

Prácticas de Ecología de Sistemas Acuáticos

Jorge Juan Montes Pérez (jmontesp@uma.es)

2023-04-12

Índice general

¿Cómo seguir la práctica?	5
1 Introducción	7
2 Estructura de la práctica	9
3 Descargar los datos de la red GLEON	11
4 Primero pasos	15
4.1 Ejercicios	20
5 Paquetes importantes y sus funciones básicas	21
5.1 Importar datos	21
5.2 Organizar tablas	23
5.3 Gráficas	24
5.4 Trabajar con fechas	30
6 Estabilidad térmica de la columna de agua	33
6.1 Perfil vertical	33
6.2 Termoclina	37
6.3 Capas en un lago estratificado	38
6.4 Estabilidad	40
6.5 Series temporales	42
7 Gráficas de contorno	45

¿Cómo seguir la práctica?

Este documento está editado con bookdown. Lo podéis leer directamente en html (recomendado) o descargarlo tanto en formato pdf como en formato mobi. En la parte superior, encontraréis un icono para desplegar u ocultar la tabla de contenidos, un icono para buscar dentro del documento, un icono para descargar los formatos pdf y mobi y varios icono para compartir a través de distintas plataformas.

A lo largo del documento entraréis “trocitos” de código R. Si pincháis en el recuadro code se desplegará todo el código y os aparecerá un icono para copiarlo. De esta manera, podréis ir copiando el código a vuestro script de R e ir siguiendo cada paso de la práctica.

Prácticas Ecología de Sistemas Acuáticos by Jorge Juan Montes Pérez is licensed under a Creative Commons Reconocimiento-NoComercial 4.0 Internacional License.

Capítulo 1

Introducción

Debido al gran avance tecnológico de los últimos años, se ha conseguido una amplia variedad de dispositivos que permiten registrar una ingente cantidad de información con relativamente poco esfuerzo. Un ejemplo de esto, son los dispositivos que registran información de variables de interés (Ej: temperatura, humedad, irradiancia) a un determinado intervalo de tiempo y de manera autónoma. Muchos de estos dispositivos pueden ser instalados en lugares remotos y transmitir la información telemáticamente o almacenarla en la memoria interna. Esto a supuesto un gran avance en el campo de la ecología acuática¹. Cómo podéis imaginar, estos dispositivos han permitido obtener una gran cantidad de información con una resolución temporal (de incluso minutos o segundos) y durante grandes periodos de tiempo (meses/años) que sería impracticable mediante los métodos tradicionales.



Figura 1.1: **Sistemas de monitoreo de alta frecuencia.** *Izquierda:* boyá flotante que realiza medidas cada 10 minutos a una profundidad de 1.5 metros en el embalse de Sau (Barcelona). *Derecha:* boyá flotante que realiza perfiles verticales desde la superficie hasta el fondo con un resolución espacial de 1 metro y una resolución temporal de 2 horas en el embalse de El Gergal (Sevilla).

Los monitoreos de alta frecuencia (HFM) tienen muchas aplicaciones dentro de la ecología acuática, tanto en el ámbito de la gestión como de la investigación. Por ejemplo, pueden ser usados para controlar la calidad del agua en un embalse que suministra agua potable a una ciudad, control de vertidos de una industria, estudiar el efecto de eutrofización en un lago de alta montaña o detectar cambios en las corrientes marinas.

Sin embargo, no todo es de color de rosas. Imaginaos que, después de dos años, vamos a recoger la información que ha almacenado nuestro sensor de temperatura y oxígeno disuelto que dejamos colocado en el centro de un lago de los pirineos. Al descargar la información, nos encontramos que tenemos 1.036.800 registros (porque claro está, queríamos registrar los datos cada segundo)... Echamos manos mano de nuestro amado Excel (Calc para los radicales del software libre) e intentamos calcular la temperatura media de esos dos años entre las 00:00 a las 08:00. A mí me ha entrado un sudor frío. Y es que, esta vasta y valiosa información tiene

¹Por supuesto, esto ha revolucionado infinidad de campos como la biomedicina, ingeniería, informática, meteorología, etc

un inconveniente... tenemos que trabajar con miles o millones de datos. Por suerte, se disponen de muchas herramientas para trabajar con los datos.

En esta práctica de Ecología de Sistemas Acuáticos, en la que vamos a trabajar con datos de HFM para estudiar la estructura térmica de un lago, me gustaría presentaros una herramienta, que a mi parecer, es fundamental para cualquier biólogo y que os permitirá realizar todo el trabajo (desde organización de la información hasta su visualización, pasando por el análisis estadístico) con un solo software libre, de código abierto y gratuito. Estamos hablando de R. Para los que no lo conocéis, Si buscáis R en google os aparecerá en las primera 4-5 entradas. Esto nos puede dar una idea de su relevancia a nivel mundial.

A partir de aquí, no os voy a engañar, R no es, por lo general, muy agradable y no suele despertar simpatías. ¡Echad un vistazo a la figura de abajo!

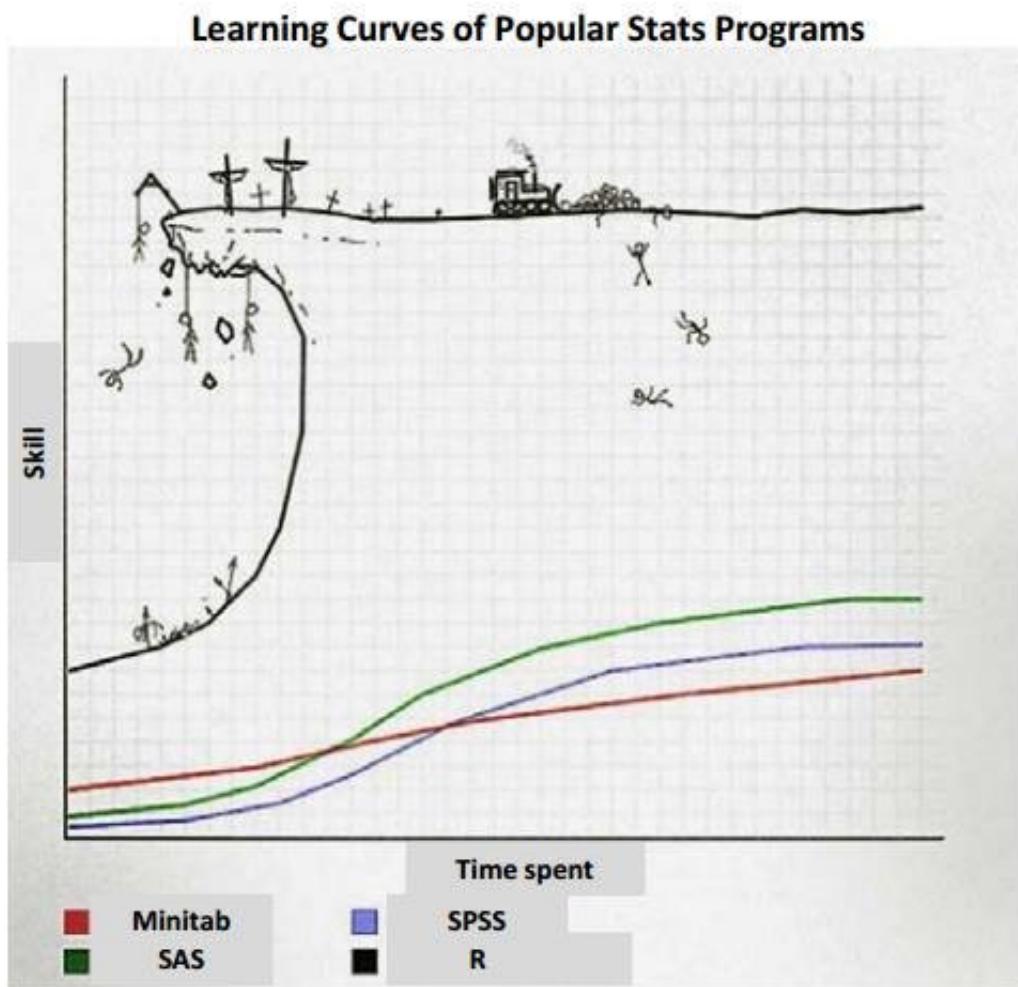


Figura 1.2: Comparación de la curva de aprendizaje de distintos software estadísticos. Fuente: <https://twitter.com/rogierK/status/730863729420701697>

Sin embargo, cuando esos momentos de flaqueza acontezcan imaginad si sería posible hacerlo con otra herramienta y el tiempo que os llevaría. Además, cada vez que tengáis que hacer un estudio o evaluación necesitaréis usar Excel para ordenar los datos, SPSS (u otro programa de estadística) para el análisis y sigma plot (o similar) para hacer gráficas decentes. Mejor no hablamos de software específicos para cada campo: VENSIM, PRIME, Ocean Data View, SURFER, etc. A la larga el tiempo invertido habrá merecido la pena.

Capítulo 2

Estructura de la práctica

1. Obtener los datos con los que vamos a trabajar. Usaremos la red [The Global Lake Ecological Observatory Network \(GLEON\)](#). Esta red pone a nuestra disposición una amplia cantidad de datos de monitoreo de alta frecuencia (HFM) de distintos lugares del mundo.
2. [Primeros pasos en R](#).
3. [Familiarizarnos con los paquetes y funciones básicos de R que nos permiten explorar y trabajar con grandes tablas de datos](#).
4. [Calcular profundidad de la termoclina y la estabilidad de la columna de agua \(número de Smidch\)](#) usando el paquete de R [rLakeAnalyzer](#).
5. [Gráficas de contorno](#).

Capítulo 3

Descargar los datos de la red GLEON

Para ello, visitamos la página de la red [GLEON](#) y nos vamos al apartado de [datos](#). En esta sección podemos encontrar la [política de datos de GLEON](#), básicamente se apuesta por una ciencia colaborativa en la que los datos quedan a disposición de la comunidad para cualquier fin de investigación, académico, educativo o cualquier otro, siempre que no haya un interés lucrativo detrás y respetando algunos principios de comunicación con los responsables de los datos. Como se muestra en esta sección, a los datos de GLEON se puede acceder a través de tres buscadores [EDI](#), [DataONE](#) o [Google data set](#).

Pues bien, para este práctica vamos a trabajar, en concreto, con datos del [lago Crystal](#). Así que utilizando el buscador que más sea de vuestro agrado lanzamos la siguiente búsqueda: *crystal lake*. Entre los resultados obtenidos (hay bastante información como podéis observar), vamos a seleccionar los datos derivados del proyecto [North Temperate Lakes Long Term Ecological Research \(NTL-LTER\)](#) que nos ofrecen datos de temperatura, oxígeno disuelto, clorofila *a* y pH desde 2011 hasta 2014. Si no pudierais encontrarlos, podéis pinchar [aquí: North Temperate Lakes LTER High Frequency Water Temperature Data, Dissolved Oxygen, Chlorophyll, pH - Crystal Lake 2011 - 2014](#).

En esa página que acabáis de abrir tenéis un sumario con toda la información necesario sobre el paquete de datos (*Title, Creators, Publication Date, Citation, Abstract, Spatial Coverage, Package ID, Resources, Intellectual Rights, Digital Object Identifier, PASTA Identifier, Code Generation, Provenance, Journal Citations*). Las que más nos van a interesar por el momento son *Abstract, Resources* y *Code Generation*. La primera de ellas es un resumen que nos explica como han sido recogido los datos y algunas particularidades que debemos saber, en la segunda tenemos directamente los archivos con los datos para descargarlos en formato *.csv y una opción muy interesante, [View Full Metadata](#), en la que si desplegamos *Data Entities* podemos ver información sobre las variables que aparecen en la tabla de datos como, por ejemplo, las unidades en las que están medidas.

En este caso, disponemos solo de un fichero. Para descargar los datos tenemos dos opciones:

1. Podemos pinchar directamente en el archivo y descargarlo a través del navegador. *Name: High Resolution Water Temperature Dissolved Oxygen Chlorophyll pH - Crystal Lake File: ntl303_v1_0.csv* (120.5 MiB; 22 downloads). Si optamos por esta opción, posteriormente habrá que importar los datos a R.
2. Otra opción mucho más cómoda es la de usar un script de R que ya nos han preparado para facilitarnos la descarga e importación. Para ello, tenemos dos opciones también:
 - Pinchamos en el [ícono de R](#) en el apartado *Code Generation*.
 - Pinchamos en la opción [tidy](#) en el apartado *Code Generation*. Os recomiendo usar esta última, es la que vemos más abajo y, además, si no lo tenemos instalado, instala automáticamente el paquete [Tidyverse](#). Tidyverse es en realidad un conjunto de paquetes de R especialmente diseñado para la ciencia de datos que nos vendrá de maravilla para esta práctica.

Independientemente por cual os decantéis, debéis abrir el archivo pinchando en *File Download: knb-lter-ntl.303.20.r* o *File Download: knb-lter-ntl.303.20.tidy* y se abrirá automáticamente con R, si no es así, lo

descargáis y lo abrís posteriormente con R. Una vez abierto ya podéis ejecutar el script (Ctrl+A y después Ctrl+Enter). Como veis también hay opción para descargar y trabajar directamente los datos con otras herramientas, si a alguno le pica la curiosidad ¡adelante!. En esta práctica como hemos dicho vamos a usar R, debido a que es una herramienta gratuita, de código abierto y libre. Además es ampliamente usado en investigación debido a su naturaleza libre y colaborativa. En fin, si más dilaciones, podéis descargar el script, abrirlo con RStudio y ejecutarlo ¡A ver qué pasa!.

Esta es la pinta que tiene el script:

```

1 # Package ID: knb-lter-ntl.303.20 Cataloging System:https://pasta.edirepository.org.
2 # Data set title: North Temperate Lakes LTER High Frequency Water Temperature Data, Dissolved
   Oxygen, Chlorophyll, pH – Crystal Lake 2011 – 2014.
3 # Data set creator: John Magnuson – University of Wisconsin
4 # Data set creator: Stephen Carpenter – University of Wisconsin
5 # Data set creator: Emily Stanley – University of Wisconsin
6 # Data set creator: NTL Lead PI – University of Wisconsin
7 # Metadata Provider: NTL Information Manager – University of Wisconsin
8 # Contact: NTL Information Manager – University of Wisconsin – ntl.infomgr@gmail.com
9 # Contact: NTL Lead PI – University of Wisconsin – ntl.leadpi@gmail.com
10 # Stylesheet for metadata conversion into program: John H. Porter, Univ. Virginia,
    jporter@Virginia.edu
11 #
12 #install package tidyverse if not already installed
13 if(!require(tidyverse)){ install.packages("tidyverse") }
14 library("tidyverse")
15 infile1 <- trimws("https://pasta.lternet.edu/package/data/eml/knb-lter-ntl/303/20/
   b9b3b932deec8f3e71fb8d70cacf6a0e")
16 infile1 <- sub("^https","http",infile1)
17 # This creates a tibble named: dt1
18 dt1 <-read_delim(infile1
19   ,delim=","
20   ,skip=1
21   , col_names=c(
22     "sampledate",
23     "year4",
24     "daynum",
25     "sample_time",
26     "depth_calculated",
27     "wtaer_temp",
28     "flag_water_temp",
29     "pH",
30     "flag_ph",
31     "chlorophylla",
32     "flag_chlorophylla",
33     "opt_do2",
34     "flag_do2",
35     "opt_dosat_raw",
36     "flag_opt_dosat_raw"    ),
37   col_types=list(
38     col_character(),
39     col_number(),
40     col_number(),
41     col_character(),
42     col_number(),
43     col_number(),
44     col_character(),
45     col_number(),
46     col_character(),
47     col_number(),
48     col_character(),
49     col_number(),
50     col_character(),
51     col_number(),
52     col_character()),
53   na=c(" ", ".", "NA"))
54
55
56 # Observed issues when reading the data. An empty list is good!
57 problems(dt1)
```

```

58 # Here is the structure of the input data tibble:
59 glimpse(dt1)
60 # And some statistical summaries of the data
61 summary(dt1)
62 # Get more details on character variables
63
64 summary(as.factor(dt1$flag_water_temp))
65 summary(as.factor(dt1$flag_ph))
66 summary(as.factor(dt1$flag_chlorophylla))
67 summary(as.factor(dt1$flag_do2))
68 summary(as.factor(dt1$flag_opt_dosat_raw))

```

Esta primera parte del script es para descargar los datos (igual que arriba, salvo que me he tomado el tiempo de comentar algunas líneas):

```

1 # Package ID: knb-lter-ntl.303.20 Cataloging System:https://pasta.edirepository.org.
2 # Data set title: North Temperate Lakes LTER High Frequency Water Temperature Data, Dissolved
   Oxygen, Chlorophyll, pH – Crystal Lake 2011 – 2014.
3 # Data set creator: John Magnuson – University of Wisconsin
4 # Data set creator: Stephen Carpenter – University of Wisconsin
5 # Data set creator: Emily Stanley – University of Wisconsin
6 # Data set creator: NTL Lead PI – University of Wisconsin
7 # Metadata Provider: NTL Information Manager – University of Wisconsin
8 # Contact: NTL Information Manager – University of Wisconsin – ntl.infomgr@gmail.com
9 # Contact: NTL Lead PI – University of Wisconsin – ntl.leadpi@gmail.com
10 # Stylesheet for metadata conversion into program: John H. Porter, Univ. Virginia,
    jporter@Virginia.edu
11 #
12 #install package tidyverse if not already installed. Aquí cargamos el conjunto de paquetes que
   están dentro de "tidyverse" y si no lo tuvieramos instalado, han pensado en nosotros y, se
   instala solo.
13 if(!require(tidyverse)){ install.packages("tidyverse") }
14 library("tidyverse")
15 infile1 <- trimws("https://pasta.lternet.edu/package/data/eml/knb-lter-ntl/303/20/
   b9b3b932deec8f3e71fb8d70cacf6a0e")
16 #Esta, de arriba, es la dirección de donde descarga los datos
17 infile1 <-sub("^https","http",infile1)
18 # This creates a tibble named: dt1
19 dt1 <-read_delim(infile1 #Crea un objeto donde descarga los datos de la dirección que le damos,
   guardada en el objeto con nombre "infile1"
20   ,delim=","
21   ,skip=1
22   , col_names=c(      #Asigna nombre a las columnas
23     "sampledate",
24     "year4",
25     "daynum",
26     "sample_time",
27     "depth_calculated",
28     "wtaer_temp",
29     "flag_water_temp",
30     "pH",
31     "flag_ph",
32     "chlorophylla",
33     "flag_chlorophylla",
34     "opt_do2",
35     "flag_do2",
36     "opt_dosat_raw",
37     "flag_opt_dosat_raw"  ),
38   col_types=list(
39     col_character(),
40     col_number() ,
41     col_number() ,
42     col_character(),
43     col_number() ,
44     col_number() ,
45     col_character(),
46     col_number() ,

```

```

47   col_character() ,
48   col_number() ,
49   col_character() ,
50   col_number() ,
51   col_character() ,
52   col_number() ,
53   col_character()) ,
54   na=c(" ", ".","NA") )

```

Este trocito de código que sigue es para corregir algún problema de formato que se haya podido introducir debido a algún error en la base de datos.

```

1 # Observed issues when reading the data. An empty list is good!
2 problems(dt1)

```

Este último trozo es simplemente para ver la estructura de los datos y un resumen de cada una de las variables. Este trozo no es necesario que lo ejecutéis.

```

1 # Here is the structure of the input data tibble:
2 glimpse(dt1)
3 # And some statistical summaries of the data
4 summary(dt1)
5 # Get more details on character variables
6
7 summary(as.factor(dt1$flag_water_temp))
8 summary(as.factor(dt1$flag_ph))
9 summary(as.factor(dt1$flag_chlorophylla))
10 summary(as.factor(dt1$flag_d02))
11 summary(as.factor(dt1$flag_opt_dosat_raw))

```

Bien, una vez ejecutado el script, ya debemos tener los datos en nuestro entorno de RStudio en un tipo de objeto denominado data.frame (básicamente una tabla de datos). Vamos a ver que pinta tienen:

```

1 head(dt1)

```

Por último, los vamos a guardar en la carpeta “Datos” que hemos creado.

```

1 write.csv(dt1, "./Datos/Datos_Crystal.csv", row.names = FALSE)

```

Capítulo 4

Primero pasos

A estas alturas es posible que ya se haya generado una histeria colectiva. A la mayoría de vosotr@s ya os habrá dado algún error y, a l@s que no, esa cantidad de código ininteligible os habrá quitado las ganas de vivir. ¡Que no cunda el pánico!

