Map-making tutorial 1- projections and datum transformations of basemaps, raster data, and sample coordinates of various file types

Hannah K. Hoff

2022-12-13

The goal of this document is to provide a template for ecologists interested in building maps to accompany their data. Here, we build a map from the ground up by performing appropriate geographic projections and transformations on basemaps, raster data, and sampling coordinates. Using packages such as ggplot2, sf, ggspatial, and tidyterra, we unlock great flexibility to customize map features using coding languages that are widely used and well-documented. Exploration of the Comprehensive R Archive Network (CRAN) will help users to explore the full capability of these packages beyond those reflected in this tutorial, and future map tutorials will build upon this framework to explore other ways to visualize spatial patterns in ecological data. Happy mapping!

Packages that may be generally useful

```
library(knitr)
library(ggplot2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(raster)
```

```
## Loading required package: sp
```

```
##
## Attaching package: 'raster'
```

```
## The following object is masked from 'package:dplyr':
##
##     select
```

```
library(ggsn)
```

```
## Loading required package: grid
```

```
##
## Attaching package: 'ggsn'
```

```
## The following object is masked from 'package:raster':
##
##     scalebar
```

```
library(rworldmap)
```

```
## ### Welcome to rworldmap ###
```

```
## For a short introduction type :   vignette('rworldmap')
```

```
library(sf)
```

```
## Linking to GEOS 3.10.2, GDAL 3.4.2, PROJ 8.2.1; sf_use_s2() is TRUE
```

```
library(gdalUtilities)
```

```
##
## Attaching package: 'gdalUtilities'
```

```
## The following object is masked from 'package:sf':
##
##     gdal_rasterize
```

```
library(ggspatial)
library(rgeos)
```

```
## rgeos version: 0.5-9, (SVN revision 684)
##  GEOS runtime version: 3.10.2-CAPI-1.16.0
##  Please note that rgeos will be retired by the end of 2023,
## plan transition to sf functions using GEOS at your earliest convenience.
##  GEOS using OverlayNG
##  Linking to sp version: 1.4-7
##  Polygon checking: TRUE
```

```
library(sp)
```

Map boundary: Load in a map boundary and convert it to long/lat coordinates (or the datum and projection of your choosing) using st_transform(). Note that if your spatial object does not already have a projection (use crs() function to check), you will need to first project it to the coordinate system it was made in, and then transform it to the system that you would like to use. Also note that all objects that we add to this map will need to be transformed to the same coordinate reference system. The basemap used was a shapefile downloaded from sciencebase.gov. Note that st_read() works for multiple file types beyond shapefiles.

```
aoi_boundary_YNP <- st_read("ab67/ab67.shp")
```

```
## Reading layer 'ab67' from data source
##   '/Users/hannahhoff/Desktop/Yellowstone data/YNP_diet_guilds/ab67/ab67.shp'
##   using driver 'ESRI Shapefile'
## Simple feature collection with 1 feature and 13 fields
## Geometry type: POLYGON
## Dimension:     XY
## Bounding box:  xmin: 469695.4 ymin: -12706.41 xmax: 573531.9 ymax: 96658.52
## Projected CRS: NAD83 / Montana
```

```
aoi_boundary_YNP
```

```
## Simple feature collection with 1 feature and 13 fields
## Geometry type: POLYGON
## Dimension:     XY
## Bounding box:  xmin: 469695.4 ymin: -12706.41 xmax: 573531.9 ymax: 96658.52
## Projected CRS: NAD83 / Montana
##         AREA PERIMETER AB67_ AB67_ID TYPE1           ADMN_TYPE1 TYPE2
## 1 8842796032  447129.5     2     649     1 Park, monument, etc.     0
##   ADMN_TYPE2 TYPE3 ADMN_TYPE3 STATE_FIPS STATE_NAME SUB_REGION
## 1       <NA>     0       <NA>         56    Wyoming        Mtn
##                         geometry
## 1 POLYGON ((559728.7 88811.52...
```

```
crs(aoi_boundary_YNP)
```

```
## [1] "PROJCRS[\"NAD83 / Montana\",\n    BASEGEOGCRS[\"NAD83\",\n        DATUM[\"North American Datum
```

```
aoi_boundary_YNP_NAD83 <- st_transform(aoi_boundary_YNP,CRS("+ proj=longlat +
                                                            datum=NAD83"))
crs(aoi_boundary_YNP_NAD83)
```

```
## [1] "GEOGCRS[\"unknown\",\n    DATUM[\"North American Datum 1983\",\n        ELLIPSOID[\"GRS 1980\",6
```

```
aoi_boundary_YNP_NAD83
```

```
## Simple feature collection with 1 feature and 13 fields
## Geometry type: POLYGON
## Dimension:     XY
## Bounding box:  xmin: -111.1543 ymin: 44.13245 xmax: -109.8339 ymax: 45.10785
## CRS:           GEOGCRS["unknown",
##     DATUM["North American Datum 1983",
##         ELLIPSOID["GRS 1980",6378137,298.257222101,
##             LENGTHUNIT["metre",1]],
##         ID["EPSG",6269]],
##     PRIMEM["Greenwich",0,
##         ANGLEUNIT["degree",0.0174532925199433],
##         ID["EPSG",8901]],
##     CS[ellipsoidal,2],
##         AXIS["longitude",east,
##             ORDER[1],
##             ANGLEUNIT["degree",0.0174532925199433,
##                 ID["EPSG",9122]]],
##         AXIS["latitude",north,
##             ORDER[2],
##             ANGLEUNIT["degree",0.0174532925199433,
##                 ID["EPSG",9122]]]]
##         AREA PERIMETER AB67_ AB67_ID TYPE1           ADMN_TYPE1 TYPE2
## 1 8842796032  447129.5     2     649     1 Park, monument, etc.     0
##   ADMN_TYPE2 TYPE3 ADMN_TYPE3 STATE_FIPS STATE_NAME SUB_REGION
## 1       <NA>     0       <NA>         56    Wyoming        Mtn
##                          geometry
## 1 POLYGON ((-110.0112 45.0478...
```

