An Introduction to R

John Verzani

CUNY/College of Staten Island Department of Mathematics

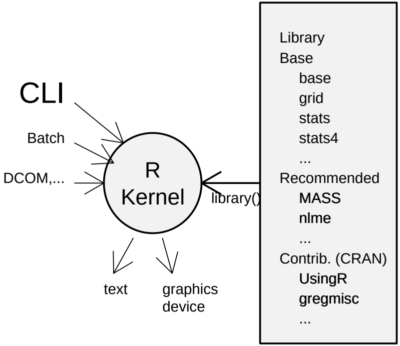
NYC ASA, CUNY Ed. Psych/May 24, 2005

Outline

R is an open-source statistical computing environment

- R is available from `http://www.r-project.org`
- R is a computing language, based on S and S-Plus, which is well suited for statistical calculations
- R has the ability to produce excellent graphics for statistical explorations and publications

The structure of R



The many faces of R

- R is ported to most modern computing platforms: Windows, MAC OS X, Unix with X11 (linux), ...
- R has an interface that varies depending on the installation:

Windows interface

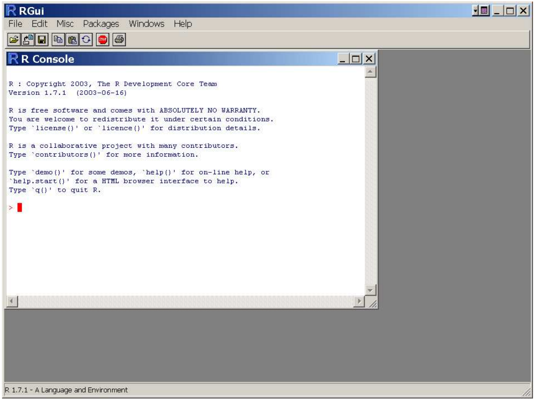


Figure: Windows gui

Mac OS X interface

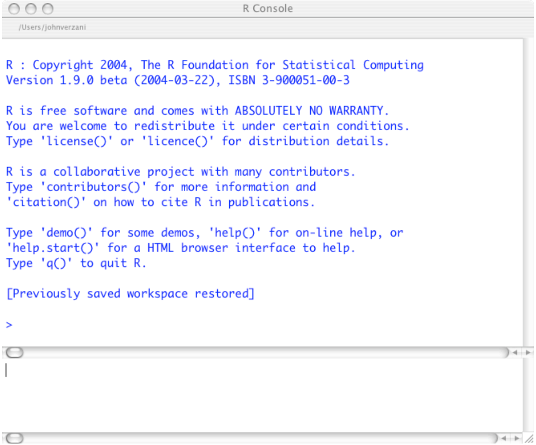


Figure: Mac OS X gui

X11 interface – the command line

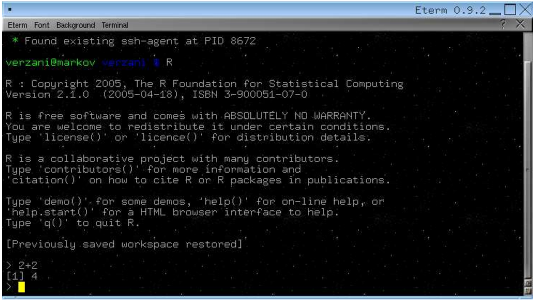


Figure: There is no standard GUI for X11 implementations. A “typical” usage may look like this screenshot. Also of interest is ESS package for (X)Emacs users.

The command line interface (CLI)

In R the typical means of interacting with the software is at the command line.

- Commands are typed at the prompt >
- Continuation lines are indicated with a +
- ENTER sends the commands off to the interpreter

2+2

```
> 2 + 2
[1] 4
```

Data types

- In statistics data comes in different types: numeric, categorical, univariate, bivariate, multivariate, etc.
- R has different data types or classes to accommodate these different types of data sets.

Some basic storage types are:

numeric vectors: created with c(), etc.

```
> somePrimes = c(2, 3, 5, 7, 11, 13, 17)
> somePrimes

[1]  2  3  5  7 11 13 17


> odds = seq(1, 15, 2)
> odds

[1]  1  3  5  7  9 11 13 15


> ones = rep(1, 10)
> ones

 [1] 1 1 1 1 1 1 1 1 1 1
```

Character strings are indicated by using matching quote, double of single.

Character variables

```
> character = c("Homer", "Marge", "Bart", "Lisa",
+       "Maggie")
> gender = c("Male", "Female", "Male", "Female",
+       "Male")
```

Categorical variables – factors

```
> gender = factor(c("Male", "Female", "Male",
+       "Female", "Male"))
> gender
```

```
[1] Male   Female Male   Female Male
Levels: Female Male
```

Factors have an extra attribute a fixed set of levels that require some care. (E.g., you can't add new levels without some work.) Factors are used instead of character vectors, as this allows R to identify certain types of data. (Storage space is smaller as well.)

Logical vectors: vectors of TRUE or FALSE

```
> somePrimes
[1]  2  3  5  7 11 13 17

> somePrimes < 10
[1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE

> somePrimes %in% c(3, 5, 7)
[1] FALSE  TRUE  TRUE  TRUE FALSE FALSE FALSE

> somePrimes == 2 | somePrimes >= 10
[1]  TRUE FALSE FALSE FALSE  TRUE  TRUE  TRUE
```

Matrices

defining matrices:matrix(), rbind(), ...

```
> M = rbind(c(1, 1), c(0, 1))
> M

     [,1] [,2]
[1,]    1    1
[2,]    0    1
```

Matrices, cont.

Operations: multiplication. (∗ is entry-by-entry

```
> M %*% M
     [,1] [,2]
[1,]    1    2
[2,]    0    1
```

Inverse is found by “solving” $Ax = b$, b an indentity matrix

```
> solve(M)
     [,1] [,2]
[1,]    1   -1
[2,]    0    1
```

Matrices, cont.

Least squares regression coefficients the hard way

```
> x = 1:5
> y = c(2, 3, 1, 4, 5)
> ones = rep(1, length(x))
> X = cbind(ones, x)
> solve(t(X) %*% X, t(X) %*% y)
     [,1]
ones  0.9
x     0.7
```

Lists

Lists are recursive structures with each level made up of components.

