01-08: Barplots

1 - Purpose

- Create barplots
- Use gradients to present a variable in a barplot
- Reorder a data frame

2 - Issues

- Can we plot in the order of a column without reordering the whole data frame?

3 - Get data

We are going to use the data file, LansingNOAA2016-3.csv, so we can access the ***dateYr*** and ***season*** columns.

```
source( file="scripts/reference.R" );
weatherData = read.csv( file="data/LansingNOAA2016-3.csv",
                        stringsAsFactors = FALSE );
```

We are going to focus on the ***precip*** column, so let's take a look at it by opening ***weatherData*** in the main window:

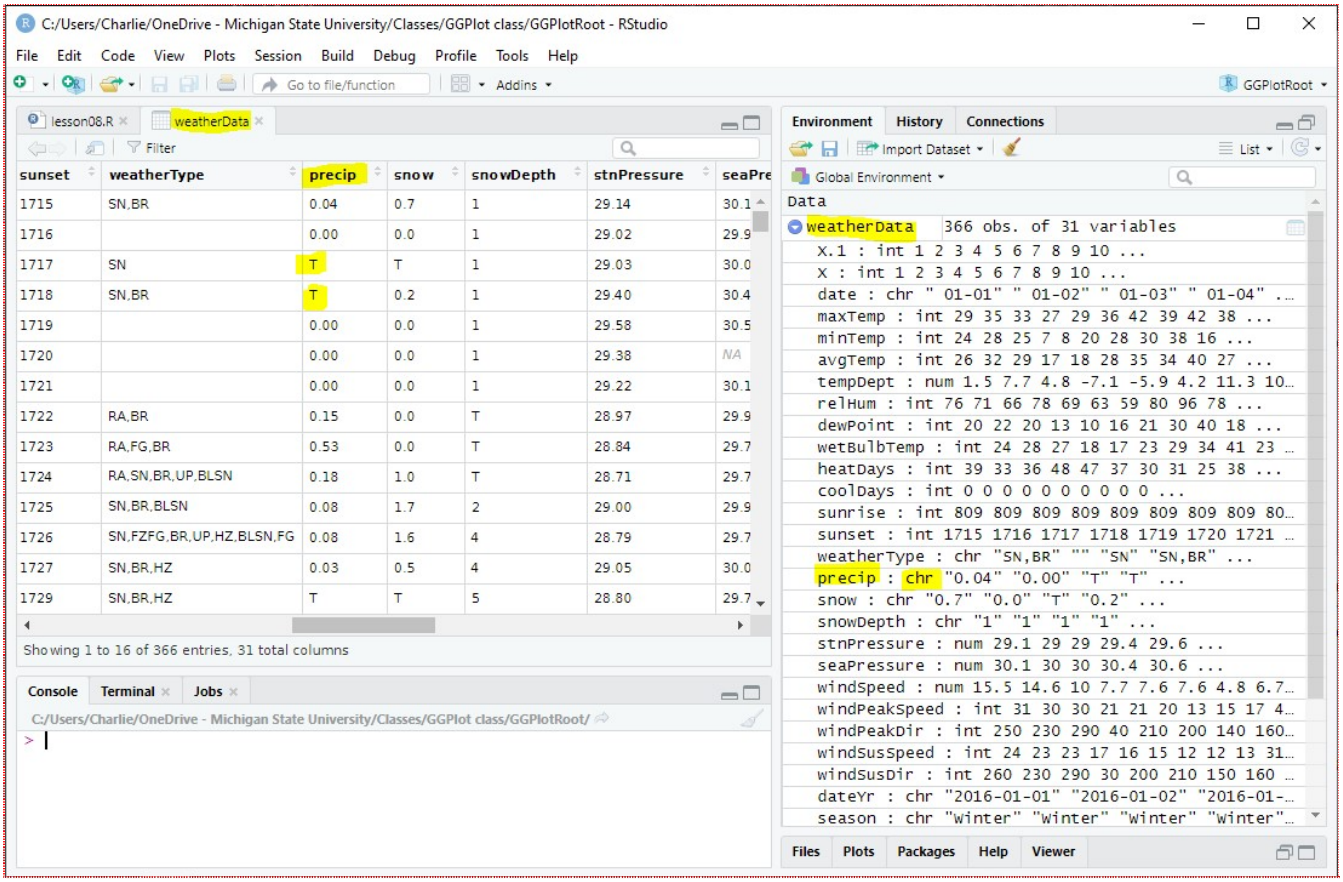

*Fig 1: The **precip** column in weatherData -- notice the non-numeric values in it.*

3.1 - Reformatting the precip column

We are going to create a bar plot of seasonal precipitation but, before we do this, we need all the values in the ***precip*** column to be numeric. The problem is that this data frame uses the character **T** to indicate there was a trace of precipitation for the day (*fig 1*). If there is even one non-numeric value in a column, *R will treat all values in the column as characters, or strings*. This means that **0.25** would be treated by R as the four characters **"0.25",** not as a number -- and you cannot do mathematical operations on characters.

So, we need to convert **T** into a number. Since the lowest measured value for precipitation is **0.01** inches, we will convert all **T** values to half that, or **0.005**.

To convert **T** to **0.005**, we will:

- copy ***precip*** to a new column, ***precipNum*** -- *note: we could just convert **precip** but it is best to maintain the original columns in the data frame*
- go through all values in the new column, ***precipNum***,and change **T** to **0.005**
- convert ***precipNum*** to a numeric column

```
#### Part 1: Convert trace rain to the numeric value 0.005
# Copy precip values to a new column, precipNum
weatherData[,"precipNum"] = weatherData[,"precip"];

# Go through all rows in weatherData
for(i in 1:nrow(weatherData))
```

```
{
  # check precipNum value -- if the value is T, change to 0.05
  if(weatherData[i,"precipNum"] == "T")
  {
    weatherData[i,"precipNum"] = 0.005;
  }
}
```

The ***for()*** converted every **T** value in ***precipNum*** to **0.005** -- so every value in ***precipNum*** consists of only numbers. But, the column will still be treated as characters *until we explicitly tell R to make it numeric* -- using ***as.numeric()***.

```
# convert precipNum column to numeric
weatherData[,"precipNum"] = as.numeric(weatherData[,"precipNum"]);
```

