Week 12: Geospatial Data

Jeff Oliver

2024-04-12

Geospatial Data

Introduction

Setup

Geospatial data are data that have some connection to specific places or areas on Earth. To work with geospatial data in R, we usually need to install one or more additional packages. The two most popular packages for geospatial data are terra and sf. In this lesson, we will only be working with terra, and you can find links for more information on terra and on sf at the bottom of the lesson. In order to use the terra package, we must first install the package with `install.packages()`:

```
install.packages("terra")
```

And we load it into memory with the `library()` command:

```
library(terra)
```

Some Terms

Geospatial data generally come in one of two types: raster data or vector data.

- **Raster** data are gridded data. They generally look like images and have a fixed resolution. This means that as you zoom in on raster data, you eventually start seeing the grids. Like when you zoom in on a photograph and it becomes "pixelated" - we will see an example of this later. Raster data are usually quantitative measurements of some sort, such as temperature, precipitation, and reflectance in certain light wavelengths.
- **Vector**, or shape, data are defined by specific coordinates on the Earth's surface and do not, technically, have a fixed resolution. This means that as you zoom in on Vector data, it never becomes "pixelated". Vector data usually take the form of points (a single pair of coordinates), lines (two or more coordinate pairs connected in a sequence), or polygons (three or more coordinate pairs connected in a closed sequence).
- Show zooming effect on both

Today's Destination

Today we will work with both raster and vector data, loading it into R, doing some exploratory visualization, and modifying the data. Ultimately, we are going to create a map of temperature data of southwestern North America, highlighting those areas that are desert biomes.

- Add final image

Geospatial Raster Data

Reading and Visualizing Data

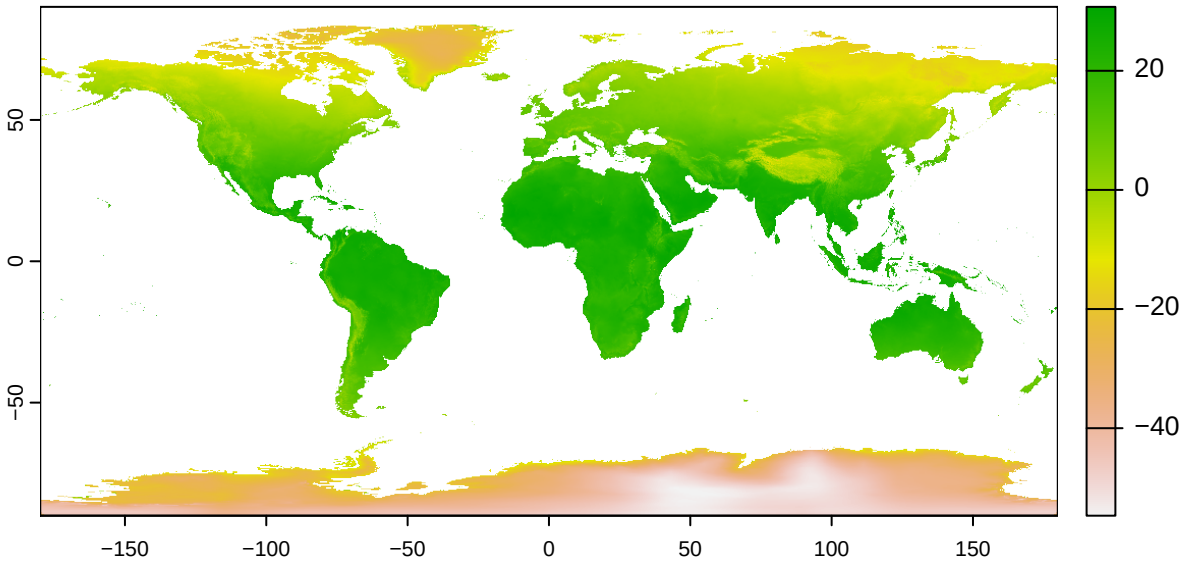
TODO: Add accompanying text for reading in the data (temperature)

```
temp <- rast("global_temperature.tif")
temp
```

```
## class       : SpatRaster
## dimensions  : 4320, 8640, 1  (nrow, ncol, nlyr)
## resolution  : 0.04166667, 0.04166667  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : lon/lat WGS 84 (EPSG:4326)
## source      : global_temperature.tif
## name        : global_temperature
## min value   :          -54.75917
## max value   :           31.16667
```

