MAT 5030
Chapter 1:
Basics

Kazuhiko Shinki

Wayne State University

Introduction: How to Use R?

Download **R** language from CRAN (Comprehensive R Archive Network) and install it to your computer.

- CRAN Website:

  `http://cran.r-project.org/`

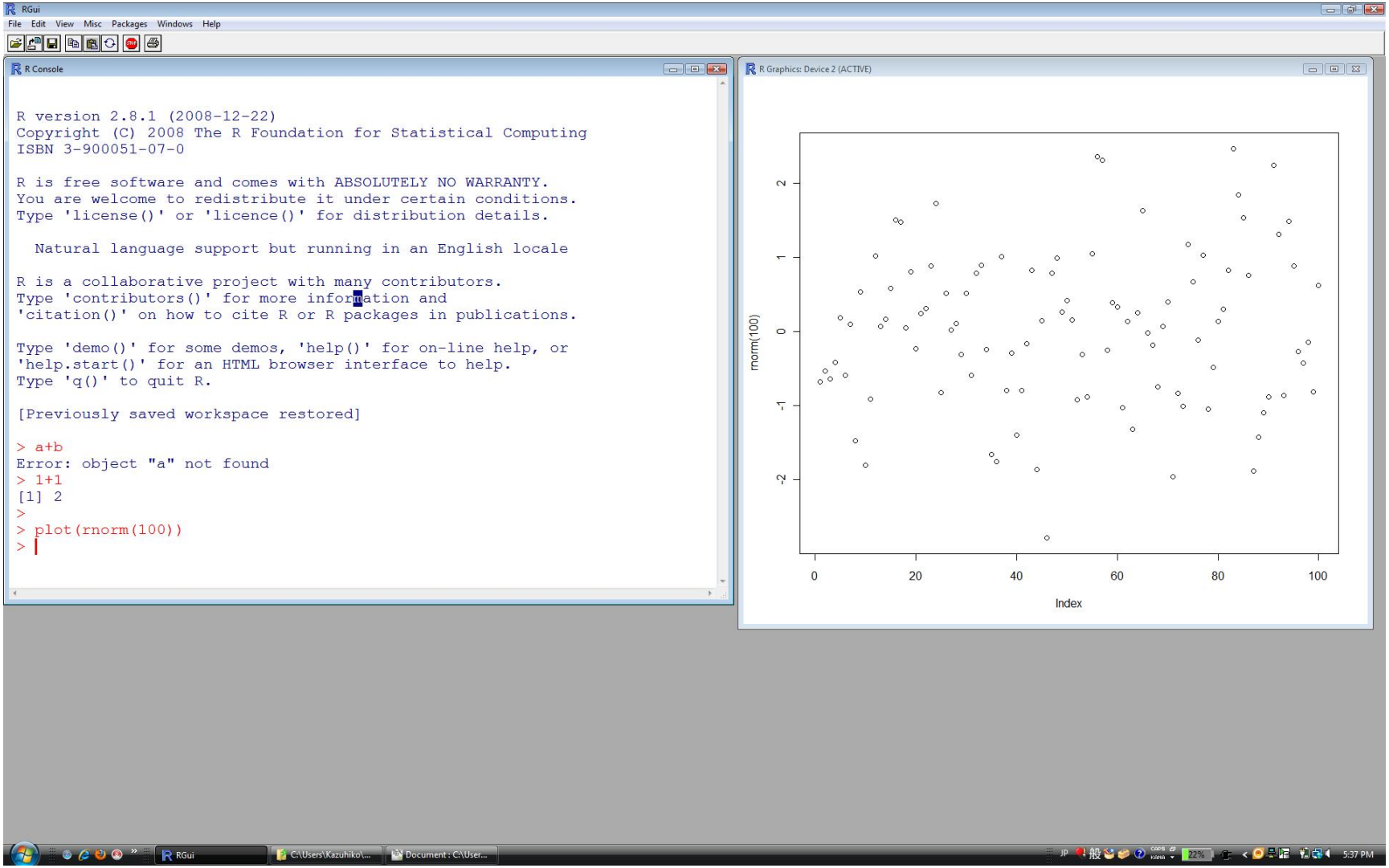
R is available for Windows, Mac OS X and Linux.
Some references:

- An Introduction to R:

  `http://cran.r-project.org/doc/manuals/R-intro.pdf`
- ”Introductory Statistics with R” (P. Dalgaard)
  - Available online at the WSU library website.
- ”The R book” (J.J. Faraway)
  - Chapter 1 (and a part of 2) are free with Amazon Kindle.

Introduction: How to Use R?

A screen shot of R:



Introduction: How to Use R?

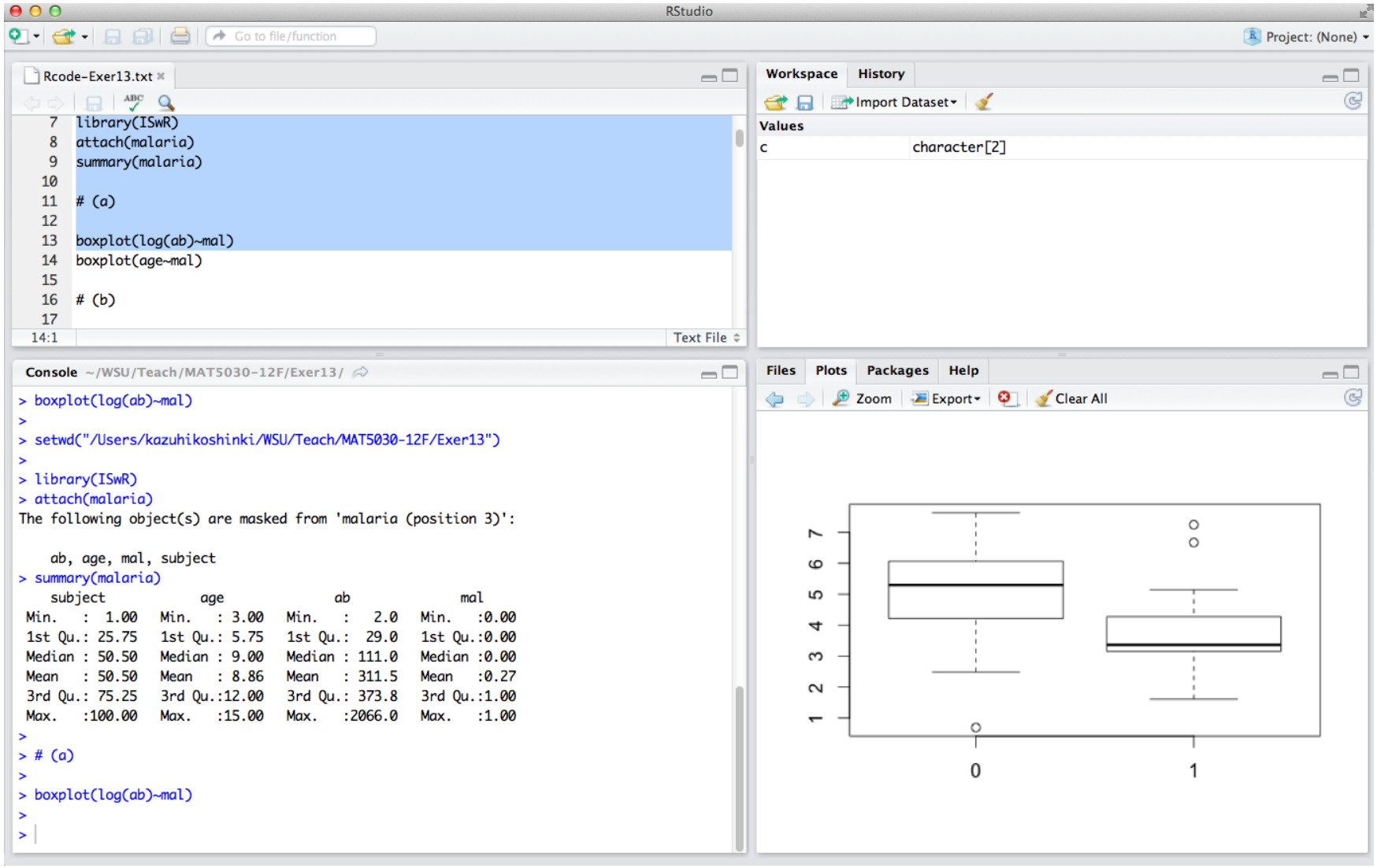
R Studio:

The R Studio is a free integrated development environment for R. It does not change the essentials of R, but it makes easier to edit codes, saves images and manage objects.

Download and install R Studio to your computer from `http://www.rstudio.com/` if you want.

Introduction: How to Use R?

A screen shot of R Studio:



Scalar

Basics

```
> 1+1 # addition
[1] 2

> 4/3-1/3 # subtraction
[1] 1

> 8*31 # multiplication
[1] 248

> 20/11 # division
[1] 1.818182
```

Scalar

Basics (cont'd)

```
> 2^3 # power
[1] 8

> sqrt(2) # square root
[1] 1.414214

> 2^(1/2) # another way for square root
[1] 1.414214

> abs(-5.5) # absolute value of -5.5
[1] 5.5
```

Calculations: Scalar – Rounding

```
> round(2.98765) # nearest integer
[1] 3
> round(2.12345)
[1] 2

> round(2.12345, 2) # rounds to x.xx
[1] 2.12

> floor(8.81) # round-down
[1] 8

> ceiling(8.31) # round-up
[1] 9
```

Scalar

Log & exponential

```
> exp(4) # e^4 (exponential)
[1] 54.59815

> log(10) # logarithm of 10 (base = e)
[1] 2.302585

> log10(100) # logarithm of 100 (base = 10)
[1] 2

> log(exp(3)) # log and exp are inverse each other.
[1] 3

> exp(log(3))
[1] 3

> (exp(0.01) - 1)/0.01 # an example of using parentheses
[1] 1.005017
```

Scalar

Triangle functions

```
> pi
[1] 3.141593

> sin(pi/2)
[1] 1

> cos(pi/4)
[1] 0.7071068

> tan(pi/4)
[1] 1
```

Scalar

Triangle functions (cont'd)

```
> asin(1/2) # arcsine
[1] 0.5235988

> acos(1/2) # arccos
[1] 1.047198

> atan(1) # arctan
[1] 0.7853982

> cosh(3) # hyperbolic cosine (= (exp(3)+exp(-3))/2)
[1] 10.06766
```

Scalar

Gamma function

Since R does not have a function for factorial (e.g., 3!), the Gamma function is useful for combinatorics.

Recall that the Gamma function is defined by

$$\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt \tag{1}$$

and satisfies

$$\Gamma(n) = (n-1)! \tag{2}$$

Scalar

Gamma function (cont'd)

```
> gamma(4) # 'gamma function of 4'  = (4-1)! = 3!
[1] 6

> factorial(3) # gamma(4) = factorial(3)
[1] 6

> 3! # "!" is not defined in R
Error: unexpected '!' in "3!"

> gamma(7) /(gamma(5) * gamma(3)) # = 6!/(4!2!) = 6 Combination 4
[1] 15
```

Scalar

Assignments

```
> x <- 2 # we define x := 2

> x # show the value of x
[1] 2

> x = 2 # another way to define x := 2

> y <- 3 # we define y := 3

> x + y
[1] 5

> 2 * x
[1] 4
```

One Vector

Calculations: Vector – Basics

```
> Weight <- c(3.1, 3.9, 4.3, 2.5, 3.6) # 5-dimensional vector

> Weight # show "Weight"
[1] 3.1 3.9 4.3 2.5 3.6

> Height <- c(0.50, 0.54, 0.53, 0.47, 0.51) # 5-dim

> BMI <- Weight/Height^2 # calculation by element (5-dim)
```

One Vector

Calculations: Vector – Elements

```
> BMI
[1] 12.40000 13.37449 15.30794 11.31734 13.84083

> BMI[2] # 2nd element (scalar)
[1] 13.37449

> BMI[2:4] # 2nd-4th elements (3-dim)
[1] 13.37449 15.30794 11.31734

> BMI[c(1,5)] # 1st and 5th elements (2-dim)
[1] 12.40000 13.84083
```

One Vector

Calculations: Vector – Sum & Mean

```
> Weight
[1] 3.1 3.9 4.3 2.5 3.6

> sum(Weight) # summation
[1] 17.4

> length(Weight) # the length of vector
[1] 5

> sum(Weight)/length(Weight) # the mean of Weight
[1] 3.48

> mean(Weight) # mean
[1] 3.48
```

One Vector

Calculations: Vector – Variance & Standard deviation

Recall that the sample variance s_x^2 of the values: $x_1, x_2, \cdots, x_n$ is defined by

$$s_x^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2 \tag{3}$$

where $\bar{x}$ is the sample mean (average).

The sample standard deviation s_x is the square root of the sample variance. Note that s_x has the same unit as $x_1, \cdots, x_n$, while s_x^2 does not. s_x is a measure of dispersion.

One Vector

Variance & Standard deviation (Cont'd)

```
> var(Weight) # sample variance of weight
[1] 0.492

> 1/4 * (sum((Weight -mean(Weight))^2)) # another way for variance
[1] 0.492

> sd(Weight) # sample standard deviation
[1] 0.7014271
```

One Vector

Sequences

```
> 5:10 # integers from 5 to 10
[1]  5  6  7  8  9 10
> 4:-3
[1]  4  3  2  1  0 -1 -2 -3

> seq(0,10, length =6) # arithmetic sequence
[1]  0  2  4  6  8 10

> seq(0,12, by = 3) # arithmetic sequence
[1]  0  3  6  9 12

> rep(1:3, length = 10) # repeat c(1,2,3) to make sequence with length = 10
 [1] 1 2 3 1 2 3 1 2 3 1

> rep(1:3, times = 3) # repeat c(1,2,3) 3 times
[1] 1 2 3 1 2 3 1 2 3
```

One Vector

Max, min, quantiles

```
> A <- 1:100 # integers from 1 to 100 (100-dim vector)
> max(A) # maximum (scalar)
[1] 100
> min(A) # minimum
[1] 1

> median(A) # median
[1] 50.5

> quantile(A, 0.25) # 25 percentile
  25%
25.75
> quantile(A, 0.75) # 75 percentile
  75%
75.25

> fivenum(A) # (min, 25%, median, 75%, max), 5-dim vector
[1]   1.0  25.5  50.5  75.5 100.0
```

One Vector

Note:
Some discrepancies exist above for quantiles, since there are numerous definitions for quantiles. The 25 percentile is at least between 25 and 26. Type ”help(quantile)” in R to see 9 definitions.

