PR4R Factor,Matrix,List,Array

강현승

2022-09-29

1. Factor

- R에서는 범주형 자료의 데이터 형태를 요인(factor)이라고 함
- 서수형(순서있는경우:수능등급), 명목형(순서없는경우:성별) 등

1.1. 명목형 자료 만들기

- factor() 함수를 이용해 생성함
- factor(vector,levels=수준,labels=이름,ordered=T)
- levels안에속하지않는변수를집어넣을경우에는로 출력
- ordered는서수형인지명목형인지를구분할때사용

```
score = factor(
  c(3, 2, 3, 4, 3, 1, 1, 3, 2, 2, 2, 2, 1, 1, 5),
  levels = c(1, 2, 3, 4),
  labels = c("A" , "B", "C", "D"),
  ordered = T
)
score
```

```
##  [1] C    B    C    D    C    A    A    C    B    B    B    B    A    A    <NA>
## Levels: A < B < C < D
```

1.2. 명목형 자료로 변환하기

- as.factor함수로변환한다.
- attributes함수로자료의속성을확인할수있다.

```
# 문자를 벡터에 입력하였을 때
fac_char = c("포도", "키위", "메론", "바나나", "딸기")
attributes(fac_char)
```

```
## NULL
```

```
# 문자벡터를 명목형 자료로 변경하였을 때
fac_char = as.factor(fac_char)
attributes(fac_char)
```

```
## $levels
## [1] "딸기"   "메론"   "바나나" "키위"   "포도"
##
## $class
## [1] "factor"
```

```
# 숫자를 벡터에 입력하여 명목형으로 변경도 가능
fac_num = 1:4
attributes(fac_num)
```

```
## NULL
```

```
fac_num = as.factor(fac_num)
attributes(fac_num)
```

```
## $levels
## [1] "1" "2" "3" "4"
##
## $class
## [1] "factor"
```

1.3. 팩터형 자료 빈도 파악

```
table(score)
```

```
## score
## A B C D
## 4 5 4 1
```

```
# 빈도가 3 이상인 데이터만 출력
tb=table(score)
tb[tb>3]
```

```
## score
## A B C
## 4 5 4
```

1.4. 서수형 자료와 명목형 자료의 차이

```
score[1] > score[3]       # (1)
```

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

```
fac_char[1] > fac_char[2] # (2)
```

```
## Warning in Ops.factor(fac_char[1], fac_char[2]): '>' not meaningful for factors
```

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

```
# 이곳에 주석으로 (1)과 (2) 차이를 서술하고 왜 그런 차이가 생기는지 각자 분석해보세요.

# (1)을 실행하였을 때 score[1]과 score[3]의 대소비교가 가능함.
# (2)를 실행하였을 때 fac_char[1]와 fac_char[2]의 대소비교가 불가능함.
# (1)의 경우 순서가 있는 factor 자료형이지만 (2)의 경우 순서가 없는 명목형 자료형이기 때문.
```

2. Matrix

- 행과 열로 구분된 2차원 형태의 자료
- 주로 실수형 자료를 넣어서 연산하는데 사용
- matrix(data,nrow=행수,ncol=열수,byrow=가로세로배열,dimnames=차원이름)

2.1. matrix 생성

```
mat = matrix(1:8,
             nrow = 2,
             ncol = 4,
             byrow = T) # 1~12의 숫자로 4행3열의 행렬 생성, 가로(열)로 배열

dim(mat)
```

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

```
length(mat) # dim함수는 행, 열 순으로 차원을 출력
```

```
## [1] 8
```

```
matrix(1:8,
       nrow = 2,
       ncol = 4,
       byrow = F)
```

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

```
matrix(1:8, nrow = 2) # 1~8의 수로 2행 배열
```

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

```
matrix(1:8, ncol = 2) # 1~8의 수로 2열 배열
```

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

```
matrix(1:8, ncol = 4, byrow = T)
```

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

```
matrix(1:9,
       nrow = 3,
       ncol = 3,
       dimnames = (list(c("r1", "r2", "r3"), c("c1", "c2", "c3"))))
```

```
##    c1 c2 c3
## r1  1  4  7
## r2  2  5  8
## r3  3  6  9
```

