# Module Visualization in R-GGPLOT and Graph Formatting

# Introduction and Learning Objective

Introduction

The **ggplot2** package, written by Hadley Wickham (2009a), provides a system for creating graphs based on the grammar of graphics described by Wilkinson (2005) and expanded by Wickham (2009b). The intention of the **ggplot2** package is to provide a comprehensive, grammar-based system for generating graphs in a unified and coherent manner, allowing users to create ne and innovative data visualizations. The power of this approach has led **ggplot2** to become an important tool for visualizing data using R.

The **ggplot2** package implements a system for creating graphics in R based on a comprehensive and coherent grammar. This provides a consistency to graph creation that’s often lacking in R and allows you to create graph types that are innovative and novel. In this section, we’ll provide an overview of **ggplot2** grammar. Before continuing, be sure the **ggplot2** package is installed.

In **ggplot2**, plots are created by chaining together functions using the plus (+) sign. Each function modifies the plot created up to that point. The **ggplot2** package is powerful and can be used to create a wide array of informative graphs. The above section has been a brief introduction to the package. A list of all **ggplot2** functions, along with examples, can be found at <http://docs.ggplot2.org>. To learn about theory underlying **ggplot2**, see the book by Wickham (2009a). Chang (2013) has written a very practical book, chock full of useful examples.

In this module, we will introduce the R package ggplot, and introduce how to format a graph with different settings.

In this module, we will introduce ggplot2 package, and how to format the graphs in different ways. For example, color the data points, format the lines, add legend and text to a graph, and format the x- and y-axis.

Learning Objectives

After this module, students will be able to

* Generate graphs using ggplot()
* Identify global graph settings and local settings
* Format graphs with the settings for points and lines
* Add title, legend and text to graphs
* Format axes

#### Learning Objective One: Construct graphs from GGPLOT

***Introduction***

Chapter 3 (“Data visualization”) of the open source text, *R for Data Science* by Garrett Grolemund and Hadley Wickham, is another excellent resource for learning more about **ggplot2**. The link is posted on D2L. Also the link for all exercises is posted. This resource goes into much greater details on geometric objects, aesthetic mappings, position adjustments, and coordinate systems. It also covers the use of **ggplot2** in graphing statistical transformations and concludes with a discussion of the layered grammar of graphics.

***Knowledge***

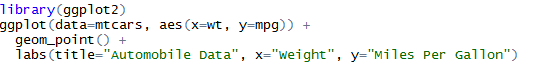
The R function to generate a ggplot is

ggplot(data= , aes(x= , y= ))+geom\_point ()+las (title= , y= , x= )

The arguments:

* data: dataset name
* aes(x= ,y= ): x variable name and y variable name
* geom\_point: plot points; can be a different geom function to generate a line chart, for example, geom\_line
* las (title= , y= , x= ): label the title and x- and y-axis

Let’s generate a scatter plot using the **mtcars** data frame in the base installation of R; it contains automotive details on 32 automobiles. The code below produces the plot in figure 1 which is a scatterplot of automobile weight by mileage.



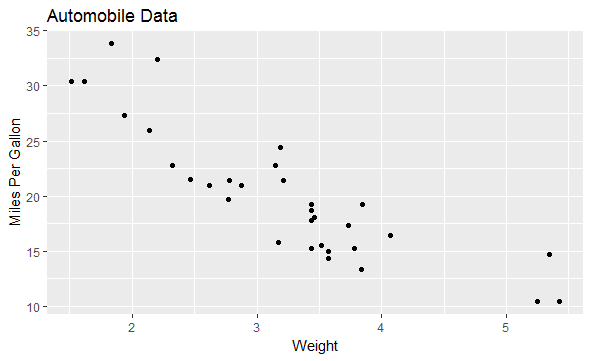


Figure 1. Scatterplot of automobile weight by mileage.

