# Intro to Spatial Analysis in R

*Jamie Afflerbach*

*8/27/2015*

## Contents

## Introduction

This is an introduction to Spatial Analysis in R. I've developed this code based on some common questions people have asked me, or that I've asked myself, over the past year. There is a lot here to help you get started, but there is also **a lot** more to learn!

The focus here will be on raster analysis, rather than vector (shapefiles, polygons, polylines, points, etc.), although there's a bit of that too.

I've drafted this informal introductory session in the form of answering a scientific question. . . .

**Where are optimal sites for Red Tuna Crab (*Pleuroncodes planipes*) aquaculture within the California Current?**

We will answer this question by taking into consideration the following spatial data:
**1. Ocean Acidification**
**2. Sea Surface Temperature**
**3. Marine Protected Areas**

Key information:

- Red tuna crabs like water temperatures between **12 and 18 degrees Celsius**
- They also prefer water that has an aragonite saturation state between **2.4 and 3**

*Disclaimer: these are completely made up parameters. Also I highly doubt red tuna crabs would ever be a viable aquaculture or wild-caught species.*

---

## Setup

### Libraries & Settings

There are a lot of spatial packages for R, we will touch on some of them here but not all of them. Here is brief overview, taken from this site:

- **raster:** Reading, writing, manipulating, analyzing and modeling of gridded spatial data
- **rgdal:** Provides the most important and basic spatial functionalities. Provides bindings to Frank Warmerdam's Geospatial Data Abstraction Library (GDAL) (>= 1.6.3, < 2) and access to projection/transformation operations from the PROJ.4 library
- **sp:** provides classes and functions for spatial data
- **rgeos:** Interface to Geometry Engine – Open Source (GEOS) using the C API for topology operations on geometries.
- **maps:** Display of maps. I use this to add a map to my raster
- **maptools:** tools for reading and writing spatial data (visualisation)
- **ncdf4:** Use with NetCDF files. Note that the `raster` package is also able to read NetCDF files and I prefer to use Raster whenever possible.

```
library(raster)       #Main raster library with nearly all functions used in this analysis
library(rgdal)        #Spatial library - most functions used from rgdal are for vectors (shapefiles)
library(rasterVis)    #Useful for raster visualizations
library(maps)         #Has a database of maps. I use this to add a map to my raster to visualize land b

library(dplyr)        #NOT spatial - this is a data wrangling library
library(RColorBrewer) #Also not spatial - used to set the spectral color scheme
```

For raster data visualization I really like to use the spectral color scheme rather than the base graphics package. I'm also setting the plotting margins much smaller so that the plots will show up larger in the viewing pane.

```
cols = rev(colorRampPalette(brewer.pal(11, 'Spectral'))(255)) # rainbow color scheme

#setting margins for plotting
par(mar=c(2,2,1,1))
```

---

## Data Prep

### Vector data

#### (1) Reading in a shapefile

Using `readOGR` from the `rgdal` package, you can read in a shapefile.

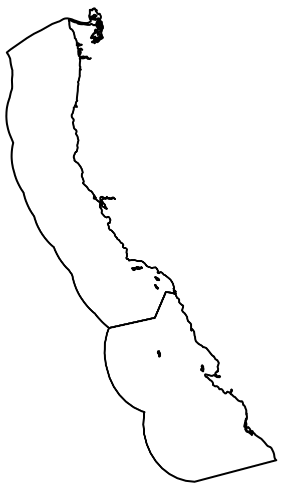
```
# dsn is the path name & layer is the name of the file. NOTE: you do not need to add an extension to th

cc = readOGR(dsn='data',layer='ca_current')
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "data", layer: "ca_current"
## with 2 features
## It has 7 fields
```

### (2) Plot a Shapefile

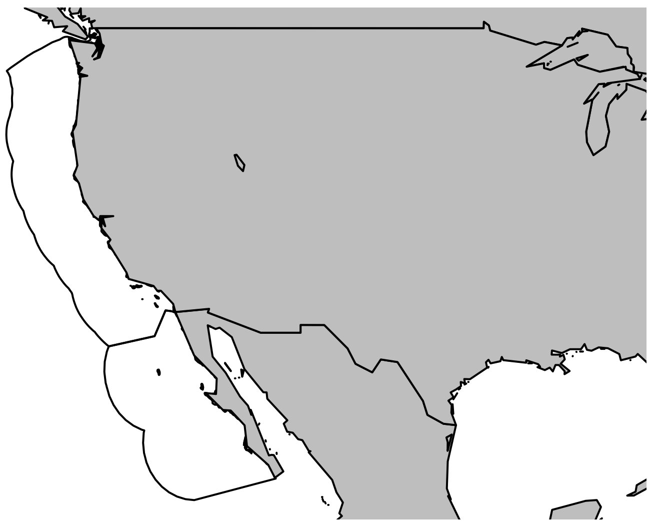
Plotting a shapefile is just as easy as:

```r
plot(cc)
```



And to add land to the map, do the following (from the `maps` package)

```r
plot(cc)
map('world',fill=T,add=T,col='gray')
```



The information in the summary of the shapefile is important if you need to understand what projection your **SpatialPolygonsDataFrame** is in, along with the extent and number of features. In this case there are just two features, the EEZs of the US West Coast and northern Baja Mexico. You can get this information by just typing in the name 'cc'

```r
cc
```

```
## class       : SpatialPolygonsDataFrame
## features    : 2
## extent      : -129.1635, -109.9721, 21.6185, 49.08721  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
## variables   : 7
## names       :                                   EEZ,       Country,  ID,     Sovereign, Remarks, Sov_
## min values  :       Mexican Exclusive Economic Zone,        Mexico, 135,        Mexico,      NA,
## max values  : United States Exclusive Economic Zone, United States, 163, United States,      NA,
```

### (3) Attribute Tables

You can look at the attribute table of a shapefile by calling …@data. The dataframe associated with the shapefile can be treated just as any other dataframe. Columns can be added, names can be changed, tables can be joined. *There are some caveats - such as filtering - that are important but I will not go into that here.*

```r
cc@data
```

```
##                                     EEZ       Country  ID     Sovereign
## 0 United States Exclusive Economic Zone United States 163 United States
## 1       Mexican Exclusive Economic Zone        Mexico 135        Mexico
##   Remarks Sov_ID Last_Chang
## 0    <NA>    163 30/03/2006
## 1    <NA>    135 30/03/2006
```

```r
#Add another column:

cc@data$short = c('USA','MEX')

cc@data
```

```
##                                     EEZ       Country  ID     Sovereign
## 0 United States Exclusive Economic Zone United States 163 United States
## 1       Mexican Exclusive Economic Zone        Mexico 135        Mexico
##   Remarks Sov_ID Last_Chang short
## 0    <NA>    163 30/03/2006   USA
## 1    <NA>    135 30/03/2006   MEX
```

---

## Raster Data

**Sea Surface Temperature** In the Data folder, there are 5 *.tif* files with the naming patterm `average_annual_sst_[year].tif`. To create a single average Sea Surface Temperature layer, do the following:

### (1) Read in the Sea Surface Temperature data

```r
sst_files = list.files('data',pattern='average_',full.names = T) #We need full file paths

sst_files
```

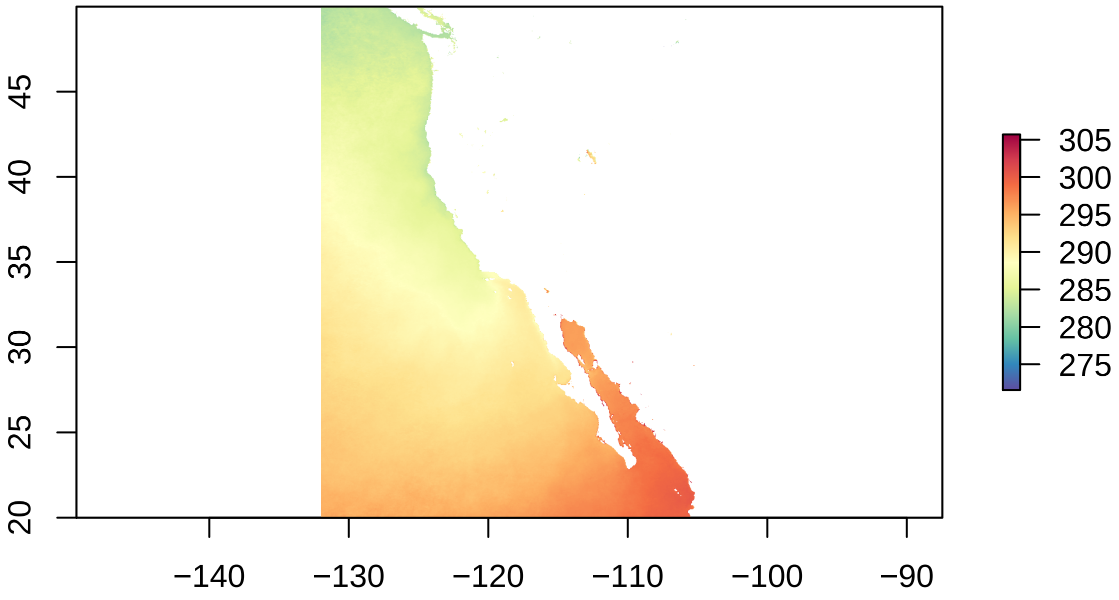
```
## [1] "data/average_annual_sst_2008.tif" "data/average_annual_sst_2009.tif"
## [3] "data/average_annual_sst_2010.tif" "data/average_annual_sst_2011.tif"
## [5] "data/average_annual_sst_2012.tif"
```

**(2) Visualize the data before running any calculation or analysis**

As a first step with new data I always recommend plotting it first and getting a sense of the data.

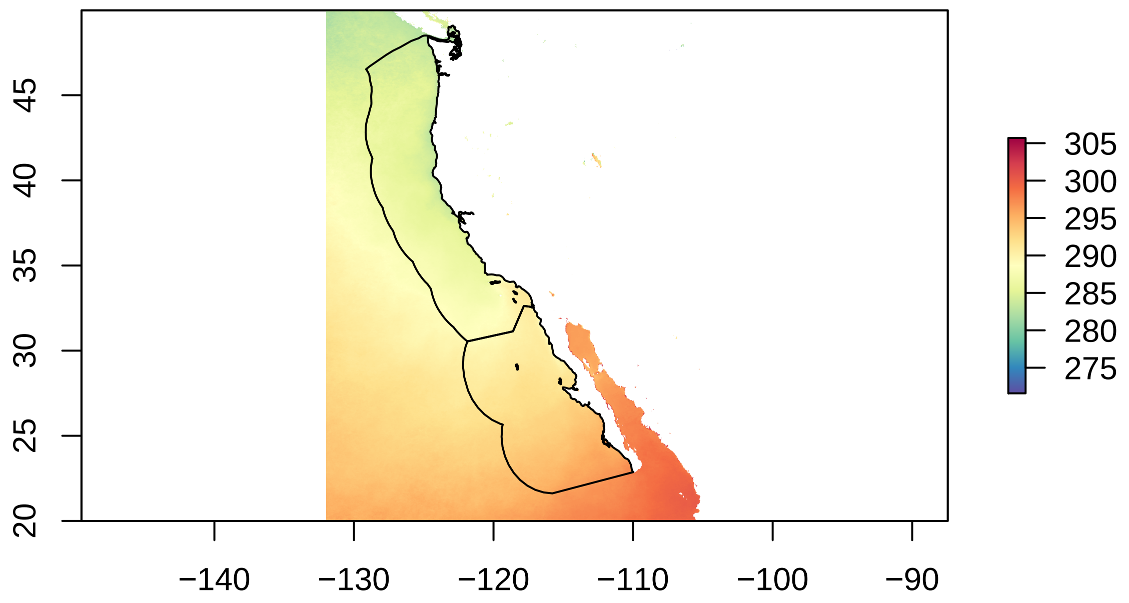
Create a raster of the first file and plot.
*Notice the data values are in Kelvin - we will change this to celsius later.*

```
r = raster(sst_files[1]) #This function reads in raster files. Think of it as similar to read.csv()

plot(r,col=cols) #remember cols was defined at the beginning to plot values in the spectral color schem
```



