01-04: Date objects and canvas styles

1 - Purpose

- Applying styles to the plot paneling and axes
- Handling and manipulating date variables
- Applying limits and spacing to discrete and continuous axes
- Modifying a data frame

2 - Get data

Like last lesson, we are going to use the weather data from the file LansingNOAA2016.csv and save it to a data frame called ***weatherData*** (*fig 1*).

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

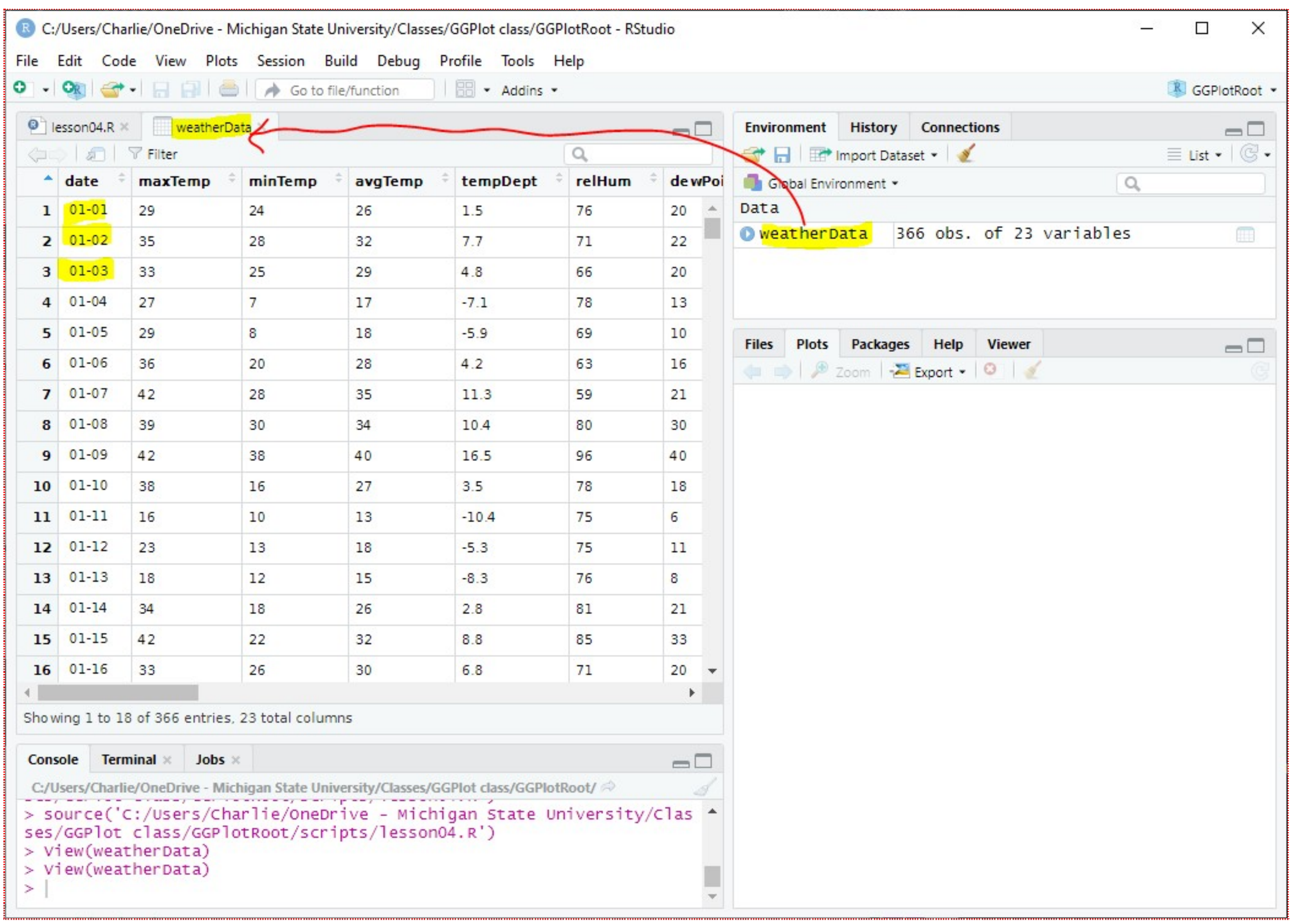

*Fig 1: **weatherData** data frame, note that the **date** column has only the month and day (no year)*

This time we are going to plot ***avgTemp*** vs. ***date***. However, if we look at the values in the ***date*** column (*fig 1*), we see that there is no year given -- the year is assumed to be 2016, since that is in the file name.

This is a problem as GGPlot will only treat a data column as dates if the values in the column are properly formed (i.e., the values have a month, date, *and year*). If the date is not properly formed, then GGPlot will treat the column as string (or character) values.

3 - Adding the year to the date values

We are going to create a data column in the ***weatherData*** data frame that has properly formed date values (i.e., with the year 2016 in it). Since *it is generally not a good idea to directly modify values in a data frame*, we are going to make a copy of the ***date*** column, called ***dateYr***, and modify the ***dateYr*** column instead.

We will break down this process into four steps:
1. Save the ***date*** column in ***weatherData*** to a vector
2. Append the year, **2016**, to all values in the vector
3. Convert the string vector into a date vector
4. Save the vector back to the data frame

Note: There are quicker ways to do this but it is important to understand the individual steps.

3.1 - Save the date column to a vector

There are many different ways in R to save a column in a data frame to a vector. The three most common ways are presented below, the last two are commented out so they will not be executed:

```
#### Part 1: Add year to date values ####
# a) save the date vector from the data frame to the variable theDate
theDate = weatherData[ , "date"];   # get all values from the dates column
# theDate = weatherData[["date"]];  # equivalent to previous line
# theDate = weatherData$date;       # equivalent to previous 2 lines
```

I choose to use the first method because it is the most explicit at showing that ***weatherData*** is a data frame, and data frames are two-dimensional objects with rows (1st value) and columns (2nd value). The blank first value in ***weatherData[, "date"]*** states we are *getting values from all rows* in the ***date*** column.

Extension: R data frames and tidyverse tibbles

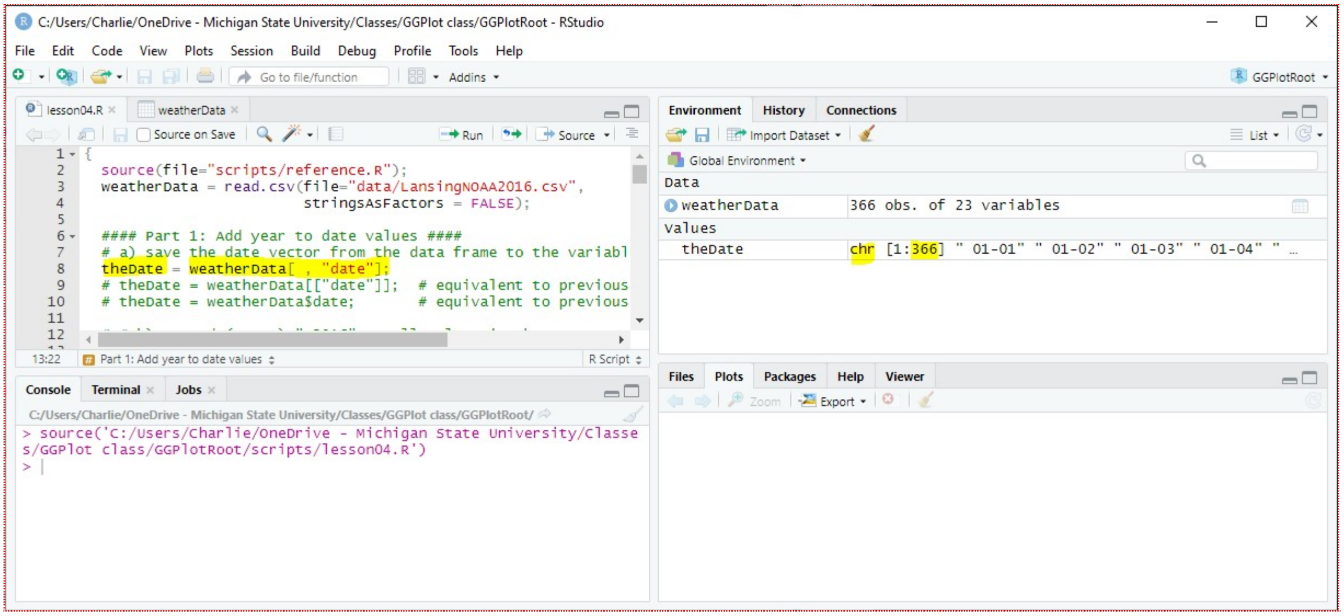

*Fig 2: Saving the date column to a vector (note: the vector contains 366 string, or **chr**, values -- not **date** values)*