- List components can be other data types, functions, additional lists, etc.
- Lists are used often as return values of functions in R. The print method is set to show only part of the values contained in the list.

Defining lists

```
> lst = list(a = somePrimes, b = M, c = mean)
> lst
$a
[1]  2  3  5  7 11 13 17

$b
     [,1] [,2]
[1,]    1    1
[2,]    0    1

$c
function (x, ...)
UseMethod("mean")
<environment: namespace:base>
```

Other data types

Data can be given extra attributes, such as a time series:

Time series have regular date information

```
> google = c(100.2, 132.6, 196, 180, 202.7,
+     191.9, 186.1, 180)
> ts(google, start = c(2004, 9), frequency = 12)

       Jan   Feb   Mar   Apr May Jun Jul Aug   Sep
2004                                         100.2
2005 202.7 191.9 186.1 180.0
       Oct   Nov   Dec
2004 132.6 196.0 180.0
2005
```

Tables – an extension of an matrix or array

The table() function (also `xtabs`, `ftable`,...)

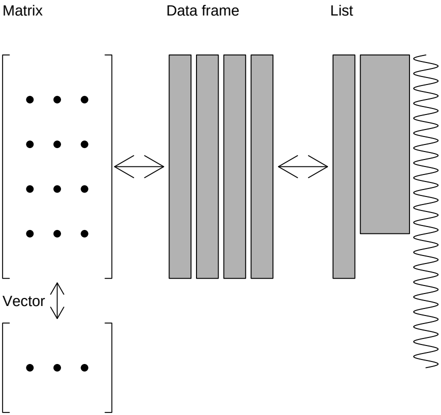
```
> table(gender)

gender
Female   Male
     2      3


> satisfaction = c(3, 4, 3, 5, 4, 3)
> category = c("a", "a", "b", "b", "a", "a")
> table(category, satisfaction)

        satisfaction
category 3 4 5
       a 2 2 0
       b 1 0 1
```

Data frames

The most common data-storage format is a data frame

- Stores rectangular data: each column a variable, typically each row data for one subject
- Columns have names for easy reference
- May be manipulated like a matrix or a list (each variable a top-level component)

Relationship between vector, matrix, data frame, list



Data frame examples

```
> role = c("Comic relief", "Parent", "troublemaker",
+      "Goody two-shoes", "Cute baby")
> theSimpsons = data.frame(name = character,
+      gender = gender, role = role)
> theSimpsons
```

```
    name gender            role
1  Homer   Male    Comic relief
2  Marge Female          Parent
3   Bart   Male    troublemaker
4   Lisa Female Goody two-shoes
5 Maggie   Male       Cute baby
```

Reading in data

Data can be built-in, entered in at the keyboard, or read in from external files. These may be formatted using fixed width format, commas separated values, tables, etc. For instance, this command reads in a data set from a url:

Reading urls

```
> f = "http://www.math.csi.cuny.edu/st/R/crackers.csv"
> crackers = read.csv(f)
> names(crackers)

 [1] "Company"               "Product"
 [3] "Crackers"              "Grams"
 [5] "Calories"              "Fat.Calories"
 [7] "Fat.Grams"             "Saturated.Fat.Grams"
 [9] "Sodium"                "Carbohydrates"
[11] "Fiber"
```

Assignment, Extraction

Values in vectors, matrices, lists, and data frames can be accessed by their components:

By index

```
> google[1:3]
[1] 100.2 132.6 196.0

> crackers[1:3, 2:3]
                                  Product Crackers
1      Country Water Cracker Crck Pepper        4
2          Country Water Cracker Klassic        4
3 Country Water Cracker Sun Dried Tomato        4
```

By index cont.

```
> M[1, ]
[1] 1 1

> lst[[1]]
[1]  2  3  5  7 11 13 17
```

Access by name

by name

```
> crackers[1:3, c("Product", "Crackers")]

                                        Product Crackers
1       Country Water Cracker Crck Pepper          4
2            Country Water Cracker Klassic          4
3 Country Water Cracker Sun Dried Tomato          4


> theSimpsons[["role"]]

[1] Comic relief     Parent           troublemaker
[4] Goody two-shoes Cute baby
5 Levels: Comic relief Cute baby ... troublemaker
```

Access by logical expressions

logical questions answered TRUE or FALSE

```
> somePrimes < 10
[1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE

> somePrimes[somePrimes < 10]
[1] 2 3 5 7
```

Recycling values

When making assignments in R we might have a situation where many values are replaced by 1, or a few. R has a means of *recycling* the values in the assignment to fill in the size mismatch.

replace coded values with NA

```
> x = c(1, 1, 0, 1, 99, 0, 1, 99, 0, 1, 99)
> x[x == 99] = NA
> x[x == 1] = "Yes"
> x[x == 0] = "No"
> x

 [1] "Yes" "Yes" "No"  "Yes" NA    "No"  "Yes" NA
 [9] "No"  "Yes" NA
```

Applying functions to data

Finding the mean

```
> fat = crackers$Fat.Grams
> mean(fat)
[1] 3.679
```

The median

```
> median(fat)
[1] 3.25
```

Extra arguments to find trimmed mean

```
> mean(fat, trim = 0.2)
[1] 3.482
```

missing data – coded NA

```
> shuttleFailures = c(0, 1, 0, NA, 0, 0, 0)
> mean(shuttleFailures, na.rm = TRUE)
[1] 0.1667
```

Functions

- Functions are called by name with a matching pair of ()
- Arguments may be indicated by position or name
- Named arguments can (and usually do) have reasonable defaults
- A special role is played by the first argument

generic functions

Interacting with R from the command line requires one to remember a lot of function names, although R helps out somewhat. In practice, many tasks may be viewed generically: E.g., "print" the values of an object, "summarize" values of an object, "plot" the object. Of course, different objects should yield different representations.

R has methods (S3, S4) to declare a function to be generic. This allows different functions to be "dispatched" based on the "class" of the first argument.

