02: Components

0.1 To-do

- Reword RStudio Project Windows

1 Purpose

- Create a script file to be read from other scripts
- Create a scatterplot in GGPlot
- Modify the scatterplot using components

1.1 Code for the lesson

The script for this lesson is here

- Save the R file, called ***lesson02-Components.R*** to the ***scripts*** folder inside your RStudio Project

The data for this lesson (Lansing weather in 2016) is here

- Save the CSV file, called ***Lansing2016NOAA.csv*** to the ***data*** folder inside your RStudio Project (this is the same file as last lesson).
- It is best to use the operating system's File Explorer/Finder to move the CSV file to the proper folder.
  - Trap: Using Excel to move files

2 Spacing Code

In GGPlot, you create a plot by initializing a GGPlot canvas, with ***ggplot()***, and then add components (e.g., a boxplot) to the canvas. Most components will, in turn, have subcomponents, (e.g., color or size).

Figure 1 shows a plot that has:

- A function to initialize GGPlot – highlighted in blue
- Five components – highlighted in yellow
- **+** to "add" components to the canvas – highlighted in green
- 10 subcomponents – highlighted in red

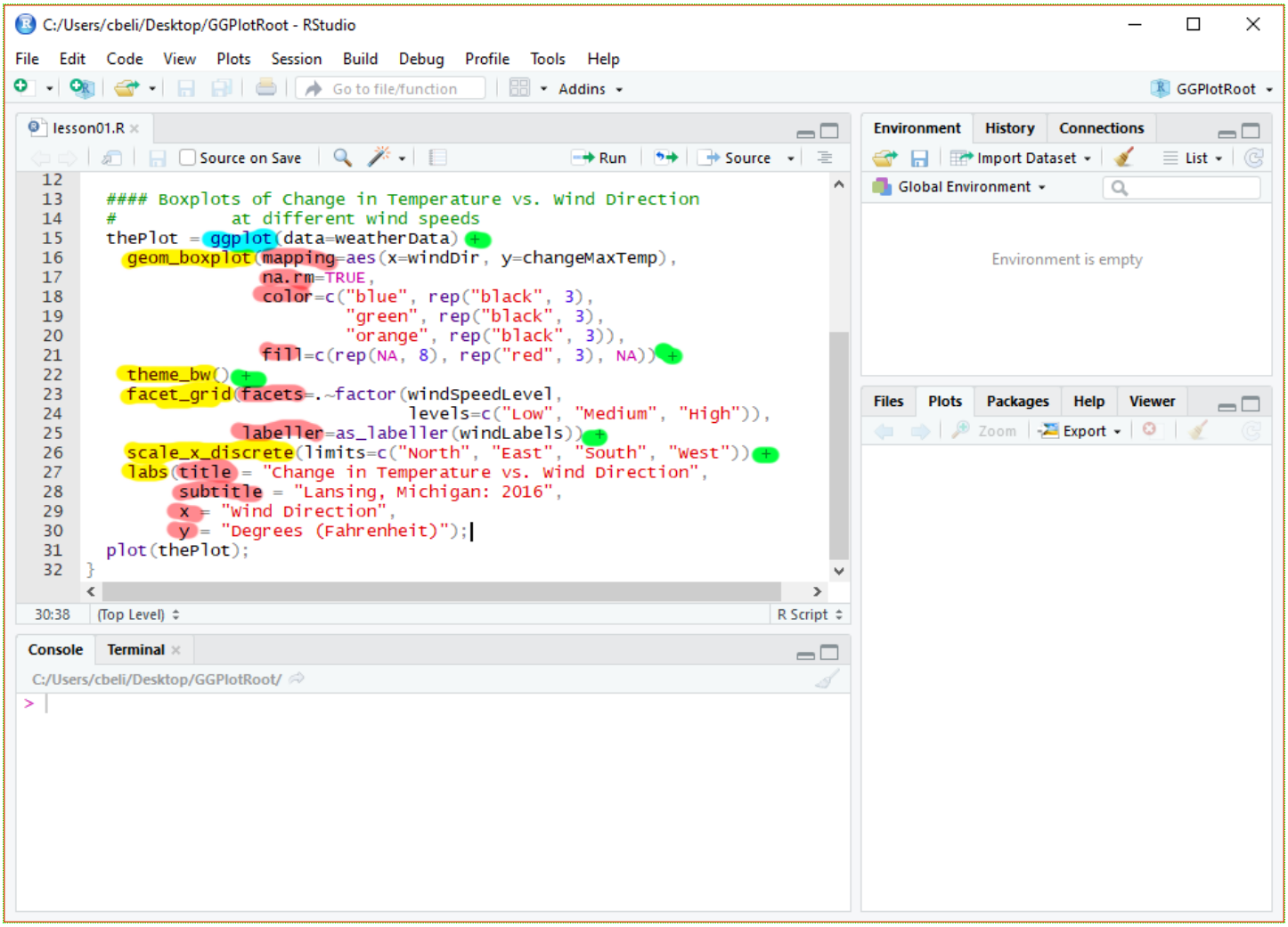

Figure 1: Spacing code to emphasize the different components of a plot

2.1 Component and subcomponent spacing

We will be going over the code in Figure 1 in much more detail in later lessons. For now, we are going to focus on the spacing. As you can see in Figure 2, the code for a plot can quickly get unwieldy– that is why it is important to consider code spacing from the very beginning. For this class, all components and subcomponents on a GGPlot will get its own line of code. This is also a requirement for all the lesson applications as it makes the code much easier to read.

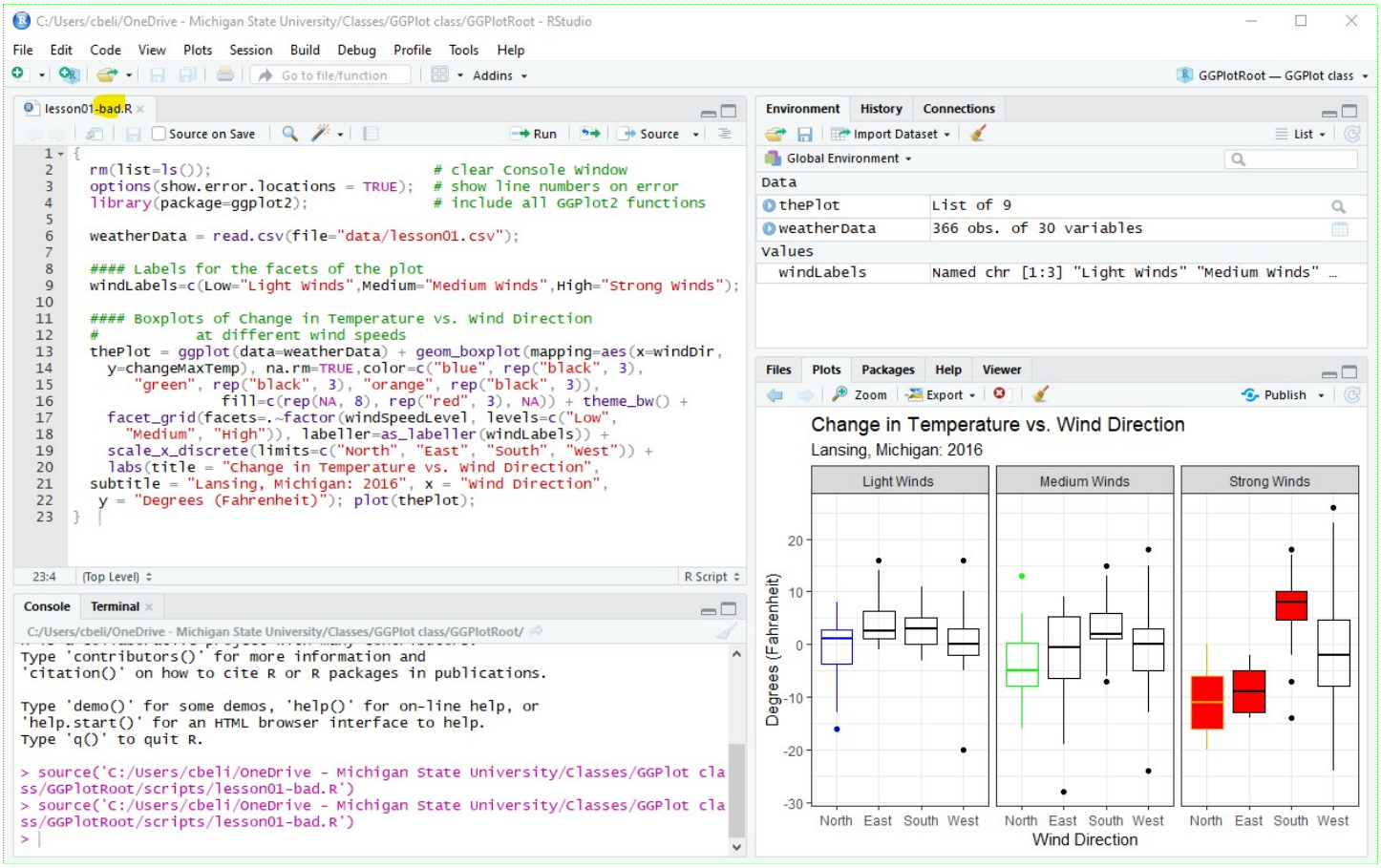

Figure 2: Poorly spaced code – it works but it is too hard to read!

