Ag Carpentry - Weather and Soil Data

Brittani

10/19/2019

Motivating Questions:

- What are the common file types in agricultural data?
- What publicly available datasets exist for my field?

Objectives with Agricultural Data Types

- Describe the contents of files generated during planting, fertilization, and harvest
- Describe the contents of files used to control seeding and fertilization rate
- Describe the format of public weather and soil datasets
- Import agricultural datasets into R
- Import weather data from the internet, eg. daymetr
- Access to elevation and ssurgo data with higher resolution
- Derive topography data from elevation data

Keypoints:

- sf is prefereable for data analysis; it is easier to access the dataframe
- Projecting your data in utm is necessary for many of the geometric operations you perform (e.g. making trial grids and splitting plots into subplot data)
- Compare different data formats, such as gpkg, shp(cpg,dbf,prj,sbn,sbx),geojson,tif

Bringing in the data from geospatial exercises

Daymet Weather Data

The Oak Ridge National Laboratory produces a datset called Daymet which contains predicted weather observations on a one meter grid. These data come from weather station climate observations in a climate model for prediction and include variables such as precipitation, snow water equivalent, temperature, day length, solar radians, and vapor pressure.

There is a package in r `daymetr` that downloads the daymet weather data within the R environment. For a single point, you can use the command `download_daymet()`. If you want to download the data for a set of points, there is also the command `download_daymet_batch()` which takes an argument a .csv of the points in lat/long.

We will use the mean latitude and longitude values from the bounding box as our point for the weather data. This should be a point near the middle of the field. We also call the site `Field1`, but this will be the name of a specific field if you use it in the future. We can choose the start and end years. If the data is not available for the year you request, an error will be reported. We choose 2000 to 2018 for this example; later we will use the historical data for comparison. The final option `internal = TRUE` means that the daymet data is brought into the R environment rather than saved in your working directory.

```
bbox <- as.matrix(st_bbox(boundary))
lon <- mean(bbox[c(1,3),])
lat <- mean(bbox[c(2,4),])
```

```
weather <- download_daymet(site = "Field1", lat = lat, lon = lon, start = 2000, end = 2018, internal = '
```

```
## Downloading DAYMET data for: Field1 at 40.7430279883331/-82.9757922217347 latitude/longitude !
```

```
## Done !
```

The object `weather` is a list of 7 objects, the last of is the data.

Exercise 1: Explore the weather data

1. Grab this object and save it as `weather_data`.
2. How is the date reported?
3. What other variables exist?
4. What are the units for the different variables? *Remember:* Sometimes you need to use a search engine to understand what objects are created from a specific R function.

Exercise 1 Solutions

```
weather_data <- weather$data
summary(weather_data)
```

```
##       year           yday        dayl..s.      prcp..mm.day.
##  Min.   :2000   Min.   :  1   Min.   :32832   Min.   :  0.000
##  1st Qu.:2004   1st Qu.: 92   1st Qu.:35942   1st Qu.:  0.000
##  Median :2009   Median :183   Median :43200   Median :  0.000
##  Mean   :2009   Mean   :183   Mean   :43201   Mean   :  2.928
##  3rd Qu.:2014   3rd Qu.:274   3rd Qu.:50458   3rd Qu.:  3.000
##  Max.   :2018   Max.   :365   Max.   :53568   Max.   :132.000
##   srad..W.m.2.    swe..kg.m.2.     tmax..deg.c.     tmin..deg.c.
##  Min.   : 38.4   Min.   : 0.000   Min.   :-15.00   Min.   :-27.50
##  1st Qu.:227.2   1st Qu.: 0.000   1st Qu.:  6.50   1st Qu.: -2.50
##  Median :313.6   Median : 0.000   Median : 17.50   Median :  5.50
##  Mean   :316.2   Mean   : 4.882   Mean   : 15.86   Mean   :  4.95
##  3rd Qu.:412.8   3rd Qu.: 0.000   3rd Qu.: 26.00   3rd Qu.: 13.50
##  Max.   :563.2   Max.   :76.000   Max.   : 37.00   Max.   : 24.00
##     vp..Pa.
##  Min.   :  80
##  1st Qu.: 520
##  Median : 880
##  Mean   :1046
##  3rd Qu.:1560
##  Max.   :3000
```

The date is reported as the year and day of the year. Other variables include day length, precipitation, solar radiation, snow water equivalent, maximum temperature, minimum temperature, and vapor pressure. The units for the variables are given after the variable name. For example, day length is in seconds and solar radiation is in watts per square meter. While precipitation and temperature have intuitive names, vapor pressure and snow water equivalent are not so apparent. Use the `datmetr` vignette to understand the meaning of these variables.

https://cran.r-project.org/web/packages/daymetr/vignettes/daymetr-vignette.html

Unit Conversions

Publicly available data are usually given in metric units as we saw in the weather data above. We may want to have these data in imperial units as these are the units we often are comparing in the United States; additionally, you may know the value of crop requirements and threshholds in imperial units rather than metric units.

