R for Statistics Research

Young Geun Kim[1]

Department of Statistics

25 Oct, 2021 (updated: 23 Oct, 2021)

[1] ygeunkim.github.io

Introductory R Functions

Class - Vector

```
(x <- c(2.1, 4.2, 3.3, 5.4))
#> [1] 2.1 4.2 3.3 5.4
class(x)
#> [1] "numeric"
```

- Vectorized code gives the fast result
- `rowSums()`, `colSums()`, `rowMeans()`, `colMeans()` are faster than `apply()` (Wickham, 2019) 🧐

Class - Data frame

```
(df <- data.frame(x = 1:2, y = 2:1, z = letters[1:2]))
#>   x y z
#> 1 1 2 a
#> 2 2 1 b
class(df)
#> [1] "data.frame"
```

- Data for data analysis might be data frame
- Indexing in data frame connects to data transformation

```
y <- MASS::Boston[, c("medv", "lstat", "age")]
head(y)
#>   medv lstat  age
#> 1 24.0  4.98 65.2
#> 2 21.6  9.14 78.9
#> 3 34.7  4.03 61.1
#> 4 33.4  2.94 45.8
#> 5 36.2  5.33 54.2
#> 6 28.7  5.21 58.7
```

Data Analysis using data frame

Multiple linear regression for `medv` ~ `lstat` + age:

```
lm(medv ~ ., data = y)
#>
#> Call:
#> lm(formula = medv ~ ., data = y)
#>
#> Coefficients:
#> (Intercept)        lstat          age
#>     33.2228      -1.0321       0.0345
```

- Data frame is proper to use with many `R` model functions.

Class - List

Contains any object in each element:

```
(z <- list(a = x, b = df))
#> $a
#> [1] 2.1 4.2 3.3 5.4
#>
#> $b
#>   x y z
#> 1 1 2 a
#> 2 2 1 b
class(z)
#> [1] "list"
```

Other Classes

- `matrix` and `array`
  - 2d matrix: linear algebra
  - more than 2d: used in deep learning ("tensor")
- `factor`
- Date: `POSIXct`, `POSIXt`, etc
- Time series: `ts`

```
ts(1:10, frequency = 4, start = c(1959, 2))
#>      Qtr1 Qtr2 Qtr3 Qtr4
#> 1959         1    2    3
#> 1960    4    5    6    7
#> 1961    8    9   10
```

Time Series Model

```
lh
#> Time Series:
#> Start = 1
#> End = 48
#> Frequency = 1
#>  [1] 2.4 2.4 2.4 2.2 2.1 1.5 2.3 2.3 2.5 2.0 1.9 1.7 2.2 1.8 3.2 3.2 2.7 2.2 2.2
#> [20] 1.9 1.9 1.8 2.7 3.0 2.3 2.0 2.0 2.9 2.9 2.7 2.7 2.3 2.6 2.4 1.8 1.7 1.5 1.4
#> [39] 2.1 3.3 3.5 3.5 3.1 2.6 2.1 3.4 3.0 2.9
class(lh)
#> [1] "ts"
```

Fit AR(1) using `arima` function:

```
arima(lh, order = c(1,0,0))
#>
#> Call:
#> arima(x = lh, order = c(1, 0, 0))
#>
#> Coefficients:
#>          ar1  intercept
#>       0.574      2.413
#> s.e.  0.116      0.147
#>
#> sigma^2 estimated as 0.197:  log likelihood = -29.4,  aic = 64.8
```

Tidy Data

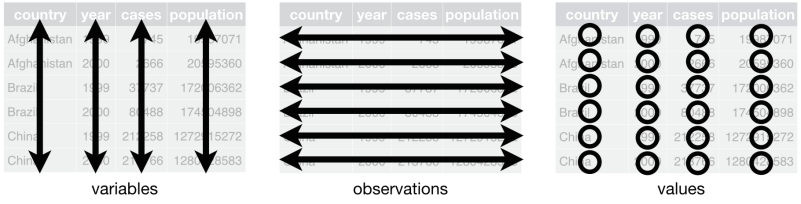


Figure 1: Tidy Data

- For easier data analysis, Wickham (2014) suggests tidy representation of dataset, called **tidy data**.
  1. Each variable must have its own column.
  2. Each observation must have its own row.
  3. Each value must have its own cell.
- `tibble` package

`read.csv` and `readr::read_csv`

- `read.csv(file)`: default function to import csv file
- `readr::read_csv(file)`: read csv file to `tibble`
  - Use this function 😎

Data Wrangling

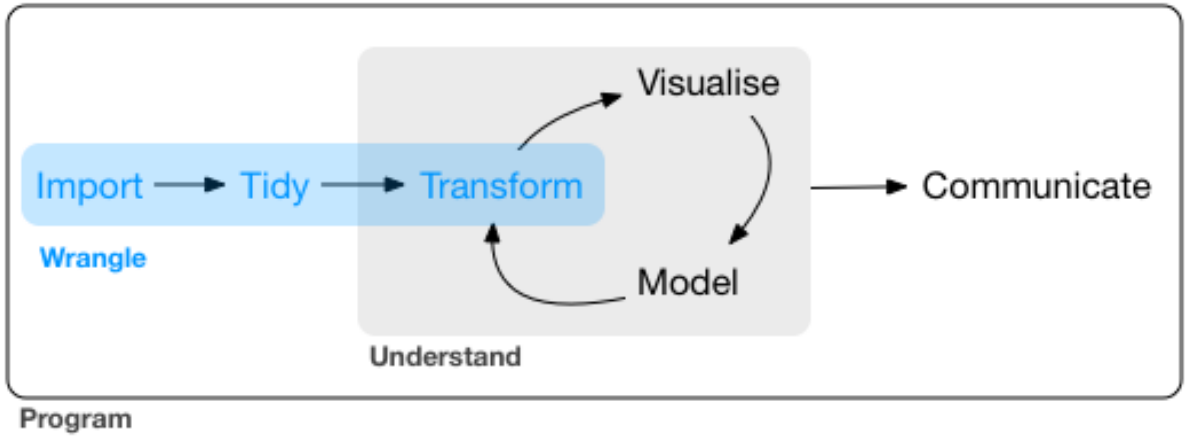


Figure 2: Steps of Data Analysis

- Since many datasets are not tidy, we need data wrangling step
- `dplyr` provides data manipulation functions
- `tidyr` package does this

Tidy Data

```
tidyr::table1
#> # A tibble: 6 x 4
#>    country      year  cases population
#>    <chr>       <int>  <int>      <int>
#> 1 Afghanistan  1999    745   19987071
#> 2 Afghanistan  2000   2666   20595360
#> 3 Brazil       1999  37737  172006362
#> 4 Brazil       2000  80488  174504898
#> 5 China        1999 212258 1272915272
#> 6 China        2000 213766 1280428583
```

dplyr

- `filter()` picks observations using logical conditions
- `arrange()` reorders the rows
- `select()` picks variables by column names
- `mutates()` creates new variables with functions
- `summarise()` collapses many values to a single value, i.e. summary statistic

Example

group_by(): excecute function for each group:

```
tidyr::table1 %>%
  group_by(country) %>%
  summarise(
    average = mean(population),
    std = sd(population)
  )
#> # A tibble: 3 x 3
#>   country          average       std
#>   <chr>              <dbl>     <dbl>
#> 1 Afghanistan    20291216.   430125.
#> 2 Brazil        173255630   1766732.
#> 3 China        1276671928.  5312713.
```

Non-tidy Data 1

```
tidyr::table4a
#> # A tibble: 3 x 3
#>   country     `1999` `2000`
#> * <chr>        <int>  <int>
#> 1 Afghanistan    745   2666
#> 2 Brazil       37737  80488
#> 3 China       212258 213766
```

- values (year) are in variable (1999, 2000)
- How to tidy? Gather these two variables into one variable.

