

# Uncertainty - Tidyverse lecture 2

## For loop and functionals

Xiaolong Yang

Introduction to Quantitative Social Science

University of Tokyo

July 5, 2022

# Today's Game Plan

## ① reducing duplication: **iteration**

- writing a for loop
- for loop v.s. functionals

**i** Today's in-class assignment: **file-drawer**

# Section 1

Iteration: for loop

# Recap: why to avoid copy-paste?

## Motivations?

```
mean(FLVoters$county)
mean(FLVoters$age)
mean(FLVoters$VTD)
median(FLVoters$county)
median(FLVoters$age)
median(FLVoters$VTD)
sd(FLVoters$county)
sd(FLVoters$age)
sd(FLVoters$VTD)
```

- ① eyes on **difference** not similarity

- ① eyes on **difference** not similarity
- ② easier to **respond to changes** in requirements

- ① eyes on **difference** not similarity
- ② easier to **respond to changes** in requirements
- ③ **fewer bugs**

# Landscape of Duplication Reduction in R

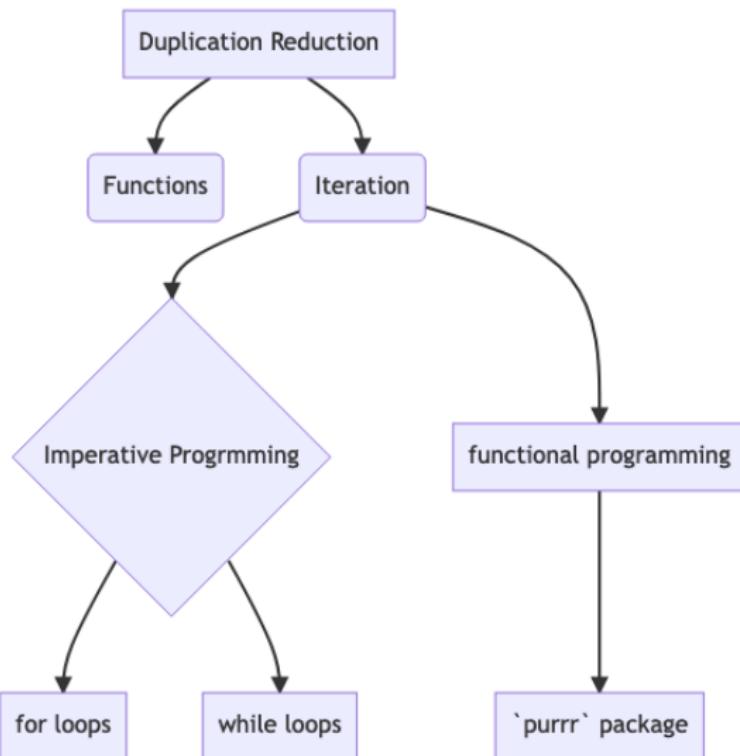


Figure 1: Duplication reduction in R

## For loops: a motivating example

**Compute the mean of each numeric variable in FLVoters**

```
mean(FLVoters$VTD)
```

```
[1] 234.725
```

```
mean(FLVoters$age)
```

```
[1] 52.42522
```

```
mean(FLVoters$county)
```

```
[1] 70.41479
```

## Compute the mean of each numeric variable in FLVoters

```
FLVoters <- FLVoters %>%
  select(where(is.numeric)) # keep numeric

output <- rep(NA, length(FLVoters))          # 1. output
for (i in seq_along(FLVoters)) {              # 2. sequence
  output[[i]] <- mean(FLVoters[[i]])          # 3. body
}
output
```

```
[1] 70.41479 234.72501 52.42522
```

## 3 components for every for loop: 1

- ➊ The **output**: create an empty vector as container beforehand

- `rep(NA, length)`
- `vector(type, length)`

```
output <- rep(NA, length(FLVoters)) # 1. output
```

```
# alternatively
```

```
output <- vector("double", length(FLVoters))
```

## 3 components for every for loop: 2

### ② The **sequence**: what to loop over

- each run `i` is assigned a different value from `seq_along(df)`
- `i` as `it`

```
seq_along(FLVoters)
```

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

```
for (i in seq_along(FLVoters)) {} # 2. sequence
```

