AMS 597: Statistical Computing

Pei-Fen Kuan (c)

Applied Math and Stats, Stony Brook University

Statistical Computing

- Computational statistics and statistical computing: computational, graphical, and numerical approaches to solving statistical problems.
- Monte Carlo methods refer to a diverse collection of methods in statistical inference and numerical analysis where simulation is used.

Introduction to R

- R is a system for statistical computation and graphics.
- Consists of a language plus a run-time environment with graphics, a debugger, access to certain system functions, and the ability to run programs stored in script files

Why R

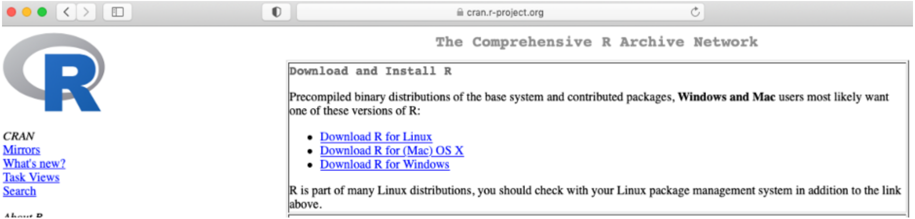
- A language and software environment for statistical computing and graphics.
- R is free!
- It is open-source and involves many developers.
- The R system is developing rapidly.
- Straightforward simple calculations and analysis.
- Allows low level control for some tasks.

Why R

- Extensive graphical abilities.
- Sometimes R is slow.

Installation of R and packages

- The way to obtain R is to download it from one of the CRAN (Comprehensive R Archive Network) sites. The main site is http://cran.r-project.org/.



- Alternatively, you can also download R studio which is an integrated development environment for R, a programming language for statistical computing and graphics https://rstudio.com/

Installation of R and packages

- R packages can be obtained from CRAN (http://cran.r-project.org/) and Bioconductor (https://www.bioconductor.org/)
- Package installation: To work through the examples and exercises in this book, you should install the ISwR package, which contains the data sets.
- If you are connected to the Internet, you can start R and from the Windows and Macintosh versions using their convenient menu interfaces.

Installation of R and packages

- On other platforms, you can type

```
install.packages("ISwR",
repos = 'http://cran.us.r-project.org')
```

- This will give off a harmless warning and install the package in the default location.
- If your R machine is not connected to the Internet, you can also download the package as a file via a different computer.

Installation of R and packages

- Starting R is straightforward, but the method will depend on your computing platform.
- Then you may need to load the package you need for your work at the command prompt, e.g.,

```
library(ISwR)
```

Installation of R and packages

- Command line interface that can be used interactively or in batch mode
- Example: evaluate the standard normal density at x=2

```
1/sqrt(2*pi)*exp(-2)
```

```
## [1] 0.05399097
```

```
dnorm(2)
```

```
## [1] 0.05399097
```

- Command prompt is >

Installation of R and packages

- A command can be continued on the next line (+)

```
plot(cars, xlab="Speed", ylab="Distance to Stop",
main="Stopping Distance for Cars in 1920")
```

Basic operations in R

- Calculating an arithmetic expression: One of the simplest possible tasks in R is to enter an arithmetic expression and receive a result.

```
exp(-2)
```

```
## [1] 0.1353353
```

- Exercise: Compute
  1. log(3.14), log10(3.14), log(3.14, 20)
  2. sin(2.1)
  3. 3^2.81
  4. sqrt(34.3)

Basic operations in R

- The [1] in front of the result is part of R's way of printing numbers and vectors. It is not useful here, but it becomes so when the result is a longer vector.
- The number in brackets is the index of the first number on that line. Consider the case of generating 20 random numbers from a normal distribution:

```r
rnorm(20)
```

```
##  [1]  1.64613825 -0.43248506 -0.07884087 -1.02264032 -2.705
##  [7]  0.04279371  0.84667458 -0.65180540 -0.82056744 -0.604
## [13]  0.93743632  1.05891377 -0.79317272 -1.32214871  1.045
## [19] -0.34380640  0.29452628
```

Basic operations in R

- Assignments:

```
x <- 2
x
```

```
## [1] 2
```

```
x+x
```

```
## [1] 4
```

```
x=2
3 -> x
x
```

```
## [1] 3
```

```
x.1 <- 2.3
x.1
```

```
## [1] 2.3
```

