Introduction to R

See DAAG Chapter 1

Short R session

- R is available on lab computers, and for free from CRAN (see course syllabus)
- After starting the R gui, you will be confronted by the command prompt

R as a calculator

- Try the following:

```
> 2+2
> 2*3*4*5
> sqrt(10)
> pi
> 2*pi*6378   # circumference of Earth
```

- You can spread commands across lines, as well as put multiple commands on one line (separated by a semi-colon)
- You can put comments following a #

Entering data at the command line

```
Year <- c(1800, 1850, 1900, 1950, 2000)
Carbon <- c(8, 54, 534, 1630, 6611)
# Now plot Carbon as a function of Year
plot( Carbon ~ Year, pch = 19 )
# Collect Year and Carbon into a data.frame
fossilfuel <- data.frame( year=Year, carbon=Carbon )
print( fossilfuel )
rm( Year, Carbon ) # removes old objects
# Recreate plot from data.frame
plot( carbon ~ year, data = fossilfuel, pch=19)
# Three ways to extract a column from a data.frame
fossilfuel[,2]; fossilfuel[,"carbon"];
fossilfuel$carbon
```

Working directory

- You can get the current working directory with `getwd()`
- You can set the current working directory with `setwd()` or by using the Rgui menus
- When you quit R (by using the `q()` function or closing the Rgui), you will be asked whether to save your workspace to resume your session later
- You can save your command history also.

Installing packages and getting help

- Packages can be installed using the Rgui menus or with the `install.packages()` function.
- If want help on a specific function (e.g. plot), use `?plot` or `help(plot)`
- If you don't know the name of the function, you can use `apropos("sort")` and `help.search("sort")`
- If you are still stuck, `help.start()` and `RSiteSearch()`

Data sources

```
help(read.table)
fossilfuel <- read.table(“fuel.txt”,
                         header = TRUE)
fossilfuel <- read.table(“fuel.csv”,
                         header = TRUE)
data(package=“datasets”)
help(load)
```

R objects

```
c(6,2,9,-1,3,-7)     # numeric vector
c(T,F,F,T,T,T)       # logical vector
c("blue","red","orange") # character
factor(c("blue","red","orange"))
                      # factor vector

# missing values
vec1 <- c(1,4,NA); vec2 <- c(4,NA,-7)
c(vec1,vec2)          # 1 4 NA 4 NA -7
0/0                   # NaN
1/0                   # Inf
```

Comparing and extracting elements

```
x <- c(-1,4,9,0)
x > 0             # FALSE TRUE TRUE FALSE
x != 0            # TRUE TRUE TRUE FALSE
x == 0            # FALSE FALSE FALSE TRUE
?Comparison; ?Logic; ?Syntax

x[2]              # 4
x[c(2,4)]         # 4 0
x[-c(2,4)]        # -1 9
x[c(T,T,F,F)]     # -1 4
x[x > 0]          # 4 9
```

Generating patterned vectors

```
?seq
4:10              # 4 5 6 7 8 9 10
10:4              # 10 9 8 7 6 5 4
seq(from=2, to=8, by=2) # 2 4 6 8
?rep
rep( 1:3, 3 ) # 1 2 3 1 2 3 1 2 3
rep( “blue”, 2 ) # “blue” “blue”
```

Factors

```
# Factors can be tricky
gender <- c(rep(“female”,4), rep(“male”,4))
levels( gender )     # NULL
gender <- factor( gender )
levels( gender )     # “female” “male”
str( gender )
gender <- factor( gender, levels =
                    c(“male”,”female”) )
levels( gender )     # “male” “female”
```

Data frames and matrices

- A data.frame is a *list* of *vectors* that all have the same length
- The columns of a data.frame can have different modes (numeric, factor, logical, character). You can check the mode of each column using `class()`
- A matrix is a 2-dimensional vector – all entries have the same mode
- `rownames(); colnames(); nrow(); ncol()`
- You can use [] indexing for data.frames
  - `my.df[rows.vector, cols.vector]`
- You can use $ indexing (used for lists) also
  - `my.df$name.of.column`
- You can also use subset
  - ```
    subset( my.df, subset = rows.logical
            , select = cols.expression )
    ```
    ```

Data frames and matrices

- Using $ indexing can be tedious
  - `plot( my.df$x, my.df$y, pch = (19:21)[my.df$group], xlab = "x", ylab = "y" )`
  - `with( my.df, plot( x, y, pch=(19:21)[group] ) )`
  - `attach( my.df )`
    `plot( x, y, pch=(19:21)[group] )`
    `detach( my.df )`
- **In general, don't use attach!!!**

