What is R?

R is an integrated statistical environment offering:

- extensive data and statistical analysis tools
- comprehensive linear and non-linear modelling facilities
- graphical facilities for data analysis and display
- a powerful programming language

What are S and S-PLUS?

- S was an ISE developed at Bell Labs (1976)
- S-PLUS is the commercial version of S
- R is a free reimplementation of S (1995)

Finding R

- already installed on lab computers
- The Comprehensive R Archive Network (CRAN)

  `cran.stat.sfu.ca` or `cran.r-project.org`

  Get the appropriate "Precompiled Binary Distribution."

Using R: Starting and Stopping

Under UNIX or Linux:

```
[yourdir]$ mkdir project1
[yourdir]$ cd project1
[project1]$ R

R : Copyright 2004, The R Foundation for Statistical Computing
Version 2.0.1  (2004-11-15), ISBN 3-900051-07-0

[ . . . ]

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for a HTML browser interface to help.
Type 'q()' to quit R.

> q()
Save workspace image? [y/n/c]: y
[project1]$ ls -a
./          ../         .RData      .Rhistory
[project1]$
```

Using R: A Sample Session

- Follow the "Introduction to R" link on the course webpage.
- Try it out (either on a version installed at home or in MSRC).
- If you make mistakes: arrow keys move through and edit command history.

Getting Help

- Use `help()`:

```
> help(q)
quit             package:base             R Documentation

Terminate an R Session

Description:

     The function 'quit' or its alias 'q' terminate
     the current R session.

Usage:

     quit(save = "default", status = 0, runLast = TRUE)

[ . . . ]
:
```

  At the ":" prompt, press SPACE for more, `b` to go back, or `q` to quit.

- Use `help.search()`:

```
> help.search("time series")
avgp(GeneTS)            Average Periodogram for
                        Multiple (Genetic) Time Series
[ . . . ]
```

- Use `help.start()` to launch web browser.
- Or, just visit `www.r-project.org`, and look at the "Documentation."

Objects in R

Everything in R (a vector of numbers, a list, a data frame, the result of a regression) is an *object* whose *class* determines how it behaves.

- Numbers (actually, numeric vectors):

```
> x <- 4
> class(x)
[1] "numeric"
> print(x)
[1] 4
> summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      4       4       4       4       4       4
> plot(x)
>
```

- Fitted models:

```
> l <- lm(demand ~ Time, data=BOD)
> class(l)
[1] "lm"
> print(l)
[ . . . ]
Coefficients:
(Intercept)         Time
      8.521        1.721
> summary(l)
[ . . . ]
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   8.5214     2.6589   3.205   0.0328 *
Time          1.7214     0.6387   2.695   0.0544 .
[ . . . ]
> plot(l)
> unclass(l)
[ . . . ]
>
```

Saving Across Sessions

```
[project1]$ R
[ . . . ]
> x <- round(runif(10))
> x
 [1] 0 1 1 0 0 1 1 0 1 0
> l <- lm(demand ~ Time, data=BOD)
> coef(l)
(Intercept)        Time
   8.521429    1.721429
> objects()
[1] "l" "x"
> q()
Save workspace image? [y/n/c]: y
[project1]$
```

Many years pass . . .

```
[project1]$ R
[ . . . ]
> objects()
[1] "l" "x"
> x
 [1] 0 1 1 0 0 1 1 0 1 0
> coef(l)
(Intercept)        Time
   8.521429    1.721429
> rm(x)
> objects()
[1] "l"
>
```


Vectors

- building with `c`

```
> x <- c(4.5, -1, 3e-2, sqrt(15))
> x
[1]  4.500000 -1.000000  0.030000  3.872983
> x[4]
[1] 3.872983
>
```

- vector arithmetic

```
> x^2+1
[1] 21.2500  2.0000  1.0009 16.0000
> mean(x)
[1] 1.850746
>
```

- sequences using ":" and `seq`

```
> 4:10
[1]  4  5  6  7  8  9 10
> 8:2
[1] 8 7 6 5 4 3 2
> 1:5*10
[1] 10 20 30 40 50
> seq(-2, 9, by=2)
[1] -2  0  2  4  6  8
>
```

- (pseudo-)random vectors

```
> rnorm(5, mean=2, sd=.1)
[1] 1.885288 2.107558 2.056270 2.048590 1.955928
> rpois(3, lambda=6)
[1] 3 4 4
>
```


Non-Numeric Vectors

- logical (boolean) vectors

```
> x <- round(10*runif(5))
> x
[1] 1 6 3 3 4
> x > 3
[1] FALSE  TRUE FALSE FALSE  TRUE
> l <- x > 3
> l
[1] FALSE  TRUE FALSE FALSE  TRUE
> x[l]
[1] 6 4
> sum(l)
[1] 2
>
> plant.height
[1] 13.41    NA 13.95 11.17    NA
> is.na(plant.height)
[1] FALSE  TRUE FALSE FALSE  TRUE
> plant.height[is.na(plant.height)] <- 0
> plant.height
[1] 13.41  0.00 13.95 11.17  0.00
>
```

- character vectors

```
> gender <- c("M","M","F","M","F","F","M")
> gender
[1] "M" "M" "F" "M" "F" "F" "M"
> table(gender)
gender
F M
3 4
>
```


Indexing Vectors

1. by a vector of positive integers

```
> x
[1]  0  8  9  7  4  2 10  0  2  1
> x[4]
[1] 7
> x[c(2,4,8)]
[1] 8 7 0
>
```

2. by a vector of negative integers

```
> x[c(-1,-2,-9)]
[1]  9  7  4  2 10  0  1
>
```

3. by a logical vector

```
> x[c(F,F,F,T,F,F,F,T,T,T)]
[1] 7 0 2 1
>
```

4. by a vector of character strings

```
> age <- c(10,2,15)
> names(age) <- c("Nancy", "Bill", "Anne")
> age
Nancy  Bill  Anne
   10     2    15
> age[c("Nancy","Anne")]
Nancy  Anne
   10    15
>
```

Factors

- Character vector with allowed set of levels:

```
> gender
[1] "M" "M" "F" "M" "F" "F" "M"
> gender <- factor(gender)
> gender
[1] M M F M F F M
Levels: F M
> gender[2] <- "G"
Warning message: invalid factor level [ . . . ]
> gender
[1] M    <NA> F    M    F    F    M
Levels: F M
> gender[2] <- "F"
> gender
[1] M F F M F F M
Levels: F M
>
> meal <- factor(c("chicken","beef","beef","chicken"),
                 levels=c("chicken","beef","vegetarian"))
> table(meal)
meal
   chicken       beef vegetarian
         2          2          0
>
```

- Ordered factors, too:

```
> treatment <- ordered(c("placebo","lowdose","lowdose",
                         "placebo","highdose"),
                       levels=c("placebo","lowdose","highdose"))
> treatment
[1] placebo  lowdose  lowdose  placebo  highdose
Levels: placebo < lowdose < highdose
>
```


Data Frames

R's usual data table object.

- Library of example datasets:

```
> data()
Data sets in package 'base':

Formaldehyde    Determination of Formaldehyde
HairEyeColor    Hair and Eye Color of Statistics Students
[ . . . ]

> class(Formaldehyde)
[1] "data.frame"
> help(Formaldehyde)
[ . . . ]
> Formaldehyde
  carb optden
1  0.1  0.086
2  0.3  0.269
3  0.5  0.446
4  0.6  0.538
5  0.7  0.626
6  0.9  0.782
> summary(Formaldehyde)
      carb            optden
 Min.   :0.1000   Min.   :0.0860
 1st Qu.:0.3500   1st Qu.:0.3132
 Median :0.5500   Median :0.4920
 Mean   :0.5167   Mean   :0.4578
 3rd Qu.:0.6750   3rd Qu.:0.6040
 Max.   :0.9000   Max.   :0.7820
>
```


Data Frames

- Building them manually:

```
> people <- data.frame(age=c(24,20,18,20,19),
                       gender=c("M","F","F","M","F"),
                       height=c(61, 55, 52, 57, 57))
> people
  age gender height
1  24      M     61
2  20      F     55
3  18      F     52
4  20      M     57
5  19      F     57
> class(people$gender)
[1] "factor"
> row.names(people)
[1] "1" "2" "3" "4" "5"
> row.names(people) <- c("Stan","Ruojia","Sam",
                         "Ahmed","Felicia")
> people
        age gender height
Stan     24      M     61
Ruojia   20      F     55
Sam      18      F     52
Ahmed    20      M     57
Felicia  19      F     57
>
```


Data Frames: Using Them

- Summarizing:

```
> summary(people)
      age       gender     height
 Min.   :18.0   F:3    Min.   :52.0
 1st Qu.:19.0   M:2    1st Qu.:55.0
 Median :20.0          Median :57.0
 Mean   :20.2          Mean   :56.4
 3rd Qu.:20.0          3rd Qu.:57.0
 Max.   :24.0          Max.   :61.0
>
```

- Selecting parts:

```
> people
        age gender height
Stan     24      M     61
Ruojia   20      F     55
Sam      18      F     52
Ahmed    20      M     57
Felicia  19      F     57
> people$age
[1] 24 20 18 20 19
> people[[2]]
[1] M F F M F
Levels: F M
> people[4,1]
[1] 20
> people[5,]
        age gender height
Felicia  19      F     57
> people[people$age >= 20 & people$gender == "M",
         c("age","height")]
      age height
Stan   24     61
Ahmed  20     57
>
```

