Git collaboration and hierarchical models

Monica Alexander

March 9 2021

1 Git collaboration

1. find a partner, add them as a collaborator to your class repo (you can/should remove them later once this is done)
2. create a text file in your repo with something in it
3. clone your partner's repo, and **on a new branch** make changes to their text file
4. add, commit, push your changes on new branch upstream
5. do a pull request of your partner
6. accept your partners pull request

I'll be able to see the history.

2 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 50.
- 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 39).

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

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

# 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"), header = T, s
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)
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)

So to complete this task successfully you will need to show me / produce:

- stan code for the model
- a plot like slide 39
- a plot like slide 50

Suggested steps

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

```
y <- log(dmn$activity)
x <- dmn$floor
u <- log(unique(dmn$Uppm))
county <- as.numeric(factor(dmn$county))

data <- list(y = y,
             x = x,
             u = u,
             county = county,
             N = length(y),
             J = length(unique(county)))
```

3. Run the model

```
fit <- stan(file = "w8.stan",
            data = data)
```

```
## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## clang -mmacosx-version-min=10.13 -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.0/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/4.0/Resources/library/RcppEigen/inclu
## In file included from /Library/Frameworks/R.framework/Versions/4.0/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/4.0/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
## ^
## /Library/Frameworks/R.framework/Versions/4.0/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
##                ^
##                ;
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.0/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/4.0/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/4.0/Resources/library/RcppEigen/include/Eigen/Core:96:10: fa
## #include <complex>
##          ^~~~~~~~~
## 3 errors generated.
## make: *** [foo.o] Error 1
##
## SAMPLING FOR MODEL 'w8' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.0001 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 1 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: 0.781982 seconds (Warm-up)
## Chain 1:                0.638388 seconds (Sampling)
## Chain 1:                1.42037 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'w8' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 4.2e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.42 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: 0.822394 seconds (Warm-up)
## Chain 2:                1.11653 seconds (Sampling)
## Chain 2:                1.93892 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'w8' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 4.6e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.46 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: 0.798864 seconds (Warm-up)
## Chain 3:                0.647342 seconds (Sampling)
## Chain 3:                1.44621 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'w8' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 4.9e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.49 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:
```