Let’s break down how the plot was produced. The **ggplot( )** function initializes the plot and specifies the data source (**mtcars**) and variables (**wt, mpg**) to be used. The options in the **aes( )** function specify what role each variable will play. (**aes** stands for *aesthetics*, or how information is represented visually.) Here, the **wt** values are mapped to distances along the x-axis, and **mpg** values are mapped to distances along the y-axis. The **ggplot( )** function sets up the graph but produces no visual output on its own. Geometric objects (called *geoms* for short), which include points, lines, bars, box plots, and shaded regions, are added to the graph using one or more *geom functions*.

Options to **geom\_point( )** set the point shape to triangles (**pch=17**), double the points’ size (**size=2**), and render them in blue (**color=”blue”**). The **geom\_smooth( )** function adds a “smoothed” line. Here a linear fit is requested (**method=”lm”**) and a red (**color=”red”**) dashed (**linetype=2**) line of size 1 (**size=1**) is produced. By default, the line includes 95% confidence intervals (the darker band).

The **ggplot2** package provides methods for grouping and faceting. *Grouping* displays two or more groups of observations in a single plot. Groups are usually differentiated by color, shape, or shading. *Faceting* displays groups of observations in separate, side-by-side plots. The **ggplot2** package uses factors when defining groups or facets.

***Key Points to Remember***

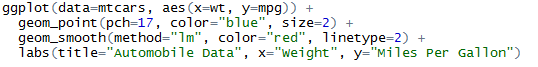
The R function **ggplot()** provides methods for complex graphing. The **geom\_** option provides the choice of graph.

***Practice and Reflection***

***Practice***

Example 1: Create a scatter plot to show the relationship between automobile weight and gas mileage, with a superimposed line of best fit and 95% confidence region.

Solution:



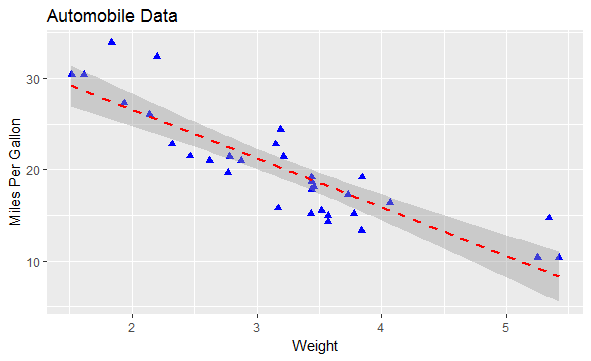
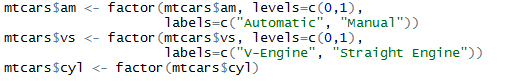


Figure 2. Scatterplot of automobile weight by gas mileage, with a superimposed line of best fit and 95% confidence region.

Example 2: Applying grouping and faceting to the **mtcars** data frame. First, the code below transform the **am**, **vs**, and **cyl** variables into factors.



***Reflection***

Activity 1: Generate a scatterplot showing the relationship between horsepower and gas mileage separately for transmission and engine type.

***Assessment***

Generate a scatter plot for the R data Orange to show the relationship between the variables age and circumference for each type of tree.

#### Learning Objective Two: Identify global setting and local setting for a plot

***Introduction***

The par() function allows us to set up many useful functions globally to be applied to all subsequent plots until we change it. The function par with argument mfrow allows us to set up multiple graphs in one plot.

***Knowledge***

The syntax of the function par() is

par(mfrow=c(x, y))

plot(…)

Viewing multiple plots in one graph help to see the differences between datasets, similarities between datasets, different views of the same dataset, and relationship between different variables of a dataset.

Other functions that can be set up globally using par() are all characteristics for a plot, such as cex for point size, pch for point type, and lty for line type, etc. (see more details of the characteristics in the locally setting section below).

The function dev.off() is to turn off the global setting. If you have applied any global settings from previous commands, state “dev.off()” at the beginning of the new script removes all previous global settings.

***Key Points to Remember***

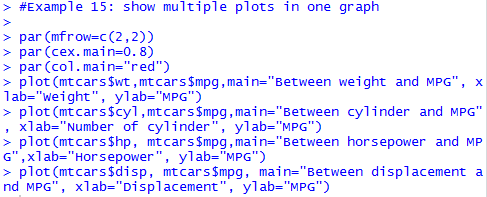
To plot several graphs in one chart, use the R function **par(mfrow=c(x, y))**. To turn off the global setting of doing multiple plots in one chart, use the R function **dev.off()**.

