Welcome to The Carpentries Etherpad!

This pad is synchronized as you type, so that everyone viewing this page sees the same text. This allows you to collaborate seamlessly on documents.

Use of this service is restricted to members of The Carpentries community; this is not for general purpose use (for that, try etherpad.wikimedia.org).

Users are expected to follow our code of conduct: https://docs.carpentries.org/topic_folders/policies/code-of-conduct.html

Welcome to the Geospatial Data Carpentry!!
Workshop website: https://uw-madison-datascience.github.io/2019-07-11-uwmadison-dc/
DC GitHub (for helpers): https://github.com/datacarpentry/r-raster-vector-geospatial/

SETUP

OSX instructions:

- Install R: https://cran.r-project.org/bin/macosx/R-3.6.1.pkg
- Install RStudio: https://download1.rstudio.org/desktop/macos/RStudio-1.2.1335.dmg
- Open RStudio and install some R packages:
- install.packages(c("dplyr", "ggplot2", "raster", "rgdal", "rasterVis", "sf"))
- Install package of geospatial libraries (also installs some stuff we don't need, but it's easier): http://www.kyngchaos.com/files/software/frameworks/GDAL_Complete-2.4.dmg
- (If you get a warning about "Unknown developer", go to System Preferences -> Security & Privacy -> click "Open Anyway")
- Follow instructions for downloading data below

Windows instructions:

- Install R: https://cran.r-project.org/bin/windows/base/R-3.6.1-win.exe
- Install RStudio: https://download1.rstudio.org/desktop/windows/RStudio-1.2.1335.exe
- Open RStudio and install some R packages:
- install.packages(c("dplyr", "ggplot2", "raster", "rgdal", "rasterVis", "sf"))
- Install RTools: https://cran.r-project.org/bin/windows/Rtools/Rtools35.exe
- Follow instructions for downloading data below

(Everyone) Data instructions:

- Download data for Intro to R lessons, unzip, and place on your Desktop: https://github.com/UW-Madison-DataScience/2019-07-11-uwmadison-dc/raw/gh-pages/files/r-geospatial.zip
- Download data for geospatial lessons, unzip, and add to r-geospatial directory now on your

Desktop (as time allows, large file): https://ndownloader.figshare.com/articles/2009586/versions/10

- Check that your final directory structure looks like this:
- ~/Desktop/r-geospatial/
    - data/
    - gapminder_data.csv
    - NEON-DS-Airborne-Remote-Sensing/
    - NEON-DS-Landsat-NDVI/
    - NEON-DS-Met-Time-Series/
    - NEON-DS-Site-Layout-Files/
    - nordic-data.csv
    - nordic-data-2.csv

Notes from the Intro R Section:

Good Enough Practices in Scientific Computing: https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1005510

R can do math: default is to print the answer to the console

3+5

output to the console: 8

Assigning values to object with the assignment operator (<-):
Recall, you can use R as a calculator (a very powerful calculator)
In R, can use <- or = to assign things to an object;
Ex: today.date <- 11 #using a dot for naming something
today_date <-11 #using _ to name something
todayYear <- 2019 #camel case to name something

#Challenge #1: Which of the following are valid R variable names?
min_height, max.height, MAxLength, celsius2kelvin
.mass (this works but it is hidden so it won't come up in your environment - best to stay away)

Project organization - VERY important (I say this from personal experience).
- best practice, keep your data in the data/ folder untouched so you can always go back to your data if something goes awry
- .txt files (sometimes called README files) are helpful to put in each folder so you know what belongs where

r-geospatial/

- data/ #data goes here, do not touch
- doc/ #documentation goes here
- results/ #analysis results
- src/ #source scripts (code goes here, sometimes you'll also see that compiled code goes here and R code goes in a folder called R/)

https://speakerdeck.com/jennybc/how-to-name-files

We created a project folder inside our r-geospatial/ folder that lives on desktop
We created a folder called `src/` which stands for 'source' to hold our scripts
In that src folder we created a new file called R_refresher.R

R has **tab completion.** This is very useful. If ever in doubt about what you can do with a function, press Tab.

