근위흉추만곡의 척추경 나사못 폭(width)의 베이즈 통계를 이용한 분석

서론

2018년의 빈도주의 통계를 이용한 분석을 베이즈 통계를 이용하여 다시 분석한다.

코드 분석

다음과 같이 세 가지 코드로 구성되었다. 각각을 알아본다.

- 레시피 코드
- Stan 코드
- 메인 코드

"file not found" error를 막기 위하여 사용되는 Rproj & here 패키지를 사용하는 셈이다. 이 패키지에서는 메인 코드와 레시피 코드의 두 가지를 사용한다.

(내 책 보고 보충할 것)

0.1 레시피 코드

레시피 코드는 데이터를 만드는 곳이다. 파일은 data.R로 다음과 같다.

```
# data.R (척추경 넓이 전용 레시피)
library(readxl)
library(tidyverse)
library(here)

# 1. 엑셀 로드
raw_df <- read_excel(here("kbpark_modify.xlsm"), skip = 1)
```

```
# data.R 2번 섹션 최종 수정
clean_df <- raw_df %>%
  select(ID, 33:66) %>%
  rename_with(~str_remove(., "\\.\\.\\..*"), -ID) %>%
  pivot_longer(cols = -ID, names_to = "key", values_to = "width") %>%
  mutate(
    Level = str_extract(key, "[TL]\\d+"),
    Side_Label = ifelse(str_detect(key, "Lt"), "Left", "Right")
  ) %>%
  filter(!is.na(width), width > 0) %>%
  # --- 여기서 T1~T6만 골라내는 '여과지'를 추가합니다 ---
  filter(Level %in% c("T1", "T2", "T3", "T4", "T5", "T6")) %>%
  # ------------------------------------------------
mutate(level_idx = as.integer(factor(Level, levels = c("T1", "T2", "T3", "T4", "T5", "T6"))))

# 3. Stan 데이터 리스트 (level_idx를 사용합니다)
stan_data <- list(
  N = nrow(clean_df),
  width = as.numeric(clean_df$width),
  vertebra_level_numeric = clean_df$level_idx,
  side_numeric = ifelse(clean_df$Side_Label == "Left", 1, 2)
)
```

코드를 하나씩 설명한다.

- `raw_df <- read_excel(here("kbpark_modify.xlsm"), skip = 1)`는 excel file을 읽어서 raw_df로 만드는 것이다. 여기서 here의 용도는 ...
-
```
clean_df <- raw_df %>%
select(ID, 33:66) %>%
rename_with(~str_remove(., "\\.\\.\\..*"), -ID) %>%
pivot_longer(cols = -ID, names_to = "key", values_to = "width") %>%
mutate(
  Level = str_extract(key, "[TL]\\d+"),
  Side_Label = ifelse(str_detect(key, "Lt"), "Left", "Right")
) %>%
filter(!is.na(width), width > 0) %>%
# --- 여기서 T1~T6만 골라내는 '여과지'를 추가합니다 ---
filter(Level %in% c("T1", "T2", "T3", "T4", "T5", "T6")) %>%
# ------------------------------------------------
```

```
mutate(level_idx = as.integer(factor(Level, levels =
            c("T1", "T2", "T3", "T4", "T5", "T6"))))
```

