Data visualization using ggplot2

Rutger Brouwer

r.w.w.brouwer@erasmusmc.nl

Center for Biomics
Erasmus Medical Center

NBIC BioAssist

Installing R

R can be obtained http://www.r-project.org/, but due to the limited band-with in this venue problems may arise in downloading the R software. It is recommended that you install R with the ggplot2 library prior to this tutorial.

In order to install ggplot2 and all of its dependencies start-up R and type the following:

```
> install.packages( 'ggplot2' )
```

This will install the following packages:
* itertools
* iterators
* reshape
* proto
* plyr
* digest
* colorspace
* ggplot2

The ggplot2 library should now be installed and ready for use.

Using R for data visualization

R is an open-source software platform that allows researchers to perform both statistics and visualize their data. On the strengths of R is that functionalities can be easily added through third-party modules. In that sense R is somewhat of a mix of traditional statistics software, such as SPSS, and scripting languages, such as Perl and Python. The [http://www.bioconductor.org bioconductor] project provides numerous extensions to R to deal with biological data. Currently bioconductor provides 417 software libraries that deal with all kinds of biological problems ranging from annotation to expression data analysis.

In this tutorial, we will focus on visualizing data with R. R is well known for it's capability to easily visualize large and differs datasets. The base package of R allows users to create xy scatterplots simply as typing plot(x_values,y_values). If you want to have a histogram, simply type in hist(x_values) and a histogram pops on the screen. This manner of plotting is ideal for simple graphs but does not scale well. Let's say, we would like to have a scatter-plot with x and y values and have the color of the data-points indicate their class(A or B). In order to do this in base::plot, the classes should be mapped to a color, the color vector should be made and a legend should be added. New libraries have been introduced in recent years that allow R to perform such model based visualizations without extensive coding. Among these, the ggplot2 library is one of the easiest to use.

The ggplot2 library has been developed by Hadley Wickham and is based on the grid graphics library. The grid library is object oriented and allows graphics objects to be modified after they've been drawn. This is advantageous for drawing complex layouts. The ggplot2 library combines this library with an underlying model that should allow for the visualization of most if not all statistical data. The ggplot2 library is quite too extensive to deal with completely in this tutorial. For further reading, we recommend the ggplot2 website (http://had.co.nz/ggplot2/) and the book (Elegant graphics for data analysis).

Getting started

Loading your data

For the purposes of this tutorial, a typical dataset has been prepared. This dataset contains a set of annotated ChIP-seq peaks with expression values associated to these genes. The genes in this set have been anonymized as this is actual research data.

To load this dataset perform the following command:

```
> data <- read.table( "course_data.txt" , header=T, sep='\t',
comment.char=‘, quote=‘ )
```

‘Note: By setting the "comment.char" and "quote" arguments to ‘, R is able to read lines that contain '#' characters’

The dataset is a regular data.frame object in R with 4701 rows and the following columns:

```
> colnames(data)
[1] "seqnames"         "width"            "chip.peak.height"
"ctrl.peak.height" "significant"
[6] "logFC.induced"    "logFC.noninduced"
```

Simple plots

When using the base graphics we would use something like the following to create an ordinary scatter-plot of the peak-heights of the ChIP versus the control:

```
> plot( data$chip.peak.height, data$ctrl.peak.height, col="red", pch=20)
```

We order R to do several in this single command, namely to plot 2 separate datasets in a single plot, to color the points red and to shape the points with pch 20 (a closed circle).

There are several issues with this method plotting your data
* There is no coupling between variables in the same dataset. It is possible to mix and match variables from diverse datasets with a single command. This may cause errors.
* Graph mark-up needs to be performed in the same command as plotting. There are no meaningful or aesthetically pleasing defaults for your graphs.
* Making a different graph of the data requires a different command.
* Data-transformations need to be performed manually.

The ggplot2 library takes care of many of these issues. To start working with the ggplot2 library type the following commands:

```
> library(ggplot2)
> p <- ggplot( data=data )
```

With the ‘library(ggplot2)’ command the ggplot2 library is loaded. This library introduces a number of new functions in to the work-space. Most of these will be used to generate or modify graphics. The ‘p <- ggplot(data=data)’ command makes a new ggplot object and assigns it to variable p. The ggplot obejct contains the data that can be displayed using the various functions in the library. As the plot functions only work in concordance with a ggplot object, a conscious decision should be made as to the various aspects in the data.