TODO: Explain console output

```
plot(temp)
```



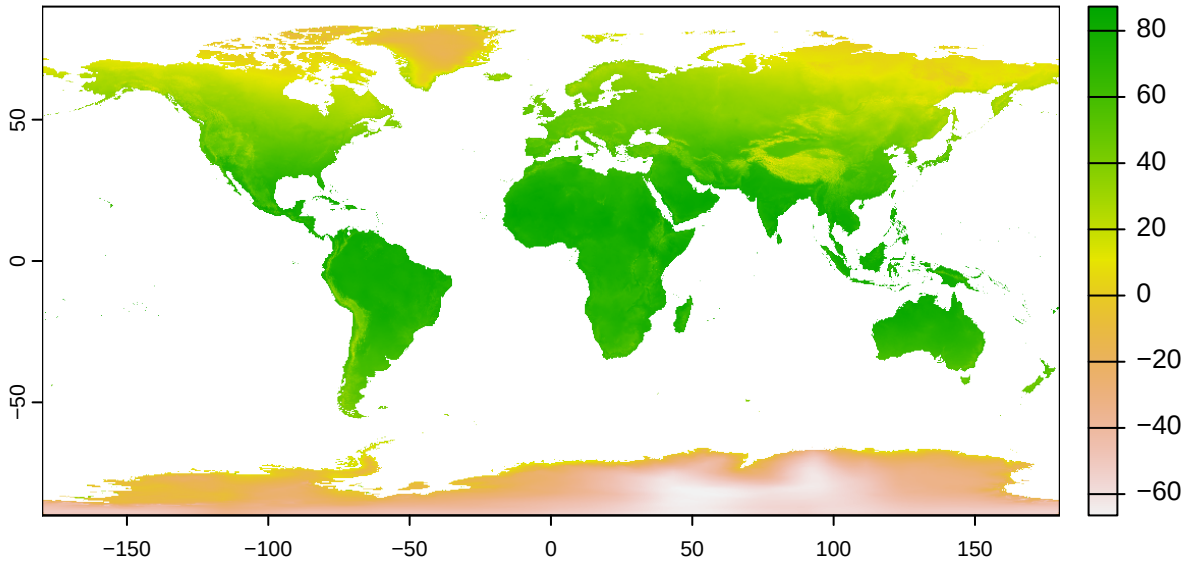
TODO: Explain plot output (legend)

Modifying data

- Do data transformation (like converting C to F)?

Figure out how many values there are. As how you would go about changing them all?

```
# (C * 9/5) + 32 = F
temp_f <- (temp * 9/5) + 32
plot(temp_f)
```



Cropping data

- Why do we need to do this?
- Defining an extent

```
# Lower right: 21.48409688629825, -97.5236750092854
# Upper left: 45.65875210983935, -125.64463895656584
southwest_ext <- ext(c(-126, -98, 21, 46))
southwest_ext
```

```
## SpatExtent : -126, -98, 21, 46 (xmin, xmax, ymin, ymax)
```

- Crop to southwest

```
temp_sw <- crop(temp, southwest_ext)
```

- Compare original (temp) to cropped (temp_sw)

```
temp
```

```
## class       : SpatRaster
## dimensions  : 4320, 8640, 1  (nrow, ncol, nlyr)
## resolution  : 0.04166667, 0.04166667  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : lon/lat WGS 84 (EPSG:4326)
## source      : global_temperature.tif
## name        : global_temperature
## min value   :          -54.75917
## max value   :           31.16667
```

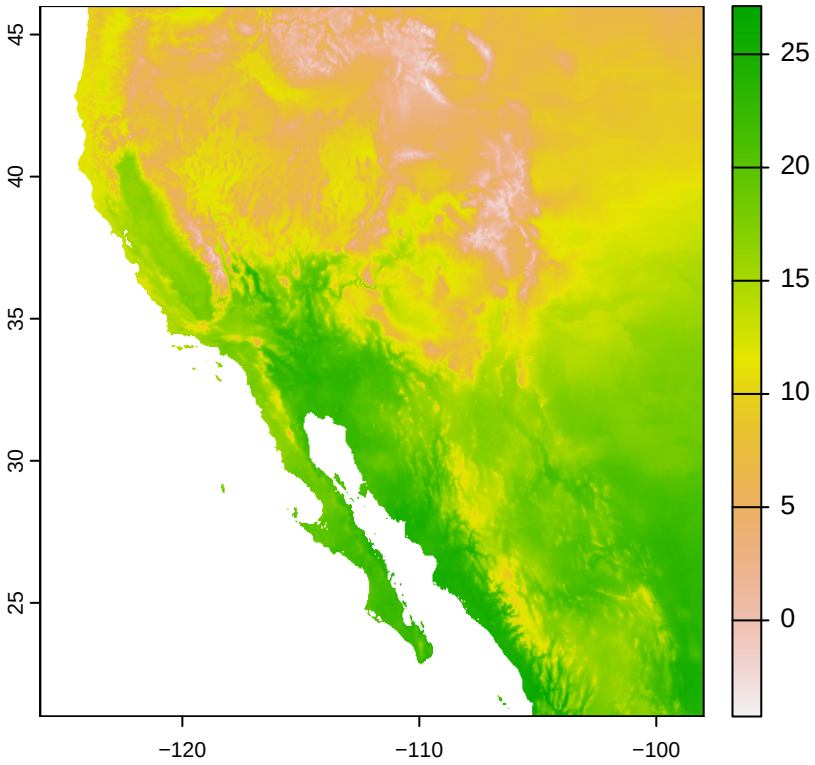
```
temp_sw
```

```
## class       : SpatRaster
## dimensions  : 600, 672, 1  (nrow, ncol, nlyr)
## resolution  : 0.04166667, 0.04166667  (x, y)
## extent      : -126, -98, 21, 46  (xmin, xmax, ymin, ymax)
## coord. ref. : lon/lat WGS 84 (EPSG:4326)
## source(s)   : memory
## varname     : global_temperature
```

```
## name        : global_temperature
## min value   :           -4.245667
## max value   :           27.129499
```

