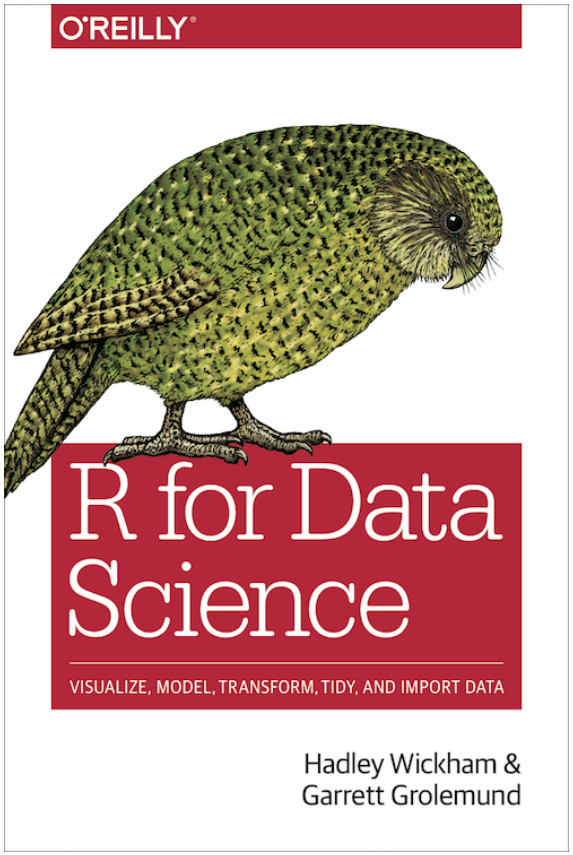
Socio-Informatics 348

Data Visualisation
Logical Vectors

Dr Lisa Martin

Department of Information Science
Stellenbosch University

Today's Reading



R for Data Science, Chapter 12

Introduction

- Logical vectors have only three possible values: `TRUE`, `FALSE`, and `NA`.
- Rare in raw data, but central to filtering, transformations, and conditional logic.
- We'll learn how to create, combine, summarise, and apply logic with `if_else()` and `case_when()`.
- Base R functions, plus `mutate()`, `filter()`, etc.

Introduction

- Early on, you were shown how basic arithmetic/functions work on vectors:

```
x <- c(1, 2, 3, 5, 7, 11, 13)
x * 2
#> [1]  2  4  6 10 14 22 26
```

- Any manipulations done to a free-floating vector can be done to a variable inside a table too:

```
df <- tibble(x)
df |>
  mutate(y = x * 2)
#> # A tibble: 7 × 2
#>       x     y
#>   <dbl> <dbl>
#> 1     1     2
#> 2     2     4
#> 3     3     6
#> 4     5    10
#> 5     7    14
#> 6    11    22
#> # i 1 more row
```

Comparisons

- Common creation through numeric comparisons:
  $<, <=, >, >=, ==, !=$.
- You've already been creating these vectors transiently in `filter()`:

```
flights |>
  filter(dep_time > 600 & dep_time < 2000 & abs(arr_delay) < 20)
#> # A tibble: 172,286 × 19
#>    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
#>   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
#> 1  2013     1     1      601            600         1      844            850
#> 2  2013     1     1      602            610        -8      812            820
#> 3  2013     1     1      602            605        -3      821            805
#> 4  2013     1     1      606            610        -4      858            910
#> 5  2013     1     1      606            610        -4      837            845
#> 6  2013     1     1      607            607         0      858            915
#> # i 172,280 more rows
#> # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>, …
```

- But you can also create logical vectors directly as variables.

Comparisons

- Common creation through numeric comparisons: $<, <=, >, >=, ==, !=$.
- You've already been creating these vectors transiently in `filter()`.
- But you can also create logical vectors directly as variables.
- Using `mutate()`:

```
flights |>
  mutate(
    daytime = dep_time > 600 & dep_time < 2000,
    approx_ontime = abs(arr_delay) < 20,
    .keep = "used"
  )
#> # A tibble: 336,776 × 4
#>   dep_time arr_delay daytime approx_ontime
#>      <int>     <dbl> <lgl>   <lgl>
#> 1      517        11 FALSE   TRUE
#> 2      533        20 FALSE   FALSE
#> 3      542        33 FALSE   FALSE
#> 4      544       -18 FALSE   TRUE
#> 5      554       -25 FALSE   FALSE
#> 6      554        12 FALSE   TRUE
#> # i 336,770 more rows
```

Comparisons: Floating point

Beware of using == with floating point numbers!

```
x <- c(1 / 49 * 49, sqrt(2) ^ 2)
x
#> [1] 1 2
```

```
x == c(1, 2)
#> [1] FALSE FALSE
```

```
print(x, digits = 16)
#> [1] 0.9999999999999999 2.0000000000000004
```

```
near(x, c(1, 2))
#> [1] TRUE TRUE
```

Comparisons: NA values

NA values are contagious!

