Base R Graphics

Cheng Peng

Lab Note: STA321 Topics of Advanced Statistics

Contents

1 Introduction **1**

2 R Graphic Devices **1**

3 Plotting with function `plot()` **1**
3.1 Outer and Inner Margins 2
3.2 Simple multi-panel Plots 4

4 Making Your Own Color **7**

1 Introduction

This note introduces the graphical capabilities of base R graphical functions. Base R graphical system contains a set of **high-level plotting functions** such as `plot()`, `hist()`, `barplot()`, etc., and also a set of **low-level functions** that are used jointly with the high-level plotting functions such as `points()`, `lines()`, `text()`, `segments()`, etc. to make a flexible graphical system.

2 R Graphic Devices

R is able to output graphics to the screen or save them directly to a file (e.g. postscript, pdf, svg, png, jpeg, etc.). The different functions for producing graphical output are known as **Graphic Devices**. For example, `pdf()` would invoke the `pdf device`, while `png()` would invoke the `png device`. Type ?Devices into the R console to see a list of graphical devices that are available to R on your system.

By default, graphical output is sent to the screen. As R is cross-platform, the graphics device for producing **screen** graphics differs by the system. The available fonts may also differ by the system and graphical device.

3 Plotting with function `plot()`

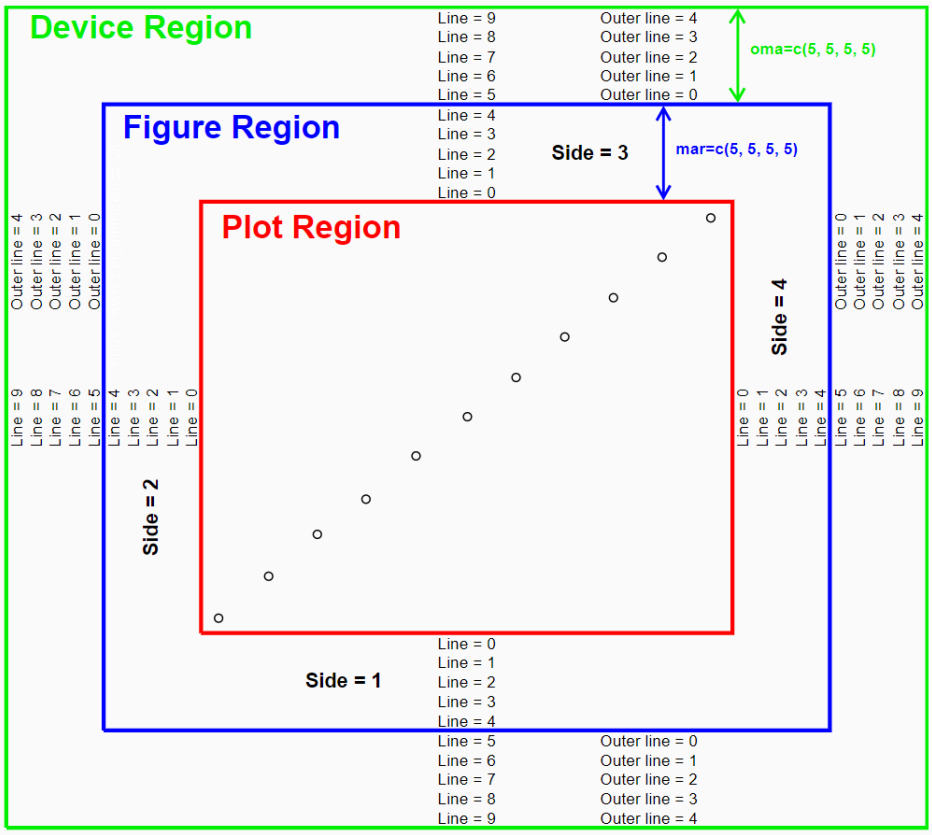
`plot()` is the most important high-level graphic function. When using `plot()`, the output is sent to the **graphics device**. This creates a "plot region" on the graphics device. The visible area of the graphics device is also known as the "device region". The plot region on the other hand is the area of the plot (usually bounded by a box) and does not include the plot axes, labels, or margins.

