01-10: Multipanel Plots

1 - Purpose

- Using grep to find values that meet multiple conditions
- Arrange multiple plots on a canvas
- Customizing plots on a canvas

2 - Future changes

- Add info about how the RStudio plot window does not always generate the plots -- need to refresh or click the Zoom button.

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 );
```

3.1 - Add library for multiple plots

In this lesson we are going to put multiple plots on one canvas. To do this we must first install an R package called ***gridExtra***.

To install ***gridExtra*** in RStudio:

- go to ***Tools*** -> ***Install Packages***
- in the Packages textbox type in ***gridExtra*** then
- click ***Install***

You will also need to add the ***gridExtra*** library to your script, you can do that in ***reference.r*** by adding to the top:

```
library(package=gridExtra);
```

4 - Conditional comparisons of vector values

If this lesson, we are going to working with the ***weatherType*** column in ***weatherData***. ***weatherType*** consists of codes that indicate the type of weather events that occurred during the day. For instance **RA** means rain, **BR** means breezy, **HZ** means hazy. The weather events are separated by commas within the cell.

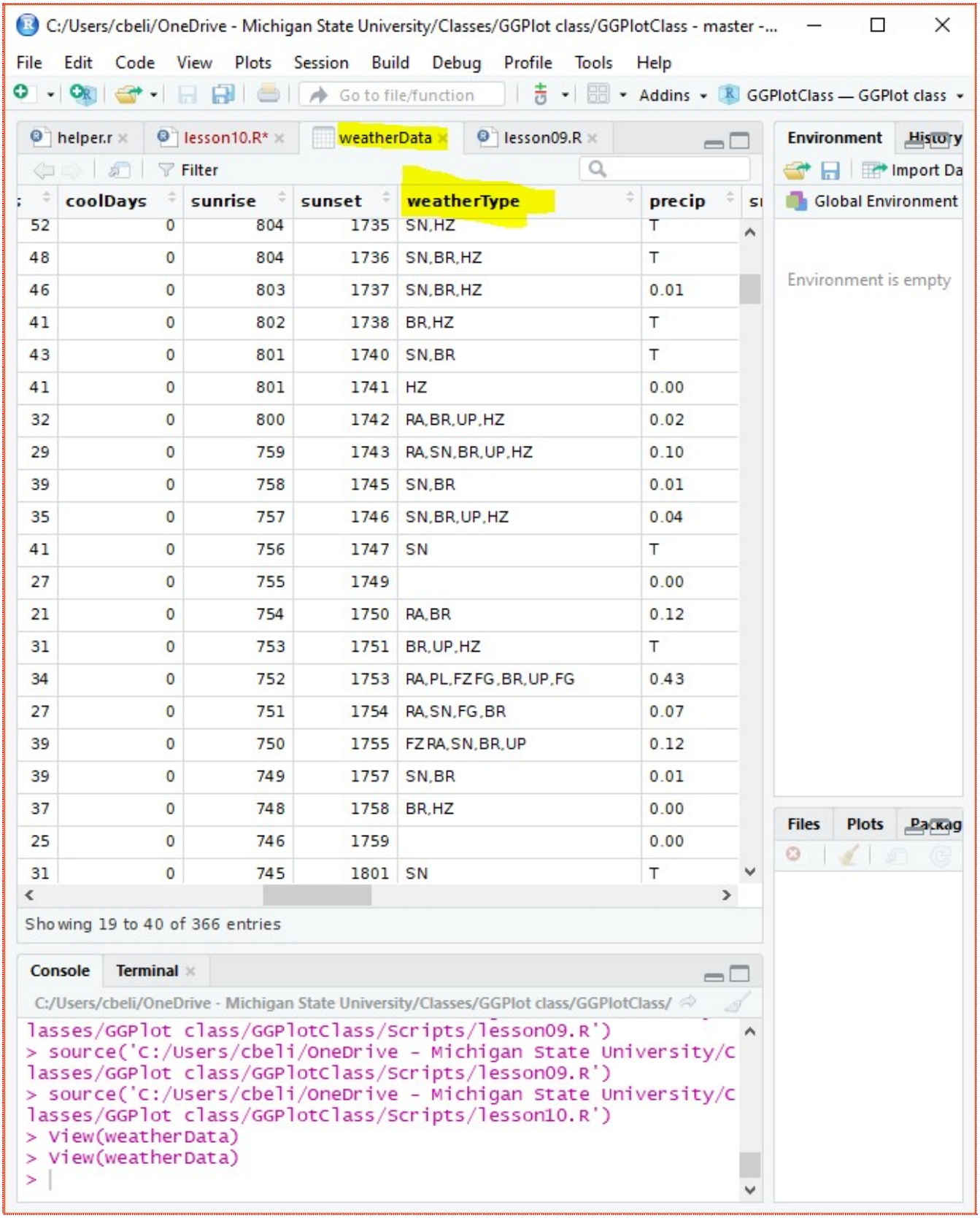

Fig 1: ***weatherType*** *column in* ***weatherData***

The problem with the way ***weatherType*** is presented is that it makes it hard to find days with one specific weather condition (like rain, ***RA***). The following code will only give you the index values of days that were exclusively rainy (i.e., no other weather event except for **RA**):

```
which(weatherType == "RA") # find rainy days
```

A common way to deal with this situation is to break the column up into multiple columns, each representing one weather condition (**RA**, **HZ**, **BR**, etc...). A more robust way is to use ***grep()*** to find the codes within the column.

4.1 - grep() to find weather event

We can use ***grep()*** to find patterns within a column. In this case, patterns that match the characters **RA** and **BR** (rainy and breezy):

```
#### Part 1: using grep to find days with a specific weather event
rainyDays = grep(weatherData$weatherType, pattern="RA");   # any day with rain
breezyDays = grep(weatherData$weatherType, pattern="BR");  # any breezy day
```

rainyDays has **124** values, meaning there were **124** days with some rain (the code **RA**)
breezyDays has **172** values, meaning there were **172** days with strong winds (the code **BR**)

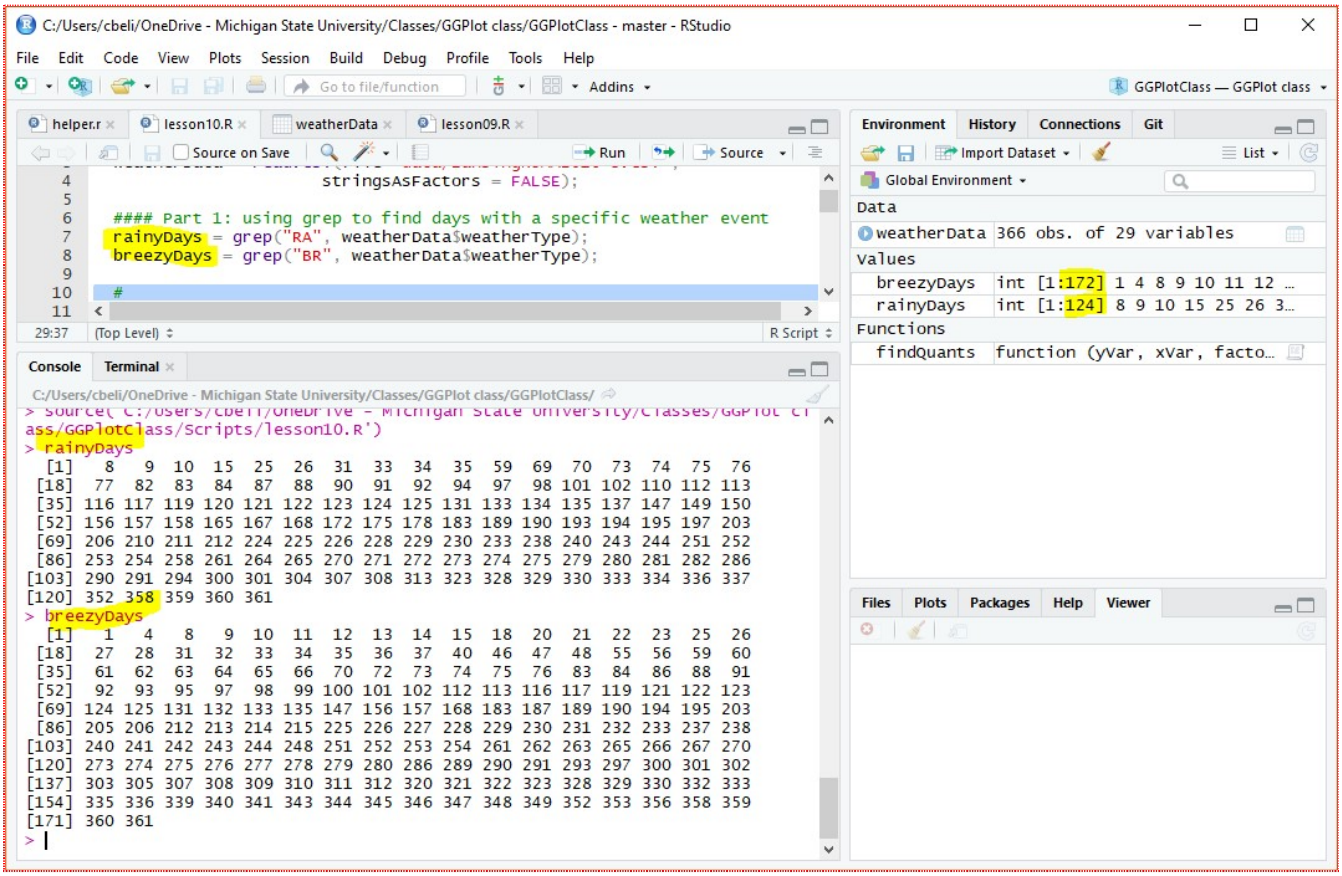

Fig 2: Finding the index values of the days with rain and those that were breezy

4.2 - Plotting a subset of values

We will now plot humidity vs temperature for only the **172** days the were breezy using ***breezyDays*** to subset of ***weatherData***:

```
#### Part 2: Scatterplot for Humidity vs. Temperature on breezy days
plot1 = ggplot(data=weatherData[breezyDays,]) +
```

```
        geom_point(mapping=aes(x=avgTemp, y=relHum)) +
        theme_classic() +
        labs(title = "Humidity vs. Temperature (Breezy Days)",
             subtitle = "Lansing, Michigan: 2016",
             x = "Degrees (Fahrenheit)",
             y = "Relative Humidity");
plot(plot1);
```

We have a plot with 172 points, matching the 172 days it was breezy (or, the 172 index values in ***breezyDays***).

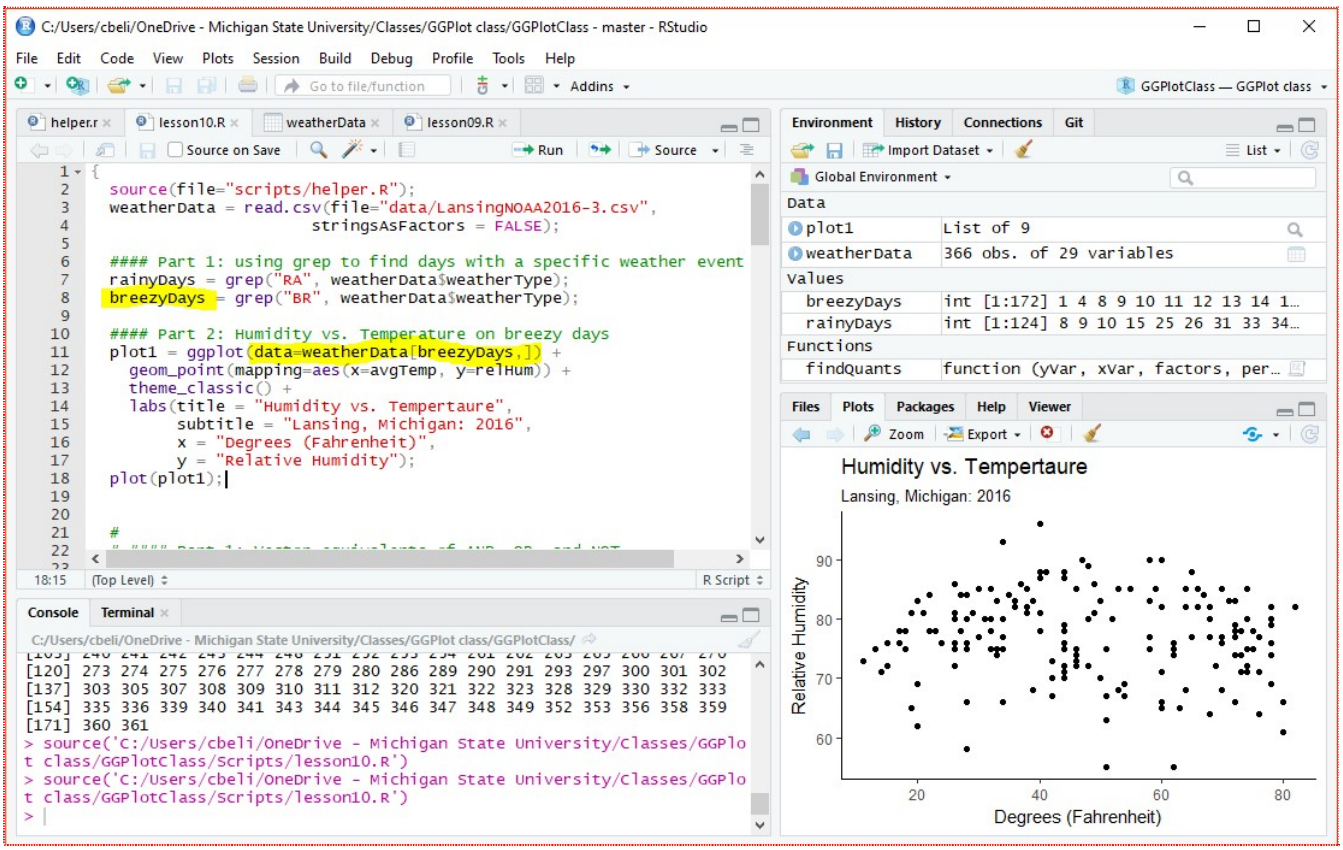

Fig 3: Scatterplot of humidity vs. temperature on breezy days

5 - Combining conditions

Next we want to find index values for various combinations of weather events for example:

- days with two events (e.g., rainy **and** breezy)
- days with at least one of two events (e.g., rainy **or** breezy)
- days with exclusively one event (e.g., rainy **and not** breezy)

R has three functions, called *set operations*, that can perform the above tasks:

- ***intersect()*** -- the vector equivalent of an **AND** statement (rainy ***&&*** breezy)
- ***union()*** -- the vector equivalent of an **OR** statement (rainy **||** breezy)
- ***setdiff()*** -- the vector equivalent of an **AND NOT** statement (rainy ***&& !***breezy)

The following code creates a vector that contains the index values for days that were *both rainy and windy*:

```
  rainyAndBreezy = intersect(rainyDays, breezyDays);
```

Let's code for the four possible combinations of windy and rainy conditions:

```
#### Part 3: Combine event using set operations
rainyAndBreezy = intersect(rainyDays, breezyDays); # days with rain AND wind
rainyOrBreezy = union(rainyDays, breezyDays);      # days with rain OR wind
rainyNotBreezy = setdiff(rainyDays, breezyDays);   # days with rain but NOT wind
breezyNotRainy = setdiff(breezyDays, rainyDays);   # days with wind but NOT rain
```

We can see that there were **82** days that were breezy but not rainy represented by the **82** index values in ***breezyNotRainy***. There were **34** days that were rainy but not breezy, represented by the **34** index values in ***rainyNotBreezy*** -- the first of those days being the 69th day of the year, which is in early April.

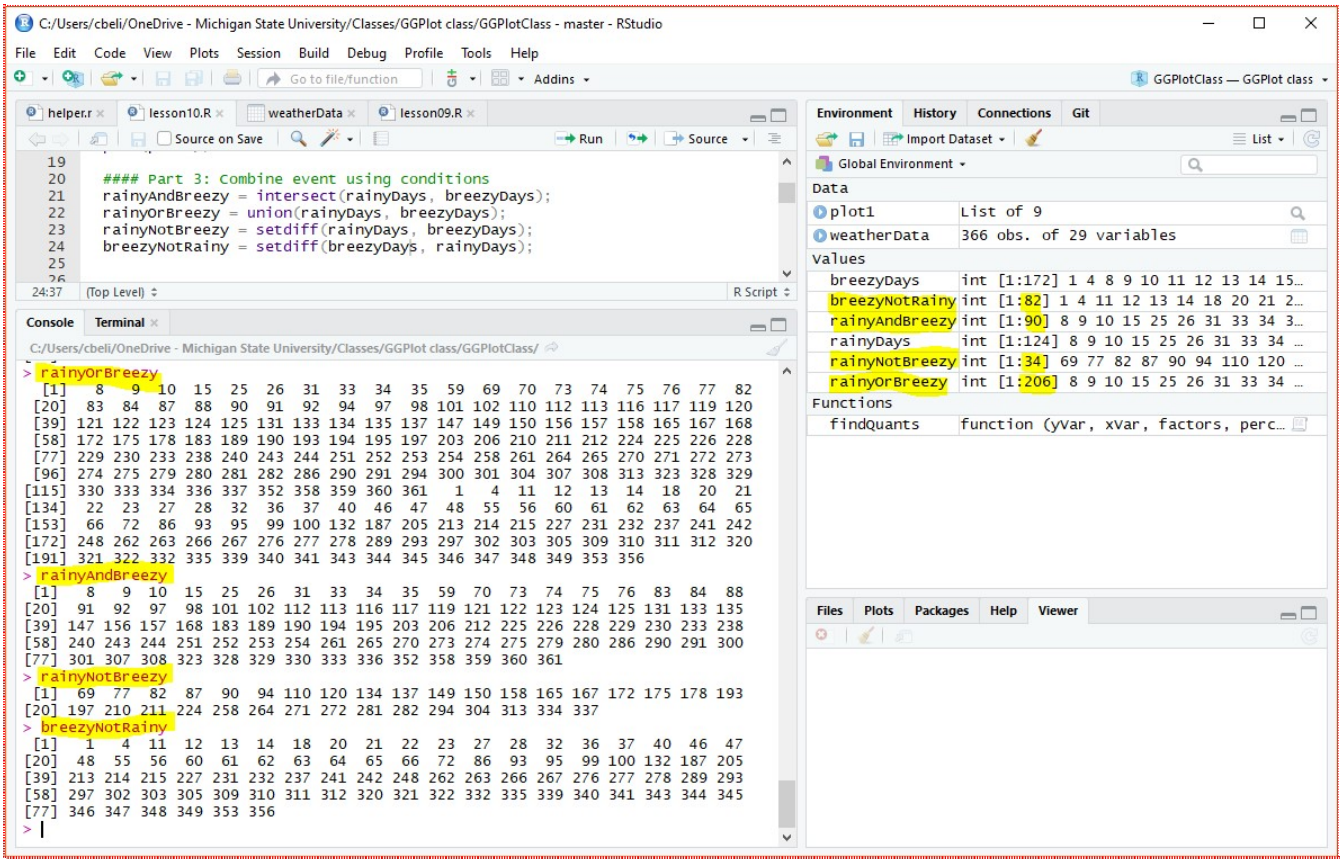

Fig 4: Vectors that contain the indexes for combined weather events

Extension: Code for days that were neither rainy nor windy

6 - Set up for multiple plots

We now have six combinations of the weather events **rainy** and **breezy** to plot:

1. breezy days (plotted in *Fig 3*)
2. rainy days
3. rainy AND breezy days
4. rainy OR breezy days
5. rainy AND NOT breezy days

6. breezy AND NOT rainy days

6.1 - Create plot data

We have already created the plot data for breeezy days (#1). We are now going to create the plot data for combinations #2 through #6. *Note: we are not printing the plots out yet -- we are just creating the data for the plots.*

