MSU Graduate Spatial Ecology Lab 2

Phoebe Zarnetske; plz@msu.edu

Sep 2016; rev. Sep 9, 2019, Aug 18, 2020

Lab 2: R Markdown, Intro to mapping in R & Scale

This lab has 2 Parts. You need to hand in Part 2 via the D2L Lab 2 Assignment Folder by start of Lab 3. When referring to the code below it may be more useful to use the .Rmd file associated with the PDF, and the GitHub site .

At the end of Lab 2 is Homework in preparation for Lab 3.

Part 1: R Markdown

R Markdown is an authoring framework for R code and output. R Markdown is part word-processor, part R output. It enables you to generate nice reports with R code, figures, tables, and text. It's handy because it produces neat summaries of your work in HTML or PDF or other formats (Word docs). This document was made in Markdown via RStudio. From now on, you'll be handing in all of your lab assignments as PDFs produced from R Markdown.

R Markdown is especially helpful with collaborative research and coursework, and is often used in supplemental materials with publications. If you haven't already watched the R Markdown tutorial, you may want to refer to it when putting together your lab write-up: http://rmarkdown.rstudio.com/lesson-1.html (including "How it Works", "Code Chunks", and "Markdown Basics"). I recommend using "KnitR" when you want to publish (i.e., click on "knit to HTML", "knit to PDF", etc. in the pull-down menu in RStudio).

R Markdown can print or hide certain portions of your code and output. This is helpful especially when you don't want to include long print-outs in a report. For lab assignments, please don't include long R printouts, e.g., printing an entire dataset.

When you click the **Knit** button in R Studio (looks like a knitting needle with ball of yarn) a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
summary(cars)
```

```
##      speed           dist
##  Min.   : 4.0   Min.   :  2.00
##  1st Qu.:12.0   1st Qu.: 26.00
##  Median :15.0   Median : 36.00
##  Mean   :15.4   Mean   : 42.98
##  3rd Qu.:19.0   3rd Qu.: 56.00
##  Max.   :25.0   Max.   :120.00
```

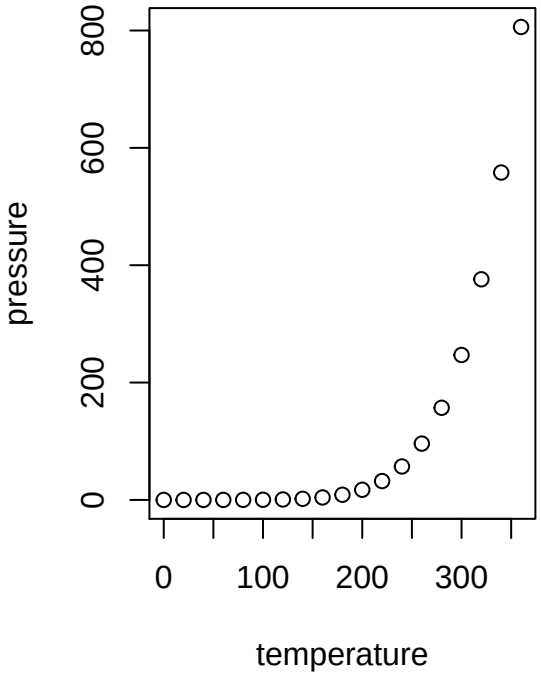
```
summary(cars)
```

`results = 'hide'` will omit the printout (suppressing the summary of the dataframe cars)

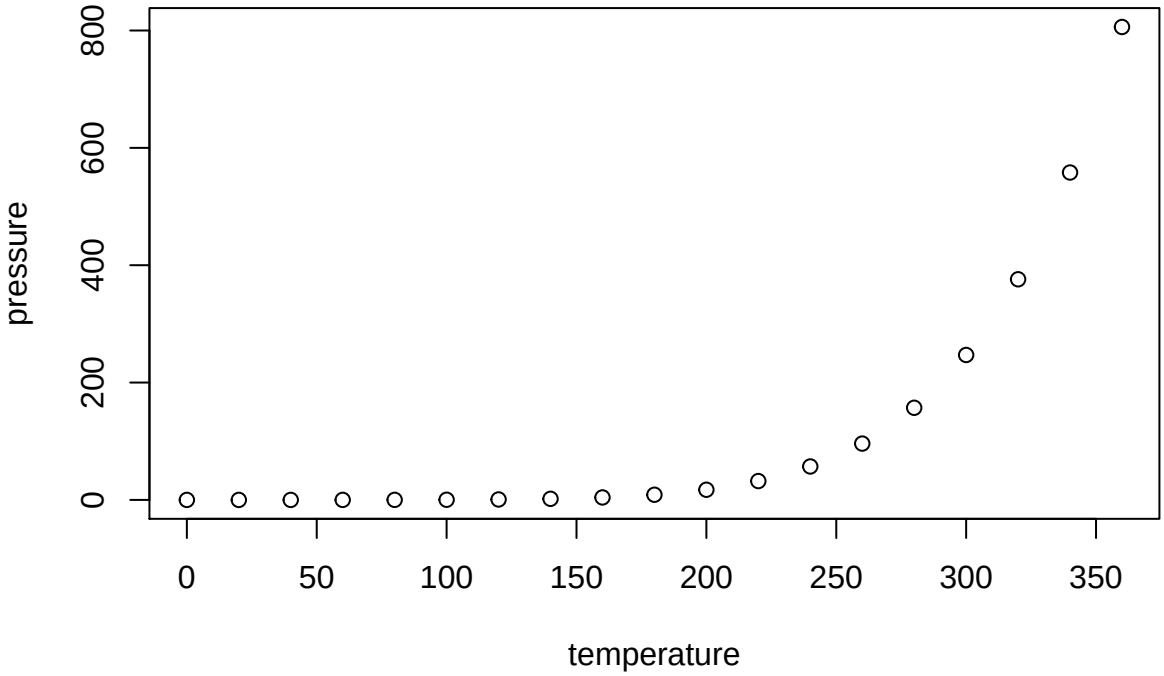
Including Plots

You can also embed plots, for example with the dataframe, pressure:

