4_Subsetting

2022-09-13

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.8
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

4 Subesetting

4.1 Introduction

Quiz

1. What is the result of subsetting a vector with positive integers, negative integers, a logical vector, or a character vector?

Select those elements only, select all but those elements, select all true elements, select all elements that match the name

2. What's the difference between [, [[, and $ when applied to a list?

Takes sub lists and returns lists. extracts element from list, short hand for [[for extracting an element

3. When should you use drop = FALSE?

To maintain a data frame / matrix when subsetting instead of returning a vector

4. If x is a matrix, what does x[] <- 0 do? How is it different from x <- 0?

Sets all values in the matrix to zero, erases the matrix and makes it a 1 element value

5. How can you use a named vector to relabel categorical variables?

Just give a vector of the named elements you want to subset

4.2 Selecting multiple elements

4.2.1 Atomic Vectors

```
x <- c(2.1, 4.2, 3.3, 5.4)
```

```
x[c(3, 1)]
```

```
## [1] 3.3 2.1
```

```
#> [1] 3.3 2.1
x[order(x)]
```

```
## [1] 2.1 3.3 4.2 5.4
```

```
#> [1] 2.1 3.3 4.2 5.4

# Duplicate indices will duplicate values
x[c(1, 1)]
```

```
## [1] 2.1 2.1
```

```
#> [1] 2.1 2.1

# Real numbers are silently truncated to integers
x[c(2.1, 2.9)]
```

```
## [1] 4.2 4.2
```

```
#> [1] 4.2 4.2
```

```
x[-c(3, 1)]
```

```
## [1] 4.2 5.4
```

```
#> [1] 4.2 5.4
```

```
x[c(-1, 2)]
```

```
## Error in x[c(-1, 2)]: only 0's may be mixed with negative subscripts
```

```
#> Error in x[c(-1, 2)]: only 0's may be mixed with negative subscripts
```

```
x[c(TRUE, TRUE, FALSE, FALSE)]
```

```
## [1] 2.1 4.2
```

```
#> [1] 2.1 4.2
x[x > 3]
```

```
## [1] 4.2 3.3 5.4
```

```
#> [1] 4.2 3.3 5.4
```

```
x[c(TRUE, FALSE)]
```

```
## [1] 2.1 3.3
```

```
#> [1] 2.1 3.3
# Equivalent to
x[c(TRUE, FALSE, TRUE, FALSE)]
```

```
## [1] 2.1 3.3
```

```
#> [1] 2.1 3.3
```

```
x[c(TRUE, TRUE, NA, FALSE)]
```

```
## [1] 2.1 4.2  NA
```

```
#> [1] 2.1 4.2  NA
```

```
x[]
```

```
## [1] 2.1 4.2 3.3 5.4
```

```
#> [1] 2.1 4.2 3.3 5.4
```

```
x[0]
```

```
## numeric(0)
```

```
#> numeric(0)
```

```
(y <- setNames(x, letters[1:4]))
```

```
##   a   b   c   d
## 2.1 4.2 3.3 5.4
```

```
#>   a   b   c   d
#> 2.1 4.2 3.3 5.4
y[c("d", "c", "a")]
```

```
##   d   c   a
## 5.4 3.3 2.1
```

```
#>   d   c   a
#> 5.4 3.3 2.1

# Like integer indices, you can repeat indices
y[c("a", "a", "a")]
```

```
##   a   a   a
## 2.1 2.1 2.1
```

```
#>   a   a   a
#> 2.1 2.1 2.1

# When subsetting with [, names are always matched exactly
z <- c(abc = 1, def = 2)
z[c("a", "d")]
```

```
## <NA> <NA>
##   NA   NA
```

```
#> <NA> <NA>
#>   NA   NA
```

```
y[factor("b")]
```

```
##   a
## 2.1
```

```
#>   a
#> 2.1
```

4.2.2 Lists

Using [always returns a list; [[and $, as described in Section 4.3, let you pull out elements of a list

4.2.3 Matrices and arrays

```
a <- matrix(1:9, nrow = 3)
colnames(a) <- c("A", "B", "C")
a
```

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

```
a[1:2, ]
```

```
##      A B C
## [1,] 1 4 7
## [2,] 2 5 8
```

```
#>      A B C
#> [1,] 1 4 7
#> [2,] 2 5 8
a[c(TRUE, FALSE, TRUE), c("B", "A")]
```

```
##      B A
## [1,] 4 1
## [2,] 6 3
```

```
#>      B A
#> [1,] 4 1
#> [2,] 6 3
a[0, -2]
```

```
##      A C
```

```
#>      A C
```

```
a[1, ]
```

```
## A B C
## 1 4 7
```

```
#> A B C
#> 1 4 7
a[1, 1]
```

```
## A
## 1
```

```
#> A
#> 1
```

```
vals <- outer(1:5, 1:5, FUN = "paste", sep = ",")
vals
```