```
#### Part 4: Creating plots for all rainy day/breezy day combinations
plot2 = ggplot(data=weatherData[rainyDays,]) +
        geom_point(mapping=aes(x=avgTemp, y=relHum)) +
        theme_classic() +
        labs(title = "Humidity vs. Temperature (breezy days)",
             subtitle = "Lansing, Michigan: 2016",
             x = "Degrees (Fahrenheit)",
             y = "Relative Humidity");

plot3 = ggplot(data=weatherData[rainyAndBreezy,]) +
        geom_point(mapping=aes(x=avgTemp, y=relHum)) +
        theme_classic() +
        labs(title = "Hum vs. Temp (Rainy AND Breezy)",
             subtitle = "Lansing, Michigan: 2016",
             x = "Degrees (Fahrenheit)",
             y = "Relative Humidity");

plot4 = ggplot(data=weatherData[rainyOrBreezy,]) +
        geom_point(mapping=aes(x=avgTemp, y=relHum)) +
        theme_classic() +
        labs(title = "Hum vs. Temp (Rainy or Breezy)",
             subtitle = "Lansing, Michigan: 2016",
             x = "Degrees (Fahrenheit)",
             y = "Relative Humidity");

plot5 = ggplot(data=weatherData[rainyNotBreezy,]) +
        geom_point(mapping=aes(x=avgTemp, y=relHum)) +
        theme_classic() +
        labs(title = "Hum vs. Temp (Rainy and NOT Breezy)",
             subtitle = "Lansing, Michigan: 2016",
             x = "Degrees (Fahrenheit)",
             y = "Relative Humidity");

plot6 = ggplot(data=weatherData[breezyNotRainy,]) +
        geom_point(mapping=aes(x=avgTemp, y=relHum)) +
```

```
        theme_classic() +
        labs(title = "Hum vs. Temp (Breezy and NOT Rainy)",
             subtitle = "Lansing, Michigan: 2016",
             x = "Degrees (Fahrenheit)",
             y = "Relative Humidity");
```

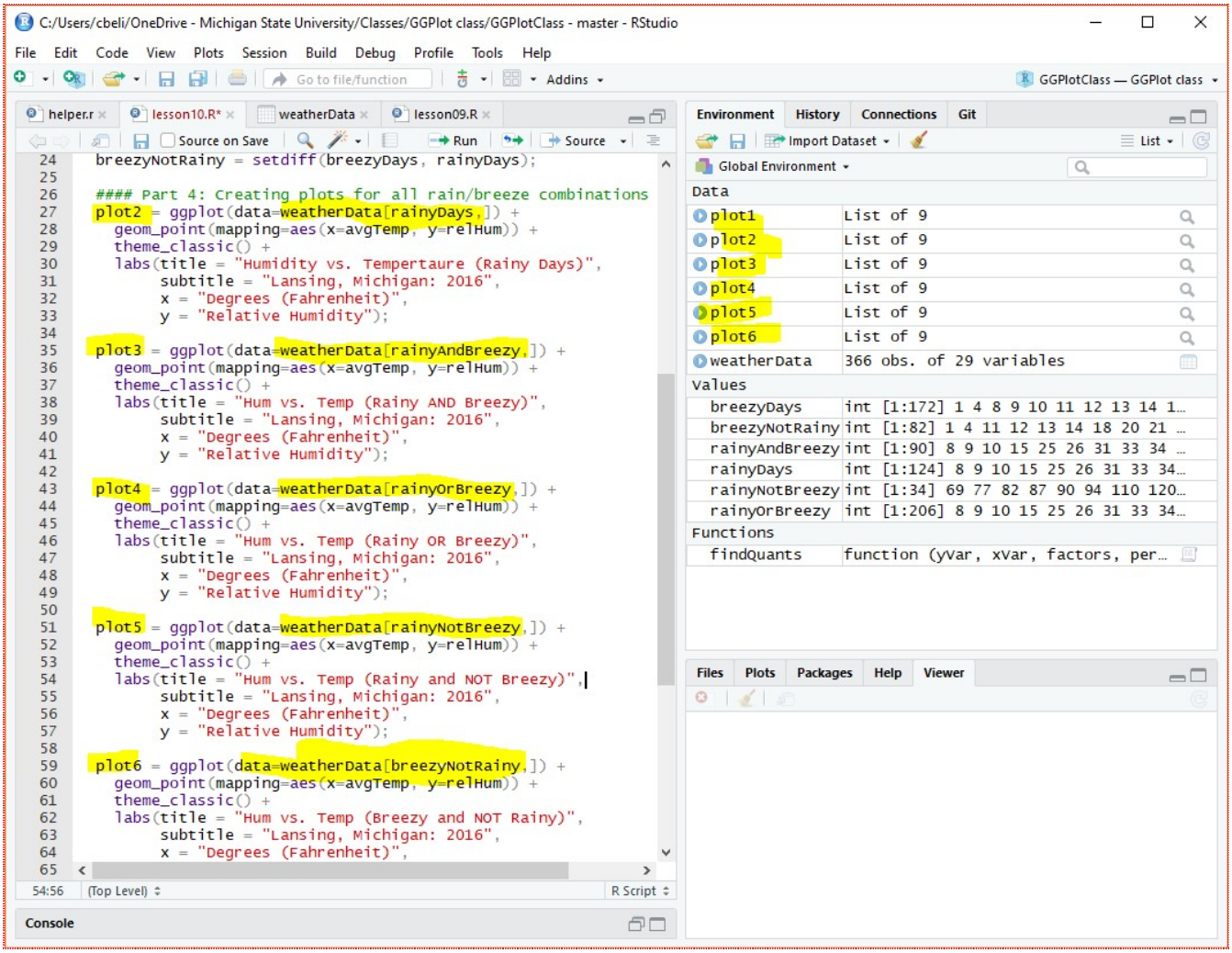

Fig 5: Creating the plot data for each event combination

7 - Multiple plots on one canvas

We have not created any plots yet, we have created the data for six different plots. Up until now, we have used ***plot()*** to put one plot on a canvas or we added **facets** to place multiple plots on one axis. Now, we are going to use ***grid.arrange()*** in the ***gridExtra*** library to *create a canvas with multiple plots*.

There is documentation for ***grid.arrange()*** but it is not very intuitive -- however, you should know that ***grob*** stands for ***gr***aphical ***ob***jects, and plots are considered graphical objects.

We are going to arrange the six plots created above (*Fig 5*) on a canvas using three different methods:

- by rows
- by columns
- customized using a matrix

7.1 - Plots in rows

The plots that will be added to the canvas must go first in ***grid.arrange()***. You can put as many plots in as you want separated by commas. After you put in all the plot names, you need to set parameters for the canvas.

For this canvas, we set the number of rows parameter, or ***nrow*** to **3**. This means ***grid.arrange()*** will place the plots listed into 3 rows (using as many columns as necessary). Since there are six plots, each of the 3 rows will have 2 plots.

