10

Introduction to ggplot2

In the part 3 of this book, beginning now, the subject is data visualization. For the next three chapters, we will learn about ggplot2, enhanced visualizations, such as interactive plots, maps, and 3D graphics, as well as how to plot graphics in Microsoft Power BI using R code.

For this chapter, more specifically, the study will be concentrated in learning the basics of ggplot2, a core library from the tidyverse package to build powerful and highly customizable visualizations. The library was created by Hadley Wickham in 2015 and evolved a lot since then, with 18 new releases. It is built on the premises of the grammar of the graphics, a concept by that states that a graphic must be created in layers, also called grammatical elements for this purpose.

In summary, this chapter will outline:

The grammar of graphics

The basic syntax of ggplot2

The most common plot types available in the library

Grammar of Graphics

Textual communication is supported by a set of rules and elements to help one building a phrase and expressing ideas. The grammatical elements like nouns, verbs, adjectives, preposition, and others are what make possible to form sentences, used for the purpose of communicating a message. With that in mind, in 1999, Leland Wilkinson wrote the book The Grammar of Graphics, where he made an analogy between the way we write and the way we can build a graphic. If we think about that for a minute, it makes sense, since both structures – the text and the graphics – serve the purpose of communicating an idea or a message through data.

The grammar of graphics has these seven elements, which we will go over one-by-one: data, geometries, aesthetics, statistics, coordinates, facets and themes. The first three are fundamental, as without those, there will be no graphic.

Data

The data element means the dataset being worked, what is being used to plot a graphic.

Geometry

The geometry, in the grammar of graphics means the type of graphic to be plotted. As previously seen, there are many kinds of graphics, each one more suitable for a purpose. Histograms are good to look at data distribution, boxplot is the best to discover outliers, scatterplot is interesting to understand correlation between two variables and line plots are more indicated for time series. So, it is a matter of choosing the best geometry based on the data to be plotted and message to be transmitted.

Aesthetics

The aesthetical elements of the graphic are what is to be seen on a plot. What is going to be the x axis, what is on the y axis, the color, fill color and shape of the geometry, the opacity, line width and line type.

Statistics

Statistics element is not always applied, but it refers to graphics where there is need to indicate the number of bins, or if the data points should be counted or summed, or yet the regression type, for the plots where it is available.

Coordinates

There are two possibilities: the cartesian and polar coordinates. Therefore, most of the time the good old x and y cartesian coordinate, which is the default, will be the one used.

Facets

In Data Science, it is not uncommon to need to plot multiple graphics for different groups within a variable. In this case, the facets are a great help, as the same figure can be used to plot all the groups, divided in separate plots. For example, if we need to see the occurrence of sun or rain by day of the week, we can use the facets to show one plot with Yes and No bars for each weekday in the same plotting area.

Themes

Themes are the graphical element available in ggplot2 to bring preset configurations like background color, presence of grid lines, font etc.

The seven elements of the grammar of graphics drive the coding in ggplot2 library. We are going to connect the dots in the next section by seeing how the code is written layer by layer to build a graphic.

Basic Syntax of ggplot2

Every phrase can be broken down into grammatical elements, such as subject, pronouns, adjectives. When read together, the words will make sense, forming a sentence and delivering a message. Likewise, as seen previously, there is a grammar for graphics as well, breaking the creation of a plot down into layers that can be added together to create a visual.

To understand the basics of how to write a ggplot2 code, we will follow a set of questions to walk us through the process smoothly. Whenever there is need to use the library, return to this template until the logic is absorbed and the coding becomes natural.

To create a basic plot in ggplot2, let us answer the following questions:

What is the dataset to be used?

What kind of graphic will be plotted?

What goes on the X axis and Y axis?

What is the graphic title?

These questions can be translated to the following code.

# What is the dataset to be used?

ggplot(data) +

# What kind of graphic to be plotted? (scatterplot)

geom\_point(

# What goes on X and Y axes?

mapping= aes(x= X, y= Y) ) +

# What is the title?

ggtitle('Title for my graphic')

Observe that the code will always start with the function ggplot(data), that will receive the dataset as its argument. That is step one. Then we will add a new layer, and for that we should use the addition sign +. The next layer is the geometry, also known as the graphic type, which can be histogram, boxplot, points, lines, bars etc. Generally, the geometries will have functions like geom\_ and the graphic type, as you see geom\_point() in the example, meaning we are plotting a scatterplot. Within that function is where the mapping of x and y axes is done, using aes(x, y). In this part, one could also add configurations like color, filling color, size and opacity. Finally, we add the last layer to put a title on the graphic, using ggtitle().