Vamos a ver alguna nociones básicas sobre R. Si os preguntáis porqué ahora y no antes: porque hasta que no surgen las dudas, no tiene sentido dar las respuestas.

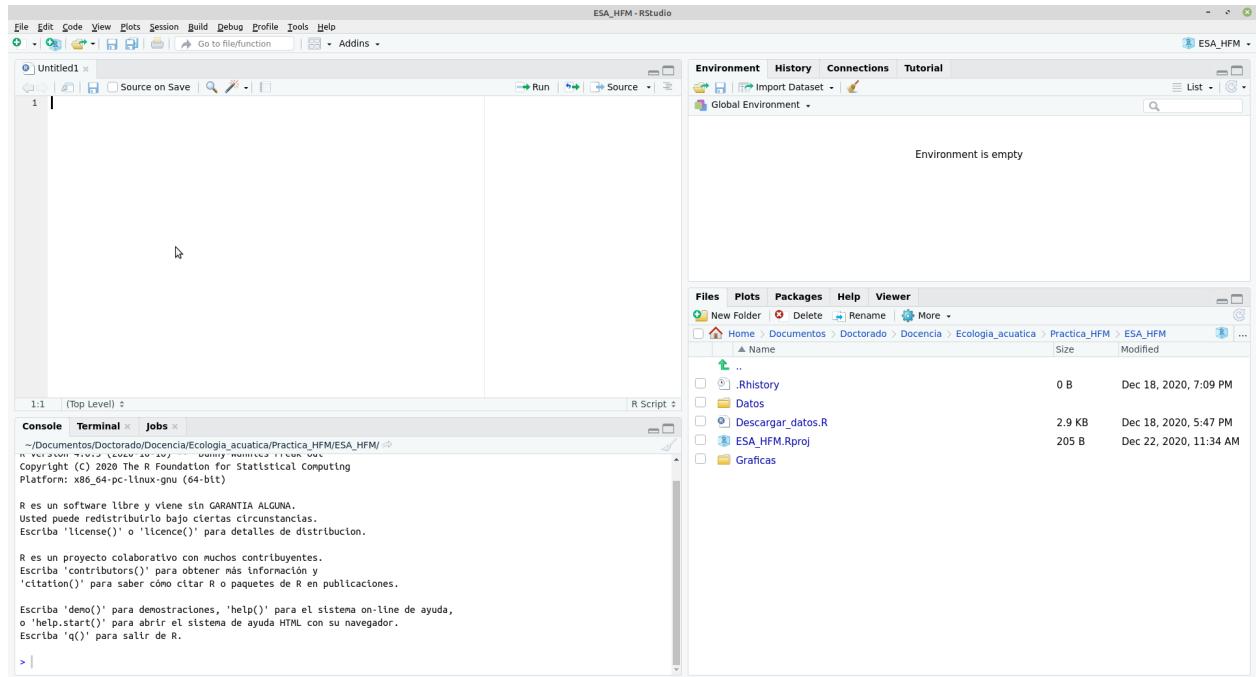


Figura 4.1: Interfaz de R.

RStudio es una IDE (entorno de desarrollo integrado) para R. En román paladino, RStudio nos permite usar R de una manera más intuitiva y con muchas herramientas que nos facilitan la vida. Como vamos usar RStudio para comunicarnos con R, cada vez que hablemos de R vamos a referirnos indistintamente a R y a RStudio. Bien, pues en la imagen de arriba vemos 4 subventanas. Veamos una a una:

En esta ventana se muestra el script. El script no es, ni más ni menos, que una hoja donde vamos escribiendo todas las ordenes que le queremos dar a R. Nos permite ir guardando todo lo que hacemos, modificar sobre lo que ya hemos hecho, ejecutar sólo la parte que nos interesa y organizar la información. Vamos a escribir

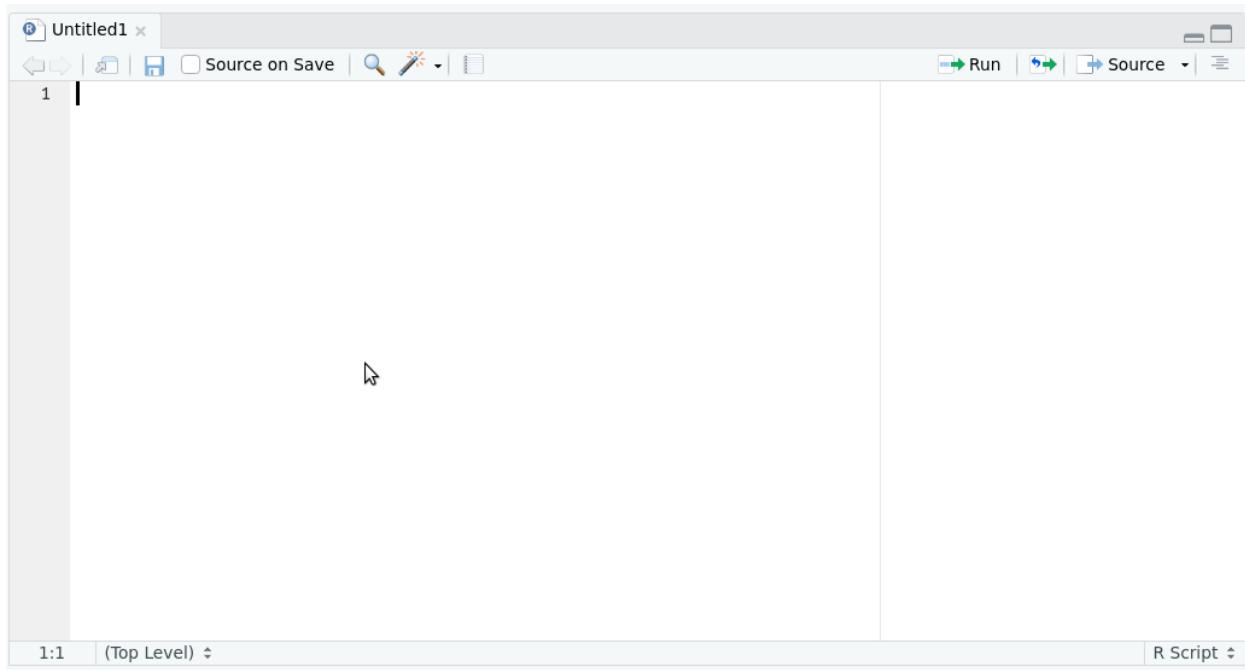


Figura 4.2: Scripts en R.

nuestra primera línea en R, usándolo como una simple calculadora (pincha sobre el recuadro code para ver el código):

```
1 2+2
```

```
1 ## [1] 4
```

Si escribís $2+2$ y pulsáis Ctrl+Enter se ejecuta solo la línea en la que estamos. Veréis que el resultado (4) os aparece en la ventana consola:

Como podéis intuir la consola es donde tiene lugar la parte importante. En ella se ejecutan las ordenes, nos informa de los errores y nos devuelve la información que le pedimos.

Volvamos al script. R es una lenguaje orientado a objetos, esto se traduce en que vamos a ir guardando la información en “objetos” virtuales y vamos a trabajar con estos objetos. Siguiente el ejemplo anterior, vamos a crear nuestros primeros objetos:

```
1 x <- 2
2 y <- 2
```

Hemos creado dos objetos, una llamado *x* y otro llamado *y*. Ambos guardan la información del valor 2. Además, si os habéis fijado, en la ventana superior derecha os han aparecido esos dos objetos:

En esta ventana aparecerán todos los objetos que hemos creado durante nuestra sesión de trabajo.

Ahora vamos a sumar estos dos objetos:

```
1 x+y
```

The screenshot shows the R console window. At the top, there are tabs for 'Console', 'Terminal', 'R Markdown', and 'Jobs'. The main area displays the R startup message, which includes the path (~/Documentos/Doctorado/Docencia/Ecologia_acuatica/Practica_HFM/), the version (R version 4.0.3 (2020-10-10) -- "Bunny-Wunnies Freak Out"), copyright information (Copyright (C) 2020 The R Foundation for Statistical Computing), and platform details (Platform: x86_64-pc-linux-gnu (64-bit)). Below the message, there are several paragraphs of text about R's license, its collaborative nature, and how to use help functions. At the bottom of the console window, there is a command prompt with the text '> 2+2' and '[1] 4'.

```

Console Terminal × R Markdown × Jobs ×
~/Documentos/Doctorado/Docencia/Ecologia_acuatica/Practica_HFM/
R version 4.0.3 (2020-10-10) -- "Bunny-Wunnies Freak Out"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R es un software libre y viene sin GARANTIA ALGUNA.
Usted puede redistribuirlo bajo ciertas circunstancias.
Escriba 'license()' o 'licence()' para detalles de distribucion.

R es un proyecto colaborativo con muchos contribuyentes.
Escriba 'contributors()' para obtener más información y
'citation()' para saber cómo citar R o paquetes de R en publicaciones.

Escriba 'demo()' para demostraciones, 'help()' para el sistema on-line de ayuda,
o 'help.start()' para abrir el sistema de ayuda HTML con su navegador.
Escriba 'q()' para salir de R.

[Workspace loaded from ~/Documentos/Doctorado/Docencia/Ecologia_acuatica/Practica_HFM/.RData]

> 2+2
[1] 4
>

```

Figura 4.3: Consola R.

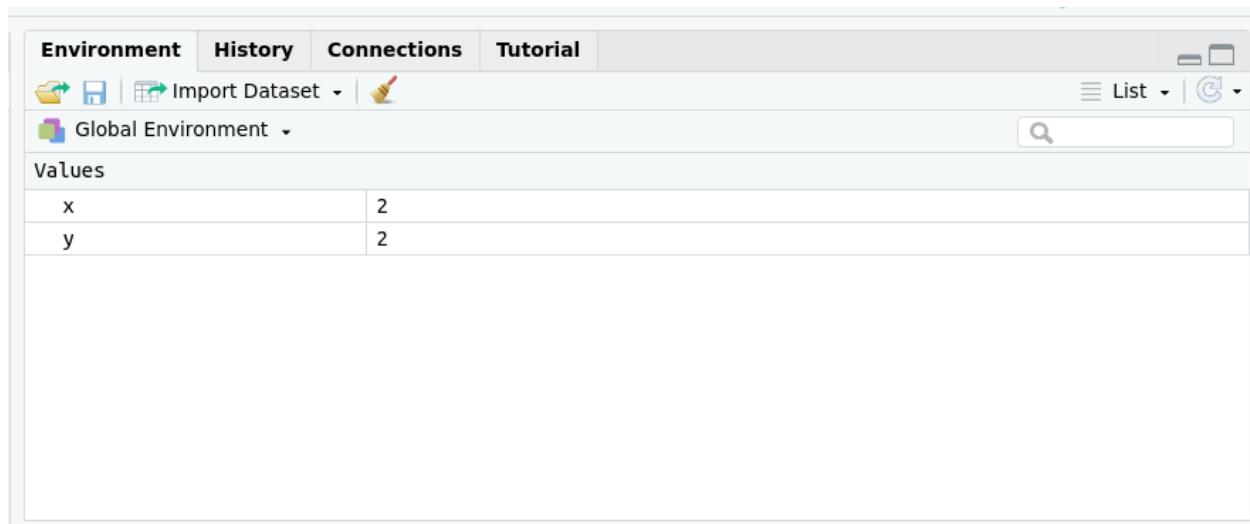


Figura 4.4: Ventana Environment.

```
1 ## [1] 4
```

¡Enhorabuena! ya habéis creado vuestros primeros objetos en R y habéis trabajado con ellos. Ahora vamos a sacarle más partido a esto. Como imaginaréis, en un objeto de R se puede almacenar muchas más información que un sólo número. Vamos a imaginar que tenemos información sobre la temperatura del agua medida en distintos sistemas epicontinentales y queremos guardar esa información en un objeto llamado Temperatura:

```
1 Temperatura <- c(23, 23.5, 20, 25)
```

Ahora vamos a calcular la temperatura media y su desviación:

```
1 mean(Temperatura)
```

```
1 ## [1] 22.875
```

```
1 sd(Temperatura)
```

```
1 ## [1] 2.096624
```

¡Ojo! R es muy quisquilloso y si no le dais el nombre exacto de Temperatura (incluyendo mayúsculas y minúsculas) no os entenderá y os devolverá un error. Por lo tanto, sed muy cuidadosos con la sintaxis y revisad que todo está escrito correctamente (que no falte ningún paréntesis o corchete).

En estas últimas líneas han sucedido un par de cosas: hemos creado un objeto nuevo, un vector, y hemos usado dos funciones `mean()` y `sd()` que calculan la media y desviación estándar.

Un vector almacena información de un mismo tipo, en este caso numérico. Existen distintos tipos de vector en función de la información que almacén:

1. Numérico: almacenan números.
2. Carácter: almacenan texto.
3. Factores: almacenan factores, generalmente texto pero que usaremos como un tratamiento en el análisis.
4. Fechas: almacenan fechas en distintos formatos.

En cuanto a las funciones, existen infinidad de ellas. R trae muchas funciones básicas pero si queremos aplicar funciones más específicas necesitaremos instalar paquetes que han sido creados específicamente para ello.

La estructura genérica de una función sería.

```
1 nombre_funcion(x = objeto_donde_aplicarla, argumento1 = "valor1", argumento2 = "valor2")
```

Si queremos obtener más información sobre una función y su uso podéis usar la función `help("funcion_que_os_interesa")` o más cómodo `?funcion_que_os_interesa`:

```
1 ?mean
```

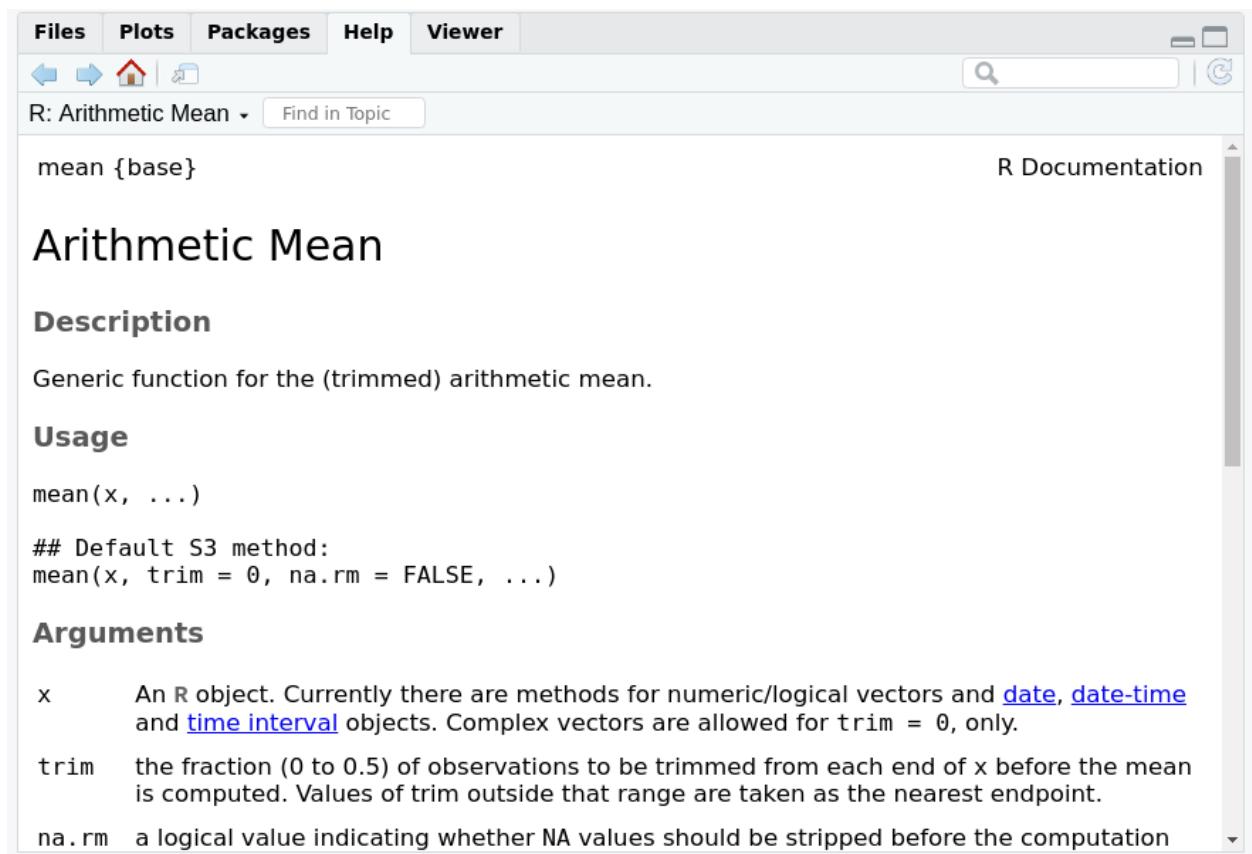


Figura 4.5: Ayuda en RStudio

Veréis que se os abrirá la última ventana que nos quedaba por ver:

Aquí encontréis la información necesaria para usar la función así como algunos ejemplos.

Por último, vamos a ver otro objeto muy útil: `data.frame`. Un `data.frame` no es más que una tabla de datos, como la que acostumbráis a hacer en Excel. Veamos un ejemplo:

Ya tenemos la temperatura del agua que se ha registrado en distintos sistemas epicontinentales (Temperatura). Ahora nos dan la información del tipo de sistema y queremos crear un tabla que recoja toda la información:

```

1 Temperatura <- c(23, 23.5, 20, 25)
2 Sistemas <- c("Lago", "Lago", "Rio", "Rio") #Creamos un vector esta vez de tipo caracter (el
   texto va entre comillas)
3
4 Temperatura_en_distintos_sistemas <- data.frame(Temperatura,
   Sistemas)
5
6 View(Temperatura_en_distintos_sistemas) #La función View() nos muestra el objeto que deseemos.

```

Se os abrirá una ventana con la información recogida en el `data.frame` `Temperatura_en_distintos_sistemas`. En este último trozo de código habéis visto un elemento nuevo: comentarios. Es altamente recomendado comentar el código que estamos escribiendo. De este modo, será mucho más claro y cuando vuelvas a abrir el script después de meses “tu yo del futuro” te lo agradecerá. Además si queréis compartir código con otr@s compañer@s es necesario añadir comentarios para que sepan que hacemos en cada sección. Para comentar, como habéis visto, sólo hay que añadir `#` delante del texto y este no se ejecutará. Si no ponéis `#` R lo entenderá como una orden y os dará error.

