Section 13

R: Graphics

张勇

yzhang@tongji.edu.cn

同济大学

2021年6月7日

Creating a Scatter Plot

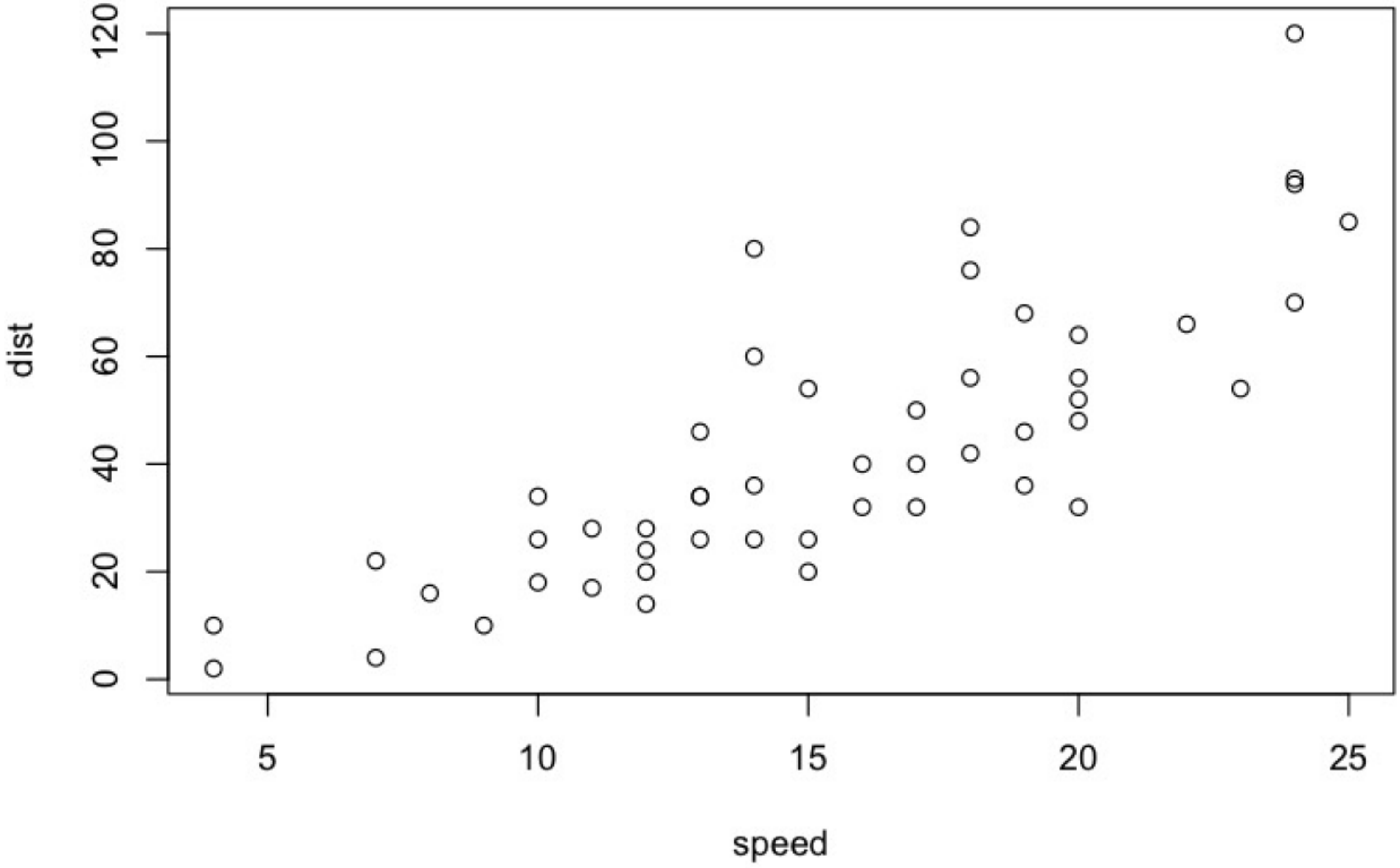
- **Problem**
  - You have paired observations: (x_1, y_1), (x_2, y_2), ..., (x_n, y_n). You want to create a scatter plot of the pairs
- **Solution**
  - If your data are held in two parallel vectors, x and y, then use them as arguments of **plot**:

    ```
    > plot(x, y)
    ```
  - If your data is held in a (two-column) data frame, plot the data frame:

    ```
    > plot(dfrm)
    ```

Creating a Scatter Plot

```
> plot(cars)
```

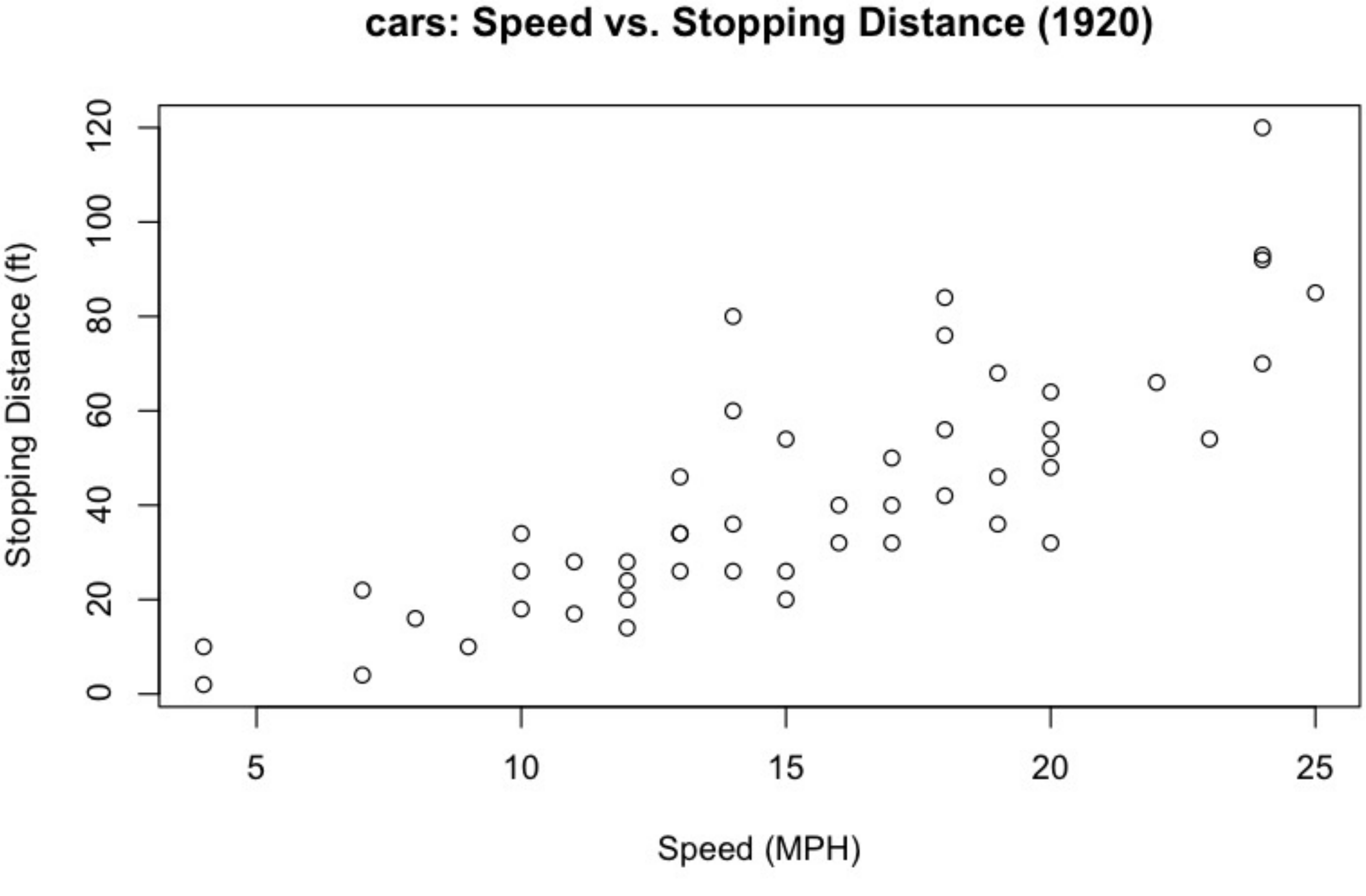


Adding a Title and Labels

- **Problem**
  - You want to add a title to your plot or add labels for the axes.
- **Solution**
  - When calling **plot**:
    - Use the **main** argument for a title;
    - Use the **xlab** argument for an x-axis label;
    - Use the **ylab** argument for a y-axis label.

Adding a Title and Labels

```
> plot(cars, main="cars: Speed vs. Stopping Distance (1920)",
+     xlab="Speed (MPH)", ylab="Stopping Distance (ft)")
```



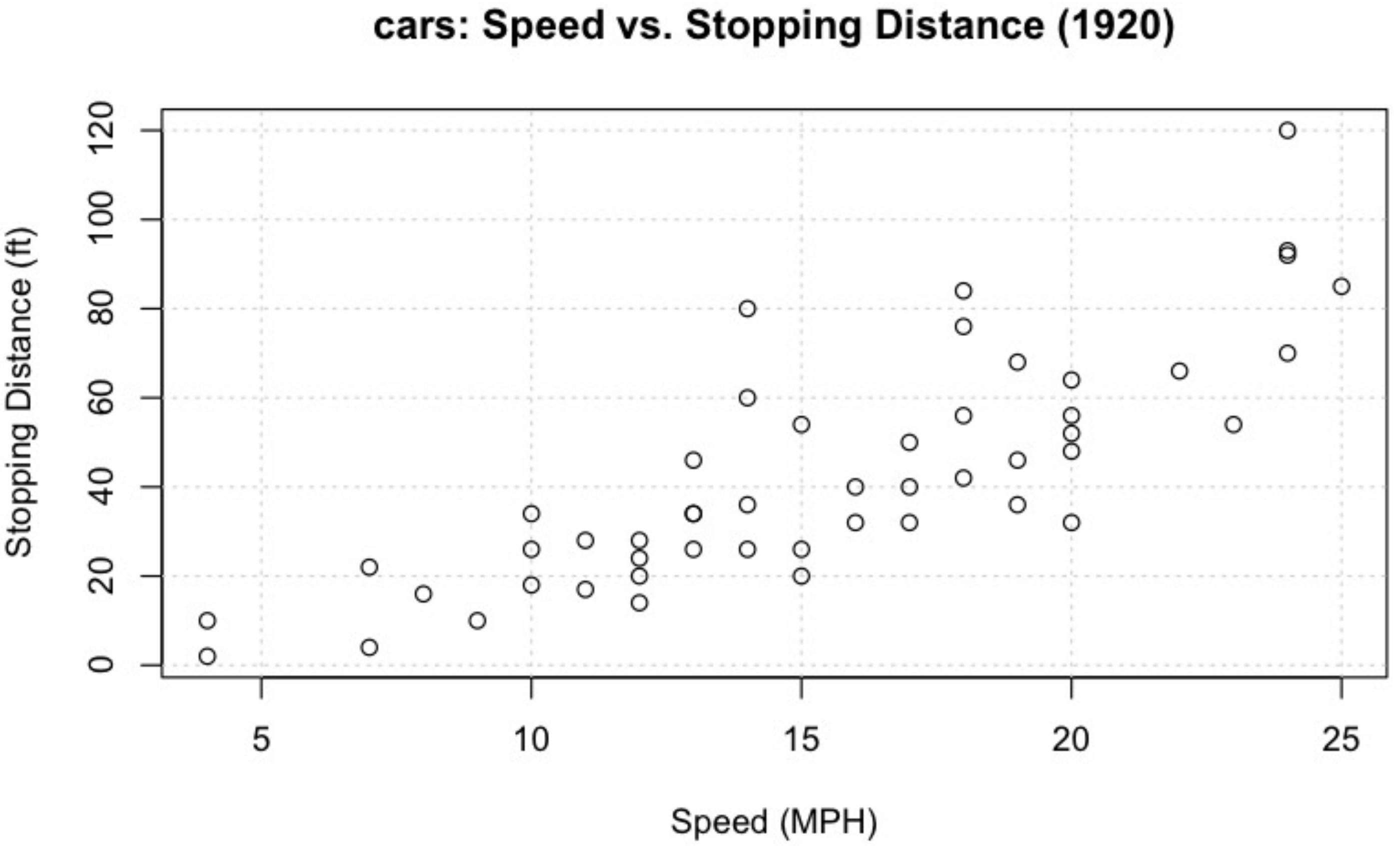
Adding a Grid

- **Problem**
  - You want to add a grid to your graphic.

- **Solution**
  - Call **plot** with **type**="n" to initialize the graphics frame without displaying the data.
  - Call the **grid** function to draw the grid.
  - Call low-level graphics functions, such as **points** and **lines**, to draw the graphics overlaid on the grid.

Adding a Grid

```
> plot(cars, main="cars: Speed vs. Stopping Distance (1920)",
+ xlab="Speed (MPH)", ylab="Stopping Distance (ft)", type="n")
> grid()
> points(cars)
```



Creating a Scatter Plot of Multiple Groups

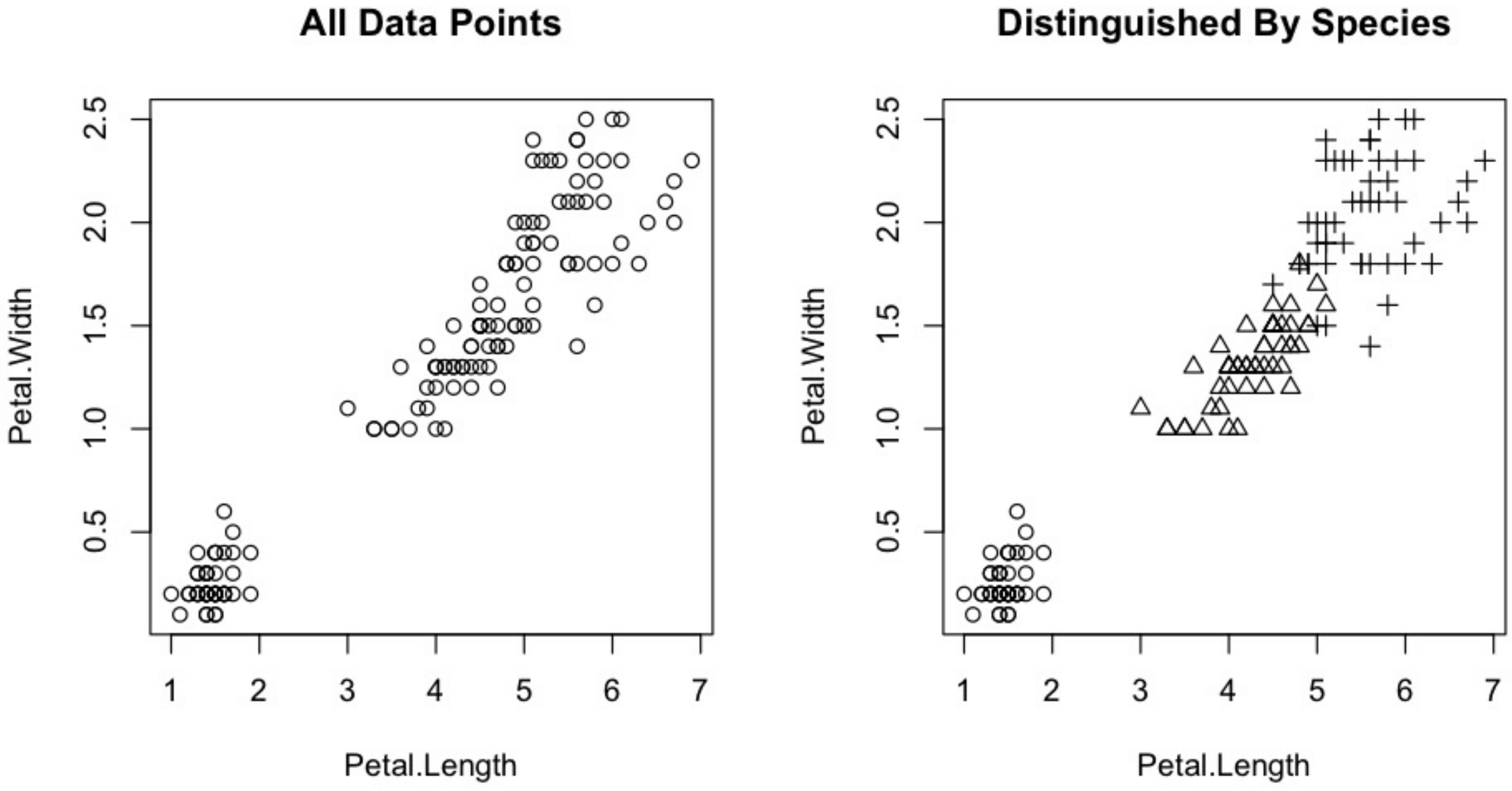
- **Problem**
  - You have paired observations in two vectors, *x* and *y*, and a parallel factor *f* that indicates their groups. You want to create a scatter plot of *x* and *y* that distinguishes among the groups.

- **Solution**
  - Use the **pch** argument of **plot**. It will plot each point with a different plotting character, according to its group:

```
> plot(x, y, pch=as.integer(f))
```

Creating a Scatter Plot of Multiple Groups

```
> par(mfrow=c(1,2))
> with(iris, plot(Petal.Length, Petal.Width, main="All Data Points"))
> with(iris, plot(Petal.Length, Petal.Width, pch=as.integer(Species),
+ main="Distinguished By Species"))
```

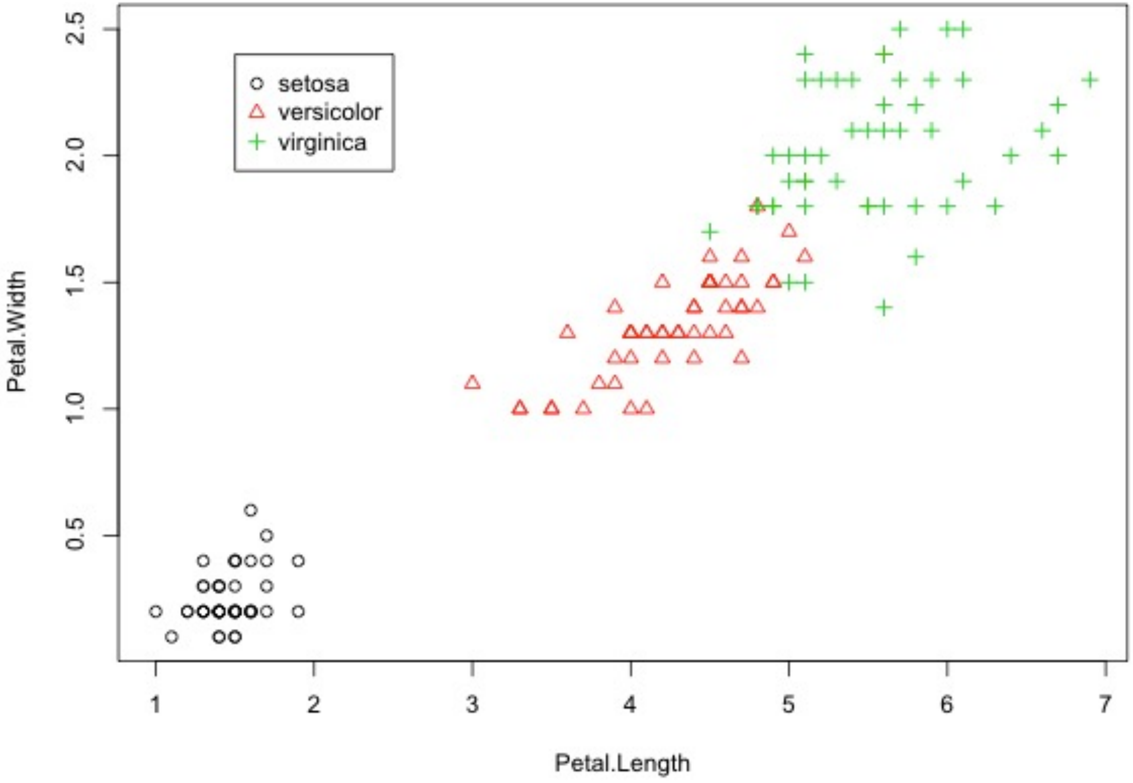


Adding a Legend

- **Problem**
  - You want your plot to include a legend, the little box that decodes the graphic for the viewer.
- **Solution**
  - After calling **plot**, call the **legend** function:
    - *Legend for points*
      - legend(x, y, labels, pch=c(*pointtype1*, *pointtype2*, ...))
    - *Legend for lines according to line type*
      - legend(x, y, labels, lty=c(linetype1, linetype2, ...))
    - *Legend for lines according to line width*
      - legend(x, y, labels, lwd=c(width1, width2, ...))
    - *Legend for colors*
      - legend(x, y, labels, col=c(*color1*, *color2*, ...))