Extension: get a range of values from the column

3.2 - Append the year to the date

Now we want to append, or paste, the year on to every date value in the string vector ***theDate***. We will use ***paste()*** and there are three parameters to set:

1. The initial values: ***theDate***
2. The value being appended or pasted: **"-2016"**
3. The separator between initial and pasted values: in this case, an empty string, ""

```
# append (paste) "-2016" to all values in theDate
theDate = paste(theDate, "-2016", sep="");
# theDate = paste(theDate, "2016", sep="-");  # functionally equivalent to the
    previous line
```

Extension: more details about **paste()** and the reason why parameter names are not used for the first two values

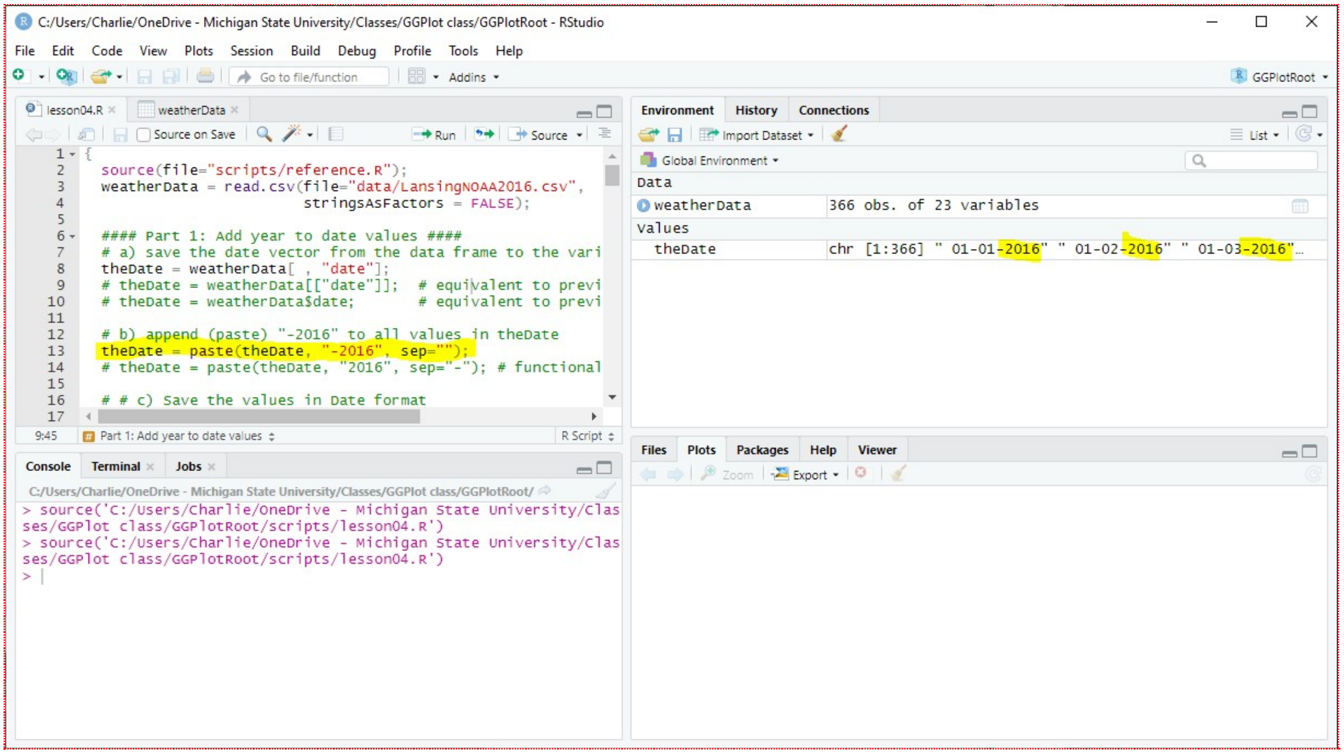

Fig 3: Pasting the year to all values in the vector

3.3 - Convert the vector into a date vector

At this point, R sees the values in ***theDate*** as strings (or ***chr***) *not **dates***.

Now we need to tell R:

- this vector contains dates (as opposed to strings) and
- how the dates values are formatted

We can do both using ***as.Date()***, and modifying two parameters:

1. the object we want to convert to dates (in this case, the vector ***theDate***)
2. the ***format*** that the dates are in, which is: **%m-%d-%y**

- ***%m***: 2-digit month (00-12)
- ***%d***: 2-digit date (00-31)
- ***%Y***: 4-digit year (2016)
    - *note: lowercase **y** is a 2-digit year whereas capital **Y** is a 4-digit year*
    - *Trap: using the wrong parameter for year (lowercase **y** vs. capital **Y**)*
- **-**: dashes used to separate month, day, and year values

```
# c) Save the values in Date format
theDate = as.Date(theDate, format="%m-%d-%Y");
```

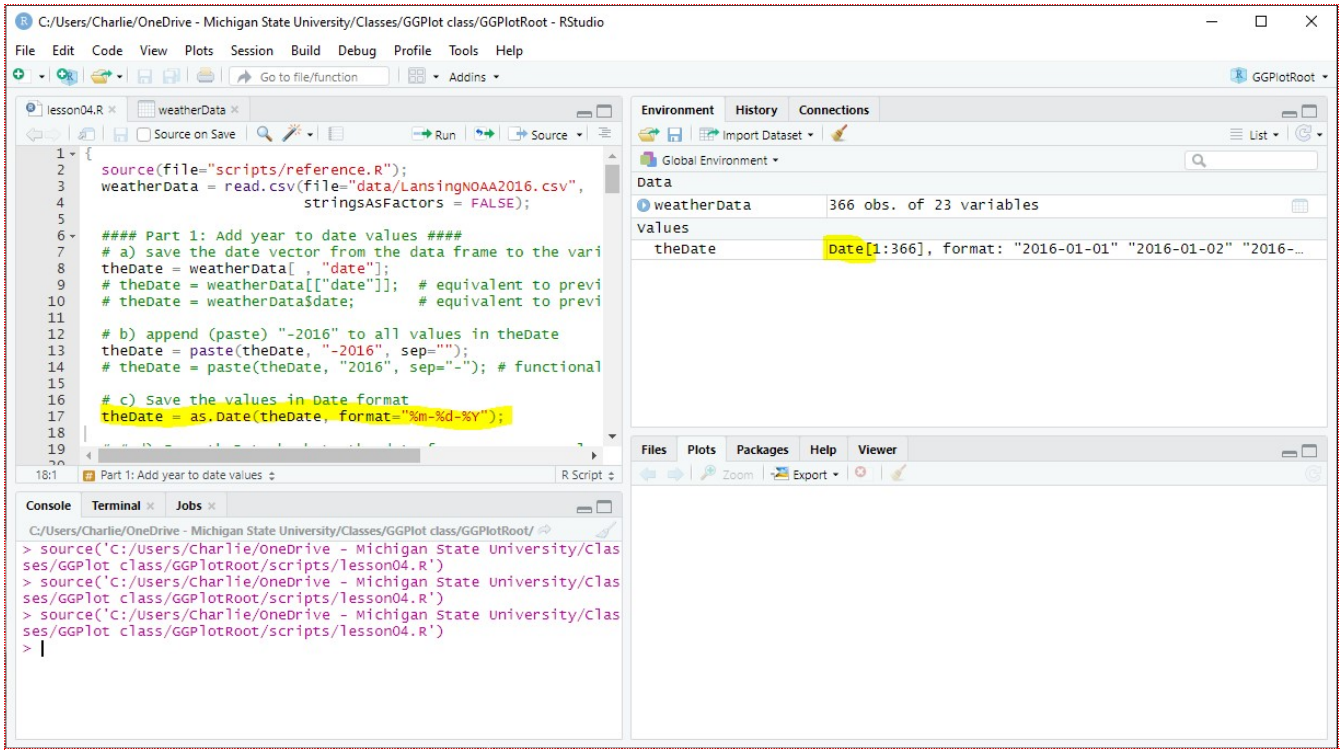

*Fig 4: Now R recognizes the values in the vector as **Date** values*

3.4 - Save the date vector to the data frame

We have the vector in date format, now we need to save the values back to a new column in the data frame. This is pretty much the reverse of the first step and, again, there are multiple ways to do it -- I present three ways and comment out the last two.