2.2 GGPlot Functions are components

The components of a GGPlot are really the functions in the GGPlot package. The subcomponents are the arguments of those functions.

One component (i.e., function) in the GGPlot package is ***labs()***, which is the labeling component in GGPlot. ***labs()*** has multiple subcomponents (i.e., arguments) representing the different parts of the plot that can be labelled. ***labs()*** was used in Figure 1 to add a ***title*** and ***subtitle*** and ***x***-axis and ***y***-axis labels. It looks like this:

```
labs(title = "Change in Temperature vs. Wind Direction",
     subtitle = "Lansing, Michigan: 2016",
     x = "Wind Direction",
     y = "Degrees (Fahrenheit)")
```

In this class, we will refer to ***labs()*** as a component and ***title***, ***subtitle***, ***x***, and ***y*** as subcomponents.

3 How to make comments in R

In the R language, comments are created using the number-sign (**#**) key. Any text on the line after the (**#**) will be treated by R as a comment and it will not be executed. In Figure 2, there are comments on lines 2, 3, 4, 8, 11, and 12. These comments are colored light green but the color of the comments depends on your color scheme ([Color Schemes (extension)]).

You are required to make use of comments for your project in this class. Aside from being a good practice, it significantly reduces the frustration level of people who are trying to evaluate your work! Commenting also significantly reduces the frustration you will feel when trying to read your own code after an extended absence.

4 Open your project

We are going to open the RStudio Project we created in the last lesson. You can open your project by any of the following three methods:

1. Go to your Project Folder using your operating system's file manager and double-clicking the ***.RProj*** file
2. In RStudio: click ***File*** -> ***Recent Projects*** -> (*choose your project*)
3. In RStudio: click ***File*** -> ***Open Project*** -> Navigate to the Project Folder and click the ***.RProj*** file

Extension: RStudio Project windows

5 Starting a new script

To create a new script in RStudio, click ***File*** -> ***New File*** -> ***R Script.***

All the scripts in this class will contain these three lines at the top:

```
rm(list=ls());                          # clear the Environment tab
options(show.error.locations = TRUE); # show line numbers on error
library(package=ggplot2);               # include all GGPlot2 functions
```

Copy the three lines to your new script file and save this script as ***lesson02.R*** inside the ***scripts*** folder in your Project (***File*** -> ***Save as...*** -> open ***scripts*** folder -> click ***Save***).

5.1 Cleaning the Environment

```
rm(list=ls());                          # clear the Environment tab
```

This line removes all the variables and data from the ***Environment*** tab. Basically, this means your script can be executed with a clean slate. This is useful when you are creating scripts that are designed to be executed as a standalone script.

5.2 Error locations

```
options(show.error.locations = TRUE);  # show line numbers on error
```

This line is good to include in your R code because it instructs R to output to ***Console*** the line number that errors occur on. However, this error detection does not work well within GGPlot functions – so, it is of limited use in this class.

5.3 GGPlot2 Package

```
library(package=ggplot2);            # include all GGPlot2 functions
```

This line gives your script access to all of the functions in the GGPlot2 package. This package is technically the third version of GGPlot – but no one uses the first two versions (***ggplot*** and ***ggplot1***) anymore.

6 GGPlot components and subcomponents

GGPlot is based more on a building metaphor where the plot is made up of a bunch of component parts. whereas R-Base plotting (what GGPlot was built to replace) is based on the metaphor or drawing on a transparency.

One way to think about the functions is that they are components of a plot. Each time you call a GGPlot function, you are either adding a component to a plot or modifying an existing component. For this class, I refer to the functions in GGPlot as components.

The arguments for GGPlots functions can be thought of as subcomponents of the components.

The full list of functions in the GGPlot package is here:
https://ggplot2.tidyverse.org/reference/

And a helpful resource for many R packages is the cheat sheets – the most recent cheat sheet for GGPlot2 can be downloaded here:
https://github.com/rstudio/cheatsheets/blob/main/data-visualization.pdf

7 Opening the data file

The data file, Lansing2016NOAA.csv, contains weather information from the NOAA for 2016.

We will use ***read.csv()*** to open the file and save the data to a data frame named ***weatherData***. ***read.csv()*** reads in the data from the CSV file – and saves the data to a variable named ***weatherData***.

```
# read in CSV file and save the content to weatherData
weatherData = read.csv(file="data/Lansing2016NOAA.csv");
```

7.1 Viewing the data frame

weatherData is a data frame variable and it appears in the ***Environment*** tab under ***Data*** as 366 obs. of 29 variables. (i.e., 366 rows and 29 columns)

- Double-clicking on ***weatherData*** opens the data frame in the ***File Viewer*** window. This is a convenient way to visualize the data frame.
- Clicking on the arrow to the left of ***weatherData*** provides information about each of the columns

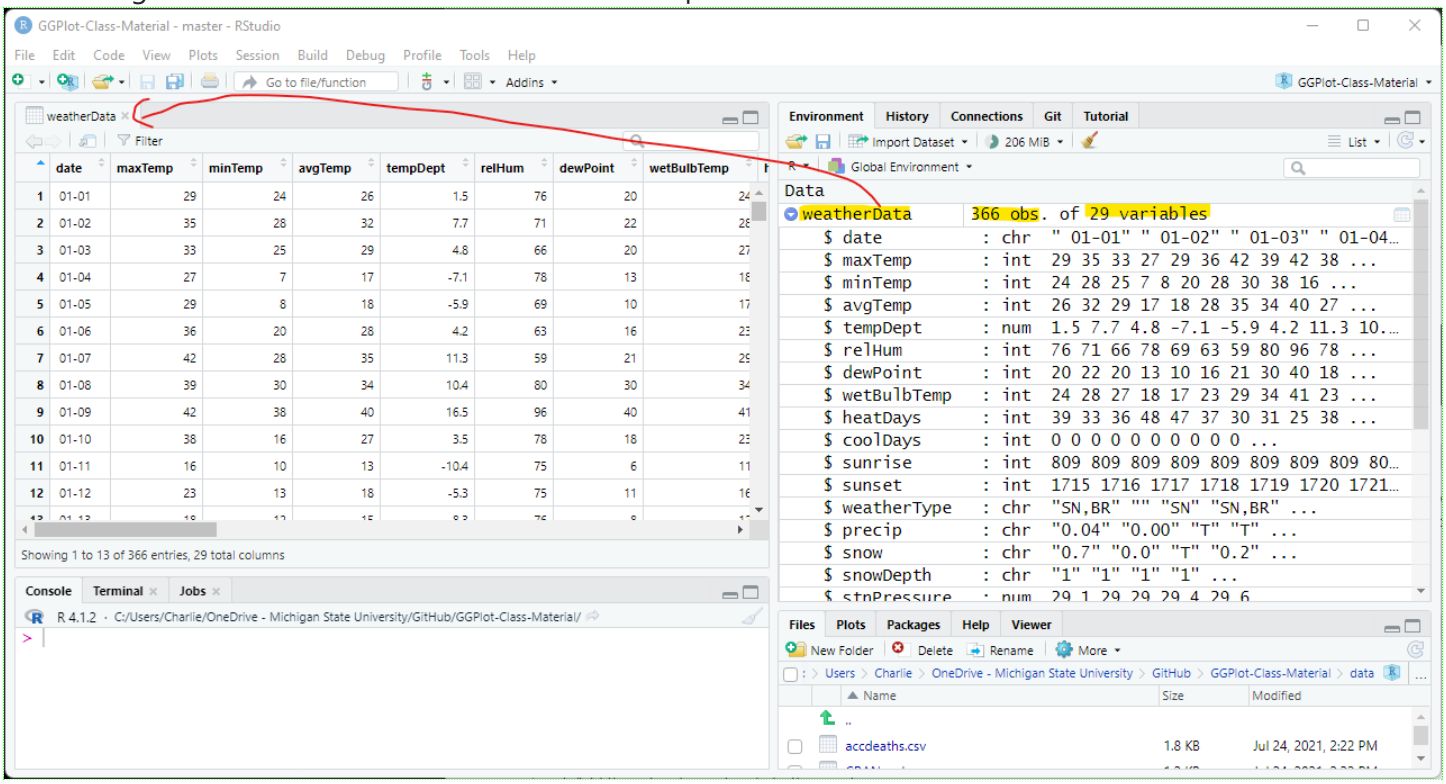

Figure 3: Viewing the data inside the data frame

8 Create plot data using GGPlot

Next, we will create a scatterplot of average temperature *(**avgTemp** column)* vs. humidity (***relHum*** column) from ***weatherData***.