Read in metadata with coordinates. Our coordinates are in Easting/Northing, which we will change to lat/long in a later step. For now, filter out samples that don't have coordinates, and give coordinates the appropriate x and y names.

```
samples <- read.csv(file="20221214_example_metadata.csv")
samples_filter_na <- subset(samples, EASTING!= "")
class(samples_filter_na)
```

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

```
samplecoords <- samples_filter_na%>%
  rename(
    x = EASTING,
    y = NORTHING)
```

Specify coordinates in dataframe, and check the class of the object (it should now be a spatial points data frame).

```
coordinates(samplecoords)<- ~x+y
class(samplecoords)
```

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

As our points were in Easting/Northing format, we want to convert to lat/long to match the projection of our map boundary. We first assign the points to the coordinate reference system that they were collected using (Easting/Northing/UTM zone 12) using CRS() and then tranform the points to our desired coordinate reference system using spTransform(). Note that if your points already match your basemap in coordinate reference system, you will not need to transform them.

```
proj4string(samplecoords) <- CRS(
  "+proj=utm +zone=12 +datum=NAD83 +units=m +no_defs")
st_crs(samplecoords)
```

```
## Coordinate Reference System:
##   User input: +proj=utm +zone=12 +datum=NAD83 +units=m +no_defs
##   wkt:
## PROJCRS["unknown",
##     BASEGEOGCRS["unknown",
##         DATUM["North American Datum 1983",
##             ELLIPSOID["GRS 1980",6378137,298.257222101,
##                 LENGTHUNIT["metre",1]],
##             ID["EPSG",6269]],
##         PRIMEM["Greenwich",0,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8901]]],
##     CONVERSION["UTM zone 12N",
##         METHOD["Transverse Mercator",
##             ID["EPSG",9807]],
```

```
##         PARAMETER["Latitude of natural origin",0,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8801]],
##         PARAMETER["Longitude of natural origin",-111,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8802]],
##         PARAMETER["Scale factor at natural origin",0.9996,
##             SCALEUNIT["unity",1],
##             ID["EPSG",8805]],
##         PARAMETER["False easting",500000,
##             LENGTHUNIT["metre",1],
##             ID["EPSG",8806]],
##         PARAMETER["False northing",0,
##             LENGTHUNIT["metre",1],
##             ID["EPSG",8807]],
##         ID["EPSG",16012]],
##     CS[Cartesian,2],
##         AXIS["(E)",east,
##             ORDER[1],
##             LENGTHUNIT["metre",1,
##                 ID["EPSG",9001]]],
##         AXIS["(N)",north,
##             ORDER[2],
##             LENGTHUNIT["metre",1,
##                 ID["EPSG",9001]]]]
```

```
samplecoords2 <- spTransform(samplecoords,CRS("+proj=longlat +datum=NAD83"))
class(samplecoords2)
```

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

```
samplecoords2
```

```
## class       : SpatialPointsDataFrame
## features    : 9
## extent      : -110.75, -110.1835, 44.86447, 45.04495  (xmin, xmax, ymin, ymax)
## crs         : +proj=longlat +datum=NAD83 +no_defs
## variables   : 6
## names       : SampleID,  SPECIES, ANIMAL_ID,  LOCATION, COLLECTION.DATE, SEASON
## min values  :       S1, Species1,       ID1, Location1,        10/15/22,   FALL
## max values  :       S8, Species5,       ID8, Location8,         12/5/22, WINTER
```

Now, in order to use ggplot to create our maps, we need to coerce our transformed points (spatial data frame) into a normal data frame (this is just what ggplot accepts).

```
samplecoords3 <- data.frame(samplecoords2)
head(samplecoords3)
```

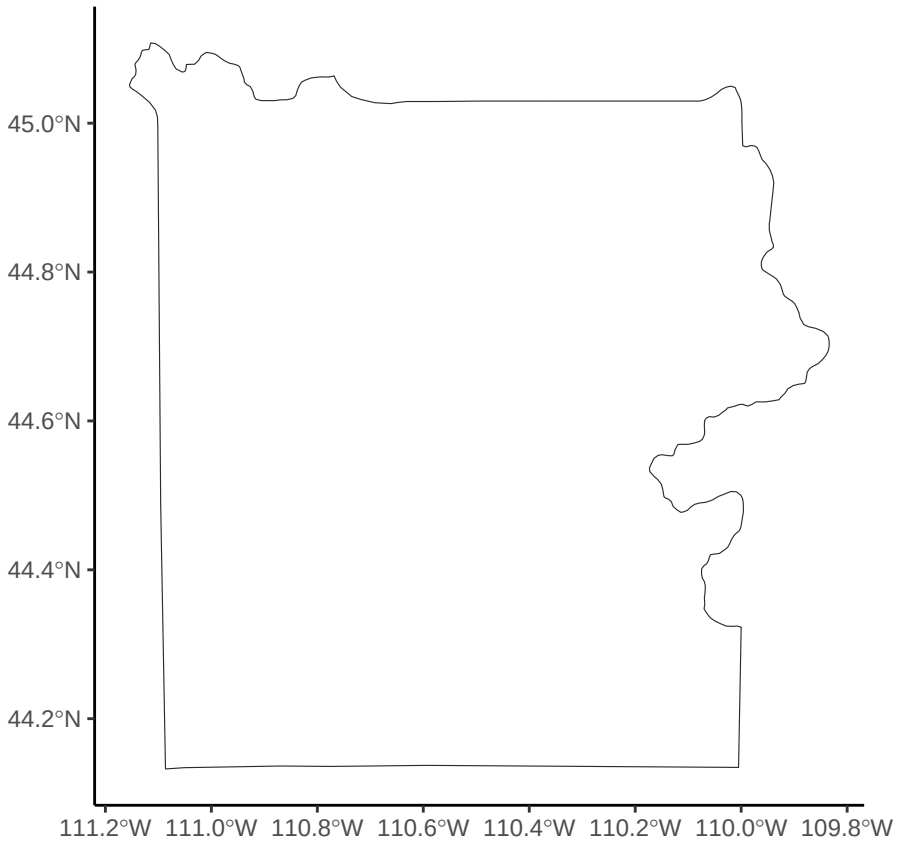
```
##   SampleID  SPECIES ANIMAL_ID  LOCATION COLLECTION.DATE SEASON         x
## 1       S1 Species1       ID1 Location1        10/15/22   FALL -110.7500
## 2       S2 Species1       ID2 Location2        10/15/22   FALL -110.1835
## 3       S3 Species2       ID3 Location3        10/30/22   FALL -110.6551
## 4       S4 Species2       ID4 Location4        10/30/22   FALL -110.6732
## 5       S5 Species3       ID5 Location5        11/15/22 WINTER -110.3509
## 6       S6 Species3       ID6 Location6        11/15/22 WINTER -110.6910
##          y optional
## 1 45.04495     TRUE
## 2 44.86447     TRUE
## 3 44.95265     TRUE
## 4 44.95353     TRUE
## 5 44.89866     TRUE
## 6 45.02206     TRUE
```

```
tail(samplecoords3)
```

```
##    SampleID  SPECIES ANIMAL_ID  LOCATION COLLECTION.DATE SEASON         x
## 4        S4 Species2       ID4 Location4        10/30/22   FALL -110.6732
## 5        S5 Species3       ID5 Location5        11/15/22 WINTER -110.3509
## 6        S6 Species3       ID6 Location6        11/15/22 WINTER -110.6910
## 7        S7 Species4       ID7 Location7        11/30/22 WINTER -110.3415
## 8        S8 Species4       ID8 Location8        11/30/22 WINTER -110.4156
## 10      S10 Species5      ID10 Location8         12/5/22 WINTER -110.3944
##           y optional
## 4  44.95353     TRUE
## 5  44.89866     TRUE
## 6  45.02206     TRUE
## 7  44.90350     TRUE
## 8  44.92019     TRUE
## 10 44.91379     TRUE
```