- Plot cropped raster

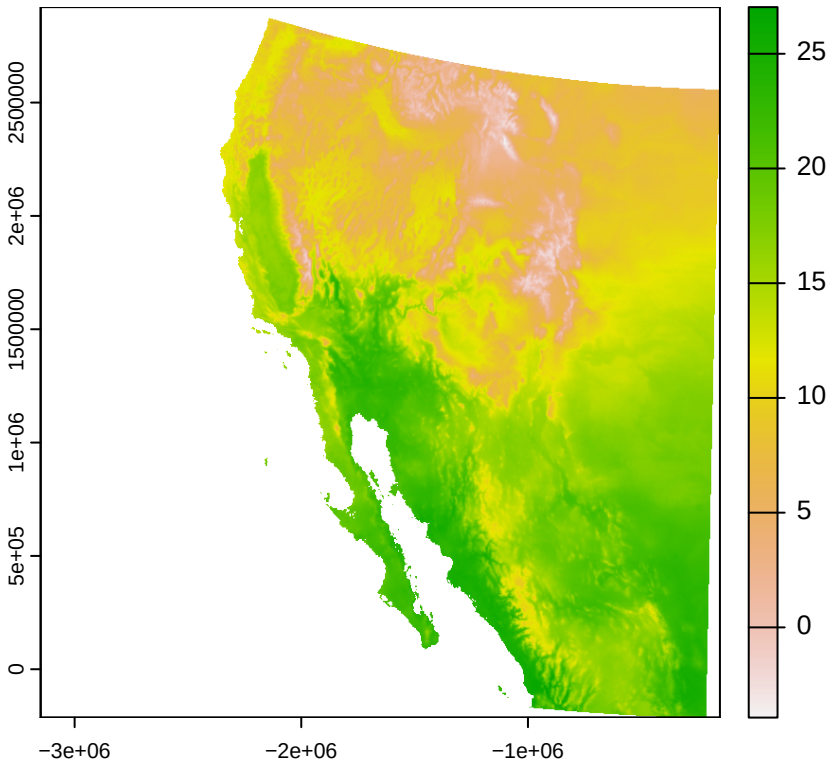
```
plot(temp_sw)
```



Aside: The world is not flat, so why is my map?

- Reproject the data

```
temp_sw_albers <- project(temp_sw, "epsg:5070")
plot(temp_sw_albers)
```