```
par(mfrow=c(1,2))
plot(pressure)
```



`echo = FALSE` will not print the code beneath it; TRUE is the default. You can also set a figure height and width in the brackets also. Units = inches.



`warning = FALSE` will not print the R warnings beneath it; TRUE is the default.

Part 2: Intro to mapping in R & Scale

R is ok for making maps. If you want something super pretty, it is probably better to go with QGIS or ArcGIS. However, you can make some nice maps in R with some basic commands, and because it's all scripted, it's useful for reproducible research.

```
##### STARTING UP R
# Clear all existing data
rm(list=ls())

# Close graphics devices
graphics.off()

# Reproducibility is essential in science. To ensure your code is reproducible
# set a path to the location of the data and/or workspace. If someone else starts
# to use your code, they only have to change these locations at the top of the
# script.
# In general, avoid spaces! use "_" or "." to separate words if possible.
# This is an example; you'll need to change it to your own paths.
# Note that I'm breaking my own rule of omitting spaces in pathnames here.
# Working from Google Drive Filestream is convenient but includes a space.

# Set paths for your computer location (these directories (folders) must already exist, and will be dif
data_path<-(file.path("data","lab2"))
output_path<-("output")

# If you previously saved your workspace and want to load it here, do so this way
# load(file.path(output_path,"lab2.RData"))

# With R Markdown, it is helpful to install packages locally before knitting (copy this into your R con
for (package in c("raster","sp","rgdal","dismo")) {
  if (!require(package, character.only=T, quietly=T)) {
    install.packages(package)
    library(package, character.only=T)
  }
}

