spatial_laboratory_2_Bee_Richness

Steppe

1/22/2022

Contents

Lab Overview

The goal of today's lab is to showcase the lovely raster data model. Our other goal is to identify suitable habitat for ground nesting bees in the Upper East River Drainage, a sub-alpine valley of Parkland Meadow near Crested Butte Colorado. Ground nesting bees make up approximately 65% of bee diversity, yet their conservation remains a challenge due to a lack of basic life history information. One of the primary challenges of studying ground-nesting bees is locating nesting sites, bees are known to make there nests on south facing slopes with less than 10% plant cover, for this lab we will identify potential suitable nesting habitat for these elusive bees.

We will use field data collected at this field station by Amelia Litz (Northwestern University), with some individual bees in challenging taxonomic groups identified by Michael Stemkovski (Utah State University).

Her field data include incidental counts of the species richness of ground nesting bees, at a resolution of 1mˆ2.

All of our raster data will be data collected and processed by Ian Breckheimer a staff scientist at the Rocky Mountain Biological Laboratory in Gothic Colorado. If you are interested in how these data were acquired please refer to the specific links below. If you are interested in spatial data science the following six part series is a must.

RMBL Spatial Data Science Webinar Series: - https://github.com/ikb-rmbl/SpatialDataScienceWebinars2020 - Collecting UAS Data: https://youtu.be/Pq8btEZRCvM

Libraries

```
# install.packages("sf", "terra", "raster", "sp", "tmap", "leaflet", "ggmap", "GGally")

# optional packages for using parallel processing at a step
# (in this case it won't actually speed step this up - it may slow it down!)
# install.packages('snow','parallel')
```

We included all but one data set we will use for this lab in the course materials on Canvas. You are going to download the final file in the same way that the files on Canvas were downloaded. Please download these materials from canvas right now, when your packages are installing.

```
shhh <- suppressPackageStartupMessages
## some of the spatial libraries create messages on start up ad nauseum.
## we will suppress these messages so we can focus on the code ahead.

shhh(library(sf))
shhh(library(raster))
shhh(library(terra))
shhh(library(tidyverse))
shhh(library(leaflet))
shhh(library(tmap))
shhh(library(ggmap))
shhh(library(GGally))

rm(shhh)
```

note we will rm(), to remove objects from the environment ALOT today. Due to the size of geospatial objects, and how long it can take to run analyses we want to both minimize the computer RAM used to store some of the objects and not confuse ourselves about whether we have completed a step yet or not (many scripts take weeks to develop).

Import Data

Half the battle can be getting the files into R. When you start to scale spatial analysis you may be feeding hundreds of files in.

Import Local Spatial Files

I made a simple feature dataset which contains all of the bee traps. We will import it using a method of listing directories (folders) sub-setting the appropriate folder, and then listing the files in that directory. We will use simple regular expressions in both instances.

```
directories <- list.dirs()
# list all folders in current directory

directories <- directories[which(stringr::str_detect(directories, "spatial_lab"))]
# filter for the right folder
```

We can supply a 'pattern' argument to the 'list.files' function, which will have R search for a certain portion of a file name using regular expressions. This is a very powerful tool, and I encourage you all to become

familiar with it. Note that my pattern here is over the top, I included a '$' at the end which specifies that we the letters 'traps.shp' must occur at the end of the file name.

Here We will search for the files ending with the pattern '.shp'. Note that to date we have largely discussed two data models; 'vector' or 'raster'. Within R we have discussed both the SP, and SF types of vector data models. The 'shapefile' abbreviated as '.shp' is a very popular format for saving vector data developed by the ESRI company (ArcMAP, ArcGIS), but with open source compatibility vector file != shapefile. shp. == shapefile! This file format is so common you will people often refer all vector data as 'shapefiles', if you do this your TA will no longer grade nicely.

```
shapefiles <- list.files(directories, pattern = "traps.shp$")
```

We can now use the 'paste0' function to concatenate our character objects 'directories', and 'shapefiles', and have them separated by a '/'. This is a quick way of allow us to save the file path, and name to a variable - 'shapefiles'. Note: 'paste0' is related to 'paste', which will combine character objects BUT include a ' ' (space!) by default between them, paste0 defaults to: '' (no space!).

```
shapefiles <- paste0(directories, "/", shapefiles)
```

Since our data are already in a spatial format (a shapefile), we can use 'st_read' to import them. This is analogous to read_csv, read_excel etc.

```
sites <- st_read(shapefiles, quiet = TRUE) %>%
  filter(SITE != '403')

rm(shapefiles)
```

The first thing we can do after importing this file is to determine whether all of our geometries are valid under the Simple Feature schema. To be honest if all you want to do is map a feature, you do not really need your features to be valid, but the second you want to use them for analyses, you will begin to have problems crop up if your features are not valid. You should ensure the validity of data you have not worked with before upon import.

```
st_is_valid(sites)
```

```
 [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[16] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[31] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

We can also check what the CRS of our vector data is right after import. When we do spatial analyses we will always transform the CRS of the vector to match the CRS of the rasters. It takes much longer to transform the CRS of a raster than a vector.

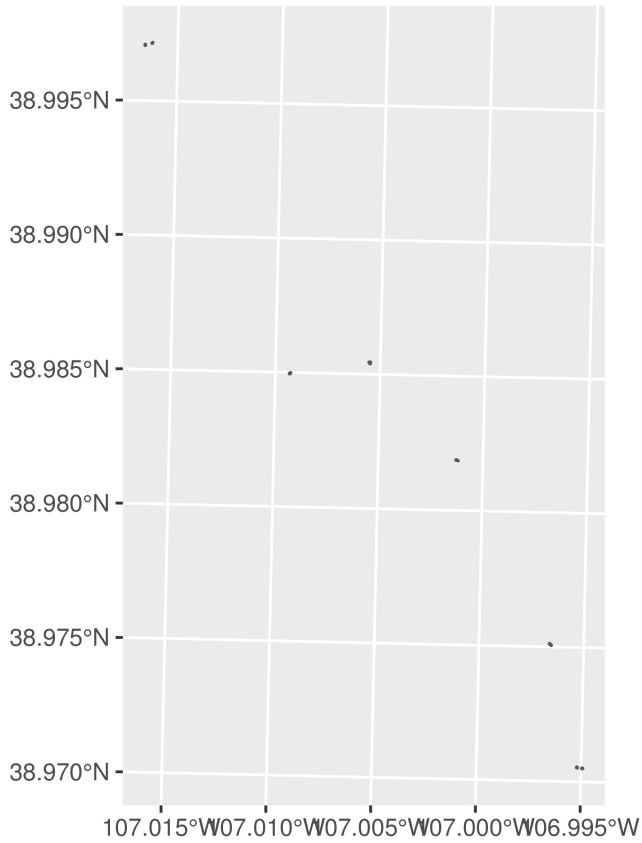
```
st_crs(sites)
```

```
Coordinate Reference System:
  User input: WGS 84 / UTM zone 13N
  wkt:
PROJCRS["WGS 84 / UTM zone 13N",
    BASEGEOGCRS["WGS 84",
        DATUM["World Geodetic System 1984",
            ELLIPSOID["WGS 84",6378137,298.257223563,
                LENGTHUNIT["metre",1]]],
        PRIMEM["Greenwich",0,
            ANGLEUNIT["degree",0.0174532925199433]],
        ID["EPSG",4326]],
    CONVERSION["UTM zone 13N",
        METHOD["Transverse Mercator",
            ID["EPSG",9807]],
```

