R –Advanced Data Management

Mathematical functions

Function	Description
abs(x)	Absolute value abs(-4) returns 4.
sqrt(x)	Square root sqrt(25) returns 5. This is the same as 25^(0.5).
ceiling(x)	Smallest integer not less than x ceiling(3.475) returns 4.
floor(x)	Largest integer not greater than x floor(3.475) returns 3.
trunc(x)	Integer formed by truncating values in x toward 0 trunc(5.99) returns 5.
round(x, digits=n)	Rounds x to the specified number of decimal places round(3.475, digits=2) returns 3.48.
signif(x, digits=n)	Rounds x to the specified number of significant digits signif(3.475, digits=2) returns 3.5.

Mathematical functions

Function	Description
cos(x), sin(x), tan(x)	Cosine, sine, and tangent cos(2) returns –0.416.
acos(x), asin(x), atan(x)	Arc-cosine, arc-sine, and arc-tangent acos(-0.416) returns 2.
cosh(x), sinh(x), tanh(x)	Hyperbolic cosine, sine, and tangent sinh(2) returns 3.627.
acosh(x),asinh(x), atanh(x)	Hyperbolic arc-cosine, arc-sine, and arc-tangent asinh(3.627) returns 2.

log(x,base=n)
log(x)
log10(x)
	Logarithm of x to the base n. For convenience:
	■ log(x) is the natural logarithm.
	■ log10(x) is the common logarithm.
	■ log(10) returns 2.3026.
	■ log10(10) returns 1.

exp(x)	Exponential function exp(2.3026) returns 10.

Statistical functions

Function	Description
mean(x)	Mean mean(c(1,2,3,4)) returns 2.5.
median(x)	Median median(c(1,2,3,4)) returns 2.5.
sd(x)	Standard deviation sd(c(1,2,3,4)) returns 1.29.
var(x)	Variance var(c(1,2,3,4)) returns 1.67.
mad(x)	Median absolute deviation mad(c(1,2,3,4)) returns 1.48.
quantile(x, probs)	Quantiles where *x* is the numeric vector, where quantiles are desired and probs is a numeric vector with probabilities in [0,1] # 30th and 84th percentiles of x y <- quantile(x, c(.3,.84))
range(x)	Range x <- c(1,2,3,4) range(x) returns c(1,4). diff(range(x)) returns 3.
sum(x)	Sum sum(c(1,2,3,4)) returns 10.

Statistical functions

Function	Description
diff(*x*, lag=*n*)	Lagged differences, with lag indicating which lag to use. The default lag is 1. x<- c(1, 5, 23, 29) diff(x) returns c(4, 18, 6).
min(*x*)	Minimum min(c(1,2,3,4)) returns 1.
max(*x*)	Maximum max(c(1,2,3,4)) returns 4.
scale(*x*, center=TRUE, scale=TRUE)	Column center (center=TRUE) or standardize (center=TRUE, scale=TRUE) data object *x*.

Statistical functions

```
> x <- c(1,2,3,4,5,6,7,8)
> n <- length(x)
> meanx <- sum(x)/n
> meanx
[1] 4.5

> css <- sum((x - meanx)^2)
> sdx <- sqrt(css / (n-1))
> sdx
[1] 2.449490
```

```
> mean(x)
[1] 4.5

> sd(x)
[1] 2.449490
```

Standardizing data-Scale() function

- The scale() function standardizes the specified columns of a matrix or data frame to a mean of 0 and a standard deviation of 1:

```
        newdata <- scale(mydata)
```

- To standardize each column to an arbitrary mean and standard deviation, you can use code similar to the following

```
        newdata <- scale(mydata)*SD + M
```

  - where *M* is the desired mean and *SD* is the desired standard deviation.
  - Using the scale() function on non-numeric columns produces an error.
- To standardize a specific column rather than an entire matrix or data frame, you can use code such as this:

```
newdata <- transform(mydata, myvar = scale(myvar) * 10+50)
```

- This code standardizes the variable myvar to a mean of 50 and standard deviation of 10.

Probability functions

- Probability functions are often used to generate simulated data with known characteristics and to calculate probability values within user-written statistical functions.

