Graphics With Base R

Sean Hellingman ©

Data Visualization and Manipulation through Scripting (ADSC1010)

shellingman@tru.ca

Fall 2024

THOMPSON RIVERS UNIVERSITY

Topics

Introduction

- Data visualization allows you to reveal patterns in your data and communicate insights.
- Base R comes with built-in graphics functionality.
- Often we use high-level functions such as `plot()` to create a plot and then add information to the plot using functions like `lines()` or `text()`.
- We can also use the *lattice* package (comes with base R but we initialize it using `library(lattice)`).
  - We can use the functionality in the *lattice* package to generate visualizations in one go.

Getting Started

- If you generate a visualization in the Console or using a script, the plot will show up in the *Plots* tab.
- In the *Plots* tab and in R Markdown, you can zoom in on your plots (using *Zoom* or *Show in New Window*).
- Visualizations generated in the Console or using a script can be saved using *Export*.
- Visualizations generated in R Markdown will be rendered in the final document unless you change the code chunk arguments to prevent this.

One Variable

- One of the most commonly used R functions for generating visualizations is the `plot()` function.
- In order to generate a scatter plot of a single variable we use `plot(data.name$variable.name)`.
- The y-axis will contain the variable values and the x-axis will contain the index of the variable (order in the data).
- Many base R visualization functions do not have a `data=` argument.
    - Instead, we can use `with(data.name,plot(variable.name)`

Two Variables

- To generate a scatterplot of two numeric variables we just need to add the `x` and `y` variables as arguments.
- We can use:
  - `plot(x = data.name$variablex.name, y = data.name$variabley.name`
  - `plot(data.name$variabley.name ~ data.name$variablex.name)` (common regression notation)
- *Pick which approach works best for you an stick with it.*

Graph Type

- You can specify different plot types using the `type=` argument
- Usage:
  - `type = "p"` plots just the points (default).
  - `type = "l"` plots just lines.
  - `type = "b"` plots both points and lines connected.
  - `type = "o"` both points and lines with the lines running through the points.
  - `type = "c"` empty points joined by lines.
- So far, the plots might seem boring, but the `plot()` function can be used in many different scenarios.

Example 1

- Import the *Football22.csv* file into R.

  1. Create a scatter plot of the *Points* variable.
  2. Create a scatter plot with `y` as *Points* and `x` as *Goal_Differential*.
  3. Use par(mfrow = c(2, 2)) and the `type=` argument to generate four different plots with `y` as *Points* and `x` as*Goal_Differential*.

- Do you notice any patterns?

Histograms

- Histograms are useful to gain information about the distribution of values in a numeric variable.

- In order to generate a histogram we use `hist(data.name$variable.name)`.

- The `y`-axis will contain the frequency of observations values and the `x`-axis will contain the values of the variable.

- The `hist()` uses the *Sturges* formula to generate the default number of breakpoints (bins).

- We may change the breakpoints (bins) and the title using the following arguments:

  - `breaks = seq(from, to, by)`
  - `main = "Your Title"`

Density

- Histograms can also be displayed as a proportion (density) using the `freq = FALSE` argument.
- We may also add a *kernel density* using the `density()` and `lines()` functions.

```
hist(data.name$variable.name),freq = FALSE)
lines(density(data.name$variable.name))
```

Example 2

- Using the *Football22.csv* data in R:

  1. Generate a histogram of the *Goals_For* variable. *Be sure to update the title and labels as needed*.
  2. Generate a density (proportional) histogram and be sure to include the kernel density and appropriate labels of the *Goals_For* variable.

- Do you notice anything about the shape?

Boxplots

- Boxplots are extremely useful for graphically summarizing data.
- Boxplots give us information about the center, the spread, or variability within the data, any departure from symmetry (skew), and identification of possible outliers.
- Usage:
  - `boxplot(data.name$variable.name,ylab = "variable")`

Comparative Boxplots I

- We can use boxplots to examine how the distribution of a variable changes for different groups.
- This can be done by simply adding the group variable into the function:
  - `boxplot(variable.name ~ group.name, data = data.name, ylab = "variable", xlab = "group name")`
    OR
  - `boxplot(data.name$variable.name ~ data.name$group.name, ylab = "variable", xlab = "group name")`

Comparative Boxplots II

- We can use boxplots to examine how the distribution of a variable changes for **two** different groups.

- This can be done by simply multiplying the second group variable to the first group variable into the function:
  - `boxplot(variable.name ~ group1.name*group2.name, data = data.name, ylab = "variable", xlab = "group names")`
    OR
  - `boxplot(data.name$variable.name ~ data.name$group1.name*data.name$group2.name, ylab = "variable", xlab = "group names")`

Example 3

- Using the *Football22.csv* data complete the following:

    1. Create a boxplot of the *Goals_For* variable for all of the clubs.
    2. Create a comparative boxplot of the *Goals_For* variable for all of the clubs in the three different leagues.
    3. Create a variable that indicates if the club was in the upper-half or lower-half of the league table in each league based on the *Position* variable (20 teams in each league).
    4. Create a comparative boxplot of the *Goals_For* variable for all of the clubs in the three different leagues and if they were in the upper or lower half of their respective league.

