Colectando datos fisicos, biologicos y ambientales en ambientes marinos

Breve introduccion a Bio-ORACLE

Table of contents

En este tutorial, nos enfocaremos brevemente en como obtener datos bioticos, geofisicos, y ambientales en capas geograficas oceanicas. Vamos a usar R e informacion de Bio-ORACLE.

1 Explorando los datos en Bio-ORACLE

Bio-ORACLE es un sitio web que permite descargar rasters sobre variables bióticas, geofísicas y ambientales en ambientes marinos bentónicos y superficiales. Todas las capas de datos están disponibles globalmente con una resolución espacial de 5 arcmin (~9,2 km en el Ecuador). Bio-ORACLE también permite descargar capas basadas en proyecciones futuras para cuatro variables: temperatura del mar, salinidad, velocidad de la corriente y espesor del hielo. En general, Bio-ORACLE puede ser usado para responder diferentes preguntas sobre ecologia y evolucion de especies, ademas de otras mas generales que involucran aspectos sociales, geologicos, entre otras.

Ejemplos de aplicaciones de capas marinas Bio-ORACLE

- Modelamiento de distribución de especies
- Modelamiento de nicho ecológico

- Genómica del paisaje marino
- Asociaciones genotipo-ambiente

1.1 Instalar y cargar los paquetes relevantes en R

Para iniciar este tutorial, necesitamos primero instalar los paquetes relevantes para los analisis.

```
library(tidyverse)
library(sdmpredictors)
library(raster)
library(sp)
library(dismo)
```

1.1.1 Exportar un archivo csv que contenga variables marinas de interés

En los siguientes pasos vamos a discutir como generar un archivo **csv** que exporte variables marinas de interes en un area de suramerica. Nos vamos a enfocar en las siguientes variables obtenidas a partir de Bio-ORACLE: temperatura del mar, salinidad, batimetría, velocidad de la corriente, oxígeno disuelto, producción primaria, concentración de fosfato, pH y concentración de silicato. Los archivos exportados contienen información adicional que es útil: códigos y descripciones de las capas, unidades de medida y resolución.

Primero, nos enfocaremos en describir las condiciones ambientales actuales:

```
# Buscar las capas marinas en la base de datos
datasets = list_datasets(terrestrial = FALSE, marine = TRUE)

# Nos vamos a enfocar en algunas de las variables unicamente
variables = c("temp","salinity","bathy","curvel","ox","pp","ph","silicate")

# Extraemos los datos actuales
present = list_layers(datasets) %>%
  # Seleccionamos algunas columnas
  dplyr::select(dataset_code, layer_code, name, units, description, contains("cellsize"),
  # Y, de nuevo, vamos a hacer enfasis en una parte de las variables
  dplyr::filter(grepl(paste(variables, collapse = "|"), layer_code))
```

```
# Desde aqui, podemos exportar los datos!
write_csv(present, "bio-oracle-present-datasets.csv")
```

Ahora, podemos hacer una descripcion de condiciones ambientales proyectadas!

```
# Future Representative Concentration Pathway (RCP)
rcp = c("RCP26","RCP45","RCP60","RCP85")

# Extrayendo las capas del set de datos
future = list_layers_future(datasets) %>%
  # Dejamos los RCP de interes
  dplyr::filter(grepl(paste(rcp, collapse = "|"), scenario)) %>%
  # Y datos para 2050 y 2100
  dplyr::filter(year == 2050 | year == 2100) %>%
  # De nuevo, las variables de interes!
  dplyr::filter(grepl(paste(variables, collapse = "|"), layer_code))
```

```
# Igual que para los datos actuales, podemos exportar los datos
write_csv(future, path = "bio-oracle-future-datasets.csv")
```