Basic operations in R

- Vectorized arithmetic: The construct c() is used to define vectors.

```
weight <- c(60, 72, 57, 90, 95, 72)
weight
```

```
## [1] 60 72 57 90 95 72
```

Basic operations in R

- You can do calculations with vectors just like ordinary numbers, as long as they are of the same length.

```
height <- c(1.75, 1.80, 1.65, 1.90, 1.74, 1.91)
bmi <- weight/height^2
bmi
```

```
## [1] 19.59184 22.22222 20.93664 24.93075 31.37799 19.73630
```

Basic operations in R

- It is in fact possible to perform arithmetic operations and calculate some basic statistics on vectors.

```
bmi^2
```

```
## [1] 383.8401 493.8272 438.3429 621.5422 984.5782 389.5216
```

```
length(bmi)
```

```
## [1] 6
```

```
sum(bmi)
```

```
## [1] 138.7957
```

```
mean(bmi)
```

```
## [1] 23.13262
```

Basic operations in R

```r
sd(bmi)
```

```
## [1] 4.493165
```

Basic operations in R

- Exercise:
  1. Compute the (A) median and (B) variance of 'weight'
  2. Compute the (A) covariance and (B) correlation of 'weight' and 'height'
  3. Implement the following operations:

```
xbar <- sum(weight)/length(weight)
weight-xbar
sqrt(sum((weight - xbar)^2)/(length(weight) - 1))
```

Basic operations in R

- Standard statistical procedures: You could run standard T-test to assess whether the six persons' BMI can be assumed to have mean 22.5 given that they come from a normal distribution.

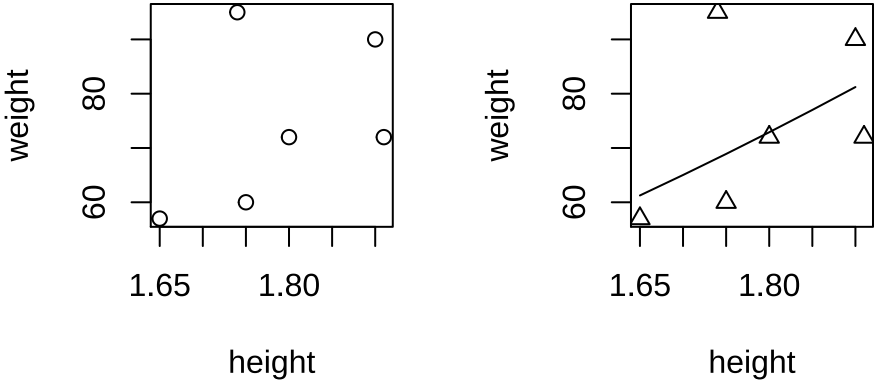
```
t.test(bmi, mu=22.5)
```

```
##
##  One Sample t-test
##
## data:  bmi
## t = 0.34488, df = 5, p-value = 0.7442
## alternative hypothesis: true mean is not equal to 22.5
## 95 percent confidence interval:
##  18.41734 27.84791
## sample estimates:
## mean of x
##  23.13262
```

Basic operations in R

- Graphics:

```
par(mfrow=c(1,2))
plot(height, weight)
plot(height, weight, pch=2)
hh <- c(1.65, 1.70, 1.75, 1.80, 1.85, 1.90)
lines(hh, 22.5*hh^2)
```



R language essentials

- Expressions and objects: The basic interaction mode in R is one of expression evaluation.
- The user enters an expression; the system evaluates it and prints the result.
- All R expressions return a value (possibly NULL), but sometimes it is "invisible" and not printed.

R language essentials

- Functions and arguments: Many things in R are done using function calls, commands that look like an application of a mathematical function of one or several variables; for example, `log(x)`.
- Vectors: We have already seen numeric vectors. There are two further types, character vectors and logical vectors. A character vector is a vector of text strings, whose elements are specified and printed in quotes.