A basic template is:

methodName(object, extraArguments)

Some common generic functions are `print()` (the default action), `summary()` (for summaries), `plot()` (for basic plots).

summary() function called on a number and factor

```
> summary(somePrimes)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   2.00    4.00    7.00    8.29   12.00   17.00

> summary(gender)
Female   Male
     2      3
```

R, like MATLAB, is naturally vectorized. For instance, to find the sample variance, $(n-1)^{-1}\sum(x_i-\overline{x})^2$ by hand involves:

sample variance (also `var()`)

```
> x = c(2, 3, 5, 7, 11, 13)
> fractions(x - mean(x))

[1] -29/6 -23/6 -11/6   1/6  25/6  37/6


> fractions((x - mean(x))^2)

[1]  841/36  529/36  121/36    1/36  625/36 1369/36


> fractions(sum((x - mean(x))^2)/(length(x) -
+     1))

[1] 581/30
```

Example: simulating a sample distribution

A simulation of the sampling distribution of $\overline{x}$ from a random sample of size 10 taken from an exponential distribution with parameter 1 naturally lends itself to a "for loop:"

for loop simulation

```
> res = c()
> for (i in 1:200) {
+     res[i] = mean(rexp(10, rate = 1))
+ }
> summary(res)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.392   0.786   0.946   1.000   1.220   2.440
```

Vectorizing a simulation

It is often faster in R to vectorize the simulation above by generating all of the random data at once, and then *applying* the mean() function to the data. The natural way to store the data is a matrix.

Simulation using a matrix

```
> m = matrix(rexp(200 * 10, rate = 1), ncol = 200)
> res = apply(m, 2, mean)
> summary(res)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.308   0.800   1.010   1.020   1.190   1.830
```

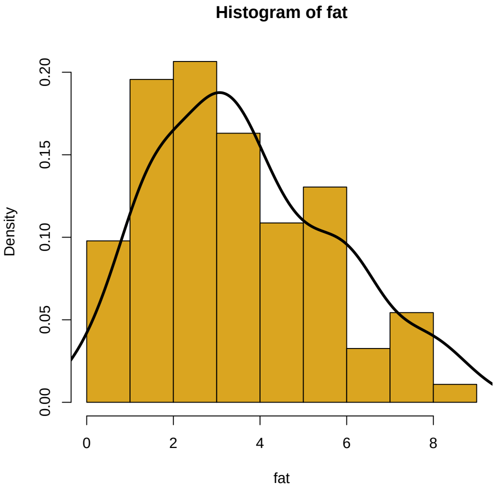
Graphics

R has many built-in graphic-producing functions, and facilities to create custom graphics. Some standard ones include:

histogram and density estimate

```
> hist(fat, probability = TRUE, col = "goldenrod")
> lines(density(fat), lwd = 3)
```

histogram and density estimate



Graphics: cont.

Quantile-Quantile plots

```
> qqnorm(fat)
```

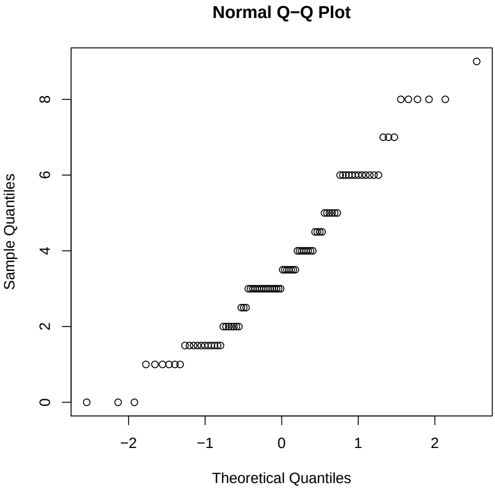
Boxplots

```
> boxplot(MPG.highway ~ Type, data = Cars93)
```

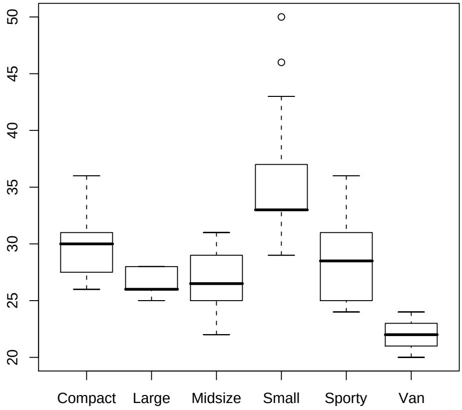
plot() is generic, last one also with

```
> plot(MPG.highway ~ Type, data = Cars93)
```

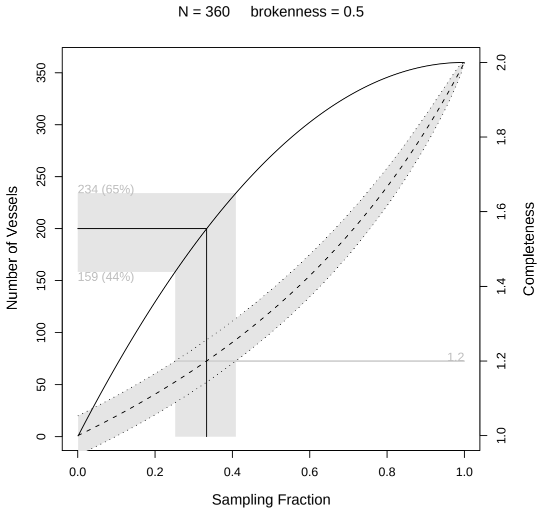
Quantile-Quantile plot



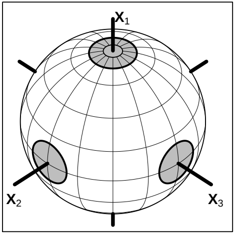
Boxplots

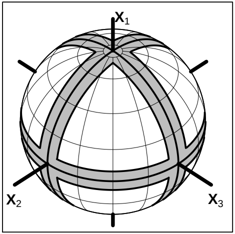


Fancy examples from upcoming book of P. Murrell



3-d graphics





Model formula notation

The boxplot example illustrates R's model formula. Many generic functions have a method to handle this type of input, allowing for easier usage with multivariate data objects.

Example with xtabs – tables

```
> df = data.frame(cat = category, sat = satisfaction)
> xtabs(~cat + sat, df)

   sat
cat 3 4 5
  a 2 2 0
  b 1 0 1
```

Model formula cont.

