Base Graphics #1
(Functions and Parameters)

Michael Malick

Updated: 15 May 2013

Graphics Packages

There are three main graphics packages for R:

- `graphics` (base graphics)
  - Don't need to load
- `lattice` (grid graphics)
  - Need to load `library(lattice)`
- `ggplot2` (grid graphics)
  - Need to install and load

Graphics Demos

```
demo(graphics)
```

Graphics Commands

Plotting commands in R are separated into three basic groups:

- **High-level commands**: plotting functions that produce a new graphical device with a plot. High-level commands generally produce a full graphic with axes, labels, and titles
- **Low-level commands**: plotting functions that add elements to graphics such as lines, points, legends
- **Interactive commands**: graphical functions that allow you to interact with the graphic (e.g., using the mouse)

Graphics Commands

Plotting commands in R are separated into three basic groups:

- **High-level commands**: plotting functions that produce a new graphical device with a plot. High-level commands generally produce a full graphic with axes, labels, and titles
- **Low-level commands**: plotting functions that add elements to graphics such as lines, points, legends
- **Interactive commands**: graphical functions that allow you to interact with the graphic (e.g., using the mouse)

The `plot()` Function

`plot()`

The `plot()` function is the primary high level command for producing graphics in R

```
x <- 1:100
y <- rnorm(100)

plot(x, y)
plot(y ~ x)
```

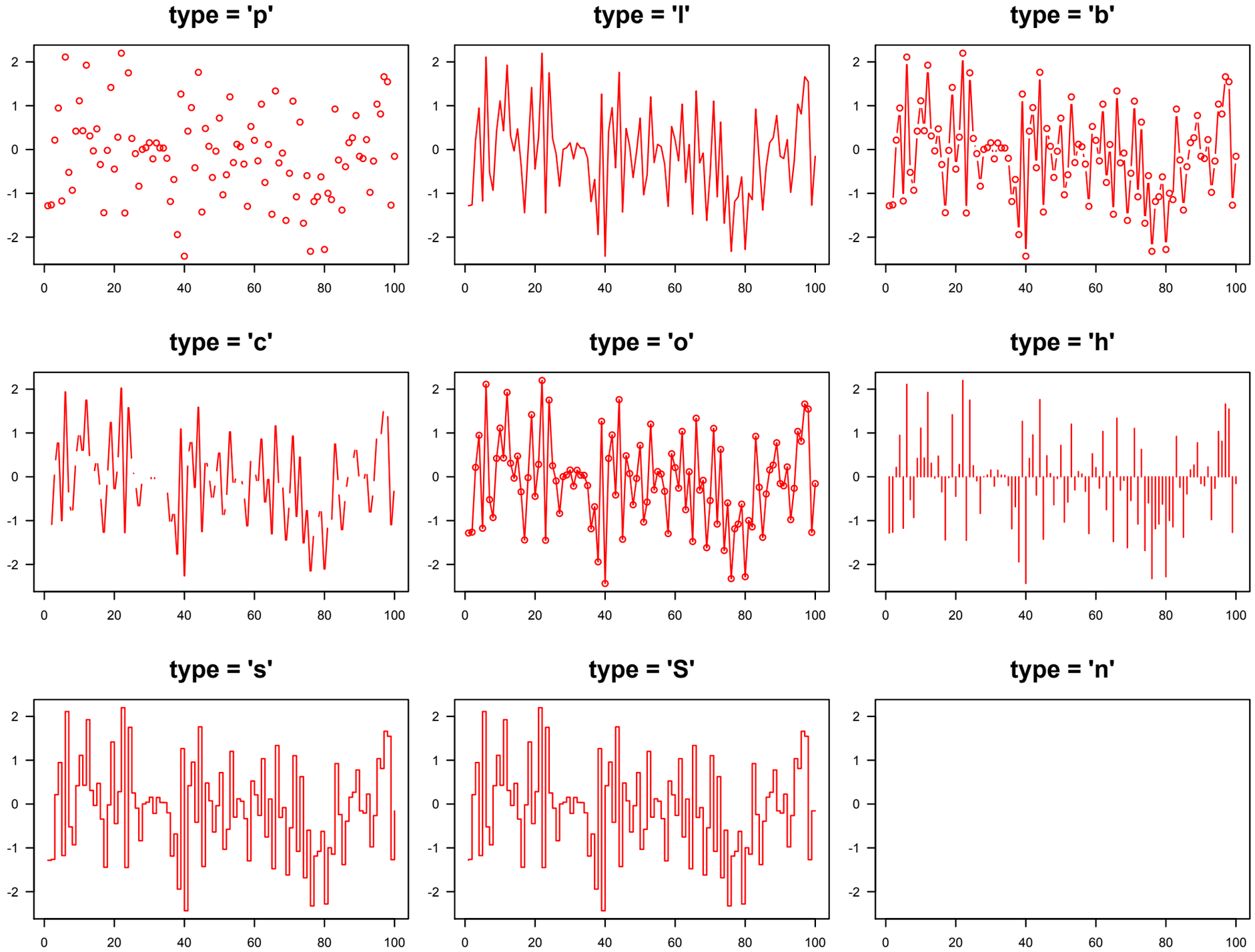
`plot()` is a Generic Function

```
# Dataframe (scatterplot matrix)
plot(iris)

# Single numeric variable (scatterplot)
plot(iris$Sepal.Length)

# Single factor variable (barplot)
plot(iris$Species)

# Two numeric variables (scatterplot)
plot(iris$Sepal.Length, iris$Petal.Length)

# Numeric and factor variable (dotplot)
plot(iris$Sepal.Length, iris$Species)

# Factor and numeric variable (boxplot)
plot(iris$Species, iris$Sepal.Length)
```

