01-09: Text Plots and Legend Modifications

1 - Purpose

- Use data values as points in a plot
- use grep to find index values
- customize gradients
- manipulate a legend

2 - Concepts

3 - Get data

For this lesson, we will work from the data file LansingNOAA2016-3.csv.

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

4 - A simple scatterplot

We will start with a simple scatterplot of Humidity (***relHum***) vs Temperature (***avgTemp***) using the component ***geom_point()***.

The component ***theme_bw()*** creates a black and white themed plot with a border around the whole plot area. The ***theme()*** subcomponents ***panel.grid.major*** and ***panel.grid.minor*** are used to remove the grid lines -- by setting them to a blank element, ***element_blank()***.

```
#### Part 1: Create a Humidity vs. Temperature scatterplot
thePlot = ggplot(data=weatherData) +
          geom_point(mapping=aes(x=avgTemp, y=relHum)) +
          theme_bw() +
          theme(panel.grid.major = element_blank(),
                panel.grid.minor = element_blank()) +
          labs(title = "Humidity vs. Temperature",
               subtitle = "Lansing, Michigan: 2016",
               x = "Degrees (Fahrenheit)",
               y = "Relative Humidity");
plot(thePlot);
```

*Reminder: **theme()** changes need to occur after **theme_bw()** is set because **theme_bw()** resets all*

***theme()** subcomponents.*

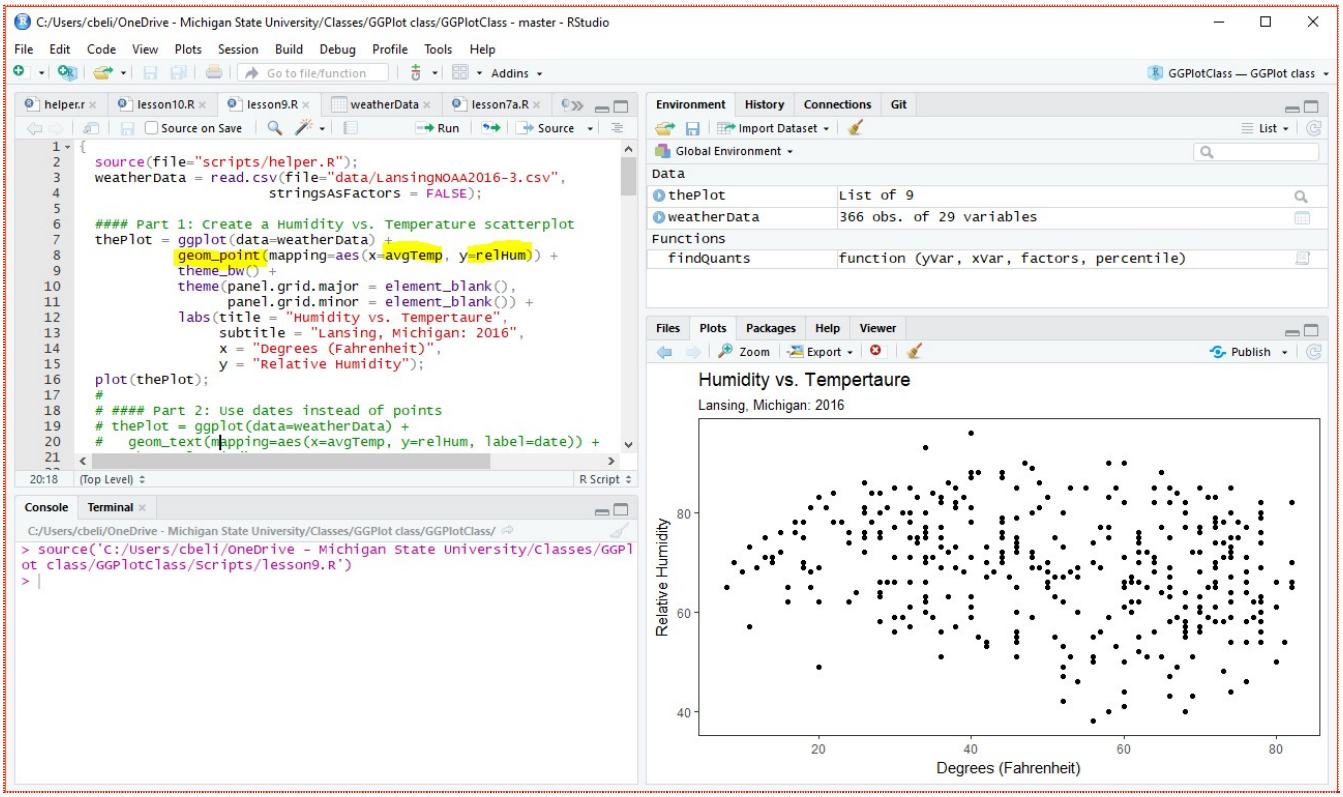

Fig 1: Scatterplot with clean background and full border around plot

5 - Using dates instead of points

We use the ***geom_text()*** component instead of ***geom_point()*** to place text values on the plot instead of points. In the ***mapping*** for ***geom_text()***, the ***label*** parameter maps ***date*** to the text on the plot. In this case, ***relHum*** vs ***avgTemp*** is plotted using the corresponding values from the ***date*** column instead of points.

```
geom_text(mapping=aes(x=avgTemp, y=relHum, label=date))
```

Looking at the full code for the plot:

```
#### Part 2: Use dates instead of points
thePlot = ggplot(data=weatherData) +
          geom_text(mapping=aes(x=avgTemp, y=relHum, label=date)) +
          theme_bw() +
          theme(panel.grid.major = element_blank(),
                panel.grid.minor = element_blank()) +
          labs(title = "Humidity vs. Temperature",
               subtitle = "Lansing, Michigan: 2016",
               x = "Degrees (Fahrenheit)",
               y = "Relative Humidity");
plot(thePlot);
```

