# 01-02: Components

## 1 - Purpose

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

## 2 - 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 Root 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 Root Folder and click the ***.RProj*** file

## 3 - Reference Script

The first script we will create is a *reference script*, which is a script that is called by other scripts but is not executed on its own. The purpose of a reference script is to contain code used by multiple script files so that the code does not have to be repeated in each of those individual script file. *Note: packages from libraries are also examples of reference scripts.*

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

For now the reference script will contain three lines:

```
1 rm(list=ls());                          # clear Console Window
2 options(show.error.locations = TRUE);   # show line numbers on error
3 library(package=ggplot2);               # include all GGPlot2 functions
```

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

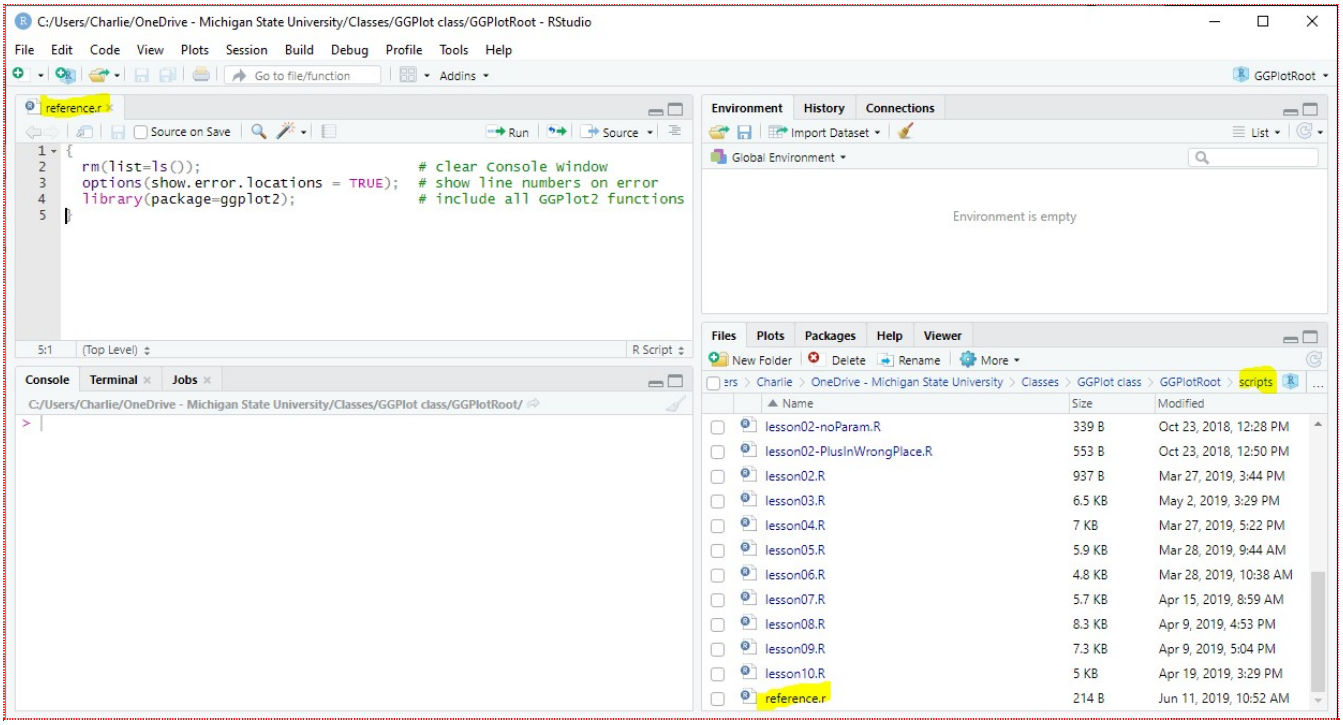


*Fig 1: The reference.r file inside the Script folder of the Project*

## 3.1 - Error locations

The second line of reference.r:

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

is a good line to always include in your R code because it helps you debug by giving you the line number that the compiler thinks an error is on. However, this error detection does not work within GGPlot code. So, it is of limited use in this class.

# 4 - Looking at a data file

A plotting program is not very useful unless it has data. So, we first need to get some data and then set up a script to read in the data.

## 4.1 - Get data for the plot

This website has hundreds of data set that are freely available. We are going to use the data set called Growth of CRAN. The data set gives the number of R Packages that were available each year from 2000 through 2014.

Save the CSV file, called ***CRANPackages.csv*** to the ***data*** folder inside your Project. *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 a CSV file*