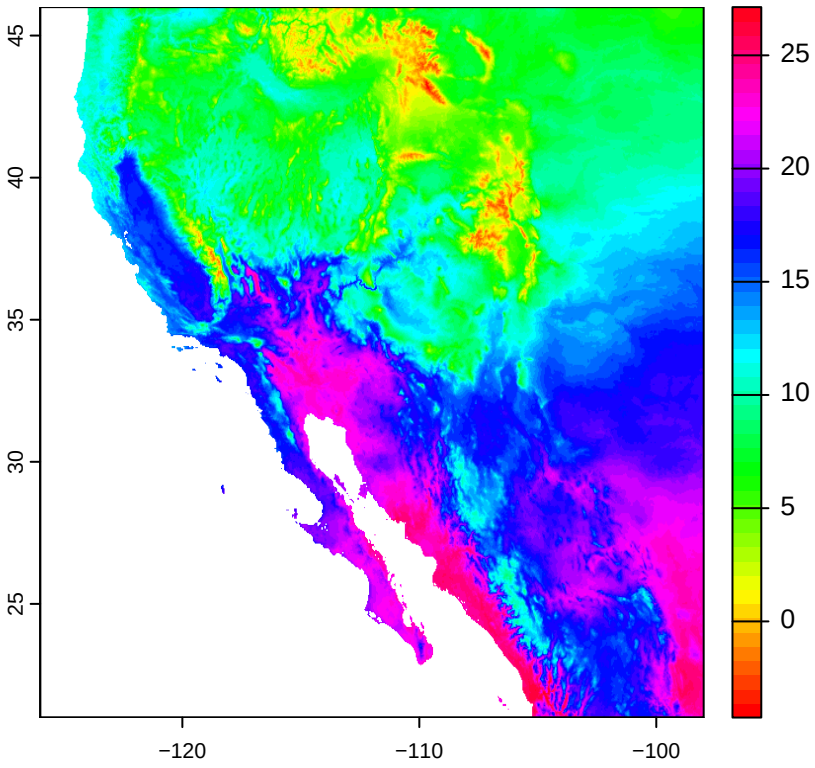


Colors on maps

- recolor (YlOrBr is good for temperature)

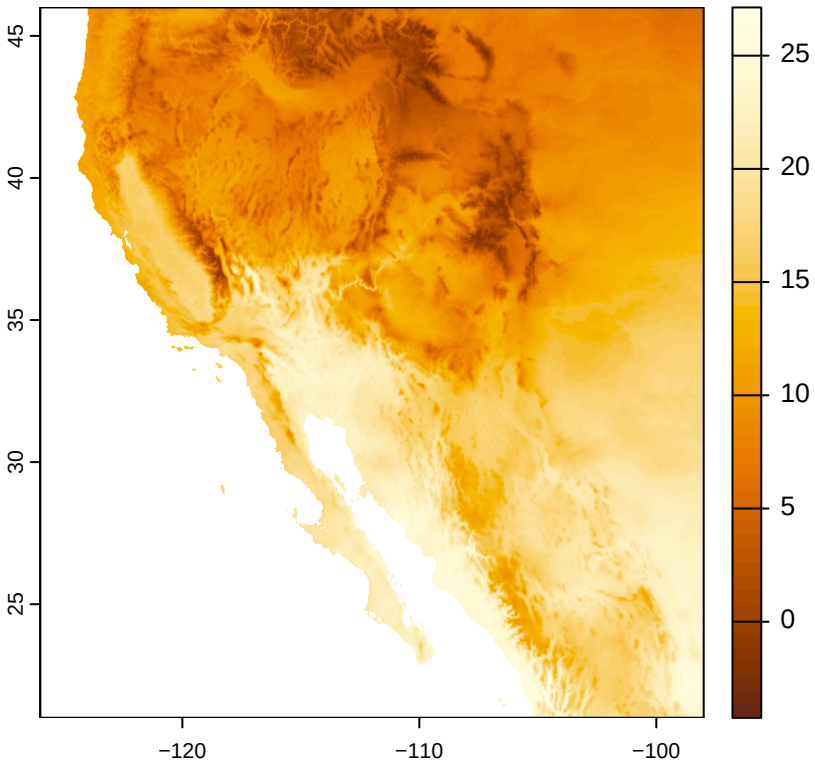
```
# Brings up R documentation for some color palettes
?rainbow
```

```
plot(temp_sw, col = rainbow(n = 50))
```

