Introduction to R

Biostatistics 615/815
Lecture 23

So far…
We have been working with C

- Strongly typed language
  - Variable and function types set explicitly
- Functional language
  - Programs are a collection of functions
- Rich set of program control options
  - `for, while, do … while, if` statements
- Computationally very efficient
  - If performance is critical, C code is usually the leanest

Today: The R Project

- Environment for statistical computing and graphics
  - Free software
- Associated with simple programming language
  - Similar to S and S-plus
- www.r-project.org

The R Project

- Versions of R exist of Windows, MacOS, Linux and various other Unix flavors
- R was originally written by Ross Ihaka and Robert Gentleman, at the University of Auckland
- It is an implementation of the S language, which was principally developed by John Chambers

On the shoulders of giants...

- In 1998, the Association for Computing Machinery gave John Chambers its Software Award. His citation reads:

- *“S has forever altered the way people analyze, visualize, and manipulate data ... It is an elegant, widely accepted, and enduring software system, with conceptual integrity.”*

Compiled C vs Interpreted R

- C requires a complete program to run
  - Program is translated into machine code
  - Can then be executed repeatedly

- R can run interactively
  - Statements converted to machine instructions as they are encountered
  - This is much more flexible, but also slower

R Function Libraries

- Implement many common statistical procedures
  - Random numbers, integration, optimization…
- Provide excellent graphics functionality
- A convenient starting point for many data analysis projects

Interactive R

- R defaults to an interactive mode
- A prompt “>” is presented to users
- Each input expression is evaluated…
- … and a result returned

R as a Calculator

```
> 1 + 1        # Simple Arithmetic
[1] 2
> 2 + 3 * 4    # Operator precedence
[1] 14
> 3 ^ 2        # Exponentiation
[1] 9
> exp(1)       # Basic mathematical functions are available
[1] 2.718282
> sqrt(10)
[1] 3.162278
> pi           # The constant pi is predefined
[1] 3.141593
> 2*pi*6378    # Circumference of earth at equator (in km)
[1] 40074.16
```

R as a Smart Calculator

```
> x <- 1                # Can define variables
> y <- 3                # using "<-" operator to set values
> z <- 4
> x * y * z
[1] 12

> X * Y * Z             # Variable names are case sensitive
Error: Object "X" not found

> This.Year <- 2004     # Variable names can include period
> This.Year
[1] 2004
```

R does a lot more!

- Definitely not just a calculator
- R can manipulate vectors, matrices and datasets
- R has many built-in statistical functions
- R produces excellent graphics
- R allows you to define your own functions

Vectors in R

- Created with
  - `c()` to concatenate elements or sub-vectors
  - `rep()` to repeat elements or patterns
  - `seq()` or `m:n` to generate sequences
- Most mathematical functions and operators can be applied to vectors
  - Without loops!
- Possible to select and edit groups of elements simultaneously

Defining Vectors

```
> rep(1,10)           # repeats the number 1, 10 times
[1] 1 1 1 1 1 1 1 1 1 1
> seq(2,6)            # sequence of integers between 2 and 6
[1] 2 3 4 5 6         # equivalent to 2:6
> seq(4,20,by=4)      # Every 4th integer between 4 and 20
[1]  4  8 12 16 20
> x <- c(2,0,0,4)     # Creates vector with elements 2,0,0,4
> y <- c(1,9,9,9)
> x + y               # Sums elements of two vectors
[1]  3  9  9 13
> x * 4               # Multiplies elements
[1]  8  0  0 16
> sqrt(x)                  # Function applies to each element
[1] 1.41 0.00 0.00 2.00 # Returns vector
```

Accessing Vector Elements

- Use the [ ] operator to select elements
- To select specific elements:
  - Use index or vector of indexes to identify them
- To exclude specific elements:
  - Negate index or vector of indexes
- Alternative:
  - Use vector of T and F values to select subset of elements

Accessing Vector Elements

```
> x <- c(2,0,0,4)
> x[1]        # Select the first element, equivalent to x[c(1)]
[1] 2
> x[-1]       # Exclude the first element
[1] 0 0 4
> x[1] <- 3 ; x
[1] 3 0 0 4
> x[-1] = 5 ; x
[1] 3 5 5 5
> y < 9       # Compares each element, returns result as vector
[1]  TRUE FALSE FALSE FALSE
> y[4] = 1
> y < 9
[1]  TRUE FALSE FALSE  TRUE
> y[y<9] = 2  # Edits elements marked as TRUE in index vector
> y
[1] 2 9 9 2
```

Functions in R

- Easy to create your own functions in R
- As tasks become complex, it is a good idea to organize code into functions that perform defined tasks
- In R, it is good practice to give default values to function arguments

Function definitions

```
name <- function(arg1, arg2, …)
    expression
```

- Arguments can be assigned default values:

  ```
  argname1 = expression
  ```

- Return value is the last evaluated expression or can be set explicitly with `return()`

Defining Functions

```
> square <- function(x = 10) x * x
> square()
[1] 100
> square(2)
[1] 4

> intsum <- function(from=1, to=10)
    {
    sum <- 0
    for (i in from:to)
      sum <- sum + i
    sum
    }
> intsum(from = 3)  # Evaluates sum from 3 to 10 …
[1] 52
> intsum(to = 3)    # Evaluates sum from 1 to 3 …
[1] 6
```

Example: Quick Union Function

```
QuickUnion <- function( N = 100, M = 100)
  {
  a <- seq(1, N)                        # initialize array

  for (dummy in seq(1,M))               # for each connection
     {
     p <- sample(N, 1)                  # sample random objects
     q <- sample(N, 1)

     # check if connected
     i = a[p]; while (a[i] != i) i <- a[i]
     j = a[q]; while (a[j] != j) j <- a[j]

     if (i == j)
        next

     a[i] = j                           # update connectivity array
     }
  }
```

Some notes on functions ...