```
#### Part 5: Arranging plots on one canvas by rows
grid.arrange(plot1, plot2, plot3, plot4, plot5, plot6,
             nrow=3);
```

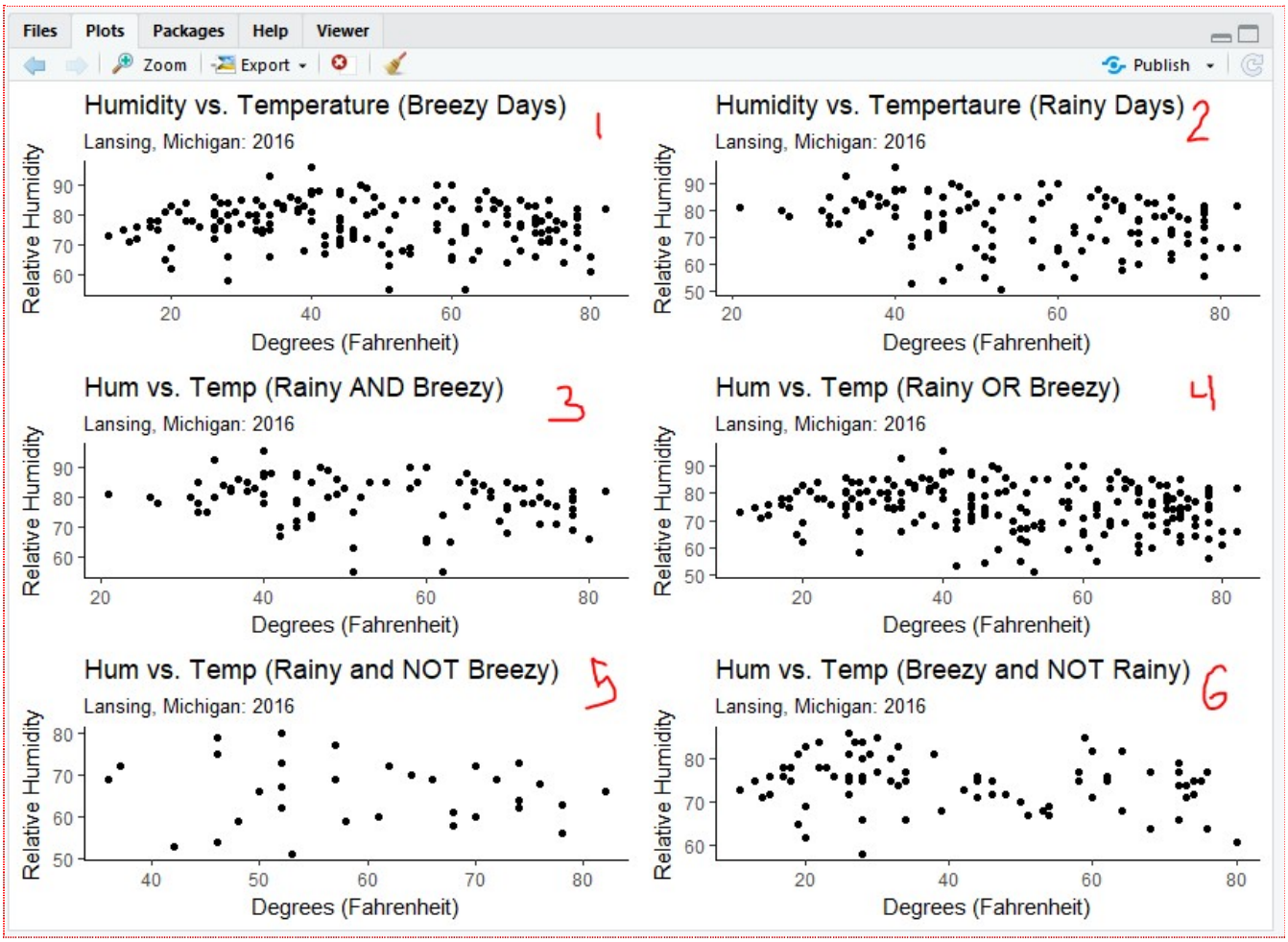

Fig 6: Arranging plots by rows in a canvas

7.2 - Plots in columns

Arranging plots by column works almost the same way as arranging by rows except we use the parameter ***ncol.***

For this canvas we are going to put the plots in the reverse order and skip the last plot (***plot1***) -- so there

are only five plots. Since we set ***ncol*** to **3** and there are **5** plots, ***grid.arrange()*** creates two rows but leaves an empty space at the end of the last row.

```
#### Part 6: Arranging plots on canvas by columns
grid.arrange(plot6, plot5, plot4, plot3, plot2,
             ncol=3);
```

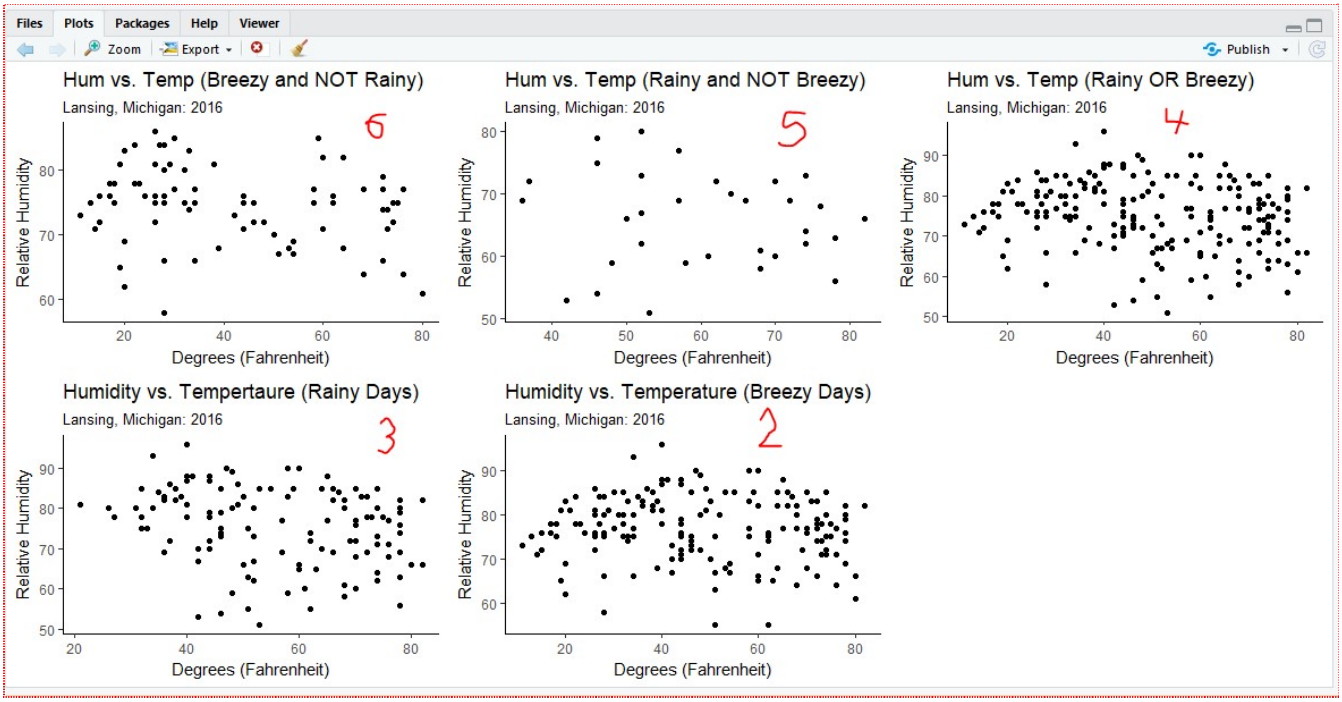

Fig 7: Arranging plots by columns in a canvas

7.3 - Customized canvas

The parameter in ***grid.arrange()*** that is probably most used is ***layout_matrix***. ***layout_matrix*** allows more flexibility in the plot layout.

layout_matrix is set to a matrix with numbers representing the plots. The rows and columns of the matrix (with their corresponding plot numbers) match the canvas layout of the plots.

In the following example:

- All 6 plots are used
- The first row on the canvas will have the 4th, 5th, and 6th plots (***plot4***, ***plot5***, and ***plot6***)
- The second row on the canvas will have the 3rd, 2nd, and 1st plots (***plot3***, ***plot2***, and ***plot1***)