```
        PARAMETER["Latitude of natural origin",0,
            ANGLEUNIT["Degree",0.0174532925199433],
            ID["EPSG",8801]],
        PARAMETER["Longitude of natural origin",-105,
            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]]],
    CS[Cartesian,2],
        AXIS["(E)",east,
            ORDER[1],
            LENGTHUNIT["metre",1]],
        AXIS["(N)",north,
            ORDER[2],
            LENGTHUNIT["metre",1]],
    ID["EPSG",32613]]
```

It is also good practice to always inspect the data using mapping techniques. This is helpful to look for errors, and outliers, but to also give you - the analyst- a sense of what these data represent and came from.

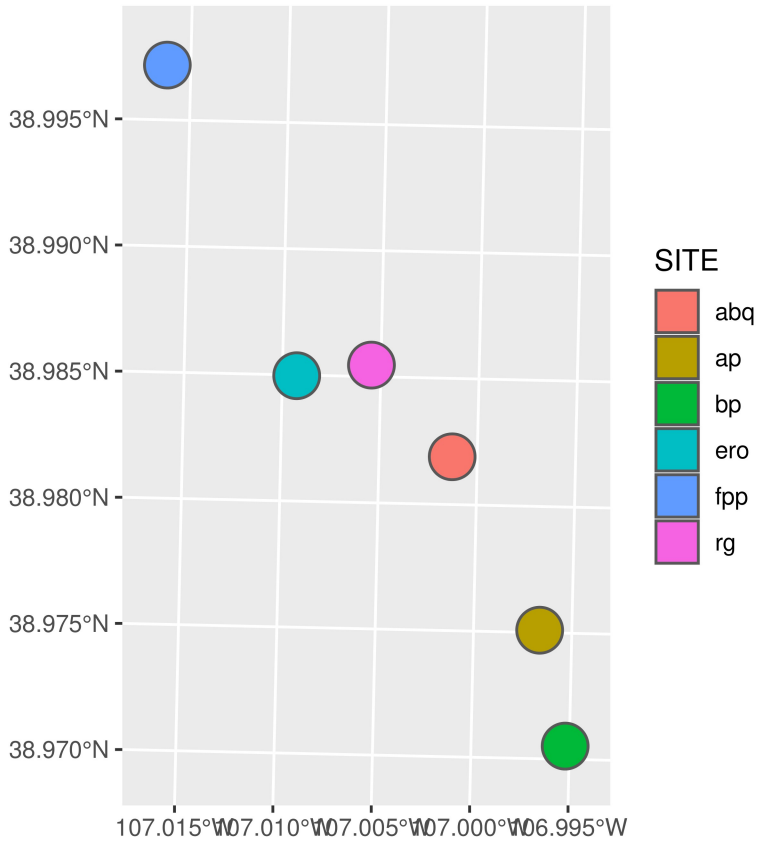
```
ggplot(sites) +
  geom_sf()
```



. . . That was not very helpful!! I always forget the world is large, and what I study is so small. In this case our plots are very small, a meter by a meter, and evidently separated by some great distance relative to there sizes. . . Maybe we can enlarge the plots somehow?

Here we will select one plot per site, and then draw out 100 meter radius circle from it, and map this.

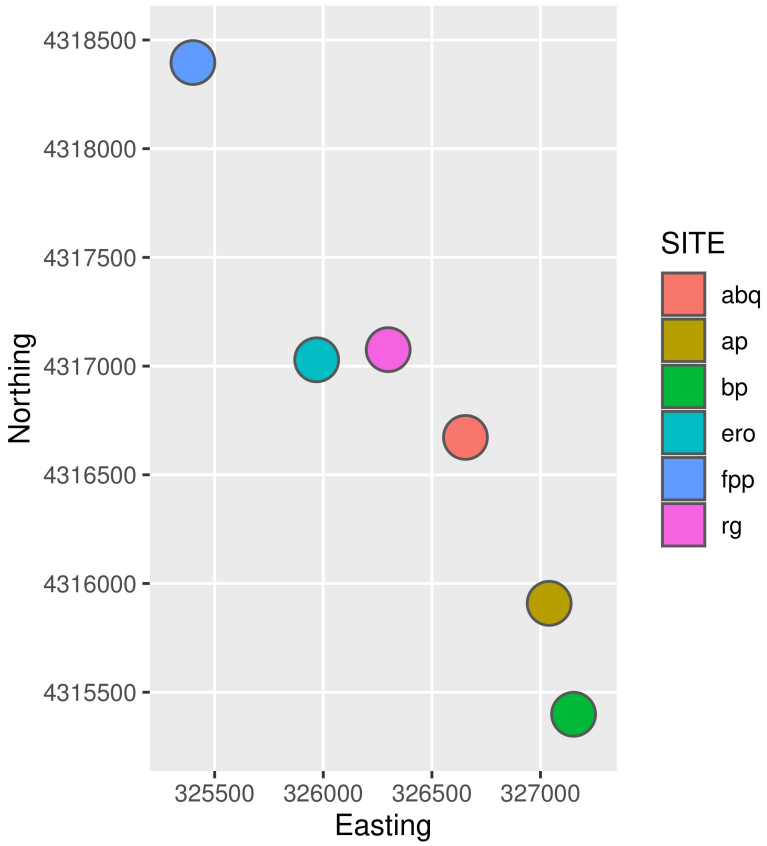
```
sites %>%
  group_by(SITE) %>% # we want to see our SITES
  slice(1) %>% # there are several PLOTs per SITE, grab one
  st_buffer(100) %>% # we will put a 200 meter diameter circle around it
  ggplot() +
  geom_sf(aes(fill = SITE))
```



There we go, now we can see the relative configuration of the study sites without squinting.

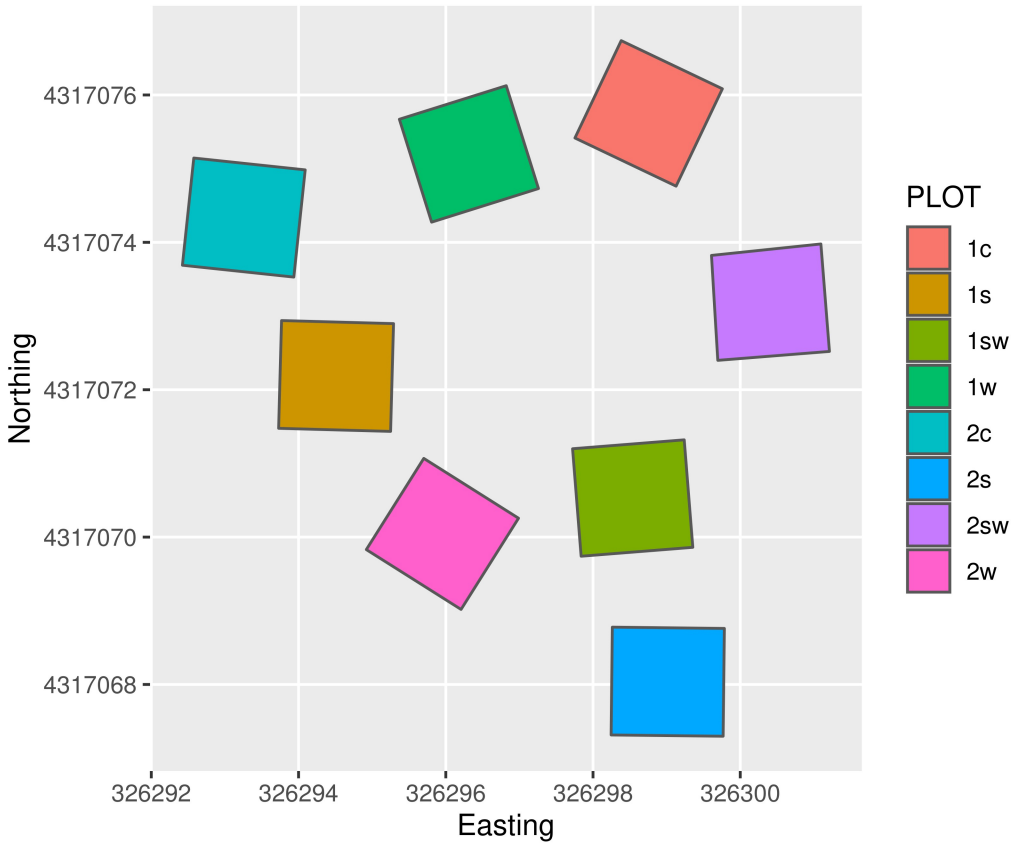
Ok nobody thinks in DD, because I know how far apart these sites are, I will let you all know that we can use a UTM projection to view these data. A UTM projection more or less takes a longitudinal section of the earth, and just flattens it (...as projections do). What is really nice and convenient about these projections is that the distances are measured in meters. So we can visually assess the distance between features in a way we can readily think about.

```
sites %>%
  group_by(SITE) %>% # we want to see our SITES
  slice(1) %>% # there are several PLOTS per SITE, grab one
  st_buffer(100) %>% # we will put a 200 meter diameter circle around it
  ggplot() +
  geom_sf(aes(fill = SITE)) +
  coord_sf(datum = st_crs(32613)) +
  labs(x = 'Easting', y = 'Northing')
```



Obviously these maps are of the relative position of the sites to each other, but we are curious to see the orientation of plots at a site as well.

```
sites %>%
  filter(SITE == 'rg') %>%
  ggplot() +
  geom_sf(aes(fill = PLOT))  +
  coord_sf(datum = st_crs(32613)) +
  labs(x = 'Easting', y = 'Northing')
```



These static maps have been helpful in understanding where these sites and plots are in relation to each other. But I still don't think we have a good grasp of where they are in the world.

R studio has some incredible built in interactivity! We can use Leaflet to launch a mini-web map, but instead of publishing our map to the web, we can just navigate around our study sites in Colorado! Take a look below.

```
leaflet(data = sites) %>%
  addTiles() %>%
  addMarkers(~LONG_WGS, ~LAT_WGS,
             popup = ~as.character(SITE),
             label = ~as.character(SITE))
```

Where are we in relation to Crested Butte? Denver? Durango? When you are done with this map click the 'x' (clear output) button in the upper left corner to exit out. If a thin blackish bar persists in your rstudio console use the scroll bar to go all the way to thebottom of your script and back up.

Import Tabular data

OK we now know where our study sites are located, but we do not have any information of attributes of them really (aside from their names). There is another file in this folder, this time a csv file which contains tabular data. Please load it, using the same method we showed you above (or a similar one).

```
tabular <- paste0(directories, "/", list.files(directories, pattern = ".csv$"))
bee_ids <- read.csv(tabular)

head(bee_ids)
```

```
              SPECIES NUMBER_IND INDIVID_ID            SITE_NAME PLOT_ID
1   lasioglossum_sedi          1          4          beaver_pond    BP2C
2 lasioglossum_nigrum          1          5 friends_pond_parking   FPP1C
3 halictus_rubicundus          1          6          beaver_pond    BP2C
4 lasioglossum_nigrum          5          7         avery_picnic    AP2C
5 lasioglossum_nigrum          5          8         avery_picnic    AP2C
6 lasioglossum_nigrum          5          9         avery_picnic    AP2C
```

Ok it looks like this is a data frame of each bee species, and how many thereof, were collected at each plot.

We will Use the code below to join the tabular data to the vector data

```
sites <- sites %>%
  mutate(across(SITE:PLOT, ~ str_to_upper(.x))) %>%
  unite('PLOT_ID', SITE, PLOT, sep = "", remove = F) %>%
  left_join(., bee_ids, by = "PLOT_ID") %>%
  mutate(across(NUMBER_IND:INDIVID_ID, ~ replace_na(.x, 0)))

rm(tabular, bee_ids)
```

Let's take a quick look at these data now.

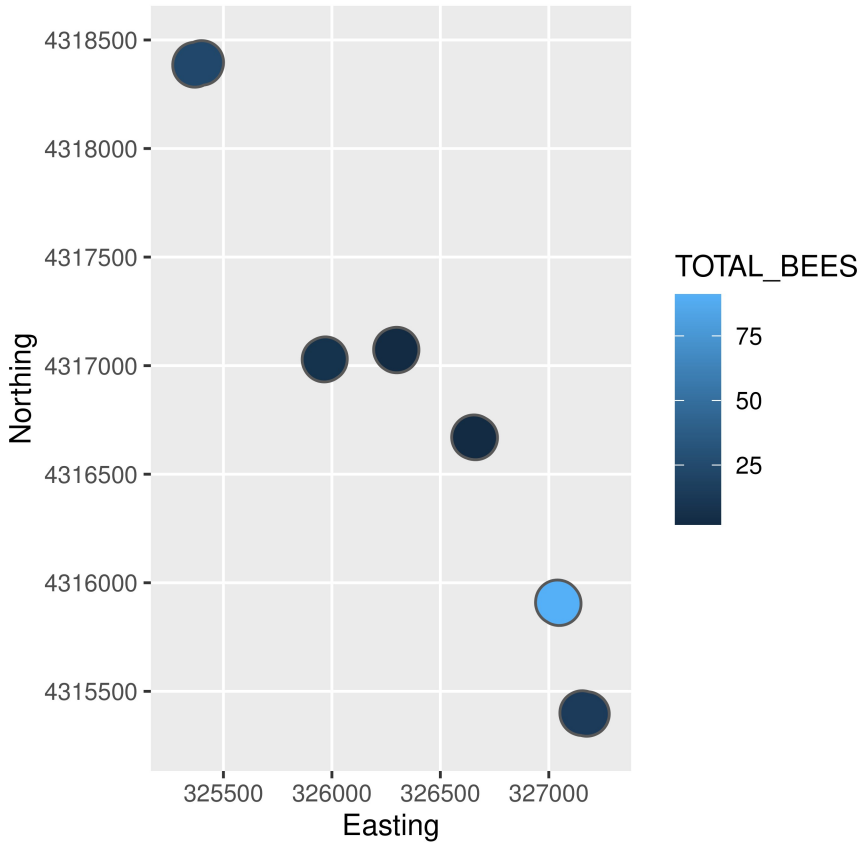
how many bees were collected per SITE?

