**Introduction**

There are many different R packages for dealing with spatial data. The main distinctions between them involve the types of data they work with — raster or vector — and the sophistication of the analyses they can do. Raster data can be thought of as pixels, similar to an image, while vector data consists of points, lines, or polygons. Spatial data manipulation can be quite complex, but creating some basic plots can be done with just a few commands. In this post, we will show simple examples of raster and vector spatial data for plotting a watershed and gage locations, and link to some other more complex examples.

**Setting up**

First, let’s download an example shapefile (a polygon) of a HUC8 from western Pennsylvania, using the sbtools package to access [ScienceBase](https://www.sciencebase.gov/catalog/). The st\_read function from the sf (for “simple features”) package reads in the shapefile. We will be using sf throughout these examples to manipulate the points and polygon for the gages and HUC. Then we’ll retrieve gages with discharge from this watershed using the dataRetrieval package. Both dataRetrieval and sbtools are covered in our [USGS Packages curriculum](https://owi.usgs.gov/R/training-curriculum/usgs-packages/).

library(sbtools)

library(dataRetrieval)

library(sf)

item\_file\_download(sb\_id = "5a83025ce4b00f54eb32956b",

names = "huc8\_05010007\_example.zip",

destinations = "huc8\_05010007\_example.zip",

overwrite\_file = TRUE)

## [1] "huc8\_05010007\_example.zip"

unzip('huc8\_05010007\_example.zip', overwrite = TRUE)

#note st\_read takes the folder, not a particular file

huc\_poly <- st\_read('huc8\_05010007\_example')

## Reading layer `wbdhu8\_alb\_simp' from data source `D:\LADData\RCode\owi-blog\content\huc8\_05010007\_example' using driver `ESRI Shapefile'

## Simple feature collection with 1 feature and 9 fields

## geometry type: POLYGON

## dimension: XY

## bbox: xmin: -79.45512 ymin: 39.91875 xmax: -78.55573 ymax: 40.77377

## epsg (SRID): 4326

## proj4string: +proj=longlat +datum=WGS84 +no\_defs

class(huc\_poly)

## [1] "sf" "data.frame"

str(huc\_poly)

## Classes 'sf' and 'data.frame': 1 obs. of 10 variables:

## $ REGION : Factor w/ 1 level "Ohio Region": 1

## $ SUBREGION : Factor w/ 1 level "Allegheny": 1

## $ BASIN : Factor w/ 1 level "Allegheny": 1

## $ SUBBASIN : Factor w/ 1 level "Conemaugh": 1

## $ HUC\_2 : Factor w/ 1 level "05": 1

## $ HUC\_4 : Factor w/ 1 level "0501": 1

## $ HUC\_6 : Factor w/ 1 level "050100": 1

## $ HUC\_8 : Factor w/ 1 level "05010007": 1

## $ HU\_8\_STATE: Factor w/ 1 level "PA": 1

## $ geometry :sfc\_POLYGON of length 1; first list element: List of 1

## ..$ : num [1:256, 1:2] -79 -79 -79 -79 -79 ...

## ..- attr(\*, "class")= chr "XY" "POLYGON" "sfg"

## - attr(\*, "sf\_column")= chr "geometry"

## - attr(\*, "agr")= Factor w/ 3 levels "constant","aggregate",..: NA NA NA NA NA NA NA NA NA

## ..- attr(\*, "names")= chr "REGION" "SUBREGION" "BASIN" "SUBBASIN" ...

st\_geometry(huc\_poly)

## Geometry set for 1 feature

## geometry type: POLYGON

## dimension: XY

## bbox: xmin: -79.45512 ymin: 39.91875 xmax: -78.55573 ymax: 40.77377

## epsg (SRID): 4326

## proj4string: +proj=longlat +datum=WGS84 +no\_defs

st\_bbox(huc\_poly)

## xmin ymin xmax ymax

## -79.45512 39.91875 -78.55573 40.77377

st\_crs(huc\_poly)

## Coordinate Reference System:

## EPSG: 4326

## proj4string: "+proj=longlat +datum=WGS84 +no\_defs"

huc\_gages <- whatNWISdata(huc = "05010007", parameterCd = "00060", service="uv")

head(huc\_gages)