How to find commands?

- Google “R + key words”.
- See documents on the Internet and books.
- Use search in R:

```
help.search("correlation test")
```

How to see the details of commands?

Use R-manual:

```
help(read.table)
  # help for the command: "read.table"
?read.table
  # same as above
```

One Vector

Data manipulation 1

```
> A <- c(20, 11, 9, 2, 12)

> sort(A) # sort by increasing order
[1]  2  9 11 12 20
> sort(A, decreasing = TRUE) # sort by decreasing order
[1] 20 12 11  9  2

> rev(A) # reverse the order of A
[1] 12  2  9 11 20

> rank(A) # e.g., "20" is the 5th smallest observation in A
[1] 5 3 2 1 4

> order(A) # e.g., 4-th element of A is the smallest
[1] 4 3 2 5 1

> A[order(A)[2]] # 2nd smallest number in A
[1] 9
```

One Vector

Data manipulation 2

```
> diff(A) # 1st difference of A: c(11-20, 9-11, 2-9, 12-2)
[1] -9 -2 -7 10

> range(A) # same as "c(min(A), max(A))"
[1]  2 20

> prod(A) # product of all elements: 20 * 11 * 9 * 2 * 12
[1] 47520
```

Two Vectors

Some calculations

```
> Mid1 <- c(10,50,70,90,30) # midterm 1 scores for 5 students
> Mid2 <- c(20,80,100,60,40) # midterm 2 scores for the 5 students

> pmax(Mid1, Mid2) # best scores for each student
[1]  20  80 100  90  40

> pmin(Mid1, Mid2) # worst scores for each student
[1] 10 50 70 60 30

> c(Mid1, Mid2) # combine vectors
 [1]  10  50  70  90  30  20  80 100  60  40

> cor(Mid1, Mid2) # correlation between the two
[1] 0.7
```

Two Vectors

Correlation Coefficient:

Recall that the (Pearson's) sample correlation coefficient r for observations $(x_1, y_1), \cdots, (x_n, y_n)$ is defined by:

$$r = \frac{s_{xy}}{s_x s_y} \tag{4}$$

where

$$\begin{aligned} s_{xy} &= \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y}) \\ \bar{x} &= \text{mean of } x_i\text{'s} \\ \bar{y} &= \text{mean of } y_i\text{'s} \end{aligned}$$

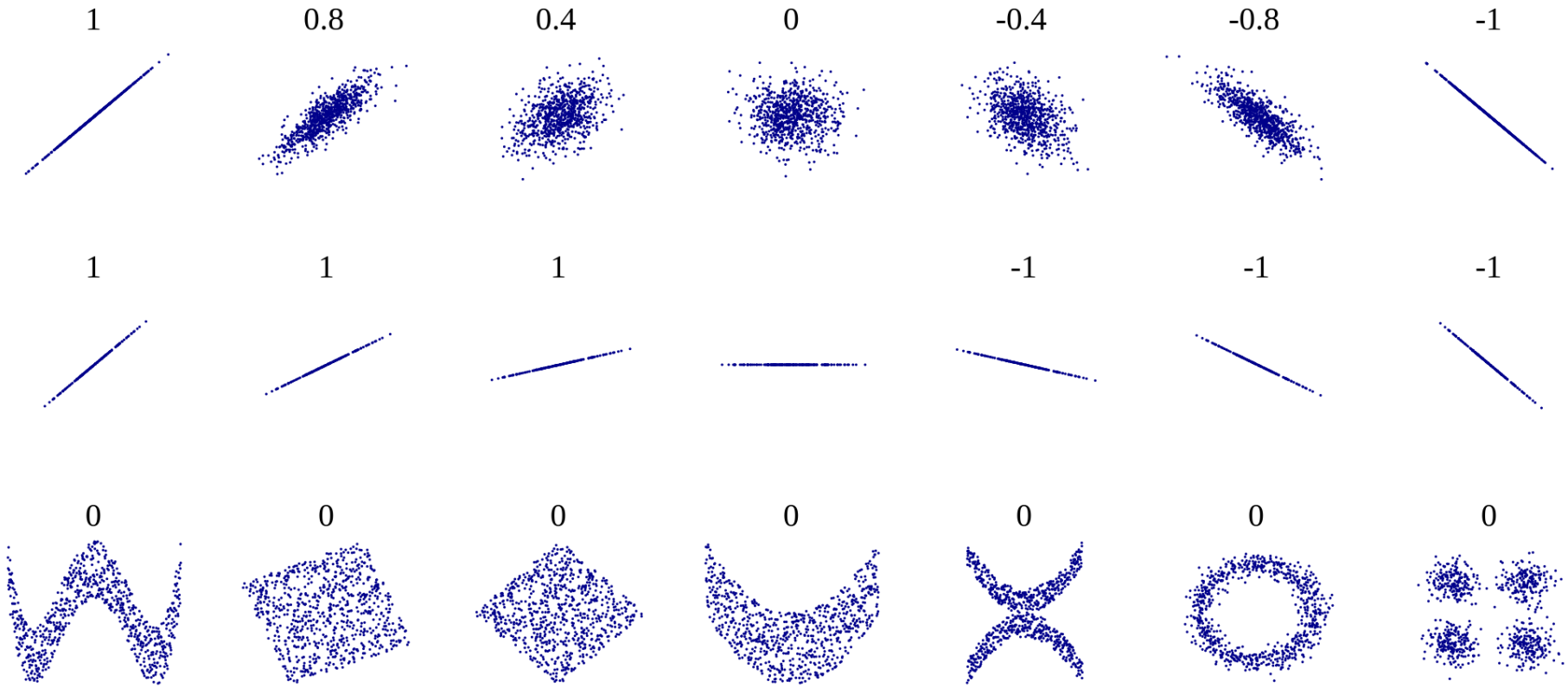
Two Vectors

Correlation coefficient

$\boldsymbol{r}$ represents the strength of linear relationship between $\boldsymbol{x}$ and $\boldsymbol{y}$.

- $-1 \leq r \leq 1$.
- $r > 0$ means $\boldsymbol{x}$ and $\boldsymbol{y}$ have positive linear relationship.
- $r < 0$ means $\boldsymbol{x}$ and $\boldsymbol{y}$ have negative linear relationship.
- $r = 0$ means $\boldsymbol{x}$ and $\boldsymbol{y}$ does not have **linear** relationship.

