Using R to work with Copernicus Marine Data

MarineData4America Workshop

David March
dmarch@ub.edu

University of Barcelona and IRBio
University of Exeter

2021-06-30

Table of contents

1 Introduction

This tutorial is aimed to provide an introductory overview on using ***R*** programming language to work with CMEMS data. ***R*** has a great potential to handle marine spatial data and perform complex spatial operations. In particular, we will use RStudio, a popular Integrated Development Environment (IDE) for coding in ***R***.

Here, you will learn how to use RStudio to:

- Import and inspect NetCDF files
- Visualize and analyze gridded products
- Extract values from a numerical model

2 Setup Environment

2.1 Install RStudio

In order to follow this tutorial, you will need to:

1. Install RStudio Desktop. The Open Source Edition provides a free license for multiple operating systems.
2. Open Rstudio and run the following code to install all required packages:

```
install.packages(c("raster", "ncdf4", "ggplot2", "lubridate", "sf", "leaflet",
                   "rasterVis", "rnaturalearth", "rnaturalearthdata"))
```

3. Start loading the required packages

```
library(ncdf4)
library(raster)
library(sf)
library(leaflet)
library(lubridate)
library(rasterVis)
library(ggplot2)
library(rnaturalearth)
library(rnaturalearthdata)
```

2.2 Download data

This tutorial uses two sample datasets: one gridded product from CMEMS in NetCDF format and the boundaries of a marine protected area (MPA) in GeoPackage format. They both will be introduced later on in this tutorial. First, we will create a new directory in current working directory to write data to

```
dir.create("data")
```

Then, sample data can be downloaded with the following commands:

```
# Download CMEMS sample data
download.file(url = "https://github.com/dmarch/Rworkshop-MarineData4America/raw/main/data/global-analysi
    destfile = "data/global-analysis-forecast-phy-001-024-monthly_1624214790015.nc",
    mode = "wb")

# Download Galapagos Islands MPA
download.file(url = "https://github.com/dmarch/Rworkshop-MarineData4America/raw/main/data/GalapagosMPA.g
    destfile = "data/GalapagosMPA.gpkg", mode = "wb")
```

3 Handling NetCDF files in R

3.1 The NetCDF format

NetCDF is a community standard for sharing scientific data. It is particularly well suited to represent multidensional data, and has been adopted as a data format for many oceanography systems, including the Copernicus Marine products. A NetCDF file contains dimensions, variables, and attributes.

- **Dimensions** represent real physical dimension (e.g. time, longitude, latitude).
- **Variables** are used to represent values of the same type (e.g. water temperature) and are described by its list of dimensions.
- **Attributes** are used to keep information about the data. They can provide information about one specific variable (e.g. units) or about the whole dataset (e.g. processing details)

3.2 Inspect NetCDF files

Here, we will import and explore a NetCDF file in R. The NetCDF correspond to a sample file from a numerical model from CMEMS. More specifically, it corresponds to a sea water temperature dataset from the Global Ocean 1/12º Physics Analysis and Forecast. In the following table, you can check the parameters used to download the dataset.

Table 1: Parameters of the sample dataset. Downloaded from Global Ocean 1/12º Physics Analysis and Forecast (Product: GLOBAL-ANALYSIS-FORECAST-PHY-001-024-MONTHLY)

Parameter	Value
Longitude (min)	-120
Longitude (max)	-20
Latitude (min)	-20
Latitude (max)	40
Start depth	0.494
End depth	0.494
Variables	thetao
Start time	2020-01-16 12:00
End time	2020-12-16 12:00

Note: CMEMS products can be automatically downloaded from R using the RCMEMS package. However, this approach requires the installation of Python and Motu client, and is beyond the scope of this tutorial.