Non-tidy Data 2

```
tidyr::table2
#> # A tibble: 12 x 4
#>    country      year type            count
#>    <chr>       <int> <chr>           <int>
#>  1 Afghanistan  1999 cases             745
#>  2 Afghanistan  1999 population    19987071
#>  3 Afghanistan  2000 cases            2666
#>  4 Afghanistan  2000 population    20595360
#>  5 Brazil       1999 cases           37737
#>  6 Brazil       1999 population   172006362
#>  7 Brazil       2000 cases           80488
#>  8 Brazil       2000 population   174504898
#>  9 China        1999 cases          212258
#> 10 China        1999 population  1272915272
#> 11 China        2000 cases          213766
#> 12 China        2000 population  1280428583
```

- Looking at `type`, each observation does not have its own row.
- Then spread `type` taking values of `count`

tidyr

`tidyr` package can tidy data-set. From Wickham and Grolemund (2017),

- `pivot_longer()`: Figure 3
- `pivot_wider()`: Figure 4

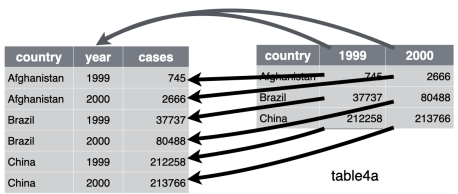


Figure 3: Gather

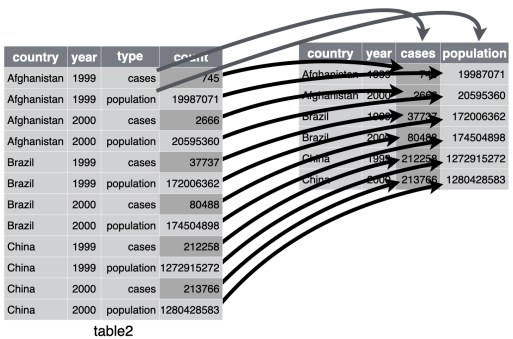


Figure 4: Spread

Large Data

- When data file is too large
- `data.table` package focuses on memory optimization
- `read.csv` or `read_csv` can mistakenly read string as factor
- but `data.table::fread()` do not

Visualization

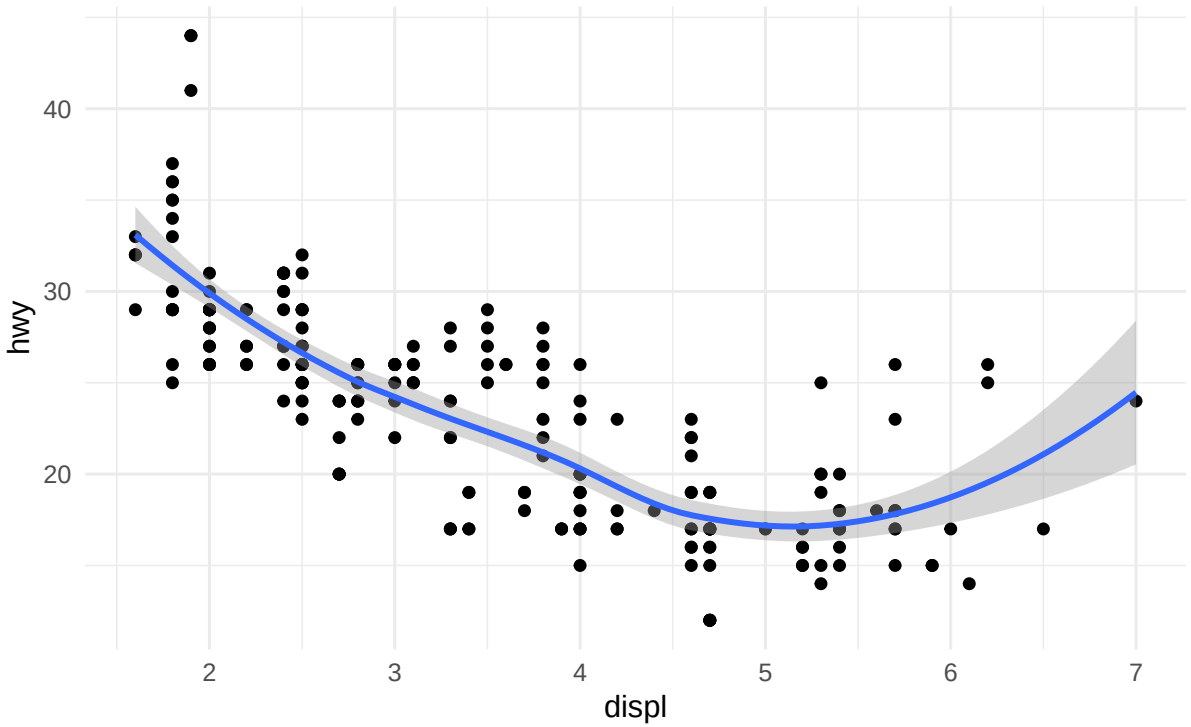
ggplot2	base plot
grammar of graphics	pen on paper model

ggplot2

- Data: data frame (`mpg`)
- Aesthetic mapping: `aes()`
  - x-axis: `displ`
  - y-axis: `hwy`
- Layers: `geom_*()` function
  - scatter plot: `geom_point()`
  - smoothing: `geom_smooth()`