***Practice and Reflection***

***Practice***

Example 2: Use the R dataset mtcars to show four scatter plots in one graph as a 2 by 2 matrix: 1) a scatter plot between weight (wt) and miles per gallon (mpg); 2) a scatter plot between cylinder (cyl) and mpg; 3) a scatter plot between gross horsepower (hp) and mpg; 4) a scatter plot between displacement (disp) and mpg. Make the size of all titles as 0.8 and red.

The R code is



The output graph is shown in figure 3

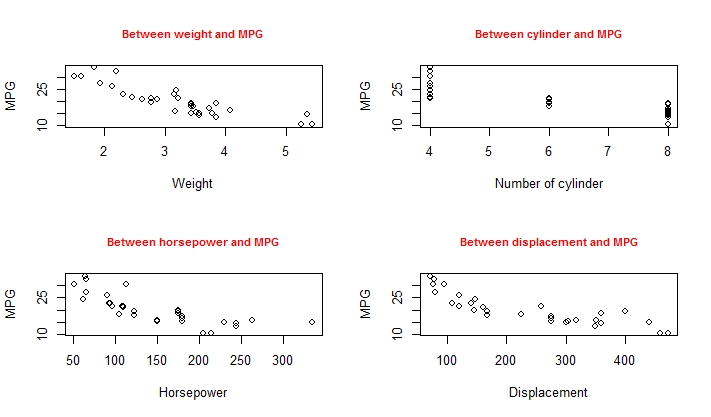


Figure 3. Showing multiple plots in one graph with global setting of title size and color.

***Reflection*** Activity 2: Use the R data set iris to generate two scatter plots in one graph and show the two plots side by side: 1) a scatter plot between sepal length and sepal width; and 2) a scatter plot between petal length and petal width. Make the two plots blue.

***Assessment***

Use the mtcars data in R to generate a box plot and a bar plot in one graph using mfrow(). The boxplot shows the five number summary of mpg for each number of cylinders and the bar plot shows the mean values of mpg for each number of cylinders. Label the x-axis Number of cylinders. Title each plot.

#### Learning Objective Three: Formatting a Plot

***Introduction***

To specify any graph settings locally for one plot, you need to simply declare that setting parameter in the plot () function. For example, if you would like the type of the plot to be both points and lines, then use plot(x, y, type=”b”) to declare that setting.

***Knowledge***

Commonly used settings for a plot is shown below as the arguments of the plot () function:

plot(x,y, type= , pty= , lty= , pch= , col.axis= , col.lab= , col.main= , font= , main= , xlab= , ylab= , xlim= c(), ylim=c())

* type: what type of plot should be drawn. Possible types are
* “p” for points
* “l” for lines
* “b” for both points and lines
* “c” for the lines part alone of “b”
* “o” for both points and lines overlapping
* “h” for histogram like vertical lines
* “s” for stair steps
* “n” for no plotting
* pty: a character specifying the type of plot region to be used; “s” generates a square plotting region and “m” generates the maximal plotting region
* lty: the line type. Possible line types are
* 0 for blank
* 1 for solid (default)
* 2 for dashed
* 3 for dotted
* 4 for dot-dash
* 5 for long-dash
* 6 for two-dash

Line types can be either specified as an integer listed above or as a character string

pch: point choice to be used in the graph. Possible point choices are shown in figure 4

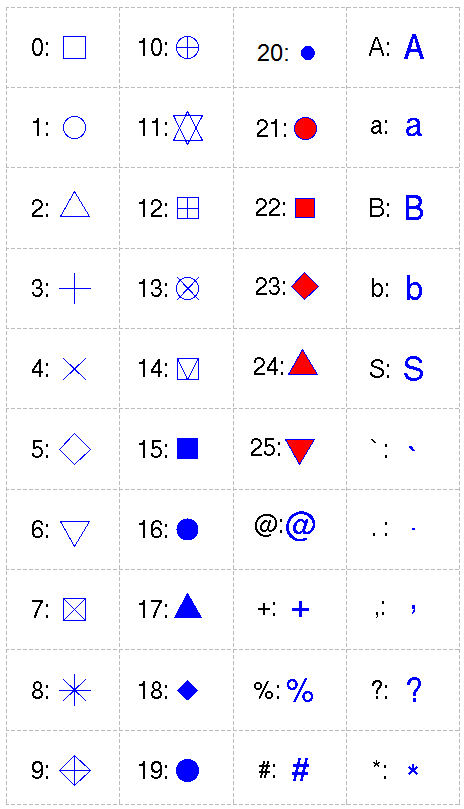


Figure 4. Possible point types in the plot() function.