The functions of ggplot2 visualize various aspects of the ggplot object. In order to make a graph of the data in ‘p’, just use the ‘+’ operator:

```
> p + geom_point( aes( x=chip.peak.height, y=ctrl.peak.height) )
```

This will make a scatter-plot of the chip.peak.height column versus the ctrl.peak.height column in the dataset. The ‘aes()’ function (aesthetic) maps the variables in the graph to variables in the dataset. In the example the x-values are mapped to the ctrl.peak.height and y-values are mapped to the ctrl.peak.height values. If we would like to introduce a fixed value for the point color, it should be added outside of ‘aes()’.

```
> p + geom_point( aes( x=chip.peak.height, y=ctrl.peak.height),
colour='red' )
```

Now the colour of the points is red instead of the default value black. We could use use the ‘aes()’ function to map different aspects of our data to properties of the data-points. For example, we might want to indicate whether or not the peaks were significant with the colour.

```
> p + geom_point( aes( x=chip.peak.height, y=ctrl.peak.height,
colour=significant) )
```

This will colour the points either green or red for the values in the signifcant column of the dataset (TRUE or FALSE). Ggplot2 also added a legend to the plot with the scale of the significant variable. In this case this variable was factorial, but we could also add continuous variables:

```
> p + geom_point( aes( x=chip.peak.height, y=ctrl.peak.height,
colour=significant, size=width) )
```

Now, an additional legend is added with the mapping of the size variable.

The graphing functions in ggplot have a fixed nomenclature that holds information on their purpose.
* ‘geom_’ indicate geometry functions that describe the type of plot made
* ‘stat_’ are statistical functions that transform the data in the plot
* ‘scale_’ controls how the data is mapped to the plot properties
* ‘coord_’ change the coordinate systems of plots
* ‘facet_’ are faceting functions allow subsets of data to be displayed
* ‘position_’ adjust the position of points in a plot and allow for fine-tuning.

In addition to these functions the ‘opts()’ function allows us to set attributes in the plots, such as the main and axis title(s) in a plot.

Using ggplot2, it is very easy to generate different graphs over the same data. We will do this now by making 2 graphs from the same ggplot object ‘p’. We will save the second plot to a variable for efficiency reasons. The ‘geom_histogram’ function implicitly calls the ‘stat_bin’ function that performs a transformation of the data. To save these results, we assign the resulting ggplot object to a new variable.

```
# make a scatterplot
> p + geom_point( aes( x=chip.peak.height, y=ctrl.peak.height,
colour=significant) )
> dev.new()
# make a histogram
> tmp <- p + geom_histogram( aes( x=chip.peak.height ) )
> print( tmp )
```

The ggplot object uses the 'print' function to print output to the graphic device. This differs from the base graphics that plot directly to the devices. In the previous example,
the histogram was assigned to 'tmp'. By doing 'print(tmp)' we make the graph. Let's say we are only interested in a specific region in the histogram (from 20 to 80). We can modify 'tmp' using a 'scale_' function to show only this region:

```
> tmp + scale_x_continuous( limits=c(20,80) )
```

To make our graph more interesting, we can try to play around with the 'coord_' functions. First we will flip our x and y axes using the 'coord_flip()' function.

```
> tmp + scale_x_continuous( limits=c(20,80) ) + coord_flip()
```

However, the resulting graph is not really more exciting than the original..

A radar plot is not really an intuitive way of looking at data, but it does look cool. To make the graph more page filling, we will make use of a log10 transformed axis.

```
> tmp + scale_x_log10( limits=c(20,80) ) + coord_polar()
```

We could also generate separate plots based on specific properties of the data (such as the significance)

```
> tmp + scale_x_log10( limits=c(20,80) ) + coord_polar() + facet_wrap(
~significant )
```

Preparing graphs for publication

Theming

The ggplot2 library also allows us to theme our graphs. The default theme with grey background was chosen to make graphs as readable as possible, but it may not be to your tastes or to a journals specification. Ggplot2 allows the appearance of every non-data element in the graph to be changed via the 'opts' and 'theme_' functions. This method is quite laborious and time-consuming as we can see in the following example:

```
# make a scatterplot with a grey background and a red border.
> p + geom_point( aes( x=chip.peak.height, y=ctrl.peak.height,
colour=significant) ) + opts( plot.background = theme_rect(fill = "grey80",
colour = "red" ) )
```

Luckily the ggplot2 library also provides a black and white theme that takes considerably less time to modify. This theme is set using the ‘theme_seq’ and ‘theme_bw’ functions:

```
# make a scatterplot with a grey background and a red border.
> theme_set( theme_bw() )
> p + geom_point( aes( x=chip.peak.height, y=ctrl.peak.height,
colour=significant) )
```

High resolution images

To quickly save your graph after you generated it on the screen, the ggplot2 library offers the ‘ggsave’ function. This function has one mandatory argument, the file-name, with which it will decide what kind of output format to generate. In order to generate a png image of the last plot just do the following:

```
# make a scatterplot with a grey background and a red border.
> ggsave( filename="mygraph.png" )
```

By default, the ggsave function will make 300 dpi images by setting the ‘dpi’ parameter different dpi's can be chosen.

```
# make a scatterplot with a grey background and a red border.
> ggsave( filename="mygraph.png", dpi=96 )
```

This function allows us to first tweak our plots and then save them without running all the commands again manually. In addition it takes care of some of R's syntax ambiguities for output devices by introducing a single parameter for dpi where R itself has multiple (‘res’ and ‘dpi’).

Concluding remarks

The ggplot2 library contains many more functions and visualizations than we have touched upon in this small tutorial. For the complete list, please read the book or go to the website (http://had.co.nz/ggplot2/). However, we hope to have giving you a feel for what is possible using this library and that is does make generating complex graphs easy.