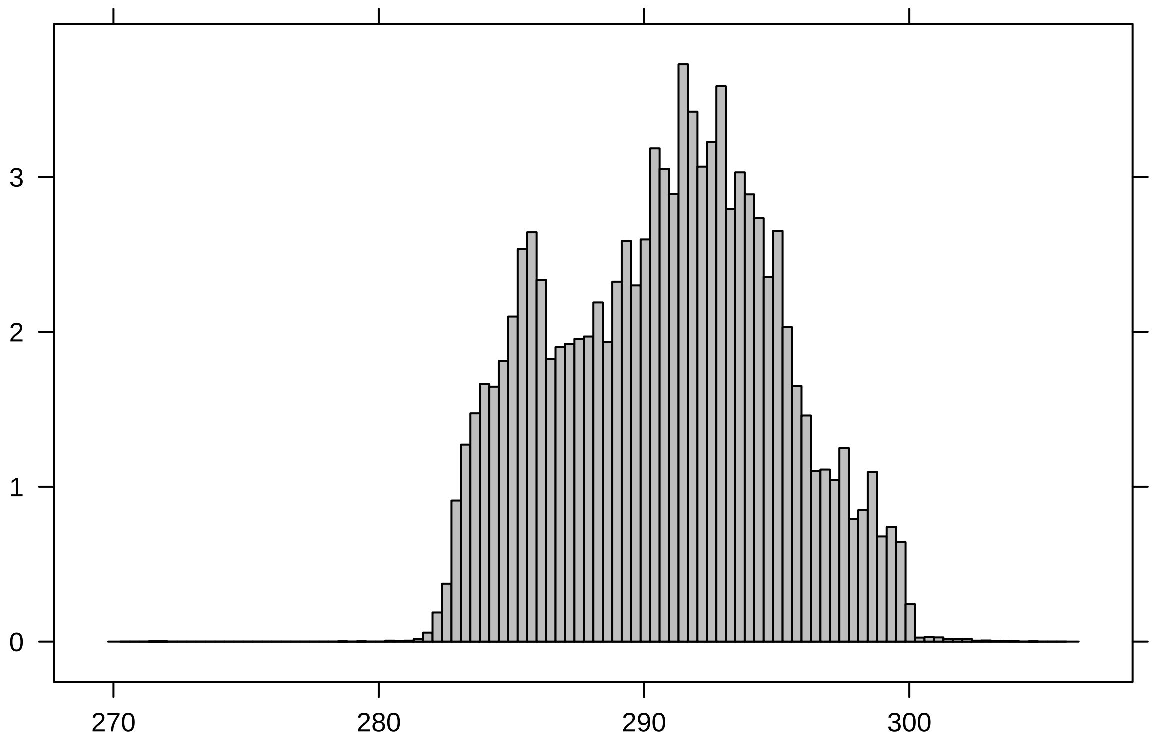
You can plot rasters or shapefiles ontop of each other, which can be really useful!

```
plot(r,col=cols)
plot(cc,add=T)
```



I also like to look at the distribution of data. Using the `histogram()` function from `rasterVis` is my preference over `hist()` from the base package. Purely because of the visual output.

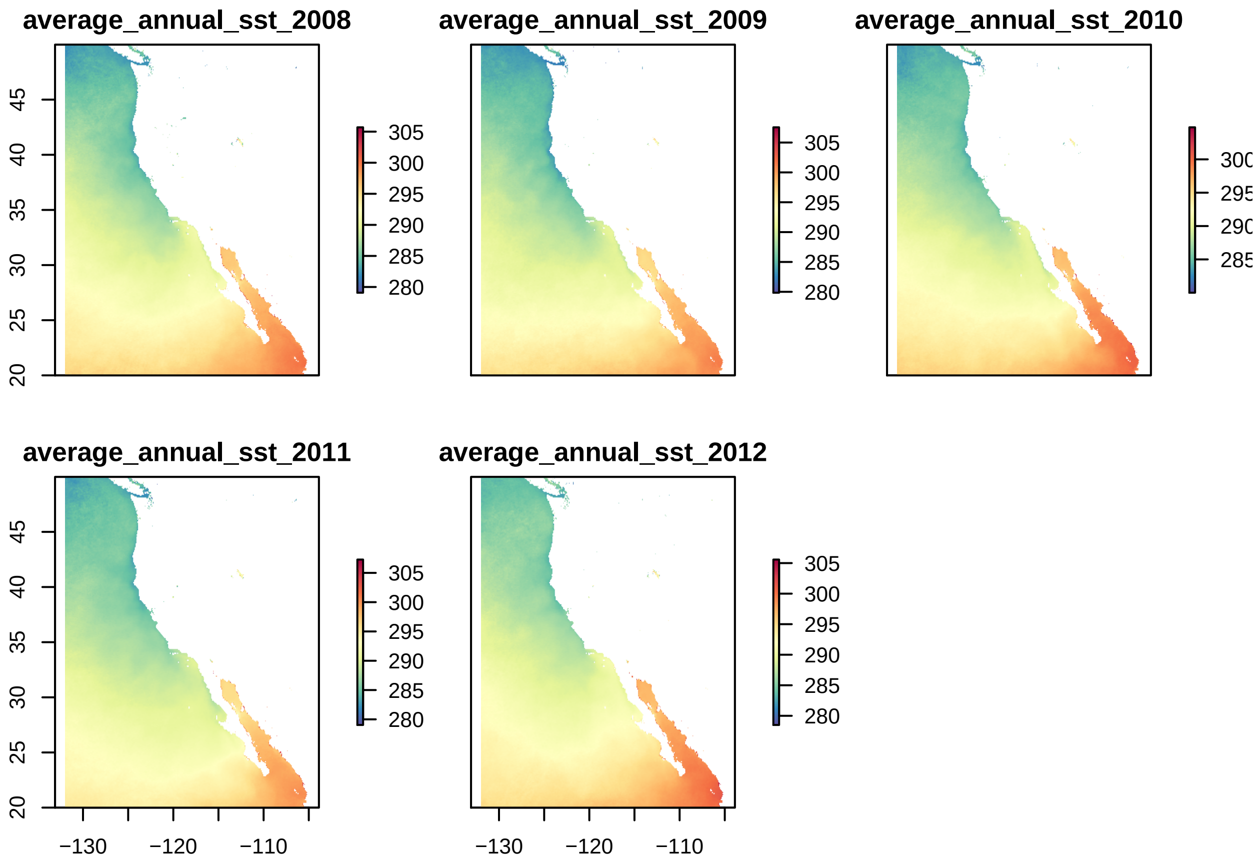
```
histogram(r)
```



**(3) Calculate average sea surface temperature over the last 5 years**

To get a single layer of average SST in degrees Celsius, first `stack` all layers and then calculate the average temperature per cell.

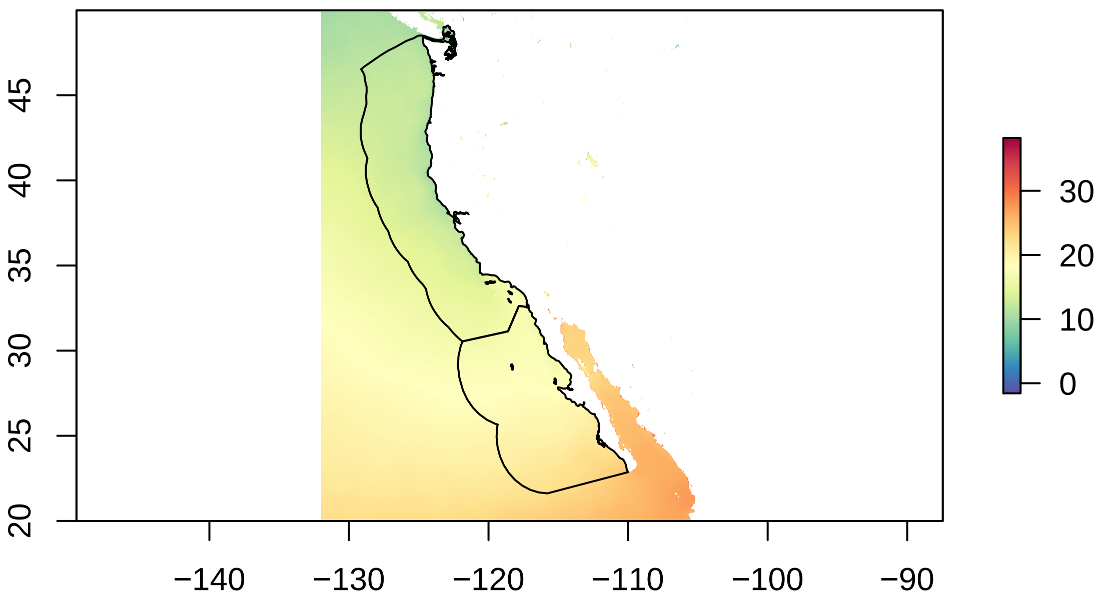
```
#stack is a function from the raster package that puts all RasterLayers into a RasterStack
sstStack = stack(sst_files)

plot(sstStack,col=cols)
```



You can perform operations on a RasterStack by using the `calc()` function from the `raster` package. `calc()` lets you define a function to apply across all layers in the stack.

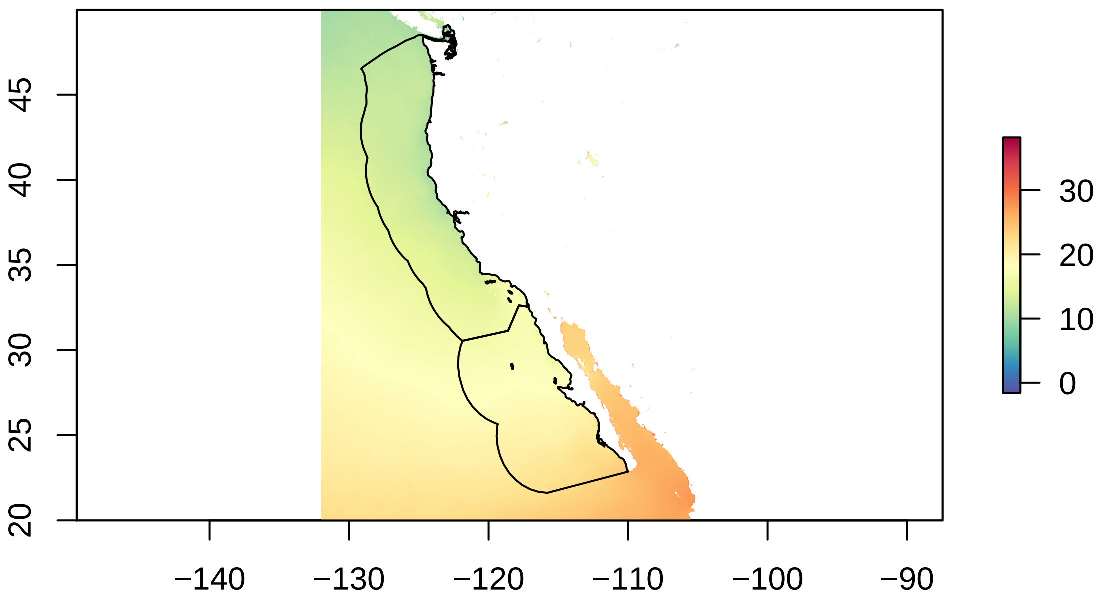
Here we will take the mean value per cell, and then convert to Celsius by subtracting 273.15.

```
sstAvg = calc(sstStack,fun=function(x){mean(x,na.rm=T)-273.15}) #calculate the average SST in Kelvin th

plot(sstAvg,col=cols);plot(cc,add=T)
```



A more compact way of doing multiple raster analysis is by using pipes…you can run `stack()` and `calc()` in one call!

```
sstAvg = stack(sst_files)%>%
          calc(.,fun=function(x){mean(x,na.rm=T)-273.15})

plot(sstAvg,col=cols);plot(cc,add=T)
```