plot() Arguments

```
plot(x, y,
  type = "p",
main = "Main-title",
sub = "Sub-title",
xlab = "X-lab",
ylab = "Y-lab",
asp = 0)
```

plot(x, y, type = ?)



xlim and ylim

You can limit the scales of the x and y axis using `xlim` and `ylim`

```
x <- rnorm(100)
y <- rnorm(100)

plot(x, y)

plot(x, y, xlim = c(-10, 10))

plot(x, y, xlim = c(-10, 10),
     ylim = c(-10, 10))
```

Graphics Functions

Base Graphics	Description	Lattice
`barplot`	Bargraph	`barchart`
`biplot`	Represents rows and columns of a matrix	
`boxplot`	Box and whisker plot	`bwplot`
`contour`	Contour plot	`contourplot`
`coplot`	Conditional plot	`xyplot`
`dotchart`	Cleveland dot chart	`dotplot`
`filled.contour`	Contour plot filled with colors	`contourplot`
`hist`	Histogram	`histogram`
`image`	High-density image plot	`levelplot`
`interaction.plot`	Interactions plot for two-factors	
`matplot`	Multiple plots specified by columns of a matrix	
`pairs`	All pairwise plots between variables	`splom`
`persp`	3D perspective plot	`wireframe, cloud`
`pie`	Pie chart	
`plot`	Plot	`xyplot`
`qqnorm`	Normal QQ plot	`qqmath`
`qqplot`	Quantile-Quantile plot	`qq`
`scatter.smooth`	Scatterplot with a smooth curve	
`stars`	Star plots for multivariate data	

You Try...

1. Using the `mtcars` dataset plot `mpg` vs. `cyl`
2. Create a histogram of the `mpg` variable in the `mtcars` dataset
3. Create boxplots for each variable in the `mtcars` dataset

Graphics Devices

Graphics Devices

All graphics have to be printed on a graphics device

Any call to a high level graphics command automatically opens a new graphics device if one is not already open

```
dev.new()
plot(rnorm(100))

dev.new(height = 8, width = 8)
    plot(runif(100))
dev.off()

graphics.off()
```

Print to a File

R can easily print graphics to different file types (devices)

- `pdf()`
- `jpeg()`
- `png()`
- `tiff()`
- `bmp()`

Print to a PDF File

```
pdf(file = "graph.pdf")
   plot(x, y, main = "Graph 1")
   plot(y, x, main = "Graph 2")
dev.off()


pdf(file = "graph.pdf", height = 4, width = 4)
   plot(x, y)
dev.off()
```

Print to a JPEG File

```
# Only one graph per file
jpeg(file = "graph.jpg")
   plot(x, y)
dev.off()
```

```
jpeg(file = "graph.jpg", width = 900,
    height = 900)
   plot(x, y)
dev.off()
```

You Try...