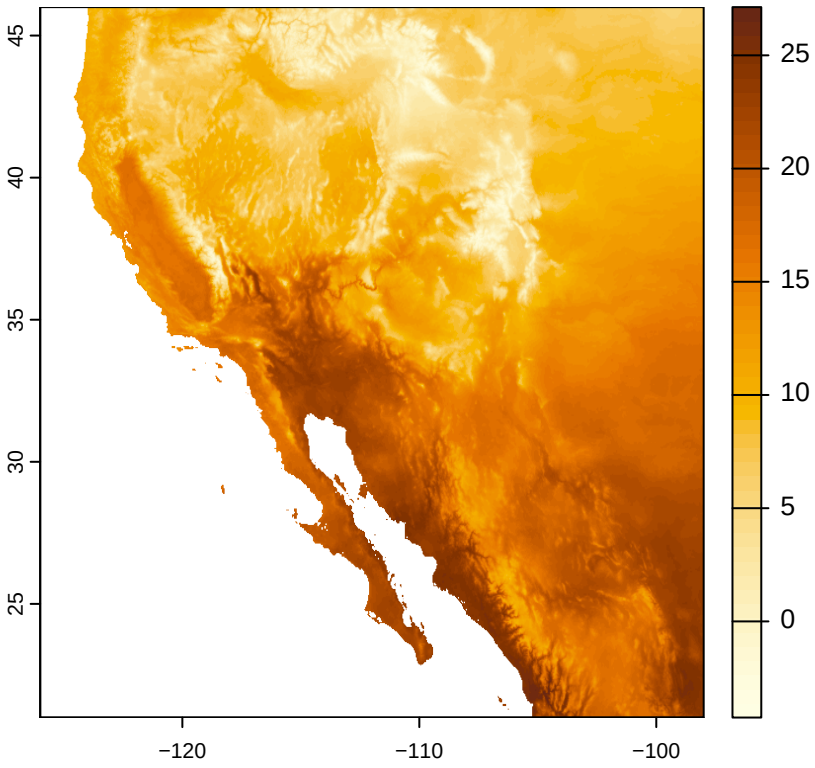


TODO Look at ColorBrewer https://colorbrewer2.org

```
plot(temp_sw, col = hcl.colors(n = 50, palette = "YlOrBr"))
```




```
temp_cols <- rev(hcl.colors(n = 50, palette = "YlOrBr"))
plot(temp_sw, col = temp_cols)
```



Geospatial Vector Data

TODO: Remind about vector data: coordinates, alone (points) or connected (lines, polygons)

Adding Points

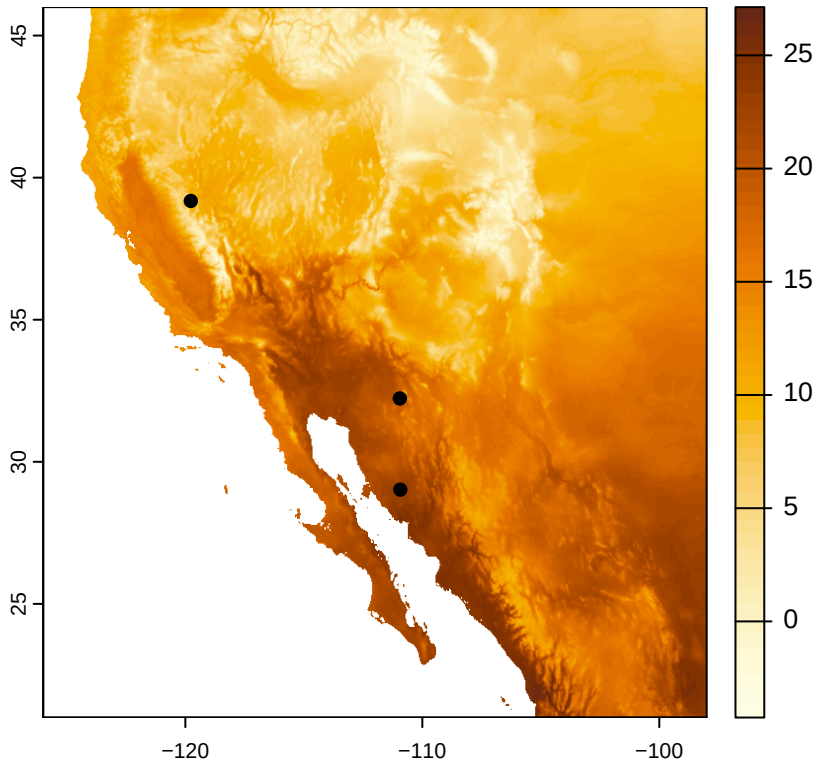
```
cities <- data.frame(city = c("Tucson", "Hermosillo", "Carson City"),
                     lat = c(32.23, 29.02, 39.18),
                     lon = c(-110.95, -110.93, -119.77))
cities
```

```
##          city   lat     lon
## 1      Tucson 32.23 -110.95
## 2  Hermosillo 29.02 -110.93
## 3 Carson City 39.18 -119.77
```