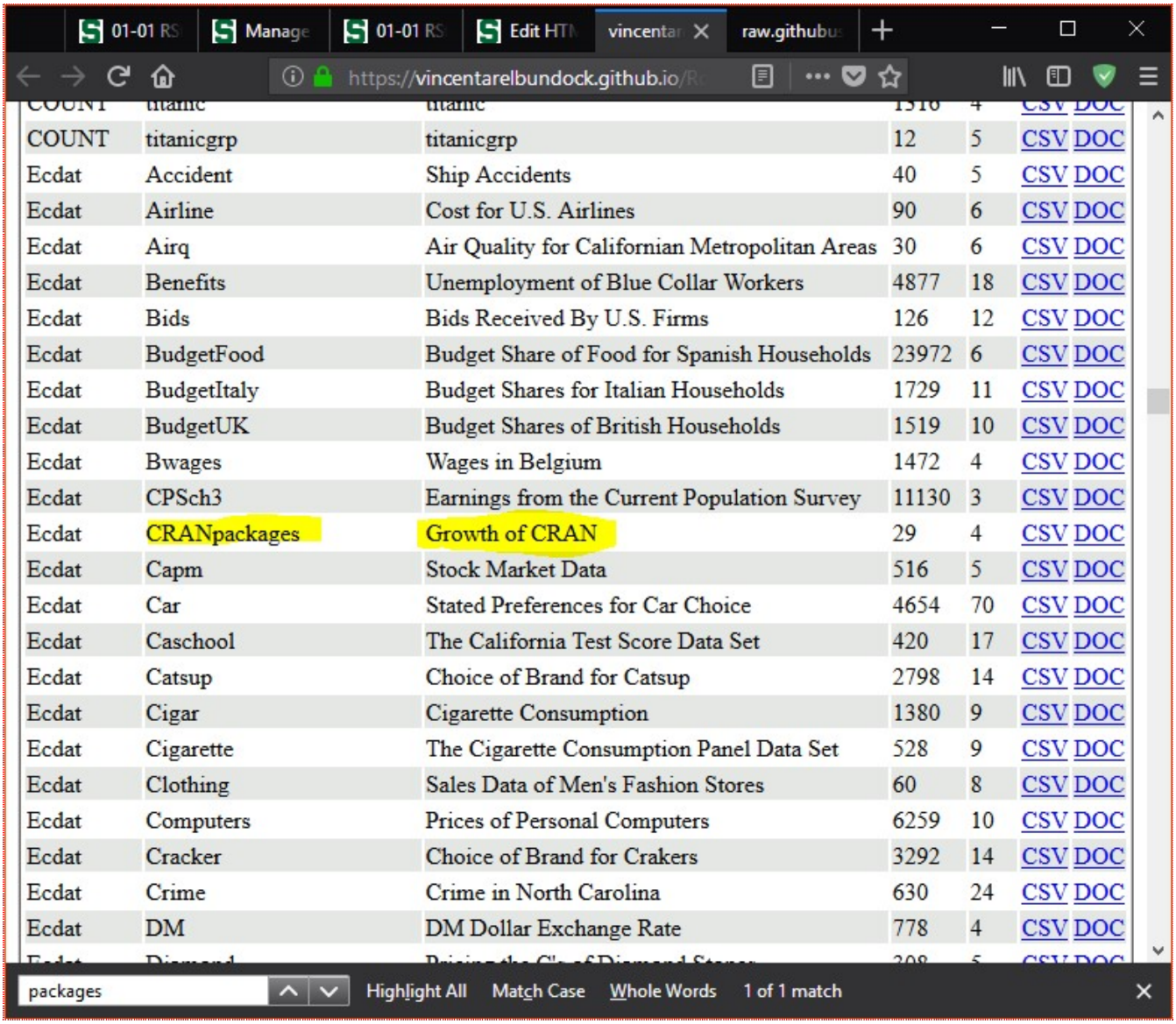


*Fig 2: The link to CRANPackage.csv*

## 4.2 - Start a new script and include the files we need

We are going to start a new script called ***lesson02.R*** and use it to read in and plot the data from ***CRANPackage.csv***.

To read in the data from ***CRANPackages.csv***:

1. Open a new script file (***File*** -> ***New File*** -> ***R Script***).
2. Add these two lines of code to the script:

```
# execute the lines of code from reference.r
source(file="scripts/reference.r");

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

Save the script file as ***lesson02.r*** inside the ***scripts*** folder in your RStudio Project.

## 4.3 - Look at the data

We execute ***lesson02.r*** by clicking ***Source*** (*fig 3*). *Extension: Run vs. Source*

The script created the variable ***packageData***, which you can see in the Environment Window (*fig 3*). ***packageData*** is a data frame with **5** columns and **29** rows populated from ***CRANPackages.csv***.

A quick look at the five columns in ***packageData*** (*fig 3*) shows that column **4** contains the number of ***Packages*** in CRAN and column **3** contains the corresponding ***Date***.

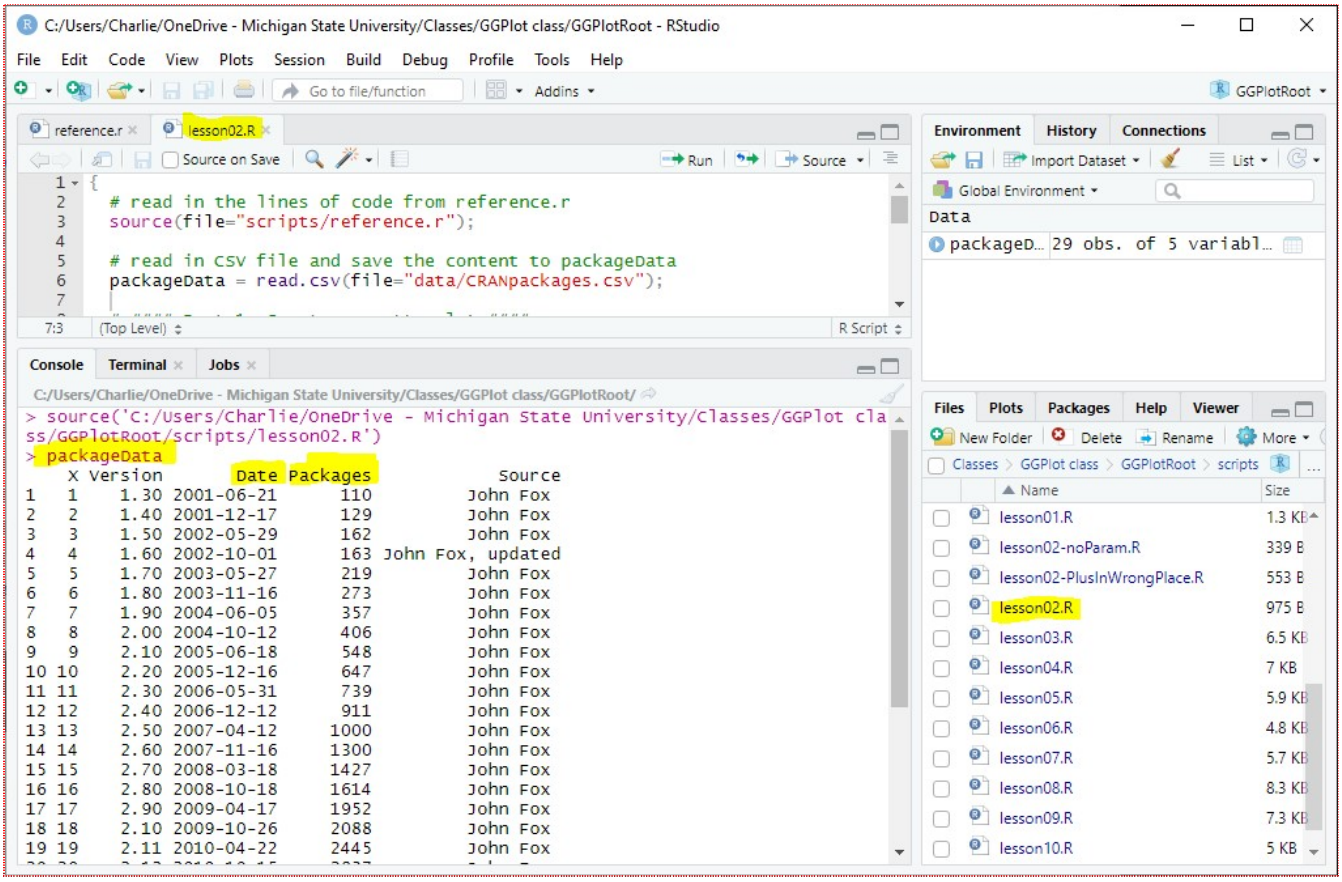


*Fig 3: The data from the **packageData** data frame*

# 5 - Create plot data using GGPlot

Next, we will create a scatterplot of ***Packages*** *(column 4)* vs ***Date*** (column 3) from ***packageData***.

The code to create a scatterplot using GGPlot is:

```
# read in the lines of code from reference.r
source(file="scripts/reference.r");

# read in CSV file and save the content to packageData
packageData = read.csv(file="data/CRANpackages.csv");

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

```
            geom_point( mapping=aes(x=Date, y=Packages) );
plot(plotData);
```

Let's ***Source*** the script -- the code will be explained in a bit and the overlapping x-axis date labels will be fixed.

