Practical Bayesian Analysis (with Stan)

Jan Scholz, Sr. Data Scientist at Architech

2016-04-11

Intro

Lecture on Bayesian Analysis held at the Toronto Probabilistic Programming Meetup at Architech, Toronto, 13 April 2016.

Multilevel Models

Radon Levels

Download …

```
wget http://www.stat.columbia.edu/~gelman/arm/examples/ARM_Data.zip
unzip ARM_Data.zip
```

… and read data into R

```
library(dplyr)
library(ggplot2)

t.orig <- read.csv("ARM_Data/radon/srrs2.dat", strip.white = TRUE)
sel <- c("LAC QUI PARLE", "AITKIN", "KOOCHICHING", "DOUGLAS", "CLAY", "STEARNS",
    "RAMSEY", "ST LOUIS")
```

Radon Levels

Massage the data

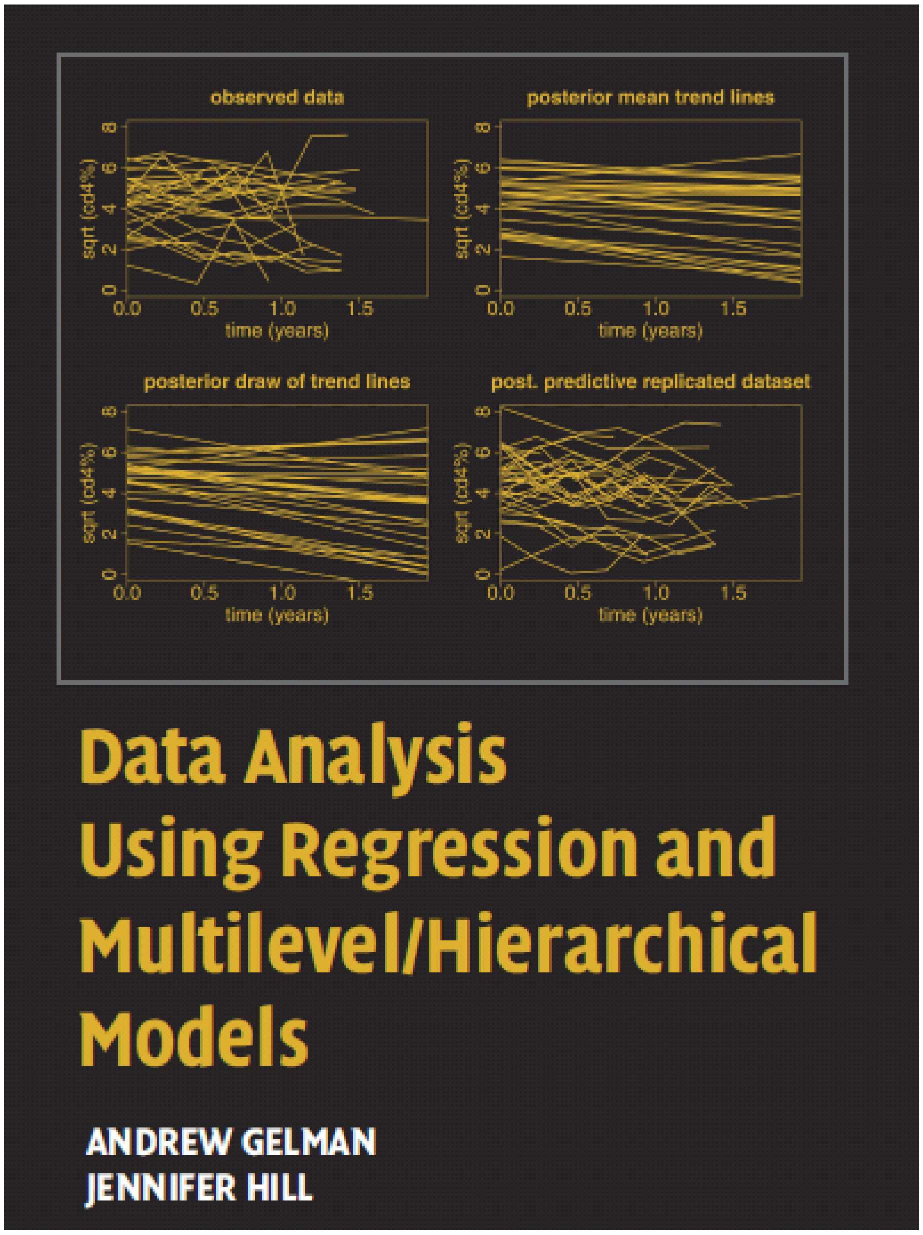

observed data
posterior mean trend lines
posterior draw of trend lines
post. predictive replicated dataset
sqrt (cd4%)
time (years)
Data Analysis Using Regression and Multilevel/Hierarchical Models
ANDREW GELMAN
JENNIFER HILL

Figure 1: Data Analysis Using Regression and Multilevel/Hierachical Models by Andrew Gelman and Jennifer Hill

```
t <- t.orig %>%
    filter(state=='MN' & county %in% sel) %>%
    droplevels() %>%
    select(state, county, basement, floor, activity) %>%
    group_by(county) %>% mutate(n=n()) %>%
    ungroup() %>%
    arrange(n, county) %>%
    mutate(floor=1-floor)

t <- within(t, county <- reorder(county, n))
t
```

```
Source: local data frame [209 x 6]

   state        county basement floor activity n
1     MN LAC QUI PARLE        Y     0     11.3 2
2     MN LAC QUI PARLE        Y     1     16.0 2
3     MN        AITKIN        N     0      2.2 4
4     MN        AITKIN        Y     1      2.2 4
5     MN        AITKIN        Y     1      2.9 4
6     MN        AITKIN        Y     1      1.0 4
7     MN   KOOCHICHING        Y     1      1.7 7
8     MN   KOOCHICHING        Y     1      1.4 7
9     MN   KOOCHICHING        N     0      2.0 7
10    MN   KOOCHICHING        N     0      1.0 7
..   ...           ...      ...   ...      ... .
```

Radon Levels

Complete Pooling

$$y = \alpha + \beta x$$

$$\log(\text{activity}) = \alpha + \beta \cdot \text{floor}$$

```
lm.pool <- lm(log(activity) ~ floor, data=t)
pred <- expand.grid(county=levels(t$county), floor=0:1)
pred$fit.pool <- predict(lm.pool, pred)
```