Example

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  geom_smooth() +
  theme_minimal()
#> `geom_smooth()` using method = 'loess' and formula 'y ~
```



🤩 Simulation 🤩

- Monte Carlo simulation (parametric bootstrap)
- Bootstrap

Monte Carlo Simulation

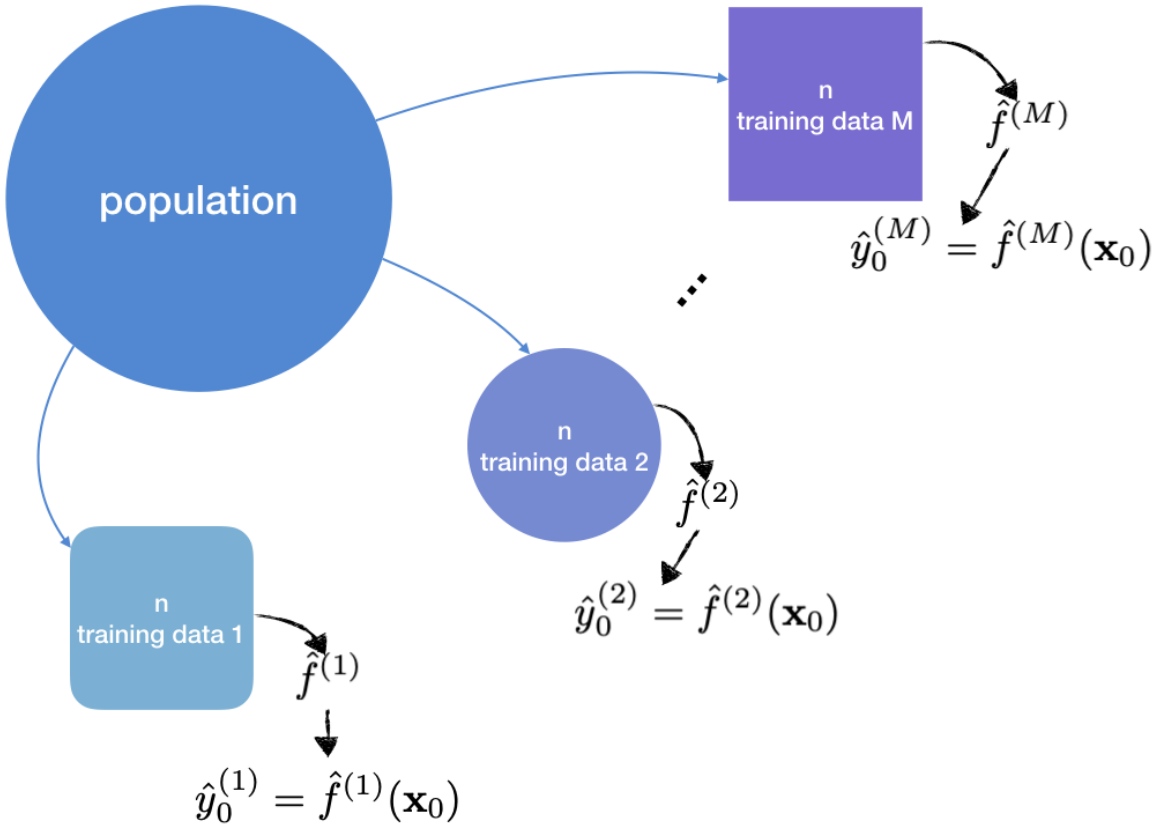


Figure 6: Simulating Our Model

Monte Carlo Simulation

Example (Any quantity of interest)

Suppose that $X_1, X_2 \overset{\text{iid}}{\sim} N(0,1)$. Estimate $\theta = \mathrm{E}|X_1 - X_2|$.

Input : distribution f, Number of iteration M

1 **for** $m \leftarrow 1$ **to** M **do**

2 | Generate $(X_1^{(m)}, X_2^{(m)}) \overset{\text{iid}}{\sim} N(0,1)$;

3 | Compute $\hat{\theta}^{(m)} = |X_1^{(m)} - X_2^{(m)}|$;

4 **end**

5 Draw a histogram;

Output: $\{\hat{\theta}^{(1)}, \ldots, \hat{\theta}^{(M)}\}$

Algorithm 1: Empirical distribution of $\hat{\theta}$

MC Estimates

- Recall if you took statistical computing (STA5011) course 😄
- MC estimator: $\bar{\hat{\theta}} = \frac{1}{M} \sum_{m=1}^{M} \hat{\theta}_m^{(m)}$
- MC standard error: $\widehat{SE}(\hat{\theta}) = \sqrt{\frac{1}{M-1} \sum_{m=1}^{M} (\hat{\theta}^{(m)} - \bar{\hat{\theta}})}$
- MSE
- CI

R programming for Example 1

- Generation can be done by matrix
- Compuation can be vectorized

```
# MC setting--------------------------
num_mc <- 2
num_iter <- 1000
set.seed(1)
# X1, X2 ~ N(0, 1)--------------------
data_mc <-
  rnorm(num_mc * num_iter) %>%
  matrix(ncol = 2, byrow = TRUE)
# theta = abs(data[,1] - data[,2])---
emp_theta <- abs(data_mc[,1] - data_mc[,2])
```

Empirical distribution

```
qplot(emp_theta, geom = "histogram", bins = 30) +
  theme_minimal() +
  xlab(expression(theta))
```

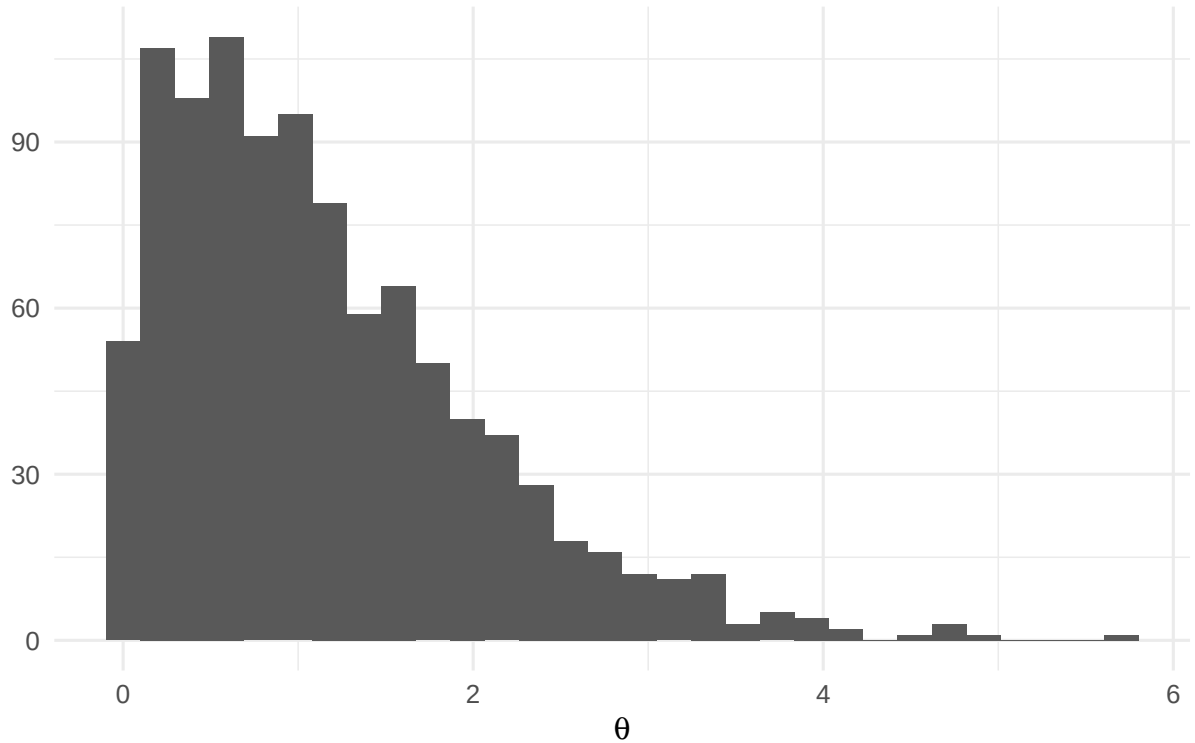


Figure 7: Empirical distribution of $\hat{\theta}$ for $|X_1 - X_2|$

Estimates

Estimator

```
mean(emp_theta)
#> [1] 1.16
```

SE:

```
sd(emp_theta)
#> [1] 0.899
```

Bootstrap

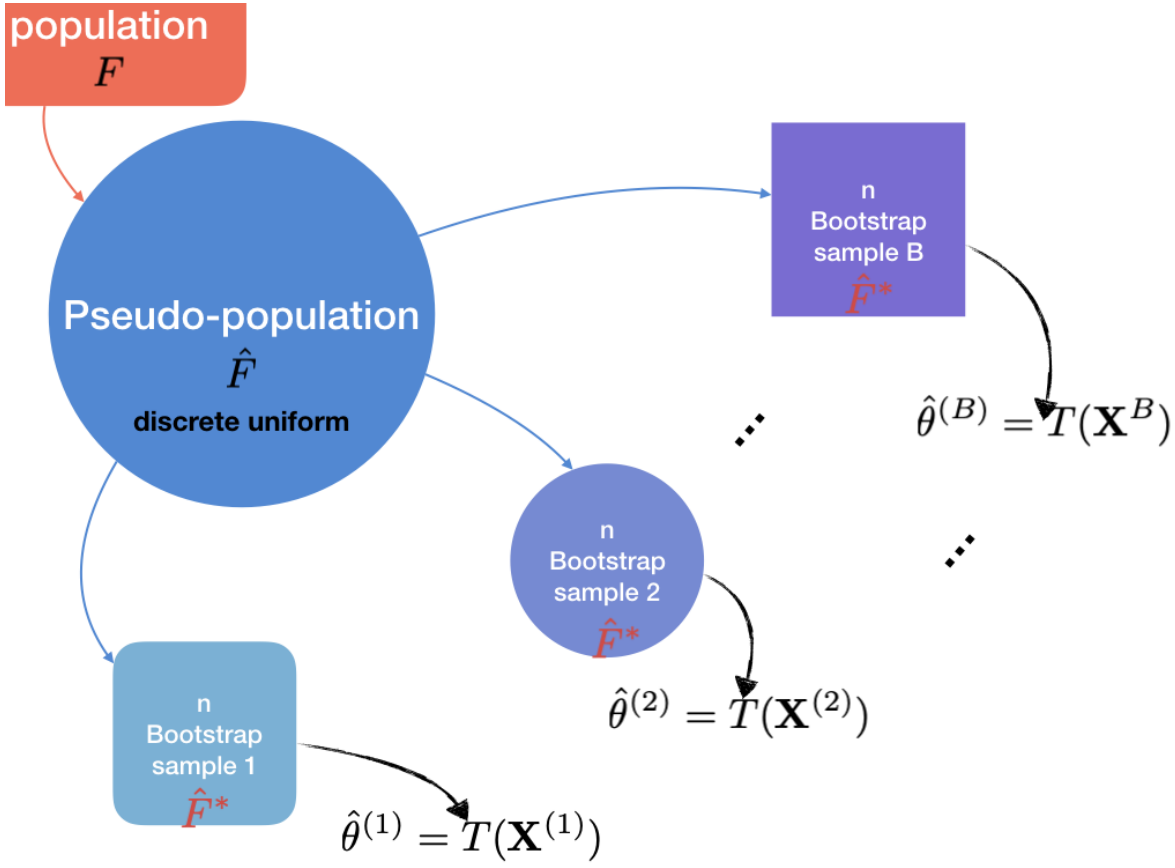


Figure 8: Empirical Distribution of Bootstrap

Bootstrap Algorithm

Data: n observations $x_1, \ldots, x_n$

input : statistic of interest $\hat{\theta}$, the number of bootstrap replicates B

1 **for** $b \leftarrow 1$ **to** B **do**

2 Sampling with replacement $X_1^{(b)}, \ldots, X_n^{(b)}$ from the observed sample;

3 Compute estimate

$$\hat{\theta}(X_1^{(b)}, \ldots, X_n^{(b)}) \equiv \hat{\theta}_b^*$$

;

4 **end**

output: Bootstrap replications $\{\hat{\theta}_1^*, \ldots, \hat{\theta}_B^*\}$

Algorithm 2: Bootstrap for $\hat{\theta}$

Bootstrap Estimates

Bootstrap standard error:

$$\hat{\sigma}_B = \left[\frac{1}{B-1} \sum_{b=1}^{B} (\hat{\theta}_b^* - \overline{\hat{\theta}^*})^2 \right]^{\frac{1}{2}}$$

where

$$\overline{\hat{\theta}^*} = \frac{1}{B} \sum_{b=1}^{B} \hat{\theta}_b^*$$

In R

```
resample <- function(x) {
  n <- length(x)
  # sampling with replacement---------------
  x[sample(1:n, size = n, replace = TRUE)] %>%
    # estimator-----------------------------
    mean()
}
# example data--------------------------------
unif_sample <- runif(50, max = 50)
```

for loop