- Do you notice any differences?

Violin Plots

- Violin plots are like a combination of a boxplot and a kernal density plot.
- We will need the *vioplot* package to create violin plots.

  ```
  if(!require(vioplot)) install.packages("vioplot")
  library(vioplot)
  ```

- Usage:
  - `vioplot((variable.name ~ group.name, data = data.name, ylab = "variable", xlab = "group names")`
  - Can change the colour using `col =`

Example 4

- Repeat Example 3 using violin plots.
- Do you notice anything different than you did in Example 3?

Dot Charts

- Identifying unusual observations (potential) outliers is an important step in any statistical modelling problems.
- The *Cleaveland* dotplot can help with outlier detection.
- To use on a single variable we can use:
`dotchart(data.name$variable.name)`
- To include a grouping variable we can use:
`dotchart(data.name$variable.name, groups = data.name$group.name)`

Example 5

- Using the *iris* data complete the following:

  1. Create dot charts for the *Petal.Width* and for the *Petal.Length* variables.
  2. Create dot charts grouped by *Species* for the *Petal.Width* and for the *Petal.Length* variables.

- Do you notice any potential outliers?

Pairs Plots

- Instead of individually creating scatter plots, we can use the `pairs()` function to create multiple scatter plots.
- This function creates a multi-panel scatter plot with all possible combinations of variables.
- We can use: `pairs(data.name[, c("var1.name", "var2.name", ...)])`
- Or we can use column indexes: `pairs(data.name[,2:5])`
- The panels on the diagonal give the **variable names of each row**.

Pairs Plots Arguments

- `panel = panel.smooth` adds a LOWESS curve to each panel.
- We can create functions that do the following:
  - `lower.panel = panel.cor` adds the correlation coefficients of each pair of variables below the diagonal.
  - `diag.panel = panel.hist` adds a histogram of each variable to the diagonal.
- *It can take some time to get used to the interpretations*

Example 6

- Using the *Football22.csv* data complete the following:
  1. Create the functions under the *Required Functions* heading.
  2. Create a pairs plot of all of the numeric variables.
  3. Create a pairs plot of all the numeric variables and include a LOWESS curve, the correlation coefficients, and the histograms.
- Does there appear to be any linear relationships between the variables?

Conditional Scatterplots I

- When examining the relationship between two numeric variables, it may be useful to determine if a third variable is obscuring or changing the possible relationships.
- Conditional Scatterplots (coplots) can be useful to plot these situations.
- The `coplot()` R function plots two variables but each plot is conditioned by a third variable (numeric or factor).
- Usage:
  - `coplot(variabley.name ~ variablex.name|variable3.name, data = data.name)`

Conditional Scatterplots II

- *The results may be a bit tricky to interpret at first.*

- The bar plot at the top indicates the range of the third variable (for the plots).
  - By default, there will be some overlap in the scatter plots based on the third variable. (can change by using: `overlap = 0`)

- Alterations:
  - To add a forth variable to the plot:
    `coplot(variabley.name ~ variablex.name|variable3.name*variable4.name, data = data.name)`

  - To add blue linear regression lines to the plots:
    `coplot(variabley.name ~ variablex.name|variable3.name, data = data.name, panel = function(x, y, ...){ points(x, y, ...)  abline(lm(y ~ x), col = "blue")})`

Example 7

- Using the *Football22.csv* data complete the following:
    1. Create a coplot with `y` the number of *Points*, `x` the *Goal_Differential* and group them by the *League*. *Be sure to include the linear regression lines*

- Does there appear to be any differences across the leagues?

Lattice Plots I

- We can also use the *lattice* R package to make the same plots.
- This package is particularly useful when we want to plot in multiple panels.
- We can use the | operator to make this happen.
  - We can *multiply* (∗) variables to examine multiple interactions.

Lattice Plots II

Graph type	lattice function	Base R function
scatterplot	`xyplot()`	`plot()`
frequency histogram	`histogram(type="count")`	`hist()`
boxplot	`bwplot()`	`boxplot()`
Cleveland dotplot	`dotplot()`	`dotchart()`
scatterplot matrix	`splom()`	`pairs()`
conditioning plot	`xyplot(y ~ x | z)`	`coplot()`

Example 8

- Use the the *Football22.csv* data and the *lattice* R package to complete the following:
  1. Generate histograms of the *Points* variable grouped by the *League* variable.
  2. Generate a conditioning plot with `y` as the *Points* variable and `x` as the *Goal_Differential* grouped by the *League* **and** the *Half* variable we created earlier.

Customising Plots

- We can use different arguments to improve the looks and functionality of our basic plots.
    - We can add titles and axis labels.
    - We can change the margins and plot limits.
    - We can change the size of our text and the shape, colour, and size of our *dots*
    - We can also add text labels to our base R plots.