  [dpqr]distribution_abbreviation()

where the first letter refers to the aspect of the *distribution* returned:

d = density
p = distribution function
q = quantile function
r = random generation (random deviates)

Probability functions

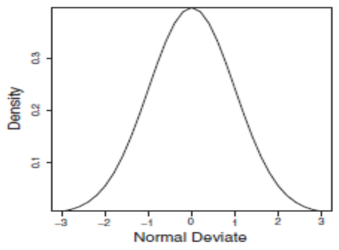
Distribution	Abbreviation	Distribution	Abbreviation
Beta	beta	Logistic	logis
Binomial	binom	Multinomial	multinom
Cauchy	cauchy	Negative binomial	nbinom
Chi-squared (noncentral)	chisq	Normal	norm
Exponential	exp	Poisson	pois
F	f	Wilcoxon signed rank	signrank
Gamma	gamma	T	t
Geometric	geom	Uniform	unif
Hypergeometric	hyper	Weibull	weibull
Lognormal	lnorm	Wilcoxon rank sum	wilcox

Normal distribution functions

- Plot the standard normal curve on the interval [–3,3].

```
x <- pretty(c(-3,3), 30)
y <- dnorm(x)
plot(x, y, type = "l", xlab = "Normal Deviate",
                       ylab = "Density",yaxs = "i")
```

Normal distribution functions



Normal distribution functions

- What is the area under the standard normal curve to the left of z=1.96?

  > pnorm(1.96)equals 0.975.
- What is the value of the 90th percentile of a normal distribution with a mean of 500 and a standard deviation of 100?

  > qnorm(.9, mean=500, sd=100) equals 628.16.
- Generate 50 random normal deviates with a mean of 50 and a standard deviation of 10.

  > rnorm(50, mean=50, sd=10)

SETTING THE SEED FOR RANDOM NUMBER GENERATION

- Each time you generate pseudo-random deviates, a different seed, and therefore different results, are produced.
- To make your results reproducible, you can specify the seed explicitly, using the set.seed() function.
- The runif() function is used to generate pseudo-random numbers from a uniform distribution on the interval 0 to 1.

```
> runif(5)
[1] 0.8725344 0.3962501 0.6826534 0.3667821 0.9255909
> runif(5)
[1] 0.4273903 0.2641101 0.3550058 0.3233044 0.6584988
> set.seed(1234)
> runif(5)
[1] 0.1137034 0.6222994 0.6092747 0.6233794 0.8609154
> set.seed(1234)
> runif(5)
[1] 0.1137034 0.6222994 0.6092747 0.6233794 0.8609154
```

GENERATING MULTIVARIATE NORMAL DATA

- In simulation research and Monte Carlo studies, you often want to draw data from a multivariate normal distribution with a given mean vector and covariance matrix.
- The mvrnorm() function in the MASS package makes this easy. The function call is

  mvrnorm(n, mean, sigma)

  - where n is the desired sample size, mean is the vector of means, and sigma is the variance-covariance (or correlation) matrix.

GENERATING MULTIVARIATE NORMAL DATA

```
> library(MASS)
> options(digits=3)
> set.seed(1234)
> mean <- c(230.7, 146.7, 3.6)
> sigma <- matrix(c(15360.8, 6721.2, -47.1, 6721.2,
                    4700.9, -16.5, - 47.1,-16.5, 0.3),
                         nrow=3, ncol=3)
> mydata <- mvrnorm(500, mean, sigma)
> mydata <- as.data.frame(mydata)
> names(mydata) <- c("y","x1","x2")
> dim(mydata)
> head(mydata, n=10)
```

Character functions

Function	Description
nchar(x)	nchar(x) Counts the number of characters of x. x <- c("ab", "cde", "fghij") length(x) returns 3. nchar(x[3]) returns 5.
substr(x, start, stop)	Extracts or replaces substrings in a character vector. x <- "abcdef" substr(x, 2, 4) returns bcd. substr(x, 2, 4) <- "22222" (x is now "a222ef").
grep(pattern, x, ignore.case=FALSE, fixed=FALSE)	Searches for pattern in x. If fixed=FALSE, then pattern is a regular expression. If fixed=TRUE, then pattern is a text string. Returns the matching indices. grep("A", c("b","A","c"), fixed=TRUE) returns 2.