Data Frames: Using Them

- Using `attach` and `detach`:

```
> age
Error: Object "age" not found
> attach(people)
> age
[1] 24 20 18 20 19
> gender
[1] M F F M F
Levels: F M
> table(gender)
gender
F M
3 2
> tapply(age, gender, mean)
 F  M
19 22
> detach()
> age
Error: Object "age" not found
>
```

- Building models:

```
> names(people)
[1] "age"    "gender" "height"
> l <- lm(height ~ age + gender, data=people)
> summary(l)
[ . . . ]
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  33.7667    12.4922   2.703    0.114
age           1.1000     0.6545   1.681    0.235
genderM       1.0333     2.7248   0.379    0.741
[ . . . ]
>
```


Getting Data into R

- From the keyboard: `c()` or `scan()`

```
> colour <- c("red", "blue", "yellow", "green")
> preference <- scan()
1: 10 2 8 6
5:
Read 4 items
> colour.type <- factor(scan(what=""))
1: primary primary primary secondary
5:
Read 4 items
> colours <- data.frame(colour=colour,
                        preference=preference,
                        type=colour.type)
> colours
  colour preference      type
1    red         10   primary
2   blue          2   primary
3 yellow          8   primary
4  green          6 secondary
>
```

- Interactively with `fix` or `edit`:

```
> fix(colours)
```

or

```
> new.colours <- edit(colours)
```


Getting Data into R

- From files: using `scan(...)`

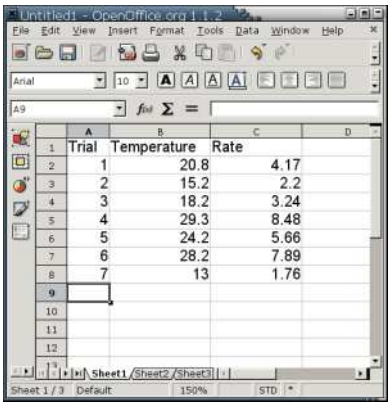
```
> system("cat numbers")
10 15 15
17 21 10.3 -8
> scan("numbers")
Read 7 items
[1] 10.0 15.0 15.0 17.0 21.0 10.3 -8.0
> system("cat strings")
one
string per
line
> scan("strings", sep="\n")
Error in scan("strings", sep = "\n") :
  "scan" expected a real, got "one"
> scan("strings", what="example", sep="\n")
Read 3 items
[1] "one"        "string per" "line"
>
```


Getting Data into R

- From files: using `read.table(...)`

```
> system("cat table")
day high.temp low.temp snowfall
Fri    0         -2       5
Sat    0         -4       2
Sun    1         -4       0
Mon    1         -1       0
> forecast <- read.table("table", header=T)
> forecast
  day high.temp low.temp snowfall
1 Fri         0       -2        5
2 Sat         0       -4        2
3 Sun         1       -4        0
4 Mon         1       -1        0
> class(forecast$day)
[1] "factor"
>
```

Exchanging Data with Spreadsheet



1. Save as a CSV (Comma-Separated Value) file.

```
> system("cat experiment.csv")
"Trial","Temperature","Rate"
1,20.8,4.17
2,15.2,2.2
3,18.2,3.24
4,29.3,8.48
5,24.2,5.66
6,28.2,7.89
7,13,1.76
>
```


Exchanging Data with Spreadsheet

2. Read using `read.csv`

```
> experiment <- read.csv("experiment.csv")
> experiment
  Trial Temperature Rate
1     1        20.8 4.17
2     2        15.2 2.20
3     3        18.2 3.24
4     4        29.3 8.48
5     5        24.2 5.66
6     6        28.2 7.89
7     7        13.0 1.76
>
```

3. Write using `write.table`

```
> experiment$Temperature <- 1.8*experiment$Temperature+32
> write.table(experiment, "newfile.csv", sep=",", col.names=NA)
>
```

See `help(write.table)` for details and examples.

Working with Columns

- adding, removing, and transforming:

```
> people
        age gender height
Stan     24      M     61
Ruojia   20      F     55
Sam      18      F     52
Ahmed    20      M     57
Felicia  19      F     57
> names(people)
[1] "age"    "gender" "height"
> names(people)[3] <- "height.inches"
> people$height.cms <- people$height.inches * 2.54
> people$age <- people$age + 0.5
> people$gender <- NULL
> people
         age height.inches height.cms
Stan    24.5            61     154.94
Ruojia  20.5            55     139.70
Sam     18.5            52     132.08
Ahmed   20.5            57     144.78
Felicia 19.5            57     144.78
>
```


Working with Columns

- acting on several at once:

```
> iris
  Sepal.Len Sepal.Wid Petal.Len Petal.Wid Species
1       5.1       3.5       1.4       0.2  setosa
2       4.9       3.0       1.4       0.2  setosa
[ . . . ]
> iris[,1:4] <- iris[,1:4]*10  # convert cm to mm
> iris
  Sepal.Len Sepal.Wid Petal.Len Petal.Wid Species
1        51        35        14         2  setosa
2        49        30        14         2  setosa
[ . . . ]
> mean(iris[,1:4])
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
    58.43333     30.57333     37.58000     11.99333
> hist(iris[,1:4])
Error in hist.default(iris[, 1:4]) : 'x' must be numeric
> par(ask=T); lapply(iris[,1:4], hist)
Hit <Return> to see next plot:
[ . . . ]
>
```

Working with Rows

- adding, removing, and replacing

```
> (people <- rbind(people,Joe=list(25,"M",64)))
       age gender height
Stan     24      M     61
Ruojia   20      F     55
Sam      18      F     52
Ahmed    20      M     57
Felicia  19      F     57
Joe      25      M     64
> people[row.names(people) != "Stan",]  # or just people[-1,]
       age gender height
Ruojia   20      F     55
Sam      18      F     52
Ahmed    20      M     57
Felicia  19      F     57
Joe      25      M     64
> people[1,] <- list(18,"M",58); people
       age gender height
Stan     18      M     58
Ruojia   20      F     55
Sam      18      F     52
Ahmed    20      M     57
Felicia  19      F     57
Joe      25      M     64
>
```


Analysis by Group

```
> mtcars
                  mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4        21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag    21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710       22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
[ . . . ]
> tapply(mtcars$mpg, mtcars$am, mean)
       0        1
17.14737 24.39231
> tapply(mtcars$mpg, list(mtcars$am, mtcars$gear), mean)
         3      4     5
0 16.10667 21.050    NA
1       NA 26.275 21.38
> by(mtcars[,c("mpg","hp","qsec")],
     list(manual=mtcars$am, gears=mtcars$gear), mean)
manual: 0
gears: 3
      mpg        hp      qsec
 16.10667 176.13333  17.69200
------------------------------------------------------------
manual: 0
gears: 4
    mpg      hp    qsec
 21.050 100.750  20.025
------------------------------------------------------------
[ . . . ]
> aggregate(mtcars[,c("mpg","hp","qsec")],
            list(am=mtcars$am, gear=mtcars$gear), mean)
  am gear      mpg       hp   qsec
1  0    3 16.10667 176.1333 17.692
2  0    4 21.05000 100.7500 20.025
3  1    4 26.27500  83.8750 18.435
4  1    5 21.38000 195.6000 15.640
>
```


Reshaping

```
> michaelson
  expt run1 run2 run3 run4 run5       run20
1    1  850  740  900 1070  930         960
2    2  960  940  960  940  880         800
3    3  880  880  880  860  720  . . .  840
4    4  890  810  810  820  800         780
5    5  890  840  780  810  760         870
> (m.long <- reshape(michaelson, idvar="expt",
                     varying=list(paste("run",1:20,sep="")),
                     timevar="run", v.names="speed",
                     direction="long"))
    expt run speed
1.1    1   1   850
2.1    2   1   960
3.1    3   1   880
4.1    4   1   890
5.1    5   1   890
1.2    1   2   740
2.2    2   2   940
[ . . . ]
> summary(aov(speed ~ factor(run) + factor(expt), data=m.long))
             Df Sum Sq Mean Sq F value   Pr(>F)
factor(run)  19 113344    5965  1.1053 0.363209
factor(expt)  4  94514   23629  4.3781 0.003071 **
Residuals    76 410166    5397
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
>
```


Cross-Tabulation

Questionnaire Data:

```
> names(quest)
 [1] "prereq.course" "prereq.when"   "stat305"       "stat404"
 [5] "stat441"       "stat442"       "stat450"       "windows"
 [9] "unix"          "s"             "r"             "sas"
[13] "matlab"        "emacs"         "c"             "java"
[17] "perl"
> attach(quest)
>
```

Frequency tables:

- One factor:

```
> prereq.course
[1] n     t     y     t     y     y     y     y     y
[ . . . ]
Levels: n < t < equiv < y
> table(prereq.course)
prereq.course
    n     t equiv     y
    4     3     2    38
>
```

- Two factor:

```
> table(prereq.course, r)
              r
prereq.course 0  1  2  3
        n     2  2  0  0
        t     0  2  1  0
        equiv 2  0  0  0
        y     2  4 20 12
>
```

Cross-Tabulation

- Multi-factor:
  - Using `table`:

```
> table(prereq.course,stat305,stat404,stat441)
, , stat404 = -, stat441 = -

             stat305
prereq.course -  n  y
        n      1  0  0
        t      0  0  1
        equiv  1  0  0
        y      0  0  2

, , stat404 = n, stat441 = -

             stat305
prereq.course -  n  y
        n      0  0  0
        t      0  0  0
        equiv  0  0  0
        y      0  0  0
[ . . . about 60 more lines . . . ]
>
```