Otra cosa que habréis podido notar es que, aunque R nos ayuda autocompletando la frase, el nombre `Temperatura_en_distintos_sistemas` no es muy funcional que se diga. Es más práctico usar nombres cortos, pero nos indiquen claramente de qué se trata, a la hora de nombrar objetos en R.

A la hora de nombrar objetos en R hay algunas nombres que debemos respetar.

1. Los nombres no pueden ser números o empezar con número. No podríamos usar 1 ó 2 ni 2datos, pero sí podríamos usar datos2.
2. No se usar caracteres especiales: `#`, `!`, `@`, etc.
3. Los espacios tampoco pueden ser usado dentro de los nombres. Usar en su lugar mayúsculas ó barra baja. Por ejemplo, usar DatosLagos o `Datos_lagos` en lugar de `Datos lagos`.

```

1 Temperatura <- c(23, 23.5, 20, 25)
2 Sistemas <- c("Lago", "Lago", "Rio", "Rio") #Creamos un vector esta vez de tipo caracter (el
   texto va entre comillas)
3
4 Temp_sistemas <- data.frame(Temperatura,
   Sistemas)
5
6 View(Temp_sistemas) #La función View() nos muestra el objeto que deseemos.

```

¡Genial! Ya hemos visto los objetos básicos. Irán saliendo más a lo largo de la práctica, pero por ahora es suficiente.

4.1 Ejercicios

1. ¿Qué sucede si corremos `Sistemas[2]`? Prueba con distintos números.
2. ¿Y si corremos `Temp_sistemas[2]`? Da error... prueba `Temp_sistemas[,2]` ¿Y si ponemos un número delante de la `,`? ¿Que indican los números que van delante de la coma y los que van detrás?
3. Además de la temperatura se midió el pH: 7.8, 7.5, 9 y 8.5. Incluye estos valores en el `data.frame` `Temp_sistemas`. Hazlo con los conocimientos que tienes y luego prueba con la función `cbind()`.
4. Nos dan también la concentración de clorofila a, pero en este caso falta un dato y lo rellenamos con `NA`, que significa que no está disponible. Clas `<-c(15.6, NA, 3, 4)`. Calcula la concentración de clorofila media. Recuerda, cuando tengas dudas sobre como utilizar una función: `?mean`.

Capítulo 5

Paquetes importantes y sus funciones básicas

Ya hemos dado nuestros primeros pasos en R, conocemos su entorno RStudio, hemos creado nuestros primeros objetos y aplicado un par de funciones básicas. Ahora vamos a dar un paso más allá y vamos a ver algunos de los más paquetes que pueden ser más importantes para nosotros y sus funciones más útiles.

Para esta parte de la práctica ya debemos haber descargado los datos de la plataforma GLEON como se indicó en la [sección 3](#).

Lo primero que nos preguntaremos, ¿Qué es un paquete? Un paquete de R es un conjunto de funciones que han sido creadas con un objetivo común, por ejemplo hacer gráficas, y que han sido agrupadas para que puedan ser cargadas en R todas juntas.

Para instalar un paquete no tenemos más que ejecutar:

```
1 install.packages("nombre_del_paquete")
```

5.1 Importar datos

La mayoría de los paquetes que vamos a ver durante la práctica han sido agrupados en una colección de paquetes que nos permite instalarlos y cargarlos todos juntos. Esta colección de paquetes se llama *tidyverse*. El primer paquete que vamos a ver es `readr`. Si aún no hemos instalado *tidyverse*:

```
1 install.packages("tidyverse")
```

Si ya lo tenemos instalado, solo tenemos que “llamarlo” o cargarlo. Cada vez que abrimos una sesión de R hay que cargar los paquetes que vayamos a usar durante la sesión. No es necesario volver a instalarlos pero sí hay que cargarlos usando la función `library()`:

```
1 #Library. Al principio de nuestro scrip creamos un apartado donde iremos cargando los paquetes  
  nos irán haciendo falta.  
2 library(readr) #Si solo quisieramos cargar el paquete readr.
```

```
1 #Library. Al principio de nuestro scrip creamos un apartado donde iremos cargando los paquetes  
  nos irán haciendo falta.  
2 library(tidyverse) #De esta forma, todos los paquetes que agrupa tidyverse: ggplot2, readr, dplyr  
  , etc. Yo usaría esta.
```

```
1 ## — Attaching packages ————— tidyverse 1.3.1 —
```

```
1 ## v ggplot2 3.3.6      v purrr    0.3.5
2 ## v tibble   3.1.8      v dplyr    1.0.10
3 ## v tidyverse 1.2.1     v stringr  1.4.1
4 ## v readr    2.1.3      v forcats  0.5.1
```

```
1 ## — Conflicts ————— tidyverse_conflicts() —
2 ## x dplyr::filter() masks stats::filter()
3 ## x dplyr::lag()   masks stats::lag()
```

Vemos que al cargar `tidyverse` nos informa de qué paquetes han sido cargados y algunas funciones que tienen conflicto con otros paquetes. Una vez cargado el paquete, la primera función que vamos a necesitar es `read_csv()`. Esta función nos permite importar tablas que se han guardado en formato `.csv`. Al principio de la práctica, creamos un proyecto de R, descargamos los datos de la plataforma y los guardamos en la carpeta Datos de nuestro directorio. Si guardasteis vuestro espacio de trabajo al salir, tendréis un objeto llamado `dt1`. Pero si no guardasteis, tendremos que importar los datos de nuevo y, además, aprovechamos para ponerle un nombre que nos sea más evocador que `dt1`, usaremos, por ejemplo, `Crystal`:

```
1 Crystal <- read_csv("Datos/Datos_Crystal.csv")
```

```
1 ## Rows: 1049068 Columns: 15
2 ## — Column specification —————
3 ## Delimiter: ","
4 ## chr (2): flag_chlorophylla, flag_opt_dosat_raw
5 ## dbl (9): year4, daynum, sample_time, depth_calculated, wtaer_temp, pH, chlo...
6 ## lgl (3): flag_water_temp, flag_ph, flag_do2
7 ## dttm (1): sampledate
8 ##
9 ## i Use `spec()` to retrieve the full column specification for this data.
10## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Al importar el archivo, la función reconoce automáticamente el tipo de elementos que se guarda en cada columna. Por ejemplo, la columna `sampledate` es de tipo fecha, mientras que `year` y `depth_calculated` son doubles que vienen a ser valores numéricos no enteros (double no es más que la forma en la que ese número se expresa en forma binaria usando 64 bits).

No os preocupéis por el aviso (warnings) que nos da, nos avisa de que esperaba encontrar un tipo de valor en una columna que no se corresponde con lo que ha entrado. A nosotros no nos preocupa la información que hay en esa columna ¹.

Vamos a ver un resumen de la información que tenemos disponible en la tabla de datos.

```
1 #Vistazo general
2 summary(Crystal)
```

¹Si no sabéis que información guarda esa columna, acordaos de que podéis mirar en los metadatos de la página donde descargasteis los datos.

```

1 ##    sampledate          year4      daynum
2 ## Min.   :2011-04-29 17:54:00.00  Min.   :2011   Min.   : 82.0
3 ## 1st Qu.:2011-10-30 02:06:45.00  1st Qu.:2011   1st Qu.:161.0
4 ## Median :2012-09-26 16:19:30.00  Median :2012   Median :208.0
5 ## Mean    :2013-01-11 09:27:48.36  Mean    :2012   Mean    :209.3
6 ## 3rd Qu.:2013-11-08 16:38:15.00  3rd Qu.:2013   3rd Qu.:257.0
7 ## Max.   :2014-11-04 06:42:00.00  Max.   :2014   Max.   :323.0
8 ##
9 ##    sample_time      depth_calculated   wtaer_temp     flag_water_temp
10 ##   Min.   : 0           Min.   : 0.000   Min.   : 0.00   Mode:logical
11 ##  1st Qu.: 600        1st Qu.: 1.000   1st Qu.: 9.79   NA's:1049068
12 ##  Median :1203        Median : 1.000   Median :16.77
13 ##  Mean   :1183        Mean   : 4.316   Mean   :15.76
14 ##  3rd Qu.:1803        3rd Qu.: 7.000   3rd Qu.:21.32
15 ##  Max.   :2359        Max.   :19.000   Max.   :27.41
16 ##  NA's   :758072
17 ##    pH            flag_ph       chlorophylla  flag_chlorophylla
18 ##   Min.   : 0.000   Mode:logical  Min.   : 0.00   Length:1049068
19 ##  1st Qu.: 7.390   NA's:1049068  1st Qu.: 1.59   Class  :character
20 ##  Median : 7.930
21 ##  Mean   : 8.028
22 ##  3rd Qu.: 8.660
23 ##  Max.   :12.020
24 ##  NA's   :76577
25 ##    opt_dosat_raw  flag_opt_dosat_raw
26 ##   Min.   : 1.00   Mode:logical  Min.   : 1.00   Length:1049068
27 ##  1st Qu.: 90.30  NA's:1049068  1st Qu.: 90.30  Class  :character
28 ##  Median :100.50
29 ##  Mean   : 94.27
30 ##  3rd Qu.:104.60
31 ##  Max.   :148.50
32 ##  NA's   :10065

```

R nos muestra para cada columna los valores mínimos, máximos, la media, la mediana y los cuartiles 1 y 3. Además, también nos dice cuantos valores faltan (NA's). Recordad que podéis usar la función View() para ver los datos directamente, también podéis hacer doble click sobre el objeto dt1.

5.2 Organizar tablas

Una cosa está clara, tenemos mucha información y, además, el archivo es bastante pesado, casi 140 MB. Vamos a centrarnos sólo en un año, 2012 por ejemplo. Para ello vamos a tirar mano de otro paquete: dplyr. Este es otro de los que ya viene preparado con tidyverse, por lo que tendremos que cargarlo. Este paquete es muy útil para organizar, filtrar y transformar la información de las tablas, aquí os dejo un “chuleta” con las funciones principales: [chuleta dplyr](#). Como queremos filtrar los datos y quedarnos solo con los del año 2012, vamos a usar la función filter().

```

1 #Nos quedamos sólo con los datos del 2012
2 Crystal_2012 <- filter(Crystal, year4 == 2012)
3 rm(Crystal) #Aprovechamos para eliminar el otro objeto que no nos va a servir.

```

Además, si no os fijasteis antes, el nombre de la columna que recoge la temperatura del agua está mal escrito, pone wtaer_temp en lugar de water_temp, como cabría esperar. Vamos a cambiárselo usando el paquete dplyr.

```

1 Crystal_2012 <- rename(Crystal_2012, water_temp = wtaer_temp)

```

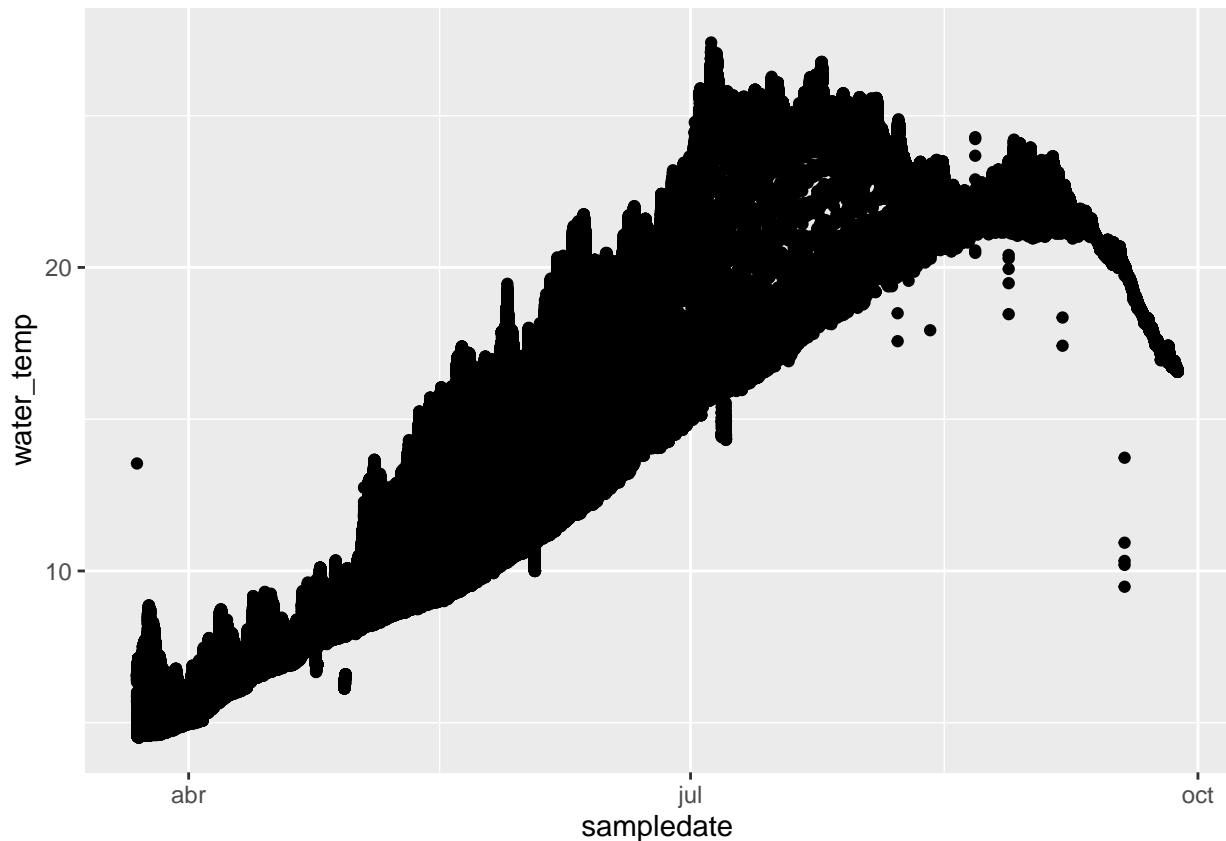
5.2.1 Ejercicios:

1. Filtra la tabla para quedarte con los datos del 2013.
2. Filtra la tabla para quedarte solo con los datos que fueron tomados a un metro de profundidad.

5.3 Gráficas

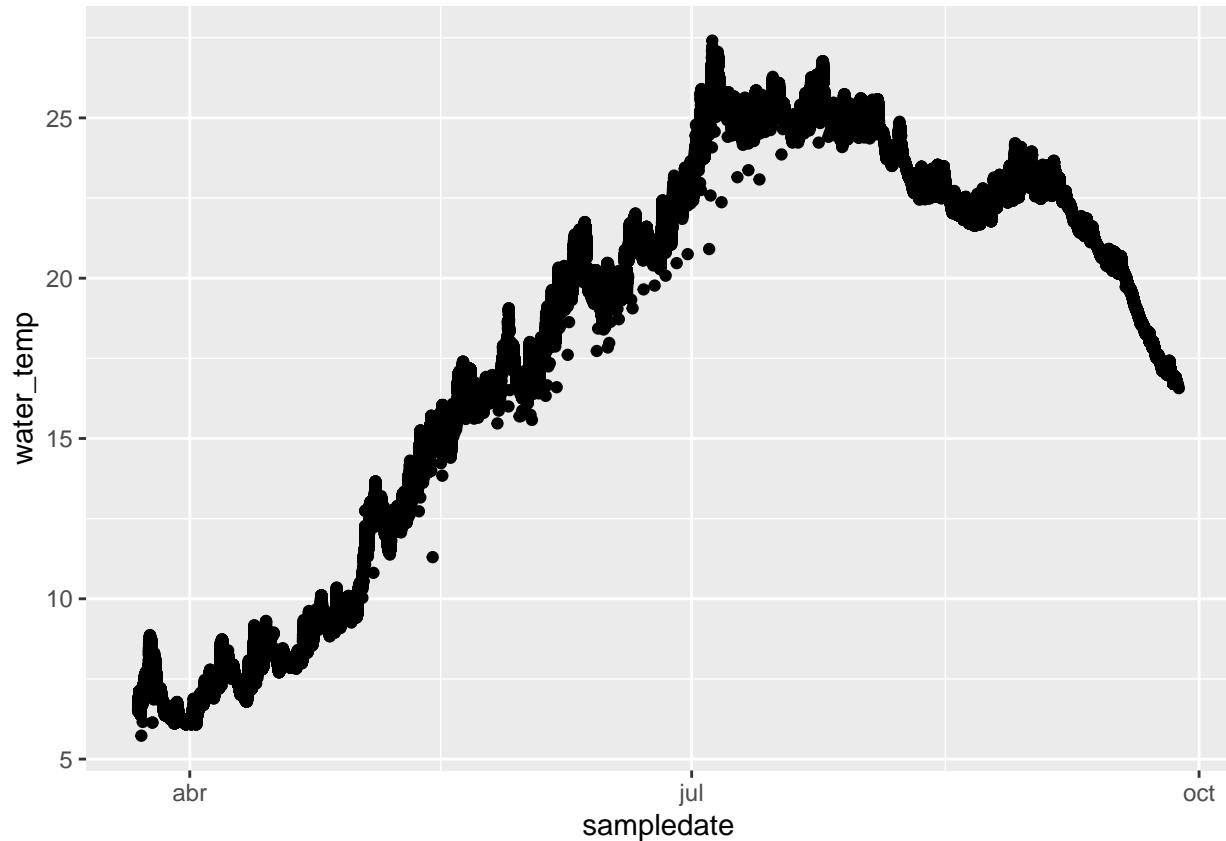
¡Perfecto! Ahora vamos a hacer alguna gráfica. Para ello vamos a usar otro paquete, también incluido en tidyverse, ggplot2. Aquí os dejo otra chuleta para usar este paquete ([pincha aquí](#)). ¿Habrá cambios en la temperatura del agua a lo largo del año? Echemos un vistazo:

```
1 ggplot(Crystal_2012, aes(x = sampledate, y= water_temp))+ #Aquí indicamos donde está la
   información que queremos representar (Crystal_2012) y que variables x e y
2   geom_point() #Aquí indicamos el tipo de gráfico, en este caso hemos elegido puntos
```



Umm... El gráfico es un poco extraño. ¿Y si seleccionamos sólo la temperatura que tenemos a 1 metro de profundidad?

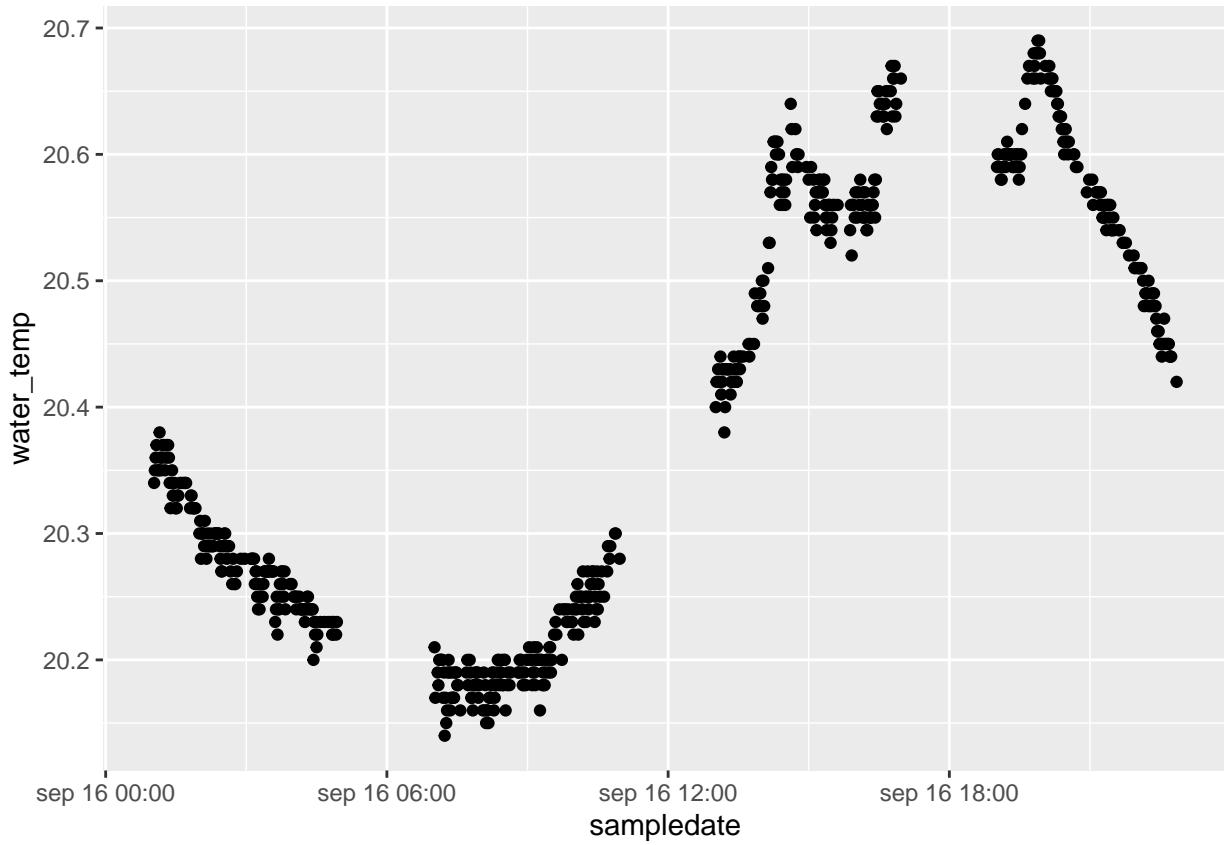
```
1 Crystal_temp_1m <- filter(Crystal_2012, depth_calculated == 1)
2 ggplot(Crystal_temp_1m, aes(x = sampledate, y= water_temp))+
   geom_point()
```



Algo mejor. Vemos que la temperatura aumenta durante la primavera y que vuelve a caer cuando se entra en el otoño. Sin embargo, a parte del patrón general, se puede observar un patrón secundario. Quizás se deba a oscilaciones diarias. Para ver eso tenemos que seleccionar la temperatura a la profundidad de 1 metros y un solo día. Con la función filter podemos hacer eso.