In each case, we are saving ***theDate*** to the column in ***weatherData*** called ***dateYr***:

```
# d) Save theDate back to the data frame as a new column
weatherData[, "dateYr"] = theDate
# weatherData[["dateYr"]] = theDate; # equivalent to previous line
# weatherData$dateYr = theDate;      # equivalent to previous 2 lines
```

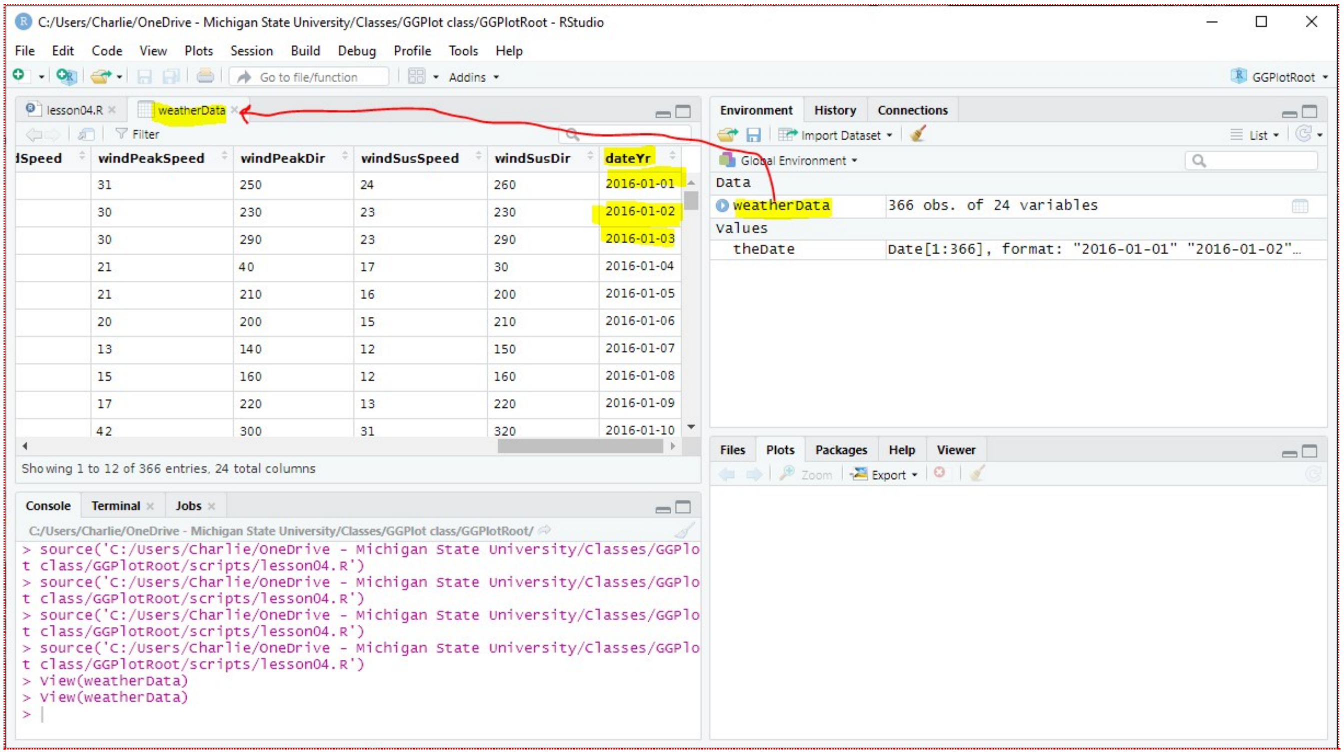

*Fig 5: The data frame now has a new column, **dateYr**, with properly formatted dates*

Now that we have a column with properly formatted date values, we can move to plotting the data....

4 - Line plots

In the last lesson, we plotted points using the plotting component ***geom_point()***. In this lesson we are going use the plotting component ***geom_line(),*** which will connect the points and make a line plot.

We are going to use ***geom_line()*** twice, to plot:

1. ***maxTemp*** *vs* **date**
2. ***minTemp*** vs ***date***

In GGPlot, you can put multiple plots on one graph using (**+**), the code below says that we are going to put two ***geom_line()*** components in the plot area :

```
plotData = ggplot(data=weatherData) +
           geom_line(mapping=aes(x=dateYr, y=maxTemp)) +
           geom_line(mapping=aes(x=dateYr, y=minTemp));
plot(plotData);
```

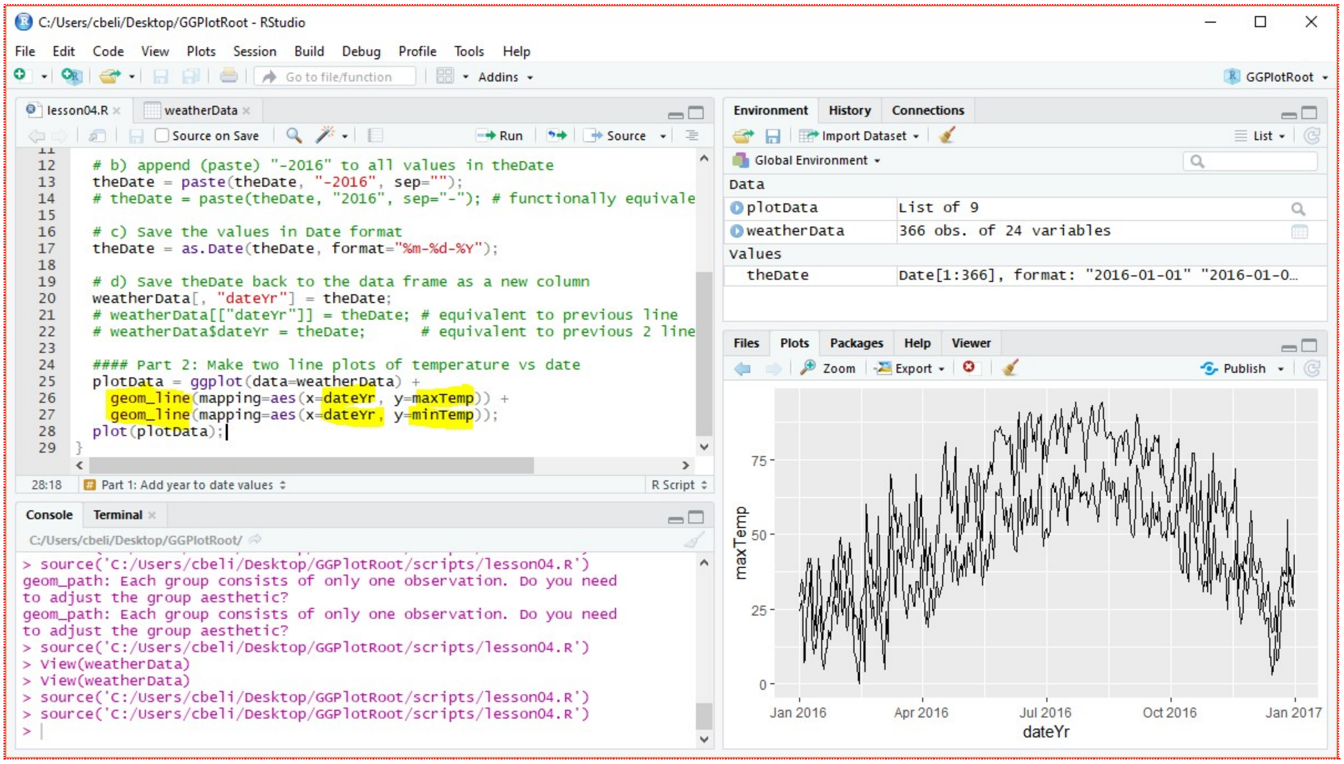

*Fig 6: Two line graphs: **maxTemp** vs **Date** and **minTemp** vs **Date***

4.1 - Add labels and coloring

Now we are going to:

- visually separate the two plots by setting the ***color*** subcomponent in ***geom_line()***
- add titles and axis labels using the ***labs()*** component

```
#### Part 3: Add labels and colors ###
plotData = ggplot(data=weatherData) +
           geom_line(mapping=aes(x=dateYr, y=maxTemp),
                     color="palevioletred1") +
           geom_line(mapping=aes(x=dateYr, y=minTemp),
                     color="aquamarine2") +
           labs(title = "Temperature vs. Date",
                subtitle = "Lansing, Michigan: 2016",
                x = "Date",
                y = "Temperature (F)");
plot(plotData);
```

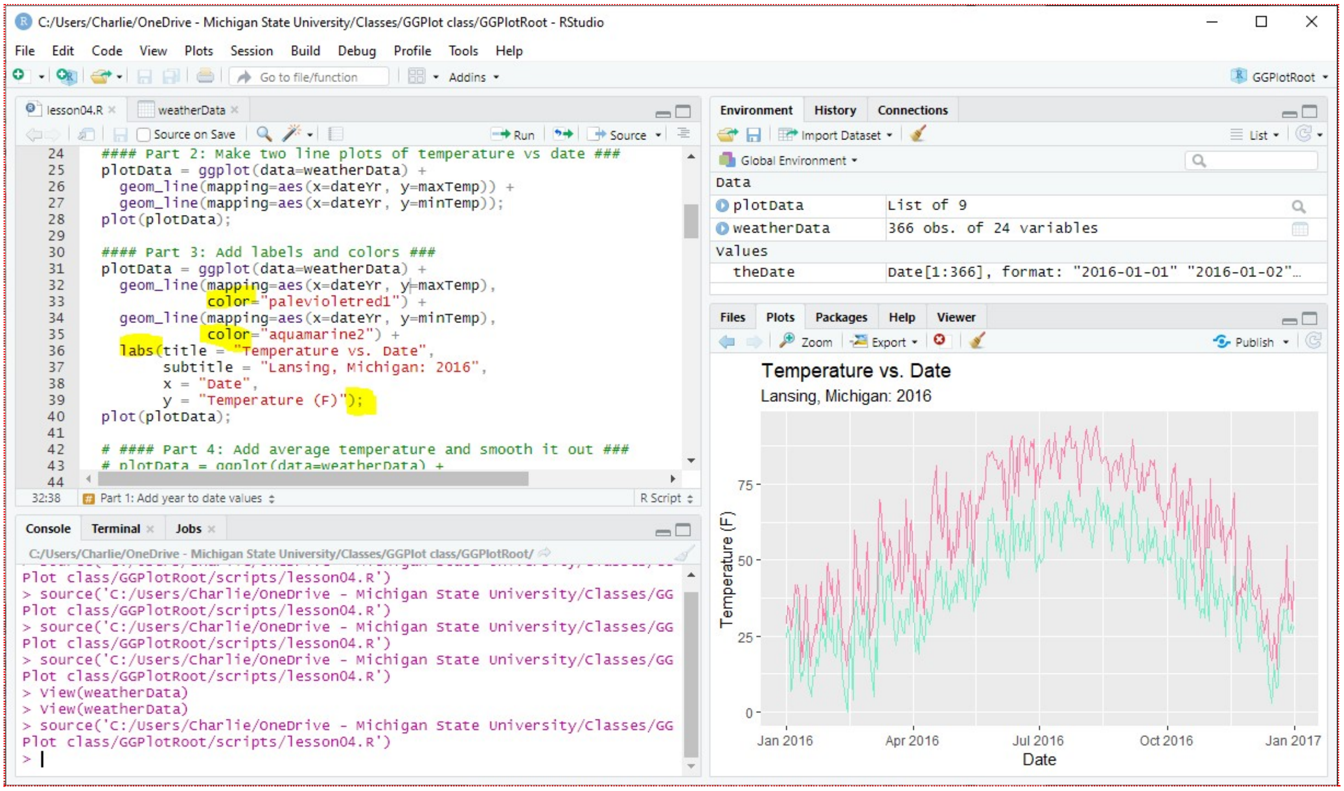

Fig 7: Changing the line plot colors and adding titles ans axis labels

We will deal with the lack of contrast between the plots and the plot area background in a bit...

4.2 - Add a smoothing function

On the graph, we are going to add another component that represents the average temperature (***avgTemp***) and use a method that smooths out the values. Like last lesson, we use ***geom_smooth()***, but we will use a different smoothing method this time (***loess***).

```
#### Part 4: Add average temperature and smooth it out ###
plotData = ggplot(data=weatherData) +
           geom_line(mapping=aes(x=dateYr, y=maxTemp),
                     color="palevioletred1") +
           geom_line(mapping=aes(x=dateYr, y=minTemp),
                     color="aquamarine2") +
           geom_smooth(mapping=aes(x=dateYr, y=avgTemp),
                       color="orange",
                       method="loess",
                       linetype=4,
                       fill="lightblue") +
           labs(title = "Temperature vs. Date",
                subtitle = "Lansing, Michigan: 2016",
                x = "Date",
                y = "Temperature (F)");
plot(plotData);
```

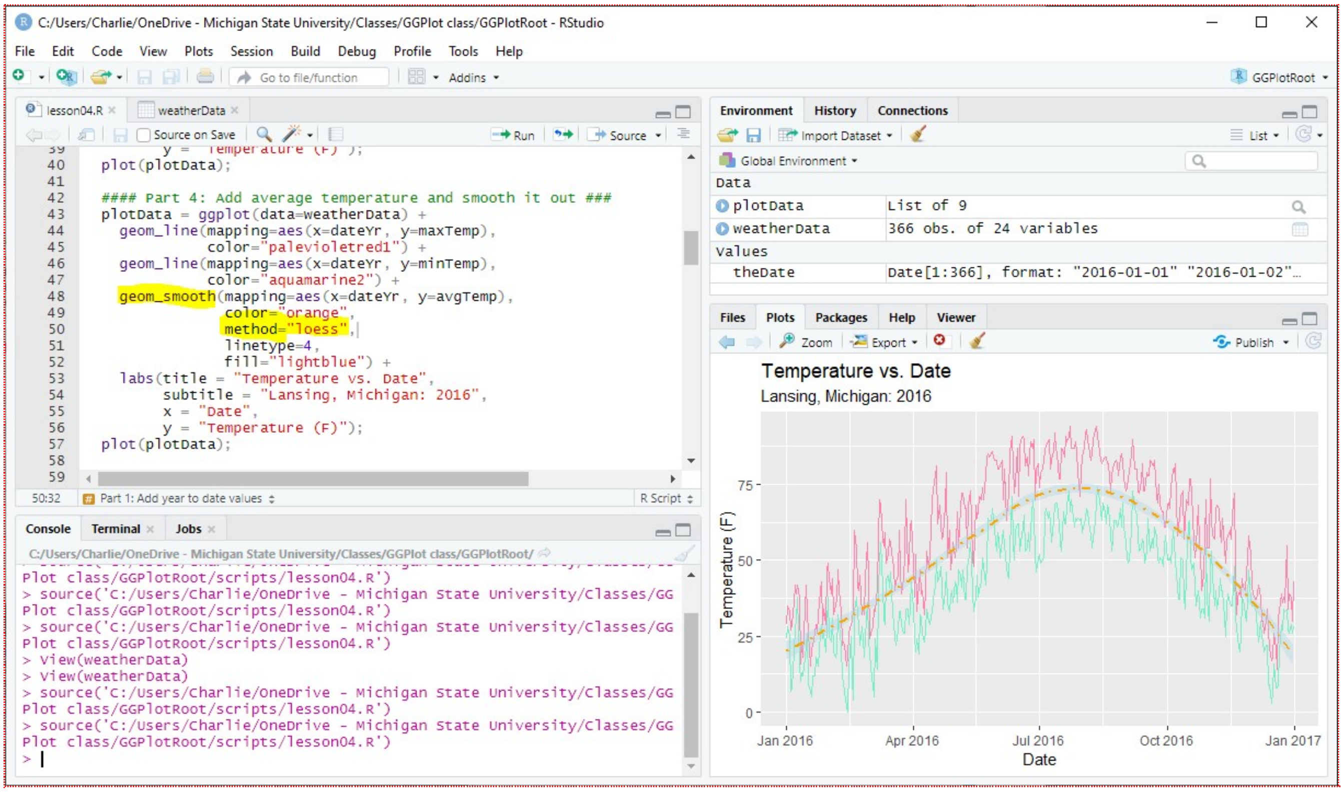

*Fig 8: Adding **avgTemp** as a smoothed plot and styled the line*

5 - Paneling changes and greyscaling

The paneling for the graph is essentially the background of the plot area, and it consists of a fill color and axes lines that go across it. In *figure 8*, the fill color is gray and the axis lines are white.