1. Using the `mtcars` dataset plot `mpg` vs. `cyl`
2. Save the plot you just created as a pdf file

Graphical Parameters

Graphics Parameters

Graphics parameters can be set to change almost any part of a graphic (over 70 parameters can be set)

- background
- labels
- colors
- margins of the plot
- font type
- font size
- orientation
- plotting characters
- aspect ratios
- tick marks
- axis label rotation
- number of plots
- line type
- line width
- plot dimensions
- box type ...

Temporary Changes to Parameters

Most graphical parameters can be set within the plot call (the exception being parameters that deal with the plot layout)

```
plot(x, y)

plot(x, y, col = 2, las = 1, pch = 19)

plot(x, y, col = 4, las = 1, pch = 12)
```

- Setting graphical parameters within the plot call only changes the parameters for that particular graph

Lasting Changes to Parameters

The `par()` function is used to set graphic parameters

- Setting graphical parameters with `par()` changes the parameters for the device (until the device is closed)

```
graphics.off()
par(pch = 19, las = 1)

plot(x, y, col = 4)

plot(x, y, col = 6)
```

Example: Multiple Plots on a Device

```
par(mfrow = c(2, 2), mar = c(3, 3, 3, 2))

plot(iris$Sepal.Length, col = 2, main = "Sepal Length",
     ylab = "Sepal length")

plot(iris$Petal.Length, col = 3, main = "Petal Length",
     ylab = "Petal length")

plot(iris$Sepal.Width, col = 5, main = "Sepal Width",
     ylab = "Sepal width")

plot(iris$Petal.Width, col = 6, main = "Petal Width",
     ylab = "Petal width")

dev.off()
```

Example: Colored Plot

```
par(bg = "black",              # background color of device
    fg = "white",              # color of axis lines and ticks
    bty = "l",                  # box type around plot
    las = 1,                    # axis labels are horizontal
    col.axis = "white",        # color of axis
    col.lab = "white",         # color of labels
    col.main = "white",        # color of title
    mar = c(4, 4, 2, 1),       # margins of the plot
    tck = -0.03,               # length of tick marks
    font.main = 2,             # bold font for title
    font.lab = 2,              # bold font for axis labels
    family = "sans",           # use sans-serif font
    cex.lab = 1.2,             # make axis labels larger
    cex.axis = 1.1,            # make axis annotations larger
    pch = 19)                  # change plotting character


plot(iris$Sepal.Length, col = 6, main = "Sepal Length",
     ylab = "Sepal length")
```

You Try...

1. Using the `mtcars` dataset plot `mpg` vs. `cyl`
2. Save the plot you just created as a pdf file
3. Change the color, plotting character, axis labels, and create a title for the plot