```
resample_for <- function(x, B = 1000) {
  res <- numeric(B)
  for (b in 1:B) {
    res[b] <- resample(x = x)
  }
  res
}
```

Result

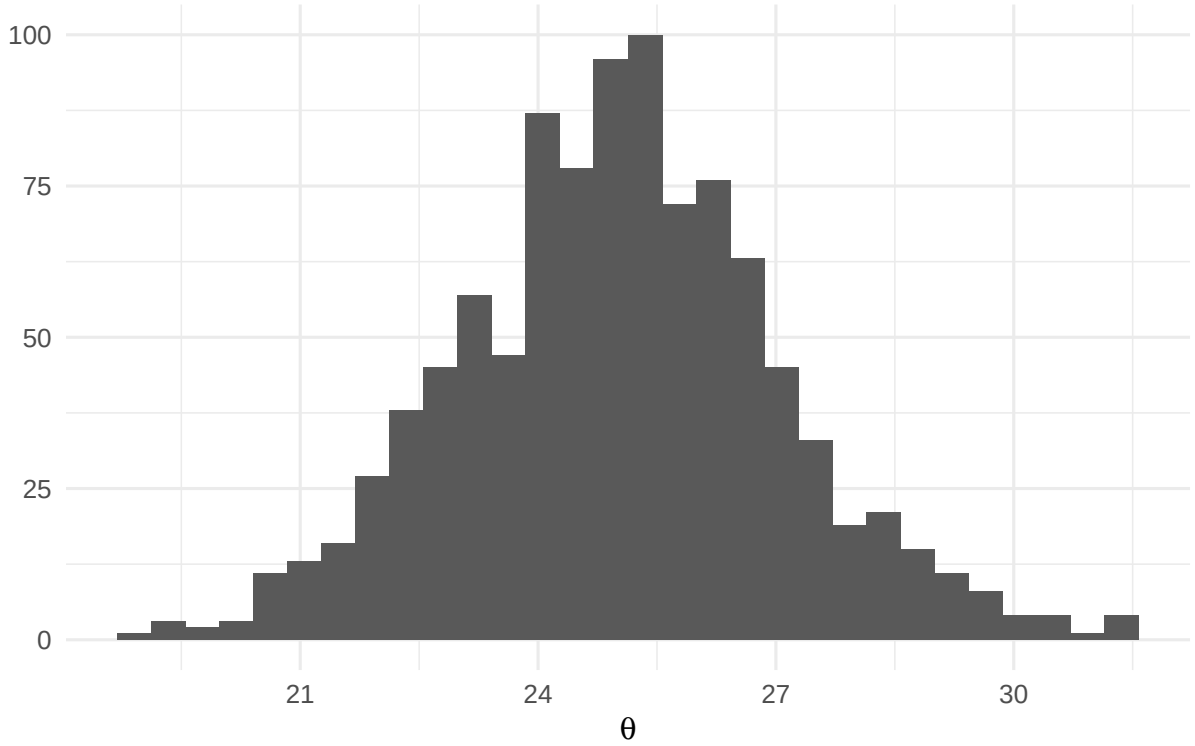


Figure 9: Bootstrap Replicates of Sample Mean

Reproducible Documents

- Data can be changed while writing the document
- R Markdown helps reproducibility by integrating Markdown and R.
- Bookdown is a package for authoring books, but it also provides a function for single document: `bookdown::*_document2`
- See Xie et al. (2018) and Xie (2016).

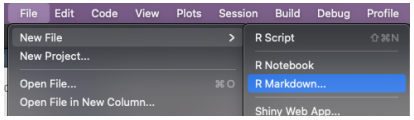


Figure 10: Making R Markdown

Figure 11: rmd Default

Style Guides

- See the `tidyverse style guide`[2]
- NOT google's

[2] https://style.tidyverse.org

Project-Oriented Workflow

- https://www.tidyverse.org/blog/2017/12/workflow-vs-script/
- Directory: the project folder
  - useful when sharing with other people
  - `renv` package: save every package in the project using into the project folder
- https://www.r-bloggers.com/2018/08/structuring-r-projects/
  - Use the efficient structure

Parallel Computations

Sources

- Books:
  - McCallum and Weston (2012)
  - Matloff (2015)
- Blogs:
  - https://psu-psychology.github.io/r-bootcamp-2018/talks/parallel_r.html
  - https://www.blasbenito.com/post/02_parallelizing_loops_with_r/

Preliminaries

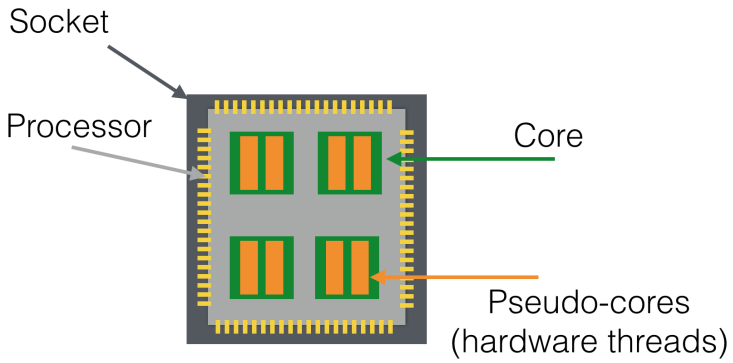


Figure 12: CPU

- Node: Single computer consisting of a motherboard and OS.
- Processor = Socket: physical unit containing
- Core: the smallest computation unit of the processor
- Multi-core processor: Physical processor in which many cores are embedded
- Hardware thread = logical core: computation unit within a core that allows the core to do multiple things at once

Parallel Computing

- Parallel excecution: divide the queue among many workers that compute at the same times
- Parallel backend
  - The way of workers group execute the iterations of the order of the director node.
  - Provides the means for the director and workers to communicate, while allocating and managing the required computing resources.
  - **PSOCK and FORK**

PSOCK

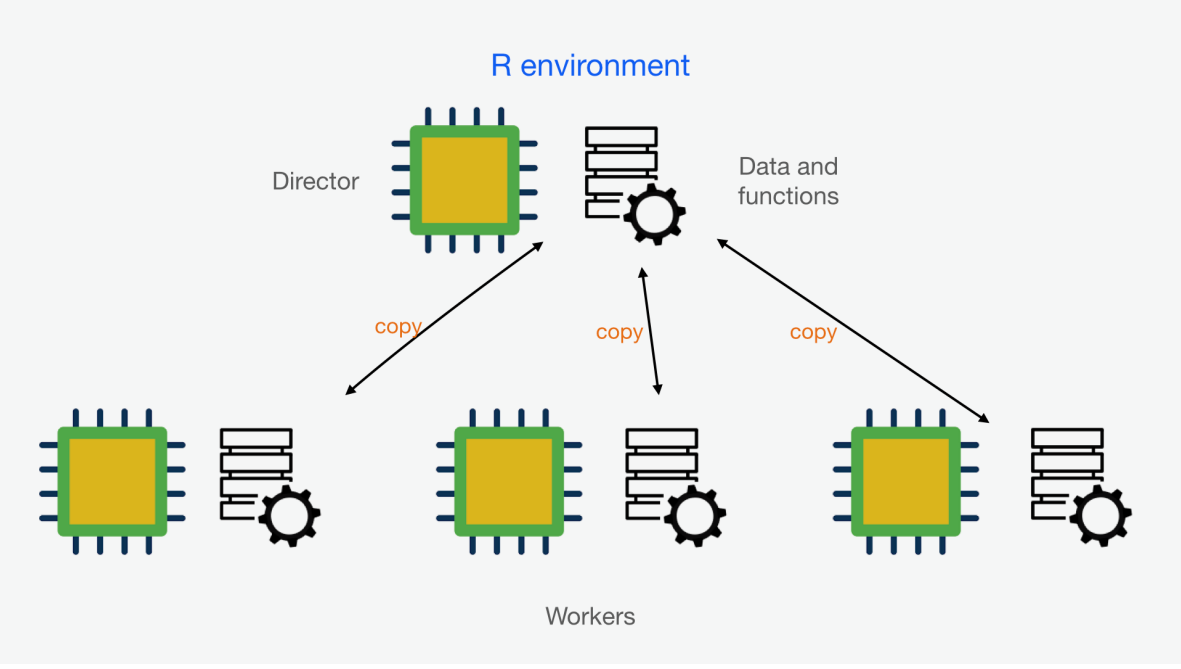


Figure 13: PSOCK backend

- Cluster computing
- launch a new environment of R on each core

FORK

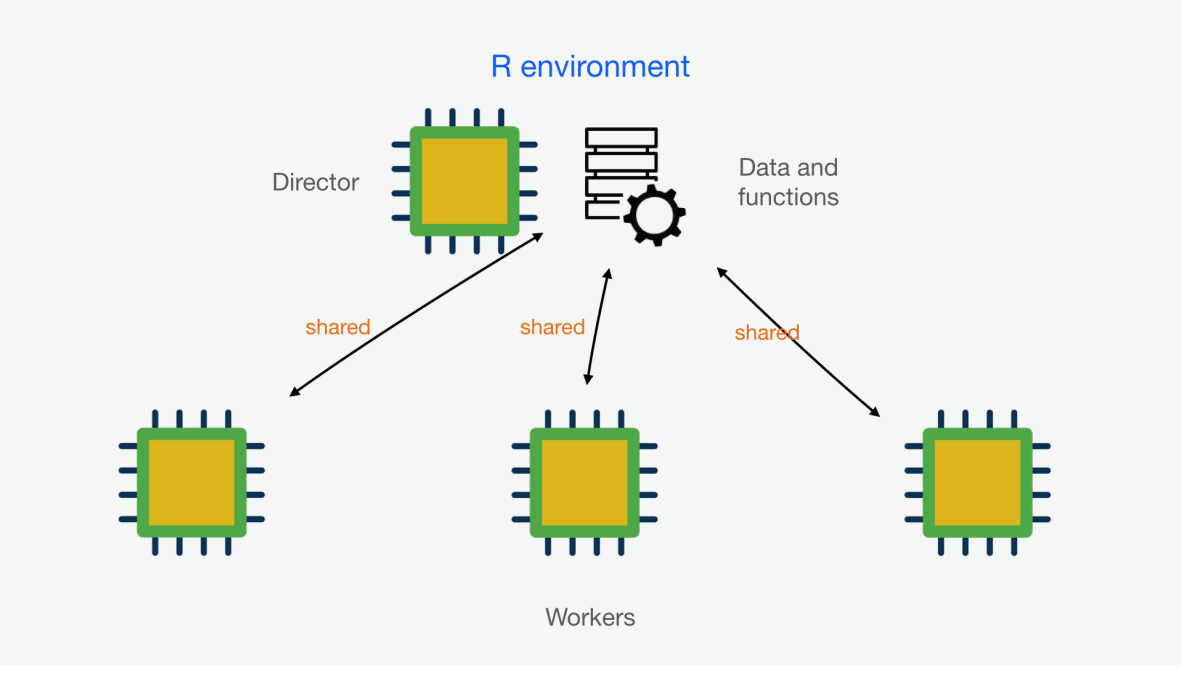


Figure 14: FORK backend

- Multi-core processor
- duplicate the entire current version of R and move it to a new core

Difference

PSOCK (Parallel Socket Cluster)	FORK
both in UNIX and windows	only in UNIX (Mac, Linux, etc)
Environment of the director needs to be copied to one of each of worker (disadvantage)	Workers share the same environment: highly efficient
Copying is tricky	Implementing forking is easy
Each process on each note is unique: cannot be cross-contaminated (advantage)	The fact that the processes are duplicates can cause issues, e.g. random number generation

~~FORK backend was much faster~~ than PSOCK.[3]

[3] https://www.r-bloggers.com/2019/06/parallel-r-socket-or-fork/

Packages

- `foreach`: Both PSOCK and FORK
- `parallel`:
  - `parLapply`, `parSapply`, and `parApply`: Both PSOCK and FORK as `foreach` does
  - `mclapply`, `pvec`: only FORK