# If you get a warning that says: there is no package called 'raster' type in:
# package(raster)
# Same for other packages.
```

Coordinate Reference Systems & Projection: Check out this helpful (and fun) video that explains projections and CRS: https://ed.ted.com/featured/uTE96JbJ

Download and read in some spatial data. Spatial data require spatial reference. A common way of defining the spatial reference is with: proj4 = projection information about the spatial data, and CRS = coordinate reference systems containing projection, datum, and ellipsoid. A CRS provides the key for how to interpret spatially referenced data given a specific geometry. It provides information about how to transform a three dimensional angular system into a 2-dimensional planar system. Great overview: https://www.nceas.ucsb.edu/~frazier/RSpatialGuides/OverviewCoordinateReferenceSystems.pdf for the proj4 strings see www.spatialreference.org or search on https://epsg.io/

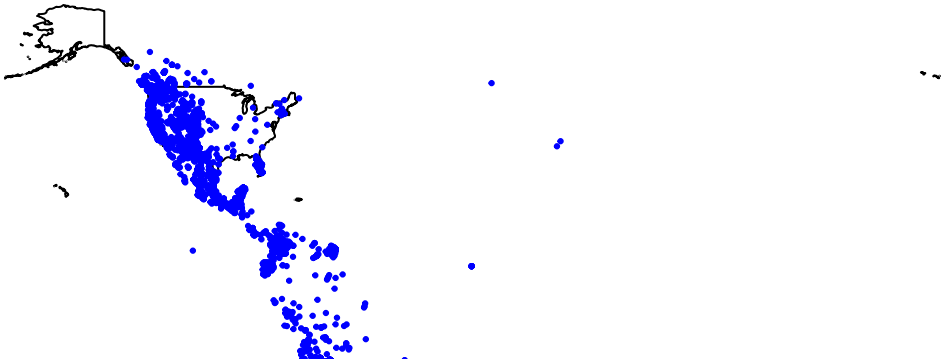
```
# USA BOUNDARY SHAPEFILE: download it from the web [note that you may
# need to change the exdir location depending on where you want it to be saved]
if(! file.exists('us.zip')){
 download.file("http://www2.census.gov/geo/tiger/GENZ2014/shp/cb_2014_us_nation_20m.zip", dest=file.patl
```

```
}

unzip (file.path(data_path,"us.zip"), exdir = data_path)
# Read in the shapefile with rgdal package
us <- readOGR(file.path(data_path,"cb_2014_us_nation_20m.shp"))

# take a look at the shapefile info
summary(us)
```

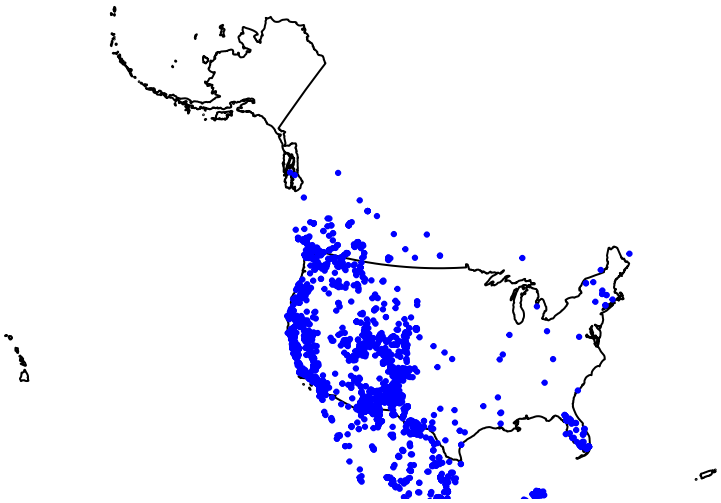
Uh oh! It's not projected, is that a problem? The proj4string states that it's a geographic coordinate system (rather than a projected coordinate system). Geographic systems include angular unit of measure, a prime meridian, and a datum (based on a spheroid). Latitude and Longitude describe any location in geographic coordinate system. If we wanted to do some exact measurements of distances or areas, it would be better to use a projection. This is because projections transform a 3D surface (Earth; a sphere) into a 2D surface and maintain constant lengths and angles, preserving area. Geographic systems have different lengths and and angles depending on the location within the spatial data. Geographic systems are often used with Google Maps.