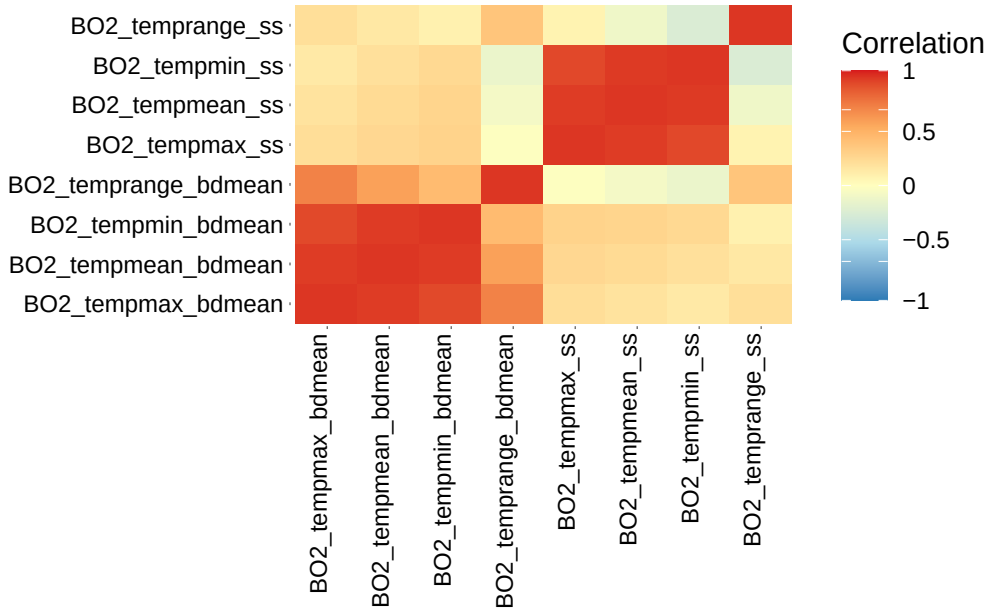
En el resto de este documento nos centraremos en analizar la batimetría y temperatura del mar. Sin embargo, el código debe aplicarse tambien a cualquiera de las otras capas en la base de datos.