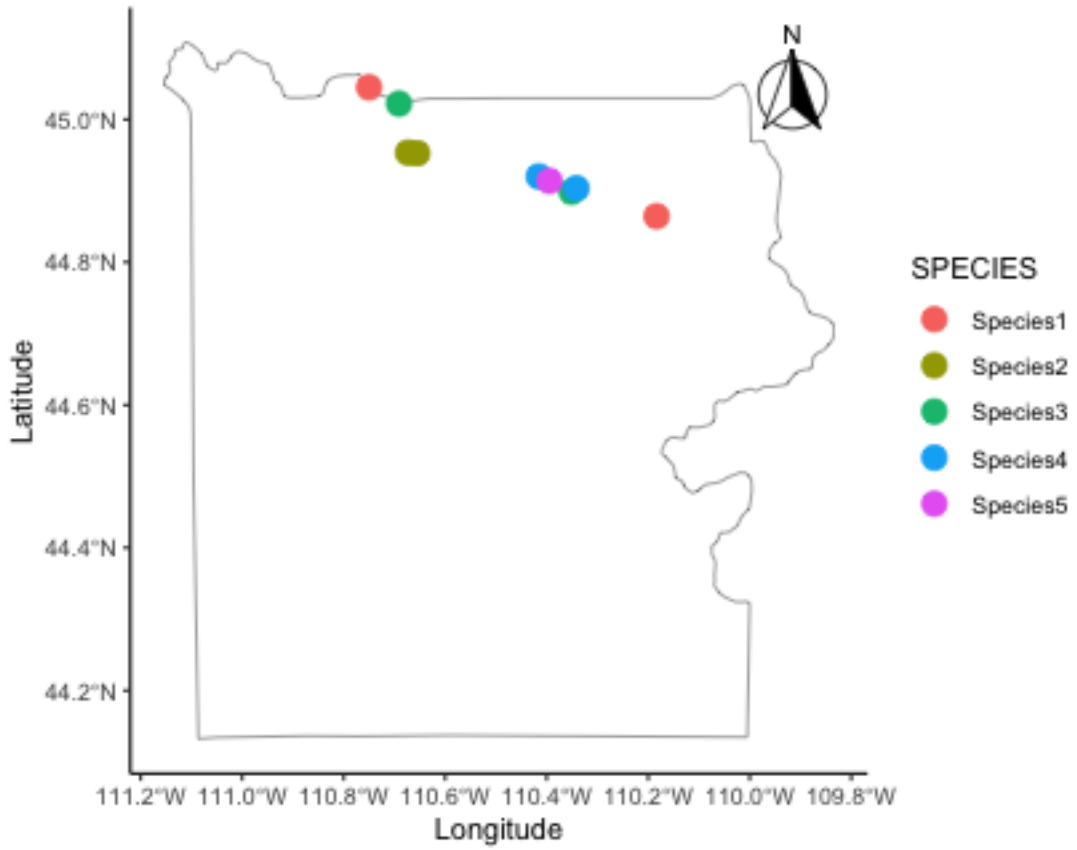
Now, we can start building the map. We can plot the base map here first, and add elements as we go (this is often a good check to make sure our elements are lining up as they should).

```
YNP_map <- ggplot() + geom_sf(data = aoi_boundary_YNP_NAD83, lwd=0, color =
    "grey17", fill=NA) + theme_classic()
YNP_map
```



Using the name of the map we created above ("YNP_map"), we can add on other map elements - like our points and a legend - note that we're just building on the above map. Note also that we are coloring our points by our "species" column and adding a north arrow; these can be customized by altering the aes() and annotation_north_arrow() code, respectively.

```
fullmap <- YNP_map + geom_point(data = samplecoords3, size = 4, aes(x=x, y=y,
  color=SPECIES)) + xlab("Longitude") + ylab("Latitude")  +
  annotation_north_arrow(location = "tr", which_north = "true",
  pad_x = unit(0.25, "cm"), pad_y = unit(0.25, "cm"),
  style = north_arrow_fancy_orienteering)
fullmap
```



