Reading data files

GOAL: Learn to read data into the R environment, practice plotting

First, load the tidyverse package

```r
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.3     v dplyr   1.0.7
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   2.0.0     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Notice the SN… files in the files tab. Those are comma separated values files containing data from temperature and light sensors. You can click on them to see what they look like.

These files are in our project folder, but they are NOT in our Environment. Note the environment tab (upper right) is empty.

To get them into our environment we must read them in.

We will do this with the following format:

NewData <- read_csv("datafile.csv")

read_csv is not the same as read.csv

read_csv is part of the tidyverse and reads the file name we pass to it and spits it out. We take that output and assign it to a variable using the <- operator. Also note the quotes around the file name. (about read.csv)

If you don't put the read_csv output into a variable, it'll just print it and not store it in the environment.

Read SN20680157.csv into a variable called Far then look at the data. What do you see? Any problems?

```r
Far <- read_csv("SN20680157.csv")
```

```
## Rows: 186 Columns: 1

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): Serial Number:20680157 --

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Let's think about how read_csv works and why this might have happened.

Is there anything in the read_csv help file that might help solve the problem?

Now read in the same file into Far again. Check it out. Better?

```
Far <- read_csv("SN20680157.csv", skip = 2)
```

```
## Rows: 185 Columns: 8

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (5): Button Down, Button Up, Host Connect, Stopped, EOF
## dbl  (2): Temp, (*F), Intensity, ( lum/ft2)
## dttm (1): Date Time, GMT -0400

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Rename the first three columns DateTime, Temp_F, Int_lmft2 Good names have what properties?

```
Far <- Far %>% rename(DateTime = `Date Time, GMT -0400`,
                      Temp_F = `Temp, (*F)`,
                      Int_lmft2 = `Intensity, ( lum/ft2)`)
```

Remove the columns beyond Int_lmft2, we don't need them

```
Far <- Far %>% select(DateTime, Temp_F, Int_lmft2)
```

The pipe allows us to do this in one set of statements instead of setting Far equal to itself every time. Use the pipe operator to read the file, change the column names, and select the first three columns all at once

Read -> rename -> select BaseR: select(rename(Read_file("file"), newname, newname), columns)

Tidyverse: Read_file("file") %>% rename(newname, newname) %>% select(column)