1.1.2 Comprobar la colinealidad entre las capas de temperatura del mar

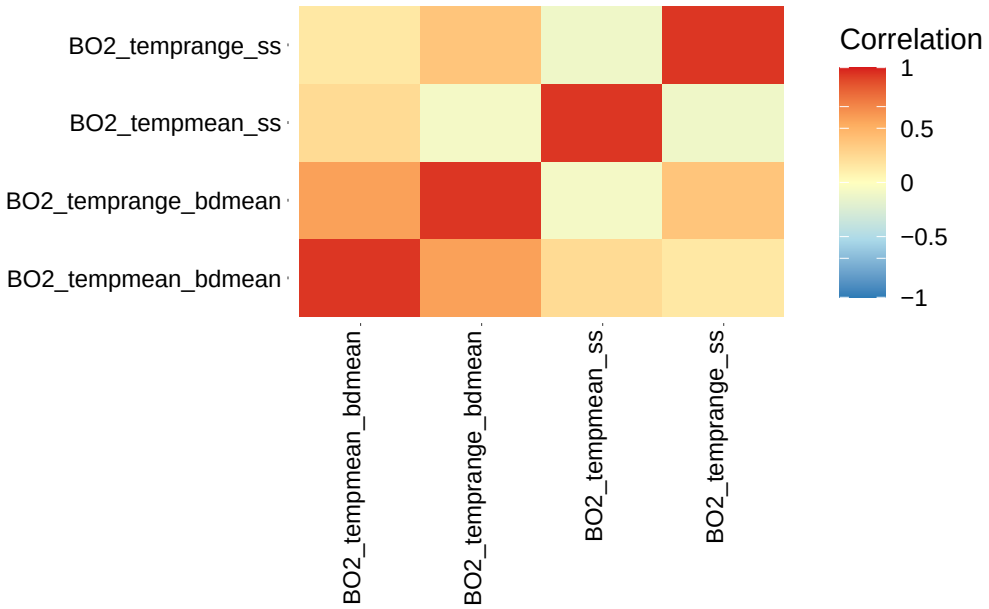
Las variables que están correlacionadas entre sí pueden afectar el rendimiento de los modelos. Por ejemplo, OLS asumen la independencia de predictores. Por lo tanto, si se considera que dos variables están correlacionadas, generalmente solo se usa una de ellas en el análisis. En el siguiente ejemplo, especificamos los códigos de capa de nuestras variables de interés y luego evaluamos su correlación. Noten que solo nos vamos a enfocar en algunas capas especificas.

```
# Creamos un vector con las capas de interes
temp.bottom = c("BO2_tempmax_bdmean",
                "BO2_tempmean_bdmean",
                "BO2_tempmin_bdmean",
                "BO2_temprange_bdmean")

temp.surface = c("BO2_tempmax_ss",
                 "BO2_tempmean_ss",
                 "BO2_tempmin_ss",
                 "BO2_temprange_ss")

temp.bottom.surface = c(temp.bottom, temp.surface)
```

```r
# Examinamos la correlacion entre los rasters
layers_correlation(temp.bottom.surface) %>% plot_correlation
```




```r
# Cuantifiquemos la correlacion esperando que no haya fuerte correlacion (-0.6 > x < 0.6)
temp.present = c("BO2_tempmean_bdmean",
                 "BO2_temprange_bdmean",
                 "BO2_tempmean_ss",
                 "BO2_temprange_ss")

layers_correlation(temp.present) %>% round(digits = 2)
##                      BO2_tempmean_bdmean BO2_temprange_bdmean BO2_tempmean_ss
## BO2_tempmean_bdmean                1.00                 0.59            0.24
## BO2_temprange_bdmean               0.59                 1.00           -0.08
## BO2_tempmean_ss                    0.24                -0.08            1.00
## BO2_temprange_ss                   0.15                 0.38           -0.11
##                      BO2_temprange_ss
## BO2_tempmean_bdmean              0.15
## BO2_temprange_bdmean             0.38
## BO2_tempmean_ss                 -0.11
## BO2_temprange_ss                 1.00
layers_correlation(temp.present) %>% plot_correlation
```



1.2 Descargar e importar rásteres de Bio-ORACLE

Vamos primero a crear dos vectores con los nombres (codigos) de las capas de interes en nuestros analisis.

```
# Los dos vectores con los nombres de la capas de interes
temp.present = gsub("BO2", "BO21", temp.present)
temp.future = future$layer_code[grep("BO21_RCP26_2050_tempmean", future$layer_code)]

# Los combinamos!
temp = c(temp.present, temp.future)
```

Desde este punto, vamos a descargar las capas en una carpeta temporal. Esta opcion puede modificarse dependiendo de su interes en dejar los datos locales.

```
# Descargamos los rasters y cargamos al environment de R
bathy.raster = load_layers("MS_bathy_5m")
## Warning in get_datadir(datadir): file.path(tempdir(), "sdmpredictors") will
## be used as datadir, set options(sdmpredictors_datadir="<directory>") to avoid
## re-downloading the data in every session or set the datadir parameter in
## load_layers
names(bathy.raster) = "MS_bathy_5m"
temp.rasters = load_layers(temp, datadir = tempdir())
```

```
## Warning in load_layers(temp, datadir = tempdir()): Layers from different eras
## (current, future, paleo) are being loaded together
```