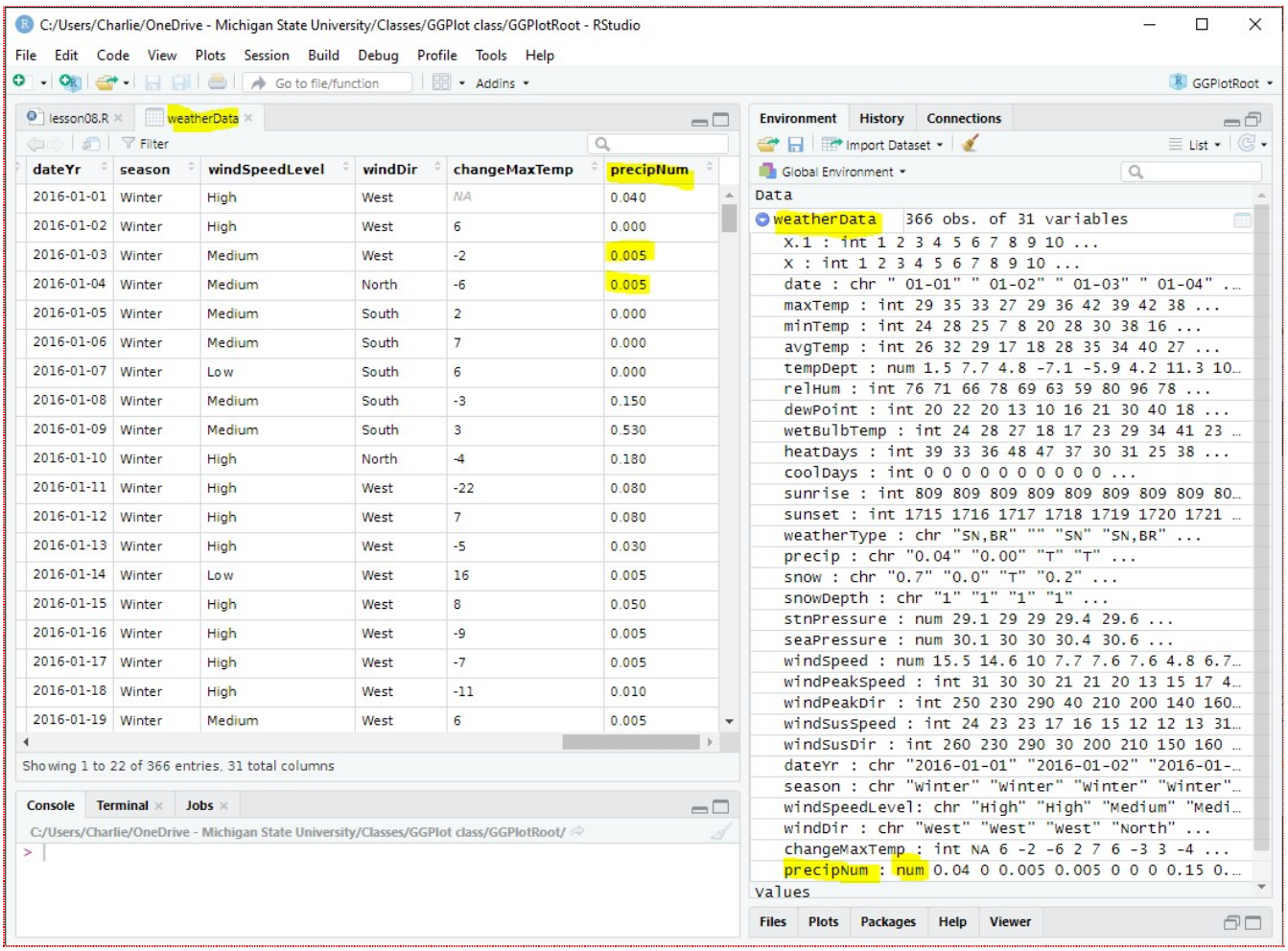

Fig 2: Precipitation values as numbers in the new column

4 - The first barplot

We are going to use the plotting component ***geom_col()*** to create a bar plot that displays the precipitation (from the ***precipNum*** column) for each ***season***. In the ***mapping*** parameter, we put ***season*** on the x-axis and ***precipNum*** on the y-axis -- GGPlot will automatically categorize ***season*** by its four values.

width, a commonly used subcomponent in ***geom_col()***, measures the thickness of the bars. ***width*** has values between **0** and **1**, with **1** meaning that the bar takes up the whole bin slot.

```
#### Part 2: Plot the precipitation for each season
thePlot = ggplot(data=weatherData) +
          geom_col(mapping=aes(x=season, y=precipNum),
                   width=0.7) +
          theme_bw() +
          labs(title = "Seasonal precipitation",
               subtitle = "Lansing, Michigan: 2016",
               x = "Seasons",
               y = "Precipitation (inches)");
plot(thePlot);
```

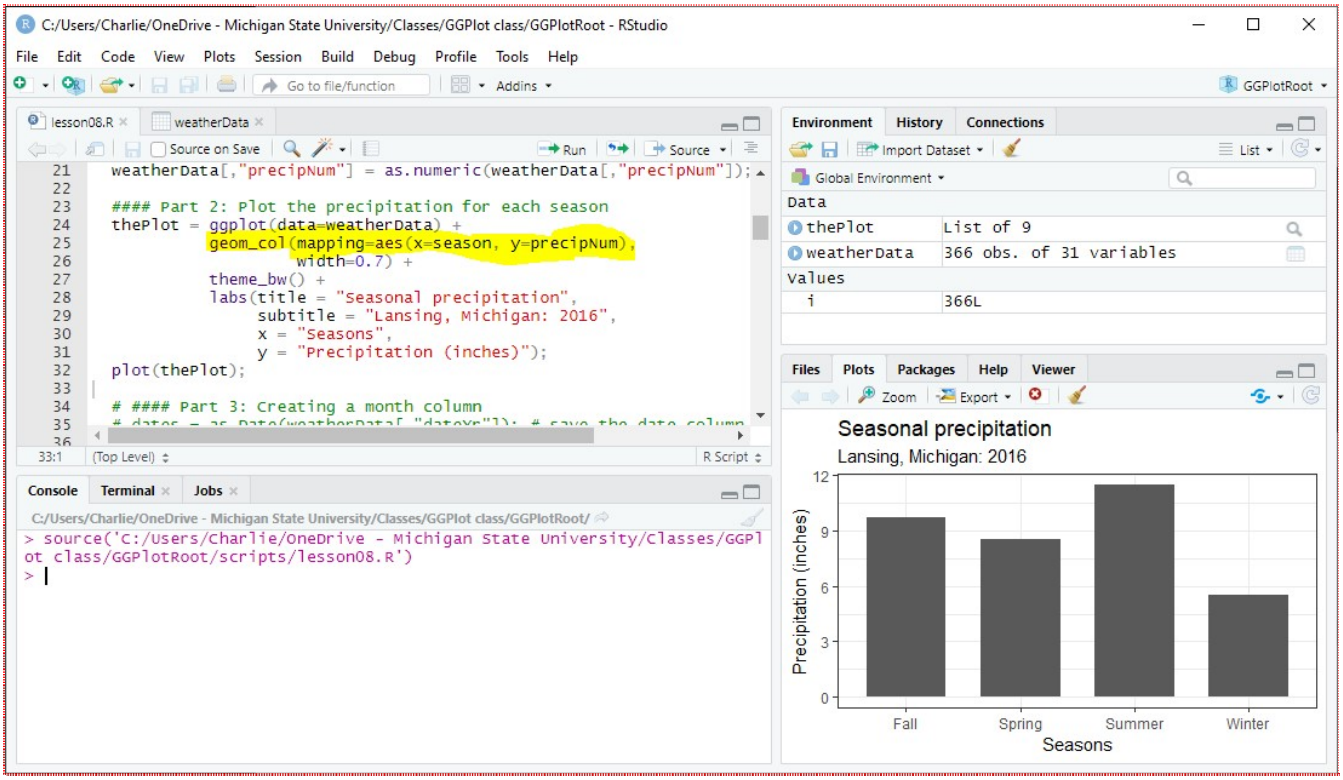

Fig 3: A bar plot of precipitation for each season

4.1 - A bar plot using months

Next, we want the precipitation for each month, but we do not have a column in the data frame for month -- instead we have a column of dates that includes the month. We are going to create a ***month*** column using the month values extracted from the ***date*** column.