```
Far <- read_csv("SN20680157.csv", skip = 2) %>%
        rename(DateTime = `Date Time, GMT -0400`,
                      Temp_F = `Temp, (*F)`,
                      Int_lmft2 = `Intensity, ( lum/ft2)`) %>%
        select(DateTime, Temp_F, Int_lmft2) %>%
        drop_na()
```

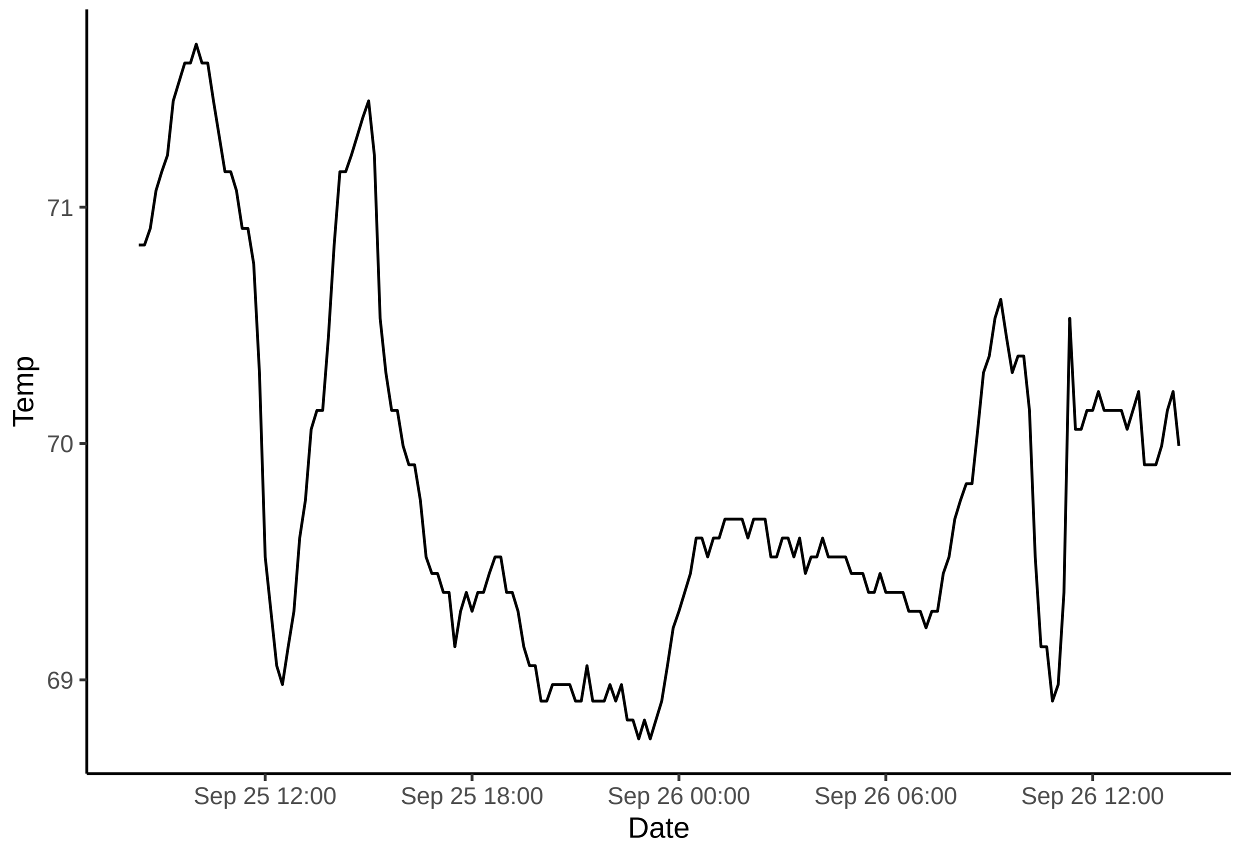
```
## Rows: 185 Columns: 8

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (5): Button Down, Button Up, Host Connect, Stopped, EOF
## dbl  (2): Temp, (*F), Intensity, ( lum/ft2)
## dttm (1): Date Time, GMT -0400
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Plot the date and time vs temperature use the classic theme plot as a line change the y and x axis labels

```
ggplot(data = Far, aes(x = DateTime, y = Temp_F))+
  geom_line()+
  labs(x = "Date", y = "Temp")+
  theme_classic()
```



Let's read in another file so we can compare it to the one we just read.

Since the sensor output is identical, we can read another sensor using the exact same code, just changing the name. This is why it is so important to keep consistent formatting!

Read the file ending in 159 and call it Near Read the file ending in 158 and call it Outside

Then check them out and see if they look ok.

```
Near <- read_csv("SN20680159.csv", skip = 2) %>%
        rename(DateTime = `Date Time, GMT -0400`,
               Temp_F = `Temp, (*F)`,
               Int_lmft2 = `Intensity, ( lum/ft2)`) %>%
        select(DateTime, Temp_F, Int_lmft2) %>%
        drop_na()
```

```
## Rows: 185 Columns: 8
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (5): Button Down, Button Up, Host Connect, Stopped, EOF
## dbl  (2): Temp, (*F), Intensity, ( lum/ft2)
## dttm (1): Date Time, GMT -0400

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
Outside <- read_csv("SN20680158.csv", skip = 2) %>%
        rename(DateTime = `Date Time, GMT -0400`,
               Temp_F = `Temp, (*F)`,
               Int_lmft2 = `Intensity, ( lum/ft2)`) %>%
        select(DateTime, Temp_F, Int_lmft2) %>%
        drop_na()
```