The code to create a scatterplot using GGPlot is:

```
#### Part 1: Create a scatterplot ####
plot1 = ggplot( data=weatherData ) +
        geom_point( mapping=aes(x=avgTemp, y=relHum) );
plot(plot1);
```

Click the ***Source*** button the script and this plot appears:

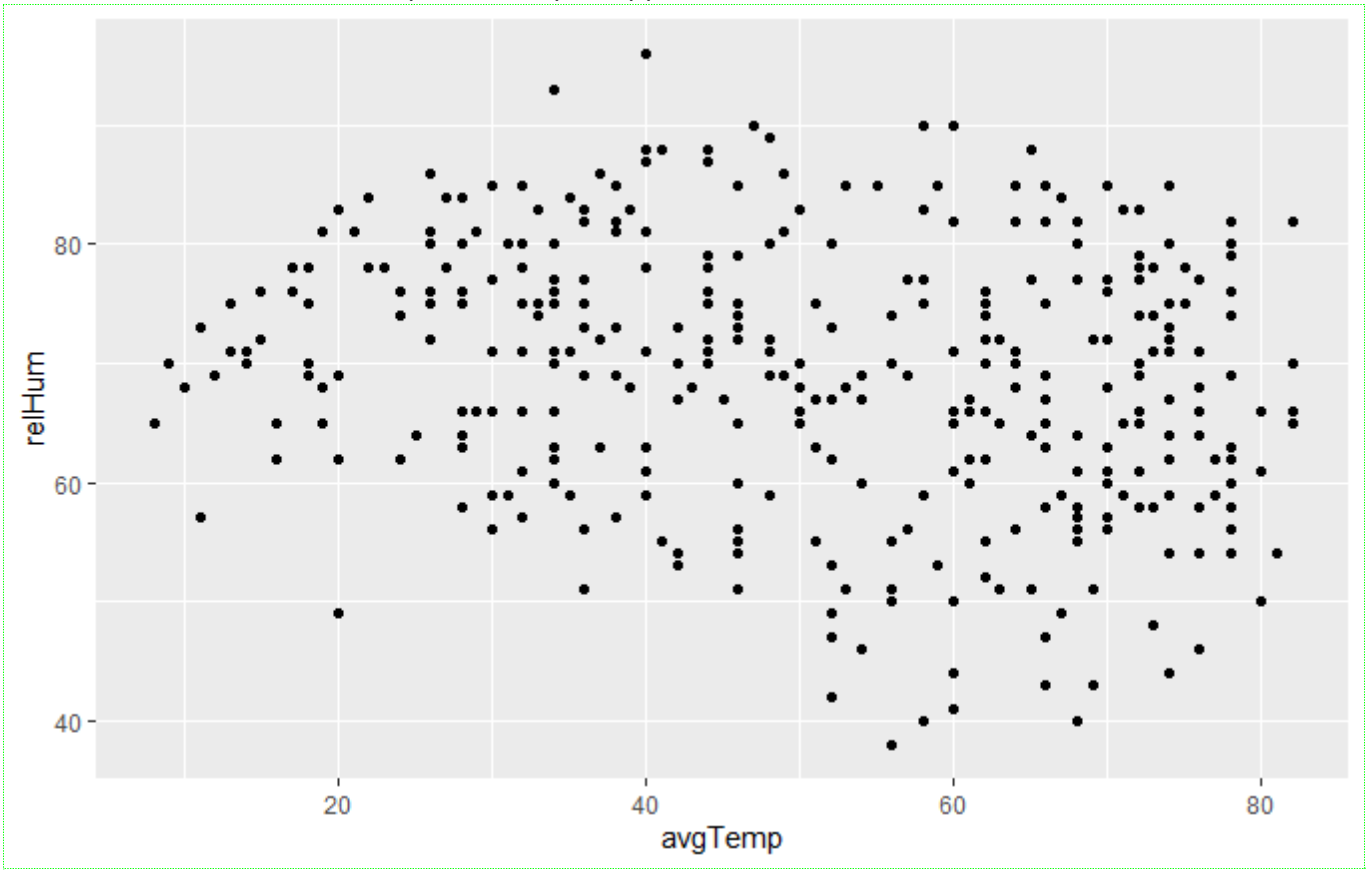

Figure 4: Our first plot using GGPlot with argument names highlighted

8.1 Taking out argument names

Below I highlight the argument names in the code:

```
#### Part 1: Create a scatterplot ####
plot1 = ggplot( data=weatherData ) +
        geom_point( mapping=aes(x=avgTemp, y=relHum) );
plot(plot1);
```

In the script below, the argument names are removed but it will still create the same plot:

```
#### Part 2: Same scatterplot without argument names ####
plot2 = ggplot( weatherData ) +
        geom_point( aes(avgTemp, relHum) );
plot(plot2);
```

8.2 Benefits of using argument names

This script without arguments works because, for this specific example, we

- only used the default arguments for each function
- used the arguments in the same order they appear in the function.

However, you should never code using these assumptions!

In this class, we will (almost always) use argument names because using argument names:

- makes the code more intuitive to the reader – and making code more intuitive should (almost always) take precedence over saving space.
- means that you can order the arguments however you want
- avoids bad assumptions about the ordering of arguments and their default values

The one exception where we will not use argument names is:

```
plot(plotData)   # no argument name here
```

instead of

```
plot(x=plotData) # x is the argument name
```

There are multiple functions in R and GGPlot where the argument name ***x*** is used generically as the name for the first argument in a function. This is not intuitive when plotting because ***x*** is also used to refer to data that maps to the x-axis.

We will use the argument name ***x*** when ***x*** refers to the x-axis (e.g., ***x=avgTemp***) but not when ***x*** is a generic name for the first argument (e.g., ***x=plotData***).

---

9 Components of a GGPlot

Let's take a more detailed look at the three lines of code that created the scatterplot.
The function ***ggplot()*** creates a canvas where the plots will be drawn. The argument ***data*** gives the data frame that will be used for the plots, ***weatherData***:

```
plot1 = ggplot( data=weatherData ) +
        geom_point( mapping=aes(x=avgTemp, y=relHum) );
plot(plot1);
```

Next, we add the component, ***geom_point()***, which creates a scatterplot using the ***avgTemp*** and ***relHum*** columns from ***weatherData***:

```
plot1 = ggplot( data=weatherData ) +
        geom_point( mapping=aes(x=avgTemp, y=relHum) );
plot(plot1);
```

The canvas is saved to a **List** variable named ***plot1:***

```
plot1 = ggplot( data=weatherData ) +
        geom_point( mapping=aes(x=avgTemp, y=relHum) );
plot(plot1);
```

And then ***plot()*** is used to display the canvas saved in ***plot1***:

```
plot1 = ggplot( data=weatherData ) +
        geom_point( mapping=aes(x=avgTemp, y=relHum) );
plot(plot1);
```

Extension: The yellow warning sign (which you might or might not see)

9.1 GGPlot components

In GGPlot, you initialize a canvas and then add components to the canvas. The (**+**) symbol is used to add components, and you can string multiple components together. In the above example, there is the initializing canvas function, ***ggplot()***, and one component, ***geom_point()***:

1) ***ggplot()*** is used to initialize a GGPlot canvas with the data from ***weatherData***:

```
plotData = ggplot( data=weatherData ) +
           geom_point( mapping=aes(x=avgTemp, y=relHum) );
```

2) ***geom_point()*** is a plotting component that creates a scatterplot

```
plotData = ggplot( data=weatherData ) +
           geom_point( mapping=aes(x=avgTemp, y=relHum) );
```

9.2 GGPlot mapping and aesthetics (aes)

All plotting components in GGPlot contain a subcomponent called **mappings**. ***mapping*** is used to describe the relationship between the data and the plot. Or, another way to put it, ***mapping*** defines what data gets represented on the plot (e.g., ***avgTemp*** and ***relHum***) and how the data gets represented (e.g., ***avgTemp*** on x-axis, ***relHum*** on y-axis):

```
plotData = ggplot( data=weatherData ) +
           geom_point( mapping=aes(x=avgTemp, y=relHum) );
```

The ***mapping*** is set to a mapping element called an ***aesthetic*** (***aes***). The concept of an aesthetic comes into play when we are generating legends and creating data categories, which we will talk about in future lessons. In the meantime, it is easier to just think of ***aes*** as a mapping element.

10 Adding more components to the canvas

Let's make the three following modifications to the plot:

1. add a title and change the axes labels
2. change the numeric tick marks on the y-axis
3. change the direction of the x-axis labels

To do this we will add *three new components* to the canvas:

1. ***labs()*** # label component
2. ***scale_x_continuous()*** # x-scaling component (there is a corresponding y-scaling component)
3. ***theme()*** # theme component

Note: you can also add more plotting components to the canvas (e.g., line plot, histogram) – this will be shown next lesson.