```
library(foreach)
#>
#> Attaching package: 'foreach'
#> The following objects are masked from 'package:purrr':
#>
#>     accumulate, when
```

foreach

- loop
- %do: non-parallel
- %dopar: parallel
- .combine: combine the result
  - e.g. .combine = rbind, .combind = c
  - This makes the code useful

Non-parallel Example 1

```
# without combine------
foreach(i = 1:2) %do% {
  i
}
#> [[1]]
#> [1] 1
#>
#> [[2]]
#> [1] 2
# combine--------------
foreach(i = 1:2, .combine = c) %do% {
  i
}
#> [1] 1 2
```

Non-parallel Example 2

```
foreach(i = 1:3, .combine = bind_rows) %do% {
  tibble(
    x = i,
    y = i + 1
  )
}
#> # A tibble: 3 x 2
#>       x     y
#>   <int> <dbl>
#> 1     1     2
#> 2     2     3
#> 3     3     4
```

Backend - Cluster

```
parallel::detectCores()
#> [1] 16
n_cores <- parallel::detectCores() / 2
```

```
(my_cluster <- parallel::makeCluster(n_cores))
#> socket cluster with 8 nodes on host 'localhost'
```

Backend - Register

```
doParallel::registerDoParallel(cl = my_cluster)
foreach::getDoParRegistered()
#> [1] TRUE
foreach::getDoParWorkers()
#> [1] 8
foreach::getDoParName()
#> [1] "doParallelSNOW"
```

```
foreach(i = 1:2, .combine = c) %dopar% {
  i
}
#> [1] 1 2
```

When finished, stop the cluster:

```
parallel::stopCluster(cl = my_cluster)
```

Example 1

If you want to other packages, you should register them:

```
my_cluster <- parallel::makeCluster(n_cores)
doParallel::registerDoParallel(cl = my_cluster)
parallel::clusterEvalQ(my_cluster, library(magrittr))
#> [[1]]
#> [1] "magrittr"  "stats"     "graphics"  "grDevices" "utils"     "datasets"
#> [7] "methods"   "base"
#>
#> [[2]]
#> [1] "magrittr"  "stats"     "graphics"  "grDevices" "utils"     "datasets"
#> [7] "methods"   "base"
#>
#> [[3]]
#> [1] "magrittr"  "stats"     "graphics"  "grDevices" "utils"     "datasets"
#> [7] "methods"   "base"
#>
#> [[4]]
#> [1] "magrittr"  "stats"     "graphics"  "grDevices" "utils"     "datasets"
#> [7] "methods"   "base"
#>
#> [[5]]
#> [1] "magrittr"  "stats"     "graphics"  "grDevices" "utils"     "datasets"
#> [7] "methods"   "base"
#>
#> [[6]]
#> [1] "magrittr"  "stats"     "graphics"  "grDevices" "utils"     "datasets"
#> [7] "methods"   "base"
#>
#> [[7]]
```

Example 1 - foreach

```
set.seed(1)
theta_foreach <- foreach(b = 1:1000, .combine = c) %dopar%
  resample(unif_sample)
}
# stop cluster----------------------
parallel::stopCluster(cl = my_cluster)
```

Example 1 - result

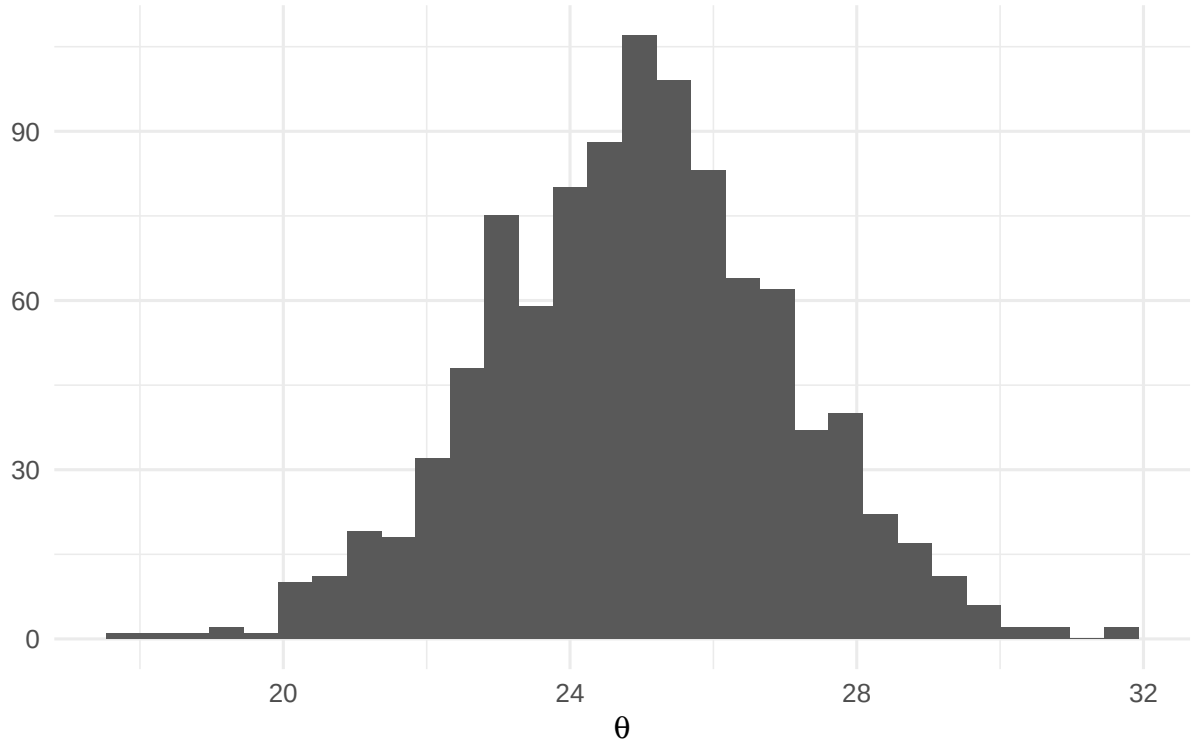


Figure 15: Bootstrap Replicates by foreach

Forking for `foreach`

- Replace `parallel::makeCluster(n_cores)` with `parallel::makeForkCluster`
- `doMC::registerDoMC`

doMC::registerDoMC

- This is quite simple
- Only one line is need
- Also, you can define function more easily.