Adding a Legend

```
> f <- factor(iris$Species)
> with(iris, plot(Petal.Length, Petal.Width, pch=as.integer(f),
+     col=as.integer(f)))
> legend(1.5, 2.4, as.character(levels(f)), pch=1:length(levels(f)),
+     col=1:length(levels(f)))
```



Plotting the Regression Line of a Scatter Plot

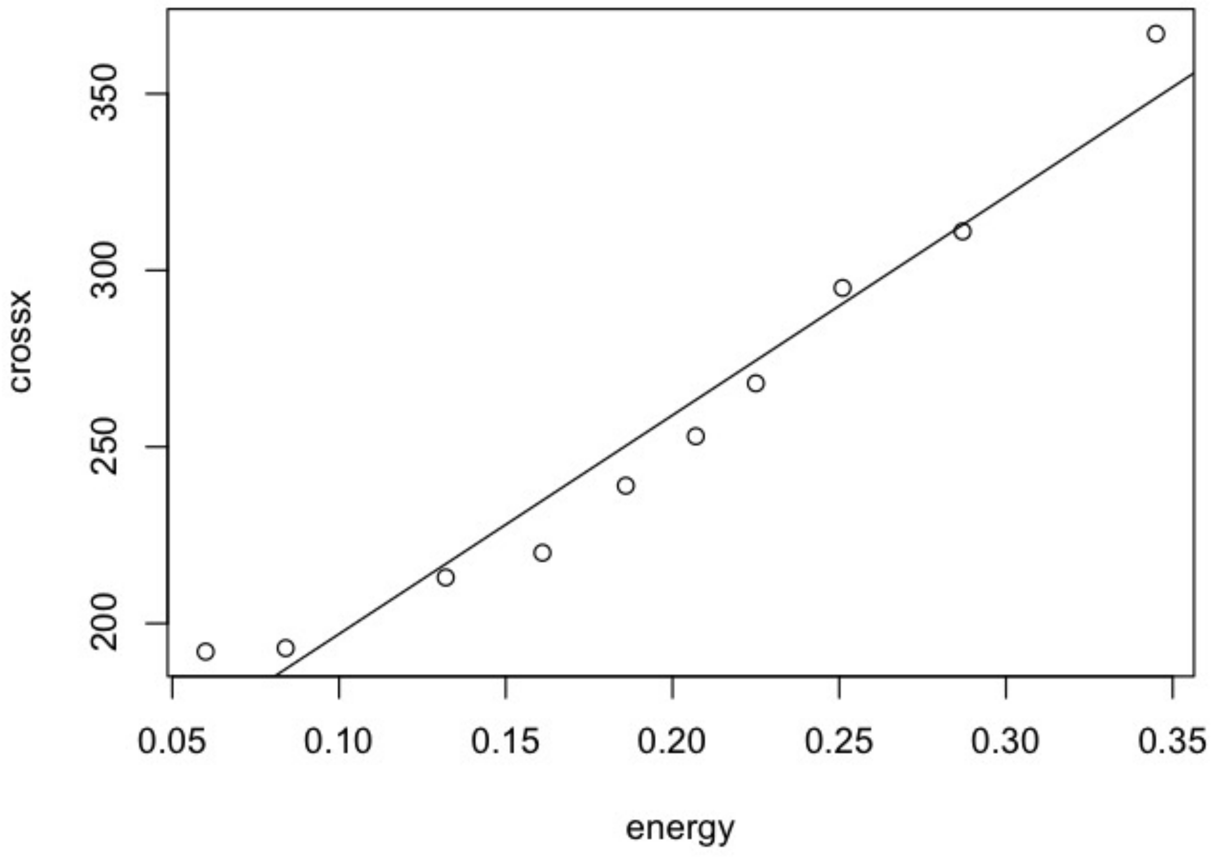
- **Problem**
  - You are plotting pairs of data points, and you want to add a line that illustrates their linear regression.

- **Solution**
  - Create a model object, plot the (x, y) pairs, and then plot the model object using the **abline** function

```
> m <- lm(y ~ x)
> plot(y ~ x)
> abline(m)
```

Plotting the Regression Line of a Scatter Plot

```
> library(faraway)
> data(strongx)
> m <- lm(crossx ~ energy, data=strongx)
> plot(crossx ~ energy, data=strongx)
> abline(m)
```



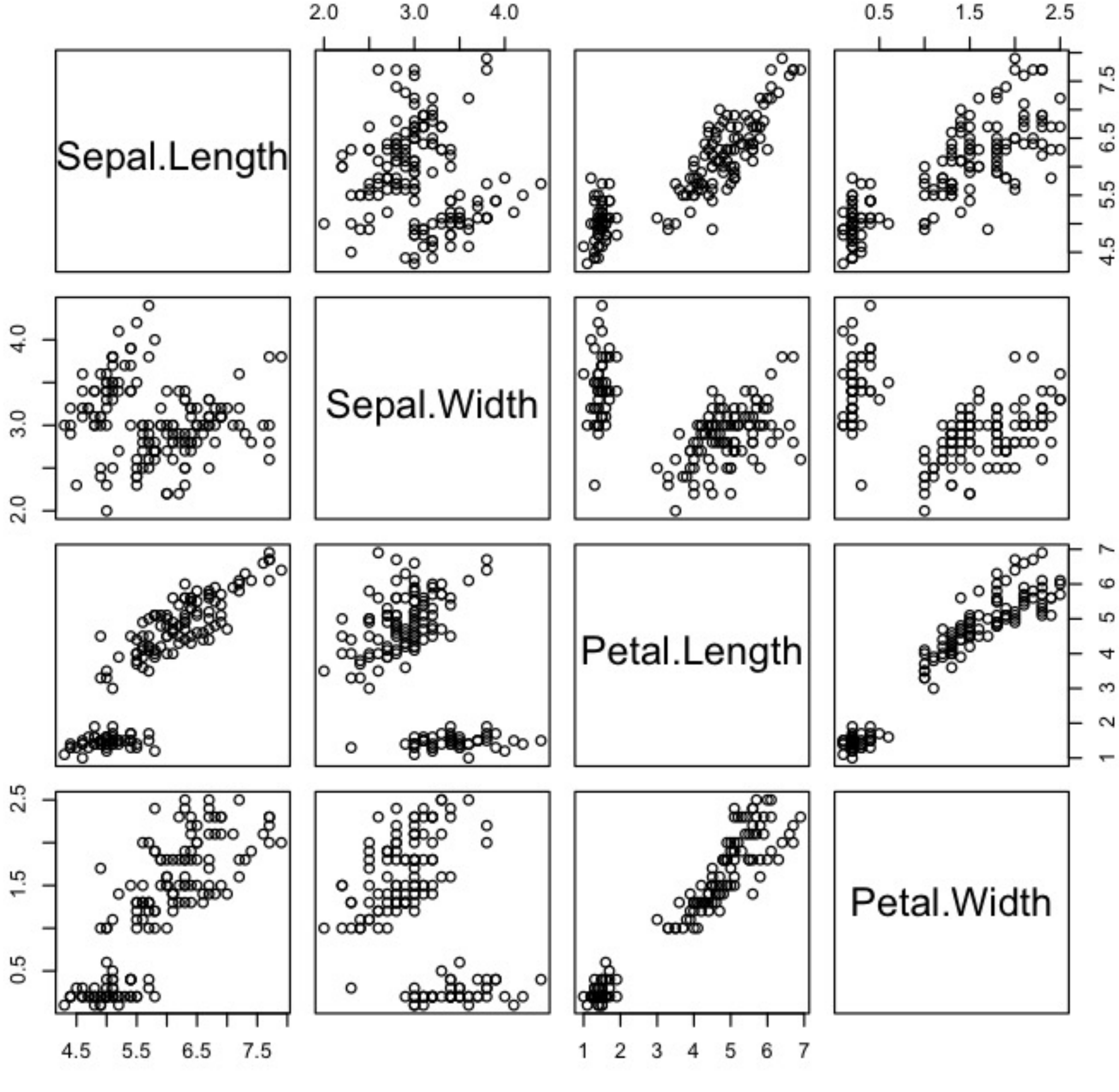
Plotting All Variables Against All Other Variables

- **Problem**
  - Your dataset contains multiple numeric variables. You want to see scatter plots for all pairs of variables.
- **Solution**
  - Place your data in a data frame and then plot the data frame. R will create one scatter plot for every pair of columns:

```
> plot(dfrm)
> head(iris)
 Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
……
```

Plotting All Variables Against All Other Variables

```
> plot(iris[,1:4])
```



Creating One Scatter Plot for Each Factor Level

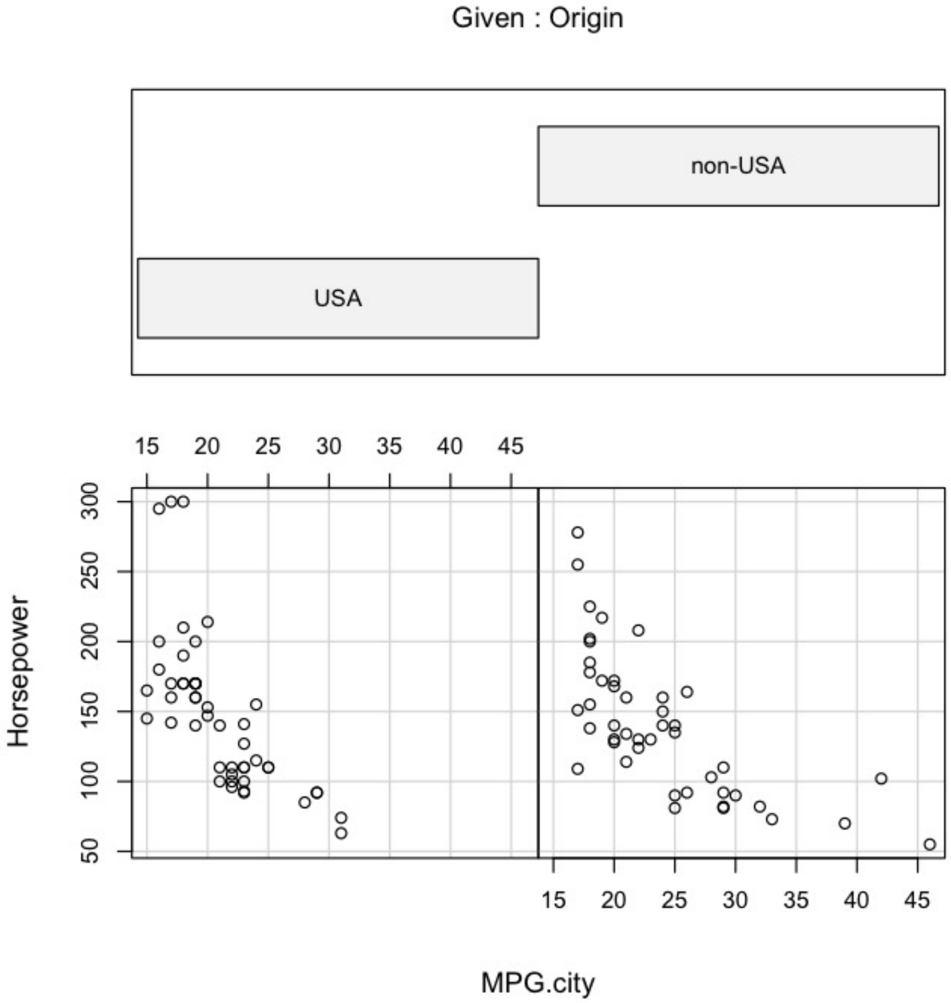
- **Problem**
  - Your dataset contains (at least) two numeric variables and a factor. You want to create several scatter plots for the numeric variables, with one scatter plot for each level of the factor.

- **Solution**
  - This kind of plot is called a *conditioning plot*, which is produced by the **coplot** function:

```
> coplot(y ~ x | f)
```

Creating One Scatter Plot for Each Factor Level

```
> data(Cars93, package="MASS")
> coplot(Horsepower ~ MPG.city | Origin, data=Cars93)
```



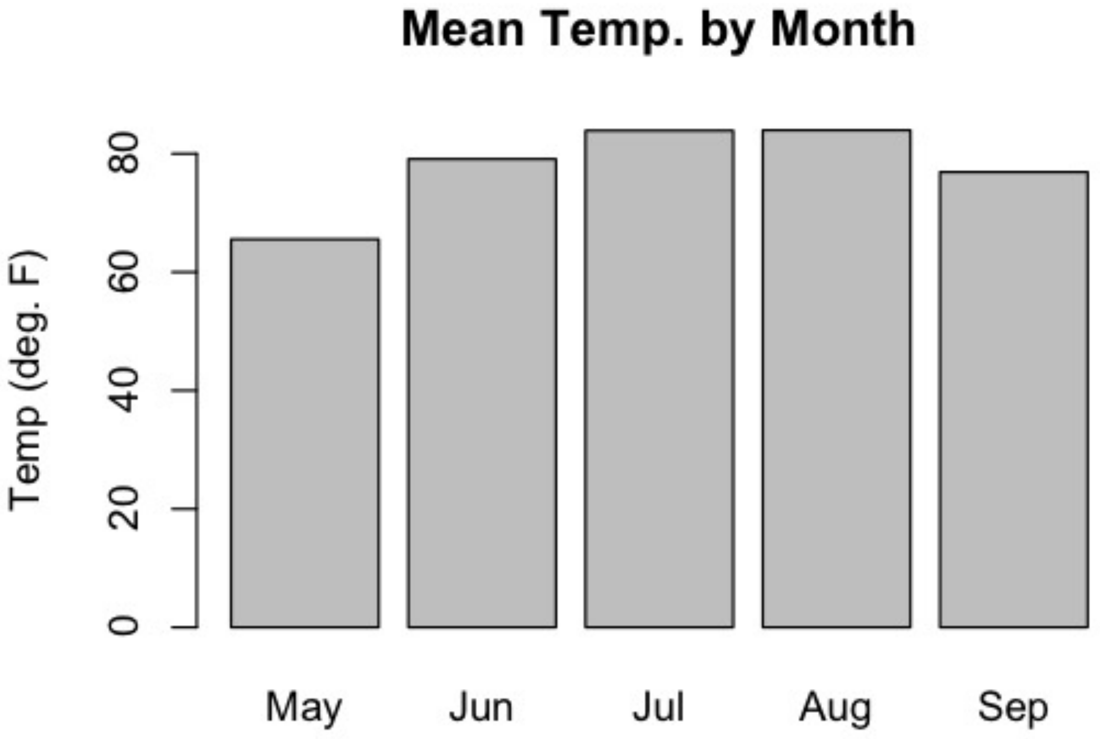
Creating a Bar Chart

- **Problem**
  - You want to create a bar chart.

- **Solution**
  - Use the **barplot** function. The first argument is a vector of bar heights:
    - > barplot(c(*height*$_1$, *height*$_2$, ..., *height*$_n$))

Creating a Bar Chart

```
> heights <- tapply(airquality$Temp, airquality$Month, mean)
> barplot(heights, main="Mean Temp. by Month",
+ names.arg=c("May", "Jun", "Jul", "Aug", "Sep"),
+ ylab="Temp (deg. F)”)
```



Adding Confidence Intervals to a Bar Chart

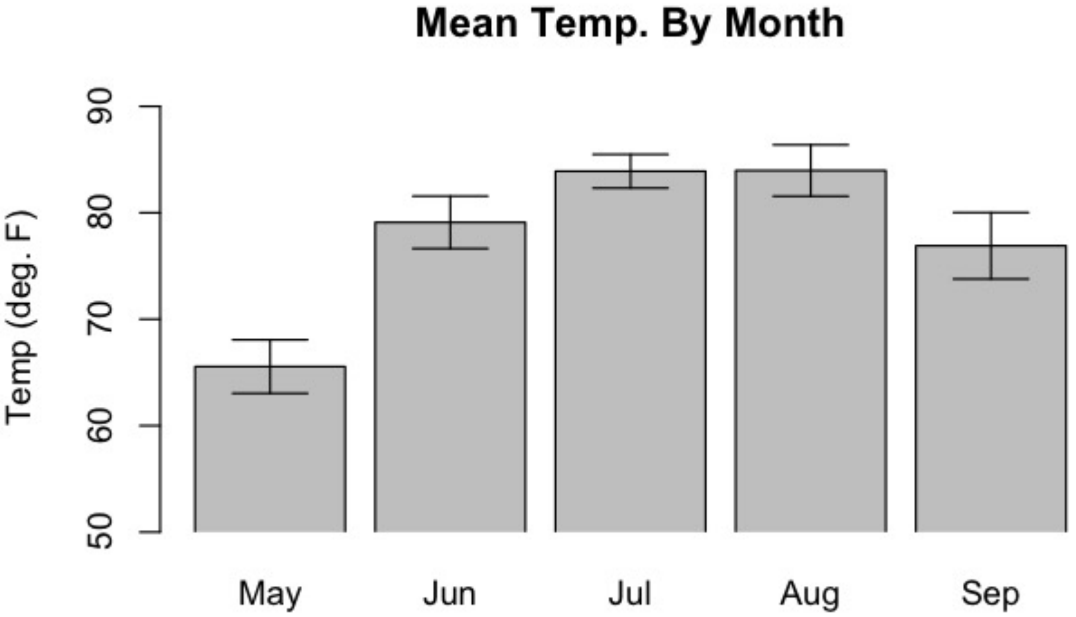
- **Problem**
  - You want to augment a bar chart with confidence intervals.

- **Solution**
  - The **barplot2** function of the **gplots** library can display a bar chart of *x* and its confidence intervals:

```
> library(gplots)
> barplot2(x, plot.ci=TRUE, ci.l=lower, ci.u=upper)
```

Adding Confidence Intervals to a Bar Chart

```
> library(gplots)
> attach(airquality)
> heights <- tapply(Temp, Month, mean)
> lower <- tapply(Temp, Month, function(v) t.test(v)$conf.int[1])
> upper <- tapply(Temp, Month, function(v) t.test(v)$conf.int[2])
> barplot2(heights, plot.ci=TRUE, ci.l=lower, ci.u=upper,
+ ylim=c(50,90), xpd=FALSE, main="Mean Temp. By Month",
+ names.arg=c("May","Jun","Jul","Aug","Sep"), ylab="Temp (deg. F)”)
```



Coloring a Bar Chart

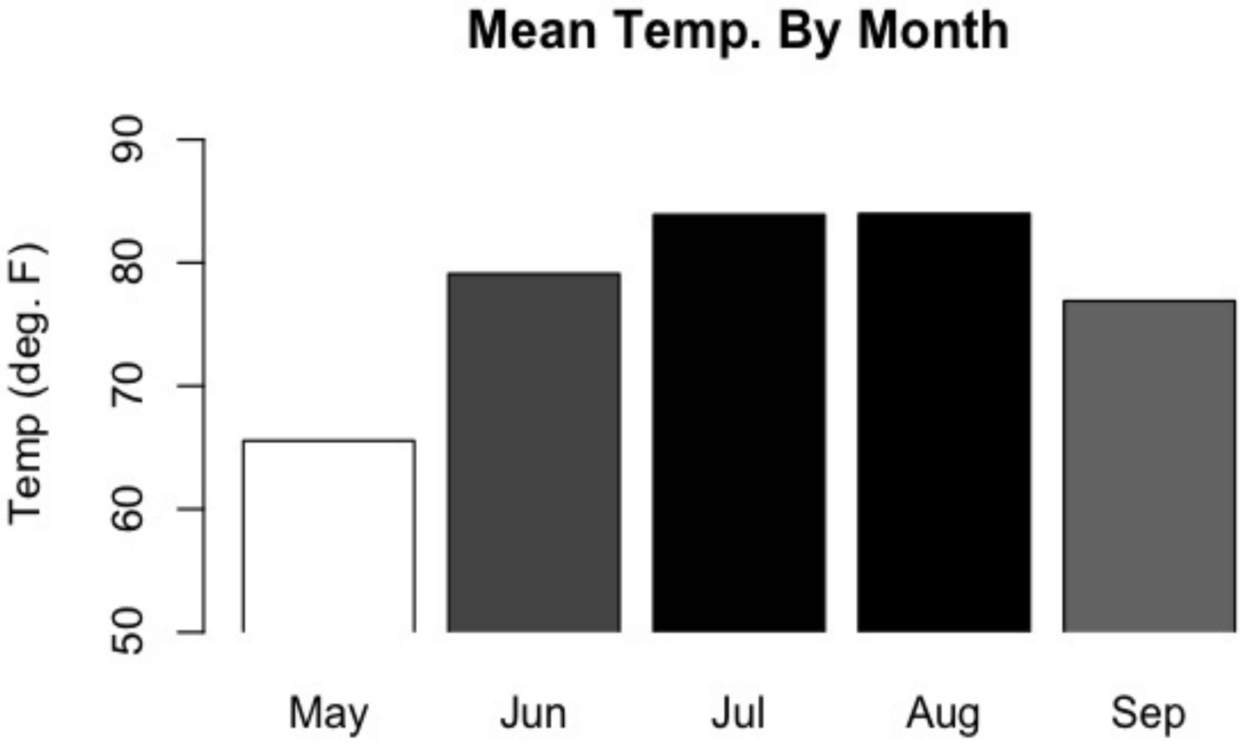
- **Problem**
  - You want to color or shade the bars of a bar chart.