The plot region, plus the axes, labels, and margins are known as the "figure region". Often, the device region and figure region can be the same size - but they are not the same thing.

3.1 Outer and Inner Margins

`par(mar=c(x, x, x, x), oma = c(x, x, x, x))`

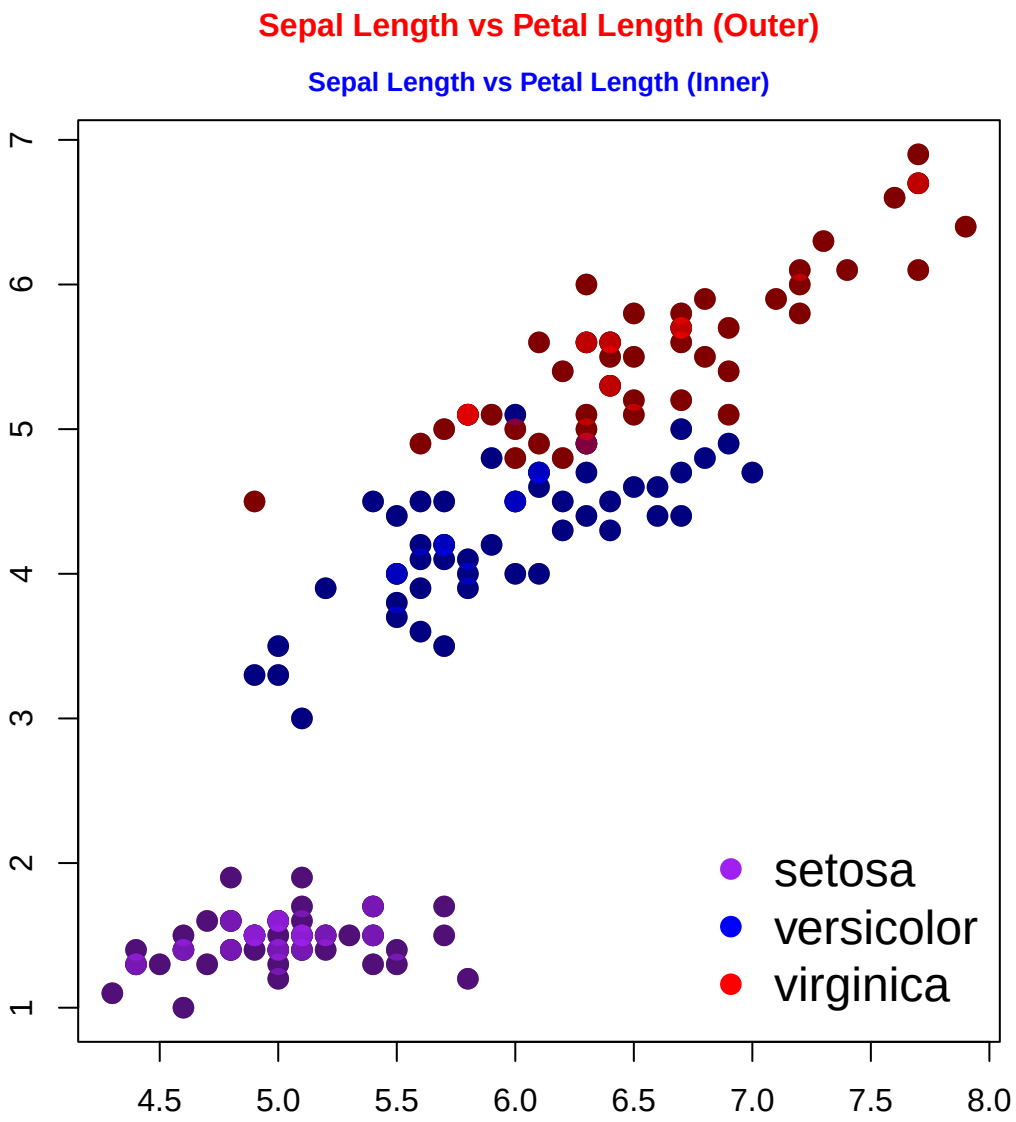
Almost all kinds of plots, charts, and graphs can be produced using base graphics, These plots can be fully customized using `par()` (graphical parameter). The following figure explains the graphical parameters we can use to customize the graphical layout.