R language essentials

- It does not matter whether you use single- or double-quote symbols, as long as the left quote is the same as the right quote.

```
c("Huey","Dewey","Louie")
```

```
## [1] "Huey"  "Dewey" "Louie"
```

```
c('Huey','Dewey','Louie')
```

```
## [1] "Huey"  "Dewey" "Louie"
```

R language essentials

- Logical vectors are constructed using the `c` function just like the other vector types

```
c(T,T,F,T)
```

```
## [1]  TRUE  TRUE FALSE  TRUE
```

R language essentials

- Exercise: Try the following R script:

```
a <- c(2, 3, exp(3.2), sin(8))
a>3
as.logical(c(1,1,0,1))
as.numeric(c(T,T,F,T))
```

R language essentials

- Quoting and escape sequences

```
cat("Huey","Dewey","Louie", "\n")
```

```
## Huey Dewey Louie
```

```
cat("What is \"R\"?\n")
```

```
## What is "R"?
```

R language essentials

- Missing values: R allows vectors to contain a special NA value as missing values.

```
a <- NA
a
```

```
## [1] NA
```

```
is.na(a)
```

```
## [1] TRUE
```

- What about `a <- "NA"`

String manipulations

- Coersion into string: `as.character()`
- String length: `nchar()`

```
a <- 12345
b <- as.character(a)
nchar(a)
```

```
## [1] 5
```

```
nchar(12)
```

```
## [1] 2
```

```
nchar(1234)
```

```
## [1] 4
```

String manipulations

```r
a = ""
nchar(a)
```

```
## [1] 0
```

```r
a <- NA
nchar(a)
```

```
## [1] NA
```

```r
a <- NULL
nchar(a)
```

```
## integer(0)
```

String manipulations

- Convert to upper/lower case

```
a <- "UPPERlower"
toupper(a)
```

```
## [1] "UPPERLOWER"
```

```
tolower(a)
```

```
## [1] "upperlower"
```

```
a <- "UPPERlower3.14159_)({:;"
toupper(a)
```

```
## [1] "UPPERLOWER3.14159_)({:;"
```

String manipulations

- Character translation

```
a <- "butter"
chartr("u","e", a)
```

```
## [1] "better"
```

```
a <- "BUTTER"
chartr("u","e", a)
```

```
## [1] "BUTTER"
```

```
old <- c("BT")
new <- c("GN")
chartr(old, new, a)
```

```
## [1] "GUNNER"
```

String manipulations

```r
chartr("T", "GN", a)
```

```
## [1] "BUGGER"
```

```r
chartr("TR", "N", a)
```

```
## Error in chartr("TR", "N", a): 'old' is longer than 'new'
```

String manipulations

- Substrings

```
a <- "12345678"
substr(a,2,2)
```

```
## [1] "2"
```

```
substr(a,2,4)
```

```
## [1] "234"
```

```
substr(a,2)
```

```
## Error in substr(a, 2): argument "stop" is missing, with no
```

String manipulations

```r
substring(a,2)
```

```
## [1] "2345678"
```

```r
substr(a, 3, 4) <- "Toyota"
a
```

```
## [1] "12To5678"
```

```r
a <- "12345678"
substring(a, 3) = "Toyota"
a
```

```
## [1] "12Toyota"
```

String manipulations

- String Concatenation

```
a = "12345"
b = "6789"
paste(a, b)
```

```
## [1] "12345 6789"
```

```
paste(a, b, sep = "")
```

```
## [1] "123456789"
```

```
paste(a, b, sep = "honda")
```

```
## [1] "12345honda6789"
```

String manipulations

```
a = c("who", "what", "why", "where", "how")
paste(a)
```

```
## [1] "who"   "what"  "why"   "where" "how"
```

```
paste(a,"?", sep = "")
```

```
## [1] "who?"   "what?"  "why?"   "where?" "how?"
```

```
paste(a,"am i?", sep = " ")
```

```
## [1] "who am i?"   "what am i?"  "why am i?"   "where am i?'
```

```
paste(a,"am i?",sep = " ", collapse = "###")
```

```
## [1] "who am i?###what am i?###why am i?###where am i?###hov
```

String manipulations

- String Splitting

```
a <- "1,2,3,4,5,6,7,8,9"
b <- strsplit(a, split =",")
b
```

```
## [[1]]
## [1] "1" "2" "3" "4" "5" "6" "7" "8" "9"
```

```
class(b)
```

```
## [1] "list"
```

```
strsplit(a, split = "")
```

```
## [[1]]
##  [1] "1" "," "2" "," "3" "," "4" "," "5" "," "6" "," "7" ",
```

String manipulations

```r
a <- c("1,2,3", "4,5,6", "7,8,9", "a,b,c", ",,asdf,,,")
strsplit(a,",")
```

```
## [[1]]
## [1] "1" "2" "3"
##
## [[2]]
## [1] "4" "5" "6"
##
## [[3]]
## [1] "7" "8" "9"
##
## [[4]]
## [1] "a" "b" "c"
##
## [[5]]
## [1] ""     ""     "asdf" ""     ""
```

