Apply Basic Plot Learning

Table of Contents

Estimated Time Needed: **30 min**

Introduction

Data visualization is the presentation of data with graphics. It's a way to summarize your findings and display it in a form that facilitates interpretation and can help in identifying patterns or trends. Having great data visualizations will make your work more interesting and clear. In this notebook you will learn how to create scatter plots, line plots, and box plots with `ggplot2`.

Scatter Plots

A scatter plot uses points and Cartesian coordinates to display the position of values between two variables.

Let's start by importing the `ggplot2` library.

[ ]:
```
library(ggplot2)
```

We can create a very simple scatterplot simply using `qplot`. Using two variables as parameters makes a scatterplot by default. Here we are using the `mtcars` dataset and are comparing `mpg` (miles per gallon) and `wt` (weight in 1000 lbs).

[ ]:
```
qplot(mpg, wt, data = mtcars)
```

We can create a similar graph using `ggplot` and `geom_point`, which is the more conventional way to create plots with ggplot.

[ ]:
```
ggplot(mtcars, aes(x = mpg, y = wt)) + geom_point(shape = 1)
```

You probably noticed that the shape of the circles changed. This is because of the `shape = 1` parameter in `geom_point`. By default, `shape = 16`.

Since we can control the shapes of the points, we can take advantage of this to plot three variables in the 2D scatter plot. You just need to add the `shape` parameter in the `aes`. So now we are visualizing `mpg`, `wt`, and `cyl`.

Notice that when you put `shape` into the `aes`, it is used to *map that aesthetic to a variable*. In comparison, having `shape` in the `geom_point` changes *all* points.

So now, let's set number of cylinders to the shape, i.e. `shape = cyl`. However, if you were to try to run:

```
ggplot(mtcars, aes(x = mpg, y = wt, shape = cyl)) +
  geom_point()
```

you will get an error because the variable set to `shape` must be *categorical*. We may interpret `cyl` (number of cylinders) to be categorical because it is discrete and there are three categories (4, 6, 8). However to R right now, it is numerical. You must explicitly tell R it is categorical by using `factor()`. In the previous lab, you saw that you can use `mutate` from tidyverse to do this. Another way is the following:

[ ]:
```
mtcars$cyl_factor <- factor(mtcars$cyl)
```

This created a new column, `cyl_factor` in the dataframe `mtcars`. Now we can plot using this new column.

[ ]:
```
ggplot(mtcars, aes(x = mpg, y = wt, shape = cyl_factor)) + geom_point()
```

Another aesthetic we change is the colors of all the points using the `color` (or `colour`) parameter.

[ ]:
```
ggplot(mtcars, aes(x = mpg, y = wt)) + geom_point(color = "blue")
```

As we did with the shapes, we can do the same with colors to add a third variable, number of cylinders. Again, you just pass the `color` parameter into `aes`.

[ ]:
```
ggplot(mtcars, aes(x = mpg, y = wt, color = cyl)) + geom_point()
```

Above, you can see that there is a gradient of blue colors (valued from 4 to 8) because we used `cyl` which is numerical. Like the graph that used shapes, we should use `cyl_factor` since we interpret it to be categorical. A unique color gets assigned for each number of cylinders.

[ ]:
```
ggplot(mtcars, aes(x = mpg, y = wt, color = cyl_factor)) + geom_point()
```

In later lessons, you will go more into detail about ways to customize plots and add labels. Here, we will add some basic labels so that people who view the graph can better understand what it means. Note that in `labs`, the parameter `color` changes the legend title.

[ ]:
```
ggplot(mtcars,aes(x = mpg, y = wt, color = cyl_factor)) +
  geom_point() +
  xlab("Miles per Gallon ") +
  ylab("Weight") +
  labs(colour = "Cylinders") +
  ggtitle("Comparing mileage and weight by number of cylinders in cars")
```

Question #1:

Create a scatter plot using the "mtcars" dataset. Set the x-axis as "hp" (gross horsepower) and the y-axis as "qsec" (1/4 mile time). Also, color the points by "gear" (number of forward gears). As a bonus, add more meaningful labels and a title to the plot.

Hint: Use "factor()" to change a variable to categorical.

[ ]:
```
# Write your code below and press Shift+Enter to execute
```

▼Click here for the solution.

```
mtcars$gear_factor <- factor(mtcars$gear)
ggplot(mtcars, aes(x = hp, y = qsec, color = gear_factor)) +
    geom_point() +
    labs(x = "Gross horsepower", y = "1/4 mile time",
         color = "Forward gears",
         title = "Gross horsepower vs 1/4 mile time by number of forward gears")
```

Line Plots

Line plots are also representations of data in which Cartesian coordinates are used. Much like scatter plots, the data is transformed into points - however, in line plots, they are connected by lines, as the name implies.

Simple Line Plots

For line graphs, we are going to use the EuStockMarkets dataset. It is also a dataset that comes included with R and it describes four European Stock markets' historical data. Namely, DAX (Germany), SMI (Switzerland), CAC (France), FTSE (UK).