2.2. matrix 각 차원에 이름 부여

```
mat
```

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

```
rownames(mat) = c("행1" , "행 2")
colnames(mat) = c("열 1", "열 2", "열 3", "열 4")
mat
```

```
##      열 1 열 2 열 3 열 4
## 행1     1    2    3    4
## 행 2    5    6    7    8
```

2.3. rbind()와 cbind()를 사용한 매트릭스 생성

```
x = 2:4
x
```

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

```
y = 9:11
y
```

```
## [1]  9 10 11
```

```
cbind(x, y)
```

```
##      x  y
## [1,] 2  9
## [2,] 3 10
## [3,] 4 11
```

```
rbind(x, y)
```

```
##   [,1] [,2] [,3]
## x    2    3    4
## y    9   10   11
```

2.4. rbind()와 cbind()를 사용한 데이터 추가

```
mat
```

```
##      열 1 열 2 열 3 열 4
## 행1     1    2    3    4
## 행 2    5    6    7    8
```

```
cbind(mat, 10:11)
```

```
##      열 1 열 2 열 3 열 4
## 행1     1    2    3    4 10
## 행 2    5    6    7    8 11
```

```
rbind(mat, 20:23)
```

```
##       열 1 열 2 열 3 열 4
## 행1      1    2    3    4
## 행 2     5    6    7    8
##         20   21   22   23
```

2.5. matrix 데이터 접근과 변환

```
X = 1:3
x
```

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

```
y = 10:12
y
```

```
## [1] 10 11 12
```

```
mat = cbind(x, y)
mat
```

```
##      x  y
## [1,] 2 10
## [2,] 3 11
## [3,] 4 12
```

```
mat[1, 1] = 100
mat
```

```
##        x  y
## [1,] 100 10
## [2,]   3 11
## [3,]   4 12
```

```
mat[2, ] = mat[2, ] / 4
mat
```

```
##           x     y
## [1,] 100.00 10.00
## [2,]   0.75  2.75
## [3,]   4.00 12.00
```

```
mat[, 2] = mat [, 2] - mat[, 1] * 3
mat
```

```
##           x      y
## [1,] 100.00 -290.0
## [2,]   0.75    0.5
## [3,]   4.00    0.0
```

3. List

- 리스트는 여러가지 유형을 가진 객체들의 집합.
- 리스트를 이루는 각 객체들을 성분(component)이라고 함.
- 서로 다른 유형을 가진 데이터들로 구성될 수 있다.
- 서로 다른 길이, 차원으로 구성될 수 있다.

3.1. 여러 벡터를 이용해 리스트 만들기

```
str_vec = c("korea", "USA", "Japan")#문자열벡터
num_vec = c(100, 200, 300, 400, 500)#숫자벡터
mat = matrix(2:9, 2, 4)#2*5매트릭스
list_tot = list(str_vec, num_vec, mat)
print(list_tot)
```

```
## [[1]]
## [1] "korea" "USA"   "Japan"
##
## [[2]]
## [1] 100 200 300 400 500
##
## [[3]]
##      [,1] [,2] [,3] [,4]
## [1,]    2    4    6    8
## [2,]    3    5    7    9
```

```
names(list_tot) = c('str_vec', 'num_vec' , 'mat')
print(list_tot)
```

```
## $str_vec
## [1] "korea" "USA"   "Japan"
##
## $num_vec
## [1] 100 200 300 400 500
##
## $mat
##      [,1] [,2] [,3] [,4]
## [1,]    2    4    6    8
## [2,]    3    5    7    9
```

3.2. list함수 내에서 성분의 이름 지정하여 리스트 만들기

```
list_tot2 = list(seq = seq(1, 10, 2),
                 str = c("토끼", "사자", "코끼리", "양"),
                 plus = rep(c('고구마', '감자', '옥수수'), 2))
print(list_tot2)
```

```
## $seq
## [1] 1 3 5 7 9
##
## $str
## [1] "토끼"   "사자"   "코끼리" "양"
##
## $plus
## [1] "고구마" "감자"   "옥수수" "고구마" "감자"   "옥수수"
```