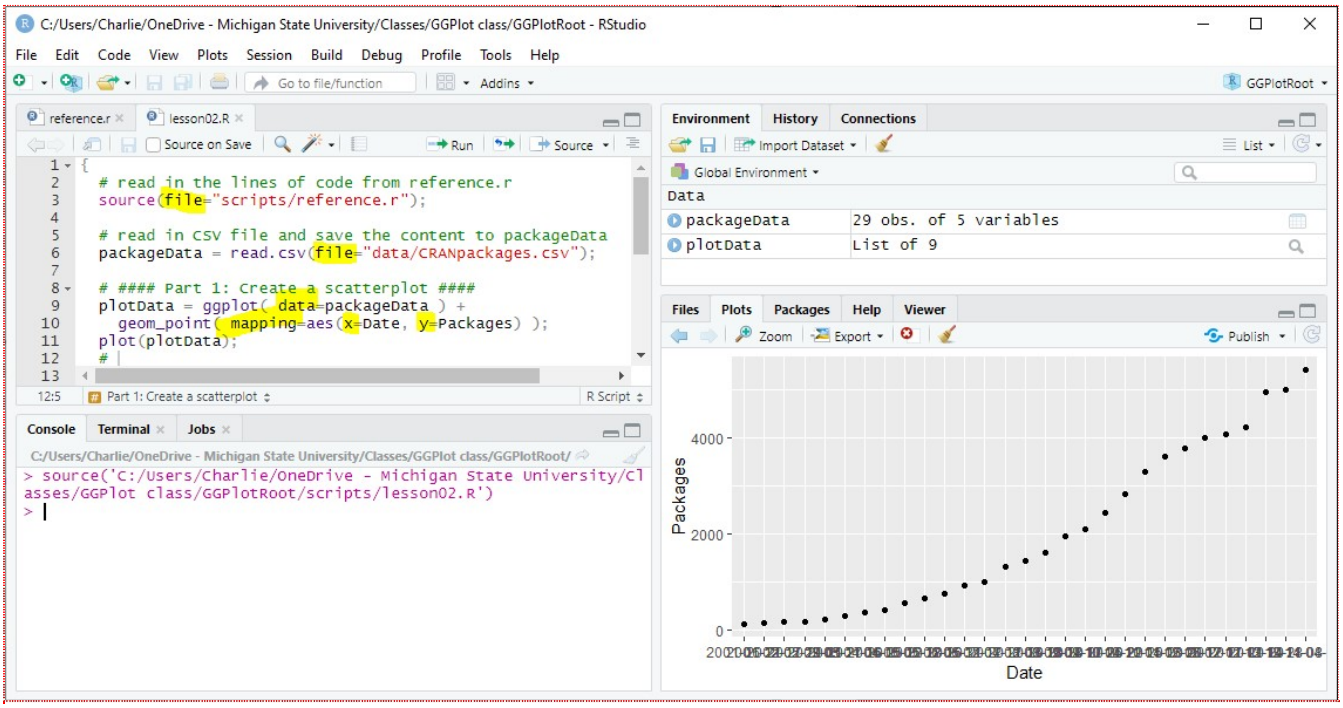


*Fig 4: Our first plot using GGPlot with parameter names highlighted*

Extension: The yellow warning sign

## 5.1 - Taking out parameter names

In *fig 4* I highlighted the parameter names in the code (which, in GGPlot, are also the subcomponents). In this case, the script will still work even if we take out all the parameter names:

```
# same five lines of code without parameter names in the functions
source("scripts/reference.r");
packageData = read.csv("data/CRANpackages.csv");

plotData = ggplot( packageData ) +
           geom_point( aes(Date, Packages) );
plot(plotData);
```

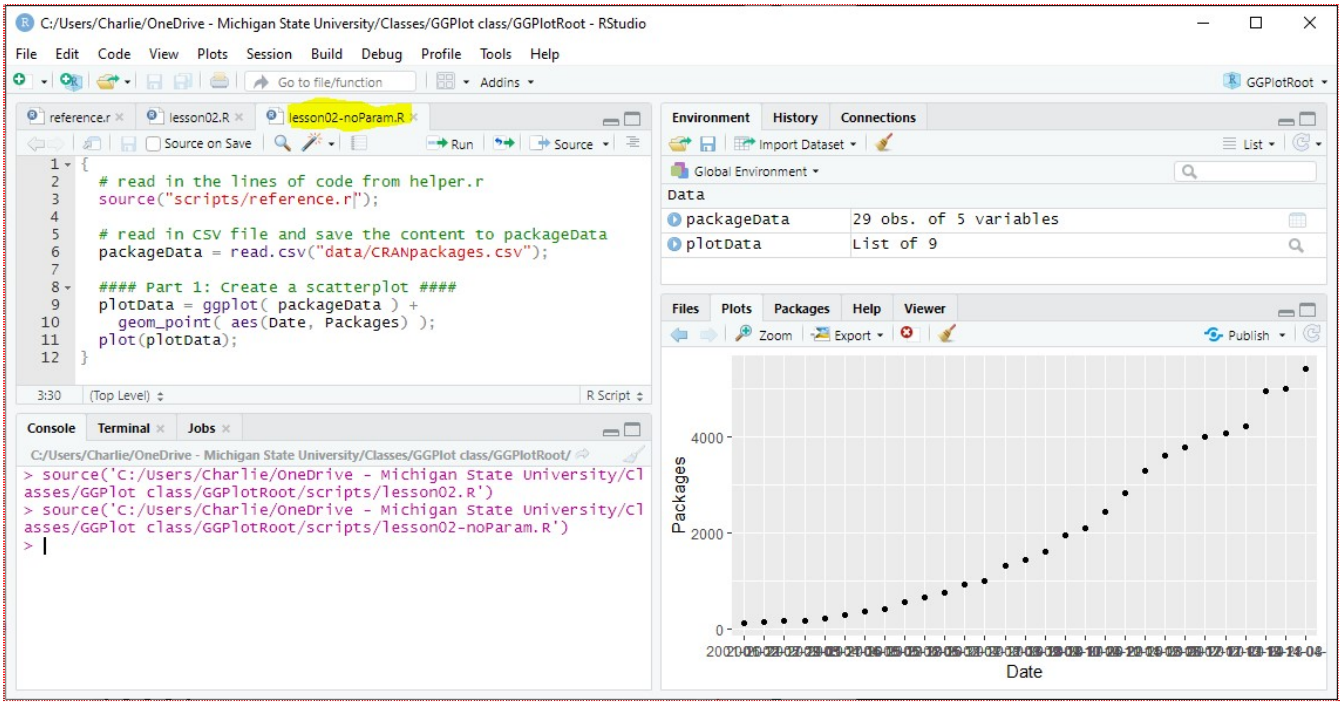


*Fig 5: Same script as above with the parameter names taken out*

## 5.2 - Benefits of using parameter names

This script works because we only used the default parameters for each function and we used the parameters in the same order as they appear in the function.