```
### PROJECTING DATA ###
## This defines the CRS for the US National Atlas Map Projection
us.atlas.proj <- CRS("+proj=laea +lat_0=45 +lon_0=-100 +x_0=0 +y_0=0 +a=6370997
                     +b=6370997 +units=m +no_defs")

par(mfcol=c(1,2)) # sets a plotting area for 2 side-by-side plots
plot(us)
# project to US National Atlas equal-area projection
us.equal <- spTransform(us, us.atlas.proj)
plot(us.equal)
```

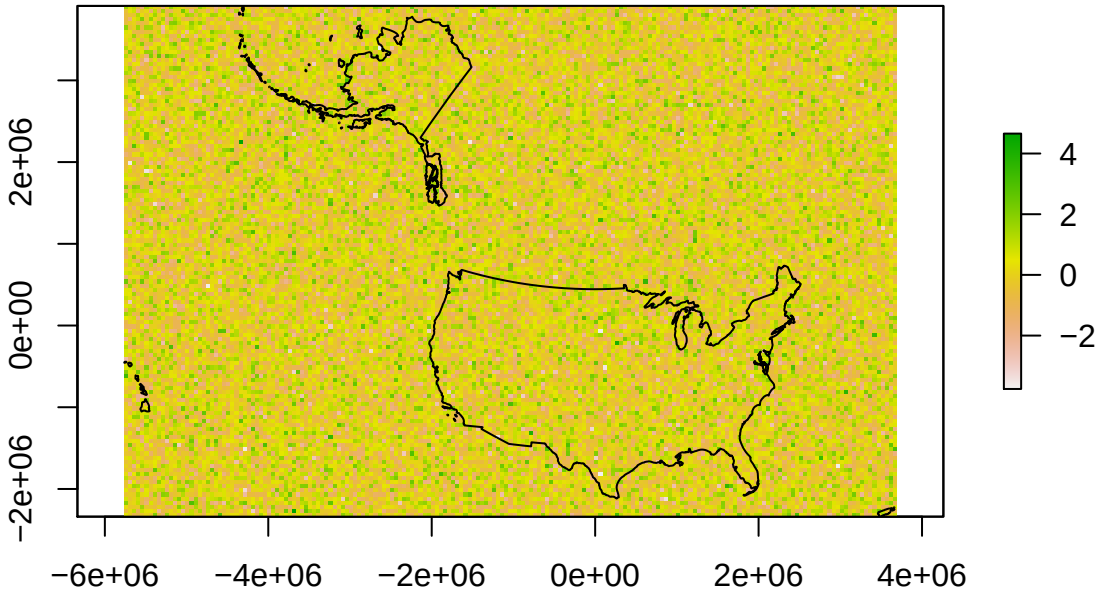
```
### READING IN DATA FROM GBIF.ORG ###
# Add some point data for a species from GBIF.org
# Get GBIF data with function gbif() from the dismo package:
# or choose your own favorite organism (in the US)
# check the GBIF site first to see how many occurrences you'll be importing
# http://www.gbif.org/occurrence/search - enter your species name in the search window
# Click on "filter" and scientific name.
# Puma concolor has 12074+ records, and may take a few mins to import
puma.gbif <- gbif("Puma", "concolor", geo=T) # plot occurrences:
par(mfcol=c(1,1))
plot(us)
points(puma.gbif$lon, puma.gbif$lat,
       pch=19, cex=0.3, col="blue")
```



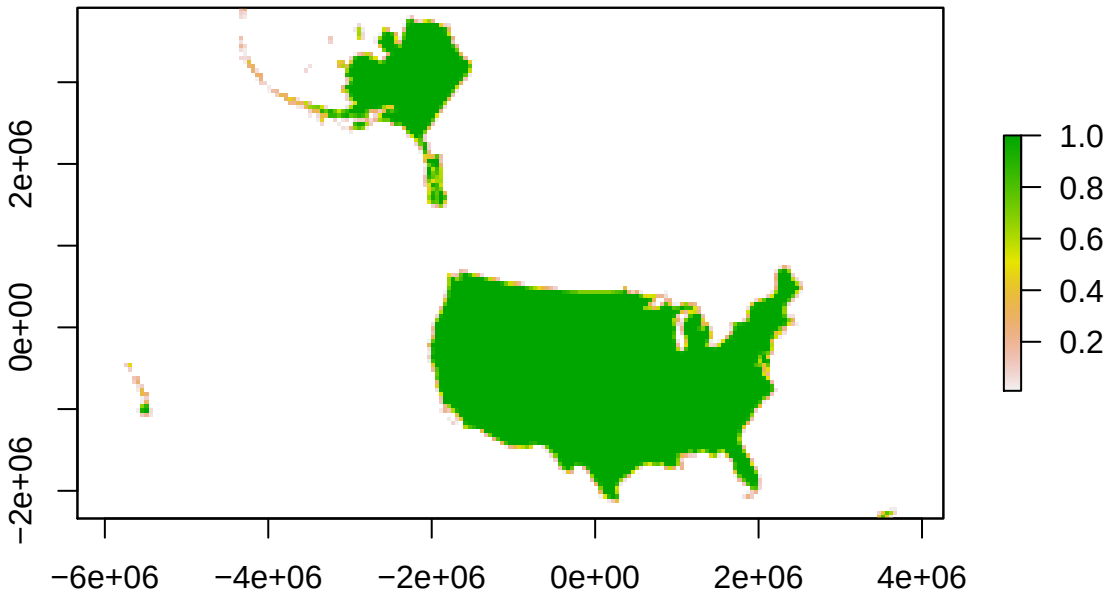
```r
# important: longitude goes first and hence the order
names(puma.gbif) # lat=column 105; lon is column 118
# remove NA values if they exist in lat or lon
puma1<-subset(puma.gbif, puma.gbif$lat != "NA",)
puma1<-subset(puma1, puma1$lon != "NA",)
dim(puma1)
# So really only 5878 records are georeferenced.