- **Solution**
  - Use the **col** argument of **barplot**:

```
> barplot(heights, col=colors)
```

Coloring a Bar Chart

```
> rel.hts <- (heights - min(heights)) / (max(heights) - min(heights))
> grays <- gray(1 - rel.hts)
> barplot(heights, col=grays, ylim=c(50,90),
+ xpd=FALSE, main="Mean Temp. By Month",
+ names.arg=c("May", "Jun", "Jul", "Aug", "Sep"), ylab="Temp (deg. F)")
```



Plotting a Line from x and y Points

- **Problem**
  - You have paired observations: (x_1, y_1), (x_2, y_2), ..., (x_n, y_n). You want to plot a series of line segments that connect the data points.

- **Solution**
  - Use the **plot** function with a plot type of "l":

    ```
    > plot(x, y, type="l")
    ```

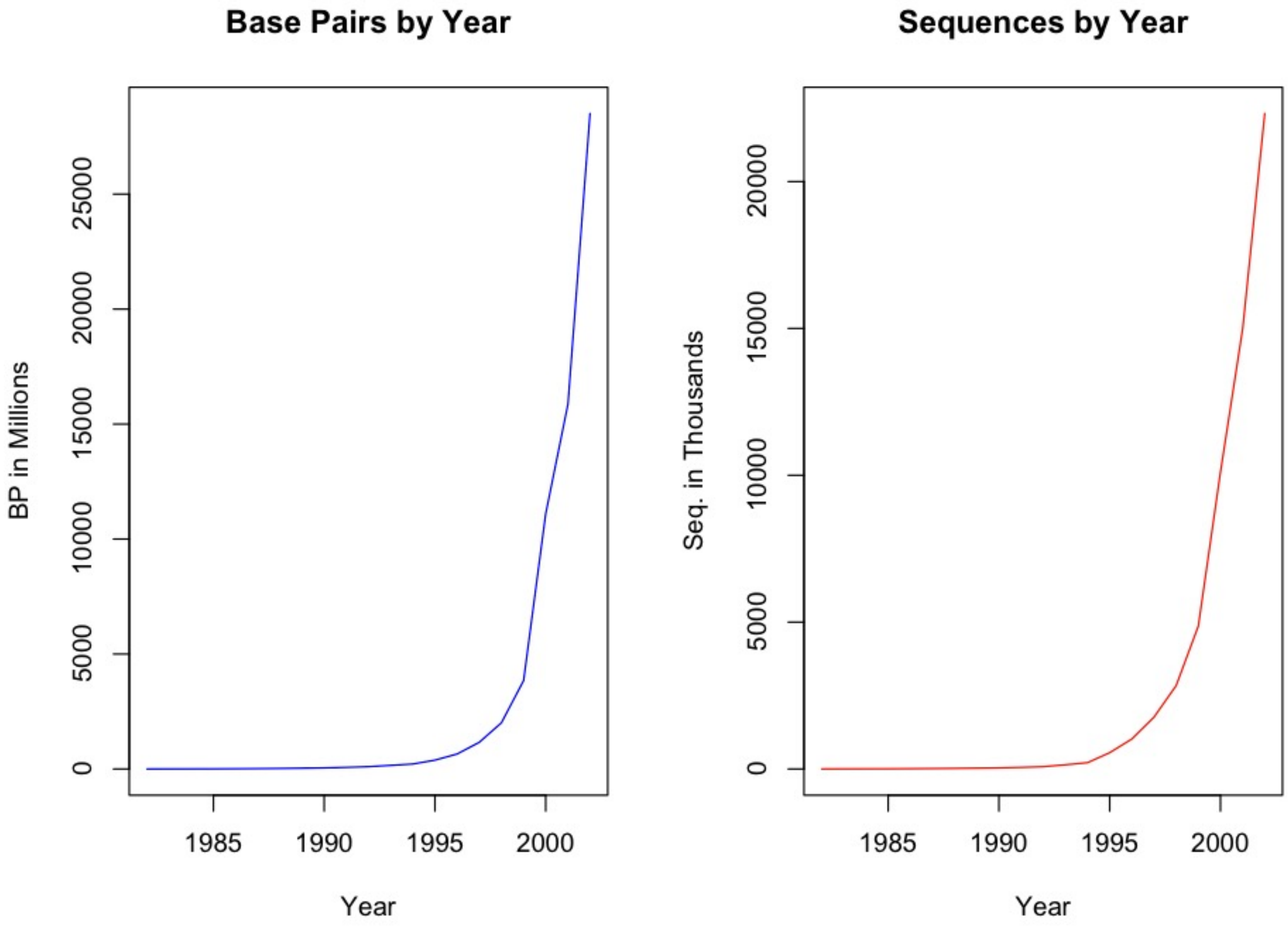
  - If your data is captured in a two-column data frame, you can plot the line segments from data frame contents:

    ```
    > plot(dfrm, type="l")
    ```

Plotting a Line from x and y Points

```
> head(NCBIdata)
  Year BasePairs Sequences
1 1982    680338       606
2 1983   2274029      2427
3 1984   3368765      4175
4 1985   5204420      5700
5 1986   9615371      9978
6 1987  15514776     14584
> plot(NCBIdata$Year, NCBIdata$BasePairs/1000000, type='l',
+    col="Blue", xlab="Year", ylab="BP in Millions",
+    main="Base Pairs by Year",)
> plot(NCBIdata$Year, NCBIdata$Sequences/1000, type='l',
+    col="Red", xlab="Year", ylab="Seq. in Thousands",
+    main="Sequences by Year”)
```

Plotting a Line from x and y Points



Changing the Type, Width, or Color of a Line

- **Problem**
  - You are plotting a line. You want to change the type, width, or color of the line.
- **Solution**
  - The **plot** function include parameters for controlling the appearance of lines. Use the **lty** parameter to control the line type:

    lty="solid" or lty=1 (default)

    lty="dashed" or lty=2

    lty="dotted" or lty=3

    lty="dotdash" or lty=4

    lty="longdash" or lty=5

    lty="twodash" or lty=6

    lty="blank" or lty=0

Changing the Type, Width, or Color of a Line

- Use the **lwd** parameter to control the line width or thickness. By default, lines have a width of 1:

```
> plot(x, y, type="l", lwd=2)
```

- Use the col parameter to control line color. By default, lines are drawn in black:

```
> plot(x, y, type="l", col="red")
```

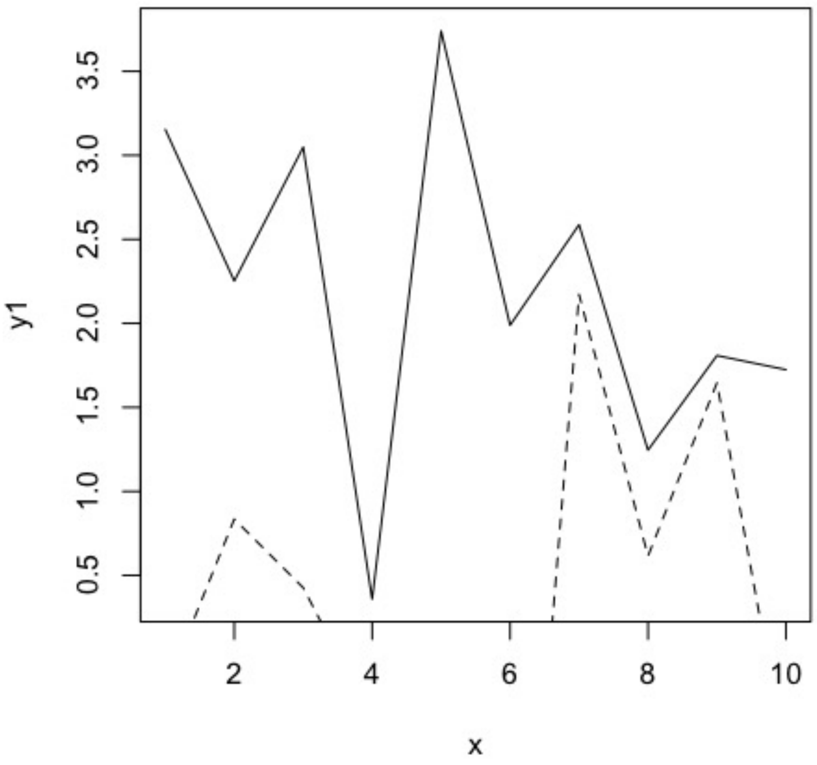
Plotting Multiple Datasets

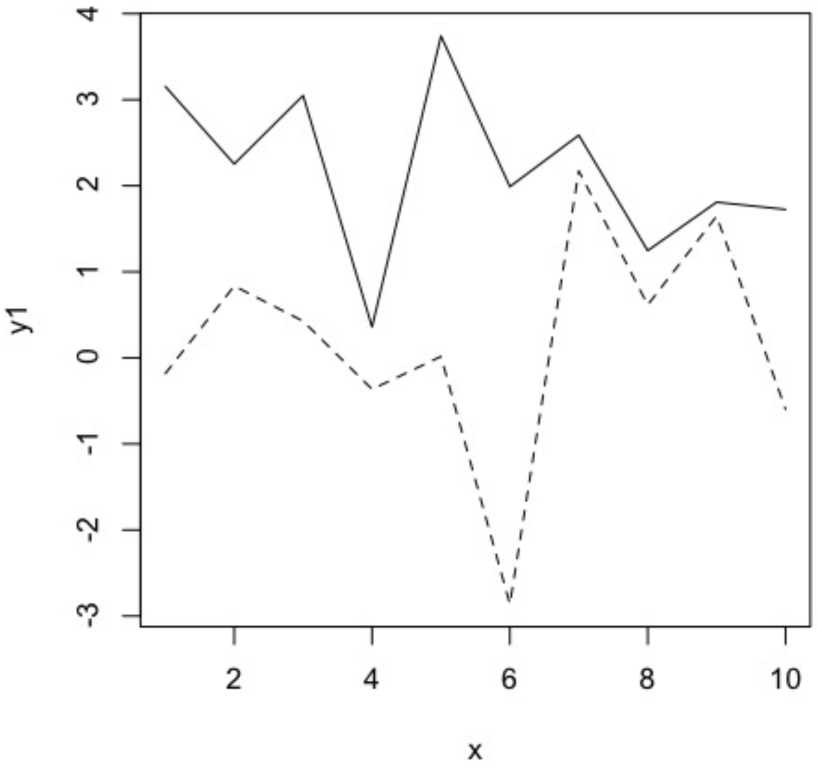
- **Problem**
  - You want to show multiple datasets in one plot.

- **Solution**
  - Initialize the plot using a high-level graphics function such as **plot** or **curve**. Then add additional datasets using low-level functions such as **lines** and **points**:

```
> plot(x1, y1, type="l")
> lines(x2, y2, lty="dashed")
> xlim <- range(c(x1,x2))
> ylim <- range(c(y1,y2))
> plot(x1, y1, type="l", xlim=xlim, ylim=ylim)
> lines(x2, y2, lty="dashed")
```

Plotting Multiple Datasets



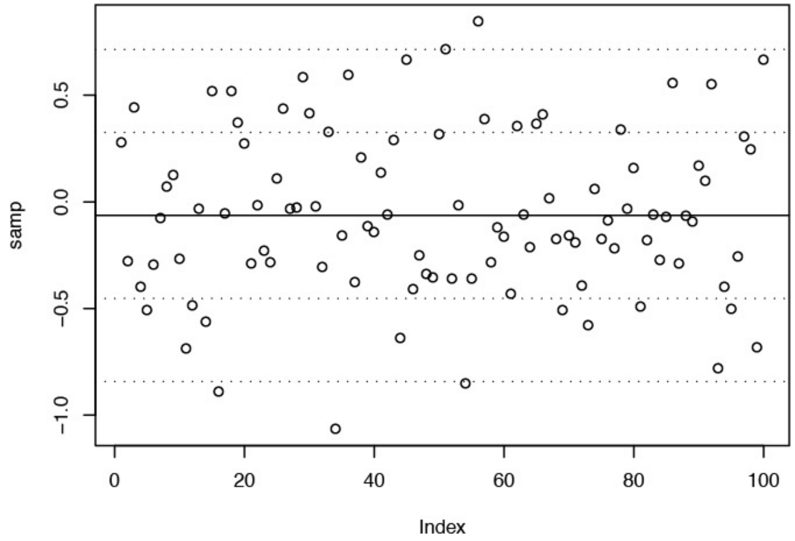


Adding Vertical or Horizontal Lines

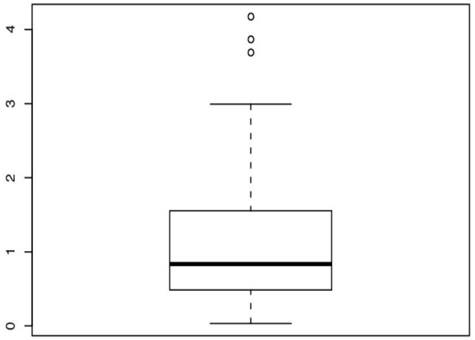
- **Problem**
  - You want to add a vertical or horizontal line to your plot, such as an axis through the origin.

- **Solution**
  - The **abline** function will draw a vertical line or horizontal line when given an argument of **v** or **h**, respectively:

```
> abline(v=x)
> abline(h=y)
```



Creating a Box Plot



- **Problem**
  - You want to create a box plot of your data.
- **Solution**
  - Use **boxplot**(x), where x is a vector of numeric values.
    - Thick line in the middle: median.
    - Top and bottom of box: Q1 and Q3.
    - Line above and below the box: the range of the data, excluding outliers.
    - The circles identify outliers. By default, an outlier is defined as any value that is farther than 1.5 × IQR away from the box. (IQR is the interquartile range, or Q3 − Q1.)

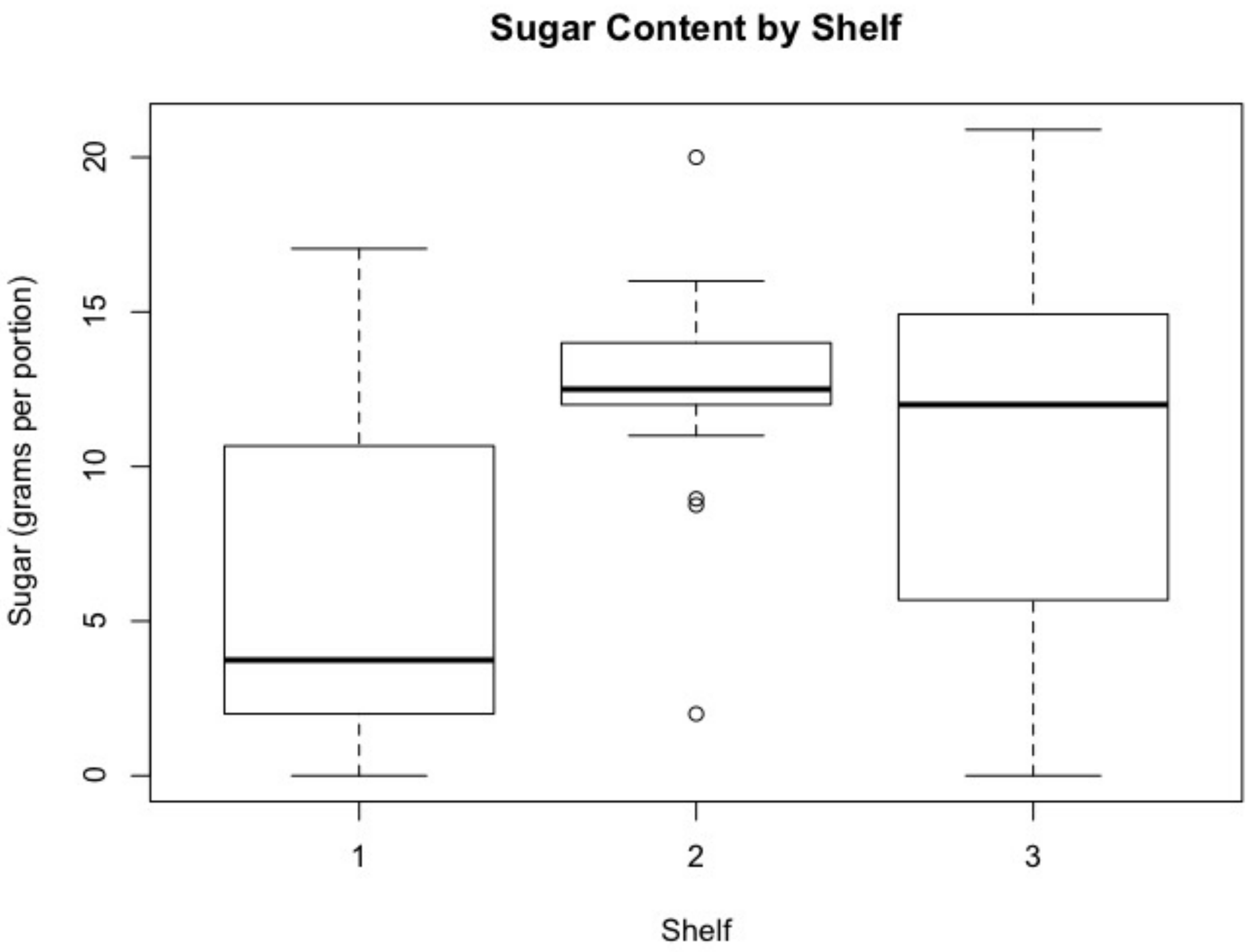
Creating One Box Plot for Each Factor Level

- **Problem**
  - Your dataset contains a numeric variable and a factor. You want to create several box plots of the numeric variable broken out by factor levels.