```
x <- 16
my.df <- data.frame( x = 0, y = -7 )
attach( my.df )
```

  - **What does print(x) return?**

Aggregation, stacking, unstacking

- `aggregate( my.df, by = my.df$group`
  `FUN = var )`
- `stack( my.df, select = 1:2 )`
- `unstack( my.df, form = x ~ group )`

More functions

```
> x <- 3 # Assign value 3 to x; no printing
> x  # equivalent to print(x)
  [1] 3
> x*2 # equivalent to print(x*2)
  [1] 6
> (x <- 3) # equivalent to: x <- 3; print(x)
  [1] 3
```

More functions

```
> table(Sex=tinting$sex, AgeGroup=tinting$agegp)
    AgeGroup
Sex younger older
  f      63    28
  m      28    63
> sapply(jobs[, -7], FUN=range)
       BC Alberta Prairies Ontario Quebec Atlantic
[1,] 1737    1366      973    5212   3167      941
[2,] 1840    1436      999    5360   3257      968
```

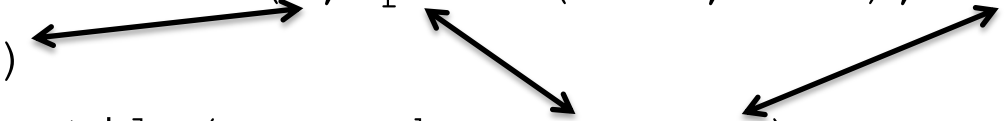
Generic functions and classes

- R has object-oriented functionality
  - Generic functions do different things depending on what class of object is passed to them

```
print()
   print.factor()
   print.data.frame()
   print.default()
plot()
   plot.formula()
   plot.ts()
   plot.default()
```

Writing your own functions

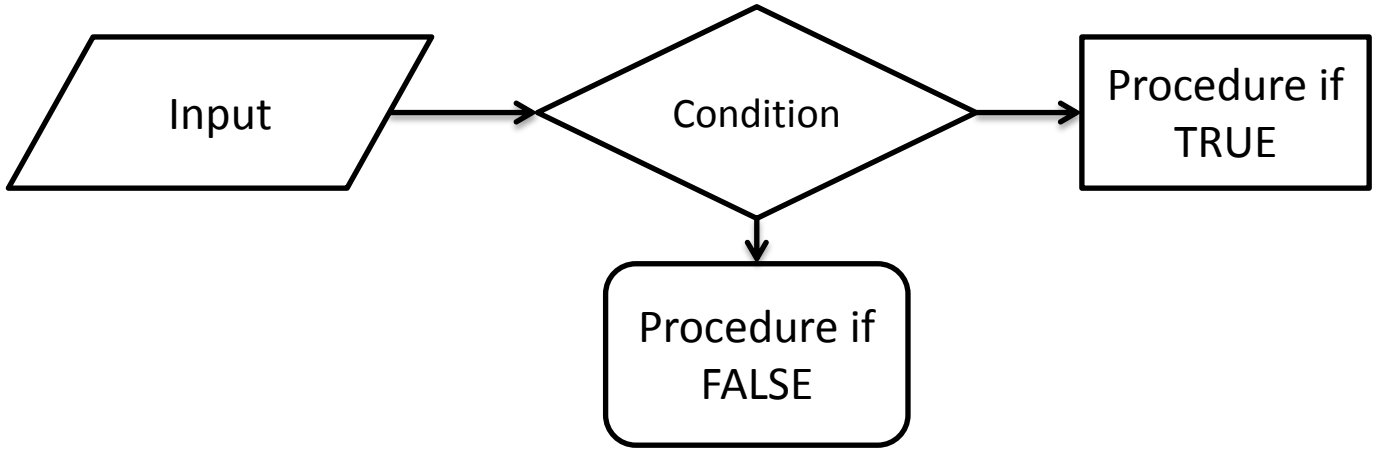
- You can write your own functions in R



```
meanQuants <- function(x, p = c(0.05,0.95), ...){
  mu <- mean(x)
  quants <- quantile(x, probs = p, ...)
c(mu, quants)
}
> meanQuants( 0:10 )
     5% 95%
5.0 0.5 9.5
> meanQuants( 0:10, p = c(0.05,0.5,0.95), type = 4 )
       5%  50%  95%
5.00 0.00 4.50 9.45
```

if

- R has flow control