We add components using (**+**) and subcomponents are the arguments within the components:

```
plot3 = ggplot( data=weatherData ) +
  geom_point( mapping=aes(x=avgTemp, y=relHum) ) +
  labs( title="Humidity vs Temperature",
        subtitle="Lansing, MI -- 2016",
        x = "Average Temperatures (Fahrenheit)",
        y = "Relative Humidity") +
  scale_x_continuous( breaks = seq(from=10, to=80, by=10) ) +
  theme( axis.text.x=element_text(angle=90, vjust=0.5) );
plot(plot3);
```

Trap: putting the (+) on the next line

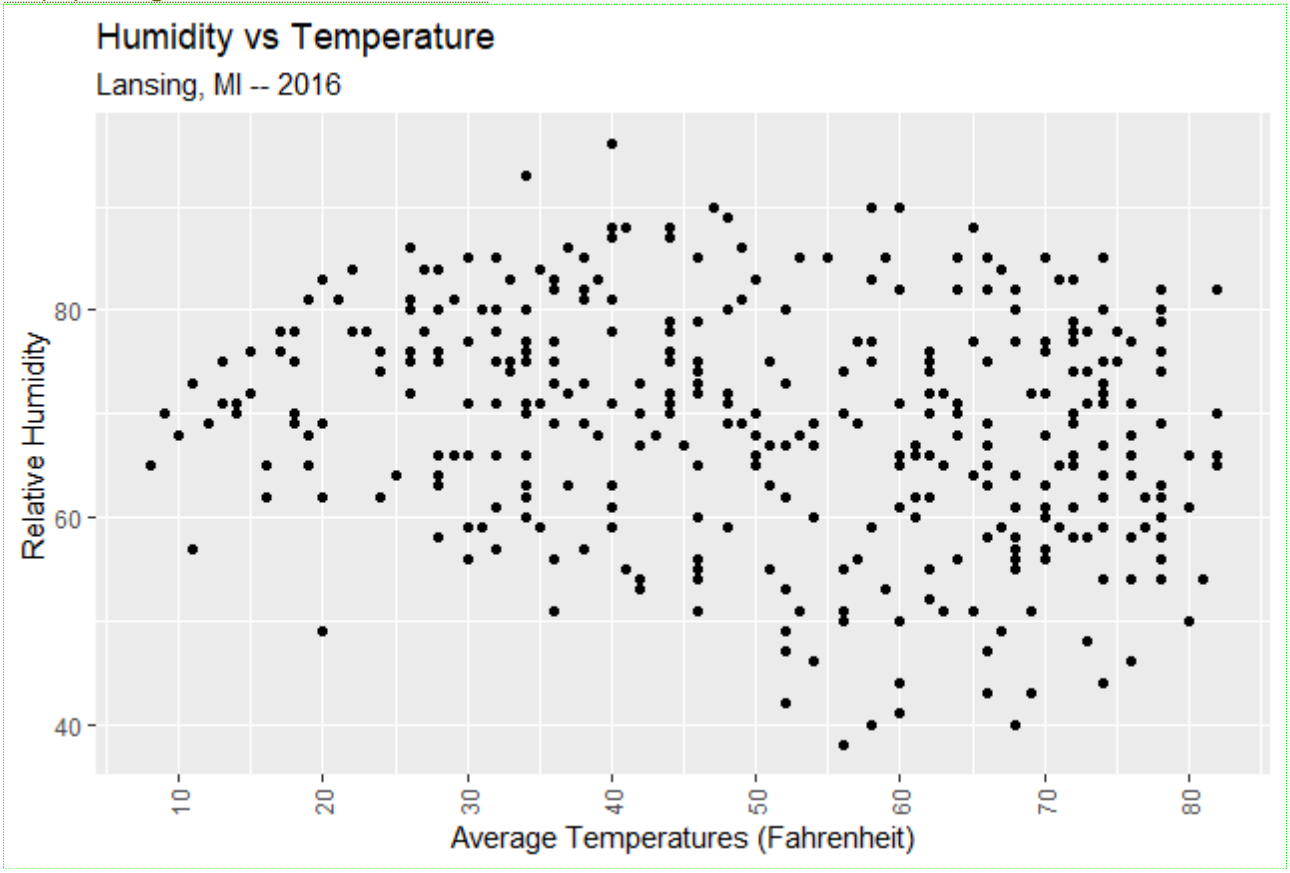

Figure 5: Scatterplot with a few added components

10.1 The Components in detail – labs()

```
    labs( title="Humidity vs Temperature",
          subtitle="Lansing, MI -- 2016",
          x = "Average Temperatures (Fahrenheit)",
          y = "Relative Humidity") +
```

When we search in the ***Help*** tab for ***labs()*** (Figure 6) we see that it has many subcomponents (or arguments) to change including:

- ***label***: the title
- ***subtitle***: a secondary title

A couple of notes about the information about ***labs()*** in the ***Help*** tab:

- There are many ways to add axes labels, ***labs()*** merges these methods into one component. Because of this, the ***Help*** section does not explicitly show the ***x*** and ***y*** arguments (although, the examples do). This is one area where the ***Help*** could do a better job reflecting the functionality of a function.
- ***waiver()*** is the default value given by the plotting function (i.e., ***waiver()*** does whatever the plotting functions thinks is best)

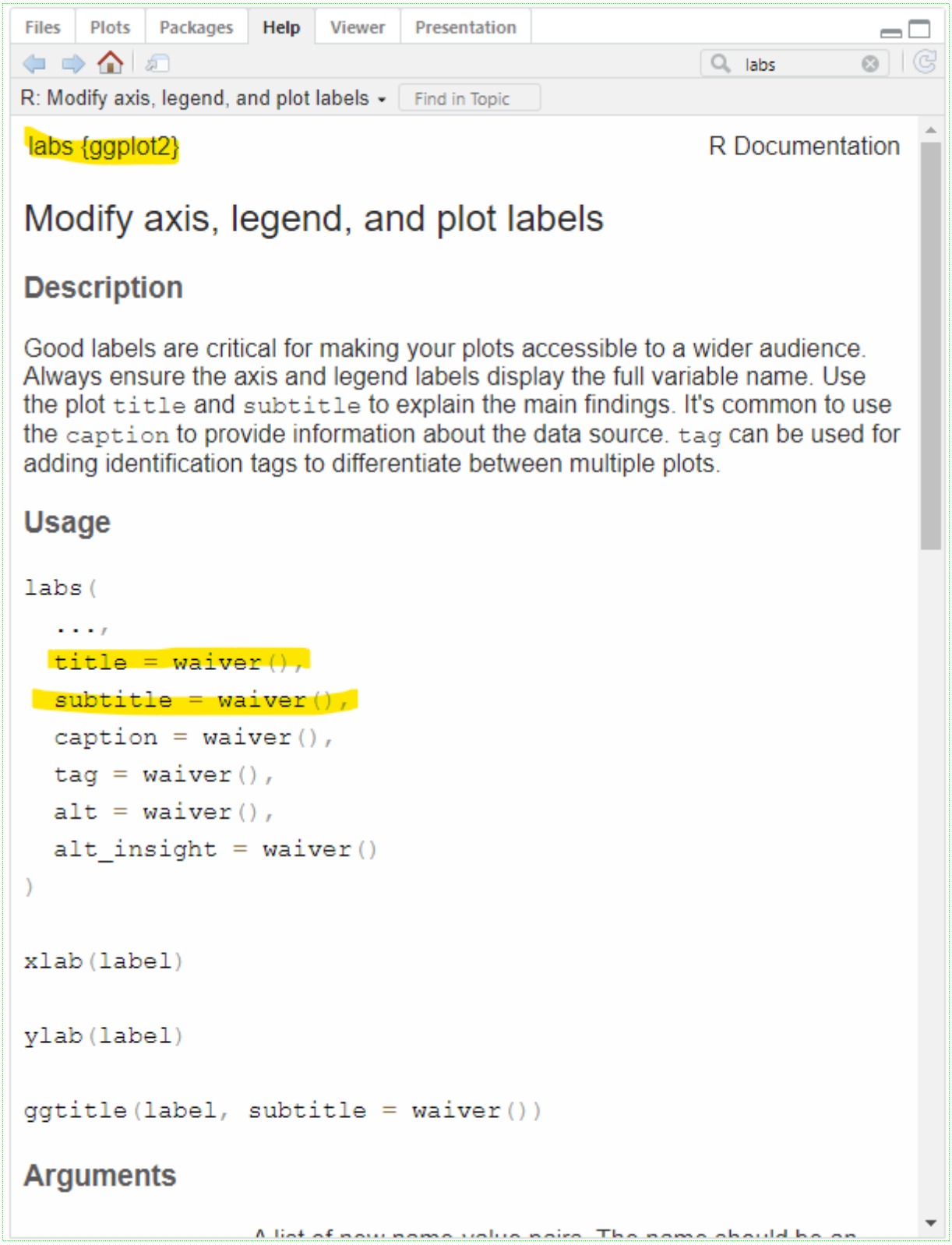

Figure 6: Using the Help Tab in RStudio to find info about GGPlot components

10.2 The Components in detail – scale_x_continuous()

```
scale_x_continuous( breaks = seq(from=10, to=80, by=10))
```

scale_x_continuous() is the component used when you want to modify an x-axis that has continuous values. There are many subcomponents (Figure 7) that can be changed in ***scale_x_continuous()*** and the corresponding ***scale_y_continuous()***. We modified one subcomponent, ***breaks***, by setting it to a sequence from **10** to **80** with numeric values place at intervals of **10**.