- These all return `NA`:
  - `NA > 5`
  - `10 == NA`
  - `NA == NA`
- Do you recall how including `NA` values affected `mean()` and `sum()` too?
- It helps to think of `NA` as 'unknown'

Comparisons: NA values

NA values are contagious!

- It helps to think of `NA` as 'unknown':

```
# We don't know how old Mary is
age_mary <- NA

# We don't know how old John is
age_john <- NA

# Are Mary and John the same age?
age_mary == age_john
#> [1] NA
# We don't know!
```

Comparisons: NA values

Use `is.na()` instead of `== NA`

```
flights |>
  filter(dep_time == NA)
#> # A tibble: 0 × 19
#> # ℹ 19 variables: year <int>, month <int>, day <int>, dep_time <int>,
#> #   sched_dep_time <int>, dep_delay <dbl>, arr_time <int>, …
```

```
flights |>
  filter(is.na(dep_time))
#> # A tibble: 8,255 × 19
#>    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
#>   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
#> 1  2013     1     1       NA           1630        NA       NA           1815
#> 2  2013     1     1       NA           1935        NA       NA           2240
#> 3  2013     1     1       NA           1500        NA       NA           1825
#> 4  2013     1     1       NA            600        NA       NA            901
#> 5  2013     1     2       NA           1540        NA       NA           1747
#> 6  2013     1     2       NA           1620        NA       NA           1746
#> # ℹ 8,249 more rows
#> # ℹ 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>, …
```

Boolean Algebra

- Combine logical vectors: `&`, `|`, `!`, `xor()`.