Character functions

Function	Description
sub(pattern, replacement, x, ignore.case=FALSE, fixed=FALSE)	Finds pattern in x and substitutes the replacement text. If fixed=FALSE, then pattern is a regular expression. If fixed=TRUE, then pattern is a text string. sub("\\s",".","Hello There") returns Hello.There. Note that "\s" is a regular expression for finding whitespace; use "\\s" instead, because "\" is R's escape character.
strsplit(x, split, fixed=FALSE)	Splits the elements of character vector x at split. If fixed=FALSE, then pattern is a regular expression. If fixed=TRUE, then pattern is a text string. y <- strsplit("abc", "") returns a one-component, three-element list containing "a" "b" "c" unlist(y)[2] and sapply(y, "[", 2) both return "b".

Character functions

Function	Description
paste(..., sep="")	Concatenates strings after using the sep string to separate them. paste("x", 1:3, sep="") returns c("x1", "x2", "x3"). paste("x",1:3,sep="M") returns c("xM1","xM2" "xM3"). paste("Today is", date()) returns Today is Mon Dec 28 14:17:32 2015
toupper(x)	Uppercase. toupper("abc") returns "ABC".
tolower(x)	Lowercase. tolower("ABC") returns "abc".

Other useful functions

Function	Description
length(x)	Returns the length of object x. x <- c(2, 5, 6, 9) length(x) returns 4.
seq(from, to, by)	Generates a sequence. indices <- seq(1,10,2) indices is c(1, 3, 5, 7, 9).
rep(x, n)	Repeats x n times. y <- rep(1:3, 2) y is c(1, 2, 3, 1, 2, 3).
cut(x, n)	Divides the continuous variable x into a factor with n levels. To create an ordered factor, include the option ordered_result = TRUE.

Other useful functions

Function	Description
pretty(x, n)	Creates pretty breakpoints. Divides a continuous variable x into n intervals by selecting n + 1 equally spaced rounded values. Often used in plotting.

cat(... , file = "myfile", append = FALSE)	Concatenates the objects in … and outputs them to the screen or to a file (if one is declared).
	name <- c("Jane")
	cat("Hello" , name, "\n")

Other useful functions

- The last example in the table demonstrates the use of escape characters in printing.
- Use \n for new lines, \t for tabs, \' for a single quote, \b for backspace, and so forth (type ?Quotes for more information).
- For example, the code

```
name <- "Bob"
cat( "Hello", name, "\b.\n", "Isn\'t R", "\t", "GREAT?\n")
```

- produces

```
Hello Bob.
 Isn't R          GREAT?
```

- Note that the second line is indented one space. When cat concatenates objects for output, it separates each by a space.
- That's why you include the backspace (\b) escape character before the period. Otherwise it would produce "Hello Bob ."

Applying functions to data objects

```
> a <- 5
> sqrt(a)
[1] 2.236068
> b <- c(1.243, 5.654, 2.99)
> round(b)
[1] 1 6 3
> c <- matrix(runif(12), nrow=3)
> c
[,1] [,2] [,3] [,4]
[1,] 0.4205 0.355 0.699 0.323
[2,] 0.0270 0.601 0.181 0.926
[3,] 0.6682 0.319 0.599 0.215
```

```
> log(c)
[,1] [,2] [,3] [,4]
[1,] -0.866 -1.036 -0.358 -1.130
[2,] -3.614 -0.508 -1.711 -0.077
[3,] -0.403 -1.144 -0.513 -1.538
> mean(c)
[1] 0.444
```

Applying functions to data objects

- Notice that the mean of matrix c results in a scalar (0.444).
- The mean() function takes the average of all 12 elements in the matrix.
- But what if you want the three row means or the four column means?
- R provides a function, apply(), that allows you to apply an arbitrary function to any dimension of a matrix, array, or data frame.
- The format for the apply() function is

  apply(x, MARGIN, FUN, ...)

- where x is the data object, MARGIN is the dimension index, FUN is a function you specify, and ... are any parameters you want to pass to FUN.
- In a matrix or data frame, MARGIN=1 indicates rows and MARGIN=2 indicates columns.