note: there is also a corresponding component named ***scale_x_discrete,*** which modifies x-axes with discrete values

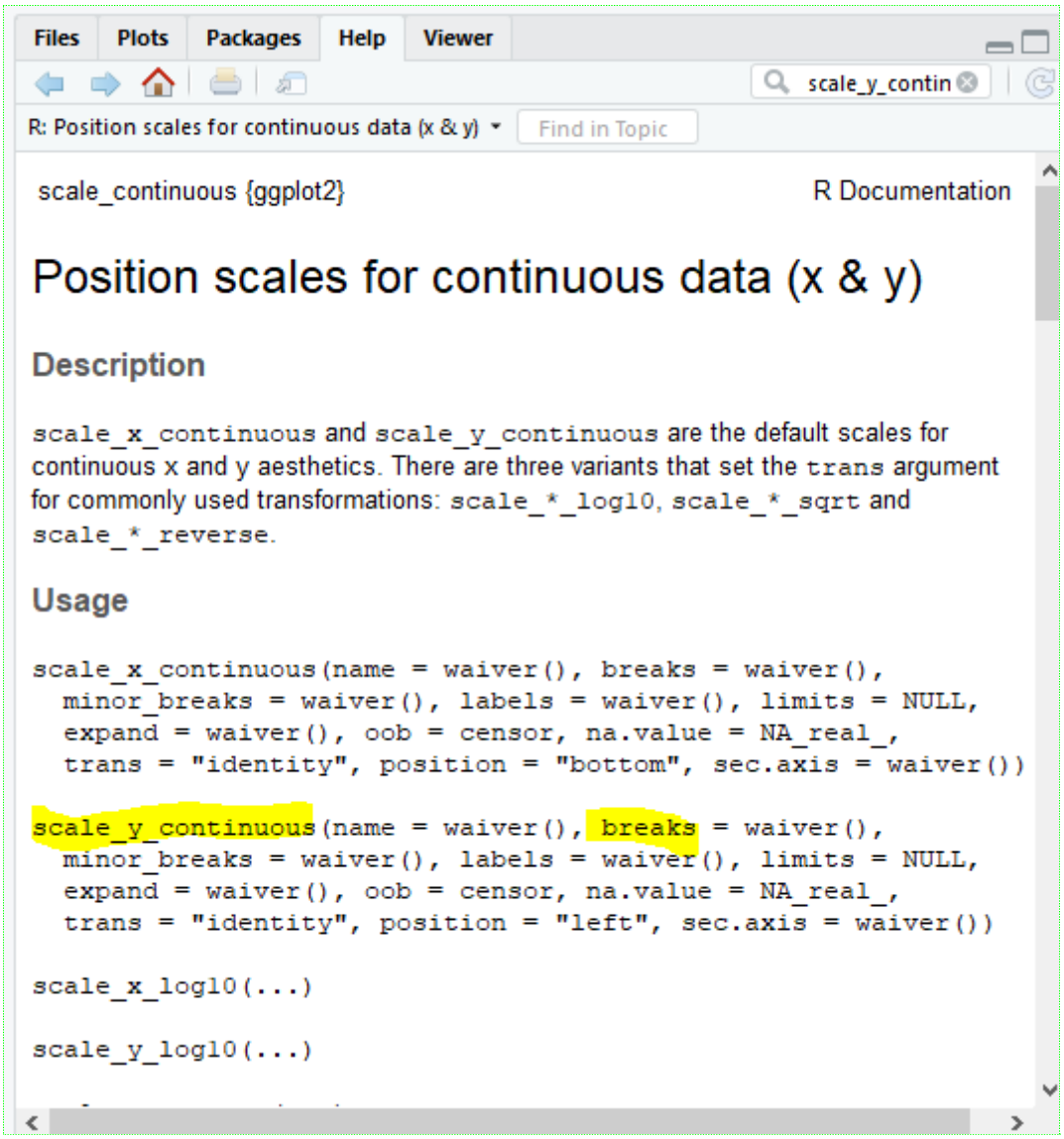

Figure 7: scale_x_continuous help page

10.3 The Components in detail – theme()

```
theme( axis.text.x=element_text(angle=90, vjust=0.5) )
```

In this example we changed one subcomponent in ***theme()*** called ***axis.text.x*** and set it to an ***element_text()*** that modifies the text by rotating it to an ***angle*** of **90** degrees and centering the text (***vjust=0.5***). Note: the default for **vjust is 1,** meaning the text will be vertically justified to the bottom. **vjust=0** means the text will be vertically justified to the top.

Broadly speaking, ***theme()*** is used to make modifications to the canvas (the plots and the background) that are not related to the data. ***theme()*** is probably the most used component in GGPlot and we could spend many lessons going through all the subcomponents of ***theme()***. We will be using ***theme()*** a lot more in future lessons and talking more about elements (e.g., ***element_text()***).

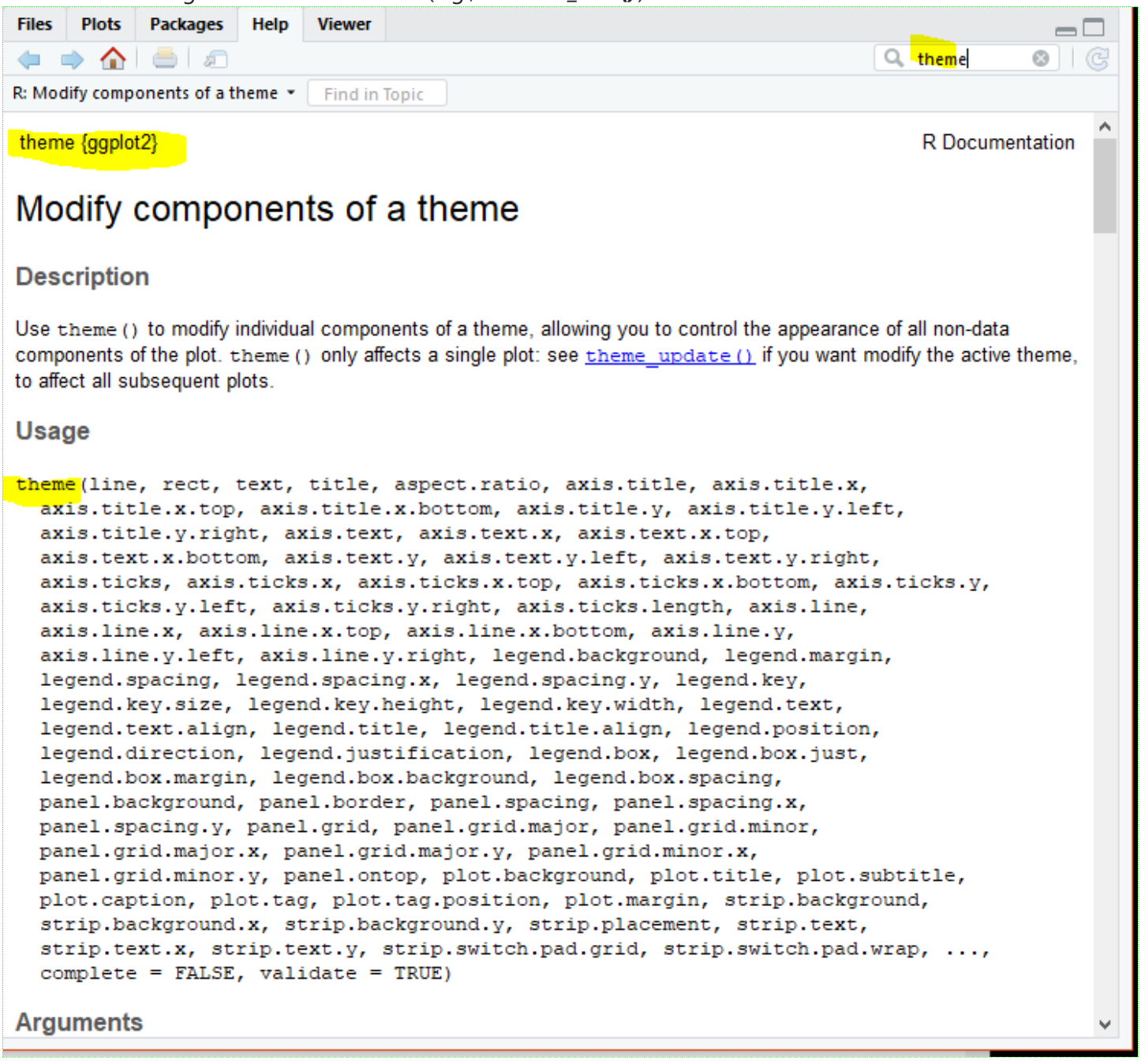

Figure 8: theme() component help page in the Help tab (yes, there is a lot there!)