Let's start looking at their helpfile.

[ ]:
```
?EuStockMarkets
```

[ ]:
```
head(EuStockMarkets)
```

As we can see, it contains 1860 observations for each, and the type is `mts`, which stands for Multivariate Time Series. The `mts` object is an object built into R in the `stats` library. There are also `ts` or time series objects with just one variable. You can also extract additional information from these objects, for example the `time()`:

[ ]:
```
head(time(EuStockMarkets))
```

`ggplot2` doesn't work with time series objects. So it must be converted to a dataframe. Now, we will:

1. Convert the time series object into a dataframe with `as.data.frame()`, and
2. Add the date column with `time()`

[ ]:
```
EuStockDF <- as.data.frame(EuStockMarkets)
EuStockDF$Date <- as.numeric(time(EuStockMarkets))
```

Now, let's check to see how the data is structured.

[ ]:
```
head(EuStockDF)
```

Now, let's start plotting!

With ggplot, you can use `geom_line` to add lines. Let's create a line graph of the "DAX" stock price.

[ ]:
```
ggplot(EuStockDF, aes(x = Date, y = DAX)) + geom_line()
```

Let's increase the width of our line with the `size` parameter. You can also change the color with the `color` parameter. We will put these parameters in the `geom_line` function.

[ ]:
```
ggplot(EuStockDF, aes(x = Date, y = DAX)) +
  geom_line(size = 1.5, color = "light blue")
```

Question #2:

Create a line plot using the "EuStockDF" dataset. Set the x-axis as "Date" and choose any other stock index price besides DAX. Also, change the color of the line to red.

[ ]:
```
# Write your code below and press Shift+Enter to execute
```

▼Click here for the solution.

```
# Example solution using CAC
ggplot(EuStockDF, aes(x = Date, y = CAC)) +
  geom_line(color = "red")
```

Multiple Line Plots

One way you could create a multiple line plot is to add `geom_line()` for every line you wish to have. Using "EuStockDF", we can add a line for every stock index. We update the `y` with the stock index we are adding and you also must specify the color or else all the lines will be the same color.

[ ]:
```
ggplot(EuStockDF, aes(x = Date)) +
  geom_line(aes(y = DAX), color = "light blue") +
  geom_line(aes(y = SMI), color = "red") +
  geom_line(aes(y = CAC), color = "purple") +
  geom_line(aes(y = FTSE), color = "green")
```

However, this is tedious since you have to manually specify the color of the line and there is no legend to label each line. Another downside of using this method is that the y-axis labels you may not realize that your y-axis may be misleading and not be the same units. It may take a lot of customizations to make this plot interpretable.

The *better* option is to put the data in a "tidy" format and then plot. For example, if the below table is the original table you can see that each column is a different stock index and each value is a price.

stock1	stock2	stock3
1	10	100
2	20	200
3	30	300

The "tidy" version would be to combine all the prices into one column and have another column to denote the stock index type.

stock	price
stock1	1
stock2	10
stock3	100
stock1	2
stock2	20
stock3	200
stock1	3
stock2	30
stock3	300

You can use the `tidyverse` package to tidy up dataframes. The package, `broom` is included, however it is not automatically loaded when `tidyverse` is loaded. So you must explicitly load it.

[ ]:
```
library(tidyverse)
library(broom)
```

There is a function `broom::tidy()` that can easily convert `mts` object types into a tidy dataframe. By default, the name of the columns will become "index", "series", and "value". So you can rename them with the `rename` function. Remember that the time is hidden in `mts` objects, so that is why the date is extracted in the tidy version.

[ ]:
```
tidy_stocks <- tidy(EuStockMarkets) %>%
  rename(Date = index, Stock_Index = series, Price = value)

head(tidy_stocks)
```

Now, the data is in the correct format. We can plot this by using `geom_line()` just once. Now, in the `aes()`, set `color = Stock_Index`. The different stock index types (DAX, CAC, FTSE, SMI) will automatically be detected. Each line will have its own color and a legend is created.

[ ]:
```
ggplot(tidy_stocks, aes(x = Date, y = Price, color = Stock_Index)) +
  geom_line()
```

As you can see, with "tidy" data, a beautiful and informative graph can easily be created with ggplot2.

Question #3:

Create a line plot using the "Orange" dataset, which is built in dataset about the growth of orange trees. Set the x-axis as "age" and set the y-axis as "circumference". Create a line for every "Tree".

Hint: This dataset is already in a "tidy" format.