# puma.gbif is just a dataframe. We need it to become spatial.
# This defines the CRS for the US in WGS 1984 (world geographic 1984).
wgs1984.proj <- CRS("+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs")

# Reference the columns with "lon" and "lat" to make a spatial points dataframe
puma <- SpatialPoints(coords=puma1[,c(118,105)], proj4string=wgs1984.proj)

# Occasionally the GBIF data are updated and column order changes. Instead of
# referencing the column number, it's better to call it out by name:
puma <- SpatialPoints(coords=puma1[,c("lon","lat")], proj4string=wgs1984.proj)

# Reproject puma to Albers Equal Area Projection so it matches
# the US Map
puma.equal <- spTransform(puma, us.atlas.proj)
plot(us.equal)
points(puma.equal, pch=19, cex=0.3, col="blue")
```



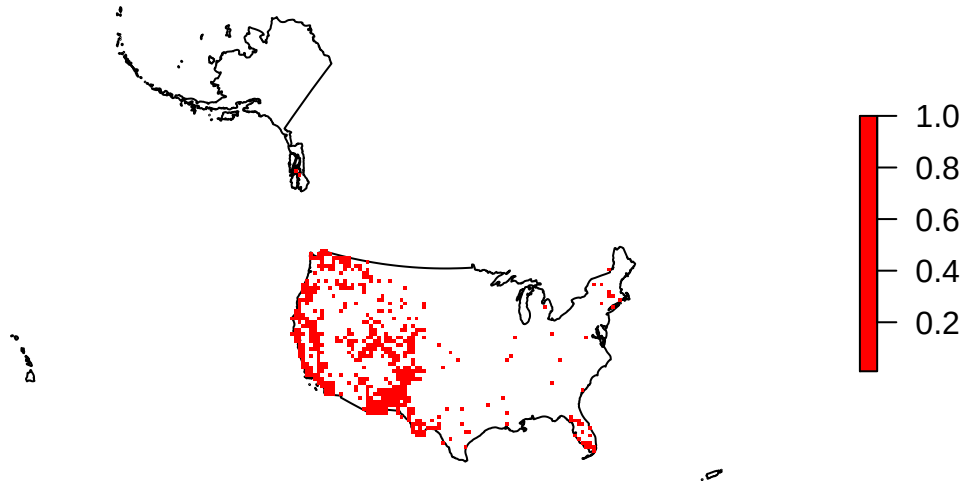
```
### CREATE AN EMPTY GRID - this will define the extent of the study area
# what are the units of us.equal?
summary(us.equal) # units = m
r <- raster(us.equal)
res(r) <- 50000      # resolution (grain size in units of us.equal = meters)
r[] <- rnorm(ncell(r))
plot(r)
plot(us.equal, add=T)
```