- ▷ `clean_df <- raw_df %>%select(ID, 33:66)`: 이 코드는 raw_df에서 열 ID, 33:66를 선택하여 clean_df라는 데이터프레임을 만들라는 것이다.
- ▷ select(ID, 33:66)에서 33열은 LtT1, 34열은 LtT2..., 50열은 RtT1,... 66열은 RtL5를 말한다. 따라서 이 코드는 좌우측 흉요추의 직경을 나타내는 열을 선택하라는 코드이다.
- ▷ `rename_with(~str_remove(., "\\.\\.\\..*"), -ID)`: 1번 흉추의 직경을 나타내는 LtT1이 R(또는 RStudio)에서는 (비슷한 이름이 여러 개 있을 때 중복을 피하기 위하여) LtT1...33으로, 2번 흉추의 직경을 나타내는 LtT2는 LtT2...34 등으로 이름이 붙여진다. 따라서 원래의 이름으로 돌려놓기 위하여 ...33 등의 번호를 제거해야 한다. 정규표현식 `\\.`는 마침표 .를 나타내고, `\\.*`는 마침표와 그 이후의 숫자를 나타낸다. 따라서 `rename_with(~str_remove(., "\\.\\.\\..*"), -ID)`는 ID 열을 제외한 나머지 열 이름에서 ...으로 시작하는 뒷부분을 모두 삭제하여 원래 이름(LtT1, LtT2 등)으로 되돌린다. str은 string의 약자이다.
- ▷ `pivot_longer(cols = -ID, names_to = "key", values_to = "width")`: 가로로 나열된 데이터(Wide format)를 세로로 긴 형태(Long format)로 변환한다. ID 열은 고정하고, LtT1 LtT9 같은 열 이름들은 key라는 새로운 열로 모으고, 그 안에 들어있던 실제 측정값(척추경 넓이)은 width라는 열로 재배치한다.
- ▷ mutate() 함수는 데이터프레임에서 새로운 열(column)을 만든다. 여기서는 Level, Side_Label의 두 열을 만드는데,
  - ∗ `str_extract(key, "[TL]\\d+")` 코드에서 [TL]은 T 혹은 L로 시작하는 글자를 찾으라는 뜻이고, `\\d+`는 그 뒤에 붙은 숫자(digit)를 모두 가져오라는 뜻이다. 그 결과 LtT1에서는 T1을, RtL5에서는 L5를 쏙 뽑아내어 Level이라는 열에 담는다.
  - ∗ `ifelse(str_detect(key, "Lt"), "Left", "Right")`는 key 열의 글자 중에 Lt라는 글자가 포함되어 있는지 찾아본다. 만약 Lt가 발견되면(TRUE) Left라고 쓰고, 발견되지 않으면(FALSE) Right라고 쓰라는 명령어이다.
- ▷ filter() 함수는 수많은 데이터 중에서 우리가 원하는 조건을 만족하는 진짜 데이터만 남기고 나머지는 걸러내는 거름망 역할을 한다. filter(A, B)는 A & B의 두 조건을 동시에 만족해야 한다는 뜻이다.
  - ∗ `filter(!is.na(width), width > 0)` 코드는 width 열의 값이 NA(Not Available)인지 물어보는 함수로, !(반대 기호)가 붙어서, '값이 비어있지 않은 데이터만 통과시켜라'는 뜻이다. 엑셀에서 측정값을 적지 않고 비워둔 행들을 걸러내는 작업이다. 또한 `width>0`에 따라 측정값이 0보다 큰 경우만 남기라는 뜻이다.
  - ∗ `filter(Level %in% c("T1", "T2", "T3", "T4", "T5", "T6"))%>%`: Level 내의 글자가 C()라는 바구니 안에 포함되어 있는 것만 취하라는 뜻이다.

• `mutate(level_idx = as.integer(factor(Level, levels = c("T1", "T2", "T3", "T4", "T5",`

이 코드의 의미는 다음과 같다.

- ▷ `factor(Level, levels = c("T1", "T2", "T3", "T4", "T5", "T6"))`는 "이 글자들은 그냥 글자가 아니라 순서와 계급이 있는 집단이다"라고 선언하는 함수이다.
- ▷ as.integer(...)는 그 순서를 정수로 바꾸라는 뜻이다.
- ▷ 변환 결과는 다음과 같다.

표 1: 변환 결과 예시

ID	Level (글자)	**level_idx (숫자)**
P01	T1	**1**
P01	T2	**2**
P01	T6	**6**

여기서 P01은 1번 환자를 나타낸다.

- ▷ 결국 이 코드는 해부학적 용어($T_1 \sim T_6$)를 컴퓨터가 이해할 수 있는 숫자 언어($1 \sim 6$)로 번역한 것이다.