- You can print the arguments for any function using `args()` command

```
> args(intsum)
function (from = 1, to = 10)
```

- You can print the contents of a function by typing only its name, without the `()`

- You can edit a function using

```
> my.func <- edit(my.old.func)
```

Debugging Functions

- Toggle debugging for a function with `debug()/undebug()` command

- With debugging enabled, R steps through function line by line
  - Use `print()` to inspect variables along the way
  - Press `<enter>` to proceed to next line

```
> debug(intsum)
> intsum(10)
```

Programming Constructs

- Grouped Expressions
- Control statements
  - `if … else …`

  - `for` loops
  - `repeat` loops
  - `while` loops

  - `next, break` statements

R Built-in and Library Functions

- R has an extensive function library

- Examples:
  - `read.table()` for loading tabular data
  - `rnorm()` for normally distributed deviates
  - `optim()` for the Nelder-Mead method
  - `optimize()` for the Golden section search
  - `integrate()` for adaptive numerical integration

`help()` and friends: The most useful R functions!

- R has a built-in help system with useful information and examples

- `help()` provides general help
- `help(plot)` will explain the plot function
- `help.search(“histogram”)` will search for topics that include the word histogram

- `example(plot)` will provide examples for the plot function

Computer Graphics

- Graphics are important for conveying important features of the data
- They can be used to examine
  - Marginal distributions
  - Relationships between variables
  - Summary of very large data
- Important complement to many statistical and computational techniques

Example Data

- The examples in this lecture will be based on a dataset with six variables:
  - Height (in cm)
  - Weight (in kg)
  - Waist Circumference (in cm)
  - Hip Circumference (in cm)
  - Systolic Blood Pressure
  - Diastolic Blood Pressure

The Data File

Height	Weight	Waist	Hip	bp.sys	bp.dia
172	72	87	94	127.5	80
166	91	109	107	172.5	100
174	80	95	101	123	64
176	79	93	100	117	76
166	55	70	94	100	60
163	76	96	99	160	87.5
154	84	98	118	130	80
165	90	108	101	139	80
155	66	80	96	120	70
146	59	77	96	112.5	75
164	62	76	93	130	47.5
159	59	76	96	109	69
163	69	96	99	155	100
143	73	97	117	137.5	85

. . .

Reading in the Data

```
> dataset <- read.table(“815data.txt", header = T)
> summary(dataset)

     Height          Weight            Waist
 Min.   :131.0   Min.   :  0.00   Min.   :  0.0
 1st Qu.:153.0   1st Qu.: 55.00   1st Qu.: 74.0
 Median :159.0   Median : 63.00   Median : 84.0
 Mean   :159.6   Mean   : 64.78   Mean   : 84.6
 3rd Qu.:166.0   3rd Qu.: 74.00   3rd Qu.: 94.0
 Max.   :196.0   Max.   :135.00   Max.   :134.0

. . .
```

Graphics in R

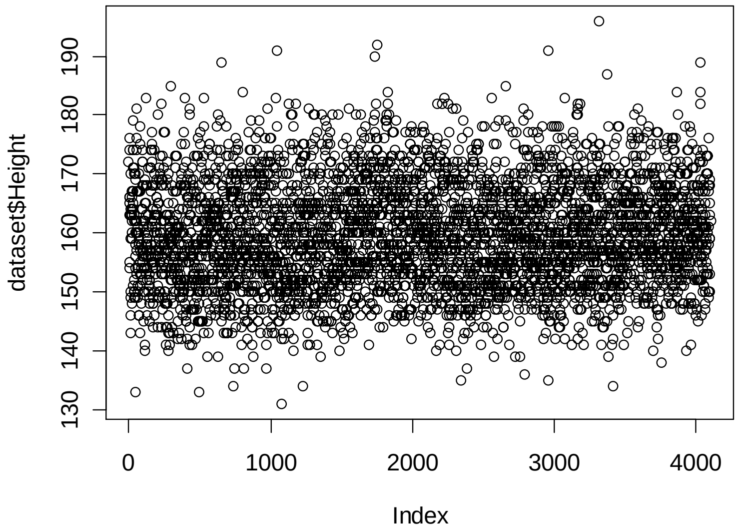
- `plot()` is the main graphing function
- Automatically produces simple plots for vectors, functions or data frames
- Many useful customization options…

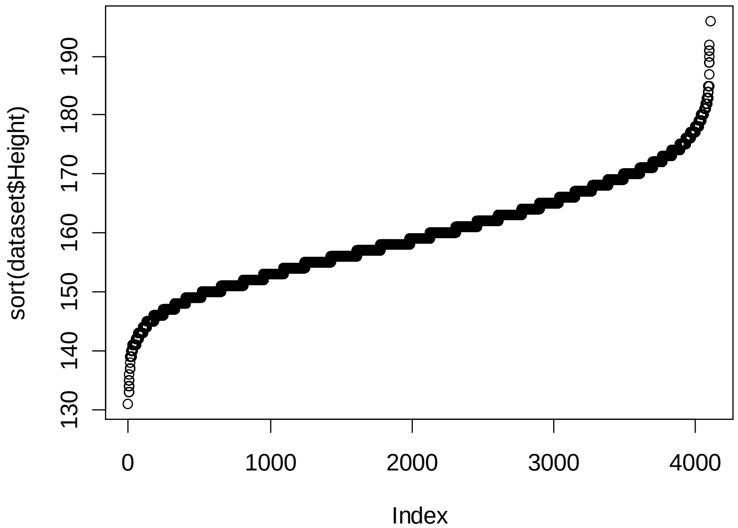
Plotting a Vector

- `plot(v)` will print the elements of the vector v according to their index

```
# Plot height for each observation
> plot(dataset$Height)
# Plot values against their ranks
> plot(sort(dataset$Height))
```