In order to inspect the NetCDF, we will use the `ncdf4` package. This package provides a R interface to NetCDF files and allow reading and editing them.

```
# Set the path for the NetCDF file
ncfile <- "data/global-analysis-forecast-phy-001-024-monthly_1624214790015.nc"

# Import NetCDF
nc <- nc_open(ncfile)

# Print information about the NetCDF file
print(nc)
```

```
## File data/global-analysis-forecast-phy-001-024-monthly_1624214790015.nc (NC_FORMAT_CLASSIC):
##
##      1 variables (excluding dimension variables):
##         short thetao[longitude,latitude,depth,time]
##             _FillValue: -32767
##             long_name: Temperature
##             standard_name: sea_water_potential_temperature
##             units: degrees_C
##             unit_long: Degrees Celsius
##             cell_methods: area: mean
##             add_offset: 21
##             scale_factor: 0.000732444226741791
##
##      4 dimensions:
##         time  Size:12
##             long_name: Time (hours since 1950-01-01)
##             standard_name: time
##             axis: T
```

```
##             units: hours since 1950-01-01
##             calendar: gregorian
##             _CoordinateAxisType: Time
##             valid_min: 613980
##             valid_max: 622020
##         depth  Size:1
##             _FillValue: NaN
##             valid_min: 0.494024991989136
##             valid_max: 0.494024991989136
##             units: m
##             positive: down
##             unit_long: Meters
##             long_name: Depth
##             standard_name: depth
##             axis: Z
##             _CoordinateAxisType: Height
##             _CoordinateZisPositive: down
##         latitude  Size:721
##             _FillValue: NaN
##             valid_min: -20
##             valid_max: 40
##             step: 0.0833358764648438
##             units: degrees_north
##             unit_long: Degrees North
##             long_name: Latitude
##             standard_name: latitude
##             axis: Y
##             _CoordinateAxisType: Lat
##         longitude  Size:1201
##             _FillValue: NaN
##             valid_min: -120
##             valid_max: -20
##             step: 0.0833282470703125
##             units: degrees_east
##             unit_long: Degrees East
##             long_name: Longitude
##             standard_name: longitude
##             axis: X
##             _CoordinateAxisType: Lon
##
##     18 global attributes:
##         title: Monthly mean fields for product GLOBAL_ANALYSIS_FORECAST_PHY_001_024
##         references: http://marine.copernicus.eu
##         credit: E.U. Copernicus Marine Service Information (CMEMS)
##         licence: http://marine.copernicus.eu/services-portfolio/service-commitments-and-licence/
##         contact: servicedesk.cmems@mercator-ocean.eu
##         producer: CMEMS - Global Monitoring and Forecasting Centre
##         institution: Mercator Ocean
##         conventions: CF-1.6
##         area: GLOBAL
##         product: GLOBAL_ANALYSIS_FORECAST_PHY_001_024
##         dataset: global-analysis-forecast-phy-001-024-monthly
##         source: MERCATOR PSY4QV3R1
##         product_user_manual: http://marine.copernicus.eu/documents/PUM/CMEMS-GLO-PUM-001-024.pdf
```

```
##          quality_information_document: http://marine.copernicus.eu/documents/QUID/CMEMS-GLO-QUID-001-0
##          _CoordSysBuilder: ucar.nc2.dataset.conv.CF1Convention
##          Conventions: CF-1.0
##          comment:
##          history: Data extracted from dataset http://localhost:8080/thredds/dodsC/global-analysis-fore
```

We can see that the NetCDF contains **1** variable, **4** dimensions and **18** global attributes.

The variable `thetao` contains specific attributes, including the names, units and complementary parameters. You can check the documentation of the product for further details.

3.3 Import NetCDF as Raster

3.3.1 Single band

The numerical model constitutes a gridded product. To manipulate and visualize gridded data, we will use the ***raster*** package.

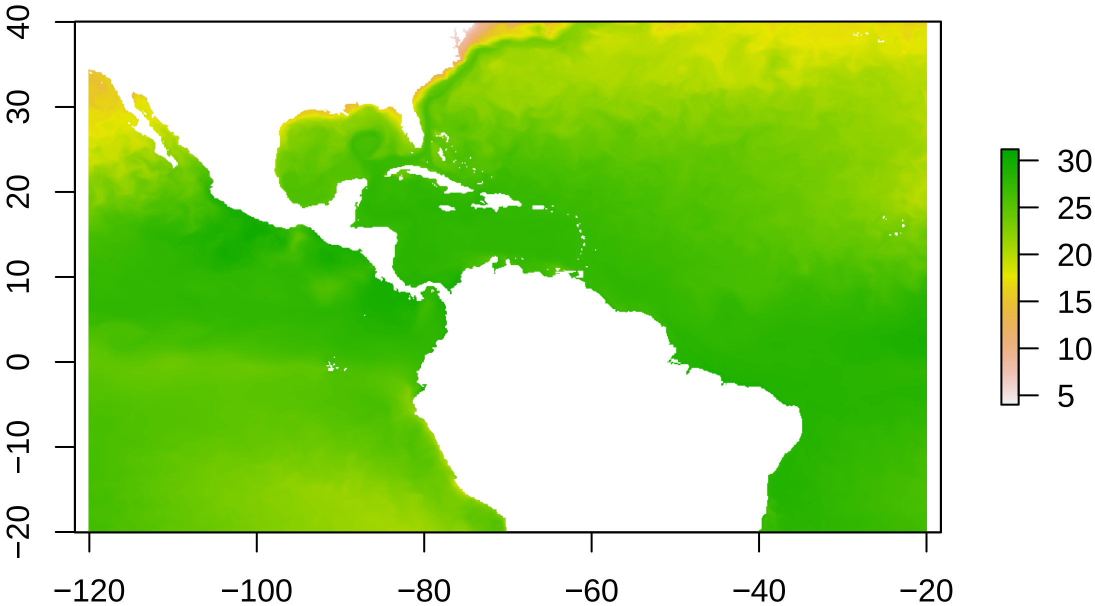
```r
# import NetCDF with raster
sst_single <- raster(ncfile)

# print a summary of the raster
sst_single
```

```
## class      : RasterLayer
## band       : 1  (of  12  bands)
## dimensions : 721, 1201, 865921  (nrow, ncol, ncell)
## resolution : 0.08333333, 0.08333333  (x, y)
## extent     : -120.0417, -19.95833, -20.04167, 40.04167  (xmin, xmax, ymin, ymax)
## crs        : +proj=longlat +datum=WGS84 +no_defs
## source     : global-analysis-forecast-phy-001-024-monthly_1624214790015.nc
## names      : Temperature
## z-value    : 2020-01-16 12:00:00
## zvar       : thetao
## level      : 1
```