The preceding code snippet covers the three fundamental elements, according to the grammar of graphics: data, geometry and aesthetics.

Let’s see that in action using a dataset. First, there must be a dataset, which will be created using random numbers: a distribution of 20 uniform numbers and a distribution of 20 normally distributed numbers.

# Create a sample dataset

df <- data.frame(

var1 = runif(20),

var2 = rnorm(20) )

And the scatterplot (points) can be built using the questions template, just like previously explained in this section.

# What is the dataset to be used?

ggplot(df) +

# What kind of graphic?

geom\_point(

# What goes on X and Y?

mapping= aes(x=var1, y=var2 ) ) +

# What is the graphic title?

ggtitle('My first ggplot2 graphic')

Here is the resultant visualization.

Figure 10.1 – Basic points plot created with ggplot2.

We have learned the basic syntax and built our intuition on how to create a simple graphic. Now it is time to raise the bar and learn other kinds of graphics, as well as adding other elements to our plots, making them more professionally looking.

Plot Types

In chapter 3, when while studying about basic data visualization, we used the toy dataset mtcars. In this chapter, we will go back to it, being able to explore the capabilities of ggplot2 library, as well as comparing it to the base R plots created back then.

To load it to an RStudio session and code along with this book, use the code data(“mtcars”). To make the codes more generic and transferrable to other data frames, I will call the dataset df.

Histograms

The histograms are created using the geom\_histogram() function. As usual, the same questions template is applicable here to write the code.

What is the dataset? df

What is the kind of graphic? Histogram

What goes on x, what is the color, fill color and number of bins? Miles per gallon, with 20 bins.

What is the title of the plot? Histogram of Miles per Gallon.

# What is the dataset to be used?

ggplot(df) +

# What kind of graphic?

geom\_histogram(

# What goes on x, what is the color, fill color and number of bins?

mapping= aes(x= mpg), bins= 20,

color=“lightgray”, fill=“royalblue”) +

# What is the graphic title?

ggtitle(“Histogram of Miles per Gallon”)

Notice that we have added some new elements to the aesthetic portion of the code. After mapping the x axis, we added bins=20, since it is expected for histograms, and also color=“lightgray” and fill=“royalblue” to setup the border and filling color of the bars. The code will display the graphic on Figure 10.2.

Figure 10.2 – Histogram of miles per gallon.

The graphic shows that we have some bins without observations, what could be interpreted as good places to divide the data in groups of MPG, but, if that does not make sense to our analysis, the gaps can be filled by decreasing the number of bins.

Boxplot

The boxplot is a good ally to find outliers, as well as it is sort of a visual T-test, being an excellent resource to compare groups averages. To create it, use the geometry function geom\_boxplot().

From now on, I won’t keep repeating the questions template, just for the sake of space, but keep them in mind when writing ggplot2 codes. The snippet to follow is for a boxplot of miles per gallon. Since it is a univariate plot, there is need to provide only one axis, which will be y, so the box is positioned vertically.

# Boxplot of MPG

# Dataset

ggplot( df ) +

# Geometry, Y and filling color

geom\_boxplot( aes(y=mpg), fill=“royalblue” ) +

# title

ggtitle(“Boxplot of Miles per gallon”)

As result, the graphic displayed is printed in the sequence.

Figure 10.3 – Boxplot of miles per gallon.

Figure 10.3 shows only one outlier for the MPG variable, somewhere around 34 miles per gallon.

Boxplots are also a good choice if you are interested in comparing groups averages. The result will be very similar to what you get when using a statistical T-test (HAIR Jr. et all, 2019), but you will have a visual return with this graphic. Let’s compare the effect of having or not a V-shaped engine on the miles per gallon variable. Notice that this time we are providing the aesthetics element aes() with the x and y axes, where x= factor(vs) indicates that the variable for the engine shape (vs) should be read as categories by ggplot2, not as numbers. The labs()function, now introduced, is used here to rename the x label to Engine shape.