- **Solution**
  - Use the **boxplot** function with a formula:

```
> boxplot(x ~ f)
> data(UScereal, package="MASS")
> boxplot(sugars ~ shelf, data=UScereal,
+         main="Sugar Content by Shelf",
+         xlab="Shelf", ylab="Sugar (grams per portion)")
```

Creating One Box Plot for Each Factor Level



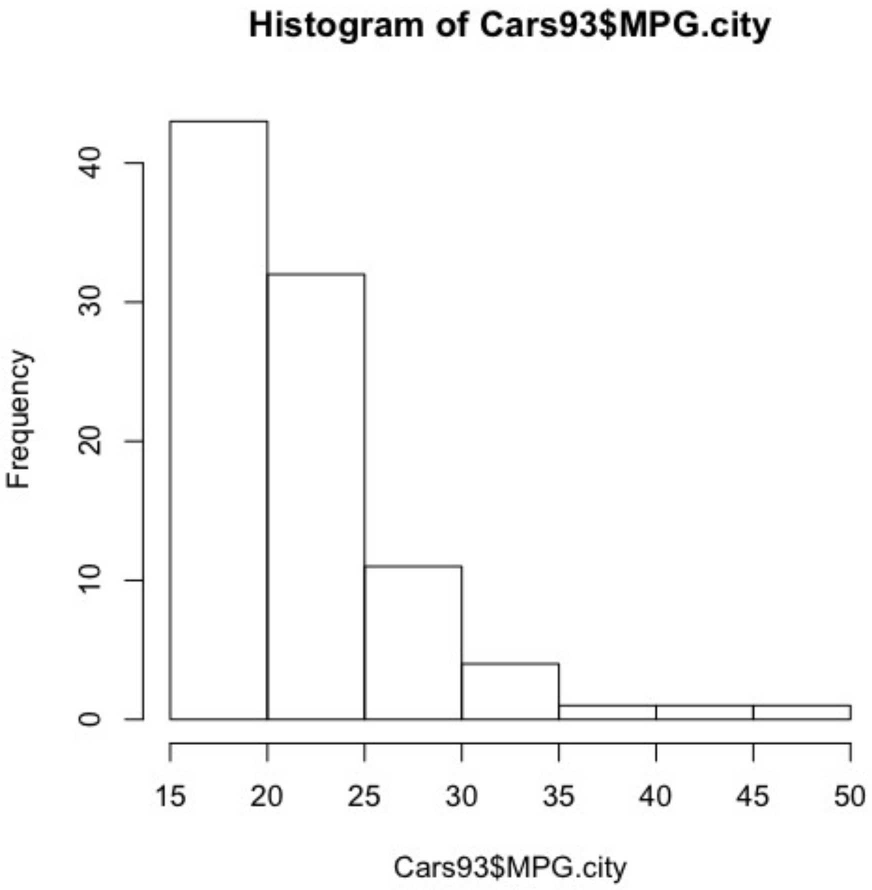
Creating a Histogram

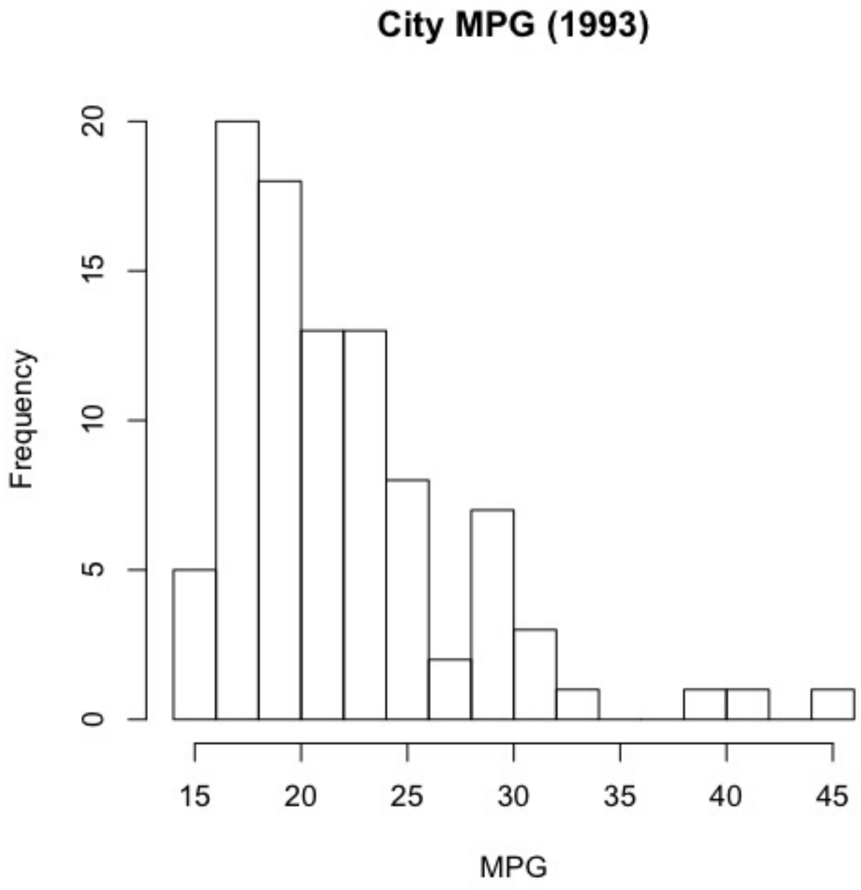
- **Problem**
  - You want to create a histogram of your data.

- **Solution**
  - Use **hist**(x), where x is a vector of numeric values.

```
> data(Cars93, package="MASS")
> hist(Cars93$MPG.city)
> hist(Cars93$MPG.city, 20, main="City MPG (1993)”,
+xlab="MPG")
```

Creating a Histogram





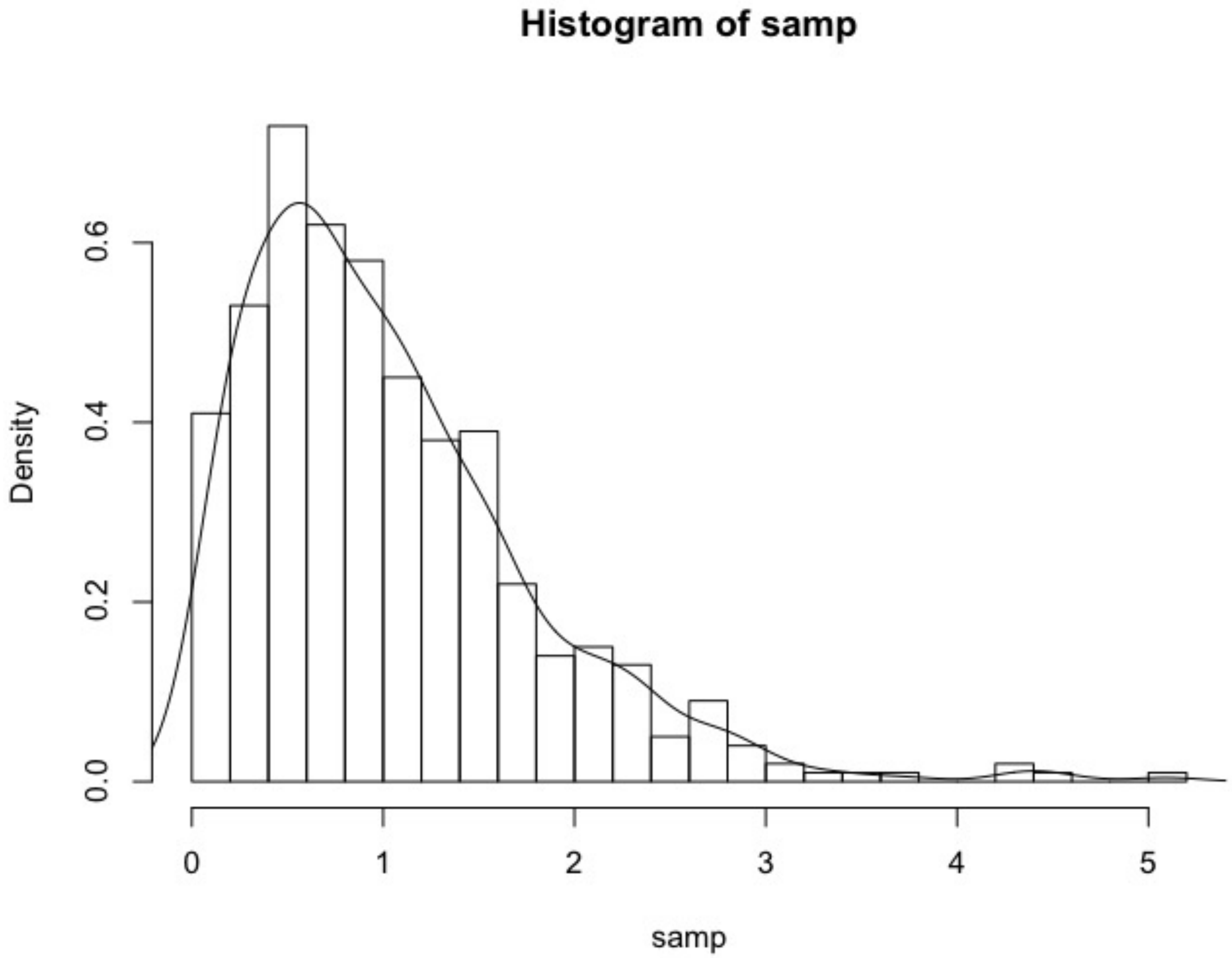
Adding a Density Estimate to a Histogram

- **Problem**
  - You have a histogram of your data sample, and you want to add a curve to illustrate the apparent density.

- **Solution**
  - Use the **density** function to approximate the sample density; then use **lines** to draw the approximation:

```
> samp <- rgamma(500, 2, 2)
> hist(samp, 20, prob=T)
> lines(density(samp))
```

Adding a Density Estimate to a Histogram



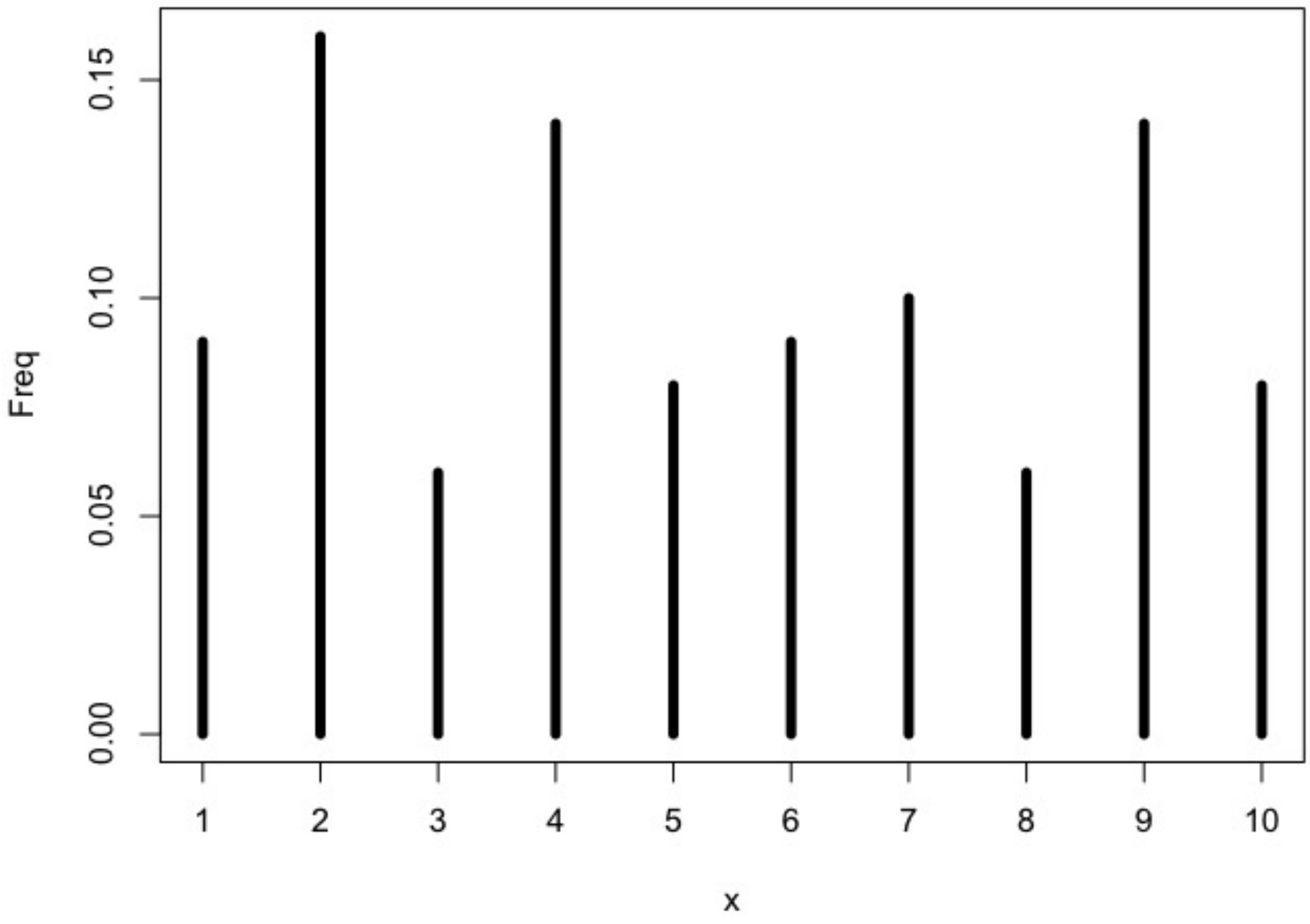
Creating a Discrete Histogram

- **Problem**
  - You want to create a histogram of discrete data.

- **Solution**
  - Use the **table** function to count occurrences. Then use the **plot** function with type="h" to graph the occurrances as a histogram:

```
> x <- sample(1:10, 100, replace=T)
> plot(table(x)/length(x), type="h", lwd=5, ylab="Freq")
```

Creating a Discrete Histogram



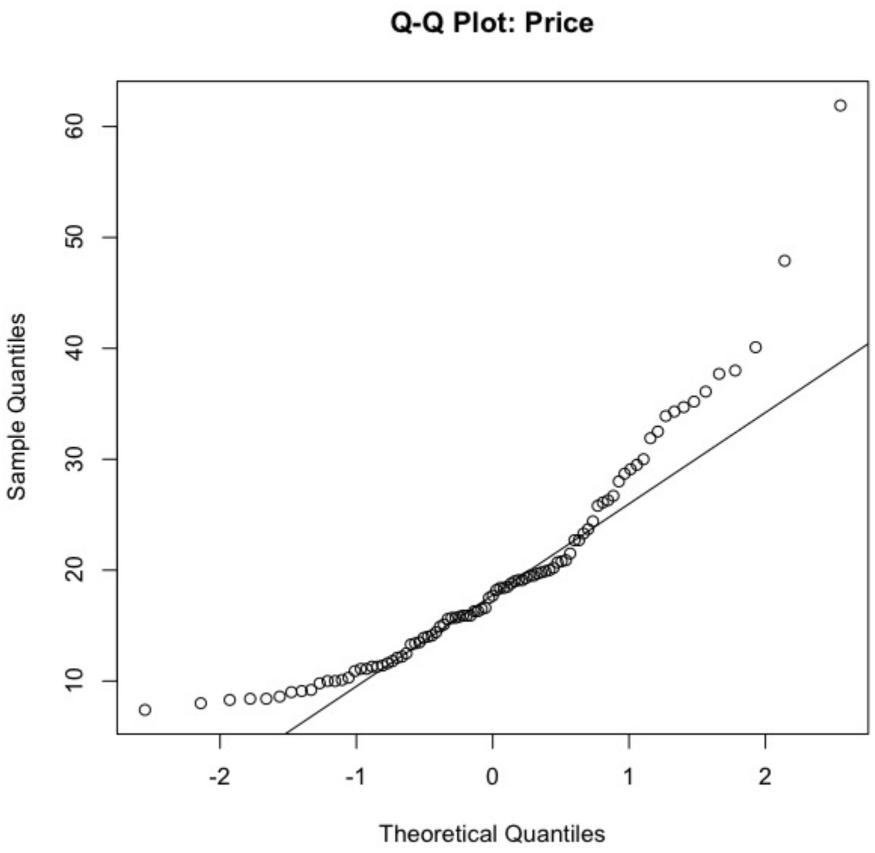
Creating a Normal Quantile-Quantile (Q-Q) Plot

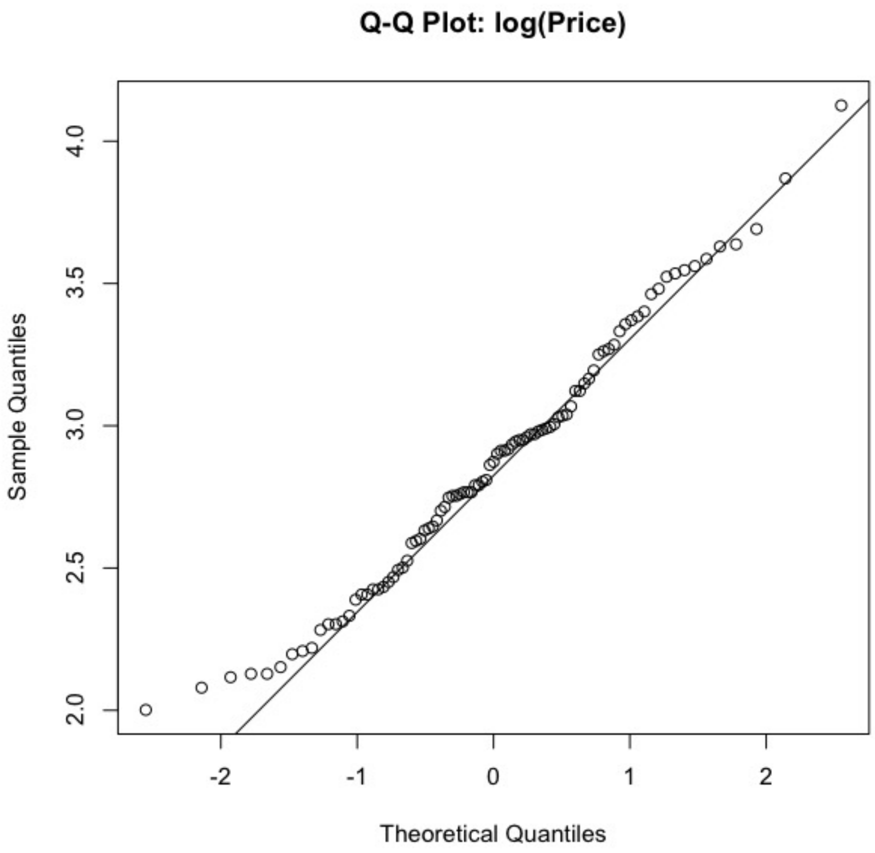
- **Problem**
  - You want to create a quantile-quantile (Q-Q) plot of your data, typically because you want to know whether the data is normally distributed.

- **Solution**
  - Use **qqnorm** function to create the basic quantile-quantile plot; then use **qqline** to augment it with a diagonal line:

```
> data(Cars93, package="MASS")
> qqnorm(Cars93$Price, main="Q-Q Plot: Price")
> qqline(Cars93$Price)
> qqnorm(log(Cars93$Price), main="Q-Q Plot: log(Price)")
> qqline(log(Cars93$Price))
```

Creating a Normal Quantile-Quantile (Q-Q) Plot





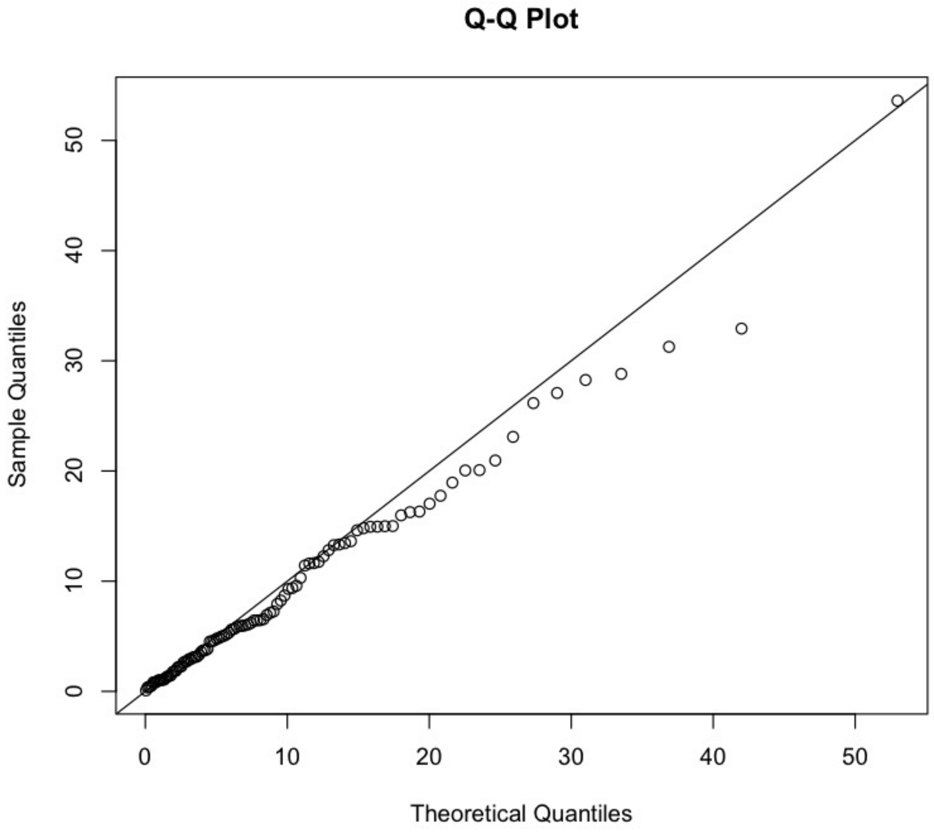
Creating Other Quantile-Quantile Plots

- **Problem**
  - You want to view a quantile-quantile plot for your data, but the data is not normally distributed.

- **Solution**
  - You must have some idea of the underlying distribution.
    - Use the **ppoints** function to generate a sequence of points between 0 and 1.
    - Transform those points into quantiles.
    - Sort the sample data.
    - Plot the sorted data against the computed quantiles.