```
r[] <- 0

### RASTERIZING (fitting data into a grid)
# rasterizing US boundary
us.raster <- rasterize(us.equal, r, getCover=TRUE)
# assign the value in the US Raster to 1 value
us.raster[us.raster>=1] <- 1
us.raster[us.raster==0] <- NA
plot(us.raster)
```



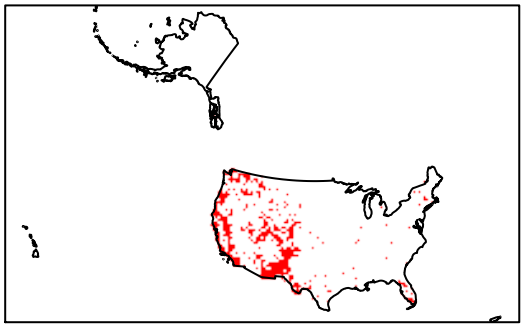

```
# Plot the US outline shapefile
plot(us.equal)
# rasterizing the point data on puma concolor
puma.raster <- rasterize(puma.equal, r)
puma.raster[puma.raster>=1] <- 1
puma.raster[puma.raster==0] <- NA

# limiting the puma point data only to US
puma.raster <- puma.raster*us.raster

plot(puma.raster, add=T, col='red')
```



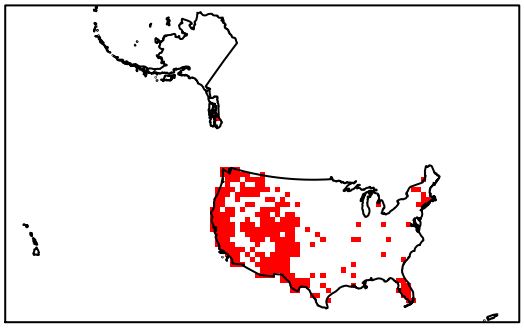

```
### AGGREGATE TO COARSE GRAIN - puma occurrences
par(mfrow=c(2,2), mai=c(0.1, 0.1, 0.5, 0.1)) # 2x2 plots with margins

# 50 x 50 resolution
plot(puma.raster, axes=FALSE, col='red',
     legend=FALSE, main="50 km x 50 km")
plot(us.equal, add=T)
```

```r
# 100 x 100 resolution
p.coarser <- aggregate(puma.raster, fact=2, fun=max)
plot(p.coarser, axes=FALSE, col='red',
     legend=FALSE, main="100 km x 100 km")
plot(us.equal, add=T)

# 200 x 200 resolution
p.coarser2 <- aggregate(p.coarser, fact=2, fun=max)
plot(p.coarser2, axes=FALSE, col='red',
     legend=FALSE, main="200 km x 200 km")
plot(us.equal, add=T)

#400 x 400 resolution
p.coarser4 <- aggregate(p.coarser2, fact=2, fun=max)
plot(p.coarser4, axes=FALSE, col='red',
     legend=FALSE, main="400 km x 400 km")
plot(us.equal, add=T)
```

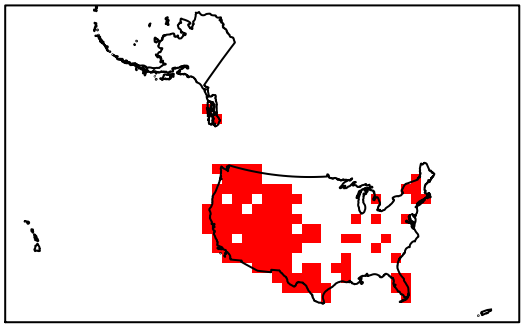
50 km x 50 km



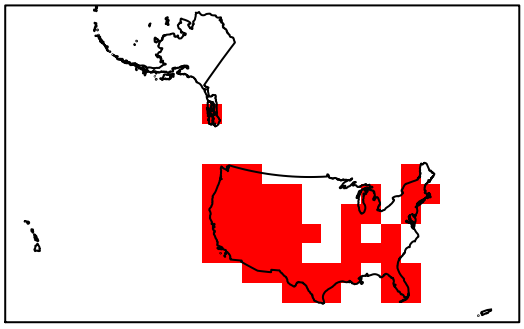
100 km x 100 km



200 km x 200 km



400 km x 400 km