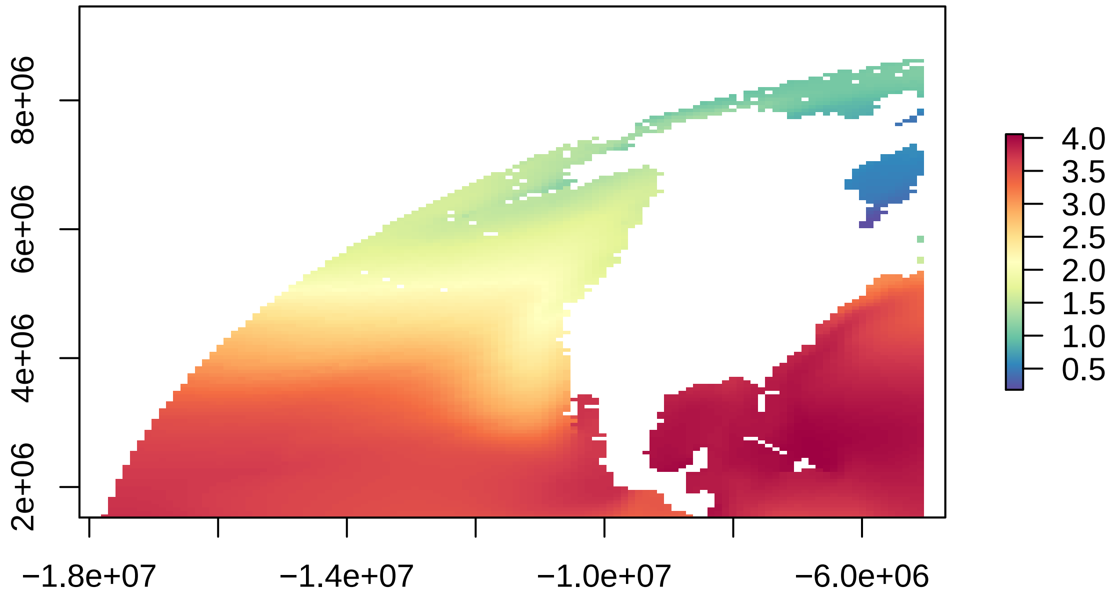
**Ocean Acidification (1) Read in the OA raster data**

Read in this data the same way as the SST data, using `raster()`. This data is the Omega aragonite saturation state of the water. For a frame of reference, values at or less than one are undersaturated in regards to aragonite and pose large problems for calcifying organisms.

```
oaAvg = raster('data/oa_avg.tif');oaAvg
```

```
## class       : RasterLayer
## dimensions  : 145, 114, 16530  (nrow, ncol, ncell)
## resolution  : 112000, 54700  (x, y)
## extent      : -17813502, -5045502, 1526148, 9457648  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=moll +lon_0=0 +x_0=0 +y_0=0 +ellps=WGS84 +units=m +no_defs
## data source : /home/afflerbach/GitHub/spatial-analysis-R/data/oa_avg.tif
## names       : oa_avg
## values      : 0.1787466, 4.056671  (min, max)
```

```
plot(oaAvg,col=cols)
```



You'll see that this is in a different projection, extent and cell size from the SST data. It is really obvious when you look at the plot, but the summary above also gives clues as to what projection/resolution/extent this data is in.

To do any sort of analysis using multiple rasters, they all need to be in the same extent, projection and cell resolution. T

First look at the differences:

```
sstAvg
```

```
## class       : RasterLayer
## dimensions  : 720, 648, 466560  (nrow, ncol, ncell)
## resolution  : 0.04166185, 0.04165702  (x, y)
## extent      : -131.9848, -104.9879, 19.99537, 49.98842  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +ellps=WGS84 +no_defs
## data source : in memory
## names       : layer
## values      : -1.57, 38.24  (min, max)
```

```r
oaAvg
```

```
## class       : RasterLayer
## dimensions  : 145, 114, 16530  (nrow, ncol, ncell)
## resolution  : 112000, 54700  (x, y)
## extent      : -17813502, -5045502, 1526148, 9457648  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=moll +lon_0=0 +x_0=0 +y_0=0 +ellps=WGS84 +units=m +no_defs
## data source : /home/afflerbach/GitHub/spatial-analysis-R/data/oa_avg.tif
## names       : oa_avg
## values      : 0.1787466, 4.056671  (min, max)
```

To get the ocean acidification data in the same format as the SST data, we need to reproject, crop and resample the data.

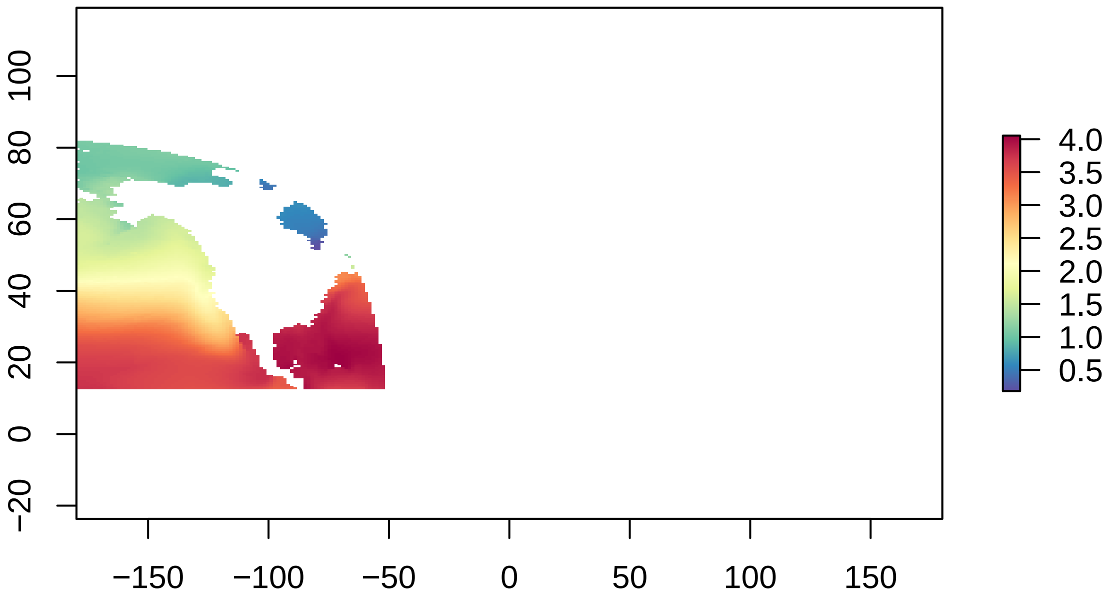
**(2) Reproject**

Use `projectRaster()` from the raster package to reproject a RasterLayer from one projection to another. You will need to define what the new projection should be by setting a coordinate reference system.

*Note:* Defining a **coordinate reference system (crs)** can be done in many ways. See Melanie's great cheat sheet for more details about Coordinate Reference Systems.

Here, we want to project from *Mollweide* to *longlat*

```r
oa_proj = projectRaster(oaAvg,crs = ('+proj=longlat'))#see a warning about finite points
plot(oa_proj,col=cols)
```

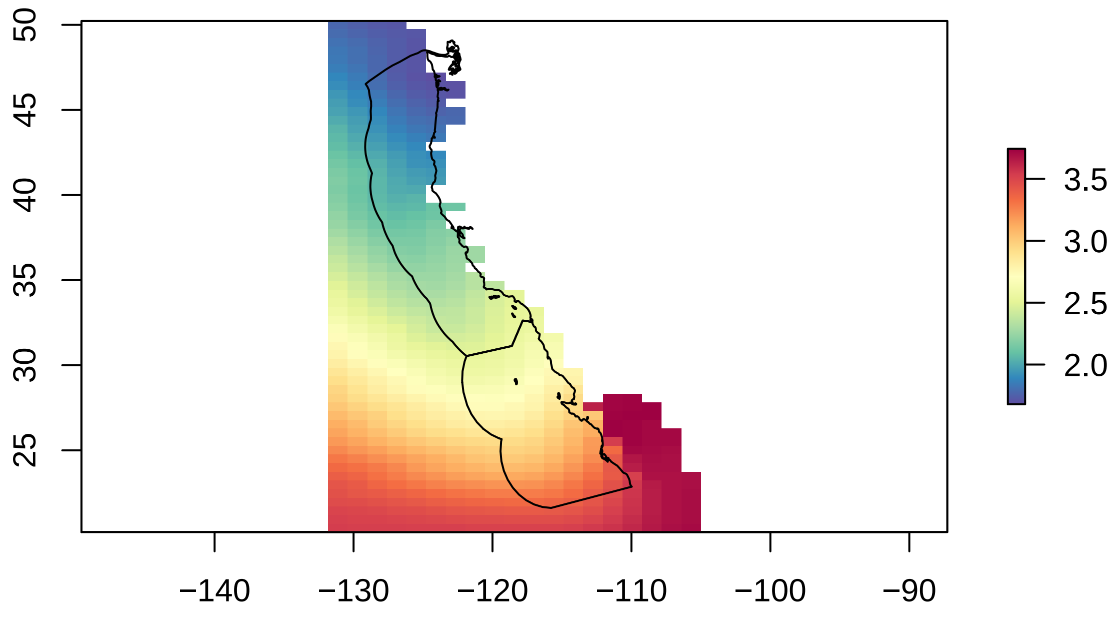


**(3) Crop**

Now that the layer is in the right projection, we need to crop it to our study area. You can set whatever extent you want using `extent()` but for two raster layers to line up, they must have the same extent.

You can use another raster object to define the extent (in this case we will use sstAvg)

```r
oa_crop = crop(oa_proj,sstAvg) #crop oa_proj to the extent of sstAvg
plot(oa_crop,col=cols);plot(cc,add=T)
```

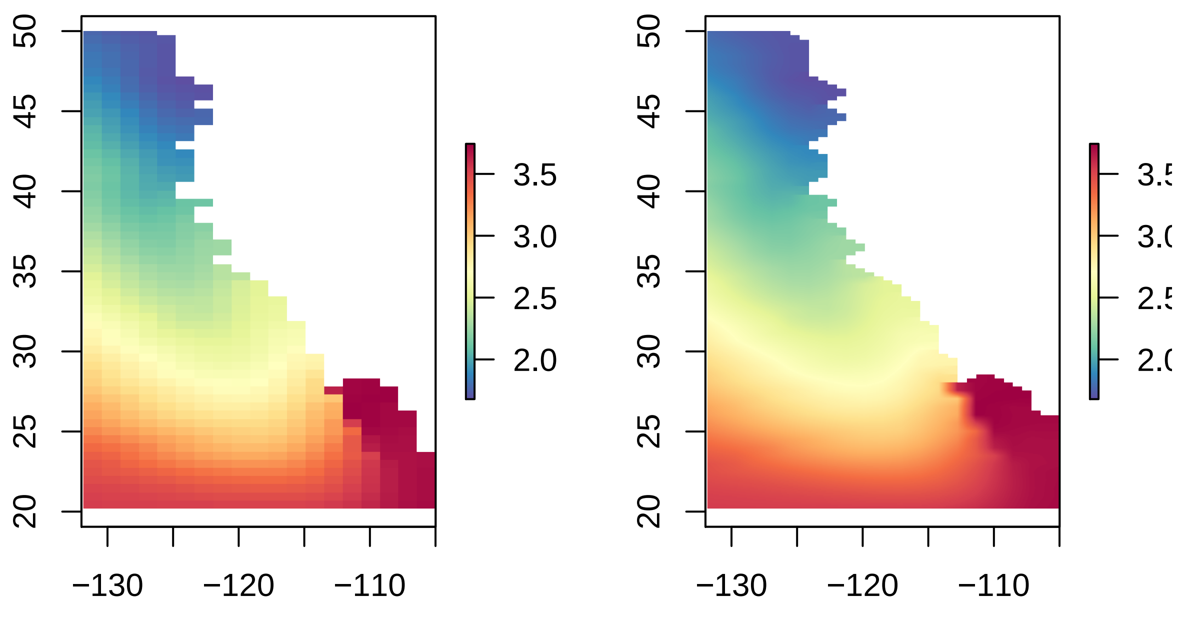


**(4) Resample**

Just by plotting both the SST and OA data, you can tell right away that these two datasets have different cell resolutions. The OA data needs to be resampled to the same cell size as SST in order to do any sort of analysis on these two. Use the *nearest neighbor* method to avoid interpolation between points. We want to keep the data values the same as the original data, but at a higher resolution.