Plotting a Vector





`plot(dataset$Height)`

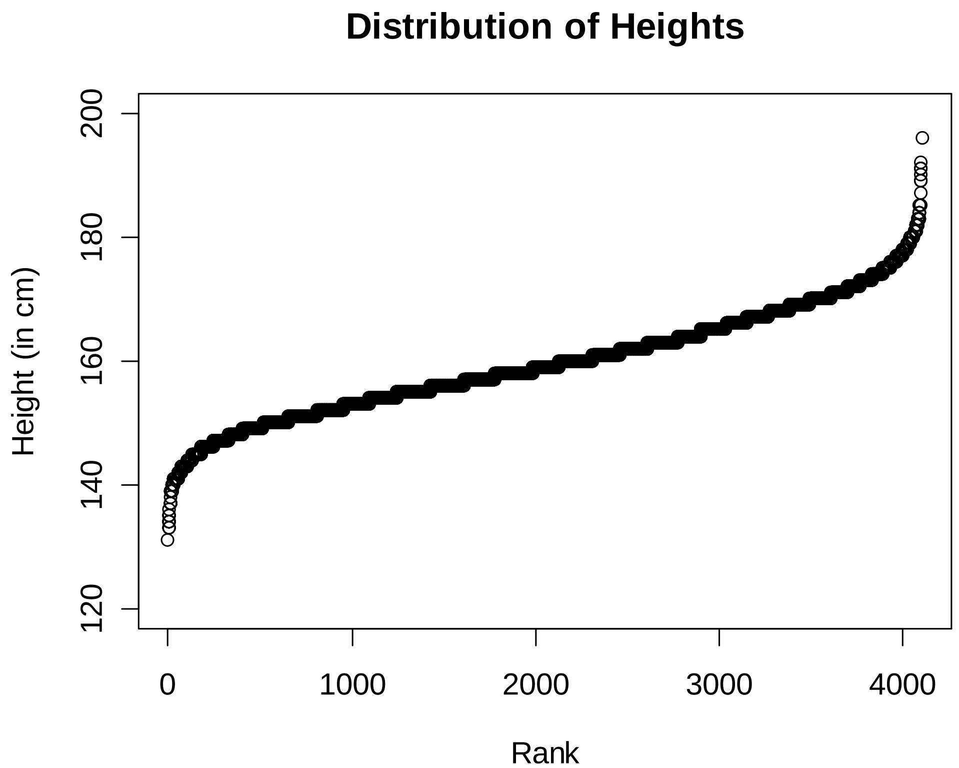
`plot(sort(dataset$Height))`

Common Parameters for `plot()`

- Specifying labels:
  - `main` – provides a title
  - `xlab` – label for the x axis
  - `ylab` – label for the y axis

- Specifying range limits
  - `ylim` – 2-element vector gives range for x axis
  - `xlim` – 2-element vector gives range for y axis

A Properly Labeled Plot

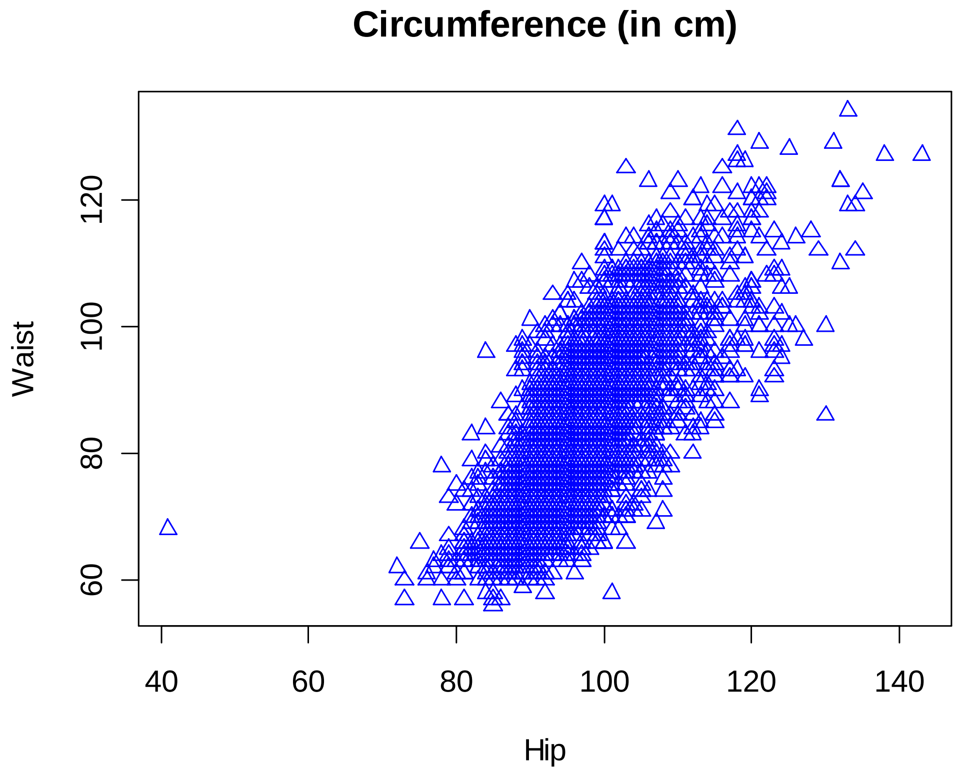



```
plot(sort(dataset$Height), ylim = c(120,200),
ylab = "Height (in cm)", xlab = "Rank", main = "Distribution of Heights")
```

Plotting Two Vectors

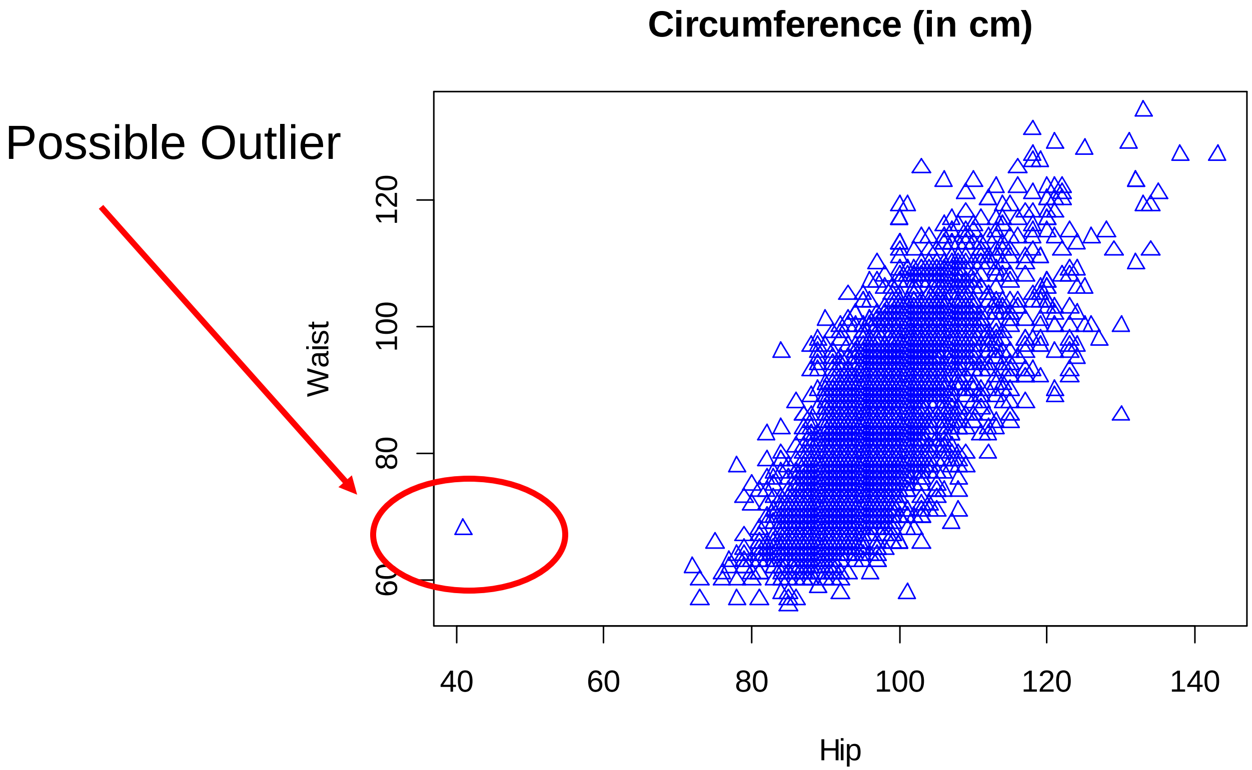
- `plot()` can pair elements from 2 vectors to produce x-y coordinates
- `plot()` and `pairs()` can also produce composite plots that pair all the variables in a data frame.