Paneling changes are made using the ***theme()*** component, and the subcomponents we are modifying are:

- ***panel.background***: a *rectangular element* representing the background of the plot area
- ***panel.grid.major***: a *line element* representing the main axis lines that go across the plot area
- ***panel.grid.minor***: a *line element* representing the the halfway axis lines that go across the plot area

We are using *greyscale* to set the color of the various objects. In GGPlot, you can choose greyscale colors between ***grey0*** (black) and ***grey100*** (white). You can think of the number after ***grey*** as the percentage of light from ***0*** (none) to ***100*** (all).

```
#### Part 5: Paneling changes ###
# grey0 is black, grey100 is white, and numbers in between are shades of grey
plotData = ggplot(data=weatherData) +
           geom_line(mapping=aes(x=dateYr, y=maxTemp),
                     color="palevioletred1") +
           geom_line(mapping=aes(x=dateYr, y=minTemp),
                     color="aquamarine2") +
           geom_smooth(mapping=aes(x=dateYr, y=avgTemp),
                       color="orange",
                       method="loess",
```

```
                       linetype=4,
                       fill="lightblue") +
          labs(title = "Temperature vs. Date",
               subtitle = "Lansing, Michigan: 2016",
               x = "Date",
               y = "Temperature (F)") +
          # size and color relate to the border, fill is the inside color
          theme(panel.background = element_rect(fill="grey25",
                                                 size=2, color="grey0"),
                panel.grid.minor = element_line(color="grey50", linetype=4),
                panel.grid.major = element_line(color="grey100"));
plot(plotData);
```

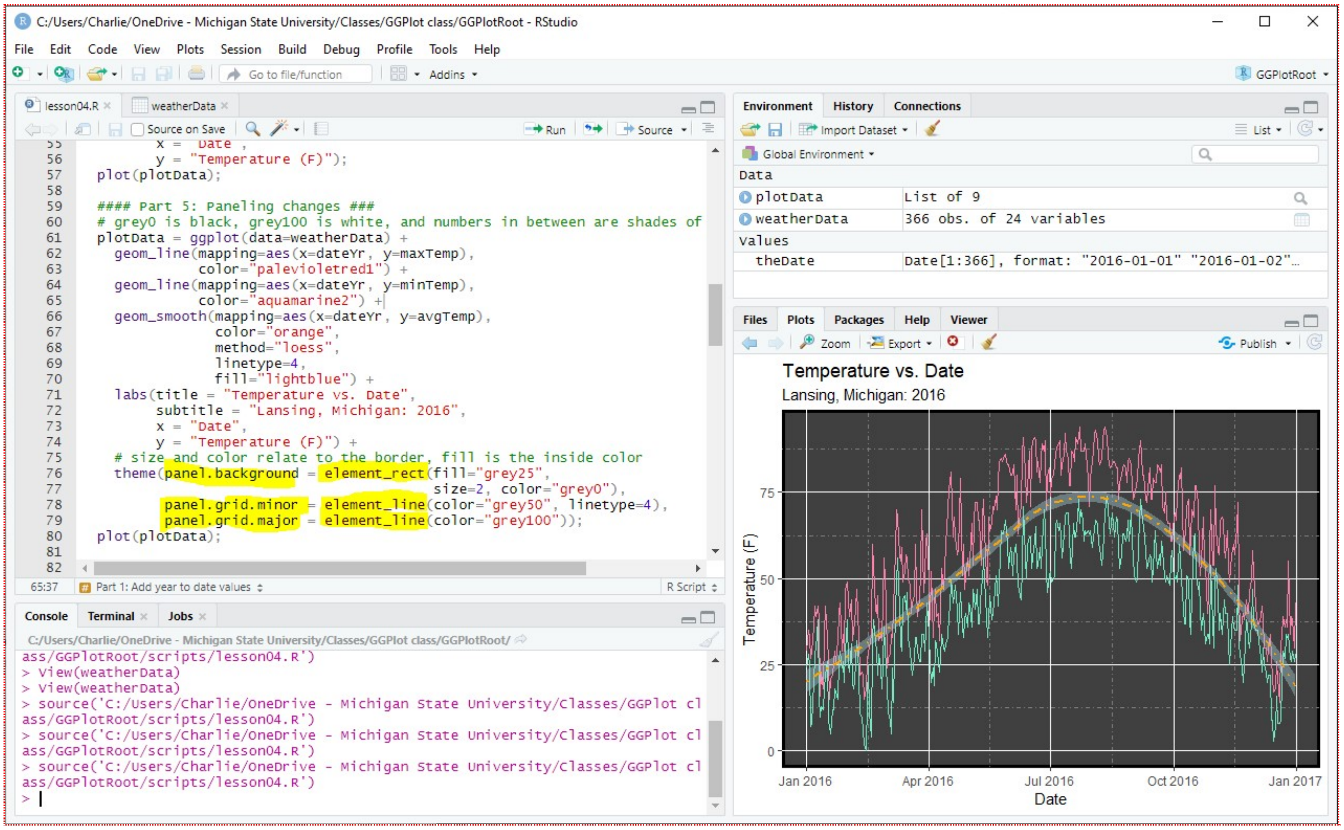

Fig 9: Paneling changes to the graph

5.1 - GGPlot Elements

In GGPlot, many of the subcomponents of a plot (including the paneling) are set to an element object (e.g., ***element_line***, ***element_rect***).

For example, if we wanted to set the background color of the plot paneling to orange, we set the plot paneling to a *rectangular element* (element object) that has a background color of orange.

In the above plot (*fig 9*), we changed:
panel.background is set to a rectangular element (***element_rect***) with parameters:

- ***fill = "grey25"***: set the inside color to a dark grey (25% light)

- ***size* = 2**: set the size of the border to 2 millimeters (default is 1mm)
- ***color = "grey0"***: set the border color to black (0% light)

panel.grid.minor is set to a line element (***element_line***) with parameters:
- ***color = "grey50"***: set the line color to a medium grey
- ***linetype = 4***: set the line type to be alternating long dashes and short dashes

panel.grid.major is a line element (***element_line***) with parameter:
- ***color = "grey100"***: set the line color to white (100% light)

6 - Outside the panel changes

Now we are going to use ***theme()*** component to change the properties of canvas area outside of the plots.