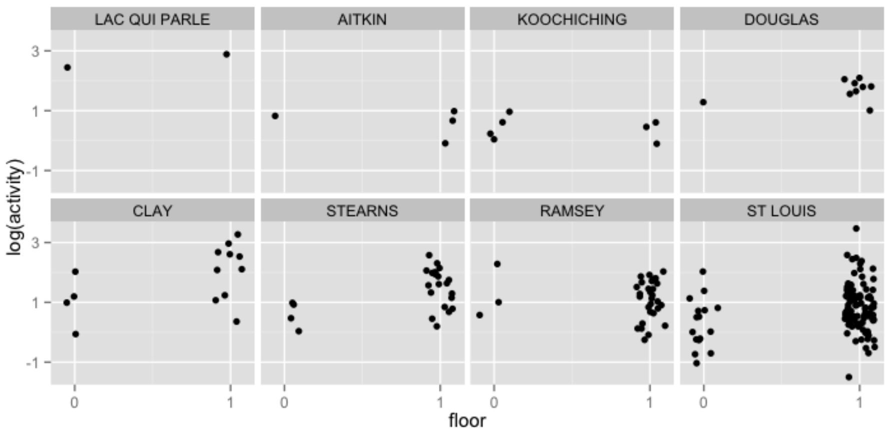


Figure 2: log radon levels depend on floor

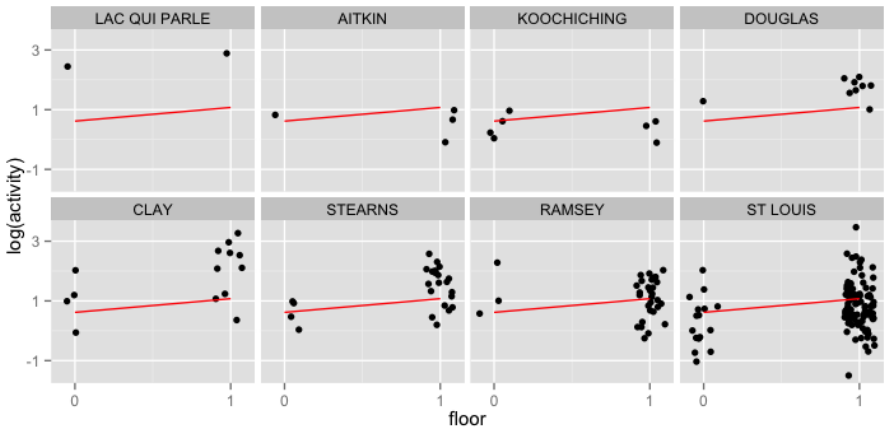


Figure 3: log radon levels depend on floor

Complete Pooling

No-Pooling: separate intercepts

$$y = \alpha_j + \beta x, \qquad j = 1..J$$

with J unique counties

```
lm.nopool <- lm(log(activity) ~ floor + county - 1, data=t)
pred$fit.nopool <- predict(lm.nopool, pred)
```

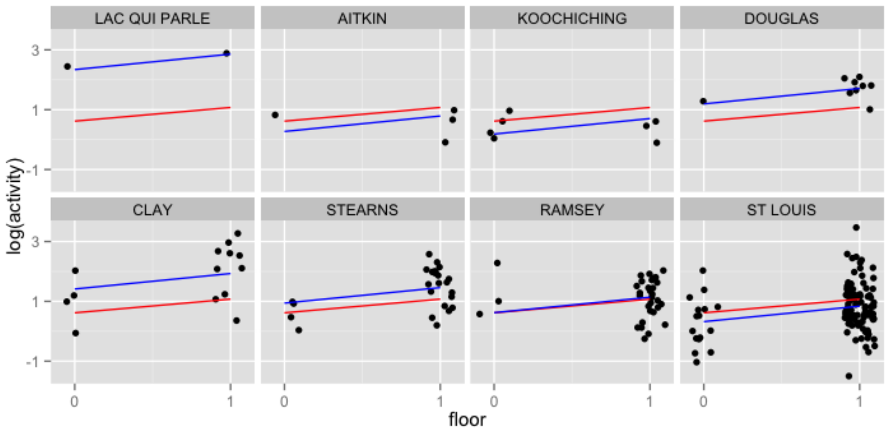


Figure 4:

complete pooling in red, no pooling in blue

Partial Pooling: separate intercepts

```
library(lme4)
lmer.partpool <- lmer(log(activity) ~ floor + (1|county), data=t)
pred$fit.partpool <- predict(lmer.partpool, pred)
```

complete pooling in red, partial pooling in black, no pooling in blue

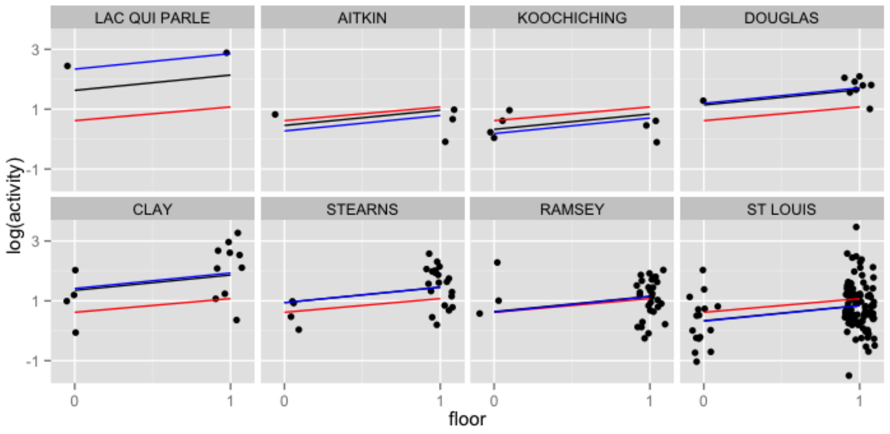


Figure 5:

Stan setup

Stan is an imperative probabilistic programming language, developed for statistical inference.

Stan is a probabilistic programming language in the sense that a random variable is a bona fide first-class object. Observed random variables are declared as data and unobserved random variables are declared as parameters.

For continuous parameters, Stan uses Hamiltonian Monte Carlo (HMC) sampling, a form of Markov chain Monte Carlo (MCMC) sampling. Stan does not provide discrete sampling for parameters. Discrete observations can be handled directly, but discrete parameters must be marginalized out of the model.

```r
library(rstan)

# For local, multicore CPU with excess RAM
#rstan_options(auto_write = TRUE)
#options(mc.cores = parallel::detectCores())

# data for stan
data <- list(N=nrow(t), J=nlevels(t$county),
             county=as.numeric(t$county), x=t$floor, y=t$activity)
```

Bayesian linear regression: complete pooling

$$y_n = \alpha + \beta x_n + \epsilon_n \qquad , \epsilon_n \sim \text{Normal}(0, \sigma)$$

```
model_string <- "
data {
  int<lower=0> N;
  vector[N] y;
  vector[N] x;
}
transformed data {
  vector[N] ylog;
  ylog <- log(y);
}
parameters {
  real alpha;
  real beta;
  real<lower=0> sigma;
}
model{
  ylog ~ normal(alpha + beta * x, sigma);
}"
```

the vectorized model is the same as the more explicit loop

```
  for (n in 1:N)
    y[n] ~ normal(alpha + beta * x[n], sigma);
```

Pop Quiz

What is the distribution of α, β?