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

```
#>      [,1]  [,2]  [,3]  [,4]  [,5]
#> [1,] "1,1" "1,2" "1,3" "1,4" "1,5"
#> [2,] "2,1" "2,2" "2,3" "2,4" "2,5"
#> [3,] "3,1" "3,2" "3,3" "3,4" "3,5"
#> [4,] "4,1" "4,2" "4,3" "4,4" "4,5"
#> [5,] "5,1" "5,2" "5,3" "5,4" "5,5"

vals[c(4, 15)]
```

```
## [1] "4,1" "5,3"
```

```
#> [1] "4,1" "5,3"
```

```
select <- matrix(ncol = 2, byrow = TRUE, c(
  1, 1,
  3, 1,
  2, 4
))
vals[select]
```

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

```
#> [1] "1,1" "3,1" "2,4"
```

4.2.4 Data frames and tibbles

```
df <- data.frame(x = 1:3, y = 3:1, z = letters[1:3])

df[df$x == 2, ]
```

```
##   x y z
## 2 2 2 b
```

```
#>   x y z
#> 2 2 2 b
df[c(1, 3), ]
```

```
##   x y z
## 1 1 3 a
## 3 3 1 c
```

```
#>   x y z
#> 1 1 3 a
#> 3 3 1 c

# There are two ways to select columns from a data frame
# Like a list
df[c("x", "z")]
```

```
##   x z
## 1 1 a
## 2 2 b
## 3 3 c
```

```
#>   x z
#> 1 1 a
#> 2 2 b
#> 3 3 c
# Like a matrix
df[, c("x", "z")]
```

```
##   x z
## 1 1 a
## 2 2 b
## 3 3 c
```

```
#>   x z
#> 1 1 a
#> 2 2 b
#> 3 3 c

# There's an important difference if you select a single
# column: matrix subsetting simplifies by default, list
# subsetting does not.
str(df["x"])
```

```
## 'data.frame':    3 obs. of  1 variable:
##  $ x: int  1 2 3
```

```
#> 'data.frame':    3 obs. of  1 variable:
#>  $ x: int  1 2 3
str(df[, "x"])
```

```
##  int [1:3] 1 2 3
```

```
#>  int [1:3] 1 2 3
```

```
df <- tibble::tibble(x = 1:3, y = 3:1, z = letters[1:3])

str(df["x"])
```

```
## tibble [3 x 1] (S3: tbl_df/tbl/data.frame)
##  $ x: int [1:3] 1 2 3
```

```
#> tibble [3 × 1] (S3: tbl_df/tbl/data.frame)
#>  $ x: int [1:3] 1 2 3
str(df[, "x"])
```

```
## tibble [3 x 1] (S3: tbl_df/tbl/data.frame)
##  $ x: int [1:3] 1 2 3
```

```
#> tibble [3 × 1] (S3: tbl_df/tbl/data.frame)
#>  $ x: int [1:3] 1 2 3
```

4.2.5 Preserving dimensionality

```
a <- matrix(1:4, nrow = 2)
str(a[1, ])
```

```
##  int [1:2] 1 3
```

```
#>  int [1:2] 1 3
```

```
str(a[1, , drop = FALSE])
```

```
##  int [1, 1:2] 1 3
```

```
#>  int [1, 1:2] 1 3
```

```
df <- data.frame(a = 1:2, b = 1:2)
str(df[, "a"])
```

```
##  int [1:2] 1 2
```

```
#>  int [1:2] 1 2
```

```
str(df[, "a", drop = FALSE])
```

```
## 'data.frame':    2 obs. of  1 variable:
##  $ a: int  1 2
```

```
#> 'data.frame':    2 obs. of  1 variable:
#>  $ a: int  1 2
```

```
z <- factor(c("a", "b"))
z[1]
```

```
## [1] a
## Levels: a b
```

```
#> [1] a
#> Levels: a b
z[1, drop = TRUE]
```

```
## [1] a
## Levels: a
```

```
#> [1] a
#> Levels: a
```