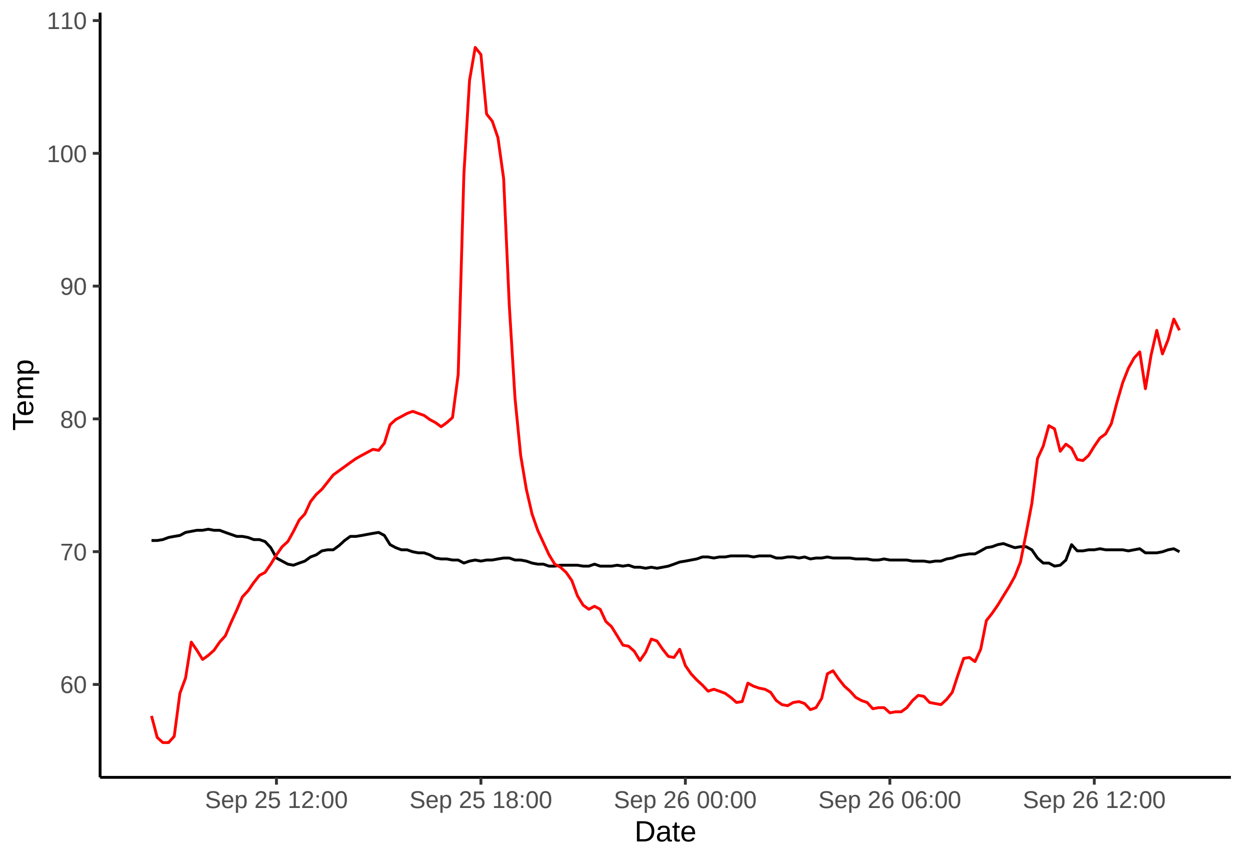
```
## Rows: 185 Columns: 8

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (5): Button Down, Button Up, Host Connect, Stopped, EOF
## dbl  (1): Temp, (*F)
## dttm (1): Date Time, GMT -0400

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Now, add the data from Far and Outside to your plot from earlier