When making maps, it is sometimes best to plot the map and legend separately so that their positions can be adjusted more easily when editing figures in software like Adobe Illustrator. To extract a legend to plot by itself, we can use the ggpubr package. Note that this code will only work if you have already plotted a figure with the legend you want to extract (like the figure and legend we created above). Legends can be customized using ggplot theme() calls.

```
library(ggpubr)
```

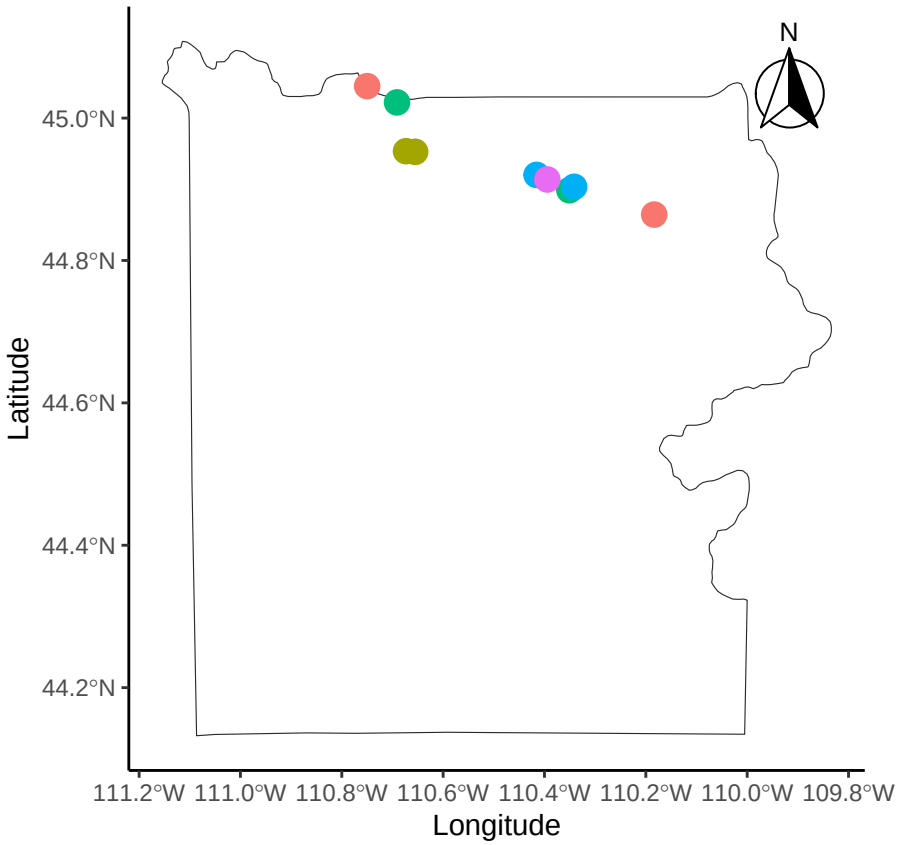
```
##
## Attaching package: 'ggpubr'
```

```
## The following object is masked from 'package:raster':
##
##     rotate
```

```
fullmap_legend <- get_legend(fullmap)
as_ggplot(fullmap_legend) +  theme(legend.key.size = unit(20,"line"))
```

Now that we've extracted our legend, we can plot our map without it using a simple theme() call.

```
fullmap + theme(legend.position="none")
```



Now that we have our coordinates on our map, we can customize it further by adding raster data. Here, we are adding elevation raster data from the "elevatr" package, and asking R to limit that raster to the bounds of our basemap. Like with other spatial objects, we will need to check the coordinate system and and re-project to same projection as our map boundary.

```
library(elevatr)
#install.packages('terra', repos='https://rspatial.r-universe.dev')
library(terra)
```

```
## Warning: package 'terra' was built under R version 4.2.2
```

```
## terra 1.6.43
```

```
##
## Attaching package: 'terra'
```

```
## The following object is masked from 'package:ggpubr':
##
##     rotate
```

```
## The following object is masked from 'package:ggsn':
##
##     north

## The following object is masked from 'package:grid':
##
##     depth

## The following object is masked from 'package:knitr':
##
##     spin
```

```
#install.packages('raster')
library(raster)
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --

## v tibble  3.1.8     v purrr   0.3.5
## v tidyr   1.2.1     v stringr 1.4.1
## v readr   2.1.2     v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x tidyr::extract() masks terra::extract(), raster::extract()
## x dplyr::filter()  masks stats::filter()
## x dplyr::lag()     masks stats::lag()
## x raster::select() masks dplyr::select()
```

```
getYNPraster <- get_elev_raster(aoi_boundary_YNP_NAD83, z = 10)
```

```
## Mosaicing & Projecting
## Note: Elevation units are in meters.
```

```
getYNPraster <- rast(getYNPraster) %>%
         mask(vect(aoi_boundary_YNP_NAD83))
```

```
## Assigning a new crs. Use 'project' to transform a SpatRaster to a new crs
## Assigning a new crs. Use 'project' to transform a SpatRaster to a new crs
```

```
## -32768
```

```
getYNPraster <- project(getYNPraster, crs(aoi_boundary_YNP_NAD83))
YNPrasterdf <- as.data.frame(getYNPraster, xy = TRUE)
names(YNPrasterdf)[3] <- "alt"
```

Now, we can create a new map with our elevation raster layer. If using an elevation raster from elevatr like in this example, replace "wiki-2.0_hypso" with your preferred raster imagery (options are returned using the unique() call for hypsometric tint palettes available in tidyterra).

```
library(ggspatial)
#remotes::install_github("dieghernan/tidyterra")
library(tidyterra)
```

```
##
## Attaching package: 'tidyterra'

## The following object is masked from 'package:raster':
##
##     select

## The following object is masked from 'package:stats':
##
##     filter
```

```
library(RColorBrewer)
library(shades)
```