```

1 Temp_1m_1dia <- filter(Crystal_2012, depth_calculated == 1 & daynum == 260) #Yo he elegido el día
   260 (16 de septiembre)
2 ggplot(Temp_1m_1dia, aes(x=sampledate, y = water_temp))+
3   geom_point()
```



Parece que sí, como cabría esperar la temperatura baja por la noche y sube por el día. Además hemos aprendido una cosa nueva en R: los operadores. Os dejo los más comunes:

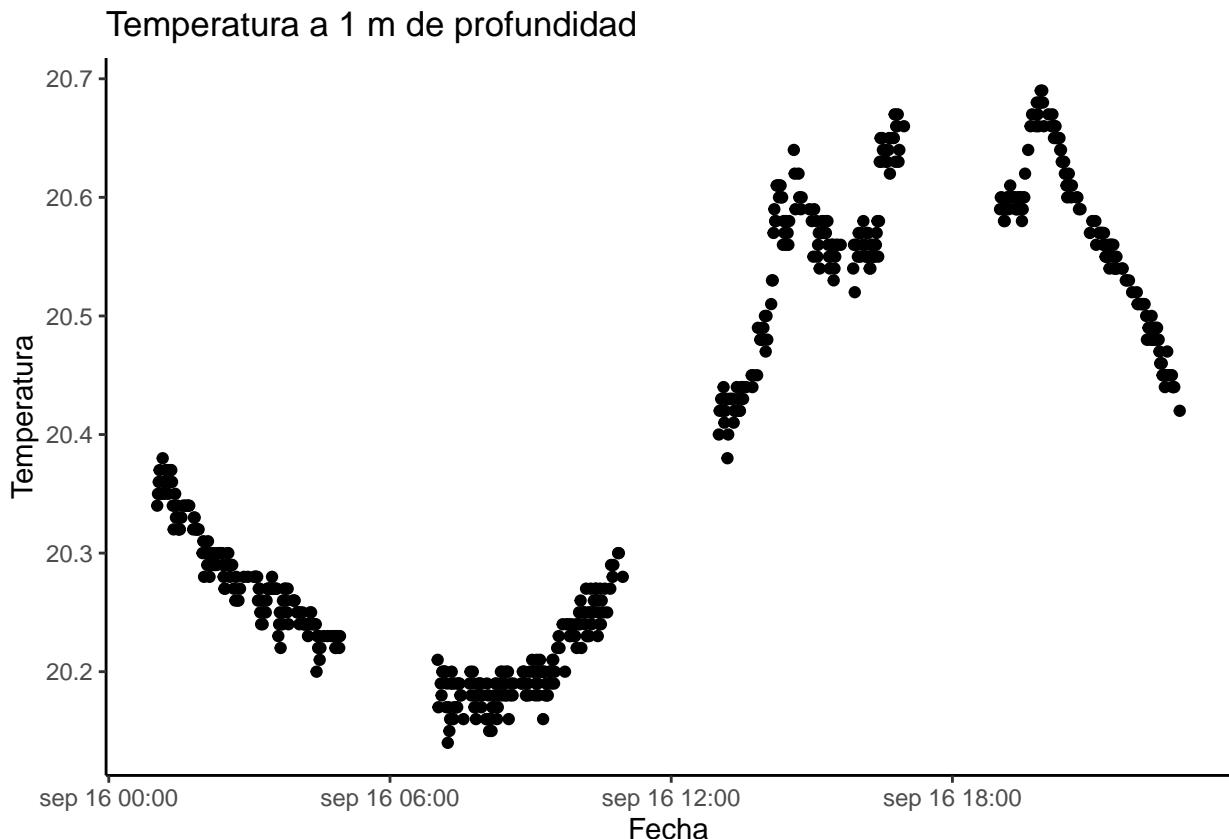
Operador	Descripción
>	mayor que
<	menor que
>=	mayor o igual que
<=	menor o igual que
==	exactamente igual que
!=	distinto a
x & y	x e y
x y	x o y

Ahora, vamos a hacer algunos arreglos estéticos (cambiar el nombre de los ejes, el fondo)

```

1 ggplot(Temp_1m_1dia, aes(x = sampledate, y= water_temp))+ #Aquí indicamos donde está la
   información que queremos representar (Crystal_2012) y qué variables x e y
2 geom_point() + #Aquí indicamos el tipo de gráfico, en este caso hemos elegido puntos
3 labs(x = "Fecha", y = "Temperatura", title = "Temperatura a 1 m de profundidad") + #Aquí
   podemos modificar las etiquetas del gráfico.
4 theme_classic() #Esta función usa un "tema" predefinido para la gráfica

```

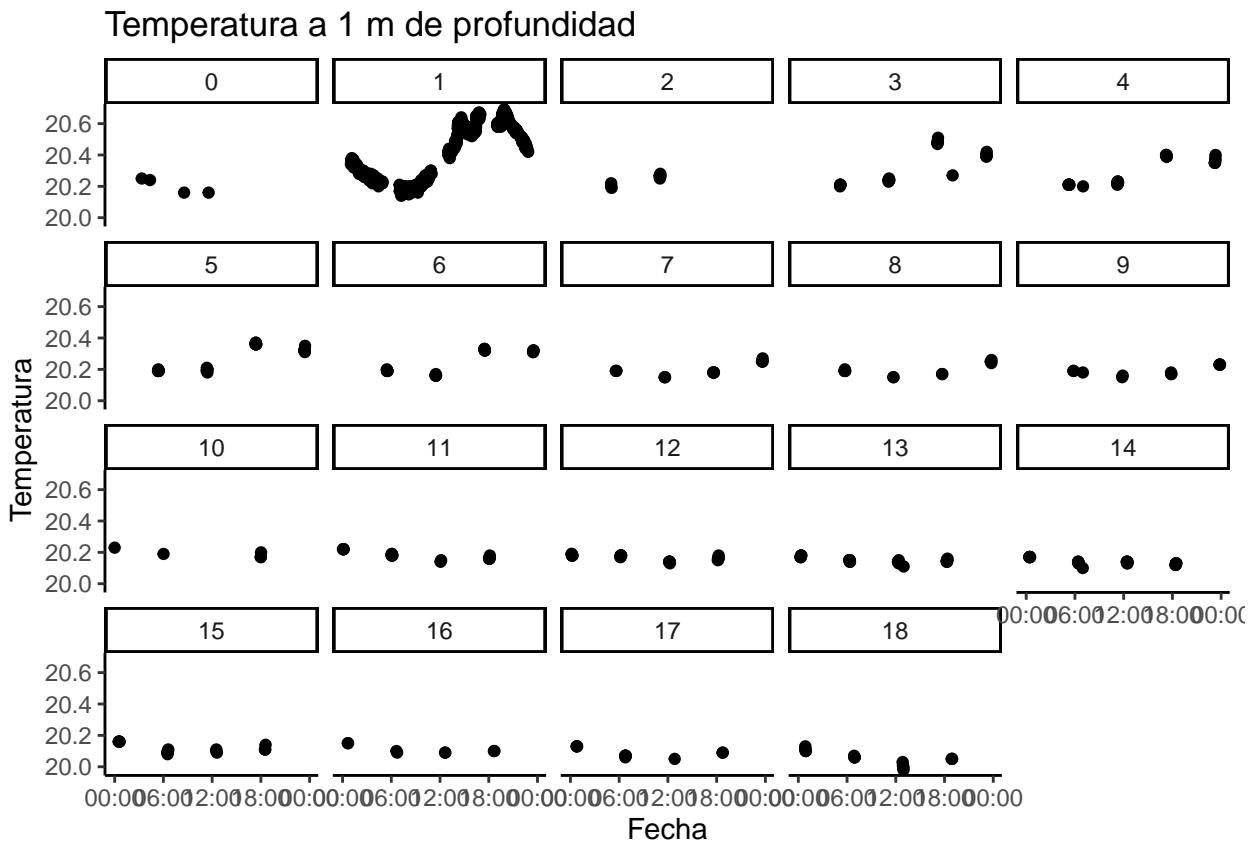


Ya tiene mejor pinta. Vamos a ver si se comporta igual a distinta profundidad. Para ello, solo tenemos que decirle a ggplot que nos haga una gráfica por cada profundidad usando `facet_wrap()`.

```

1 Crystal_temp_dia260 <- filter(Crystal_2012, daynum == 260)
2 ggplot(Crystal_temp_dia260, aes(x = sampledate, y = water_temp))+ #Aquí indicamos donde está la
   información que queremos representar (Crystal_2012) y que variables x e y
3 geom_point() + #Aquí indicamos el tipo de gráfico, en este caso hemos elegido puntos
4 labs(x = "Fecha", y = "Temperatura", title = "Temperatura a 1 m de profundidad") + #Aquí
   podemos modificar las etiquetas del gráfico.
5 facet_wrap(~depth_calculated)+ #En esta línea le indicamos que haga una gráfica por cada
   profundidad.
6 scale_x_datetime(date_labels = "%R") + #He añadido esta línea para que el formato de la fecha
   ponga solo la hora.
7 theme_classic() #Esta función usa un "tema" predefinido para la gráfica

```



Aquí podemos ver varias cosas interesantes. Por un lado, la temperatura parece oscilar a lo largo del día (día-noche) sólo en las capas más superficiales pero en las capas profundas la variación es mínima ¿alguna idea? ¿Podría estar la masa de agua estratificada? Lo veremos más adelante... Por otro lado, vemos que la cantidad de información (puntos) no es la misma en cada profundidad, a 1 metros tenemos muchos más registros de temperatura que a 10 o 12 metros. Vamos a comprobar si es algo general para todos los días:

```

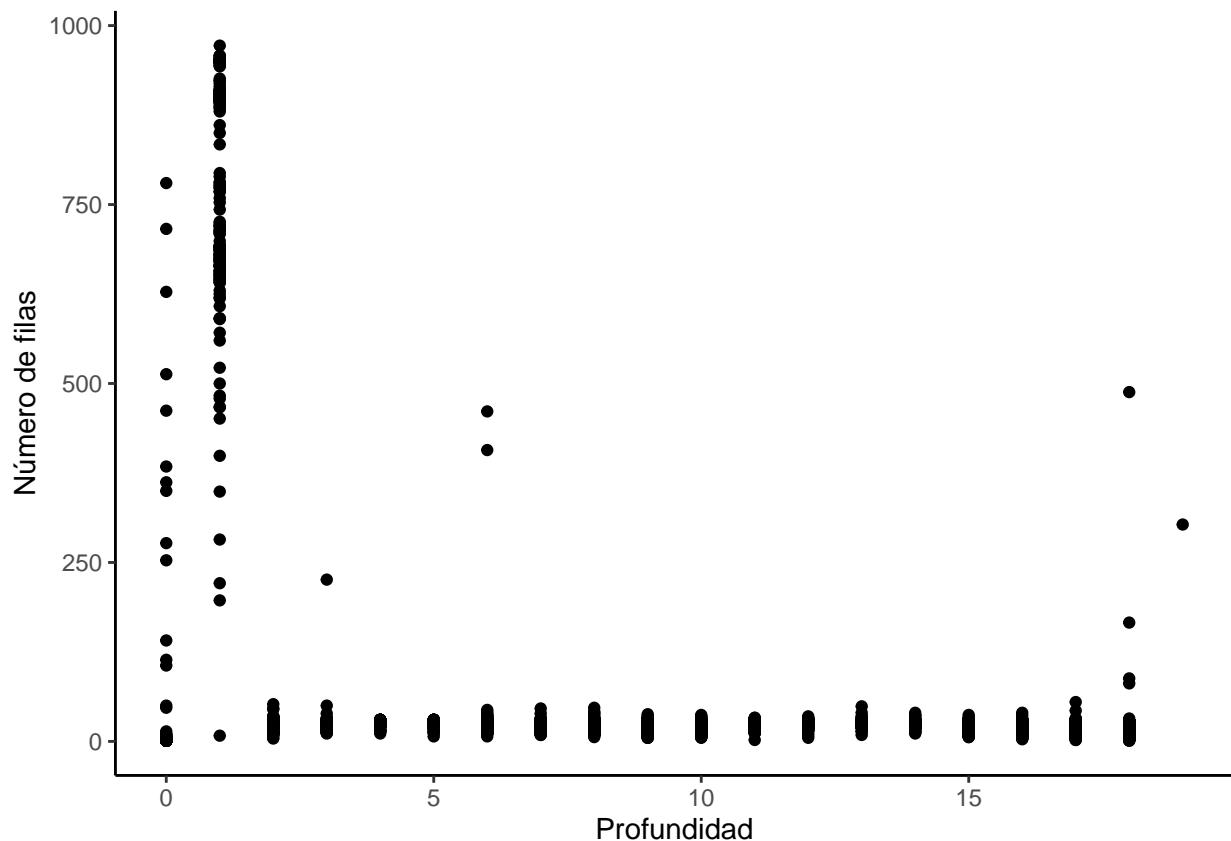
1 #Contamos el número de filas que hay para cada profundidad y día
2 Test_profundidades <- Crystal_2012 %>% group_by(daynum, depth_calculated) %>% summarise(n_filas =
    n())
  
```

```

1 ## `summarise()` has grouped output by 'daynum'. You can override using the
2 ## `.`groups` argument.
  
```

```

1 #Lo representamos en una gráfica
2 ggplot(Test_profundidades, aes(x = depth_calculated, y = n_filas))+ 
  geom_point()+
  labs(x = "Profundidad", y = "Número de filas")+
  theme_classic()
  
```



Vemos que en las profundidades de 0 metros y 1 metro hay muchos más registros que en el resto. Por lo tanto, la variabilidad diaria puede no estar igual de bien recogida para todas las profundidades. En la siguiente sección vamos a seleccionar una franja horaria y a obtener un valor medio por profundidad.

En el código que acabamos de ejecutar habréis encontrado algunos elementos nuevos. Tres funciones nuevas, `group_by()`, `summarise()` y `n()`, que nos permiten agrupar en función de las variables que queramos, resumir la información de la tabla y contar el número de elementos que hay en un grupo, respectivamente. Y un nuevo operador denominado “pipe”, `%>%`. El operador `%>%` nos permite concatenar funciones de una manera más fácil de visualizar. Lo vemos con un ejemplo más sencillo:

```

1 #Si queremos seleccionar la variable temperatura pero sólo de la profundidad de 1 metro
2 #Puedo hacerlo incluyendo funciones dentro de funciones:
3 select(filter(Crystal_2012, depth_calculated == 1), daynum)

```

```

1 ## # A tibble: 149,282 x 1
2 ##   daynum
3 ##   <dbl>
4 ##     1     82
5 ##     2     82
6 ##     3     82
7 ##     4     82
8 ##     5     82
9 ##     6     82
10 ##    7     82
11 ##    8     82
12 ##    9     82
13 ##   10     82
14 ## # ... with 149,272 more rows

```

```

1 #O concatenandolas:

```

```
2 | Crystal_2012 %>% filter(depth_calculated == 1) %>% select(daynum)
```

```
1 ## # A tibble: 149,282 x 1
2 ##   daynum
3 ##   <dbl>
4 ##     1     82
5 ##     2     82
6 ##     3     82
7 ##     4     82
8 ##     5     82
9 ##     6     82
10 ##    7     82
11 ##    8     82
12 ##    9     82
13 ##   10     82
14 ## # ... with 149,272 more rows
```

Esta última es más fácil de visualizar que se está haciendo en cada paso y, por lo tanto, facilita la lectura del código.

5.3.1 Ejercicios

1. Representa otra variable, la que a tí te apetezca. Ej: oxígeno disuelto.
2. Prueba, en lugar de `theme_classic()`, otro tema diferente: por ejemplo: `theme_ligh()`.
3. Prueba a usar `geom_line()` en lugar de `geom_point()`.
4. ¿Y si en lugar de "%R" usáis "%H"? Ejecuta `?strptime` y mira los formatos disponibles. Experimenta con ellos.
5. Si ya estas muy mosquead@ con R (cosa que entiendo perfectamente), prueba a abrir la tabla con excel y representar la temperatura en el lago Crystal en cada profundidad para el día 260 del año 2012. ¿Cuánto tiempo te ha llevado?

5.4 Trabajar con fechas

Como dijimos en la sección anterior, vamos a obtener un valor medio de temperatura por profundidad y día. Para trabajar con fechas, horas, etc. en R tenemos un paquete que nos facilita la vida: `lubridate`. Para este paquete también disponemos de “chuleta”, [aquí os la dejo](#). Este paquete, aunque también se instala con tidyverse, hay que cargarlo de manera independiente.