# Dataset

ggplot( df ) +

# Geometry, X, Y and filling color

geom\_boxplot( aes(x= factor(vs), y=mpg), fill=“royalblue” ) +

# overwrite the X label

labs(x=“Engine shape”) +

# Title

ggtitle(“A comparison between V-shaped vs line-shaped engines and the effect on MPG”)

The comparison coded previously outputs the Figure 10.4.

Figure 10.4 – Average comparison of MPG by engine shape with boxplots.

According to the test, the cars with line-shaped engines (1) are more economic than the V-shaped equipped cars.

Scatterplot

The scatterplot is also known as points plot. Ergo, that was the name of the geometry chosen by the library’s creators. Use geom\_point() to create a scatterplot.

This graphic type is very useful for understanding relationships between variables and correlations. Observe the code that follows. There is the basic ggplot(df) to create the figure, the geometry will be geom\_point(), the aesthetics provided to the geometry are x and y axes, the color, size (15 means squares) and shape of the markers, and we added the alpha argument this time, which mean the opacity of the points. Then, we used the function labs() to rename x and y, as well as to include a title and subtitle to the graphic.

# Scatterplot weight versus mpg

ggplot(df) +

geom\_point( aes(x= wt, y= mpg),

color= “royalblue”, size=4, shape=15, alpha=0.7 ) +

labs(x= “Weight of the cars”, y= “Miles per gallon”,

title= “How the weight affects MPG in cars?”,

subtitle= “As the weight increases, the car will make less miles per gallon”)

The graphic plotted is the subsequently displayed.

Figure 10.5 – Customized scatterplot of MPG versus weight.

The plot from Figure 10.5 looks very professional. Observe that the customizations made increase the readability of the graphic. The addition of subtitle helps us to know what to expect from the data, which is a negative correlation between the variables. The opacity makes easier to see where the points overlap. This is especially good when using datasets with more observations, where there is a lot of overlaps between points.

Bar plot

There are many kinds of graphics, but not many as simple to read as a bar graphic. For categorical plot, the bar or column plots are essential, showing counts or values for each category represented. To create such graphic, use the geometry functions geom\_bar() or geom\_col(). There is a slight difference between them, which will be explained next.

When using the geom\_bar() function, we are usually looking at a single variable. If we want to plot a count of observations by transmission type (am), we can call the geometry function and pass factor(am) as the categories to be counted and rename the label x using labs(). Notice one specificity: this time we passed the fill=factor(am) inside the aesthetics function. That small change makes a good difference because it will make ggplot2 to understand that we want to fill one color for each category. So, fill argument within aes() means one color per category, while fill argument outside aes() means the same color for everyone. Let’s see that in action.

# Bar plot

ggplot(df) +

geom\_bar( aes(x= factor(am), fill= factor(am) ) ) +

labs(x=“Automatic(0) | Manual(1)”) +

ggtitle(“Count of observations by transmission type”)

Figure 10.6 is what comes out of the preceding code.

Figure 10.6 – Bar graphic created with the function geom\_bar().

In this dataset, there are more cars with automatic transmission, what is a bit surprising for a 1974 sample of cars.

The bar plot geometry comes with the count statistic associated with it. Hence, if it is needed to change that to mean, for example, we will have to use the statistics functions, which is one of the seven grammatical elements. It is basically the same plot, but adding the arguments stat= “summary” and the indicating that we want to use the mean calculation (fun= “mean”)

ggplot(df) +

# statistic calculation - mean value

geom\_bar(aes(x= factor(am), y= mpg, fill= factor(am)),

stat = "summary", fun=“mean”)

The result will be a bar graphic with the bar being the average value of MPG for each transmission type.

Figure 10.7 – Average MPG by transmission type.

Remember that we did not include a title or renamed the x axis, thus the Figure 10.7 won’t display those updates.

There is yet another syntax, where instead of calling geom\_bar(), we will call the stat\_summary() function, providing it with the aesthetics, plus the function to be used for the y value (fun= “mean”) and the geometry wanted (geom= “bar”).

# Bar plot with two variables

# Another syntax, using stat

ggplot(df) +

# statistic calculation - mean value

stat\_summary(aes(x= factor(am), y= mpg, fill= factor(am)), fun=“mean”, geom=“bar”)

The resultant plot will be equal to the Figure 10.7.