In this class, *we will (almost always) use parameter names* because using parameter 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 parameters however you want
- avoids bad assumptions about the ordering of parameters and their default values

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

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

instead of

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

There are multiple functions in R and GGPlot where the parameter name ***x*** is used generically as the name for the first parameter in a function. This is not intutive because ***x*** is also used to refer to data that goes on the x-axis.

We will use the parameter name ***x*** only when ***x*** refers to an axis (line 10: ***x=Date***) but not when ***x*** is a generic first-parameter name (line 11: ***x=plotData***).

# 6 - Components of a GGPlot

Let's take a more detailed look at the three lines of code that created the scatterplot in *figure 5*...

First we initialize the canvas for the plotting area using ***ggplot()*** and the data frame we are going to use, ***packageData***:

```
plotData = ggplot( data=packageData ) +
           geom_point( mapping=aes(x=Date, y=Packages) );
plot(plotData);
```

Next, we add the component, ***geom_point()***, which creates a scatterplot using the ***Date*** and ***Packages*** columns from ***packageData***:

```
plotData = ggplot( data=packageData ) +
           geom_point( mapping=aes(x=Date, y=Packages) );
plot(plotData);
```

The ggplot information is saved to a **List** variable named ***plotData:***

```
plotData = ggplot( data=packageData ) +
           geom_point( mapping=aes(x=Date, y=Packages) );
plot(plotData);
```

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

```
plotData = ggplot(data=packageData ) +
           geom_point( mapping=aes(x=Date, y=Packages) );
plot(plotData);
```

## 6.1 - GGPlot components

In GGPlot, you initialize a canvas and then add components (often called layers) to the canvas. The **+** symbol is used to add components. In the above example (*fig 4*), there is the initializing canvas function and one component:

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

```
plotData = ggplot( data=packageData ) +
           geom_point( mapping=aes(x=Date, y=Packages) );
```

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

```
plotData = ggplot( data=packageData ) +
           geom_point( mapping=aes(x=Date, y=Packages) );
```

## 6.2 - GGPlot mapping and aesthetics (aes)

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

```
plotData = ggplot( data=packageData ) +
           geom_point( mapping=aes(x=Date, y=Packages) );
```

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 probably easier to just think of ***aes*** as a mapping element.

## 7 - Adding more components to the canvas

Let's say we want to make the three following modifications to the plot:

1. add a title
2. change the numeric tick marks on the y-axis
3. change the direction of the x-axis labels (so we can read the labels)

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

1. ***ggtitle()*** # title component
2. ***scale_Y_continuous()*** # y-scaling component (there is a corresponding x-scaling component)
3. ***theme()*** # theme component

We add components using ( **+** ).

```
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);
```

*Trap: putting the ( + ) on the next line*

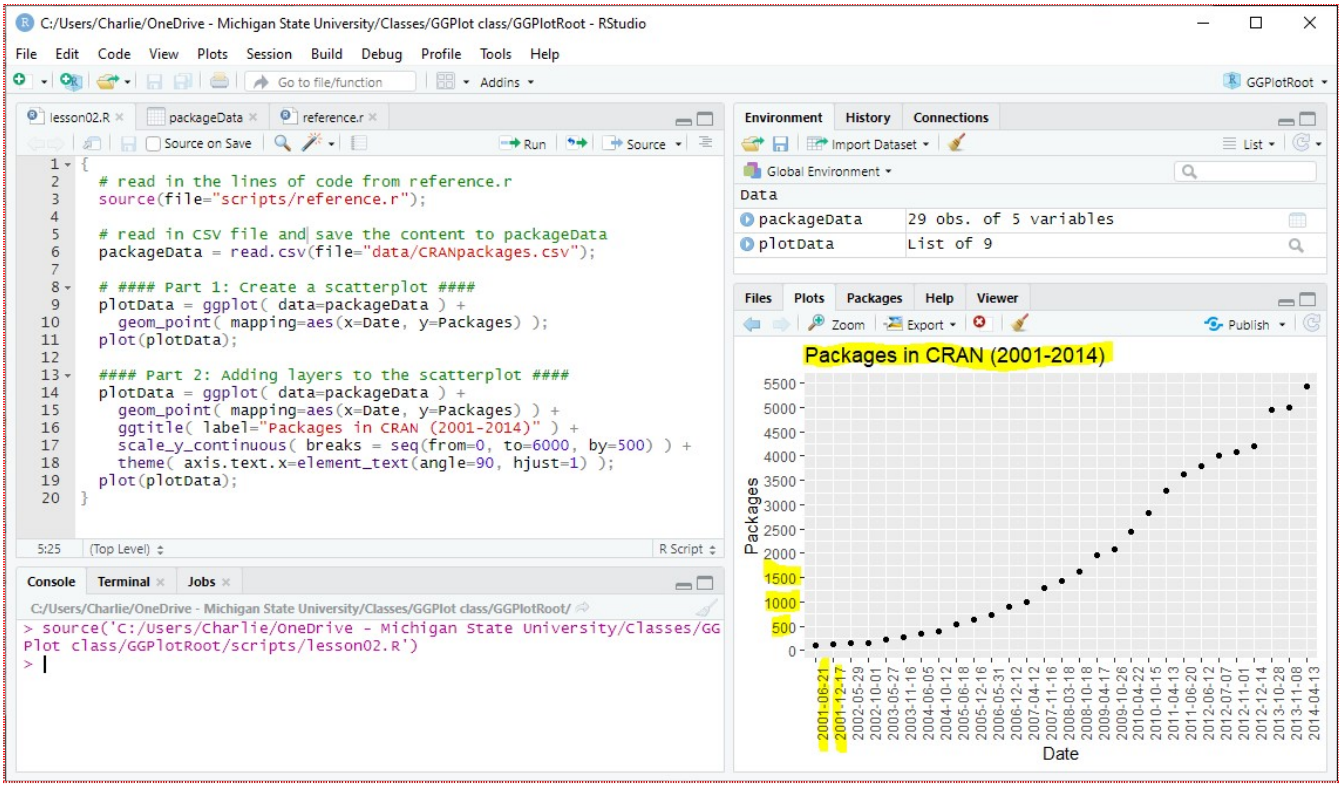


*Fig 6: Scatterplot with a few added components*

## 7.1 - The Components in detail

```
ggtitle( label="Packages in CRAN (2001-2014)" )
```

When we search in the ***Help*** tab for ***ggtitle()*** (*fig 7*) we see that it has two subcomponents (or parameters) to change:

- ***label***: the title
- ***subtitle*** (not used in this example): a secondary title that has no value, or **NULL**, as a default

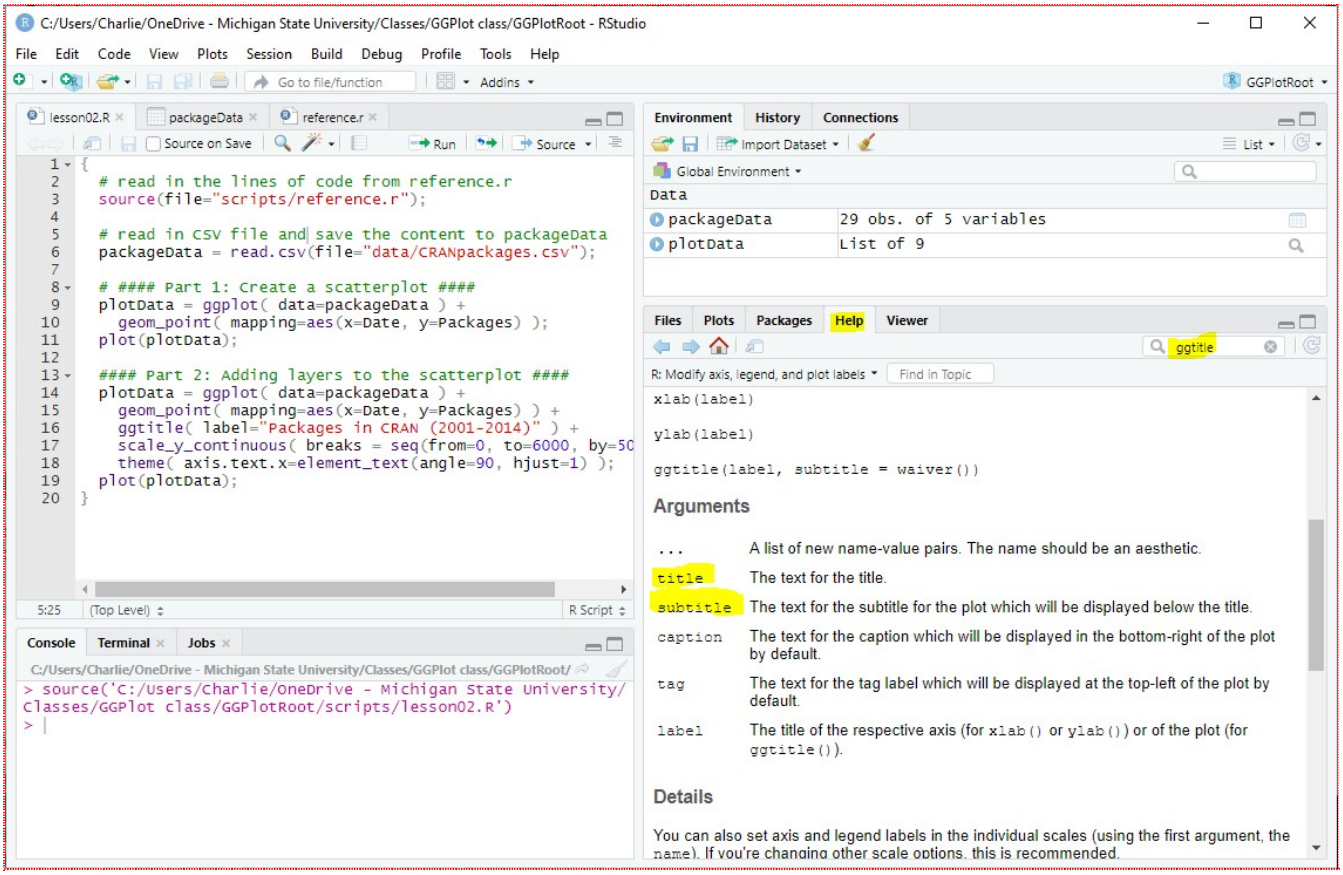


*Fig 7: Using the Help Tab in RStudio to find info about GGPlot components*

```
scale_y_continuous( breaks = seq(from=0, to=6000, by=500) )
```

***scale_y_continuous()*** is the component used when you want to modify a y-axis that has continuous values. There are many subcomponents (*fig 8*) that can be changed in ***scale_y_continuous()***. We modified one subcomponent, ***breaks***, by setting it to a sequence from **0** to **6000** and the tick marks were placed at intervals of **500**. *Note: **waiver()**, which is used as a default value for many of the subcomponents, is a somewhat unintuituve way of saying to use the values calculated by the plotting function (i.e. default values).*