Loading in data:

```
nordic <- read.csv("data/nordic-data.csv") #remember, our project folder is r-geospatial. From there we want to load in the nordic data that lives in our data/ folder.
nordic$country
nordic$lifeExp
nordic$country + 2
nordic$country + nordic$lifeExp #R gets angry because these are factors
#classes
class(3.14) #numeric
class(1L) #this is an integer, that's what the L tells us
class(1+1i) #this is a complex number
class(TRUE) #Logicals

class("banana") #character, either single or double quotes work
class(factor("banana")) #tricky factors, particularly useful for telling R it is a category (if you're familiar with SAS, declaring something as a factor is equivalent to saying it's a class)

#working with vectors
my_vector <- c(2,4,6) #c is for combine or concatenate multiple values into a vector
class(my_vector) #this is a numeric vector

experiment <- c(2,4, "6") #note the quotation marks arouns "6"
class(experiment) #this is a character vector. Since our "6" was a character, it coerces everything else to a character data type
character_vector_example <- c("0","2","4")
class(character_vector_example) #this is a character vector
character_coerced_to_numeric <- as.numeric(character_vector_example) #this as.numeric() function will coerce our character vector to a numeric vector
class(character_coerced_to_numeric) #this is now a numeric vector!

ab_vector <-c('a','b')
ab_vector
ab_vector <- c(ab_vector, "c","d") #adding elements to an existing vector
ab_vector

my_series <- 1:10 #this will give me a series of values from 1 to 10, incremented by 1
my_sequence <- seq(10)
my_sequence
```

```
my_other_sequence <- seq(1,10, by=0.1) #increment of 0.1

#looking at vectors
head(my_other_sequence) #this gives you the first 6 elements
 head(my_other_sequence, 2)
 tail(my_other_sequence)
 length(my_other_sequence) #how many elements are in my vector

 my_example <- 5:8
 names(my_example) <- c("a", "b","c", "d")
 names(my_example)

 #what if you give it fewer names than there are elements?
 my_test <- 5:8
 names(my_test) <- c("a","b", "c")
 my_test

 #factors  - represent categorical information
 nordic_countries <- c("Norway", "Finland", "Denmark", "Iceland", "Sweden")
 class(nordic_countries) #character vector
 categories <- factor(nordic_countries) #convert character vector to factors!
 str(categories) # str() function stands for structure. str() is one of my favorite R functions!
 #each of the categories have been encoded into values, but they are categories; level values are
encoded alphabetrically
 #side note: you can also relevel factors using the relevel() function


 #Challenge #2: Is there a factor in the nordic data frame? What is it's name? Try using read.csv() to
figure out how to keep text columns as character vectors. HINT: check out help pages: ?afunctionname
#     read.csv() has an option - stringsAsFactors, where the default is set to TRUE, but you can will it to
FALSE
```

- nordic <- read.csv("data/nordic-data.csv", stringsAsFactors=FALSE)
- str(nordic)

```
 #subsetting & indexing
 x <- c(5.4, 6.2, 7.1, 4.8, 7.5)
 names(x) <- c("a", "b", "c", "d", "e")
x[1]
x[c(1,3)]
x[1:4]
x[c(1,1,3)]
x[6] #gives us an NA but not an error
x[0] #gives a named numeric(0) things
x[x>7] #values where elements in x > 7
y <- 4:8
x[y > 7]
#skip and remove elements
```

```
x[-2] #skip the second element, return everything else in x
x[-c(1,3)]
x[-1:3] #this returns an error
-1:3
x[-c(1:3)]
```