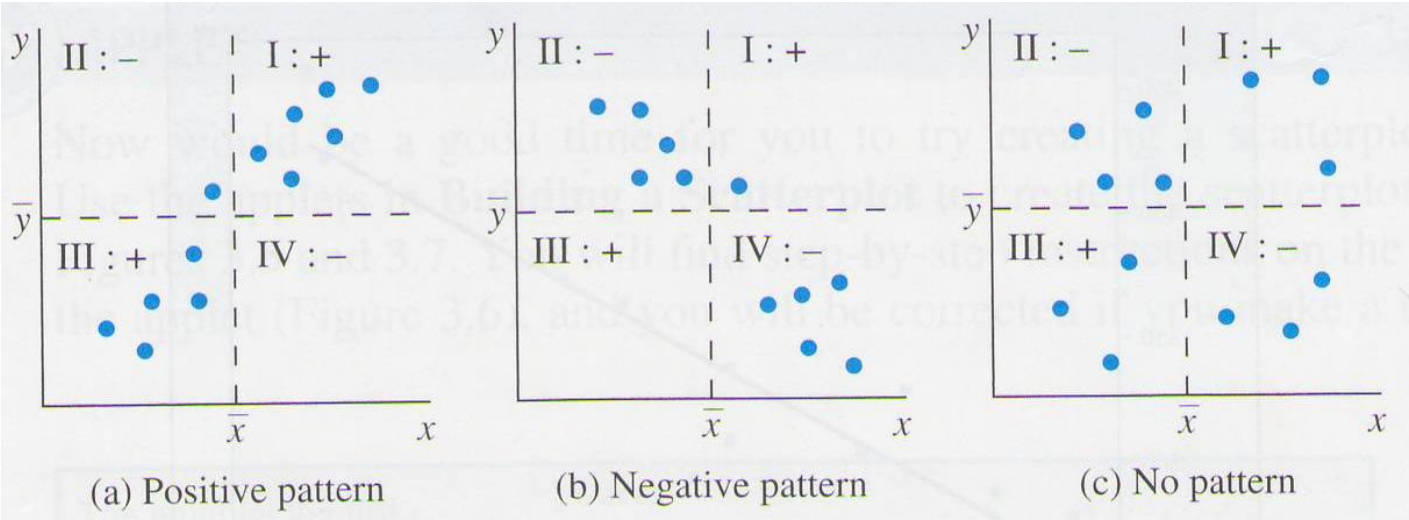
Two Vectors



Two Vectors

The signs of $(x_i - \bar{x})(y_i - \bar{y})$:

By definition, the sum of $(x_i - \bar{x})(y_i - \bar{y})$ (over all i's) determines the sign of r. One can see that positive $(x_i - \bar{x})(y_i - \bar{y})$'s correspond to a positive relationship.



Two Vectors

Recalculate ***r*** from the definition.

```
x <- Mid1; y <- Mid2; # ";" can divide the line

> Sxy <- 1/4 * sum((x - mean(x))*(y-mean(y)))

> r <- Sxy/(sd(x)*sd(y))
> r
[1] 0.7
```

We got the same answer as ”cor(Mid1,Mid2)”.

Statistical Procedures

Paired t-test

(Note: we will study t-test later in Chapter 5.) Want to test whether or not one exam has significantly higher mean than the other.

```
> t.test(Mid1-Mid2) # paired t-test

        One Sample t-test

data:  Mid1 - Mid2
t = -0.9129, df = 4, p-value = 0.4130
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 -40.41443  20.41443
sample estimates:
mean of x
      -10
```

Statistical Procedures

Our null hypothesis is that the two means are the same. The output says that:

- The t-statistic follows t-distribution with degrees of freedom 4 under the null.
- The t-statistic is -0.9129.
- The p-value is 0.4130.

Since $\boldsymbol{p - value > 0.05}$, the null hypothesis is not rejected. There are no evidence that the two means are significantly different.

Character vectors

Generating character vectors

```
> BabyName <- c("Andy", "Brian", "Cathy") # 3 by 1 vector
> BabyName
[1] "Andy"  "Brian" "Cathy"

> BabyName[2]
[1] "Brian"
```

Factor Vectors

Statistical analysis often deals with categorical variables such as gender, country of origin and ratings (e.g., ”excellent”, ”good” or ”poor”). R can recognize these as positive integers.

```
> G <- c("male","female","male","female","unknown") # character vector
> G
[1] "male"    "female"  "male"    "female"  "unknown"

> GF <- as.factor(G) # "factor" vector
> GF
[1] male    female  male    female  unknown
Levels: female male unknown

> GN <- as.numeric(GF) # as a numeric vector
> GN
[1] 2 1 2 1 3

> G2 <- as.character(GN) # as a character vector
> G2
[1] "2" "1" "2" "1" "3"
```

Matrices

Most statistical inferences (such as regression analysis and analysis of variance) are better described when we use matrix representation. Statistical languages such as R/S-Plus, Matlab, Gauss and Ox specialize in matrix calculation.

Matrices

Generating matrices

```
> A <- matrix(1:6, nrow=2, ncol=3) # generate a 2 by 3 matrix
> A
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

> B <- matrix(1:6, nrow=2, ncol=3, byrow=T) # generate a 2 by 3 matrix
> B
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

Generating matrices 2

```
> A <- matrix(1:6, nrow=2) # generate a 2 by 3 matrix
> A
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

> A <- matrix(1:6, 2, 3) # generate a 2 by 3 matrix
> A
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

Matrices

Checking matrices

```
> nrow(A) # the number of rows
[1] 2
>
> ncol(A) # the number of columns
[1] 3
>
> dim(A) # dimension
[1] 2 3
>
> row(A) # row number for each element
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    2    2    2
>
> col(A) # column number for each element
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    1    2    3
```

Matrices

Extracting Elements

```
> A[2,2] # (2,2)-th element of A
[1] 4

> A[1,] # 1st row as a 3-dimensional vector
[1] 1 3 5

> A[,2] # 2nd column as a 2-dimensional vector
[1] 3 4

> as.matrix(A[,2]) # 2nd column as a matrix
     [,1]
[1,]    3
[2,]    4

> A[1,c(1,3)] # 1st row and 1st and 3rd column
[1] 1 5
```

Matrices

```
> A[2,3] <- 100   # substitute (2,3)-th element by 100
> A
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4  100
```

Matrices

Vectors to a matrix

```
> W <- c(3.1, 2.9, 3.8)
> H <- c(50, 46, 53)

> cbind(W,H) # bind columns
       W  H
[1,] 3.1 50
[2,] 2.9 46
[3,] 3.8 53
>
> rbind(W,H) # bind rows
  [,1] [,2] [,3]
W  3.1  2.9  3.8
H 50.0 46.0 53.0
```

Matrices

Calculation

```
> A <- rbind(c(1,2), c(3,4))
> A
     [,1] [,2]
[1,]    1    2
[2,]    3    4
> B <- rbind(c(3,0), c(0,2))
> B
     [,1] [,2]
[1,]    3    0
[2,]    0    2

> A + B # addition
     [,1] [,2]
[1,]    4    2
[2,]    3    6
> A - B # subtraction
     [,1] [,2]
[1,]   -2    2
[2,]    3    2
```

Matrices

Calculation

```
> A * B  # product element by element
     [,1] [,2]
[1,]    3    0
[2,]    0    8

> A %*% B # matrix product
     [,1] [,2]
[1,]    3    4
[2,]    9    8

> solve(A) # inverse of A
     [,1] [,2]
[1,] -2.0  1.0
[2,]  1.5 -0.5

> A %*% solve(A) # A times A inverse = I (some rounding errors)
     [,1]         [,2]
[1,]    1 1.110223e-16
[2,]    0 1.000000e+00
```