```
resample_foreach <- function(x, B) {
  foreach(b = 1:B, .combine = c) %dopar% {
    resample(x)
  }
}
# implement--------------------------
doMC::registerDoMC(cores = n_cores)
theta_domc <- resample_foreach(unif_sample, 1000)
```

parLapply, parSapply, and parApply

- Functionals
- Parallel backend like `foreach`
- Need additional `parallel::clusterExport(cl, varlist)` for object and function

```
my_cluster <- parallel::makeCluster(n_cores)
doParallel::registerDoParallel(cl = my_cluster)
# parallel::clusterExport(my_cluster, c("unif_sample", "res
parallel::clusterEvalQ(my_cluster, library(magrittr))
#> [[1]]
#> [1] "magrittr"  "stats"     "graphics"  "grDevices" "ut
#> [7] "methods"   "base"
#>
#> [[2]]
#> [1] "magrittr"  "stats"     "graphics"  "grDevices" "ut
#> [7] "methods"   "base"
```

Example 1 - parSapply

```
set.seed(1)
theta_apply <- parallel::parSapply(
  cl = my_cluster,
  1:1000,
  function(x) {
    resample(unif_sample)
  }
)
# stop cluster----------------------
parallel::stopCluster(cl = my_cluster)
```

Example 1 - result of `parSapply`

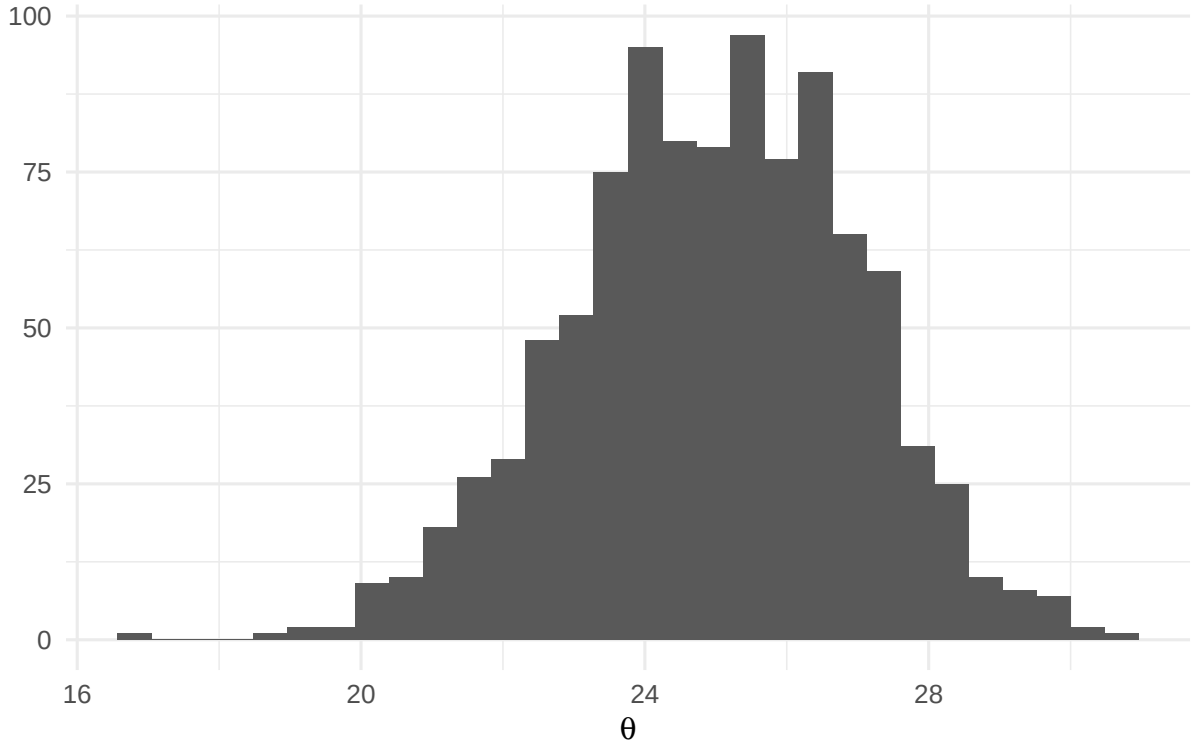


Figure 16: Bootstrap Replicates by mclapply

mclapply

- `mclapply` is FORK backend
- Easy to implement
- `lapply`-way and additional `mc.cores` option