The other arguments are:

* col.axis: color of the x- and y-axis
* col.lab: color of the labels for the axes
* col.main: color of the title of the graph
* font: font size
* main: title of the graph
* xlab and ylab: label of the x-axis and y-axis
* xlim: a vector with two numbers c(a, b), with the lower limit a and upper limit b for the x-axis
* ylim: a vector with two numbers c(a, b) representing the lower and upper limit of y-axis

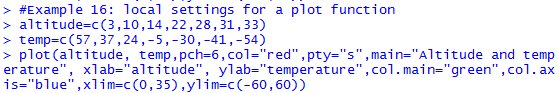
***Practice and Reflection***

***Practice***

Example 3: The research recorded the temperature at 7 altitudes, starting at 6327 feet. Use the scatter plot to show the relationship between altitude and temperature. Plot the graph with red hollow triangle points in a squared plotting region. Make the title green, and the label of the axes blue. Set the x-axis from 0 to 35, and the y-axis from -60 to 60.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Altitude (x) | 3 | 10 | 14 | 22 | 28 | 31 | 33 |
| Temperature | 57 | 37 | 24 | -5 | -30 | -41 | -54 |

The R code is



The output graph is shown in figure 5.

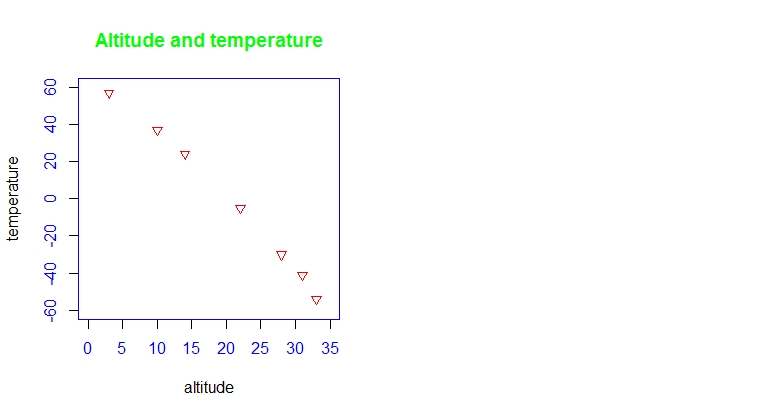


Figure 5. A scatter plot with specific local setting for the point type and color, axes’ color and limits, plot type and title color.

***Reflection***

Activity 3 Use the iris data in R to generate a scatter plot showing the relationship between petal length and petal width. Make the points as blue solid squares. Color the title and axes’ labels as red. Make the x-axis go from 1 to 7, and y-axis from 0 to 3.

***Assessment***

Use the mtcars data in R to generate a box plot and a bar plot in one graph using mfrow(). The boxplot shows the five number summary of mpg for each number of cylinders and the bar plot shows the mean values of mpg for each number of cylinders. Add legend and color to show the difference of the three types of cylinders. Use the title() function to add the titles.

#### Learning Objective Four: Add title, legend, and text to a plot

***Introduction***

We can add titles, axes, labels of axes, text of points, and legend to a plot.

***Knowledge***

To add title, we can either use the argument *main=* in the plot () function as shown in the previous example, or use the function title(). For example, using the altitude and temperature data from previous example, if we do not have the argument *main=* in the plot function, we can add title() after the plot () function



To add a legend to a plot, use the R function legend(x=, legend=, pch=, col=, border =, horiz=) after plot(). In the arguments:

* x: character or number to show the position where the legend should appear
* legend: a character expression vector to appear as the legend
* pch: point types appearing in the legend
* col: the color of points or lines appearing in the legend
* border: the border color for the box of the legend
* horiz: logical; if TRUE (default), set the legend horizontally; otherwise, vertically.