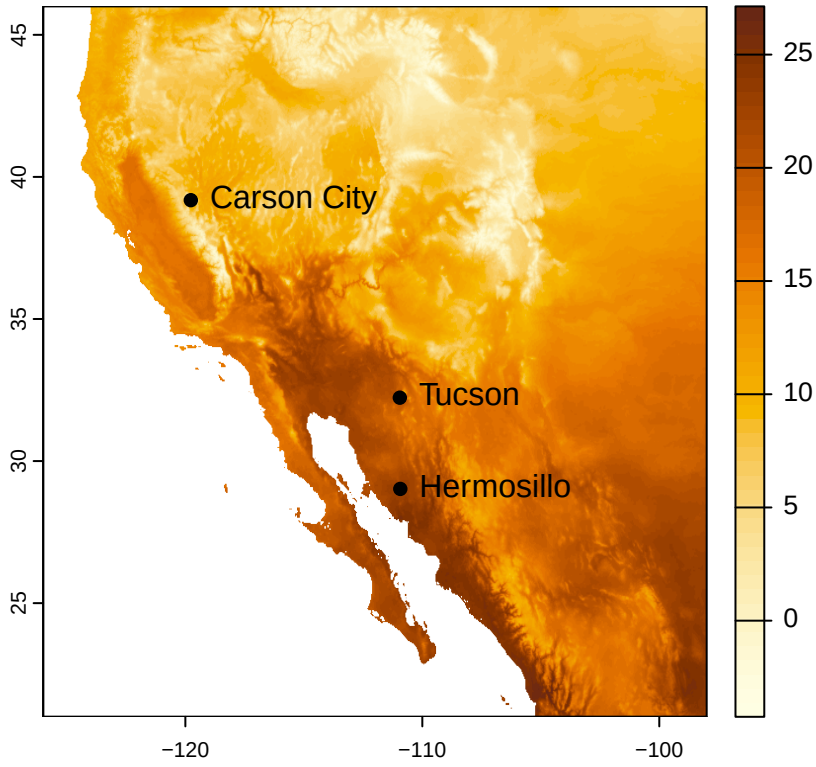
```
cities_vect <- vect(cities, crs = crs(temp_sw))
cities_vect
```

```
##  class       : SpatVector
##  geometry    : points
##  dimensions  : 3, 1  (geometries, attributes)
##  extent      : -119.77, -110.93, 29.02, 39.18  (xmin, xmax, ymin, ymax)
##  coord. ref. : lon/lat WGS 84 (EPSG:4326)
##  names       :        city
##  type        :       <chr>
##  values      :      Tucson
##                  Hermosillo
##                 Carson City
```

```
plot(temp_sw, col = temp_cols)
plot(cities_vect, add = TRUE)
```




```
plot(temp_sw, col = temp_cols)
plot(cities_vect, add = TRUE)
text(cities_vect, labels = "city", pos = 4)
```



Adding Areas (aka Polygons)

Categorize areas as desert with prec_raster < 250 (general definition), turn 0 values to NA, and then convert to polygons with terra::as.polygons Add this to plot to outline desert areas

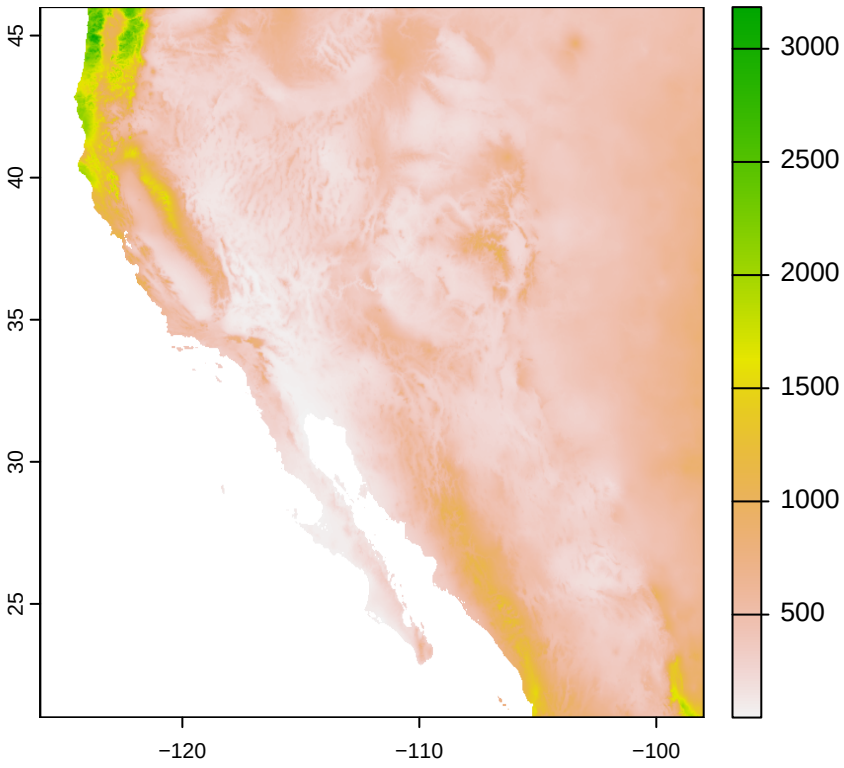
```
prec <- rast("global_precipitation.tif")
prec
```

```
## class       : SpatRaster
## dimensions  : 4320, 8640, 1  (nrow, ncol, nlyr)
## resolution  : 0.04166667, 0.04166667  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : lon/lat WGS 84 (EPSG:4326)
## source      : global_precipitation.tif
## name        : global_precipitation
## min value   :                   0
## max value   :               11246
```