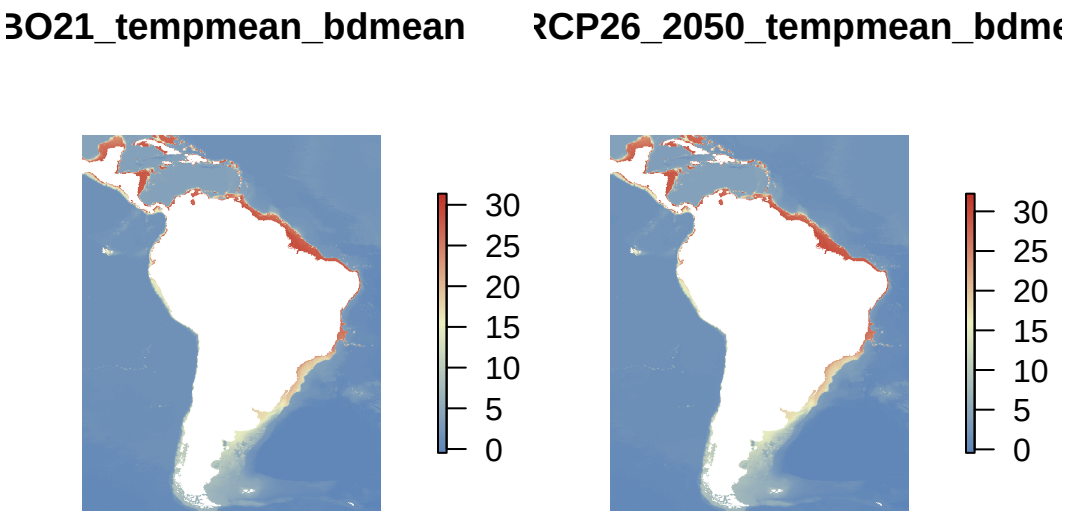
1.2.1 Visualizar los rasters

Primero, vamos a definit los limites de visualizacion de los rasters a analizar.

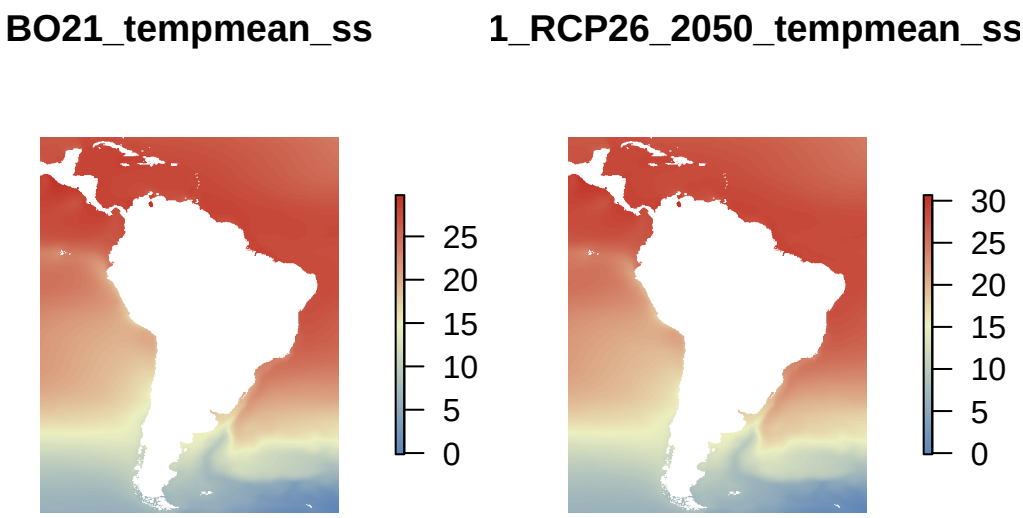
```
# Definimos los limites
boundary = extent(c(xmin = -96, xmax = -30, ymin = -56, ymax = 24))

# Recortamos los rasters a los limites definidos anteriormente
bathy.raster = crop(bathy.raster, boundary)
temp.rasters = crop(temp.rasters, boundary)
```

Ahora, visualizamos los raster usando la función `raster::plot()`.

```
# Definimos los colores
cols = colorRampPalette(c("#5E85B8","#EDF0C0","#C13127"))

# Graficamos el promedio de temperatura (en profundidad)
raster::subset(temp.rasters, grep("tempmean_bdmean",
                                  names(temp.rasters), value = TRUE)) %>%
  plot(col = cols(100), axes = FALSE, box = FALSE)
```

