Data Visualization in R

An introduction to Statistics and their use in R

Disclaimer

The purpose of this tutorial is simply to provide an introduction to a variety of topics in R. It is in no way a substitute for a full for-credit course on R, and should students want to pursue R on a more serious level they should look to the university courses on the subject

Knowing your audience

It is always important when creating a data visualization of any type that we are asking ourselves this important question: Where will your visualizations be seen? A scientific conference, an analytics consulting presentation? What will this audience understand, what won't they understand? As you learn to use R for this task, be aware of what you may be using it for in the future.

Base Graphics

R has a wide range of built in plotting functions in R, these are called the ***base graphics***. We will use those before we start talking about more advanced plotting software.

plot()

The most basic plotting function in r is simply called *plot()*. Like with most functions in R, plot() has a minimum number of inputs, but there are many additional arguments that you can add.

Code: plot(x variable, y variable)

This will create quick and simple scatter plot of our data.

A note on x and y variables:* In statistics, we differentiate between two types of variables: a ***Response variable* and an ***Explanatory variable***. A response variable is a random variables whose behavior we often want to predict using the values of the explanatory variable. Simply put, a response variable is the particular quantity that we ask a question about in our study. An explanatory variable is any factor that can influence the response variable.

Generally speaking, we will plot our explanatory (or independent) variable on the X axis, and our response variable on the Y axis.

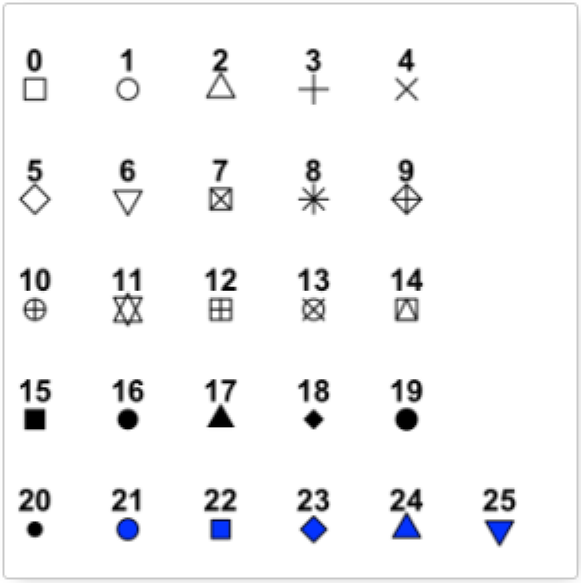
Adding arguments to our plot

Once we have plotted our basic data we notice a few things, firstly, our display is very bland, and the labels for the axis's could be confusing to an outside observer. Here is where we decide to add additional elements to our plot:

xlab = "X-axis label" <- labels your X-axis
ylab = "Y-axis label" <- labels your Y-axis

xlim = c(lower limit, upper limit) <- Limits what data is shown on X-axis
ylim = c(lower limit, upper limit) <- Limits what data is shown on Y-axis
main = "Title of plot" <-gives your chart a title
You also can use a function **title()** after your plot for the same effect
legend = <- adds a legend to your plot

Data Point Style
col = "Color" <- changes the color of your data points
pch = Shape ID <-changes the shape of your data points
Shape options:



Adding Text
To add text to a chart we use a function called **text()**

Example:
text(x value of location, y value of location, text you want displayed)

Optional additional arguments:
cex = size of font
pos = position (kind of like an offset)
col =color
font = changes font style. 1 = standard, 2 = bold, 3 = italic, 4 = bold italic
Family = changes font. Common types are: "serif", "sans", "mono"

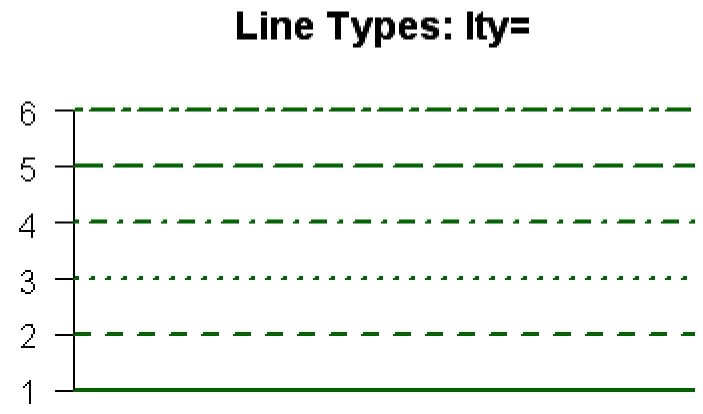
Adding a line to our plot
Say we want to add a straight line to our graph, something with a function y = a + bx, where a is the y intercept and x is the slope of the linear line. To do this we use a function called abline.

abline(v = #) <- adds a vertical line
abline(h = #) <- adds a horizontal line
abline(a, b) <- adds a line with y-intercept a and slope b

Want to add a curved line? Use the **Curve** function.