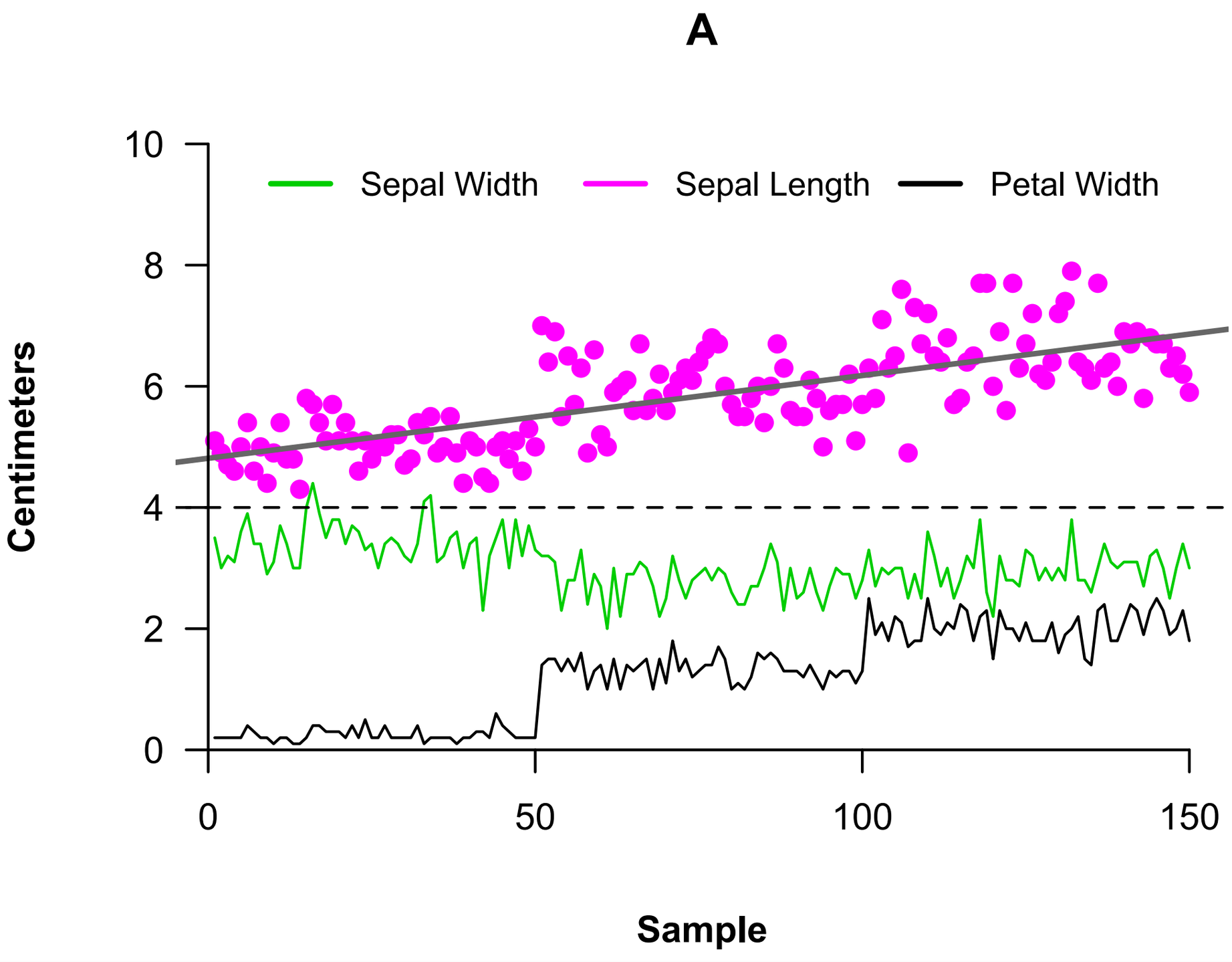
Base Graphics #2
(Lines, Points, Legends, Axis)

Graphics Commands

Plotting commands in R are separated into three basic groups:

- **High-level commands**: plotting functions that produce a new graphical device with a plot. High-level commands generally produce a full graphic with axes, labels, and titles
- **Low-level commands**: plotting functions that add elements to graphics such as lines, points, legends
- **Interactive commands**: graphical functions that allow you to interact with the graphic (e.g., using the mouse)

Graphic to Create ...



Functions We Need

- `plot()`
- `par()`
- `points()`
- `lines()`
- `abline()`
- `axis()`
- `legend()`

Step #1: Call `plot()`

Need to call a high-level plotting command first (`plot()`)

```
attach(iris)

dev.new(width = 5, height = 4)
par(pch = 19, las = 1, font.lab = 2)

plot(Sepal.Width, type = "l", ylim = c(0, 10), col = 3,
     ylab = "Centimeters", axes = FALSE, xlab =
     "Sample", main = "A")
```

Step #2: Add `axis()`

The `axis()` function adds axes to a plot

```
dev.new(width = 5, height = 4)
par(pch = 19, las = 1, font.lab = 2)

plot(Sepal.Width, type = "l", ylim = c(0, 10), col = 3,
     ylab = "Centimeters", axes = FALSE, xlab =
     "Sample", main = "A")

axis(1, pos = 0) # x-axis
axis(2, pos = 0) # y-axis
```