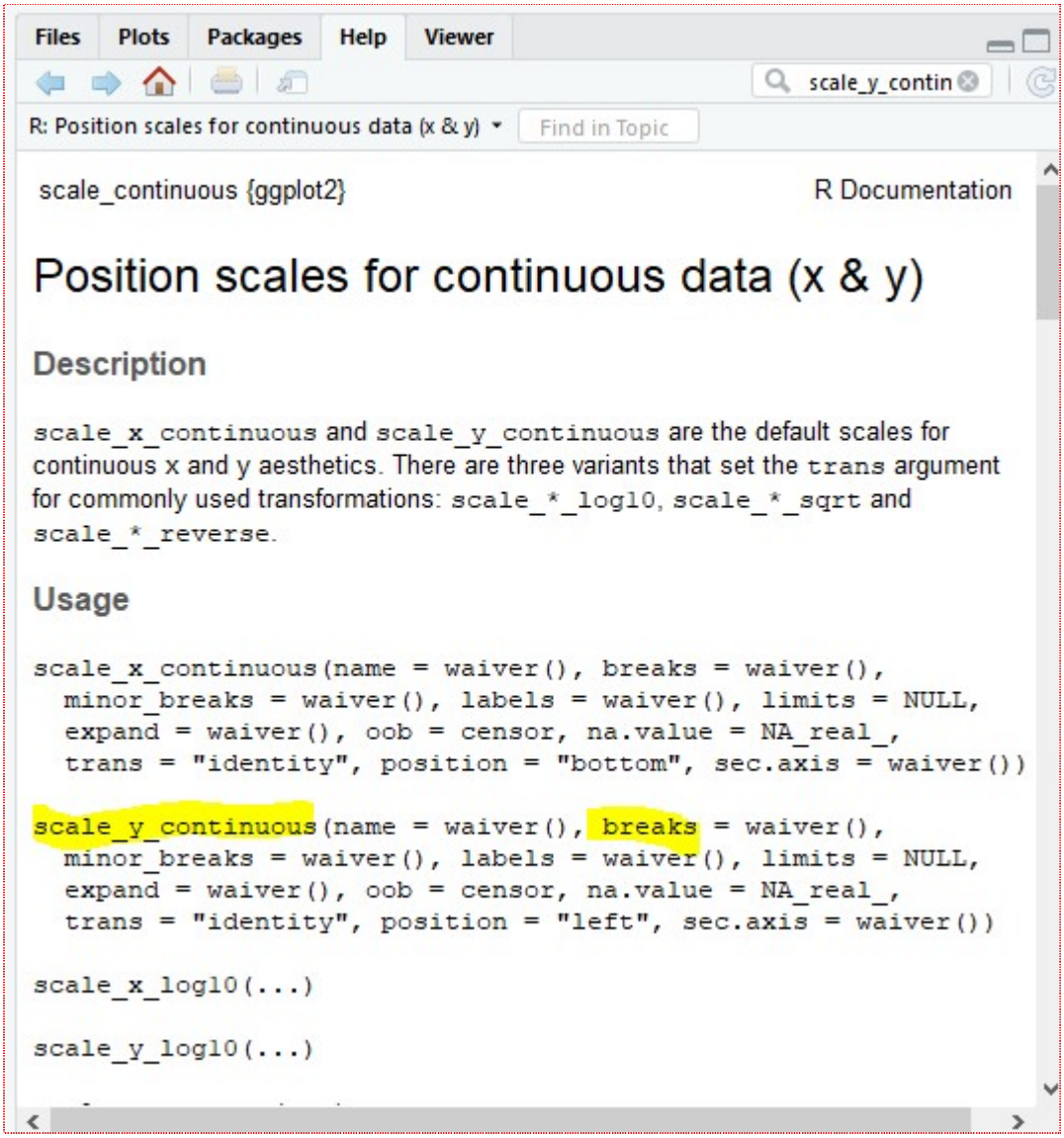


*Fig 8: Scale_y_continuous help page*

```
theme( axis.text.x=element_text(angle=90, hjust=1) )
```

In this example we changed one subcomponent in ***theme()*** called ***axis.text.x*** and set it to a ***element_text()*** that modifies the text by rotating it to an ***angle*** of **90** degrees and right-justifying the text (***hjust*=1**). *Note: **hjust=0** left-justifies the text, **hjust=0.5** centers the text.*

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

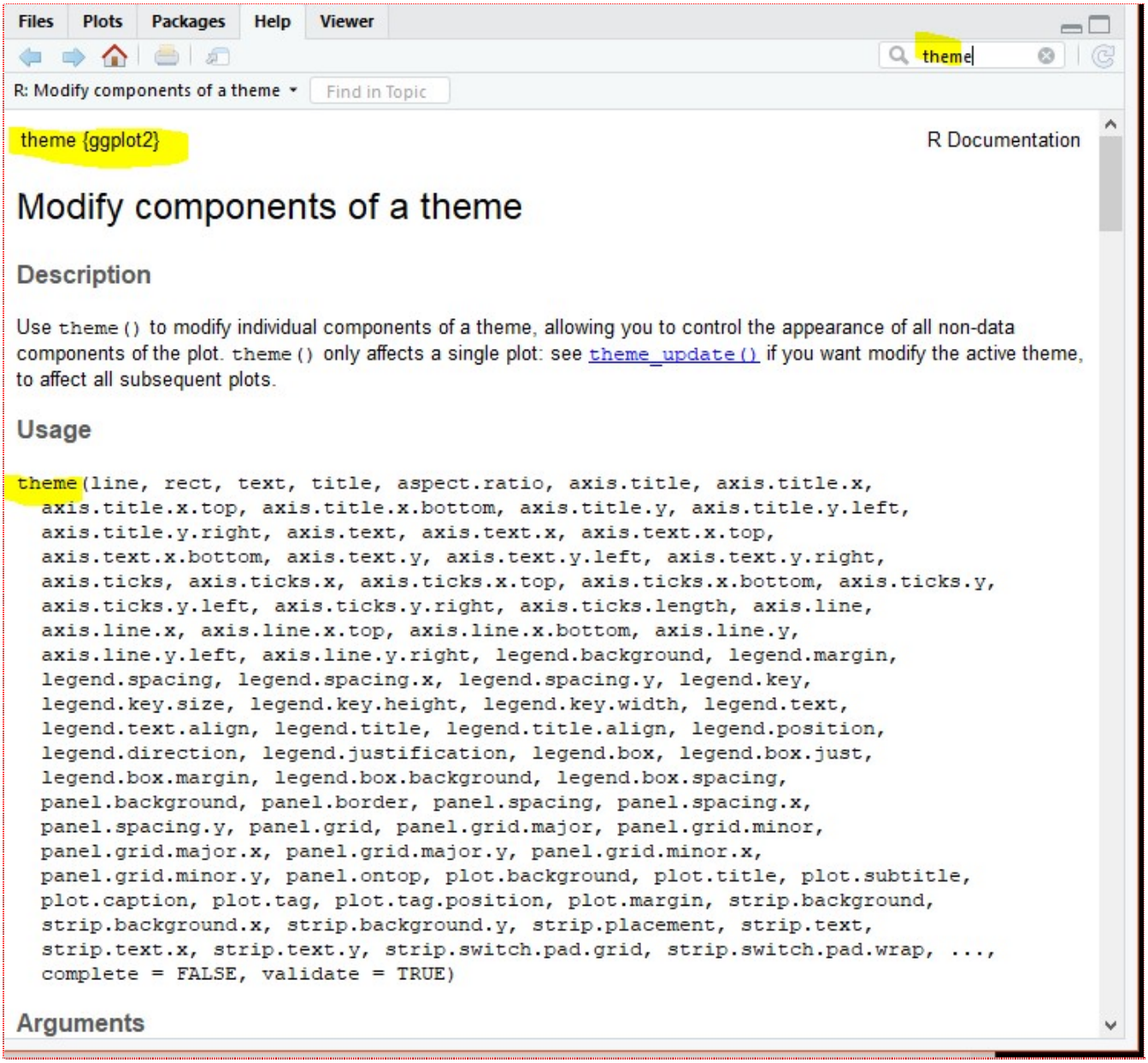


*Fig 9: **theme()** component help page (yes, there is a lot there!)*

## 7.2 - 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. The ***Help*** tab provides information directly from https://ggplot2.tidyverse.org/reference/, which is the official webpage for GGPlot.