```
1 library(lubridate)
```

```
1 ##
2 ## Attaching package: 'lubridate'
```

```
1 ## The following objects are masked from 'package:base':
2 ##
3 ##     date, intersect, setdiff, union
```

En la columna sampledate tenemos recogida la información de la fecha y la hora a la que se tomó la medida. Con el paquete lubridate podemos extraer esta información de manera sencilla:

```

1 date(Crystal_2012$sampleddate) #Nos da la fecha
2 hour(Crystal_2012$sampleddate) #Nos da la hora
3 minute(Crystal_2012$sampleddate) #Nos da los minutos
4 yday(Crystal_2012$sampleddate) #el día del año

```

Con esta pequeña presentación y uniendo lo que hemos visto hasta ahora, quizás, podríamos intentar calcular la temperatura media de cada profundidad para cada día teniendo en cuenta sólo los valores medidos entre las 11:00 de la mañana y las 16:00 de la tarde.

Vamos a intentarlo:

```

1 #nos quedamos sólo con las muestras tomadas entre las 11:00 y las 16:00
2 Crystal_dia <- Crystal_2012 %>% filter(hour(sampleddate) >= 11 & hour(sampleddate) <= 16)
3 #Calculamos la temperatura media por día y profundidad
4 Crystal_dia %>% group_by(daynum, depth_calculated) %>% summarise(mean(water_temp))

```

```

1 ## `summarise()` has grouped output by 'daynum'. You can override using the
2 ## `.`groups` argument.

```

```

1 ## # A tibble: 3,365 x 3
2 ## # Groups:   daynum [189]
3 ##   daynum depth_calculated `mean(water_temp)` 
4 ##   <dbl>      <dbl>          <dbl>
5 ##     1        82            0       8.32
6 ##     2        82            1       6.54
7 ##     3        82            2       6.41
8 ##     4        82            3       5.92
9 ##     5        82            4       5.74
10 ##    6        82            5       5.66
11 ##    7        82            6       5.51
12 ##    8        82            7       5.30
13 ##    9        82            8       5.14
14 ##   10        82            9       5.03
15 ## # ... with 3,355 more rows

```

¡Perfecto! Ya lo tenemos! Si le echáis un vistazo a la chuleta que os dejé del paquete `dplyr`, hay dos funciones que nos pueden resultar interesantes: `between()` y `summarise_if()`. La primera nos simplifica la vida cuando queremos filtrar dentro de un rango determinado y la segunda nos permite calcular la media de todas las variables numéricas de una vez.

5.4.1 Ejercicios

1. Intenta aplicar la función `between()` para filtrar tabla y quedarnos solo con los valores que están entre las 11:00 y las 16:00.
2. Intenta aplicar la función `summarise_if()` para calcular la media, no sólo de la temperatura del agua, si no de todas las variables numéricas que tenemos.

Capítulo 6

Estabilidad térmica de la columna de agua

Llegados a este punto de la práctica ya podemos adentrarnos en cuestiones más puramente limnológicas. Hemos echado un vistazo a los datos que descargamos y decidimos seleccionar el año 2012. Además, para el eliminar las variaciones diarias y centrarnos en una tendencia estacional, hemos obtenido un perfil promedio del lago Crystal para cada día. Si en el apartado anterior no conseguisteis hacer el promedio diario para todas las variables os dejo el código aquí para que partamos tod@s desde el mismo punto:

```
1 #Library
2 library(tidyverse) #Recordad siempre cargar los paquetes que vais a necesitar al inicio del
3 # script
4 #Lo he recogido todo en una sola línea.
5 Crystal_dia <- Crystal_2012 %>% filter(between(hour(sampledate), 11, 16)) %>% group_by(daynum,
6 depth_calculated) %>% summarise_if(is.numeric, mean, na.rm = TRUE)
7 #Exportamos los datos
8 write_csv(Crystal_dia, "Datos/Crystal_dia.csv") #Guardamos la información en un archivo
```

6.1 Perfil vertical

Todavía no hemos visualizado lo que acabamos de hacer. Vamos a ver como ha quedado la tabla.

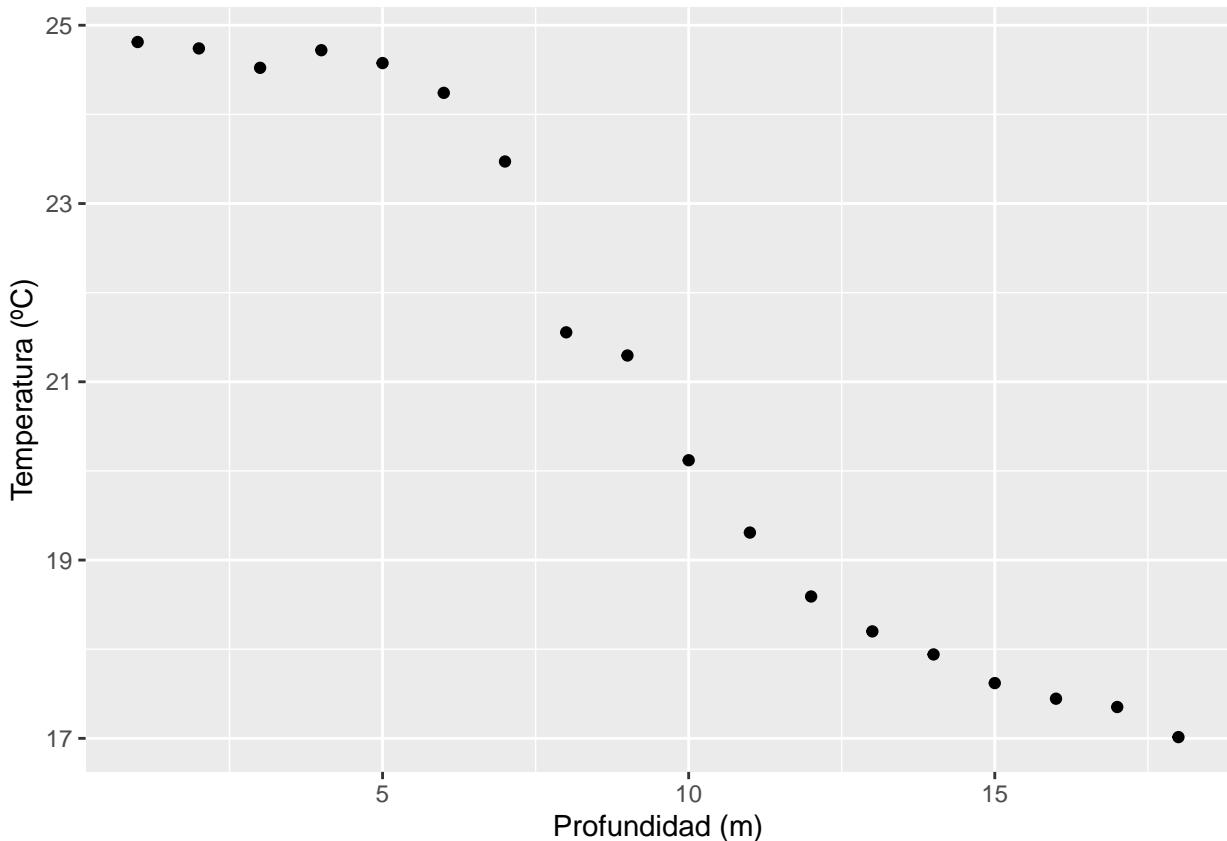
```
1 Crystal_dia
```

```
1 ## # A tibble: 3,365 x 9
2 ##   daynum depth_calculated year4 sample_~1 water~2     pH chlor~3 opt_d~2 opt_d~4
3 ##   <dbl>          <dbl> <dbl> <lgl>    <dbl> <dbl> <dbl> <dbl> <dbl>
4 ## 1     82            0  2012 NA      8.32  7.59  1.06  9.63  87.3
5 ## 2     82            1  2012 NA      6.54  6.56  1.24  9.73  84.5
6 ## 3     82            2  2012 NA      6.41  6.27  1.35  9.71  84.0
7 ## 4     82            3  2012 NA      5.92  6.10  1.81  9.57  81.8
8 ## 5     82            4  2012 NA      5.74  6.10  2.07  9.49  80.7
9 ## 6     82            5  2012 NA      5.66  6.37  2.31  9.44  80.2
10 ## 7     82            6  2012 NA      5.51  6.34  2.63  9.38  79.3
11 ## 8     82            7  2012 NA      5.30  6.34  2.86  9.28  78.1
12 ## 9     82            8  2012 NA      5.14  6.32  2.95  9.08  76.1
13 ## 10    82            9  2012 NA      5.03  6.38  3.31  8.95  74.8
14 ## # ... with 3,355 more rows, and abbreviated variable names 1: sample_time,
15 ## #   2: water_temp, 3: chlorophylla , 4: opt_dosat_raw
```

Tiene justo lo que queríamos. Vamos a representar el perfil vertical de temperatura para un día concreto.

```

1 #Elegimos un día
2 Dia_elegido <- Crystal_dia %>% filter(daynum == 200)
3
4 #Representamos
5 ggplot(Dia_elegido, aes(x= depth_calculated , y = water_temp))+ 
6   geom_point()+
7   labs(x = "Profundidad (m)" , y = "Temperatura (°C)")
```



Tiene buena pinta pero es extraño. Nosotr@s estamos acostumbrados a ver los perfiles así:

Pues vamos a hacer algunos cambios en el código para mejorar esto.

```

1 ggplot(Dia_elegido, aes(y= depth_calculated , x = water_temp))+ 
2   geom_point()+
3   scale_y_reverse() + # Invertimos la profundidad para que la parte superior sea 0 metros.
4   scale_x_continuous(position = "top") + #Colocamos el eje arriba
5   labs(x = "Temperatura (°C)" , y = "Profundidad (m)")+
6   theme_classic()
```

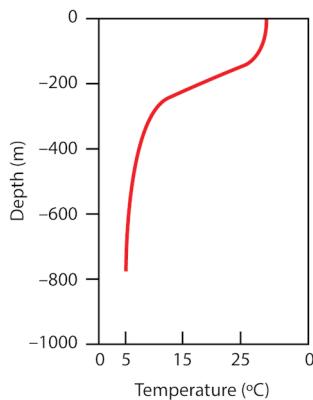
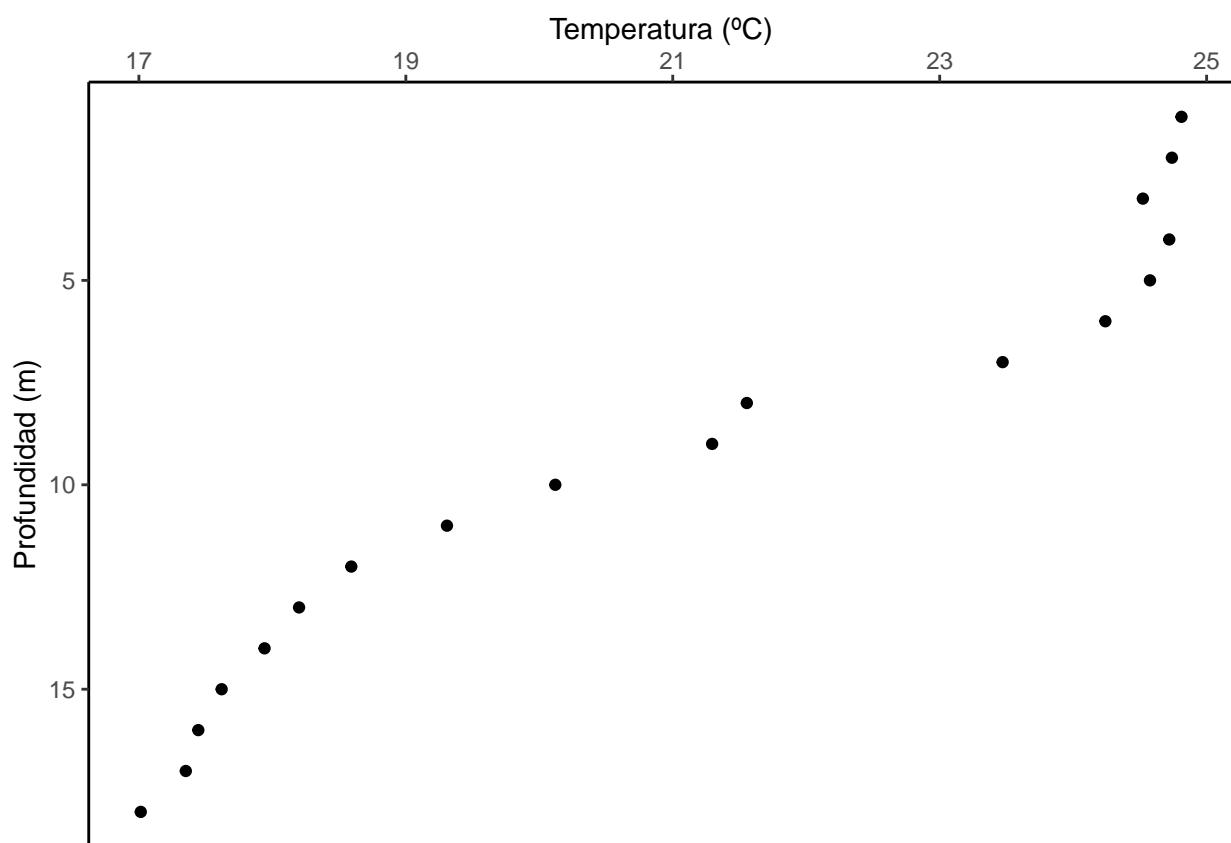


Figura 6.1: Representación clásica de un perfil vertical de temperatura.

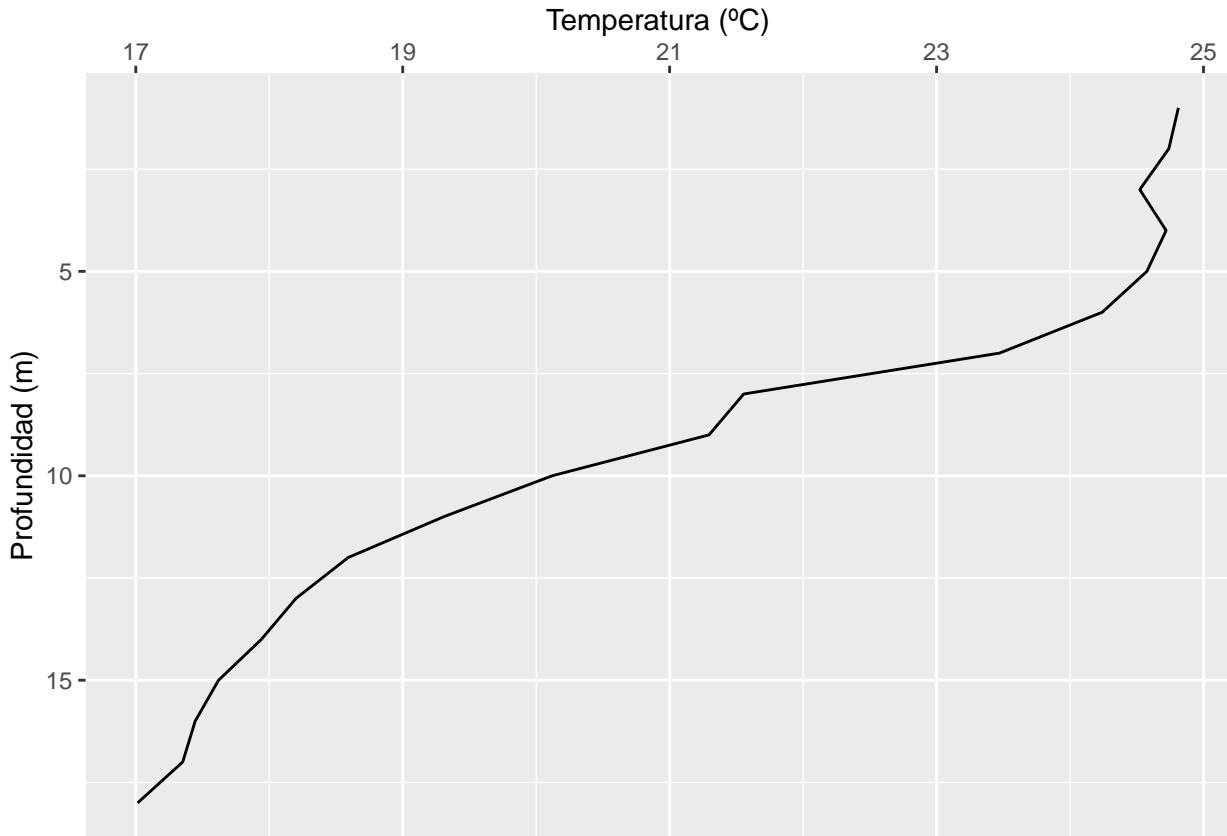


Esto tiene mejor pinta. Prueba a usar `geom_line()` en lugar de `geom_point()`. Sucede algo extraño entorno a los 5 metros de profundidad ¿verdad? Esto sucede porque `geom_line()` une las observaciones ordenadas por la variable x. Prueba mejor con `geom_path()`. Otra opción es asignar la profundidad al eje x y cambiar los ejes (es un poco lioso, os dejo el código por si queréis indagar).

```

1 ggplot(Dia_elegido, aes(x= depth_calculated , y = water_temp))+
2   geom_line()+
3   coord_flip() + #Cambia el eje x por el y
4   scale_x_reverse() + # Invertimos la profundidad para que la parte superior sea 0 metros.
5   scale_y_continuous(position = "right")+
6   labs(x = "Profundidad (m)", y = "Temperatura (°C)")

```



Quitando esta curiosidad con respecto a como se representan las variables, ya tenemos nuestro perfil vertical de temperatura:

```

1 ggplot(Dia_elegido, aes(y= depth_calculated, x = water_temp))+  

2   geom_path(color = "gray", size = 3)+ #Cambiamos un poco la estética dándole color y grosor a la  

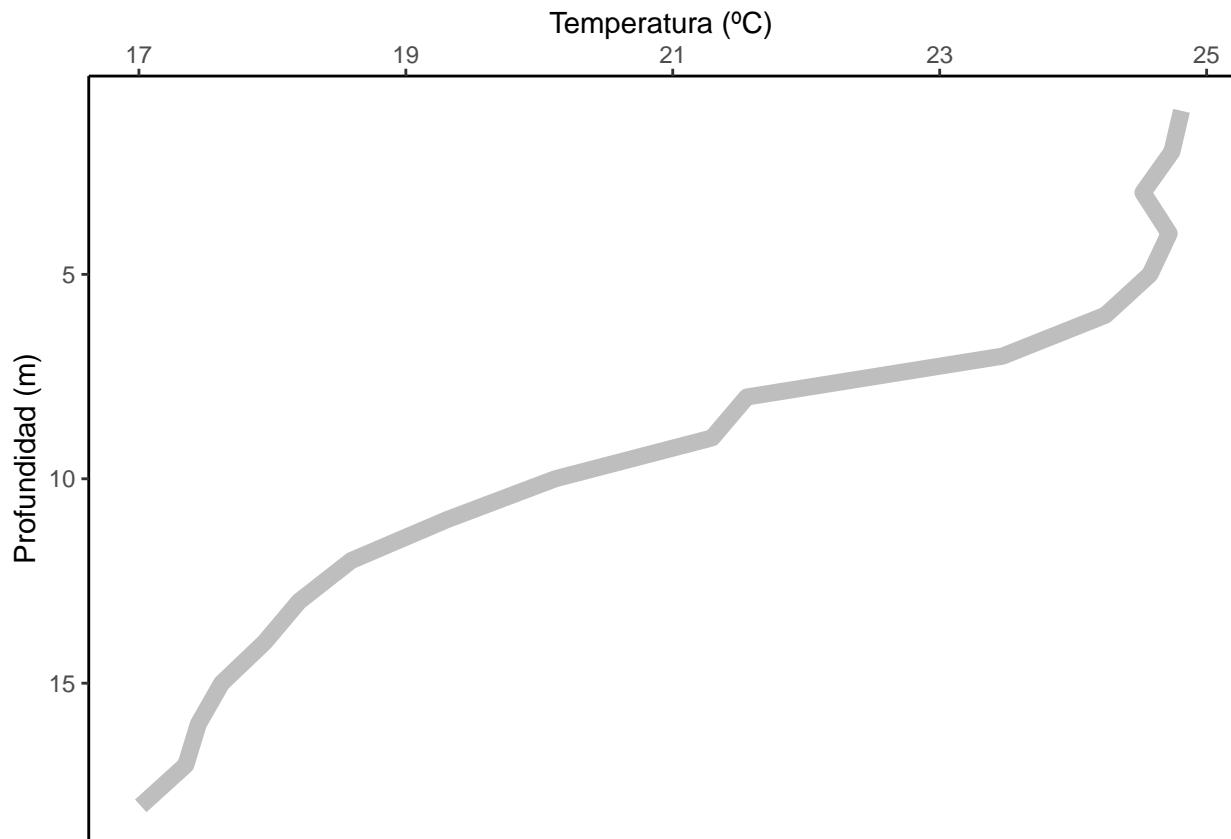
3   línea  

4   scale_y_reverse() + # Invertimos la profundidad para que la parte superior sea 0 metros.  