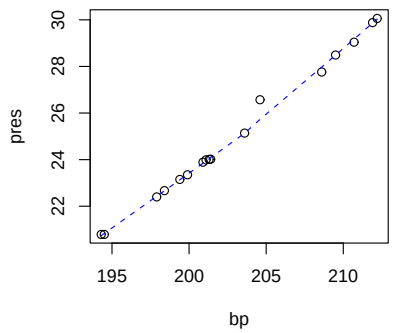

Cross-Tabulation

- Multi-factor:
  - Summarized in a data frame:

```
> aggregate(row.names(quest),
            list(prereq.course,stat305,stat404,stat441),
            length)
   Group.1 Group.2 Group.3 Group.4 factor(x)
1        n       -       -       -         1
2    equiv       -       -       -         1
3        t       y       -       -         1
4        y       y       -       -         2
[ . . . ]
> aggregate(list(freq=row.names(quest)),
            list(prereq=prereq.course,s305=stat305,
                 s404=stat404,s441=stat441),
            length)
   prereq s305 s404 s441 freq
1       n    -    -    -    1
2   equiv    -    -    -    1
3       t    y    -    -    1
4       y    y    -    -    2
5       y    y    y    -    1
6       n    n    n    n    2
7       t    n    n    n    1
8   equiv    n    n    n    1
9       y    n    n    n    2
10      n    y    n    n    1
11      t    y    n    n    1
12      y    y    n    n    8
13      y    y    y    n   10
14      y    y    -    y    1
15      y    y    y    y   14
>
```


Forbes' Alps Data

```
> library(MASS)
> forbes
      bp  pres
1  194.5 20.79
2  194.3 20.79
3  197.9 22.40
[ . . . ]
16 211.9 29.88
17 212.2 30.06
> summary(forbes)
       bp             pres
 Min.   :194.3   Min.   :20.79
 1st Qu.:199.4   1st Qu.:23.15
 Median :201.3   Median :24.01
 Mean   :203.0   Mean   :25.06
 3rd Qu.:208.6   3rd Qu.:27.76
 Max.   :212.2   Max.   :30.06
> attach(forbes)
> plot(bp, pres)
> lines(lowess(bp, pres), lty="dashed", col="blue")
>
```



Simple Linear Regression

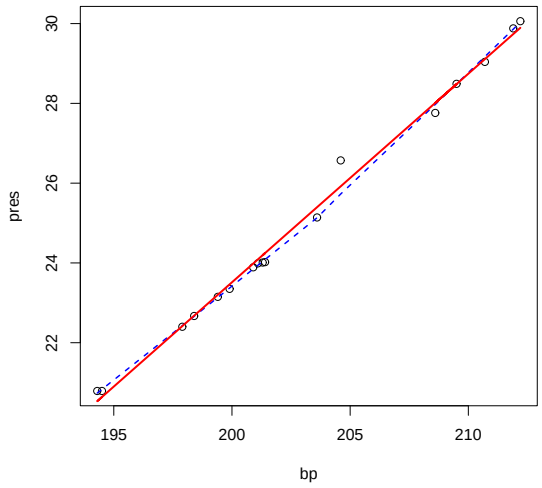
```
> forbes.lm <- lm(pres ~ bp, data=forbes)
> forbes.lm

Call:
lm(formula = pres ~ bp, data = forbes)

Coefficients:
(Intercept)           bp
   -81.0637       0.5229

> summary(forbes.lm)

Call:
lm(formula = pres ~ bp, data = forbes)

Residuals:
     Min       1Q   Median       3Q      Max
-0.25717 -0.11246 -0.05102  0.14283  0.64994

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -81.06373    2.05182  -39.51   <2e-16 ***
bp            0.52289    0.01011   51.74   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.2328 on 15 degrees of freedom
Multiple R-Squared: 0.9944,    Adjusted R-squared: 0.9941
F-statistic:  2677 on 1 and 15 DF,  p-value: < 2.2e-16

>
```

Simple Linear Regression

```
> coef(forbes.lm)
(Intercept)          bp
-81.0637271   0.5228924
> resid(forbes.lm)
           1            2            3            4
 0.151155176  0.255733656 -0.016678987 -0.008125187
[ . . . ]
> fitted(forbes.lm)  # or predict(forbes.lm)
       1        2        3        4        5        6
20.63884 20.53427 22.41668 22.67813 23.20102 23.46246
[ . . . ]
> predict(forbes.lm, data.frame(bp=c(197,207)))
       1        2
21.94608 27.17500
>

> model.matrix(forbes.lm)
   (Intercept)    bp
1            1 194.5
2            1 194.3
3            1 197.9
4            1 198.4
5            1 199.4
[ . . . ]
13           1 209.5
14           1 208.6
15           1 210.7
16           1 211.9
17           1 212.2
attr(,"assign")
[1] 0 1
>
```

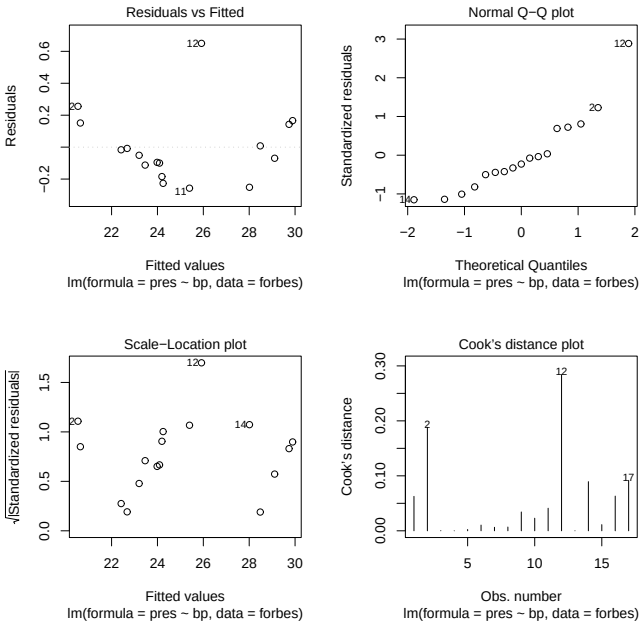

Simple Linear Regression

```
> lines(bp, fitted(forbes.lm), col="red")
```



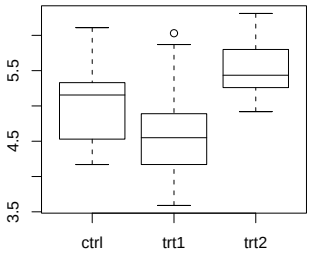
Regression Diagnostics

```
> plot(forbes.lm)
```



Plant Growth

```
> PlantGrowth
   weight group
1    4.17  ctrl
2    5.58  ctrl
[ . . . ]
29   5.80  trt2
30   5.26  trt2
> summary(PlantGrowth)
     weight       group
 Min.   :3.590   ctrl:10
 1st Qu.:4.550   trt1:10
 Median :5.155   trt2:10
 Mean   :5.073
 3rd Qu.:5.530
 Max.   :6.310
> attach(PlantGrowth)
> tapply(weight,group,mean)
 ctrl  trt1  trt2
5.032 4.661 5.526
> tapply(weight,group,sd)
     ctrl      trt1      trt2
0.5830914 0.7936757 0.4425733
> plot(group,weight)
>
```



One-Factor ANOVA

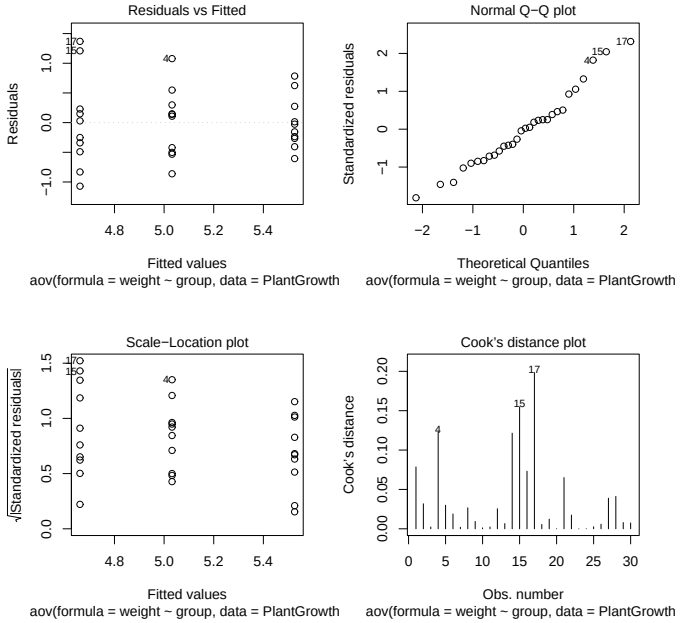
```
> pg.aov <- aov(weight ~ group, data=PlantGrowth)
> pg.aov
Call:
   aov(formula = weight ~ group, data = PlantGrowth)

Terms:
                    group Residuals
Sum of Squares    3.76634  10.49209
Deg. of Freedom         2        27

Residual standard error: 0.6233746
Estimated effects may be unbalanced
> summary(pg.aov)
            Df  Sum Sq Mean Sq F value  Pr(>F)
group        2  3.7663  1.8832  4.8461 0.01591 *
Residuals   27 10.4921  0.3886
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
> coef(pg.aov)
(Intercept)   grouptrt1   grouptrt2
      5.032      -0.371       0.494
> resid(pg.aov)
     1      2      3      4      5      6      7
-0.862  0.548  0.148  1.078 -0.532 -0.422  0.138  [ . . . ]
>
```