The summary provides relevant information to understand the imported dataset, including the number of bands, spatial resolution, spatial extent, coordinate reference system (CRS), name of variable, time stamps. In this example, we can see that the importater raster contains one single band that correspond to the SST map of January 2020.

The ***raster*** package provides a `plot()` function to visualize gridded products.

```r
# plot raster dataset
plot(sst_single)
```



There are more advanced functions to plot raster data in R, using packages such as ***rasterVis*** or ***ggplot2***. In addition, packages such as ***leaflet*** offer the possibility to build interactive maps. We will explore them later.

3.3.2 Multiband layers

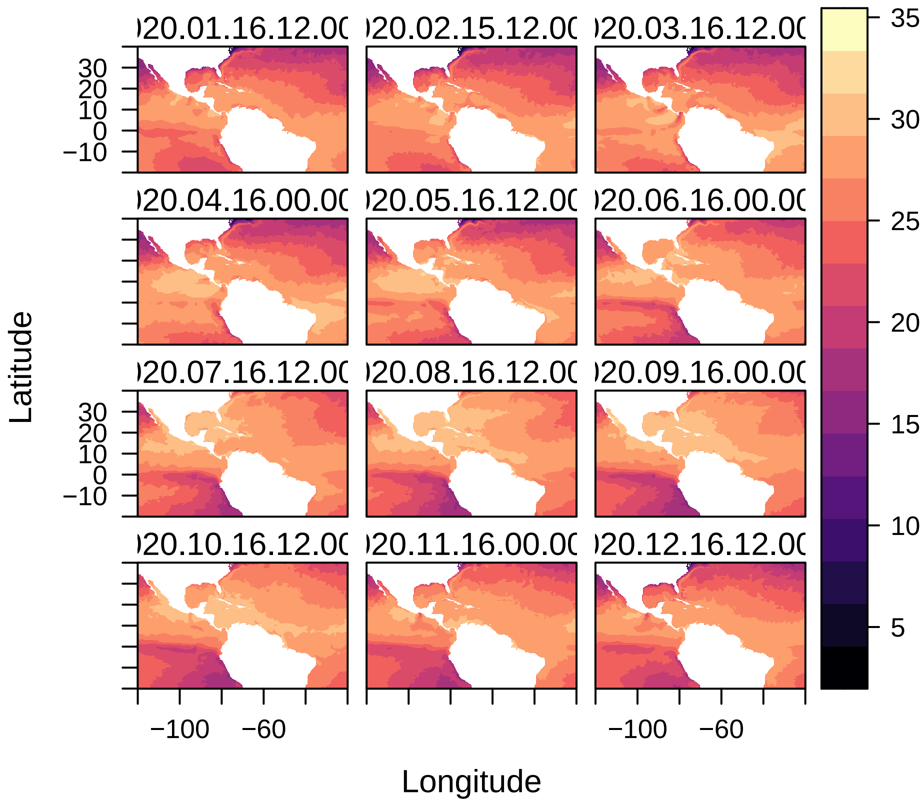
The function `raster()` only work for single band layers. In order to import multiple bands, we can use `brick()` or `stack()` functions.

```
# import multi-band NetCDF file
sst_multi <- brick(ncfile)

# print a summary of the brick
sst_multi
```

```
## class      : RasterBrick
## dimensions : 721, 1201, 865921, 12  (nrow, ncol, ncell, nlayers)
## resolution : 0.08333333, 0.08333333  (x, y)
## extent     : -120.0417, -19.95833, -20.04167, 40.04167  (xmin, xmax, ymin, ymax)
## crs        : +proj=longlat +datum=WGS84 +no_defs
## source     : global-analysis-forecast-phy-001-024-monthly_1624214790015.nc
## names      : X2020.01.16.12.00.00, X2020.02.15.12.00.00, X2020.03.16.12.00.00, X2020.04.16.00.00.00,
## Date/time  : 2020-01-16 12:00:00, 2020-02-15 12:00:00, 2020-03-16 12:00:00, 2020-04-16 00:00:00, 202
## varname    : thetao
## level      : 1
```