3.3. 데이터의 속성을 확인하는 다양한 함수

```
class(list_tot)
```

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

```
length(list_tot)
```

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

```
attributes(list_tot)
```

```
## $names
## [1] "str_vec" "num_vec" "mat"
```

```
str(list_tot)
```

```
## List of 3
##  $ str_vec: chr [1:3] "korea" "USA" "Japan"
##  $ num_vec: num [1:5] 100 200 300 400 500
##  $ mat    : int [1:2, 1:4] 2 3 4 5 6 7 8 9
```

3.4. 리스트의 성분에 접근하기

```
# []연산자 또 $연산자를 활용해 추출

list_tot2[1] # 첫 번째 성분
```

```
## $seq
## [1] 1 3 5 7 9
```

```
list_tot2[3]
```

```
## $plus
## [1] "고구마" "감자"   "옥수수" "고구마" "감자"   "옥수수"
```

```
list_tot2[1:2]
```

```
## $seq
## [1] 1 3 5 7 9
##
## $str
## [1] "토끼"   "사자"   "코끼리" "양"
```

```
list_tot2$seq # 'seq'라는 성분
```

```
## [1] 1 3 5 7 9
```

```
list_tot2$str
```

```
## [1] "토끼"   "사자"   "코끼리" "양"
```

3.5. 리스트의 성분 안에 있는 원소에 접근하기

```
# [[]]연산자 또는 $연산자와 []로 추출

list_tot[[3]][1] # 2 번째 성분의 첫 번째 원소
```

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

```
list_tot2$seq[3] # seq성분의 세 번째 원소
```

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

```
list_tot2$str[1:2]
```

```
## [1] "토끼" "사자"
```

3.6. 리스트의 성분이나 원소 조작하기

```
# 성분이나 원소 삭제 또는 추가하기

list_tot[1] = NULL # 첫 번째 성분 삭제
str(list_tot)
```

```
## List of 2
##  $ num_vec: num [1:5] 100 200 300 400 500
##  $ mat    : int [1:2, 1:4] 2 3 4 5 6 7 8 9
```

```
list_tot2$str[1] = "고양이" # str 성분 첫 번째 원소 덮어쓰기
str(list_tot2)
```

```
## List of 3
##  $ seq : num [1:5] 1 3 5 7 9
##  $ str : chr [1:4] "고양이" "사자" "코끼리" "양"
##  $ plus: chr [1:6] "고구마" "감자" "옥수수" "고구마" ...
```

```
list_tot$NEW = 2:5 # 새로운 성분 추가
str(list_tot)
```

```
## List of 3
##  $ num_vec: num [1:5] 100 200 300 400 500
##  $ mat    : int [1:2, 1:4] 2 3 4 5 6 7 8 9
##  $ NEW    : int [1:4] 2 3 4 5
```

3.7. 리스트의 성분에 하위 리스트 추가하여 계층적으로 리스트 만들기

```
list_tot$hierarchy[[1]] = list_tot2 # 리스트에 hierarchy라는 성분에 list_ex2를 할당
str(list_tot)
```

```
## List of 4
##  $ num_vec  : num [1:5] 100 200 300 400 500
##  $ mat      : int [1:2, 1:4] 2 3 4 5 6 7 8 9
##  $ NEW      : int [1:4] 2 3 4 5
##  $ hierarchy:List of 1
##   ..$ :List of 3
##   .. ..$ seq : num [1:5] 1 3 5 7 9
##   .. ..$ str : chr [1:4] "고양이" "사자" "코끼리" "양"
##   .. ..$ plus: chr [1:6] "고구마" "감자" "옥수수" "고구마" ...
```