```
#### Part 7: Customize arrangements using matrix
grid.arrange(plot1, plot2, plot3, plot4, plot5, plot6,
             layout_matrix = rbind(c(4,5,6),
                                   c(3,2,1)));
```

Note: The numbers in the matrix represent the order the plots are listed.

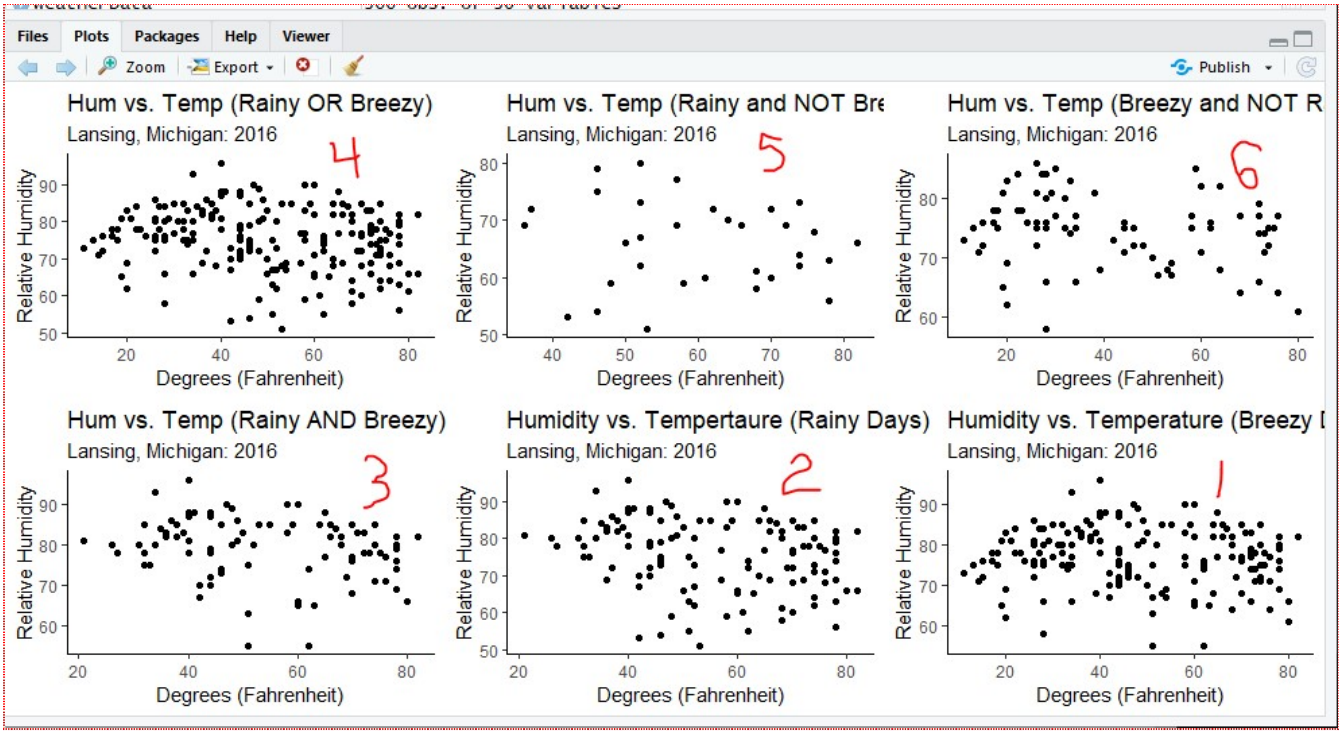

Fig 8: The 6 plots using the matrix layout

7.4 - Empty spaces within the canvas

We can use ***NA*** to represent an empty space (i.e., no plot).

The code:

```
layout_matrix = rbind(c(NA,1,2),     # row 1: nothing, 1st plot, 2nd plot
                      c(3,NA,NA)));  # row 2: 3rd plot, nothing, nothing
```

Tells ***grid.arrange()*** to create a canvas that has 2 rows and 3 columns.

- Row 1 has nothing in the first column, the 1st plot in the second column, and the 2nd plot in the third column
- Row 2 has the 3rd plot in the first column and nothing in the second or third column

In the following code, ***plot3*** *is the 1st plot*, ***plot4*** is the 2nd plot, and ***plot5*** is the 3rd plot:

```
#### Part 8: Add empty spaces to customized arrangement
grid.arrange(plot3, plot4, plot5,
             layout_matrix = rbind(c(NA,1,2),
                                   c(3,NA,NA)));
```

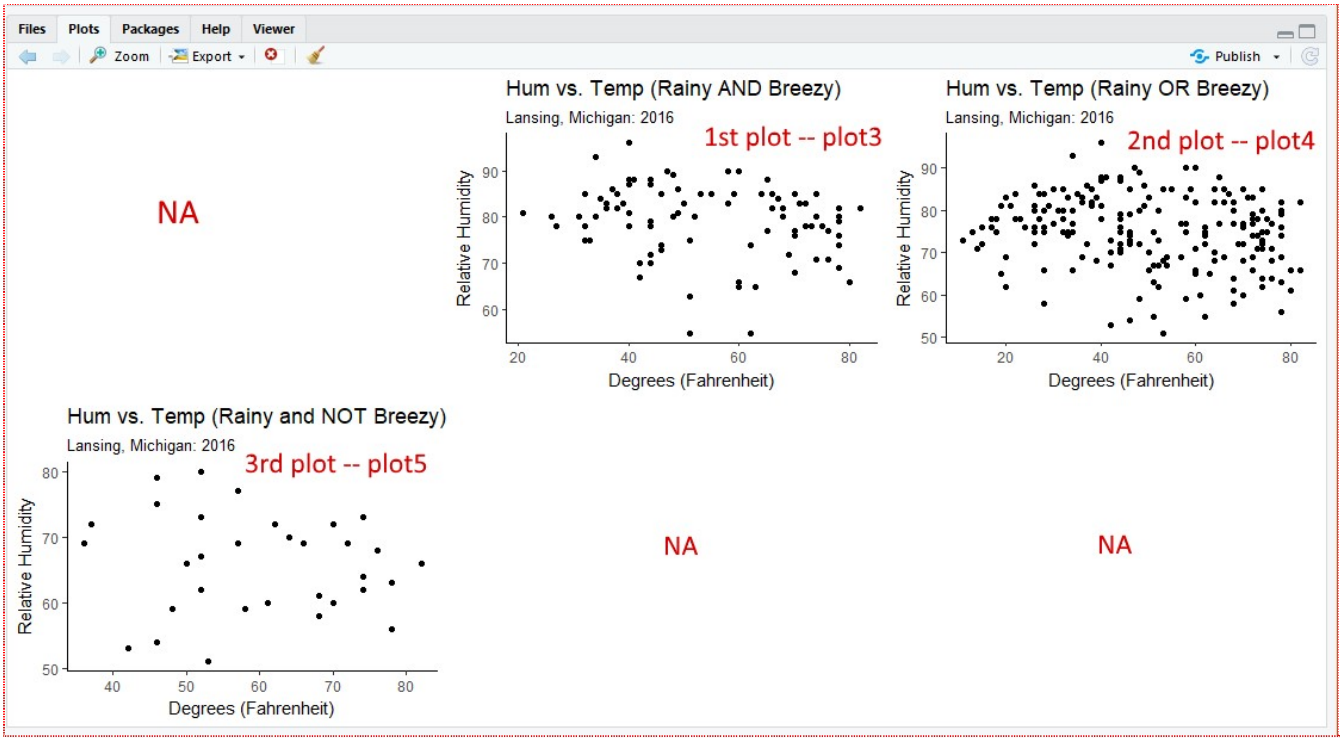

*Fig 9: Using **layout_matrix** to customize the canvas*

7.5 - Sizing plots

We can also use ***layout_matrix*** to resize plots by extending them across rows and columns.

The following code makes the:

- 1st plot 2 columns by 2 rows,
- 2nd plot 1 column by 1 row
- 3rd plot 2 columns by 1 row
- 4th plot 1 column by 2 rows

```
layout_matrix = rbind(c(1,1,2),
                      c(1,1,NA),
                      c(4,3,3),
                      c(4,NA,NA)));
```

Let's put the **4x3** matrix in ***grid.arrange()***:

```
#### Part 9: Extending plots across rows and columns
grid.arrange(plot1, plot2, plot3, plot4,
             layout_matrix = rbind(c(1,1,2),
                                   c(1,1,NA),
                                   c(4,3,3),
                                   c(4,NA,NA)));
```

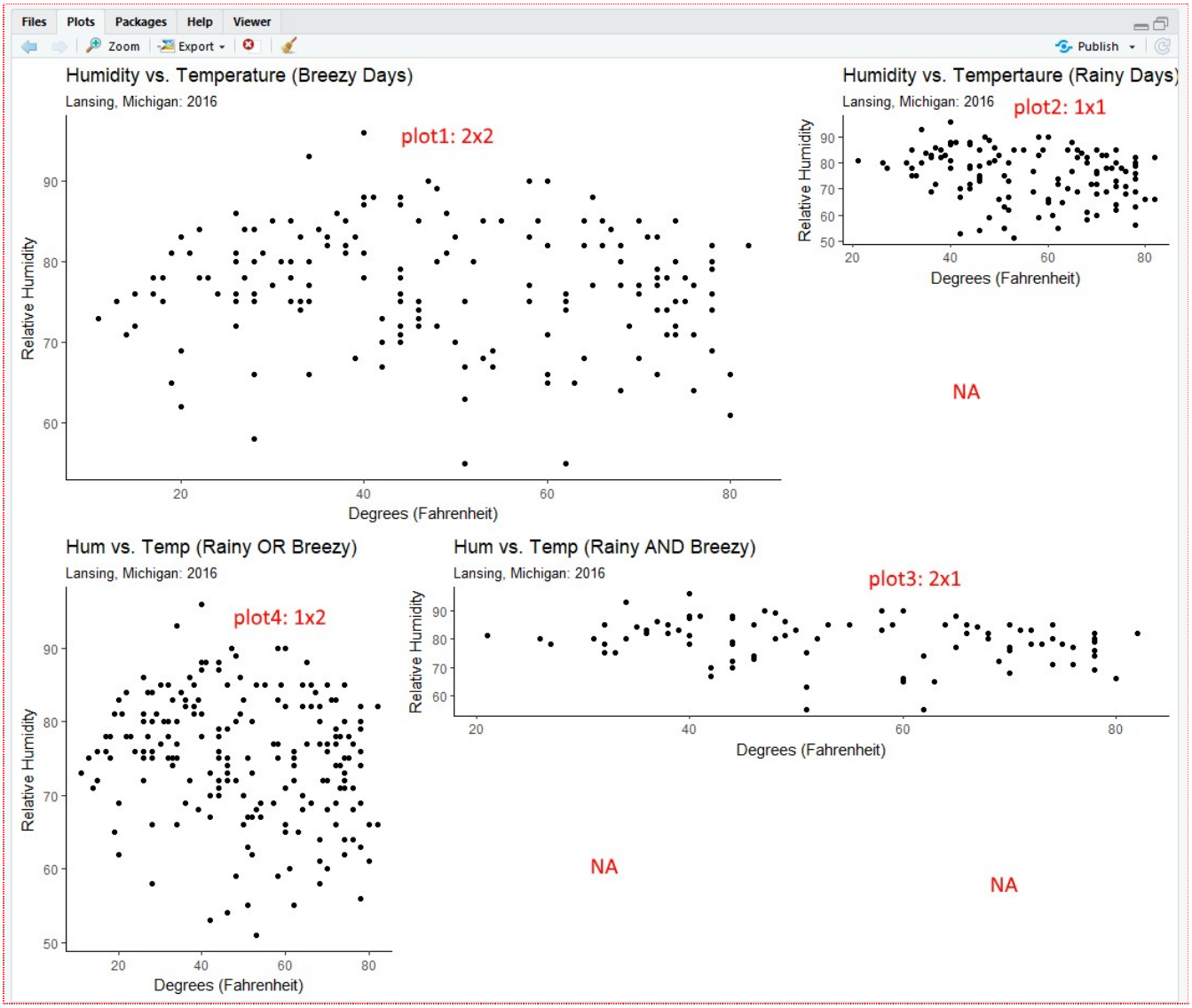

Fig 10: Extending plots across rows and down columns

8 - Errors in layout_matrix

***grid.arrange()** is very sensitive* and often produces unintuitive errors. In this section, we will look at some of the issues and the errors associated with the issues.

8.1 - Issue 1: Plots listed must be used in matrix

All plots listed in ***grid.arrange()*** must be used in the matrix

Issue: ***plot2*** is listed but not used in the matrix:

```
#### Issue 1: Plots listed must be used in matrix
####          Plot 2 is not used in the layout_matrix
grid.arrange(plot1, plot2, plot3, plot4, plot5
             layout_matrix = rbind(c(1,1,5),
                                   c(1,1,NA),
```

```
6                                   c(4,3,3),
7                                   c(4,NA,NA)));
```

Executing this grid without using Plot 2 produces the error:
Error in t:b : NA/NaN argument

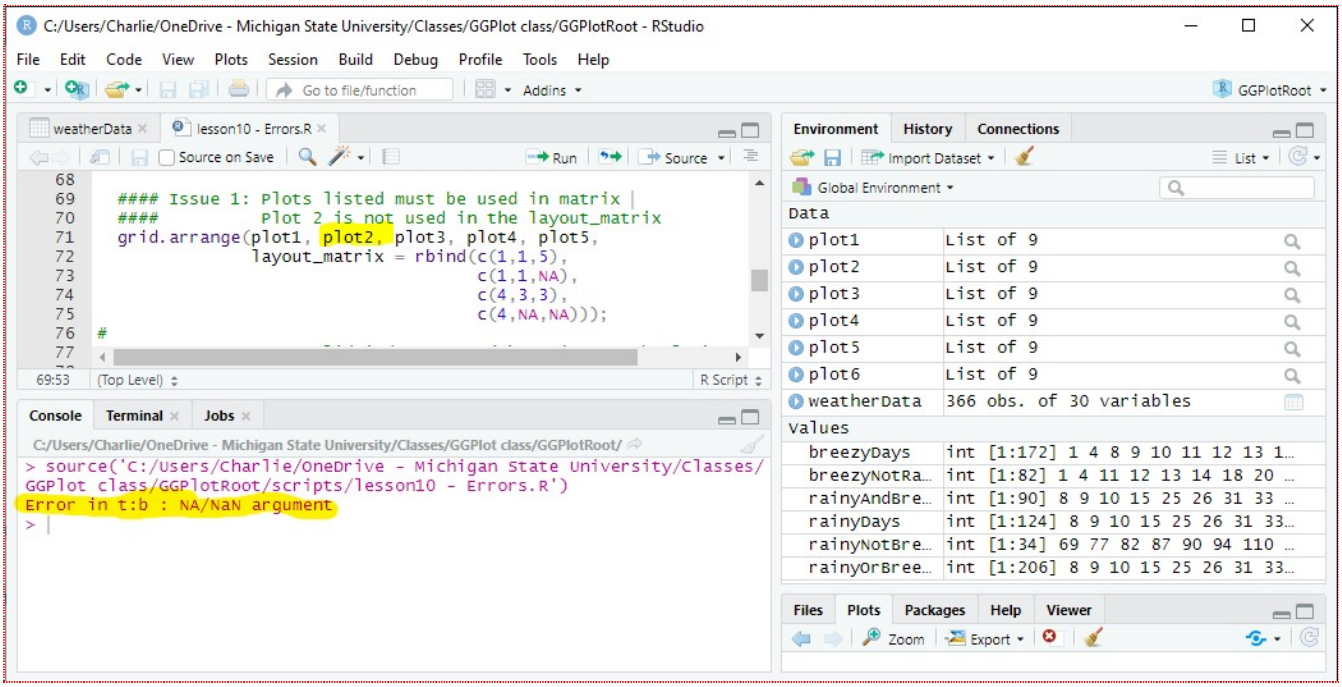

*Fig 11: Plot put into **grid.arrange()** but not used in **layout_matrix()***

8.2 - Invalid index numbers in matrix

Issue: index **5** is used in the matrix but there is no 5th plot

```
1 #### Issue 2: Invalid index error (there is no 5th plot)
2 grid.arrange(plot1, plot2, plot3, plot4
3              layout_matrix = rbind(c(1,1,5),
4                                    c(1,1,NA),
5                                    c(4,3,3),
6                                    c(4,NA,NA)));
```

When executing this grid, the number will be ignored by ***layout_matrix()*** and nothing will be plotted in that area.

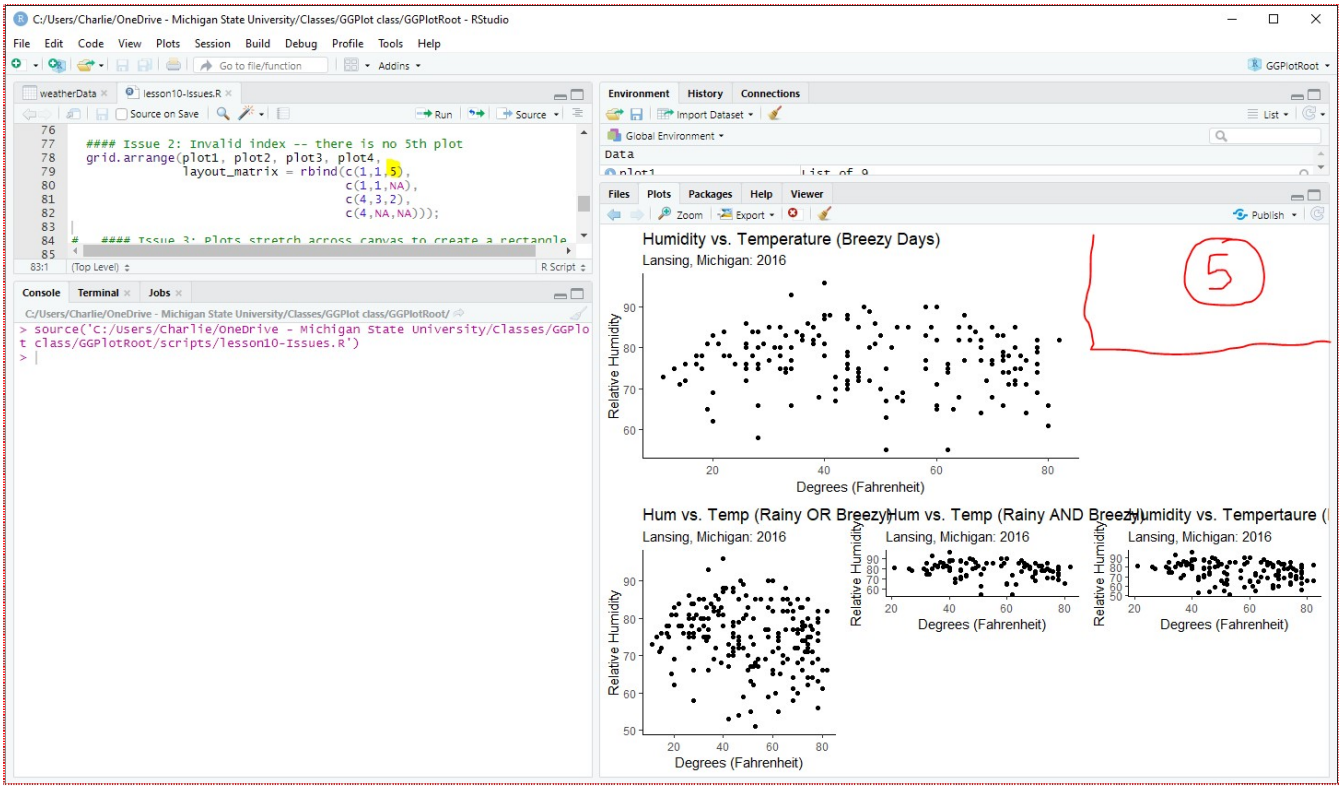

Fig 12: Nothing is plotted in the area of the grid with an invalid plot index (5).

8.3 - Extending plots to fill a rectangle

grid.arrange() makes two important assumptions:

1. All plots take up a rectangular space on the grid
2. All plots are represented only once on the canvas

If you arrange the index numbers in the matrix in a non-rectangular fashion, ***grid.arrange()*** will "fill" in the rest of the rectangle.

In the following code, ***grid.arrange()*** will extend ***plot1*** across the 2 rows and 2 columns