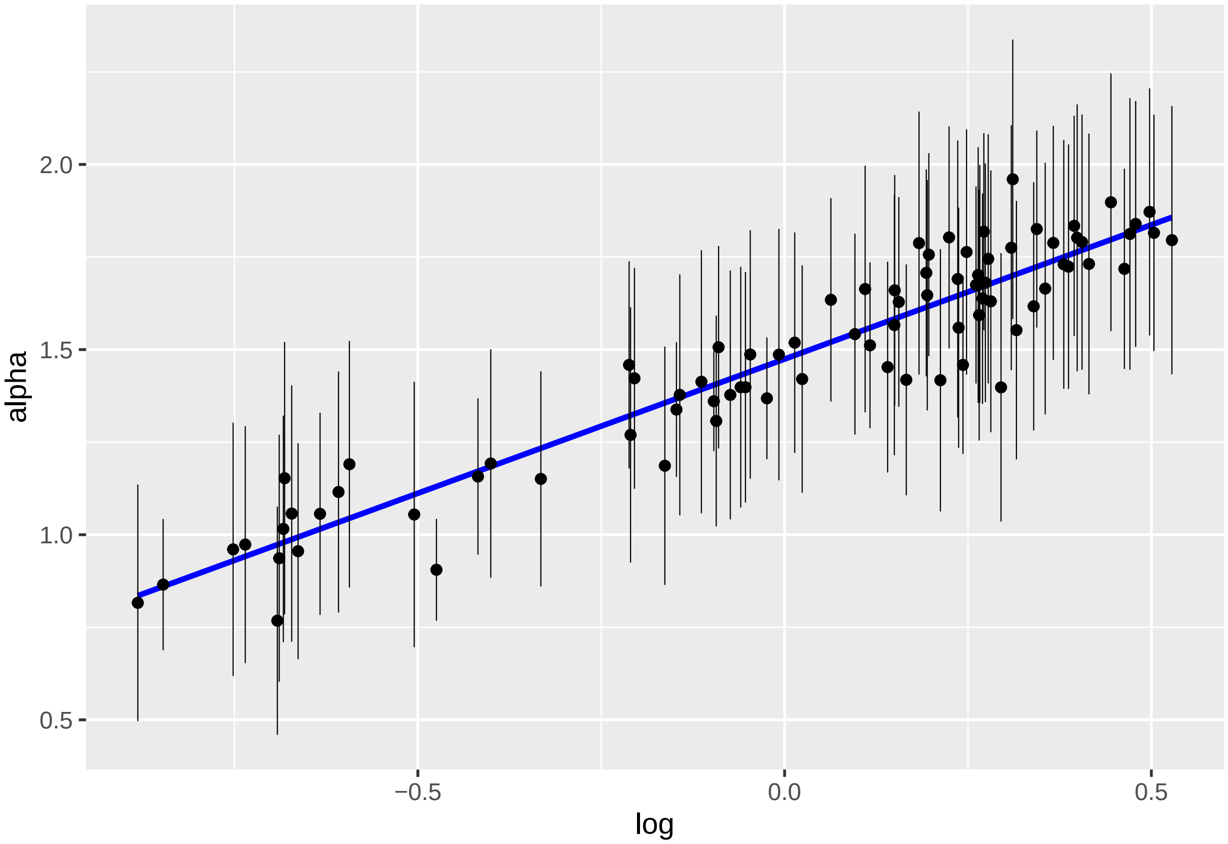
```
## Chain 4:  Elapsed Time: 0.911784 seconds (Warm-up)
## Chain 4:                0.761377 seconds (Sampling)
## Chain 4:                1.67316 seconds (Total)
## Chain 4:
```

```
summary(fit)$summary[c("gamma0", "gamma1", "beta", "sigma", "sigma_alpha"),]
```

```
##                   mean      se_mean         sd        2.5%        25%
## gamma0       1.4708916 0.0014018666 0.04079398  1.39210602  1.4436489
## gamma1       0.7300503 0.0026325859 0.09580947  0.53394701  0.6670911
## beta        -0.6666482 0.0014674153 0.07003224 -0.80360384 -0.7131227
## sigma        0.7690927 0.0003219409 0.01884490  0.73324480  0.7561566
## sigma_alpha  0.1705554 0.0032246865 0.04874034  0.08471495  0.1358755
##                    50%        75%      97.5%     n_eff     Rhat
## gamma0       1.4710847  1.4979258  1.5521666  846.7958 1.004285
## gamma1       0.7309389  0.7942480  0.9159101 1324.4993 1.001299
## beta        -0.6660525 -0.6199167 -0.5314242 2277.6653 1.000335
## sigma        0.7682427  0.7811189  0.8075339 3426.3781 1.000151
## sigma_alpha  0.1681527  0.2020586  0.2711206  228.4558 1.019478
```