Creating Other Quantile-Quantile Plots

```
> RATE <- 1/10
> N <- 100
> y <- rexp(N, rate=RATE)
> plot(qexp(ppoints(N), rate=RATE), sort(y), main="Q-Q Plot",
+    xlab="Theoretical Quantiles", ylab="Sample Quantiles")
> abline(a=0,b=1)
```



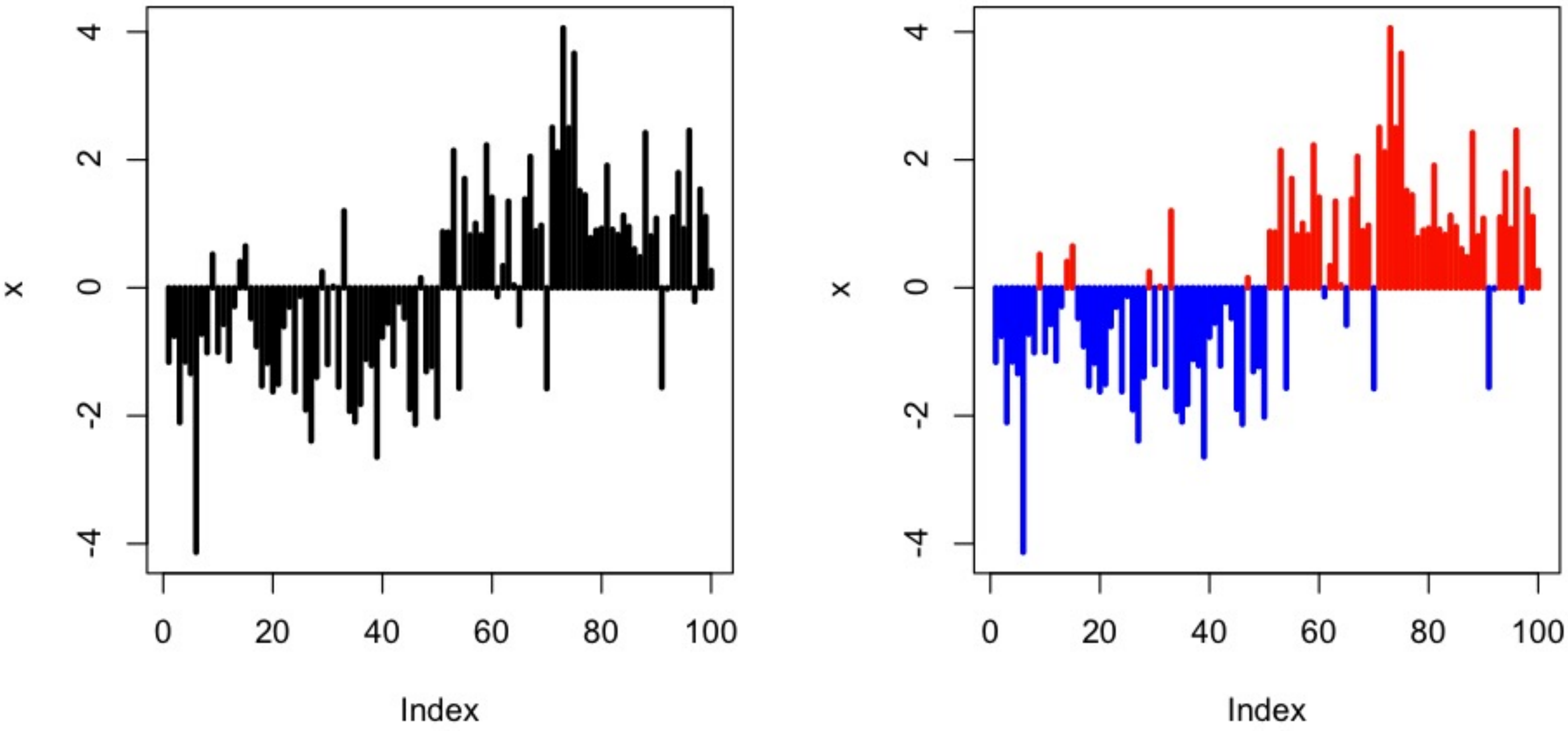
Plotting a Variable in Multiple Colors

- **Problem**
  - You want to plot your data in multiple colors, typically to make the plot more informative, readable, or interesting.

- **Solution**
  - Use the **col** argument of the plot function:

```
> plot(x, col=colors)
> x <- c(rnorm(50, mean=-1), rnorm(50, mean=1))
> plot(x, type="h", lwd=3)
> colors <- ifelse(x >= 0, "red", "blue")
> plot(x, type='h', lwd=3, col=colors)
```

Plotting a Variable in Multiple Colors



Graphing a Function

- **Problem**
  - You want to graph the value of a function.

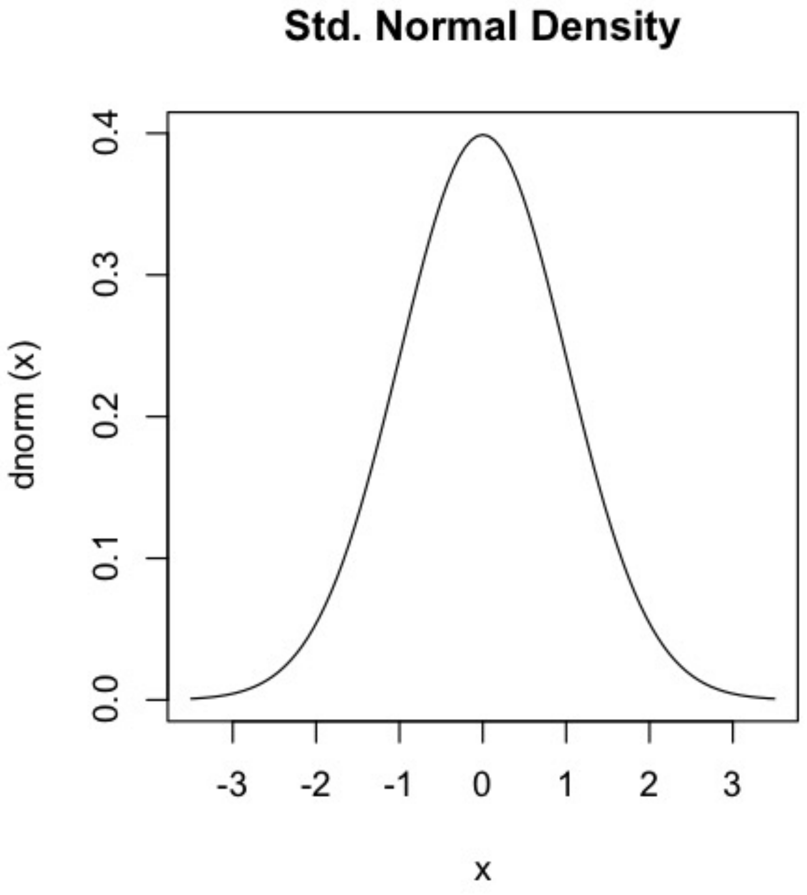
- **Solution**
  - The **curve** function can graph a function, given the function and the limits of its domain:

```
> curve(dnorm, -3.5, 3.5, main="Std. Normal Density")
```

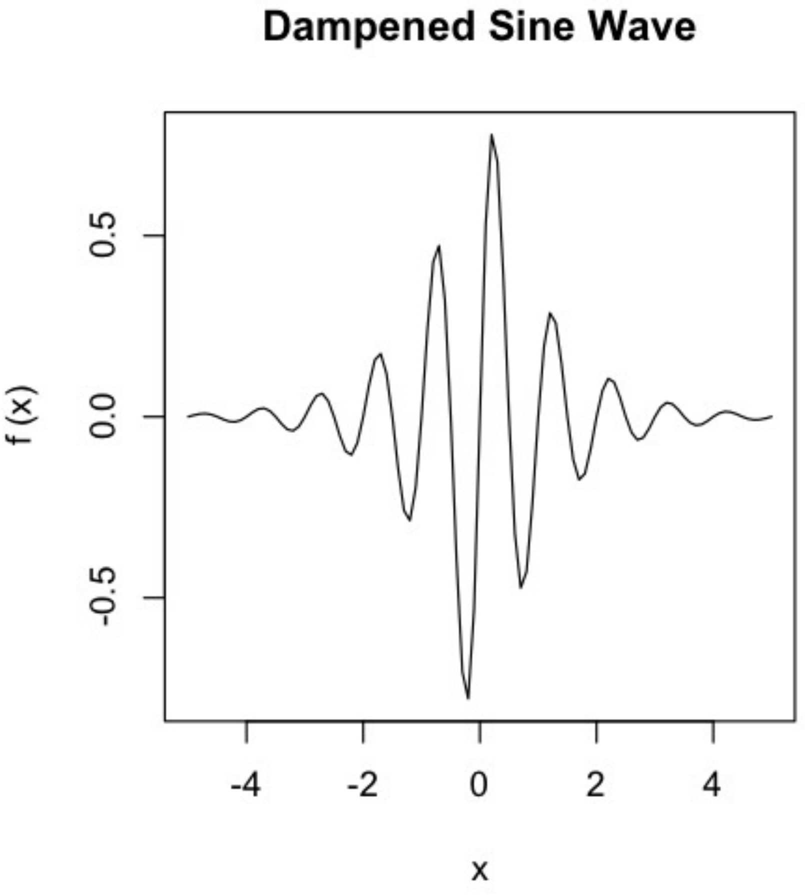
  - The **curve** function can graph any function that takes one argument and returns one value:

```
> f <- function(x) exp(-abs(x)) * sin(2*pi*x)
> curve(f, -5, +5, main="Dampened Sine Wave")
```

Graphing a Function



Std. Normal Density



Dampened Sine Wave

Pausing Between Plots

- **Problem**
  - You are creating several plots, and each plot is overwriting the previous one. You want R to pause between plots so you can view each one before it's overwritten.
- **Solution**
  - There is a global graphics option called **ask**. Set it to TRUE, and R will pause before each new plot:

    ```
    > par(ask=TRUE)
    ```
  - When you are tired of R pausing between plots, set it to FALSE:

    ```
    > par(ask=FALSE)
    ```

Displaying Several Figures on One Page

- **Problem**
  - You want to display several plots side by side on one page.

- **Solution**
  - Divide the graphics window into a matrix of N rows and M columns by setting the graphics parameter called **mfrow** or **mfcol**. Its value is a two-element vector giving the number of rows and columns:

```
> par(mfrow=(c(N,M))    # fill the graphics window row by row
> par(mfcol=(c(N,M))     # fill the graphics window col by col
```

Writing Your Plot to a File

- **Problem**
  - You want to save your graphics in a file, such as a PNG, JPEG, or PDF file.
- **Solution**
  - Call a function to open a new graphics file, such as pdf(…), png(...) or jpeg(...).
  - Generate the graphics image.
  - Call dev.off() to close the graphics file.

```
> png("myPlot.png", width=648, height=432)
> plot(x, y, main="Scatterplot of X, Y")
> dev.off()
```

Changing Graphical Parameters

- **Problem**
  - You want to change a global parameter of the graphics software, such as line type, background color, or font size.

- **Solution**
  - Use the **par** function, which lets you set values of global graphics parameters. For example, this call to **par** will change the default line width from 1 to 2:

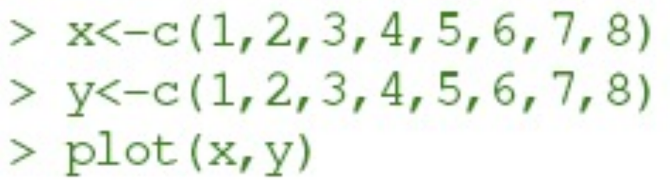
```
> par(lwd=2)
```

Parameter and type	Purpose	Example
`ask=`*`logical`*	Pause before every new graph if TRUE (Recipe 10.25)	`par(ask=TRUE)`
`bg="`*`color`*`"`	Background color	`par(bg="lightyellow")`
`cex=`*`number`*	Height of text and plotted points, expressed as a multiple of normal size	`par(cex=1.5)`
`col="`*`color`*`"`	Default plotting color	`par(col="blue")`
`fg="`*`color`*`"`	Foreground color	`par(fg="gray")`
`lty="`*`linetype`*`"`	Type of line: solid, dotted, dashed, etc. (Recipe 10.13)	`par(lty="dotted")`
`lwd=`*`number`*	Line width: 1 = normal, 2 = thicker, 3 = even thicker, etc.	`par(lwd=2)`
`mfcol=c(`*`nr`*`,`*`nc`*`)` or `mfrow=c(`*`nr`*`,`*`nc`*`)`	Create a multifigure plot matrix with *nr* rows and *nc* columns (Recipe 10.26)	`par(mfrow=c(2,2))`
`new=`*`logical`*	Used to plot one figure on top of another	`par(new=TRUE)`
`pch=`*`pointtype`*	Default point type (see help page for `points` function)	`par(pch=21)`
`xlog=`*`logical`*	Use logarithmic X scale	`par(xlog=TRUE)`
`ylog=`*`logical`*	Use logarithmic Y scale	`par(ylog=TRUE)`

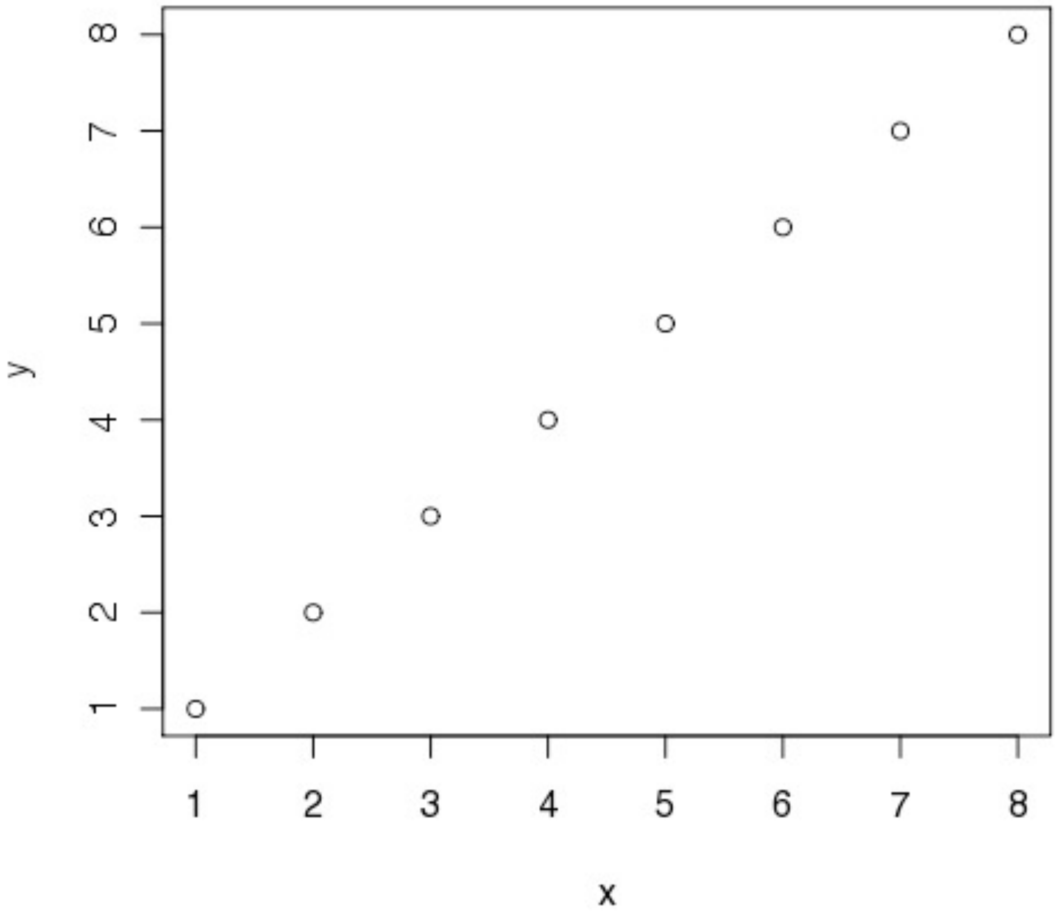
Graphics Technology in R

- It’s very useful to explore data in a graphical format using R.
- Three types of commands:
  - High-level plotting functions
    - create a new plot on the graphics device
  - Low-level plotting functions
    - add additional information to an existing plot
  - Graphical parameter functions
    - control the graphics window
    - fine-tune the appearance of graphics with colors, text, fonts, etc

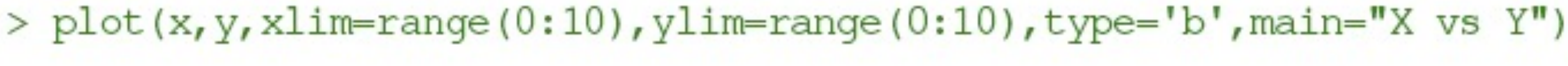
High-Level Plotting Functions



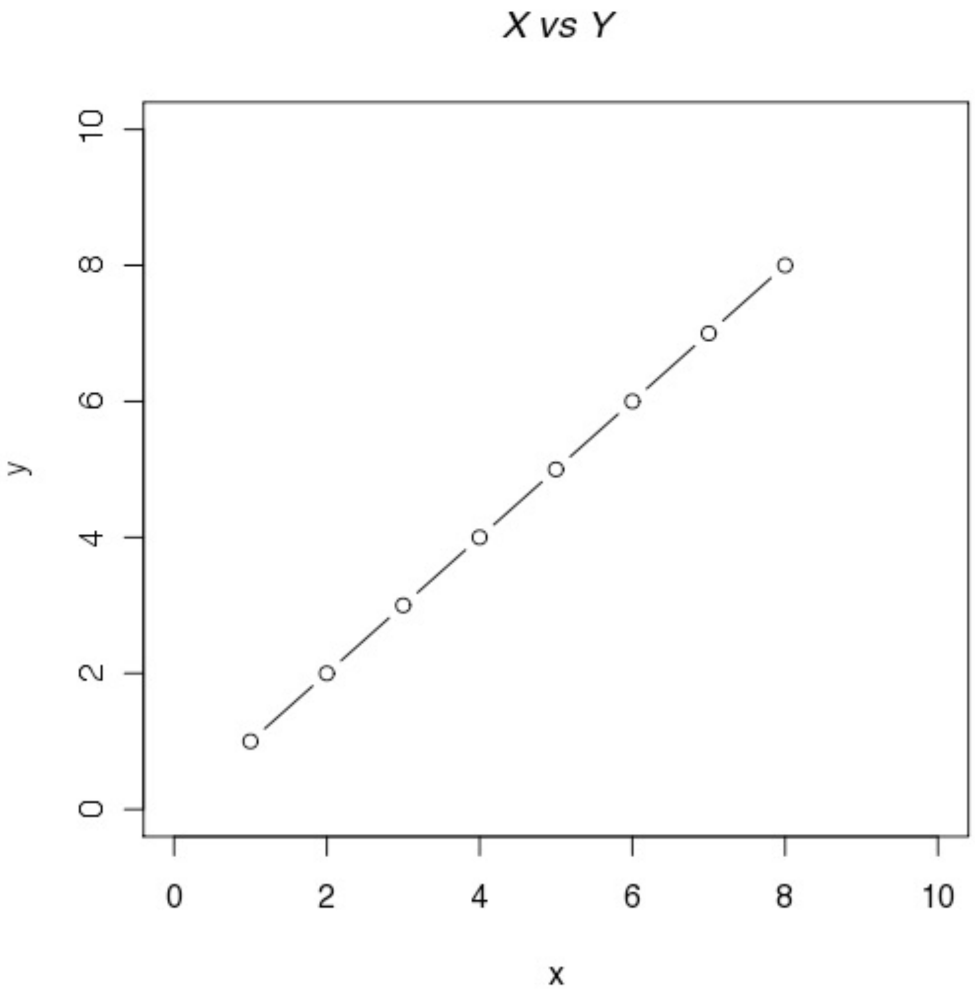
```
> x<-c(1,2,3,4,5,6,7,8)
> y<-c(1,2,3,4,5,6,7,8)
> plot(x,y)
```