#Challenge #3: Try out the different ways to call variables, observations, and elemnts from data frames. HINT: use the class() function

```
#Data Frames
gapminder <- read.csv(file = "data/gapminder_data.csv")
str(gapminder)
head(gapminder)
length(gapminder)
nrow(gapminder) # number of rows in df
ncol(gapminder) #number of columns
dim(gapminder) #gives you both the rows and the columns of the dataframe

#adding columns or rows
below_average <- gapminder$lifeExp < 70.5 #returns a logical (TRUE or FALSE)
head(cbind(gapminder, below_average)) #add a vector to a dataframe and return only first 6 rows of this
test_average <- c("TRUE", "TRUE", "FALSE", "TRUE")
head(cbind(gapminder, test_average))
#because the length of test_average is shorter than and a factor of the number of rows in gapminder, R recycles the vector test_average, which is probably not what you want it to do - be careful!

new_row <- list("Norway", 2016, 5000000, "Nordic", 80.3, 49754,  TRUE)
new_row #this is now a list of 7 elements
gapminder_norway <- rbind(gapminder, new_row)
head(gapminder)
levels(gapminder$continent) <- c(levels(gapminder$continent), "Nordic")
levels(gapminder$continent)
tail(gapminder_norway)

#slicing data frames
head(gapminder[3])
head(gapminder[["lifeExp"]])
gapminder[1:3]
head(gapminder[1:3]) #this returns columns
gapminder[1:3,] #this gives us all columns, but first 3 rows

#Data frame manipulation using the dplyr package.
#if you haven't installed dplyr, run:
  #install.packages("dplyr")
#if you have it installed, you can just load it into your workspace:
  library(dplyr) #side note, library(tidyverse) will also give you dplyr as well as ggplot2 packages
  #selecting using dplyr
  year_country_gdp <- select(gapminder, year, country, gdpPercap) #select is a function from the dplyr package, The first argument (gapminder) is the name of the dataframe. The subsequent arguments (year,
```

country, gdpPercap) tell it which columns to select from that dataframe
head(year_country_gdp)

#Pipe operator: %>%
#if you're familiar with Bash/Unix Shell, this is equivalent to the vertical bar, | , which passes arguments from the left side to the function on the right side. In R, it just looks like %>%

#selecting
year_country_gdp <- gapminder %>% select(year, country, gdpPercap)

#filtering
year_country_gdp <- gapminder %>%

- filter(continent == "Europe") %>%
  - select(year, country, gdpPercap)

- #summarize and group_by
- gdp_bycontinents <- gapminder %>%
  - group_by(continent) %>% #grouping by continent
  - summarize(mean_gdpPercap = mean(gdpPercap)) #find mean gdp per continent
- gdp_bycontinents <- gapminder %>%
  - group_by(continent, year) %>% #grouping by continent
  - summarize(mean_gdpPercap = mean(gdpPercap)) #find mean gdp per continent

- gdp_bycontinents_byyear <- gapminder %>%
  - group_by(continent,year) %>%
  - summarize(mean_gdpPercap = mean(gdpPercap), sd_gdpPercap = sd(gdpPercap), mean_pop = mean(pop), sd = sd(pop))

#side note: if you have NA's in your data and you want to use a summary function such as mean(), or sd(), you will need to add the argument, na.rm=TRUE, otherwise, the entire vector will be returned as NA
#Ex: if there are NA's in your data

- #gdp_bycontinents_byyear_withNAsexample <- gapminder %>%
  - #group_by(continent,year) %>%
  - #summarize(mean_pop = mean(pop, na.rm=TRUE), sd = sd(pop, na.rm=TRUE))
- #count() and n()
- #count() is like group_by() + n() functions, all in one
- ncountries <- gapminder %>% filter(year == 2002) %>%
  - count(continent, sort=TRUE)
- new_summary <- gapminder %>%
  - group_by(continent) %>%
  - summarize(se_le = sd(lifeExp)/sqrt(n()))
- new_summary
- #Visualization using ggplot2 package
- library(ggplot2)
- ggplot(data=gapminder, aes(x=lifeExp)) + geom_histogram()
- gapminder_small <- filter(gapminder, year == 2007, continent == "Americas")
- ggplot(data = gapminder_small, aes(x=country, y = gdpPercap)) +

        - geom_col() +
        - coord_flip()
- #writing data out from R
- austr_subset <- filter(gapminder, country == "Australia")
- write.csv(austr_subset, file = "results/gapminder_aus.csv") # we created a new folder called results/ that lives under our r-geospatial/ master folder

Questions:
Can you discuss what the advantages are of using an R project? How does it differ from just setting a working directory location?

- Tobin: you can zip the project folder and send it to a collaborator. the relative file paths will work on their machine
- Sarah: Works well with version control.
- More info: https://support.rstudio.com/hc/en-us/articles/200526207-Using-Projects
    - https://www.r-bloggers.com/%F0%9F%93%81-project-oriented-workflow/

Will we have access to this etherpad (or the info on it) after this class?