[ ]:
```
# Write your code below and press Shift+Enter to execute
```

▼Click here for the solution.
```
ggplot(Orange, aes(x = age, y = circumference, color = Tree)) +
  geom_line()
```

Box Plots

Box plots are good to indicate dispertion. They do so by providing visual representations in terms of quartiles. The main parts of a box plot are:

- The **median** (second quartile or Q2) of the data, which represents the middle datapoint. In the plot, it is the thickest line in the middle of the rectangle.
- The **Upper Quartile** (third quartile or Q3), which is the 75th percentile. In the plot, it is the top of the rectangle (or rightmost of the rectangle if it is oriented horizontally).
- The **Lower Quartile** (first quantile or Q1), which is the 25th percentile. In the plot, it is the bottom of the rectangle (or leftmost of the rectangle if it is oriented horizontally).
- The **Interquartile Range** (IQR), which is the difference between the first and third quartiles.
- **Outliers** are defined as values less than Q1 - 1.5 _ IQR or greater than Q3 + 1.5 _ IQR. In the plot, these are the dots.

A nice library to use with boxplots is "plotly" because it makes the plot interactive and you can see all of these parts. So first, "plotly" should be installed.

[ ]:
```
install.packages("plotly")
library(plotly)
```

With ggplot, adding `geom_boxplot` creates the boxplot. If you want to visualize just one numerical variable, you can set it to `y` in `aes`. In this example, we look at `mpg` in the "mtcars" dataset. Then, you can pass in the ggplot object to `ggplotly` (from the "plotly" library) to make it interactive.

[ ]:
```
p <- ggplot(mtcars, aes(y = mpg)) + geom_boxplot()
ggplotly(p)
```

In the interactive boxplot above, you'll notice that hovering above it with your mouse will display the different components of it. For example, the median is 19.20 here.

Additionally, if you wanted to break down a *numerical* variable by a *categorical* variable, you could set the x for categorical and y for numerical. For example, say you want to look more into the `mpg` variable and break it down by the number of cylinders. Remember, you have to use the *categorical* version of number of cylinders, `cyl_factor`. You can set `x = cyl_factor` and `y = mpg`.

[ ]:
```
ggplot(mtcars, aes(x = cyl_factor, y = mpg)) + geom_boxplot()
```

As a good exercise on your own, you can try making the plot above interactive with `ggplotly`.

Since throughout this lab we needed the categorical version of `cyl` several times, it was better to just create the `cyl_factor` column and call `factor()` once.

```
ggplot(mtcars, aes(x = cyl_factor, y = mpg)) + geom_boxplot()
```

However, it is possible to just use `factor()` inside ggplot instead of creating that extra variable like so:

```
ggplot(mtcars, aes(x = factor(cyl), y = mpg)) + geom_boxplot()
```

Using `ggplot() + geom_boxplot()` is more common in practice. However if you did want to use `qplot()`, you could set `geom = "boxplot"`:

```
qplot(cyl_factor, mpg, data = mtcars, geom = "boxplot")
```

Question #4:

Create a box plot using the "mtcars" dataset. Set the x-axis as "hp" (gross horsepower) and break down the plot by the categorical variable "gear" (number of forward gears). Do not use qplot().

Hint: Use the variable "gear_factor" you created earlier.

[ ]:
```
# Write your code below and press Shift+Enter to execute
```

▼Click here for the solution.

```
ggplot(mtcars, aes(x = gear_factor, y = hp)) + geom_boxplot()
```

Conclusion

In this lab, you learned how to create visualizations with the format `ggplot() + geom_function()`. We focused on three plot types:

- **Scatter plots** - Use geometry function `geom_point()`. Visualizes two continuous variables with the x and y axis. You can visualize another variable with the `shape` or `color` parameters.
- **Line plots** - Use geometry function `geom_line()`. Visualizes continuous variables and is useful for time series data.
- **Box plots** - Use geometry function `geom_boxplot()`. Similar to histograms, it visualizes the distribution of a numerical variable. The numerical variable can be broken down by a categorical variable as well. Also, `ggplotly()` from library "plotly" makes plots interactive.

Feel free to further explore these plots with different datasets and variables!

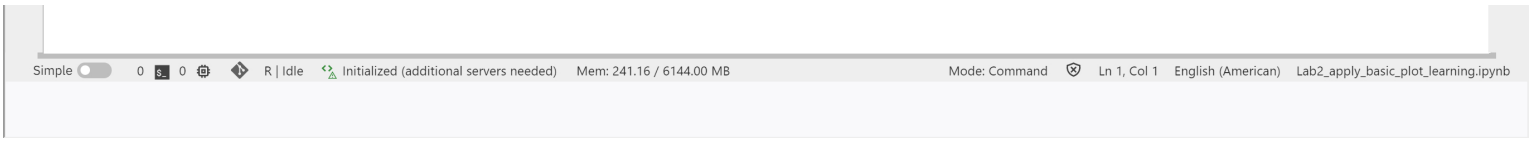
Thanks for completing this lab!

About the Author:

Hi! It's Yiwen Li and Tiffany Zhu, the authors of this notebook. We hope you found R easy to learn! There's lots more to learn about R but you're well on your way. Feel free to connect with us if you have any questions.

Other contributors:

Francisco Magioli and Erich Natsubori Sato


Simple
0
0
R | Idle
Initialized (additional servers needed)
Mem: 241.16 / 6144.00 MB
Mode: Command
Ln 1, Col 1
English (American)
Lab2_apply_basic_plot_learning.ipynb