The process is to:

- Save the values from the ***dateYr*** column as dates to a vector
  - ***as.Date()*** explicitly tells R that these values are dates -- not characters
- Extract the month from ***dateYr*** and save it to a new vector

- ***format()*** is a generic R function that can format all types of values
- ***format="%b"*** gives the abbreviate month (e.g., Jan, Feb),
  - *Note: **format="%B"** gives the full month name (e.g., January, February)*
- Save the months vector to ***weatherData*** as the column ***month***

```
#### Part 3: Creating a month column in the data frame
dates = as.Date(weatherData[,"dateYr"]); # save the date column to a vector
months = format(dates, format="%b");     # extract the month -- save to vector
weatherData[,"month"] = months;          # save months to data frame as new column

# The above three lines could be written as:
# weatherData[,"month"]= format.Date(weatherData[,"dateYr"], format="%b");
```

Now we have a new column called ***month*** in ***weatherData.***

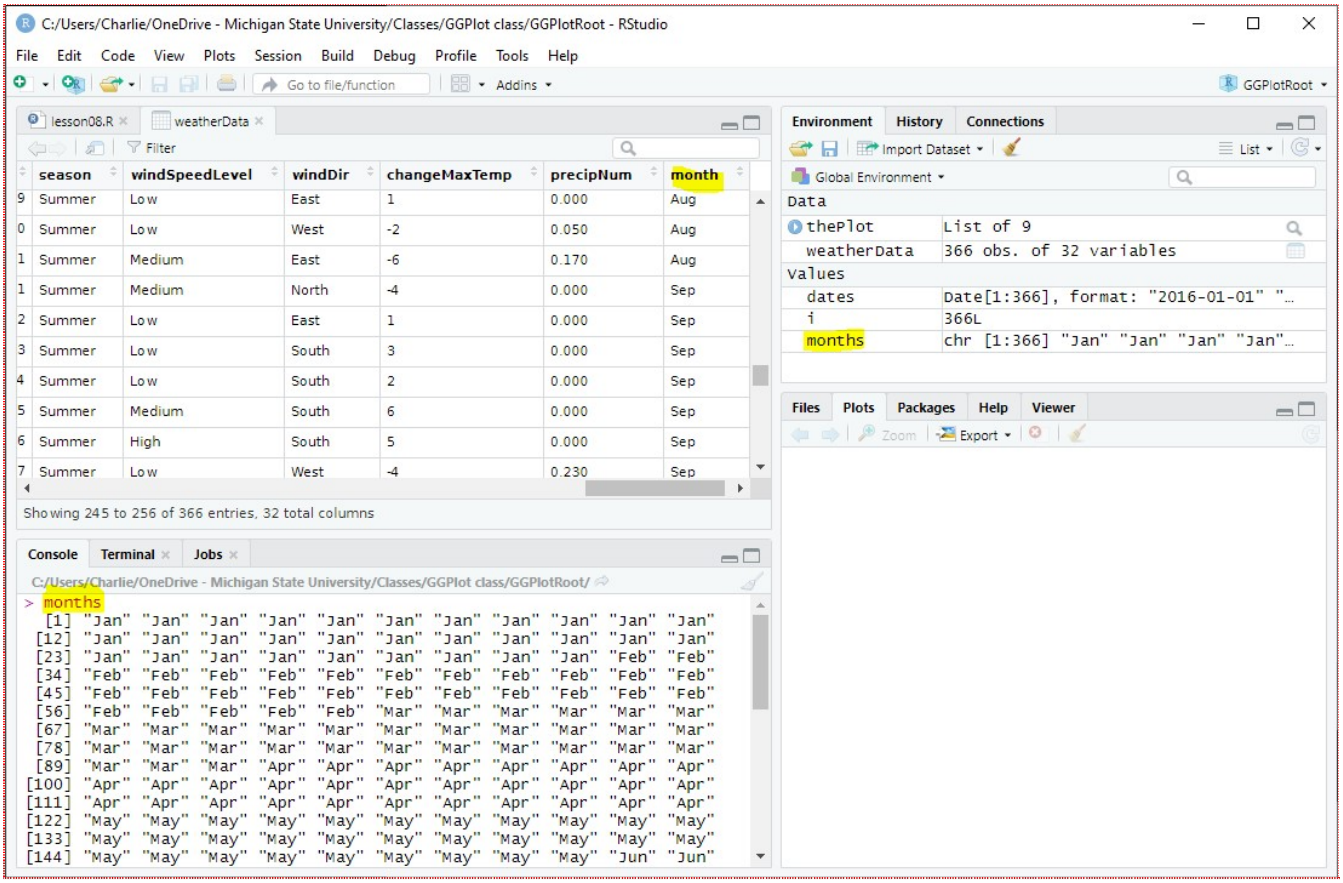

Fig 4: The month indexes the month for each row in the column

4.2 - Plotting by month

We will use the same code as above to make a barplot of ***precipitation*** for each ***month.***

```
#### Part 4: Plot precipitation for each month
thePlot = ggplot(data=weatherData) +
geom_col(mapping=aes(x=month, y=precipNum),
                     width=0.7) +
```

```
        theme_bw() +
        labs(title = "Monthly precipitation",
             subtitle = "Lansing, Michigan: 2016",
             x = "Month",
             y = "Precipitation (inches)");
plot(thePlot);
```