## agency\_cd site\_no station\_nm

## 197 USGS 03039925 North Fork Bens Creek at North Fork Reservoir, PA

## 591 USGS 03040000 Stonycreek River at Ferndale, PA

## 621 USGS 03041029 Conemaugh River at Minersville, PA

## 731 USGS 03041500 Conemaugh River at Seward, PA

## 824 USGS 03042000 Blacklick Creek at Josephine, PA

## 964 USGS 03042280 Yellow Creek near Homer City, PA

## site\_tp\_cd dec\_lat\_va dec\_long\_va coord\_acy\_cd dec\_coord\_datum\_cd

## 197 ST 40.26619 -79.01669 S NAD83

## 591 ST 40.28563 -78.92058 S NAD83

## 621 ST 40.34139 -78.92611 S NAD83

## 731 ST 40.41924 -79.02614 S NAD83

## 824 ST 40.47701 -79.18670 S NAD83

## 964 ST 40.57257 -79.10337 S NAD83

## alt\_va alt\_acy\_va alt\_datum\_cd huc\_cd data\_type\_cd parm\_cd stat\_cd

## 197 05010007 uv 00060

## 591 1184.06 .01 NGVD29 05010007 uv 00060

## 621 1140 .01 NGVD29 05010007 uv 00060

## 731 1076.01 .01 NGVD29 05010007 uv 00060

## 824 954.46 .01 NAVD88 05010007 uv 00060

## 964 1140 .01 NGVD29 05010007 uv 00060

## ts\_id loc\_web\_ds medium\_grp\_cd parm\_grp\_cd srs\_id access\_cd

## 197 221326 wat 1645423 0

## 591 122456 wat 1645423 0

## 621 122464 wat 1645423 0

## 731 122468 wat 1645423 0

## 824 122471 wat 1645423 0

## 964 122477 wat 1645423 0

## begin\_date end\_date count\_nu

## 197 1987-10-03 1998-09-26 4011

## 591 1991-10-01 2018-08-16 9816

## 621 2001-12-13 2018-08-16 6090

## 731 1991-10-01 2018-08-16 9816

## 824 1991-10-15 2018-08-16 9802

## 964 1991-10-01 2018-08-16 9816

The huc\_poly object is a new type of object that we haven’t seen — it has classes sf as well as data.frame. Looking inside the object with the str command, we can see it is structured very much like a data.frame with several factor columns, except for the geometry column, which is of type sfc\_POLYGON. sf provides various functions to extract useful information from this kind object, generally prefixed with st\_. st\_geometry extracts the entire geometry part of the object; st\_bbox extracts the bounding box from the geometry; and st\_crs extracts the coordinate reference system. There are others, but we will use these three to get the parts of the sf object we need for plotting.

Now that we understand this new object, let’s make some maps.

**Raster map example**

For the raster map, we will use the ggmap package to create a road and satellite basemaps for the HUC. Since the basemaps that ggmap uses are quite detailed, they are too large to include with the package and must be retrieved from the web with the get\_map function. For the location argument, we are getting the bbox from the huc\_poly object. st\_bbox returns the bbox in the format we need, except for the names, which we add with setNames. The ggmap function is analogous to the ggplot function in the ggplot2 package that you have likely already used. It creates the base map, which we can then add things to. Many of the commands used here are from the ggplot2 package (ggmap imports them), and others could be used to further customize this map.

Note that ggmap is probably not a good choice if you need your data to be in a particular projection. Compared to base plotting, it provides simplicity at the cost of control.

library(ggmap)

bbox <- setNames(st\_bbox(huc\_poly), c("left", "bottom", "right", "top"))

#setting zoom to 9 gives us a bit of padding around the bounding box

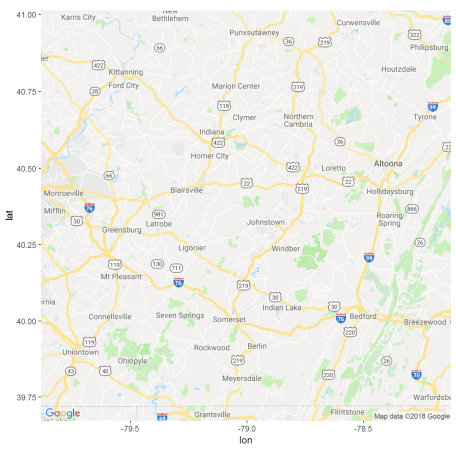
basemap\_streets <- get\_map(maptype = "roadmap", location = bbox, zoom = 9)

basemap\_satellite <- get\_map(maptype = "satellite", location = bbox, zoom = 9)