Example 1: A typical graphic with the required graphical components: title, subtitle (if needed), axis labels, legends, etc.

```
url = "https://raw.githubusercontent.com/pengdsci/sta321/main/ww02/w02-iris.txt"
iris = read.table(url, header = TRUE)
par(mfrow = c(1,1),      # graphical layout: one row and one column
    oma=c(1, 1, 1, 1),   # outer margin area: oma=c(bottom, left, top, right)
    mar=c(2, 2, 2, 2))   # inner margin: mar=c(bottom, left, top, right)
# single plot
plot(iris$SepalLength, iris$PetalLength, pch = 16, cex = 1.5)
title(   main = "Sepal Length vs Petal Length (Outer)",
        outer = TRUE,       # place the title in outer margin
     col.main = "red",      # the color of the title
     cex.main = 1)          # font size. default = 1
title(   main = "Sepal Length vs Petal Length (Inner)",
        outer = FALSE,      # place a subtitle in the inner margin of the graphic
     col.main = "blue",     # the color of the subtitle
     cex.main = 0.8)        # font size of the subtitle
## Coloring points with the transparency level
## species ID
setosa = which(iris$Classification == "Iris-setosa")  # identify individual species
```

```r
versicolor = which(iris$Classification == "Iris-versicolor")
virginica = which(iris$Classification == "Iris-virginica")
## adding points
points(iris$SepalLength[setosa], iris$PetalLength[setosa],
       pch = 16,              # point type, ranged 1 - 21
       col = alpha("purple", 0.5),  # add a transparency level to the point
       cex = 1.5)             # point size
points(iris$SepalLength[versicolor], iris$PetalLength[versicolor],
       pch = 16,
       col = alpha("blue", 0.5),
       cex = 1.5)
points(iris$SepalLength[virginica], iris$PetalLength[virginica],
       pch = 16,
       col = alpha("red", 0.5),
       cex = 1.5)
legend("bottomright", c("setosa", "versicolor", "virginica"),
       col = c("purple", "blue", "red"),  #  order reflects the points
       pch = rep(16,3),  # corresponding point type. The same point type in this case
       cex = rep(1.5,3), # corresponding point size. The same font size in this case
       bty = "n")        # exclude the box of the legend
```



3.2 Simple multi-panel Plots

A simple **multi-panel plot** (also known as a **faceted plot** or **small multiples**) is a plot with multi sub-plots with equal dimension. We use base R plot functions `layout()` or `par(mfrow = c())` to achieve this task. We can add graphical parameters to create multi-panels with different dimensions.

From effective visual design perspective, a multi-panel plot should be used when

- **Compare Subgroups or Categories** - When you want to visualize how different subsets of data (e.g., groups, conditions, time periods) behave relative to each other.
- **Avoid Overcrowding in a Single Plot** - If too many lines, bars, or points in a single graph make it unreadable, splitting them into panels improves clarity.
- **Highlight Interactions or Conditional Relationships** - When the relationship between variables changes based on another factor (e.g., a categorical variable).
- **Display Multi-Dimensional Data** - When you have more than 3 variables and need a structured way to show interactions.

- **Standardize Comparisons** - Ensures all subplots share the same scale, making comparisons easier than with overlayed data.
- **Show Temporal or Spatial Patterns** - Useful for time series (e.g., each panel represents a year) or geospatial data (e.g., maps split by region).

However, if the subgroups are too numerous (e.g., 50+ panels), consider aggregation or sampling first and then make a single panel plot or multi-panel with no more than 6 sub-plots.

In multi-panel plots, titles play a crucial role in guiding interpretation.

- **Main Title** - The overarching title for the entire figure. It is placed outer margin.
- **Panel Titles (Subtitles)** - Labels for individual panels (often placed at the top or within each panel)
- **Row/Column Titles** - They are used when panels are rganized in a grid.
- **Axis Labels** - If axes represent the same information, we should use shared labels to increase the clarity of the visual representation.