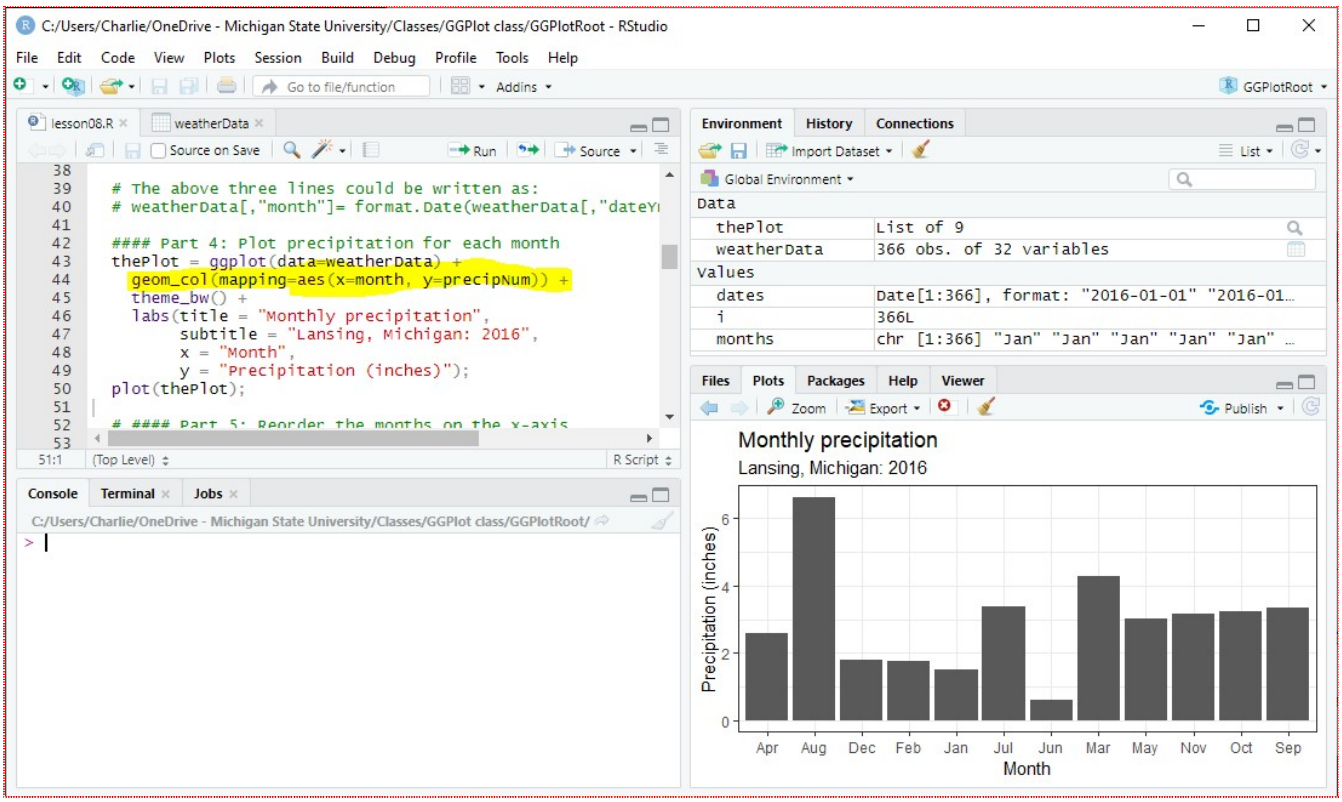

Fig 5: Precipitation plotted for each month -- in alphabetical order

4.3 - Ordering the months correctly

GGPlot saw the months as 12 categories and, not so helpfully in this case, put them in alphabetical order (*fig 5*). To put the month in the more familiar order, we will use the ***scale_x_discrete()*** component. *Note: there are analogous components for the y-axis and continuous values -- **scale_y_discrete()**, **scale_x_continuous()**, and **scale_y_continuous()**.*

The parameter in ***scale_x_discrete()*** we need to set is ***limits***. On the ***GGPlot help page for scale_discrete()***, ***limits*** is defined as "a character vector that defines [all] *possible values* of the scale and their order."

We want a character vector of the months (i.e., **month[1]** = "**Jan**", **month[2]**="**Feb**"...)-- we could create our own but already exists a month vector predefined in R called ***month.abb***.

```
#### Part 5: Reorder the months on the x-axis
thePlot = ggplot(data=weatherData) +
          geom_col(mapping=aes(x=month, y=precipNum),
                   width=0.4) +
```

```
        scale_x_discrete(limits = month.abb) +
        theme_bw() +
        labs(title = "Monthly precipitation",
             subtitle = "Lansing, Michigan: 2016",
             x = "Month",
             y = "Precipitation (inches)");
plot(thePlot);
```

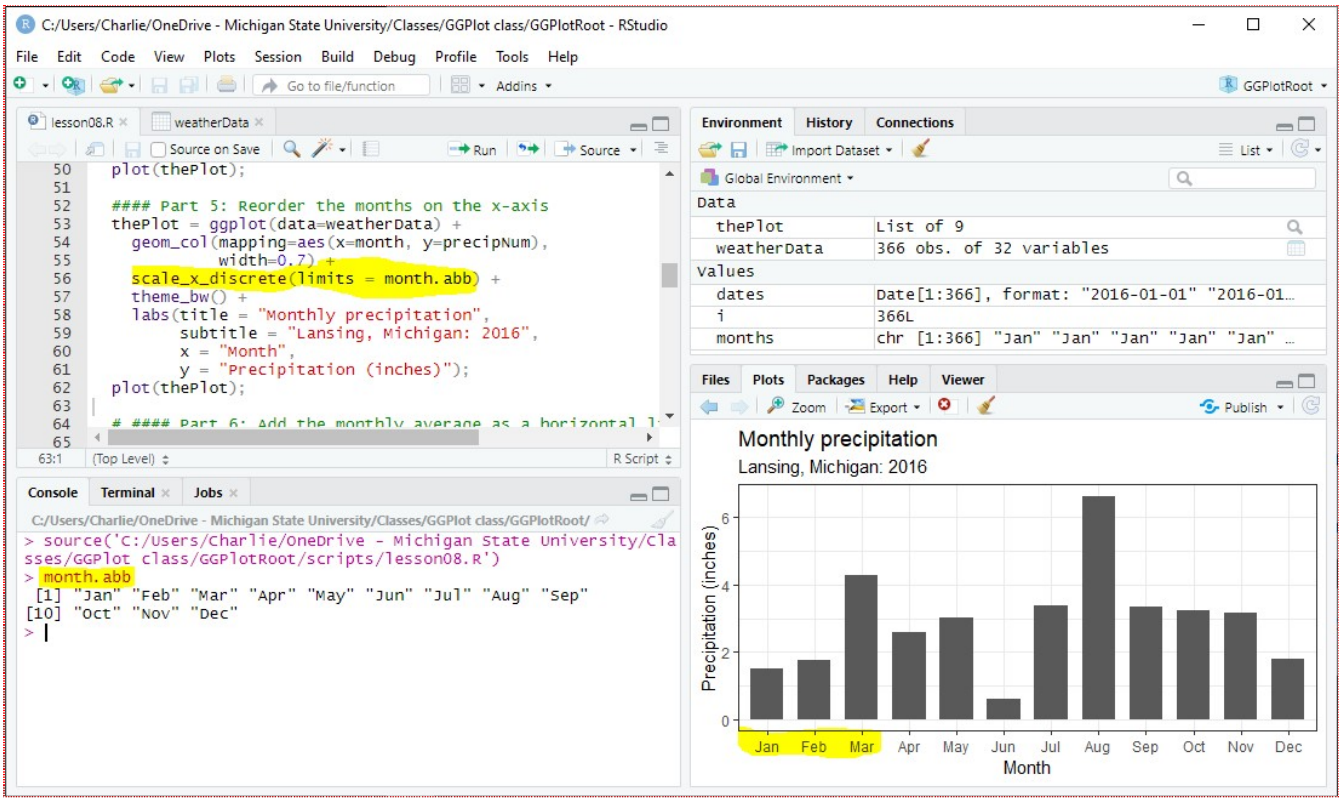

Fig 6: Precipitation for each month -- with the months in order!

5 - Adding lines to a plot

It would be nice to see average precipitation on the plot -- so we will add a horizontal line at the value of the average precipitation using the ***geom_hline()*** component.