10.4 For more help with components

A good place to find more information about components in GGPlot is the **Help** tab in the lower-right corner of RStudio (Figure 8). The ***Help*** tab provides information directly from https://ggplot2.tidyverse.org/reference/, which is the official webpage for GGPlot.

11 Getting rid of the grey (themes)

The default GGplot theme, which uses the gray background is not one of my favorite. Luckily, GGPlot makes it easy to change the theme. The components that do this, called ***complete themes***, are on this page. I will change to the black-white theme, ***theme_bw()***:

```
#### Part 4: Changing the theme ####
plot5 = ggplot( data=weatherData ) +
  geom_point( mapping=aes(x=avgTemp, y=relHum) ) +
  labs( title="Humidity vs Temperature",
        subtitle="Lansing, MI -- 2016",
        x = "Average Temperatures (Fahrenheit)",
        y = "Relative Humidity") +
  scale_x_continuous( breaks = seq(from=10, to=80, by=10) ) +
  theme_bw() +
  theme( axis.text.x=element_text(angle=90, vjust=0.5) );
plot(plot5);
```

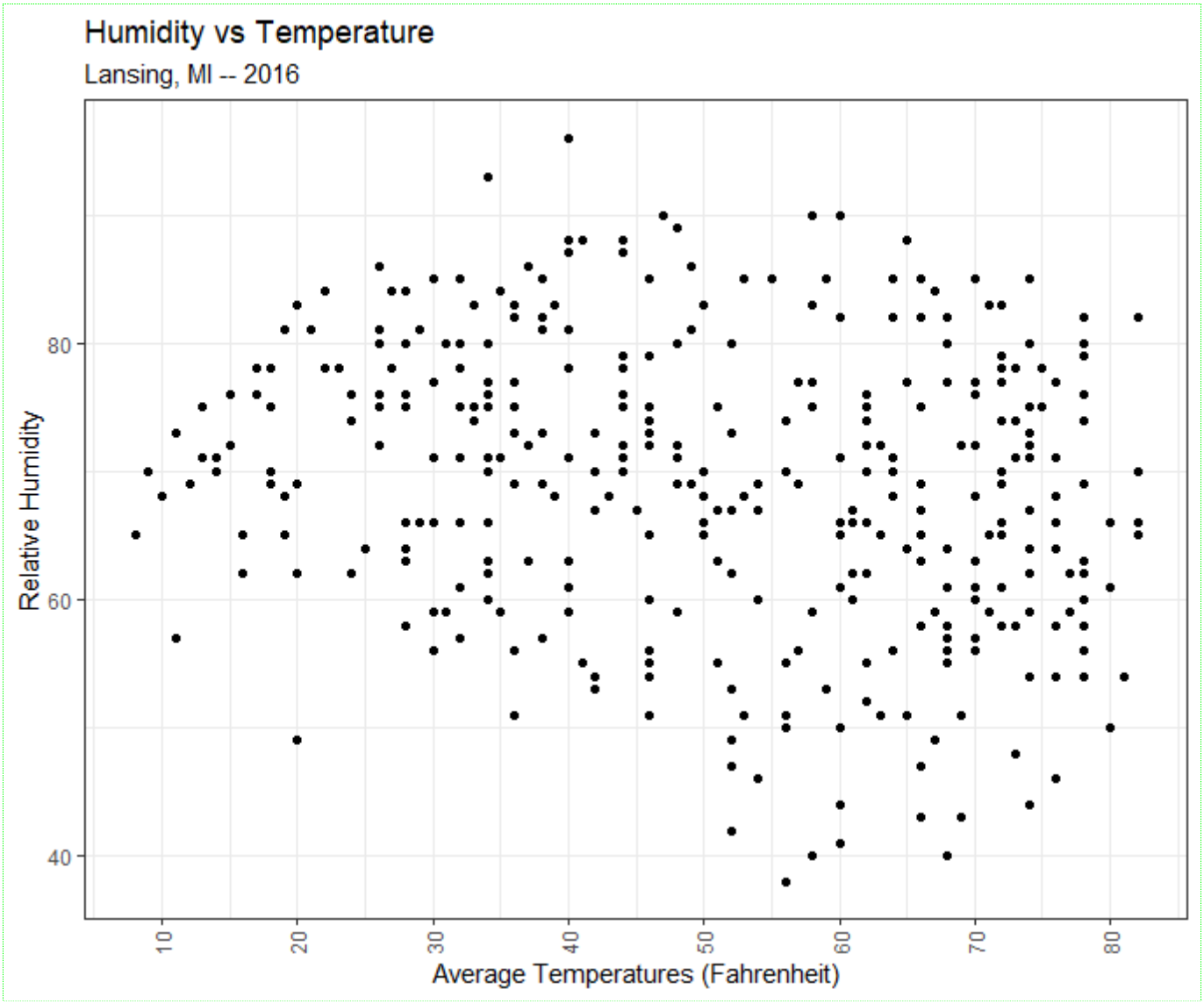

Figure 9: Setting the GGPlot theme to black and white

11.1 Complete themes must come before theme changes

A complete theme (e.g., **theme_bw()**) overwrites the whole theme for the canvas. This means it will overwrite any ***theme*** changes you previously made:

```
#### Part 5: Changing the theme --- oops, undoes theme ####
plot5 = ggplot( data=weatherData ) +
  geom_point( mapping=aes(x=avgTemp, y=relHum) ) +
  labs( title="Humidity vs Temperature",
        subtitle="Lansing, MI -- 2016",
        x = "Average Temperatures (Fahrenheit)",
        y = "Relative Humidity") +
  scale_x_continuous( breaks = seq(from=10, to=80, by=10) ) +
  theme( axis.text.x=element_text(angle=90, vjust=0.5) ) +
  theme_bw();  # this complete theme change will remove the theme change
       above
plot(plot5);
```

So, make sure you put your ***theme*** changes after your complete theme.