4.2.6 Exercises

```
#mtcars[mtcars$cyl = 4, ]
mtcars[mtcars$cyl == 4, ]
```

1. Fix each of the following common data frame subsetting errors:

```
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230       22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona  21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

```
#mtcars[-1:4, ]
mtcars[-c(1:4), ]
```

```
##                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
```

```
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

```
#mtcars[mtcars$cyl <= 5]
mtcars[mtcars$cyl <= 5,]
```

```
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230       22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona  21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

```
#mtcars[mtcars$cyl == 4 | 6, ]
mtcars[mtcars$cyl == c(4,6), ]
```

```
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Valiant        18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Merc 230       22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280       19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corona  21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
```

```
x <- 1:5
x[NA]
```

2. Why does the following code yield five missing values? (Hint: why is it different from x[NA_real_]?)

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

```
#> [1] NA NA NA NA NA
x[NA_real_]
```

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

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

NA is not applicable so it gives 5 NAs because it can't tell for each number if it is the same or not. With x[NA_real_] you are pulling real NAs and they don't exist so you get NA

```
x <- outer(1:5, 1:5, FUN = "*")
x
```

3. What does upper.tri() return? How does subsetting a matrix with it work? Do we need any additional subsetting rules to describe its behaviour?

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    2    4    6    8   10
## [3,]    3    6    9   12   15
## [4,]    4    8   12   16   20
## [5,]    5   10   15   20   25
```

```
x[upper.tri(x)]
```

```
##  [1]  2  3  6  4  8 12  5 10 15 20
```

```
x[lower.tri(x)]
```

```
##  [1]  2  3  4  5  6  8 10 12 15 20
```

Upper triangle of the matrix not including the diagonal. It returns a same size matrix of T and F. Matrix subsetting with a matrix which strips the dimensionality of the matrix and makes it a 1D vector

```
dim(mtcars)
```

4. Why does mtcars[1:20] return an error? How does it differ from the similar mtcars[1:20,]?

```
## [1] 32 11
```

There's only 11 columns so `mtcars[1:20]` is asking for more columns than exist in the data

```
diag_alt <- function(m){
  mat <- matrix(FALSE,
                nrow = dim(m)[1],
                ncol = dim(m)[2])
  for(i in 1:(min(dim(m)))){
```

```
    mat[i,i] <- TRUE
  }
  return(m[mat])
}

x <- outer(1:5, 1:5, FUN = "*")
diag(x)
```

5. Implement your own function that extracts the diagonal entries from a matrix (it should behave like diag(x) where x is a matrix).

```
## [1]  1  4  9 16 25
```

```
diag_alt(x)
```

```
## [1]  1  4  9 16 25
```

6. What does df[is.na(df)] <- 0 do? How does it work? Any NA elements in the data frame are converted to 0 using a logical matrix

4.3 Selecting a single element

4.3.1 "[["

```
x <- list(1:3, "a", 4:6)
```

```
# Bad
for (i in 2:length(x)) {
  out[i] <- fun(x[i], out[i - 1])
}
```

```
## Error in fun(x[i], out[i - 1]): could not find function "fun"
```

```
# Good
for (i in 2:length(x)) {
  out[[i]] <- fun(x[[i]], out[[i - 1]])
}
```

```
## Error in fun(x[[i]], out[[i - 1]]): could not find function "fun"
```

4.3.2 "$"

```
var <- "cyl"
# Doesn't work - mtcars$var translated to mtcars[["var"]]
mtcars$var
```

```
## NULL
```

```
#> NULL

# Instead use [[
mtcars[[var]]
```

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

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

```
# Partial match with $
x <- list(abc = 1)
x$a
```

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

```
#> [1] 1
x[["a"]]
```

```
## NULL
```

```
#> NULL
```

```
options(warnPartialMatchDollar = TRUE)
x$a
```

```
## Warning in x$a: partial match of 'a' to 'abc'

## [1] 1
```

```
#> Warning in x$a: partial match of 'a' to 'abc'
#> [1] 1
```

4.3.3 Missing and out-of-bounds indices

```
x <- list(
  a = list(1, 7, 3),
  b = list(3, 4, 5)
)

purrr::pluck(x, "a", 2)
```

```
## [1] 7
```

```
#> [1] 1

purrr::pluck(x, "c", 1)
```

```
## NULL
```

```
#> NULL
```

```
purrr::pluck(x, "c", 1, .default = NA)
```

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

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

4.3.4 @ and slot()

There are two additional subsetting operators, which are needed for S4 objects: @ (equivalent to $), and slot() (equivalent to [[). @ is more restrictive than $ in that it will return an error if the slot does not exist. These are described in more detail in Chapter 15.