```
sites %>%
  group_by(SITE) %>%
  st_drop_geometry() %>% # just for printing !!!
  summarize(TOTAL_BEES = sum(NUMBER_IND))
```

```
# A tibble: 6 x 2
  SITE  TOTAL_BEES
  <chr>      <int>
1 ABQ            2
2 AP            91
3 BP            15
4 ERO            8
5 FPP           24
6 RG             2
```

Can we map the intensity of collections in space?

```
sites %>%
  group_by(SITE) %>% # we want to see our SITES
  summarize(TOTAL_BEES = sum(NUMBER_IND)) %>%
  st_buffer(100) %>% # we will put a 200 meter diameter circle around it
  ggplot() +
  geom_sf(aes(fill = TOTAL_BEES)) +
  coord_sf(datum = st_crs(32613)) +
  labs(x = 'Easting', y = 'Northing')
```



We see that sites vary greatly in how many bees live at them. AP, has enormous amounts of bees, RG, and ABQ do not. We will need to keep this in mind when we make our simple predictions.

Download Data from from the RMBL Spatial Data Science Platform.

While we now have spatial data with features, we do not have any environmental variables associated with these sites. Remember these types of variables are almost always stored in Raster Data Format. We will import some Rasters from the Rocky Mountain Biological Stations (RMBL) Spatial Data Platform (SDP). For the sake of internet bandwidth, and pedagogical purposes we will import two rasters from online today, and import four from the class data folder.

We may access all of the RMBL spatial products directly from R !!! :-) !!!

we can pull in a csv of all the spatial products available for this research station directly through the web link below. These products have just begun to be actively developed and released in the last couple years. Hence these are early phase products, and you will see many are produced in the process of producing other products. For example high resolution digital elevation models, are essential for vegetation canopy height models, and nearly all hydrological models.

```
sdp_prods <-
  read.csv("https://www.rmbl.org/wp-content/uploads/2021/04/SDP_product_table_4_26_2021.csv")
```

An important part of GIS work is understanding 1) what geographic *extent* your study will cover, at 2) what *resolution*, and what 3) *thematic content* you need to research your study question.

We should explore these topics using the contents of this CSV. Please try to tell me what *resolution* (i.e. the pixel/grid cell size of the rasters) of data are available, what described spatial *extents* are available, and what some major and interesting *thematic content* types are (either to you as a researcher, or for this project)?

```r
## Using the 'spatial data products (SDP)' inventory, please tell me
head(sdp_prods)
```

```
   Release  Type                                        Product Domain
1 Release1 Hydro  Large Stream Flowlines (Multi-flow Direction)    UER
2 Release1 Hydro Large Stream Flowlines (Single-flow Direction)    UER
3 Release1 Hydro       Large Watersheds (Multi-flow Direction)    UER
4 Release1 Hydro             Multi-direction Flow Accumulation    UER
5 Release1 Hydro      Small Watersheds (Single Flow Direction)    UER
6 Release1 Hydro            Single Direction Flow Accumulation    UER
  Resolution
1         1m
2         1m
3         1m
4         1m
5         1m
6         1m
                                                                                                                     Data
1 https://rmbl-sdp.s3.us-east-2.amazonaws.com/data_products/released/release1/UER_streams_512k_mfd_1m_v
2 https://rmbl-sdp.s3.us-east-2.amazonaws.com/data_products/released/release1/UER_streams_512k_sfd_1m_v
3  https://rmbl-sdp.s3.us-east-2.amazonaws.com/data_products/released/release1/UER_basins_512k_mfd_1m_v
4   https://rmbl-sdp.s3.us-east-2.amazonaws.com/data_products/released/release1/UER_flow_accum_mfd_1m_v
5   https://rmbl-sdp.s3.us-east-2.amazonaws.com/data_products/released/release1/UER_basins_32k_sfd_1m_v
6   https://rmbl-sdp.s3.us-east-2.amazonaws.com/data_products/released/release1/UER_flow_accum_sfd_1m_v

1 https://rmbl-sdp.s3.us-east-2.amazonaws.com/data_products/released/release1/UER_streams_512k_mfd_1m_v
2 https://rmbl-sdp.s3.us-east-2.amazonaws.com/data_products/released/release1/UER_streams_512k_sfd_1m_v
3  https://rmbl-sdp.s3.us-east-2.amazonaws.com/data_products/released/release1/UER_basins_512k_mfd_1m_v
4   https://rmbl-sdp.s3.us-east-2.amazonaws.com/data_products/released/release1/UER_flow_accum_mfd_1m_v
5   https://rmbl-sdp.s3.us-east-2.amazonaws.com/data_products/released/release1/UER_basins_32k_sfd_1m_v
6   https://rmbl-sdp.s3.us-east-2.amazonaws.com/data_products/released/release1/UER_flow_accum_sfd_1m_v
```

```r
## what geographic extents do the products cover ?
distinct(sdp_prods, Domain)
```

```
  Domain
1    UER
2     UG
3     GT
```

```r
count(sdp_prods, Domain)
```

```
  Domain  n
1     GT  6
2    UER 43
3     UG 25
```

```r
## what resolutions are the products at ?
distinct(sdp_prods, Resolution)
```

```
  Resolution
1         1m
2         3m
3     0.333m
4        5cm
5         2m
```

```
count(sdp_prods, Resolution)
```

```
  Resolution  n
1     0.333m  1
2         1m 43
3         2m  7
4         3m 17
5        5cm  6
```

```
## what types of data do most of these early releases cover?
distinct(sdp_prods, Type)
```

```
           Type
1         Hydro
2          Maks
3          Mask
4          Topo
5    Vegetation
6      Planning
7     Radiation
8          Snow
9       Imagery
10 Supplemental
```

```
count(sdp_prods, Type)
```

```
           Type  n
1         Hydro 16
2       Imagery  7
3          Maks  1
4          Mask  4
5      Planning  2
6     Radiation  8
7          Snow  4
8  Supplemental  4
9          Topo 13
10   Vegetation 15
```

If you are all done and the class has not caught up, do you have any idea of a content theme we can use to study our question?

Now we will begin the process of downloading a particular data set. We will start with the basic land cover product, for the Upper East River (UER) Domain. Here we will extract the URL leading to one of these products.

```
landcover_uri <- sdp_prods %>%
  filter(Product == 'Basic Landcover' & Domain == "UER") %>%
  pull(Data.URL)

## Using the SDP csv, please repeat this step for another product,
## the 'Summer Bare-Earth Solar Radiation' product.

solar_radiation_uri <- sdp_prods %>%
  filter(Product == 'Summer Bare-earth Solar Radiation') %>%
  pull(Data.URL)

## I am going to also extract one of the URL's for metadata for an example
```

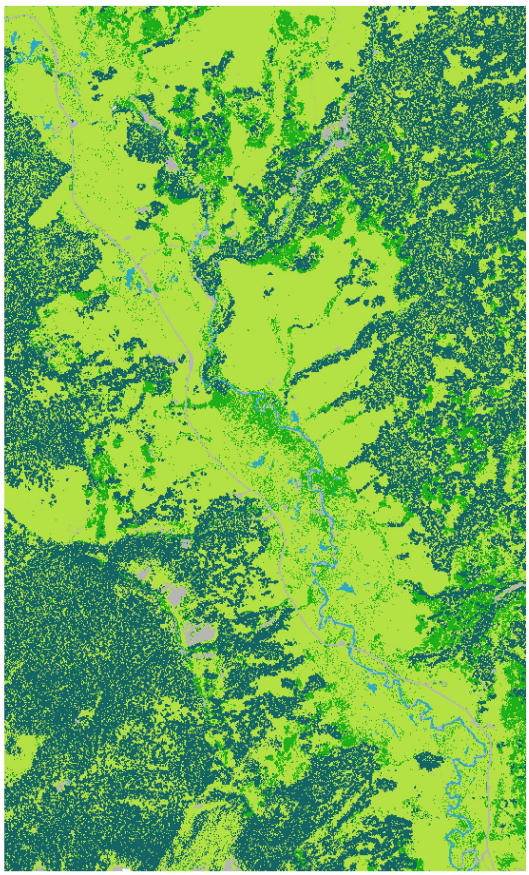
```
## which you will see below.

landcover_md_uri <- sdp_prods %>%
  filter(Product == 'Basic Landcover' & Domain == "UER") %>%
  pull(Metadata.URL)

rm(sdp_prods)
```

Here we will go through the steps of downloading our land cover product from Amazon Cloud. We are going to use access options created by the *Geospatial Data Abstraction Library* for reading in data from remote sources over the internet.