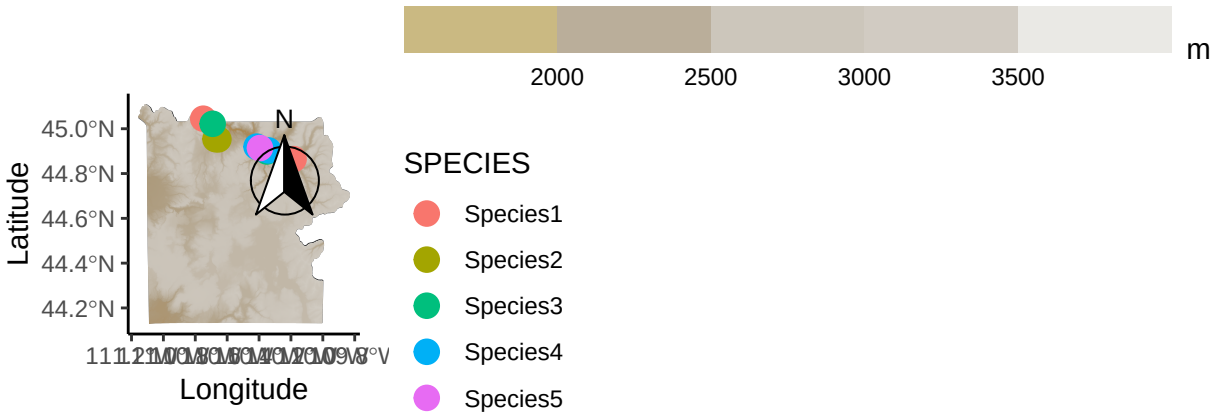
```
##
## Attaching package: 'shades'

## The following objects are masked from 'package:terra':
##
##     distance, shade

## The following object is masked from 'package:raster':
##
##     distance
```

```
palettes <- unique(hypsometric_tints_db$pal)
palettes
```

```
##  [1] "arctic"                  "arctic_bathy"
##  [3] "arctic_hypso"            "c3t1"
##  [5] "colombia"                "colombia_bathy"
##  [7] "colombia_hypso"          "dem_poster"
##  [9] "dem_print"               "dem_screen"
## [11] "etopo1"                  "etopo1_bathy"
## [13] "etopo1_hypso"            "gmt_globe"
## [15] "gmt_globe_bathy"         "gmt_globe_hypso"
## [17] "meyers"                  "meyers_bathy"
```

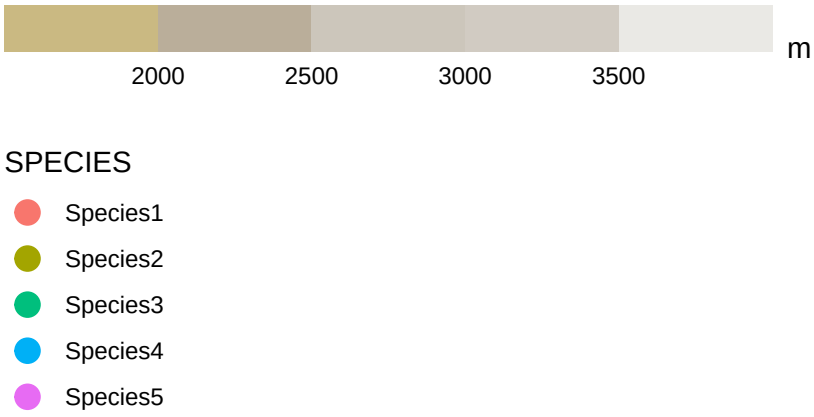
```
## [19] "meyers_hypso"              "moon"
## [21] "moon_bathy"                "moon_hypso"
## [23] "nordisk-familjebok"        "nordisk-familjebok_bathy"
## [25] "nordisk-familjebok_hypso" "pakistan"
## [27] "spain"                     "usgs-gswa2"
## [29] "utah_1"                    "wiki-2.0"
## [31] "wiki-2.0_bathy"            "wiki-2.0_hypso"
## [33] "wiki-schwarzwald-cont"
```

```
fullmapwithelevation <- YNP_map + geom_raster(data = YNPrasterdf,
    aes(x, y, fill = alt)) + geom_point(data = samplecoords3, size = 4,
    aes(x=x, y=y, color=SPECIES)) + xlab("Longitude") + ylab("Latitude")  +
    annotation_north_arrow(location = "tr", which_north = "true",
    pad_x = unit(0.25, "cm"), pad_y = unit(0.25, "cm"),
    style = north_arrow_fancy_orienteering) + guides(fill =
    guide_colorsteps(direction = "horizontal", barwidth = 20,
    title.position = "right")) + scale_fill_hypso_tint_c(palette =
    "wiki-2.0_hypso", breaks = c(2000, 2500, 3000, 3500)) + labs(fill = "m")
fullmapwithelevation
```



We can once again extract a legend for the map, this time with the elevation raster layer. Note again that this code won't work if you've excluded legends in your previous figure call.