0.2 Stan 코드

파일은 simple_model.stan으로 다음과 같다. (좌우 구분 버전)

```
// This model estimates a mean pedicle width difference
//                  for each vertebral level (T1-T6).

data {
  int<lower=0> N;
  array[N] int<lower=1, upper=6> vertebra_level_numeric;
  array[N] int<lower=1, upper=2> side_numeric; // 1: Concave, 2: Convex
  array[N] real width; // 척추경의 절대 넓이 (mm)
}

parameters {
  // 6개 레벨 x 2개 사이드 = 총 12개의 평균값을 추정한다.
  matrix[6, 2] mu;
  real<lower=0> sigma;
}
```

```
model {
  to_vector(mu) ~ normal(5, 3);
  sigma ~ exponential(1);

  for (i in 1:N) {
    width[i] ~ normal(mu[vertebra_level_numeric[i], side_numeric[i]], sigma);
  }
}

generated quantities {
  // 1. 모든 레벨(T1-T6)의 좌우 평균 차이 계산
  matrix[6, 2] mu_prob_narrow;
  for (k in 1:6) {
    for (s in 1:2) {
     // 각 레벨의 각 측면 평균이 3mm 미만일 확률 (지표 변수)
    // 3mm는 나사가 들어갈 수 있는 최소한의 안전 마진 예시입니다.
      mu_prob_narrow[k, s] = (mu[k, s] < 3.0);
    }
  }

// 2. 전체 데이터에 대한 사후 예측 (모델 검증용)
  array[N] real y_rep;
  for (n in 1:N) {
    y_rep[n] = normal_rng(mu[vertebra_level_numeric[n], side_numeric[n]], sigma);
  }
}
```

---

stan 파일의 코드를 살펴 본다.

- 모수(parameter)는 mu와 sigma의 둘이다. mu는 척추경의 넓이를 나타내고, sigma는 그 표준편차이다.
- model 블록의 to_vector와 sigma는 사전분포(prior distribution)를 나타낸다. mu는 행렬로, Stan은 행렬 자체를 정규분포에 할당하는 것을 허용하지 않는다.

  to_vector(mu) 함수는 행렬(matrix) 형태인 mu를 하나의 긴 열벡터로 변환하여, 모든 원소가 normal(5,3)이라는 사전분포를 따르도록 한꺼번에 선언하기 위해 사용한다.

  ▷ 사전분포는 데이터를 관찰하기 전, 우리가 모델의 모수(parameter)에 대해 가지고 있는 배경 지식이나 가정을 나타낸다.

▷ `to_vector(mu) ~ normal(5, 3)`: 척추경의 넓이인 mu가 정규분포 N(5,3)을 따를 것이라고 사전에 가정하는 것이다. 척추경의 넓이가 대략 5mm 내외일 것이라는 의학적 경험을 반영하는 것으로, 베이즈 통계에서 말하는 '사전분포에 전문가 지식(Expert knowledge)을 반영하는 것'을 잘 보여주는 예시가 된다.

▷ `sigma ~ exponential(1)`: 오차의 표준편차인 sigma가 지수분포를 따를 것이라고 가정하는 것이다. sigma는 항상 양수이므로 지수분포를 주로 사용한다.

- ```
  for (i in 1:N) {
    width[i] ~ normal(mu[vertebra_level_numeric[i], side_numeric[i]], sigma);
  }
  ```

  ▷ for 문장은 가능도(likelihood) 부분이다. 가능도는 모수가 주어졌을 때 관측된 데이터(width)가 나타날 확률을 의미한다.

  ▷ width[i]는 우리가 실제로 측정한 데이터(척추경의 넓이)이다. 이 데이터가 mu(level, side)라는 평균과 sigma라는 표준편차를 가진 정규분포에서 추출되었다고 가정하고 있다. 관측치 하나하나가 모델이 예측하는 분포에 얼마나 적합한지를 계산하는 과정이므로 가능도를 정의하는 것이 맞다.

- 현재의 stan 파일은 모든 환자가 동일한 평균을 가진다고 가정하는 pooling 모델에 가깝다. 이와 대비되는 모델이 환자별 개인차를 반영할 수 있는 계층적 모델(hierarchical model)이다.