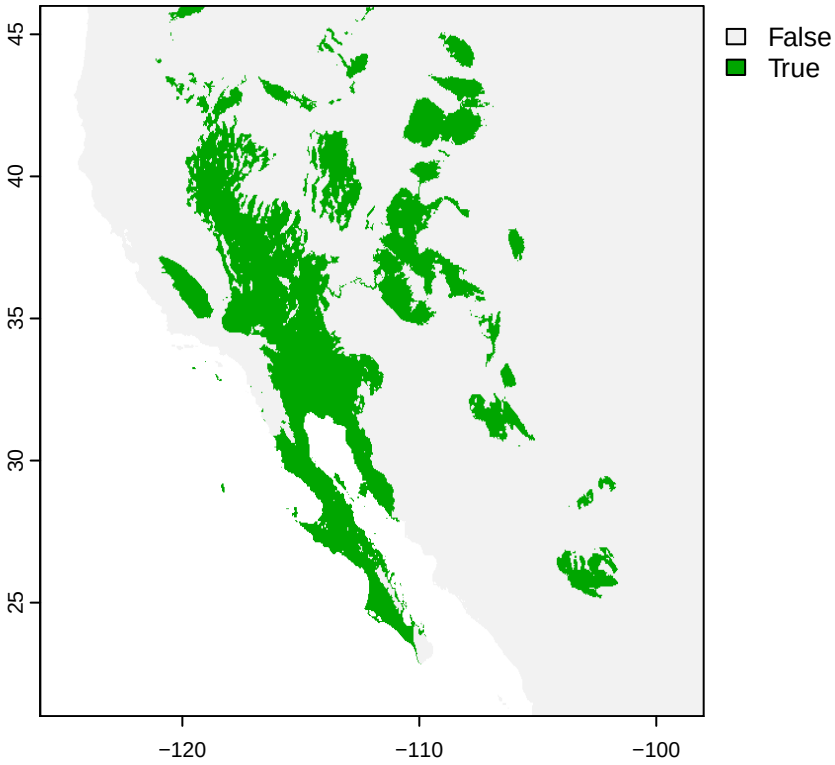
```
prec_sw <- crop(prec, southwest_ext)
prec_sw
```

```
## class       : SpatRaster
## dimensions  : 600, 672, 1  (nrow, ncol, nlyr)
## resolution  : 0.04166667, 0.04166667  (x, y)
## extent      : -126, -98, 21, 46  (xmin, xmax, ymin, ymax)
## coord. ref. : lon/lat WGS 84 (EPSG:4326)
## source(s)   : memory
## varname     : global_precipitation
## name        : global_precipitation
## min value   :                  45
## max value   :                3184
```

```
plot(prec_sw)
```

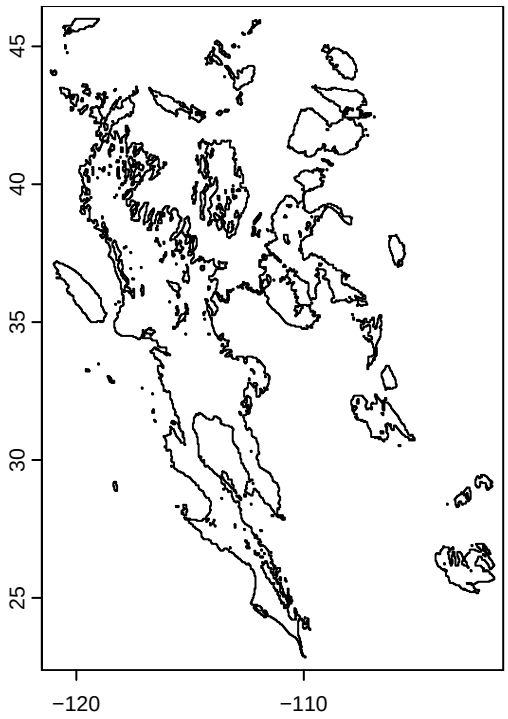



```
desert_max <- 250
prec_desert <- prec_sw < desert_max
plot(prec_desert)
```