- Yes. As long as the (global) carpentries organization continues to host etherpads it will remain up but... you can also export it using the two arrows buttons in the right hand corner.

Does the $ symbol in R work the same way as period does in sql?

- I'm not super familiar with sql but yes that is how it works in R. The $ with a column name after it will specify only that specific column (as a vector)

Can you clarify what %>% does?

- The pipe takes the 'output' of what comes before and uses it as the input(the first argument) of the next function on the right side.

How does the tibble affect future charting, plotting, or use of a dataframe?

- Other instructors correct this if needed, for the most part I tihnk they work similarlly to dataframes but often display a little nicer. If you'd like to read more this link (https://r4ds.had.co.nz/tibbles.html) might be of interest.

Is it easier to read code if each pipe is on its own line? If so, how do I tell R to do make that happen?

- A lot of R users will preser to code such that each dplyr or ggplot statement is on one line (so the code is longer not wider).To do so, you can just press Enter. Be sure that the new line does not start with an operator though (such as +, or %>%) - this will cause an error. <- Also be sure the first line above ends in a + or a %>% because then R knows to keep looking for the rest of the command on the next line. Side note: the indentation makes it easier to read but is not required for the code to run.
- Example:
    - new_summary <- gapminder %>%
        - group_by(continent) %>%
        - summarize(se_le = sd(lifeExp)/sqrt(n()))

#INTRO TO GEOSPATIAL CONCEPTS

Altneratives for 'point': https://www.thesaurus.com/browse/main%20point

- Not sure any captures it perfectly but "main idea" comes close..
- Depending on intended usage "comment." e.g. when thanking some participant for their great comment.

NEON sites: https://www.neonscience.org/about-neon-field-sites

The ultimate list of GIS format and geospatial file extensions: https://gisgeography.com/gis-formats/

 # Geospatial Data in R

```
# Load Libraries
library(ggplot2)
library(dplyr)
library(raster)
library(rgdal)

# view ratser file atttributes
GDALinfo("data/NEON-DS-Airborne-Remote-Sensing/HARV/DSM/HARV_dsmCrop.tif")

# Open a raster in R
DSM_HARV <- raster("data/NEON-DS-Airborne-Remote-Sensing/HARV/DSM/HARV_dsmCrop.tif")
summary(DSM_HARV)
summary(DSM_HARV, maxsamp=ncell(DSM_HARV))

DSM_HARV_df <- as.data.frame(DSM_HARV, xy=TRUE)
head(DSM_HARV_df)
str(DSM_HARV_df)

# make a super cool plot!
ggplot() +
   geom_raster(data = DSM_HARV_df, aes(x = x, y = y, fill = HARV_dsmCrop)) +
   scale_fill_viridis_c() +
   coord_quickmap()

# view raster CRS (coodinate reference system) in R
crs(DSM_HARV)

# CHALLENGE 1: What units are our data in?
+proj=utm +zone=18 +datum=WGS84 +units=m +no_defs +ellps=WGS84 +towgs84=0,0,0
```

- A: Meters

```
# Calculate raster min/max values
minValue(DSM_HARV)
maxValue(DSM_HARV)
nlayers(DSM_HARV)


# CHALLENGE 2: Use GDALinfo() to figure out the what the no data value is for our raster?
```

