

# Intro To Geospatial R

## Packages for Geospatial Analysis

There are MANY packages for geospatial analysis in R, whether you are working with Raster or Vector data.

- ▶ For rasters, the `raster` package makes for easy raster analysis.
- ▶ For vectors, the `rgdal` and `sp` packages allow for data to be easily read in and manipulated.
- ▶ For visualization, the `mapview` packages allows users to display their data on OSM maps.

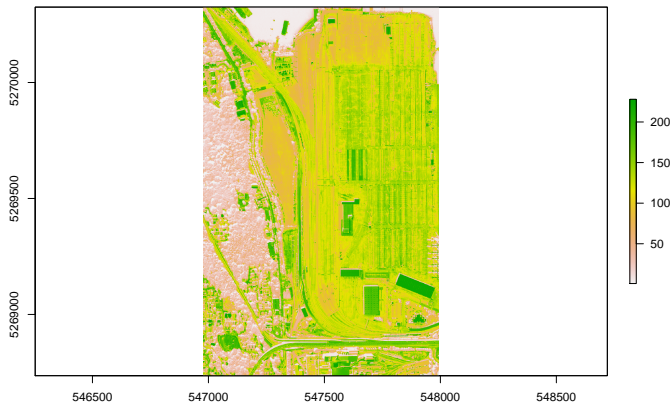
## Reading in Raster Data

```
require(raster)
seattle <- raster("../tutorials/data/Seattle_NAIP.tif", band = 1)
```

This assigns the first band of 4-band NAIP image to an object we've titled "seattle". We can use this object to visualize the data...

## Plotting Raster Data with plot() Function

```
plot(seattle)
```



## Reading in Multi-Band Rasters

```
seattle_4band <- brick("../tutorials/data/Seattle_NAIP.tif")
```

Raster bricks and stacks are important components of the raster package that allow us to read in multiple bands of a raster.

A raster brick is one multi-layer file or object which is loaded into memory. - Faster and more efficient processing

A raster stack can be “virtually” connected to multiple raster objects that are written to different files in memory. - Can performed pixel-based calculations on separate raster layers

## Plotting RGB Raster Data

```
plotRGB(seattle_4band, r = 1, g = 2, b = 3)
```



## Do Raster Calculations

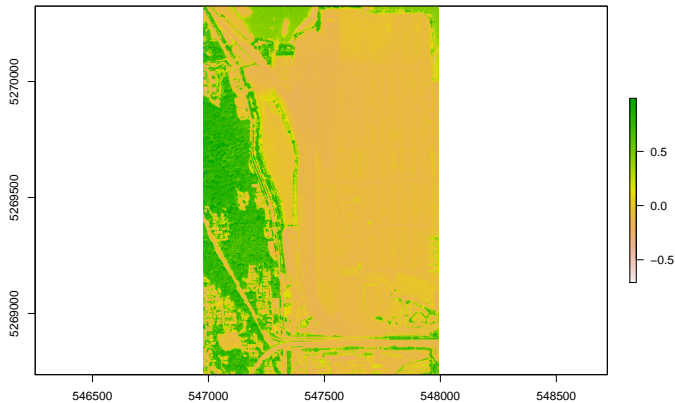
We may all be familiar with the infamous “raster calculators” in our favorite GIS applications. Often times we need to do some manipulation with our raster data sets, whether it’s multiplying all the values by 2, or subtracting the values of one raster from another.

R makes it EXTREMELY easy to manipulate your rasters and perform calculations. Here we calculate the NDVI from our raster stack. NDVI is a way of measuring vegetation health using the red and near infrared bands of multispectral images.

```
NDVI <- (seattle_4band$Seattle_NAIP.4 -  
         seattle_4band$Seattle_NAIP.1)/  
         (seattle_4band$Seattle_NAIP.4 +  
         seattle_4band$Seattle_NAIP.1)
```

## Do Raster Calculations

```
plot(NDVI)
```





## Reading in Vector Data

Points, Lines, and Polygons are common forms of vector data that we may use on a day-to-day basis in spatial analysis. There are two primary packages for reading in vector data that are used by frequent R users: the `sp` and `rgdal` packages.

You may be familiar with `gdal` if you are familiar with python. `Rgdal` is `gdal` adapted for use in `r`. The `gdalUtils` package also provides access to all `gdal` functions and arguments.

The `readOGR` function is the primary function used for reading in vector data... But it should be noted that this function is a bit complicated. Luckily, the folks that created the `raster` package helped to simplify this by creating a wrap-around function called `shapefile` which more easily reads in shapefiles. They both do essentially the same job...

## Reading in Vector Data

```
require(rgdal)
require(sp)
require(raster)
```

```
trees <- shapefile("../tutorials/data/alkiTrees_UTM.shp")
class(trees)
```

```
## [1] "SpatialPointsDataFrame"
## attr(,"package")
## [1] "sp"
```

## Reading in Vector Data

```
trees2 <- readOGR(dsn = "../tutorials/data/",  
                  layer = "alkiTrees_UTM")
```

```
## OGR data source with driver: ESRI Shapefile  
## Source: "../tutorials/data/", layer: "alkiTrees_UTM"  
## with 300 features  
## It has 1 fields
```

```
class(trees2)
```

```
## [1] "SpatialPointsDataFrame"  
## attr(,"package")  
## [1] "sp"
```

## Reading in Vector Data

When the data is read in by either the `shapefile` function or the `readOGR` function, the result is a “SpatialPointsDataFrame”, an `sp` object!