Running the model

```
fit <- stan(model_code=model_string, data=data,
            pars=c("alpha", "beta", "sigma"),
            chains=3, iter=500)
```

```
COMPILING THE C++ CODE FOR MODEL 'model_string' NOW.

SAMPLING FOR MODEL 'model_string' NOW (CHAIN 1).

Chain 1, Iteration:   1 / 500 [  0%]  (Warmup)
Chain 1, Iteration:  50 / 500 [ 10%]  (Warmup)
Chain 1, Iteration: 100 / 500 [ 20%]  (Warmup)
Chain 1, Iteration: 150 / 500 [ 30%]  (Warmup)
Chain 1, Iteration: 200 / 500 [ 40%]  (Warmup)
Chain 1, Iteration: 250 / 500 [ 50%]  (Warmup)
Chain 1, Iteration: 251 / 500 [ 50%]  (Sampling)
Chain 1, Iteration: 300 / 500 [ 60%]  (Sampling)
Chain 1, Iteration: 350 / 500 [ 70%]  (Sampling)
Chain 1, Iteration: 400 / 500 [ 80%]  (Sampling)
Chain 1, Iteration: 450 / 500 [ 90%]  (Sampling)
Chain 1, Iteration: 500 / 500 [100%]  (Sampling)
#  Elapsed Time: 0.021228 seconds (Warm-up)
#                0.020529 seconds (Sampling)
#                0.041757 seconds (Total)


SAMPLING FOR MODEL 'model_string' NOW (CHAIN 2).

Chain 2, Iteration:   1 / 500 [  0%]  (Warmup)
Chain 2, Iteration:  50 / 500 [ 10%]  (Warmup)
Chain 2, Iteration: 100 / 500 [ 20%]  (Warmup)
Chain 2, Iteration: 150 / 500 [ 30%]  (Warmup)
Chain 2, Iteration: 200 / 500 [ 40%]  (Warmup)
Chain 2, Iteration: 250 / 500 [ 50%]  (Warmup)
Chain 2, Iteration: 251 / 500 [ 50%]  (Sampling)
Chain 2, Iteration: 300 / 500 [ 60%]  (Sampling)
Chain 2, Iteration: 350 / 500 [ 70%]  (Sampling)
Chain 2, Iteration: 400 / 500 [ 80%]  (Sampling)
Chain 2, Iteration: 450 / 500 [ 90%]  (Sampling)
Chain 2, Iteration: 500 / 500 [100%]  (Sampling)
#  Elapsed Time: 0.024614 seconds (Warm-up)
#                0.022027 seconds (Sampling)
#                0.046641 seconds (Total)


SAMPLING FOR MODEL 'model_string' NOW (CHAIN 3).

Chain 3, Iteration:   1 / 500 [  0%]  (Warmup)
Chain 3, Iteration:  50 / 500 [ 10%]  (Warmup)
Chain 3, Iteration: 100 / 500 [ 20%]  (Warmup)
Chain 3, Iteration: 150 / 500 [ 30%]  (Warmup)
Chain 3, Iteration: 200 / 500 [ 40%]  (Warmup)
Chain 3, Iteration: 250 / 500 [ 50%]  (Warmup)
Chain 3, Iteration: 251 / 500 [ 50%]  (Sampling)
Chain 3, Iteration: 300 / 500 [ 60%]  (Sampling)
Chain 3, Iteration: 350 / 500 [ 70%]  (Sampling)
Chain 3, Iteration: 400 / 500 [ 80%]  (Sampling)
Chain 3, Iteration: 450 / 500 [ 90%]  (Sampling)
Chain 3, Iteration: 500 / 500 [100%]  (Sampling)
```

```
#  Elapsed Time: 0.028866 seconds (Warm-up)
#                0.022698 seconds (Sampling)
#                0.051564 seconds (Total)
```

Output

```
print(fit, probs=c(0.025,0.5,0.975))
```

```
Inference for Stan model: model_string.
3 chains, each with iter=500; warmup=250; thin=1;
post-warmup draws per chain=250, total post-warmup draws=750.

        mean se_mean   sd   2.5%    50%  97.5% n_eff Rhat
alpha   0.61    0.01 0.14   0.33   0.62   0.87   180 1.01
beta    0.46    0.01 0.16   0.17   0.45   0.79   185 1.01
sigma   0.83    0.00 0.04   0.75   0.83   0.92   268 1.01
lp__  -64.08    0.11 1.34 -67.66 -63.73 -62.61   154 1.00

Samples were drawn using NUTS(diag_e) at Wed Apr 20 22:10:43 2016.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

Traceplot

```
traceplot(fit, pars=c("alpha", "beta", "sigma"), window=c(50,500))
```

Correlation Matrix