Plotting Two Vectors



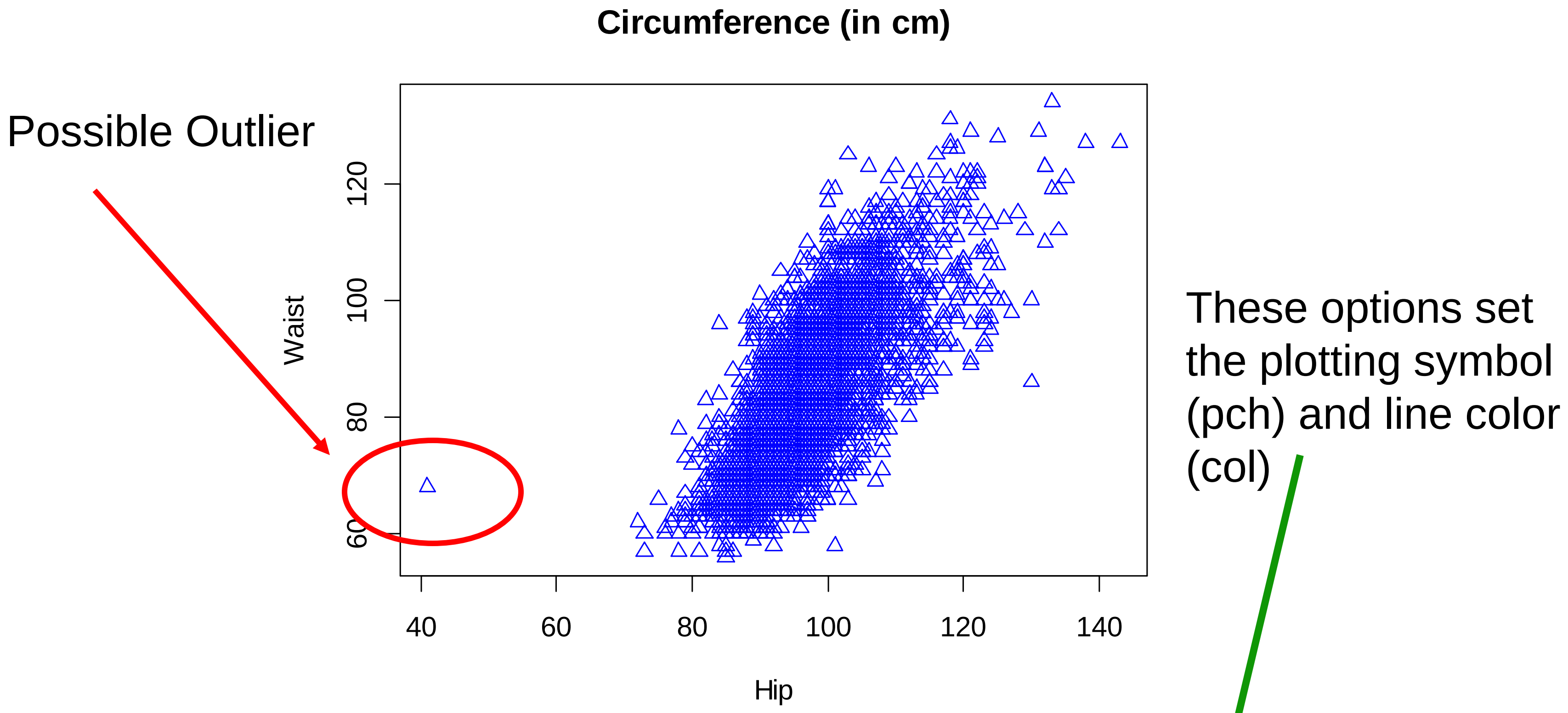

```
plot(dataset$Hip, dataset$Waist,
xlab = "Hip", ylab = "Waist",
main = "Circumference (in cm)", pch = 2, col = "blue")
```