5   scale_x_continuous(position = "top") + #Colocamos el eje arriba  

6   labs(x = "Temperatura (°C)", y = "Profundidad (m)")+  

    theme_classic()
  
```



6.1.1 Ejercicio

1. Haz un perfil con la concentración de oxígeno disuelto y otro con el pH.

6.2 Termoclina

Ahora, en base a el perfil de temperatura que hemos representado, ¿sabrías si este día el lago está estratificado? Si estuviera estratificado, ¿serías capaz de decir a que profundidad se encuentra la termoclina?

A ojo podríamos acercarnos bastante pero imaginad que luego quisiéramos calcularlo para cada día... Gracias al altruismo y la ciencia colaborativa tenemos a nuestra disposición un paquete desarrollado especialmente para este tipo de cosas que nos gustan a los limnolog@s. El paquete es [rLakeAnalyzer](#) y contiene métodos estandarizados para calcular propiedades físicas del lago, como la termoclina, la estabilidad de Schmidt, etc. Para instalarlo: `install.packages("rLakeAnalyzer")`. Si echáis un vistazo al paquete veréis un montón de funciones interesantes. Nosotros, ahora mismo, queríamos saber a que profundidad tenemos la termoclina. Para eso, vamos a pedir ayuda para aprender a usar la función `?thermo.depth()`. Aquí podemos ver los argumentos que necesitamos. Por ejemplo, podemos indicarle la densidad mínima para considerar que existe termoclina o la temperatura por debajo de la cual no queremos que calcule termoclina. No vamos a reparar en estos. Sin embargo, el argumento `seasonal` viene por defecto como verdadero, situando la termoclina en el gradiente de densidad más profundo. Nosotros vamos a cambiarlo a FALSE. Estos son argumentos opcionales, los indispensables son un vector numérico con la temperatura del agua (`wtr`) y otro vector numérico con la profundidad que corresponde a cada temperatura. Estos los tenemos, así que nos podemos poner manos a la obra:

```

1 #Library
2 library(rLakeAnalyzer)
3 #Creamos dos vectores con la información que nos pide
4 Temp <- Dia_elegido$water_temp
5 Prof <- Dia_elegido$depth_calculated
6

```

```

7 | #Calculamos la termoclinia
8 | thermo.depth(wtr = Temp, depths = Prof, seasonal = FALSE)

```

```

1 | ## [1] 7.397819

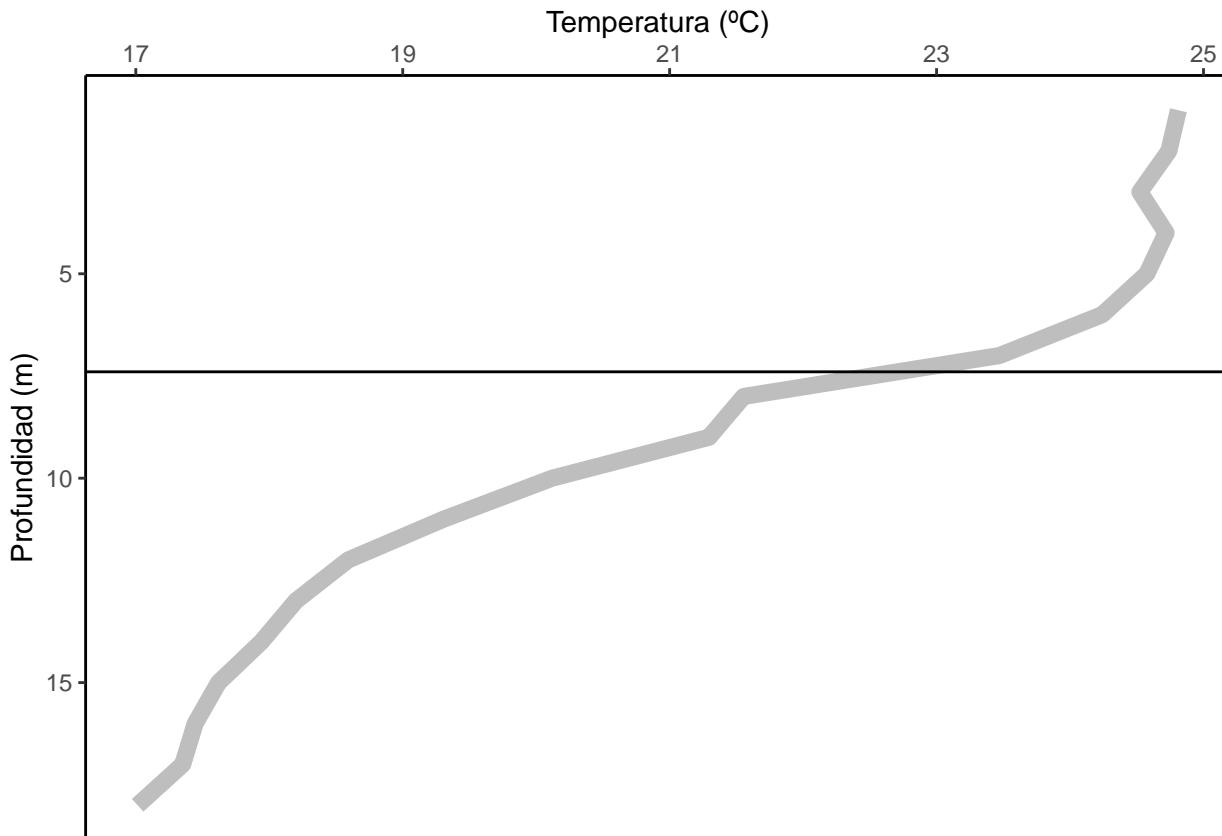
```

¡Voilá! Ya tenemos la profundidad a la que está la termoclinia. Vamos a representarla:

```

1 | #Vamos a guardar la profundidad de la termoclinia en un objeto
2 | Termoclinia <- thermo.depth(wtr = Temp, depths = Prof, seasonal = FALSE)
3 |
4 | ggplot(Dia_elegido, aes(y= depth_calculated , x = water_temp))+
5 |   geom_path(color = "gray", size = 3)+ #Cambiemos un poco la estética dándole color y grosor a la
6 |   linea
7 |   geom_hline(yintercept = Termoclinia)+ #Añadimos la termoclinia
8 |   scale_y_reverse() + # Invertimos la profundidad para que la parte superior sea 0 metros.
9 |   scale_x_continuous(position = "top") + #Colocamos el eje arriba
10 |  labs(x = "Temperatura (°C)", y = "Profundidad (m)")+
11 |  theme_classic()

```



6.3 Capas en un lago estratificado

Como sabéis, cuando un lago (o un embalse) se encuentra estratificado se puede diferenciar en distintas capas. Una capa superficial, que está mezclada y en contacto con la atmósfera, el **epilimnion**, otra capa profunda, aislada de la superior debido al fuerte gradiente de densidad, el **hipolimnion** y una capa intermedia que hace de interfase entre estas dos capas, el **metalimnion**. Esto determinan los procesos que tendrán lugar a cada profundidad. La capa superficial al estar en contacto con la atmósfera estará bien oxigenada y

dará lugar a que se desarrolle la vida aeróbica sin problemas. La capa profunda, sin embargo, puede acabar volviéndose anóxica y limitando la degradación de la materia orgánica a mecanismos de respiración anaeróbica (desnitrificación, sulfato reducción, etc) y acumulando metales que pueden ser tóxicos y comprometer su uso para abastecimiento humano. Por lo tanto, si estamos estudiando un sistema estratificado es clave reconocer y definir estas capas.

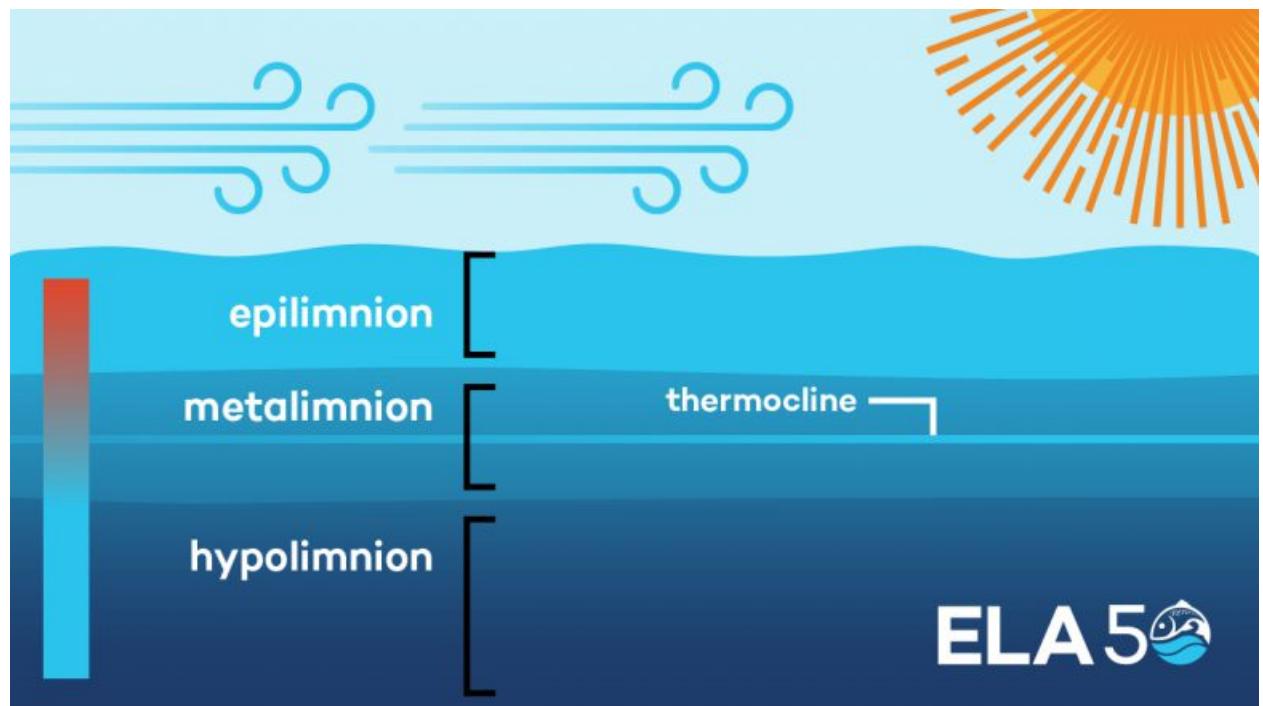


Figura 6.2: Esquema de un lago estratificado. Fuente: IISD Experimental Lakes Area (IISD-ELA).

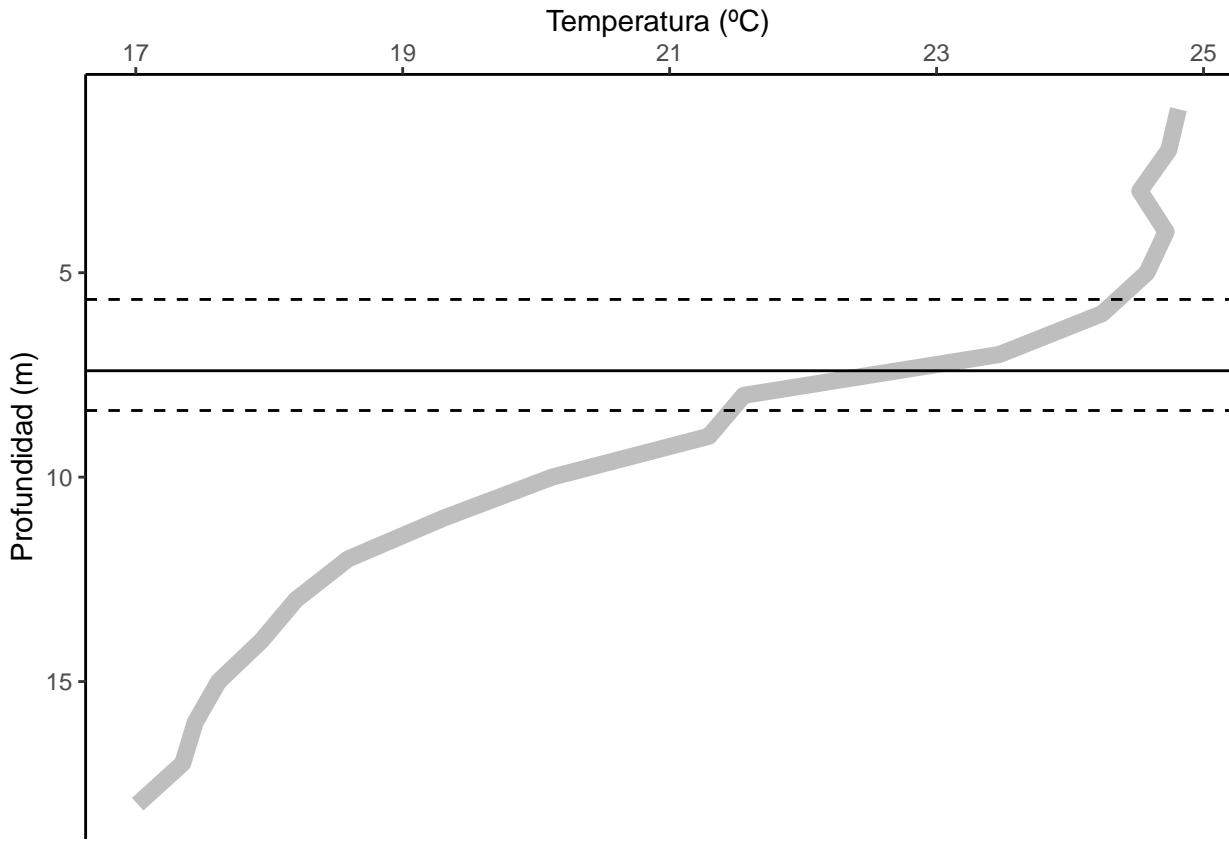
Para determinar donde empiezan y donde acaban estas capas, el paquete rLakeAnalyzer dispone de un función, meta.depths(). Si leemos la información sobre la función vemos que nos devuelve el límite superior e inferior del metalimnion, que a su vez son el límite inferior del epilimnion y el límite superior del hipolimnion. Además, los argumentos obligatorios son los mismo que usamos con thermo.depth(). Así que podemos aplicarla sin problemas.

```
1 meta.depths(wtr = Temp, depths = Prof, seasonal = FALSE)
```

```
1 ## [1] 5.654340 8.370172
```

¡Perfecto! Vamos a representarlo todo junto.

```
1 #Guardo los límites del metalimnion
2 metalimnion <- meta.depths(wtr = Temp, depths = Prof, seasonal = FALSE)
3
4 ggplot(Dia_elegido, aes(y= depth_calculated , x = water_temp))+
  geom_path(color = "gray", size = 3)+ #Cambiemos un poco la estética dándole color y grosor a la
  #línea
  geom_hline(yintercept = Termocлина)+ #Línea en la termocicina
  geom_hline(yintercept = metalimnion , linetype = 2)+ #Líneas discontinuas defininedo el
  #metalimnion
  scale_y_reverse()# Invertimos la profundidad para que la parte superior sea 0 metros.
  scale_x_continuous(position = "top")#Colocamos el eje arriba
  labs(x = "Temperatura (°C)", y = "Profundidad (m)")+
  theme_classic()
```



6.3.1 Ejercicios

1. Prueba a cambiar el argumento seasonal. ¿Qué sucede?
2. ¿Podrías calcular la profundidad de la termoclina para todos los días del año en el lago Crystal?

6.4 Estabilidad

La estabilidad de la columna de agua nos va a dar una idea sobre su tendencia a mezclarse o permanecer estratificada. Uno de los índices utilizados para representar la estabilidad de la masa de agua en un lago es el índice de estabilidad de Schmidt. Este indice da un valor sobre la energía que tendríamos que aplicar sobre la superficie para mezclar la columna de agua. Las unidades son, por lo tanto, J/m^2 .

El paquete `rLakeAnalyzer` también nos provee de una función para calcular la estabilidad de Schmidt, `schmidt.stability()`. Echar una ojeada a como se usa.

Si os habéis fijado, en este caso, a parte de la temperatura y la profundidad, necesitamos darle información sobre la batimetría del lago. `bthA` es el área que tiene cada capa a una determinada profundidad `bthB`. Quizás lo veis mejor en la siguiente imagen.

Pues bien, a priori, no disponemos de esta información. Quizás buscando en la base de datos de GLEON podríamos encontrar la batimetría del Lago Crystal. De hecho, está pero en un archivo `.shp` y deberíamos calcular el área. Todo esto se puede hacer con R (de hecho, la imagen de arriba está hecha con R) pero no vamos a abrir esa puerta ahora, ya tenemos bastante...

De nuevo, el paquete `rLakeAnalyzer` nos proporciona la solución a nuestros problemas. Existe una función para estimar la curva área-profundidad (también conocida como curva hipsográfica) para un lago. Sólo tenemos

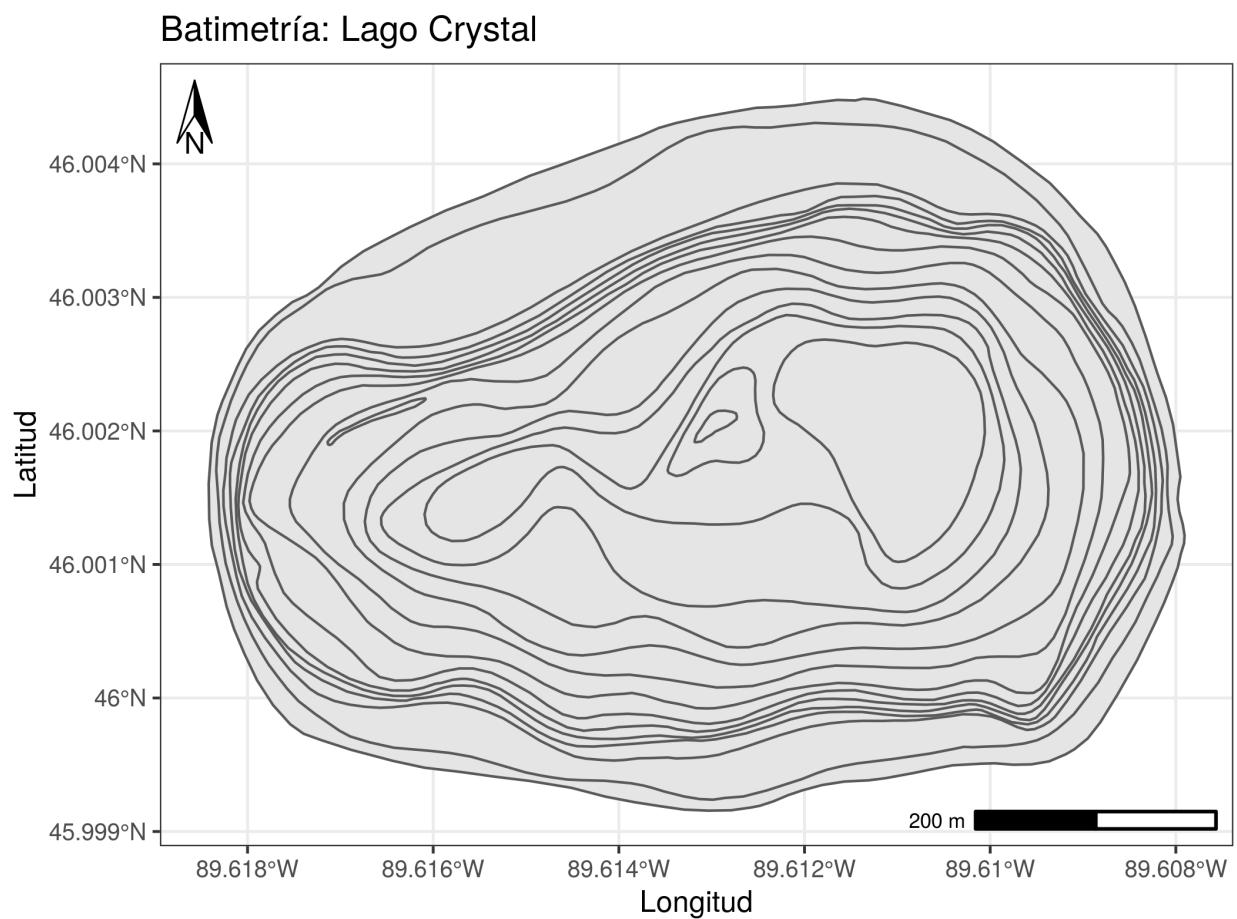


Figura 6.3: Batimetría del lago Crystal.

que proporcionarle el área de la superficie del lago, la profundidad máxima y la profundidad media. Toda esta información la tenemos disponible en la página de NTL-LTER (North Temperate Lakes US Long-Term Ecological Research Network). Podéis pinchar [aquí](#). La profundidad máxima es de 20.4 m, la superficie del lago es de 367000 m² y la profundidad media es de 11.4 m. ¡Al lío!