```
## Create a full url path for download
## note that 'vsi' refers to a 'virtual file system' (developed by GDAL),
## and will allow us to 'load' this file into our environment
## through the internet.
landcover_path <- paste0("/vsicurl/",landcover_uri)

## we use the 'raster' function to import these data, we will also tell
## raster to create a text bar so we know how the process is going
# landcover <- raster(landcover_path, progress='text')

## if you cannot access this product via the web it is also in your folder.
landcover <- raster('spatial_lab_data/landcover.gri')

## we now crop the raster to the extent of our study area
landcover <- crop(landcover,extent(sites))

## how big is this raster?
os_lc <- object.size(landcover)
## remember 1000 (or 1024... (preferred)) Mb in a Gb (Gib)
print(os_lc, units = "auto")
```

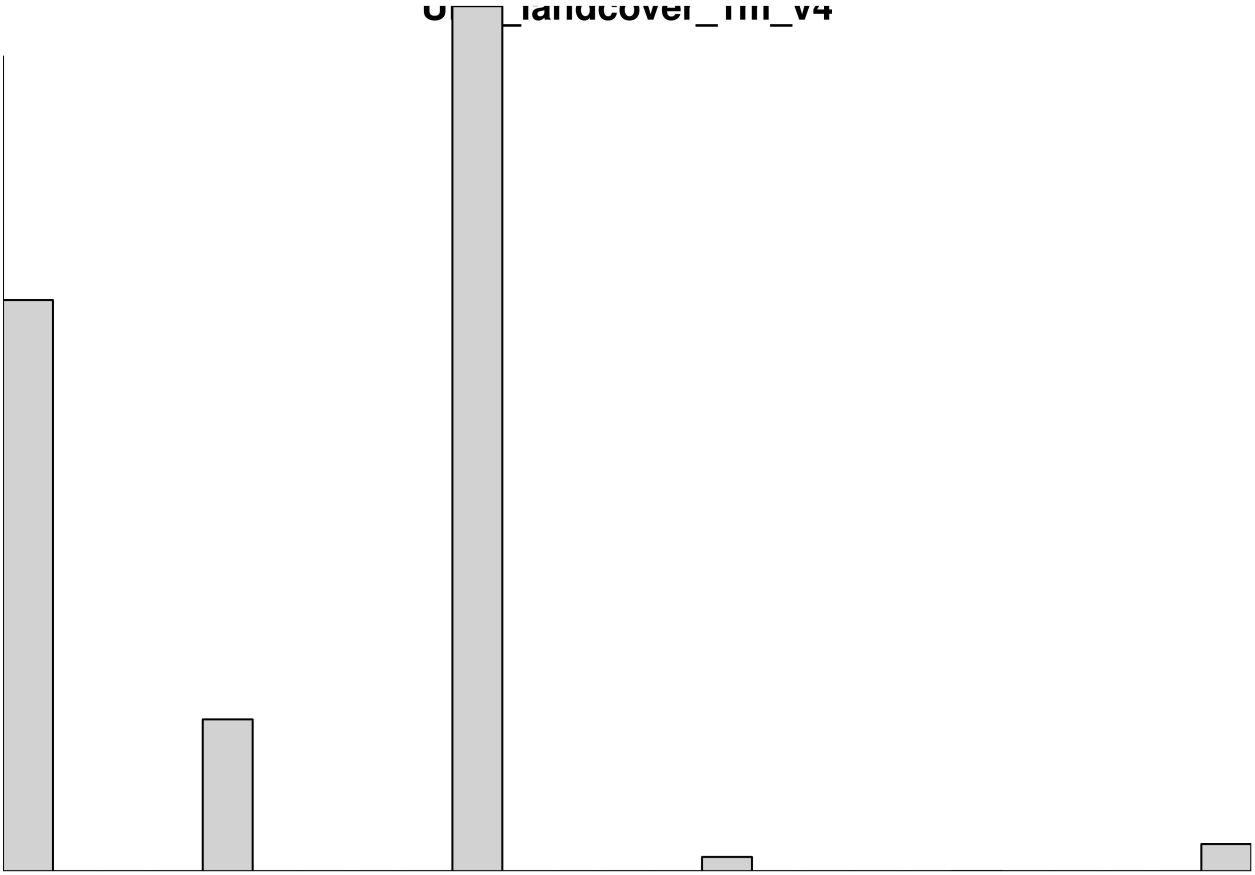
```
16.8 Kb
```

```
## what do these data look like?
plot(landcover, main = "Landcover")
```



```
## we can also look at the underlying distributions of values
## using a number of simple exploratory data analyses approaches.
## for example what is the distribution of cover classes?
hist(landcover)
```

```
Warning in .hist1(x, maxpixels = maxpixels, main = main, plot = plot, ...): 2%
of the raster cells were used. 100000 values used.
```



```
## clear the environment so we can focus on the variables we still need.
rm(landcover_uri, landcover_path, os_lc)
```

The code below requires two additional packages to run, so just look up to the screen and you can see the results. Note that a very important part of acquiring other folks data is understanding exactly what they collected, and what is represented in it. It rarely totally lines up with what you want and you may need to consider how to think about your results through the lens the products provide. In this raster, categorical data are represented as integers, and for us to determine what they represent we need to access the metadata associated with the raster.

```
landcover_metadata <- xml2::read_xml(landcover_md_uri, "text")
landcover_metadata <- XML::xmlParse(landcover_metadata)
print(landcover_metadata)

rm(landcover_metadata, landcover_md_uri)
```

I have copied the look-up table from the metadata file below.

Look up table for the raster:

- 1 = “needle-leaf trees and shrubs”
- 2 = “deciduous trees and shrubs”
- 3 = “deciduous meadow and subshrub”
- 4 = “bare rock, soil, gravel and asphalt”
- 5 = “water”
- 6 = “snow”
- 7 = “buildings”

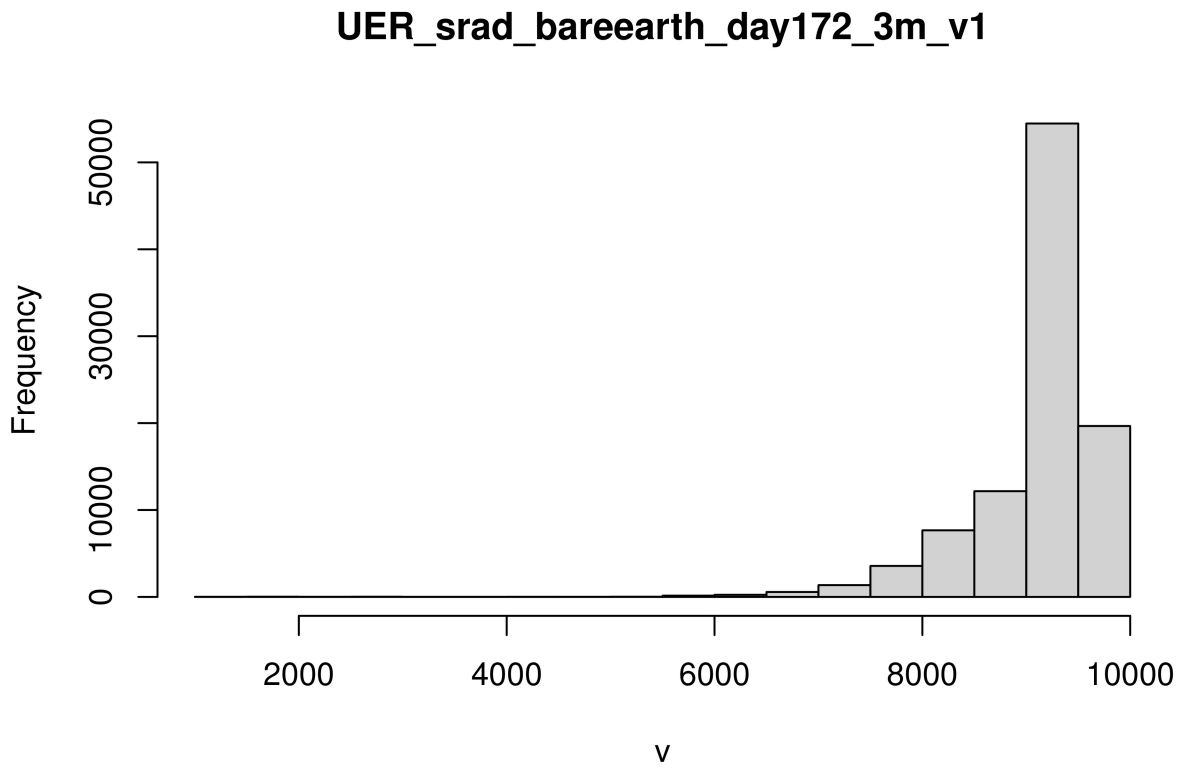
We can now repeat the steps to load the solar radiation predictor into R.

```
solar_radiation_path <- paste0("/vsicurl/", solar_radiation_uri)
# solar_radiation <- raster(solar_radiation_path, progress='text')

## if you cannot access this product via the web it is also in your folder.
solar_radiation <- raster('spatial_lab_data/solar_radiation.gri')

solar_radiation <- crop(solar_radiation, extent(sites))

os_sr <- object.size(solar_radiation)
print(os_sr, units = "auto")
```

```
## 14.4 Kb
```

```
# remember 1000 Mb in a Gb
hist(solar_radiation)
```

```
## Warning in .hist1(x, maxpixels = maxpixels, main = main, plot = plot, ...): 16%
## of the raster cells were used. 100000 values used.
```



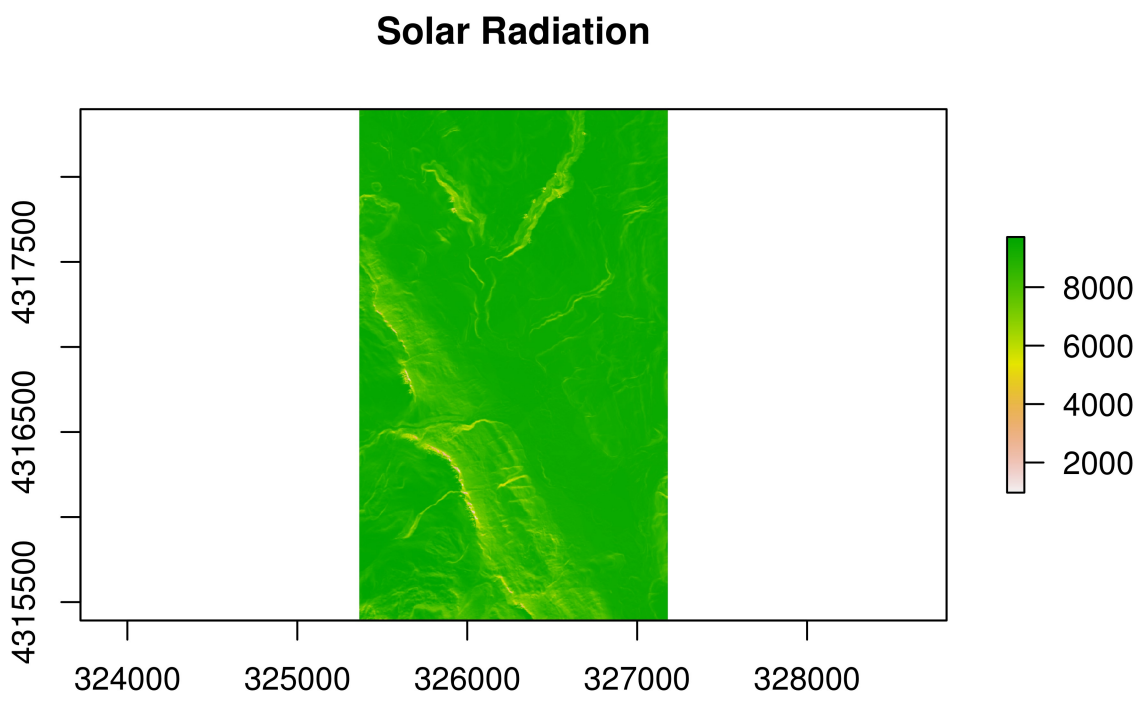