Regular expression

- String matching involves searching a string for substrings: `grep()`, `grepl()`, `regexpr()` and `gregexpr()`
- It is recommended to set `perl=TRUE` to use regular expressions with R
- By default, all matching are case sensitive. You can bypass this by setting `ignore.case=TRUE`
- For `grep()`, if `value=TRUE` then it returns the actual elements satisfying the matching criteria

Regular expression

```r
a <- c("asdf","asbf","dfdf", "12365)")
grep("d", a, perl=TRUE)
```

```
## [1] 1 3
```

```r
grep("d", a, perl=TRUE, value=TRUE)
```

```
## [1] "asdf" "dfdf"
```

```r
grep("9", a)
```

```
## integer(0)
```

```r
grep("df", a, value=TRUE)
```

```
## [1] "asdf" "dfdf"
```

Regular expression

- `grepl()` is almost similar to `grep()`, except that `value` argument is not supported
- `grepl()` returns a logical vector with the same length as the input vector.

```
grepl("df", a)
```

```
## [1]  TRUE FALSE  TRUE FALSE
```

Regular expression

- `regexpr()` returns an integer vector with the same length as the input vector
- Each element in the returned vector indicates the character position in each corresponding string element in the input vector at which the (first) regular expression match was found

```
regexpr("df", a, perl=TRUE)
```

```
## [1]  3 -1  1 -1
## attr(,"match.length")
## [1]  2 -1  2 -1
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
```

Regular expression

- `gregexpr()` returns a list with the same length as the input vector
- Each element is another vector, with one element for each match found in the string indicating the character position at which that match was found

```
gregexpr("df", a, perl=TRUE)
```

```
## [[1]]
## [1] 3
## attr(,"match.length")
## [1] 2
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
##
## [[2]]
## [1] -1
```

Regular expression

- Some commonly used matching patterns

+	match 1 or more times
*	match 0 or more times
{n}	match exactly n times
{n,}	match at least n times
{n,m}	match at least n times but not more than m times
^	match at the beginning of the line
$	match at the end of the line
.	match any character
[a-z]	match any lower case letters
[[:lower:]]	match any lower case letters
[0 – 9]	match any digits between 0 and 9
[[:digit:]]	match any digits between 0 and 9
[[:space:]]	match to a space

Regular expression

- Some commonly used matching patterns
- “\w”: match a word character (letter, digit, or underscore)
- “\W”: match a non-word character
- “\s”: match a whitespace character
- “\S”: match a non-whitespace character
- “\d”: match a digit character
- “\D”: match a non-digit character

Regular expression

```r
x <- "ab"
grep("^ac{3}b$", x, perl=TRUE, value=TRUE)
```

```
## character(0)
```

```r
x <- "acccb"
grep("^ac{3}b$", x, perl=TRUE, value=TRUE)
```

```
## [1] "acccb"
```

Regular expression

- Replacing regular expression matches in string vectors can be done using `gsub()`
- Place the entire regular expression in a capturing group `()` and then use "\1" to insert the whole regular expression match

```
gsub("(df)", "8\\18", a, perl=TRUE)
```

```
## [1] "as8df8"   "asbf"     "8df88df8" "12365)"
```

Regular expression

- Use “\U” and “\L” to change the text inserted to uppercase or lowercase

```
gsub("(df)", "8\\U\\18", a, perl=TRUE)
```

```
## [1] "as8DF8"   "asbf"     "8DF88DF8" "12365)"
```

Regular expression

- Easy to use regular expression wrapper functions are implemented in `stringr` package
- Check `str_subset()`, `str_detect()`, `str_extract()`, `str_match()`, `str_locate()`, `str_locate_all()`, `str_replace()`, `str_replace_all()`

Regular expression

Exercise: How do you check if a string is a palindrome?