4.3.5 Exercises

```
mtcars[3,2]
```

1. Brainstorm as many ways as possible to extract the third value from the cyl variable in the mtcars dataset.

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

```
mtcars$cyl[3]
```

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

```
mtcars[["cyl"]][3]
```

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

```
as.matrix(mtcars)[35]
```

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

```
as.matrix(mtcars)[[35]]
```

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

```
mod <- lm(mpg ~ wt, data = mtcars)
mod$df.residual
```

2. Given a linear model, e.g., mod <- lm(mpg ~ wt, data = mtcars), extract the residual degrees of freedom. Then extract the R squared from the model summary (summary(mod))

```
## [1] 30
```

```r
summod <- summary(mod)
summod$r.squared
```

```
## [1] 0.7528328
```

4.4 Subsetting and assignment

```r
x <- 1:5
x[c(1, 2)] <- c(101, 102)
x
```

```
## [1] 101 102   3   4   5
```

```
#> [1] 101 102   3   4   5
```

```r
x <- list(a = 1, b = 2)
x[["b"]] <- NULL
str(x)
```

```
## List of 1
##  $ a: num 1
```

```
#> List of 1
#>  $ a: num 1
```

```r
y <- list(a = 1, b = 2)
y["b"] <- list(NULL)
str(y)
```

```
## List of 2
##  $ a: num 1
##  $ b: NULL
```

```
#> List of 2
#>  $ a: num 1
#>  $ b: NULL
```

```r
mtcars[] <- lapply(mtcars, as.integer)
is.data.frame(mtcars)
```

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

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

```r
mtcars <- lapply(mtcars, as.integer)
is.data.frame(mtcars)
```

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

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

4.5 Applications

4.5.1 Lookup tables (character subsetting)

```
x <- c("m", "f", "u", "f", "f", "m", "m")
lookup <- c(m = "Male", f = "Female", u = NA)
lookup[x]
```

```
##        m        f        u        f        f        m        m
##   "Male" "Female"       NA "Female" "Female"   "Male"   "Male"
```

```
#>        m        f        u        f        f        m        m
#>   "Male" "Female"       NA "Female" "Female"   "Male"   "Male"
```

```
unname(lookup[x])
```

```
## [1] "Male"   "Female" NA       "Female" "Female" "Male"   "Male"
```

```
#> [1] "Male"   "Female" NA       "Female" "Female" "Male"   "Male"
```

4.5.2 Matching and merging by hand (integer subsetting)

```
grades <- c(1, 2, 2, 3, 1)

info <- data.frame(
  grade = 3:1,
  desc = c("Excellent", "Good", "Poor"),
  fail = c(F, F, T)
)
```

```
id <- match(grades, info$grade)
id
```

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

```
#> [1] 3 2 2 1 3
info[id, ]
```

```
##     grade      desc  fail
## 3       1      Poor  TRUE
## 2       2      Good FALSE
## 2.1     2      Good FALSE
## 1       3 Excellent FALSE
## 3.1     1      Poor  TRUE
```

```
#>     grade      desc  fail
#> 3       1      Poor  TRUE
#> 2       2      Good FALSE
#> 2.1     2      Good FALSE
#> 1       3 Excellent FALSE
#> 3.1     1      Poor  TRUE
```

4.5.3 Random samples and bootstraps (integer subsetting)

```
df <- data.frame(x = c(1, 2, 3, 1, 2), y = 5:1, z = letters[1:5])

# Randomly reorder
df[sample(nrow(df)), ]
```

```
##   x y z
## 1 1 5 a
## 4 1 2 d
## 5 2 1 e
## 2 2 4 b
## 3 3 3 c
```

```
#>   x y z
#> 5 2 1 e
#> 3 3 3 c
#> 4 1 2 d
#> 1 1 5 a
#> 2 2 4 b

# Select 3 random rows
df[sample(nrow(df), 3), ]
```

```
##   x y z
## 5 2 1 e
## 1 1 5 a
## 2 2 4 b
```

```
#>   x y z
#> 4 1 2 d
#> 2 2 4 b
#> 1 1 5 a

# Select 6 bootstrap replicates
df[sample(nrow(df), 6, replace = TRUE), ]
```

```
##     x y z
## 5   2 1 e
## 1   1 5 a
## 3   3 3 c
## 2   2 4 b
## 3.1 3 3 c
## 4   1 2 d
```

```
#>     x y z
#> 5   2 1 e
#> 5.1 2 1 e
#> 5.2 2 1 e
#> 2   2 4 b
#> 3   3 3 c
#> 3.1 3 3 c
```