ANOVA Model Matrix

```
> model.matrix(pg.aov)
   (Intercept) grouptrt1 grouptrt2
1            1         0         0
2            1         0         0
3            1         0         0
4            1         0         0
5            1         0         0
6            1         0         0
7            1         0         0
8            1         0         0
9            1         0         0
10           1         0         0
11           1         1         0
12           1         1         0
13           1         1         0
14           1         1         0
15           1         1         0
16           1         1         0
17           1         1         0
18           1         1         0
19           1         1         0
20           1         1         0
21           1         0         1
22           1         0         1
23           1         0         1
24           1         0         1
25           1         0         1
26           1         0         1
27           1         0         1
28           1         0         1
29           1         0         1
30           1         0         1
[ . . . ]
>
```


Regression Diagnostics

```
> plot(pg.aov)
```



Compare `aov` and `lm`

```
> pg.aov <- aov(weight ~ group, data=PlantGrowth)
> pg.lm <- lm(weight ~ group, data=PlantGrowth)
> pg.aov
Call:
   aov(formula = weight ~ group, data = PlantGrowth)

Terms:
                    group Residuals
Sum of Squares    3.76634  10.49209
Deg. of Freedom         2        27

Residual standard error: 0.6233746
Estimated effects may be unbalanced
> pg.lm

Call:
lm(formula = weight ~ group, data = PlantGrowth)

Coefficients:
(Intercept)    grouptrt1    grouptrt2
      5.032       -0.371        0.494
>
```

Compare `aov` and `lm`

```
> summary(pg.aov)
            Df  Sum Sq Mean Sq F value  Pr(>F)
group        2  3.7663  1.8832  4.8461 0.01591 *
Residuals   27 10.4921  0.3886
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
> summary(pg.lm)

Call:
lm(formula = weight ~ group, data = PlantGrowth)

Residuals:
    Min      1Q  Median      3Q     Max
-1.0710 -0.4180 -0.0060  0.2627  1.3690

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   5.0320     0.1971  25.527   <2e-16 ***
grouptrt1    -0.3710     0.2788  -1.331   0.1944
grouptrt2     0.4940     0.2788   1.772   0.0877 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.6234 on 27 degrees of freedom
Multiple R-Squared: 0.2641,Adjusted R-squared: 0.2096
F-statistic: 4.846 on 2 and 27 DF,  p-value: 0.01591


>
```

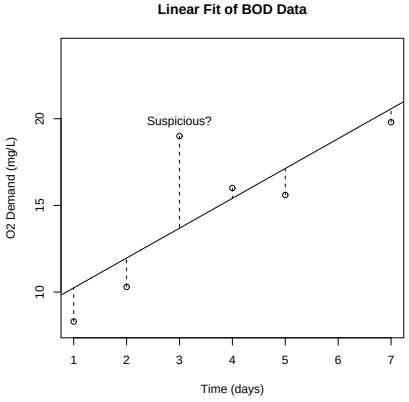

Compare `aov` and `lm`

```
> coef(pg.aov)
(Intercept)   grouptrt1   grouptrt2
      5.032      -0.371       0.494
> coef(pg.lm)
(Intercept)   grouptrt1   grouptrt2
      5.032      -0.371       0.494
> resid(pg.aov)
     1      2      3      4      5      6      7
-0.862  0.548  0.148  1.078 -0.532 -0.422  0.138 [ . . . ]
> resid(pg.lm)
     1      2      3      4      5      6      7
-0.862  0.548  0.148  1.078 -0.532 -0.422  0.138 [ . . . ]
>
```


R Graphics

- Base Graphics
  - high-level, plot-generating functions (e.g., `plot`, `boxplot`)
  - lower-level, annotating functions (e.g., `lines`, `points`, `text`)

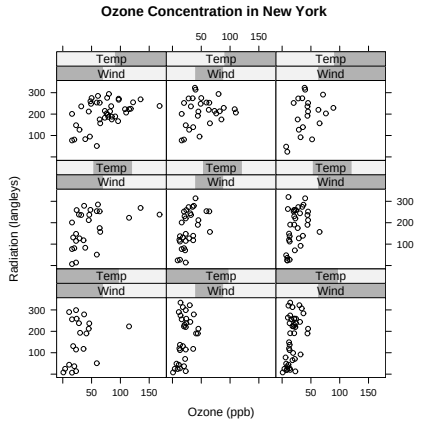
```
> plot(Time, demand,
       main="Linear Fit of BOD Data",  ylim=c(8,24),
       xlab="Time (days)", ylab="O2 Demand (mg/L)")
> abline(coef(l))
> segments(Time, demand, Time, fitted(l), lty="dashed")
> text(Time[3], demand[3]+par("cxy")[2], "Suspicious?")
NULL
>
```



R Graphics

- Lattice Graphics*
  - Created “all at once”
  - Well-suited for multivariate data

```
> library(lattice)
> data(environmental)
> environmental$Wind <- equal.count(environmental$wind, 3)
> environmental$Temp <- equal.count(environmental$temp, 3)
> xyplot(radiation ~ ozone | Wind * Temp, data=environmental,
         main="Ozone Concentration in New York",
         xlab="Ozone (ppb)", ylab="Radiation (langleys)")
>
```

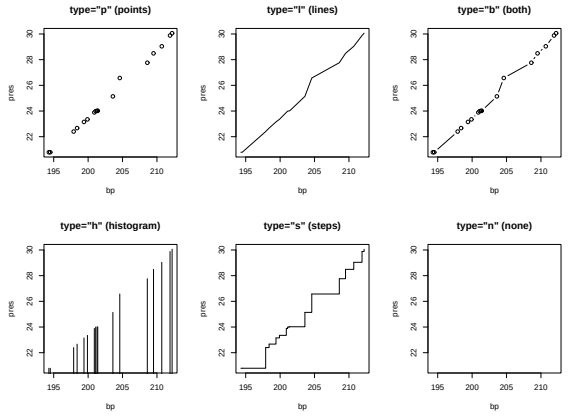


*Also called “Trellis Graphics” in the S-PLUS world.

`plot`: Bivariate Scatterplots

Form: `> plot(`*numeric*`, `*numeric*`, type="`*?*`")`

Various types:



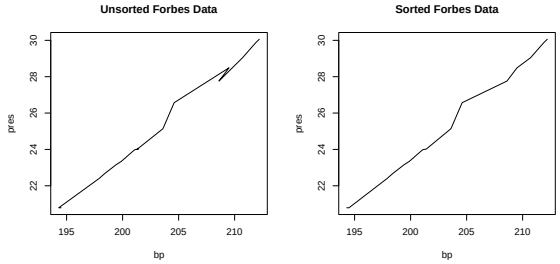
all generated with

```
> plot(bp, pres, type="x")  # bp and pres from the forbes data
```


Sorting Obs. for Line Graphs

Actually, the Forbes data looks different:

```
> library(MASS)
> attach(forbes)
> plot(bp, pres, type="l", main="Unsorted Forbes Data")
> detach()
> forbesS <- forbes[sort.list(forbes$bp),]
> attach(forbesS)
> plot(bp, pres, type="l", main="Sorted Forbes Data")
>
```

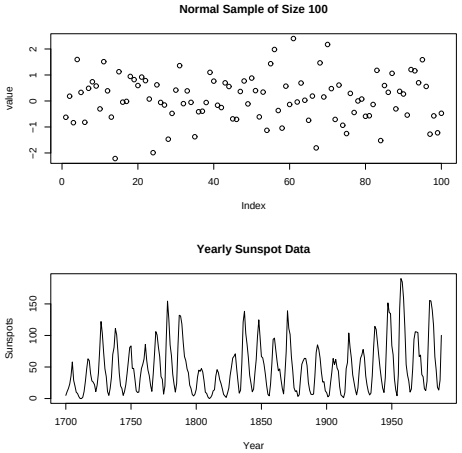



```
> forbes$bp
 [1] 194.5 194.3 197.9 198.4 199.4 199.9 200.9 201.1 201.4 [ . . . ]
> sort(forbes$bp)
 [1] 194.3 194.5 197.9 198.4 199.4 199.9 200.9 201.1 201.3 [ . . . ]
> sort.list(forbes$bp)
 [1]  2  1  3  4  5  6  7  8 10  9 11 12 14 13 15 16 17
>
```


`plot`: Univariate Scatterplots

Form: `> plot(`*numeric vector or time series*`)`

```
> plot(rnorm(100), main="Normal Sample of Size 100",
       ylab="value")
> sunspot.year
Time Series:
Start = 1700
End = 1988
Frequency = 1
  [1]   5.0  11.0  16.0  23.0  36.0  58.0  29.0  [ . . . ]
> plot(sunspot.year, main="Yearly Sunspot Data",
       ylab="Sunspots", xlab="Year")
>
```



`plot`: Barplots

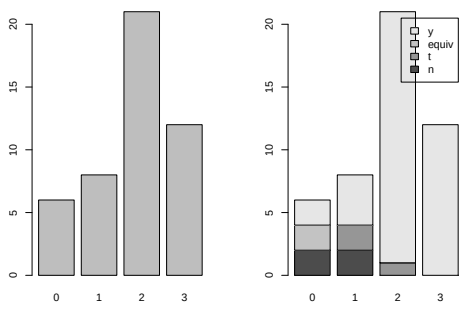
Form: `> plot(`*factor[*, *factor]*`)`

- barplot of a single factor

```
> attach(quest)
> table(r)
r
 0  1  2  3
 6  8 21 12
> plot(r)
>
```