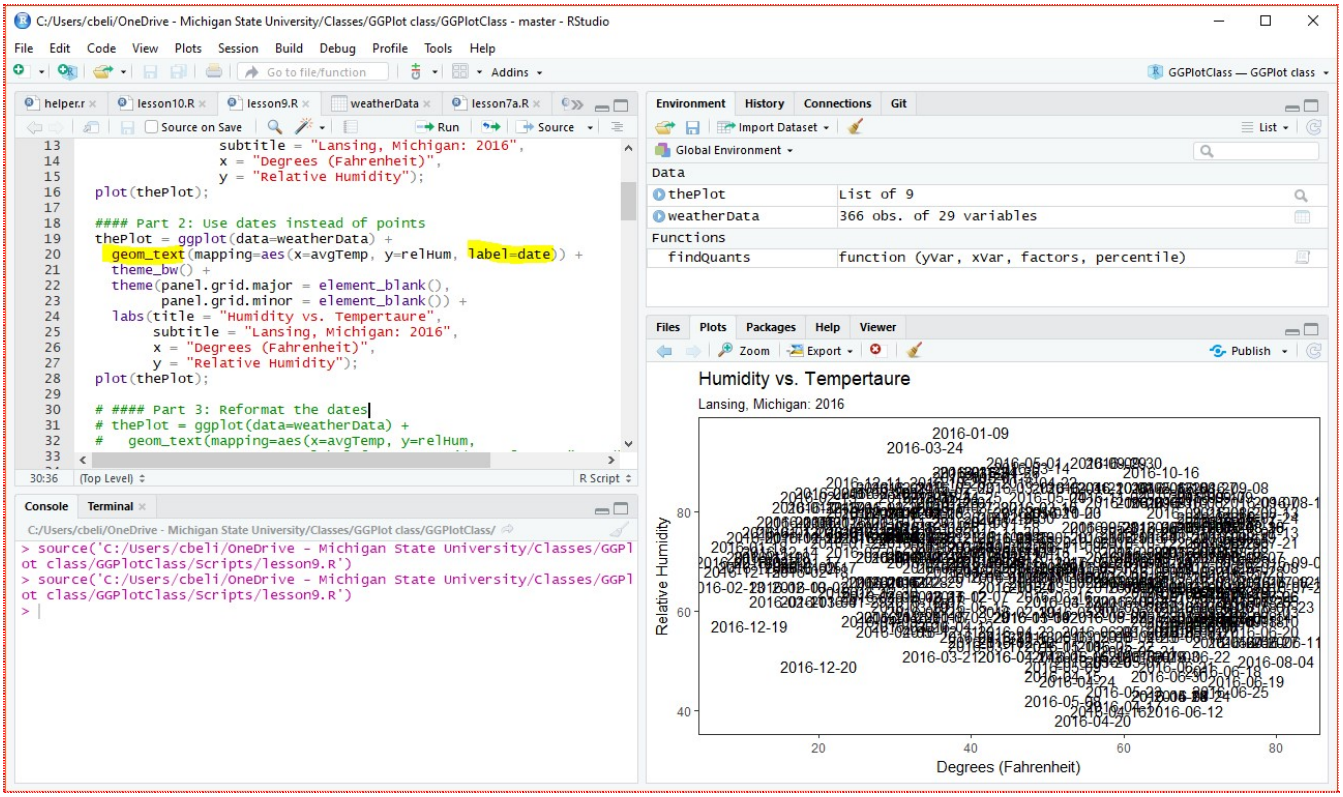

Fig 2: A scatterplot of humidity that uses labels (date values) instead of points

5.1 - Formatting dates

The overlapping dates makes the plot unreadable, so we will change the ***size*** of the text to **2.5** (***size*** is in millimeters) to save space and make the plot more readable:

```
#### Part 3: Reformat the dates
thePlot = ggplot(data=weatherData) +
          geom_text(mapping=aes(x=avgTemp, y=relHum, label=date),
                    color="darkgreen",
                    size=2.5) +   # change size
          theme_bw() +
          theme(panel.grid.major = element_blank(),
                panel.grid.minor = element_blank()) +
          labs(title = "Humidity vs. Temperature",
               subtitle = "Lansing, Michigan: 2016",
               x = "Degrees (Fahrenheit)",
               y = "Relative Humidity");
plot(thePlot);
```