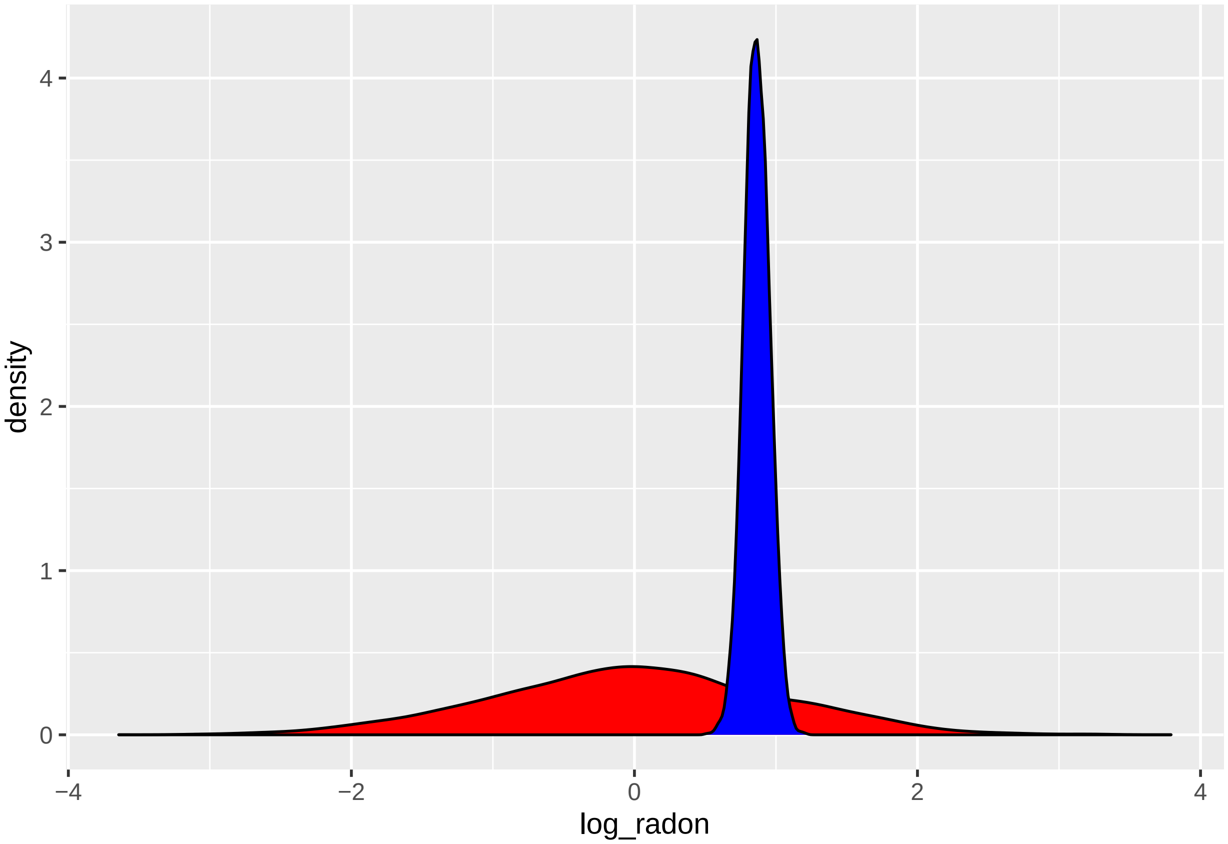
4. For α plot, get median estimates of alpha's, and the 2.5th and 97.5th percentiles. Also get the median (mean fine, easier to pull from summary) of the gamma0 and gamma1. You can then use `geom_abline()` to plot mean regression line.

```
alpha = summary(fit)$summary[c(6:90), c(4,8)]
alpha_df = as.data.frame(alpha)
alpha_df$"log_u" = u
colnames(alpha_df) <- c("min", "max", "log")
alpha_df$"mean" = (alpha_df$min + alpha_df$max)/2
colnames(alpha_df) <- c("alpha", "alpha", "log")
alpha_new = rbind(alpha_df[,c(1,3)], alpha_df[,c(2,3)])

ggplot(data = alpha_new, aes(x = log, y = alpha)) +
  stat_smooth(method = "lm", col = "blue", se=FALSE) + stat_summary(
  size = 0.2,
  fun.min = min,
  fun.max = max,
  fun = mean)
```



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

```
more_data <- extract(fit)
compare = data.frame(new = more_data$mu_alpha)
compare2 = data.frame(new = more_data$alpha[,2])
colnames(compare) <- c("log_radon")

ggplot() +
  geom_density(data = compare, aes(x = log_radon), fill = "red") +
  geom_density(data = compare2, aes(x=new), fill = "blue")
```


density
4
3
2
1
0
−4
−2
0
2
4
log_radon