```r
### Save your Workspace ###
## If you want to save your workspace and all the R objects within it:
save.image(file.path(output_path,"lab2.RData"))
```

ASSIGNMENT - Hand in a R Markdown-produced PDF to D2L including the following:

1. Repeat the process above, but create 2 side-by-side plots for 2 new resolutions of your choice. Create both of the plots at the extent of the US Northeast (or if you use another species and it's not in the Northeast, clip it to another US region). Clip the rasters so they only plot for the extent of the Northeast (or another region).

a) ADD A SCALEBAR: See scalebar.R on D2L in this Lab 2 folder for one approach. Also: https://www.rdocumentation.org/packages/raster/versions/2.6-7/topics/scalebar hint: look at the units (and extent) of a raster by typing the name of it at the R prompt.

b) ADD A NORTH ARROW: hint: https://www.rdocumentation.org/packages/GISTools/versions/0.7-4/topics/North%20Arrow

c) CHANGE COLORS: Make each one of the plots differ in color for pixels of the species occurrence raster.

d) OVERLAY the species occurrence points on one plot. Use a contrasting color for the points.

e) ADD A LEGEND. hint: https://biologyforfun.wordpress.com/2013/03/11/taking-control-of-the-legend-in-raster-in-r/ Some tips on plotting LEGENDS with rasters ("distribution" below) and shapefiles ("occurrence" below) together. "left" tells R where to put the legend (lower left). You can also specify actual coordinates. pch is the base R code for the point shape: http://www.statmethods.net/advgraphs/parameters.html - here it's referencing 21 and 15 which are the circles and squares for "occurrence" and "distribution" respectively. Assign their colors with the "col=" command, and their size with "pt.cex". Adding a title to the legend is important, otherwise we don't know what we're looking at. ?legend will tell you more.

2. Repeat 1a-e. above, but this time reproject the data into another projection for North America (your choice).

3. Briefly answer the following: How do the different projections and resolutions change your interpretation of the distribution of the species? Keep in mind that distribution does not just include extent or range size. What would be an appropriate resolution and extent of occurrence data for the species you chose if you are trying to represent its full distribution? Which type of Coordinate System (projected or geographic), if any, would be more appropriate to use with these data if you wanted to measure distances between a) two extreme ends of the species range? and b) two nearby species occurrences locations?

Here are some useful bits of code:

```
# Regions of US shapefile
download.file("http://www2.census.gov/geo/tiger/GENZ2014/shp/cb_2014_us_region_20m.zip",
              dest=file.path(data_path,"usr.zip"), mode="wb")
unzip( file.path(data_path,"usr.zip"), exdir = data_path)

# read in the shapefile
usr <- readOGR(file.path(data_path,"cb_2014_us_region_20m.shp"))
# take a look at the shapefile info
summary(usr)
# follow the steps above for projecting it to WGS 1984,
# then to US National Atlas Equal-Area.
# Hint for plotting region: extract a portion of a shapefile and create a
# new one. Look at the first 6 lines of data in the shapefile attribute table.
# head(usr.equal)
# make a shapefile of just Northeast
# ne <- usr.equal[usr.equal$NAME =="Northeast",]
```

Homework in preparation for Lab 3:

In Lab 3 we will be working with landcover data from the National Land Cover Database (NLCD). Read about NLCD here (https://www.mrlc.gov/) and view the short video: https://www.mrlc.gov/nlcd-2016-video.

In past years we have used the MSU HPCC for this lab, but we will not be doing so this year.

OPTIONAL: If you're interested in learning more about the HPCC here is the go-to Introduction website: https://wiki.hpcc.msu.edu/display/TEAC/Introduction+to+HPCC

There are also Intro to HPCC workshops offered monthly through iCER; free but registration required https://icer.msu.edu/upcoming-workshops)

If you are not familiar with basic Linux commands, you can review them here: https://linuxconfig.org/bash-scripting-tutorial-for-beginners#h1-bash-shell-scripting-definition. We will be working through the HPCC lab together, so it is not essential to know all of these ahead.