lixuanze_lab8

Quarto

Quarto enables you to weave together content and executable code into a finished document. To learn more about Quarto see https://quarto.org.

Radon

The goal of this lab is to fit this model to the radon data:

$$ \begin{aligned} y_{i} | \alpha_{j[i]} & \sim N\left(\alpha_{j[i]}+\beta x_{i}, \sigma_{y}^{2}\right), \text { for } i=1,2, \ldots, n \\ \alpha_{j} & \sim N\left(\gamma_{0}+\gamma_{1} u_{j}, \sigma_{\alpha}^{2}\right), \text { for } j=1,2, \ldots, J \end{aligned} $$

i.e. varying intercepts, fixed slope on floor. I want you to

- reproduce the graph on slide 53
- plot samples from the posterior predictive distribution for a new household in county 2 with basement level measurement, compared to samples from the posterior distribution of the mean county effect in county 2 (i.e., a graph similar to slide 45).

Here's code to get the data into a useful format:

```
library(tidyverse)
```

```
── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
✔ dplyr     1.1.4     ✔ readr     2.1.5
✔ forcats   1.0.0     ✔ stringr   1.5.1
✔ ggplot2   3.5.0     ✔ tibble    3.2.1
✔ lubridate 1.9.3     ✔ tidyr     1.3.1
✔ purrr     1.0.2
── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to
become errors
```

```
# house level data
d <- read.table(url("http://www.stat.columbia.edu/~gelman/arm/examples/radon/srrs2.dat"),

# deal with zeros, select what we want, makke a fips variable to match on
d <- d |>
  mutate(activity = ifelse(activity==0, 0.1, activity)) |>
  mutate(fips = stfips * 1000 + cntyfips) |>
```

```
  dplyr::select(fips, state, county, floor, activity)

# county level data
cty <- read.table(url("http://www.stat.columbia.edu/~gelman/arm/examples/radon/cty.dat"),
cty <- cty |> mutate(fips = 1000 * stfips + ctfips) |> dplyr::select(fips, Uppm)

# filter to just be minnesota, join them and then select the variables of interest.
dmn <- d |>
  filter(state=="MN") |>
  dplyr::select(fips, county, floor, activity) |>
  left_join(cty)
```

```
Joining with `by = join_by(fips)`

Warning in left_join(dplyr::select(filter(d, state == "MN"), fips, county, : Detected an
unexpected many-to-many relationship between `x` and `y`.
ℹ Row 102 of `x` matches multiple rows in `y`.
ℹ Row 1327 of `y` matches multiple rows in `x`.
ℹ If a many-to-many relationship is expected, set `relationship =
  "many-to-many"` to silence this warning.
```

```
head(dmn)
```

```
   fips              county floor activity     Uppm
1 27001 AITKIN                1      2.2 0.502054
2 27001 AITKIN                0      2.2 0.502054
3 27001 AITKIN                0      2.9 0.502054
4 27001 AITKIN                0      1.0 0.502054
5 27003 ANOKA                 0      3.1 0.428565
6 27003 ANOKA                 0      2.5 0.428565
```

Note, in the model:

- y_i is log(activity)
- x_i is floor
- u_i is log(Uppm)

Suggested steps

1. write Stan model (note, you will need samples from post pred distribution, either do in Stan or later in R)

Answer: Please see radon.stan file.

2. Get data in stan format

```
library(rstan)
```

```
Loading required package: StanHeaders
```

```
rstan version 2.32.6 (Stan version 2.32.2)

For execution on a local, multicore CPU with excess RAM we recommend calling
options(mc.cores = parallel::detectCores()).
To avoid recompilation of unchanged Stan programs, we recommend calling
rstan_options(auto_write = TRUE)
For within-chain threading using `reduce_sum()` or `map_rect()` Stan functions,
change `threads_per_chain` option:
rstan_options(threads_per_chain = 1)


Attaching package: 'rstan'

The following object is masked from 'package:tidyr':

    extract
```

```r
# data transformations
N_obs <- nrow(dmn)
u <- log(dmn |> group_by(county) |> slice(1) |> select(Uppm) |> pull())
```

```
Adding missing grouping variables: `county`
```

```r
x <- dmn$floor
y <- log(dmn$activity)
county <- as.numeric(as.factor(dmn$county))
J <- length(unique(dmn$county))

stan_data <- list(y = y, x = x, u = u, N = N_obs, J = J, county = county)
```