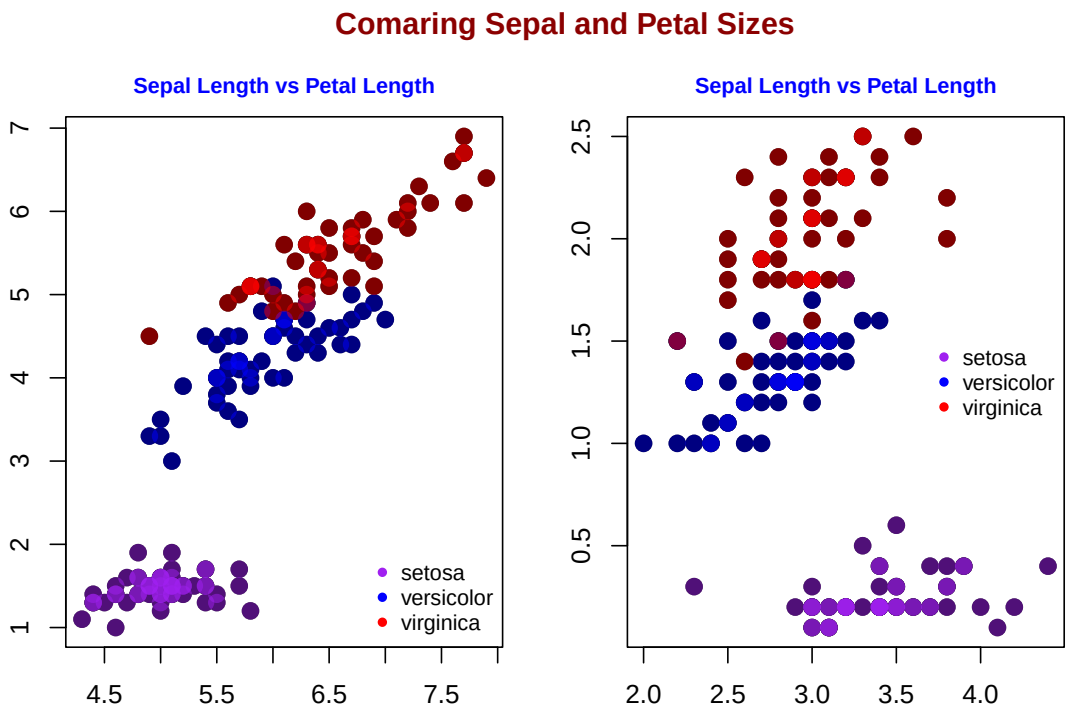
The next example shows how to create a simple multi-panel plots.

Example 2: This is a representative of simple multi-panel plots using the above design ideas.

```
url = "https://raw.githubusercontent.com/pengdsci/sta321/main/ww02/w02-iris.txt"
iris = read.table(url, header = TRUE)
## species ID
setosa = which(iris$Classification == "Iris-setosa")
versicolor = which(iris$Classification == "Iris-versicolor")
virginica = which(iris$Classification == "Iris-virginica")
###
## par() ==> parameters of the graphic
par(mfrow = c(1,2),       # create a graphical layout with on row and two columns
                          # mfrow() stands for "multi-frame row-wise"
    oma=c(2, 2, 2, 2),    # specify outer margin: oma = c(bottom, left, top, right)
                          # the unit is number of lines. default c(0,0,0,0), no outer margin
    mar=c(2, 2, 2, 2))

# first graphic panel
plot(iris$SepalLength, iris$PetalLength, pch = 16, cex = 1.5,
            xlab = "Sepal Width",
            ylab = "Petal Width")
title(main="Sepal Length vs Petal Length",
         outer = FALSE,    # subtitle of the panel on top of the panel
      col.main = "blue",
      cex.main = 0.8)
## adding points
points(iris$SepalLength[setosa], iris$PetalLength[setosa],
       pch = 16,
       col = alpha("purple", 0.5),
       cex = 1.5)
points(iris$SepalLength[versicolor], iris$PetalLength[versicolor],
       pch = 16,
       col = alpha("blue", 0.5),
       cex = 1.5)
points(iris$SepalLength[virginica], iris$PetalLength[virginica],
```

```
       pch = 16,
       col = alpha("red", 0.5),
       cex = 1.5)
legend("bottomright", c("setosa", "versicolor", "virginica"),
       col=c("purple", "blue", "red"),
       pch=rep(16,3), cex=0.8, bty="n")
##
plot(iris$SepalWidth, iris$PetalWidth, pch = 16, cex = 1.5,
            xlab = "Sepal Width",
            ylab = "Petal Width")
title(main="Sepal Length vs Petal Length",
      outer=FALSE,   # subtitle of the panel on top of the panel
      col.main = "blue",
      cex.main = 0.8)
## adding points
points(iris$SepalWidth[setosa], iris$PetalWidth[setosa],
       pch = 16,
       col = alpha("purple", 0.5),
       cex = 1.5)
points(iris$SepalWidth[versicolor], iris$PetalWidth[versicolor],
       pch = 16,
       col = alpha("blue", 0.5),
       cex = 1.5)
points(iris$SepalWidth[virginica], iris$PetalWidth[virginica],
       pch = 16,
       col = alpha("red", 0.5),
       cex = 1.5)
legend("right", c("setosa", "versicolor", "virginica"),
       col = c("purple", "blue", "red"),
       pch = rep(16,3), cex=0.8, bty="n")
## Overall Title
title( main = "Comaring Sepal and Petal Sizes",
      outer = TRUE,    # place the main title in the outer margin
      col.main = "darkred",
      cex.main = 1.2)
```