- A: -9999

```
# create histogram
ggplot() +
  geom_histogram(data = DSM_HARV_df, aes(HARV_dsmCrop))

# CHALLENGE #3 Work as a group: Use GDALinf() to determine the following about the NEON-DS-Airborne-Remote-Sensing/HARV/DSM/HARV_DSMhill.tif file:
    - Does this file have the same CRS as DMS_HARV
    - What is the NoDataValue?
    - What is the resolution of the rater data?
    - How large would a 5x5 pixel area be on the Earth's surface?
    - Is the file a multi- or single-band raster?



# plotting using breaks (bins)
custom_bins <- c(300, 350, 400, 450)
class(custom_bins)

# use dplyr to create a vector of meaningful bins
DSM_HARV_df <- DSM_HARV_df %>%
    mutate(fct_elevation_2  = cut(HARV_dsmCrop,
                                  breaks = custom_bins))
head(DSM_HARV_df)
unique(DSM_HARV_df$fct_elevation_2)

# visualize bin
ggplot() +
  geom_bar(data = DSM_HARV_df, aes(fct_elevation_2))

# another cool plot
ggplot() +
    geom_raster(data = DSM_HARV_df,
                aes(x = x, y = y, fill = fct_elevation_2 ))+
    coord_quickmap()


# Making the cool plot cooler
```

```
ggplot() +
  geom_raster(data = DSM_HARV_df,
              aes(x = x, y = y, fill = fct_elevation_2 )) +
  coord_quickmap() +
  scale_fill_manual(values = terrain.colors(3)) +
  xlab("UTM Westing Coordinate (m)") +
  ylab("UTM Northing Coordinate (m)") +
  ggtitle("Classified Elevation Map")

# CHALLENGE #4 : Create a plot of the Harvard Forest Digital Surface Model (DSM) (Hint use
DSM_HARV_df) that has:
    - Six classified ranges of values (break points) that are evenly divided among the range of pixel values.
    - Axis labels
    - A plot title

DSM_HARV_df <- DSM_HARV_df  %>%
              mutate(fct_elevation_6 = cut(HARV_dsmCrop, breaks = 6))

head(DSM_HARV_df)
unique(DSM_HARV_df$fct_elevation_6)

ggplot() +
    geom_raster(data = DSM_HARV_df , aes(x = x, y = y,
                                         fill = fct_elevation_6)) +
    scale_fill_manual(values = terrain.colors(6), name = "Elevation") +
    ggtitle("Classified Elevation Map - 6 Levels") +
    xlab("UTM Westing Coordinate (m)") +
    ylab("UTM Northing Coordinate (m)") +
    coord_quickmap()

# layering rasters
DSM_hill_HARV <-
  raster("data/NEON-DS-Airborne-Remote-Sensing/HARV/DSM/HARV_DSMhill.tif")

# attributes of the raster
DSM_hill_HARV

# convert raster to data.frame
DSM_hill_HARV_df <- as.data.frame(DSM_hill_HARV, xy = TRUE)
head(DSM_hill_HARV_df)

# plot!
ggplot() +
  # first raster
  geom_raster(data = DSM_hill_HARV_df,
              aes(x = x, y = y, fill = HARV_DSMhill)) +
  # second raster
  geom_raster(data = DSM_hill_HARV_df,
              aes(x = x, y = y, alpha = HARV_DSMhill)) +
```

```
  scale_fill_viridis_c() +
  scale_alpha(range =  c(0.15, 0.65)) +
  coord_quickmap()

# raster calculations
# start here if you need to catch up after the morning break!

# load in terrain model
DTM_HARV <-
  raster("data/NEON-DS-Airborne-Remote-Sensing/HARV/DTM/HARV_dtmCrop.tif")

DTM_HARV

# raster math!
CHM_HARV <- DSM_HARV - DTM_HARV

# convert raster to data.frame
CHM_HARV_df <- as.data.frame(CHM_HARV, xy = TRUE)

head(CHM_HARV_df)

# plot for canopy height model
ggplot() +
   geom_raster(data = CHM_HARV_df ,
               aes(x = x, y = y, fill = layer)) +
   scale_fill_gradientn(name = "Canopy Height",
                        colors = terrain.colors(10)) +
   coord_quickmap()

# histogram for canopy height model
ggplot() +
  geom_histogram(data=CHM_HARV_df, aes(layer))

# CHALLENGE: It's often a good idea to explore the range of values in a raster dataset just like we
might explore a dataset that we collected in the field.

# What is the min and maximum value for the Harvard Forest Canopy Height Model (CHM_HARV) that
we just created?

minValue(CHM_HARV)
maxValue(CHM_HARV)

# What are two ways you can check this range of data for CHM_HARV?

min(CHM_HARV_df$layer, na.rm = TRUE)
max(CHM_HARV_df$layer, na.rm = TRUE)

# overlay
CHM_ov_HARV <- overlay(DSM_HARV,
```

```
                DTM_HARV,
                fun = function(r1, r2) { return( r1 - r2) })

class(CHM_ov_HARV)

# convert raster to data.frame
CHM_ov_HARV_df <- as.data.frame(CHM_ov_HARV, xy = TRUE)

# plot!
ggplot() +
  geom_raster(data = CHM_ov_HARV_df,
              aes(x = x, y = y, fill = layer)) +
  # legend title is both meaningful and cool!
  scale_fill_gradientn(name = "Canopy Height", colors = terrain.colors(10)) +
  coord_quickmap()

# export a GeoTIFF
writeRaster(CHM_ov_HARV, "results/CHM_HARV.tiff",
            format="GTiff",
            overwrite=TRUE,
            NAflag=-9999)



# work with multi-band rasters

# load raster
RGB_band1_HARV <-
raster("data/NEON-DS-Airborne-Remote-Sensing/HARV/RGB_Imagery/HARV_RGB_Ortho.tif")

# attributes of the raster
RGB_band1_HARV
class(RGB_band1_HARV)
nlayers(RGB_band1_HARV)
summary(RGB_band1_HARV)

# convert raster to data.frame
RGB_band1_HARV_df  <- as.data.frame(RGB_band1_HARV, xy = TRUE)

# plot!
ggplot() +
 geom_raster(data = RGB_band1_HARV_df,
             aes(x = x, y = y, alpha = HARV_RGB_Ortho)) +
 coord_quickmap()

# load specific band
RGB_band2_HARV <-
raster("data/NEON-DS-Airborne-Remote-Sensing/HARV/RGB_Imagery/HARV_RGB_Ortho.tif", band
= 2)
```

```
# create raster stack
RGB_stack_HARV <-
stack("data/NEON-DS-Airborne-Remote-Sensing/HARV/RGB_Imagery/HARV_RGB_Ortho.tif")

# attributes of the stack
RGB_stack_HARV
RGB_stack_HARV@layers

# plot!!!
plotRGB(RGB_stack_HARV,
      r = 1, g = 2, b = 3)
plotRGB(RGB_stack_HARV,
      r = 1, g = 2, b = 3,
      stretch = "lin")
plotRGB(RGB_stack_HARV,
      r = 1, g = 2, b = 3,
      stretch = "hist")

# create raster stack
RGB_brick_HARV <- brick(RGB_stack_HARV)

# view the size
object.size(RGB_brick_HARV)
object.size(RGB_stack_HARV)

# CHALLENGE: What Functions Can Be Used on an R Object of a particular class?

methods(class=class(RGB_stack_HARV))
```