4. Array

4.1. Array 생성하기

```
# array 함수로 array 생성하기

arr = array(1:18, dim = c(3, 3, 2),
            dimnames = list(
              c("KOR", "CHI", "JAP"),
              c("GDP.R", "USD.R", "Cuur.Acc"),
              c("2011Y", "2012Y")
            ))
arr
```

```
## , , 2011Y
##
##     GDP.R USD.R Cuur.Acc
## KOR     1     4        7
## CHI     2     5        8
## JAP     3     6        9
##
## , , 2012Y
##
##     GDP.R USD.R Cuur.Acc
## KOR    10    13       16
## CHI    11    14       17
## JAP    12    15       18
```

```
# 벡터 생성 후 차원을 부여하여 array로 변환하기

arr1 = 1:18
dim(arr1) = c(3, 3, 2)
dimnames(arr1) = list(c("KOR", "CHI", "JAP"),
                      c("GDP.R" , "USD.R", "Cuur.Acc"),
                      c("2011Y", "2012Y"))
arr1
```

```
## , , 2011Y
##
##     GDP.R USD.R Cuur.Acc
## KOR     1     4        7
## CHI     2     5        8
## JAP     3     6        9
##
## , , 2012Y
##
##     GDP.R USD.R Cuur.Acc
## KOR    10    13       16
## CHI    11    14       17
## JAP    12    15       18
```

```
arr1 = arr # 앞에서 만든 배열과 같은지 비교
```

4.2. Array 조작 방법

4.2.1. '[,,]' 인덱싱으로 각 원소에 접근하기

```
arr
```

```
## , , 2011Y
##
##     GDP.R USD.R Cuur.Acc
## KOR     1     4         7
## CHI     2     5         8
## JAP     3     6         9
##
## , , 2012Y
##
##     GDP.R USD.R Cuur.Acc
## KOR    10    13        16
## CHI    11    14        17
## JAP    12    15        18
```

```
arr[1,,] # 한국의 연도별 자료
```

```
##          2011Y 2012Y
## GDP.R        1    10
## USD.R        4    13
## Cuur.Acc     7    16
```

```
arr[,-2,] # 3개국의 GDP.R와 Cuur.Acc
```

```
## , , 2011Y
##
##     GDP.R Cuur.Acc
## KOR     1         7
## CHI     2         8
## JAP     3         9
##
## , , 2012Y
##
##     GDP.R Cuur.Acc
## KOR    10        16
## CHI    11        17
## JAP    12        18
```

```
arr[,,2] # 3개국의 2012년 자료
```

```
##     GDP.R USD.R Cuur.Acc
## KOR    10    13       16
## CHI    11    14       17
## JAP    12    15       18
```

```
arr[,,"2012Y"] # 이름으로 추출 (3 개국의 2012년 자료)
```

```
##     GDP.R USD.R Cuur.Acc
## KOR    10    13       16
## CHI    11    14       17
## JAP    12    15       18
```

```
arr[c(T,T,F),, 2] # 한국, 중국의 2012년 자료 - 일본 제외
```

```
##     GDP.R USD.R Cuur.Acc
## KOR    10    13       16
## CHI    11    14       17
```

```
arr[-2,,2] # 한국, 일본의 2012년 자료 - 중국 제외
```

```
##     GDP.R USD.R Cuur.Acc
## KOR    10    13       16
## JAP    12    15       18
```

4.2.2. 배열 원소의 추출 및 수정

```
arr.tmp = arr
arr.tmp
```

```
## , , 2011Y
##
##     GDP.R USD.R Cuur.Acc
## KOR     1     4        7
## CHI     2     5        8
## JAP     3     6        9
##
## , , 2012Y
##
##     GDP.R USD.R Cuur.Acc
## KOR    10    13       16
## CHI    11    14       17
## JAP    12    15       18
```

```
arr.tmp[, , 1] = c(5, 6, 4)
arr.tmp
```

```
## , , 2011Y
##
##     GDP.R USD.R Cuur.Acc
## KOR     5     5         5
## CHI     6     6         6
## JAP     4     4         4
##
## , , 2012Y
##
##     GDP.R USD.R Cuur.Acc
## KOR    10    13        16
## CHI    11    14        17
## JAP    12    15        18
```

```
arr.tmp[, 1, 2] = NA
arr.tmp
```

```
## , , 2011Y
##
##     GDP.R USD.R Cuur.Acc
## KOR     5     5         5
## CHI     6     6         6
## JAP     4     4         4
##
## , , 2012Y
##
##     GDP.R USD.R Cuur.Acc
## KOR    NA    13        16
## CHI    NA    14        17
## JAP    NA    15        18
```

```
arr.tmp[is.na(arr.tmp)] = c(8, 1)
```

```
## Warning in arr.tmp[is.na(arr.tmp)] = c(8, 1): number of items to replace is not
## a multiple of replacement length
```

```
arr.tmp
```

```
## , , 2011Y
##
##     GDP.R USD.R Cuur.Acc
## KOR     5     5         5
## CHI     6     6         6
## JAP     4     4         4
##
## , , 2012Y
##
##     GDP.R USD.R Cuur.Acc
## KOR     8    13        16
## CHI     1    14        17
## JAP     8    15        18
```