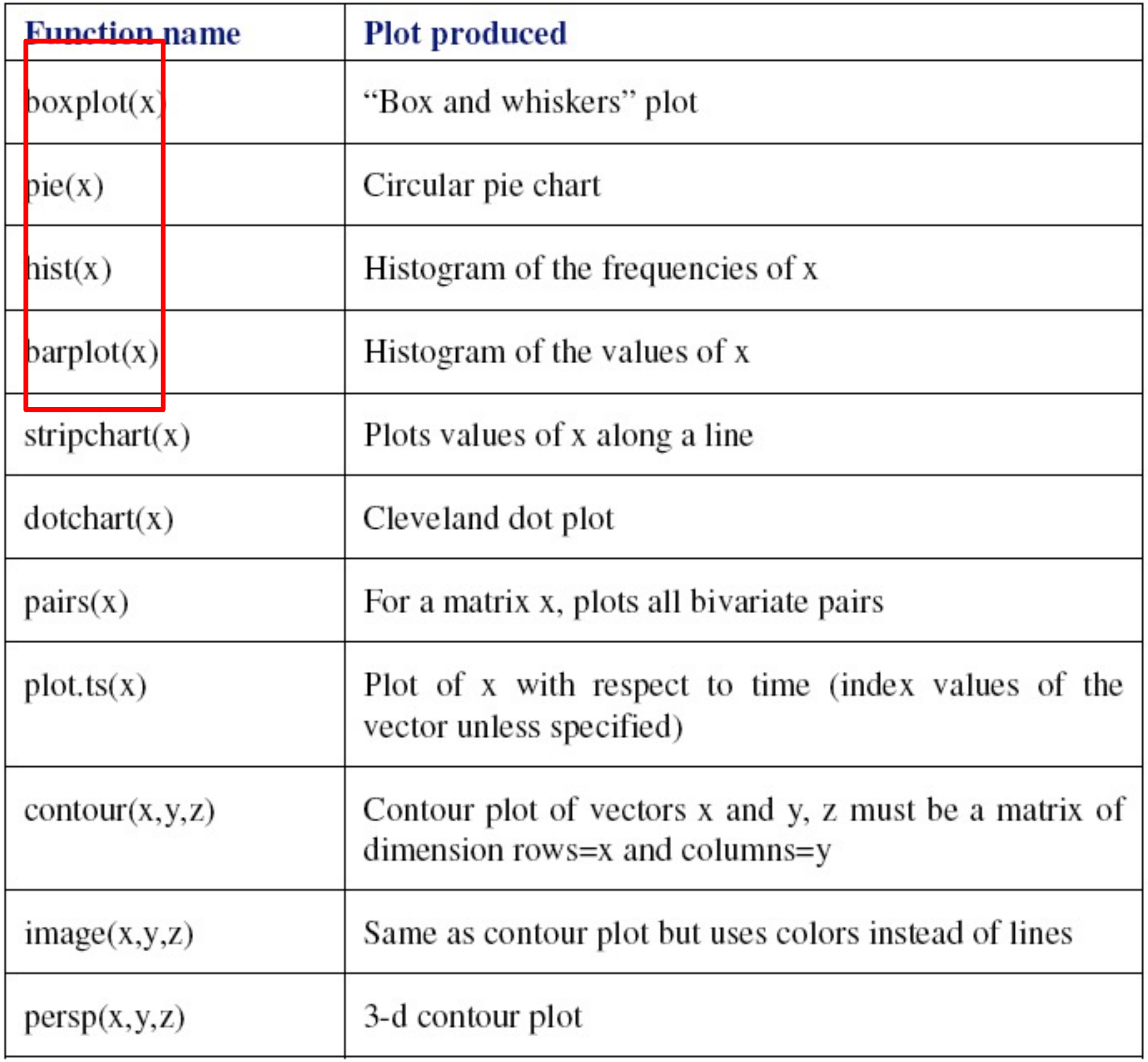
High-Level Plotting Functions



```
> plot(x,y,xlim=range(0:10),ylim=range(0:10),type='b',main="X vs Y")
```



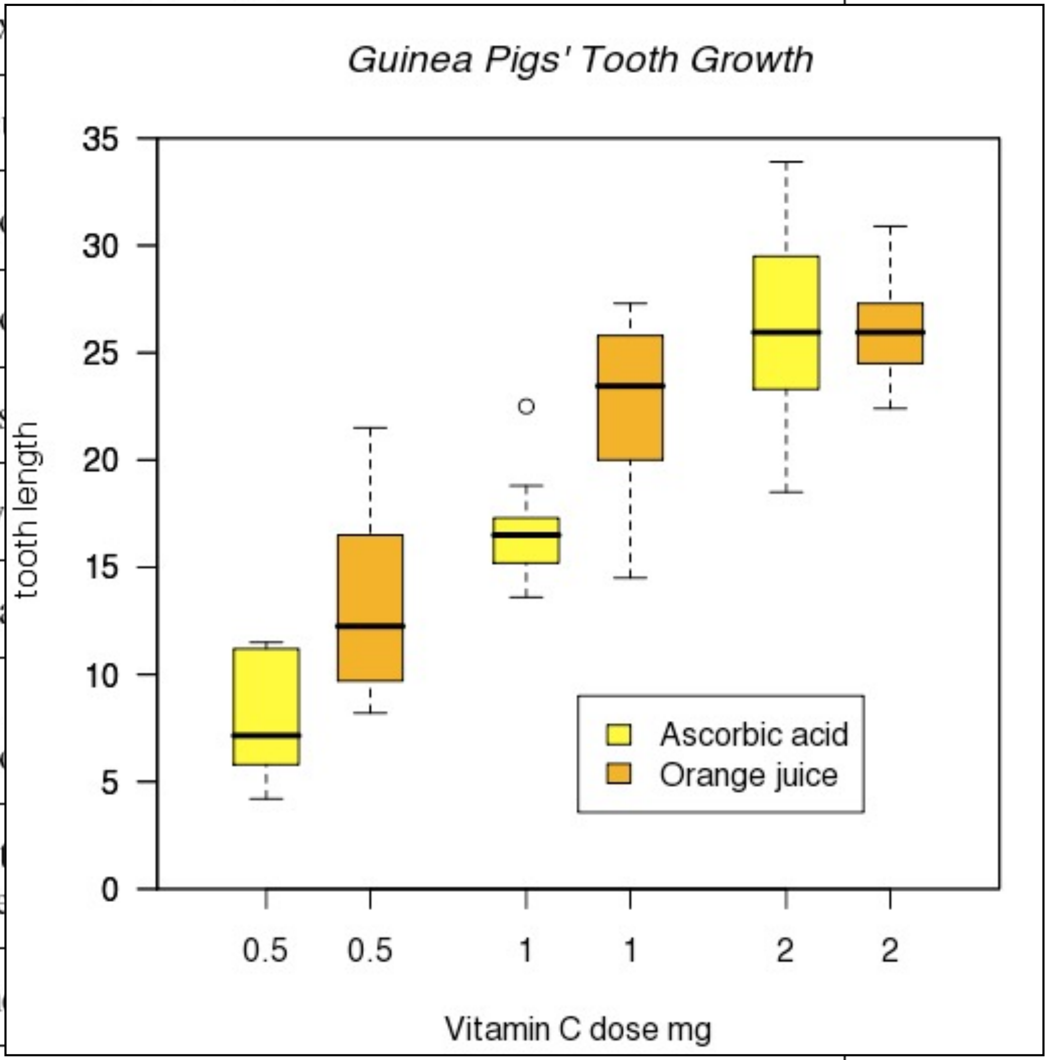
Selected High-Level Plotting Functions



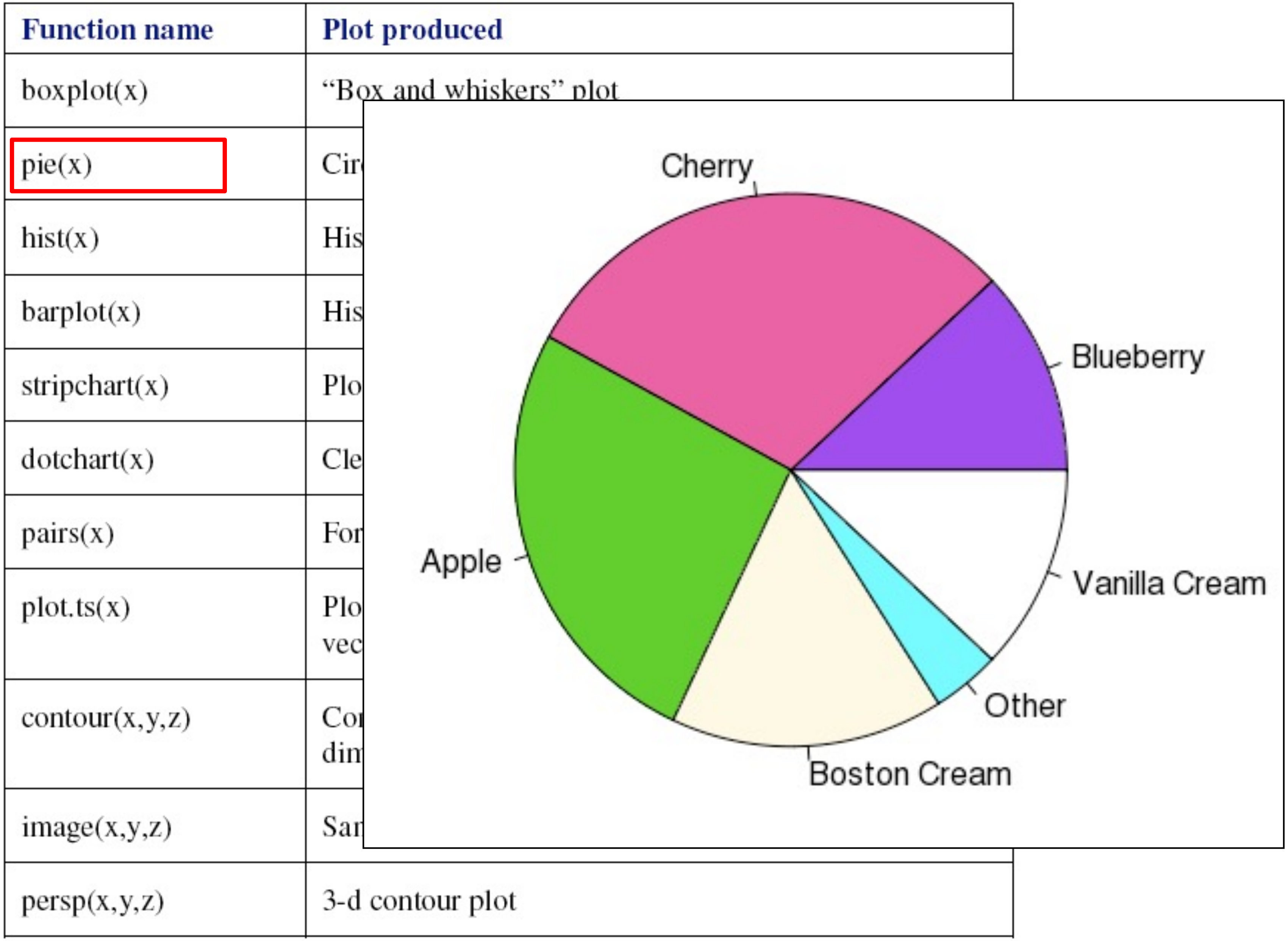
Function name	Plot produced
boxplot(x)	"Box and whiskers" plot
pie(x)	Circular pie chart
hist(x)	Histogram of the frequencies of x
barplot(x)	Histogram of the values of x
stripchart(x)	Plots values of x along a line
dotchart(x)	Cleveland dot plot
pairs(x)	For a matrix x, plots all bivariate pairs
plot.ts(x)	Plot of x with respect to time (index values of the vector unless specified)
contour(x,y,z)	Contour plot of vectors x and y, z must be a matrix of dimension rows=x and columns=y
image(x,y,z)	Same as contour plot but uses colors instead of lines
persp(x,y,z)	3-d contour plot

Selected High-Level Plotting Functions

Function name	Plot produced
boxplot(x)	“Box
pie(x)	Circ
hist(x)	Hist
barplot(x)	Hist
stripchart(x)	Plots
dotchart(x)	Clev
pairs(x)	For a
plot.ts(x)	Plot vect
contour(x,y,z)	Cont dime
image(x,y,z)	Sam
persp(x,y,z)	3-d contour plot



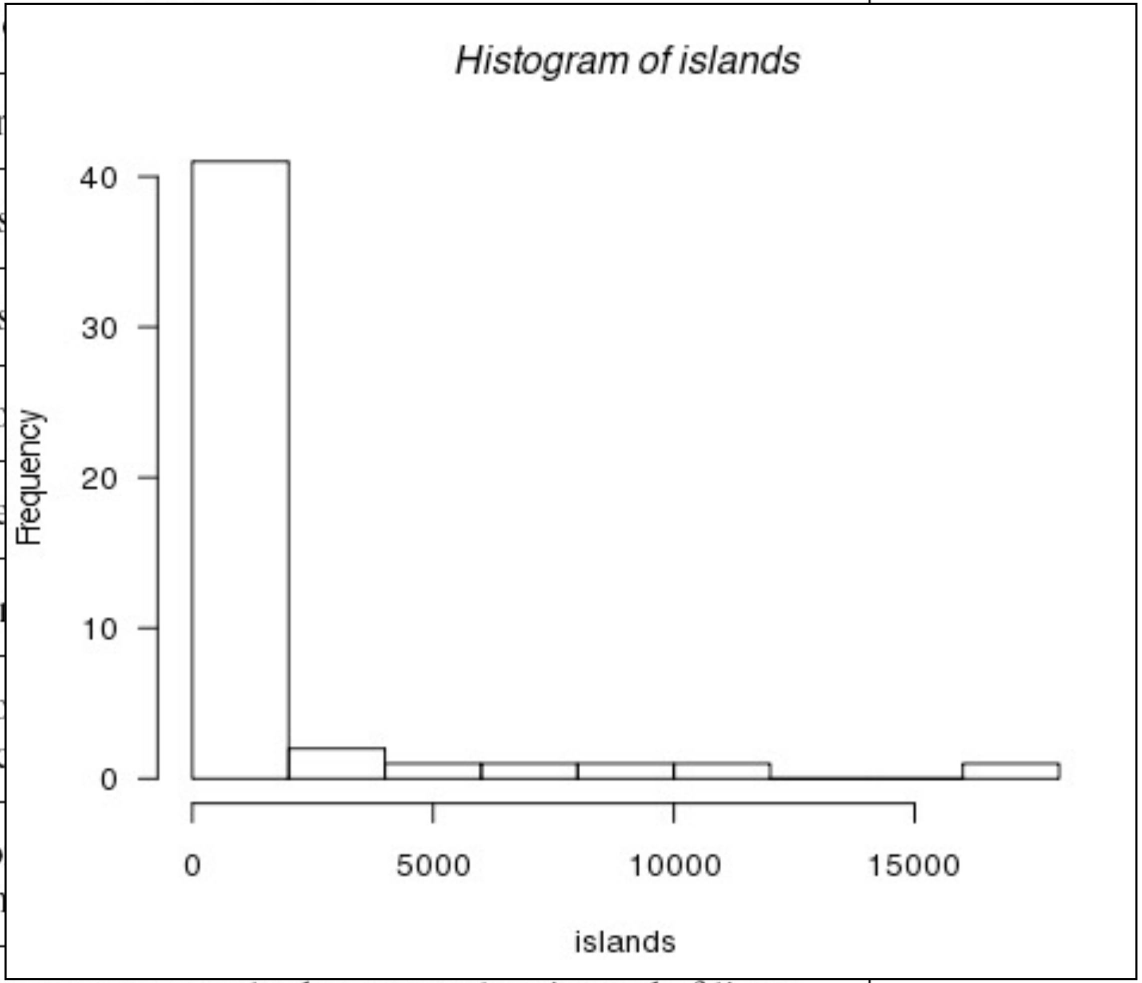
Selected High-Level Plotting Functions



Function name	Plot produced
boxplot(x)	"Box and whiskers" plot
pie(x)	Cir
hist(x)	His
barplot(x)	His
stripchart(x)	Plo
dotchart(x)	Cle
pairs(x)	For
plot.ts(x)	Plo vec
contour(x,y,z)	Cor dim
image(x,y,z)	Sar
persp(x,y,z)	3-d contour plot

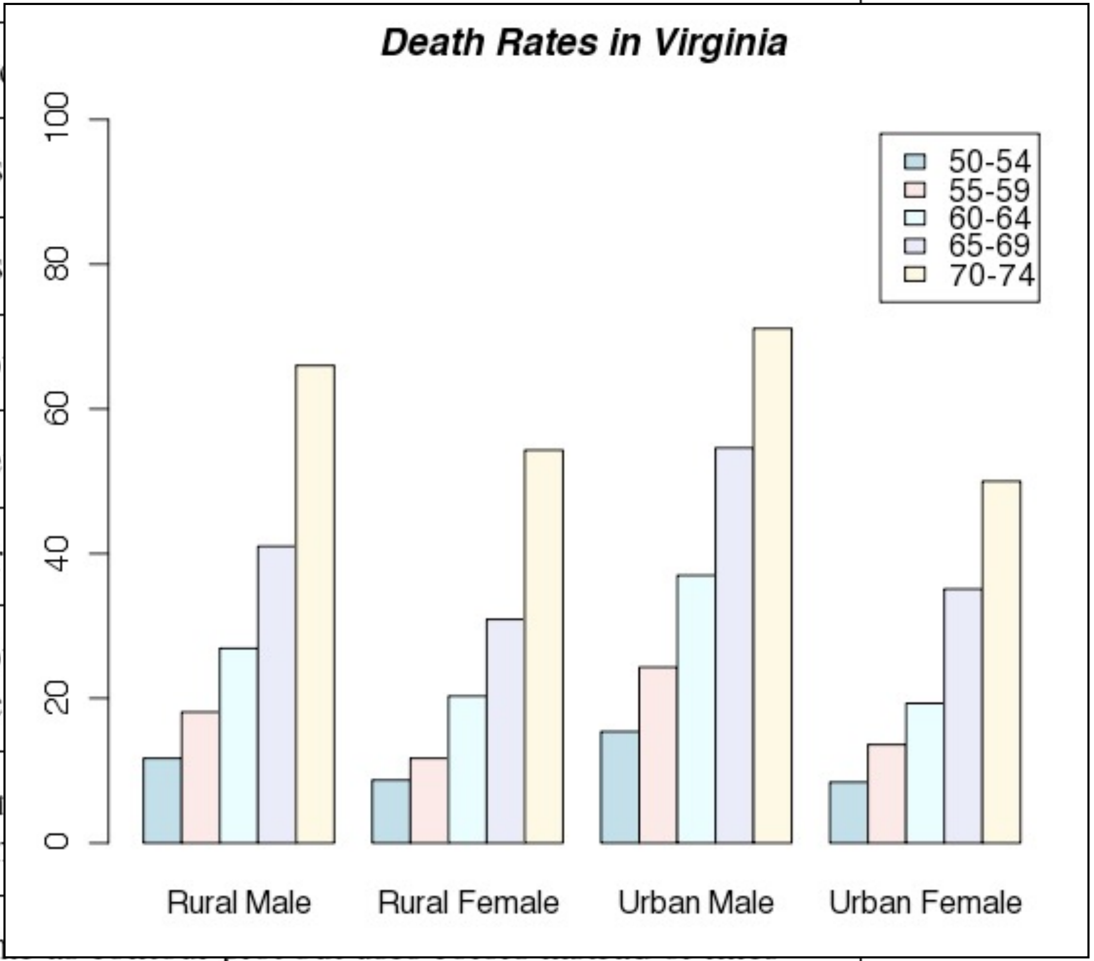
Selected High-Level Plotting Functions

Function name	Plot produced
boxplot(x)	“B
pie(x)	Cir
hist(x)	His
barplot(x)	His
stripchart(x)	Plo
dotchart(x)	Cle
pairs(x)	For
plot.ts(x)	Plo vec
contour(x,y,z)	Co dir
image(x,y,z)	Same as contour plot but uses colors instead of lines
persp(x,y,z)	3-d contour plot