Plotting Two Vectors



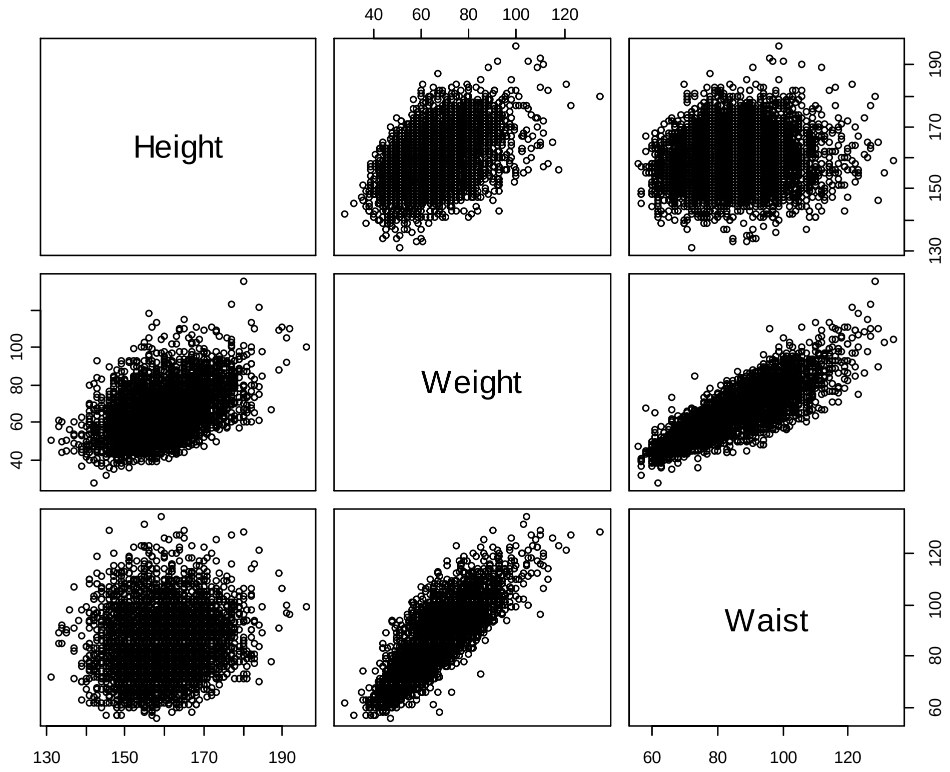

```
plot(dataset$Hip, dataset$Waist,
 xlab = "Hip", ylab = "Waist",
main = "Circumference (in cm)", pch = 2, col = "blue")
```

Plotting Two Vectors



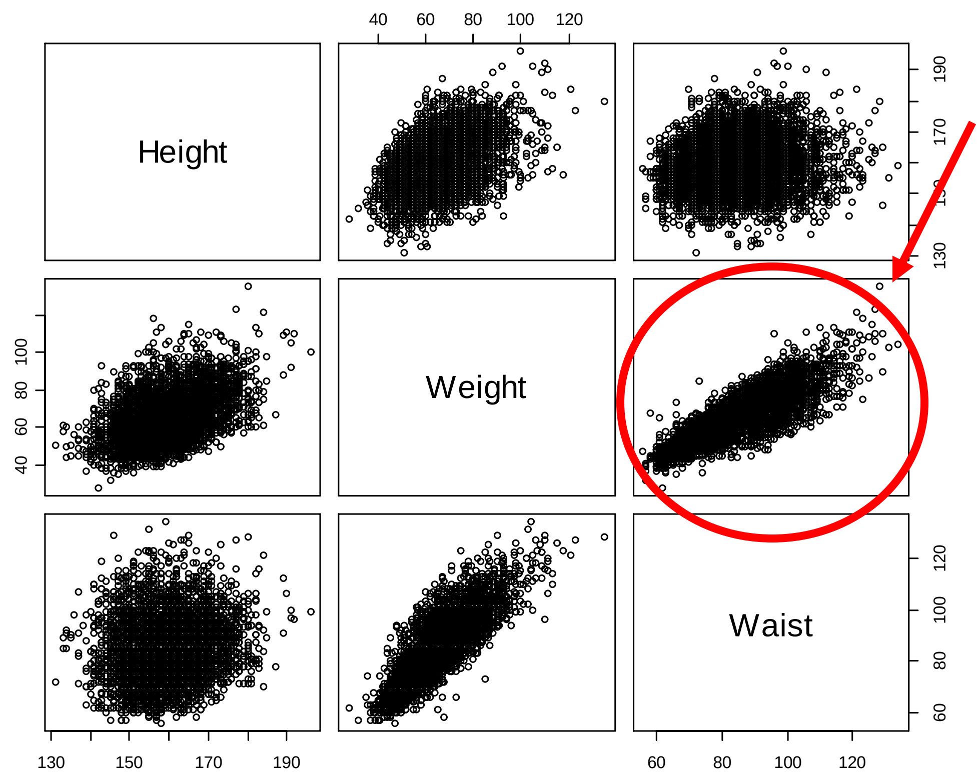

```
plot(dataset$Hip, dataset$Waist,
xlab = "Hip", ylab = "Waist",
main = "Circumference (in cm)", pch = 2, col = "blue")
```

Plotting Contents of a Dataset




```
plot(dataset[-c(4,5,6)])
```

Plotting Contents of a Dataset



Weight and Waist Circumference Appear Strongly Correlated

You could check this with the `cor()` function.

```
plot(dataset[-c(4,5,6)])
```