Matrices

Vector and matrix

```
> A
     [,1] [,2]
[1,]    1    3
[2,]    2    4

> X <- c(1,2)

> A %*% X   # X is recognized as a column vector (2 by 1)
     [,1]
[1,]    7
[2,]   10

> X %*% A   # X is recognized as a row vector (1 by 2)
     [,1] [,2]
[1,]    5   11
```

Matrices

Diagonal and transposed matrices

```
> diag(2)   # 2 by 2 identity matrix
     [,1] [,2]
[1,]    1    0
[2,]    0    1

> diag(3:2) # 2 by 2 diagonal matrix
     [,1] [,2]
[1,]    3    0
[2,]    0    2
>
> t(A)   # transposed A
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

Determinant, eigenvectors

```
> det(A) # determinant of A
[1] -2
>
> E <- eigen(A)  # eigenvalues and eigenvectors of A
> E
$values
[1]  5.3722813 -0.3722813

$vectors
           [,1]       [,2]
[1,] -0.4159736 -0.8245648
[2,] -0.9093767  0.5657675
```

Verifying eigenvectors

```
> E$vectors[,1] # eigenvector 1
[1] -0.4159736 -0.9093767

> A %*% E$vectors[,1]
          [,1]
[1,] -2.234727
[2,] -4.885428

> -2.234727/-0.4159736
[1] 5.372281
> -4.885428/-0.9093767
[1] 5.372282
```

Matrices

Row and column names

```
> A <- matrix(0, nrow=3, ncol = 3)
> rownames(A) <- c("Alex","Bob","Chris") # row names
> colnames(A) <- c("Height", "Weight", "BMI") # column names
> A
      Height Weight BMI
Alex       0      0   0
Bob        0      0   0
Chris      0      0   0

> rownames(A) # show row names
[1] "Alex"  "Bob"   "Chris"
```

Array

Generating arrays

A matrix has two indices; row and column. R can also produce arrays with more than two indices.

```
> A <- array(1:24, dim=c(3,4,2))
> A
, , 1

     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

, , 2

     [,1] [,2] [,3] [,4]
[1,]   13   16   19   22
[2,]   14   17   20   23
[3,]   15   18   21   24
```

Array

Transposed Array

```
> aperm(A, perm = c(2,1,3)) # interchange 1st and 2nd indices
, , 1

     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12

, , 2

     [,1] [,2] [,3]
[1,]   13   14   15
[2,]   16   17   18
[3,]   19   20   21
[4,]   22   23   24
```

List

We sometimes want to combine multiple numbers, vectors and matrices into an object. It is called a ”**list**” in R.

```
> Instructor <- c("Kaz")
> Time <- rbind(c(1250,1345),c(1250,1345),c(1250,1345))
> rownames(Time) <- c("Mon","Wed","Fri")
> colnames(Time) <- c("Start","End")
> M5030 <- list(Inst=Instructor, Time=Time)
> M5030
$Inst
[1] "Kaz"

$Time
    Start  End
Mon  1250 1345
Wed  1250 1345
Fri  1250 1345
```

List

Elements of a list

```
> M5030$Inst # "Inst" of M5030
[1] "Kaz"

M5030[[1]] # 1st object of M5030
[1] "Kaz"

> M5030[[2]][3,] # 3rd row of the 2nd object
Start   End
 1250  1345

> M5030[[2]][3,] <- c(1200, 1800) # change 3rd row of 2nd object
> M5030[[2]]
    Start  End
Mon  1250 1345
Wed  1250 1345
Fri  1200 1800
```

List

Example: t-test

Statistical procedures often output a list.

```
> TT <- t.test(Mid1-Mid2) # paired t-test (see the earlier slide)
> TT[[1]]
        t
-0.912871
> TT[[2]]
df
 4
> TT[[3]]
[1] 0.4129544
> TT[[4]]
[1] -40.41443  20.41443
attr(,"conf.level")
[1] 0.95
```

If you just type ”TT”, the format of the results is a little different (see the earlier slide).

List

The elements of a list may have names. You can see more details by typing ”help(t.test)”.

```
> TT$statistic   # T statistic of the test
        t
-0.912871

> TT$p.value   # p-value of the test
[1] 0.4129544

> TT$conf.int  # Confidence interval of the test
[1] -40.41443  20.41443
attr(,"conf.level")
[1] 0.95
```

Data Frame

Data frame is a set of equal-sized vectors, for example, gender, weight and height of 4 patients.

```
> Gender <- c("male","female","male","female") # gender: factor
> Height <- c(174,160, 182, 166) # height in cm: numeric
> Weight <- c(62, 59, 95, 75)  # weight in kg: numeric
> D<- data.frame(Gender, Height, Weight)
> D
  Gender Height Weight
1   male    174     62
2 female    160     59
3   male    182     95
4 female    166     75
```

Data frame allows vectors to have different types (e.g., numeric, character, factor), while matrix requires all elements to have the same type.

Data Frame

You can use similar commands for data frames as for matrices.

```
> D$Gender
[1] male   female male   female
Levels: female male
> D[2,2]
[1] 160
```

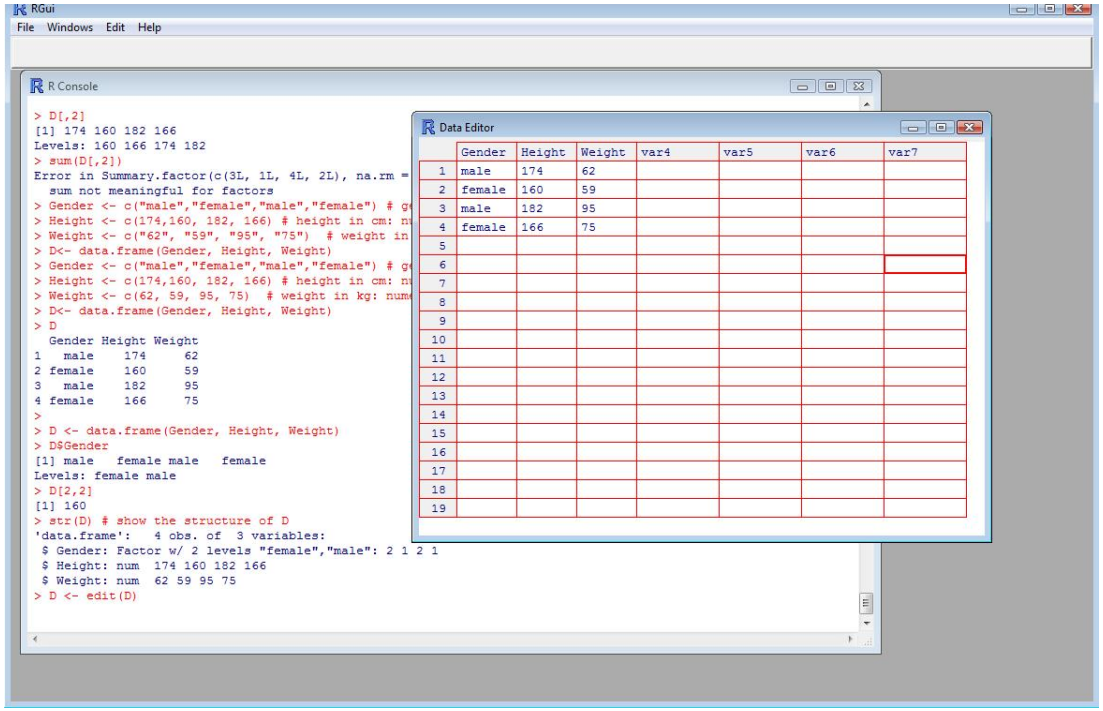
A useful command to see a data frame at a glance.