```
ggplot(data = Far, aes(x = DateTime, y = Temp_F))+
  geom_line()+
  labs(x = "Date", y = "Temp")+
  theme_classic()+
  geom_line(data = Outside, aes(x = DateTime, y = Temp_F), color = "red")
```



It is almost always better to combine your data into one tibble and then plot from that.

How could we combine all three sets of data into 1? Don't think about code, just in general.

DateTime Temp Int ID 12 . 5 . 6 . Near 12 . 5 . 6 . near 12 . 5 . 6 . far 12 . 5 . 6 . far 12 . 5 . 6 . outside 12 . 5 . 6 . outside

We can do this in R using bind_rows()

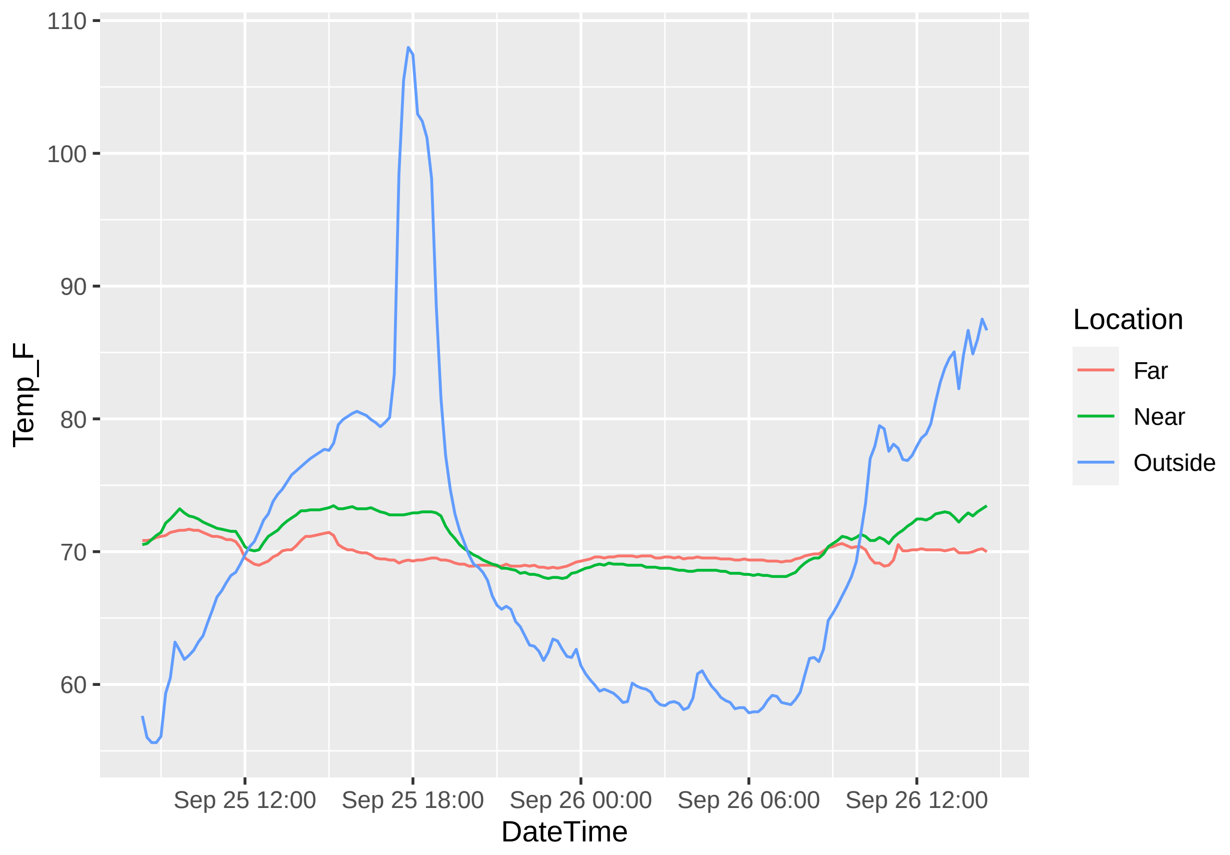
The syntax is

NewData <- bind_rows("How to ID data" = data, "How to id data" = data, .id = "What to call the new column that id's the data")