- split barplot of two factors

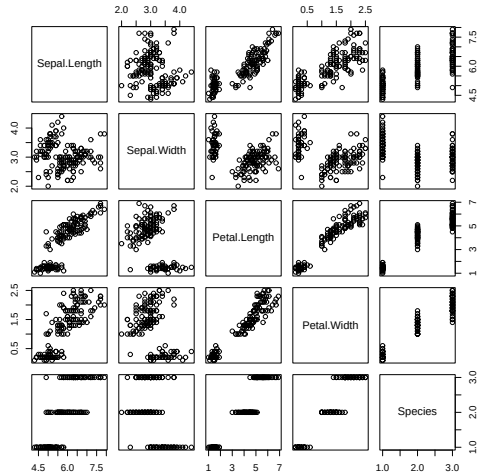
```
> table(prereq.course)
prereq.course
    n     t equiv     y
    4     3     2    38
> plot(r, prereq.course)
>
```



Other Forms of `plot`

- Boxplots by Group (see slide 32)
  Form: `> plot(`*factor*`,` *numeric*`)`
- Diagnostic Plots (see slides 31 and 35)
  Form: `> plot(`*fitted model*`)`
- Data Frames
  Form: `> plot(`*data frame*`)`

```
> plot(iris)
```

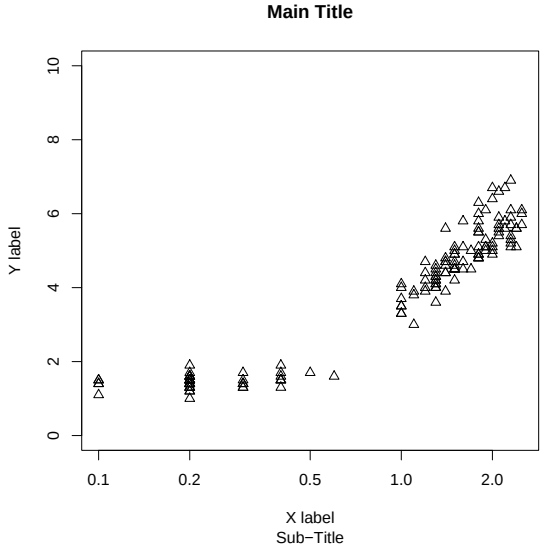


`plot` Parameters

- `type`: plot type
- `main`, `sub`: main and sub-titles
- `xlab`, `ylab`: axis labels
- `xlim=c(-5,5)`, `ylim=c(0,20)`: axis ranges
- `log="x"`, `="y"`, or `="xy"`: use logarithmic axes
- `col="red"`: colour
- `lty="dashed"`, `lwd=2`: line type and width for line plots
- `pch=1` or `pch="A"`: symbols or characters for point plots

Example of Some Parameters

```
> plot(Petal.Width, Petal.Length,
       main="Main Title", sub="Sub-Title",
       xlab="X label", ylab="Y label",
       ylim=c(0,10), log="x", pch=2)
>
```



Getting Help on `plot`

```
> help(plot)
plot             package:graphics             R Documentation

Generic X-Y Plotting

Description:

     Generic function for plotting of R objects.  For more
     details about the graphical parameter arguments, see 'par'.

[ . . . ]
Details:

     For simple scatter plots, 'plot.default' will be
     used. However, there are 'plot' methods for many R objects,
     including 'function's, 'data.frame's, 'density' objects,
     etc.  Use 'methods(plot)' and the documentation for these.

[ . . . ]
See Also:

     'plot.default', 'plot.formula' and other methods; 'points',
     'lines', 'par'.
>
```

Generic Functions and Methods

Some functions are *generic*: they use a different version when used on different objects:

```
> lm.object <- lm(y ~ x, data=mydata)
> aov.object <- aov(y ~ x, data=mydata)
> summary(lm.object)   # actually runs summary.lm(lm.object)
> summary(aov.object)  # actually runs summary.aov(aov.object)
> plot(lm.object)      # actually runs plot.lm(lm.object)
> plot(x, y)           # actually runs plot.default(x, y)
>
```

Generic functions will have one or more methods, a default method and object-specific methods:

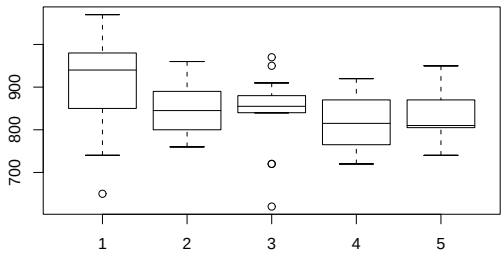
```
> methods(plot)
[ . . . ]
 [7] plot.data.frame*    plot.decomposed.ts* plot.default
[10] plot.dendrogram*    plot.density        plot.ecdf
[13] plot.factor*        plot.formula*       plot.hclust*
[16] plot.histogram*     plot.isoreg*        plot.lm
[ . . . ]
> methods(summary)
[ . . . ]
 [4] summary.aov         summary.aovlist     summary.connection
 [7] summary.data.frame  summary.default     summary.ecdf*
[10] summary.factor      summary.glm         summary.infl
[13] summary.lm          summary.loess*      summary.manova
[ . . . ]
> methods(search)
no methods were found
>
```

Getting help:

```
> help(plot)          # get generic, and useless, help
> help(plot.lm)       # get useful help
> help(plot.default)  # ditto
>
```


More High-Level Plots: `boxplot`

```
> morley
    Expt Run Speed
1      1   1   850
2      1   2   740
3      1   3   900
[ . . . ]
98     5  18   800
99     5  19   810
100    5  20   870
> split(morley$Speed, morley$Expt)
$"1"
 [1]  850  740  900 1070  930  850  950  980  980  880 1000  [ . . . ]
$"2"
 [1] 960 940 960 940 880 800 850 880 900 840 830 790 810 880 [ . . . ]
[ . . . ]
> boxplot(split(morley$Speed, morley$Expt))
>
```



`boxplot` Parameters

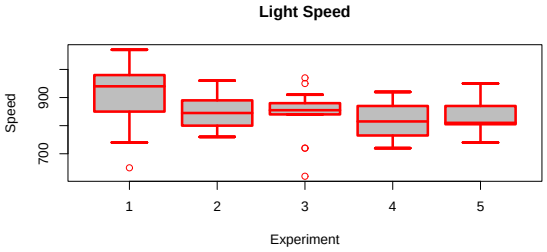
- `plot` parameters: `main`, `sub`, `xlab`, `ylab`, `ylim`, `log`, `lty`, `lwd`, `pch`
- `boxplot`-specific parameters
  - `varwidth=T`: box widths prop. to $\sqrt{n}$
  - `notch=T`: add "confidence notches"
  - `outline=F`: don't draw outliers
  - `names`: box labels
  - `boxwex=0.8`: scale factor for box widths
  - `border="pink"`: colour(s) for box outlines and outlier symbols
  - `col="purple"`: colour(s) for box bodies
  - `horizontal=T`: draw horizontal boxes
- `add=T`: add this plot to current plot

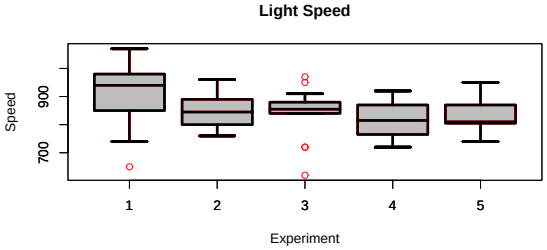
The following looks "best" on a colour display:

```
> boxplot(split(morley$Speed, morley$Expt),
          main="Ugliest Boxplot on Earth",
          xlab="Experiment", ylab="Speed",
          col=1:5, border=6:2, lty=1:5, lwd=1:5,
          pch=c("A","B","C","D","E"), ylim=c(600,1100))
>
```


A Cool Trick: Red Outliers

```
> boxplot(split(morley$Speed, morley$Expt),
          main="Light Speed", xlab="Experiment", ylab="Speed",
          border="red", lwd=3, lty=1, col="grey")
> boxplot(split(morley$Speed, morley$Expt),
          border="black", lwd=3, lty=1, outline=F, add=T)
>
```



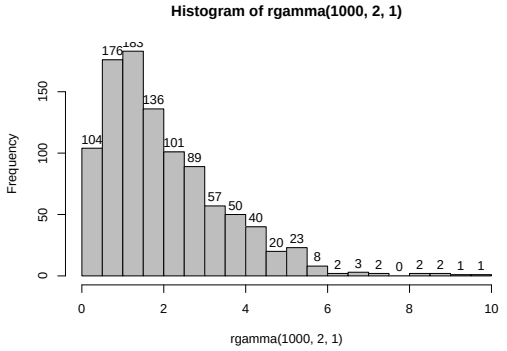


More High-Level Plots: `hist`

Parameters:

- `plot` parameters and `add=T`
- `breaks`: method to select breakpoints (e.g., `"Sturges"`), number of cells, or vector of breakpoints
- `probability=T`: display relative frequencies (probabilities) instead of raw frequencies (counts)
- `labels`: show freqs on top of bars

```
> hist(rgamma(1000,2,1), labels=T, col="grey", breaks=15)
>
```

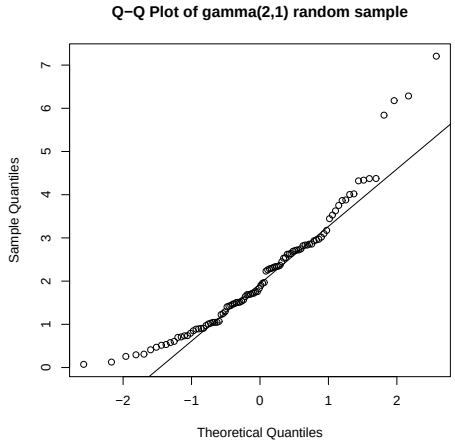


More High-Level Plots: `qqnorm`

Parameters:

- `plot` parameters
- `datax=T`: put data on x-axis instead

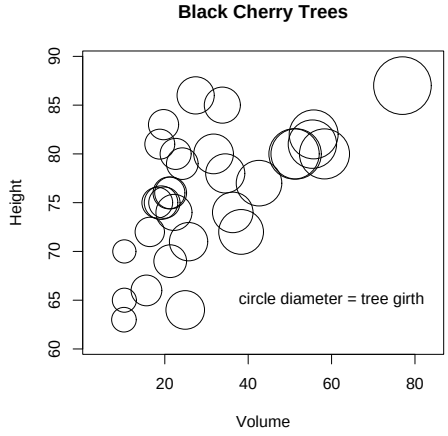
```
> r <- rgamma(100,2,1)
> qqnorm(r, main="Q-Q Plot of gamma(2,1) random sample")
> qqline(r)
>
```



Multivariate Data: `symbols`

Plots (variable-size) circles, squares, rectangles, stars, thermometers, and boxplots.