The package `measurements` in R converts observations from one unit to another. The command '`conv_unit()` converts the column from one stated unit to another unit. To see the possible units for a specific kind of measure, look at the `conv_unit_options` for the specific measure you are converting (e.g. length, area, weight, etc.).

If we want to convert the daily precipitation from milimeters to inches, we will first look at the unit options for length. Here we can see that inches are "inch" and milimeters are "mm" in the `measurements` framework. We then use `conv_unit()` with these arguments and our `prcp..mm.day.` column to create a new column called `prec`.

```
conv_unit_options$length
```

```
##  [1] "angstrom"   "nm"         "um"         "mm"         "cm"
##  [6] "dm"         "m"          "km"         "inch"       "ft"
## [11] "foot"       "feet"       "yd"         "yard"       "fathom"
## [16] "mi"         "mile"       "naut_mi"    "au"         "light_yr"
## [21] "light_year" "parsec"     "point"
```

```
weather_data$prec <- conv_unit(weather_data$prcp..mm.day., "mm", "inch")
```

Exercise 2: Unit Conversions

1. Look at the possible units for temperature in `measurements`.
2. Convert the two temperature variables into fahrenheit from celsius with the names `tmax` and `tmin`.

Execrise 2 Solutions

```
conv_unit_options$temperature
```

```
## [1] "C" "F" "K" "R"
```

```
weather_data$tmax <- conv_unit(weather_data$tmax..deg.c., "C", "F")
weather_data$tmin <- conv_unit(weather_data$tmin..deg.c., "C", "F")
head(weather_data$tmax)
```

```
## [1] 48.2 57.2 58.1 54.5 37.4 34.7
```

Dates in Dataframes

There is a class within R for dates. Once a column is of the `date` class, we can subset or order the dataset by time. `as.Date()` converts a column to a data, but here if we try the command `weather_data$date <- as.Date(weather_data$yday)`, we will receive an error saying an origin must be supplied.

The function can see that the date is in days after some starting time or origin. The name `yday` means this is the day of the year, so the origin should be the last day of the previous year. There are multiple years in our dataframe, so the origin should change for each year. This is accomplished by pasting `weather_data$year-1` and `"-12-31"` together using the function `paste0()` so that the origin is always the last day of the previous year.

```
weather_data$date <- as.Date(weather_data$yday, origin = paste0(weather_data$year-1, "-12-31"))
head(weather_data$date)
```

```
## [1] "2000-01-01" "2000-01-02" "2000-01-03" "2000-01-04" "2000-01-05"
## [6] "2000-01-06"
```

Precipitation Graph

Perhaps you want to see what the weather this year was like compared to the average historic weather for the same area. We will make a graph showing the total monthly precipitation from 2018 compared to the average precipitation from the years 2000 to 2017. This is a common way to look at seasonal rainfall and allows us to look at the rainfall during the critical months of July and August.

Currently, there is no month variable in our dataframe. There is a package called `lubridate` that can facilitate easy transformations of dates in R. We use the command `month()` to add a variable called `month` to the dataframe. The option `label = TRUE` creates a string with the month name instead of a number.

```
weather_data$month <- lubridate::month(weather_data$date, label = TRUE)
```

Now, we need to sum the daily precipitation for each year and month combination. There is a package called `dplyr` that helps with many kinds of data manipulation. A popular task is to perform an action over a group. To specify the grouping variables, you use `group_by()` then add the additional command `summarise()` which defines the action.

The next steps organize a dataframe for a graph of the average monthly precipitation from 2000 to 2017.

1. Create subset of `weather_data` taking out the 2018 observations.
2. Create a monthly precipitation variable for each year and month combination called `prec_month` using the `dplyr` commands.
3. Take an average of `prec_month` for each month.