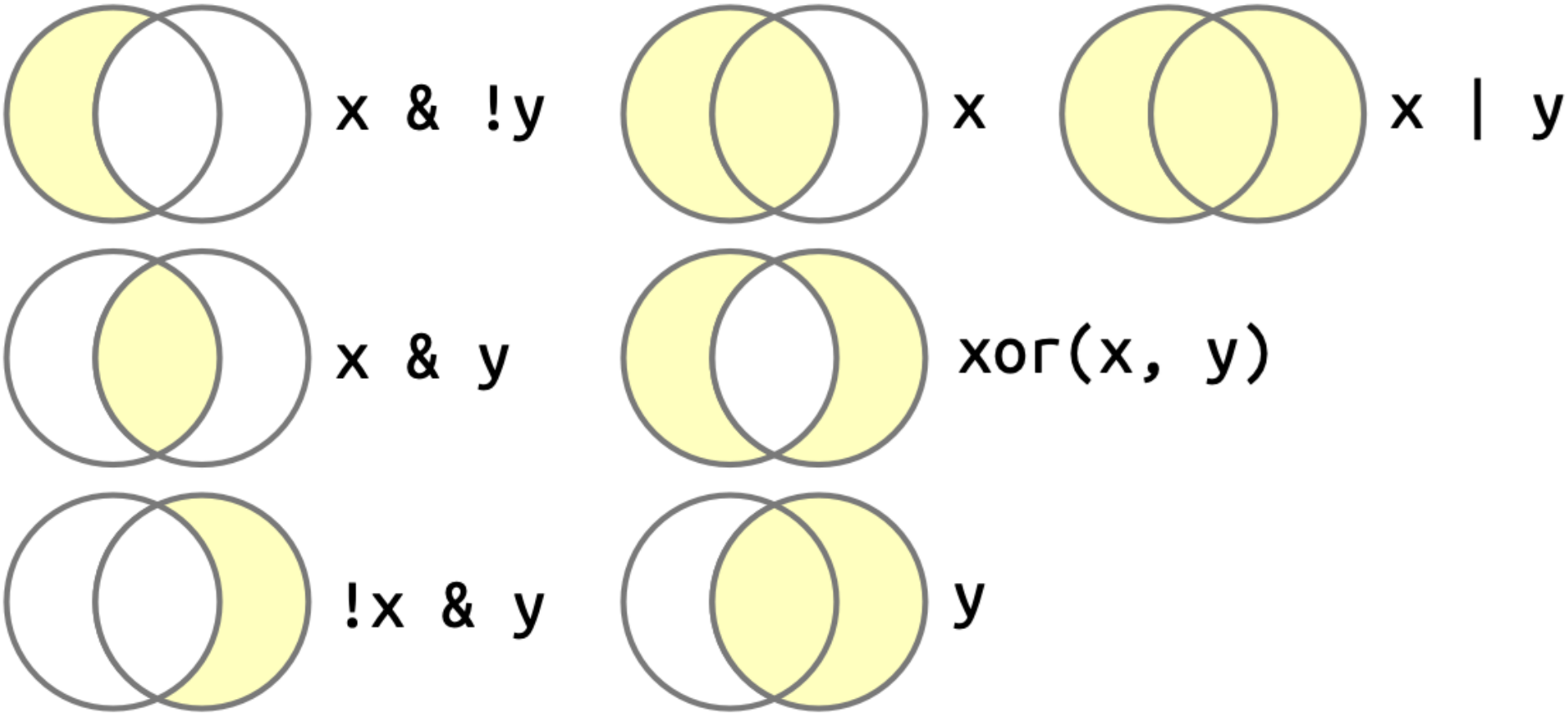


Figure 12.1 from R for Data Science

Boolean Algebra

Order of operations:

```
flights |>
   filter(month == 11 | month == 12)
```

```
flights |>
   filter(month == 11 | 12)
#> # A tibble: 336,776 × 19
#>    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
#>   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
#> 1  2013     1     1      517            515         2      830            819
#> 2  2013     1     1      533            529         4      850            830
#> 3  2013     1     1      542            540         2      923            850
#> 4  2013     1     1      544            545        -1     1004           1022
#> 5  2013     1     1      554            600        -6      812            837
#> 6  2013     1     1      554            558        -4      740            728
#> # i 336,770 more rows
#> # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>, …
```

- In these operations, any number higher than 0 is treated as `TRUE`.
- `(anything | TRUE)` is always `TRUE`

Boolean Algebra

%in%:

- Short-hand for multiple == comparisons combined with |.

```
1:12 %in% c(1, 5, 11)
#>  [1]  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
letters[1:10] %in% c("a", "e", "i", "o", "u")
#>  [1]  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE
```

```
flights |>
  filter(month %in% c(11, 12))
```

- Can also be used with NA values (`NA %in% NA` returns `TRUE`):

```
c(1, 2, NA) == NA
#> [1] NA NA NA
c(1, 2, NA) %in% NA
#> [1] FALSE FALSE  TRUE
```

- But best practice is to use `is.na()`.

Summaries of Logical Vectors

There are two main logical summaries:

- `any(x)` $\longrightarrow$ `TRUE` if any element is `TRUE`;
  `all(x)` $\longrightarrow$ `TRUE` only if all are `TRUE`.

```
flights |>
  group_by(year, month, day) |>
  summarize(
    all_delayed = all(dep_delay <= 60, na.rm = TRUE),
    any_long_delay = any(arr_delay >= 300, na.rm = TRUE),
    .groups = "drop"
  )
#> # A tibble: 365 × 5
#>    year month   day all_delayed any_long_delay
#>   <int> <int> <int> <lgl>       <lgl>
#> 1  2013     1     1 FALSE       TRUE
#> 2  2013     1     2 FALSE       TRUE
#> 3  2013     1     3 FALSE       FALSE
#> 4  2013     1     4 FALSE       FALSE
#> 5  2013     1     5 FALSE       TRUE
#> 6  2013     1     6 FALSE       FALSE
#> # ℹ 359 more rows
```

all delayed by AT MOST an hour

Summaries of Logical Vectors

Numeric Summaries:

- We can get more info out of these logical vectors by treating them as numbers.
- In R, `TRUE` is 1 and `FALSE` is 0.
- `sum(x)` counts how many are `TRUE`
- `mean(x)` gives proportion (because it's `sum(x) / length(x)`)