```
## now that we have a continuous variable let's use summary on it.
summary(solar_radiation)
```

```
## Warning in .local(object, ...): summary is an estimate based on a sample of 1e+05 cells (16.5% of al
```

```
##         UER_srad_bareearth_day172_3m_v1
## Min.                            969.513
## 1st Qu.                        8973.962
## Median                         9348.508
```

```
## 3rd Qu.                          9470.703
## Max.                             9716.818
## NA's                                0.000
```

```
plot(solar_radiation, main = "Solar Radiation")
```




```
rm(solar_radiation_uri, solar_radiation_path, os_sr)
```

Solar Radiation: "...This data set represents potential clear-sky incident solar radiation (in w/m^2) for day of year 172 (summer solstice)..."

Create a Raster Stack

Prior to class I also downloaded some other data products from this website. These are saved in your 'spatial_lab_data' folder. Four of these raster layers have the same extent, resolution, and we can form a rasterstack with them.

We have rasters of two different extents: The East River Valley, and The Upper Gunnison Basin. All of these products are for the Upper Gunnison Basin.

Please procure the file names, and paths to these files using the systems we used above for retrieving these information. Note that the pattern which these files end with is "gri"

```
files <- list.files(path = directories, pattern = "gri$")
files <- paste0(directories, "/", files)
print(files)
```

```
[1] "./spatial_lab_data/Aspect_southness.gri"
[2] "./spatial_lab_data/Aspect_westness.gri"
```

```
[3] "./spatial_lab_data/DEM.gri"
[4] "./spatial_lab_data/landcover.gri"
[5] "./spatial_lab_data/Slope_angle.gri"
[6] "./spatial_lab_data/Solar_radiation.gri"
[7] "./spatial_lab_data/Suitable_Bee_Habitat.gri"
```

```
files <- files[which(!stringr::str_detect(files, "landcover|Solar|Bee"))]
print(files)
```

```
[1] "./spatial_lab_data/Aspect_southness.gri"
[2] "./spatial_lab_data/Aspect_westness.gri"
[3] "./spatial_lab_data/DEM.gri"
[4] "./spatial_lab_data/Slope_angle.gri"
```

```
rasters_UG <- stack(files)
rasters_UG
```

```
class      : RasterStack
dimensions : 3004, 1813, 5446252, 4  (nrow, ncol, ncell, nlayers)
resolution : 1, 1  (x, y)
extent     : 325366, 327179, 4315393, 4318397  (xmin, xmax, ymin, ymax)
crs        : +proj=utm +zone=13 +datum=WGS84 +units=m +no_defs
names      : UG_dem_aspect_southness_1m_v1, UG_dem_aspect_westness_1m_v1, UG_dem_1m_v1, UG_dem_slope_1m
min values :                        -1.00,                       -1.00,      2918.62,              0
max values :                      1.00000,                     1.00000,   3340.43994,          84.77
```

```
rm(files)
```

"*Topographic Aspect Southness Metadata:* "North-facing aspects have a value of -1 and south-facing aspects have a value of 1. East and west-facing aspects have a value of 0."

"*Topographic Aspect Westness Metadata:* "east-facing aspects have a value of -1 and west-facing aspects have a value of 1. North and south-facing aspects both have a value of 0."

Digital Elevation Model (DEM) Recorded in meters.

Slope Angle Metadata: "... This is a 1 m resolution map of topographic slope (measured in degrees) computed using a 3*3 pixel kernel and Horn's formula...."

Remember earlier I mentioned that we are looking at the early stage of the spatial releases for this field station? Can you recognize how these products here are related to each other?

Extract Values from a Raster to a Vector

In general when we are performing spatial operations, we want to extract values from rasters which we can use as independent (predictor) variables for our dependent variables of interested. Hence we need to be able to extract values from rasters to our data.

Basically all we are doing is overlaying our points on a piece of paper and writing the values from the paper to our points. This is very easy, we just need to make a few decisions ahead of time of how to deal with funny problems, like when a point lands on the edge of two raster cells/pixels - which value should we add to our point then? Because we know that rasters have discrete numeric values (even when representing continuous processes), should we write that number to our point directly, or should we consider the raster cells which border the cell our point fell on too? If we choose to aggregate the values held in our raster cells to the point we are extracting them too how should we do it? Should we calculate the Mean? find the median? The max value, or the minimum?

Well... many of these things can come down to your study, but are still straightforward with just a couple arguments to the extract function in the Raster package.

Note this section has the code for running this process in parallel, if you are interested ask about it.

```
#mycores <- parallel::detectCores()
#snow::beginCluster(n = mycores)

UG_vals <- raster::extract(x = rasters_UG, # from raster
                           y = sites, # to vector
                  method = 'bilinear',
# create a 'smoothed' interpolation based on the four nearest raster cells
# only for continuous data without classes, else use method = 'simple'
# and a different fun(ction).
                  buffer = 0.25,
# we buffer each 1m^2 plot by 0.5m diameter
                  fun = mean
# the values from all cells intersected by
# our plots will be averaged
                  )
#snow::endCluster()
#rm(mycores)

## now try this for the remaining two rasters !

solar_radiation_vals <- raster::extract(solar_radiation, sites, method = 'bilinear',
                                buffer = 0.25, fun = mean, df = T)
solar_radiation_vals <- solar_radiation_vals[,2]

landcover_vals <- raster::extract(landcover, sites, method = 'simple',
                          buffer = 0.25, fun = min, df = T)
landcover_vals <- landcover_vals[,2]

extracted_vals <- data.frame(cbind(UG_vals, landcover_vals, solar_radiation_vals))

sites <- cbind(sites, extracted_vals)

rm(UG_vals, landcover_vals, solar_radiation_vals)
```

Rasters are usually fully of integers which oftentimes represent categorical data.

In our lecture I mentioned some of the great byte saving qualities of integers. Now that we have extracted these values from a raster We will convert these data to their under laying categorical data. We will use the 'case_when' function from the tidyverse. It works a nested if_else statement, BUT is much more readable, and simple to construct. We could also update these values to the raster directly, but modifying the extracted values is more sensible here.

```
sites <- sites %>%
  mutate(LNDCVR_NAME = case_when(
    landcover_vals == 1 ~ "needle-leaf trees and shrubs",
    landcover_vals == 2 ~ "deciduous trees and shrubs",
    landcover_vals == 3 ~ "deciduous meadow and subshrub",
    landcover_vals == 4 ~ "bare rock, soil, gravel and asphalt",
    landcover_vals == 5 ~ "water",
    landcover_vals == 6 ~ "snow",
    landcover_vals == 7 ~ "buildings"
  )
) %>%
  mutate(LNDCVR_NAME = str_replace_all(LNDCVR_NAME, " ", "_"))
```

It is good practice to write out the attributes of our sites to a shapefile so we do not need to run these extractions again. Running the extractions took both time and resources, even a minor fee was charged to the Field Station for Amazon to fulfill our requests for rasters.

```
sites <- sites %>%
  rename_with(~str_remove(., "UG_dem_aspect_|UG_dem_"))
st_write(sites, paste0(directories, '/Study_Sites_w_Pred_val.gpkg'), append = F)
```

```
## Deleting layer `Study_Sites_w_Pred_val' using driver `GPKG'
## Writing layer `Study_Sites_w_Pred_val' to data source
##   `./spatial_lab_data/Study_Sites_w_Pred_val.gpkg' using driver `GPKG'
## Writing 101 features with 19 fields and geometry type Polygon.
```

Does bee habitat and the general habitat of RMBL differ?

Is there actually a difference between the areas which the bees are sampled at and the areas which they were not sampled at? We can analyze each of our attributes as a pair.