We now can see that all bands from the NetCDF file (each one for a different time stamp) have been imported. In order to visualize multiple bands that share a common scale, we can use the function `levelplot()` from the ***rasterVis*** package.

```
# plot brick dataset
levelplot(sst_multi)
```



4 Raster analysis

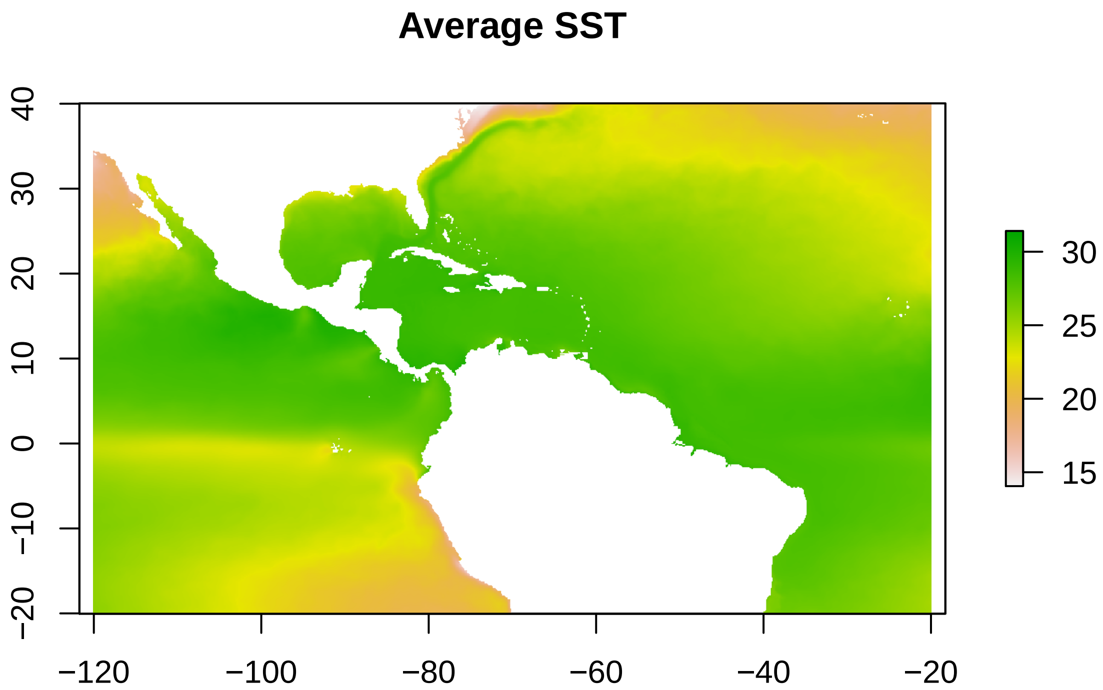
4.1 Summary statistics

In this section, we will combine all monthly maps (n=12) to calculate the mean and standard deviation (SD). These are two common metrics used to describe de centrality (middle value) and dispersion (how spread out the values are from the average) of the data. We will use the function `calc()`, which gathers all monthly values for each single grid cell and applies the given function (i.e. mean or sd).

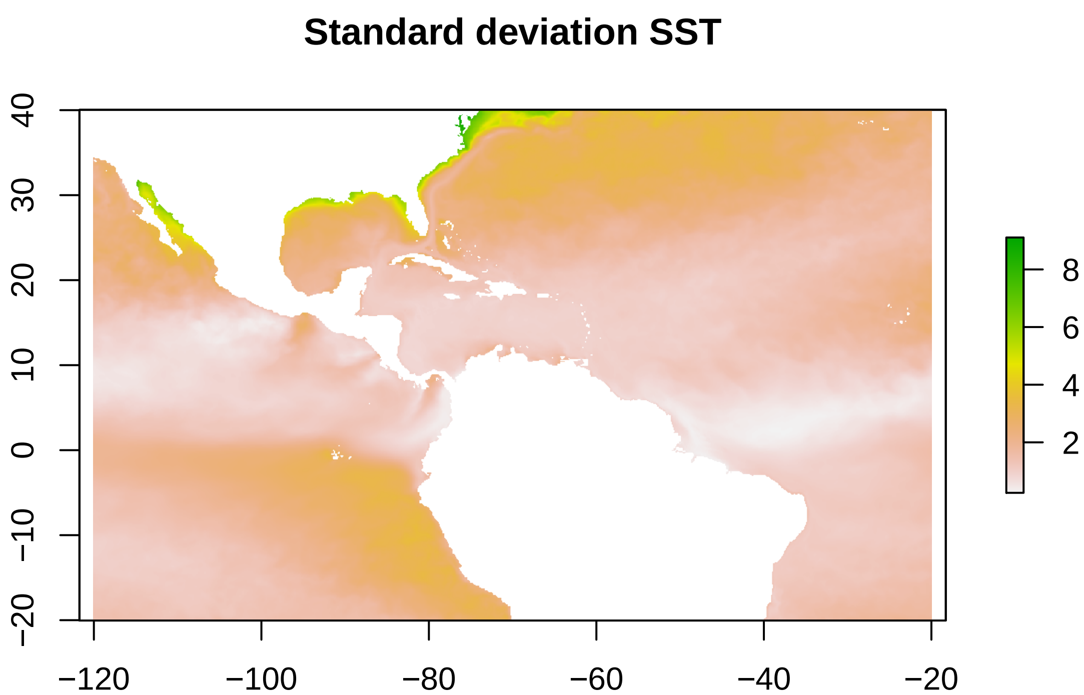
```
# calculate average and SD
sst_mean <- calc(sst_multi, fun = mean)
sst_sd <- calc(sst_multi, fun = sd)
```