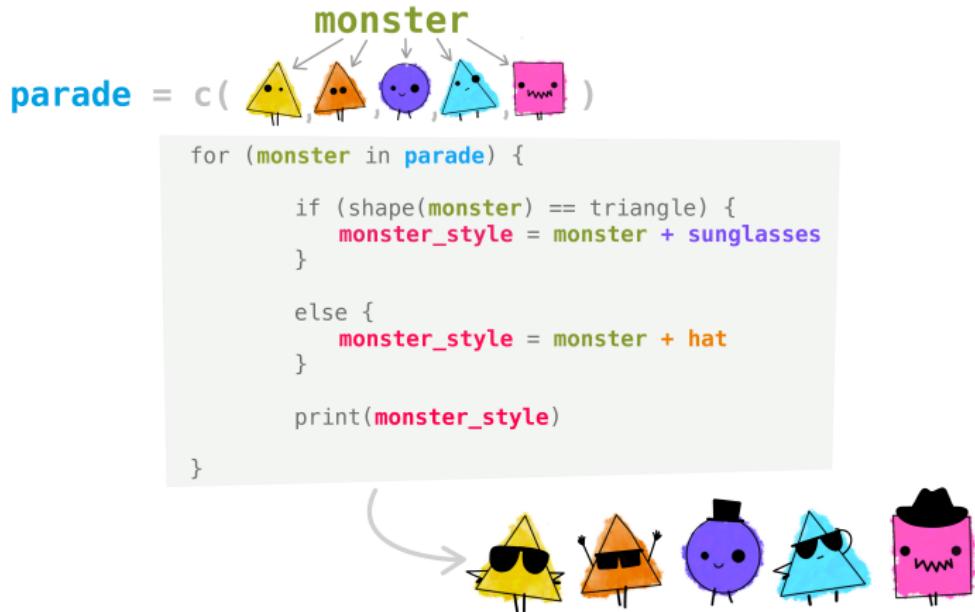
## 3 components for every for loop: 3

- ③ The **body**: code that run repeatedly, each time with a different i

```
for (i in seq_along(FLVoters)) {  
  output[[i]] <- mean(FLVoters[[i]])          # 3. body  
}
```

- `output[[1]] <- mean(FLVoters[[1]])`
- `output[[2]] <- mean(FLVoters[[2]])`
- ...

 Subsetting R objects: `[]`, `[[]]`, `$` ([link](#))



@PadavanWork

Figure 2: Monster for loop

# For loops v.s. functionals

- R as functional programming language: one can **wrap for loops in a function**, and call that function instead



Figure 3: For loops in R

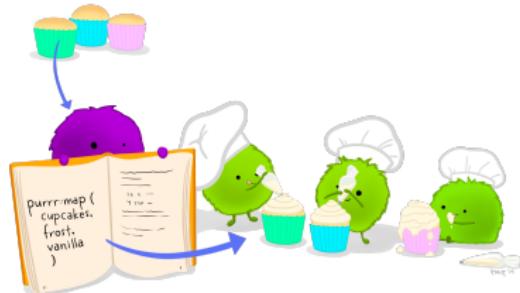


Figure 4: Functionals in R

# For loops v.s. functionals

Consider a simple motivating example

- **compute the mean of every numeric variable (every column)**

```
output <- rep(NA, length(FLVoters))          # 1. output
for (i in seq_along(FLVoters)) {              # 2. sequence
  output[[i]] <- mean(FLVoters[[i]])           # 3. body
}
output
```

# For loops v.s. functionals

Since we do this so frequently, how about **extracting it into a function?**

```
col_mean <- function(df) {  
  output <- rep(NA, length(df))  
  for (i in seq_along(df)) {  
    output[[i]] <- mean(df[[i]])  
  }  
  output  
}  
  
col_mean(FLVoters)
```

```
[1] 70.41479 234.72501 52.42522
```

# For loops v.s. functionals

But wait, surely **median**, **standard deviation** matter too!

## For loops v.s. functionals

```
col_median <- function(df) {  
  output <- rep(NA, length(df))  
  for (i in seq_along(df)) {  
    output[[i]] <- median(df[[i]])  
  }  
  output  
}
```

```
col_sd <- function(df) {  
  output <- rep(NA, length(df))  
  for (i in seq_along(df)) {  
    output[[i]] <- sd(df[[i]])  
  }  
  output  
}
```

Side effects: hard to see the difference; bug → **generalize**

# Generalizing for loops by writing functions

Add an argument to supply different functions to each column

- create a `col_summary()` with `df` and `fn` arguments

```
col_summary <- function(df, fn) {  
  out <- rep(NA, length(df))  
  for (i in seq_along(df)) {  
    out[[i]] <- fn(df[[i]])  
  }  
  out  
}
```

```
col_summary(FLVoters, median)
```

```
[1] 86 121 53
```

```
col_summary(FLVoters, sd)
```

```
[1] 37.19531 281.61961 18.48903
```

# Summary

## What we learnt

- iteration
  - for loop
  - generalize for loop with functionals (purrr)



# Final Game Plan

- iteration: **for loop variations**
- tidyverse wrap-up
- new functions in **chapter 7: Uncertainty (7.3)**

# Reference

- Quantitative Social Science: An Introduction in tidyverse
- R for Data Science
- Advanced R
- Artwork by @allison\_horst