R language essentials

- Functions that create vectors: We introduce three functions, `c`, `seq`, and `rep`, that are used to create vectors in various situations.

```
c(42,57,12,39,1,3,4)
```

```
## [1] 42 57 12 39  1  3  4
```

```
x <- c(1, 2, 3)
y <- c(10, 20)
c(x, y, 5)
```

```
## [1]  1  2  3 10 20  5
```

R language essentials

```r
x <- c(red="Huey", blue="Dewey", green="Louie")
x
```

```
##      red    blue   green
##   "Huey" "Dewey" "Louie"
```

```r
names(x)
```

```
## [1] "red"   "blue"  "green"
```

R language essentials

```r
c(FALSE,3)
```

```
## [1] 0 3
```

```r
c(pi,"abc")
```

```
## [1] "3.14159265358979" "abc"
```

```r
c(FALSE,"abc")
```

```
## [1] "FALSE" "abc"
```

R language essentials

```r
seq(4,9)
```

```
## [1] 4 5 6 7 8 9
```

```r
seq(4,10,2)
```

```
## [1]  4  6  8 10
```

```r
4:9
```

```
## [1] 4 5 6 7 8 9
```

R language essentials

```r
oops <- c(7,9,13)
rep(oops,3)
```

```
## [1]  7  9 13  7  9 13  7  9 13
```

```r
rep(oops,1:3)
```

```
## [1]  7  9  9 13 13 13
```

```r
rep(oops,each=3)
```

```
## [1]  7  7  7  9  9  9 13 13 13
```

R language essentials

- Matrices and arrays

```
x <- 1:12
dim(x) <- c(3,4)
x
```

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

```
matrix(1:12,nrow=3,byrow=T)
```

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

R language essentials

```r
x <- matrix(1:12,nrow=3,byrow=T)
rownames(x) <- LETTERS[1:3]
t(x)
```

```
##      A B  C
## [1,] 1 5  9
## [2,] 2 6 10
## [3,] 3 7 11
## [4,] 4 8 12
```

R language essentials

```r
cbind(A=1:4,B=5:8,C=9:12)
```

```
##      A B  C
## [1,] 1 5  9
## [2,] 2 6 10
## [3,] 3 7 11
## [4,] 4 8 12
```

```r
rbind(A=1:4,B=5:8,C=9:12)
```

```
##   [,1] [,2] [,3] [,4]
## A    1    2    3    4
## B    5    6    7    8
## C    9   10   11   12
```

- Inverse `solve()` and matrix multiplication `%*%`

R language essentials

- Factors: It is common in statistical data to have categorical variables, indicating some subdivision of data.
- Such variables should be specified as factors in R. The terminology is that a `factor` has a set of `levels`.

R language essentials

```r
pain <- c(0,3,2,2,1)
fpain <- factor(pain,levels=0:3)
levels(fpain) <- c("none","mild","medium","severe")
fpain
```

```
## [1] none   severe medium medium mild
## Levels: none mild medium severe
```

```r
as.numeric(fpain)
```

```
## [1] 1 4 3 3 2
```

```r
levels(fpain)
```

```
## [1] "none"   "mild"   "medium" "severe"
```

R language essentials

```
fpain2 <- factor(fpain, levels= c("medium","none","mild","seve
fpain2
```

```
## [1] none   severe medium medium mild
## Levels: medium none mild severe
```

```
fpain3 <- relevel(fpain, "medium")
fpain3
```

```
## [1] none   severe medium medium mild
## Levels: medium none mild severe
```

R language essentials

- Lists: It is sometimes useful to combine a collection of objects into a larger composite object. This can be done using lists.

```
intake.pre <- c(5260, 5470, 5640, 6180, 6390, 6515,
    6805, 7515, 7515, 8230, 8770)
intake.post <- c(3910, 4220, 3885, 5160, 5645, 4680,
    5265, 5975, 6790, 6900, 7335)
mylist <- list(before = intake.pre, after = intake.post)
mylist
```

```
## $before
##  [1] 5260 5470 5640 6180 6390 6515 6805 7515 7515 8230 8770
##
## $after
##  [1] 3910 4220 3885 5160 5645 4680 5265 5975 6790 6900 7335
```

R language essentials

```r
mylist$before
```

```
##  [1] 5260 5470 5640 6180 6390 6515 6805 7515 7515 8230 8770
```

```r
mylist$after
```

```
##  [1] 3910 4220 3885 5160 5645 4680 5265 5975 6790 6900 7335
```

R language essentials

- Data frame: A data frame corresponds to what other statistical packages call a "data matrix" or a "data set". You can create data frames from pre-existing variables.

```
d <- data.frame(intake.pre, intake.post)
d
```

```
##    intake.pre intake.post
## 1        5260        3910
## 2        5470        4220
## 3        5640        3885
## 4        6180        5160
## 5        6390        5645
## 6        6515        4680
## 7        6805        5265
## 8        7515        5975
## 9        7515        6790
## 10       8230        6900
```