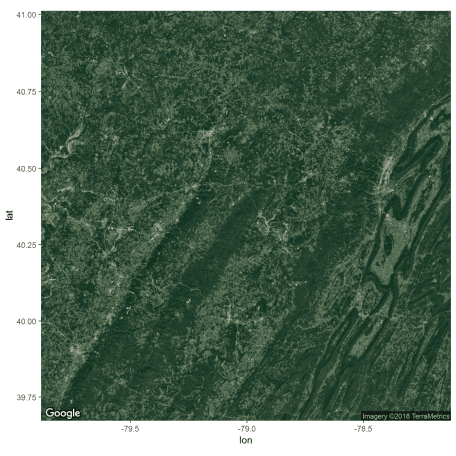
street\_map <- ggmap(basemap\_streets)

satellite\_map <- ggmap(basemap\_satellite)

print(street\_map)



print(satellite\_map)



Now we can start adding to our maps. First, we convert the huc\_gages data.frame to an sf object using st\_as\_sf, assigning it the same coordinate reference system as huc\_poly using st\_crs. ggplot functions like geom\_sf and geom\_text add to your base map.

huc\_gages\_sf <- st\_as\_sf(huc\_gages, coords = c("dec\_long\_va", "dec\_lat\_va"),

crs = st\_crs(huc\_poly), remove = FALSE)

satellite\_map + geom\_sf(data = huc\_poly,

inherit.aes = FALSE,

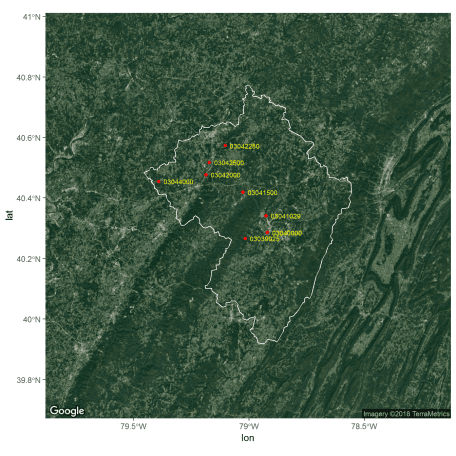
color = "white", fill = NA) +

geom\_sf(data = huc\_gages\_sf, inherit.aes = FALSE, color = "red") +

geom\_text(data = huc\_gages\_sf,

aes(label = site\_no, x = dec\_long\_va, y = dec\_lat\_va),

hjust = 0, size=2.5, nudge\_x = 0.02, col = "yellow")



street\_map + geom\_sf(data = huc\_poly,

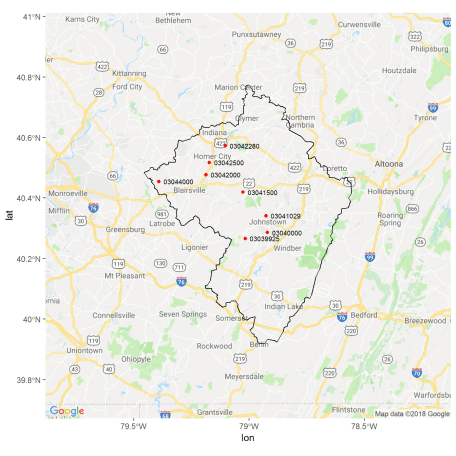
inherit.aes = FALSE,

color = "black", fill = NA) +

geom\_sf(data = huc\_gages\_sf, inherit.aes = FALSE, color = "red") +

geom\_text(data = huc\_gages\_sf, aes(label = site\_no, x = dec\_long\_va, y = dec\_lat\_va),

hjust = 0, size=2.5, nudge\_x = 0.02, col = "black")



**Vector map example**

If we don’t want any raster background to our maps, we can use base plotting and the maps package. This style of map can be nicer for insets or large scale maps that would be cluttered with a raster background. The maps package provides easily accessible political boundary maps, that can be overlaid with other shapefiles. As with regular base plotting, you can think of creating maps like painting — every layer has to go on in the right order. You can’t remove things without starting over. Fortunately, you can start over with just a few keystrokes since you are scripting your plot! Run these commands one by one to see the map take shape — first we create a blank state map, then add county lines in white, then the HUC boundary, then the gage points, and then the legend. Note that we use the st\_geometry function inside of the plot command so that we only plot the huc\_poly and huc\_gages\_sf geometry, and not the other information in their data frames.