Suppose `x`, `y` are numeric variables, and `f` is a factor. The basic model formulas have these interpretations:

`y ~ 1`	$y_i = \beta_0 + \varepsilon_i$	
`y ~ x`	$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$	
`y ~ x - 1`	$y_i = \beta_1 x_i + \varepsilon_i$	remove the intercept
`y ~ x | f`	$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$	grouped by levels of *f*
`y ~ f`	$y_{ij} = \tau_i + \varepsilon_{ij}$	

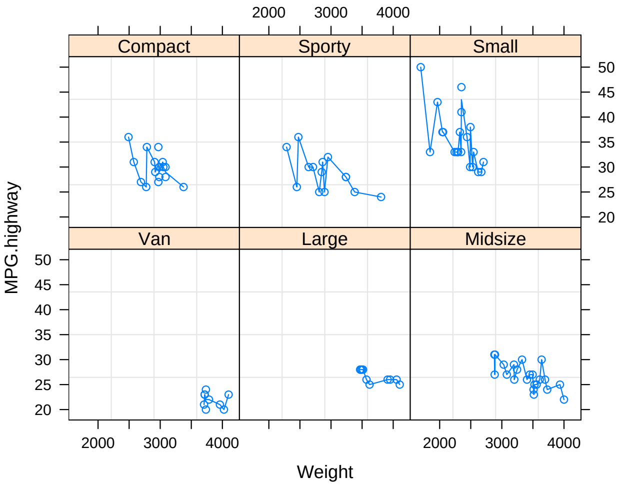
The last usage suggests storing multivariate data in two variables – a numeric variable with the measurements, and a factor indicating the treatment.

Lattice graphics

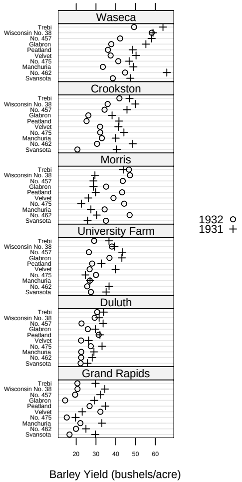
Lattice graphics can effectively display multivariate data that are naturally defined by some grouping variable. (Slightly more complicated than need be to show `groupedData()` function in `nlme` package.)

lattice graphics

```
> cars = groupedData(MPG.highway ~ Weight |
+      Type, Cars93)
> plot(cars)
```


Compact
Sporty
Small
Van
Large
Midsize
MPG.highway
Weight
2000
3000
4000
20
25
30
35
40
45
50

more lattice graphics (Murrell's book)



Significance tests

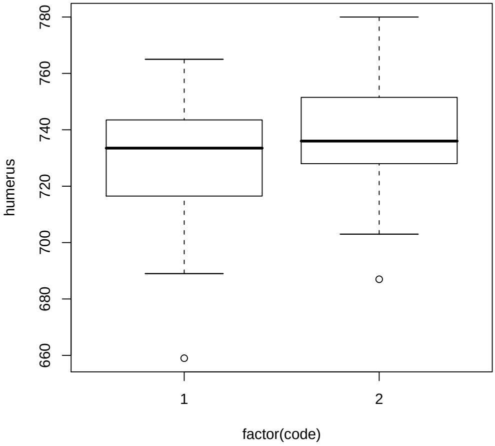
There are several functions for performing classical statistical tests of significance: `t.test()`, `prop.test()`, `oneway.test()`, `wilcox.test()`, `chisq.test()`, ...
These produce a *p*-value, and summaries of the computations.

The Bumpus data set (Ramsey and Shafer) contains data from 1898 lecture supporting evolution (Some birds survived a harsh winter storm)

two-sample t test

```
> Bumpus = read.table("Bumpus.txt", header = TRUE)
> plot(humerus ~ factor(code), data = Bumpus)
```

Diagnostic plot



t.test() output

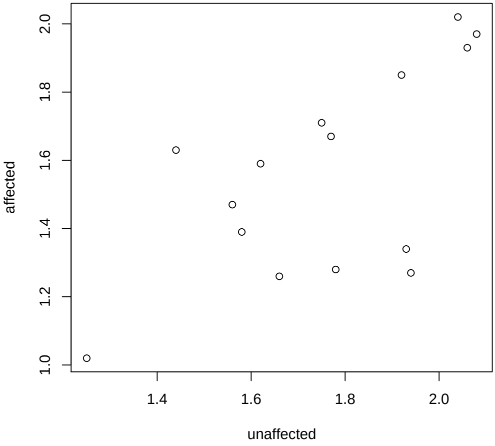
```
> t.test(humerus ~ code, data = Bumpus)


        Welch Two Sample t-test

data:  humerus by code
t = -1.721, df = 43.82, p-value = 0.09236
alternative hypothesis: true difference in means is not equ
95 percent confidence interval:
 -21.895   1.728
sample estimates:
mean in group 1 mean in group 2
          727.9           738.0
```

The SchizoTwins data set (R&S) contains data on 15 pairs of monozygotic twins. Measured values are of volume of left hippocampus.

t-tests: paired

```
> twins = read.table("SchizoTwins.txt", header = TRUE)
> plot(affected ~ unaffected, data = twins)
> attach(twins)
> t.test(affected - unaffected)$p.value

[1] 0.006062

> t.test(affected, unaffected, paired = TRUE)$p.value

[1] 0.006062

> detach(twins)
```

Diagnostic plot



Confidence intervals

Confidence intervals are computed as part of the output of many of these functions. The default is to do 95% CIs, which may be adjusted using `conf.level=`.