Histograms

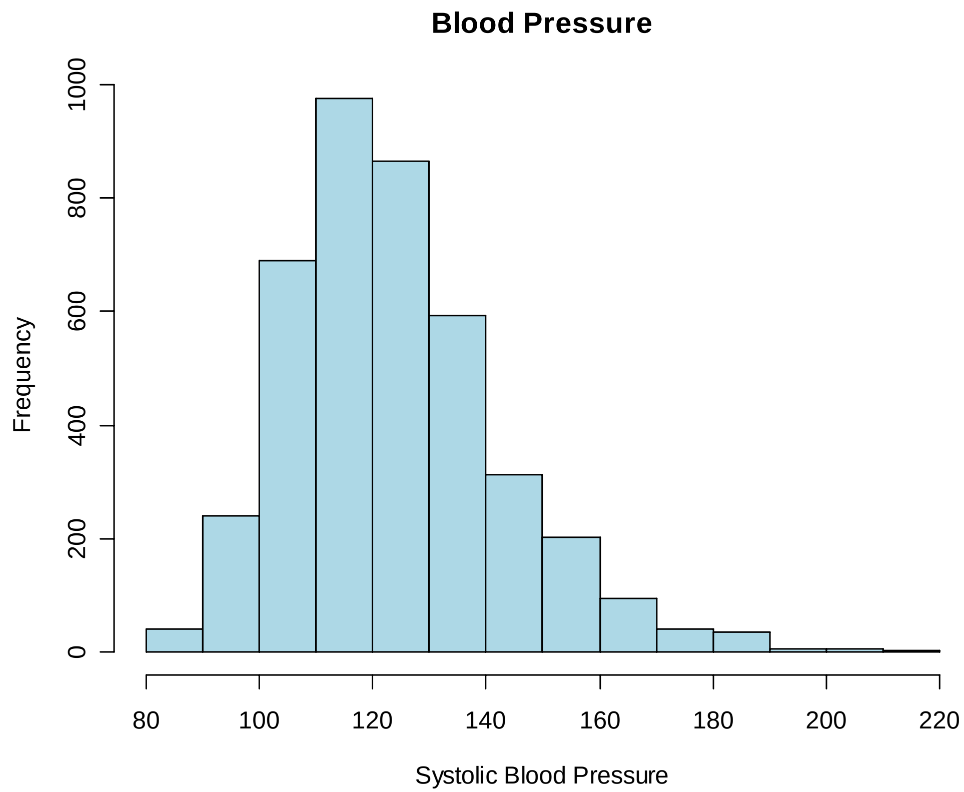
- Generated by the `hist()` function

- The parameter `breaks` is key
  - Specifies the number of categories to plot

  or
  - Specifies the breakpoints for each category

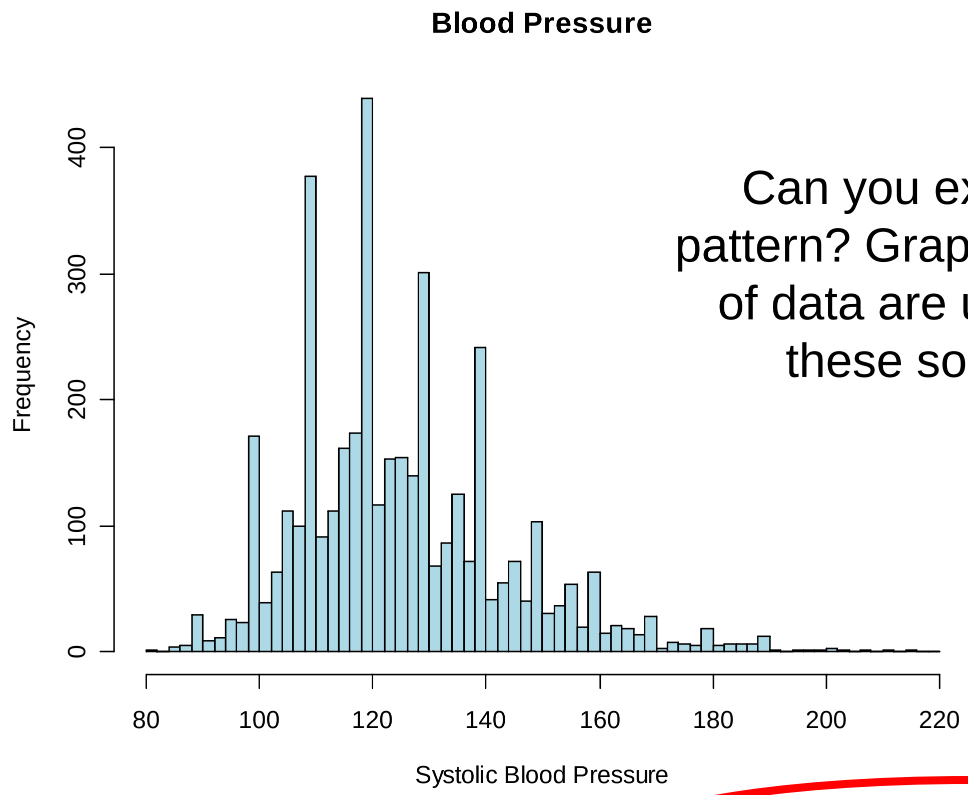
- The `xlab, ylab, xlim, ylim` options work as expected

Histogram




```
hist(dataset$bp.sys, col = "lightblue",
xlab = "Systolic Blood Pressure", main = "Blood Pressure")
```

Histogram , Changed breaks



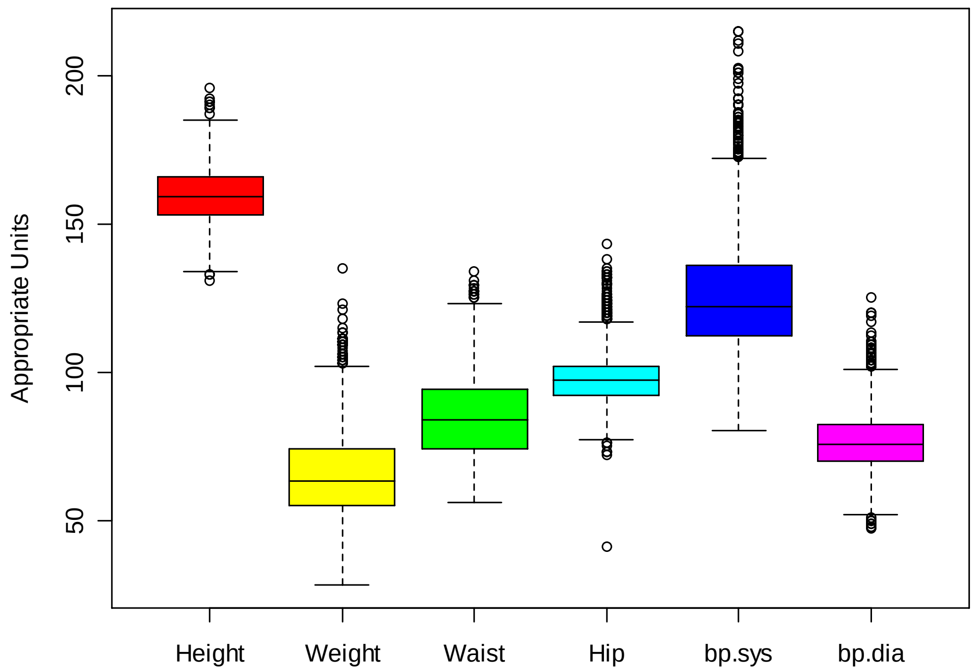
Can you explain the peculiar pattern? Graphical representations of data are useful at identifying these sorts of artifacts…

```
hist(dataset$bp.sys, col = "lightblue", breaks = seq(80,220,by=2),
    xlab = "Systolic Blood Pressure", main = "Blood Pressure")
```

Boxplots

- Generated by the boxplot() function
- Draws plot summarizing
  - Median
  - Quartiles (Q1, Q3)
  - Outliers – by default, observations more than 1.5 * (Q1 – Q3) distant from nearest quartile