5. 기타

5.1. NA 값 다루기

```
# is.na 함수와 complete.cases 함수를 사용해 결측 값 파악하기

x = c(1, 2, NA, 4, NA, 5)
is.na(x) # NA 값 여부에 대한 논리 판단 결과
```

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

```
bad = is.na(x)
y = x[!bad]
mean(y)
```

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

```
x = c(1, 2, NA, 4, NA, 5)
good = complete.cases(x)
x[good]
```

```
## [1] 1 2 4 5
```

연습문제

연습 1-3 공통 보기

```
load('satellite.RData')
```

연습 1

```
sum(names(paper) == '위성영상')
```

```
## [1] 19
```

```
sum(names(paper) == '딥러닝')
```

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

연습 2

```
sum(names(paper) == '&') # & 삭제 전
```

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

```
paper[paper == paper$'&'] = NULL
sum(names(paper) == '&') # & 삭제 후
```

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

연습 3

```
paper[19] = strsplit(paper[[19]], '\\+')
paper[19]
```

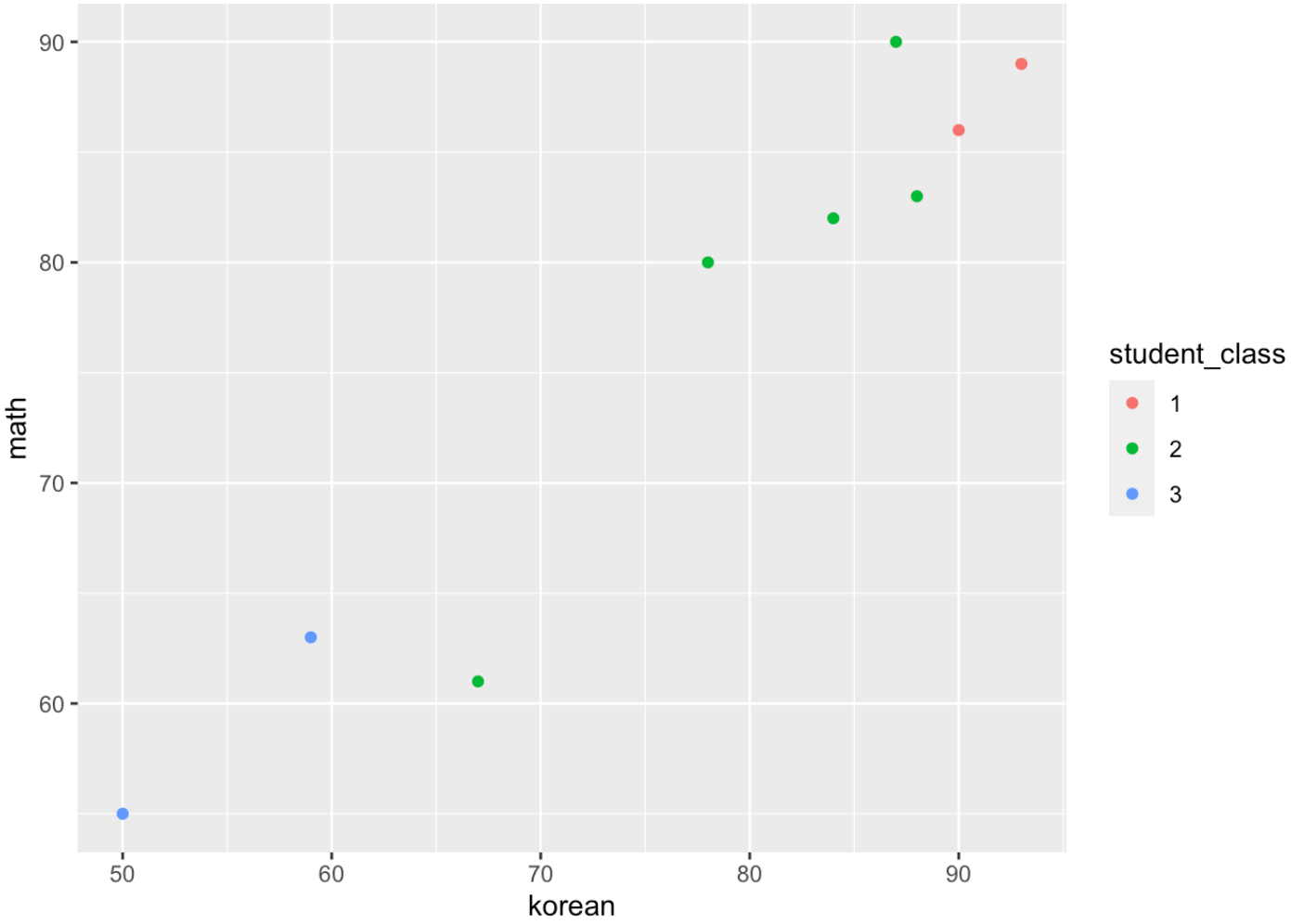
```
## $방법론의
## [1] "방법론/NC" "의/JC"
```

연습 4

```
# 문제점: student_class 데이터는 3개의 종류만 가지고 있지만 이를 가시적으로 확인하기 어려움.
# 해결: student_class의 자료형을 vector 형에서 순서가 있는 factor 형으로 바꾸어 student_class를
 명확히 구분할 수 있도록 함.

korean = c(93, 78, 50, 90, 88, 87, 67, 59, 84)
math = c(89, 80, 55, 86, 83, 90, 61, 63, 82)
student_class = factor(c(1, 2, 3, 1, 2, 2, 2, 3, 2), levels = c(1, 2, 3))

library(ggplot2)