Here you can see the difference:

```
#include progress='text' in any raster function to see a progress bar as the function runs! This is one
oa_res = resample(oa_crop,sstAvg,method='ngb')#,progress='text')
oa_bil = resample(oa_crop,sstAvg,method='bilinear')#,progress='text')

par(mfrow=c(1,2))
plot(oa_res,col=cols);plot(oa_bil,col=cols)
```

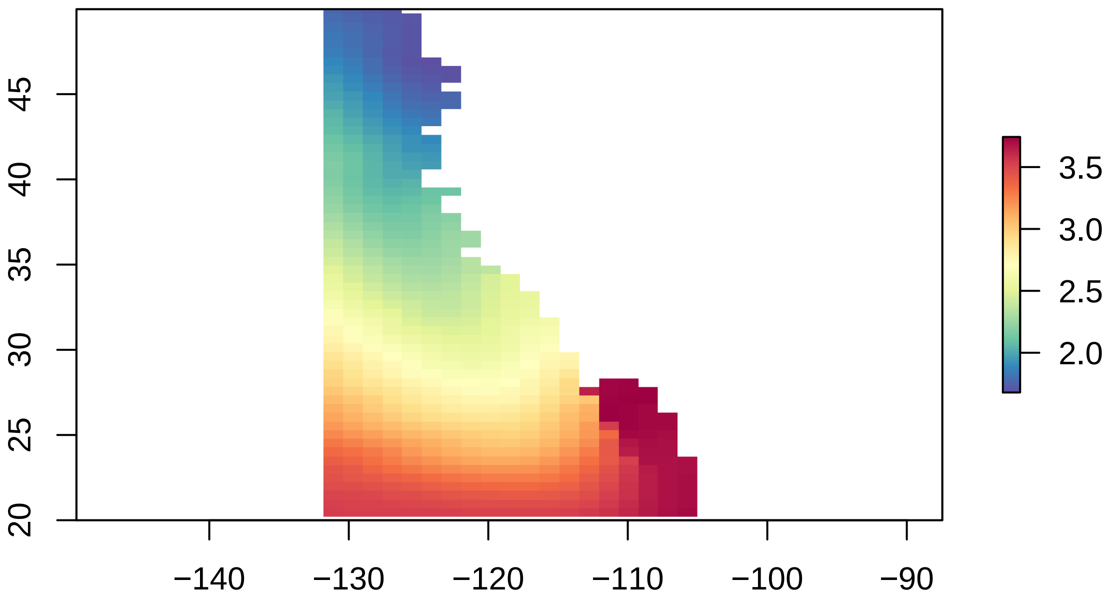


```
#dev.off()
```

*NOTE: Typically you'll want to disaggregate cells to match data of a higher resolution. Otherwise, if we aggregate the cells from the SST data, we would lose data.*

Again we can condense this script by using pipes!

```
oa = projectRaster(oaAvg,crs=('+proj=longlat'))%>%
        crop(.,sstAvg)%>%
          resample(.,sstAvg,method='ngb')#,progress='text')

plot(oa,col=cols)
```



**(5) Check prepped data**
Check to see that we can use the SST and OA data together now

```
stack(oa,sstAvg) #No error and the stack has two layers so it looks good!
```

```
## class       : RasterStack
## dimensions  : 720, 648, 466560, 2  (nrow, ncol, ncell, nlayers)
## resolution  : 0.04166185, 0.04165702  (x, y)
## extent      : -131.9848, -104.9879, 19.99537, 49.98842  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +ellps=WGS84 +no_defs
## names       :    oa_avg,     layer
## min values  :  1.678623, -1.570000
## max values  :  3.743462, 38.240000
```

## Analysis

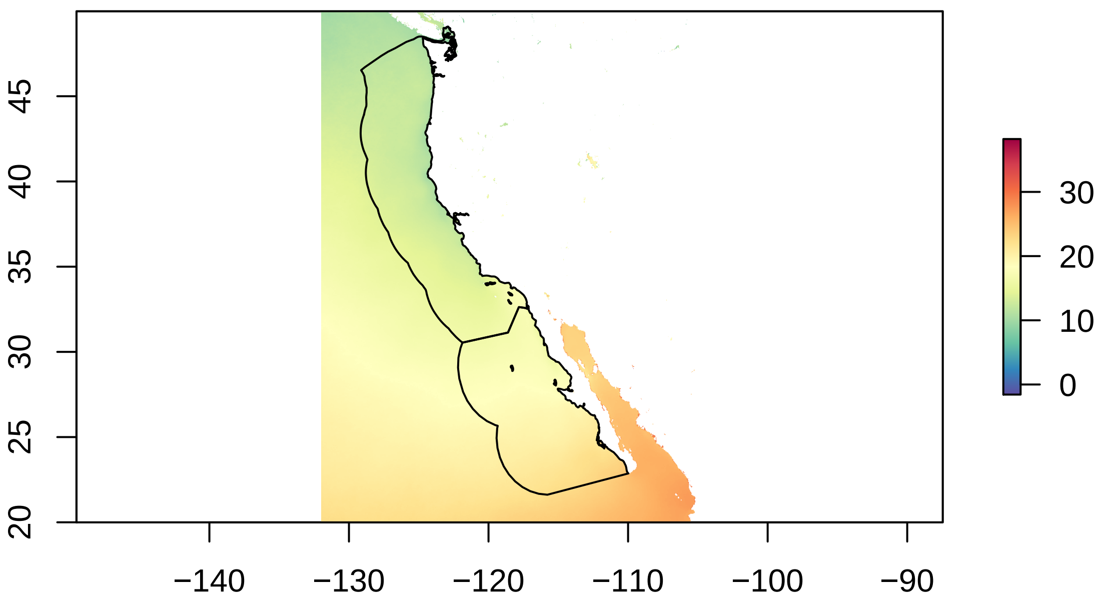
**Select cells**

**(1) Remove all cells from the Sea Surface Temperature layer that fall out of the species temperature range**

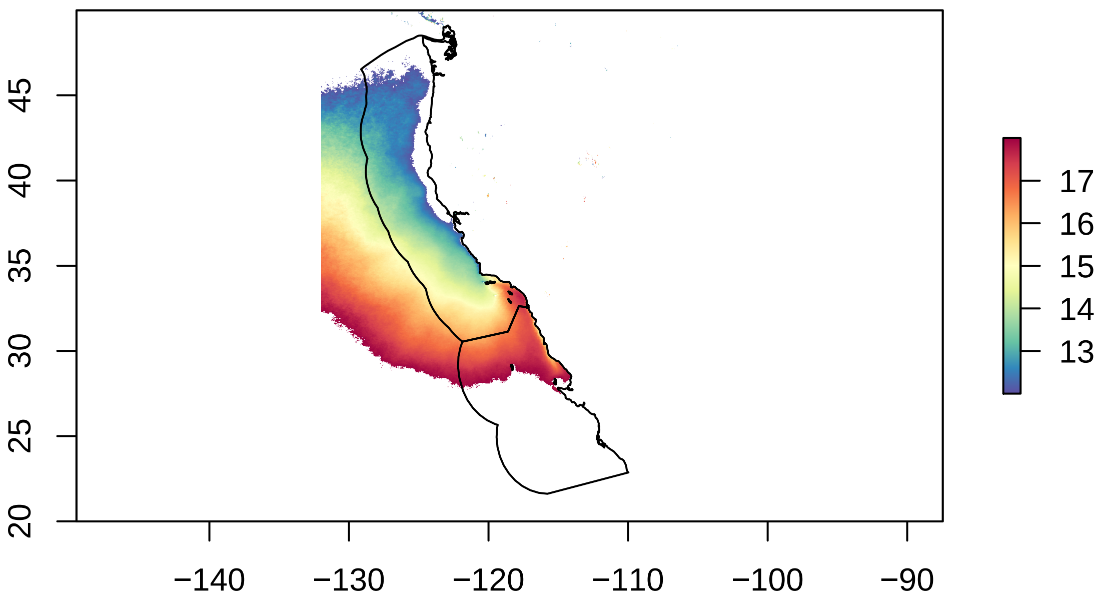
The Red Tuna Crab grows best in waters that are **between 12 and 18 degrees Celsius.**

Remembering that `sstAvg` is our current SST layer, you can eliminate all cells with values outside of your range in a few different ways. I prefer to do a simple subset using brackets, but first assigning a new variable so you don't overwrite `sstAvg`

```
plot(sstAvg,col=cols);plot(cc,add=T)
```



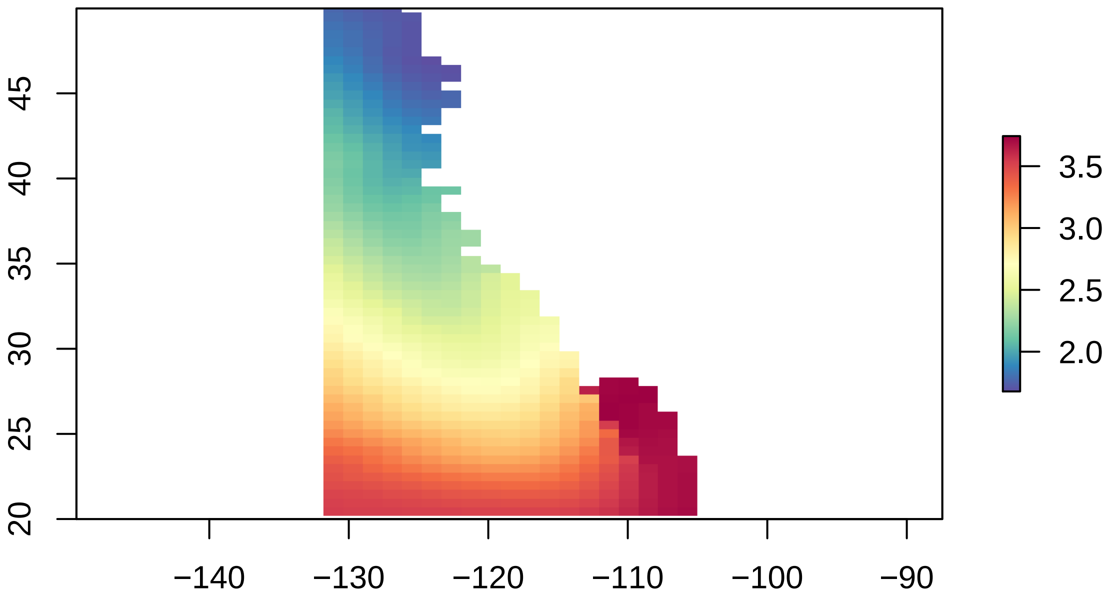
```
# trying something like this gives you all of the cell values that fit your condition. Not exactly what
#sstPref=sstAvg[sstAvg>=12 & sstAvg<=18]

#I prefer the following

sstPref = sstAvg
sstPref[sstPref<12 | sstPref>18]<-NA #set all cells outside of our range to NA
plot(sstPref,col=cols);plot(cc,add=T)
```



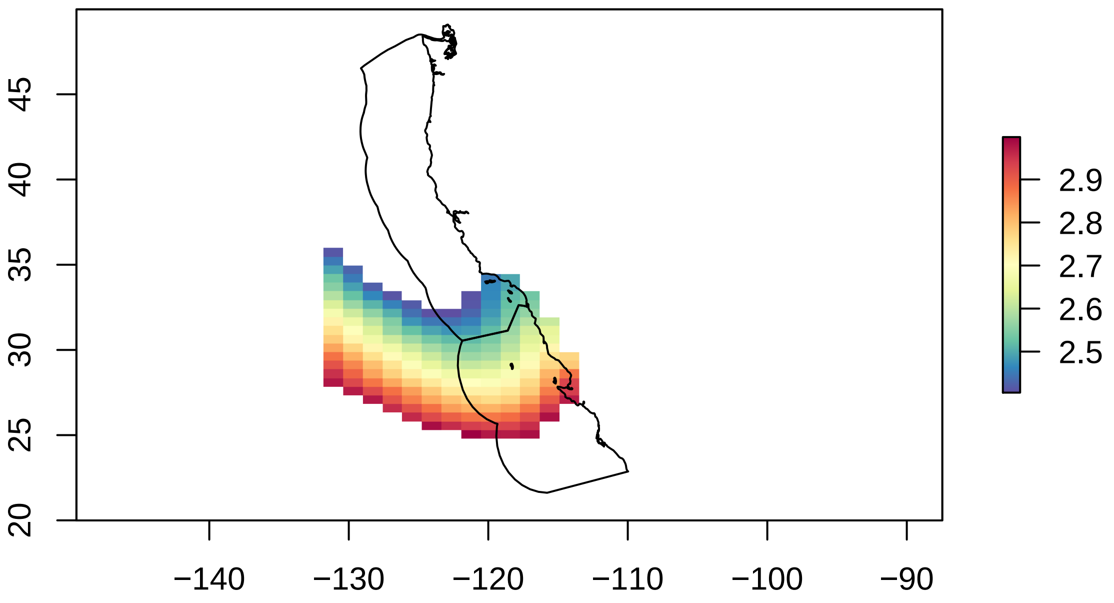
**(2) Remove all cells from the Ocean Acidification layer that fall out of the species preferred range**