4.5.4 Ordering (integer subsetting)

```
x <- c("b", "c", "a")
order(x)
```

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

```
#> [1] 3 1 2
x[order(x)]
```

```
## [1] "a" "b" "c"
```

```
#> [1] "a" "b" "c"
```

By default, any missing values will be put at the end of the vector; however, you can remove them with na.last = NA or put them at the front with na.last = FALSE.

```
# Randomly reorder df
df2 <- df[sample(nrow(df)), 3:1]
df2
```

```
##   z y x
## 5 e 1 2
## 2 b 4 2
## 1 a 5 1
## 4 d 2 1
## 3 c 3 3
```

```
#>   z y x
#> 5 e 1 2
#> 1 a 5 1
#> 4 d 2 1
#> 2 b 4 2
#> 3 c 3 3

df2[order(df2$x), ]
```

```
##   z y x
## 1 a 5 1
## 4 d 2 1
## 5 e 1 2
## 2 b 4 2
## 3 c 3 3
```

```
#>   z y x
#> 1 a 5 1
#> 4 d 2 1
#> 5 e 1 2
#> 2 b 4 2
#> 3 c 3 3
df2[, order(names(df2))]
```

```
##   x y z
## 5 2 1 e
## 2 2 4 b
## 1 1 5 a
## 4 1 2 d
## 3 3 3 c
```

```
#>   x y z
#> 5 2 1 e
#> 1 1 5 a
#> 4 1 2 d
#> 2 2 4 b
#> 3 3 3 c
```

4.5.5 Expanding aggregated counts (integer subsetting)

```
df <- data.frame(x = c(2, 4, 1), y = c(9, 11, 6), n = c(3, 5, 1))
df
```

```
##   x  y n
## 1 2  9 3
## 2 4 11 5
## 3 1  6 1
```

```
rep(1:nrow(df), df$n)
```

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

```
#> [1] 1 1 1 2 2 2 2 2 3
```

```
df[rep(1:nrow(df), df$n), ]
```

```
##     x  y n
## 1   2  9 3
## 1.1 2  9 3
## 1.2 2  9 3
## 2   4 11 5
## 2.1 4 11 5
## 2.2 4 11 5
## 2.3 4 11 5
## 2.4 4 11 5
## 3   1  6 1
```

```
#>     x  y n
#> 1   2  9 3
#> 1.1 2  9 3
#> 1.2 2  9 3
#> 2   4 11 5
#> 2.1 4 11 5
#> 2.2 4 11 5
#> 2.3 4 11 5
#> 2.4 4 11 5
#> 3   1  6 1
```

4.5.6 Removing columns from data frames (character)

```
df <- data.frame(x = 1:3, y = 3:1, z = letters[1:3])
df$z <- NULL
```

```
df <- data.frame(x = 1:3, y = 3:1, z = letters[1:3])
df[c("x", "y")]
```

```
##   x y
## 1 1 3
## 2 2 2
## 3 3 1
```

```
#>   x y
#> 1 1 3
#> 2 2 2
#> 3 3 1
```

```
df[setdiff(names(df), "z")]
```

```
##   x y
## 1 1 3
## 2 2 2
## 3 3 1
```

```
#>   x y
#> 1 1 3
#> 2 2 2
#> 3 3 1
```