Step #3: Add `lines()`

The `lines()` function adds lines to an existing plot

```
dev.new(width = 5, height = 4)
par(pch = 19, las = 1, font.lab = 2)

plot(Sepal.Width, type = "l", ylim = c(0, 10), col = 3,
     ylab = "Centimeters", axes = FALSE, xlab =
     "Sample", main = "A")

axis(1, pos = 0) # x-axis
axis(2, pos = 0) # y-axis

lines(Petal.Width)
```

Step #4: Add `points()`

The `points()` function adds points to an existing plot

```
dev.new(width = 5, height = 4)
par(pch = 19, las = 1, font.lab = 2)

plot(Sepal.Width, type = "l", ylim = c(0, 10), col = 3,
     ylab = "Centimeters", axes = FALSE, xlab =
     "Sample", main = "A")

axis(1, pos = 0) # x-axis
axis(2, pos = 0) # y-axis

lines(Petal.Width)
points(Sepal.Length, col = 6)
```

Step #5: Add Reference Line

The `abline()` function adds vertical or horizontal reference lines

```
dev.new(width = 5, height = 4)
par(pch = 19, las = 1, font.lab = 2)

plot(Sepal.Width, type = "l", ylim = c(0, 10), col = 3,
     ylab = "Centimeters", axes = FALSE, xlab =
     "Sample", main = "A")

axis(1, pos = 0) # x-axis
axis(2, pos = 0) # y-axis

lines(Petal.Width)
points(Sepal.Length, col = 6)

abline(h = 4, lty = 2)
```

Step #6: Add Regression Line

The `abline()` function can also add regression lines to an existing plot

```
dev.new(width = 5, height = 4)
par(pch = 19, las = 1, font.lab = 2)

plot(Sepal.Width, type = "l", ylim = c(0, 10), col = 3,
     ylab = "Centimeters", axes = FALSE, xlab = "Sample",
     main = "A")

axis(1, pos = 0) # x-axis
axis(2, pos = 0) # y-axis

lines(Petal.Width)
points(Sepal.Length, col = 6)

abline(h = 4, lty = 2)

x <- 1:150
abline(lm(Sepal.Length ~ x), lwd = 2, col = "grey40")
```

Step #7: Add `legend()`

The `legend()` function creates a legend for the plot

```
dev.new(width = 5, height = 4)
par(pch = 19, las = 1, font.lab = 2)

plot(Sepal.Width, type = "l", ylim = c(0, 10), col = 3,
      ylab = "Centimeters", axes = FALSE, xlab = "Sample", main = "A")

axis(1, pos = 0)              # x-axis
axis(2, pos = 0, las = 1) # y-axis

lines(Petal.Width)
points(Sepal.Length, col = 6)

abline(h = 4, lty = 2)

x <- 1:150
abline(lm(Sepal.Length ~ x), lwd = 2, col = "grey40")

legend(x = 5, y = 10, legend = c("Sepal Width", "Sepal Length",
    "Petal Width"), col = c(3, 6, 1), lwd = 2, bty = "n",
    horiz = TRUE, cex = 0.9)
```

Entire Code

```
attach(iris)
dev.new(width = 5, height = 4)
par(pch = 19, las = 1, font.lab = 2)

plot(Sepal.Width, type = "l", ylim = c(0, 10), col = 3,
    ylab = "Centimeters", axes = FALSE, xlab = "Sample", main = "A")

axis(1, pos = 0) # x-axis
axis(2, pos = 0) # y-axis

lines(Petal.Width)
points(Sepal.Length, col = 6)

abline(h = 4, lty = 2)

x <- 1:150
abline(lm(Sepal.Length ~ x), lwd = 2, col = "grey40")

legend(x = 5, y = 10, legend = c("Sepal Width", "Sepal Length",
    "Petal Width"), col = c(3, 6, 1), lwd = 2, bty = "n",
    horiz = TRUE, cex = 0.9)
```

You Try...

1. Using the `mtcars` dataset plot `mpg` vs. `cyl`
2. Save the plot you just created as a pdf file
3. Change the color, plotting character, axis labels, and create a title for the plot
4. Add to the plot the `qsec` vs. `cyl` using a different color
   - Hint: use the `points()` function
5. Add a legend to the plot to differentiate between `mpg` and `qsec`