```
pairs(fit, pars=c("alpha", "beta", "sigma"))
```

Diagnostics

```
plot(get_sampler_params(fit)[[1]][,2], ylim=c(0,1), ylab='stepsize')
```

```
get_sampler_params(fit)[[1]][245:255,]
```

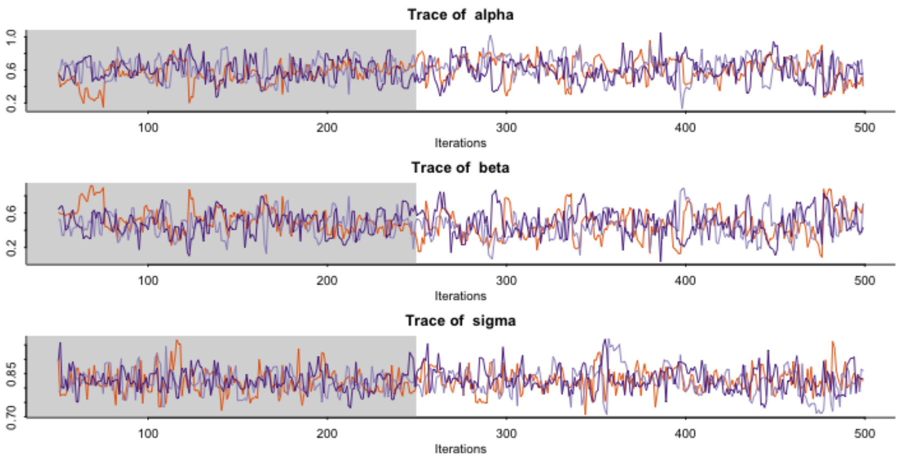


Figure 6:

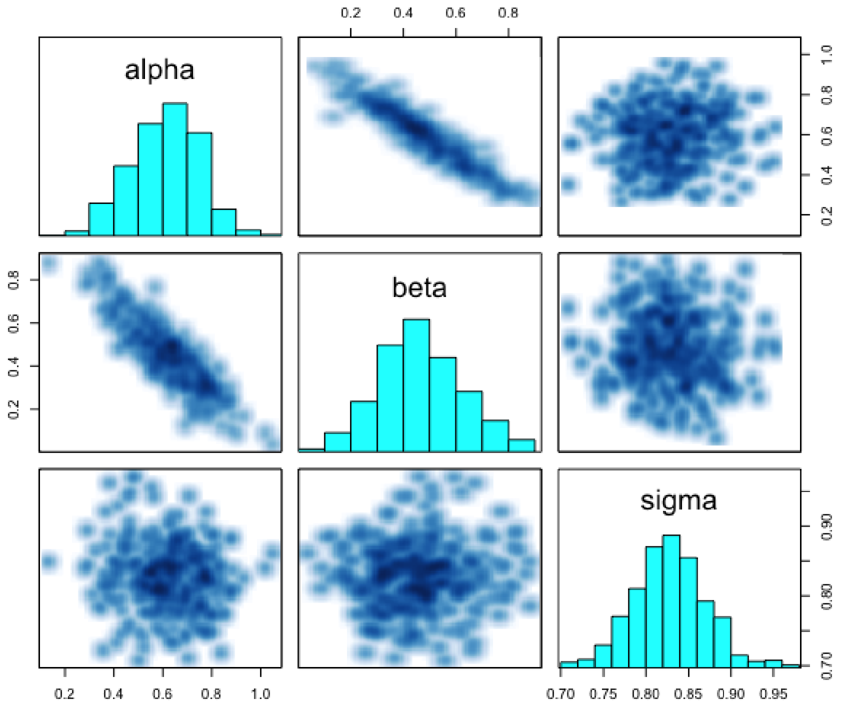


Figure 7:

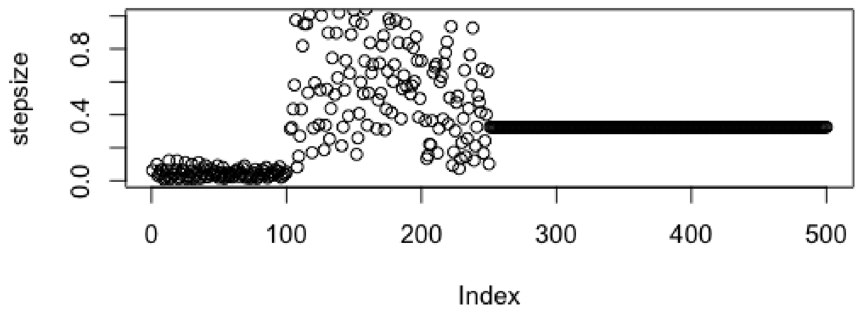


Figure 8:

```
      accept_stat__ stepsize__ treedepth__ n_leapfrog__ n_divergent__
 [1,]   0.215409358  0.6795622           2            3             0
 [2,]   0.940335877  0.1669584           3            7             0
 [3,]   0.997952450  0.2421507           3            7             0
 [4,]   1.000000000  0.4012850           1            1             0
 [5,]   0.002458872  0.6635630           1            1             0
 [6,]   0.996209609  0.1021665           5           31             0
 [7,]   0.775731648  0.3257149           3            5             0
 [8,]   1.000000000  0.3257149           2            3             0
 [9,]   0.998846767  0.3257149           4           15             0
[10,]   0.811089357  0.3257149           2            3             0
[11,]   0.996502162  0.3257149           4           15             0
```

Bayesian linear regression: complete pooling

```
s <- extract(fit) # samples

set.seed(1)
ggplot(t, aes(floor, log(activity))) +
    geom_point(position = position_jitter(width=0.1)) +
    facet_wrap(~county, ncol=4) +
    scale_x_continuous(breaks = c(0,1)) +
    scale_y_continuous(breaks = c(-1,1,3)) +
    geom_abline(intercept=mean(s$a), slope=mean(s$b), color='red')
```

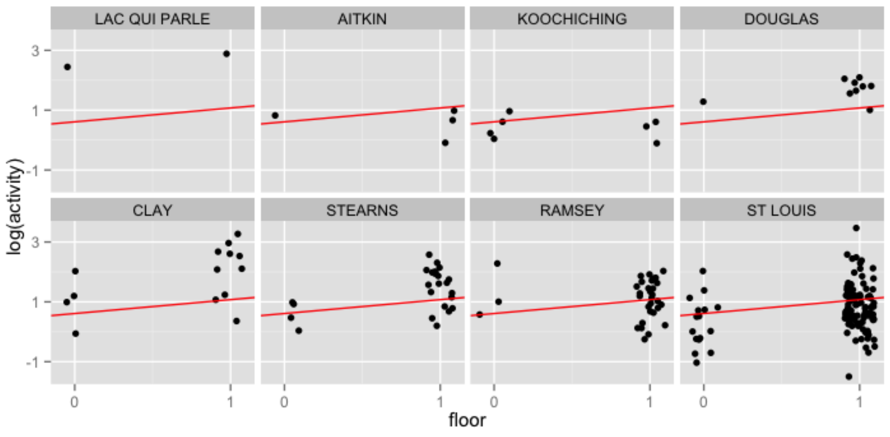


Figure 9:

Bayesian linear regression: no pooling

Model definition

```
model_string <- "
data {
  int<lower=0> N;
  int<lower=0> J;           // ++ number of counties
  vector[N] y;
  vector[N] x;
  int<lower=0> county[N];  // ++ county
}
transformed data {
  vector[N] ylog;
  ylog <- log(y);
}
parameters {
  real alpha[J];            // ++ random effec
  real beta;
  real<lower=0> sigma;
}
model{
  for (n in 1:N)            // vv county indicator
    ylog[n] ~ normal(alpha[county[n]] + beta * x[n], sigma);
}"

fit.nopool <- stan(model_code=model_string, data=data, pars=c("alpha", "beta", "sigma"), cha
COMPILING THE C++ CODE FOR MODEL 'model_string' NOW.

SAMPLING FOR MODEL 'model_string' NOW (CHAIN 1).