```
x <- runif(1) # Draw a random uniform number
if( x > 0.5 ){
  print("Heads")}else{
  print("Tails")}
?Control
```

Looping

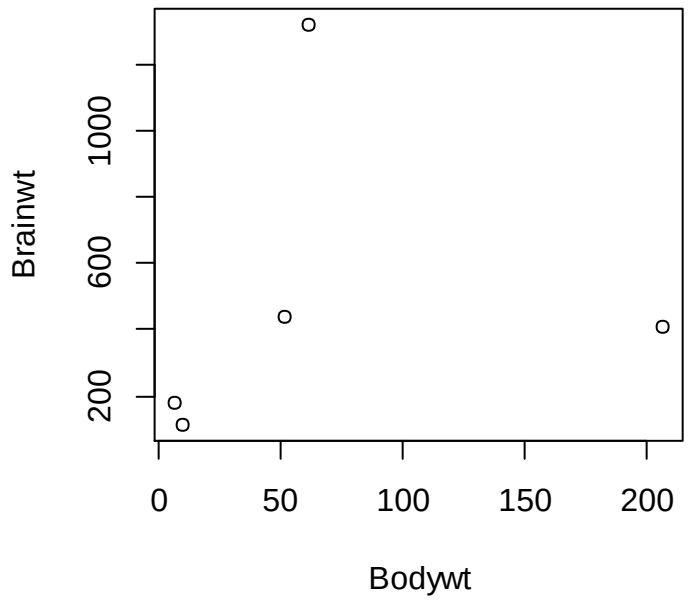
- For loops, while loops, and repeat loops are all implemented in R
  - `for( i in 1:3 ){ print(i) }`
  - ```
    i <- 1
    while( i < 4 ){ print(i); i <- i + 1 }
    ```
  - ```
    i <- 1
    repeat{ print(i); i <- i + 1;
            if( i > 3 ) break }
    ```

Graphics – the basics

```
demo(graphics)
plot(x,y)
points(x,y) # Add points to an existing plot
lines(x,y) # Add a line to an existing plot
text(x,y,labels) # Add text to an existing plot
mtext(text,side,line) # Add text to the margin of an
                      # existing plot
axis(side,...) # Add an axis to a plot
par(mfrow=c(nrow,ncol)) # multipanel by row
par(mfcol=c(nrow,ncol)) # multipanel by col
?par  # listing and setting graphics parameters
oldpar <- par( mar = c(4,4,2,2) ) # save old
           # graphics parameters to reset them later
```

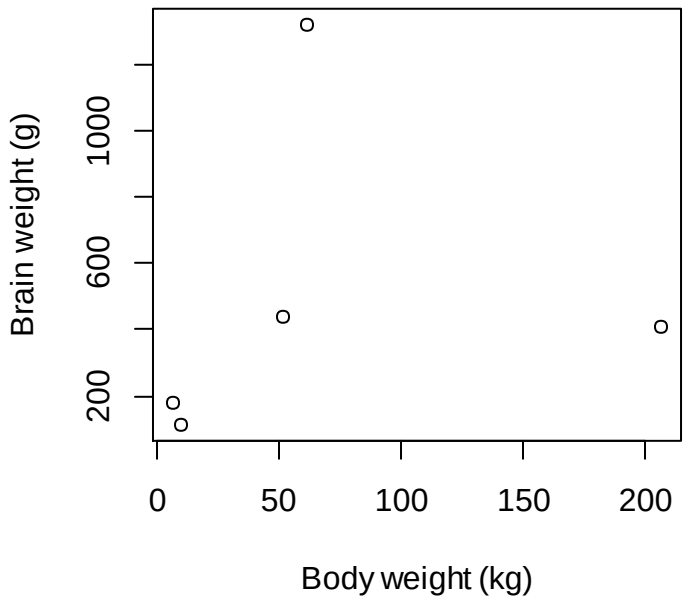
Graphics – the basics

```
with( primates, plot( Bodywt, Brainwt ) )
```



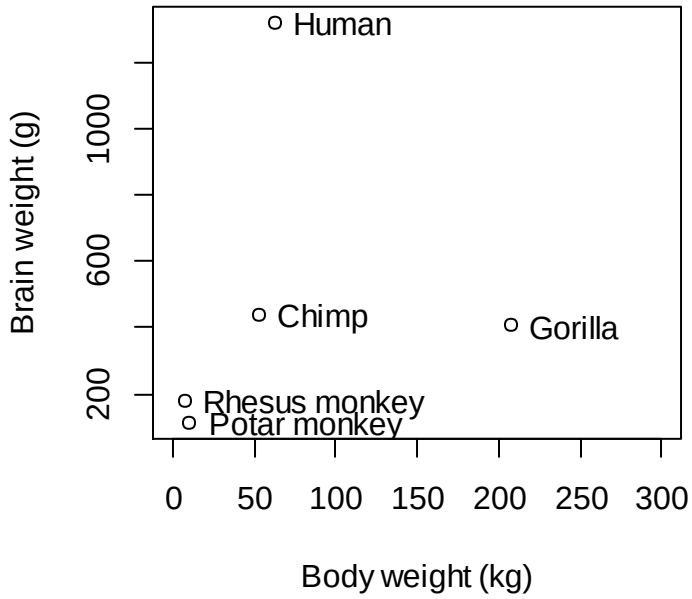
Graphics – the basics