We can then explore the generated maps

```
# plot raster dataset
plot(sst_mean, main = "Average SST")
```




```
plot(sst_sd, main = "Standard deviation SST")
```



A more advanced visualization of these maps can be conducted with the ***ggplot2*** package. This package offers a higher degree of customization and allows the generation of high-quality images to use in scientific journals or reports.

First, we need to transform the raster object into a `data.frame` class.

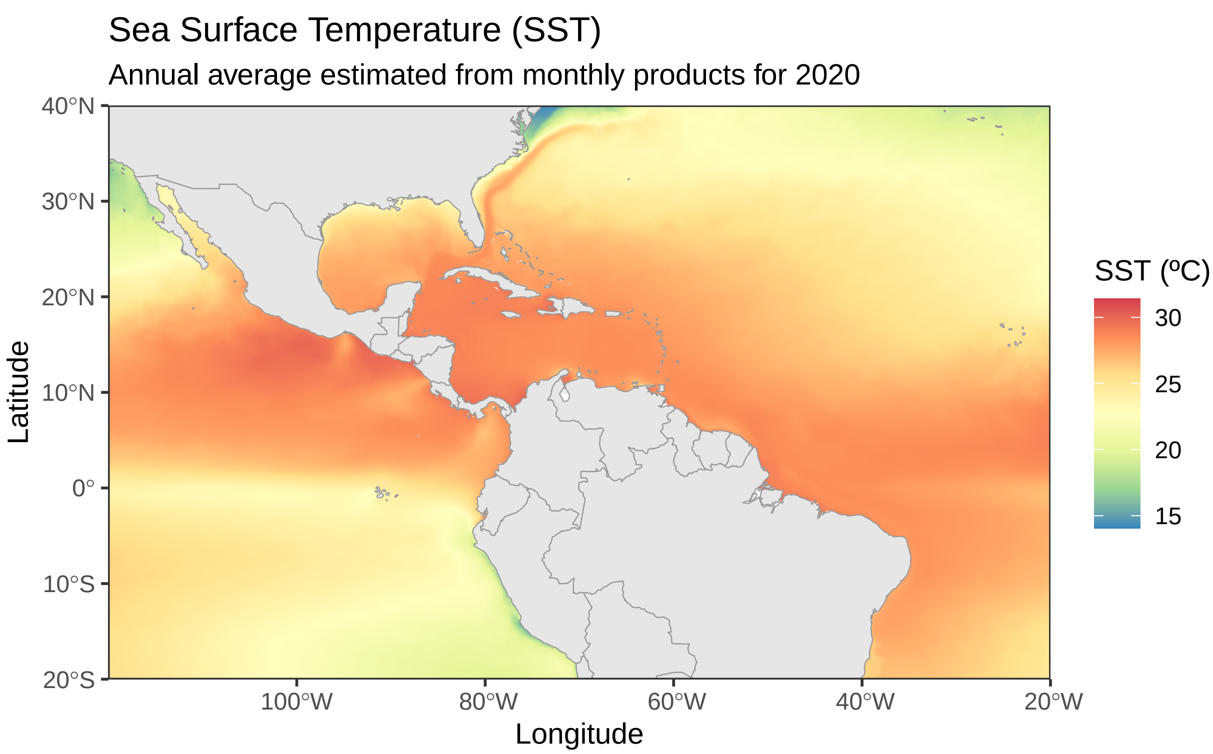
```
# convert raster to data.frame
sst_df <- as.data.frame(sst_mean, xy=TRUE, na.rm=TRUE)
```