The other way to create a bar or column plot is using geom\_col(). This function requires x and y axes and returns bars with the summed amount of the y axis variables by category.

# Column plot for MPG by transmission type

ggplot(df) +

geom\_col( aes(x= factor(am), y=mpg, fill=factor(am) ) ) +

labs(x=“Automatic(0) | Manual(1)”)

The only change in the code compared to the bar plot is the geometry function, that now is geom\_col(). The code will result in the graphic below. Be aware that values on the y axis are summed, and that amount does not make much sense for our analysis. What we would prefer in this case would be a count or another statistic, like mean or median value.

Figure 10.8 – Column plot of MPG by transmission type.

A work around that is to create a subset with the mean values of the groups and then plot it using geom\_col(), as we did many times in the EDA of the chapter 9.

For the bar plots, we can also change the position argument to make it stacked, stacked filling the entire axis or side by side. So, next, while checking what is the dominant type of engine by cylinder in the dataset, we will plot three bar plots and compare their styles. The code to construct it is similar to past codes, but here we should add the position=“stack” argument in the geom\_bar() function.

# Bar plot stacked

ggplot(df) +

geom\_bar( aes(x= factor(cyl), fill=factor(vs) ),

position = “stack”)

Similarly, we can plot the bars side-by-side.

# Bar plot side

ggplot(df) +

geom\_bar( aes(x= factor(cyl), fill=factor(vs) ),

position = “dodge”)

Or yet, plot them filling the entire plot area, as a 100% bar and the respective categories representation out of the whole.

# Bar plot fill

ggplot(df) +

geom\_bar( aes(x= factor(cyl), fill=factor(vs) ),

position = “fill”)

The result for each plot is on Figure 10.9.

Figure 10.9 – The three positions options for a bar plot.

There they are the three types of position for bar plots, with the cylinders on the x axis and the observation counts on y axis. The coral color is for the V-shaped engines and the teal is for straight line engines. Observe that the engines in line dominate the 4 cylinders models on the left-hand side bars, then the 6 cylinders engines are almost equally split, and the 8 cylinders engines are all V-shaped in this dataset. It is hard to tell which option is the best, as this may vary from a project to another. So, you can choose the one that fits better to your project. In this case, with just two categories, the first plot on the left is more compact and delivers a clear message.

Next, let’s learn about line plots.

Line Plot

The line plots are very indicated to show progression over time. Let’s put the mtcars data aside for a little and suppose that there is a dataset with car sales throughout the months and we want to visualize that information, the line plot would be one of the best indications.

It is easy to create a dataset with months and car sales random numbers to exemplify our case. See the dataset created in the table of Figure 10.10.

|  |  |  |
| --- | --- | --- |
| month | sales | sales2 |
| 1 | 13,709 | 15,277 |
| 2 | 5,646 | 3,066 |
| 3 | 3,631 | 1,466 |
| 4 | 6,328 | 6,995 |
| 5 | 4,042 | 3,126 |
| 6 | 1,061 | 29,221 |
| 7 | 15,115 | 26,845 |
| 8 | 946 | 14,521 |
| 9 | 20,184 | 3,373 |
| 10 | 627 | 19,594 |
| 11 | 13,048 | 1,891 |
| 12 | 22,866 | 13,361 |

Figure 10.10 – Dataset created with random numbers.

To create a line plot, use the geometry function geom\_line(), adding to the aesthetics the x, y, and also a argument called group=1, that is needed to group the sales numbers together, as a single group, so the library knows which points to connect. The complement is size, color and the title.

# Simple Line plot

ggplot(sales) +

geom\_line( aes(x=month, y= sales, group=1),

size=1, color=“darkgreen”) +

ggtitle(“Car sales throughout the months”)

The code displays this graphic shown subsequently.

Figure 10.11 – A simple line plot.

There will be times when we need to plot multiple lines in the same figure, for comparison. Imagine that there are sales numbers from two years in our dataset. In this case, we can use a similar code, just adding another geom\_line() for the new variable to be added.