This is because the `rgdal` package requires the `sp` package to work.

Essentially, `sp` provides a framework for storing data, and `rgdal` provides a framework for reading and writing data.

## How Vector data is stored in sp

You may have noticed the term “SpatialPointsDataFrame” contains **data frame**... This is because the data is stored in a data frame object that contains additional geospatial information.

```
head(trees@data)
```

```
##              Cmmn_Nm
## 1 Pacific Sunset Maple
## 2      Norway Maple
## 3 Pacific Sunset Maple
## 4  Cherry/Plum/Laurel
## 5 Pacific Sunset Maple
## 6 Pacific Sunset Maple
```

## What is in a spatial points data frame?

```
trees
```

```
## class      : SpatialPointsDataFrame
## features   : 300
## extent     : 546980.1, 547992.5, 5268745, 5270317 (xmin, xmax, ymi
## coord. ref. : +proj=utm +zone=10 +datum=NAD83 +units=m +no_defs +ell
## variables  : 1
## names      :                               Cmmn_Nm
## min values  : Autumn Flowering Cherry
## max values  :                               Western Red Cedar
```

## What is in a spatial points data frame?

```
str(trees)
```

```
## Formal class 'SpatialPointsDataFrame' [package "sp"] with 5 slots
##   ..@ data          : 'data.frame':  300 obs. of  1 variable:
##   .. ..$ Cmmn_Nm: chr [1:300] "Pacific Sunset Maple" "Norway Maple"
##   ..@ coords.nrs  : num(0)
##   ..@ coords      : num [1:300, 1:2] 547195 547875 547312 547304 5472
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : NULL
##   .. .. ..$ : chr [1:2] "coords.x1" "coords.x2"
##   ..@ bbox        : num [1:2, 1:2] 546980 5268745 547992 5270317
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:2] "coords.x1" "coords.x2"
##   .. .. ..$ : chr [1:2] "min" "max"
##   ..@ proj4string: Formal class 'CRS' [package "sp"] with 1 slot
##   .. .. ..@ projargs: chr "+proj=utm +zone=10 +datum=NAD83 +units=m
```

## Spatial Pixels Data Frames

```
seattle_4band.sp <- as(seattle_4band, "SpatialPixelsDataFrame")  
head(seattle_4band.sp@data)
```

##	Seattle_NAIP.1	Seattle_NAIP.2	Seattle_NAIP.3	Seattle_NAIP.4
## 1	65	122	81	212
## 2	100	117	83	198
## 3	93	111	79	198
## 4	52	98	60	205
## 5	42	91	56	207
## 6	30	73	51	194



## Spatial Pixels Data Frames

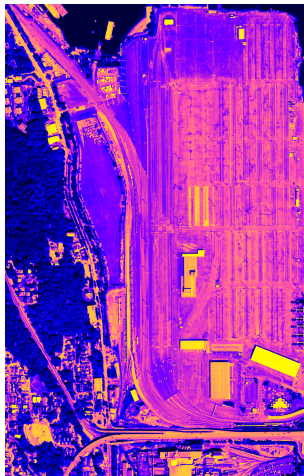
These can be extremely useful when working with many bands (i.e. hyperspectral data) as they are more efficient to store than rasters.

I personally use these for building classification and/or regression models. Rather than having to extract the values for each cell, this format has everything in a nice and easy to access table.

And, you can still plot the data!

## Spatial Pixels Data Frames

```
plot(seattle_4band.sp)
```



## sf - The Latest in Geospatial R

`sf` == “simple features”

From the makers of `sp` comes `sf`. `sf` changes the class structure of spatial objects, taking on a form that more resembles how spatial objects are store in Postgres.

- ▶ Based on tibbles.
- ▶ Has a spatial column, rather than a dataframe attached to a spatial object.
- ▶ Geometry operations are simplified and use function names similar to Postgres.
- ▶ Faster operations, more flexible objects.
- ▶ Geometries are represented as wkt/wkb, an international standard.

Simple Features on Wikipedia

## sf - Simplifies the Workflow

sf brings many of the geometry and spatial I/O operations under one package.

- ▶ rgdal
- ▶ rgeos
- ▶ deldir

sf *does not* support rasters operations or classes. These are still handled by the raster library. Compatibility between the two libraries is spotty at the moment, but conversions between sf and sp objects are simple enough.

## What We Covered

The raster package is a great way to read in raster objects, including multi-band rasters. But, when you start getting into larger multi-band rasters (i.e. hyperspectral data), it's best to switch to a "spatial pixels data frame" format, which is built into the sp package.

For reading and manipulating vector data, we recommend using `rgdal` (remember the function `readOGR`) or the nice wraparound for `readOGR` which is built into the raster package (the function is called `shapefile`).

Whether you are using `readOGR` or `shapefile`, the objects will be read in as "Spatial\_\_\_\_DataFrame". Fill in the blank with pixels, points, or polygons.

`sf` is the latest and greatest in spatial vector processing. It is currently a work in progress but is already getting traction as the future of geospatial R. At this time, it is important to be aware of both `sp` and `sf`.