Our species also prefers water that has an aragonite saturation state **between 2.4 and 3**

```r
plot(oa,col=cols)
```



```r
oaPref = oa
oaPref[oaPref<2.4|oaPref>3]<-NA
plot(oaPref,col=cols); plot(cc,add=T)
```

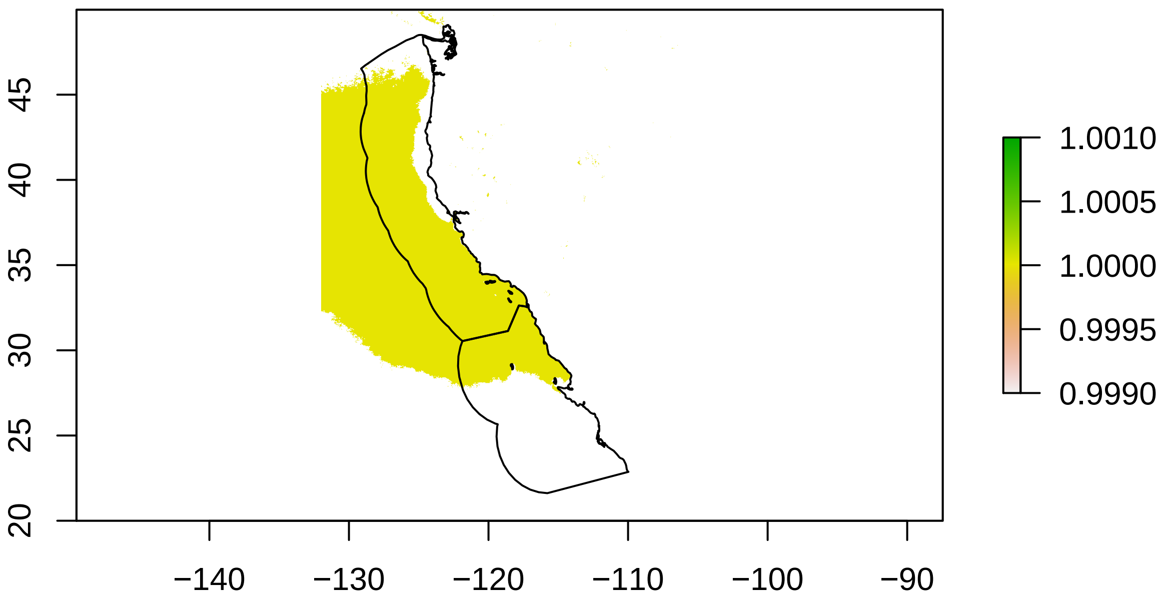


**(3) Change all remaining cell values to 1**

Now that we have our two rasters with suitable parameters, there are a lot of different ways to select overlapping cells. Here is the way I think about it:

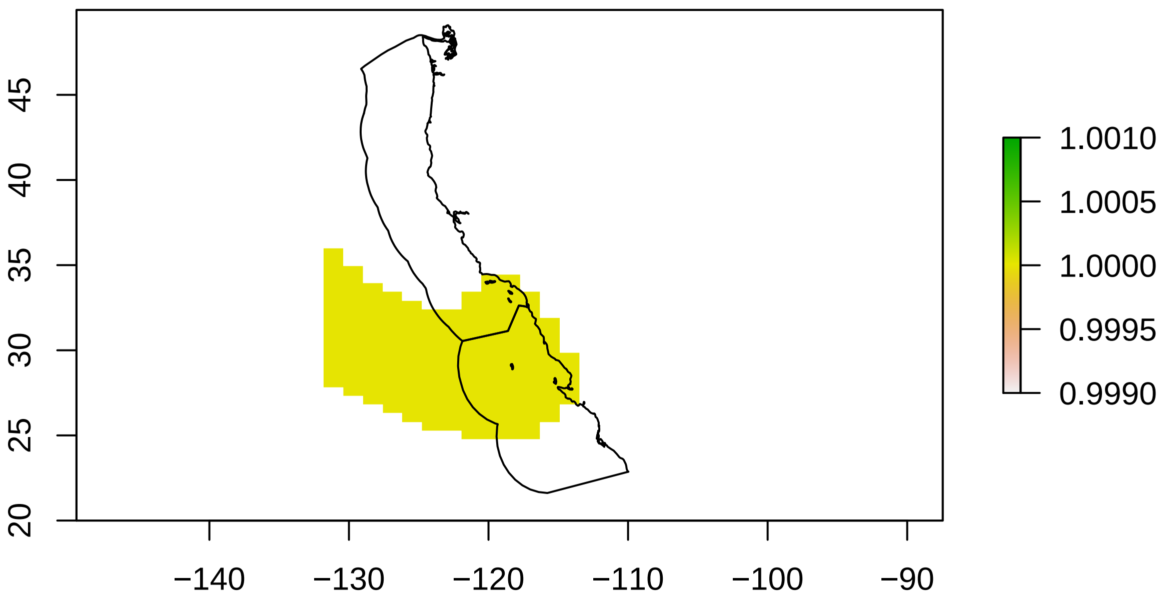
Set all viable cells in SST to equal 1

```
sstBin = sstPref             #assigning new variable sstBin (binary) so that sstPref is not overwritten
sstBin[!is.na(sstBin)]<-1    #setting all cells that are not NA to 1
plot(sstBin);plot(cc,add=T)
```



…and same for OA

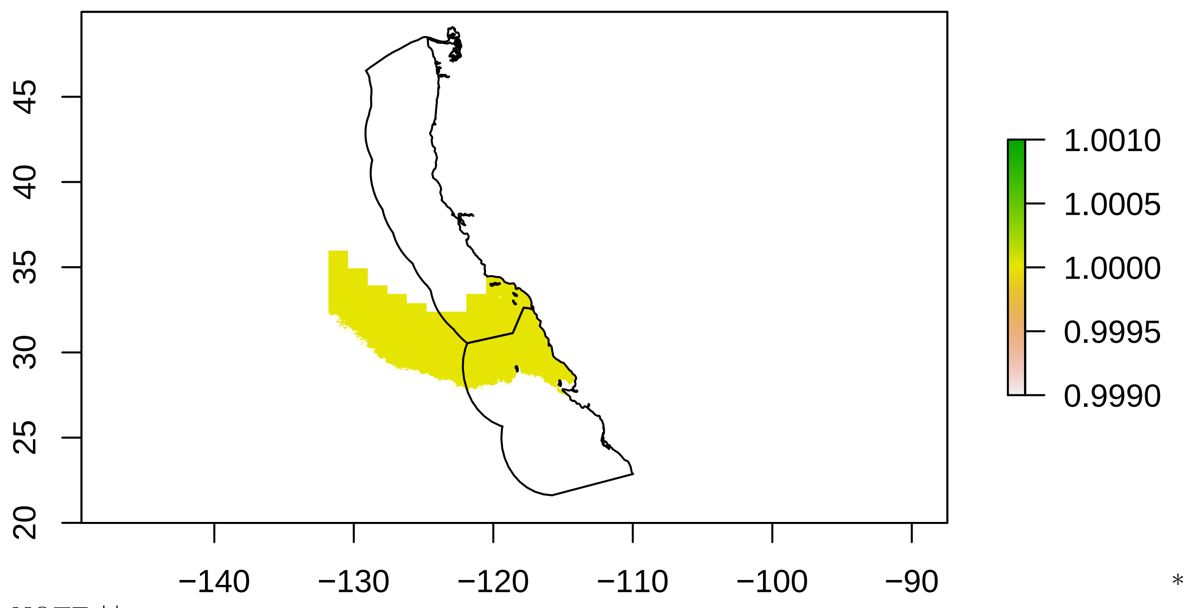
```
oaBin = oaPref
oaBin[!is.na(oaBin)]<-1
plot(oaBin);plot(cc,add=T)
```



Now that we have these two binary layers, we can combine them by summing all cells from both layers and those cells with a value equal to 2 will be our viable cells!

Multiply the two rasters together using `overlay()` from the raster package and the resulting cells, equal to 1, are the cells that meet both SST and OA requirements

```
cells = overlay(sstBin,oaBin,fun=function(x,y){x*y})
plot(cells);plot(cc,add=T)
```