First, we need to find the monthly average. I am going to cheat a bit and just add up all **366** precipitation values in the ***precipNum*** column, using the ***sum()*** function and then divide the sum by **12**:

```
monthlyAvg = sum(weatherData$precipNum)/12;
```

We want to add a horizontal line that intercepts the ***monthlyAvg*** point. In the mapping for ***geom_hline()***, we use the ***yintercept*** to map the ***monthlyAvg*** point to a line on the plot.

```
#### Part 6: Add the monthly average as a horizontal line
monthlyAvg = sum(weatherData$precipNum)/12;

thePlot = ggplot(data=weatherData) +
```

```
          geom_col(mapping=aes(x=month, y=precipNum),
                   width=0.5) +
          scale_x_discrete(limits = month.abb) +
          geom_hline(mapping = aes( yintercept = monthlyAvg ),
                     color="red",
                     size=1.5,
                     linetype=2) +
          theme_bw() +
          labs(title = "Monthly precipitation",
               subtitle = "Lansing, Michigan: 2016",
               x = "Month",
               y = "Precipitation (inches)");
plot(thePlot);
```

*Note: We also modified the **color**, **size**, and **linetype** parameters in **geom_hline()**.*

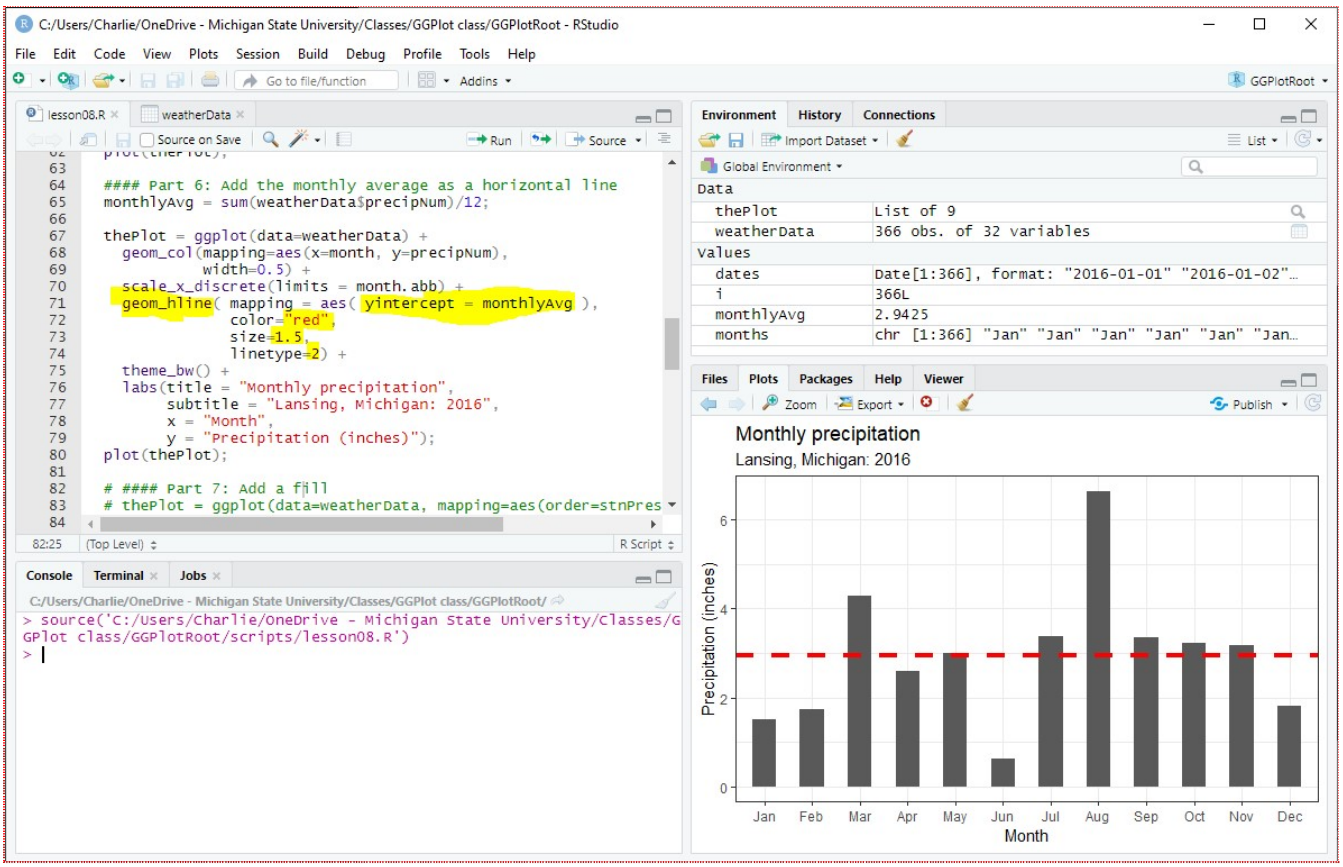

Fig 7: Adding a horizontal line to the plot

6 - Coloring by variable

We can add more information to the bar plots by using a ***fill*** color to represent another variable (in this case: atmospheric pressure). We will ***fill*** the bar in the barplot using the data from the ***stnPressure*** column (atmospheric pressure).

fill maps the data in ***stnPressure*** to the bars on the plot, so ***fill*** is added to GGPlot as a ***mapping***

parameter.

```
#### Part 7: Add a fill
thePlot = ggplot(data=weatherData) +
          geom_col(mapping=aes(x=month, y=precipNum, fill=stnPressure),
                   width=0.5) +
          scale_x_discrete(limits = month.abb) +
          geom_hline(mapping = aes( yintercept= sum(precipNum)/12 ),
                     color="red",
                     size=2,
                     linetype=2) +
          theme_bw() +
          labs(title = "Monthly precipitation",
               subtitle = "Lansing, Michigan: 2016",
               x = "Month",
               y = "Precipitation (inches)");
plot(thePlot);
```

*Note: GGPlot will automatically produce a legend when you use **fill** as a **mapping** parameter.*