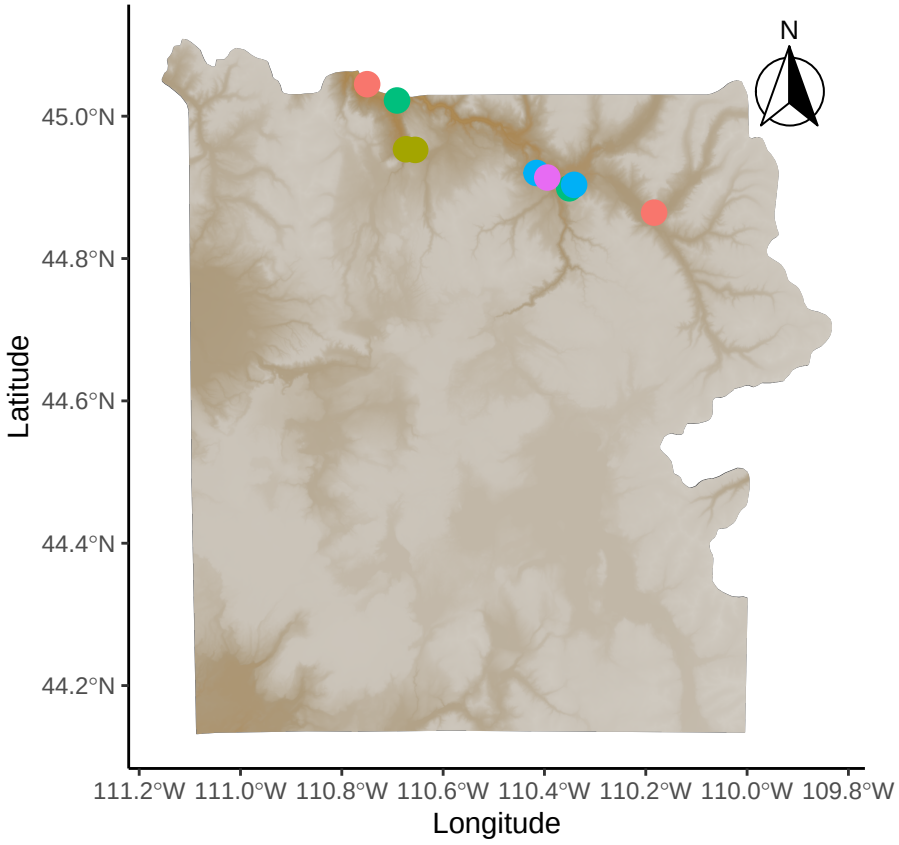
```
library(ggpubr)
fullmapwithelevation_legend <- get_legend(fullmapwithelevation)
as_ggplot(fullmapwithelevation_legend) +  theme(legend.key.size =
                                                  unit(20,"line"))
```



Now that we have our desired legends, we can plot a cleaner map. Note that there are many customization options using ggplot2 convention that are not reflected here but are available on the tidyverse and CRAN websites, in addition to many other sources.

```
library(ggspatial)
#remotes::install_github("dieghernan/tidyterra")
library(tidyterra)
library(RColorBrewer)
library(shades)
```

```
fullmapwithelevation <- YNP_map + geom_raster(data = YNPrasterdf,
    aes(x, y, fill = alt)) + geom_point(data = samplecoords3, size = 4,
    aes(x=x, y=y, color=SPECIES)) + xlab("Longitude") + ylab("Latitude")  +
    annotation_north_arrow(location = "tr", which_north = "true",
    pad_x = unit(0.25, "cm"), pad_y = unit(0.25, "cm"),
    style = north_arrow_fancy_orienteering) + guides(fill =
    guide_colorsteps(direction = "horizontal", barwidth = 20,
    title.position = "right")) + scale_fill_hypso_tint_c(palette =
    "wiki-2.0_hypso", breaks = c(2000, 2500, 3000, 3500)) + labs(fill =
    "m") + theme(legend.position = "none")
fullmapwithelevation
```

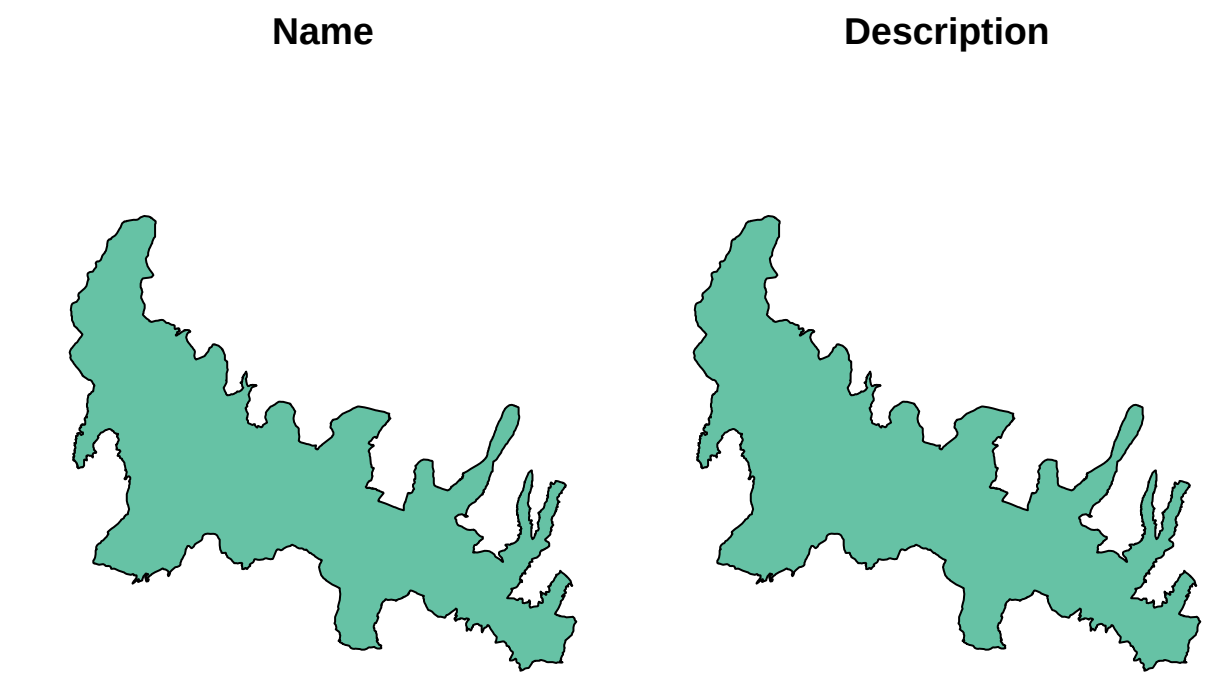


In some cases, maps elements that we want to include are not readily available in shapefile format. However, st_read() can accomodate numerous different file types. Here is an example using a .kml file of the Yellowstone northern range; we once again check the coordinate reference system and transform it to our desired projection.

```
northernrange <- st_read("nr98.kml")
```

```
## Reading layer 'Latlonnr98.shp' from data source
##   '/Users/hannahhoff/Desktop/Yellowstone data/YNP_diet_guilds/nr98.kml'
##   using driver 'KML'
## Simple feature collection with 1 feature and 2 fields
## Geometry type: POLYGON
## Dimension:     XYZ
## Bounding box:  xmin: -110.905 ymin: 44.7654 xmax: -110.027 ymax: 45.3222
## z_range:       zmin: 0 zmax: 0
## Geodetic CRS:  WGS 84
```

```
plot(northernrange)
```