```
#### Issue 3: The grid will extend discontinuous plots to fill a rectangle
####          plot1 will be extended to a rectangle that is 2x2
grid.arrange(plot1, plot2,
layout_matrix = rbind(c(1,NA,2),
                      c(NA,1,NA)));
```

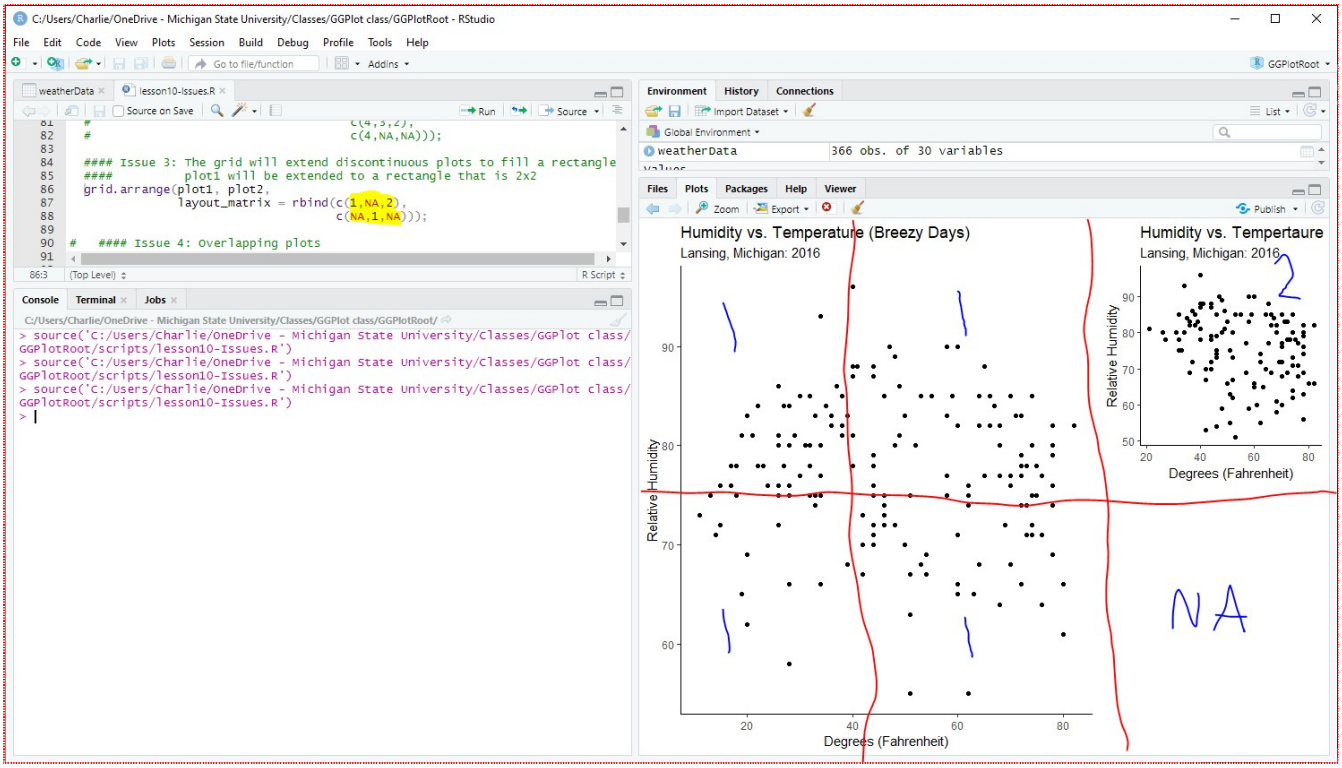

Fig 13: Plots are extended to fill the rectangular area they represent in the grid.

8.4 - Overlapping plots

In this example, ***grid.arrange()*** assumes that ***plot1*** is **2** rows x **3** columns (it takes up the whole plot area). This means that plot1 and plot2 both use the spot on the 1st row, 3rd column.

In this case, ***plot2*** overlaps ***plot1*** in the one grid space because ***plot2*** comes after ***plot1*** in ***grid.arrange()***.

```
#### Issue 4: Overlapping plots -- priority goes to the later plot
####          In this case, plot2 overlaps plot1
grid.arrange(plot1, plot2,
layout_matrix = rbind(c(1,NA,2),
                      c(NA,NA,1)));
```

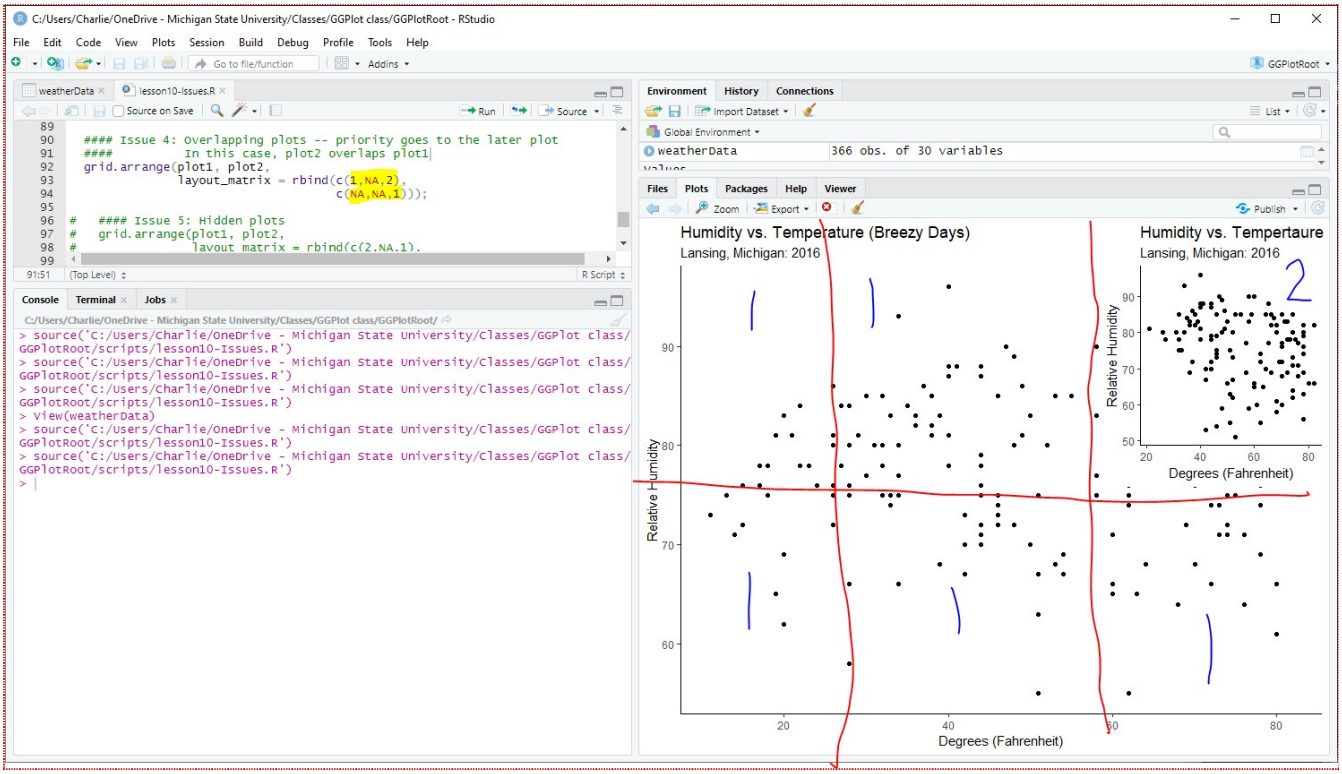

*Fig 14: Plots overlapping in **layout_matrix()**, the later plot get priority.*

8.5 - Hidden plots due to overlapping

grid.arrange() puts plots with higher indexes on top. So, if we reverse the previous canvas and stretch ***plot2*** across the whole canvas then ***plot2*** will still overlap ***plot1.*** This effectively hides ***plot1***.