Steps

1. Create subset of `weather_data` taking out the 2018 observations.

```
hist_data <- subset(weather_data,year != 2018)
```

2. Create a monthly precipitation for each year called `prec_month` using the `dplyr` commands.

```
by_month_year <- hist_data %>% dplyr::group_by(month, year) %>% dplyr::summarise(prec_month = sum(prec))
```

3. Take an average of `prec_month` for each month.

```
by_month <- by_month_year %>% dplyr::group_by(month) %>% dplyr::summarise(prec_avg = mean(prec_month))
```

Exercise 3: Using `dplyr`

1. Create a new dataframe called `weather_2018` from the `weather_data` using only 2018 observations.
2. Create a monthly precipitation variable for each month in 2018 called `prec_month` using the `dplyr` commands.

Exercise 3 Solutions

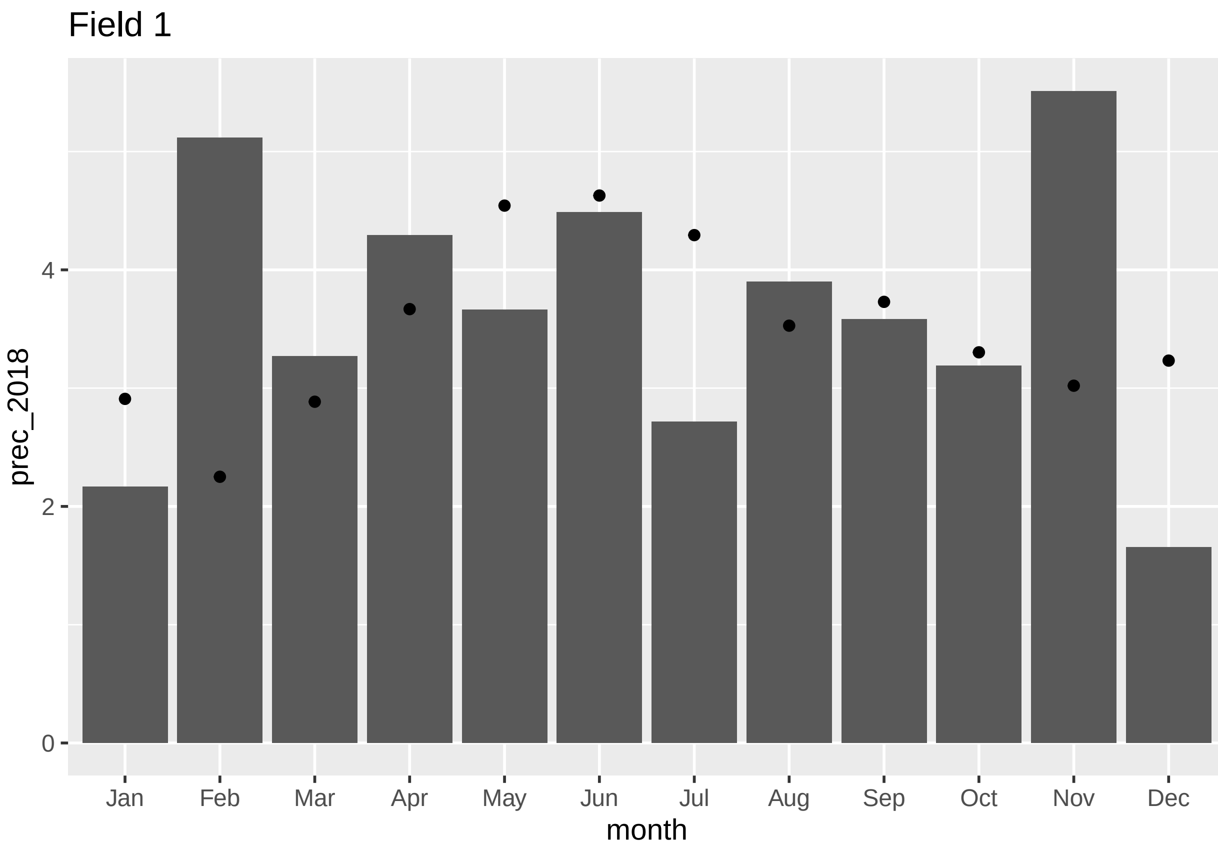
```
weather_2018 <- subset(weather_data, year==2018)
by_month_2018 <- weather_2018 %>% dplyr::group_by(month) %>% dplyr::summarise(prec_2018 = sum(prec))
```