The subcomponents we are going to change are:
- ***plot.background***: the background outside the plot area (rectangular element)
- ***plot.title:*** the text that represents the title of the graph (text element)
- ***plot.subtitle:*** the text that represent the subtitle of the graph (text element)
- ***axis.text:*** the text that represents the labels on the x and y axes (text element)

```
#### Part 6: Making changes outside the panel ###
plotData = ggplot(data=weatherData) +
           geom_line(mapping=aes(x=dateYr, y=maxTemp),
                     color="palevioletred1") +
           geom_line(mapping=aes(x=dateYr, y=minTemp),
                     color="aquamarine2") +
           geom_smooth(mapping=aes(x=dateYr, y=avgTemp),
                       color="orange",
                       method="loess",
                       linetype=4,
                       fill="lightblue") +
           labs(title = "Temperature vs. Date",
                subtitle = "Lansing, Michigan: 2016",
                x = "Date",
                y = "Temperature (F)") +
           # size and color relate to the border, fill is the inside color
           theme(panel.background = element_rect(fill="grey25",
                                                 size=2, color="grey0"),
                 panel.grid.minor = element_line(color="grey50", linetype=4),
                 panel.grid.major = element_line(color="grey100"),
                 plot.background = element_rect(fill = "lightgreen"),
                 plot.title = element_text(hjust = 0.45),
                 plot.subtitle = element_text(hjust = 0.42),
                 axis.text = element_text(color="blue", family="mono", size=9));
plot(plotData);
```

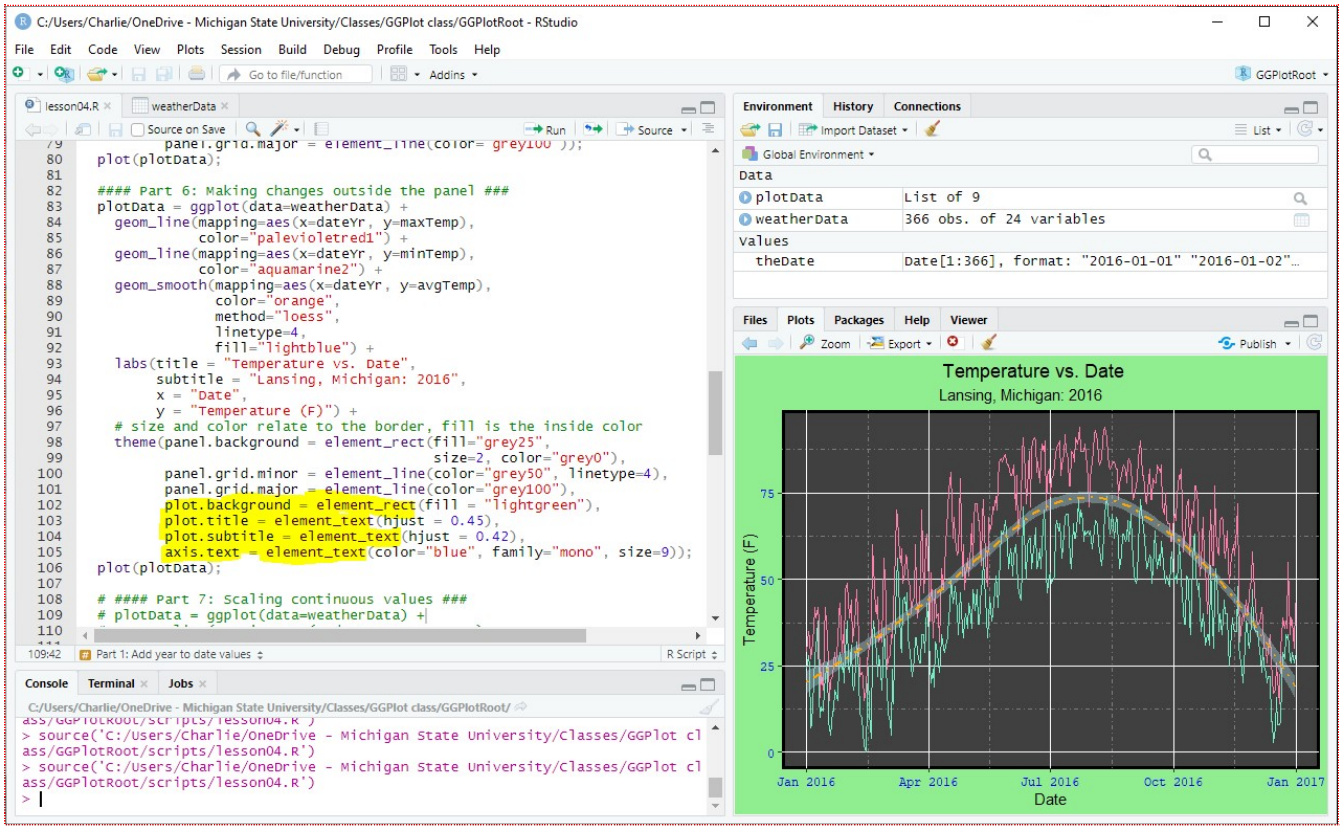

Fig 10: Style changes outside of the plot area

panel.background is set to a rectangular element (***element_rect***) with parameter:

- ***fill = "lightgreen"***: set the background color outside the plot area to light green

plot.title is set to a text element (***element_text***) with parameter:

- ***hjust= 0.45***: set the horizontal justification of the text to almost the center

note: ***hjust*** has values from ***0.0*** (left justified text) to ***1.0*** (right justified text) and ***0.5*** is centered

plot.subtitle is set to a text element (***element_text***) with parameter:

- ***hjust= 0.42***: set the horizontal justification of the text to line up underneath the title

axis.text is set to a text element (***element_text***) that represents the text labels on the x and y axes. We change the parameters:

- ***color= "blue"***: set the text color to **blue**
- ***family="mono"***: set the font type to **mono**
- **size=9:** set the font size to **9** pixels

If you want to individually change the x-axis text style or y-axis text style you can use the components: ***axis.text.x*** and ***axis.text.y***

Note: Changing the font family can cause issues because different computer have different fonts installed.

7 - Scaling axis values

GGPlot does a decent job of generating axes values and setting the distance between tick marks, but there are times you want to change these values.

In the above plot (*fig 10*), the *x-axis has discrete values* and the *y-axis has continuous values* -- we will look at these two cases separately.

7.1 - Scaling continuous axis values

Temperature is on the y-axis of the plot and it is continuous, so the component of GGPlot that controls its scaling is: ***scale_y_continuous.***

We need to set two subcomponents of ***scale_y_continuous***:

- ***limits***: a vector that contains the low and high value of the y-axis (set to **-15** and **90**)
- ***breaks***: a sequence that gives the tick marks (set to interval of **20** between **-15** and **90**)

```
#### Part 7: Scaling continuous values ###
plotData = ggplot(data=weatherData) +
           geom_line(mapping=aes(x=dateYr, y=maxTemp),
                     color="palevioletred1") +
           geom_line(mapping=aes(x=dateYr, y=minTemp),
                     color="aquamarine2") +
           geom_smooth(mapping=aes(x=dateYr, y=avgTemp),
                       color="orange",
                       method="loess",
                       linetype=4,
                       fill="lightblue") +
          labs(title = "Temperature vs. Date",
                subtitle = "Lansing, Michigan: 2016",
                x = "Date",
                y = "Temperature (F)") +
           theme(panel.background = element_rect(fill="grey25",
                                                 size=2, color="grey0"),
                 panel.grid.minor = element_line(color="grey50", linetype=4),
                 panel.grid.major = element_line(color="grey100"),
                 plot.background = element_rect(fill = "lightgreen"),
                 plot.title = element_text(hjust = 0.45),
                 plot.subtitle = element_text(hjust = 0.42),
                 axis.text = element_text(color="blue", family="mono", size=9))+
           scale_y_continuous(limits = c(-15,90),
                              breaks = seq(from=-15, to=90, by=20));

plot(plotData);
```