An example R code is



This is to add the legend at the top right corner with the three colors representing three age groups.

To add text to a particular point or place in the plot, use the R function

text(x, y, labels=“ “, adj= ,srt = , font= , cex = , col = ) after the plot () function.

In the arguments,

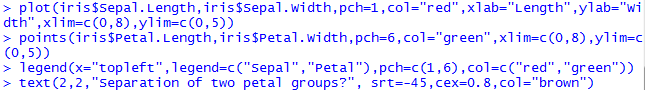
* x and y: the coordinates where the text labels should be written.
* labels: the character vector that specifies the text to appear
* adj: values of 0.5, 0 or 1, where 0.5 (default) is centered, 0 is left-justified, and 1 is right-justified
* srt: the angle that the text appears
* font: font of the text
* cex: size of the text
* col: color of the text

***Practice and Reflection***

***Practice***

Example 4: Use the R data iris to generate a scatter plot showing the relationship between sepal length and sepal width in red color, and the relationship between petal length and petal width in green color. Plot the sepal using red circles (pch=1) and plot the petal using green triangles (pch=6). Add a legend to show the difference between the petal and sepal. Add text “separation of two petal groups?” at the position (2,2) with -45, in brown color with size of 0.8.

The R code is:



The output graph is shown in figure 6

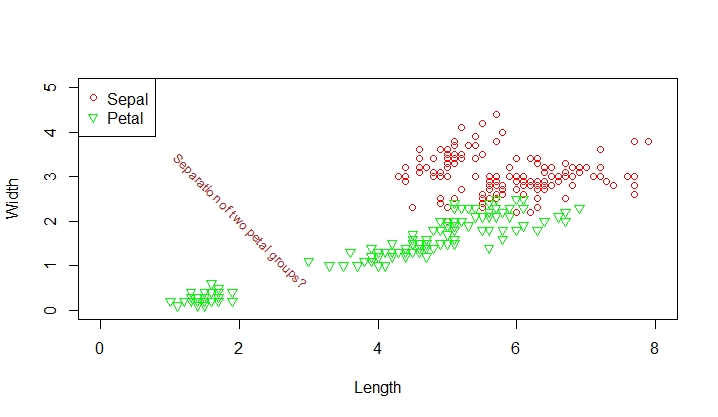


Figure 6. Add legend, and text to a scatter plot.

***Reflection***

Activity 4: The data below shows the rain fall amount in a city at two different years. Generate two line plots in one graph to show the trend of rain fall through time in the two years. Label the x-axis as month, and y-axis as rain fall amount. Make the line for year 2000 red and green for year 2010. Use the type of line across points for both years (type=”o”). Make the y-axis start from 0 and end at 40 with an increment of 5. Add legend to the top right corner of the graph showing the two different years with different color (red for 2000 and green for 2010), with size “cex=0.8”. Add text “maximum point” at the maximum point with black color and size “cex=1”.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | March | April | May | June | July | August | Sep | October |
| Year2000 | 7 | 12 | 28 | 30 | 41 | 39 | 22 | 7 |
| Year2010 | 10 | 9 | 19 | 28 | 32 | 27 | 18 | 3 |

***Assessment***

Use the R dataset Orange to generate five boxplots in one graph. Each boxplot shows the five number summary of the circumference of each tree. The factor Tree is ordered as 3<1<5<2<4, which is not the original order. Rename the factor Tree in ascending order. Add a legend showing “Tree1”, “Tree2”, “Tree3”, “Tree4”, and “Tree5” on the top left corner of the graph, with red being tree1, green being tree2, blue being tree3, yellow being tree4, and gold being tree5. Make the size of the legends 0.8, and use solid dots. Give the graph a title “Orange Tree Circumferences”.

#### Learning Objective Five: Format Axes

***Introduction***

When using the plot() function, R will label the x- tick marks as 1, 2, 3, etc. in an ascending order, and label y-axis tick markers based on the values provided. We can manipulate the axis attributes using the R function axis().