  계층적 모델은 따로 다룰 것이다.

- Generated quantities 블록(GQB)에 대해서 알아본다.

  ▷ (곧 다루게 될) 메인 코드에서 적합(fitting)을 하게 되면 사후분포가 나온다.

  * 사후분포로 나오는 것은 모델이 데이터를 학습한 후 업데이트한 모수(파라미터, parameter) 값들이다.
  * 이 모델의 모수는 각 추체 마디의 척추경의 평균 넓이인 mu(12개)와 전체적이 오차 범위인 sigma이다.
  * Stan은 하나의 모수 평균값을 내 놓는 것이 아니라 4000개의 가능성 있는 샘플(숫자 뭉치)을 내놓는다. 이 4000개의 샘플은 데이터 2021개를 가장 잘 설명하는 파라미터들의 확률적 집합이다.

  ▷ GQB에서의 확률적 시뮬레이션 (The Generative Process)

  * 4000개의 파라미터 세트가 GQB라는 공장으로 들어간다. GQB는 샘플링 단계에서 생성된 모수 μ, σ 4000세트를 전달받는 것이다.
  * 4000개 모수 가운데 어떤 한 개를 "T4의 $\mu = 1.06$이고, $\sigma = 0.82$"라고 가정해 보자. 이 모수로 실제 데이터 2021개의 예측치(y_rep_i)를 만든다.
  * 모수 4000세트에 대해서 각각 2021개의 데이터로 예측치를 만들면, 그 결과물(y_rep)은 행 2021개, 열 4000개의 거대한 행렬이 된다.
  ```

▷ y_rep의 구조를 보면,

* 가로(열): 2021개의 데이터
* 세로(행): 4000번의 시뮬레이션(iteration)

▷ 여기서 궁금한 점은 2021이라는 숫자가 어떻게 나왔냐는 점이다.

* 2021은 사후분포의 개수가 아니라 데이터의 개수이다.
* 사후분포의 개수는 T1∼T6의 좌, 우 척추경의 평균 넓이 12개 + 전체 데이터의 변동성을 나타내는 표준편차인 σ 1개 등 총 13개이다.

▷ 2021은 다음의 세 가지 요소가 곱해진 결과이다.

* 환자 수: 데이터 셋에 포함된 유효 환자 수 (약 170명 내외)
* 부위(Level): T1∼T6 (6개 부위)
* 방향(Side): Left, Right 2개

계산: 170×6×2=2040

여기서 filter 명령에 따라 !is.na(width), width> 0 코드가 실행되면서 데이터의 개수가 2021개가 된 것이다.

▷ y_rep의 총 개수는?: 총 4000개의 사후분포 모수 세트가 한번에 하나씩 GQB에 들어옴에 따라 매번 2021개의 y_rep가 생성된다. 따라서 y_rep의 총 개수는 "4000×2021"이 된다. (이에 관해서는 우리 논문6의 15, 16페이지에 잘 정리되어 있다.)

▷ 지금까지 얘기한 내용을 실행하는 코드는 GQB에 있는 다음 코드이다.

```
array[N] real y_rep;
for (n in 1:N) {
  y_rep[n] = normal_rng(mu[vertebra_level_numeric[n], side_numeric[n]], sigma);
```

- GQB에 있는 다음 코드에 대해서도 알아보자.