Rasterbrick vs rasterstack
Key distinction seems to be that the multiple layers in a rasterbrick must all be contained in one file, whereas in a rasterstack, the multiple layers might each be in a separate file. This means the rasterbrick is more restrictive / less general, but also means there's an opportunity for greater efficiency.

See this page: https://rspatial.org/spatial/4-rasterdata.html#rasterstack-and-rasterbrick
Also this: https://www.rdocumentation.org/packages/raster/versions/2.9-5/topics/brick

Projections & Re-Projections:
For anyone who was interested in projections (and re-projections), here is a link to that lesson:
https://datacarpentry.org/r-raster-vector-geospatial/03-raster-reproject-in-r/index.html

Questions:
Where did HARV_dsmCrop come from for the fill?

- A: It was the other data column name and it was the name of the original raster file.

Can you define no data value is this null / 0 / or blank spaces?

- A: It is like a "null"/ missing data value

How do we know if we are actually missing data (just if we have areas of no data, but not where it is)?

- A: One way is to look at the summary of the raster data using summary(DSM_HARV)
- A: Another way is to use the data frame we created from raste
    - sum(DSM_HARV_df$HARV_dsmCrop==-9999)
    - This identifies the number of rows in the df that are equal to our no data value of -9999

What's the difference between a brick and a stack? It's still a little confusing...

- A: "The raster package has two classes for multi-layer data the RasterStack and the RasterBrick. The principal difference between these two classes is that a RasterBrick can only be linked to a single (multi-layer) file. In contrast, a RasterStack can be formed from separate files and/or from a few layers (‘bands’) from a single file."