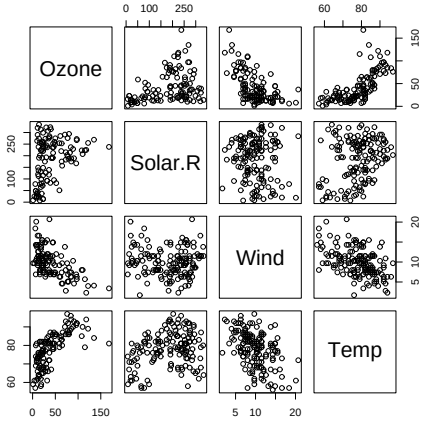
```
> trees
   Girth Height Volume
1    8.3     70   10.3
2    8.6     65   10.3
[ . . . ]
31  20.6     87   77.0
> symbols(Volume, Height, circles=Girth, inches=.3,
          main="Black Cherry Trees")
> text(60,65,"circle diameter = tree girth")
>
```



Multivariate Data: `pairs`

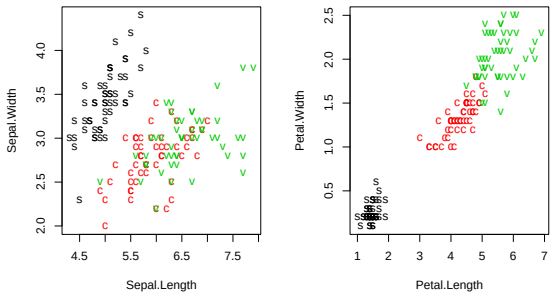
Produces scatterplots of all pairs of columns in a matrix. Good way to quickly gauge relationships between several continuous variables.

```
> airquality  # some New York air quality measurements
    Ozone Solar.R Wind Temp Month Day
1      41     190  7.4   67     5   1
2      36     118  8.0   72     5   2
[ . . . ]
> attach(airquality)
> pairs(cbind(Ozone,Solar.R,Wind,Temp))
>
```



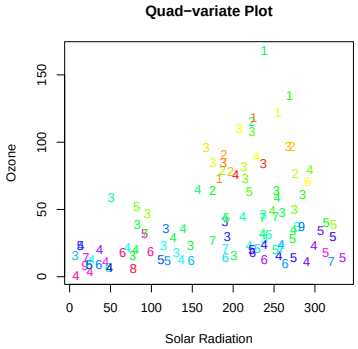
Multivariate Data: `col` and `pch`

```
> 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
> attach(iris)
> levels(Species)
[1] "setosa"     "versicolor" "virginica"
> as.numeric(Species)
  [1] 1 1 1 [ . . . ] 3 3 3
> plot(Sepal.Length, Sepal.Width, col=as.numeric(Species),
       pch=c("s","c","v")[as.numeric(Species)])
> plot(Petal.Length, Petal.Width, col=as.numeric(Species),
       pch=c("s","c","v")[as.numeric(Species)])
>
```



Multivariate Data: `col` and `pch`

```
> attach(airquality)
> names(airquality)
[1] "Ozone"   "Solar.R" "Wind"    "Temp"    "Month"   "Day"
> range(Wind)
[1]  1.7 20.7
> range(Temp)
[1] 56 97
> Wind.char <- as.character(floor(9.9*Wind/max(Wind)))
> Wind.char
  [1] "3" "3" "6" "5" "6" "7" "4" "6" "9" "4" "3" "4" "4" [ . . . ]
> Temp.col <- rev(rainbow(diff(range(Temp))+1))[Temp-min(Temp)+1]
> Temp.col
  [1] "#4900FF" "#006DFF" "#00B6FF" "#FF00FF" "#FF0024" [ . . . ]
> plot(Solar.R, Ozone, pch=Wind.char, col=Temp.col,
     main="Quad-variate Plot",
     xlab="Solar Radiation", ylab="Ozone")
>
```



`points`, `text`, and `identify`

- `points(`*x*`, `*y*`, pch=`*plot chars*`)`

  Adds points `(x[1],y[1])`, . . . , `(x[`n`],x[`n`])`. The `pch` vector is *recycled*: it can specify a symbol number 0–25 or a single character.

  Some other parameters:

  - `type`: as for `plot(...)`
  - `col`: symbol colour (recycled)
  - `bg`: fill colour for `pch=21:25` (recycled)
  - `cex`: symbol scale (recycled)

- `text(`*x*`, `*y*`, `*labels*`)`

  Adds labels to given points.

  Some other parameters:

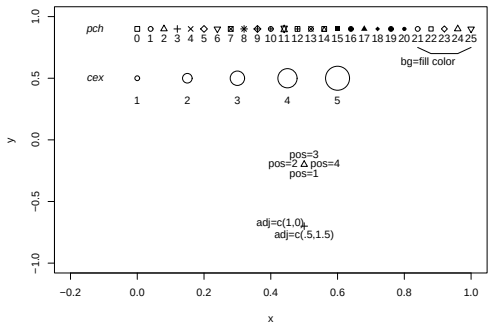
  - `pos=1,2,3,4` position below, left, above, or right
  - `offset=0.5` offset for `pos`
  - `adj=c(0.5,0.5)` alternative to `pos`
  - `col`, `cex`: as above
  - `font=1,2,3,4` for normal, bold, italic, bold italic

- `identify(`*x*`, `*y*`, `*labels*`)`

  Like `text`, but add labels interactively with left mouse button and finish with right button. Also, default labels are $1, 2, \ldots$

`points` and `text`

```
> plot(c(-.2,1),c(-1,1),type="n",xlab="x",ylab="y")
> x <- seq(from=0,to=1,length=26)
> text(-.1, .9, "pch", adj=1, font=3)
> points(x, rep(.9,26), pch=0:25)
> text(x, rep(.9,26), 0:25, pos=1)
> lines(x[c(22,23,25,26)],c(.75,.7,.7,.75))
> text(x[24],.7,"bg=fill colour",adj=c(.8,1.5))
> x <- seq(from=0,to=.6,length=5)
> text(-.1, .5, "cex", adj=1, font=3)
> points(x, rep(.5,5), pch=1, cex=1:5)
> text(x, rep(.5,5), 1:5, pos=1, offset=1.5)
> points(.5,-.2,pch=2)
> text(.5,-.2,paste("pos=",1:4,sep=""),pos=1:4)
> points(.5,-.7,pch=3)
> text(.5,-.7,"adj=c(1,0)",adj=c(1,0))
> text(.5,-.7,"adj=c(.5,1.5)",adj=c(.5,1.5))
```

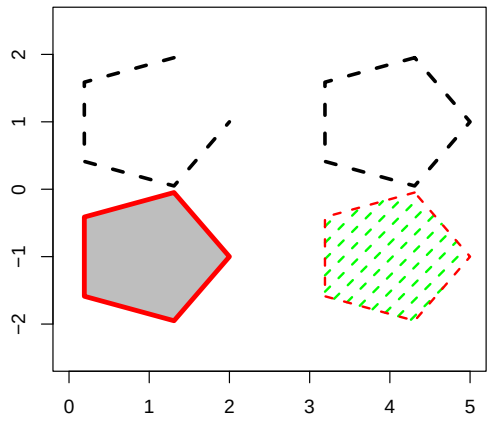


lines and polygon

- `lines(x, y)`
  Draws lines connecting points (x[1],y[1]),...,(x[n],y[n]). Any NA coordinates add a break, making multiple lines.
  Some parameters:
  - `type`: as for `plot(...)`
  - `col`: line colour
  - `lty`: line type
  - `lwd`: line width
- `polygon(x, y)`
  Creates a polygon with given vertices. Any NA coords separate multiple polygons.
  Some parameters:
  - `col`: fill colour
  - `density`, `angle`: hash shading
  - `border`: border colour
  - `lty`, `lwd`: for border (and hash shading)

lines and polygon

```
> plot(0,0, type="n", xlim=c(0,5),ylim=c(-2.5,2.5),
       xlab="", ylab="")
> x <- cos(seq(from=0,to=2*pi,length=6)[-1])
> y <- sin(seq(from=0,to=2*pi,length=6)[-1])
> lines(1+x,1+y, lwd=3, lty="dashed")
> polygon(4+x,1+y, lwd=3, lty="dashed")
> polygon(1+x,-1+y, col="grey", border="red", lwd=4)
> polygon(4+x,-1+y, col="green", density=10, border="red",
          lty="dashed", lwd=2)
```



segments, arrows, and rect

All of form: `segments`(x_1,y_1,x_2,y_2)

