Statistics with Recitation: TA Session

Danny Po-Hsien Kang (康柏賢)

October 28, 2025

Today's agenda

1. Logical expressions
   - Comparison Operators: `>`, `<`, `=`, `!`
   - Logical Combination: `&`, `&&`, `|`, `||`
   - `%in%`
   - Parenthesis: `()`
2. Data Wrangling
   - `library(dplyr)` v.s. `library(tidyverse)`
   - Pipeline: `%>%`
   - `filter()`
   - `select()`
   - `mutate()`
   - `arrange()`

Today's Dataset

- Please download the following dataset:
  - `babies.csv` (example)
  - `restaurants.csv` (practice)

  from the TA Session's Website (`restaurants.csv`) and the OpenIntro website.
- The dataset includes the babies born in 1960-1967 in the San Francisco East Bay area.
- Please import the following datasets into RStudio.

```
babies <- read.csv("data/babies.csv")
restaurants <- read.csv("data/restaurants.csv")
```

Comparison Operators: >, <, =, !

- `>` (Greater than): e.g. `5 > 4` (`TRUE`)
- `>=` (Greater or equal): e.g. `3 >= 4` (`FALSE`)
- `<` (Smaller than): e.g. `5 < 4` (`FALSE`)
- `<=` (Smaller or equal): e.g. `3 <= 4` (`TRUE`)
- `==` (Equal to): e.g. `4 == 4` (`TRUE`)
- `!=` (Not equal to): e.g. `"H" != "I"` (`TRUE`)

Logical Combination: &, &&, |, ||

- AND: & (element wise), && (first element only)

```
x <- c(3, 5, 8)
x > 4 & x < 7 # FALSE TRUE FALSE
x > 4 && x < 7 # Error: 'length = 3' in coercion to 'logical(1)
```

- OR: | (element wise), || (first element only)

```
x <- c(3, 5, 8)
x > 4 | x < 7 # TRUE TRUE TRUE
x > 4 || x < 7 # Error: 'length = 3' in coercion to 'logical(1)
```

Logical Combination: &, &&, |, ||

- How to choose between & and &&? Use && if:
  - You only want to compare the first element in the vector.
  - Inside an if statement. e.g. if (x && y).

```
if (TRUE || FALSE){
  print("TRUE!")
}
```

- & and | are useful as filters:

```
df <- data.frame(age = c(16, 22, 30, 40),
                 score = c(80, 90, 55, 60))
df[df$age > 18 & df$score > 70, ]

#Output
  age score
2  22    90
```

Check if object inside a vector or list: %in%

- **Syntax:**

```
x %in% y # x and y can be data, vec or list
```

- **Output:**

```
fruit <- c("apple", "banana", "pear")
basket <- c("banana", "orange", "pear", "kiwi")
fruit %in% basket # FALSE TRUE TRUE

3 %in% list(1, 2, 3, 4, 5) # TRUE
3 %in% list(1:5, 6:10) # FALSE
1:5 %in% list(1:5, 6:10) # FALSE FALSE FALSE FALSE FALSE
list(1:5) %in% list(1:5, 6:10) # TRUE
```

Use () for clearer expressions

- When the logical expression is long, it is recommended to add () at proper places for readability.
-
  ```
  x <- 7

  # comparisons (>, <, ==) are evaluated before &
  # & is evaluated before |
  x > 4 & x < 10 | x == 2 & x > 8 # TRUE
  # this is equivalent to
  (x > 4 & x < 10) | (x == 2 & x > 8) # TRUE

  # however, if you meant to calculate from left to right
  (((x > 4 & x < 10) | x == 2) & x > 8) # FALSE
  ```

library(dplyr) v.s. library(tidyverse)

- **dplyr** : grammar of data manipulation in R
  - Core functions: filter(), select(), mutate(), arrange(), summarise(), group_by().
  - Designed for readable, pipeable workflows (%>%).
- **tidyverse**: a collection of packages that work together
  - Includes: dplyr, ggplot2, tidyr, readr, etc.
  - We used to load tidyverse package because usually we need ggplot2 and dplyr packages.
    - You can change it to library(ggplot2), library(dplyr) if you want.