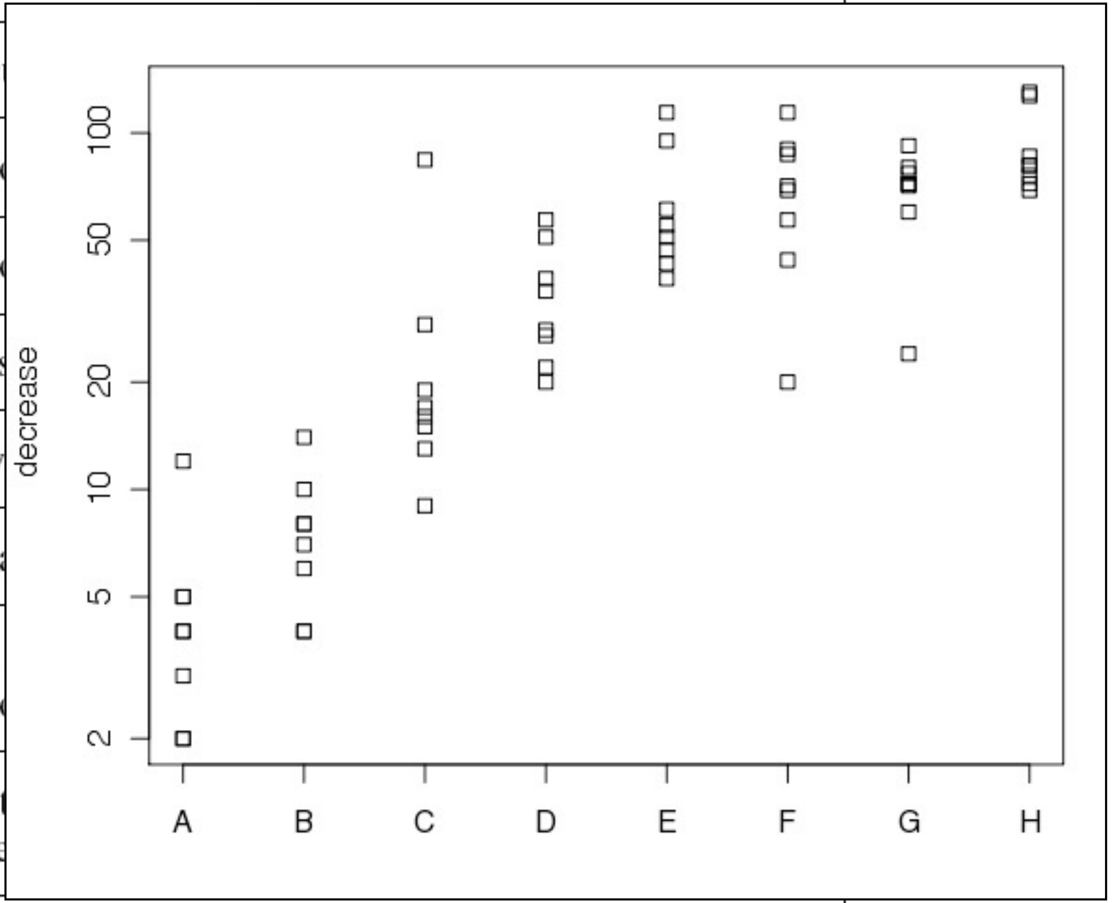
Selected High-Level Plotting Functions

Function name	Plot produced
boxplot(x)	“Box and whiskers” plot
pie(x)	Cir
hist(x)	His
barplot(x)	His
stripchart(x)	Plo
dotchart(x)	Cle
pairs(x)	For
plot.ts(x)	Plo vec
contour(x,y,z)	Cor dim
image(x,y,z)	San
persp(x,y,z)	3-d contour plot



Selected High-Level Plotting Functions

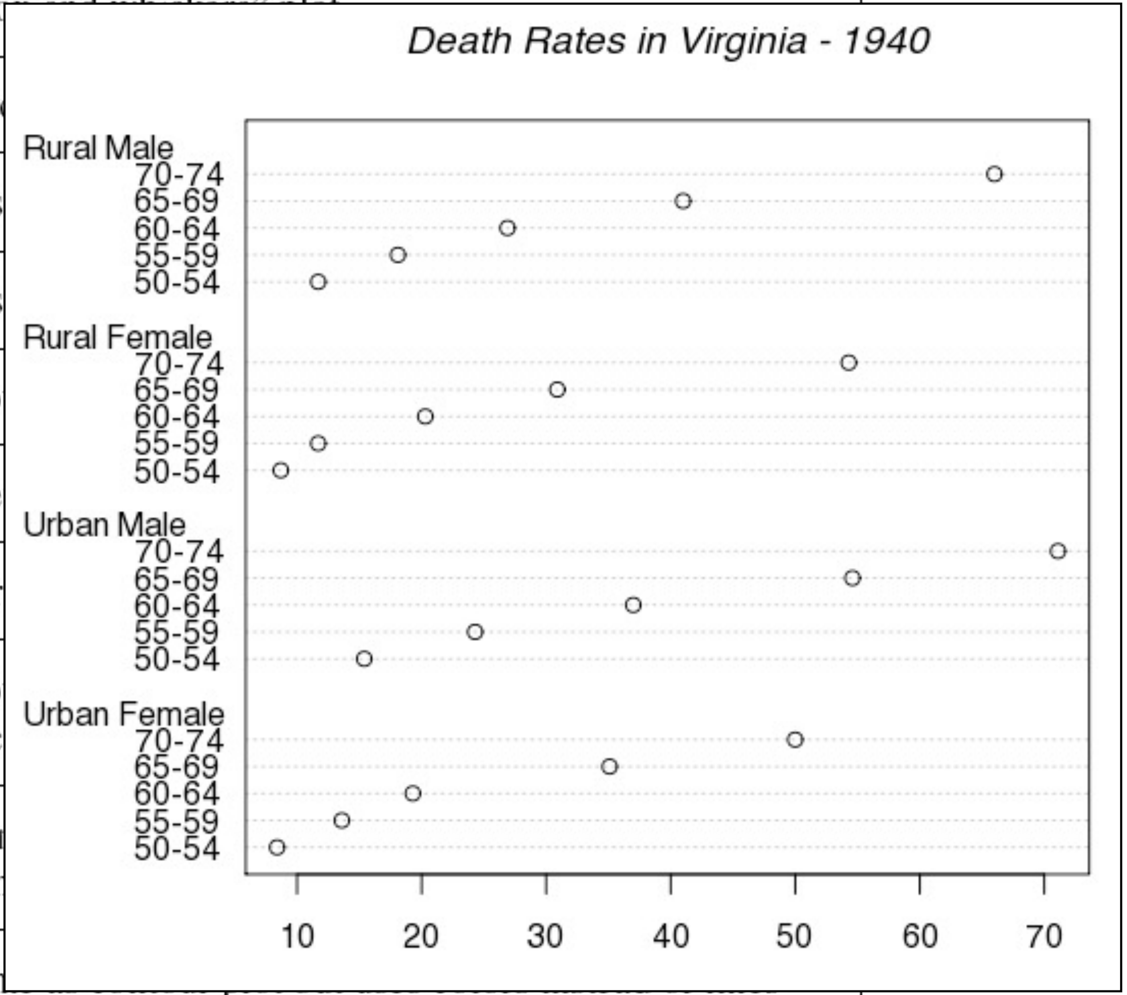
Function name	Plot produced
boxplot(x)	"Box and whiskers" plot
pie(x)	Circ
hist(x)	Hist
barplot(x)	Hist
stripchart(x)	Plots
dotchart(x)	Clev
pairs(x)	For a
plot.ts(x)	Plot vect
contour(x,y,z)	Cont dime
image(x,y,z)	Same as contour plot but uses colors instead of lines
persp(x,y,z)	3-d contour plot



stripchart: a good alternative to boxplot's when sample sizes are small.

Selected High-Level Plotting Functions

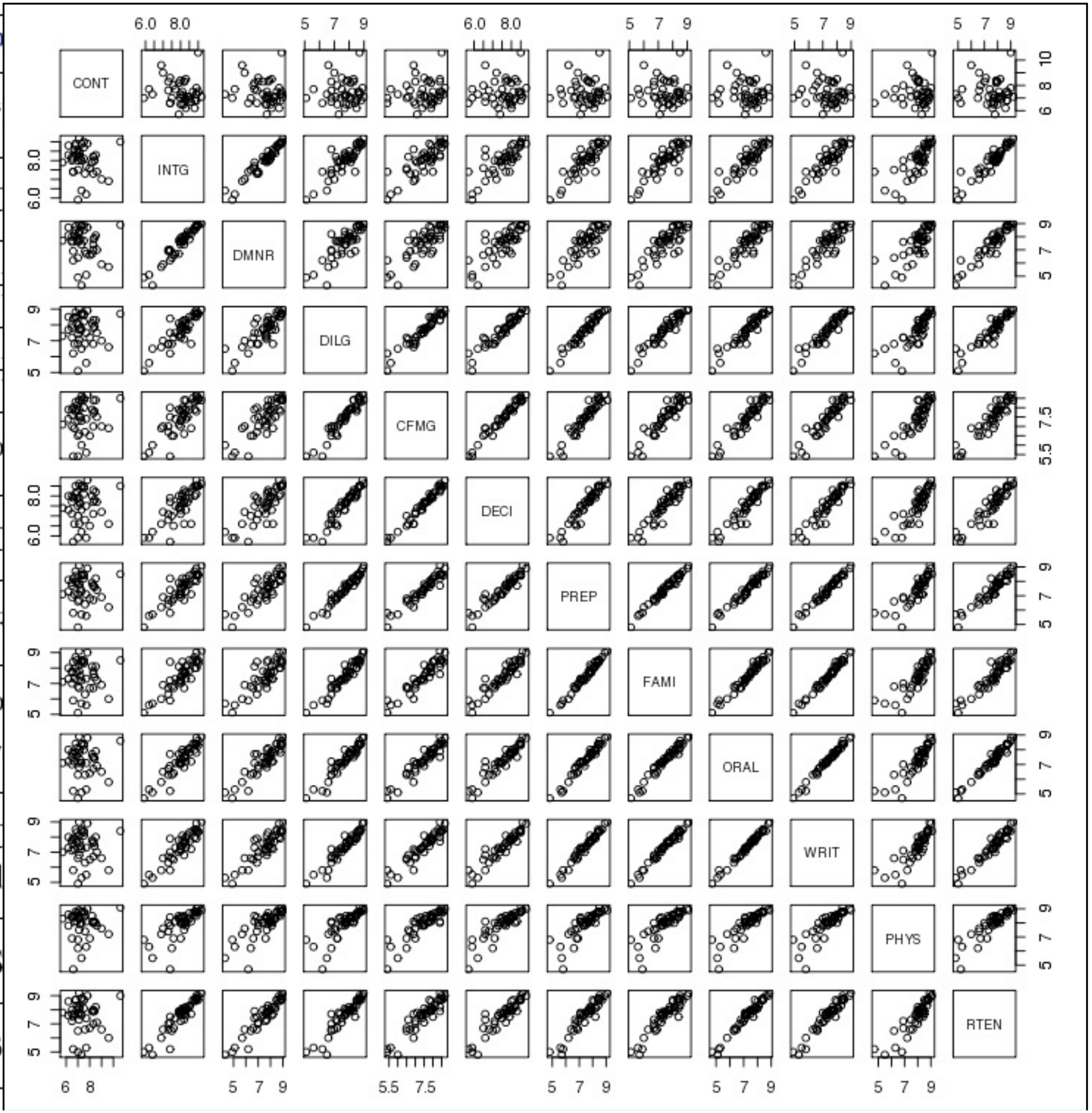
Function name	Plot produced
boxplot(x)	“Box and whiskers” plot
pie(x)	Cir
hist(x)	His
barplot(x)	His
stripchart(x)	Plo
dotchart(x)	Cle
pairs(x)	For
plot.ts(x)	Plo vec
contour(x,y,z)	Con dim
image(x,y,z)	San
persp(x,y,z)	3-d contour plot



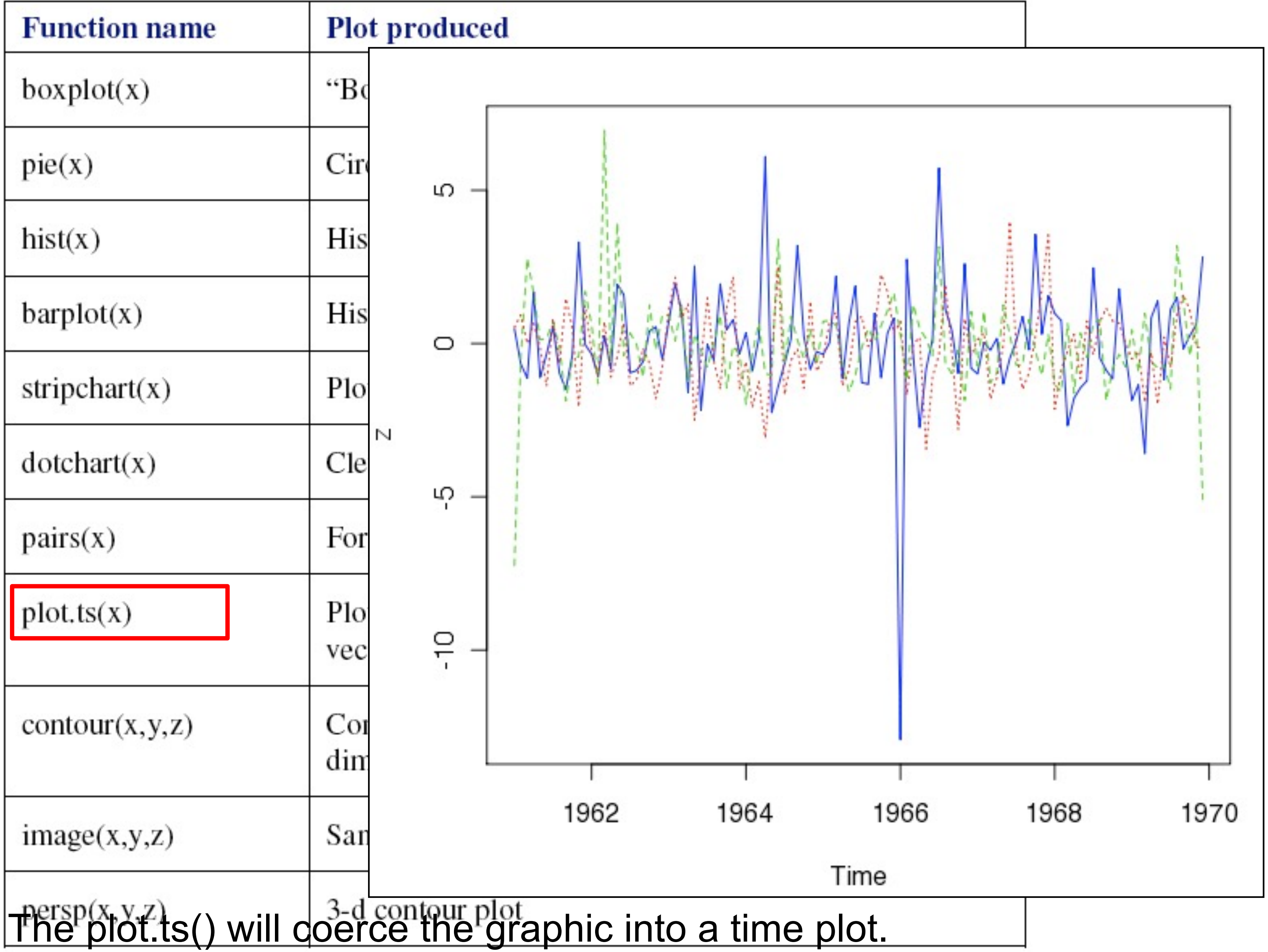
Dotcharts are a reasonable substitute for barplots.

Selected High-Level Plotting Functions

Function name	P
boxplot(x)	“
pie(x)	C
hist(x)	H
barplot(x)	H
stripchart(x)	P
dotchart(x)	C
pairs(x)	H
plot.ts(x)	P v
contour(x,y,z)	C d
image(x,y,z)	S
persp(x,y,z)	3



Selected High-Level Plotting Functions

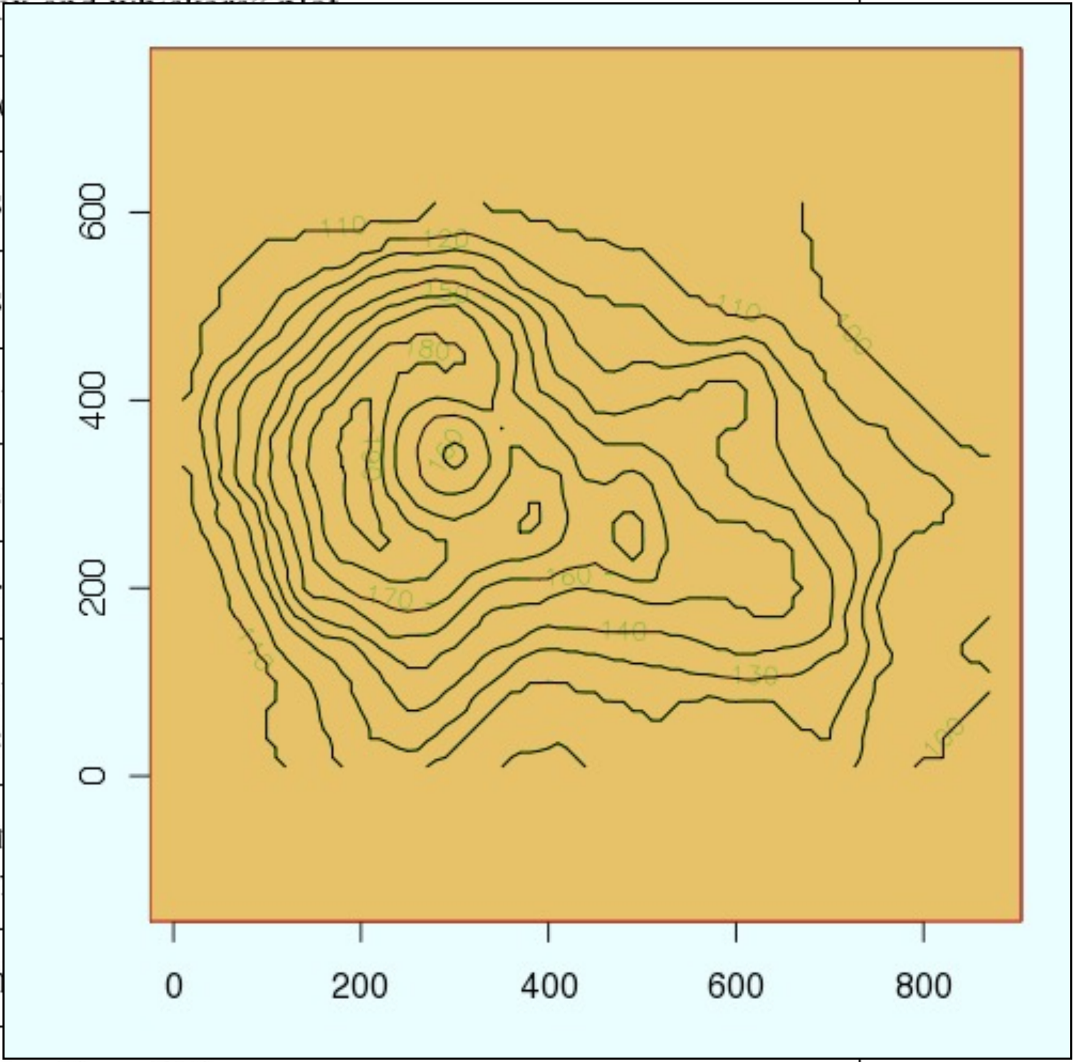


Function name	Plot produced
boxplot(x)	“Bo
pie(x)	Cir
hist(x)	His
barplot(x)	His
stripchart(x)	Plo
dotchart(x)	Cle
pairs(x)	For
plot.ts(x)	Plo vec
contour(x,y,z)	Co dim
image(x,y,z)	San
persp(x,y,z)	3-d contour plot

The plot.ts() will coerce the graphic into a time plot.