# Line plot with two variables

ggplot(sales) +

geom\_line( aes(x=month, y= sales, group=1, color=“sales year 1”),

size=1, linetype=2) +

geom\_line( aes(x=month, y= sales2, group=1, color=“sales year 2”),

size=1, linetype=1) +

ggtitle(“Car sales throughout the months - Two year comparison”)

One observation: the color argument was placed within the aes() function and we provided the name for the legend. The code will show the next graphic on the Figure 10.12.

Figure 10.12 – Line plot of two series.

The graphics from ggplot2 are very elegant. And there are more cool features to be presented. Let’s move on.

Smooth geometry

The geom\_smooth() function, as per the documentation, calculates a smoothed line that helps us to see trends in the points, using methods like linear regression, general linear model, polynomial regression and others, to create the trend line that helps in the graphic interpretation.

The use is similar to other geometries. In this case, we will plot a smooth line on top of a scatterplot of the effect of horsepower over miles per gallon. Observe that we add the geom\_point() first and then the geom\_smooth(), inputting the same x and y axes to the function. By default, the method used is a polynomial function (“loess”), but there are linear regression (“lm”), generalized additive models (“gam”), robust linear models (“rlm”) and generalized linear model (“glm”).

# Smooth line geometry

ggplot(df) +

geom\_point( aes(x= hp, y= mpg, color= factor(vs)) ) +

geom\_smooth( aes(x= hp, y= mpg))

The result shows the smoothed curve with a confidence interval band. Very impressive tool.

Figure 10.13 – Smooth line on top of a scatterplot.

The smoothed line on the plot from Figure 10.13 helps to easier see the effect that horsepower has over MPG in this data. Up to 200hp, there is a clear down trend in the miles per gallon, but after that, the effect starts to fade and the cars won’t lose too much more efficiency as they increase the engine power.

Hadley Wickham, the creator of ggplot2, says that we should notice that the previous code is not as efficient as it should be. The axes are being repeated twice since we added two geometries in the plot. That can be easily fixed if we pass x and y axes in the aesthetics function together with the ggplot() function that holds the dataset. This simple change will make the library apply those axes to all the geometries added. Let’s recode the last snippet and see how it will look.

# Smooth line

ggplot(df, aes(x= hp, y= mpg) ) +

geom\_point(aes(color= factor(vs))) +

geom\_smooth()

The result is graphic identical to the Figure 10.13.

Themes

Another grammatical element to support graphic creation in ggplot2 is the theme. Themes are preset visual configurations that one can add to the code as a layer and create a plot that makes more sense with the style of the project or (why not?) the one you like the best.

To add a there, you can choose one of the functions and add that as a coding layer: theme\_bw(), theme\_classic(), theme\_light(), theme\_dark(), theme\_gray(), theme\_linedraw(), theme\_minimal(), theme\_void().

# Adding Theme BW

g = ggplot(df) +

geom\_bar( aes(x= factor(am) ), fill= 'royalblue' )

Next, the figure shows the themes and respective plots.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Figure 10.14 – Themes available in ggplot2.

The Figure 10.14 shows how each theme looks like.

There is a lot more that ggplot2 offers. It is a complete library for plotting of any kind and, if I tried to show everything here, that would be an entire book only for this. Therefore, I highly encourage you to examine the further reading section and look for the documentation to learn everything amazing that this package can do.

Next chapter we will go a little deeper in visualization to create some enhanced types of graphics, such as facet grids, maps, and 3D.

Summary

In this chapter, we studied one of the main graphic packages in the marketing. The library ggplot2 is capable of so much that it was even translated to other languages, such as Python.

We debuted the chapter talking about the interesting theory of the grammar of graphics, which makes an analogy of the textual grammatical elements with the elements needed to construct and plot a good visualization. ggplot2 was built on top of that concept, enabling analysts to code the graphic by layers, adding one piece at a time. We went over a template of questions to help organize the thinking to create the code: (1) start with a dataset; (2) choose a geometry; (3) provide axes and aesthetics; (4) add title, labels, statistics, and themes.

After familiarizing with the syntax, we studied the code for the most used kinds of graphics, like histograms, boxplots, scatterplots, bar plots, and line plots. Then we introduced the smooth geometry, that helps us to create smoothed lines or even linear regression lines on top of graphics, and we ended the chapter showing how to add a theme to the graphic.

Next chapter, we will continue the study about data visualization.