ch*), pero como a nosotros nos interesa trabajar con numeros, hemos hecho una transformación de los datos a formato grados. Tambien podemos acceder a la información de anem1, usando su posición dentro de la lista Datos. Antes vimos que *anem1* es el primero de la lista

```
head(Datos[[1]])
```

```
##   s.since_1.1.1970    date_string_hour Mean Max    Dir_ch Dir_deg
## 1       1540303537 23/10/2018 16:05:37  1.1 2.6 Northeast      45
## 2       1540303115 23/10/2018 15:58:35  2.1 4.8 Northeast      45
## 3       1540302694 23/10/2018 15:51:34  1.6 7.2 Northeast      45
## 4       1540302271 23/10/2018 15:44:31  0.9 6.5     North       0
## 5       1540301850 23/10/2018 15:37:30  1.1 4.8 Northeast      45
## 6       1540301428 23/10/2018 15:30:28  2.5 5.7 Northeast      45
```

Nótese que en el caso de las listas tenemos que usar doble corchete "[[ ]]" a diferencia de tablas y vectores en las que únicamente usabamos corchetes simples "[ ]"

Empezando a tratar con data.frames

A continuación extraeremos de la lista de toda la información solo la tabla de "anem1", para seguir trabajando avanzando en el análisis de datos. Para ello guardaremos la informacion contenida en "anem1" en una nueva variable que se añadirá a nuestro enviroment.

```
anem1<- Datos$anem1
```

Problemas: datos tipo factor

Una de las cosas que aun no llego a controlar del todo, es lo siguiente.Si miramos individualizadamente la columna de velocidad del viento media (Mean). Nos damos cuenta de que interpreta que son datos tipo "factor", lo cual no nos interesa a la hora de realizar una análisis de estos datos. A nosotros nos interesa que los datos sean tipo "numeric".

```
class(anem1$Mean)
```

```
## [1] "factor"
```

Transformación de factor a numérico

Que sea tipo factor significa que está metiendo los datos dentro de categorías o tipos. Pero nosotros no queremos que categorice la velocidad del viento. A nosotros nos interesa que sea un vector numérico y **punto**. Para realizar esta transformación de factor a numérico se pueden emplear dos formas:

Método 1

```
anem1$Mean<-as.numeric(levels(anem1$Mean))[anem1$Mean]
```

```
class(anem1$Mean)
```

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

Método 2

```
anem1$Mean<-as.numeric(as.character(anem1$Mean))
```

```
class(anem1$Mean)
```

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

Esto parece que está de más, pero es una error muy frecuente el de trabajar con factores y que las cosas no nos salgan como queremos. Por ello esto hay que tenerlo en cuenta.

Análisis preliminar de los datos

Comando Summary()

Existen varios comandos que arrojan información rápida acerca de una serie de datos. Quizás el más representativo es el comando **summary()**. Este comando nos hace un resumen de los datos que le introducimos.

```
summary(anem1$Mean)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.0000  0.3000  0.5313  0.8000  5.8000
```

Está muy bien porque nos informa del minimo, maximo, medio, mediana, y cuartiles. De esta manera podemos tener una visión genérica de los datos que tenemos.

Tambien podemos acceder a esta informacion de manera individualizada.

```
#Media
mean(anem1$Mean)
```

```
## [1] 0.5313172
```

```
#Mediana
median(anem1$Mean)
```

```
## [1] 0.3
```

```
#Quartiles
quantile(anem1$Mean)
```

```
##   0%  25%  50%  75% 100%
##  0.0  0.0  0.3  0.8  5.8
```

```
#Máximo y mínimo
max(anem1$Mean)
```

```
## [1] 5.8
```

```
min(anem1$Mean)
```

```
## [1] 0
```

Comandos familia which()

Imaginemos ahora que necesitamos saber cuando se produjo el máximo o el mínimo o cuando se dió un evento determinado. Para esta acción tenemos los comandos **which(), whic.max() y whic.min()** Busquemos la velocidad media máxima.

```
which.max(anem1$Mean)
```

```
## [1] 23142
```

Al ejecutar este comando buscamos el máximo. Pero ya vimos antes que el máximo es de 5.8 m/s. Este comando nos da la posición del máximo en el vector/tabla de la velocidad media. Este comando nos puede servir para identificar cuando se produjo el máximo. Lo que hacemos es buscar esa posición (fila) en la tabla *anem1*. Ya vimos como se buscaba una fila determinada en una tabla o matriz.

```
anem1[which.max(anem1$Mean),]
```

```
##       s.since_1.1.1970  date_string_hour Mean  Max         Dir_ch Dir_deg
## 23142       1530484860 2/7/2018 00:41:00  5.8 12.7 West-southwest     245
```