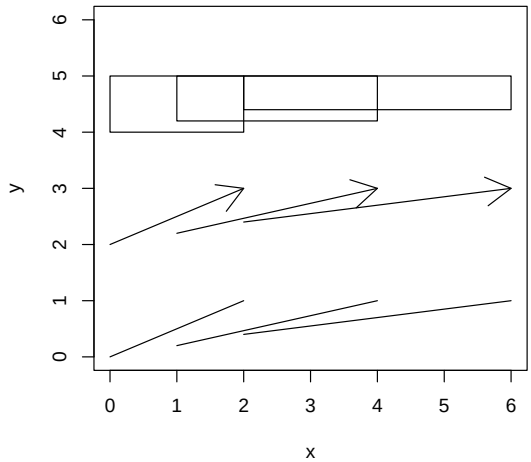
For the pairs of points

$(x_1[1], y_1[1])$ and $(x_2[1], y_2[1])$
$(x_1[2], y_1[2])$ and $(x_2[2], y_2[2])$
⋮
$(x_1[n], y_1[n])$ and $(x_2[n], y_2[n])$

- `segments` draws `n` line segments;
- `arrows` draws `n` arrows (heads at second point);
- `rect` draws `n` rectangles with given diagonally opposite points.

segments, arrows, and rect

```
> plot(c(0,6),c(0,6),type="n",xlab="x",ylab="y")
> x1 <- c(0,1,2); y1 <- c(0,.2,.4)
> x2 <- c(2,4,6); y2 <- c(1,1,1)
> segments(x1,y1,x2,y2)
> arrows(x1,2+y1,x2,2+y2)
> rect(x1,4+y1,x2,4+y2)
```



Other Annotating Functions

- `abline`
  - `abline(`*intercept*`,`*slope*`)`
  - `abline(h=c(0,2,3))`: horizontal lines with given y-values
  - `abline(v=c(-1,3,4))`: vertical lines with given x-values
- `matpoints` and `matlines`

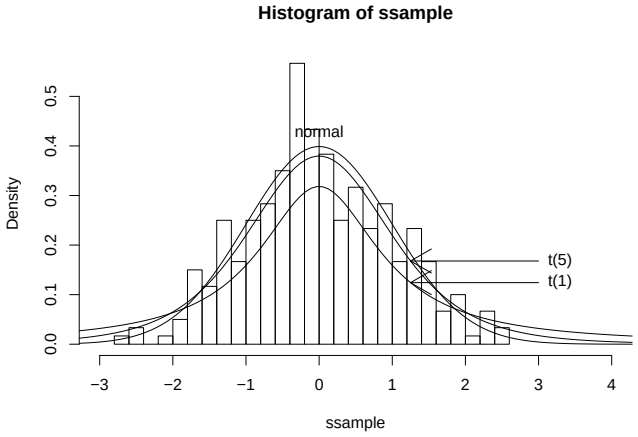
  Take matrices for x and y. Useful for plotting several columns of y-values against the same vector of x-values.
- `curve`

  Plots a function at 101 equally spaced values.

```
> curve(sin(x),from=0,to=2*pi)
> curve(cos(x),from=0,to=2*pi,add=T)
```


Example: Annotated Histogram

```
> ssample <- (sample-mean(sample))/sd(sample)
> hist(ssample, probability=T, nclass=25, xlim=c(-3,4))
> curve(dnorm(x), add=T)
> for (df in c(1,5))
    curve(dt(x, df), add=T)
> text(0, dnorm(0),
      "normal", pos=3)
> y <- dt(1.25, df=c(5,1))
> text(3, y, c("t(5)", "t(1)"), pos=4)
> arrows(3, y, 1.25, y)
```

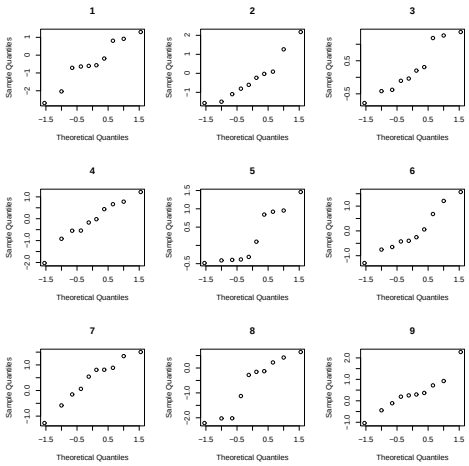


Multiple Graphs on a Page

- `par(mfrow=c(`$nrow$`,`$ncol$`))`
  `par(mfcol=c(`$nrow$`,`$ncol$`))`

  Lay out next $nrow \times ncol$ plots in an array. With `mfrow`, fill in row-by-row; with `mfcol`, fill in col-by-col.

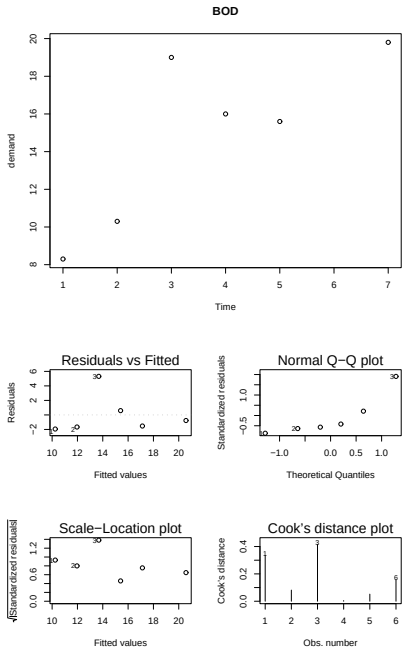
```
> par(mfrow=c(3,3))
> for (i in 1:9) qqnorm(rnorm(10), main=i)
> par(mfrow=c(1,1))  # go back to normal
```



Multiple Graphs on a Page

- `layout(...)`

```
> layout(rbind(c(1,1),c(2,3),c(4,5)),heights=c(2,1,1))
> with(BOD, plot(Time, demand, main="BOD"))
> plot(lm(demand ~ Time, data=BOD))
```



Graphical Parameters: `par(...)`

See `help(par)` for detailed explanations of graphical parameters including `col`, `lty`, etc. Also, use `par(...)` to get and globally set graphical parameters:

```
> par()
$adj
[1] 0.5
[ . .  . and 72 more . . .]
> par("pch","lwd")
$pch
[1] 1

$lwd
[1] 1
> par(pch=0, lwd=3)
>
```

From now on (until the graphics device is reset), default plot character is a square and all lines are triple-thick. To make a semi-permanent change:

```
> opar <- par(mfrow=c(2,2), cex=2, lty="dashed")
> opar
$mfrow
[1] 1 1

$cex
[1] 0.83

$lty
[1] "solid"
> [ . . . do some plots . . . ]
> par(opar)  # back to normal
>
```

Note: an important one is `par(ask=T)`.

Graphics Devices

R has a list of graphics devices:

```
> dev.list()       # just after starting R
NULL
> plot(rnorm(100)) # will auto-open a new window
> dev.list()
X11
  2
> X11()            # create another X11 window
> dev.list()
X11 X11
  2   3
> hist(rgamma(100,1,2))
> dev.off()        # close hist window
> dev.off()        # close original window
```

or

```
> graphics.off()   # close all graphics devices
```

Devices besides `X11()` are available:

```
> qqnorm(rnorm(10), main="Appears on screen")
> postscript("myplots.ps")
> qqnorm(rnorm(10), main="First page of PostScript file")
> qqnorm(rt(10,5), main="Second page of PostScript file")
> dev.off()
X11
  2
> qqnorm(rnorm(100), main="Back to the screen")
>
```

See `help(postscript)` and `help(pdf)` for parameters.

Functions

- Displaying functions:

```
> qqplot
function (x, y, plot.it = TRUE, xlab = deparse(substitute(x)),
    ylab = deparse(substitute(y)), ...)
{
    sx <- sort(x)
    sy <- sort(y)
    lenx <- length(sx)
    leny <- length(sy)
    if (leny < lenx)
        sx <- approx(1:lenx, sx, n = leny)$y
    if (leny > lenx)
        sy <- approx(1:leny, sy, n = lenx)$y
    if (plot.it)
        plot(sx, sy, xlab = xlab, ylab = ylab, ...)
    invisible(list(x = sx, y = sy))
}
<environment: namespace:stats>
>
```

- Defining functions:

```
> f <- function() { print("Hi, I'm a function!") }
> class(f)
[1] "function"
> f
function() { print("Hi, I'm a function!") }
> f()
[1] "Hi, I'm a function!"
>
```

- Editting functions:

```
> options(editor="pico")
> fix(f)
```

If you get "stuck" in `vi`, the default editor, use `:q!`⟨Enter⟩ to escape.