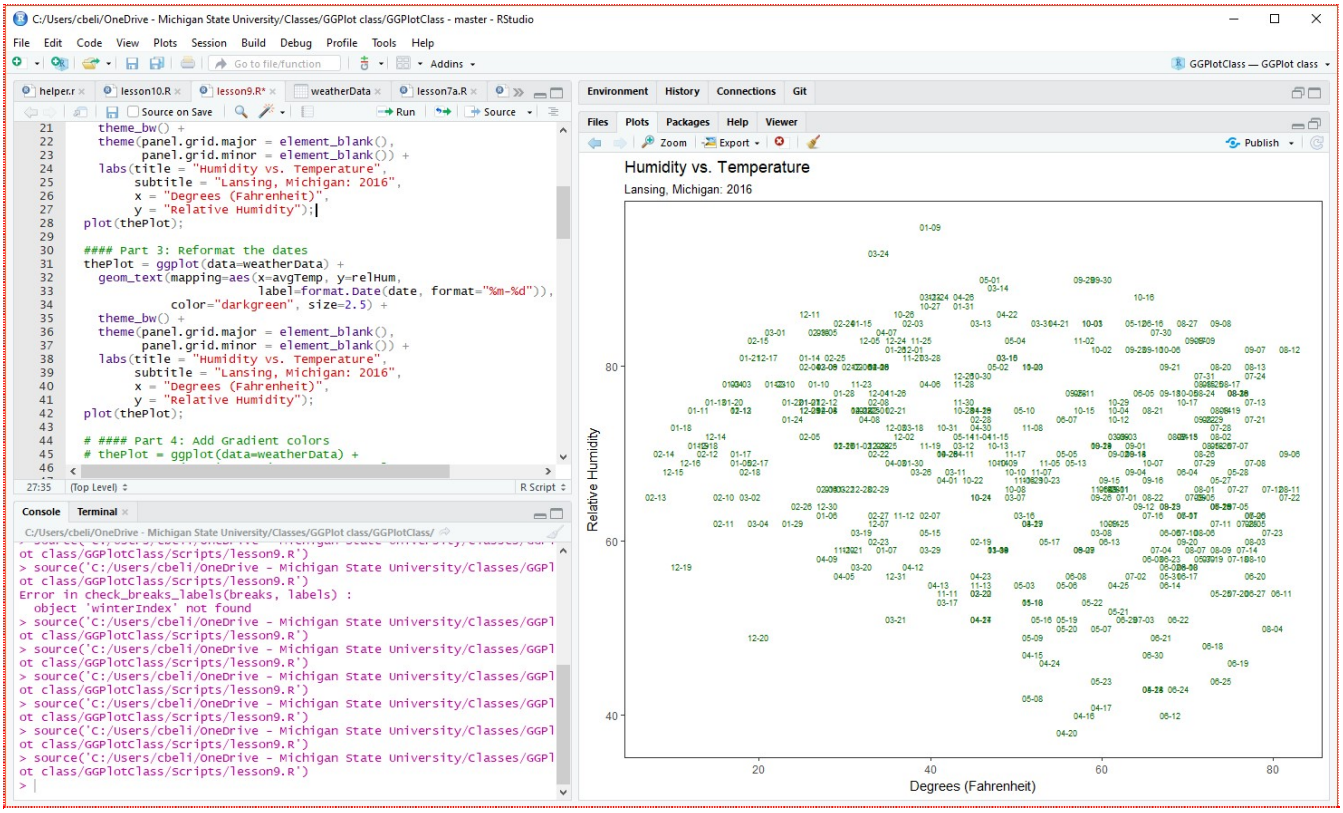

Fig 3: Formatting and resizing the date values in the plot to remove some of the overlap

6 - Gradients

In the previous example, we changed the **color** of the **date** values to **darkgreen**. However, we can use multiple colors to better represent where the date values are in the year. For instance, we can make winter value more blue and summer values more red. In other words, we want to map ***color*** to the data. To do this, we need to move ***color*** inside the ***mapping*** for ***geom_text()***.

We need to define vector with 366 colors. For this example, we will define a vector and GGPlot will add the colors.

The easiest way is to set ***color*** to the sequence **1:366** or, for a more future-proof solution, set ***color*** to **1:nrow(weatherData),** which is equivalent to **1:366**.

```
#### Part 4: Add gradient colors
thePlot = ggplot(data=weatherData) +
          geom_text(mapping=aes(x=avgTemp, y=relHum, color=1:nrow(weatherData),
                                label=date),
                    size=2.5) +
          theme_bw() +
          theme(panel.grid.major = element_blank(),
                panel.grid.minor = element_blank()) +
          labs(title = "Humidity vs. Temperature",
               subtitle = "Lansing, Michigan: 2016",
               x = "Degrees (Fahrenheit)",
```

```
                y = "Relative Humidity");
plot(thePlot);
```

Just like for ***fill***, GGPlot automatically places a legend in the plot when ***color*** is used. The default color gradient goes from dark blue (**01-01**, representing Jan 1) to light blue (**12-31**, representing December 31).