Applying functions to data objects

```
> mydata <- matrix(rnorm(30), nrow=6)
> mydata
[,1] [,2] [,3] [,4] [,5]
[1,] 0.71298 1.368 -0.8320 -1.234 -0.790
[2,] -0.15096 -1.149 -1.0001 -0.725 0.506
[3,] -1.77770 0.519 -0.6675 0.721 -1.350
[4,] -0.00132 -0.308 0.9117 -1.391 1.558
[5,] -0.00543 0.378 -0.0906 -1.485 -0.350
[6,] -0.52178 -0.539 -1.7347 2.050 1.569
> apply(mydata, 1, mean)
[1] -0.155 -0.504 -0.511 0.154 -0.310 0.165
> apply(mydata, 2, mean)
[1] -0.2907 0.0449 -0.5688 -0.3442 0.1906
> apply(mydata, 2, mean, trim=0.2)
[1] -0.1699 0.0127 -0.6475 -0.6575 0.2312
```

A data-management challenge

- Let's consider a data management problem. A group of students have taken exams in math, science, and English.
- You want to combine these scores in order to determine a single performance
- indicator for each student.
- Additionally, you want to assign an A to the top 20% of students, a B to the next 20%, and so on. Finally, you want to sort the students alphabetically.
- Looking at this dataset, several obstacles are immediately evident. First, scores on the three exams aren't comparable. They have widely different means and standard deviations, so averaging them doesn't make sense. You must transform the exam scores into comparable units before combining them.
- Second, you'll need a method of determining a student's percentile rank on this score in order to assign a grade.
- Third, there's a single field for names, complicating the task of sorting students. You'll need to split their names into first name and last name in order to sort them properly.
- Each of these tasks can be accomplished through the judicious use of R's numerical and character functions.

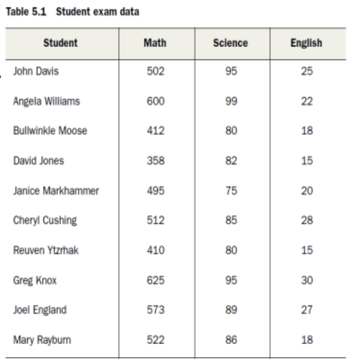


Table 5.1 Student exam data

Student	Math	Science	English
John Davis	502	95	25
Angela Williams	600	99	22
Bullwinkle Moose	412	80	18
David Jones	358	82	15
Janice Markhammer	495	75	20
Cheryl Cushing	512	85	28
Reuven Ytzrhak	410	80	15
Greg Knox	625	95	30
Joel England	573	89	27
Mary Rayburn	522	86	18

A solution for the data-management challenge

```
> options(digits=2)
> Student <- c("John Davis", "Angela Williams", "Bullwinkle Moose",
               "David Jones", "Janice Markhammer", "Cheryl Cushing",
               "Reuven Ytzrhak", "Greg Knox", "Joel England", "Mary Rayburn")

> Math <- c(502, 600, 412, 358, 495, 512, 410, 625, 573, 522)
> Science <- c(95, 99, 80, 82, 75, 85, 80, 95, 89, 86)
> English <- c(25, 22, 18, 15, 20, 28, 15, 30, 27, 18)

> roster <- data.frame(Student, Math, Science, English, stringsAsFactors=FALSE)
```

A solution for the data-management challenge

```
# Obtain the performance scores
> z <- scale(roster[,2:4])
> score <- apply(z, 1, mean)
> roster <- cbind(roster, score)

# Grade the students
> y <- quantile(score, c(.8,.6,.4,.2))
> roster$grade[score >= y[1]] <- "A"
> roster$grade[score < y[1] & score >= y[2]] <- "B"
> roster$grade[score < y[2] & score >= y[3]] <- "C"
> roster$grade[score < y[3] & score >= y[4]] <- "D"
> roster$grade[score < y[4]] <- "F"
```

A solution for the data-management challenge

```
# Extracts the last and first names
> name <- strsplit((roster$Student), " ")
> Lastname <- sapply(name, "[", 2)
> Firstname <- sapply(name, "[", 1)
> roster <- cbind(Firstname,Lastname, roster[,-1])

# Sort the students based on last and first names
> roster <- roster[order(Lastname,Firstname),]
> roster
```