***Knowledge***

If we need to change the labels on the x-tick marks and y-tick marks, we need to firstly claim axes=FALSE in the plot() function to remove all the labels of the x- and y-axis. Then use the function axis() to add all the labels and tick marks. The R syntax of the axis () function is

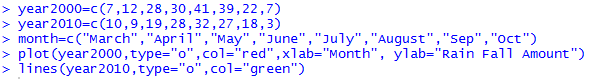
axis(side, at = , labels = , tick = , las=, lwd = , col = , col.ticks = )

* side: 1 for horizontal x-axis; and 2 for vertical y-axis
* at: numeric vector where the tick marks are to be drawn
* labels: character or numeric vector of the labels placing by the tick marks
* tick: a logical value specifying whether tick marks and an axis line should be drawn. The default is TRUE
* las: 0 for parallel labels to the axis, 2 for perpendicular labels to the axis
* lwd: line widths for the axis and the tick marks
* col: color for the axis line and tick marks.
* col.ticks: color for the tick marks if different from the axis color.

***Practice and Reflection***

***Practice***

Example 5: Use the data from Activity 4. If we write the R code as following



Then the graph will not show appropriate x-axis labels as the eight months in words. Instead, the labels of the x-axis will be the default numbers (figure 7 a). If we include the argument axes=FALSE in the plot function

and add the following lines to the code



The output graph will show the correct x-axis label as the eight months (figure 7 b).

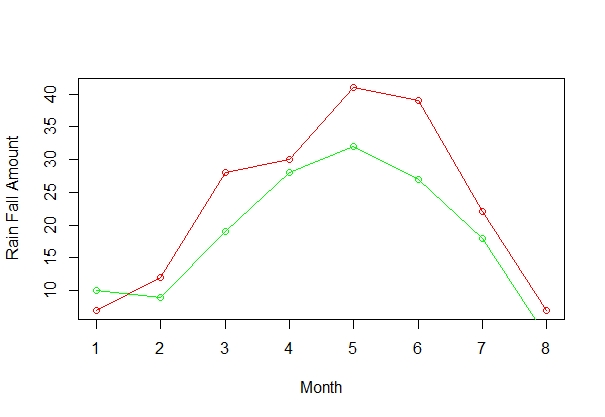


Figure 7 a. Plot of rain fall amount over months without the correct x-labels

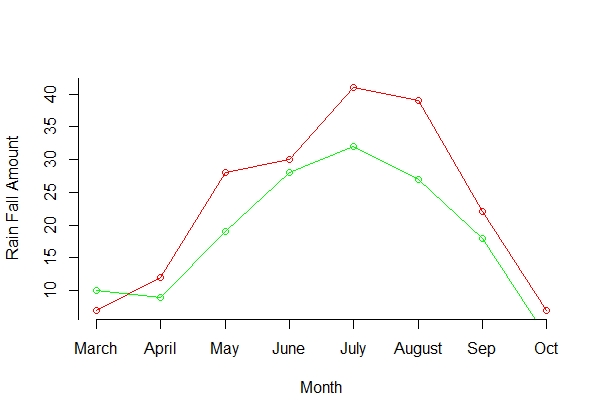


Figure 7 b. Plot of rain fall amount over months with the correct x-labels.

***Reflection***: Activity 5: The data below shows the average price of each type of the car in Minnesota. Genearte a bar graph to show the average price of each type of the car. Label the x-axis tick marks as the five types of cars. Label the y-axis from 0 to 55 with an increment of 5. Include x-label as “Cars” and y-label as “Average Price”. Put the y-axis labels perpendicular to the axis.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Car | BMW | Toyota | Honda | Ford | Volvo |
| Average Price (k) | 52 | 26 | 24 | 22 | 54 |

***Assessment***

Use the R dataset ToothGrowth to generate multiple box plots in one graph to show the five number summary of tooth length(len) for the two supplement type (supp). Add x- and y-axis labels. Add title and legend to the graph. Place the text “Highest Median” right below the highest median line on the graph.