```
// 1. 모든 레벨(T1-T6)의 좌우 평균 차이 계산
  matrix[6, 2] mu_prob_narrow;
  for (k in 1:6) {
    for (s in 1:2) {
     // 각 레벨의 각 측면 평균이 3mm 미만일 확률 (지표 변수)
    // 3mm는 나사가 들어갈 수 있는 최소한의 안전 마진 예시입니다.
      mu_prob_narrow[k, s] = (mu[k, s] < 3.0);
    }
```

▷ `matrix[6, 2] mu_prob_narrow`: T1∼T6의 좌, 우 척추경의 넓이(width)이 3mm 보다 작을 확률을 구하여 기록할 6×2 행렬을 만들라는 뜻이다.

- ▷ 예를 들면, T4, left 척추경의 넓이의 평균을 $\mu[4,2]$라고 표시하고, $\mu[4,2]$ <3.0mm일 확률을 구한다. 12개의 척추경 각각에 대하여 4000번 측정하여 3.0mm 이하일 확률을 측정하여 만약 3.0보다 작으면 1(TRUE)를 반환하고, 크거나 같으면 0(FALSE)를 반납한다.
- ▷ 이 작업이 4000번의 샘플링 회차마다 매번 실행된다. 만약 어떤 마디에서 4000번 중 400번이 1(TRUE)로 판정되었다면 최종 결과에서는 10%라는 확률로 기록된다.
- ▷ 척추경 width가 3mm 미만일 확률을 보고 싶으면 콘솔에 다음 코드를 치면 된다. 이에 관해서는 메인 코드에서 상세히 설명할 것이다.

```
prob_summary <- summary_fit[grep("mu_prob", rownames(summary_fit)), ]
```

0.3 메인 코드

파일은 main_analysis.R로 코드는 다음과 같다.

```
# main_analysis.R (실행 및 시각화)
library(rstan)
library(here)

# main_analysis.R 상단에 추가 (멀티코어 지원)
options(mc.cores = parallel::detectCores()) # CPU 코어를 모두 사용합니다.
rstan_options(auto_write = TRUE)            # 바뀐 게 없다면 재컴파일을 건너뜁니다.

# 1. 레시피 호출 (데이터 준비)
source(here("data.R"))

# 2. Stan 모델 가동 (어제 만든 simple_model.stan 사용)
fit <- stan(
  file = here("simple_model.stan"),
  data = stan_data,
  iter = 2000, chains = 4, core = 4
)

# 3. 결과 확인: T2 레벨의 오목/볼록측 차이 확인
print(fit, pars = c("mu")) # 평균값 추정치 확인

# 4. 시각화 (선생님이 가장 좋아하시는 사후분포 그래프)
plot(fit, show_density = TRUE, pars = "mu")

# main_analysis.R 의 마지막 부분에 추가하세요
```

```
# --- 4. 통계 결과 요약 및 정렬 (기존 코드) ---
summary_fit <- summary(fit)$summary
# 1. 전체 파라미터(lp__, y_rep 포함)를 평균 순으로 정렬
sorted_summary <- summary_fit[order(summary_fit[, "mean"]), c("mean", "sd", "2.5%", "97.5%

print("=== 1. 전체 결과 정렬 (lp__ 포함 버전) ===")
print(head(sorted_summary, 10)) # 너무 기니까 상위 10개만 먼저 봅니다.


# --- 4-1. mu(마디별 평균)만 골라내기 (새로운 코드 추가) ---
# 2. 행 이름(rownames) 중에 "mu["가 포함된 것만 필터링합니다.
mu_summary <- summary_fit[grep("mu\\[", rownames(summary_fit)), ]

# 3. 골라낸 mu들만 다시 평균 순으로 정렬합니다.
sorted_mu <- mu_summary[order(mu_summary[, "mean"]), c("mean", "sd", "2.5%", "97.5%")]

print("=== 2. 핵심 마디(mu)만 정렬 (교수님 맞춤형) ===")
print(sorted_mu)

# --- 5. 시각화 (LA 아침의 선물 세트) ---
library(bayesplot)
library(ggplot2)

# 파라미터 이름을 직관적으로 변경 (논문용 이름표)
posterior <- as.matrix(fit)
# mu[1,1]부터 mu[6,2]까지 순서대로 이름을 붙입니다.
# 1은 Concave(Left), 2는 Convex(Right)를 의미합니다.
target_names <- c("T1_Concave", "T1_Convex", "T2_Concave", "T2_Convex",
                  "T3_Concave", "T3_Convex", "T4_Concave", "T4_Convex",
                  "T5_Concave", "T5_Convex", "T6_Concave", "T6_Convex")
colnames(posterior)[1:12] <- target_names
posterior[1:6, 1:5]