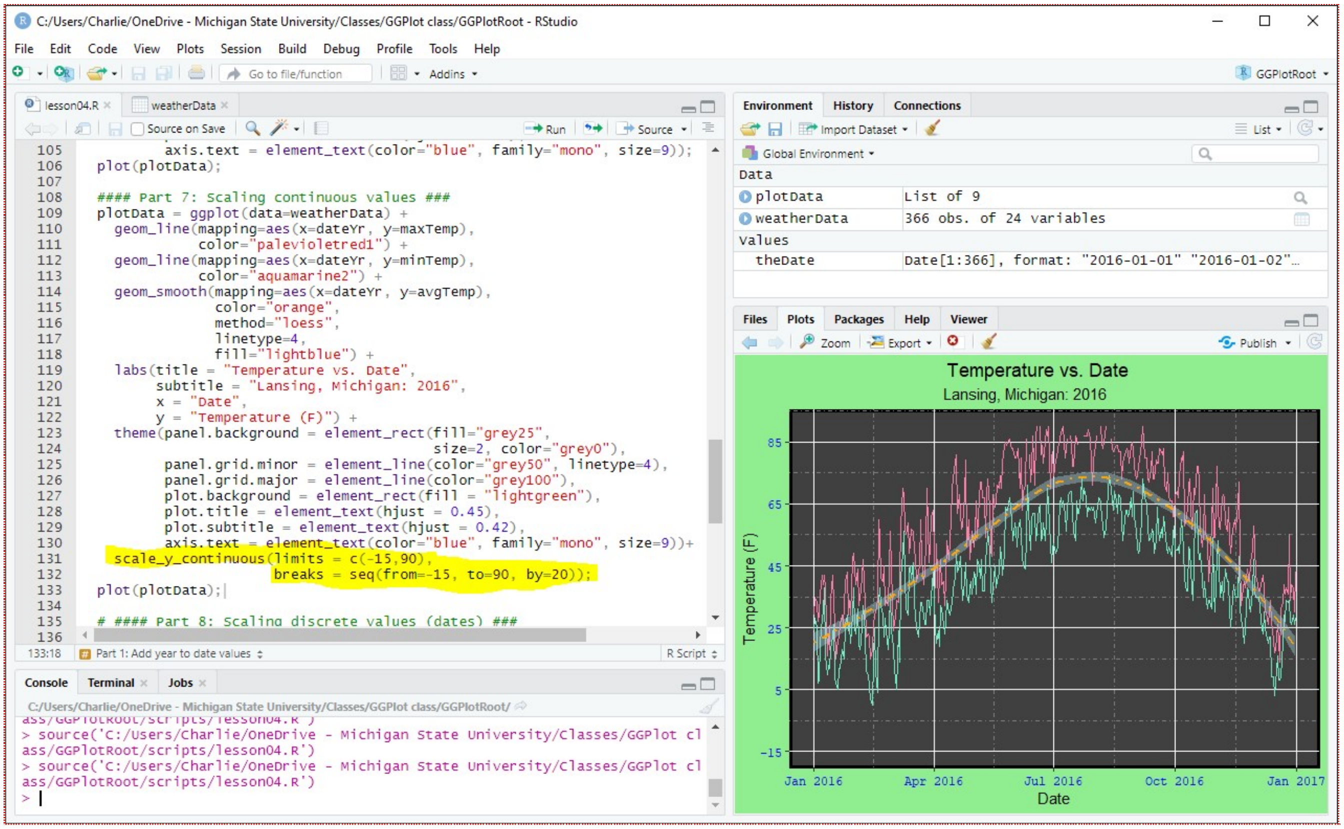

Fig 11: Scaling the continuous y-axis (temperatures)

The y-axis (*fig 11*) goes a little beyond the limits of **-15** and **90** -- this is because GGPlot always adds a bit of cushioning at the ends.

The tick marks on the y-axis (*fig 11*) are at: **5, 25, 45, 65**, and **85** -- and there are minor tick marks halfway in-between at: **-5, 15, 35, 55,** and **75**.

7.2 - Scaling date axis values

Date is on the x-axis of the plot and the component of GGPlot that controls its scaling is: ***scale_x_date.***

We need to set two subcomponents of ***scale_x_date***:

- ***limits***: a vector that gives the low and high value of the x-axis (set to March 21 and December 21)
- ***date_breaks***: a sequence that gives the tick marks (set to 6 week intervals)

We are also going to format the dates to 2-digit months and 2-digit days using the ***date_labels*** parameter.

date_labels = format("%m/%d") means the date will be represented by:

- a 2-digit month (***%m***)
- a front-slash separator (***/***)
- a 2-digit day (***%d***)

```
#### Part 8: Scaling discrete values (dates) ###
plotData = ggplot(data=weatherData) +
           geom_line(mapping=aes(x=dateYr, y=maxTemp),
                     color="palevioletred1") +
```

```
           geom_line(mapping=aes(x=dateYr, y=minTemp),
                     color="aquamarine2") +
           geom_smooth(mapping=aes(x=dateYr, y=avgTemp),
                       color="orange",
                       method="loess",
                       linetype=4,
                       fill="lightblue") +
          labs(title = "Temperature vs. Date",
                subtitle = "Lansing, Michigan: 2016",
                x = "Date",
                y = "Temperature (F)") +
           # size and color relate to the border, fill is the inside color
           theme(panel.background = element_rect(fill="grey25",
                                                  size=2, color="grey0"),
                 panel.grid.minor = element_line(color="grey50", linetype=4),
                 panel.grid.major = element_line(color="grey100"),
                 plot.background = element_rect(fill = "lightgreen"),
                 plot.title = element_text(hjust = 0.45),
                 plot.subtitle = element_text(hjust = 0.42),
                 axis.text = element_text(color="blue", family="mono", size=9))+
           scale_y_continuous(limits = c(-15,90),
                              breaks = seq(from=-15, to=90, by=20)) +
           scale_x_date(limits=c(as.Date("2016-03-21"),
                        as.Date("2016-12-21")),
                        date_breaks = "6 weeks",
                        date_labels = format("%m/%d"));
plot(plotData);
```

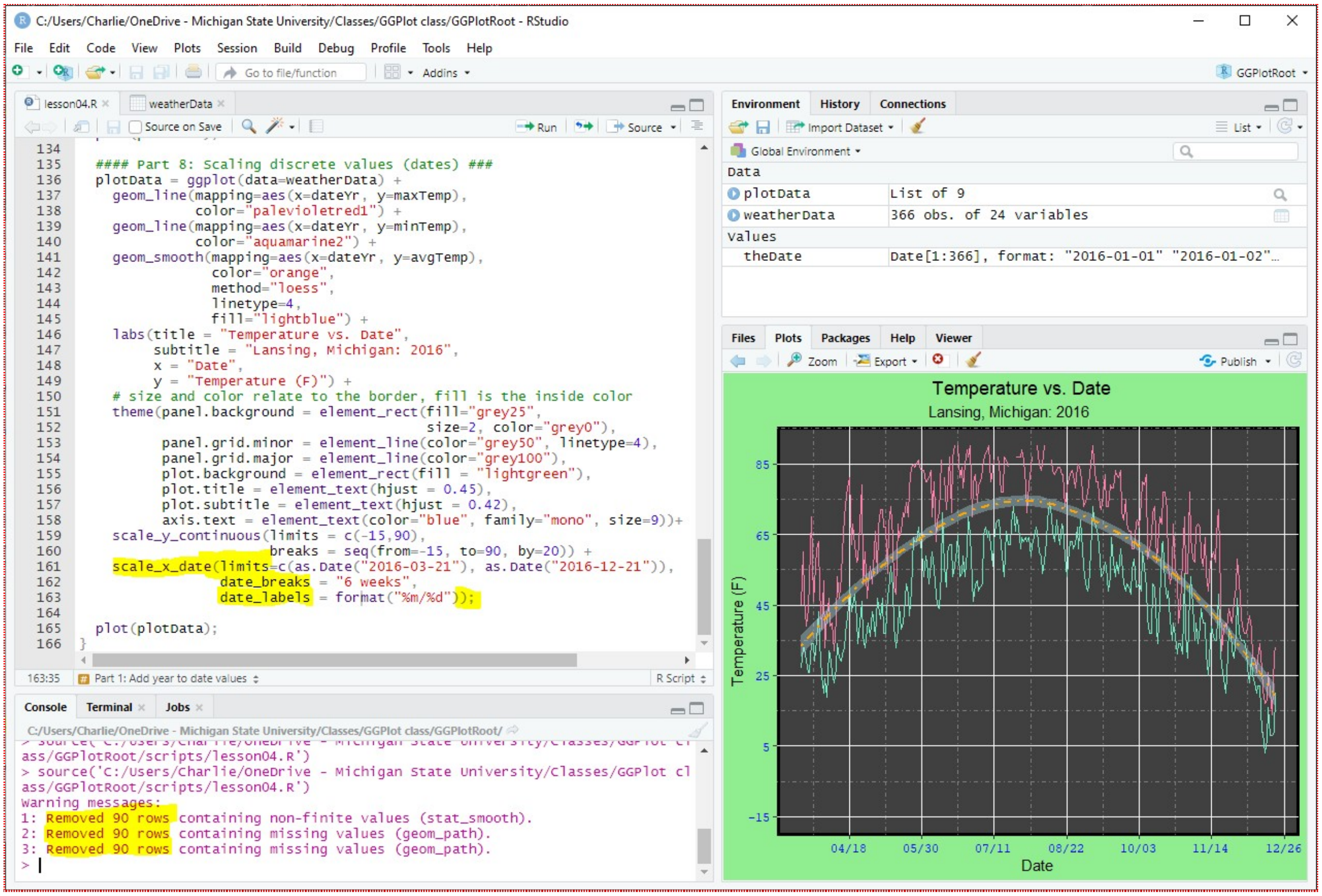

Fig 12: Scaling the Date axis

The warning message in the Console Window (*fig 12*) is pointing out that you have cut off 90 days of data from each of the three plot. The 90 days represent **January 1** to **March 20** and **December 22** to **December 31** -- the days outside the **limits** set in **scale_x_date**.