4 Making Your Own Color

You can make transparent colors using R and the `rgb() command`. These colors can be useful for charts and graphics with overlapping elements.

The `rgb() command` defines a new color using numerical values (0–255) for red, green and blue. In addition, we also set an alpha value (also 0–255), which sets the transparency (0 being fully transparent and 255 being "solid").

The following example takes the standard blue and makes it transparent (~50%):

```
mycol <- rgb(0, 0, 255, max = 255, alpha = 125, names = "blue50")
```

You can also use `col2rgb()` to check the composition of an existing named R color and modify it to define your own favorite colors.

Example 3: We check the RGB code from several named R colors in the following

```
col2rgb("darkred")
```

```
##       [,1]
## red    139
## green    0
## blue     0
```

```
col2rgb("skyblue")
```

```
##       [,1]
## red    135
## green  206
## blue   235
```

```
col2rgb("gold")
```

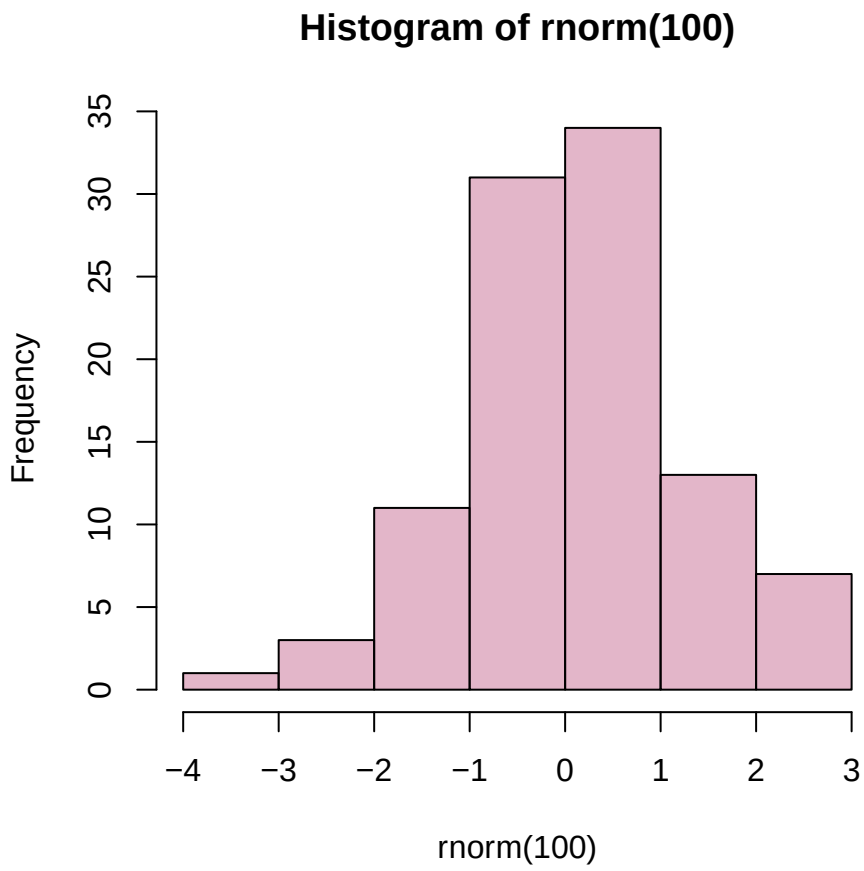
```
##       [,1]
## red    255
## green  215
## blue     0
```

```
col2rgb("purple")
```