```
> str(D) # show the structure of D
'data.frame':   4 obs. of  3 variables:
 $ Gender: Factor w/ 2 levels "female","male": 2 1 2 1
 $ Height: num  174 160 182 166
 $ Weight: num  62 59 95 75
```

Data Frame

Spreadsheet

You can edit the dataframe on a spreadsheet by ”D <- edit(D)” (”D” is the name of the data frame).



”apply” family

The ”**apply**” function (and its analogues) calculate statistics such as mean and standard deviation for each row or column.

apply

```
> A <- c( 7,10, 8, 9); B <- c( 7, 8, 9,10); C <- c(12, 8, 9, 7);
> D <- data.frame(A,B,C)
> D
   A  B  C
1  7  7 12
2 10  8  8
3  8  9  9
4  9 10  7
>
> apply(D, 1, sum) # sum for each row
[1] 26 26 26 26
> apply(D, 2, mean) # mean for each column
  A   B   C
8.5 8.5 9.0
```

”apply” family

lapply, sapply

```
> E <- 2.5; F <- c(0.7,0.8,0,9); G <- c(0.1,0.2,0.3,0.4,0.5);
> L1 <- list(E=E,F=F,G=G)
> L1
$E
[1] 2.5
$F
[1] 0.7 0.8 0.0 9.0
$G
[1] 0.1 0.2 0.3 0.4 0.5

> sapply(L1, mean) # mean of each object as a vector
    E     F     G
2.500 2.625 0.300
```

”apply” family

```
> lapply(L1, mean) # mean of each object as a list
$E
[1] 2.5

$F
[1] 2.625

$G
[1] 0.3
```

”apply” family

tapply

The ”tapply” calculates group means by factor.

```
> Gender <- as.factor(c("M","F","M","F","M")) # factor vector
> Gender
[1] M F M F M
Levels: F M
> BP <- c(120, 111, 132, 94, 85)   # numeric vector
> D <- data.frame(Gender, BP)
> tapply(BP,Gender,mean) # mean by Gender
       F        M
102.5000 112.3333
> tapply(BP,Gender,sd) # standard deviation by Gender
       F        M
12.02082 24.41994
```

Logical Operations

Equality, inequality

```
> X <- 5; Y <- 6
> X == 5  # X = 5?
[1] TRUE

> X > 5  # X > 5?
[1] FALSE

> c(X,Y) > c(6, 5) # X > 6?, Y > 5?
[1] FALSE  TRUE

> X != 4 # "X is not equal to 4" is true?
[1] TRUE

> X >= 4  # X is larger than or equal to 4?
[1] TRUE
> X <= 4  # X is smaller than or equal to 4?
[1] FALSE
```

Logical Operations

AND, OR, XOR and NOT

```
> (X == 5) & (Y == 6) # Is "X = 5 and Y = 6" true?
[1] TRUE

> (X == 5) | (Y == 5) # Is "X = 5 or Y = 5" true?
[1] TRUE

> xor(X == 5, Y == 6) # Exactly one of the two conditions is true?
[1] FALSE

> !(1 == 0)  # "Not 1 = 0" is true?
[1] TRUE
```

Note: ”&&” and ”||” behave similar to ”&” and ”|” respectively, but only ”&” and ”|” can apply for vectors.

Conditioning

Recall the data frame ”D” with Gender, Height and Weight.

```
> Cond <- D$Gender == "female"  # condition: female
> Cond
[1] FALSE  TRUE FALSE  TRUE

> D[Cond,]  # extract rows for female
  Gender Height Weight
2 female    160     59
4 female    166     75
```

Missing Values

Missing values are left as NA.

```
> Weight <- c(70, 80, 55, NA, 99)
> Weight
[1] 70 80 55 NA 99

> mean(Weight) # mean of all observations
[1] NA
> mean(Weight, na.rm=TRUE) # mean, ignoring NAs
[1] 76

> is.na(Weight) # logical operation to look for NAs.
[1] FALSE FALSE FALSE  TRUE FALSE

> Weight[!is.na(Weight)] # Exclude NAs from the data
[1] 70 80 55 99

> Weight == NA  # This does not work!
[1] NA NA NA NA NA
```

Missing Values

```
> Gender <- c("M","F","M","F", NA)
> Gender
[1] "M" "F" "M" "F" NA

> Gender == "M"
[1]  TRUE FALSE  TRUE FALSE    NA
```

Plots

R generates scatter plots, bar charts, histograms etc., in a different window. You can save the figure as jpg, eps, pdf or some other formats. To save the figure, choose ”File > Save as” in the menu bar when the figure window is active.

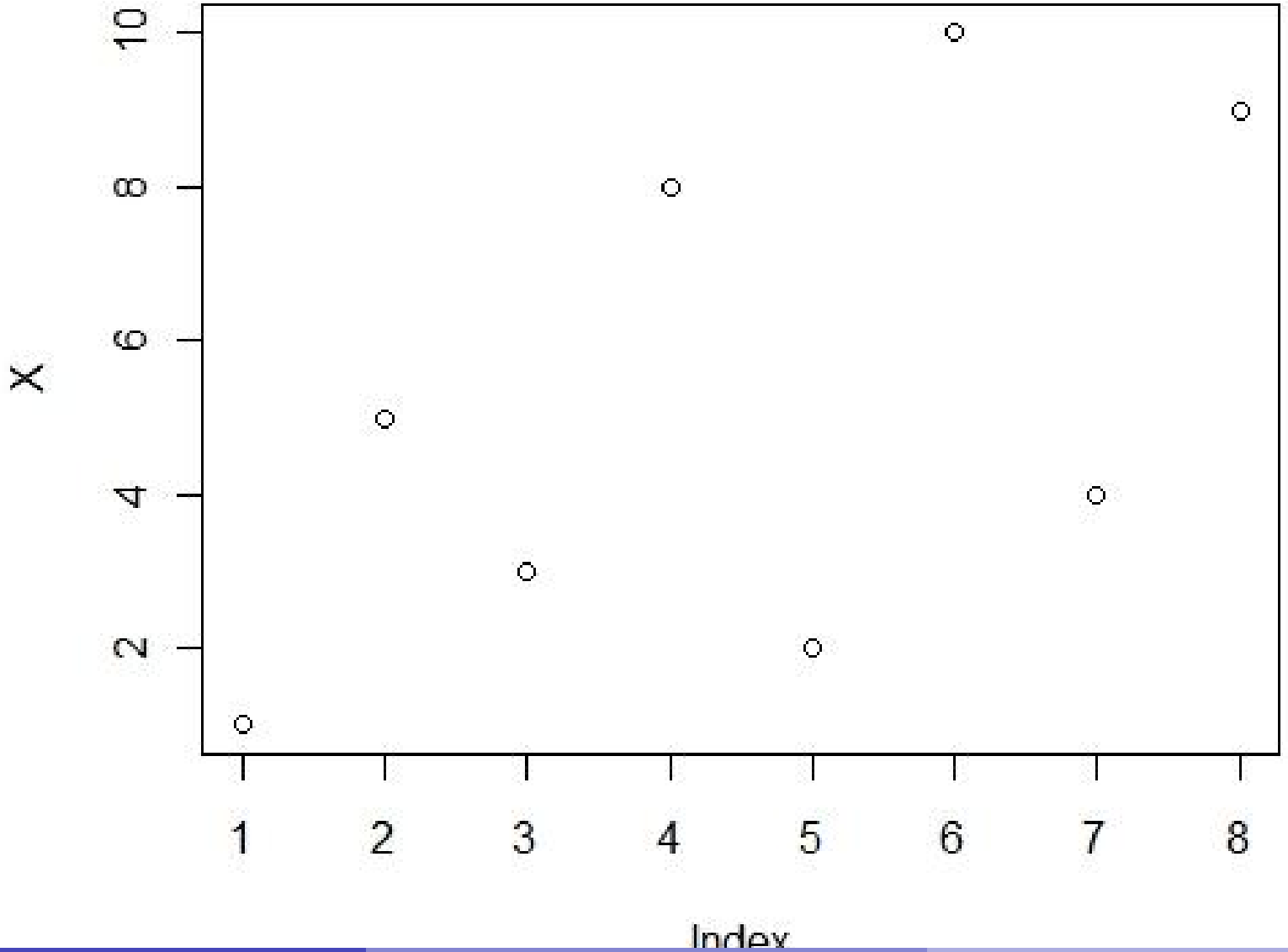
```
X <- c(1,5,3,8,2,10,4,9)
Y <- c(2,6,4,7,1,9,3,10)