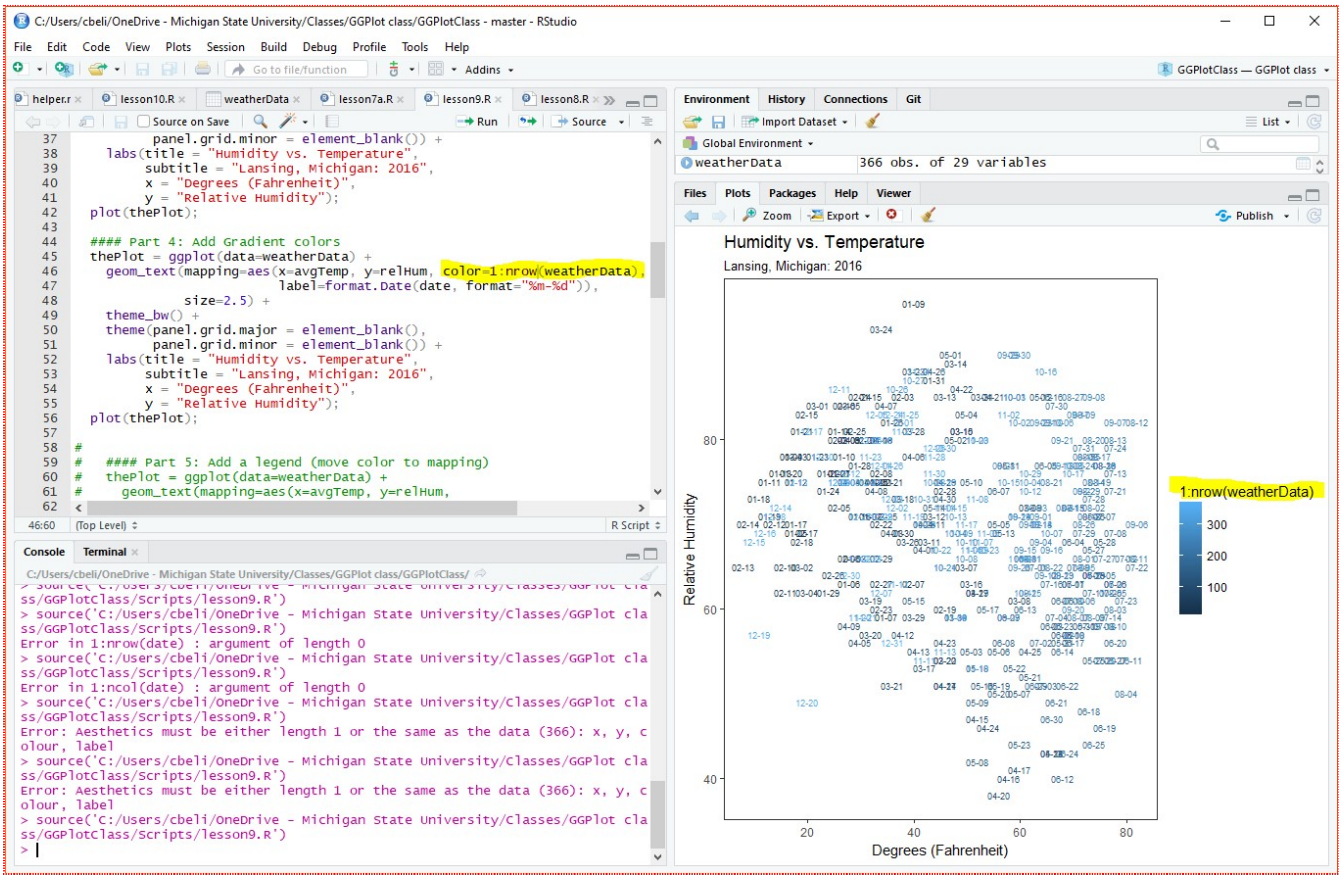

Fig 4: Mapping the data to a gradient color

6.1 - Customizing gradient colors

To customize the gradient color, we will use the ***scale_color_gradientn()*** component. The ***n*** in ***gradientn*** refers to the fact that you can apply as many, or **n**, colors to the gradient as you want.

The default gradient (*fig 4*) was from light blue to dark blue -- it was a two color gradient.

We are going to create a ***colors*** gradient with five colors:

- the first value (**blue**) is the color of the first plotted point (representing Jan 1)
- the last value (**blue**) is the color of the last plotted point (representing Dec 31)
- the colors in between (**orange**, **red**, **green**) are evenly spaced (approximately the beginning of spring, summer, and fall)

Note: I used blue at the beginning and end of a gradient to have a smooth transition between December 31 and January 1.

The following code says **blue** is at both ends (0^{th} and 100^{th} percentile) and **orange**, **red**, and **green** will be evenly spaced (at the 25^{th}, 50^{th} and 75^{th} percentile, respectively).

```
    scale_color_gradientn(colors=c("blue","orange","red","green","blue"))
```

```
1  #### Part 5: Customize gradient colors
2  thePlot = ggplot(data=weatherData) +
3            geom_text(mapping=aes(x=avgTemp, y=relHum, color=1:nrow(weatherData),
4                                  label=date),
5                      size=2.5) +
6            scale_color_gradientn(colors=c("blue","orange","red","green","blue")) +
7            theme_bw() +
8            theme(panel.grid.major = element_blank(),
9                  panel.grid.minor = element_blank()) +
10           labs(title = "Humidity vs. Temperature",
11                subtitle = "Lansing, Michigan: 2016",
12                x = "Degrees (Fahrenheit)",
13                y = "Relative Humidity");
14 plot(thePlot);
```

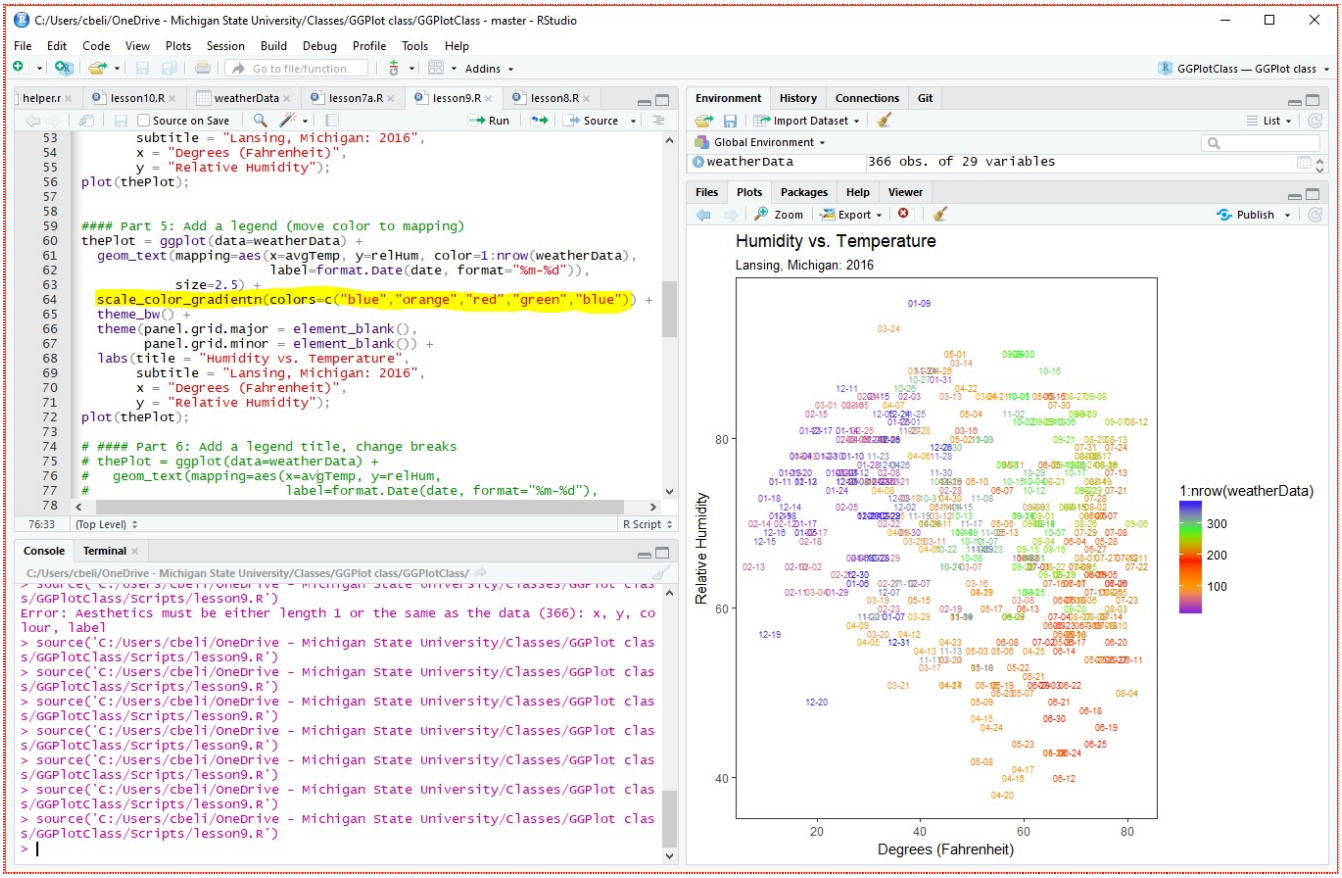

Fig 5: A text plot where the color gradient represent the time of year

7 - Modifying the legend

We are going to make the following *modifications to the legend*:

- *title*: in ***labs()*** component, ***color*** subcomponent -- this is because the legend maps color to the plot

- *height*: in ***theme()*** component, ***legend.key.height*** subcomponent -- uses ***unit()*** with a value and unit type, ***units***
- *tick mark locations*: in ***scale_color_gradientn()*** component, ***breaks*** subcomponent -- creates three tick marks, or ***breaks***
- *labels*: in ***scale_color_gradientn()*** component, ***labels*** subcomponent -- *the number of labels must be the same as the number of tick marks* (in this case, 3)

```
#### Part 6: Customize legend
thePlot = ggplot(data=weatherData) +
          geom_text(mapping=aes(x=avgTemp, y=relHum, color=1:nrow(weatherData),
                                label=date),
                    size=2.5) +
          scale_color_gradientn(colors=c("blue","brown","red","green","blue"),
                                breaks=c(91, 183, 274),  # these are guesses
                                labels=c("Mar-21", "Jun-21", "Sep-21")) +
          theme_bw() +
          theme(panel.grid.major = element_blank(),
                panel.grid.minor = element_blank(),
                legend.key.height = unit(40, units="pt")) +  # height of legend
          labs(title = "Humidity vs. Temperature",
               subtitle = "Lansing, Michigan: 2016",
               x = "Degrees (Fahrenheit)",
               y = "Relative Humidity",
               color = "Dates");            # give a better explanation! title of
    legend
plot(thePlot);
```

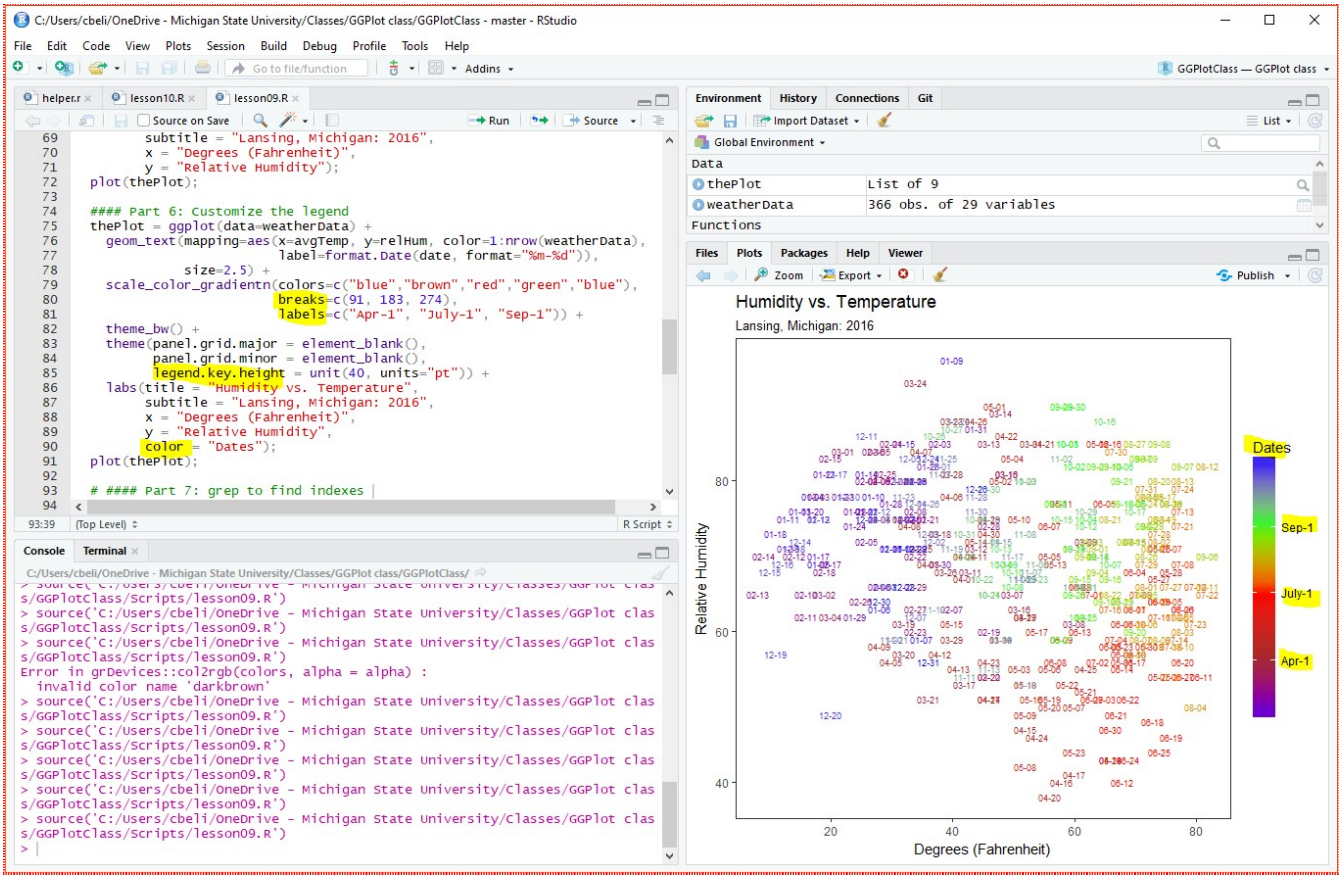

Fig 6: Customizing the legend's title, size, tick marks, and labels

8 - Using grep

differences between which() and grep()

In the last code snippet, I incorrectly guessed that March 21, June 21, and September 21 were, respectively, the days with index numbers 91, 183, and 274.

```
  breaks=c(91, 183, 274),
  labels=c("Mar-21", "Jun-21", "Sep-21")
```

To get the actual index numbers for the days, we can use ***grep()*** to find the index of a vector that contain a certain value.