A Simple Boxplot

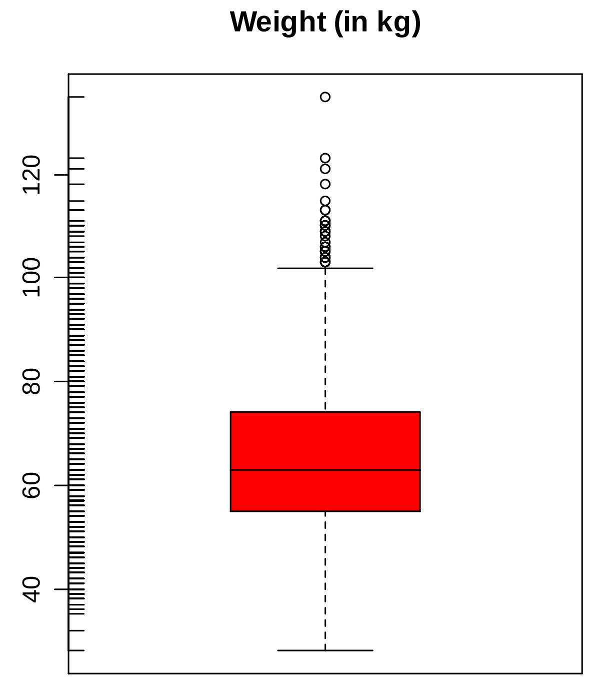



```
boxplot(dataset, col = rainbow(6), ylab = "Appropriate Units")
```

Adding Individual Observations

- `rug()` can add a tick for each observation to the side of a boxplot() and other plots.

- The `side` parameter specifies where tickmarks are drawn




```
> boxplot(dataset$Weight,
          main = "Weight (in kg)",
          col = "red")
> rug(dataset$Weight, side = 2)
```

Customizing Plots

- R provides a series of functions for adding text, lines and points to a plot

- We will illustrate some useful ones, but look at `demo(graphics)` for more examples

Drawing on a plot

- To add additional data use
  - `points(x,y)`
  - `lines(x,y)`

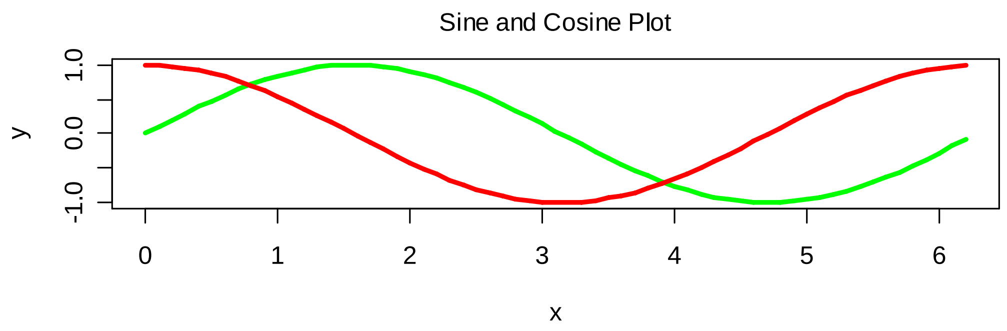
- For freehand drawing use
  - `polygon()`
  - `rect()`

Text Drawing

- Two commonly used functions:
  - `text()` – writes inside the plot region, could be used to label datapoints
  - `mtext()` – writes on the margins, can be used to add multiline legends
- These two functions can print mathematical expressions created with `expression()`

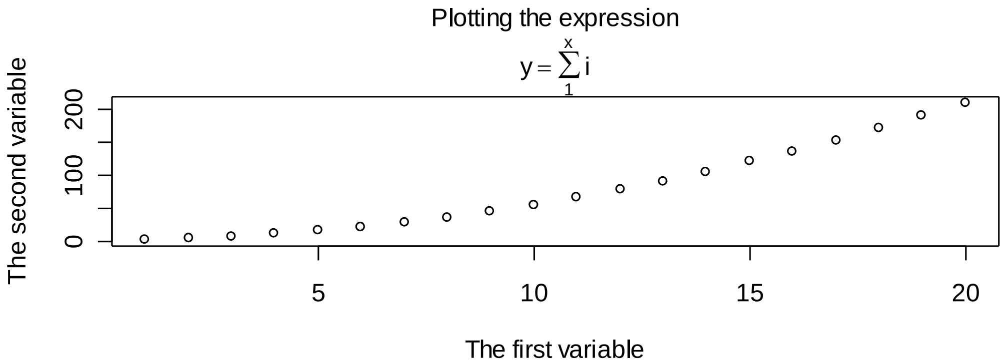
Plotting Two Data Series

```
> x <- seq(0,2*pi, by = 0.1)
> y <- sin(x)
> y1 <- cos(x)
> plot(x,y, col = "green", type = "l", lwd = 3)
> lines(x,y1, col = "red", lwd = 3)
> mtext("Sine and Cosine Plot", side = 3, line = 1)
```