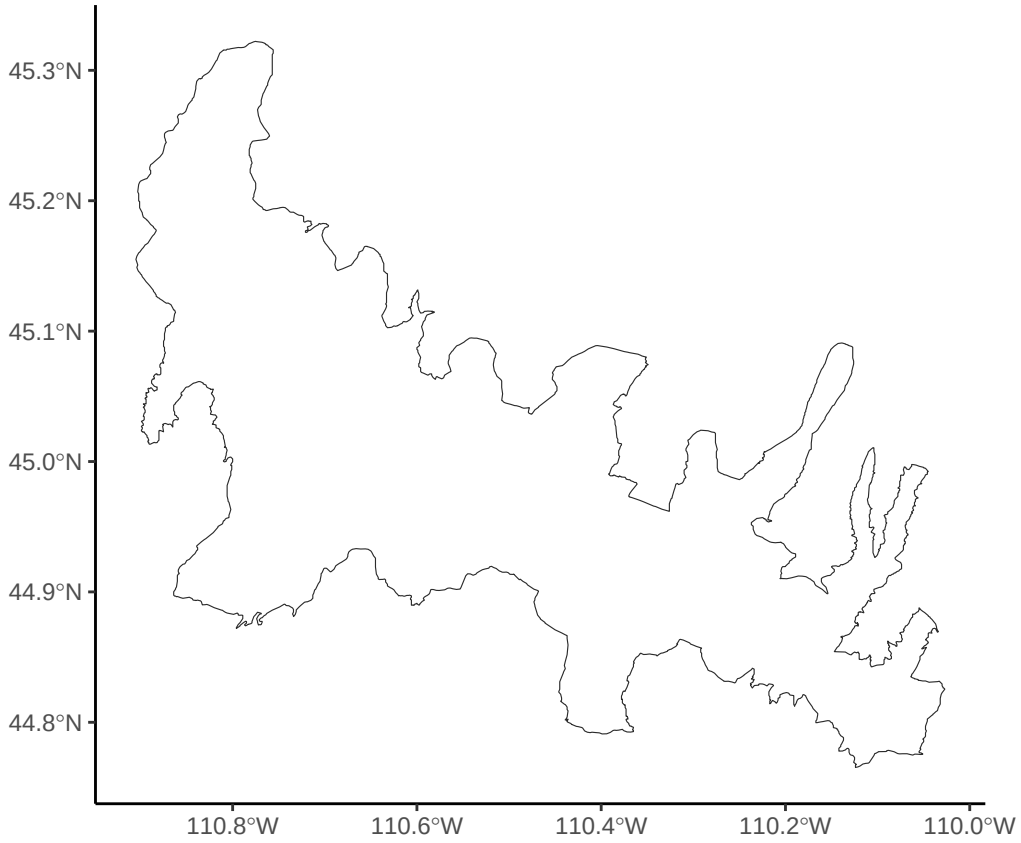
```
crs(northernrange)
```

```
## [1] "GEOGCRS[\"WGS 84\",\n    DATUM[\"World Geodetic System 1984\",\n        ELLIPSOID[\"WGS 84\",63
```

```
northernrange_NAD83 <- st_transform(northernrange,CRS("+proj=longlat +
                                                       datum=NAD83"))
crs(northernrange_NAD83)
```

```
## [1] "GEOGCRS[\"unknown\",\n    DATUM[\"North American Datum 1983\",\n        ELLIPSOID[\"GRS 1980\",
```

```
northernrangemap <- ggplot() + geom_sf(data = northernrange_NAD83, lwd=0,
    color = "grey17", fill=NA) + theme_classic()
northernrangemap
```



To add raster data to this object, we follow a similar pipeline as before - in the case of the .kml file, we need to first drop the z-dimension from the feature geometry using st_zm().

```
northernrangezm <- st_zm(northernrange_NAD83)
getnorthernraster <- get_elev_raster(northernrangezm, z = 10)
```

```
## Mosaicing & Projecting

## Note: Elevation units are in meters.
```