Summaries of Logical Vectors

Numeric Summaries:

```
flights |>
  group_by(year, month, day) |>
  summarize(
    proportion_delayed = mean(dep_delay <= 60, na.rm = TRUE),
    count_long_delay = sum(arr_delay >= 300, na.rm = TRUE),
    .groups = "drop"
  )
#> # A tibble: 365 × 5
#>    year month   day proportion_delayed count_long_delay
#>   <int> <int> <int>              <dbl>             <int>
#> 1  2013     1     1              0.939                 3
#> 2  2013     1     2              0.914                 3
#> 3  2013     1     3              0.941                 0
#> 4  2013     1     4              0.953                 0
#> 5  2013     1     5              0.964                 1
#> 6  2013     1     6              0.959                 0
#> # ℹ 359 more rows
```

Summaries of Logical Vectors

Logical Subsetting:

- When interested in a sub-group, one option is to use `filter()`.

```
flights |>
  filter(arr_delay > 0) |>
  group_by(year, month, day) |>
  summarize(
    behind = mean(arr_delay),
    n = n(),
    .groups = "drop"
  )
#> # A tibble: 365 × 5
#>    year month   day behind     n
#>   <int> <int> <int>  <dbl> <int>
#> 1  2013     1     1   32.5   461
#> 2  2013     1     2   32.0   535
#> 3  2013     1     3   27.7   460
#> 4  2013     1     4   28.3   297
#> 5  2013     1     5   22.6   238
#> 6  2013     1     6   24.4   381
#> # i 359 more rows
```

- But what if you have multiple sub-groups you're interested in?

Summaries of Logical Vectors

Logical Subsetting:

- Alternative: Use the subset operator []

```
flights |>
  group_by(year, month, day) |>
  summarize(
    behind = mean(arr_delay[arr_delay > 0], na.rm = TRUE),
    ahead = mean(arr_delay[arr_delay < 0], na.rm = TRUE),
    n = n(),
    .groups = "drop"
  )
#> # A tibble: 365 × 6
#>    year month   day behind ahead     n
#>   <int> <int> <int>  <dbl> <dbl> <int>
#> 1  2013     1     1   32.5 -12.5   842
#> 2  2013     1     2   32.0 -14.3   943
#> 3  2013     1     3   27.7 -18.2   914
#> 4  2013     1     4   28.3 -17.0   915
#> 5  2013     1     5   22.6 -14.0   720
#> 6  2013     1     6   24.4 -13.6   832
#> # ℹ 359 more rows
```

Conditional Transformations — `if_else()`

- `if_else(condition, true, false, [missing])`

```
x <- c(-3:3, NA)
if_else(x > 0, "+ve", "-ve")
#> [1] "-ve" "-ve" "-ve" "-ve" "+ve" "+ve" "+ve" NA
```

- Handling missing:

```
if_else(x > 0, "+ve", "-ve", "???")
#> [1] "-ve" "-ve" "-ve" "-ve" "+ve" "+ve" "+ve" "???"
```

- Nest `if_else()` to handle zero explicitly.

```
if_else(x == 0, "0", if_else(x < 0, "-ve", "+ve"), "???")
#> [1] "-ve" "-ve" "-ve" "0"   "+ve" "+ve" "+ve" "???"
```

Conditional Transformations — `case_when()`

- Flexible alternative to `if_else()`.
- Inspired by SQL's `CASE` statement.
- Example labeling flight status:

```
flights |>
  mutate(
    status = case_when(
      is.na(arr_delay)      ~ "cancelled",
      arr_delay < -30       ~ "very early",
      arr_delay < -15       ~ "early",
      abs(arr_delay) <= 15  ~ "on time",
      arr_delay < 60        ~ "late",
      arr_delay < Inf       ~ "very late",
    ),
```

- Note: ORDER MATTERS; Only first matching condition applies.

Conditional Transformations — `case_when()`

```
flights |>
  mutate(
    status = case_when(
      is.na(arr_delay)      ~ "cancelled",
      arr_delay < -30       ~ "very early",
      arr_delay < -15       ~ "early",
      abs(arr_delay) <= 15  ~ "on time",
      arr_delay < 60        ~ "late",
      arr_delay < Inf       ~ "very late",
    ),
    .keep = "used"
  )
#> # A tibble: 336,776 × 2
#>   arr_delay status
#>       <dbl> <chr>
#> 1        11 on time
#> 2        20 late
#> 3        33 late
#> 4       -18 early
#> 5       -25 early
#> 6        12 on time
#> # ℹ 336,770 more rows
```

Compatible Types

- `if_else()` and `case_when()` require compatible output types.
- Compatible types include:
  - Numeric and logical
  - Strings and factors
  - Dates and date-times
  - NA is compatible with everything