In the following example, I call ***grep()*** with the vector I want to look at, ***weatherData$date***, and the ***pattern*** I am seeking within that vector. ***grep()*** will return the first index value in the vector that matches the ***pattern***.

```
1 #### Part 7: grep to find indexes
2 springIndex = grep(weatherData$date, pattern="3-21");
3 summerIndex = grep(weatherData$date, pattern="6-21");
4 fallIndex = grep(weatherData$date, pattern="9-21");
5 winterIndex = grep(weatherData$date, pattern="12-21");
```

So, spring starts on day **81**, summer on day **173**, fall on day **265**, and winter on day **356.**

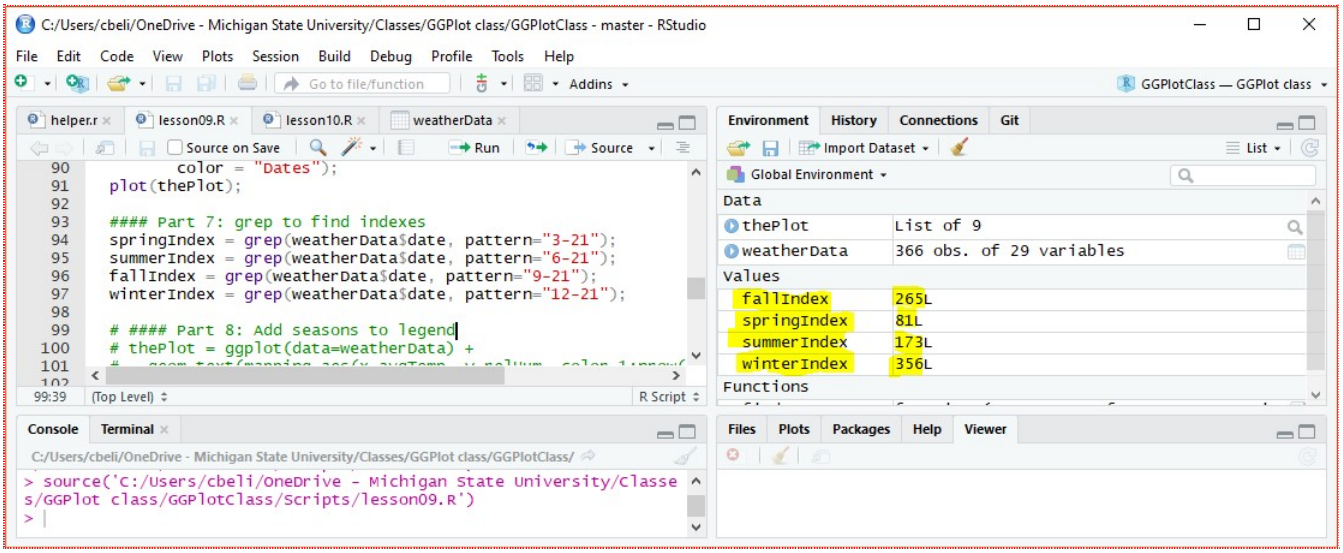

Fig 7: Using grep to get the index values from the beginning of each season

grep() is a very powerful tool for doing pattern recognition within data, most of which is beyond the scope of this class. The power arises from the flexibility of the ***pattern*** parameter, which can be used to do much more complex searches. *Extension: the **pattern** parameter.*

8.1 - Using the index values from grep()

Now we will replace the ***breaks*** values with the ***season*** index values calculated using ***grep()*** and replace the ***labels*** values with the appropriate season:

```
#### Part 8: Add seasons to the legend
thePlot = ggplot(data=weatherData) +
          geom_text(mapping=aes(x=avgTemp, y=relHum, color=1:nrow(weatherData),
                                label=date),
                    size=2.5) +
          scale_color_gradientn(colors=c("blue","brown","red","green","blue"),
                                breaks=c(winterIndex, springIndex,
                                         summerIndex, fallIndex),
                                labels=c("winter","spring",
                                         "summer","fall")) +
          theme_bw() +
          theme(panel.grid.major = element_blank(),
                panel.grid.minor = element_blank(),
                legend.key.height = unit(40, units="pt")) +  # height of legend
          labs(title = "Humidity vs. Temperature",
               subtitle = "Lansing, Michigan: 2016",
               x = "Degrees (Fahrenheit)",
               y = "Relative Humidity",
               color = "Dates");    # title of legend
```

```
20 plot(thePlot);
```

The tick marks are now at the first day of each season and labelled with the season's name.