```
#### Issue 5: Hidden plots due to overlapping
####          In this case, plot2 completely covers up plot1
grid.arrange(plot1, plot2,
layout_matrix = rbind(c(2,NA,1),
                      c(NA,NA,2)));
```

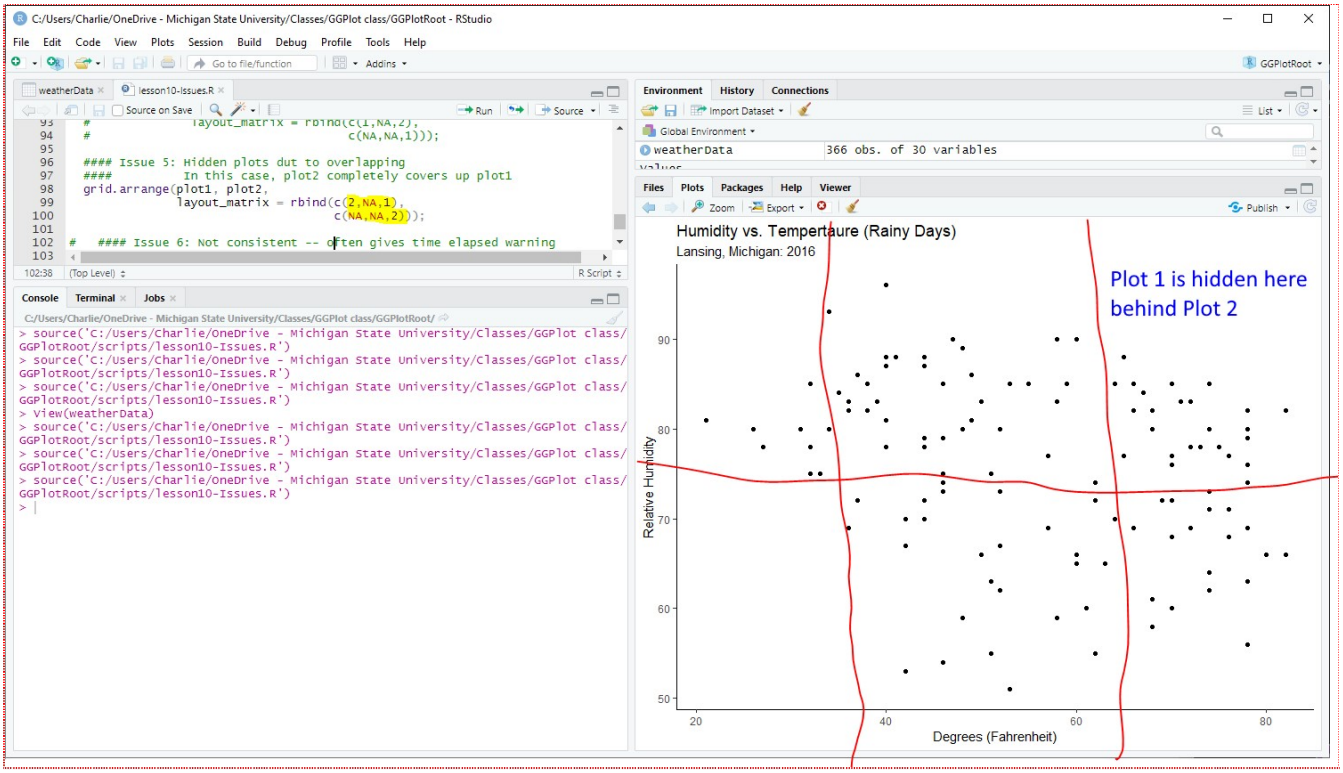

Fig 15: A plot is hidden because it is overlapped by another plot with higher priority.

8.6 - Time elapsed warning

When the canvas area is complicated, the plot will sometimes not be fully drawn. This is an inconsistent error that happens in RStudio on slower computers.

Theoretically, this should show ***plot4*** across the whole canvas, covering up the other plots.

```
#### Part 10: Error in plot placement
grid.arrange(plot1, plot2, plot3, plot4,
             layout_matrix = rbind(c(1,1,4),
                                   c(1,2,NA),
                                   c(4,3,3),
                                   c(4,NA,NA)));
```

This code will sometimes give an elapsed time warning like:
In unique.default(lengths(x)) : reached elapsed time limit

If the full canvas does not get drawn when you execute the script, you can press the ***Zoom*** button to open the canvas in a new window.

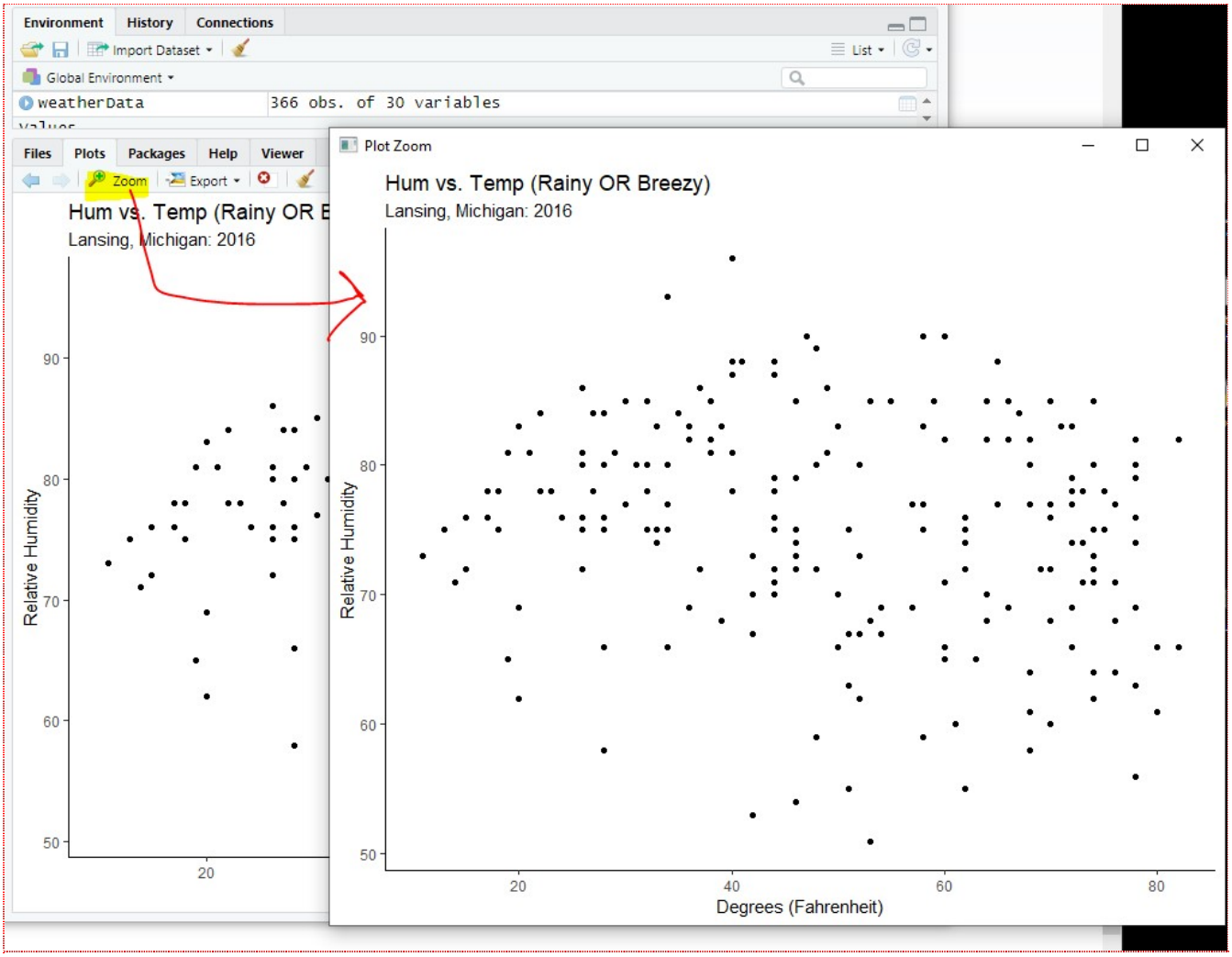

Fig 16: Using the Zoom button to open a plot in a new window.

9 - Application

Find how different weather conditions in the ***weatherType*** column correlate with ***tempDept***. *Note: **tempDept** is how far the temperature for the day was from the historic average temperature for that day.*

- Plot out ***tempDept*** histograms for three different conditions
- On each plot, annotate the average ***tempDept*** for the condition
- Make plots that combine ***weatherType*** conditions using ***union()*** and ***intersect()***
- Using ***grid.arrange()***, put at least 5 plots on one canvas
- Using ***grid.arrange()***, have at least 2 of the plots take up more than 1 cell.

10 - Extension: Coding for neither condition

```
notRainyAndNotBreezy = setdiff(1:366, rainyOrBreezy);
```