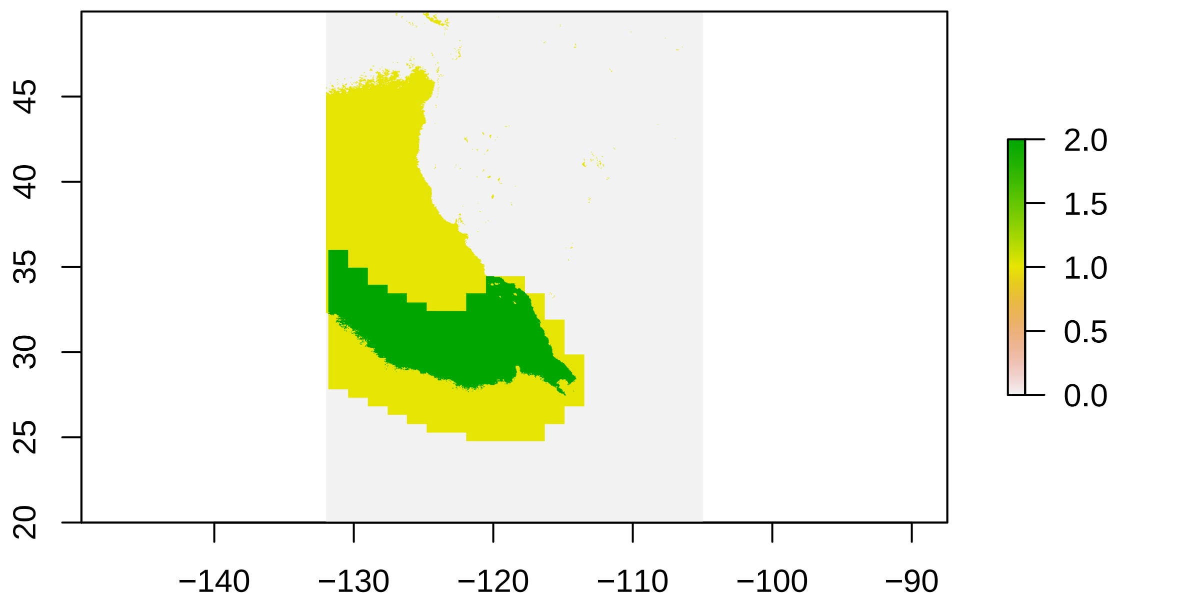


* SIDE

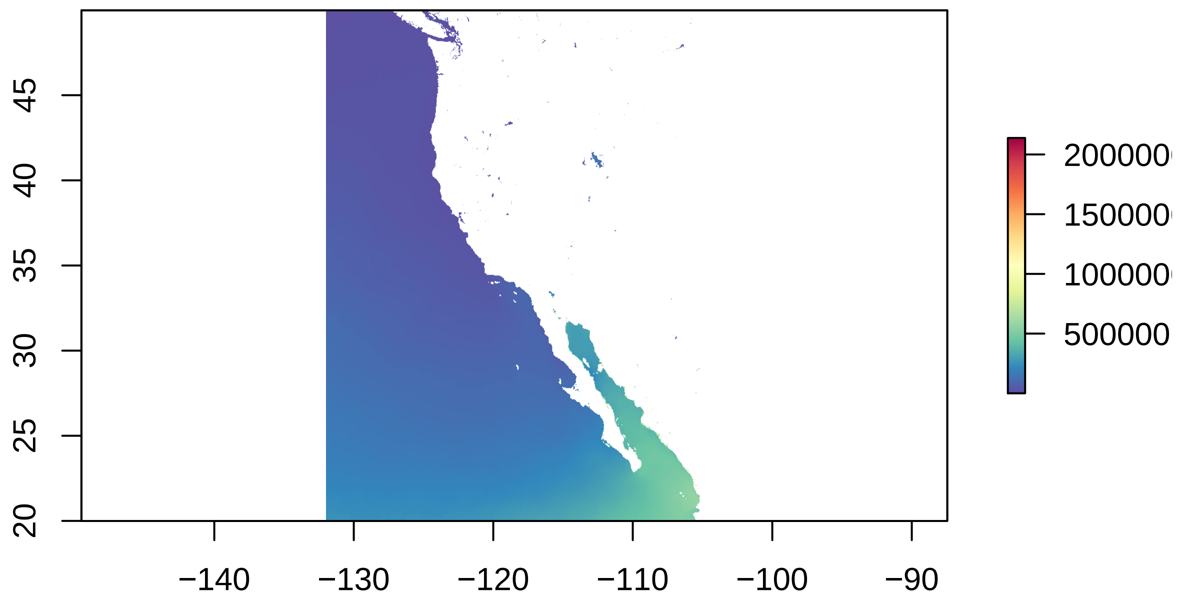
NOTE:**

You can perform mathematical operations on single or multiple raster layers using base R functions. Both `calc()` and `overlay()` are useful when you have complex functions working on these layers. Here are some examples of how you can use raster layers with base R functions:

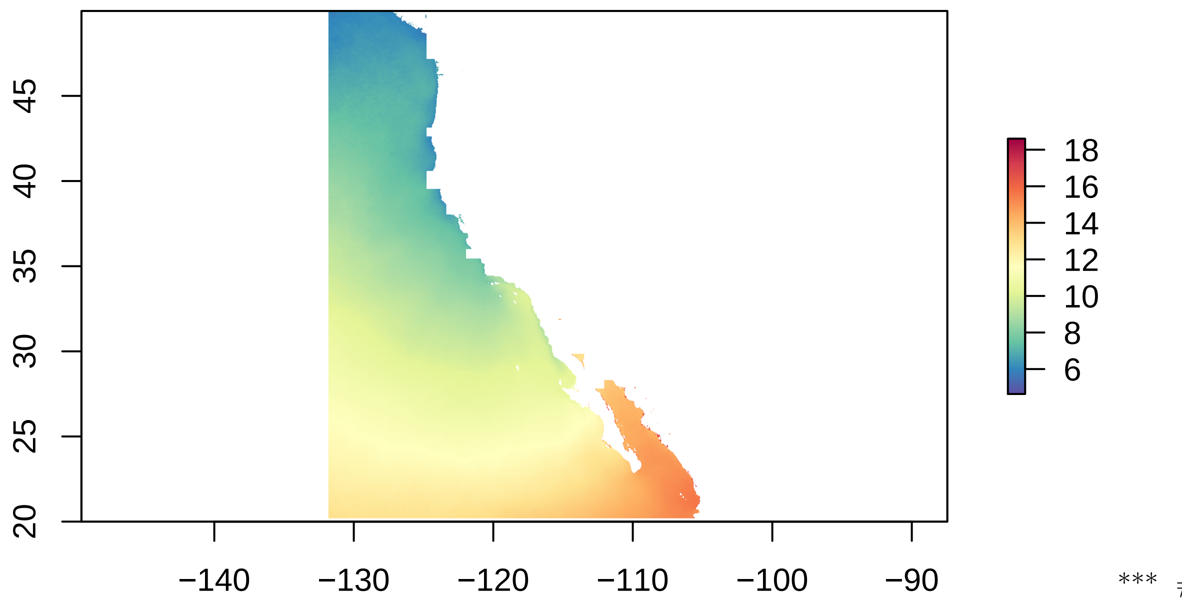
```
#sum
sum = sum(sstBin,oaBin,na.rm=T);plot(sum) #gives you cells equal to 0, 1 and 2.
```



```
#power
power = sstAvg^4;plot(power,col=cols)
```



```r
#average
avg = mean(sstAvg,oa);plot(avg,col=cols)
```



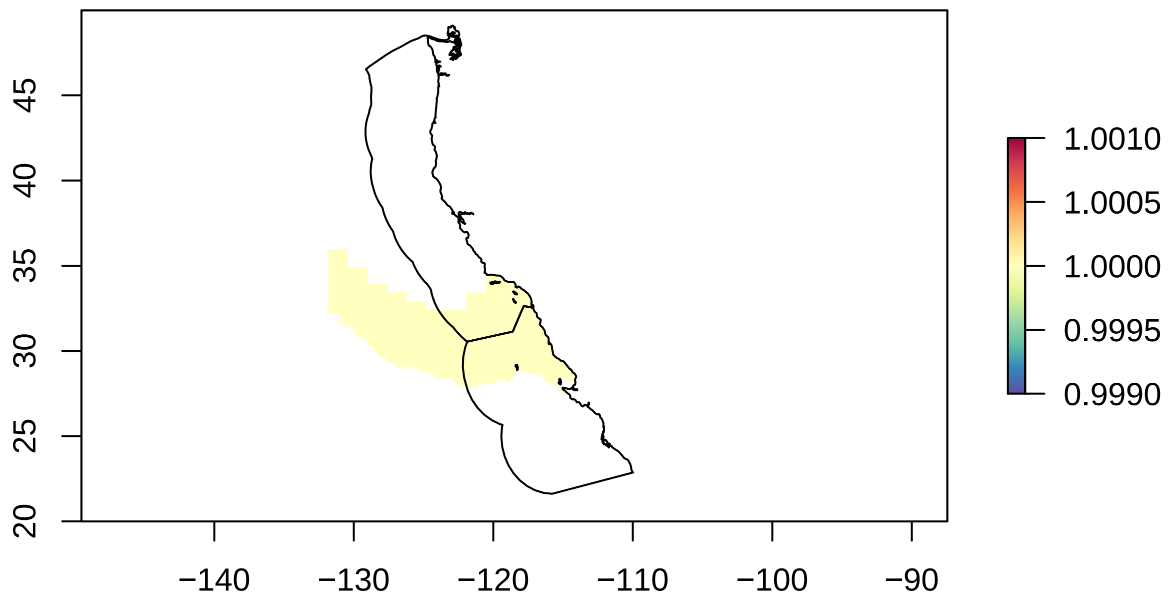
*** #Additional Functions

This could be all you need - but I want to show some additional steps to highlight more functionality in R.

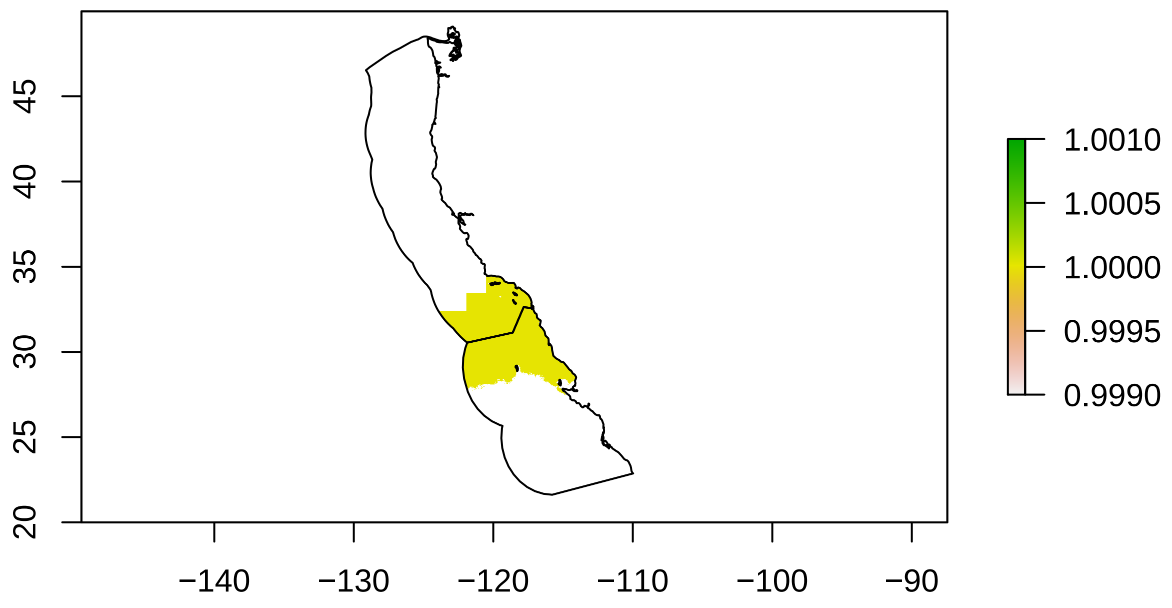
### Mask

You can remove cells outside of your region by using the `mask()` function. Here, you only want to keep cells that are within the Mexican or US EEZs.

```r
plot(cells, col=cols);plot(cc,add=T)
```



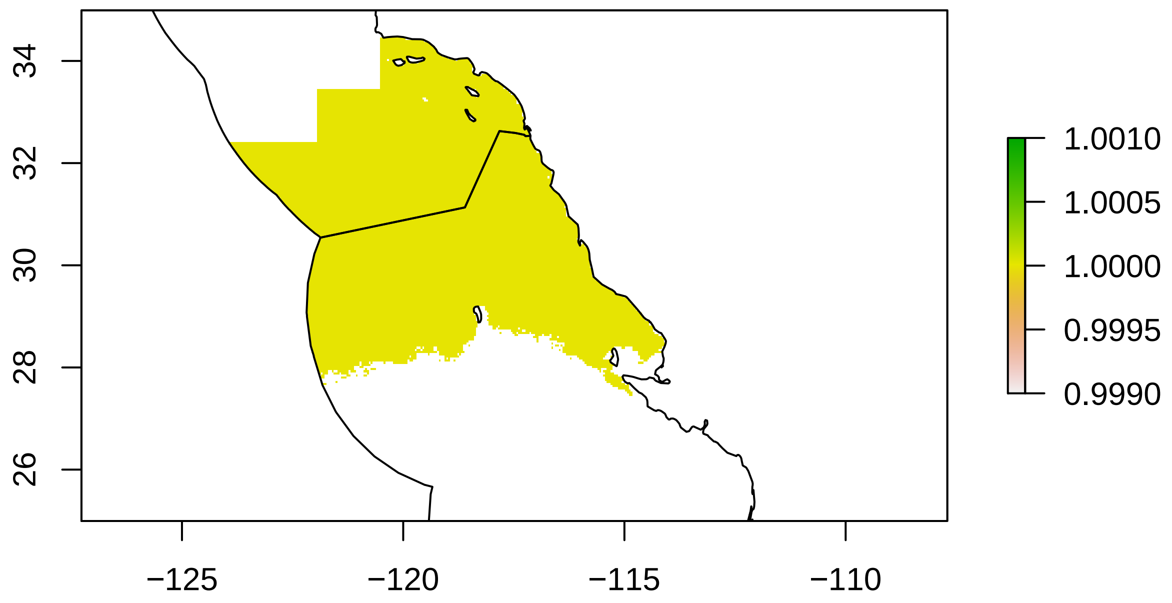
```
cellsEEZ = mask(cells,cc)

plot(cellsEEZ);plot(cc,add=T)
```