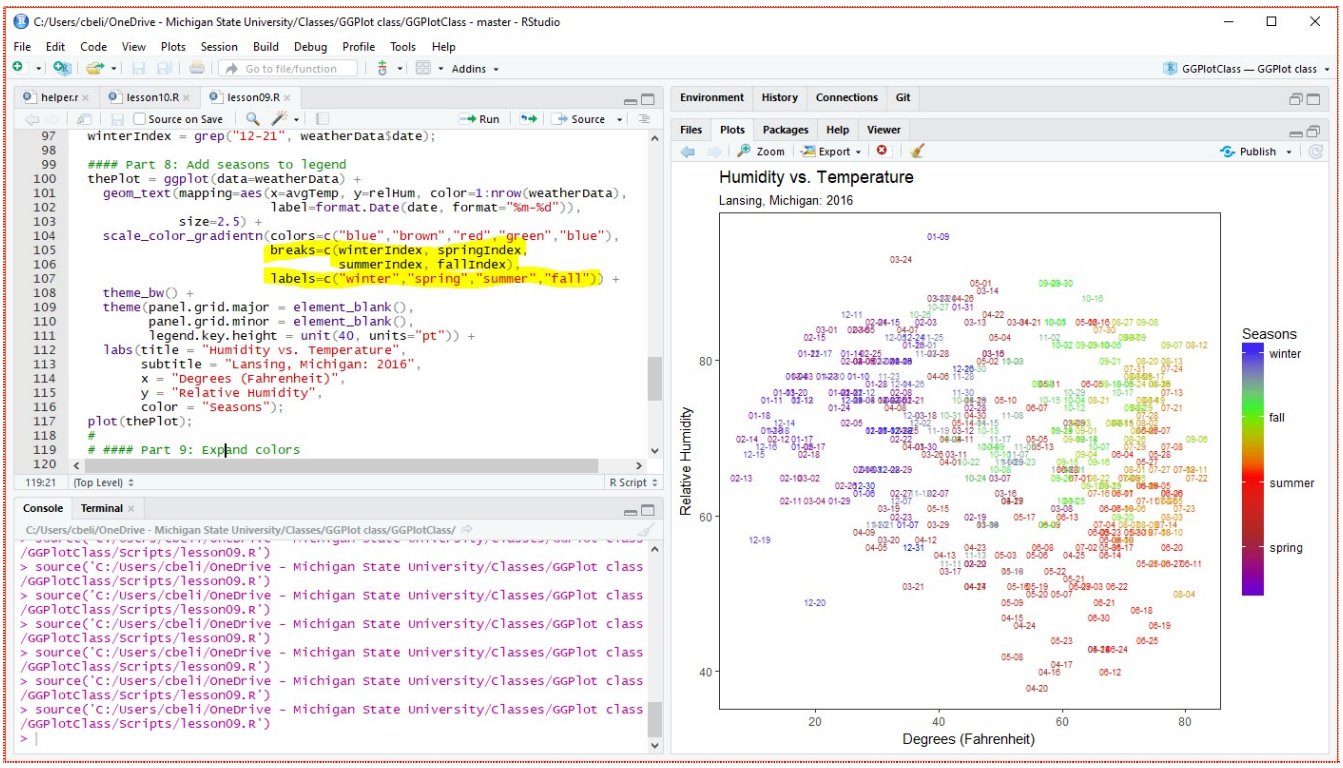

Fig 8: Adding seasons to the legend

8.2 - Changing gradient breaks

The gradient colors do not need to be place equidistant on the legend (in this case, at the 25th, 50th, and 75th percentiles), we can define the location of the gradient colors.

We used five ***colors*** values in ***scale_color_gradientn()***, so we need to define five percentile ***values*** to place the color. The percentile ***values*** are between **0** and **1**.

```
colors=c("blue","brown","red","green","blue"),
values=c(0, 0.2, 0.55, 0.85, 1),
```

In the above code, **blue** is put at both the top and bottom (**0**, **1**), **brown** is put at 20% (**0.2**), **red** at 55% (**0.55**), and **green** at 85% (**0.85**).

```
1 #### Part 9: Change gradient breaks
2 thePlot = ggplot(data=weatherData) +
3           geom_text(mapping=aes(x=avgTemp, y=relHum, color=1:nrow(weatherData),
4                                 label=date),
5                     size=2.5) +
6           scale_color_gradientn(colors=c("blue","brown","red","green","blue"),
7                                 values=c(0, 0.2, 0.55, 0.85, 1),
8                                 breaks=c(winterIndex, springIndex,
```

```
                                        summerIndex, fallIndex),
                                    labels=c("winter","spring","summer","fall")) +
          theme_bw() +
          theme(panel.grid.major = element_blank(),
                panel.grid.minor = element_blank(),
                legend.key.height = unit(25, units="pt")) +
          labs(title = "Humidity vs. Temperature",
               subtitle = "Lansing, Michigan: 2016",
               x = "Degrees (Fahrenheit)",
               y = "Relative Humidity",
               color = "Dates");
plot(thePlot);
```

The gradient colors in the legend have shifted to match the new ***color*** placement.