De esta manera podemos saber que el máximo se produjo el 2/7/2018. De la misma manera podemos buscar el mínimo empleando **whic.min()**. Empleando **which()** podemos buscar cualquier valor que queramos usando operaciones lógicas. Por ejemplo vamos a buscar las filas en las que el viento medio se encuentra por encima de 3 m/s.

```
anem1[which(anem1$Mean > 3),]
```

—

Recordatorio: los comandos which nos dan la posición dentro de una tabla en la que se cumple la condición que hemos puesto, de esta manera estamos sacando por consola todas las filas en las que el viento medio se encuentra por encima de 3 m/s y todas las columnas de la tabla.

Operaciones lógicas.

Es muy habitual tener que buscar información dentro de una tabla que atienda a unas especifícaciones. Para esto usamos operaciones lógicas.

- "==": ponemos dos iguales seguidos para buscar dentro de un vector valores concretos.

```
#Dime que valores son iguales que 2
anem1$Mean == 2
```

- "<=": menor o igual

```
#Dime que valores son menores o iguales que 2
anem1$Mean <= 2
```

- ">=": mayor o igual

```
#Dime que valores son mayores o iguales que 2
anem1$Mean >= 2
```

- "!=": diferente de.

```
#Dime que valores son diferentes de 2
head(anem1$Mean != 2)
```

```
## [1] TRUE TRUE TRUE TRUE TRUE TRUE
```

```
#Pongo un head() para que no me saque por consola todos los valores.
```

Observamos que al ejecutar estos comandos el resultado es un vector del mismo tamaño de *anem1$Mean* lleno de **TRUE o FALSE**, esto es lo que se llama un *vector lógico*. Nos da información de si se cumple la condición en cada elemento del vector o tabla. Estos vectores lógicos son útiles porque los podemos utilizar para generar una tabla nueva en la que se cumplan las especifícaciones, sin necesidad de usar el which().

```
#Devuelveme todas las filas en las que la velocidad
# del viento sea superior a 5 m/s

anem1[anem1$Mean > 5,]
```

```
##       s.since_1.1.1970   date_string_hour Mean  Max         Dir_ch Dir_deg
## 23142       1530484860  2/7/2018 00:41:00  5.8 12.7 West-southwest     245
## 27080       1528823945 12/6/2018 19:19:05  5.1 12.5 West-southwest     245
## 27108       1528812133 12/6/2018 16:02:13  5.1 11.9 West-southwest     245
```

```
## 27116       1528808758 12/6/2018 15:05:58  5.2 14.1 West-southwest     245
## 27118       1528807914 12/6/2018 14:51:54  5.4 12.7      Southwest     225
## 27119       1528807492 12/6/2018 14:44:52  5.6 12.4 West-southwest     245
## 27121       1528806648 12/6/2018 14:30:48  5.3 13.2 West-southwest     245
```

Tambien podemos usar *puertas* lógicas normales como pueden ser **AND, OR, XOR, NOT y NAND.**

- "&": operación AND

```
#Devuelve las filas een las el la velocidad del viento
# sea mayor que 3 y menor que 3.2.
anem1[anem1$Mean > 3 & anem1$Mean < 3.2 ,]
```

- "|": operación OR

```
#Devuelve las filas en las el la velocidad del viento
# este en el intervalo (3 , 3.2) o en el intervalo (4 , 4.2)
anem1[(anem1$Mean > 3 & anem1$Mean < 3.2 ) | (anem1$Mean > 4 & anem1$Mean < 4.2 ) ,]
```

- "!": operación NOT

```
#Devuelve las filas en las el la velocidad del viento
# NO este en el intervalo (3 , 3.2) o en el intervalo (4 , 4.2)
anem1[ !((anem1$Mean > 3 & anem1$Mean < 3.2 ) | (anem1$Mean > 4 & anem1$Mean < 4.2 ) ),]
```

Simplemente añadiendo un "!" conseguimos la negada de la operación deseada. —

Nótese la importancia del correcto uso de los parentesis a la hora de realizar las operaciones lógicas. Esto es una fuente de errores bastante frecuente

---