**Crop to region**

You can then crop your raster to a smaller extent

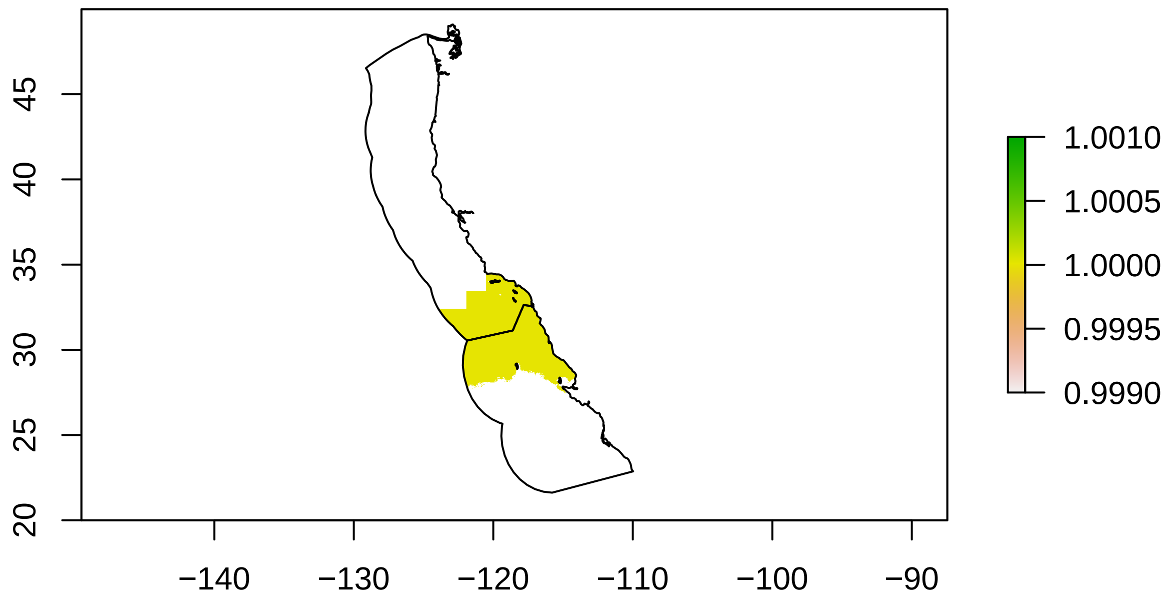
```
cellsCrop = crop(cellsEEZ,extent(-125,-110,25,35)) #setting extent by eyeing it
plot(cellsCrop);plot(cc,add=T)
```



**Draw Extent**

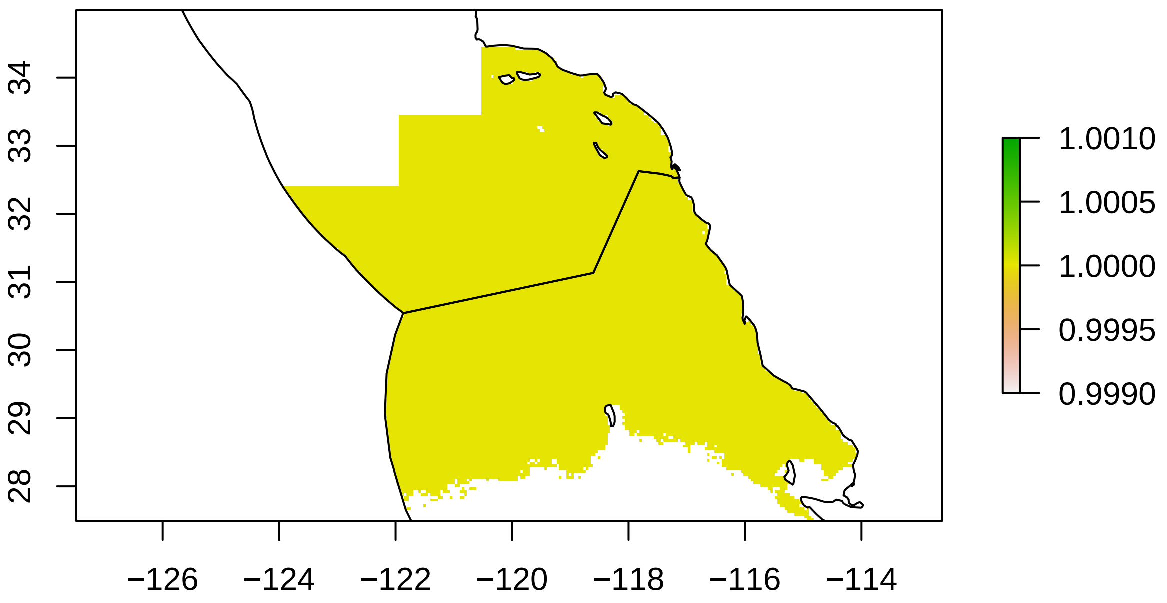
Another nifty thing - if you don't know the extent, you can draw it and R will tell you! You can then save this extent and crop to it

```
plot(cellsEEZ);plot(cc,add=T)
ext <- drawExtent(show=TRUE,col='red')
```



```
#class       : Extent
#xmin        : -126.729
#xmax        : -113.3571
#ymin        : 27.48709
#ymax        : 35.00825
```

```r
ext=extent(-126.729,-113.3571,27.48709,35.00825) #I picked this one,
cropCells=crop(cellsEEZ,ext)
plot(cropCells);plot(cc,add=T)
```



**Mask out Marine Protected Areas and National Marine Sanctuaries**

There are two shapefiles in the **data** folder. One is 'MPA_State' which are state MPAs, and one is 'National Marine Sanctuaries'. To go along with the aquaculture analysis, lets say that cells within these regions must be excluded from the suitability analysis.

```r
#marine protected areas
mpa = readOGR(dsn='data',layer='MPA_State',verbose=F);mpa
```

```
## class       : SpatialPolygonsDataFrame
## features    : 101
## extent      : -373440.2, 259220.6, -590425.8, 259522.3  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=aea +lat_1=34 +lat_2=40.5 +lat_0=0 +lon_0=-120 +x_0=0 +y_0=-4000000 +datum=NAD83
## variables   : 7
## names       :                  NAME,            Type, Length_nmi,                 CCR,    Area_nmi, Pe
## min values  :          Abalone Cove,            SMCA,       0.00,    Section 632 (b)1,  0.01000000,
## max values  : White Rock (Cambria), Special Closure,       9.66, Yet to be published,  9.86000000,
```

```r
plot(mpa)
```

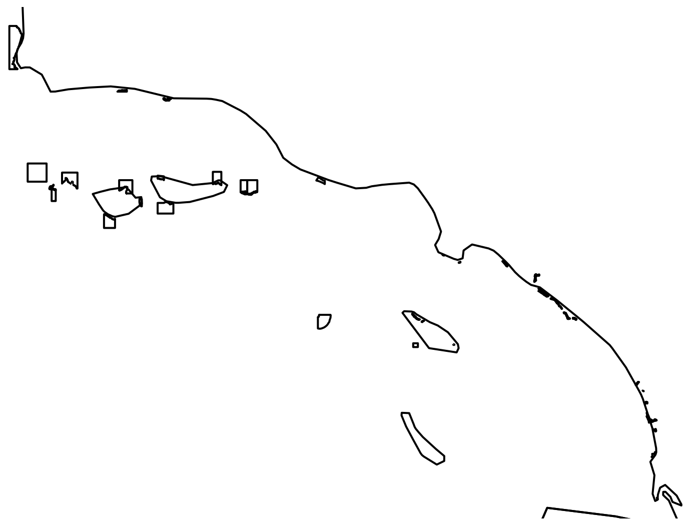
```r
# try to crop to our extent
mpa_c = crop(mpa,ext);mpa_c
```

```
## NULL
```

Notice that when creaing `mpa_c` there is no error. But when you call it the output is NULL, indicating a NULL object. This is because the MPA shapefile is **not in the same projection**

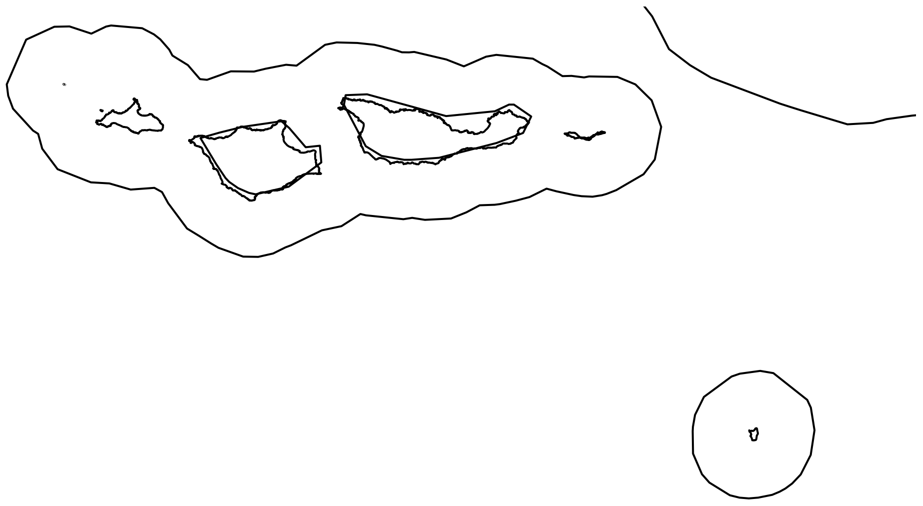
Using `spTransform()` from the `rgdal` package, you can project a shapefile in a similar manner as raster.

```r
mpa = spTransform(mpa,CRS("+proj=longlat")) #spTransform is part of rgdal package

#crop again
mpa_c = crop(mpa,ext);plot(mpa_c);plot(cc,add=T)
```



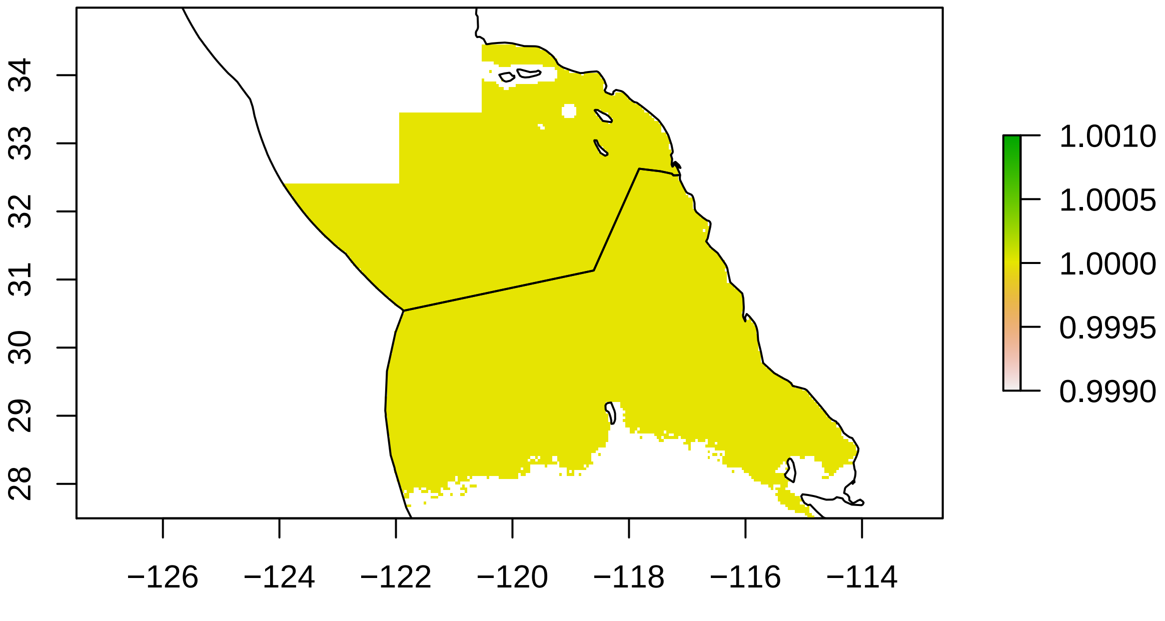
The same can be done with the National Marine Sanctuary shapefile, using pipes!

```
#national marine sanctuaries
nms = readOGR(dsn='data',layer='National Marine Sanctuaries',verbose=F)%>%
          spTransform(.,CRS("+proj=longlat"))%>%
            crop(.,ext)

plot(nms);plot(cc,add=T)
```



To remove cells in the MPAs and NMS just use the `mask()` function but set `inverse=T` so that all cells **outside** of the polygons are kept and those inside are set to NA.

```
cellsAQ = mask(cropCells,mpa_c,inverse=T)%>%
            mask(.,nms,inverse=T)

plot(cellsAQ);plot(cc,add=T)
```



### Rasterize & Zonal Stats

With any raster analysis, you likely aren't just creating a pretty map. Here is an example of running **zonal statistics** on a raster.