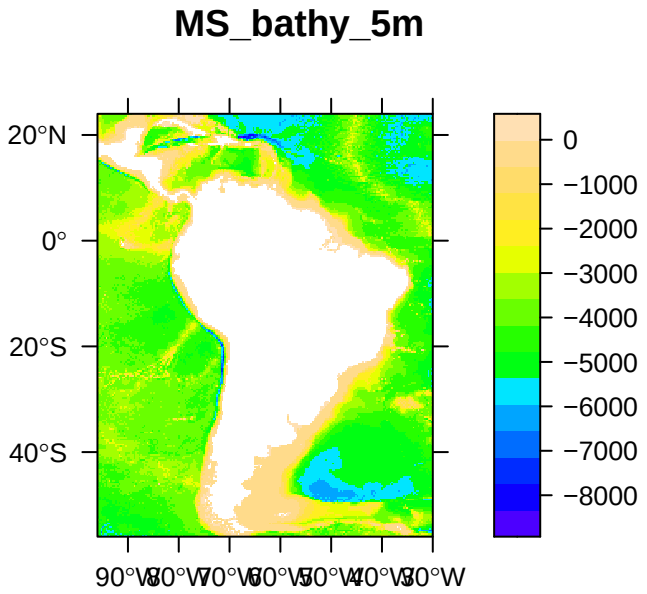


```r
# Repetimos el mismo proceso para la temperatura superficial
raster::subset(temp.rasters, grep("tempmean_ss",
                                  names(temp.rasters), value = TRUE)) %>%
  plot(col = cols(100), axes = FALSE, box = FALSE)
```



Podemos generar visualizaciones un poco mas esteticas usando `sp::spplot()`.

```r
# Un grafico de profundidad
sp::spplot(bathy.raster, main = names(bathy.raster),
           scales = list(draw=TRUE), col.regions = topo.colors(100))
```



1.2.2 Extraer datos de los rasters

1.2.2.1 Preparar los puntos de observacion

Para esta seccion necesitamos importar o crear una tabla con datos de longitud y latitud. Para este tutorial, vamos a crear 100 observaciones en una de las capas del raster.

```
set.seed(123)
random.pts = randomPoints(bathy.raster, n = 100) %>% as_tibble()
```

Vamos a transformar nuesto `tibble` en `SpatialPoints` y ajustar el sistema de referencia de coordenadas (CRS) para nuestros puntos.

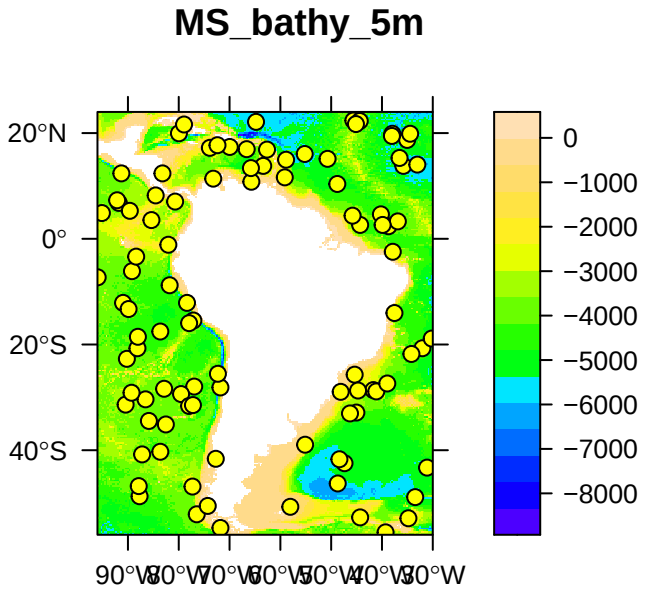
```
random.pts = SpatialPoints(random.pts, proj4string = CRS("+proj=longlat +datum=WGS84"))
random.pts
## class       : SpatialPoints
## features    : 100
## extent      : -95.95833, -30.20833, -55.54167, 22.375  (xmin, xmax, ymin, ymax)
## crs         : +proj=longlat +datum=WGS84 +no_defs
```