```
1 #Estimamos la batimetría
2 approx.bathy(Zmax = 20.4, lkeArea = 367000, Zmean = 11.4)
```

```
1 ##      depths   Area.at.z
2 ## 1      0 367000.0000
3 ## 2      1 331901.4802
4 ## 3      2 298566.7051
5 ## 4      3 266995.6747
6 ## 5      4 237188.3891
7 ## 6      5 209144.8481
8 ## 7      6 182865.0519
9 ## 8      7 158349.0004
10 ## 9      8 135596.6936
11 ## 10     9 114608.1315
12 ## 11    10 95383.3141
13 ## 12    11 77922.2414
14 ## 13    12 62224.9135
15 ## 14    13 48291.3303
16 ## 15    14 36121.4917
17 ## 16    15 25715.3979
18 ## 17    16 17073.0488
19 ## 18    17 10194.4444
20 ## 19    18 5079.5848
21 ## 20    19 1728.4698
22 ## 21    20 141.0996
```

Ahora ya tenemos toda la información necesaria para calcular la estabilidad de Schmidt.

```
1 #Guardamos la estimación en un objeto
2 Batimetria <- approx.bathy(Zmax = 20.4, lkeArea = 367000, Zmean = 11.4)
3
4 #Calculamos la estabilidad de Schmidt
5 schmidt.stability(wtr = Temp, depths = Prof, bthA = Batimetria$Area.at.z, bthD = Batimetria$
```

```
1 ##      [,1]
2 ## [1,] 121.69
```

La estabilidad de Schmidt para el día 200 es de 121.69 J/m². Este número nos puede decir poco pero... ¿y si comparamos distintos días?

6.4.1 Ejercicios

1. ¿Para qué sirve el argumento *method* de la función `approx.bathy()`? ¿Qué efecto tiene?
2. Vamos a comparar la estabilidad de Schmidt para tres días: el primero que tenemos de 2012, el día 200 y el último que tenemos para 2012.

6.5 Series temporales

Las funciones que hemos visto hasta ahora del paquete rLakeAnalyzer las hemos aplicado a perfiles vertical de un día en concreto. En uno de los ejercicios os animé a calcularais la termoclina para cada día del año 2012.

Gracias al paquete dplyr esta tarea no es realmente complicada. Sin embargo, el paquete rLakeAnalyzer posee una versión de cada una de las funciones que hemos visto pero que se puede aplicar a series temporales como la que nosotros tenemos. Echad un vistazo a `?ts.thermo.depth`. Si os das cuenta, el único inconveniente en que nos piden una estructura de los datos concreta. Para poder aplicarla tenemos que reorganizar la tabla. La tabla tiene que tener una columna llamada datetime con formato de fecha en lugar de día del año y una columna para cada profundidad. Lo primero que vamos a hacer es quedarnos solamente con la información del día, de la profundidad y de la temperatura del agua.

```
1 #Seleccionamos las variables que nos interesan
2 Crystal_temp <- Crystal_dia %>% select(daynum, depth_calculated, water_temp)
```

Una vez que hemos seleccionado sólo lo que nos interesa, vamos a convertirlo a forma ancho usando la función `spread()`.

```
1 #Mantenemos la fecha y convertimos la profundidad en variables (columnas) incluyendo el valor de
  #temperatura en cada columna.
2 Crystal_ts <- Crystal_temp %>% spread(key = depth_calculated, value = water_temp)
```

Vamos a convertir el día del año en formato fecha y a cambiarle el nombre a las columnas.

```
1 #Convertimos los días del año en fechas
2 Crystal_ts <- Crystal_ts %>% mutate_at(vars(daynum), as.Date, origin = "2012-01-01")
3 #Cambiamos el nombre de la columna "daynum" a "datetime"
4 Crystal_ts <- Crystal_ts %>% rename(datetime = daynum)
5 #Añadimos "wtr_" delante del número de cada profundidad
6 Crystal_ts <- Crystal_ts %>% rename_if(is.numeric, ~paste("wtr_", ., sep = ""))
7 Crystal_ts
```

```
1 ## # A tibble: 189 x 20
2 ##   datetime    wtr_0  wtr_1  wtr_2  wtr_3  wtr_4  wtr_5  wtr_6  wtr_7  wtr_8  wtr_9  wtr_10
3 ##   <date>     <dbl> <dbl>
4 ## 1 2012-03-23  8.32  6.54  6.41  5.92  5.74  5.66  5.51  5.30  5.14  5.03  4.98
5 ## 2 2012-03-24 NA     7.03  6.79  6.23  5.85  5.79  5.61  5.43  5.29  5.13  5.04
6 ## 3 2012-03-25 NA     7.78  7.12  6.32  6.06  5.90  5.78  5.67  5.42  5.28  5.11
7 ## 4 2012-03-26  8.20  8.11  7.99  7.07  6.04  5.88  5.66  5.60  5.43  5.26  5.06
8 ## 5 2012-03-27 NA     7.18  7.04  7     6.05  NA     NA     NA     NA     5.11  5.13
9 ## 6 2012-03-28 NA     6.50  6.35  6.32  6.30  6.26  6.20  6.16  5.98  5.83  5.46
10 ## 7 2012-03-29 NA     6.36  6.34  6.32  6.31  6.31  6.28  6.12  5.81  5.74  5.69
11 ## 8 2012-03-30  6.02  6.60  6.28  6.25  6.22  6.18  6.16  6.15  6.04  5.98  5.86
12 ## 9 2012-03-31 NA     6.23  6.23  6.21  6.18  6.16  6.15  6.13  6.12  6.12  6.06
13 ## 10 2012-04-01  6.18  6.16  6.10  6.08  6.07  6.06  6.06  6.04  6.04  6.04  6.02
14 ## # ... with 179 more rows, and 8 more variables: wtr_11 <dbl>, wtr_12 <dbl>,
15 ## #   wtr_13 <dbl>, wtr_14 <dbl>, wtr_15 <dbl>, wtr_16 <dbl>, wtr_17 <dbl>,
16 ## #   wtr_18 <dbl>
```

Ya tenemos los datos ordenados para aplicar la función `ts.thermo.depth()`. Ahora vamos a intentar calcular la profundidad de la termoclinia para cada día.

```
1 termoclinia_diaria <- ts.thermo.depth(Crystal_ts)
```

Vemos que tenemos muchos días con valor NA. Esto se debe a que si la función encuentra un valor de NA en el perfil de temperatura devuelve NA. Para evitar esto podemos añadir el argumento `na.rm = TRUE`.

```
1 termoclinia_diaria <- ts.thermo.depth(Crystal_ts, na.rm = TRUE)
```

Ahora vamos a guardar la información en la carpeta Datos.

```
1 #Guardamos los perfiles de temperatura de cada día
2 write_csv(Crystal_ts, "Datos/Crystal_ts.csv")
3 #Guardamos la profundidad de la termoclina
4 write_csv(termocлина_diaria, "Datos/Termocлина.csv")
```

6.5.1 Ejercicios

1. *Representa la profundidad de la termoclina para cada día.*
2. *Calcula la estabilidad de Schmidt para todos los días.*
3. *Representa el valor de la estabilidad de Schmidt para cada día.*
4. *Calcula los límites del metalimnion para todos los días.*

Capítulo 7

Gráficas de contorno

Por último y aprovechando que tenemos información diaria de la temperatura del lago a distintas profundidades, quizás, sería interesante realizar una gráfica de contorno del periodo estudiado. Estas gráficas son muy visuales y ayudan a entender mejor el comportamiento del lago. Las vemos a menudo en los artículos.

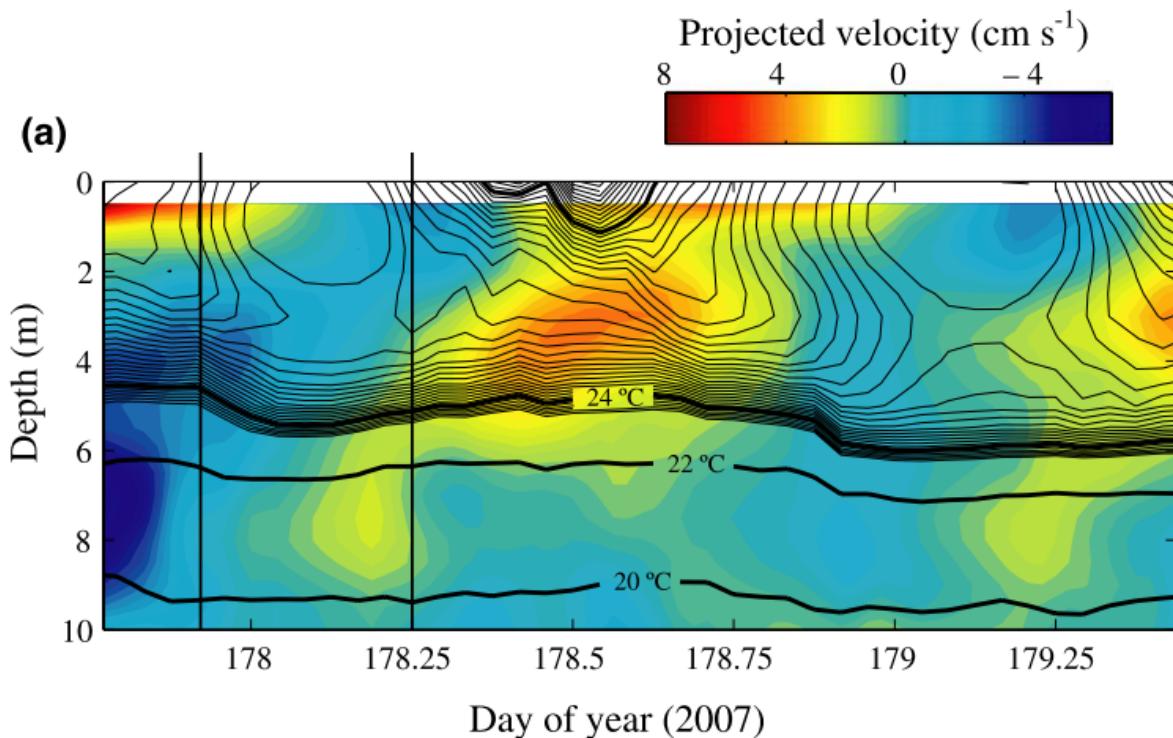


Figura 7.1: Imagen ejemplo de gráfica de contorno extraída de @Vidal2010.

Existen distintos programas que permiten hacer estas gráficas como son Surfer, SigmaPlot o OceanDataView, con el inconveniente que la mayoría de ellos son privativos y de pago (OceanDataView se salva!). Además, ya estamos metidos en harina y, como hemos dicho anteriormente, R nos permite trabajar desde los datos crudos hasta la generación de gráficas e informes (estos documentos están hechos directamente desde R).

Vamos a partir de la tabla de datos en formato “ancho” que creamos para calcular la profundidad de la termoclinia `Crystal_ts`.

¹ `#Cargamos tidyverse si cerramos la sesión anterior`

```

2 library(tidyverse)
3 #Importamos los datos de temperatura
4 Crystal_ts <- read_csv("Datos/Crystal_ts.csv")

```

```

1 ## Rows: 189 Columns: 20
2 ## — Column specification —
3 ## Delimiter: ","
4 ## dbl (19): wtr_0, wtr_1, wtr_2, wtr_3, wtr_4, wtr_5, wtr_6, wtr_7, wtr_8, wt...
5 ## date (1): datetime
6 ##
7 ## i Use `spec()` to retrieve the full column specification for this data.
8 ## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

```

Cuando transformamos el objeto a formato “ancho” homogeneizamos las profundidades a las que tenemos datos de temperatura. Es decir, tenemos el mismo número de medidas y a las mismas profundidades para cada momento. Sin embargo, como los datos no estaban completos se han introducidos NAs en aquellas profundidades donde no teníamos información. Por ejemplo, en el perfil del 2012-03-27, tenemos datos de temperatura a 4 y a 9 metros pero no para las profundidades entre 5-8 m. Para solucionar esto, podemos interpolar los datos. Para ello, vamos a usar un nuevo paquete, [zoo](#).

```

1 #Cargamos el paquete. Tiene que estar previamente instalado
2 library(zoo)

```

```

1 ##
2 ## Attaching package: 'zoo'

```

```

1 ## The following objects are masked from 'package:base':
2 ##
3 ##     as.Date, as.Date.numeric

```

```

1 #Interpolamos y extrapolamos (rule = 2) la temperatura para las profundidades que nos faltan.
2 Crystal_ts[,-1] <- na.approx(Crystal_ts[,-1], rule = 2)

```

Ahora podemos devolver los datos a su formato largo.

```

1 #Cargamos el paquete tidyverse
2   library(tidyverse)
3 #Devolvemos la tabla al formato "largo"
4   Temp_largo <- gather(Crystal_ts, key = "depth", value = "wtemp", -datetime)
5 #Le quitamos "wtr_" a la variable profundidad para que se quede sólo los números.
6   Temp_largo <- Temp_largo %>% mutate(depth = parse_number(depth))

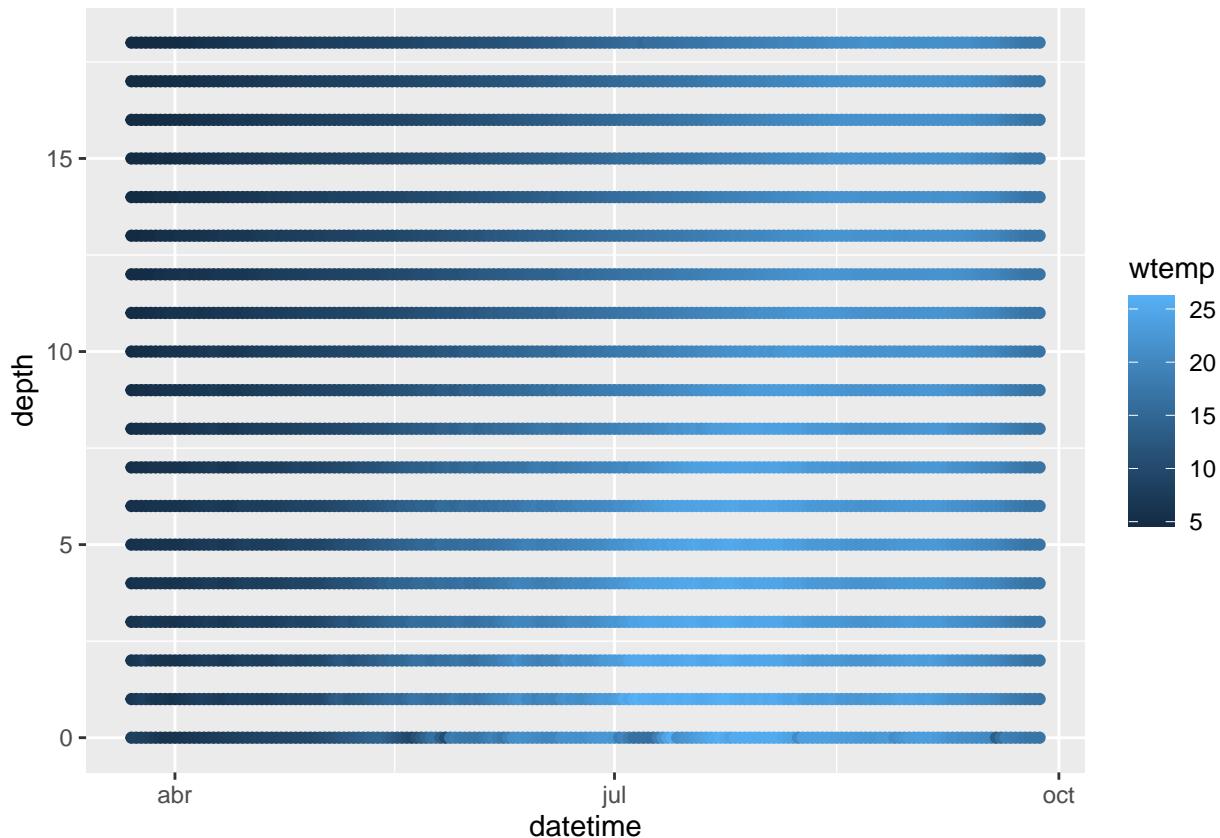
```

E intentamos representar.

```

1 ggplot(Temp_largo, aes(x = datetime, y = depth, color = wtemp))+
2   geom_point()

```



Parece que estamos bastante cerca, sin embargo, hay muchos huecos en blanco. Para solucionar esto vamos a crear una matriz y a usar una interpolación espacial multinivel b-spline para completar la información que nos falta. Para esto, usamos el paquete [MBA](#).