We now have two separate dataframes `by_month_2018` and `by_month` with the rainfall for each month. We can use the common variable `month` to merge them into one dataframe with the average monthly rainfall and the 2018 monthly rainfall using the `merge()` function.

```
prec_plot <- merge(by_month,by_month_2018, by = "month")
```

We will now use `ggplot` to create a graph with a bar representing the monthly precipitation in 2018 and a point with the average rainfall from 2000 to 2017. In the function `geom_bar()` `stat = identity` creates a bar graph where the height of the bar is the value of the variable rather than the count of the observations, the common use of a bar chart.

```
monthly_prec <- ggplot(prec_plot) +
  geom_bar(aes(x = month, y = prec_2018), stat = 'identity')
monthly_prec + geom_point(aes(month, prec_avg), show.legend = TRUE) + ggtitle("Field 1")
```



The most notable feature of the weather graph is the below average rainfall in July, the most critical growing period for corn. To understand whether this affected yield on the field, we woud also need to look at historic yield. But on your field, you will know those historic average and be able to have a pretty clear idea of weather impacted the average yield in a growing season.

Another possible graph you could make with these data is on the accumulated GDD each month.

SSURGO Soil Data

The SSURGO data is probably a dataset you are familiar with already. You can obtain a soil description of your field on the Web Soil Survey website below. The SSURGO dataset has been developed over a century of surveying land and analyzing soil samples across the United States. While the website is one way to access the soil data, R also has a package called `FedData` that has a function `get_ssurgo()` for accessing the soil data in the R environment.

https://websoilsurvey.sc.egov.usda.gov/App/WebSoilSurvey.aspx

The next line brings the SSURGO data into the R environment with the name `ssurgo` and the object `boundary` from the geospatial lesson. Note here that the class of `boundary` needs to be `spatial` rather than `sf`, so we trasnform the object with `as(boundary,"Spatial")`.

```
bound.sp <- as(boundary, "Spatial")
ssurgo <- get_ssurgo(bound.sp, "field1")
```

The downloaded `ssurgo` is a list with 2 objects, `spatial` and `tabular`. The `spatial` object contains the polygons of soil types for the field, and `tabular` contains many dataframes with attributes collected for the soil and soil horizons. Note that these dataframes and their relationships with one another ar very complex. To use these data, you must carefully read the SSURGO documentation. Merging the dataframes to have one value of the attributes for each soil polygon requires reducing the dimension of the data, often by weighting the attributes by horizon depth. For an example of how this is done with clay, silt, and sand content, contact the workshop instructors.

Let's make a map of the soil types on this field. First, we need to locate the part of `tabular` with the soil names; these can be found in `muaggatt`.

```
names <- ssurgo$tabular$muaggatt
```

Exercise 4: What are the soil types present on the field as seen in `names`? Are the soil defined by anything other than the soil type?

Exercise 4 Solution

```
names
```

```
## # A tibble: 9 x 40
##   musym muname mustatus slopegraddcp slopegradwta brockdepmin wtdepannmin
##   <chr> <chr>  <lgl>           <dbl>        <dbl> <lgl>             <dbl>
## 1 AdB   Alexa~ NA                  3          3   NA                  153
## 2 BgB   Benni~ NA                  4          3.9 NA                   22
## 3 Bw    Bono ~ NA                  1          1   NA                    7
## 4 EtA   Ellio~ NA                  2          2   NA                   31
## 5 Lu    Luray~ NA                  1          1   NA                    7
## 6 Pm    Pewam~ NA                  1          1   NA                   15
## 7 TrA   Tiro ~ NA                  1          1   NA                   22
## 8 BeA   Benni~ NA                  1          1.2 NA                   22
## 9 Crd1~ Cardi~ NA                  3          2.8 NA                   46
## # ... with 33 more variables: wtdepaprjunmin <dbl>, flodfreqdcd <chr>,
## #   flodfreqmax <chr>, pondfreqprs <dbl>, aws025wta <dbl>,
## #   aws050wta <dbl>, aws0100wta <dbl>, aws0150wta <dbl>, drclassdcd <chr>,
## #   drclasswettest <chr>, hydgrpdcd <chr>, iccdcd <lgl>, iccdcdpct <dbl>,
## #   niccdcd <dbl>, niccdcdpct <dbl>, engdwobdcd <chr>, engdwbdcd <chr>,
## #   engdwbll <chr>, engdwbml <chr>, engstafdcd <chr>, engstafll <chr>,
## #   engstafml <chr>, engsldcd <chr>, engsldcp <chr>, englrsdcd <chr>,
## #   engcmssdcd <chr>, engcmssmp <chr>, urbrecptdcd <chr>,
## #   urbrecptwta <dbl>, forpehrtdcp <chr>, hydclprs <dbl>,
## #   awmmfpwwta <dbl>, mukey <dbl>
```

Looking at `names` we can see there are eight types of soil on the field, and the dataframe reports areas with different slopes with different names. We often know the slope of the field, and so we may want to combine areas of the field with the same soil type and different slopes.