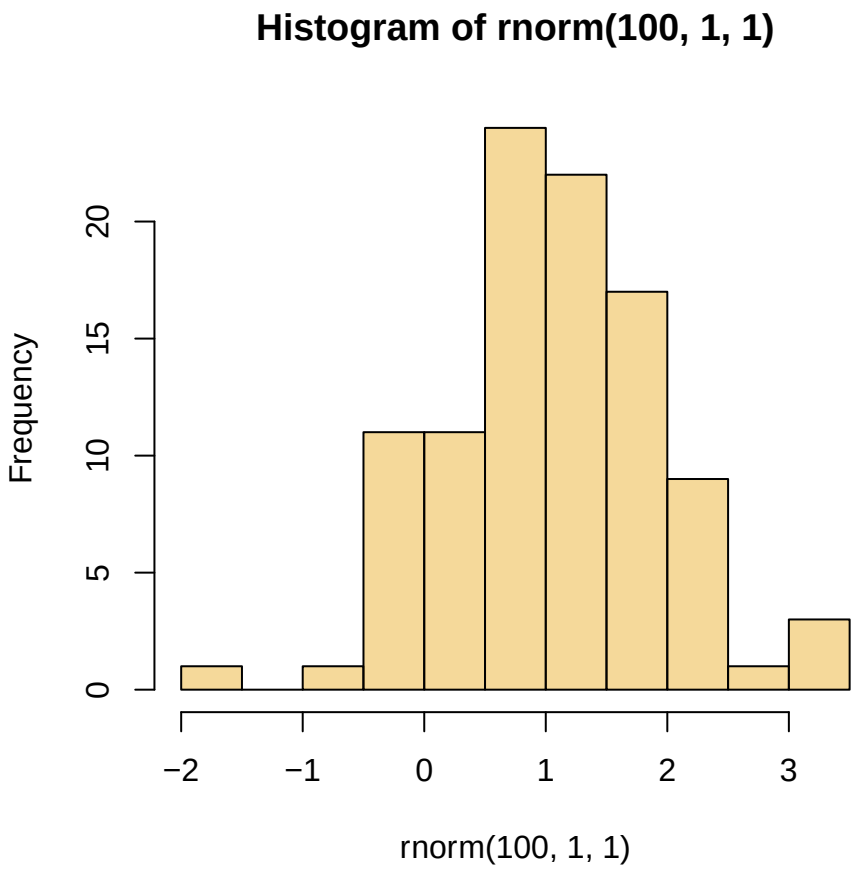
```
##       [,1]
## red    160
## green   32
## blue   240
```

Example 3 We define a color by modifying "gold" and "purple". I take the average of the RGB codes of "gold" and "purple" to see what the color looks like.

```
ratio = 0.4
avggoldpurple = round(ratio*col2rgb("gold") + (1-ratio)*col2rgb("purple"))
##
mycol.1 = rgb(avggoldpurple[1,1], avggoldpurple[2,1], avggoldpurple[3,1],
              max = 255,
              alpha = 125,
              names = "goldpuple50")
###
hist(rnorm(100),  col = mycol.1)
```




```
ratio = 0.8
avggoldpurple = round(ratio*col2rgb("gold") + (1-ratio)*col2rgb("purple"))
##
mycol.2 = rgb(avggoldpurple[1,1], avggoldpurple[2,1], avggoldpurple[3,1], max = 255, alpha = 125, names
###
hist(rnorm(100,1,1),  col = mycol.2)
```


Histogram of rnorm(100, 1, 1)
Frequency
0
5
10
15
20
−2
−1
0
1
2
3
rnorm(100, 1, 1)