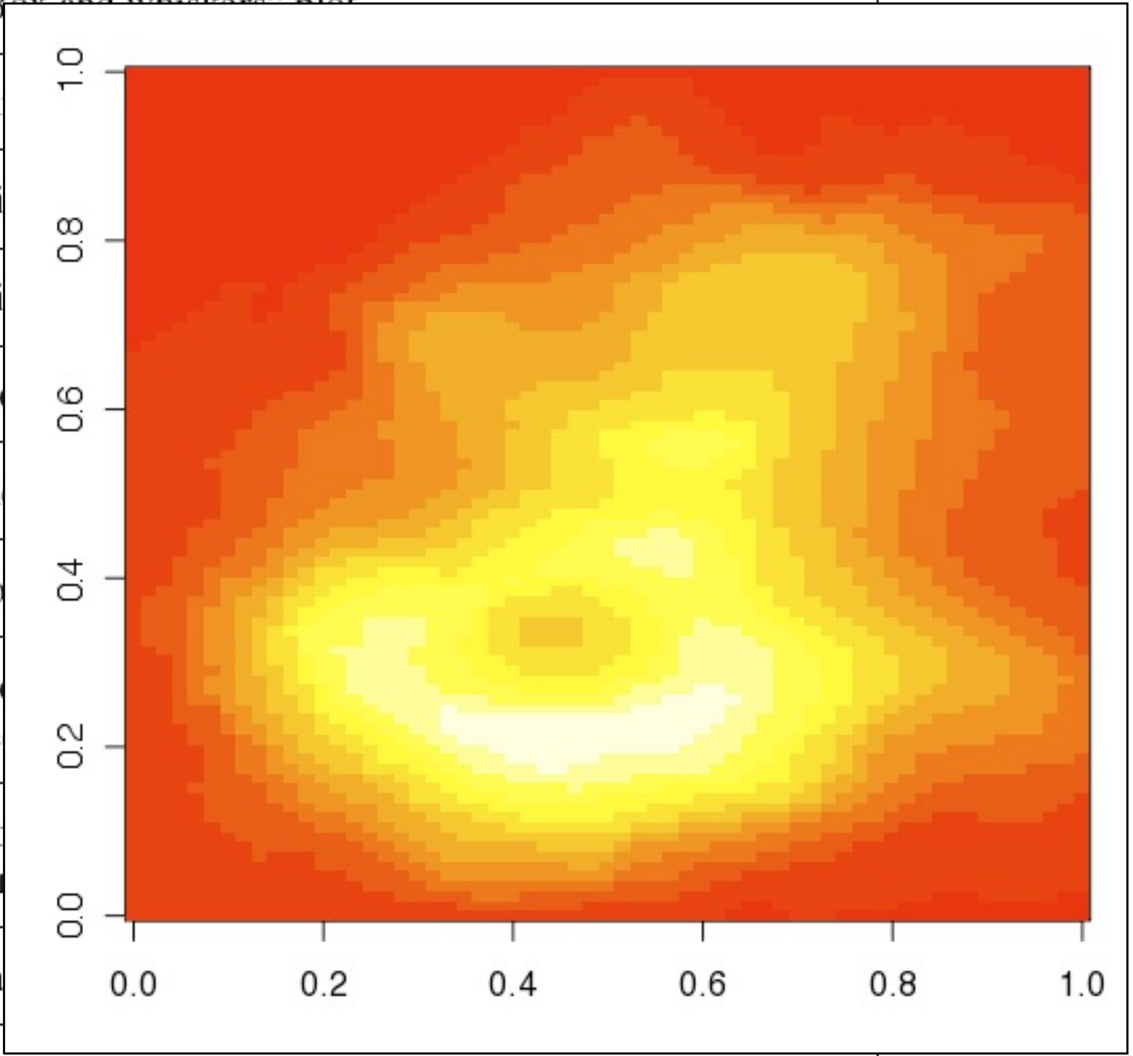
Selected High-Level Plotting Functions

Function name	Plot produced
boxplot(x)	“Bo
pie(x)	Cir
hist(x)	His
barplot(x)	His
stripchart(x)	Plo
dotchart(x)	Cle
pairs(x)	For
plot.ts(x)	Plo vec
contour(x,y,z)	Cor dir
image(x,y,z)	Sar
persp(x,y,z)	3-d contour plot



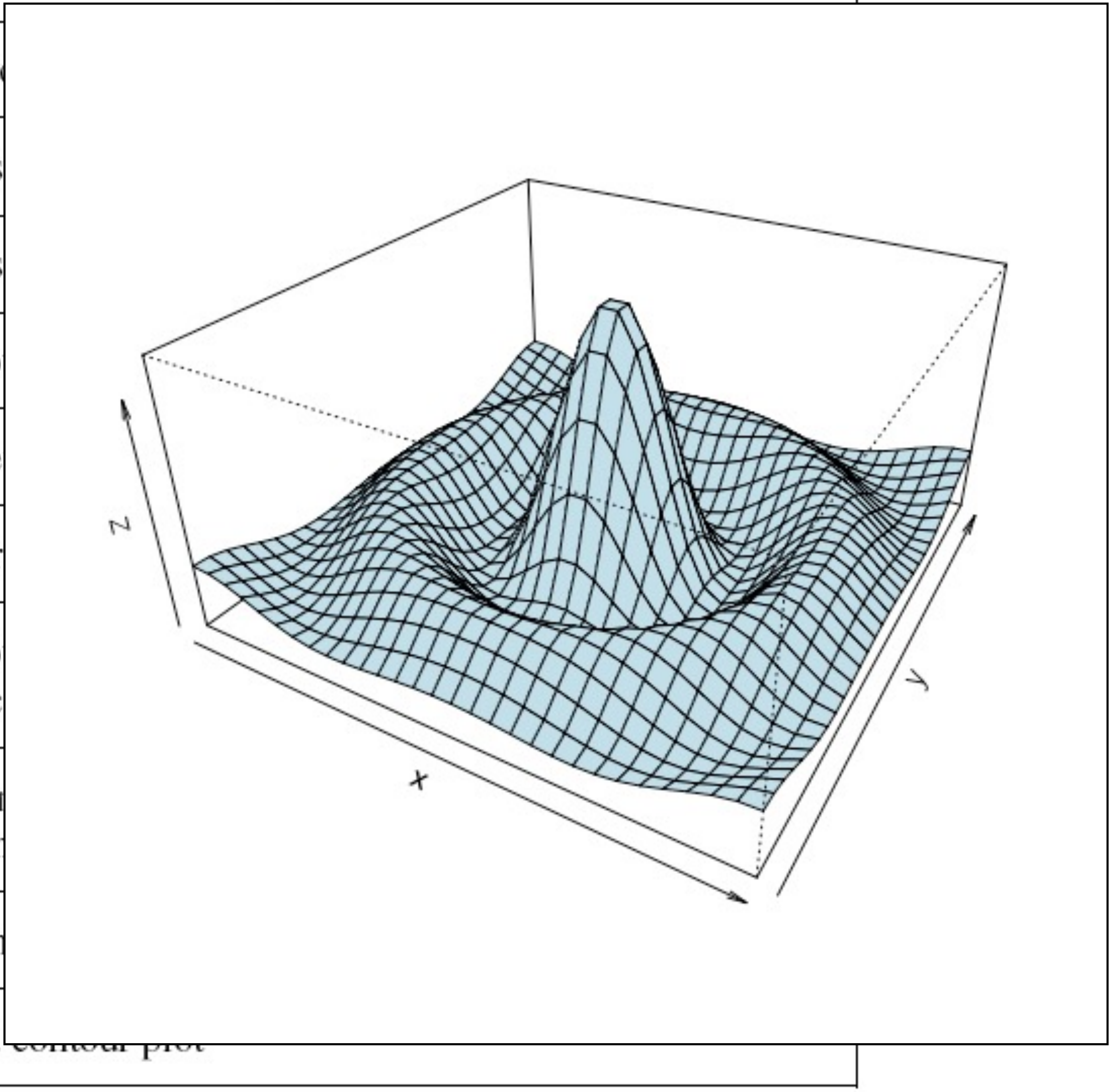
Selected High-Level Plotting Functions

Function name	Plot produced
boxplot(x)	“Box and whiskers” plot
pie(x)	Ci
hist(x)	Hi
barplot(x)	Hi
stripchart(x)	Pl
dotchart(x)	Cl
pairs(x)	Fo
plot.ts(x)	Pl ve
contour(x,y,z)	Co di
image(x,y,z)	Sa
persp(x,y,z)	3-d contour plot



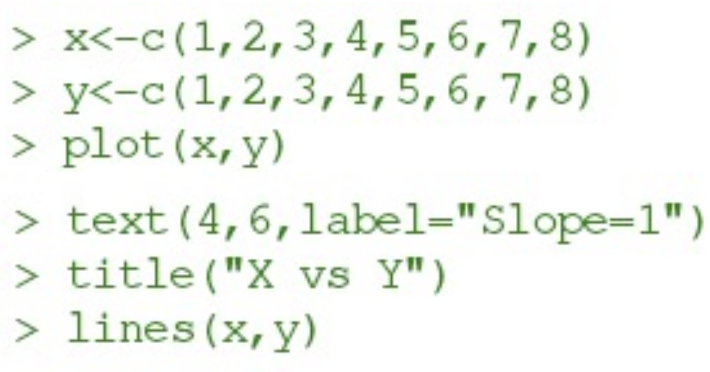
Selected High-Level Plotting Functions

Function name	Plot produced
boxplot(x)	"Box and whiskers" plot
pie(x)	Cir
hist(x)	His
barplot(x)	His
stripchart(x)	Plo
dotchart(x)	Cle
pairs(x)	For
plot.ts(x)	Plo vec
contour(x,y,z)	Cor dim
image(x,y,z)	San
persp(x,y,z)	3-d contour plot

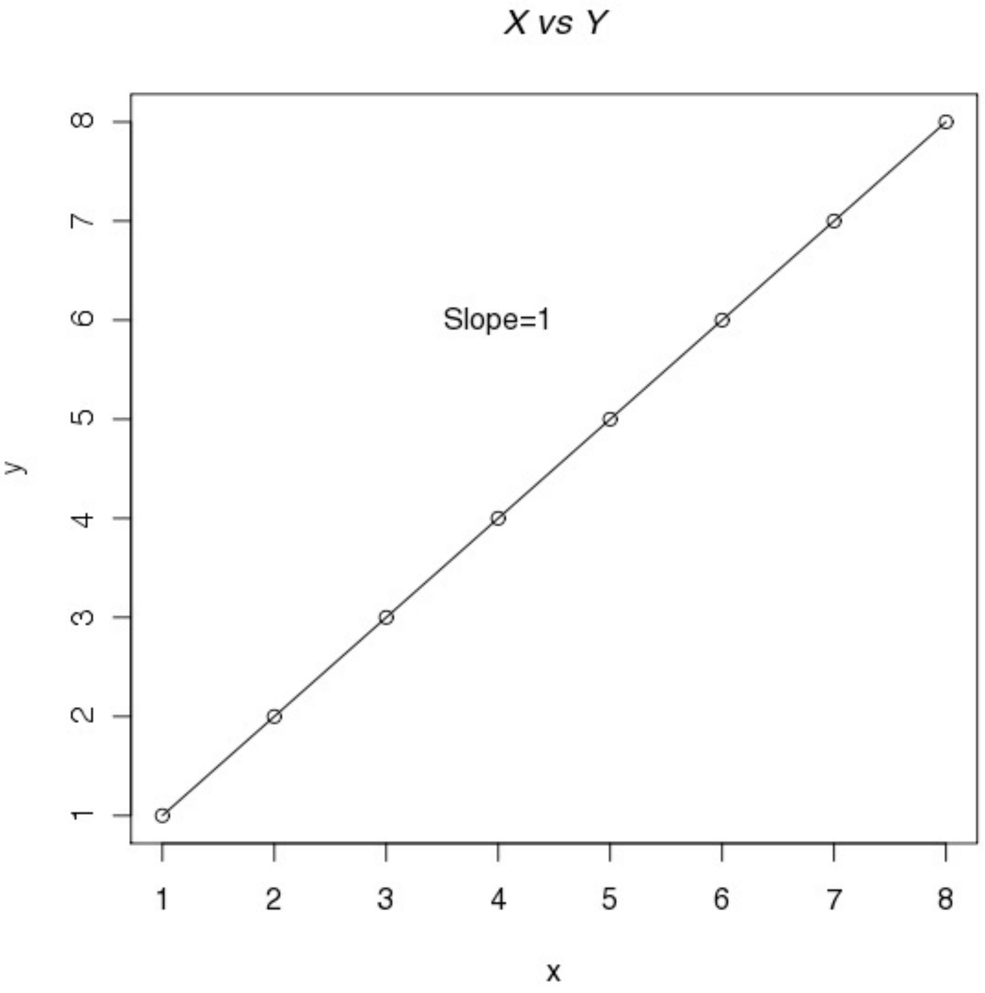


Low-Level Plotting Functions

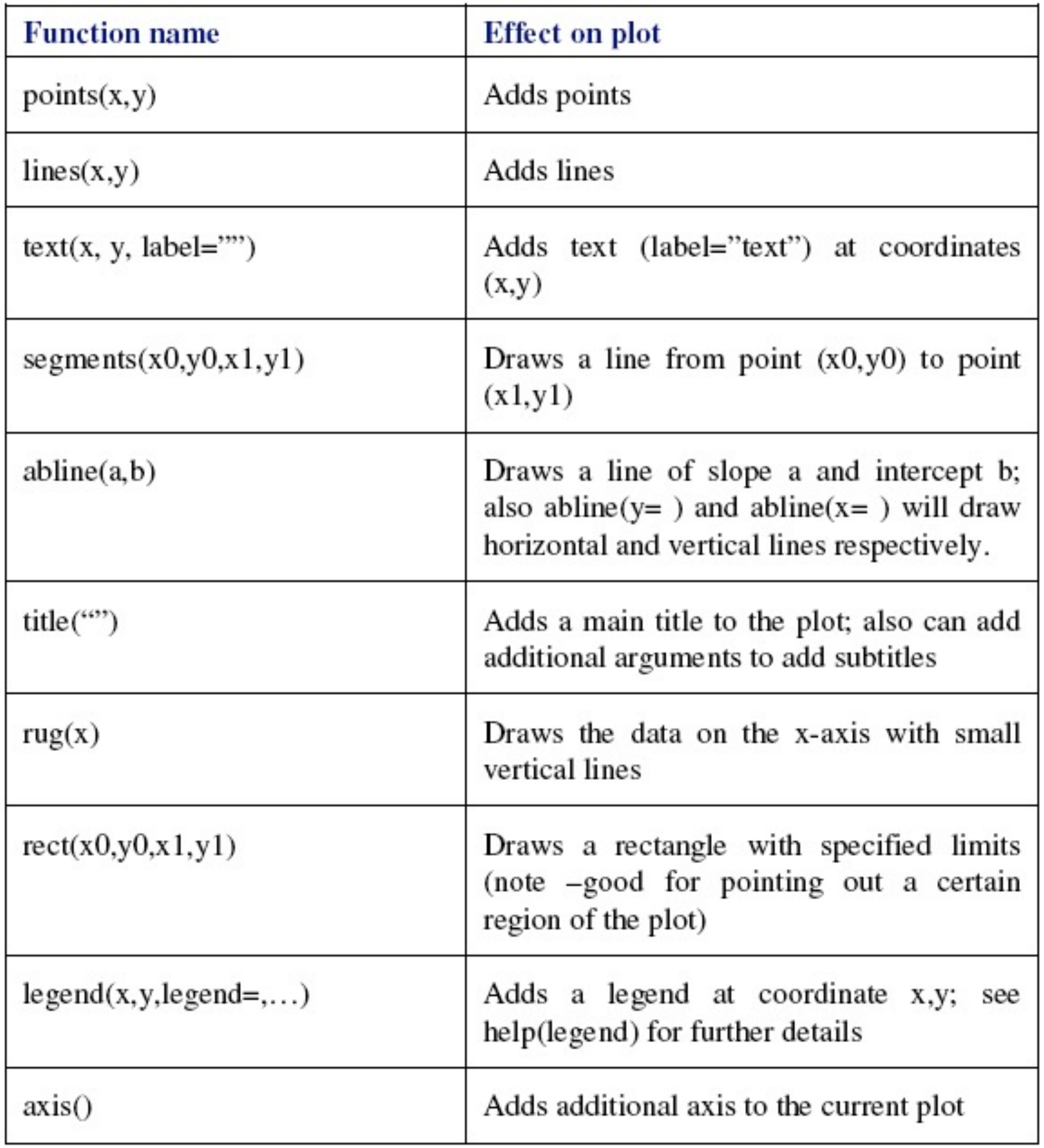
- There is some redundancy of low-level plotting functions with arguments of high-level plotting functions.



```
> x<-c(1,2,3,4,5,6,7,8)
> y<-c(1,2,3,4,5,6,7,8)
> plot(x,y)
> text(4,6,label="Slope=1")
> title("X vs Y")
> lines(x,y)
```



Selected Low-Level Plotting Functions

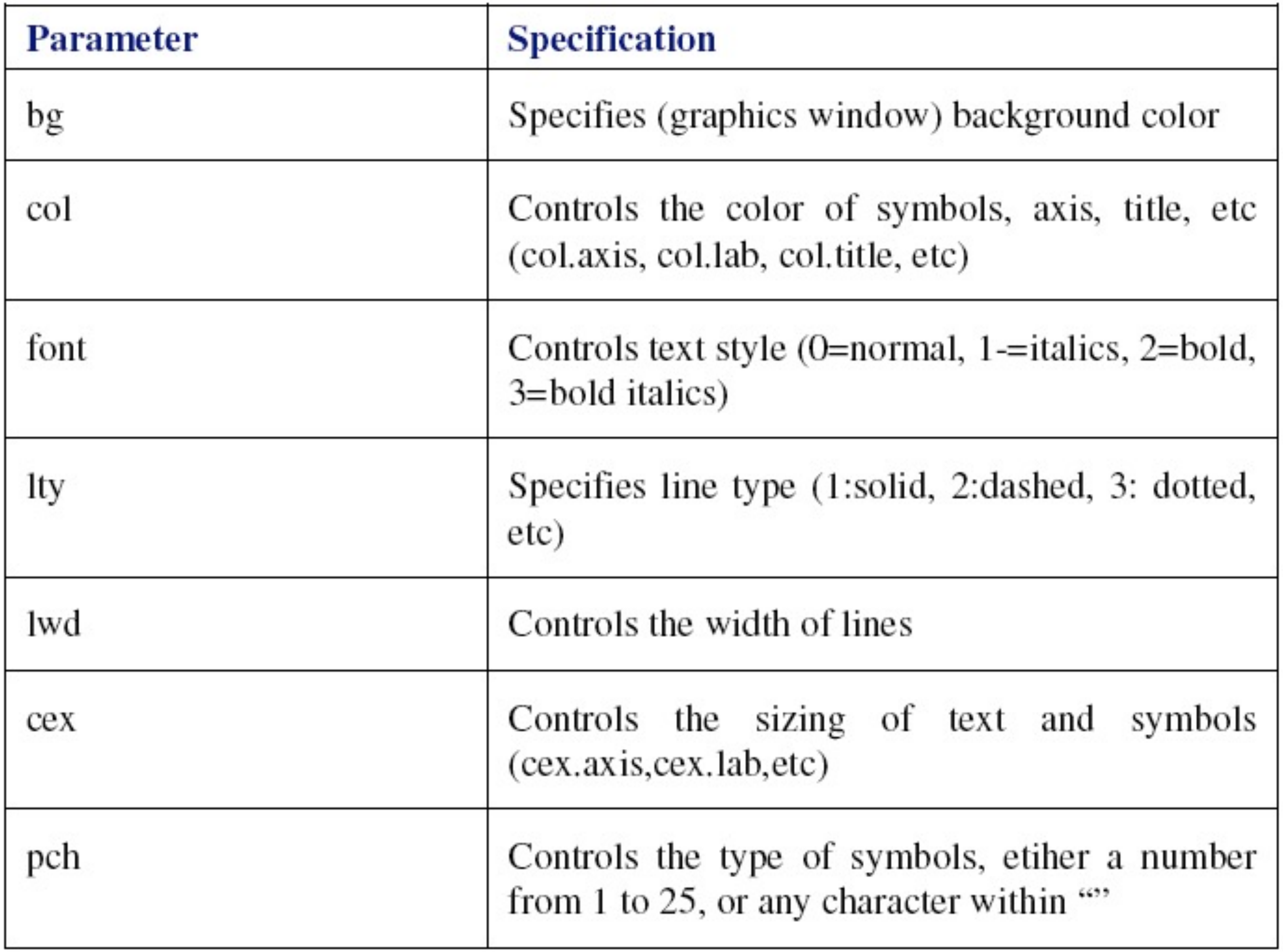


Function name	Effect on plot
points(x,y)	Adds points
lines(x,y)	Adds lines
text(x, y, label="")	Adds text (label="text") at coordinates (x,y)
segments(x0,y0,x1,y1)	Draws a line from point (x0,y0) to point (x1,y1)
abline(a,b)	Draws a line of slope a and intercept b; also abline(y=) and abline(x=) will draw horizontal and vertical lines respectively.
title("")	Adds a main title to the plot; also can add additional arguments to add subtitles
rug(x)	Draws the data on the x-axis with small vertical lines
rect(x0,y0,x1,y1)	Draws a rectangle with specified limits (note –good for pointing out a certain region of the plot)
legend(x,y,legend=,…)	Adds a legend at coordinate x,y; see help(legend) for further details
axis()	Adds additional axis to the current plot

Graphical Parameter Functions

- The par function
  - to access and modify settings of the graphics device
  - E.g. to split the graphics screen to display more than one plot on the graphic device at one time.
    - par(mfrow=c(rows,columns)) or par(mfcol=c(rows,columns))
    - mfrow draws the plots in row order (row 1 column 1, row 1 column 2, etc)
    - mfcol draws plots in column order (row 1 column 1, row 2 column 1)

Selected Graphical Parameters



Parameter	Specification
bg	Specifies (graphics window) background color
col	Controls the color of symbols, axis, title, etc (col.axis, col.lab, col.title, etc)
font	Controls text style (0=normal, 1-=italics, 2=bold, 3=bold italics)
lty	Specifies line type (1:solid, 2:dashed, 3: dotted, etc)
lwd	Controls the width of lines
cex	Controls the sizing of text and symbols (cex.axis,cex.lab,etc)
pch	Controls the type of symbols, etiher a number from 1 to 25, or any character within “”

Summary

- R Cookbook
  - Chapter 10. Graphics