# 밀도 그래프 출력
mcmc_areas(posterior, pars = target_names, prob = 0.95) +
  geom_vline(xintercept = 3, linetype = "dashed", color = "red") + # 3mm 가이드라인 추가
  ggtitle("Posterior Distributions of Pedicle Width (T1-T6)") +
  xlab("Width (mm)") +
  theme_minimal()
```

위의 코드를 설명하면 다음과 같다.

- `source(here("data.R"))` 코드는 방금 설명한 data.R 코드를 원격 실행하게 하는 코드이다. here()는 현재 작업 중인 R 프로젝트 있는 곳을 작업 디렉토리(working directory)로 자동 인식하게 해 주면서 '작업 디렉토리 꼬임 방지' 역할을 한다.
- stan() 함수로 fitting을 하고 나면, 사후분포의 샘플 정보가 담긴 fit 객체가 생성된다.
- 사후분포는 단일한 값이 아니라 수천 개의 시뮬레이션 샘플로 나타나며, 이를 확인하기 위한 대표적인 명령어들은 print(), summary(), plot(), bayesplot(), mcmc_trace() 등이 있다. 괄호 안에는 `fit`을 넣거나, 특정한 모수만을 볼 때는 '`fit, pars = c("mu", "sigma")`'를 넣으면 된다.
- `summary_fit <- summary(fit)$summary`: summary(fit)에서 summary만 뽑으라는 뜻이다.
- ```
  sorted_summary <- summary_fit[order(summary_fit[, "mean"]),
                   c("mean", "sd", "2.5%", "97.5%")]
  ```

  summary(fit)에서 `c("mean", "sd", "2.5%", "97.5%")`을 열로 하여 데이터프레임을 만드는데 mean을 기준으로 올림 차순으로 만들고 그걸 sorted_summary로 저장하라는 뜻이다.
- 결과를 보면 정작 우리가 보고 싶은 mu는 보이지 않고 다음과 같이 값이 작은 lp__, mu_prob_narrow[]들이 상위를 차지한다.

```
> print(head(sorted_summary, 10)) # 너무 기니까 상위 10개만 먼저 봅니다.
                           mean         sd         2.5%        97.5%
lp__                -604.3350136 2.62843248 -610.6240740 -600.3004578
mu_prob_narrow[1,1]    0.0000000 0.00000000    0.0000000    0.0000000
mu_prob_narrow[1,2]    0.0000000 0.00000000    0.0000000    0.0000000
mu_prob_narrow[2,1]    0.0975000 0.29667449    0.0000000    1.0000000
sigma                  0.8158896 0.01288854    0.7915202    0.8417731
mu_prob_narrow[2,2]    1.0000000 0.00000000    1.0000000    1.0000000
mu_prob_narrow[3,1]    1.0000000 0.00000000    1.0000000    1.0000000
mu_prob_narrow[3,2]    1.0000000 0.00000000    1.0000000    1.0000000
mu_prob_narrow[4,1]    1.0000000 0.00000000    1.0000000    1.0000000
mu_prob_narrow[4,2]    1.0000000 0.00000000    1.0000000    1.0000000
```

- 거대한 도서관(summary_fit)에서 만약 mu에 관한 데이터만 보려면 다음 코드를 사용한다.

```
mu_summary <- summary_fit[grep("mu\\[", rownames(summary_fit)), ]
```

이 코드는 아주 유용한 코드이다. (참고: 이 코드에 대한 설명은 "우리 논문-7, 17페이지"에 정리되어 있다.)
```