```
set.seed(1)
theta_mcapply <- parallel::mclapply(
  1:1000,
  function(x) {
    resample(unif_sample)
  },
  mc.cores = n_cores
)
```

Example 1 - result of `mclapply`

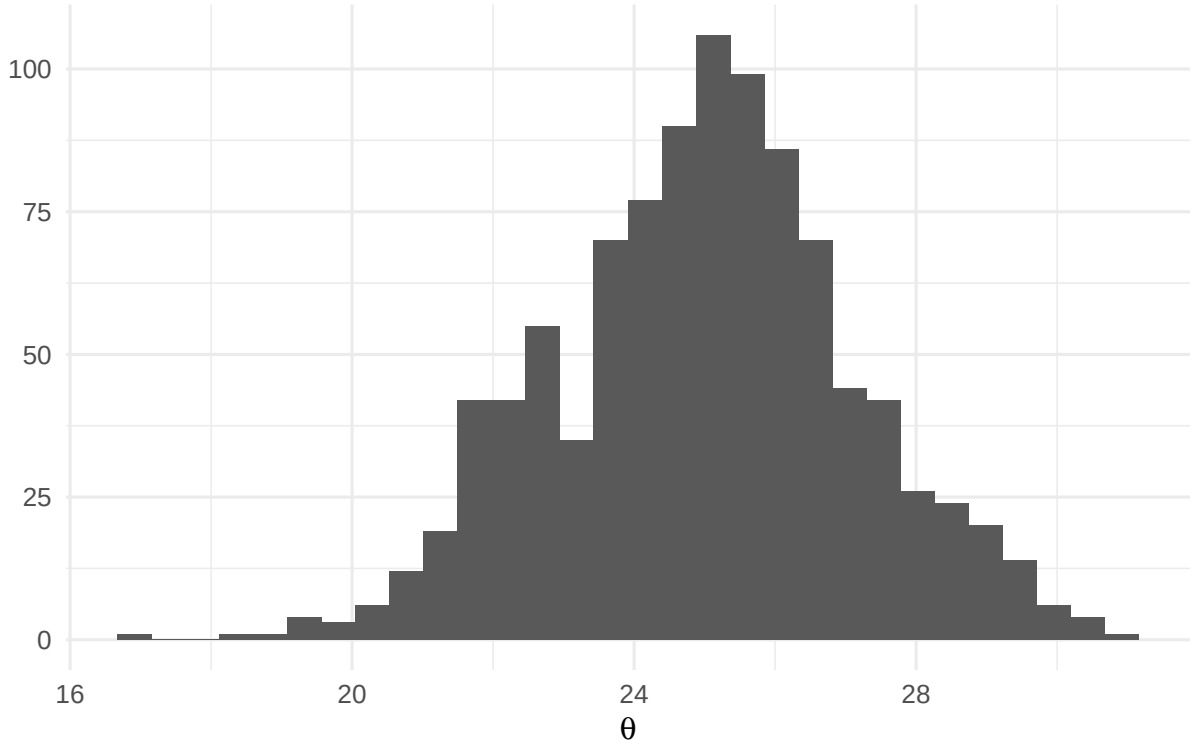


Figure 17: Bootstrap Replicates by parSapply

pvec

- Vector map based on forking backend
- same as `sapply` option

Reproducible Results

- `mclapply` result give not identical results by `set.seed()`
- It needs some methods
- See McCallum and Weston (2012) and https://psu-psychology.github.io/r-bootcamp-2018/talks/parallel_r.html

Parallel Options in Popular Funtions

- `glmnet::cv.glmnet` has `parallel = FALSE` option.
  - Use `foreach`
- `boot` has `parallel = c("no", "multicore", "snow")`

Rcpp

- Although parallelization helps performance of our codes, `Rcpp` might be basically the fastest in the `for` loop scheme
- Simple Gibbs sampler code in the official site: https://gallery.rcpp.org/articles/gibbs-sampler/

R Packages

Sources

- Wickham (2015)
- Recommend: Presentation of tidyverse team in last weak
  - https://youtu.be/EpTkT6Rkgbs

devtools

```
install.packages("devtools")
```

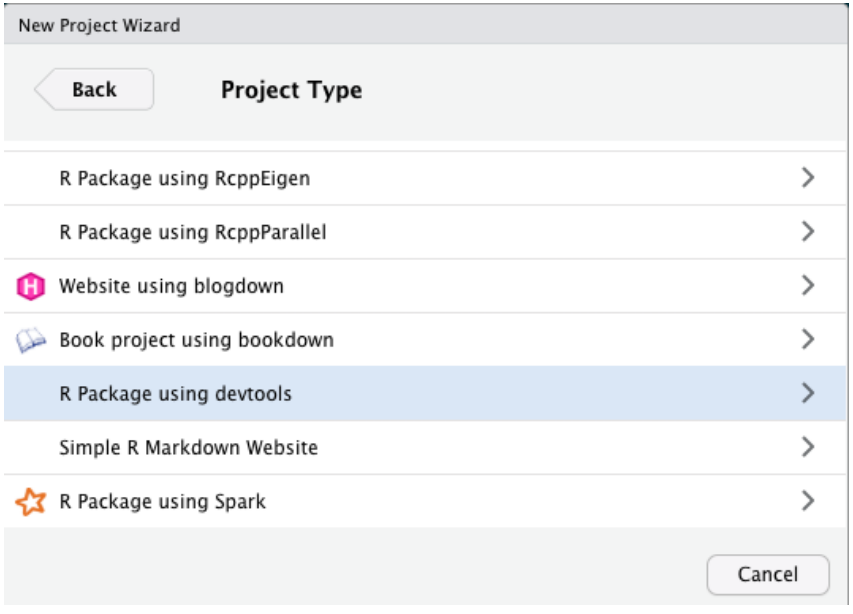


Figure 18: R Package Project

Metadata

```
Package: mypackage
Title: What The Package Does (one line, title case required)
Version: 0.1
Authors@R: person("First", "Last", email = "first.last@example.com",
                  role = c("aut", "cre"))
Description: What the package does (one paragraph)
Depends: R (>= 3.1.0)
License: What license is it under?
LazyData: true
```

`usethis`

- When write or make files, additional settings are required.
- `usethis` package do this instead of us
- e.g. `usethis::use_mit_license("Name")` add Licence file and add this file, write in the description above license, and list into `.Rbuildignore`.

Documentation

- Using `roxygen2` package, make help documents

Build

- There is `build & reload` button in RStudio
- but it is not perfect
- vignettes is not build
- If you want vignettes, use `devtools::build()`

Check

- Check button
- `devtools::check()`

Vignettes

- Long form documentation
- `usethis::use_vignette("vignette-name")`

Test

- `testthat packate`
- `usethis::use_testthat()`

Data

- Adding data in the package
- `usethis::use_data_raw()`
- `usethis::use_data()`

Additional Tips

xlsx2csv

- xlsx file: hard to deal with in R
  - readxl package
  - also has issue
- Just change the file into csv
- https://github.com/dilshod/xlsx2csv
- `xlsx2csv -s 3 -m NJClPFECA_SciHub_Data.xlsx ../processed/njcipeca.csv`

Pipe

- `marittr` package in `tidyverse` family: %>%
  - RStudio shortcut: command + shift + m or ctrl + shift + m
- R default pipe operator from 4.1: |>
  - similar to %>%
  - but it cannot be referenced by . like %>%
  - e.g. `data %>% lm(y ~ x, data = .)`

kable and kableExtra

- `knitr::kable`: make latex, html, markdown, or pandoc table in R markdown
- `kableExtra` package: build complex table based on `kable`.
  - e.g. multirow or multicol latex table

Why `tidyverse`?

- https://stackoverflow.com/questions/tagged/r
- If you ask or look at the answer in the stackoverflow, people ask the questions with `tidyverse`
- `tidyverse` provides the insufficiency of pre-processing with base data frame.

tidyverts

- `forecast`: Time series package made by Rob Hyndman
- Additionaly, `tidyverts` family was made by the team for tidy version of time series (`ts`)

Sources I

Matloff, N. (2015). *Parallel Computing for Data Science: With Examples in R, C++ and CUDA*. CRC Press.

McCallum, Q. E. and Weston, S. (2012). *Parallel R*. O'Reilly Media.

Wickham, H. (2014). Tidy data. *Journal of Statistical Software*, 59(10):1–23.

Wickham, H. (2015). *R Packages*. O'Reilly Media.

Wickham, H. (2019). *Advanced R, Second Edition*. CRC Press.

Wickham, H. and Grolemund, G. (2017). *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. O'Reilly Media, Inc., 1st edition.

Sources II

Xie, Y. (2016). *bookdown: Authoring Books and Technical Documents with R Markdown*. CRC Press.

Xie, Y., Allaire, J., and Grolemund, G. (2018). *R Markdown: The Definitive Guide*. CRC Press.