3. Run the model

```r
hirechy_model <- stan(data = stan_data, file = "radon.stan")
```

```
Trying to compile a simple C file

Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
using C compiler: ‘Apple clang version 15.0.0 (clang-1500.3.9.4)’
using SDK: ‘MacOSX14.4.sdk’
clang -arch arm64 -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -
I"/Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/Rcpp/include/"  -
I"/Library/Frameworks/R.framework/Versions/4.3-
arm64/Resources/library/RcppEigen/include/"  -
I"/Library/Frameworks/R.framework/Versions/4.3-
arm64/Resources/library/RcppEigen/include/unsupported"  -
I"/Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/BH/include" -
I"/Library/Frameworks/R.framework/Versions/4.3-
arm64/Resources/library/StanHeaders/include/src/"  -
I"/Library/Frameworks/R.framework/Versions/4.3-
```

```
arm64/Resources/library/StanHeaders/include/"  -
I"/Library/Frameworks/R.framework/Versions/4.3-
arm64/Resources/library/RcppParallel/include/"  -
I"/Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/rstan/include" -
DEIGEN_NO_DEBUG  -DBOOST_DISABLE_ASSERTS  -DBOOST_PENDING_INTEGER_LOG2_HPP  -
DSTAN_THREADS  -DUSE_STANC3 -DSTRICT_R_HEADERS  -DBOOST_PHOENIX_NO_VARIADIC_EXPRESSION  -
D_HAS_AUTO_PTR_ETC=0  -include '/Library/Frameworks/R.framework/Versions/4.3-
arm64/Resources/library/StanHeaders/include/stan/math/prim/fun/Eigen.hpp'  -D_REENTRANT -
DRCPP_PARALLEL_USE_TBB=1   -I/opt/R/arm64/include    -fPIC  -falign-functions=64 -Wall -g
-O2  -c foo.c -o foo.o
In file included from <built-in>:1:
In file included from /Library/Frameworks/R.framework/Versions/4.3-
arm64/Resources/library/StanHeaders/include/stan/math/prim/fun/Eigen.hpp:22:
In file included from /Library/Frameworks/R.framework/Versions/4.3-
arm64/Resources/library/RcppEigen/include/Eigen/Dense:1:
In file included from /Library/Frameworks/R.framework/Versions/4.3-
arm64/Resources/library/RcppEigen/include/Eigen/Core:19:
/Library/Frameworks/R.framework/Versions/4.3-
arm64/Resources/library/RcppEigen/include/Eigen/src/Core/util/Macros.h:679:10: fatal
error: 'cmath' file not found
#include <cmath>
         ^~~~~~
1 error generated.
make: *** [foo.o] Error 1

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
Chain 1:
Chain 1: Gradient evaluation took 9.1e-05 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.91 seconds.
Chain 1: Adjust your expectations accordingly!
Chain 1:
Chain 1:
Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0.645 seconds (Warm-up)
Chain 1:                0.603 seconds (Sampling)
Chain 1:                1.248 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
```

```
Chain 2:
Chain 2: Gradient evaluation took 2.1e-05 seconds
Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.21 seconds.
Chain 2: Adjust your expectations accordingly!
Chain 2:
Chain 2:
Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 2:
Chain 2:  Elapsed Time: 0.39 seconds (Warm-up)
Chain 2:                0.342 seconds (Sampling)
Chain 2:                0.732 seconds (Total)
Chain 2:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
Chain 3:
Chain 3: Gradient evaluation took 2.8e-05 seconds
Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.28 seconds.
Chain 3: Adjust your expectations accordingly!
Chain 3:
Chain 3:
Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 3:
Chain 3:  Elapsed Time: 0.393 seconds (Warm-up)
Chain 3:                0.314 seconds (Sampling)
Chain 3:                0.707 seconds (Total)
Chain 3:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
Chain 4:
```

```
Chain 4: Gradient evaluation took 2.9e-05 seconds
Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.29 seconds.
Chain 4: Adjust your expectations accordingly!
Chain 4:
Chain 4:
Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 4:
Chain 4:  Elapsed Time: 0.475 seconds (Warm-up)
Chain 4:                0.514 seconds (Sampling)
Chain 4:                0.989 seconds (Total)
Chain 4:
```

```
Warning: There were 2 chains where the estimated Bayesian Fraction of Missing Information
was low. See
https://mc-stan.org/misc/warnings.html#bfmi-low

Warning: Examine the pairs() plot to diagnose sampling problems

Warning: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and
medians may be unreliable.
Running the chains for more iterations may help. See
https://mc-stan.org/misc/warnings.html#bulk-ess

Warning: Tail Effective Samples Size (ESS) is too low, indicating posterior variances and
tail quantiles may be unreliable.
Running the chains for more iterations may help. See
https://mc-stan.org/misc/warnings.html#tail-ess
```