```
path_random_pts <- paste0(directories, "/",
       list.files(directories, pattern = "Random_Points_w_Pred_val.gpkg"))
values <- paste0(directories, "/",
       list.files(directories, pattern = "Study_Sites_w_Pred_val.gpkg"))

random_pts <- st_read(path_random_pts) %>%
  st_drop_geometry() %>%
  dplyr::select(southness_1m_v1:solar_radiation_vals1) %>%
  mutate(GRP = 'RANDOM') %>%
  rename(solar_radiation_vals = solar_radiation_vals1, landcover_vals = landcover_vals1)
```

```
## Reading layer `Random_Points_w_Pred_val' from data source
##   `C:\R_datsci_2022\Spatial_lecture_developement\spatial_lab_data\Random_Points_w_Pred_val.gpkg'
##   using driver `GPKG'
## Simple feature collection with 100 features and 7 fields
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: 325396 ymin: 4315436 xmax: 327156 ymax: 4318381
## Projected CRS: WGS 84 / UTM zone 13N
```

```
values <- st_read(values, quiet = T) %>%
  dplyr::select(southness_1m_v1:solar_radiation_vals) %>%
  st_drop_geometry() %>%
  mutate(GRP = 'TARGET')

multi_plot <- rbind(random_pts, values) %>%
  mutate(GRP = as_factor(GRP)) %>%
  rename_with(~str_remove(., "_1m_v1|_vals")) %>%
  rename(elevation = X1m_v1 )

multi_plot %>%
  ggparcoord(
    columns = 1:6, groupColumn = 7, order = "anyClass",
    scale="uniminmax",
    showPoints = TRUE,
    title = "Distribution of values at current study sites and across the Study Area",
```

```
    alphaLines = 1
    ) +
  scale_color_manual(values=c("#DCDCDC", "#00bbb7") ) +  #"#69b3a2"
  theme_classic()+
  theme(
    plot.title = element_text(hjust = 0.5),
    legend.position="Default"
  ) +
  xlab("") +
  ylab("Rescaled Values")
```

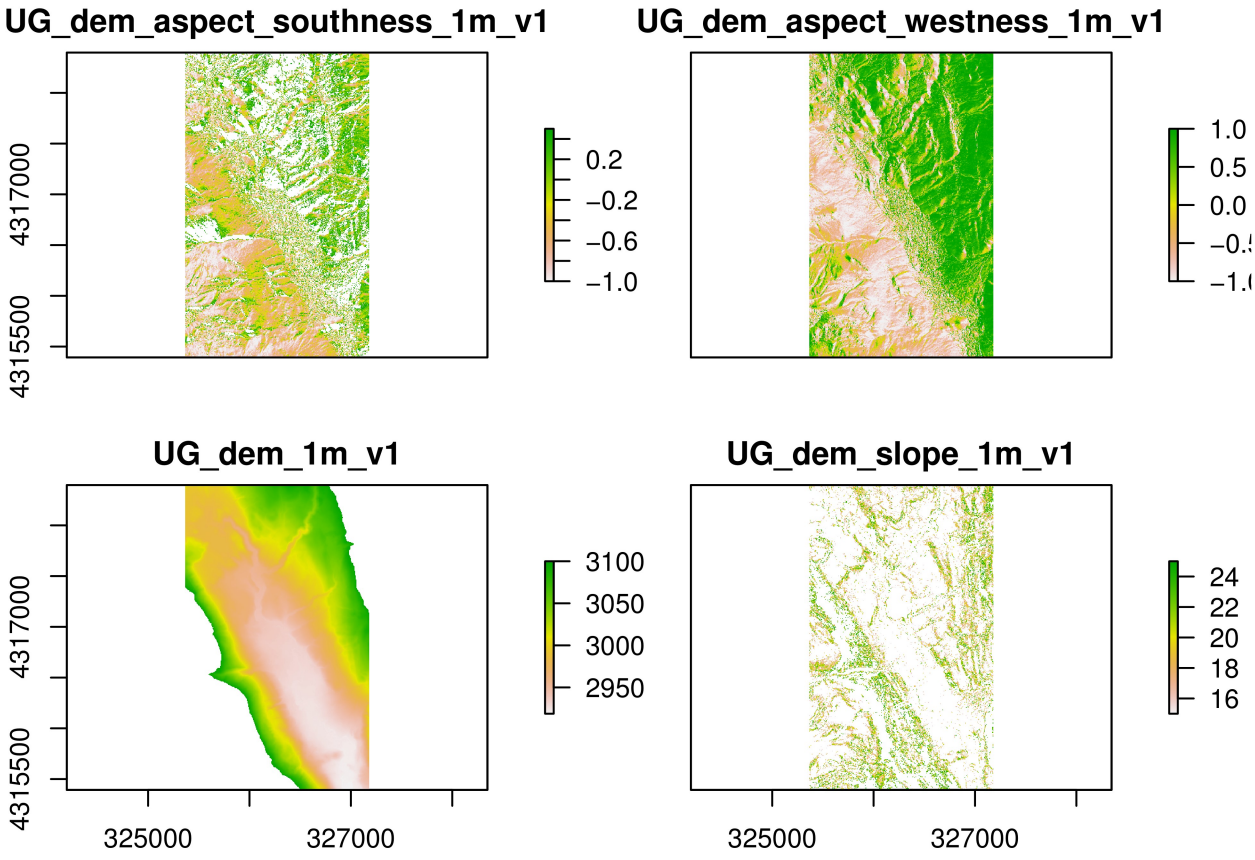

```
rm(random_pts, values, multi_plot, path_random_pts)
```

Mask Rasters to find target sites

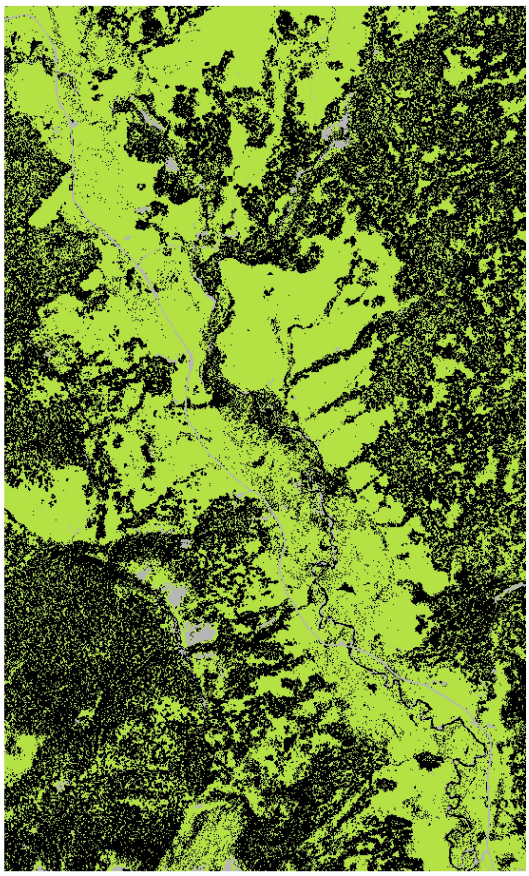
Just as we can filter a dataframe or vector, we can use logical tests on a raster. The core results of our lab will be based on subsetting a raster using logical test, and setting the remaining values in the raster as NA. By turning non target cells into NA, we will effectively be manipulating our raster to show us only the areas or interest.

```
landcover[landcover %in% c(1,2,4,5)] <- NA

solar_radiation[solar_radiation < 8950] <- NA
solar_radiation[solar_radiation > 9370] <- NA

rasters_UG@layers[[1]][rasters_UG@layers[[1]] > 0.5] <- NA
```

```
rasters_UG@layers[[3]][rasters_UG@layers[[3]] < 2900] <- NA
rasters_UG@layers[[3]][rasters_UG@layers[[3]] > 3100] <- NA

rasters_UG@layers[[4]][rasters_UG@layers[[4]] < 15] <- NA
rasters_UG@layers[[4]][rasters_UG@layers[[4]] > 25] <- NA

plot(rasters_UG)
```




```
plot(landcover)
```



Now that we have identified the cells which have values of interest for our variables in each of our six raster layers, we can 'overlay' these areas. Here we will use an approach of extracting the raster values to create dataframes, and then join each suitable cell in the raster grids using their coordinates.

This is not an approach you would use for a large dataset!

Note we will be using inner join here. This type of join only returns records which are present in both data sets. Note this join works, because we require that the geographic area must be considered a target area in *each* of the layers present.

```
sr_df <- as.data.frame(solar_radiation, xy = T)
sr_df <- sr_df[complete.cases(sr_df),]

lc_df <- as.data.frame(landcover, xy = T)
lc_df <- lc_df[complete.cases(lc_df),]

upper_gunni <- as.data.frame(rasters_UG, xy = T)
upper_gunni2 <- upper_gunni[complete.cases(upper_gunni),]

sr_lc <- inner_join(sr_df, lc_df, by = c('x' = 'x', 'y' = 'y'))
upper_gunni2 <- inner_join(sr_lc, upper_gunni2, by = c('x' = 'x', 'y' = 'y'))

rm(sr_df, lc_df, upper_gunni, sr_lc, rasters_UG, solar_radiation)
```

It is not uncommon in geospatial applications that our products and models are saved as predictions in space. While we have done some work utilizing tabular formats of our rasters to develop our predictions, we now need to store our predictions in a geospatial format.

Create Prediction Raster

We can actually create a raster of the same *extent*, *cell size* & number, and *crs* as one of our 1m resolution input rasters.

```
prediction <- raster(landcover)
```

Similar to how we joined the target cells from each raster layer, we will now use a left join to ensure that our target cells find the appropriate cell in the new empty raster so they we can populate it with them.

First we will pull out a set of the coordinates of each cell.

```
prediction_vals <- as.data.frame(prediction, xy = T)
```

When we use a left join, we will keep every cell in our new raster (as represented by the data frame), and when relevant add cells which match our features of interest to it (as also represented by a data frame)

```
prediction_vals <- left_join(prediction_vals, upper_gunni2,
                             by = c('x' = 'x', 'y' = 'y')) %>%
  dplyr::select(-layer) %>%
  mutate(SUITABILITY =
           if_any(.cols = everything(), ~ is.na(.x))) %>%
# identify cells without values from our earlier rasters
  mutate(SUITABILITY =
           if_else(SUITABILITY == T, 0, 1)) %>%
# make the cells we did not join '0' indicating they are not target locations
  dplyr::select(SUITABILITY) %>%
  pull(SUITABILITY)
```

We can now fill our empty raster with values indicating whether the cell is target habitat or not. Please note you will see many examples of people myself included

```
prediction_raster <- setValues(prediction, prediction_vals)

rm(prediction_vals, prediction)
```

While we created a raster above, in this use case, it is also valuable to have a vector copy of our results around. We will go over vector functionality in our final lecture this quarter, and make great use of vector data in our homework assignment. Here we will convert our predictions, which are saved in the raster format, to a vector format.

Make a Vector copy of our predictions

The raster and vector data model are both representing features on the planet, and in certain instances they can be *both intelligently* be used to represent certain features. In fact, in our case it behooves us to create a vector copy of our data so that we may run certain types of statistics on it, and to subset it for sampling.

To map our predictions we will create a copy of our raster where we have masked all of the non-suitable habitat values with NA, so that they will be transparent. We will also use this copy of our predictions raster to create our vector data.

```
prediction_raster_2plot <- prediction_raster
prediction_raster_2plot[prediction_raster_2plot == 0] <- NA
```

We can create a vector copy of it using the code below: notice we will need to pass through an intermediate SP object to create a simple feature.