Then, we will use the ***rnaturalearth*** package to download a global country map to use in the plot.

```
# import countries layer from Natural Earth
countries <- ne_countries(scale = "medium", returnclass = "sf")
```

Finally, we use the ggplot() function from ***ggplot2*** package to create a customized map. Note that each map component is added with a +. You can find further information about the ***ggplot2*** package here.

```
# plot
ggplot()+
  # add raster layer
  geom_raster(aes(x=x, y=y, fill=layer), data=sst_df) +
  # define color palette of raster layer
  scale_fill_distiller(palette = "Spectral", name = "SST (ºC)") +
  # add countries layers
  geom_sf(fill=grey(0.9), color=grey(0.6), lwd = 0.2, data=countries) +
  # define spatial extent
  coord_sf(xlim = range(sst_df$x), ylim = range(sst_df$y), expand = F, ndiscr = 500) +
  # labels
  labs(title = "Sea Surface Temperature (SST)",
       subtitle = "Annual average estimated from monthly products for 2020",
       x = "Longitude",
       y = "Latitude") +
  # theme
  theme_bw()
```



4.2 Extract values from a numerical model

Raster values can be sampled from any type of vector data (i.e. points, lines, polygons). In the marine realm, such type of vector data can be used to represent different objects:

- ***Points:*** Points can represent sampling locations, static platforms such as oceanographic buoys or moving objects such as animal tracks.

- ***Lines:*** Lines can be used to represent survey transects, shoreline, cables, pipelines.
- ***Polygons:*** This geometry type can be used to represent marine boundaries, such as marine protected areas (MPAs) or jurisdictional waters.

The function `extract()` offers a common approach to sample raster values from any of these three types of vector data. In the following example, we will use the boundaries of the Galapagos Islands marine protected area. This boundary has been acquired from the World Database on Protected Areas and converted to GeoPackage format (.gpkg). GeoPackage is an open standard used in GIS to store geoespatial information, and constitutes a modern alternative to the Shapefile format.

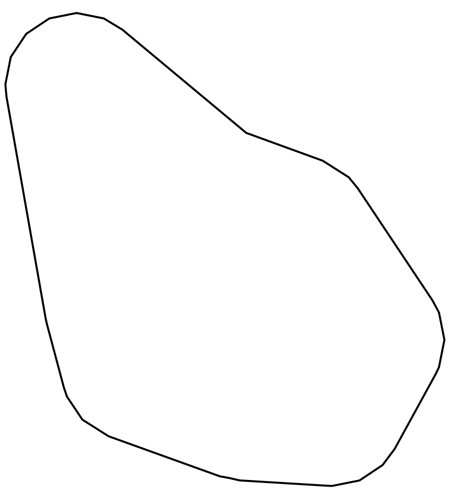
To import vector data in R, we will use the ***sf*** package.

```
# Import boundaries of Galapagos Islands Marine Protected Area
mpa <- st_read("data/GalapagosMPA.gpkg")
```

```
## Reading layer `GalapagosMPA' from data source
##   `/Users/dmarch/Git/Rworkshop-MarineData4America/data/GalapagosMPA.gpkg'
##   using driver `GPKG'
## Simple feature collection with 1 feature and 30 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -92.68288 ymin: -2.07455 xmax: -88.57545 ymax: 2.34955
## Geodetic CRS:  WGS 84
```

We can visualize the boundaries of Galapagos Islands MPA, with the basic `plot()` function.