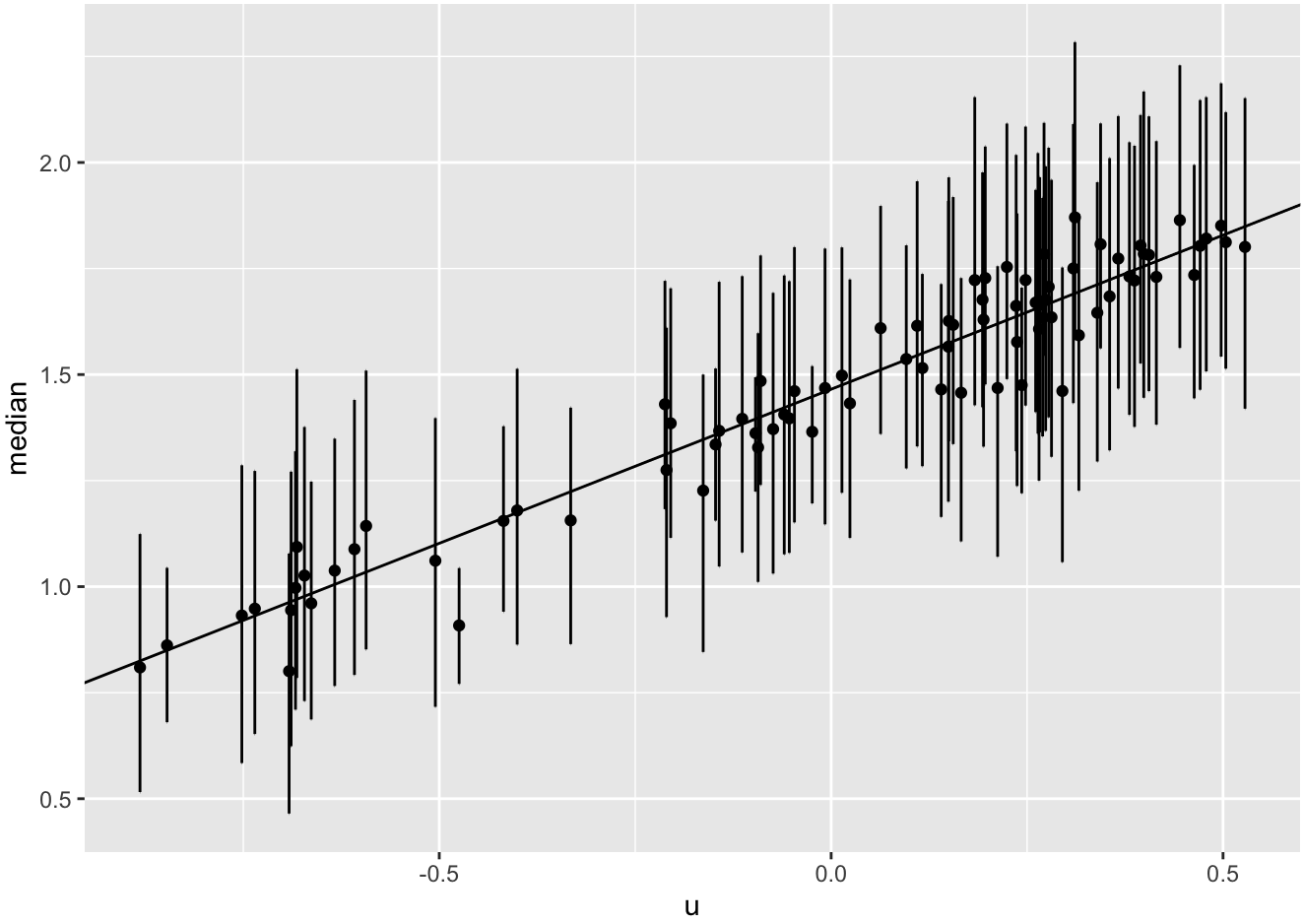
4. For α plot, get median estimates of alpha's, and the 2.5th and 97.5th percentiles. Also get the median (mean fine, easier to pull from summary) of the gamma0 and gamma1. You can then use `geom_abline()` to plot mean regression line.

```
extracted_samples <- rstan::extract(hirechy_model)
names(extracted_samples)
```

```
[1] "alpha"       "beta"        "gamma0"      "gamma1"      "sigma_y"
[6] "sigma_alpha" "lp__"
```