8 - Application

1. Create a script file in your Project's ***script*** folder called ***app04.r***.
2. Create a date column that includes the year (same as in lesson)
3. Convert the three temperature columns in the data frame from Fahrenheit to Celsius

$$C = \frac{5}{9}\left(F - 32\right)$$

4. For the Celsius Temperatures:
   - make a line plot of high temp vs date
   - make a line plot of low temp vs date
   - make a smoothed plot of average temp
5. Add appropriate labels to the axis and a title.
6. Right-justify the title and subtitle
7. Only show Spring and Summer dates on the plot (i.e., March 21 through Sept 21)
8. Set the axis limits and tick marks so that:
   A. the full range of high and low temperatures can be seen
   B. there are 3 ticks on the x-axis and 3 ticks on the y-axis
9. Set the color of the x-axis labels to blue
10. Set the color of the y-axis labels to red

11. Display the dates in this format: Jun-28-2016 (abbreviated month, 2-digits for days, and four digits for years)
    - Go to this page for help on formatted the dates

9 - Extension: Getting a range of values in a column

```
weatherData[3, "date"];   # get the 3rd value from the date column
weatherData[3:7, "date"];   # get the 3rd through 7th values from the date column
weatherData[seq(from=2, by=2), "date");   # get all even values in the date column
```

10 - Trap: Using the wrong parameter for year

%Y means that the year is in the 4-digit format, whereas
%y means that the year is in the 2-digit format

If you put in the wrong parameter value (lowercase ***y*** instead of uppercase ***Y***):

```
theDate = as.Date(theDate, format="%m-%d-%y");
```

then you will get an unusual result:

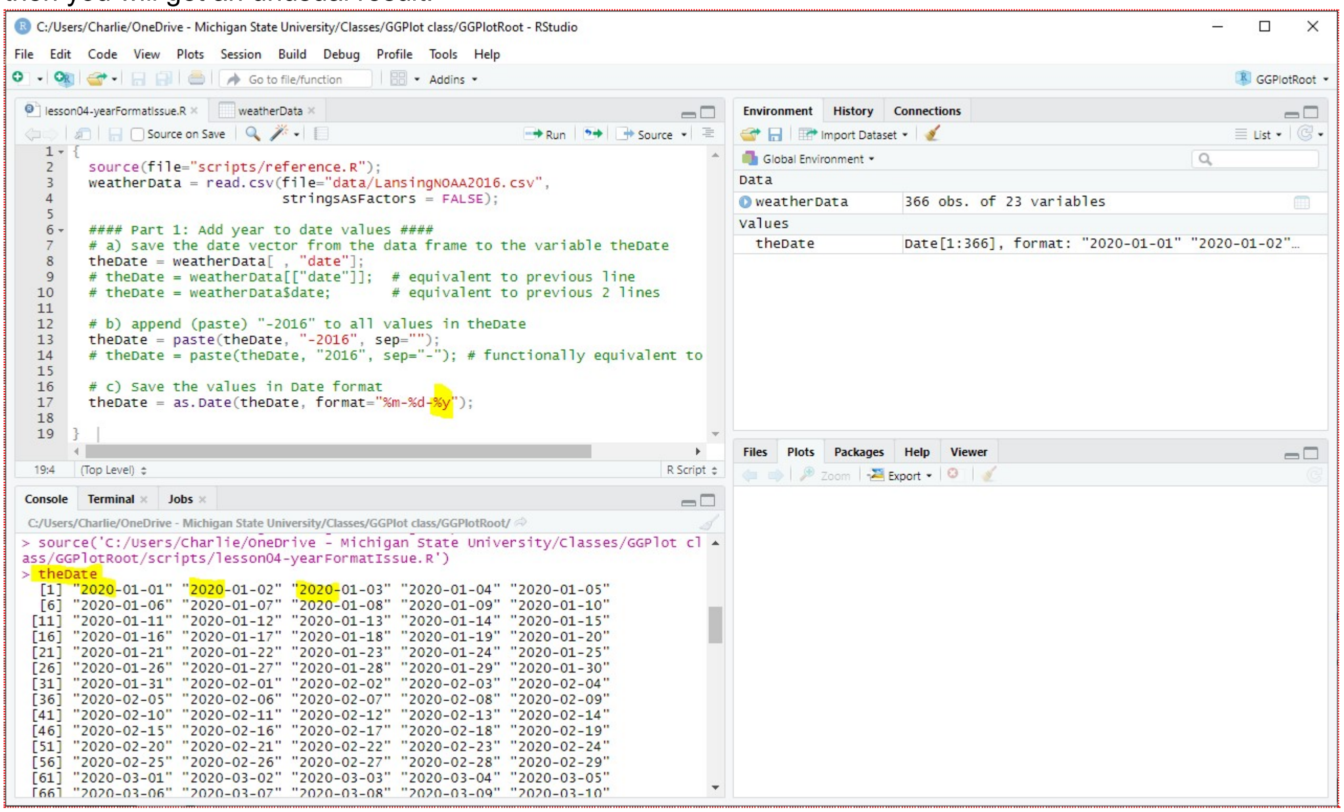

Fig 13: The year is now 2020 instead of 2016

%y tells R to take the first two values of the number and make that the year. The first two values of the year **2016** is **20** -- so the **16** is dropped. R then assumes we are in the 21st century so R assumes the year is 20**20**.

11 - Extension: more about paste()

The help page can be accessed in RStudio by going to the ***Help*** tab in the lower-left corner and searching for **paste**.

If we go to the help page for ***paste()***, it gives the usage for ***paste()*** as:

```
paste (..., sep = " ", collapse = NULL)
```

(...,) means that ***paste()*** will accept any number of initials values -- these are the values that ***paste()*** will attempt to paste together. *These values do not have a parameter name*. When using ***paste()***, you must use parameter names for ***sep*** and ***collapse***, otherwise ***paste()*** will see these value as part of the initial set of values to be pasted together. There are many functions in R that accept an indeterminate number of unnamed (i.e., no parameter name) values at the beginning of a function.

sep = " " means the default separator value is one space. So, if you do not set ***sep***, you will get one space between all the values that ***paste()*** is pasting together.

12 - Extension: R data frames and tidyverse tibbles

In this lesson I introduce three ways to access a column from a data frame and save it to a vector:

```
#### Part 1: Add year to date values ####
# a) save the date vector from the data frame to the variable theDate
theDate = weatherData[ , "date"];   # get all values from the dates column
# theDate = weatherData[["date"]];  # equivalent to previous line
# theDate = weatherData$date;       # equivalent to previous 2 lines
```

All three of these methods are functionally the same on R data frames but they are functionally different if you are working with ***tibbles***, which is a modern take on data frames used by packages in the TidyVerse.

The second and third methods are the same but the first method:

```
theDate = weatherData[ , "date"];   # get all values from the dates column
```

will save the ***date*** column *as a one-column tibble* as opposed to a vector.

Because of this behavior, I would probably recommend using the third method to access a column from a data frame as it is more consistent:

```
theDate = weatherData$date;
```

The above method also has the advantage that as soon as you type ***weatherData$*** in RStudio, RStudio will give you suggestions. In future iterations of this course, I will probably switch to this method of accessing columns (even though I still think the first method is the most intuitive!).