We want to look at the total area (km2) in Mexico and California for aquaculture of Red Tuna Crab.

#### (1) Rasterize shapefile

First you want to turn a shapefile (california current) into a raster of the same projection/extent/resolution as your cells.

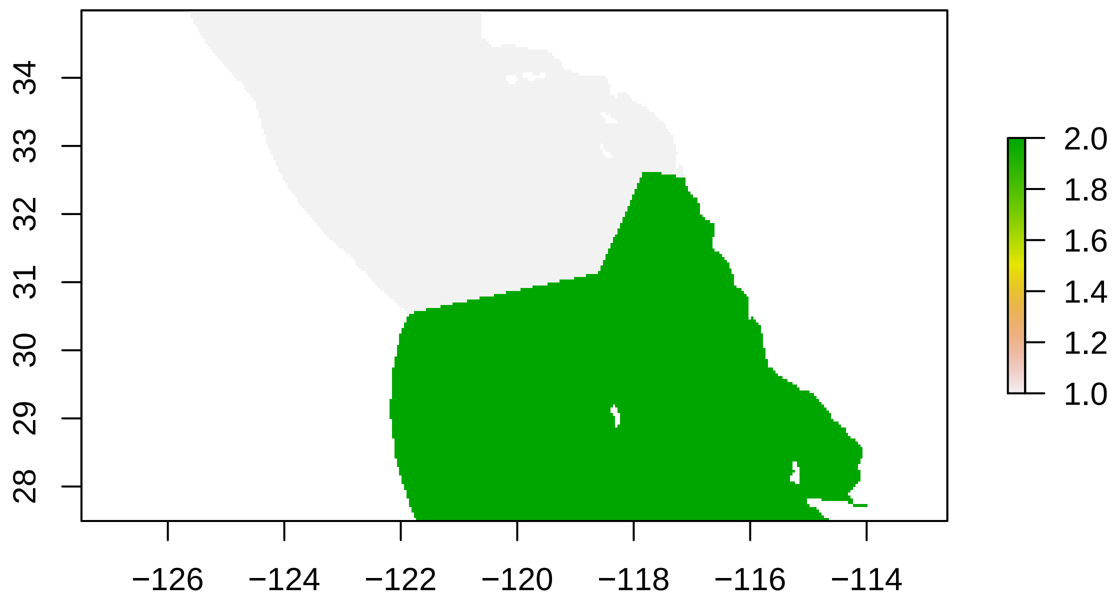
```
#let's take a look at what the CC dataframe looks like again
head(cc)
```

```
##                                         EEZ       Country  ID     Sovereign
## 0 United States Exclusive Economic Zone United States 163 United States
## 1       Mexican Exclusive Economic Zone        Mexico 135        Mexico
##   Remarks Sov_ID Last_Chang short
## 0    <NA>    163 30/03/2006   USA
## 1    <NA>    135 30/03/2006   MEX
```

```
ccZones = rasterize(cc,cellsAQ)#,progress='text')
ccZones
```

```
## class       : RasterLayer
## dimensions  : 180, 321, 57780  (nrow, ncol, ncell)
## resolution  : 0.04166185, 0.04165702  (x, y)
## extent      : -126.7354, -113.3619, 27.49363, 34.99189  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 1, 2  (min, max)
## attributes  :
##  ID                                     EEZ       Country ID.2     Sovereign
##   1 United States Exclusive Economic Zone United States  163 United States
##   2       Mexican Exclusive Economic Zone        Mexico  135        Mexico
##  Remarks Sov_ID Last_Chang short
##     <NA>    163 30/03/2006   USA
##     <NA>    135 30/03/2006   MEX
```

```
plot(ccZones)
```

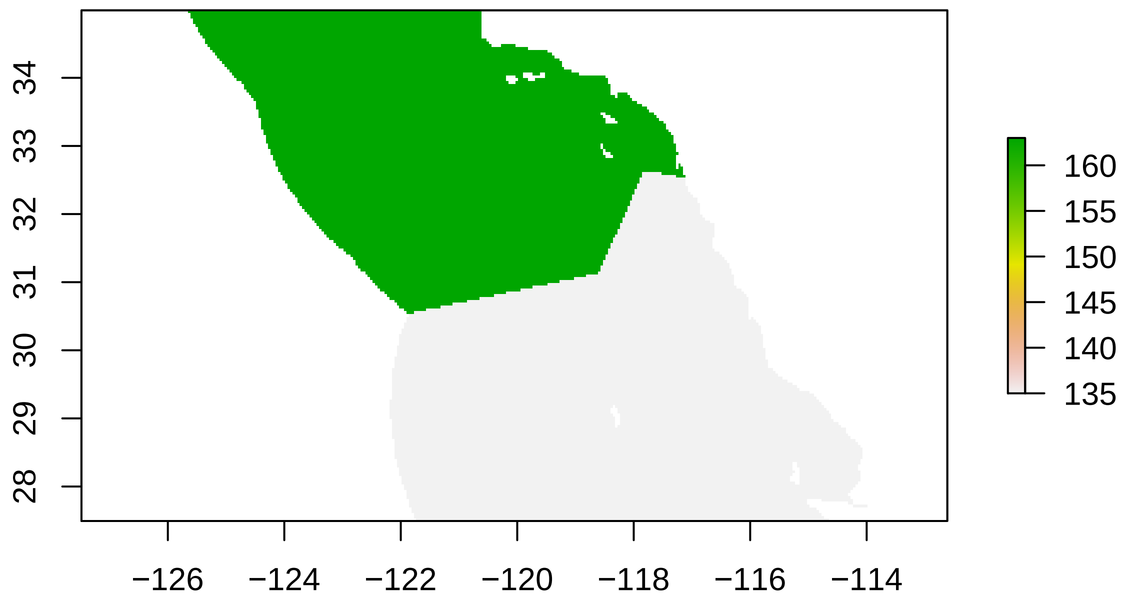


R automatically assigned 1 and 2 as the cell values. You can set your own values too based on a field or another defined vector of ids.

```
#using a field from the shapefile

cc_ras_ID = rasterize(cc,cellsAQ,field="ID");cc_ras_ID
```

```
## class       : RasterLayer
## dimensions  : 180, 321, 57780  (nrow, ncol, ncell)
## resolution  : 0.04166185, 0.04165702  (x, y)
## extent      : -126.7354, -113.3619, 27.49363, 34.99189  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 135, 163  (min, max)
```

```
plot(cc_ras_ID)
```

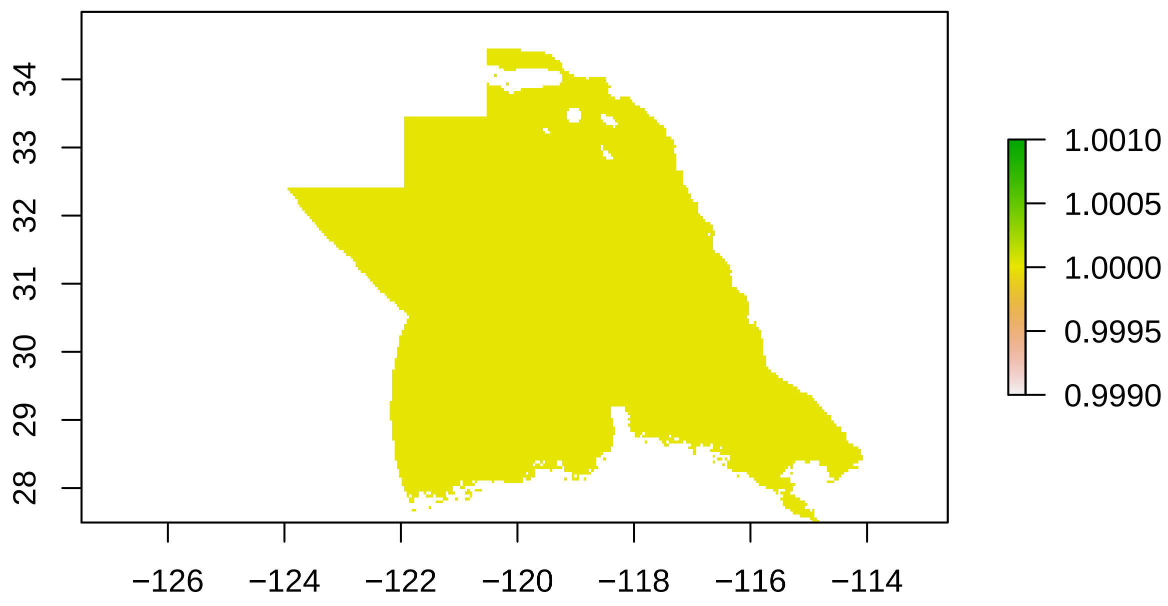


**(2) Run zonal stats**

To get the total viable area for aquaculture of Red Tuna Crab in the California Current, run `zonal()` using `ccZones`. The `zonal()` function is given any sort of function you define including sum, mean, max, etc.

Since the current values of `prefRas` are all equal to 1, we can simply sum up the number of cells in each EEZ.

```r
plot(cellsAQ)
```

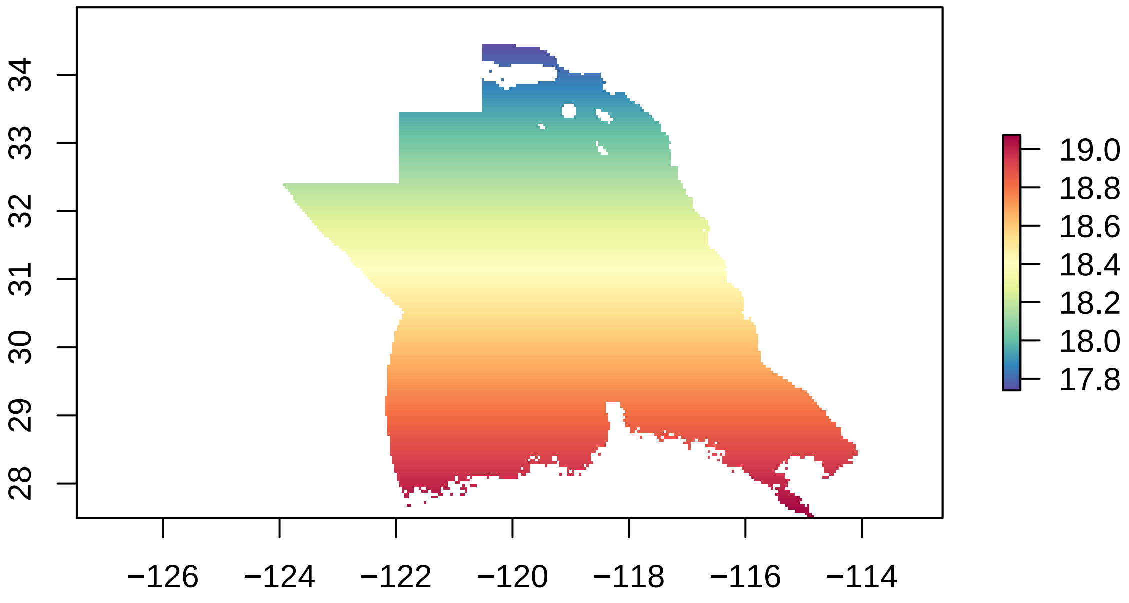


```r
cellsSum = zonal(cellsAQ,ccZones,fun='sum') #total number of cells since they are all equal to 1
cellsSum
```

```
##      zone   sum
## [1,]    1  7920
## [2,]    2 11773
```

But that isn't as useful as calculating the actual area in km2. Using the `area()` function from the `raster` package you can create a new raster with cell values equal to their area, and then run `zonal()`.

```
cellsArea = area(cellsAQ,na.rm=T);plot(cellsArea,col=cols) #gives are in km2
```



```
area = zonal(cellsArea,ccZones,fun='sum');area
```

```
##      zone      sum
## [1,]    1 143913.5
## [2,]    2 219806.2
```