Chain Your Operation with Pipeline: %>%

- The pipeline operator %>% is used to chain multiple operations on data together. For example,

```
num <- -3:3
num %>%
  cumsum() %>%
  print()
```

- **Output:**

```
[1] -3 -5 -6 -6 -5 -3  0
```

Dataset: babies.csv

- case: id number
- bwt: birthweight
- age: mother's age in years
- height (weight): mother's height (weight) in inches

case	bwt	gestation	parity	age	height	weight	smoke
1	120	284	0	27	62	100	0
2	113	282	0	33	64	135	0
3	128	279	0	28	64	115	1
4	123	NA	0	36	69	190	0
5	108	282	0	23	67	125	1

Filter Sample in Dataset: `filter()`

- Suppose we want to filter the mothers with height 62-64. (only first 5 rows are printed)

case	bwt	gestation	parity	age	height	weight	smoke
1	120	284	0	27	62	100	0
2	113	282	0	33	64	135	0
3	128	279	0	28	64	115	1
6	136	286	0	25	62	93	0
7	138	244	0	33	62	178	0

Filter Sample in Dataset: select()

- **Syntax:**

```
data %>%
  filter(condition)
```

- **Example:**

```
babies_height_62_64 <- babies %>%
  filter(height %in% 62:64)
```

- You can use multiple conditions inside filter:
- **Example:**

```
babies_bwt_gt120_smoke <- babies %>%
  filter(bwt > 120 & smoke == 1)
```

Select Variables in Dataset: `select()`

- Suppose we only want to keep columns `bwt` and age.

bwt	age
120	27
113	33
128	28
123	36
108	23

Select Variables in Dataset: select()

- **Syntax:**

```r
data %>%
  select(cols)
```

- **Example:**

```r
babies_age_bwt <- babies %>%
  select(bwt, age)
```

- You may also select columns by deletion.
- **Example:**

```r
babies_wo_case_bwt <- babies %>%
  select(-c(case, bwt)) # delete column case and bwt
```

Create New Columns by Existing Vars: mutate()

- Suppose we want to add a column ht_plus_wt that is the sum of height and weight.

case	bwt	gestation	parity	age	height	weight	smoke	ht_plus_wt
1	120	284	0	27	62	100	0	162
2	113	282	0	33	64	135	0	199
3	128	279	0	28	64	115	1	179
4	123	NA	0	36	69	190	0	259
5	108	282	0	23	67	125	1	192

Create New Columns by Existing Vars: mutate()

- **Syntax:**

```
data %>%
  mutate(new_var = expr)
```

- **Example:**

```
babies_ht_plus_wt <- babies %>%
  mutate(ht_plus_wt = height + weight)
```

Create New Columns by Existing Vars: `mutate()`

- `mutate()` is very powerful on condition grouping!
  - Extremely useful in practice.
- If the grouping condition is binary, use together with `ifelse()`.
- **Example:**

```r
babies_age_gt30 <- babies %>%
  mutate(age_30 = ifelse(age > 30, "age > 30", "age <= 30"))
```

case	bwt	gestation	parity	age	height	weight	smoke	age_30
1	120	284	0	27	62	100	0	age <= 30
2	113	282	0	33	64	135	0	age > 30
3	128	279	0	28	64	115	1	age <= 30
4	123	NA	0	36	69	190	0	age > 30
5	108	282	0	23	67	125	1	age <= 30

Create New Columns by Existing Vars: mutate()

- If groups $>= 3$, use together with case_when().
- **Example:**

```
babies_ht_group <- babies %>%
  mutate(ht_group = case_when(
    is.na(height) ~ NA,
    height <= 63 ~ "ht <= 63",
    height > 63 & height < 67 ~ "63 < ht < 67",
    TRUE ~ "ht >= 67"))
```

case	bwt	gestation	parity	age	height	weight	smoke	ht_group
1	120	284	0	27	62	100	0	ht <= 63
2	113	282	0	33	64	135	0	63 < ht < 67
3	128	279	0	28	64	115	1	63 < ht < 67
4	123	NA	0	36	69	190	0	ht >= 67
5	108	282	0	23	67	125	1	ht >= 67

Sort the Datasets: arrange()

- Suppose we want the rows to be in descending order according to the mother's height.

case	bwt	gestation	parity	age	height	weight	smoke
176	122	278	0	31	72	155	1
86	129	274	0	29	71	NA	1
148	160	300	0	29	71	175	1
288	148	279	0	27	71	189	0
294	121	276	1	23	71	152	1

Sort the Datasets: `arrange()`

- **Syntax:**

```
data %>%
  arrange(desc(var))
```

- **Example:**

```
babies_ht_desc <- babies %>%
  arrange(desc(height))
```

- By default, `arrange()` uses ascending order:
- **Example:**

```
babies_ht_asc <- babies %>%
  arrange(height)
```

Importance of using %>%

- Use pipeline wisely to make your code more readable and maintainable.
- **Comparison:**

```
# Without %>%
arrange(select(filter(babies, height >= 60),
               bwt,
               height),
        desc(bwt)
        )

# With %>%
babies %>%
  filter(height >= 60) %>%
  select(bwt, height) %>%
  arrange(desc(bwt))
```