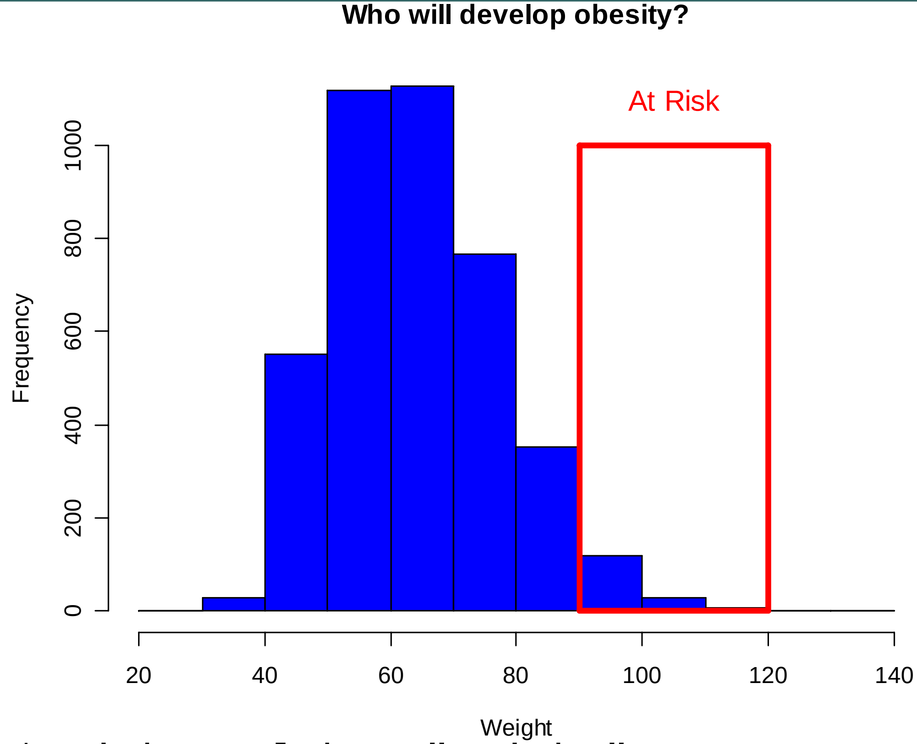


Printing on Margins, Using Symbolic Expressions

```
> f <- function(x) x * (x + 1) / 2
> x <- 1:20
> y <- f(x)
> plot(x, y, xlab = "", ylab = "")
> mtext("Plotting the expression", side = 3, line = 2.5)
> mtext(expression(y == sum(i,1,x,i)), side = 3, line = 0)
> mtext("The first variable", side = 1, line = 3)
> mtext("The second variable", side = 2, line = 3)
```



Adding a Label Inside a Plot




```
> hist(dataset$Weight, xlab = "Weight",
       main = "Who will develop obesity?", col = "blue")
> rect(90, 0, 120, 1000, border = "red", lwd = 4)
> text(105, 1100, "At Risk", col = "red", cex = 1.25)
```

Symbolic Math
Example from demo(plotmath)

Big Operators	
sum(x[i], i = 1, n)	$\sum_{1}^{n} x_i$
prod(plain(P)(X == x), x)	$\prod_{x} \mathrm{P}(X = x)$
integral(f(x) * dx, a, b)	$\int_a^b f(x) dx$
union(A[i], i == 1, n)	$\bigcup_{i=1}^{n} A_i$
intersect(A[i], i == 1, n)	$\bigcap_{i=1}^{n} A_i$
lim(f(x), x %->% 0)	$\lim_{x \to 0} f(x)$
min(g(x), x >= 0)	$\min_{x \geq 0} g(x)$
inf(S)	$\inf S$
sup(S)	$\sup S$

Further Customization

- The `par()` function can change defaults for graphics parameters, affecting subsequent calls to `plot()` and friends.

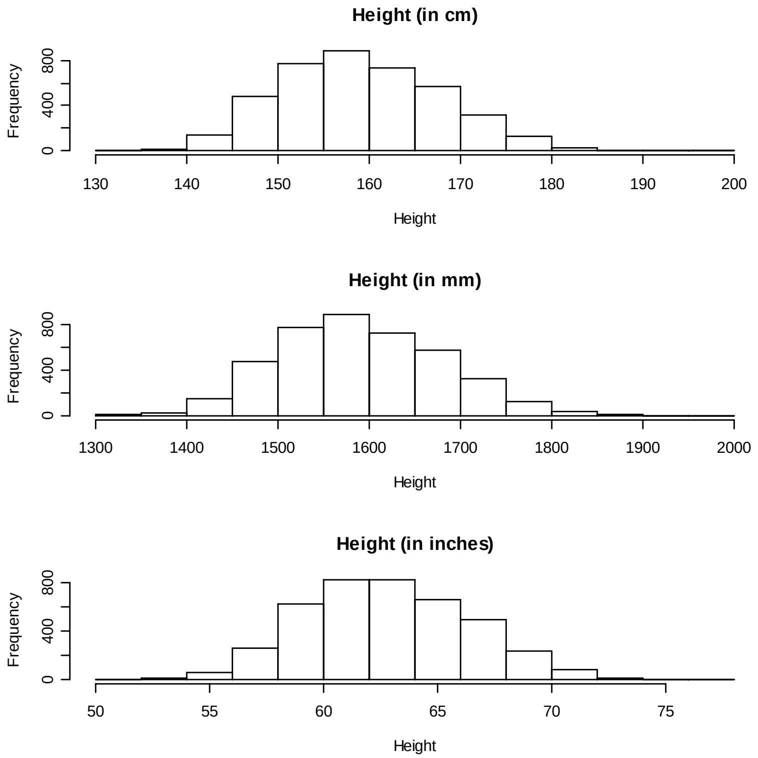
- Parameters include:
  - `cex, mex` – text character and margin size
  - `pch` – plotting character
  - `xlog, ylog` – to select logarithmic axis scaling

Multiple Plots on A Page

- Set the `mfrow` or `mfcol` options
  - Take 2 dimensional vector as an argument
  - The first value specifies the number of rows
  - The second specifies the number of columns
- The 2 options differ in the order individual plots are printed

Multiple Plots

```
> par(mfcol = c(3,1))
> hist(dataset$Height,
  breaks = 10,
  main = "Height (in cm)",
  xlab = "Height")
> hist(dataset$Height * 10,
  breaks = 10,
  main = "Height (in mm)",
  xlab = "Height")
> hist(dataset$Height / 2.54,
  breaks = 10,
  main = "Height (in inches)",
  xlab = "Height")
```



Outputting R Plots

- R usually generates output to the screen
- In Windows and the Mac, you can point and click on a graph to copy it to the clipboard
- However, R can also save its graphics output in a file that you can distribute or include in a document prepared with Word or LATEX

Selecting a Graphics Device

- To redirect graphics output, first select a device:
  - `pdf()` – high quality, portable format
  - `postscript()` – high quality format
  - `png()` – low quality, but suitable for the web

- After you generate your graphics, simply close the device
  - `dev.off()`

Example of Output Redirection

```
> x <- runif(100)
> y <- runif(100) * 0.5 + x * 0.5

# This graph is plotted on the screen
> plot(x, y, ylab = “This is a simple graph”)

# This graph is plotted to the PDF file
> pdf(“my_graph.pdf”)
> plot(x, y, ylab = “This is a simple graph”)
> dev.close()

# Where does this one go?
> plot(x, y, ylab = “This is a simple graph”)
```

Today

- Introduction to Graphics in R
- Examples of commonly used graphics functions
- Common options for customizing graphs

Learning More About R

- Excellent documentation is available at www.r-project.org
  - “An Introduction to R”
    by Venables and Smith
    in the Documentation Section
- Good book to browse is “Data Analysis and Graphics in R” by Maindonald and Braun