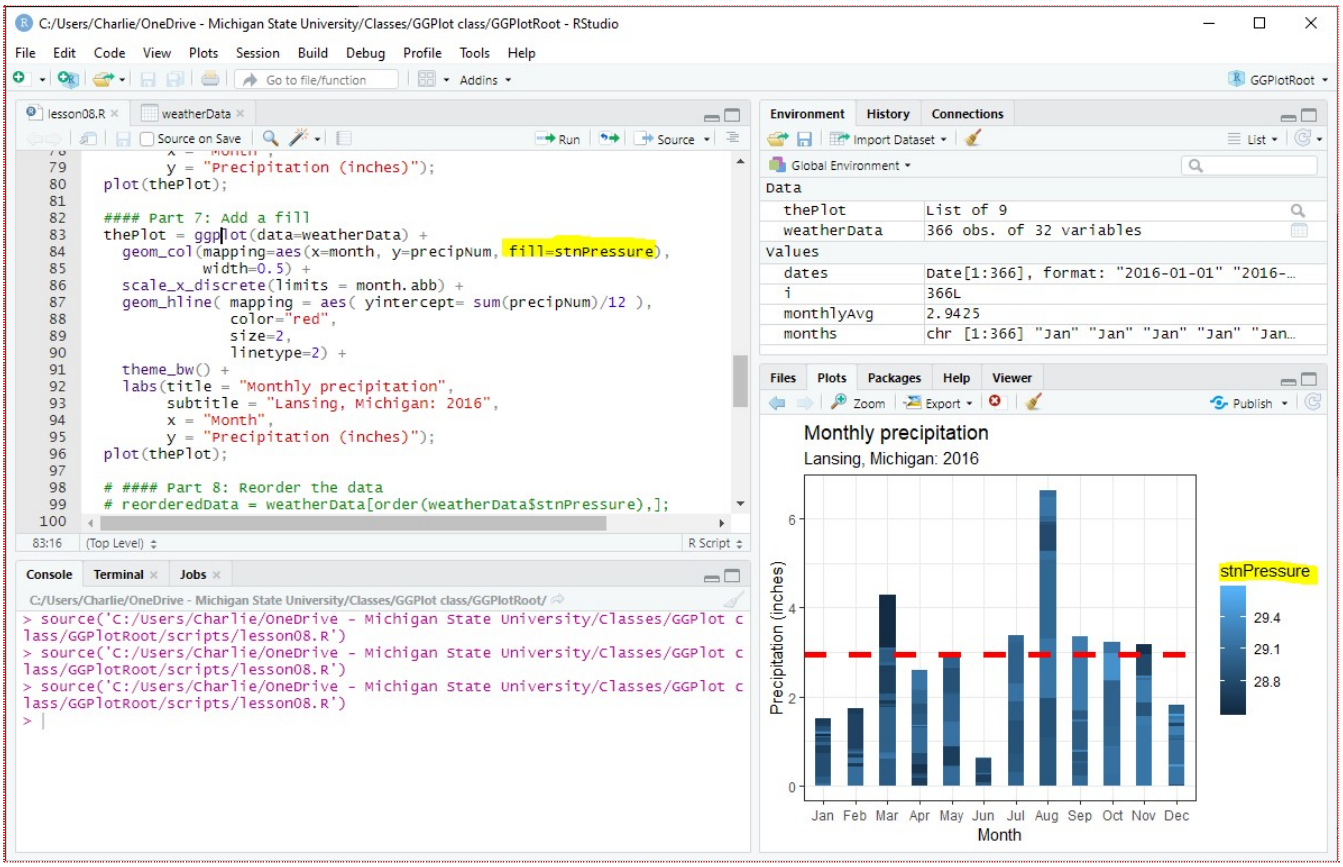

Fig 8: Adding a fill to the bar plot.

6.1 - Reordering the data frame

GGPlot generated a legend and produced a color chart for the range of values in ***stnPressure***. However, the color mapping on the bars is a little confusing. GGplot maps ***stnPressure*** in order by date. This is because the rows of the data frame go in order of date. So, the top of March's bar (dark blue) has the color for the ***stnPressure*** on March 1 and the bottom (medium blue) has the color for the ***stnPressure*** on March 31.

In would be more helpful if ***fill*** was presented in the order of the values in ***stnPressure*** instead of ***dates***.

There is no easy way to reorder the plot by the ***stnPressure*** values -- every solution involves reordering the data frame so the rows go in order of increasing values of ***stnPressure***.

So, we will reorder the rows of the ***weatherData*** data frame using ***order()***:

```
reorderedData = weatherData[order(weatherData$stnPressure), ];
```

What the line of code says is:

- take the ***weatherData*** data frame
- reorder the rows in ***weatherData*** by the values in ***stnPressure***
- save the reordered data frame to ***reorderedData***

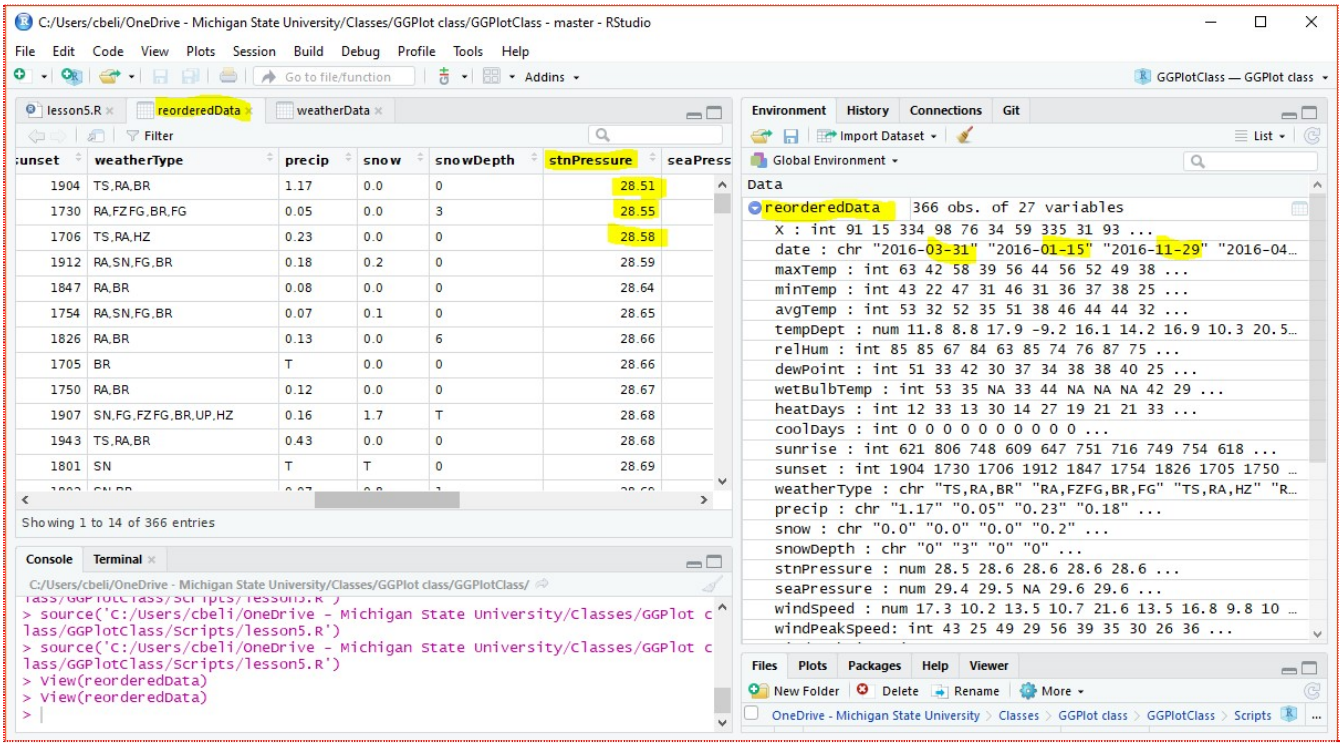

Fig 9: The reordered data frame

The rows of the new data frame go in order of ***stnPressure*** instead of ***date***. Notice that **March 31** is the first date and there was **1.17** inches of rain that day -- not a coincidence that is was also the day with the lowest ***stnPressure***!