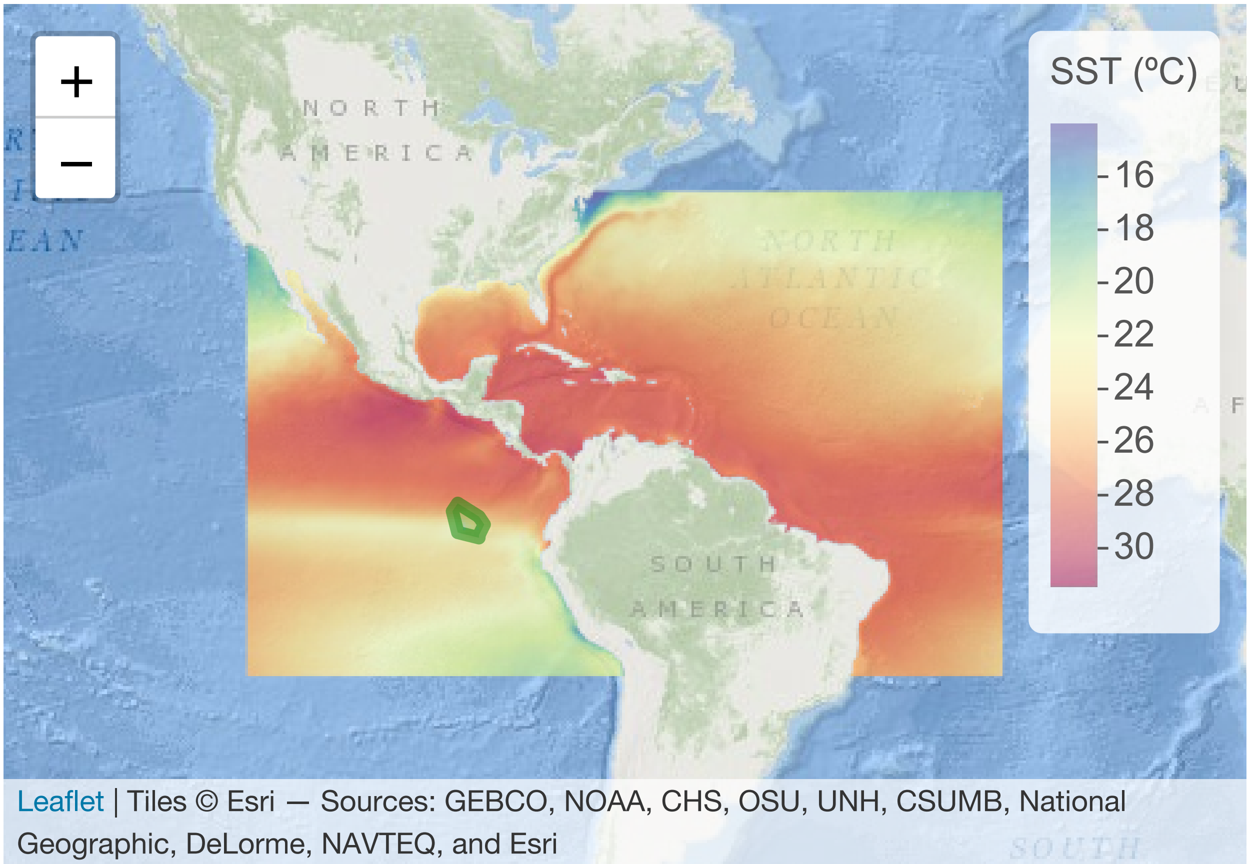
```
# basic plot
plot(st_geometry(mpa))
```



However, in this occasion, we lack context information. In order to improve the visualization, we will use the ***leaflet*** package, which provides dynamic visualizations and uses web services to add base maps. In this plot, we will combine the previous average map of SST with the new boundary of the MPA.

```
# Create color palette for CMEMS maps
palRaster <- colorNumeric("Spectral", domain = sst_mean@data@values, reverse = TRUE,
                          na.color = "transparent")

# Plot the boundary of MPA with a base map
leaflet(mpa) %>%
  # add base map
  addProviderTiles("Esri.OceanBasemap") %>%
```

```r
  # add raster map
  addRasterImage(sst_mean, colors = palRaster, opacity = 0.8) %>%
  # add legend
  addLegend(pal = palRaster, values = values(sst_mean), title = "SST (ºC)") %>%
  # add MPA boundary
  addPolygons(color = "green")
```



Note the ***leaflet*** package uses the pipe operator (`%>%`) to concatenate functions. This is similar to the `+`in a ***ggplot2*** statement.

In this last exercise, we will summarize raster values over polygonal areas, commonly referred to as ***zonal statistics***. We will use the `extract()` function from ***raster*** package

```r
# extract values from MPA and summarize values using the mean and standard deviation
mpa_sst_avg <- extract(sst_multi, mpa, fun=mean, na.rm=T)
mpa_sst_sd <- extract(sst_multi, mpa, fun=sd, na.rm=T)
```

We will combine the extracted data into a `data.frame`. But before that, we will also extract time data information form the raster to get the monthly dates.