```
with( primates, plot( Bodywt, Brainwt
    , xlab = "Body weight (kg)"
    , ylab = "Brain weight (g)" ) )
```



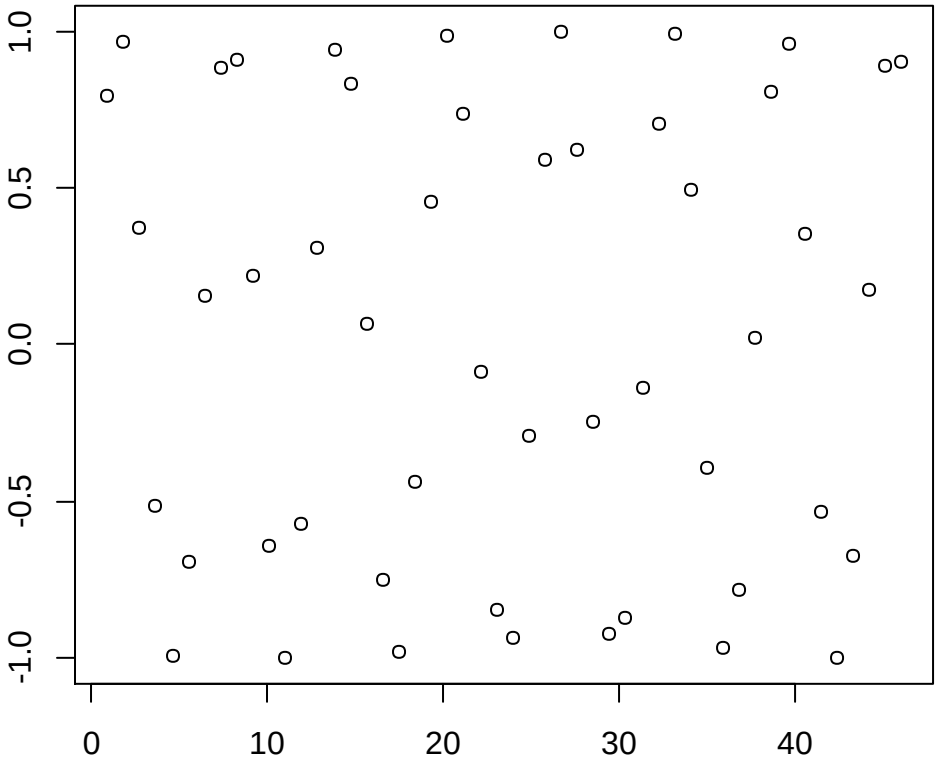
Graphics – the basics