Vector Data in R section:

Objects we will be working with:
polygon - aoi_boundry_HARV
line - lines_HARV
point - point_HARV
Good practice: always add a little header with information about what the script is doing and the date

```
# Load your libraries
library(raster)
library(rgdal)
library(ggplot2)
library(dplyr)
library(sf) --> if this doesn't load becuase of something relate to "group_map()", update your version of dplyr, then try again.

# Import AOI boundry file
aoi_boundary_HARV <- st_read("data/NEON-DS-Site-Layout-Files/HARV/HarClip_UTMZ18.shp")

# look at geometry type
st_geometry_type(aoi_boundary_HARV)

# look at CRS
st_crs(aoi_boundary_HARV)

# exent
st_bbox(aoi_boundary_HARV)

ggplot()  +
  geom_sf(data = aoi_boundary_HARV, size = 3, color = "black", fill = "cyan1") +
  ggtitle("AOI Boundary Plot") +
  coord_sf()
```

Challenge: Use the steps above, import the HARV_roads as: lines_HARV and HARVtower_UTM18N as: point_HARV
1. What type of R spatial objects is created when you import each layer?
2. What is the CRS and extent for each obejct?
3. How many spatial obects are in each file?

```
#import road data
lines_HARV <- st_read("data/NEON-DS-Site-Layout-Files/HARV/HARV_roads.shp")
point_HARV <- st_read("data/NEON-DS-Site-Layout-Files/HARV/HARVtower_UTM18N.shp")

# editoralizing from Christina+1
# I always save my data location as its own variable and then load it, like:
road_datafile <- "data/NEON-DS-Site-Layout-Files/HARV/HARV_roads.shp"
lines_HARV <- st_read(road_datafile)
# which is more lines of code, but is sometimes easier to read

# CRS
st_crs(lines_HARV)
st_crs(point_HARV)

#extent
st_bbox(lines_HARV)
st_bbox(point_HARV)

#vector type
st_geometry_type(lines_HARV)
st_geometry_type(point_HARV)
```

missed some notes here....:(, I think

```
# how many columns do we have in the line data?
ncol(lines_HARV)

# what are the names of the columnsk
names(lines_HARV)

# Look at 1st 6 rows
head(lines_HARV)


# look at the TYPE (the column called type in the data set)  of lines
lines_HARV$TYPE

# How many levels do we have?
levels(lines_HARV$TYPE)

# filter for footpath type of road
footpath_HARV <- lines_HARV %>% filter(TYPE == "footpath")
```

```
nrow(footpath_HARV)

# Let's plot it! the footpath data
ggplot() +
   geom_sf(data = footpath_HARV) +
   ggtitle("NEON Harvard Forest Field Site", subtitle = "Footpaths") +
   coord_sf()



# Change line color and size
ggplot() +
   geom_sf(data = footpath_HARV, aes(color = factor(OBJECTID)), size = 1.5) +
 labs(color = 'Footpath ID') +
   ggtitle("NEON Harvard Forest Field Site", subtitle = "Footpaths") +
   coord_sf()
#plot only the board lines
boardwalk_HARV <- lines_HARV %>%
filter(TYPE == "boardwalk")

nrow(boardwalk_HARV)

ggplot() +
```

- `geom_sf(data = boardwalk_HARV, size = 1.5) +`
- `ggtitle"NEON Harvard Forest Field Site", subtitle = "Boardwalks") +`
- `coord_sf()`

```
stoneWall_HARV <- lines _HARV %>% filter(TYPE == "stone wall")

nrow(stoneWall_HARV)

ggplot() +
   geom_sf(data = stoneWall_HARV, size = 1.5, aes(color = factor(OBJECTID)) +
   ggtitle("NEON harvard Forest Field Site", subtitle = "stone walls") +
   coord_sf()

# Changing the legend to lines
ggplot() +
   geom_sf(data = stoneWall_HARV, size = 1.5, aes(color = factor(OBJECTID)), show.legend = "line")
+
   ggtitle("NEON harvard Forest Field Site", subtitle = "stone walls") +
   coord_sf()

ggplot() +
  geom_sf(data = lines_HARV, color = "black") +
  geom_sf(data = aoi_boundary_HARV, color = "grey20", size = 1) +
  geom_sf(data = point_HARV, pch = 8) +
  coord_sf()
```

CHALLENGE:

Plot raster and vector data together.

Create a plot that uses the NEON aoi canopy height model (NEON-DS-Airborne-Remote-Sensing/HARV/CHM/HARV_chmCrop.tif) as a base layer.