- ▷ `grep("mu\\[", ...)`: grep은 수많은 텍스트 중에서 특정 글자 패턴을 찾는 탐정 역할을 하는데 여기서는 " mu["까지만 정확히 찾으라고 명령하는 것이다. 여기서 \\[와 같이 대괄호 앞에 백슬래시 두 개를 붙이는 이유는 특정한 기능이 있는 대괄호가 아니라 진짜 대괄호를 찾으라고 알려주는 일종의 암호(Escape character)이다.
- ▷ 이렇게 찾은 mu의 위치 번호들을 가지고 원본 표에서 해당 줄(Row)들만 통째로 도려낸다.
- ▷ `rownames(summary_fit)), ]`: 쉼표(comma) 뒤의 빈칸은 '해당 행의 모든 열(mean, sd, 신뢰구간 등)을 다 가져오라'는 뜻이다.
- ▷ 이 코드의 결과는 다음과 같이 mu의 결과를 잘 보여준다.

```
> print(sorted_mu)
          mean         sd      2.5%    97.5%
mu[4,2] 1.055265 0.07124917 0.9164127 1.191523
mu[3,2] 1.370167 0.07070239 1.2298499 1.506397
mu[5,2] 1.478759 0.06583415 1.3538639 1.605857
mu[6,2] 1.837771 0.06264470 1.7119079 1.960576
mu[6,1] 2.058397 0.05999232 1.9424473 2.176263
mu[5,1] 2.182983 0.06144841 2.0647396 2.300585
mu[4,1] 2.274216 0.06025104 2.1604288 2.390060
mu[2,2] 2.448928 0.06308703 2.3239928 2.571021
mu[3,1] 2.581710 0.06142642 2.4615003 2.697481
mu[2,1] 3.080214 0.06224230 2.9577498 3.204482
mu[1,2] 3.700279 0.05873870 3.5867184 3.817770
mu[1,1] 3.720804 0.06125955 3.6045285 3.839657
```

- 만약 "나사 삽입 위험 확률(mu_prob_narrow)" 결과만 따로 보고 싶다면 코드의 따옴표 안을 다음과 같이 바꾸면 된다.

```
prob_summary <- summary_fit[grep("mu_prob", rownames(summary_fit)), ]
```

-
```
    # 파라미터 이름을 직관적으로 변경 (논문용 이름표)
posterior <- as.matrix(fit)
# mu[1,1]부터 mu[6,2]까지 순서대로 이름을 붙입니다.
# 1은 Concave(Left), 2는 Convex(Right)를 의미합니다.
target_names <- c("T1_Concave", "T1_Convex", "T2_Concave", "T2_Convex",
                  "T3_Concave", "T3_Convex", "T4_Concave", "T4_Convex",
                  "T5_Concave", "T5_Convex", "T6_Concave", "T6_Convex")
colnames(posterior)[1:12] <- target_names
posterior[1:6, 1:12]
```

- ▷ 사후분포가 담긴 fit 객체를 행렬로 만들어 posterior이라고 한다. dim(posterior)은 4000*2046

으로 posterior는 한 눈에 볼 수 없는 거대한 행렬이다. mu[1,1]부터 mu[6,2]까지 직관적인 이름을 붙이고 그 일부를 보면 다음과 같다.

```
        posterior[1:6, 1:6]
          parameters
iterations T1_Concave T1_Convex T2_Concave T2_Convex T3_Concave T3_Convex
      [1,]   3.768710  3.061559   2.428785  2.255876   2.092251  2.050292
      [2,]   3.693651  3.063508   2.448809  2.323445   2.207049  1.994187
      [3,]   3.670822  3.099799   2.661109  2.255480   2.182055  2.109348
      [4,]   3.775132  3.059331   2.499402  2.293443   2.137739  2.049032
      [5,]   3.687730  3.085335   2.613132  2.258085   2.213609  2.091706
      [6,]   3.690146  3.045044   2.579941  2.195650   2.241927  2.008903
```

- 밀도 그래프는 다음과 같은 코드로 볼 수 있다.

```
mcmc_areas(posterior, pars = target_names, prob = 0.95) +
  geom_vline(xintercept = 3, linetype = "dashed", color = "red")
                + # 3mm 가이드라인 추가
  ggtitle("Posterior Distributions of Pedicle Width (T1-T6)") +
  xlab("Width (mm)") +
  theme_minimal()
```

이때 library(bayesplot), library(ggplot2)를 설치해야 한다.