95% CI humerus length overall

```
> t.test(Bumpus$humerus)

...
 95 percent confidence interval:
     728.2 739.6
...
```

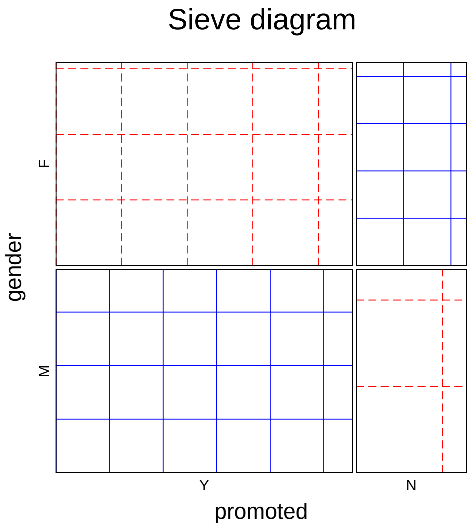
Chi-square tests

Goodness of fit tests are available through `chisq.test()` and others. For instance, data from Rosen and Jerdee (1974, from R&S) on the promotion of candidates based on gender:

gender data

```
> rj = rbind(c(21, 3), c(14, 10))
> dimnames(rj) = list(gender = c("M", "F"),
+      promoted = c("Y", "N"))
> rj

      promoted
gender   Y  N
     M  21  3
     F  14 10
```

sieveplot(rj)



chi-squared test *p*-value

```
> chisq.test(rj)$p.value
[1] 0.05132
```

Fischer's exact test

```
> fisher.test(rj, alt = "greater")$p.value
[1] 0.02450
```

Fitting linear models

Linear models are fit using `lm()`:

- This function uses the syntax for model formula
- Model objects are reticent – you need to ask them for more information
- This is done with extractor functions: `summary()`, `resid()`, `fitted()`, `coef()`, `predict()`, `anova()`, `deviance()`,...

Body fat data set

```
> source("http://www.math.csi.cuny.edu/st/R/fat.R")
> names(fat)
 [1] "case"         "body.fat"     "body.fat.siri"
 [4] "density"      "age"          "weight"
 [7] "height"       "BMI"          "ffweight"
[10] "neck"         "chest"        "abdomen"
[13] "hip"          "thigh"        "knee"
[16] "ankle"        "bicep"        "forearm"
[19] "wrist"
```

Fitting a simple linear regression model

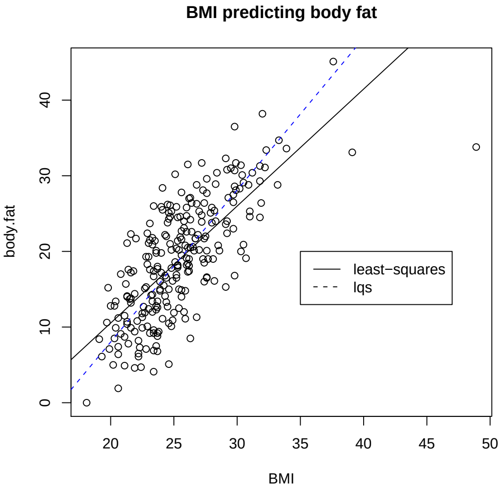
Basic fit is done with `lm`: response ~ predictor(s)

```
> res = lm(body.fat ~ BMI, data = fat)
> res


Call:
lm(formula = body.fat ~ BMI, data = fat)

Coefficients:
(Intercept)          BMI
     -20.41         1.55
```

Making scatterplot, adding the regression line

```
> plot(body.fat ~ BMI, data = fat)
> abline(res)
> res.robust = lqs(body.fat ~ BMI, data = fat)
> abline(res.robust, lty = 2, col = "blue")
> title("BMI predicting body fat")
> legend(35, 20, legend = c("least-squares",
+     "lqs"), lty = 1:2)
```

Scatterplot with regression line



Basic output is minimal, more is given by summary()

```
> summary(res)

Call:
lm(formula = body.fat ~ BMI, data = fat)

Residuals:
     Min       1Q   Median       3Q      Max
-21.4292  -3.4478   0.2113   3.8663  11.7826

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -20.40508    2.36723   -8.62 7.78e-16 ***
BMI           1.54671    0.09212   16.79  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '
```

summary() output cont.

```
Residual standard error: 5.324 on 250 degrees of freedom
Multiple R-Squared:  0.53,      Adjusted R-squared: 0.5281
F-statistic: 281.9 on 1 and 250 DF,  p-value: < 2.2e-16
```

Extractor functions used to extract information

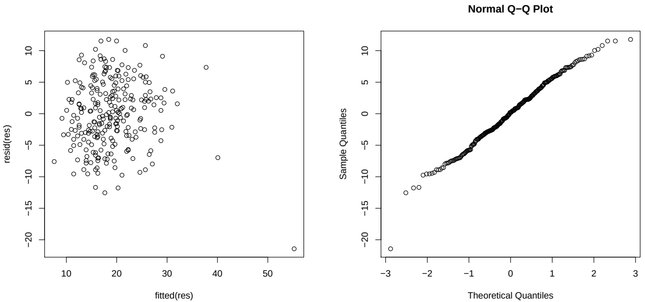
extractor functions

```
> coef(res)
(Intercept)          BMI
    -20.405        1.547

> summary(residuals(res))
     Min.   1st Qu.    Median      Mean   3rd Qu.
-2.14e+01 -3.45e+00  2.11e-01  3.52e-17  3.87e+00
     Max.
 1.18e+01
```

Residual plots to test model assumptions

```
> plot(fitted(res), resid(res))

> qqnorm(resid(res))
```

Residual plots



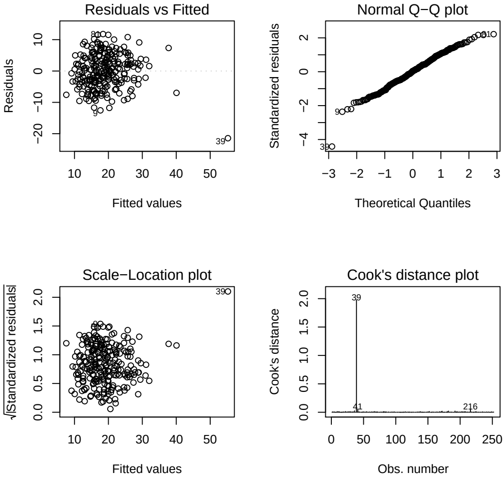
Default diagnostic plots

Model objects, such as the output of `lm()`, have default plots associated with them. For `lm()` there are four plots.