```

1 #Cargamos lubridate, aunque viene con tidyverse hay que "llamarlo" a parte.
2 library(lubridate)
3 #Primero tenemos que convertir las fechas en un vector numérico
4 Temp_largo$datetime <- decimal_date(Temp_largo$datetime)
5 #Cargamos el paquete MBA, hay que instalarlo previamente.
6 library(MBA)
7 # Aquí creamos una matriz con mayor resolución usando una interpolación espacial multinivel b-
8 spline
9 Temp_mba <- mba.surf(Temp_largo, no.X = 500, no.Y = 500, extend = T)
10 #Aquí están las fecha con la nueva resolución (en este caso es mayor de la que teníamos, 500 "
  "perfiles", frente a los 189 que teníamos)
11 head(Temp_mba$xyz.est$x)

```

```

1 ## [1] 2012.224 2012.225 2012.226 2012.227 2012.228 2012.229

```

```

1 #Aquí tenemos las nuevas profundidades, 500 profundidades en lugar de las 19 que teníamos antes
2 head(Temp_mba$xyz.est$y)

```

```

1 ## [1] 0.00000000 0.03607214 0.07214429 0.10821643 0.14428858 0.18036072

```

```

1 #Estos son los datos de temperatura

```

```
2 | head(Temp_mba$xyz.est$z) [1:10]
```

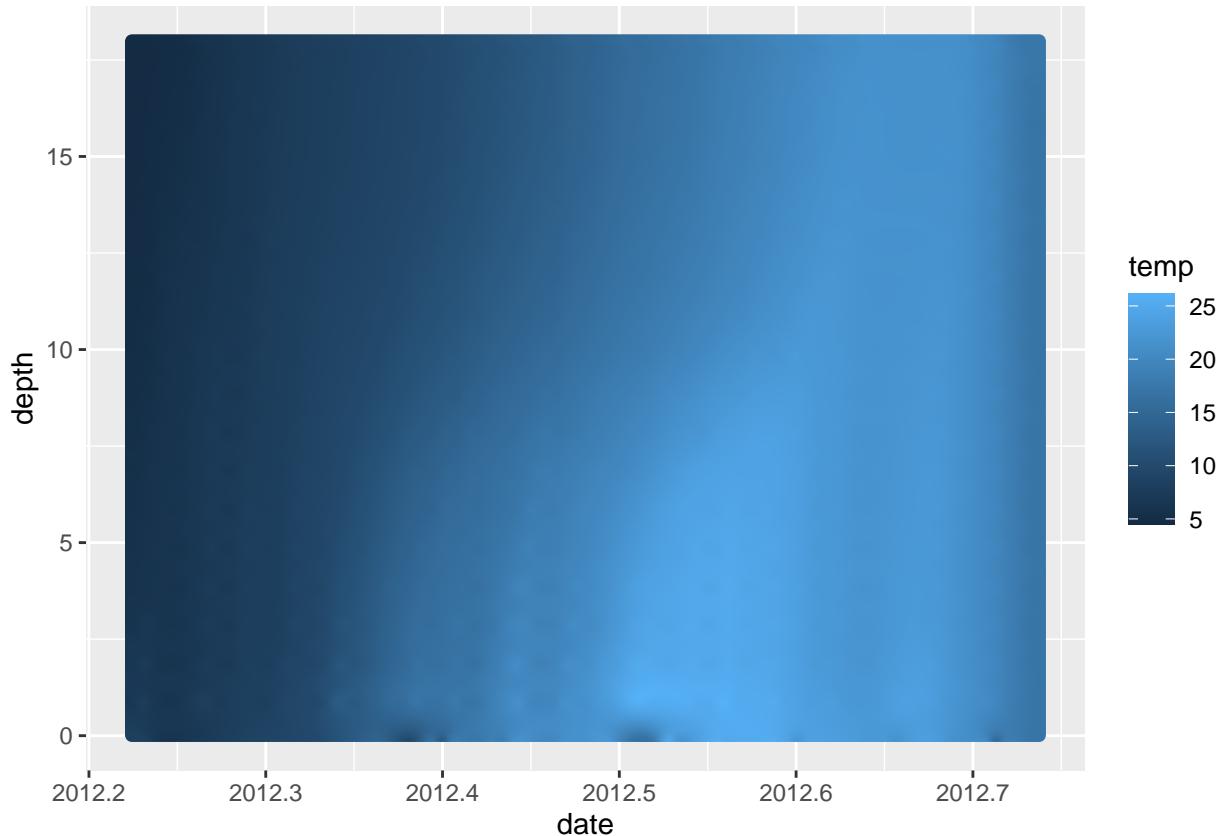
```
1 ## [1] 8.336292 8.334450 8.329062 8.320237 8.308084 8.290180 8.312388 8.310018
2 ## [9] 8.303618 8.293271
```

```
1 #Los juntamos todos
2 Temp_prof <- as.data.frame(Temp_mba$xyz.est$z)
3 colnames(Temp_prof) <- Temp_mba$xyz.est$x
4 Temp_prof <- bind_cols(date = Temp_mba$xyz.est$x, Temp_prof)
5 #Y los volvemos al formato largo.
6 Temp_mba <- gather(Temp_prof, key = "depth", value = 'temp', -date) %>% mutate(temp = round(temp
7 , 3))
8 #Ponemos la profundidad en numérico, se nos había quedado como carácter
9 Temp_mba <- Temp_mba %>% mutate(depth = as.numeric(depth))
10
11 #Esta es la pinta de los datos:
12 head(Temp_mba)
```

```
1 ##      date depth  temp
2 ## 1 2012.224     0 8.336
3 ## 2 2012.225     0 8.334
4 ## 3 2012.226     0 8.329
5 ## 4 2012.227     0 8.320
6 ## 5 2012.228     0 8.308
7 ## 6 2012.229     0 8.290
```

Ahora representamos de nuevo.

```
1 ggplot(Temp_mba, aes(x = date, y = depth, color = temp))+
2   geom_point()
```

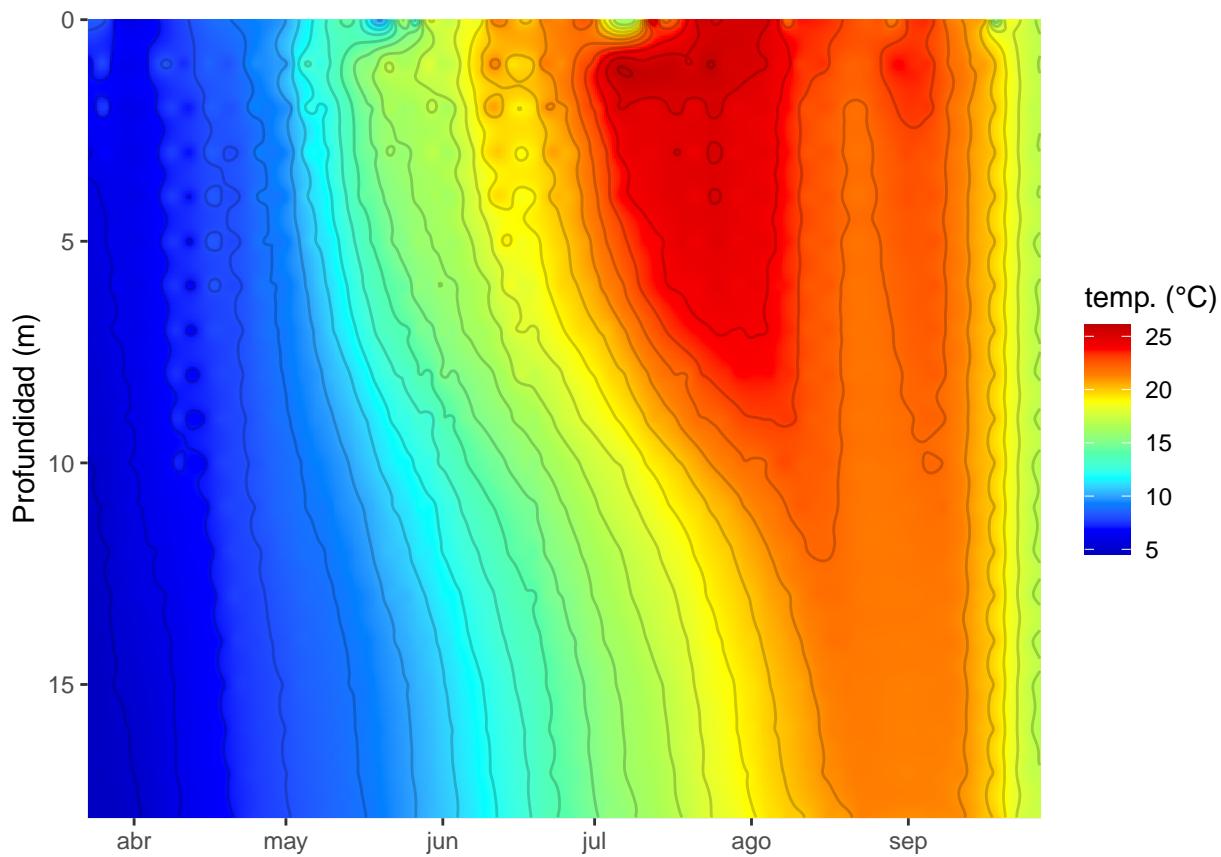


¡Mucho mejor! Vamos a cambiar algunos aspectos estéticos para que quede más resultona.

```

1 #Vamos a devolverle el formato de fecha
2 Temp_mba$date <- date_decimal(Temp_mba$date)
3 #Cargamos un paquete para usar una paleta de color más común
4 library(colorRamps)
5 Grafica_temp <- ggplot(data = Temp_mba, aes(x = date, y = depth)) +
6   geom_tile(aes(fill = temp)) + #Usamos esta capa que viene mejor para este tipo de gráficos pero
    podíamos haber usado geom_point
7   scale_y_reverse()+
8   scale_fill_gradientn(colours = matlab.like2(10)) +
9   geom_contour(aes(z = temp), binwidth = 1, colour = "black", alpha = 0.2) +
10  labs(y = "Profundidad (m)", x = NULL, fill = "temp. (°C)") +
11  coord_cartesian(expand = 0)
12 Grafica_temp

```



Hemos cambiado el formato de la fecha, hemos invertido el eje profundidad para que sea más intuitivo, hemos cambiado las etiquetas, el color y añadido unas líneas de contorno.

Vamos a probar a añadirle la profundidad de la capa de mezcla.

```
1 Termocлина <- read_csv("Datos/Termocлина.csv")
```

```
1 ## Row: 189 Columns: 2
2 ## — Column specification —
3 ## Delimiter: ","
4 ## dbl (1): thermo.depth
5 ## date (1): datetime
6 ##
7 ## i Use `spec()` to retrieve the full column specification for this data.
8 ## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
1 Grafica_temp_zmix <- Grafica_temp + geom_line(data = Termocлина, aes(x=datetime, y = thermo.
    depth, color = "Termocлина"), size = 0.2) +
  scale_color_manual(values = "black") + labs(color = NULL)
2
3 Grafica_temp_zmix
```

```
1 ## Error: Invalid input: time_trans works with objects of class POSIXct only
```

¡Vaya! Tenemos un error. Esto saca de quicio pero os iréis acostumbrando, poco a poco empezaréis a comprender que quiere decir el mensaje de error y buscar en qué os habéis equivocado. En este caso, el error nos

dice que al transforma un objeto de tiempo no ha podido porque trabaja con objetos de clase POSIXct. Si os fijáis en el data.frame que usamos para hacer la gráfica de contorno (Temp_mba) la fecha está en formato POSIXct y en el data.frame de Termocina está en formato Date. Sólo tenemos que cambiar el formato de este último y estará solucionado.

```

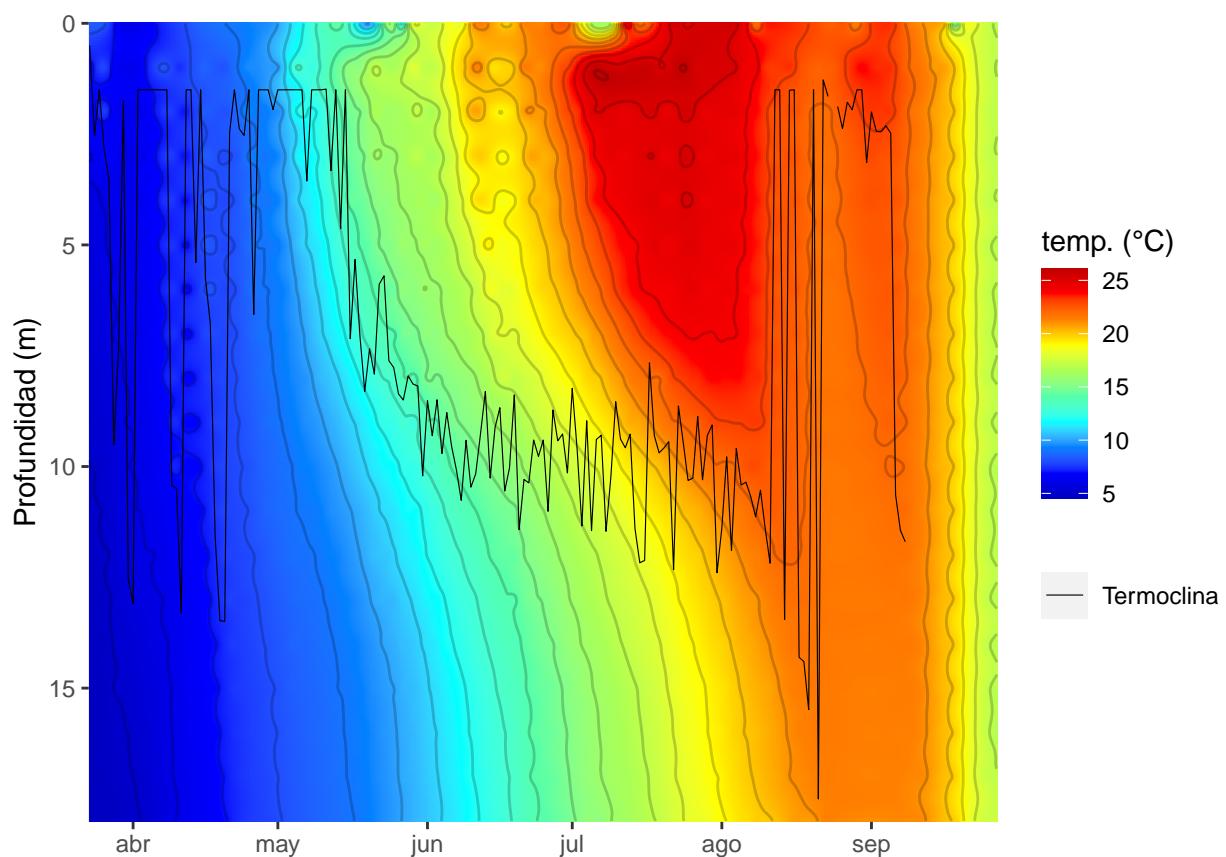
1 Termocina <- Termocina %>% mutate(datetime = as.POSIXct(datetime))
2
3 Grafica_temp_zmix <- Grafica_temp + geom_line(data = Termocina, aes(x=datetime, y = thermo.depth
, color = "Termocina"), size = 0.2) +
  scale_color_manual(values = "black") + labs(color = NULL)
4
5
6 Grafica_temp_zmix

```

```

1 ## Warning: Removed 9 row(s) containing missing values (geom_path).

```



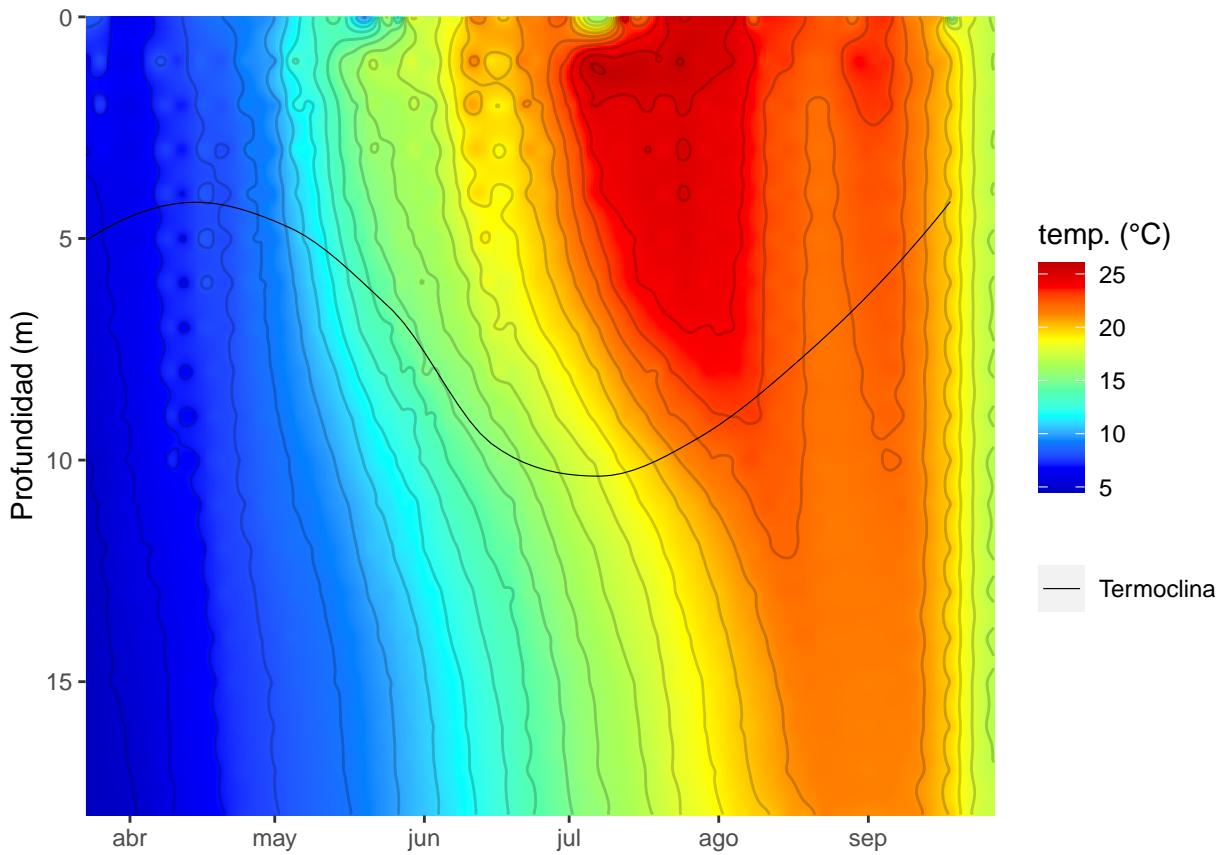
Como vemos hay cierta oscilación en la termocina y sobretodo en los momentos en lo que la estratificación está comenzando o se está rompiendo. A rasgos generales podemos decir que la estratificación comienza a finales de mayo y se prolonga hasta mediados de agosto. A mediados de agosto vemos que se rompe la estratificación y vuelve a estratificarse suavemente pero a mediados de septiembre el lago está totalmente mezclado con las isotermas totalmente verticales. Si queremos, para visualizar la termocina podemos usar un suavizado:

```

1 Grafica_temp_zmix <- Grafica_temp + geom_smooth(data = Termocina, aes(x=datetime, y = thermo.
  depth, color = "Termocina"), size = 0.2, na.rm = TRUE, se = FALSE) +
  scale_color_manual(values = "black") + labs(color = NULL)
2
3
4 Grafica_temp_zmix

```

```
1 ## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Y ya la tenemos lista para exportar. Podéis guardarla en el formato que más os guste (.jpeg, .pdf, .bmp, ...) simplemente cambiando el nombre con el que lo guardáis o indicándolo con el argumento device. Además también podéis fijar el ancho y el largo, así como otras opciones que podéis ver en ?ggsave().

```
1 ggsave("./Graficas/Grafica_temp.png", Grafica_temp_zmix, width = 20, height = 10, units = "cm")
```

7.0.1 Ejercicios

1. Añade el límite superior e inferior del metalimnion a la gráfica Gráfica_temp.
2. Haz una gráfica de contorno para el oxígeno disuelto.