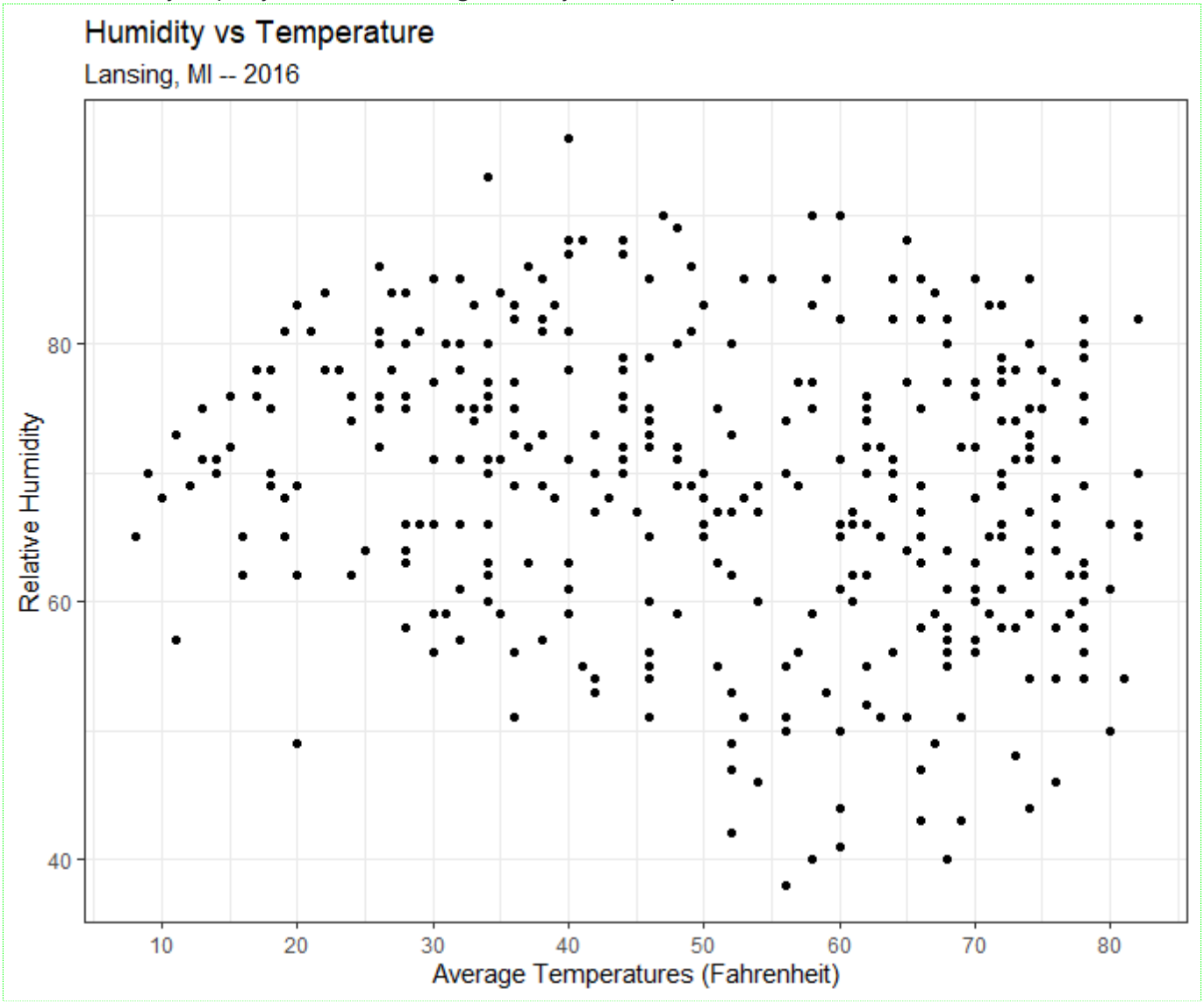

Figure 10: The complete theme change removed the theme change above it (the axis labels are no longer at 90 degrees)

12 Application

A) Looking at the GGPlot cheat sheet (or the GGPlot functions page) from Section 6, answer the following in comments in your script file

1. What component (function) would be used to create a text plot?
2. What component would you use to change the breaks on the x-axis if the values were in date format?

C) Create a scatterplot in GGPlot in the ***app1-02.r*** script:

1. Create a scatterplot of ***stnPressure*** vs ***windSpeed*** using the data from Lansing2016NOAA.csv
2. Add a title and labels the axes appropriately
3. Change the plot theme to one of your choice
4. Change the angle of the ***stnPressure*** axis labels to 45 degrees
5. Change the ***stnPressure*** breaks to go up by 0.1

6. Have the **wind-speed** axis only display three values: **3, 12, 21**
7. Challenge: Use the ***limits*** argument in ***scale_x_continuous*** and ***scale_y_continuous*** to remove the top and right parts of the plot where there are only a couple points.
    - ***limits*** is set equal to a vector with two values – so ***limits = c(0,100)*** would mean the axis would go from 0 to 100

If you have any questions regarding this application, feel free to email them to Charlie Belinsky at belinsky@msu.edu.

12.1 Questions to answer

Answer the following in comments inside your application script:

1. What was your level of comfort with the lesson/application?
2. What areas of the lesson/application confused or still confuses you?
3. What are some things you would like to know more about that is related to, but not covered in, this lesson?

12.2 Turn in on GitHub

Save the script as ***app02.r*** in your ***scripts*** folder, ***Push*** the changes to your GitHub repository, create an Issue in GitHub that says you have finished the application and assign the issue to ***belinskyc***.

13 Extension: RStudio Project windows

An RStudio Project takes the whole RStudio window – also called an RStudio Session. If you want to open up a second RStudio Project, you need to start a new RStudio Session (i.e., a new RStudio window). This can be done by clicking ***File -> Open Project in New Session...***

14 Trap: Using Excel to move files

On many computers, Microsoft Excel is the default application for opening CSV files – so double-clicking on a CSV file opens it in Excel. So, it is common for people to open a CSV file in Excel and then save it to a different folder.

There are a couple of issues with using Excel to move CSV files:

1. Some versions of Excel will ask you to save the file with an XLSX extension – make sure you ignore that. This will convert the file from a CSV to an XLSX, and the file will be unreadable in R if you use ***read.csv()***. There are packages that can read XLSX files but you are unnecessarily adding complexity.
2. Excel will occasionally change the format of a column. For instance, if you have a column with values that look like this: ***01-01, 01-02, 01-03*** then Excel will likely switch those values to dates like this: ***Jan-1, Jan-2, Jan-3***

You should not use Excel to move a CSV. Instead, use the system's File Explorer (Windows) / Finder (Mac) to move the file. You can also safely open the CSV file in RStudio and save it to another location.

15 Trap: Putting the (+) on the next line

The (+) commands strings together the components of a GGPlot. A common mistake is to put the (+) at the beginning of the following line:

```
source(file="scripts/reference.R");
packageData = read.csv(file="data/CRANpackages.csv");

plotData = ggplot( data=packageData )
    + geom_point( mapping=aes(x=Date, y=Packages) )
    + ggtitle(label="Packages in CRAN (2001-2014)")
    + scale_y_continuous(breaks = seq(from=0, to=6000, by=500))
    + theme(axis.text.x=element_text(angle=90, hjust=1));
plot(plotData);
```

This will result in an error and a surprisingly wise assessment of the problem fro the R debugger.

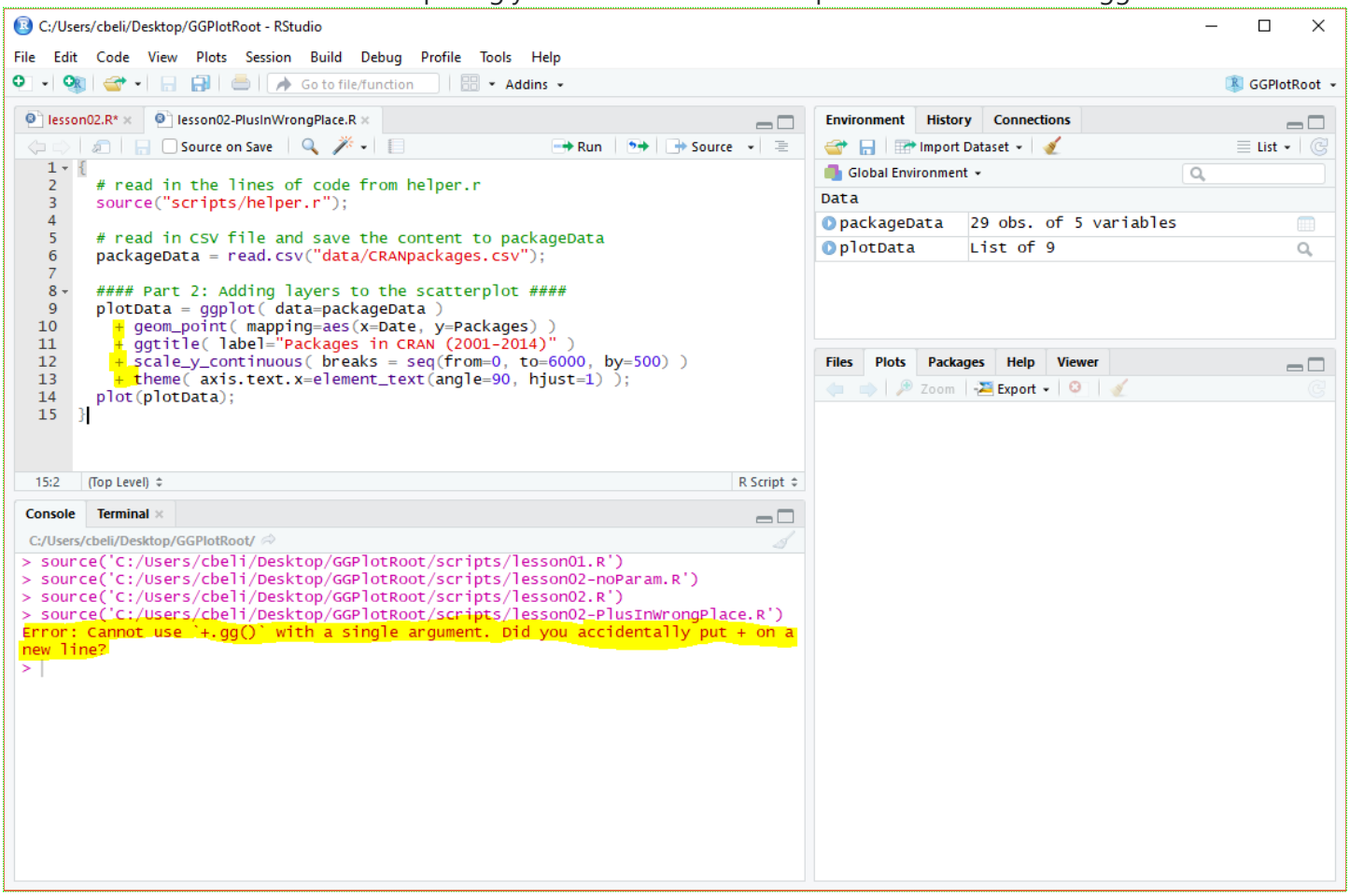

Figure 11: Error when putting the (+) on the next line

The reason for this error is that R thinks that line 5:

```
plotData = ggplot( data=packageData )
```

is a fully-formed and completed command

And R does not understand why line 6 starts a new command with a (**+**)

```
 + geom_point( mapping=aes(x=Date, y=Packages) )
```

A (**+**) at the end of a line tells R to append the next line to the current line. A (**+**) at the beginning of a line tells R to perform the mathematical operation addition.