# 8 - Application

***Create a scatterplot:***

1. Download the Accidental Deaths in the US 1973-1978 data file and save to your project's ***data*** folder
2. Create a new R script file in your project's ***script*** folder called ***app02.R***
3. Include the reference file, **reference.r**, in ***app02.R***
4. Do a scatterplot of ***accdeaths*** vs ***time*** using the data from the downloaded data file
5. Add a title to the plot

6. Change the angle of the x-axis labels to 45 degrees
7. Change the ***x-axis*** ticks to go by half-year instead of whole year (i.e., **1973**, **1973.5**, **1974, 1974.5**...)
   - Note: you can treat the years as numbers
8. Have the **y-*axis*** only display three values: **7000**, **9000**, and **11000**
9. Zip your Root Folder with ***app02.r*** and the downloaded data file
10. *Email the zipped Root Folder to the instructor*

**Windows: Zip your Root Folder** (*fig 10*)**:**

1. In your File Manager (not in RStudio), right-click on the Root Folder
2. Choose ***Send to***
3. Choose ***Compressed (zipped) folder***
4. A zipped file named ***<Root Folder>.zip*** with all your Root Folder contents is created in the same folder

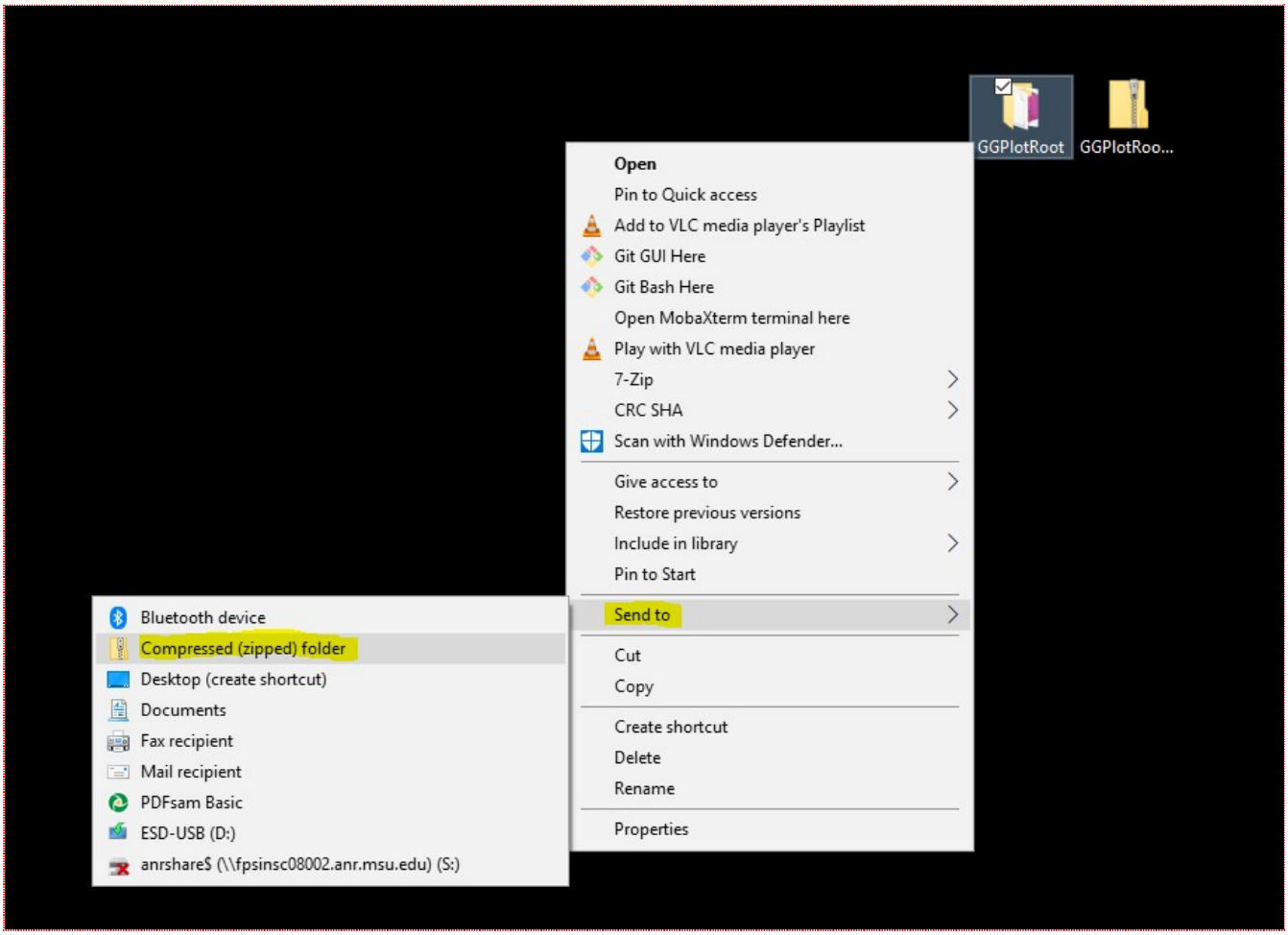


*Fig 10: Zipping your Root Folder*

**Mac: Zip your Root Folder:**

1. In your File Manager (not in RStudio), right-click on the Root Folder
2. Choose ***Compress "<Root Folder>"***
3. A zipped file named ***<Root Folder>.zip*** with all your Root Folder contents is created in the same folder

## 9 - 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 window). This can be dome by clicking ***File -> Open Project in New Session.***

RStudio considers only files saved within the Root Folder (and subfolders) of the Project to be a part of the Project. You can create a script file that is independent of the RStudio Project you are working on, you just need to save the script file outside of the Project's Root Folder.

## 10 - 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.
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.

## 11 - Extension: Run vs. Source

Technically speaking, the difference between ***Run*** and ***Source*** is:

- ***Source*** will execute all the code in a script file.
- ***Run*** will execute only the line that the cursor is on or all lines that are highlighted.

The real difference lies in a historical discussion of scripting vs. programming, which is a discussion beyond this class. Suffice to say, R was originally intended to be a quick-and-dirty scripting language where users could immediately pull in data and produce a plot or perform statistical analysis. Using this method, an R-user would produce and execute one line of code at a time using the equivalent of the ***Run*** button.

As R has grown, the focus has shifted into developing well-structured code that can be easily shared, tweaked, and reused -- similar to a modern programming language like C. Using this method, an R-user would produce a full script and execute it all using the equivalent of the ***Source*** button. This is the method we will be using in this class.