plot(res)

```
> par(mfrow = c(2, 2))
> plot(res)
```

Diagnostic plots for `lm()`



Predictions done using the predict() extractor function

```
> vals = seq(15, 35, by = 2)
> names(vals) = vals
> predict(res, newdata = data.frame(BMI = vals))
    15     17     19     21     23     25     27
 2.796  5.889  8.982 12.076 15.169 18.263 21.356
    29     31     33     35
24.450 27.543 30.636 33.730
```

Multiple regression

Multiple regression models are modeled with `lm()` as well. Extra covariates are specified using the following notations:

- + adds terms (- subtracts them, such as `-1`)
- Math expressions can (mostly) be used as is: log, exp,...
- `I()` used to insulate certain math expressions
- `a:b` adds an interaction between `a` and `b`. Also, `*`, `^` are shortcuts for more complicated interactions

To illustrate, we model the `body.fat` variable, by measurements that are easy to compute

Modeling body fat

```
> res = lm(body.fat ~ age + weight + height +
+      chest + abdomen + hip + thigh, data = fat)
> res


Call:
lm(formula = body.fat ~ age + weight + height + chest + abc

Coefficients:
(Intercept)          age       weight       height
  -33.27351      0.00986     -0.12846     -0.09557
      chest      abdomen          hip        thigh
   -0.00150      0.89851     -0.17687      0.27132
```

Model selection using AIC

```
> stepAIC(res, trace = 0)

Call:
lm(formula = body.fat ~ weight + abdomen + thigh, data = fa

Coefficients:
(Intercept)       weight      abdomen        thigh
    -48.039       -0.170        0.917        0.209
```

(Set `trace=1` to get diagnostic output.)

Model selection using F-statistic

```
> res.sub = lm(body.fat ~ weight + height +
+       abdomen, fat)
> anova(res.sub, res)

Analysis of Variance Table

Model 1: body.fat ~ weight + height + abdomen
Model 2: body.fat ~ age + weight + height + chest + abdomer
  Res.Df  RSS  Df Sum of Sq    F Pr(>F)
1    248 4206
2    244 4119   4        88 1.3   0.27
```

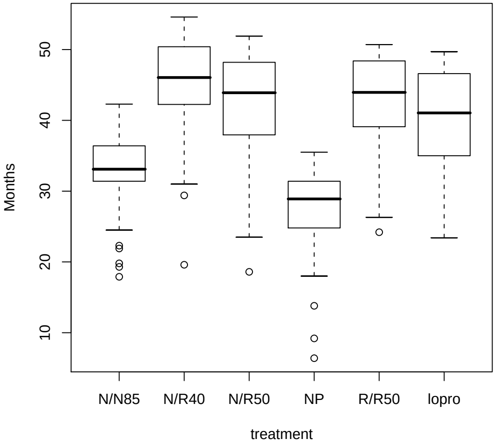
Analysis of variance models

A simple one-way analysis of variance test is done using, `oneway.test()` with a model formula of the type `y ~ f`.
For instance, we look at data on the lifetime of mice who have been given a type of diet (R&S).

read in data, make plot

```
> mice = read.table("mice-life.txt", header = TRUE)
> plot(lifetime ~ treatment, data = mice, ylab = "Months")
```

Plot of months survived by diet



One way test of equivalence of means

```
> oneway.test(lifetime ~ treatment, data = mice,
+     var.equal = TRUE)


        One-way analysis of means

data:  lifetime and treatment
F = 57.1, num df = 5, denom df = 343, p-value <
2.2e-16
```

(`var.equal=TRUE` for assumption of equal variances, not default)

Using lm() for ANOVA

Modeling is usually done with a modeling function. The lm() function can also fit a one-way ANOVA, again with the same model formula

Using lm()

```
> res = lm(lifetime ~ treatment, data = mice)
> anova(res)

Analysis of Variance Table

Response: lifetime
           Df Sum Sq Mean Sq F value Pr(>F)
treatment   5  12734    2547    57.1 <2e-16 ***
Residuals 343  15297      45
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '
```

Following Ramsey and Schafer we ask: Does lifetime on 50kcal/wk exceed that of 85 kcal/month? We can take advantage of the use of treatment contrasts to investigate this (difference in mean from first level is estimated).

Treatment contrasts, set β_1

```
> treat = relevel(mice$treatment, "N/R50")
> res = lm(lifetime ~ treat, data = mice)
> coef(summary(res))
```

```
            Estimate Std. Error t value Pr(>|t|)
(Intercept)    42.30        0.8   53.37 3.4e-168
treatN/N85     -9.61        1.2   -8.09  1.1e-14
treatN/R40      2.82        1.2    2.41  1.7e-02
treatNP       -14.90        1.2  -12.01  5.7e-28
treatR/R50      0.59        1.2    0.49  6.2e-01
treatlopro     -2.61        1.2   -2.19  2.9e-02
```

logistic regression

Logistic regression extends the linear regression framework to binary response variables. It may be seen as a special case of a *generalized linear model* which consists of:

- A response y and covariates $x_1, x_2, \ldots, x_p$
- A linear predictor $\eta = \beta_1 x_1 + \cdots + \beta_p x_p$.
- A specific *family* of distributions which describe the random variable y with mean response, $\mu_{y|x}$, related to η through a *link function*, m^{-1}, where

$$\mu_{y|x} = m(\eta), \qquad \text{or } \eta = m^{-1}(\mu_{y|x})$$

logistic regression example

- Linear regression would be m being the identity and the distribution being the normal distribution.
- For logistic regression, the response variable is Bernoulli, so the mean is also the probability of success. The link function is the *logit*, or log-odds, function and the family is Bernoulli, a special case of the binomial.

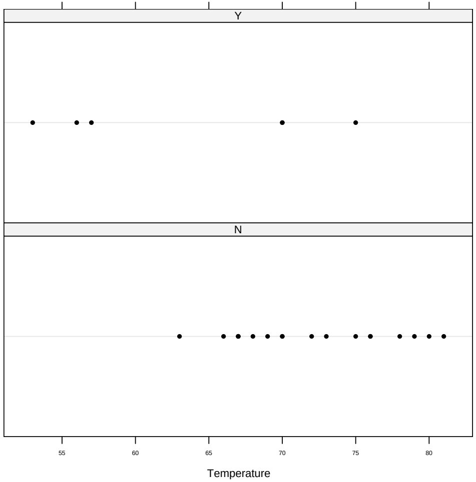
We illustrate the implementation in R with an example from Devore on the failure of space shuttle rings (a binary response variable) in terms of take off temperature.