6.2 - Plotting with reordered dataframe

Now we will use ***reorderedData*** instead of ***weatherData***:

```
#### Part 8: Reorder the data
reorderedData = weatherData[order(weatherData$stnPressure),];
```

```
3
4  thePlot = ggplot(data=reorderedData) +
5            geom_col(mapping=aes(x=month, y=precipNum, fill=stnPressure),
6                     width=0.5) +
7            scale_x_discrete(limits = month.abb) +
8            geom_hline(mapping = aes( yintercept= sum(precipNum)/12 ),
9                       color="red",
10                      size=1.5,
11                      linetype=2) +
12           theme_bw() +
13           labs(title = "Monthly precipitation",
14                subtitle = "Lansing, Michigan: 2016",
15                x = "Month",
16                y = "Precipitation (inches)",
17                fill= "Pressure");
18 plot(thePlot);
```

*Note: I also changed the legend title using the **fill** parameter in **labs().***

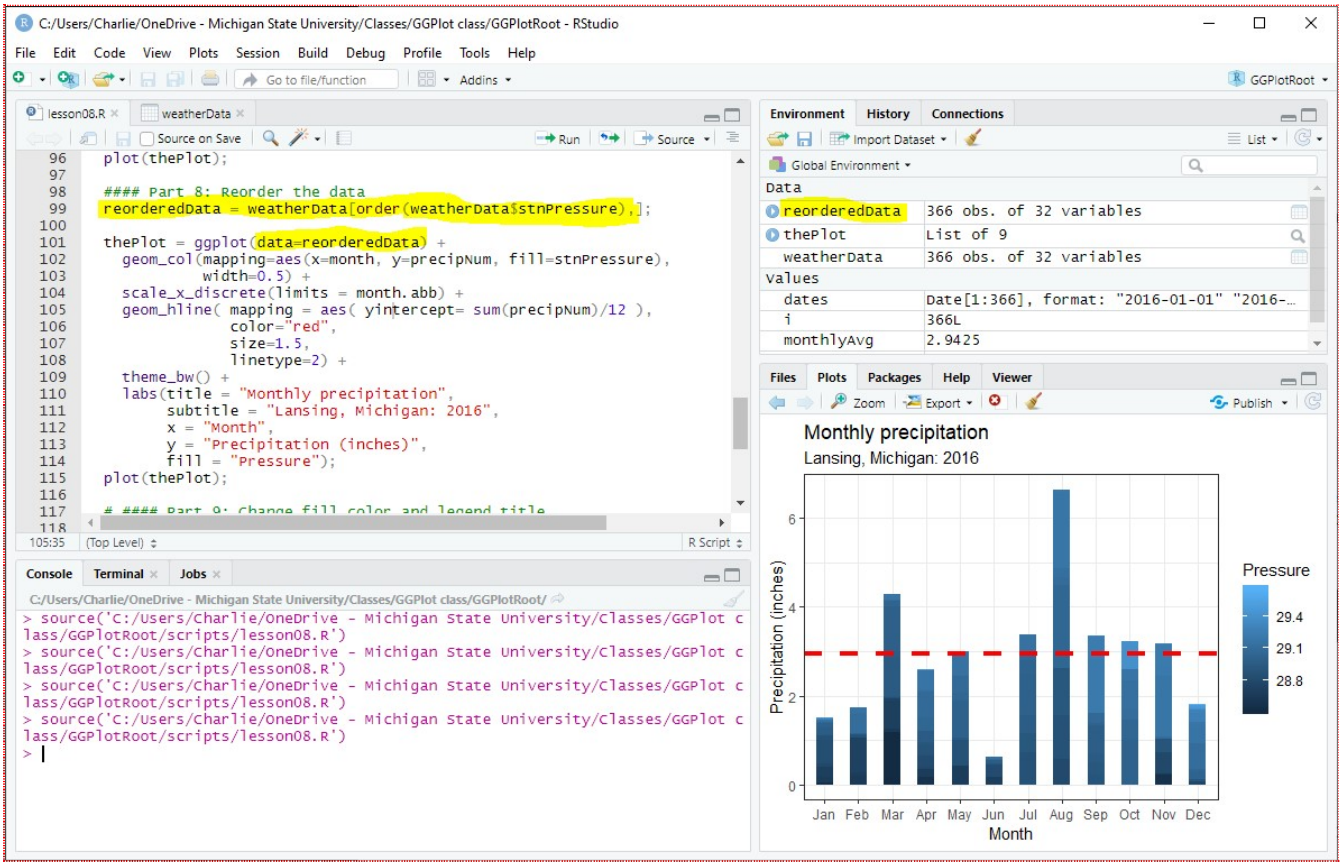

*Fig 10: The bar plot reflects the numeric ordering of **stnPressure***

7 - Legends and mapping

The default ***fill*** is a gradient from dark blue (low ***precip*** values) to light blue (high ***precip*** values). We can

define our own gradient to use for ***fill***. We use the ***scale_fill_gradient()*** component to define the gradient color range. The two parameters we need to set are ***low*** and ***high*** (i.e., the two ends of the gradient).

You can use color names for ***low*** and ***high*** but I decided to use **rgb** (red-green-blue) colors:

```
scale_fill_gradient(low = rgb(red=1, green=0.5, blue=0),
                    high = rgb(red=0, green=0, blue=1))
```

RGB colors represent the three *primary light colors:* red, green, and blue.

The ***low*** I used is all red light and half green light (this makes orange): **rgb(red=1, green=0.5, blue=0)** , and ***high*** is all blue light: **rgb(red=0, green=0, blue=1)**.

*Note: I also change the horizontal line to a dark green **rgb(red=0, green=0.5, blue=0)** to give it better contrast with the boxplots.*

```
#### Part 9: Change fill color and legend title
reorderedData = weatherData[order(weatherData$stnPressure),];
thePlot = ggplot(data=reorderedData) +
          geom_col(mapping=aes(x=month, y=precipNum, fill=stnPressure),
                   width=0.4) +
          scale_x_discrete(limits = month.abb) +
          geom_hline(mapping = aes( yintercept= sum(precipNum)/12 ),
                     color= rgb(red=0, green=0.5, blue=0),
                     size=2,
                     linetype=2) +
          scale_fill_gradient(low = rgb(red=1, green=0.5, blue=0),    # orange

                              high = rgb(red=0, green=0, blue=1)) +  # blue
          theme_bw() +
          labs(title = "Monthly precipitation",
               subtitle = "Lansing, Michigan: 2016",
               x = "Month",
               y = "Precipitation (inches)",
               fill = "Pressure");
plot(thePlot);
```