ggplot() + geom_point(aes(x = korean, y = math, colour = student_class))
```


90
80
70
60
math
50
60
70
80
90
korean
student_class
1
2
3

```r
#install.packages('digest')
library(digest)
block1 <- list(
  number = 1,
  timestamp = "2022-09-28",
  data = "세형",
  parent_hash = "0"
)
block1$hash = digest(block1, "sha256")
block2 <- list(
  number = 2,
  timestamp = "2022-09-28",
  data = "재형",
  parent_hash = block1$hash
)
block2$hash = digest(block2, "sha256")
block3 <- list(
  number = 3,
  timestamp = "2022-09-30",
  data = "민철",
  parent_hash = block2$hash
)
block3$hash = digest(block3, "sha256") # sha256 방식의 암호화를 사용하여 block3 객체의 해시값을 block3['hash']에 첨부함.
# 여기까지 기존 장부의 내용
# block4 에 "현승"의 정보를 넣는다.
block4 = list(
  number = 4, # 장부 내역의 순서
  timestamp = "2022-09-30", # 데이터 인입 일자
  data = "현승", # 데이터
  parent_hash = block3$hash # 바로 전의 블록을 암호화해서 첨부하여 이전 블록과의 연속성을 검증할 수 있도록 하면서 동시에 unique한 parent_hash의 값을 이용해서 변조 가능성을 차단함.
)
block4$hash = digest(block4, "sha256") # 이후 block5에서 block4의 해시를 확인할 수 있도록 block4를 암호화하여 block4['hash']에 첨부함.
block4
```

```
## $number
## [1] 4
##
## $timestamp
## [1] "2022-09-30"
##
## $data
## [1] "현승"
##
## $parent_hash
## [1] "b26a18af49a8bfd2731484210e4dc6e355025555572c38266dde40b79da528cd"
##
## $hash
## [1] "7cd7c84324cb799de18958a8c079c1dbc4b63eb5921e1a332487ec42c94e7c5c"
```