Chain 1, Iteration:   1 / 500 [  0%]  (Warmup)
Chain 1, Iteration:  50 / 500 [ 10%]  (Warmup)
Chain 1, Iteration: 100 / 500 [ 20%]  (Warmup)
Chain 1, Iteration: 150 / 500 [ 30%]  (Warmup)
Chain 1, Iteration: 200 / 500 [ 40%]  (Warmup)
Chain 1, Iteration: 250 / 500 [ 50%]  (Warmup)
Chain 1, Iteration: 251 / 500 [ 50%]  (Sampling)
Chain 1, Iteration: 300 / 500 [ 60%]  (Sampling)
Chain 1, Iteration: 350 / 500 [ 70%]  (Sampling)
Chain 1, Iteration: 400 / 500 [ 80%]  (Sampling)
Chain 1, Iteration: 450 / 500 [ 90%]  (Sampling)
Chain 1, Iteration: 500 / 500 [100%]  (Sampling)
#  Elapsed Time: 0.079405 seconds (Warm-up)
#                0.051515 seconds (Sampling)
#                0.13092 seconds (Total)


SAMPLING FOR MODEL 'model_string' NOW (CHAIN 2).

Chain 2, Iteration:   1 / 500 [  0%]  (Warmup)
Chain 2, Iteration:  50 / 500 [ 10%]  (Warmup)
Chain 2, Iteration: 100 / 500 [ 20%]  (Warmup)
Chain 2, Iteration: 150 / 500 [ 30%]  (Warmup)
Chain 2, Iteration: 200 / 500 [ 40%]  (Warmup)
Chain 2, Iteration: 250 / 500 [ 50%]  (Warmup)
Chain 2, Iteration: 251 / 500 [ 50%]  (Sampling)
Chain 2, Iteration: 300 / 500 [ 60%]  (Sampling)
Chain 2, Iteration: 350 / 500 [ 70%]  (Sampling)
Chain 2, Iteration: 400 / 500 [ 80%]  (Sampling)
Chain 2, Iteration: 450 / 500 [ 90%]  (Sampling)
Chain 2, Iteration: 500 / 500 [100%]  (Sampling)
#  Elapsed Time: 0.083076 seconds (Warm-up)
#                0.052725 seconds (Sampling)
#                0.135801 seconds (Total)


SAMPLING FOR MODEL 'model_string' NOW (CHAIN 3).

Chain 3, Iteration:   1 / 500 [  0%]  (Warmup)
Chain 3, Iteration:  50 / 500 [ 10%]  (Warmup)
Chain 3, Iteration: 100 / 500 [ 20%]  (Warmup)
Chain 3, Iteration: 150 / 500 [ 30%]  (Warmup)
Chain 3, Iteration: 200 / 500 [ 40%]  (Warmup)
Chain 3, Iteration: 250 / 500 [ 50%]  (Warmup)
Chain 3, Iteration: 251 / 500 [ 50%]  (Sampling)
Chain 3, Iteration: 300 / 500 [ 60%]  (Sampling)
Chain 3, Iteration: 350 / 500 [ 70%]  (Sampling)
Chain 3, Iteration: 400 / 500 [ 80%]  (Sampling)
Chain 3, Iteration: 450 / 500 [ 90%]  (Sampling)
Chain 3, Iteration: 500 / 500 [100%]  (Sampling)
#  Elapsed Time: 0.067091 seconds (Warm-up)
#                0.048676 seconds (Sampling)
#                0.115767 seconds (Total)
```

```
print(fit.nopool, probs=c(0.025,0.5,0.975))
```

```
Inference for Stan model: model_string.
3 chains, each with iter=500; warmup=250; thin=1;
post-warmup draws per chain=250, total post-warmup draws=750.

           mean se_mean   sd   2.5%    50%  97.5% n_eff Rhat
alpha[1]   2.33    0.02 0.51   1.35   2.32   3.30   528    1
alpha[2]   0.28    0.02 0.40  -0.50   0.29   1.05   553    1
alpha[3]   0.19    0.01 0.30  -0.31   0.17   0.81   454    1
alpha[4]   1.19    0.01 0.28   0.67   1.18   1.78   453    1
alpha[5]   1.40    0.01 0.23   0.95   1.41   1.85   426    1
alpha[6]   0.93    0.01 0.19   0.52   0.93   1.30   375    1
alpha[7]   0.61    0.01 0.19   0.26   0.62   1.00   355    1
alpha[8]   0.32    0.01 0.15   0.03   0.32   0.62   345    1
beta       0.53    0.01 0.15   0.22   0.53   0.82   296    1
sigma      0.74    0.00 0.04   0.67   0.74   0.82   430    1
lp__     -41.04    0.13 2.37 -46.38 -40.78 -37.49   344    1

Samples were drawn using NUTS(diag_e) at Wed Apr 20 22:11:02 2016.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

Bayesian linear regression: no pooling

```
s.nopool <- extract(fit.nopool) # samples
pred$alpha <- apply(s.nopool[['alpha']], 2, mean)
pred$beta  <- mean(s.nopool[['beta']])
pred <- pred %>%
    mutate(bayes.nopool = alpha + beta*floor)
```

```
set.seed(1)
ggplot(t, aes(floor, log(activity))) +
    geom_point(position = position_jitter(width=0.1)) +
    facet_wrap(~county, ncol=4) +
    scale_x_continuous(breaks = c(0,1)) +
    scale_y_continuous(breaks = c(-1,1,3)) +
    geom_line(data=pred, aes(floor, bayes.nopool), color='blue') +
    geom_line(data=pred, aes(floor, fit.nopool), color='red', linetype="dashed")
```

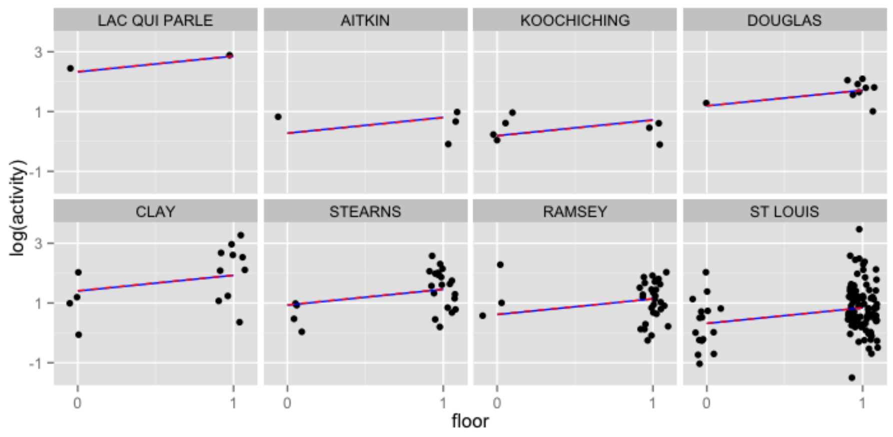


Figure 10:

Bayesian linear regression: partial pooling

Model definition

```
model_string <- "
data {
  int<lower=0> N;
  int<lower=0> J;
  vector[N] y;
  vector[N] x;
  int<lower=0> county[N];
}
transformed data {
  vector[N] ylog;
  ylog <- log(y);
}
```

```
parameters {
  real alpha[J];
  real beta;
  real<lower=0> sigma;
  real mu;                           // ++ hyper prior
  real<lower=0> sigma_mu;            // ++ hyper prior
}
model{
  mu ~ normal(0, 100);               // ++ hyper prior
  sigma_mu ~ cauchy(0,5);            // ++ hyper prior

  sigma ~ cauchy(0,5);               // ++ prior
  for (j in 1:J)                     // ++ prior
    alpha[j] ~ normal(mu,sigma_mu); // ++ prior

  for (n in 1:N)
    ylog[n] ~ normal(alpha[county[n]] + beta * x[n], sigma);

}"
```

```
fit.partpool <- stan(model_code=model_string, data=data, pars=c("alpha", "beta", "sigma"), c
```

```
COMPILING THE C++ CODE FOR MODEL 'model_string' NOW.

SAMPLING FOR MODEL 'model_string' NOW (CHAIN 1).

Chain 1, Iteration:   1 / 500 [  0%]  (Warmup)
Chain 1, Iteration:  50 / 500 [ 10%]  (Warmup)
Chain 1, Iteration: 100 / 500 [ 20%]  (Warmup)
Chain 1, Iteration: 150 / 500 [ 30%]  (Warmup)
Chain 1, Iteration: 200 / 500 [ 40%]  (Warmup)
Chain 1, Iteration: 250 / 500 [ 50%]  (Warmup)
Chain 1, Iteration: 251 / 500 [ 50%]  (Sampling)
Chain 1, Iteration: 300 / 500 [ 60%]  (Sampling)
Chain 1, Iteration: 350 / 500 [ 70%]  (Sampling)
Chain 1, Iteration: 400 / 500 [ 80%]  (Sampling)
Chain 1, Iteration: 450 / 500 [ 90%]  (Sampling)
Chain 1, Iteration: 500 / 500 [100%]  (Sampling)
#  Elapsed Time: 0.07958 seconds (Warm-up)
#                0.060445 seconds (Sampling)
#                0.140025 seconds (Total)


SAMPLING FOR MODEL 'model_string' NOW (CHAIN 2).

Chain 2, Iteration:   1 / 500 [  0%]  (Warmup)
Chain 2, Iteration:  50 / 500 [ 10%]  (Warmup)
Chain 2, Iteration: 100 / 500 [ 20%]  (Warmup)
Chain 2, Iteration: 150 / 500 [ 30%]  (Warmup)
Chain 2, Iteration: 200 / 500 [ 40%]  (Warmup)
Chain 2, Iteration: 250 / 500 [ 50%]  (Warmup)
Chain 2, Iteration: 251 / 500 [ 50%]  (Sampling)
Chain 2, Iteration: 300 / 500 [ 60%]  (Sampling)
Chain 2, Iteration: 350 / 500 [ 70%]  (Sampling)
Chain 2, Iteration: 400 / 500 [ 80%]  (Sampling)
Chain 2, Iteration: 450 / 500 [ 90%]  (Sampling)
Chain 2, Iteration: 500 / 500 [100%]  (Sampling)
#  Elapsed Time: 0.080551 seconds (Warm-up)
#                0.053712 seconds (Sampling)
#                0.134263 seconds (Total)


SAMPLING FOR MODEL 'model_string' NOW (CHAIN 3).

Chain 3, Iteration:   1 / 500 [  0%]  (Warmup)
Chain 3, Iteration:  50 / 500 [ 10%]  (Warmup)
Chain 3, Iteration: 100 / 500 [ 20%]  (Warmup)
Chain 3, Iteration: 150 / 500 [ 30%]  (Warmup)
Chain 3, Iteration: 200 / 500 [ 40%]  (Warmup)
Chain 3, Iteration: 250 / 500 [ 50%]  (Warmup)
Chain 3, Iteration: 251 / 500 [ 50%]  (Sampling)
Chain 3, Iteration: 300 / 500 [ 60%]  (Sampling)
Chain 3, Iteration: 350 / 500 [ 70%]  (Sampling)
Chain 3, Iteration: 400 / 500 [ 80%]  (Sampling)
Chain 3, Iteration: 450 / 500 [ 90%]  (Sampling)
Chain 3, Iteration: 500 / 500 [100%]  (Sampling)
#  Elapsed Time: 0.065033 seconds (Warm-up)
#                0.03887 seconds (Sampling)
#                0.103903 seconds (Total)
```

```
print(fit.partpool, probs=c(0.025,0.5,0.975))
```

```
Inference for Stan model: model_string.
3 chains, each with iter=500; warmup=250; thin=1;
post-warmup draws per chain=250, total post-warmup draws=750.

           mean se_mean   sd  2.5%   50% 97.5% n_eff Rhat
alpha[1]   1.73    0.03 0.50  0.80  1.71  2.79   370 1.00
alpha[2]   0.41    0.02 0.35 -0.30  0.43  1.05   363 1.01
alpha[3]   0.30    0.01 0.28 -0.29  0.32  0.85   533 1.00
alpha[4]   1.15    0.01 0.27  0.67  1.15  1.70   410 1.00
```

```
alpha[5]    1.35    0.01 0.23   0.89   1.35   1.78    372 1.00
alpha[6]    0.93    0.01 0.19   0.56   0.93   1.28    272 1.00
alpha[7]    0.63    0.01 0.18   0.28   0.63   0.96    267 1.00
alpha[8]    0.32    0.01 0.14   0.05   0.32   0.59    212 1.00
beta        0.52    0.01 0.14   0.25   0.52   0.81    205 1.00
sigma       0.74    0.00 0.04   0.67   0.74   0.81    585 1.00
lp__      -42.10    0.18 2.66 -47.93 -41.72 -37.96    219 1.01

Samples were drawn using NUTS(diag_e) at Wed Apr 20 22:11:20 2016.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

Bayesian linear regression: partial pooling

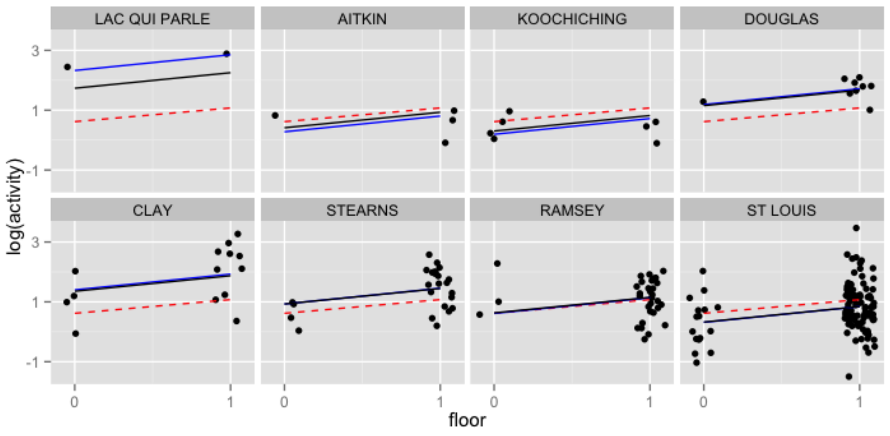


Figure 11:

Why Bayes?

Robust Noise Models, e.g. noise term as a Student-t distribution