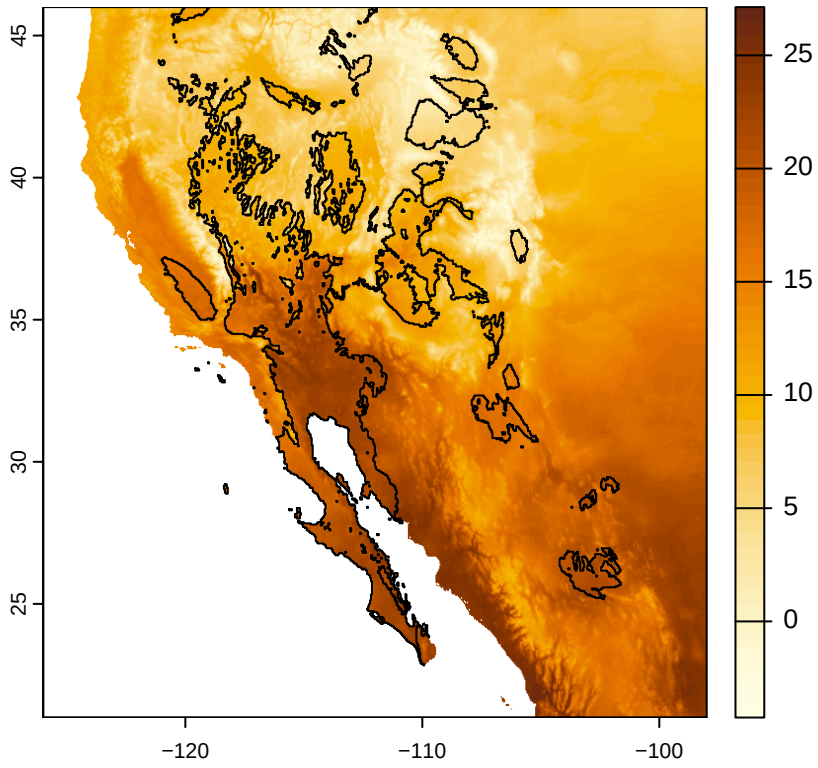



```
# plot(prec_desert, grid = TRUE, pax = list(col = "gray"))
```

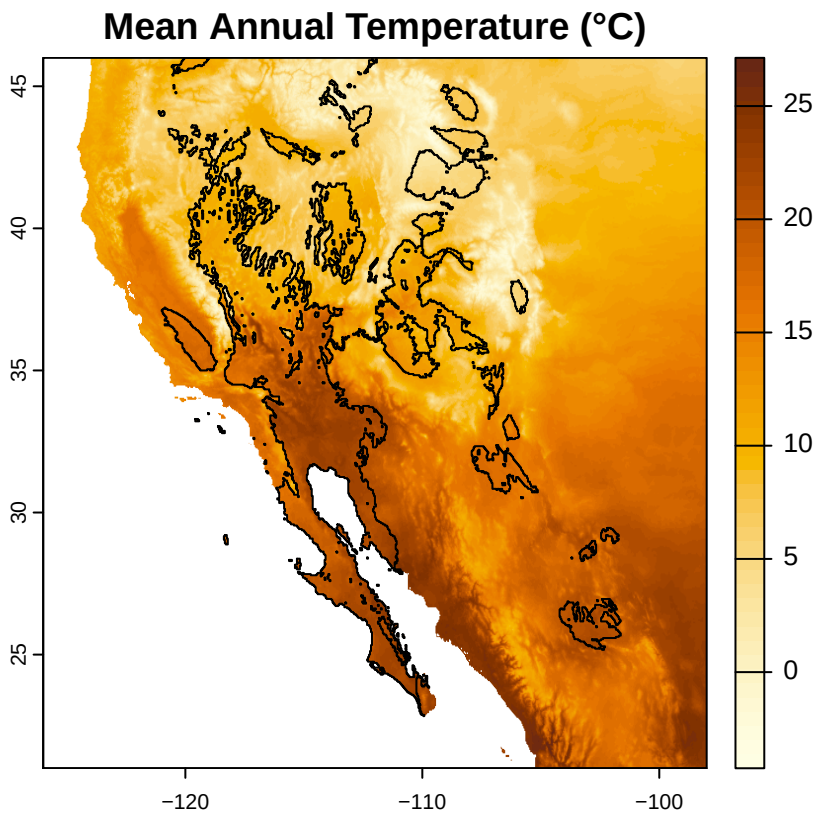
```r
# Change all FALSE values to missing data
prec_desert[!prec_desert] <- NA
# Convert remaining non-missing values to a polygon (vector data)
desert_poly <- as.polygons(prec_desert)
# Plot polygons (reality check)
plot(desert_poly)
```




```r
plot(temp_sw, col = temp_cols)
polys(x = desert_poly)
```




```
plot(temp_sw, col = temp_cols, main = "Mean Annual Temperature (\u00B0C)")
polys(x = desert_poly)
```



Other Resources

- sf package
- rspatial (terra et al.)
- osmdata for OpenStreetMaps access
- ggplot2 for geospatial? => tidyterra
- Data Carpentry