## 12 - 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:

```
1 source(file="scripts/reference.R");
2 packageData = read.csv(file="data/CRANpackages.csv");
3
4 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.

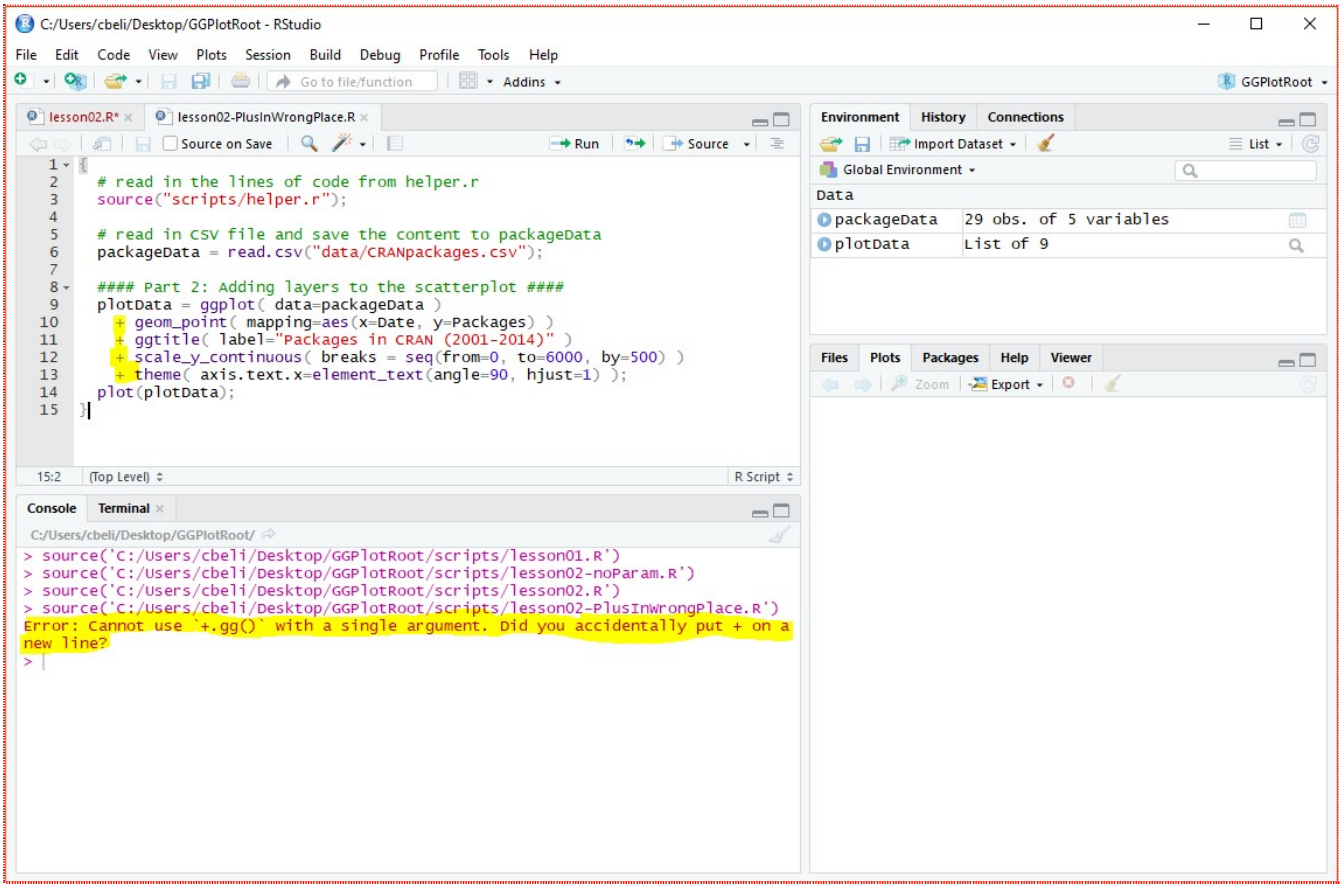


*Fig 11: Error when putting the ( + ) on the next line*

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.

## 13 - Extension: The yellow warning sign

When you are working in GGPlot and have debugging features turned on in RStudio, you will almost always see multiple yellow warning signs on the side of your code (*fig 12*). 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.

Unfortunately, this is just a limitation of the RStudio debugger.

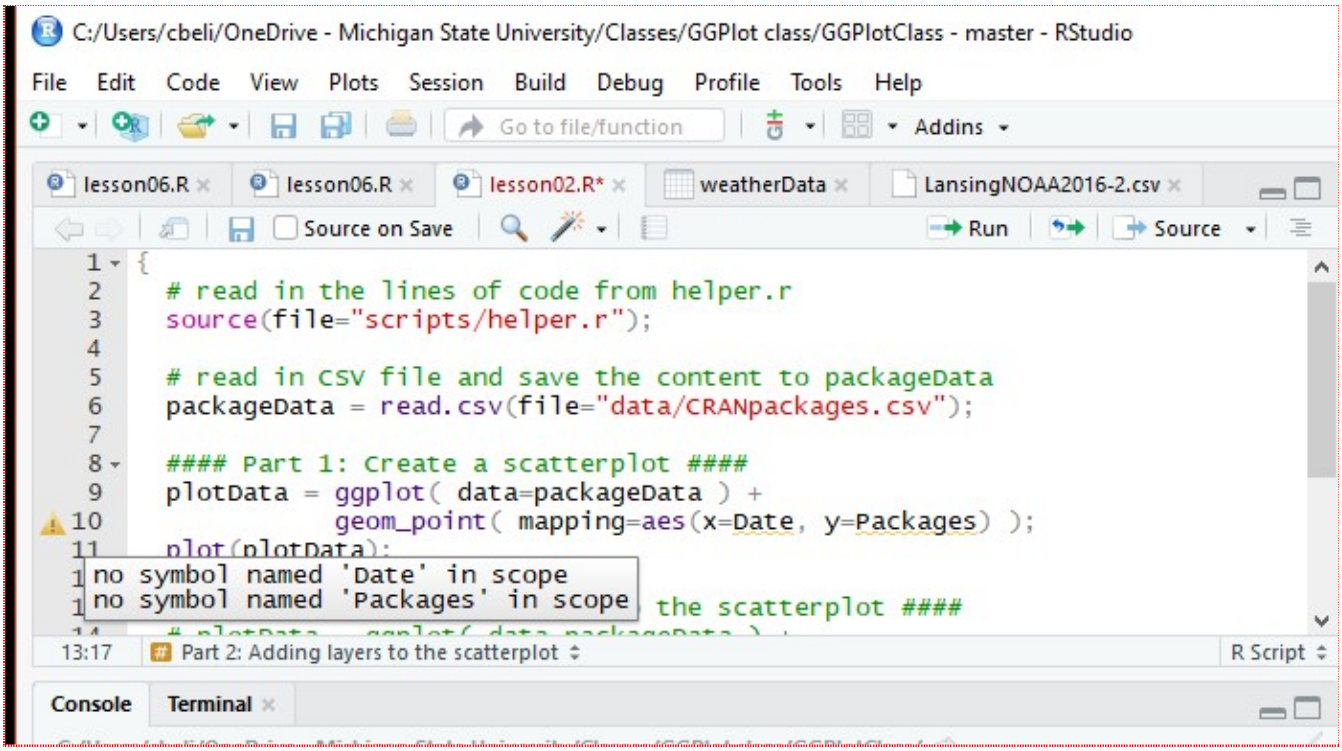


*Fig 12: Warning about variables within the GGPlot functions*