On top of the CHM, please add:

- the study site AOI.
- roads
- the tower location.

Be sure to give your plot a meaningful title.

```
# use the CHM as A BASEMAP FOR THE PLOT
CHM_HARV <-  raster("data/NEON-DS-Airborne-Remote-Sensing/HARV/CHM/HARV_chmCrop.tif")
CHM_HARV_df <- as.data.frame(CHM_HARV, xy=TRUE)

ggplot() +
  geom_raster(data = CHM_HARV_df, aes(x = x, y = y, fill = HARV_chmCrop)) +
  geom_sf(data = lines_HARV, color = "black") +
  geom_sf(data = aoi_boundary_HARV, color = "grey20", size = 1) +
  geom_sf(data = point_HARV, pch = 8) +
  ggtitle("NEON Havard Forest Field Site", subtitle = "with CHM") +
  coord_sf()

#how to have the aoi_boundary box be completely transparent (set fill=NA) in the geom_sf() function for aoi_boundary_HARV
ggplot() +
  geom_raster(data = CHM_HARV_df, aes(x = x, y = y, fill = HARV_chmCrop)) +
  geom_sf(data = lines_HARV, color = "black") +
  geom_sf(data = aoi_boundary_HARV, color = "grey20", fill=NA,size = 1) +
  geom_sf(data = point_HARV, pch = 8) +
  ggtitle("NEON Havard Forest Field Site", subtitle = "with CHM") +
  coord_sf()

# import .csv file
plot_locations_HARV <-
read.csv("data/NEON-DS-Site-Layout-Files/HARV/HARV_PlotLocations.csv")

str(plot_lcoations_HARV)
# x  == easting
# y == northing

# peek into one of those columns
head(plot_locations_HARV$easting)

# do we have projection information?
UTM zone 18!

# we can reuse the projection info from an object we had in the past
```

```
st_crs(point_HARV)

utm19nCRS <- st_crs(point_HARV)

# convert csv file into an sf object
plot_locations_sp_HARV <- st_as_sf(plot_locations_HARV, coords = c("easting", "northing"), crs =
utm19nCRS)

# plot our filed plot data
ggplot() +
  geom_sf(data = plot_locations_sp_HARV) +
  ggtitle("Our plot locations")

# write out a shape file
st_write(plot_locations_sp_HARV, "data/PlotLocations_HARV.shp", driver = "ESRI Shapefile")
```

- missed some stuff...;(

```
CHM_HARV_Cropped <- crop(x = CHM_HARV, y = as(aoi_boundary_HARV, "Spatial"))


ggplot() +
   geom_raster(data = CHM_HARV_Cropped_df, aes(x = x, y = y, fill = HARV_chmCrop)) +
   geom_sf(data = aoi_boundary_HARV, color="blue", fill= NA) +
   coord_sf()

# Can also crop by extents

#extract raster information from a polygon (mean canopy height)
mean_tree_height_AOI <- extract(x = CHM_HARV,
                                y = as(aoi_boundary_HARV, "Spatial"),
                                fun= mean )
```

Questions?:

Q: How to center the plot title?
A:
```
ggplot() +
 ggtitle("My Title") +
  theme(plot.title = element_text(hjust = 0.5))
```
https://stackoverflow.com/questions/40675778/center-plot-title-in-ggplot2

Q: How would you set the projection if you didn't have an existing file to pull it from?
A: You will need to set the crs of the sf object sf.object using the st_crs(sf.object) by typing st_crs(sfc) = 4326 where 4326 is the EPSG code. It will also work for proj4string. Therefore, you can pipe the sf object

sf.object to the wrapper function that can set the crs of the sf object by typing points %>% st_set_crs(st_crs(polygons)). Hope this helps!

Q:what if we want to show utm axis labels and not decimal degrees?
A: I think there may be a way to supress that with theme() but I'm not sure off the top of my head

Cheatsheets for Spatial Analysis using R

Spatial manipulation with sf: https://github.com/rstudio/cheatsheets/raw/master/sf.pdf

ArcGIS to R spatial cheat sheet:
http://www.seascapemodels.org/data/ArcGIS_to_R_Spatial_CheatSheet.pdf