Defining Functions

An example:

```
f <- function(x, y, diag=FALSE)
{
   l <- lm(y ~ x)
   opar <- par(ask=T)
   plot(x, y)
   abline(l)
   if(diag)
     plot(l)
   par(opar)
   l   # last expression is return value of f
}
```

Things to note:

- zero or more named arguments, optionally with default values;
- one or more expressions in the function "body" (that may reference the named arguments);
- the last function call or expression, to be used as the value of the function call (except if the function is terminated early using `return(value)`).

```
> l1 <- f(forbes$bp, forbes$pres, diag=T)
[ . . . shows plot of fit and diag plots . . . ]
> coef(l1)
(Intercept)           x
-81.0637271   0.5228924
>
```

Arguments

Given the function from the last slide:

```
f <- function(x, y, diag=FALSE)
{
  . . .
}
```

consider:

```
> f()
Error in eval(expr, envir, enclos) :
Argument "y" is missing, with no default
> f(forbes$bp, forbes$pres)              # uses default diag=FALSE
> f(forbes$bp, forbes$pres, F)           # same effect
> f(x=forbes$pres, diag=F, y=forbes$bp)  # any order okay if explicit
> f(forbes$pres, x=forbes$bp, diag=F)    # more complicated example
>
```

The actual rules for argument assignment are a little complicated. You don't have to worry about them if you call functions like this:

```
> text(c(1,2,3),rep(0,3),c("a","b","c"), pos=1, offset=0.75,cex=0.7)
```

with initial unnamed arguments assigned in order and extra, named arguments (if any) following the unnamed arguments.

Control Structures

- Blocks: Use braces {..} to group expressions together for function bodies and for...
- Conditional Execution:

```
if (n >= 25) {
  warning("n >=25, so using normal approximation")
  [ . . . code to use normal approximation . . . ]
} else {
  [ . . . special small sample code . . . ]
}
```

The logical operators `&&` and `||` are useful in conditional expressions:

```
if (iters >= max.iters || accuracy < tolerance) {
  return(answer)
}
```

Also, there's a vector version:

```
> ifelse(c(T,T,F,T),c(1,2,3,4),c(10,20,30,40))
[1]  1  2 30  4
>
```


Control Structures

- Repetition:
  - `for` loop

```
> for (i in 1:1000) { [ . . . repeated 1000 times . . . ] }
> student.names <- c("Sally", "Joe", "William")
> for (name in c("Bill","Joe","Sally")) {
    if (!name %in% student.names)
      warning("Missing student: ", name)
  }
Warning message:
Missing student: Bill
>
```

  - `while` loop

```
while (accuracy < tolerance && iters < max.iters) {
  [ . . . repeat while condition holds . . . ]
}
```

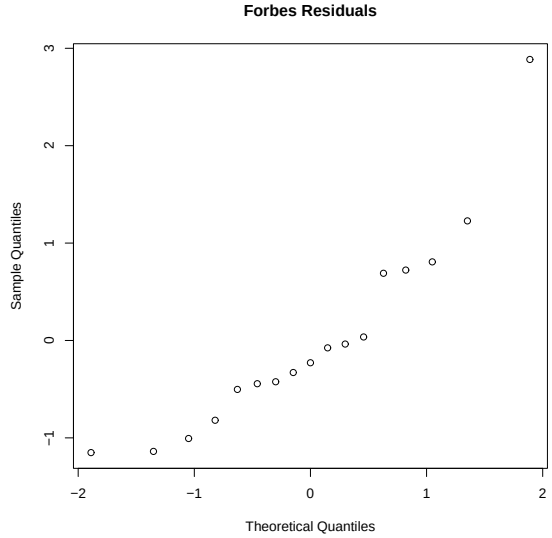
  - `repeat` loop

```
> repeat { print("I hate R!") }
```

Inside a loop, `next` moves on to the next iteration, `break` terminates the loop prematurely, and `return(...)` ends the loop and returns from the function.

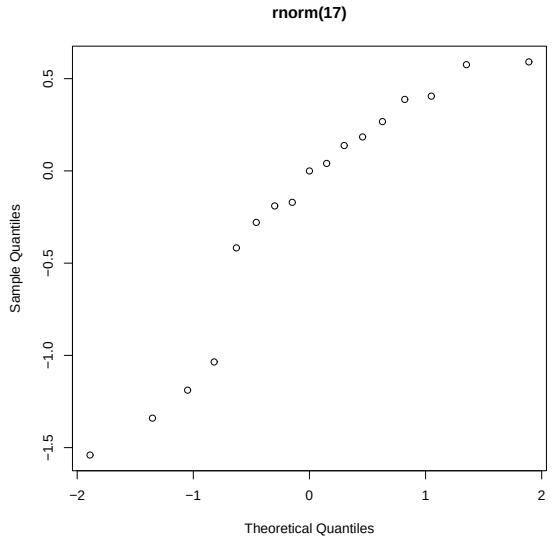
Evaluating a `qqnorm` Plot

```
> forbes.lm <- lm(pres ~ bp, data=forbes)
> r <- rstandard(forbes.lm)
> length(r)
[1] 17
> qqnorm(r, main="Forbes Residuals")
```



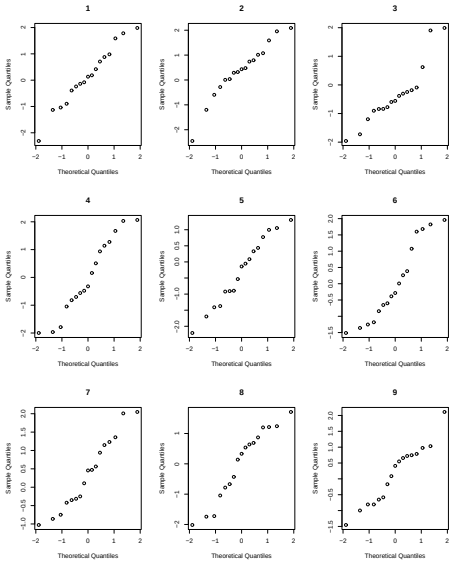
Evaluating a qqnorm Plot

```
> qqnorm(rnorm(17), main="rnorm(17)")
```



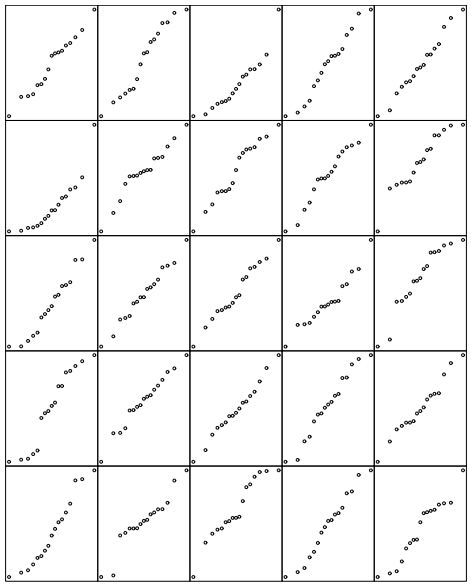
Evaluating a qqnorm Plot

```
> opar <- par(mfrow=c(3,3))
> for (i in 1:9)
    qqnorm(rnorm(17), main=i)
> par(opar)
```



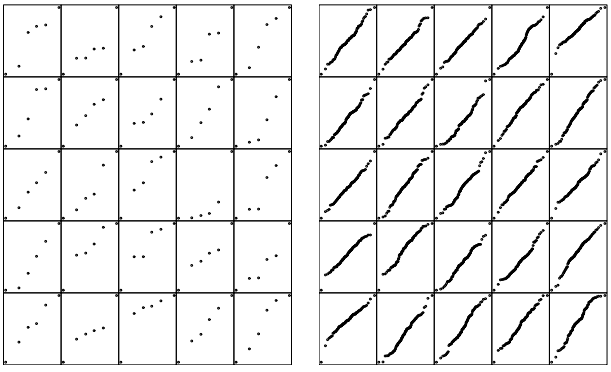
Evaluating a qqnorm Plot

```
> opar <- par(mfrow=c(5,5),mar=c(0,0,0,0),ann=F,xaxt="n",yaxt="n")
> for (i in 1:25)
    qqnorm(rnorm(17))
> par(opar)
```



Evaluating More qqnorm Plots

```
> qqnorm.sample <- function(n, nrow=5, ncol=5, annotate=F)
{
  if (annotate)
    opar <- par(mfrow=c(nrow,ncol),ask=F)
  else
    opar <- par(mfrow=c(nrow,ncol),ask=F,mar=c(0,0,0,0),ann=F,
                xaxt="n",yaxt="n")
  for (i in 1:(nrow*ncol)) {
    qqnorm(rnorm(n), main=i)
  }
  par(opar)
}
> qqnorm.sample(6)
> qqnorm.sample(100)
```



qqnorm Confidence Bounds

```
> print(qqnorm(sort(rnorm(5)),plot.it=F))
$x
[1] -1.1797611 -0.4972006  0.0000000  0.4972006  1.1797611
$y
[1] -1.7138151 -1.2574695 -0.7910166  0.3965363  0.8228089
> print(qqnorm(sort(rnorm(5)),plot.it=F))
$x
[1] -1.1797611 -0.4972006  0.0000000  0.4972006  1.1797611
$y
[1] -0.83008885 -0.04778618  0.14088821  0.20620674  0.41256715
> qqnorm.conf <- function(sample, simulations=100, ...)
{
  n <- length(sample)
  q <- qqnorm(sort(rnorm(n)), plot.it=F)
  x <- q$x
  y <- q$y
  for (i in 1:(simulations-1)) {
    q <-  qqnorm(sort(rnorm(n)),plot.it=F)
    y <- rbind(y, q$y)
  }
  bands <- apply(y, 2, quantile, probs=c(.025,.975))
  qqnorm(sample, ...)
  lines(c(x,NA,x), c(bands[1,],NA,bands[2,]), lty="dashed")
}
>
```


qqnorm Confidence Bounds

```
> qqnorm.conf(r, main="Forbes Standard Residuals")
> qqnorm.conf(r, main="Forbes Standard Residuals")
>
```