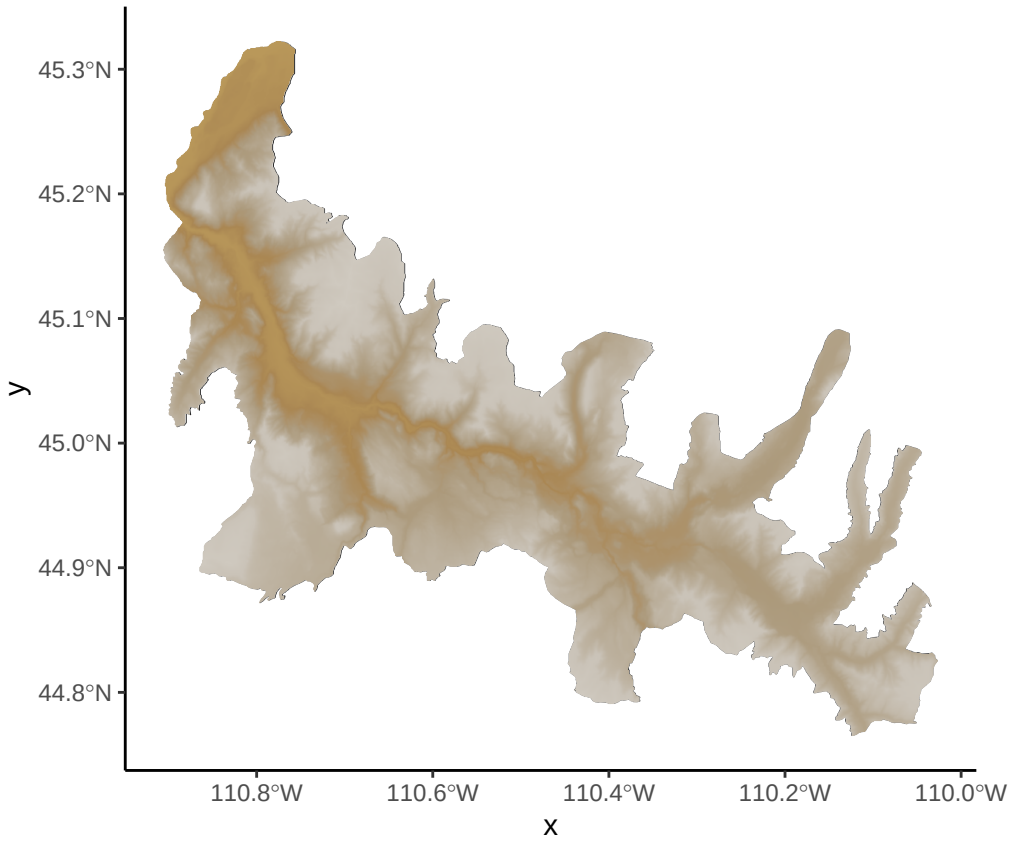
```
getnorthernraster <- rast(getnorthernraster) %>%
         mask(vect(northernrange_NAD83))
```

```
## Assigning a new crs. Use 'project' to transform a SpatRaster to a new crs

## Assigning a new crs. Use 'project' to transform a SpatRaster to a new crs

## -32768
```

```
getnorthernraster <- project(getnorthernraster, crs(northernrange_NAD83))
northernrasterdf <- as.data.frame(getnorthernraster, xy = TRUE)
names(northernrasterdf)[3] <- "alt"
```

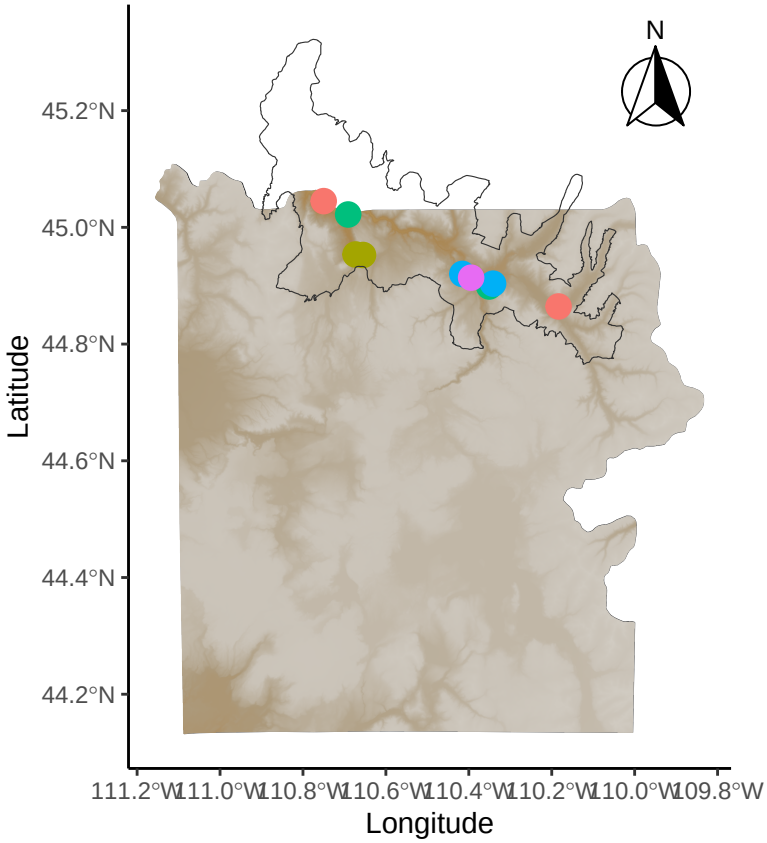
Once our .kml file has been converted, we can map it by itself. Note that "breaks" in elevation should change based on the elevations of your mapped areas and how you choose to "bin" them.

```
northernrangemap <- ggplot() + geom_sf(data = northernrange_NAD83, lwd=1,
    color = "grey17", fill=NA) + theme_classic()
northernrangepluselevation <- northernrangemap + geom_raster(data =
    northernrasterdf, aes(x, y, fill = alt)) + scale_fill_hypso_tint_c(palette =
    "wiki-2.0_hypso", breaks = c(2000, 2500, 3000, 3500)) +
    theme(legend.position = "none")
northernrangepluselevation
```



We can also add this feature to the full map we created before. Here, I'm just adding it as an outline but this can easily be changed (adjust the section that says "fill = transparent"). We now have a map with a boundary, raster layer with elevation, sample points, and a shapefile (northernrange) overlaid on top.

```
fullmapwithelevation <- YNP_map + geom_raster(data = YNPrasterdf,
  aes(x, y, fill = alt)) + geom_point(data = samplecoords3, size = 4,
  aes(x=x, y=y, color=SPECIES)) + xlab("Longitude") + ylab("Latitude")  +
  annotation_north_arrow(location = "tr", which_north = "true",
  pad_x = unit(0.25, "cm"), pad_y = unit(0.25, "cm"),
  style = north_arrow_fancy_orienteering) + guides(fill =
  guide_colorsteps(direction = "horizontal", barwidth = 20, title.position =
  "right")) + scale_fill_hypso_tint_c(palette = "wiki-2.0_hypso", breaks =
  c(2000, 2500, 3000, 3500)) + labs(fill = "m") + theme(legend.position =
  "none") + geom_sf(fill = "transparent", color = "gray20", size = 1, data =
  northernrange_NAD83)
fullmapwithelevation
```


N
45.2°N
45.0°N
44.8°N
44.6°N
44.4°N
44.2°N
Latitude
111.2°W
111.0°W
110.8°W
110.6°W
110.4°W
110.2°W
110.0°W
109.8°W
Longitude