```
AllData <- bind_rows("Far" = Far,
                     "Near" = Near,
                     "Outside" = Outside,
                     .id = "Location")
```

Now plot Date against temp but assign color to Location

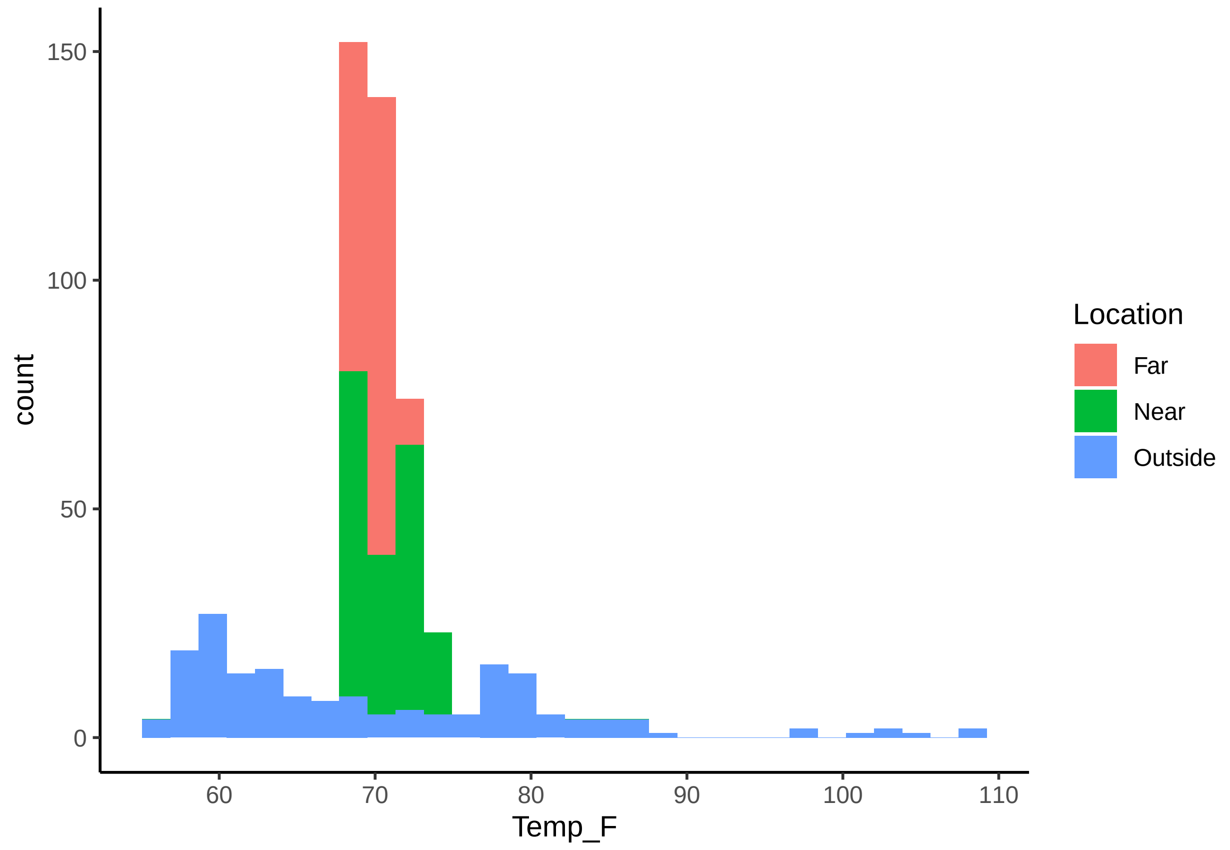
```
AllData %>% ggplot(aes(DateTime, Temp_F, color = Location))+
  geom_line()
```



Let’s look at the distribution of data in all three locations using a histogram

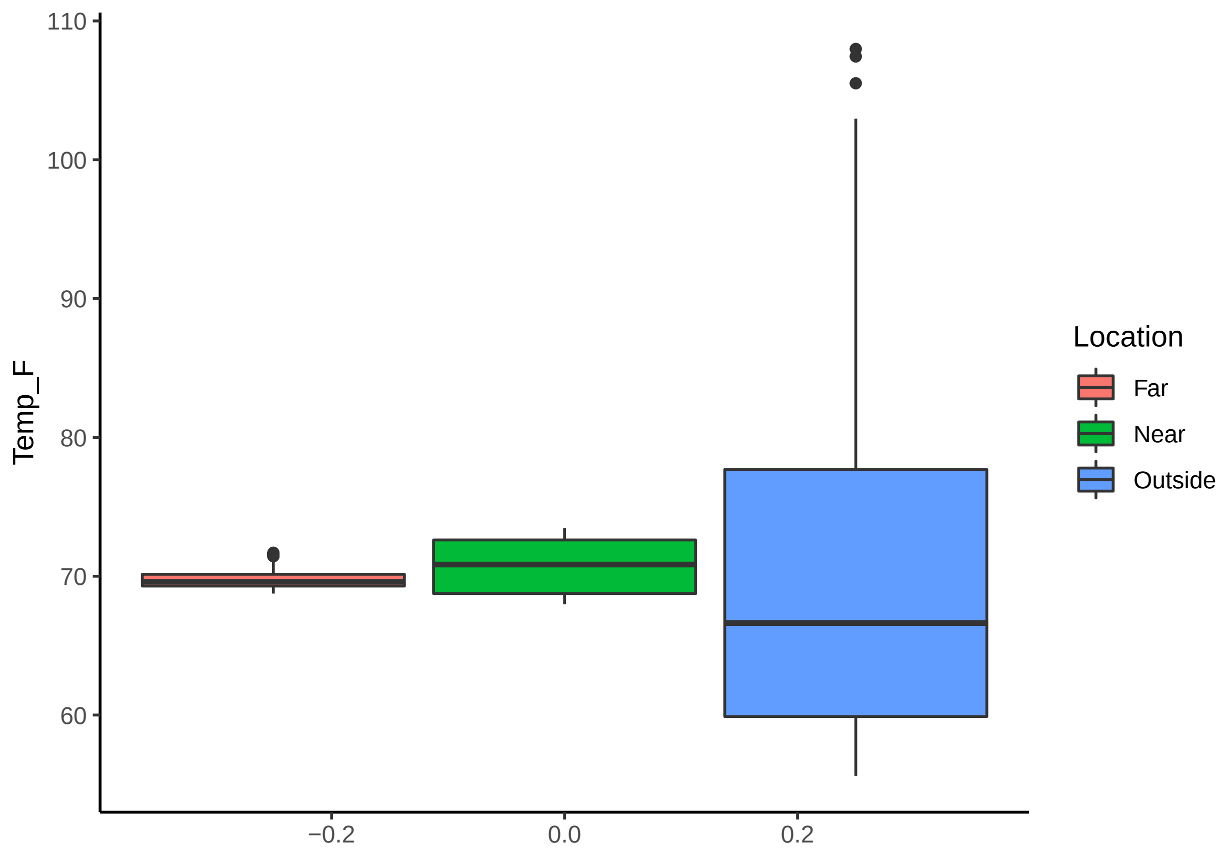
```
AllData %>% ggplot(aes(x = Temp_F, fill = Location))+
 geom_histogram()+
 theme_classic()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Let's look at the distribution as boxplots

```
AllData %>% ggplot(aes(y = Temp_F, fill = Location))+
 geom_boxplot()+
 theme_classic()
```



Make a plot of date vs temp, use the classic theme, and label the x and y axes appropriately, and show a separate plot for each location using the facet function of ggplot