plot(X) # plot (horizontal: index, vertical: X)
plot(X, Y) # scatter plot (horizontal: X, vertical:Y)

RN <- rnorm(100) # 100 standard normal random variables
hist(RN)  # histogram of RN
```

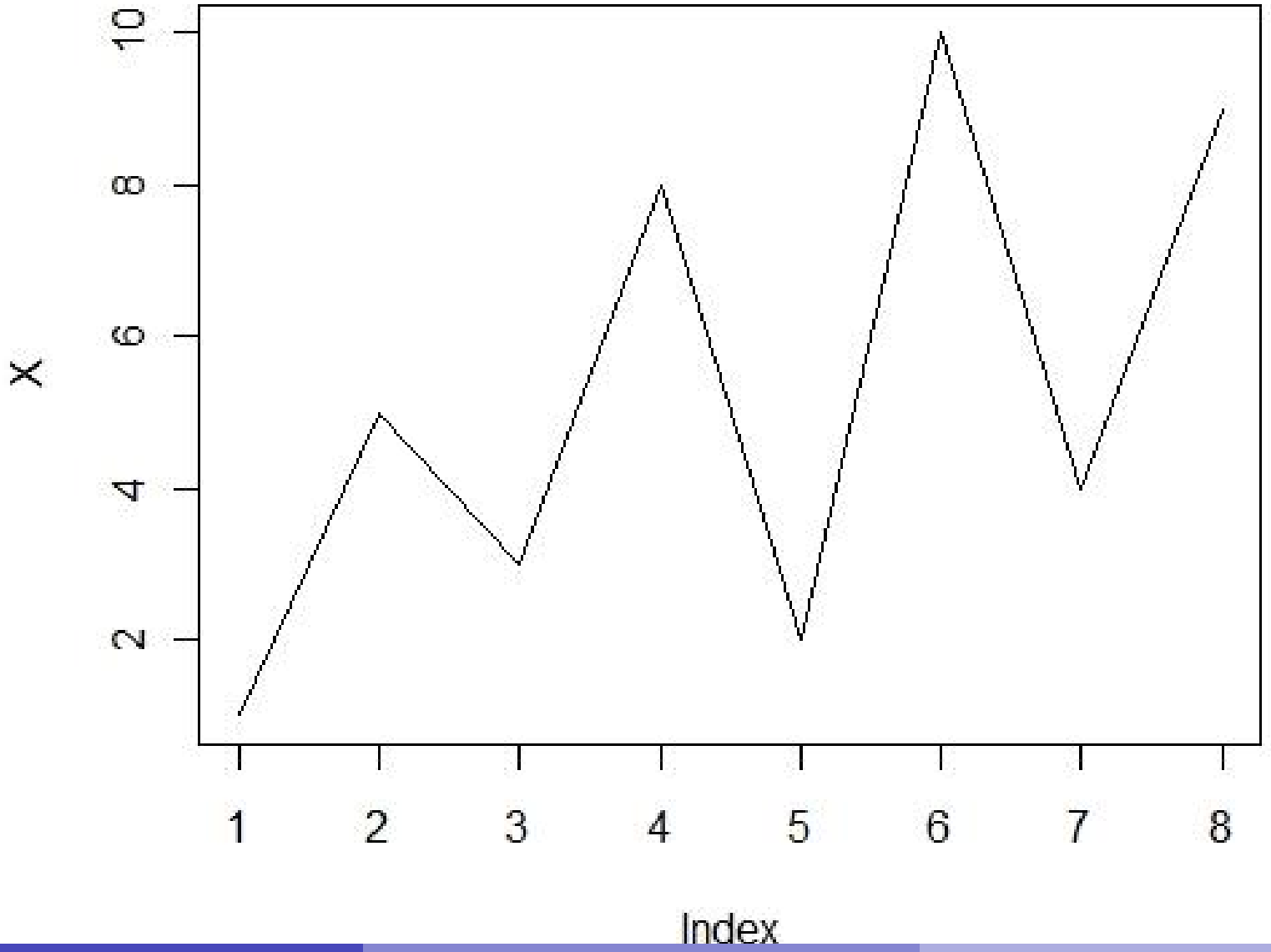
Plots

plot(X), plot against index



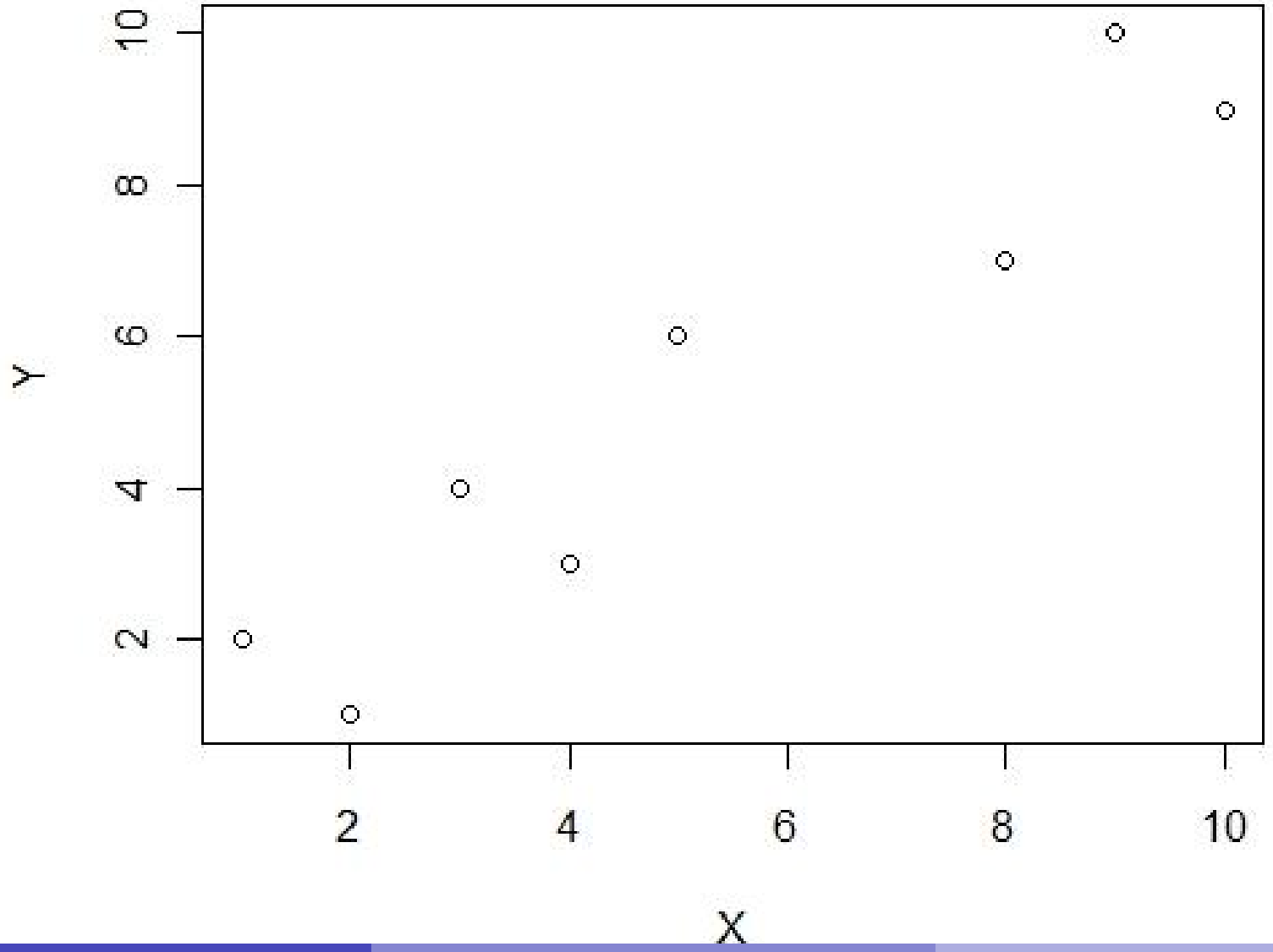
Plots

plot(X, type="l"), plot against index (line plot)