Space shuttle data

read in data, diagnostic plot

```
> shuttle = read.table("shuttle.txt", header = TRUE)
> dotplot(~Temperature | Failure, data = shuttle,
+     layout = c(1, 2))
```

Lift-off temperature by O-Ring failure/success (Y/N)



Fitting a logistic model

We need to specify the *formula*, the *family*, and the *link* to the `glm()` function:

Specify model, family, optional link

```
> res.glm = glm(Failure ~ Temperature, data = shuttle,
+      family = binomial(link = logit))
> coef(summary(res.glm))

             Estimate Std. Error z value Pr(>|z|)
(Intercept)   11.7464     6.0214   1.951  0.05108
Temperature   -0.1884     0.0891  -2.115  0.03443
```

(Actually, `link=logit` is default for `binomial` family.)

mixed effects

Mixed-effects models are fit using the `nlme` package (or its new replacement `lmer` which can also do logistic regression).

- Need to specify the fixed and random effects
- Can optionally specify structure beyond independence for the error terms.
- The implemention is well documented in Pinheiro and Bates (2000)

Mixed-effects example

We present an example from Piheiro and Bates on a data set involving a dental measurement taken over time for the same set of subjects – longitudinal data.
The key variables

- `Orthodont` the data frame containing:
- `distance` – measurement
- `age` – age of subject at time of measurement
- `Sex` – gender of subject
- `subject` – subject code

Fit model with lm(), check

The simple regression model is

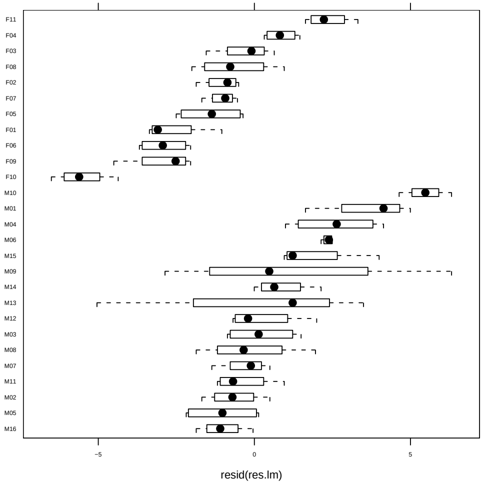
$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i,$$

where ε_i is $\mathcal{N}(0, \sigma^2)$.

model using lm()

```
> res.lm = lm(distance ~ I(age - 11), Orthodont)
> res.lm
> bwplot(getGroups(Orthodont) ~ resid(res.lm))

Call:
lm(formula = distance ~ I(age - 11), data = Orthodont)

Coefficients:
(Intercept)  I(age - 11)
      24.02         0.66
```

Boxplots of residuals by group



Fit each group

A linear model fit for each group is this model

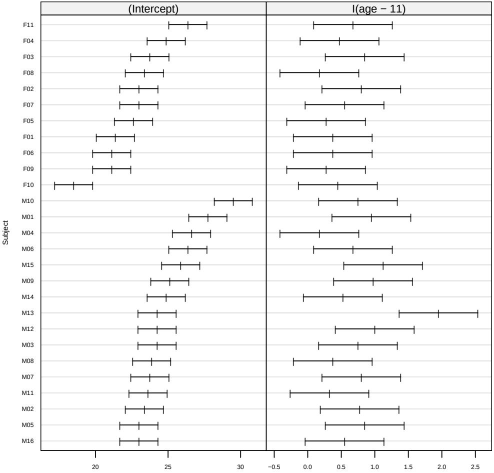
$$y_{ij} = \beta_{i0} + \beta_{i1} x_{ij} + \varepsilon_{ij},$$

where ε_{ij} are $\mathcal{N}(0, \sigma^2)$.

fit each group with lmList()

```
> res.lmlist = lmList(distance ~ I(age - 11) |
+       Subject, data = Orthodont)
> plot(intervals(res.lmlist))
```

Intervals of slope, intercept estimate by group



Fit with random effect

This fits the model

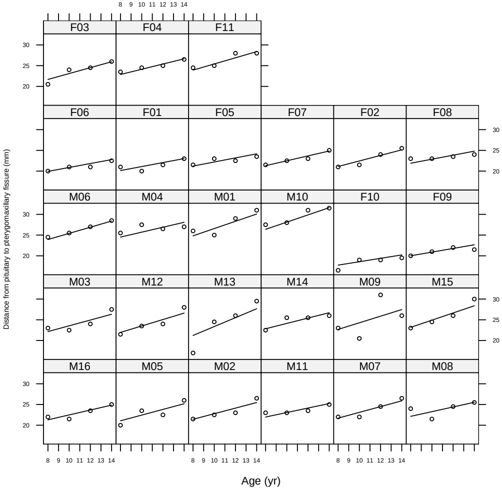
$$y_{ij} = (\beta_0 + b_{0i}) + (\beta_1 + b_{1i})x_{ij} + \varepsilon_{ij},$$

with $b_{.i}$ $\mathcal{N}(0, \Psi)$. ε_{ij} $\mathcal{N}(0, \sigma^2)$.