```
with( primates, plot( Bodywt, Brainwt
    , xlab = "Body weight (kg)“
    , ylab = "Brain weight (g)", xlim = c(0,300) ) )
with( primates, text( Bodywt, Brainwt
    , labels = row.names(primates), pos = 4 ) )
```



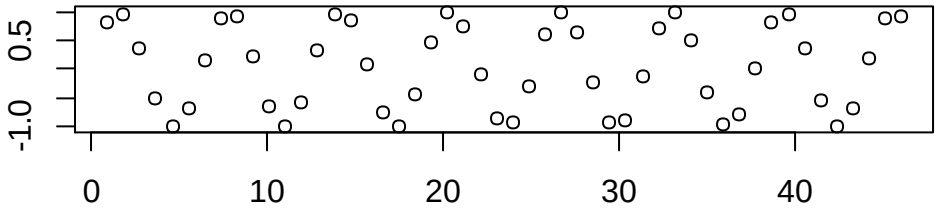
Graphics – aspect ratio

What is this a plot of?



Graphics – aspect ratio

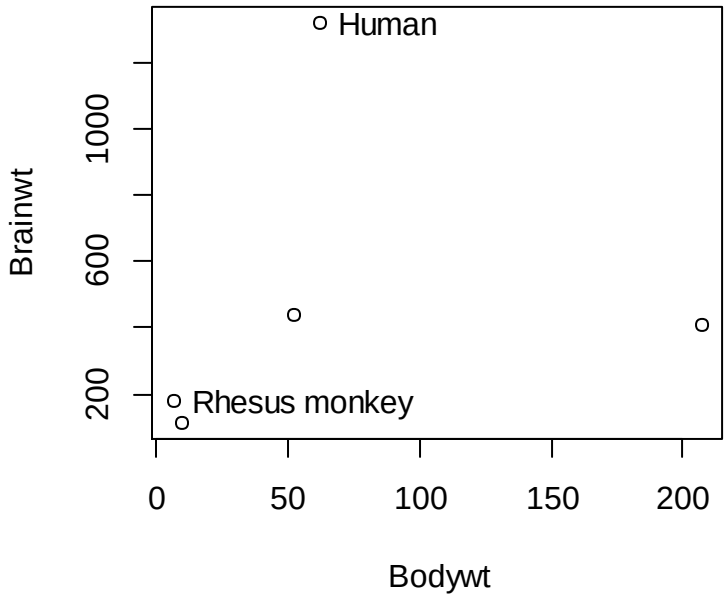
How about if we change the aspect ratio?



- Patterns that are nearly vertical or nearly horizontal are difficult for the human eye to recognize.

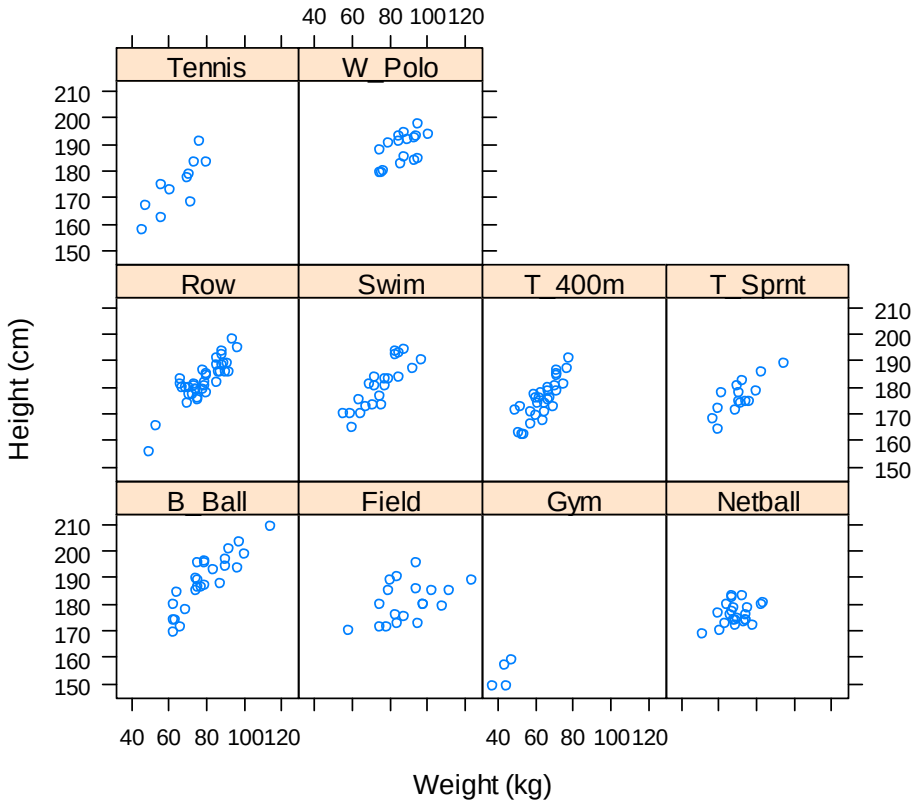
Graphics – identifying and labelling

```
locator() # returns the position of points
identify() # labels points
with( primates, plot( Bodywt, Brainwt ) )
with( primates, identify( Bodywt, Brainwt,
          , labels = row.names(primates), n=2 ) )
# This will allow us to label two of the points
# using the row names of primates using our mouse
```



Graphics – lattice graphics

```
library(lattice)
# A conditioning plot
xyplot( ht ~ wt | sport, aspect = 1, data = ais
      , xlab = "Weight (kg)", ylab = "Height (cm)" )
```



Graphics – lattice graphics

```
dotplot()          # Cleveland dot plot
stripplot()        # One-dimensional plot
barchart()         # Barplot
histogram()        # Histogram
densityplot()      # Density plot
bwplot()           # Box and whisker plot
qqmath()           # Normal probability plot
splom()            # Scatterplot matrix
parallel()         # Parallel coordinate plots
cloud()            # 3D scatterplot
wireframe()        # 3D surface plot
```