```
suitable_sites.sp <- rasterToPolygons(prediction_raster_2plot)
suitable_sites.sf <- st_as_sf(suitable_sites.sp, crs = 32613)
```

We should inspect the new object to make sure that the transformation form a raster to a spatial polygon, to a simple feature worked. While this is good to do, we will not run this activity in class because it is time intensive. The code to do so is below.

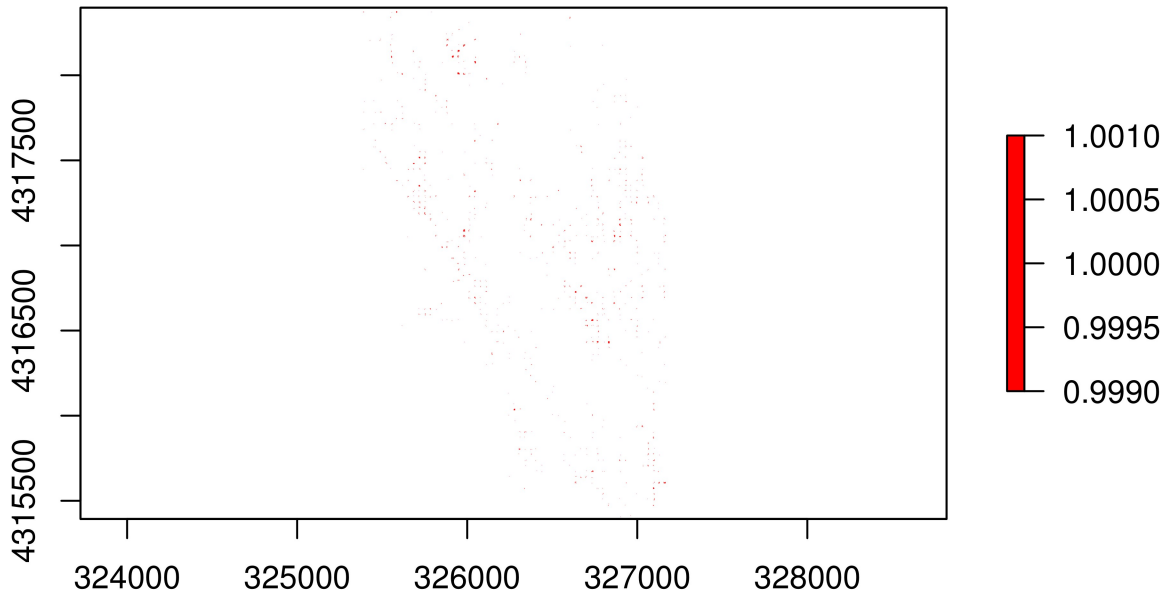
```
is_valid <- st_is_valid(suitable_sites.sf)
which(is_valid == F) # hopefully nothing pops up!

## and now we will make a quick map to QC our work.
ggplot(suitable_sites.sf) +
  geom_sf()
```

```
rm(suitable_sites.sp, is_valid)
```

Ok so naturally at some point we will need to communicate our results to others. What the plot is to the statistician, the map is to the geospatial analyst (... well the plot too but not in exclusivity!). Remember maps in this context are meant to communicate information, and should not serve as a distraction from the results. You all will have some pretty fancy code for making publication quality maps in your notes from your next lecture, but here we will make some quicker maps.

Take a quick look at your results again to determine how we may need to go about emphasizing certain features.

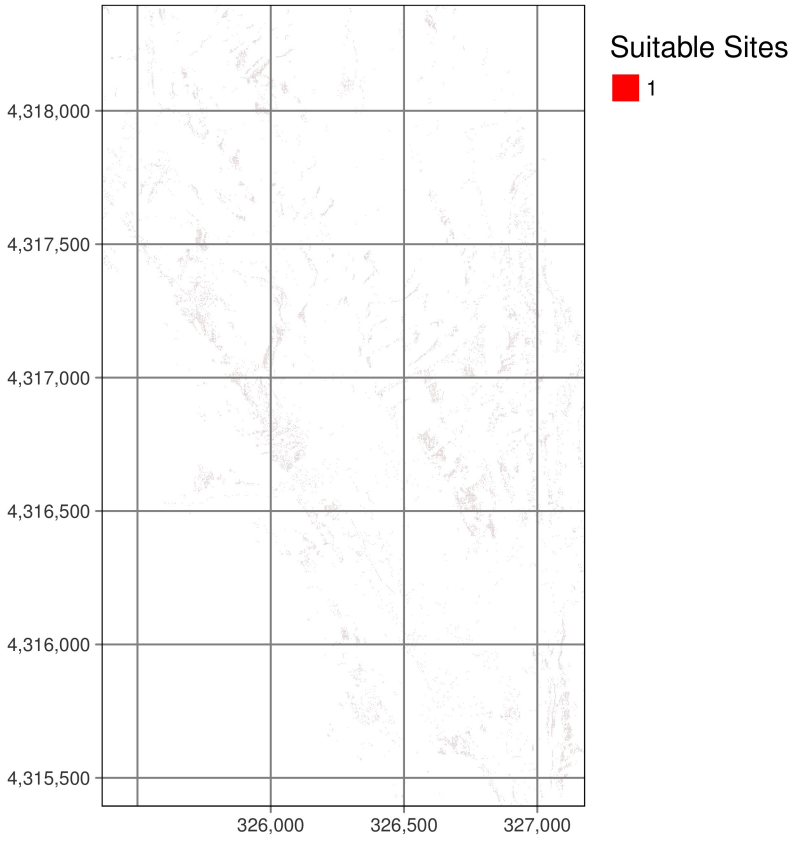
```
plot(prediction_raster_2plot, col = "red")
```



Obviously that plot function is missing a few things, we can make more elaborate maps of our the prediction using tmap. which More of less follows the grammar of graphics.

```
raster_map <- tmap::tm_shape(prediction_raster_2plot, raster.downsample = F) +
   tmap::tm_raster(palette = "red", title = "Suitable Sites") +
```

```
   tmap::tm_legend(outside = TRUE) +
   tmap::tm_grid(labels.inside.frame = FALSE,
          n.x = 3, n.y = 5)
raster_map
```




```
tmap_save(raster_map, "Bee_habitat_pred_vector.png", height = 8, units = "cm")
```

```
Map saved to C:\R_datsci_2022\Spatial_lecture_developement\Bee_habitat_pred_vector.png
```

```
Resolution: 570.2633 by 944.8819 pixels
```

```
Size: 1.900878 by 3.149606 inches (300 dpi)
```

```
rm(raster_map, prediction_raster_2plot)
```

Look at those tiny flecks, seems like such a small area? how do you want to quickly check how much area we have? each raster cell is a meter.

```
cellStats(prediction_raster, sum)
```

```
[1] 16273
```

Mao our results to share them

Tmap is great for making rasters look nice, but to be honest vector data are better for mapping. We are going to switch over to ggmap and map our vector data real quick.

Before, when we made our exploratory maps, it was glaringly obvious we needed a basemap to provide some context. Now it is worth grabbing one. We will also download this from online. We will use the 'ggmap'

package to do this. To request basemaps from ggmps we need to provide it a an *extent*, tell it a zoom level ~ which is the *resolution* of the basemap and define the *thematic content* of our map.

GGmap is mostly great, but it has some problems. The first one is it uses a CRS called 'Web Mercator' it is what google maps made oh so popular, and to be frank is now probably the default map projection people think about. However, more competent geospatial analysts than myself assure me Web Mercator, is an abomination of all they hold dear - in essence it does not work very well - and at least worse than any alternative.

To work with ggmap we will do a few steps, such as buffer our points before requesting our map extent (if we do not our points will be noticeably OFF the basemap).

Note google maps also use to provide there map themes for free through ggmap, and they more or less still do, but this requires an account which we will not set up. We will instead download a map generated by Stamen which is a really cool open source map making group based out of the San Francisco Bay.

```
site_bbox <- sites %>%
  st_buffer(250) %>%
  st_transform(4326) %>%
  st_bbox(sites)
names(site_bbox) <- c('left', 'bottom', 'right', 'top')

basemap <- ggmap::get_stamenmap(site_bbox, zoom = 14, maptype = "terrain", messaging = F)
```

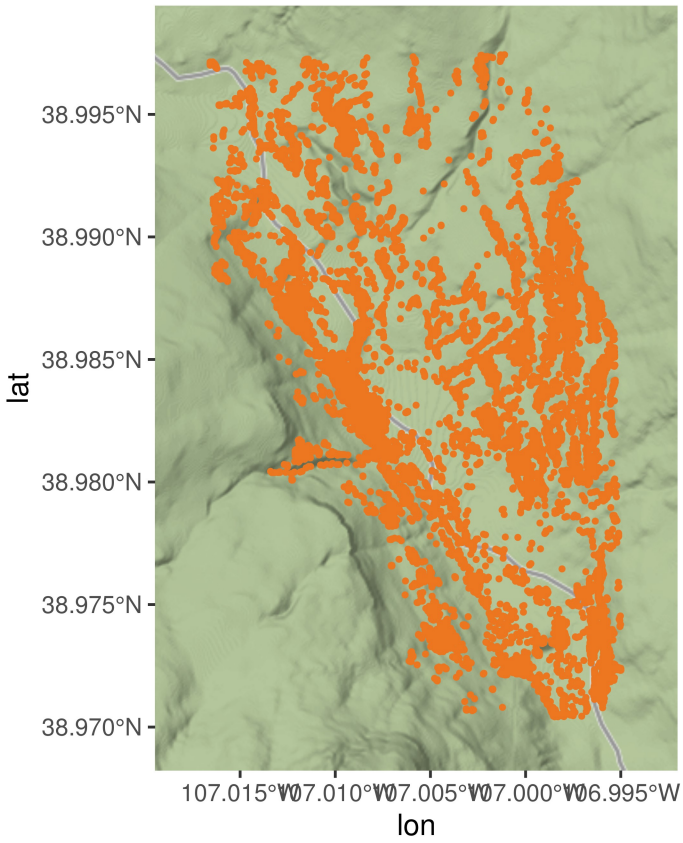
```
Source : http://tile.stamen.com/terrain/14/3321/6261.png

Source : http://tile.stamen.com/terrain/14/3322/6261.png

Source : http://tile.stamen.com/terrain/14/3321/6262.png

Source : http://tile.stamen.com/terrain/14/3322/6262.png

Source : http://tile.stamen.com/terrain/14/3321/6263.png

Source : http://tile.stamen.com/terrain/14/3322/6263.png
```

Now finally we can plot the map below using ggmap.