library(maps)

map(database = 'state', regions = 'Pennsylvania', col = "tan", fill = TRUE, border = NA)

#this draws all PA counties since the regions argument uses partial matching

map(database = 'county', regions = 'Pennsylvania', col = "white", fill = FALSE, add = TRUE)

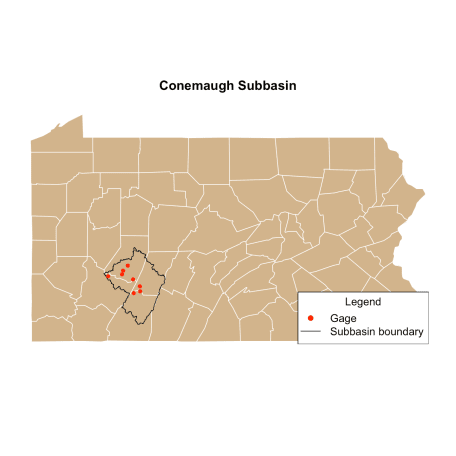
plot(st\_geometry(huc\_poly), col = NA, add = TRUE)

plot(st\_geometry(huc\_gages\_sf), add = TRUE, col = "red", pch = 19, cex = 0.7)

legend("bottomright", legend = c("Gage", "Subbasin boundary"), pch = c(19,NA), lty = c(NA, 1),

col = c("red", "black"), title = "Legend")

title("Conemaugh Subbasin")



Similarly, we can create a map zoomed in to the HUC polygon. Note that we set the x and y limits of the map by extracting the limits of the bbox object we created earlier. We can use the names left, right, etc. because bbox is a named vector.

map(database = 'county', regions = 'Pennsylvania', col = "lightgray",

xlim = bbox[c('left', 'right')], ylim = bbox[c('bottom', 'top')])

plot(st\_geometry(huc\_poly), col = "dodgerblue", border = NA, add = TRUE)

box()

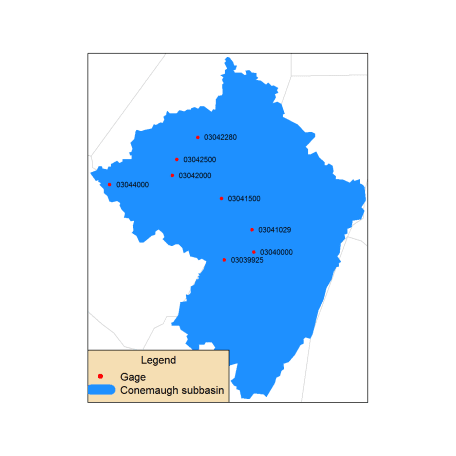
plot(st\_geometry(huc\_gages\_sf), add = TRUE, col = "red", pch = 19, cex = 0.7)

legend("bottomleft", legend = c("Gage", "Conemaugh subbasin"), pch = c(19,NA), lty = c(NA, 1),

col = c("red", "dodgerblue"), title = "Legend", lwd = c(1,15), bg = "wheat")

text(x = huc\_gages$dec\_long\_va, y = huc\_gages$dec\_lat\_va, labels = huc\_gages$site\_no,

pos = 4, cex = 0.7)



**Other packages and examples**

Like plotting in R, there are endless intricacies to making maps, and we are only really scratching the surface here. Some other packages that you may find useful for certain applications include:

* [raster](https://cran.r-project.org/web/packages/raster/index.html): For working with your own raster data
* [sp](https://cran.r-project.org/web/packages/sp/index.html): The original workhorse package for handling spatial data. sf is largely replacing it, but you will see it a lot when Googling things.
* [geoknife](https://cran.r-project.org/web/packages/geoknife/index.html): A USGS package that utilizes the [Geo Data Portal](https://cida.usgs.gov/gdp/how-to-gdp/) for processing gridded data. Covered in the [packages curriculum](https://owi.usgs.gov/R/training-curriculum/usgs-packages/).
* [inlmisc](https://cran.r-project.org/web/packages/inlmisc/index.html): Another USGS package for creating high-level graphics, [demonstrated in this blog post by Jason Fisher](https://owi.usgs.gov/blog/inlmiscmaps/).