Titles, Labels and Margins

- `main` = *Plot title*
- `xlab` = *x-axis label*
- `ylab` = *y-axis label*

- If we want to include superscripts or other mathematical expressions we can use the `expression()` function:

  Example: `ylab = expression(paste("Area (cm"^"2",")"))`

- To adjust the margins (sizes of the margins):

  `par(mar = c(bottom, left, top, right)`

Axes Arguments

- `xlim = c(min, max)` *Adjusts the range of the x-axis*
- `ylim = c(min, max)` *Adjusts the range of the y-axis*

- If we want to remove the box around the outside of our plot we can use the `bty = "l"` argument.

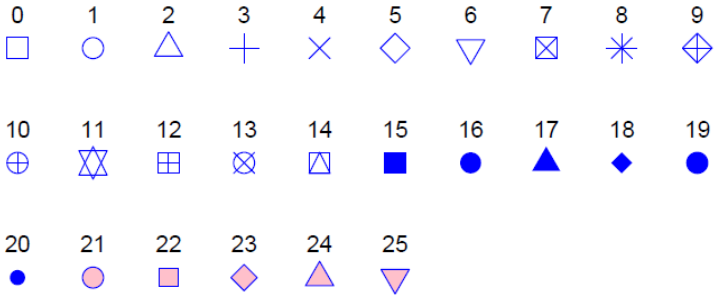
- `par(mar = c(bottom, left, top, right), xaxs = "i", yaxs = "i")` adjusts the axes so they meet at the origin (0,0).
- The `las = 1` argument rotates the y-axis tick marks and changes the orientation of the text.
- `cex.axis = multiplier` sets the amount by which the text will be magnified (relative to 1).
- `tcl = -0.2` shortens the tick marks (positive increases them).

Plotting Symbol Adjustments and Text

- `pch` = changes the type of plotting symbol (see next slide)
- `col` = changes the colour of the plotting symbol (see link in references for colours).
- `cex` = changes the size of the plotting symbol (multiplyer).
- We can add text to our plots:

  `text(x = x.location, y = y.location, label = "Your Label", cex =` *`text size`*`, col =` *`label colour`*`)`

Symbol Shapes (`pch =`)



Plot Arguments I

Argument	Description
`adj`	controls justification of the text (0 left justified, 0.5 centered, 1 right justified)
`bg`	specifies the background colour of the plot(i.e. : `bg = "red"`, `bg = "blue"`)
`bty`	controls the type of box drawn around the plot, values include: `"o"`, `"l"`, `"7"`, `"c"`, `"u"` , `"]"` (the box looks like the corresponding character); if `bty = "n"` the box is not drawn
`cex`	controls the size of text and symbols in the plotting area with respect to the default value of 1. Similar commands include: `cex.axis` controls the numbers on the axes, `cex.lab` numbers on the axis labels, `cex.main` the title and `cex.sub` the sub-title
`col`	controls the colour of symbols; additional argument include: `col.axis`, `col.lab`, `col.main`, `col.sub`
`font`	an integer controlling the style of text (1: normal, 2: bold, 3: italics, 4: bold italics); other argument include `font.axis`, `font.lab`, `font.main`, `font.sub`
`las`	an integer which controls the orientation of the axis labels (0: parallel to the axes, 1: horizontal, 2: perpendicular to the axes, 3: vertical)
`lty`	controls the line style, can be an integer (1: solid, 2: dashed, 3: dotted, 4: dotdash, 5: longdash, 6: twodash)

Plot Arguments II

Argument	Description
`lwd`	a numeric which controls the width of lines. Works as per `cex`
`pch`	controls the type of symbol, either an integer between 0 and 25, or any single character within quotes `" "`
`ps`	an integer which controls the size in points of texts and symbols
`pty`	a character which specifies the type of the plotting region, "s": square, "m": maximal
`tck`	a value which specifies the length of tick marks on the axes as a fraction of the width or height of the plot; if `tck = 1` a grid is drawn
`tcl`	a value which specifies the length of tick marks on the axes as a fraction of the height of a line of text (by default `tcl = -0.5`)

Example 9

- Using the *Football22.csv* data complete the following:
  1. Create a scatterplot in base R with `y` as the *Points* variable and `x` as the *Goals_For* variable.
  2. Take some time to make improvements to your scatterplot using the plot customisation tools.
- Do you think the resulting plot is useful?

Exercise 1

- The best way to improve your understanding of the plotting functions in base R is to take some time to generate some plots, make customisations, and think about what your plots actually mean.
- I encourage you to take some time with your project data (or other real data) and try to generate some insightful plots using the base R functions we covered.

References & Resources

1. Douglas, A., Roos, D., Mancini, F., Couto, A., & Lusseau, D. (2023). *An introduction to R*. Retrieved from https://intro2r.com/

- https://r-charts.com/colors/
- https://lattice.r-forge.r-project.org/
- https://cran.r-project.org/web/packages/vioplot/vioplot.pdf