```
vector_map <- ggmap(basemap) +
  geom_sf(data = suitable_sites.sf,
          alpha = 0.5,
          fill = "chocolate2",
          color = "chocolate2",
          show.legend = "line",
          size = 1.2,
          inherit.aes = FALSE) +
  coord_sf(crs = st_crs(4326))
```

```
## Coordinate system already present. Adding new coordinate system, which will replace the existing one
```

```
vector_map
```




```
ggsave(filename="Bee_habitat_pred_vector.png", plot=vector_map,height=8, units="cm")
```

```
## Saving 16.5 x 8 cm image
```

```
rm(upper_gunni2, vector_map, basemap, site_bbox)
```

We can see that ggmaps is not only based on the *g*rammar of *g*raphics, but **gg*plot itself.

Writing Geo-spatial Data

It is good practice to write our the results of geospatial analysis to disk. While this lab did not take long, some folks regularly run analyses which take days (..weeks... months...).

```
st_write(suitable_sites.sf, "spatial_lab_data/Suitable_Bee_Habitat.shp",
         quiet = TRUE,
         append = F)

rm(suitable_sites.sf)
```

The raster package has a wide range of formats it can save data in. In my research I pretty much always deal with: .gri, gtiff, ascii, but I suspect some of these other types may be more common for earth and planetary science related things (please speak up if this true and indicate which if possible!).

```
raster::writeFormats()
```

```
      name        long_name
 [1,] "raster"    "R-raster"
 [2,] "SAGA"      "SAGA GIS"
 [3,] "IDRISI"    "IDRISI"
```

```
 [4,] "IDRISIold" "IDRISI (img/doc)"
 [5,] "BIL"       "Band by Line"
 [6,] "BSQ"       "Band Sequential"
 [7,] "BIP"       "Band by Pixel"
 [8,] "ascii"     "Arc ASCII"
 [9,] "CDF"       "NetCDF"
[10,] "ADRG"      "ARC Digitized Raster Graphics"
[11,] "BAG"       "Bathymetry Attributed Grid"
[12,] "BMP"       "MS Windows Device Independent Bitmap"
[13,] "BT"        "VTP .bt (Binary Terrain) 1.3 Format"
[14,] "BYN"       "Natural Resources Canada's Geoid"
[15,] "CTable2"   "CTable2 Datum Grid Shift"
[16,] "EHdr"      "ESRI .hdr Labelled"
[17,] "ELAS"      "ELAS"
[18,] "ENVI"      "ENVI .hdr Labelled"
[19,] "ERS"       "ERMapper .ers Labelled"
[20,] "FITS"      "Flexible Image Transport System"
[21,] "GPKG"      "GeoPackage"
[22,] "GS7BG"     "Golden Software 7 Binary Grid (.grd)"
[23,] "GSBG"      "Golden Software Binary Grid (.grd)"
[24,] "GTiff"     "GeoTIFF"
[25,] "GTX"       "NOAA Vertical Datum .GTX"
[26,] "HDF4Image" "HDF4 Dataset"
[27,] "HFA"       "Erdas Imagine Images (.img)"
[28,] "IDA"       "Image Data and Analysis"
[29,] "ILWIS"     "ILWIS Raster Map"
[30,] "INGR"      "Intergraph Raster"
[31,] "ISCE"      "ISCE raster"
[32,] "ISIS2"     "USGS Astrogeology ISIS cube (Version 2)"
[33,] "ISIS3"     "USGS Astrogeology ISIS cube (Version 3)"
[34,] "KRO"       "KOLOR Raw"
[35,] "LAN"       "Erdas .LAN/.GIS"
[36,] "Leveller"  "Leveller heightfield"
[37,] "MBTiles"   "MBTiles"
[38,] "MRF"       "Meta Raster Format"
[39,] "netCDF"    "Network Common Data Format"
[40,] "NGW"       "NextGIS Web"
[41,] "NITF"      "National Imagery Transmission Format"
[42,] "NTv2"      "NTv2 Datum Grid Shift"
[43,] "NWT_GRD"   "Northwood Numeric Grid Format .grd/.tab"
[44,] "PAux"      "PCI .aux Labelled"
[45,] "PCIDSK"    "PCIDSK Database File"
[46,] "PCRaster"  "PCRaster Raster File"
[47,] "PDF"       "Geospatial PDF"
[48,] "PDS4"      "NASA Planetary Data System 4"
[49,] "PNM"       "Portable Pixmap Format (netpbm)"
[50,] "RMF"       "Raster Matrix Format"
[51,] "ROI_PAC"   "ROI_PAC raster"
[52,] "RRASTER"   "R Raster"
[53,] "RST"       "Idrisi Raster A.1"
[54,] "SAGA"      "SAGA GIS Binary Grid (.sdat, .sg-grd-z)"
[55,] "SGI"       "SGI Image File Format 1.0"
[56,] "Terragen"  "Terragen heightfield"
[57,] "VICAR"     "MIPL VICAR file"
```

```
writeRaster(prediction_raster, filename = "spatial_lab_data/Suitable_Bee_Habitat",  overwrite=TRUE)
```

```
class      : RasterLayer
dimensions : 3004, 1813, 5446252  (nrow, ncol, ncell)
resolution : 1, 1  (x, y)
extent     : 325366, 327179, 4315393, 4318397  (xmin, xmax, ymin, ymax)
crs        : +proj=utm +zone=13 +datum=WGS84 +units=m +no_defs
source     : Suitable_Bee_Habitat.grd
names      : layer
values     : 0, 1  (min, max)
```

```
rm(sites, directories, landcover, extracted_vals, prediction_raster)
```

LAB ENDS HERE! LAB ENDS HERE! LAB ENDS HERE!

In the sake of reproducibility, the code below were some steps that were taken to develop the materials for this activity.

I created vector files of each plot via the following steps:

```
data_dir <- "Digitize_plots/"
files <- list.files(data_dir)
all_files <- paste0(data_dir, files)

raw <- all_files %>%
  map_dfr(readxl::read_xlsx) %>%
  mutate(across(c('Other','Other2'), ~str_to_lower(.))) %>%
  dplyr::select('Other', 'Other2', 'POINT_X', 'POINT_Y') %>%
  arrange(Other, Other2) %>%
  filter(!str_detect(Other2, 'cen')) %>%
  rename(Site = Other,
         Plot = Other2)

clean <- raw %>%
  group_by(Site, Plot) %>%
  add_count() %>%
  filter(n == 4)%>%
  slice(1) %>%
  dplyr::select(-n) %>%
  bind_rows(., raw) %>%
  group_by(Site, Plot) %>%
  add_count() %>%
  filter(n == 5)

points <- clean %>%
  st_as_sf(coords = c(x = 'POINT_X', y = 'POINT_Y'), crs = 32613, remove = F) %>%
  group_by(Site, Plot) %>%
  mutate(group_id = cur_group_id())

list_poly = st_sf(
  aggregate(points,
    by=list(ID=points$group_id),
#    do_union=FALSE,
    FUN=function(vals){vals[1]}))
```

```
polygons = st_cast(list_poly, 'POLYGON') %>%
  dplyr::select(-ID, -n) %>%
  rename_with(str_to_upper, 1:5) %>%
  dplyr::select(GROUP_ID, SITE, PLOT,
    LONG_UTM = POINT_X,
    LAT_UTM = POINT_Y) %>%
  st_make_valid() %>%
  st_convex_hull()

coordinates <- polygons %>%
  st_transform(4326) %>%
  st_centroid() %>%
  mutate(LONG_WGS = unlist(map(.$geometry,1)),
           LAT_WGS = unlist(map(.$geometry,2))) %>%
  st_drop_geometry() %>%
  dplyr::select(LONG_WGS, LAT_WGS)
polygons <- cbind(polygons, coordinates)

polygons %>%
  filter(SITE == 'rg') %>%
  ggplot() +
  geom_sf(aes(fill = PLOT))  +
  coord_sf(datum = st_crs(32613)) +
  labs(x = 'Easting', y = 'Northing')

#dir.create("spatial_lab_data")
# st_write(polygons, "spatial_lab_data/Emergence_traps.shp", quiet = TRUE, append = F)

rm(data_dir, files, all_files, raw, clean, points, list_poly, polygons, coordinates)
```

Metadata: "... This is a 1 m resolution map of topographic slope (measured in degrees) computed using a 3*3 pixel kernel and Horn's formula...."

Metadata: "North-facing aspects have a value of -1 and south-facing aspects have a value of 1. East and west-facing aspects have a value of 0."

We can also reduce the space of our raster for some sub-setting operations.

First we will define the extent of the area which we are interested in creating our model in.

Let us import another vector file for this. It is in your data folder.

Extract values from 100 random points over study extent

```
rastext <- extent(landcover)
ras_samples <- as.data.frame(cbind(
  X = sample(c(rastext[1]:rastext[2]), size = 100),
  Y = sample(c(rastext[3]:rastext[4]), size = 100)
)) %>%
  mutate(ID = 1:nrow(.)) %>%
  st_as_sf(coords = c(x = 'X', y = 'Y'),
           crs = st_crs(landcover))

UG_vals1 <- terra::extract(x = rasters_UG, y = ras_samples, method = 'simple', fun = mean)
solar_radiation_vals1 <- terra::extract(solar_radiation, ras_samples, method = 'simple', fun = mean, df
```

```r
solar_radiation_vals1 <- solar_radiation_vals1[,2]
landcover_vals1 <- terra::extract(landcover, ras_samples, method = 'simple', fun = min, df = T)
landcover_vals1 <- landcover_vals1[,2]

extracted_vals1 <- data.frame(cbind(UG_vals1, landcover_vals1, solar_radiation_vals1))
ras_samples <- cbind(ras_samples, extracted_vals1) %>% st_as_sf()

ras_samples <- ras_samples %>%
  rename_with(~str_remove(., "UG_dem_aspect_|UG_dem_"))

st_write(ras_samples, 'spatial_lab_data/Random_Points_w_Pred_val.gpkg', append = F)

rm(rastext, UG_vals1, solar_radiation_vals1, landcover_vals1, extracted_vals1)
```