Chapter 5

Advanced data management

Instructor: Li, Han

Contents

- Statistics: mean, sd, var, median,quantile,…
- Generate random variables from
  - Binomial distribution
  - Poisson distribution
  - Normal distribution
  - Uniform distribution
- Pseudo-random variable
- Inversion method

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

Statistical functions

- mean: mean(x)
- variance: var(x)
- standard deviation: sd(x)
- quantile: quantile(x, probs=c(...))
- median: median(x)
- covariance: cov(x,y)
- correlation: cor(x, y)

- min: min(x)
- max: max(x)
- range: range(x)
- sum: sum(x)
- scale: scale(x)

scale(x) means setting x*=(x-a)/b, such that mean(x*)=0, sd(x*)=1.

Example 5

```
x <- rnorm(100, 5, 1)
mean(x)
var(x)
sd(x)
quantile(x, probs=c(0.05, 0.5, 0.95))
median(x)
```

```
min(x)
max(x)
range(x)
sum(x)
x.new <- scale(x)
mean(x.new)
sd(x.new)

y <- runif(100, 0, 10)
cov(x, y)
cor(x, y)
```

Generate random variables

R could generate random variables from different distributions, for instance,

- binomial distribution
- poisson distribution
- normal distribution
- uniform distribution

[rdpq] distribution_abbreviation()

Where the first letter refers to the aspect of the distribution.

r=random generation
d=density
p=distribution function
q=quantile function

binomial distribution

$X \sim binomial(m,p)$

m independent trials, each with success probability p, and X is the total number of success.

- rbinom(n, size, prob)
- dbinom(x, size, prob, log = FALSE)
- pbinom(q, size, prob, lower.tail = TRUE, log.p = FALSE)
- qbinom(p, size, prob, lower.tail = TRUE, log.p = FALSE)

Here n=number of samples, size=m, prob=p.

lower.tail=TRUE means calculating $P(X \leq x)$.

Example 1

```
#generate n=5000 samples with m=20, p=0.5
rv <- rbinom(5000, 20, 0.5)

pbinom(c(5, 10, 15, 20), size=20, prob=0.5)
qbinom(c(0.25, 0.5, 0.75, 1), 10, 0.5)
```

Poisson distribution

$X \sim poisson(\lambda)$

- rpois(n, lambda)
- dpois(x, lambda, log = FALSE)
- ppois(q, lambda, lower.tail = TRUE, log.p = FALSE)
- qpois(p, lambda, lower.tail = TRUE, log.p = FALSE)

Example 2

```
rv <- rpois(2000, 10)
ppois(c(5,10,15,20), lambda=10)
qpois(c(0.3, 0.6, 0.8), 10)
```

Normal distribution

$$X \sim N(\mu,\sigma^2)$$

- rnorm(n, mean = 0, sd = 1)
- dnorm(x, mean = 0, sd = 1, log = FALSE)
- pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
- qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)

Example 3

```
pnorm(0)
round(qnorm(0.95),3)
rv1 <- rnorm(500, 0, 1)
rv2 <- rnorm(100000, 0, 1)
```

Uniform distributions

(1) discrete uniform distribution

$$P(X=x_1)=P(X=x_2)=...=P(X=x_k)=1/k$$

sample(x, size, replace = FALSE, prob = NULL)

- x: a vector of numbers to choose, here $x=(x_1,...,x_k)$.
- size: the number of samples
- replace= FALSE: non-replacing sampling
- prob=the corresponding probability to choose each item. By default, it is equal probability.

(2) continuous uniform distribution

runif(n, min = 0, max = 1)

By default, X ~ unif(0,1).

- n: the number of samples
- min: the minimum value of X
- max: the maximum value of X

runif(n, min=a, max=b) means X ~ unif(a, b)

Example 4

```
sample(1:10, 5)
sample(1:10, 5, replace=TRUE)
sample(1:5,10, replace=TRUE, prob=c(0.5, 0.1, 0.2, 0.1, 0.1))

runif(10)
runif(20, 1, 10)
```

A summary of distributions

Table 5.4 Probability distributions

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`

Listing 5.4 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
```

`c` is a matrix

`mean(c)` returns the average of all 12 elements of `c`

apply() function

apply() could calculate the statistics of a matrix or a data frame for all its rows or columns simultaneously.

apply(X, MARGIN, FUN, ...)

- X: a matrix or data frame
- MARGIN=1: calculation by row
- MARGIN=2: calculation by column
- FUN: the function to do the calculation

Listing 5.5 Applying a function to the rows (columns) of a matrix

```
 > 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
```

1. Generates data
2. Calculates the row means
3. Calculates the column means

Example 6

```
x <- rnorm(10)
y <- rnorm(10, 5, 3)
z <- cbind(x, y)
z
apply(z, 1, mean)
apply(z, 2, mean)
apply(z, 1, sd)
apply(z, 2, sd)
```

Example 7

```
attach(mtcars)
z <- mtcars[c("mpg", "hp", "wt")]
apply(z, 2, mean)
apply(z, 2, sd)
```

"Random" variables

Random variables are really random?

Note that machine can only execute explicit commands. Thus we need to give a rule initially how to generate the samples, and this results in a pseudo-random variable, which are not actually random.

Pseudo-random variables

A sequence of pseudo-random numbers

$$x_1,\ldots,x_n$$

Their statistical properties are very similar to those of true random variables, thus we regard them as if they were really random.

How to generate pseudo random variables?

The most widely used method of generating pseudo-random numbers are the congruential generators:

$$X_i = (aX_{i-1} + c) \bmod M$$
$$U_i = X_i / M$$

for a multiplier a, shift c, and modulus M, all integers.

Given M, the values of a and c are chosen to maximize the period of the numbers, and to ensure the generator has good statistical properties.

Some examples:

M	a	c
2^{59}	13^{13}	0
2^{32}	69069	1
2^{31}-1	630360016	0
2^{32}	2147001325	715136305

set. seed()

How to re-generate the same sequence of random variables? Use

set. seed()

Everytime you generate pseudo-random variables, the software uses a different seed, thus generate different samples. If you fix the seed, your results will be reproducible!

Example 8

```
runif(5)
runif(5)

set.seed(123)
runif(5)
set.seed(123)
runif(5)
```

Generate discrete distribution

Question:

How to generate random variables from a specified distribution that has probability p(k), k=0,1,…,N?

Inversion method

The inversion method reduces to searching for an index j that satisfies

$$p(0)+p(1)+\ldots+p(j-1) \leq U < p(0)+p(1)+\ldots+p(j-1)+p(j),$$

Where $U \sim \text{Uniform}(0,1)$.

Example 9

Assume p(0)=0.5, p(1)=0.3, p(2)=0.2, then

if $0 \le U < 0.5$, return X=0;

if $0.5 \le U < 0.8$, return X=1;

if $0.8 \le U < 1$, return X=2.

Question:

How to generate random variables from a specified continuous distribution F(.)?

Summary

- how to generate random variables from different distributions
- calculate the statistics, for instance, the mean, sd, quantile of sample variables
- pseudo random variables
- using inversion method to generate random variables from a specific discrete distribution