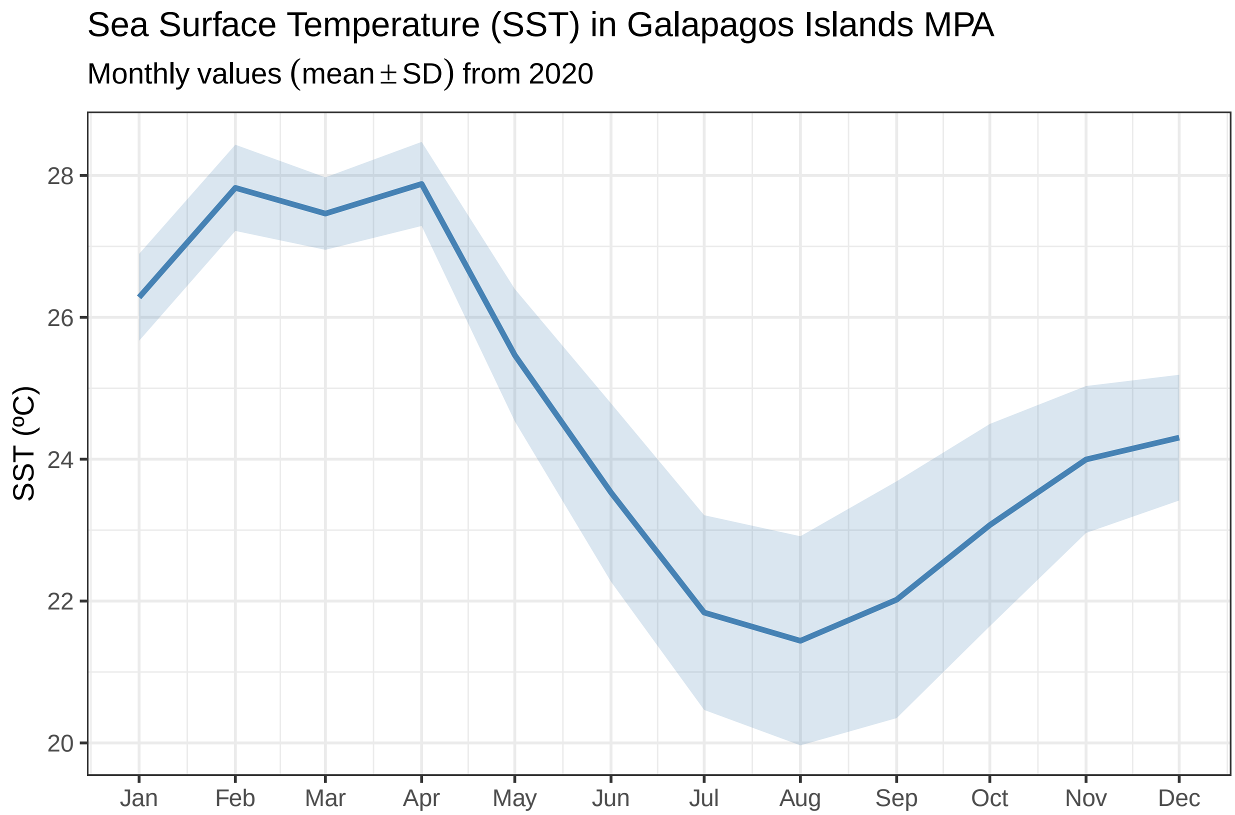
```r
# get date
date_sst <- sst_multi %>%
  # get time stamps from multi raster
  getZ() %>%
  # parse character to POSIXct class (time)
  parse_date_time("Ymd HMS") %>%
  # get the first day of each month
  floor_date("month")
```

```
# generate data.frame with three new columns (time, mean, sd)
mpa_sst <- data.frame(date = date_sst, sst_avg = c(mpa_sst_avg), sst_sd = c(mpa_sst_sd))

# inspect data.frame
mpa_sst
```

```
##          date  sst_avg    sst_sd
## 1  2020-01-01 26.28223 0.6124181
## 2  2020-02-01 27.82590 0.6077927
## 3  2020-03-01 27.46260 0.5108500
## 4  2020-04-01 27.88070 0.5926351
## 5  2020-05-01 25.46692 0.9326977
## 6  2020-06-01 23.52970 1.2574829
## 7  2020-07-01 21.83812 1.3727375
## 8  2020-08-01 21.43969 1.4742606
## 9  2020-09-01 22.01954 1.6695039
## 10 2020-10-01 23.07151 1.4256645
## 11 2020-11-01 23.99604 1.0358462
## 12 2020-12-01 24.30445 0.8860909
```

In order to visualize the extracted information, we will use again the package ***ggplot2*** to plot the time series.

```
# plot data
ggplot(mpa_sst, aes(x = date)) +
  # add ribbon to represent mean +- SD
  geom_ribbon(aes(ymin = sst_avg-sst_sd, ymax = sst_avg+sst_sd),  alpha=.2, linetype=0, fill="steelblue
  # add line to represent mean value
  geom_line(aes(y = sst_avg), size = 1, color="steelblue") +
  # define frequency of x-axis and date labels
  scale_x_datetime(date_breaks = "1 month", date_labels = "%b") +
  # plot labels labels
  labs(title = "Sea Surface Temperature (SST) in Galapagos Islands MPA",
       # note we use `expression()` to add +- symbol
       subtitle = expression(Monthly~values~(mean %+-% SD)~from~2020),
       x = "",
       y = "SST (ºC)") +
  # theme
  theme_bw()
```



5 Conclusion

This tutorial has provided a brief introduction to some of the potential capabilities of R to work with CMEMS data. Note that R can also be used to manipulate no only numerical model products, but also in-situ observations. In addition, it also offers the possibility to manipulate large datasets using cloud and parallel computing.