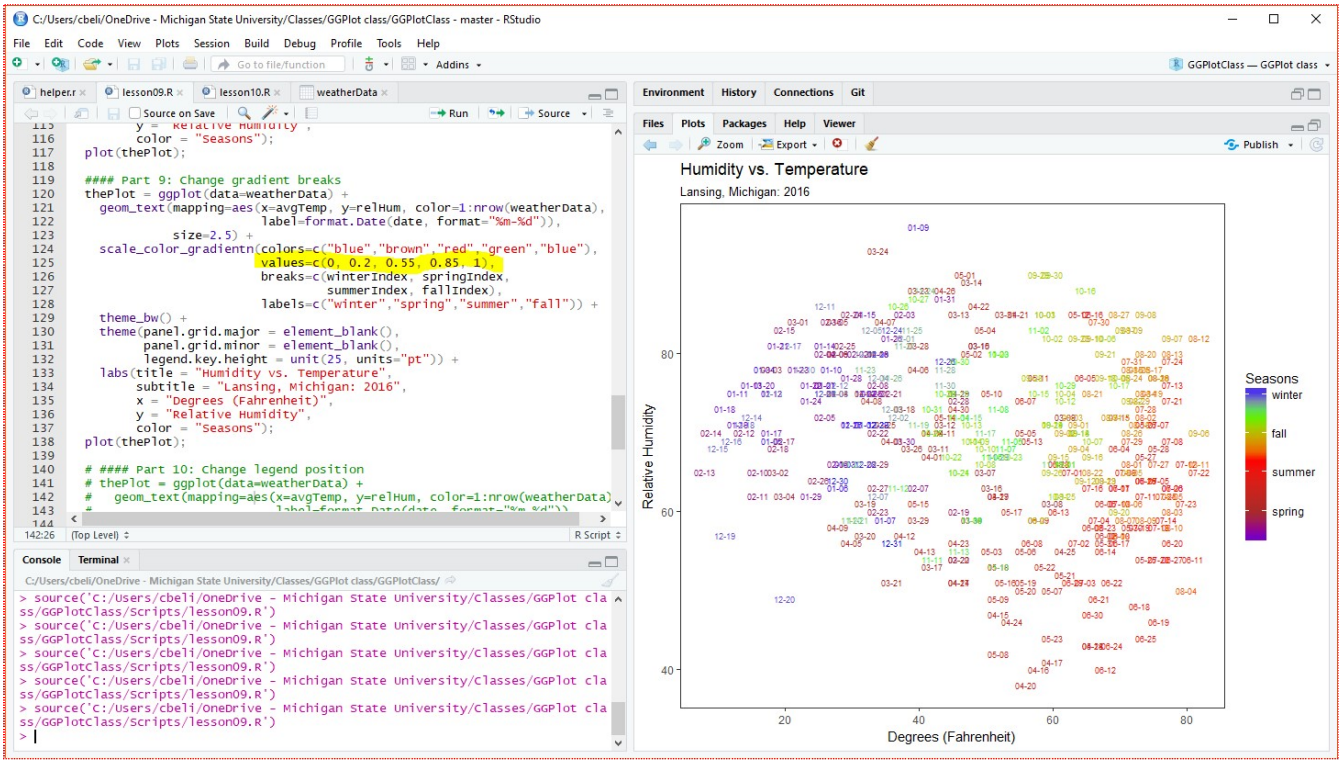

Fig 9: Changing the position of the gradient colors

9 - Legend modifications

We will finish off the lesson by modifying the legend's:

- width
- height
- alignment
- position

All of this can be done in the ***theme()*** component.

```
#### Part 10: Legend modifications
thePlot = ggplot(data=weatherData) +
          geom_text(mapping=aes(x=avgTemp, y=relHum, color=1:nrow(weatherData),
                                label=date),
                    size=2.5) +
          scale_color_gradientn(colors=c("blue","brown","red","green","blue"),
                                values=c(0, 0.2, 0.55, 0.85, 1),
                                breaks=c(winterIndex, springIndex,
                                         summerIndex, fallIndex),
                                labels=c("winter","spring","summer","fall")) +
          theme_bw() +
          theme(panel.grid.major = element_blank(),
                panel.grid.minor = element_blank(),
                legend.key.height = unit(15, units="pt"),  # height
                legend.key.width = unit(40, units="pt"),   # width
                legend.direction = "horizontal",           # alignment
                legend.position = c(0.25, 0.08)) +         # position

          labs(title = "Humidity vs. Temperature",
               subtitle = "Lansing, Michigan: 2016",
               x = "Degrees (Fahrenheit)",
               y = "Relative Humidity",
               color = "");
plot(thePlot);
```

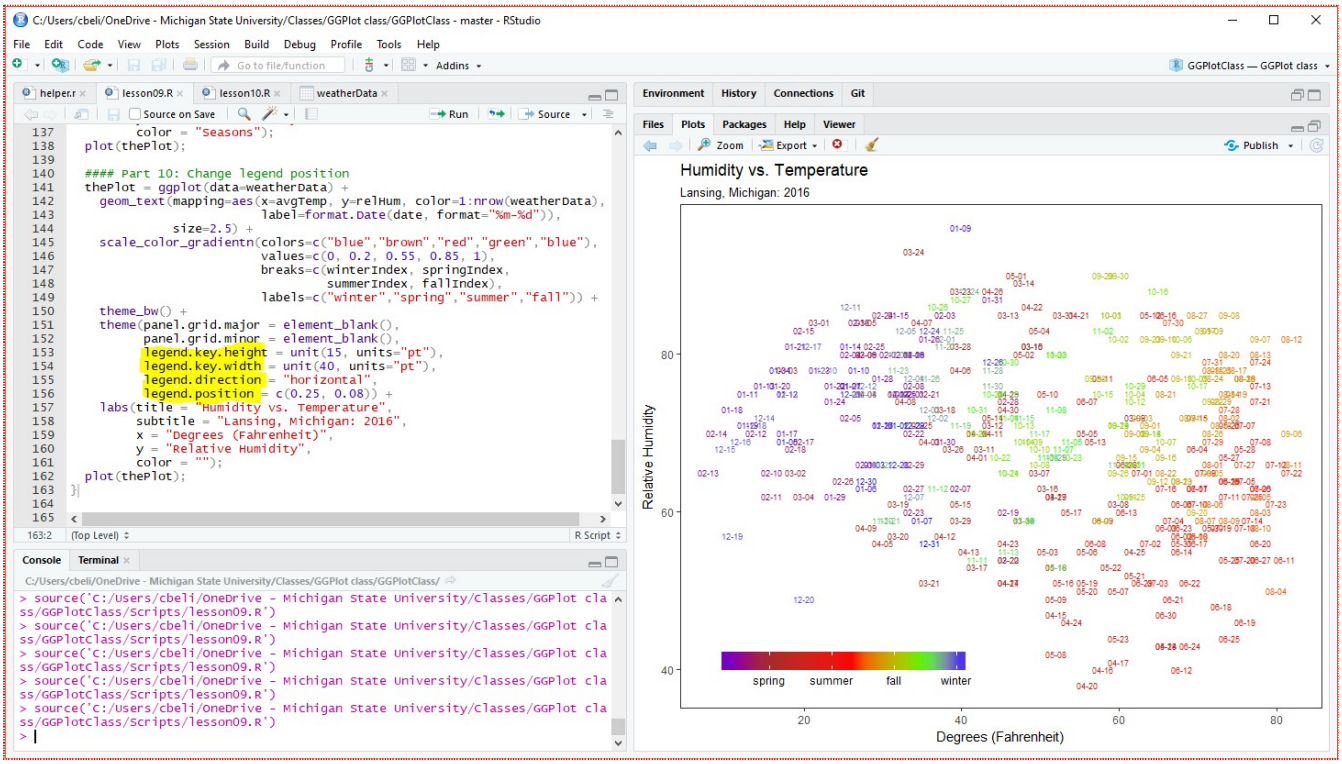

Fig 10: Changing the height, width, alignment, and position of the legend

10 - Application

1. Create a script file in your GGPlot Class Project called ***app09.r***.
2. Create a text plot of Precipitation vs Humidity
3. Label the points with ***avgTemp***
4. Color the text points using a gradient that goes from blue (coldest temp) to darkgreen (middle), to red (hottest)
5. Change the y-axis to a logarithmic scale
   - This can be done by adding the component: ***scale_y_continuous(trans="log10")***
6. Move the legend into the plot area making sure not to cover any points.

11 - Extension: the pattern parameter in grep()