We need one dataframe with both the soil name and spatial data. We will merge the soil data and the spatial data by the `musym`. Note that in one of the dataframes the variable is capitalized and not in the other. We must rename the variable for consistency using `rename()` from `dplyr`.

```
spatial <- as(ssurgo$spatial, "sf")
spatial <- dplyr::rename(spatial, musym = MUSYM)
spatial <- merge(spatial, names, by = "musym")
head(spatial$muname)
```

```
## [1] "Alexandria silt loam, 2 to 6 percent slopes"
## [2] "Bennington silt loam, 0 to 2 percent slopes"
## [3] "Bennington silt loam, 2 to 6 percent slopes"
## [4] "Bono silty clay loam"
## [5] "Bono silty clay loam"
## [6] "Bono silty clay loam"
```

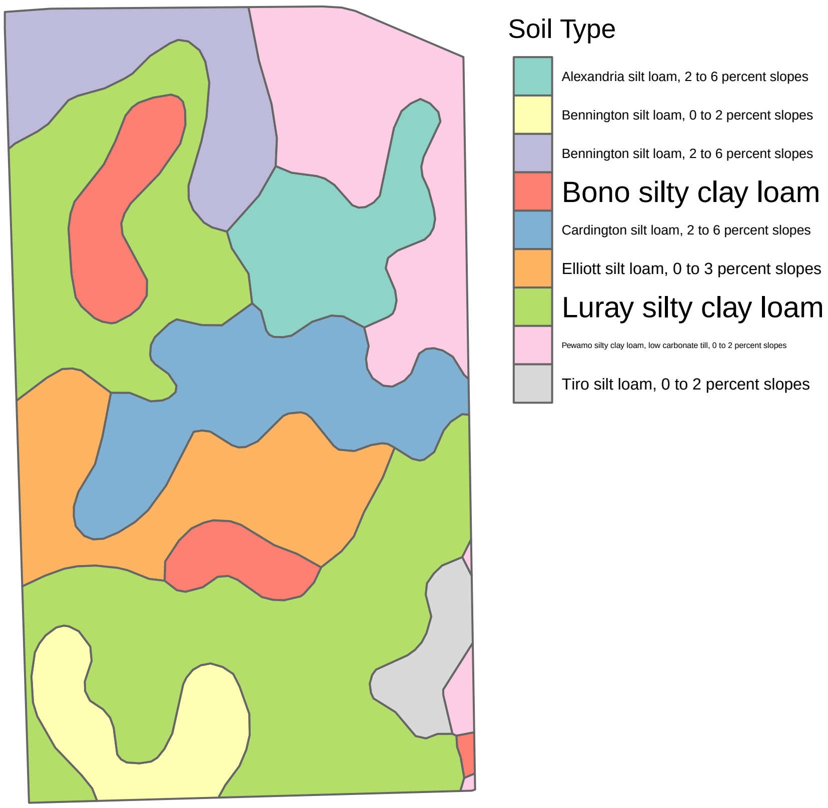
Exercise 5: Create the Soil Map

Use `tmap` to make a map where the polygon color is informed by the soil names in `muname`. *Hint*: use `tm_polygons()`.

Exercise 5 Solution

```
map_soil <- tm_shape(spatial) + tm_polygons('muname', title = "Soil Type") + tm_layout(legend.outside =
  tm_legend(text.size = 1,
            title.size = 1,
            width = 100,
            bg.color = "white"
           )
map_soil
```

```
## Legend labels were too wide. The labels have been resized to 0.43, 0.42, 0.42, 0.89, 0.42, 0.49, 0.8
```



The map shows that there are quite a few soil types on the field, and several show up in different section of the field. However, most of the soil are silt loam. It might be difficult to understand the different soils without more information about soil weathering and texture. This is also provided within SSURGO, and is likely, something you know about in your own field.