Curve(function of x) <- adds a curve based on some function of x

Change the ***line type*** by adding an argument called **lty** = #



Other types of charts
***Histogram:* hist()**
The simplest way to quickly understand the distribution of a set of data

Code: hist(iris$Sepal.Length)

***Barplot*: barplot()**
A bar plot simply displays using height the frequency of different items in a parameter.

The special parameters that apply to a bar plot are:
- beside = TRUE/FALSE (False is the automatic value)
- horiz = TRUE/FALSE (False is the automatic value)

***Boxplot*: boxplot()**
A boxplot is a display that shows the distribution of the data using median and and the inter quartile range.

The special parameters that apply to a box plot are:
- varwidth=TRUE/FALSE
- horizontal=TRUE/FALSE

Code: boxplot(iris$Sepal.Length, iris$Petal.Length, horizontal = TRUE, beside = TRUE)

***Pie Chart*: pie()**
Don't use pie charts.

***Pairwise Scatter plots*: pairs()**
Creates scatter plots for every possible combination of some data set, good for quick visualization

***Quantile Quantile plot*: qqplot()**
A QQ-plot compares two distributions of data and will show the relationship between the data. This could demonstrate whether or not a dataset is skewed in one direction.

Ggplot2
ggplot2 is based on an idea called the ***grammar of graphics***, the concept being that you can build every graph from the same few components: a **data set**, a set of **geoms**—visual marks that represent data points, and a **coordinate system.**

Advantages of ggplot2

- consistent underlying grammar of graphics
- plot specification at a high level of abstraction
- very flexible
- theme system for polishing plot appearance
- mature and complete graphics system
- many users, active mailing list

Disadvantages

- Tends to overcomplicate the simple graph types
- Can only take data that is in a data frame
- Uses a different system for adding plot elements

Installation and use
First we need to install the package ggplot2

Code: install.packages("ggplot2")

Now we can start using it! As mentioned above, every plot in ggplot follows the same basic code structure:

ggplot(*data set*, *geoms*, *coordinate system*)

Good cheat sheet for using ggplot2: https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf

Pipe operator
%>%
Dplyr package

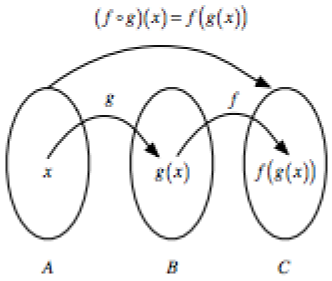

$$\begin{tikzcd} x \arrow[r, bend left, "g"] \arrow[rr, bend left=60, "(f \circ g)(x) = f(g(x))"] & g(x) \arrow[r, bend left, "f"] & f(g(x)) \end{tikzcd}$$

Image Credit: James Balamuta, "Piping Data"

GGplot2
(Courtesy of Leon Cui in the SCM department)

<u>Basics with geom point</u>

dat% > %ggplot(aes(x = displ, y = hwy)) + geom point()

<u>Details of each feature</u>

Use ggplot() function to create a blank canvas. Feed in aesthetics ("aes") which give details on what each component is going to look like. Then add layers or components using "+". e.g. +geom point() adds points to the plot. You can also add aesthetics to each component (see cheat sheet for complete list). For geom point the important ones are color, shape, size.

e.g. dat% > %ggplot(aes(x = displ, y = hwy)) + geom point(aes(col = class))

Adding in shape of each point for cylinder: First try: dat% > %ggplot(aes(x = displ, y = hwy)) + geom point(aes(col = class, shape = cyl))

OK but one issue is that it thinks "cylinders" is a continuous variable because it's numeric.

Solution is to convert it to a factor first. dat% > %ggplot(aes(x = displ, y = hwy)) + geom point(aes(col = class, shape = factor(cyl)))

if you wanted to name the title or the axes you can add in +labs(title = "newtitleoo , x = "newxoo , y = "newyoo)

-jitter (description when you pull it up)
-discrete variables overlap so you cannot see density
-moves points by small amount so there is no overlap

-ggpairs

-plot for each pair of factors including discrete(finite numeric)/categorical (finite qualitative)