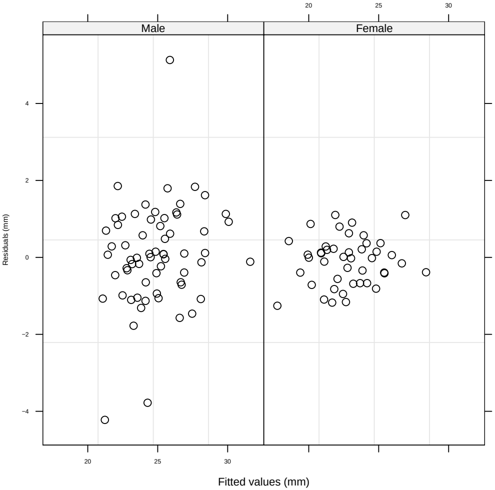
fit with `lme()`

```
> res.lme = lme(distance ~ I(age - 11), data = Orthodont,
+       random = ~I(age - 11) | Subject)
> plot(augPred(res.lme))
> plot(res.lme, resid(.) ~ fitted(.) | Sex)
```

Predictions based on random-effects model



Residuals by gender, subject



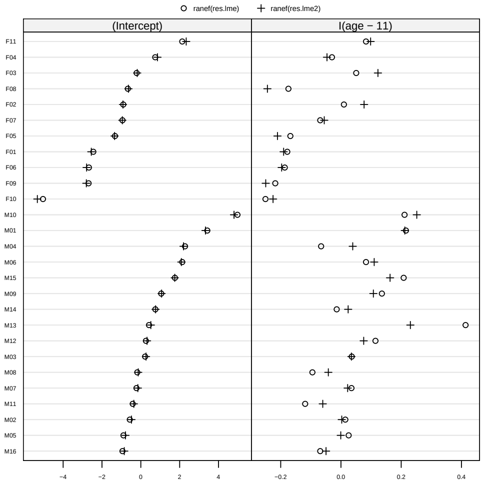
Adjust the variance

Adjust the variance for different groups is done by specifying a formula to the `weights=` argument:

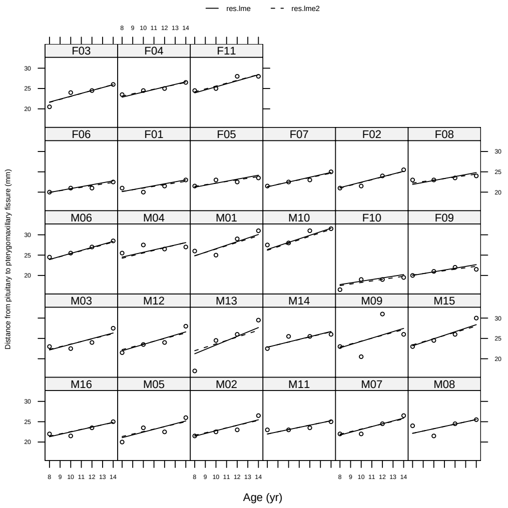
Adjust σ for each gender

```
> res.lme2 = update(res.lme, weights = varIdent(form = ~1
+      Sex))
> plot(compareFits(ranef(res.lme), ranef(res.lme2)))
> plot(comparePred(res.lme, res.lme2))
```

Compare random effects BLUPs for two models



Compare the different predictions



compare models using anova()

These nested models can be formally compared using a likelihood ratio test. The details are carried out by the anova() method:

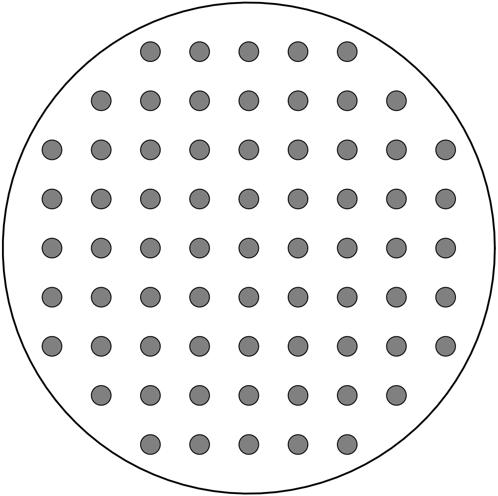
compare nested models

```
> anova(res.lme, res.lme2)
         Model df   AIC   BIC logLik   Test L.Ratio
res.lme      1  6 454.6 470.6 -221.3
res.lme2     2  7 435.6 454.3 -210.8 1 vs 2   20.99
         p-value
res.lme
res.lme2  <.0001
```

Extending R: writing functions

R can be extended by writing functions. These may be defined in separate files and read into R, or defined within an R session. For instance, how to create the following diagram?

Pizza and pepperoni



helper functions

```
plotCircle = function(x,radius=1,...) {
  t = seq(0,2*pi,length=100)
  polygon(x[1]+ radius*cos(t), x[2]+radius*sin(t), ...)
}

doPlot = function(x,r,R,...) {
  if(sqrt(sum(x^2))+r < R) plotCircle(x,r=r,...)
}
```

Key points

- functions are defined using `function()`
- Arguments are matched by position or name
- Named arguments may be abbreviated
- The special argument `...` is used to pass along extra arguments
- Command blocks are indicated using braces
- Functions can be defined on the command line, used anonymously, or be stored in files to be `source`d in.

Calling function

```
plotPizza = function(n,R=n,r=0.2) {
  par(mai=c(0,0,0,0))
  plot.new();plot.window(xlim=c(-n,n),ylim=c(-n,n),asp=1)
  plotCircle(c(0,0),radius=R,lwd=2)

  x = rep(-n:n,rep(2*n+1,2*n+1))
  y = rep(-n:n,length.out=(2*n+1)^2)
  apply(cbind(x,y),1,function(x) doPlot(x,r,R,col=gray(.5)))
}

plotPizza(5)
```

Extending R: add-on packages

One can extend R using add-on packages. Some are built-in (≈ 10), over 400 contributed packages are hosted on CRAN (`http://cran.r-project.org`), others are on author's websites. For instance, neglecting issues with permissions, to download and install a basic GUI for R can be done with the command

Installing a package

```
> install.packages("Rcmdr")
```

This GUI, available for the three main platforms, allows one to select variables, and fill in function arguments with a mouse. The command `install.packages()` allows one to browse the available packages.

Learning more

R has several different ways that you can learn more:
The built in help pages. Some key fuctions are

- `help.start()` – to start web interface
- `?functionName` – to find specific help for the named function
- `apropos("word")` To search through searchlist for a word
- `help.search()` matches in more places than `apropos()`

More free documentation

The accompanying manuals: Included with R are 5 manuals in pdf or html form. *An Introduction to R* contains *lots* of information.

Contributed documentation: Contributed documentation: On the R project webpage `http://www.r-project.org` is a link to contributed documentation. There are quite a few documents of significant size, including a few that have made it into book form.

The R mailing list: The R mailing list is full of information. Questions should only be asked after a reading of the FAQ or you are likely to have a rather terse response.

Books on R (Also numerous S-plus titles)