Ahora verificamos que el CRS de los puntoscoincide con el CRS del ráster.

```
projection(random.pts) == projection(bathy.raster)
## [1] TRUE
```

Podemos ahora visualizar los puntos sobre el raster de batimetria!

```
sp::spplot(bathy.raster,  main = names(bathy.raster),
           scales = list(draw=TRUE), col.regions = topo.colors(100),
           sp.layout = c("sp.points", random.pts, pch = 21, cex = 1,
                         fill = "yellow", col = "black")
           )
```



Ahora vamos a extraer datos de los rasters en un `tibble` o `data.frame` para cada punto.

```
marine.data = tibble(ID = 1:nrow(random.pts@coords),
                     Lon = random.pts$x,
                     Lat = random.pts$y
                     )
marine.data
## # A tibble: 100 x 3
##       ID   Lon    Lat
##    <int> <dbl>  <dbl>
##  1     1 -70.0  17.4
##  2     2 -77.3 -46.9
##  3     3 -80.0  20.0
##  4     4 -44.3   2.62
##  5     5 -45.0 -32.9
##  6     6 -64.8  22.1
```

```
##  7     7 -79.0  21.6
##  8     8 -35.8  13.8
##  9     9 -73.2  11.4
## 10    10 -91.8   6.79
## # ... with 90 more rows
```

1.2.2.2 Extraer datos para cada punto

Primero combinamos los rasters en una sola unidad (stack).

```
rasters = raster::stack(bathy.raster, temp.rasters)
nlayers(rasters)
## [1] 9
```

Extraemos los datos de cada ráster para cada punto y los almacenamos en una lista.

```
store_data = list()
for (i in 1:nlayers(rasters)){
  store_data[[i]] = raster::extract(rasters[[i]], random.pts)
}
```

Por ultimo, re-formateamos los datos extraídos y los adicionamos como nuevas columnas a `marine.data`.

```
# Name variables in the list and then combine data
names(store_data) = names(rasters)
marine.data = bind_cols(marine.data, as_tibble(store_data))
marine.data
## # A tibble: 100 x 12
##       ID   Lon    Lat MS_bathy_5m BO21_tempmean_bdmean BO21_temprange_bdmean
##    <int> <dbl>  <dbl>       <dbl>                <dbl>                  <dbl>
##  1     1 -70.0  17.4        -4929                 3.17                  0.119
##  2     2 -77.3 -46.9        -3298                 1.31                  0.146
##  3     3 -80.0  20.0        -2650                 3.60                  0.0195
##  4     4 -44.3   2.62       -4156                 1.85                  0.244
##  5     5 -45.0 -32.9        -4156                 0.316                 0.480
##  6     6 -64.8  22.1        -5808                 1.47                  0.0455
##  7     7 -79.0  21.6         -135                27.8                   5.54
##  8     8 -35.8  13.8        -5481                 1.76                  0.0367
##  9     9 -73.2  11.4           -7                27.3                   5.13
## 10    10 -91.8   6.79       -3598                 1.39                  0.0265
```

```
## # ... with 90 more rows, and 6 more variables: BO21_tempmean_ss <dbl>,
## #   BO21_temprange_ss <dbl>, BO21_RCP26_2050_tempmean_bdmax <dbl>,
## #   BO21_RCP26_2050_tempmean_bdmean <dbl>,
## #   BO21_RCP26_2050_tempmean_bdmin <dbl>, BO21_RCP26_2050_tempmean_ss <dbl>
```

Removemos los NAs si es requerido (cuando algun punto esta fuera del area de cobertura de un raster).

```
# Revisamos si hay NAs
na.check = map_int(marine.data, ~sum(is.na(.)))
summary(na.check > 0)
##    Mode   FALSE
## logical      12
```