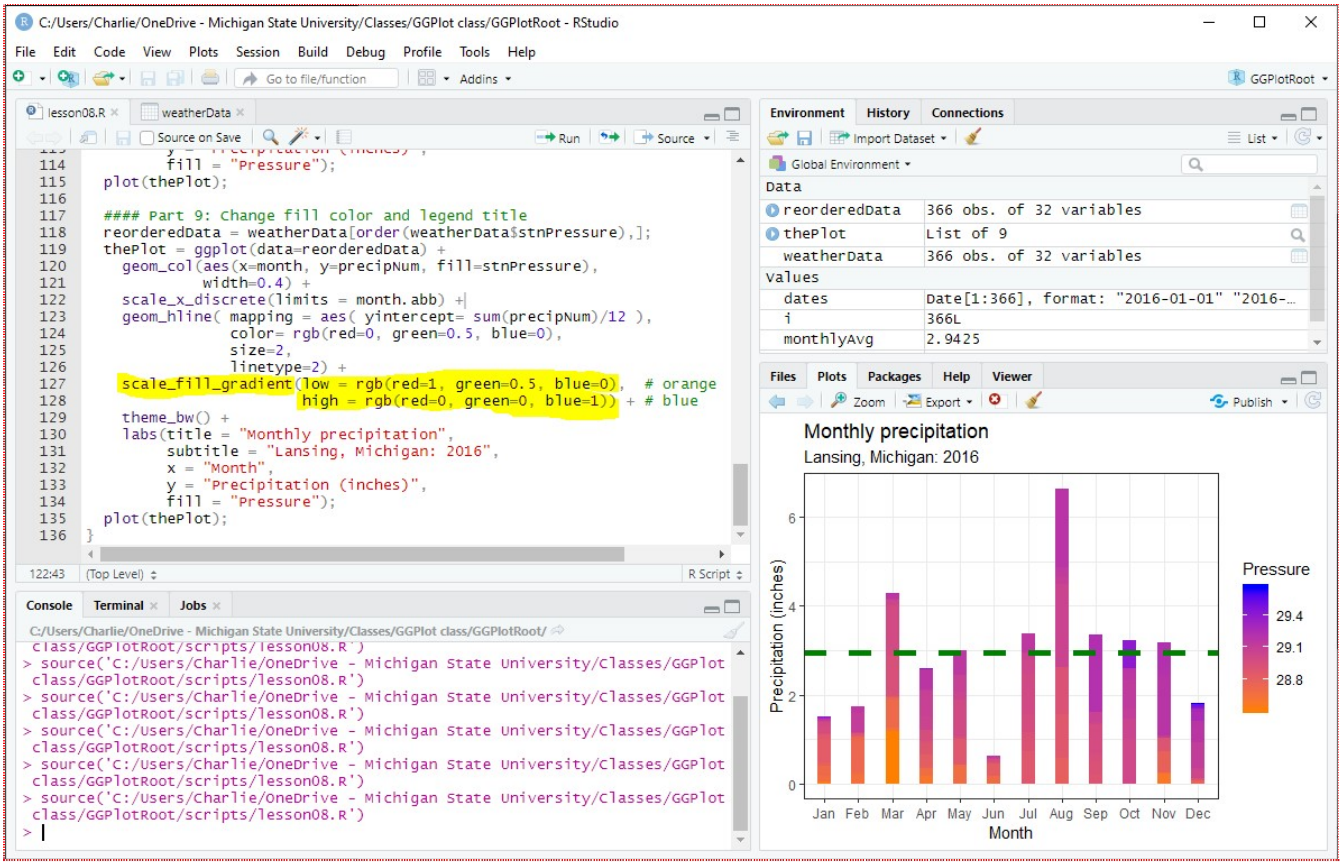

Fig 11: Adding a color gradient to the bar plot fill.

This plot shows that, while March did not have the most rain, it had the most severe weather (bright orange color representing the lowest pressure). And, as expected, rain falls tend to be greater when there is lower pressure.

8 - Application

For this application, we will be using the ***heatDays*** and ***coolDays*** columns in ***weatherData***. These columns measure how much heating or cooling is needed on a day to make a place comfortable. So, on a cold winter day, ***coolDays*** would be **zero** and ***heatDays*** would be high. On a hot summer day, ***heatDays*** would be zero and ***coolDays*** would be high.

Specifically, the number represents how far the average temperature of the day is from 65 (which was determined to be the most comfortable temperature).

1. Create a script file in your GGPlot Class Project called ***app08.r***.
2. Create a bar plot of Cooling Days (***coolDays***) for each month
   A. Add a title, and put appropriate labels on the axes.
   B. Change the bar width to 0.6
   C. Create a vector that consists of the first two values in the ***weatherType*** column (i.e., the new vector will have at most one weather condition for the day).
   D. Use the vector created in the previous step to fill the bars (for fun, you can try to just use ***weatherType*** as the fill and see what happens...)
   E. Customize the colors of the 8 weather conditions (one of the conditions is empty). Do this using component ***scale_fill_manual*** and the subcomponent ***values***, which will be set a to a vector. *Note: this is the discrete version of the component* ***scale_fill_gradient****.*

3. Create a bar plot of Heating Days (***heatDays***) for each month
   A. Add a title, and put appropriate labels on the axes.
   B. Change the bar width to 0.6
   C. Create a vector that consists of the first two values in the ***weatherType*** column.
   D. Add a horizontal line to the plot that represent the sum of all ***coolDays*** for the year (it should be in the 800s)
   E. Add a label (or, annotate) that describes what the line represents.
4. Combine Heating and Cooling Days into one barplot (i.e., side-by-side)
   A. Add a title, and put appropriate labels on the axes.
   B. Change the bar **widths** of both to **0.4**
   C. Change the bar **colors** (**red** for ***heatDays*** , **blue** for ***coolDays***)

9 - Side-by-side bar plots

This is one of the more unusually challenging topics in GGPlot. The most common solution found online is to reshape your data frame using the ***dplyr*** package. This author sees this technique as overkill for what is needed.

The second technique is to use the ***position_nudge*** attribute in the ***position*** subcomponent in the graphing component, which in this lesson is ***geom_col()***. For the example in this lesson this would look like:

```
thePlot = ggplot(data=weatherData) +
          geom_col(mapping=aes(x=month, y=heatDays),
                   fill = "red",
                   width=0.6,
                   position=position_nudge(x=-0.5)) +
          scale_x_discrete(limits = month.abb) +
          theme_bw() +
          labs(title = "Heating and Cooling Days",
               subtitle = "Lansing, Michigan: 2016",
               x = "Month",
               y = "Cumulative Degrees (F)");
plot(thePlot);
```

This technique successfully moves the bars 0.7 units to the left but it also changes the way the bar is presented. What is happening is that the bar has been changed from stack to dodge. This author has not found a way to keep the bars stacks while nudging the position and assumes this is a bug in GGPlot.

The third technique, and the one this authors prefer because it universally works and does not require data frame manipulation is to make the x-axis numeric and then move it over in the mapping. To do this, you need to:

1. Cast the x-axis variable as a factor
2. Cast the factor as numeric
3. Add or subtract values in the mapping

To switch ***month*** into a numeric vector (this combines step 1 and 2):

```
monthNum = as.numeric(factor(weatherData$month, levels=unique(weatherData$month)));
```

To shift the bar over in the mapping (in this case, left by 0.6 units):

```
        geom_col(mapping=aes(x=monthNum -0.6, y=heatDays),
                 fill = "red",
                 width=0.6) +
```