```
data {
  real<lower=0> nu;
  ...
}
...
```

```
model {
  for (n in 1:N)
    y[n] ~ student_t(nu, alpha + beta * x[n], sigma);
  ...
}
```

```
  ylog[n] ~ normal(alpha[county[n]] + beta * x[n], sigma);
```

Why Bayes?

Extensibility

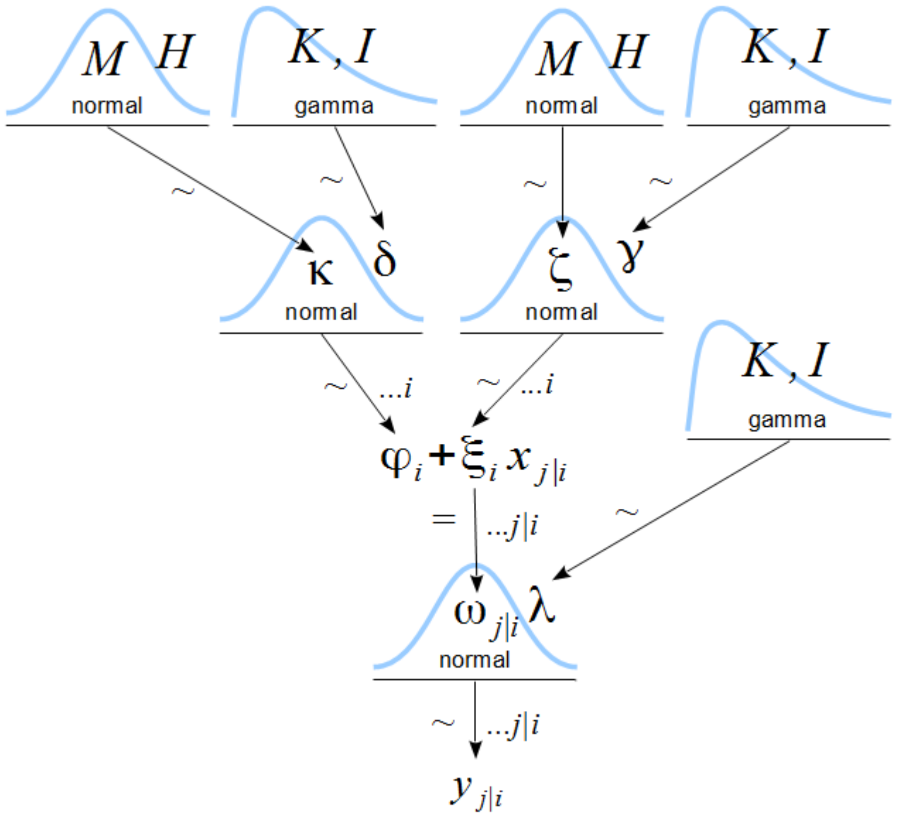


Figure 12: It's natural to add levels of hierachy and constraints

Censoring

```
model_string <- "
data {
  int<lower=0> N;
  int<lower=0> N_cens;
  vector[N] y;
  vector[N] x;
  vector[N_cens] x_cens;
  real<lower=max(y)> U;
}
parameters {
  real alpha;
  real beta;
  real<lower=0> sigma;
  vector<lower=U>[N_cens] y_cens;
}
model{
  y      ~ normal(alpha + beta * x,      sigma);
  y_cens ~ normal(alpha + beta * x_cens, sigma);
}"

U <- 2
t$censor <- as.numeric(log(t$activity) > U)
tc <- t %>% filter(censor==0)
data.censor <- list(N=sum(1-t$censor), N_cens=sum(t$censor), x=tc$floor, y=log(tc$activity)

fit.censor <- stan(model_code=model_string, data=data.censor, pars=c("alpha", "beta", "sigma
```

```
COMPILING THE C++ CODE FOR MODEL 'model_string' NOW.

SAMPLING FOR MODEL 'model_string' NOW (CHAIN 1).

Chain 1, Iteration:   1 / 500 [  0%]  (Warmup)
Chain 1, Iteration:  50 / 500 [ 10%]  (Warmup)
Chain 1, Iteration: 100 / 500 [ 20%]  (Warmup)
Chain 1, Iteration: 150 / 500 [ 30%]  (Warmup)
Chain 1, Iteration: 200 / 500 [ 40%]  (Warmup)
Chain 1, Iteration: 250 / 500 [ 50%]  (Warmup)
Chain 1, Iteration: 251 / 500 [ 50%]  (Sampling)
Chain 1, Iteration: 300 / 500 [ 60%]  (Sampling)
Chain 1, Iteration: 350 / 500 [ 70%]  (Sampling)
Chain 1, Iteration: 400 / 500 [ 80%]  (Sampling)
Chain 1, Iteration: 450 / 500 [ 90%]  (Sampling)
Chain 1, Iteration: 500 / 500 [100%]  (Sampling)
```

```
#  Elapsed Time: 0.207793 seconds (Warm-up)
#                0.057627 seconds (Sampling)
#                0.26542 seconds (Total)


SAMPLING FOR MODEL 'model_string' NOW (CHAIN 2).

Chain 2, Iteration:   1 / 500 [  0%]  (Warmup)
Chain 2, Iteration:  50 / 500 [ 10%]  (Warmup)
Chain 2, Iteration: 100 / 500 [ 20%]  (Warmup)
Chain 2, Iteration: 150 / 500 [ 30%]  (Warmup)
Chain 2, Iteration: 200 / 500 [ 40%]  (Warmup)
Chain 2, Iteration: 250 / 500 [ 50%]  (Warmup)
Chain 2, Iteration: 251 / 500 [ 50%]  (Sampling)
Chain 2, Iteration: 300 / 500 [ 60%]  (Sampling)
Chain 2, Iteration: 350 / 500 [ 70%]  (Sampling)
Chain 2, Iteration: 400 / 500 [ 80%]  (Sampling)
Chain 2, Iteration: 450 / 500 [ 90%]  (Sampling)
Chain 2, Iteration: 500 / 500 [100%]  (Sampling)
#  Elapsed Time: 0.206245 seconds (Warm-up)
#                0.049896 seconds (Sampling)
#                0.256141 seconds (Total)


SAMPLING FOR MODEL 'model_string' NOW (CHAIN 3).