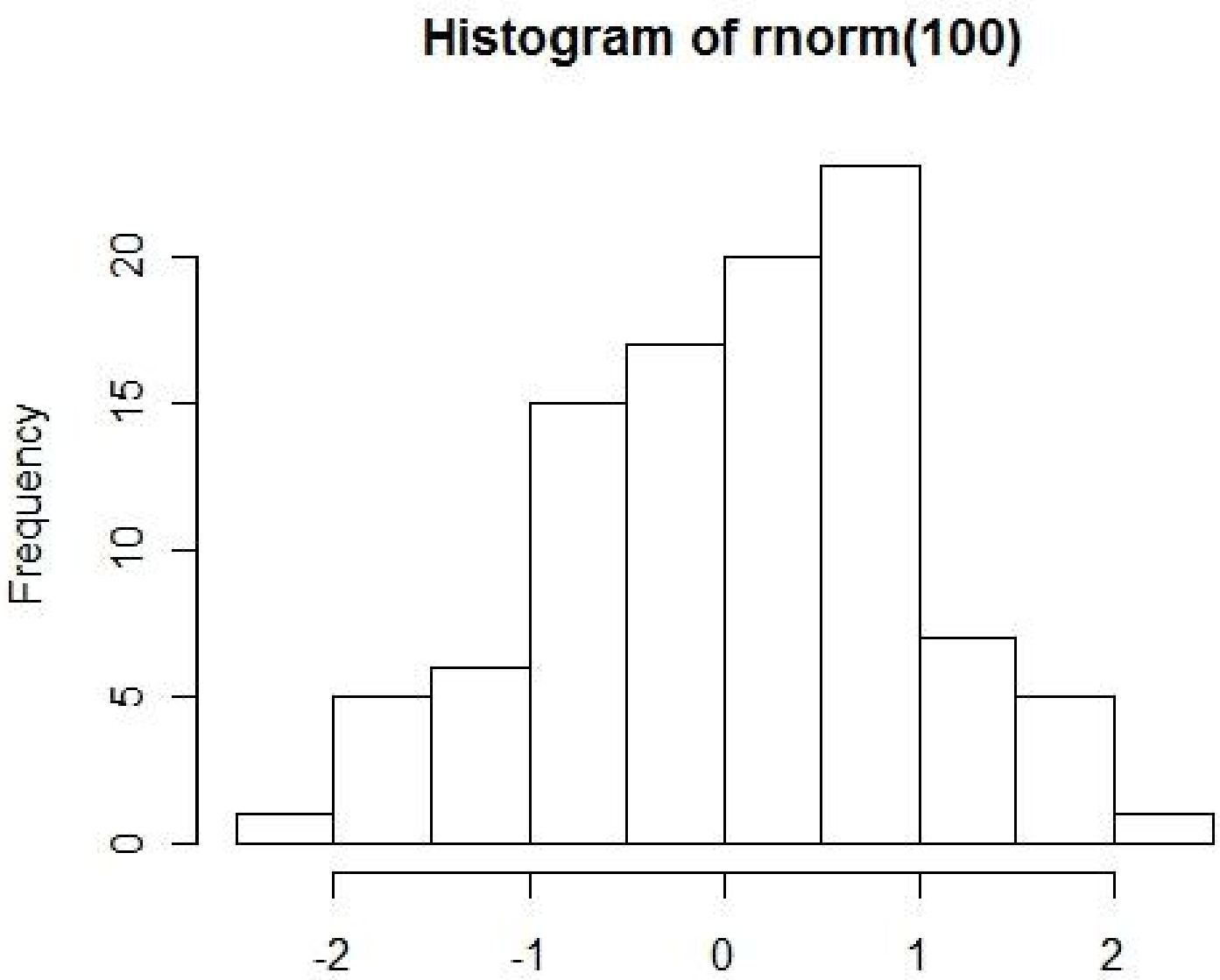
Plots

plot(X,Y), scatter plot



Plots

hist(RN), histogram



Plots

Note:
See ”help(plot)” and ”help(par)” (or search documents on the Internet) for more graphic options.

Customizing figures in R is not very convenient, while technically you can customize almost everything. You may consider to use a different software such as MS-Excel for complicated figures.