R language essentials

```r
d$intake.pre
```

```
##  [1] 5260 5470 5640 6180 6390 6515 6805 7515 7515 8230 877(
```

- Indexing: If you need a particular element in a vector, for instance the pre-migraine supplement intake for person no. 5, you can do the following:

```r
intake.pre[5]
```

```
## [1] 6390
```

```r
intake.pre[c(3, 5, 7)]
```

```
## [1] 5640 6390 6805
```

R language essentials

```
v <- c(3, 5, 7)
intake.pre[v]
```

```
## [1] 5640 6390 6805
```

```
intake.pre[1:5]
```

```
## [1] 5260 5470 5640 6180 6390
```

```
intake.pre[-c(3, 5, 7)]
```

```
## [1] 5260 5470 6180 6515 7515 7515 8230 8770
```

R language essentials

- Conditional selection:

```
intake.post[intake.pre > 7000]
```

```
## [1] 5975 6790 6900 7335
```

```
intake.post[intake.pre > 7000 & intake.pre <= 8000]
```

```
## [1] 5975 6790
```

```
intake.pre > 7000 & intake.pre <= 8000
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE
```

R language essentials

```r
d <- data.frame(intake.pre, intake.post)
d[5, 1]
```

```
## [1] 6390
```

```r
d[5, ]
```

```
##   intake.pre intake.post
## 5       6390        5645
```

```r
d[d$intake.pre > 7000, ]
```

```
##    intake.pre intake.post
## 8        7515        5975
## 9        7515        6790
## 10       8230        6900
## 11       8770        7335
```

R language essentials

```
sel <- d$intake.pre > 7000
sel
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE
```

R language essentials

- `match()` and `%in%` operator

```
a <- c("a", "a", "c", "a", "c", "b")
match(a, c("a", "b"))
```

```
## [1]  1  1 NA  1 NA  2
```

```
a %in% c("a", "b")
```

```
## [1]  TRUE  TRUE FALSE  TRUE FALSE  TRUE
```

R language essentials

- Exercise: Subset the rows which correspond to ‘f’ in the following data.frame using `match()` and `%in%` operator

```
d1 <- data.frame(gender = c(rep(c("f", "m"), c(3, 8))),
    intake.pre, intake.post)
```

R: Commonly Used Operators

Description	R symbol	Example
Comment	`#`	`#this is a comment`
Assignment	`<-`	`x <- log2(2)`
Concatenation operator	`c`	`c(3,2,2)`
Elementwise multiplication	`*`	`a * b`
Exponentiation	`^`	`2^1.5`
x mod y	`x %% y`	`25 %% 3`
Integer division	`%/%`	`25 %/% 3`
Sequence from a to b by h	`seq`	`seq(a,b,h)`
Sequence operator	`:`	`0:20`

R: Commonly Used Operators

Description	R symbol
Square root	`sqrt`
$\lfloor x \rfloor$, $\lceil x \rceil$	`floor`, `ceiling`
Natural logarithm	`log`
Exponential function e^x	`exp`
Factorial	`factorial`
Random Uniform numbers	`runif`
Random Normal numbers	`rnorm`
Normal distribution	`pnorm`, `dnorm`, `qnorm`
Rank, sort	`rank`, `sort`
Variance, covariance	`var`, `cov`
Std. dev., correlation	`sd`, `cor`
Frequency tables	`table`
Missing values	`NA`, `is.na`

R: Commonly Used Operators

Description	R symbol	Example
Zero vector	`numeric(n)`	`x <- numeric(n)`
	`integer(n)`	`x <- integer(n)`
	`rep(0,n)`	`x <- rep(0,n)`
Zero matrix	`matrix(0,n,m)`	`x <- matrix(0,n,m)`
i^{th} element of vector a	`a[i]`	`a[i] <- 0`
j^{th} column of a matrix A	`A[,j]`	`sum(A[,j])`
ij^{th} entry of matrix A	`A[i,j]`	`x <- A[i,j]`
Matrix multiplication	`%*%`	`a %*% b`
Elementwise multiplication	`*`	`a * b`
Matrix transpose	`t`	`t(A)`
Matrix inverse	`solve`	`solve(A)`