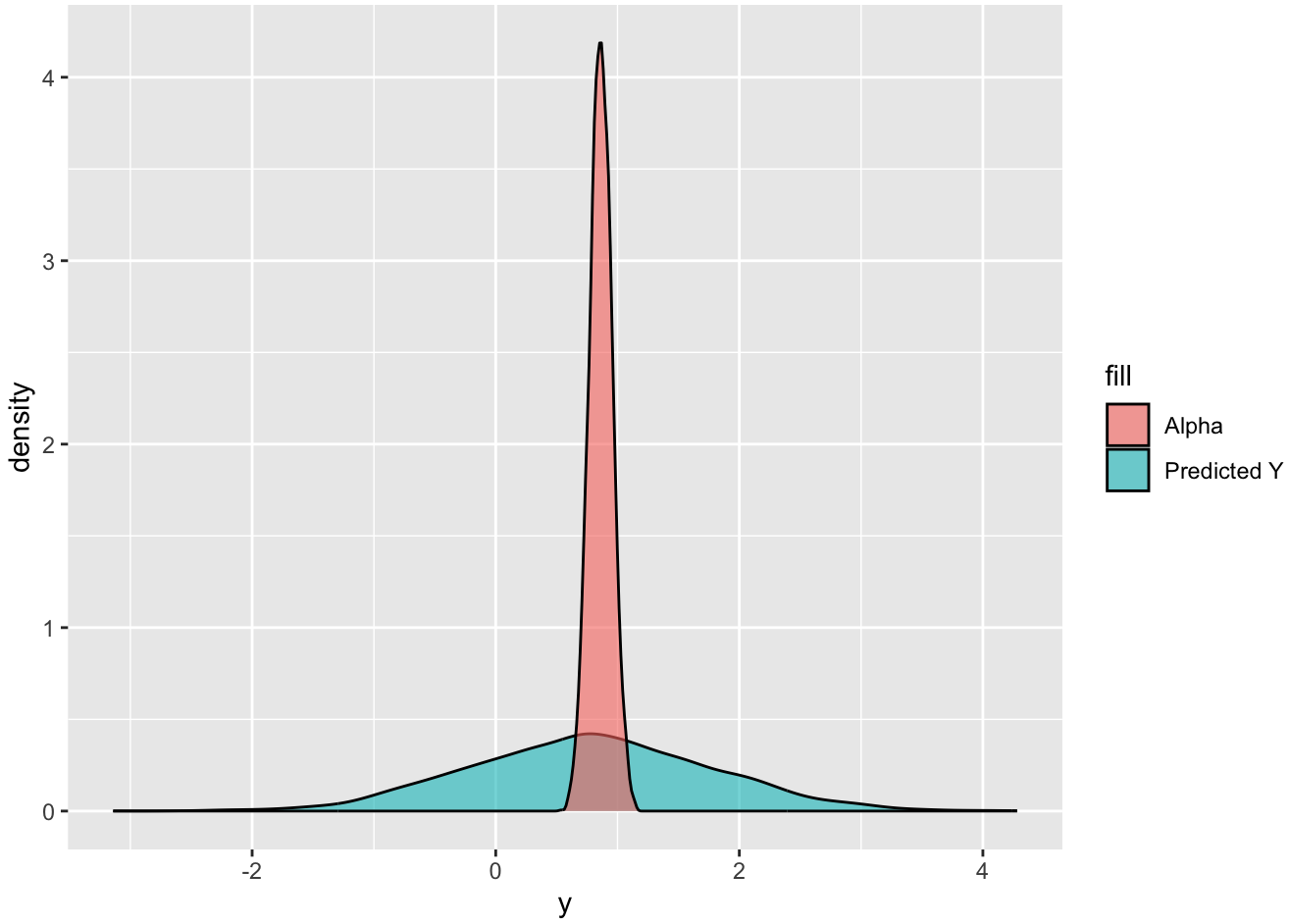
```
alpha_ <- apply(extracted_samples[["alpha"]], 2, median)
alpha_lower <- apply(extracted_samples[["alpha"]], 2, quantile, 0.025)
```

```
alpha_upper <- apply(extracted_samples[["alpha"]], 2, quantile, 0.975)

alpha_df <- tibble(county = 1:J, median = alpha_, lower = alpha_lower, upper = alpha_uppe
gamma0_ <- median(extracted_samples[["gamma0"]])
gamma1_ <- median(extracted_samples[["gamma1"]])

ggplot(alpha_df, aes(u, median)) +
  geom_point()+
  geom_errorbar(aes(ymin = lower, ymax = upper))+
  geom_abline(intercept = gamma0_, slope = gamma1_)
```



5. For the predicted y plot, you will need your posterior predictive samples for y's and then just use `geom_density()`

```
alpha_2 <- extracted_samples[["alpha"]][,2]
sigma_y <- extracted_samples[["sigma_y"]]
y_replicated <- rnorm(alpha_2, sigma_y)

tibble(alpha = alpha_2, y = y_replicated) |>
  ggplot(aes(y)) +
  geom_density(aes(fill = "Predicted Y"), alpha = 0.6)+
  geom_density(aes(alpha, fill = "Alpha"), alpha = 0.6)
```


density
4
3
2
1
0
-2
0
2
4
y
fill
Alpha
Predicted Y