4.5.7 Selecting rows based on a condition (logical subsetting)

```
rm(mtcars)
mtcars[mtcars$gear == 5, ]
```

```
##                 mpg cyl  disp  hp drat    wt qsec vs am gear carb
```

```
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.7  0  1    5    2
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
## Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.5  0  1    5    4
## Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.5  0  1    5    6
## Maserati Bora  15.0   8 301.0 335 3.54 3.570 14.6  0  1    5    8
```

```
#>                mpg cyl  disp  hp drat   wt qsec vs am gear carb
#> Porsche 914-2  26.0   4 120.3  91 4.43 2.14 16.7  0  1    5    2
#> Lotus Europa   30.4   4  95.1 113 3.77 1.51 16.9  1  1    5    2
#> Ford Pantera L 15.8   8 351.0 264 4.22 3.17 14.5  0  1    5    4
#> Ferrari Dino   19.7   6 145.0 175 3.62 2.77 15.5  0  1    5    6
#> Maserati Bora  15.0   8 301.0 335 3.54 3.57 14.6  0  1    5    8
```

```
mtcars[mtcars$gear == 5 & mtcars$cyl == 4, ]
```

```
##                mpg cyl  disp  hp drat    wt qsec vs am gear carb
## Porsche 914-2 26.0   4 120.3  91 4.43 2.140 16.7  0  1    5    2
## Lotus Europa  30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
```

```
#>                mpg cyl  disp  hp drat   wt qsec vs am gear carb
#> Porsche 914-2 26.0   4 120.3  91 4.43 2.14 16.7  0  1    5    2
#> Lotus Europa  30.4   4  95.1 113 3.77 1.51 16.9  1  1    5    2
```

!(X & Y) is the same as !X | !Y !(X | Y) is the same as !X & !Y

4.5.8 Boolean algebra versus sets (logical and integer)

```
x <- sample(10) < 4
which(x)
```

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

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

```
unwhich <- function(x, n) {
  out <- rep_len(FALSE, n)
  out[x] <- TRUE
  out
}
unwhich(which(x), 10)
```

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

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

```
(x1 <- 1:10 %% 2 == 0)
```

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

```
#>  [1] FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE
(x2 <- which(x1))
```

```
## [1]  2  4  6  8 10
```

```
#> [1]  2  4  6  8 10
(y1 <- 1:10 %% 5 == 0)
```

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

```
#>  [1] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE
(y2 <- which(y1))
```

```
## [1]  5 10
```

```
#> [1]  5 10

# X & Y <-> intersect(x, y)
x1 & y1
```

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

```
#>  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
intersect(x2, y2)
```

```
## [1] 10
```

```
#> [1] 10

# X | Y <-> union(x, y)
x1 | y1
```

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

```
#>  [1] FALSE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE
union(x2, y2)
```

```
## [1]  2  4  6  8 10  5
```

```
#> [1]  2  4  6  8 10  5

# X & !Y <-> setdiff(x, y)
x1 & !y1
```

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

```
#>  [1] FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE FALSE
setdiff(x2, y2)
```

```
## [1] 2 4 6 8
```

```
#> [1] 2 4 6 8

# xor(X, Y) <-> setdiff(union(x, y), intersect(x, y))
xor(x1, y1)
```

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

```
#>  [1] FALSE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE
setdiff(union(x2, y2), intersect(x2, y2))
```

```
## [1] 2 4 6 8 5
```

```
#> [1] 2 4 6 8 5
```

4.5.9 Exercises

```
x <- matrix(1:25, ncol = 5)
x
```

1. How would you randomly permute the columns of a data frame? (This is an important technique in random forests.) Can you simultaneously permute the rows and columns in one step?

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    6   11   16   21
## [2,]    2    7   12   17   22
## [3,]    3    8   13   18   23
## [4,]    4    9   14   19   24
## [5,]    5   10   15   20   25
```

```
x[,sample(ncol(x))]
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    6    1   21   11   16
## [2,]    7    2   22   12   17
## [3,]    8    3   23   13   18
## [4,]    9    4   24   14   19
## [5,]   10    5   25   15   20
```

```
x[sample(nrow(x)),sample(ncol(x))]
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   18    8   13   23    3
## [2,]   17    7   12   22    2
## [3,]   20   10   15   25    5
## [4,]   19    9   14   24    4
## [5,]   16    6   11   21    1
```

```
m <- 3
start <- sample(nrow(x) - m + 1, 1)
end <- start + m -1
x[start:end, , drop = F]
```

2. How would you select a random sample of m rows from a data frame? What if the sample had to be contiguous (i.e., with an initial row, a final row, and every row in between)?

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    2    7   12   17   22
## [2,]    3    8   13   18   23
## [3,]    4    9   14   19   24
```

```
df <- data.frame(a = 1, b = 2, d = 4, e = 5, c = 3)
df
```

3. How could you put the columns in a data frame in alphabetical order?

```
##   a b d e c
## 1 1 2 4 5 3
```

```
df[,order(df)]
```

```
## Warning in xtfrm.data.frame(x): cannot xtfrm data frames
```

```
##   a b c d e
## 1 1 2 3 4 5
```