16 Extension: The yellow warning sign

When you are working in GGPlot and have diagnostic features turned on in RStudio (located at ***Tools -> Global Options... -> Code -> Diagnostics***):

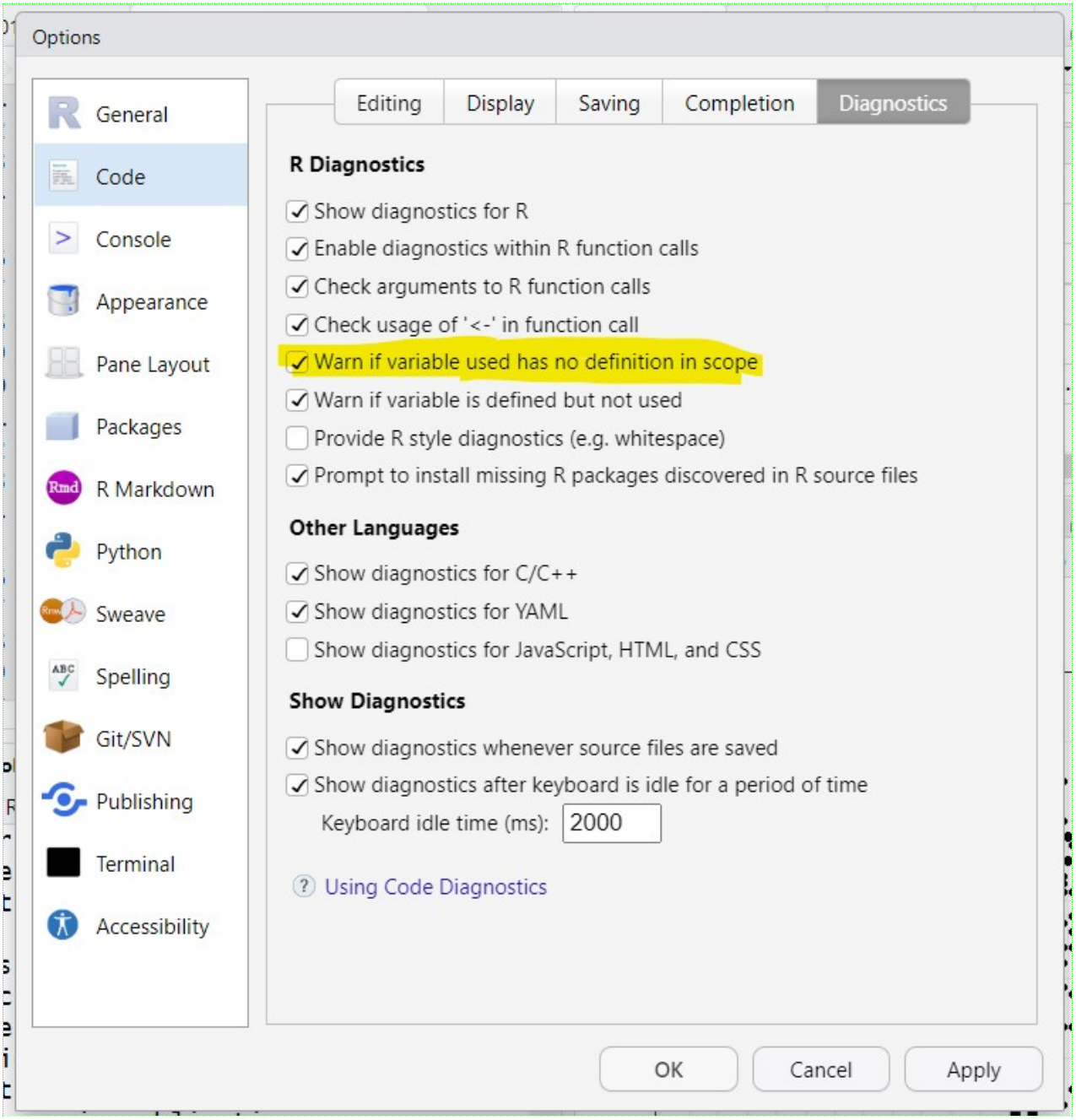

Figure 12: The diagnostic features in RStudio – the highlighted line is causing the scope warning

You will almost always see multiple yellow warning signs on the side of your code (Figure 13) The warning no symbol named 'Date' in scope means that RStudio does not recognize ***Date*** as a variable or a function. This is because ***Date*** is a variable within the GGPlot function ***geom_point()***, and the debugger is not sophisticated enough to always search through the GGPlot functions.

This is just a limitation of the RStudio debugger and does not reflect an actual issue.

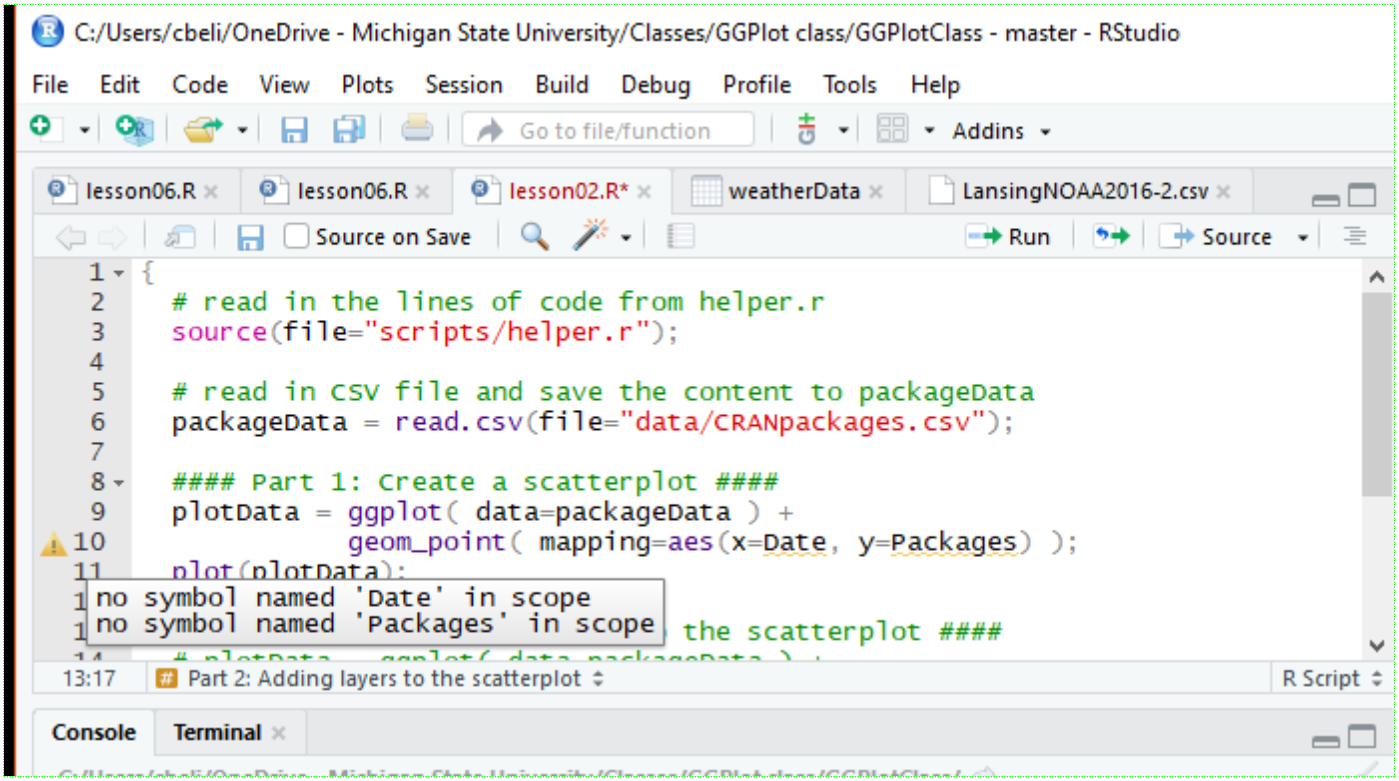

Figure 13: Warning about variables within the GGPlot functions