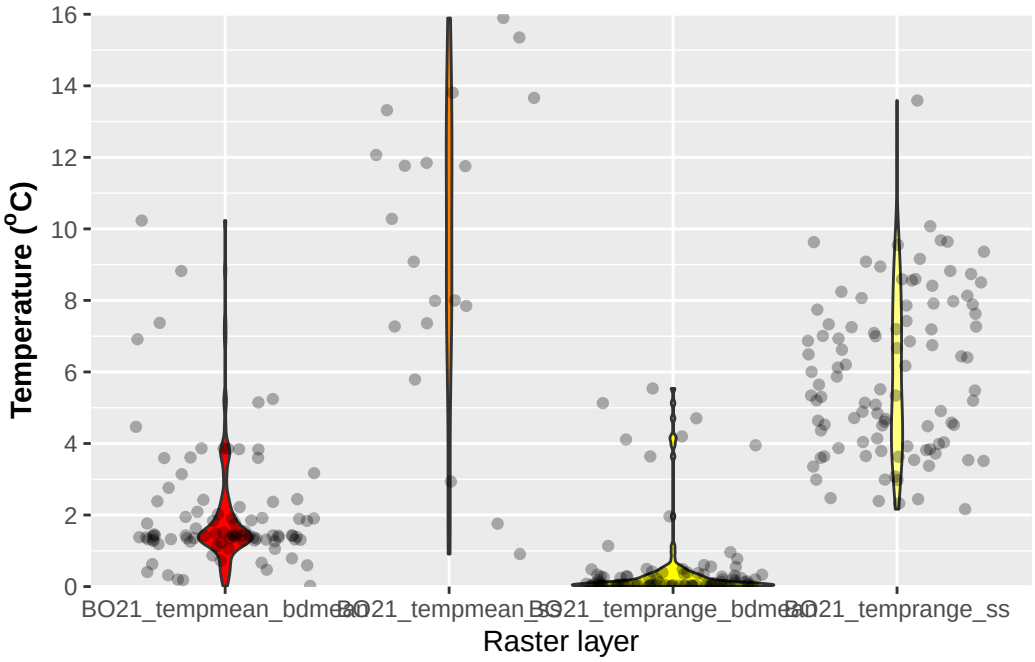
```
# Removemos los NAs!
marine.data = marine.data %>% drop_na
```

Podemos redondear los valores de temperatura en nuestro set de datos.

```
marine.data[-(1:4)] = apply(marine.data[-(1:4)], MARGIN = 2, FUN = round, digits = 3)
```

Ahora podemos visualizar los valores actuales de temperatura en los puntos muestreados!

```
# Graficos de violin y datos en bruto
marine.data %>%
  # Seleccionamos solo las columnas de temperatura actual
  dplyr::select(5:8) %>%
  # Transformamos a distribucion long para graficar
  pivot_longer(names_to = "Variable", values_to = "Values", cols = everything()) %>%
  # plot data
  ggplot(data = .)+
    geom_violin(aes(x = Variable, y = Values, fill = Variable), show.legend = FALSE)+
    geom_jitter(aes(x = Variable, y = Values), show.legend = FALSE, alpha = 0.30)+
    scale_y_continuous(expand = c(0,0), limits = c(0,16), breaks = c(seq(0,16,2)))+
    scale_fill_manual(values = heat.colors(4))+
    xlab("Raster layer")+
    ylab(expression(bold("Temperature ("^o*"C)")))
## Warning: Removed 89 rows containing non-finite values (`stat_ydensity()`).
## Warning: Removed 89 rows containing missing values (`geom_point()`).
```



Calculemos los valores de profundidad en los puntos mas superficiales y profundos en nuestro set de datos.

```
marine.data %>%
  summarise(MasProfundo = min(MS_bathy_5m), MasSuperficial = max(MS_bathy_5m))
## # A tibble: 1 x 2
##   MasProfundo MasSuperficial
##         <dbl>          <dbl>
## 1       -5808             -7
```

Por ultimo, podemos exportar nuestros datos en un archivo `csv`.

```
write_csv(marine.data, path = "marine_data.csv")
```