Chain 3, Iteration:   1 / 500 [  0%]  (Warmup)
Chain 3, Iteration:  50 / 500 [ 10%]  (Warmup)
Chain 3, Iteration: 100 / 500 [ 20%]  (Warmup)
Chain 3, Iteration: 150 / 500 [ 30%]  (Warmup)
Chain 3, Iteration: 200 / 500 [ 40%]  (Warmup)
Chain 3, Iteration: 250 / 500 [ 50%]  (Warmup)
Chain 3, Iteration: 251 / 500 [ 50%]  (Sampling)
Chain 3, Iteration: 300 / 500 [ 60%]  (Sampling)
Chain 3, Iteration: 350 / 500 [ 70%]  (Sampling)
Chain 3, Iteration: 400 / 500 [ 80%]  (Sampling)
Chain 3, Iteration: 450 / 500 [ 90%]  (Sampling)
Chain 3, Iteration: 500 / 500 [100%]  (Sampling)
#  Elapsed Time: 0.195814 seconds (Warm-up)
#                0.055355 seconds (Sampling)
#                0.251169 seconds (Total)
```

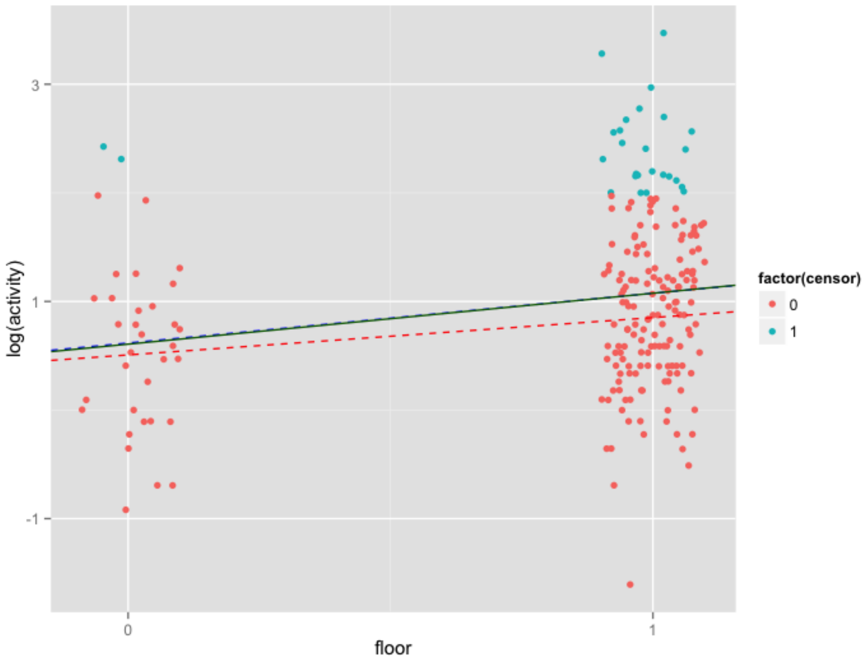


Figure 13:

Censoring

truth in blue, without censored values in red, and censor model in green

Books

Stan: http://mc-stan.org

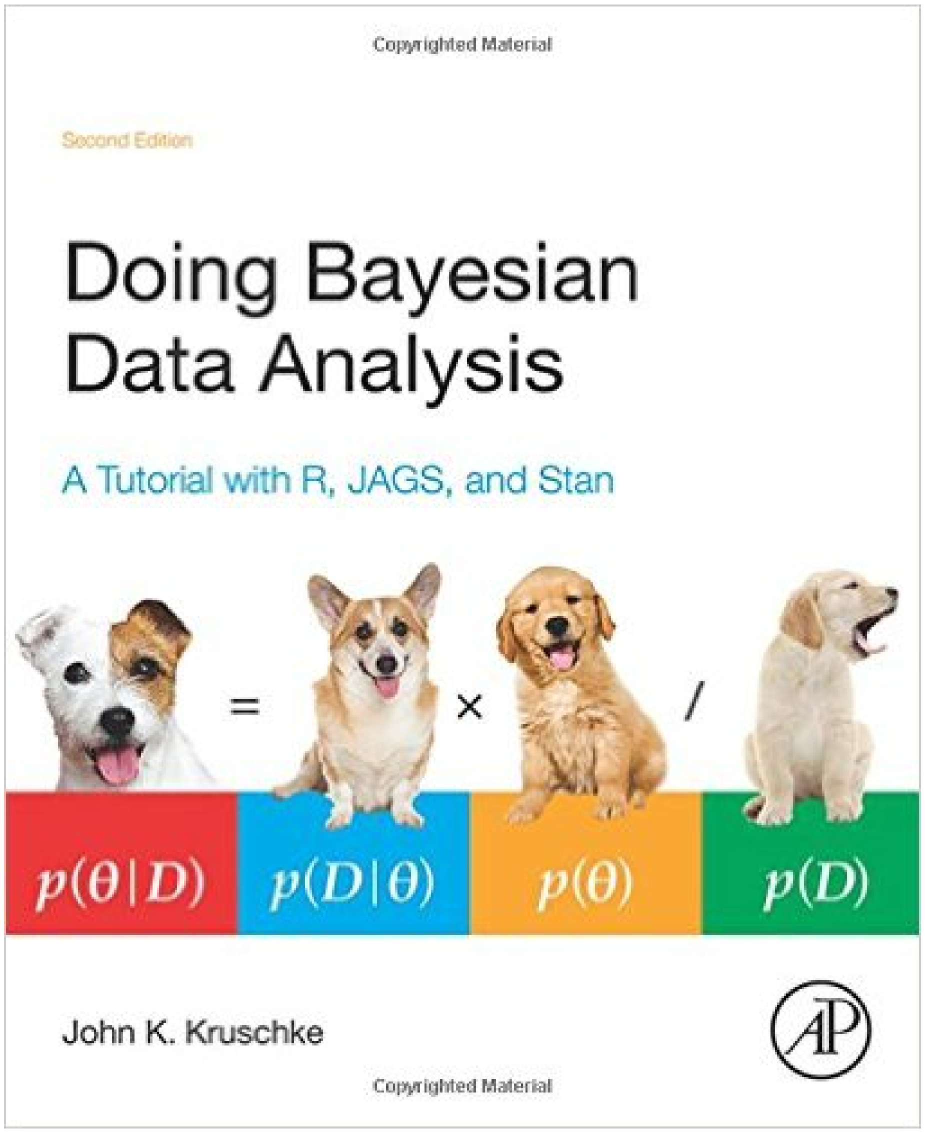
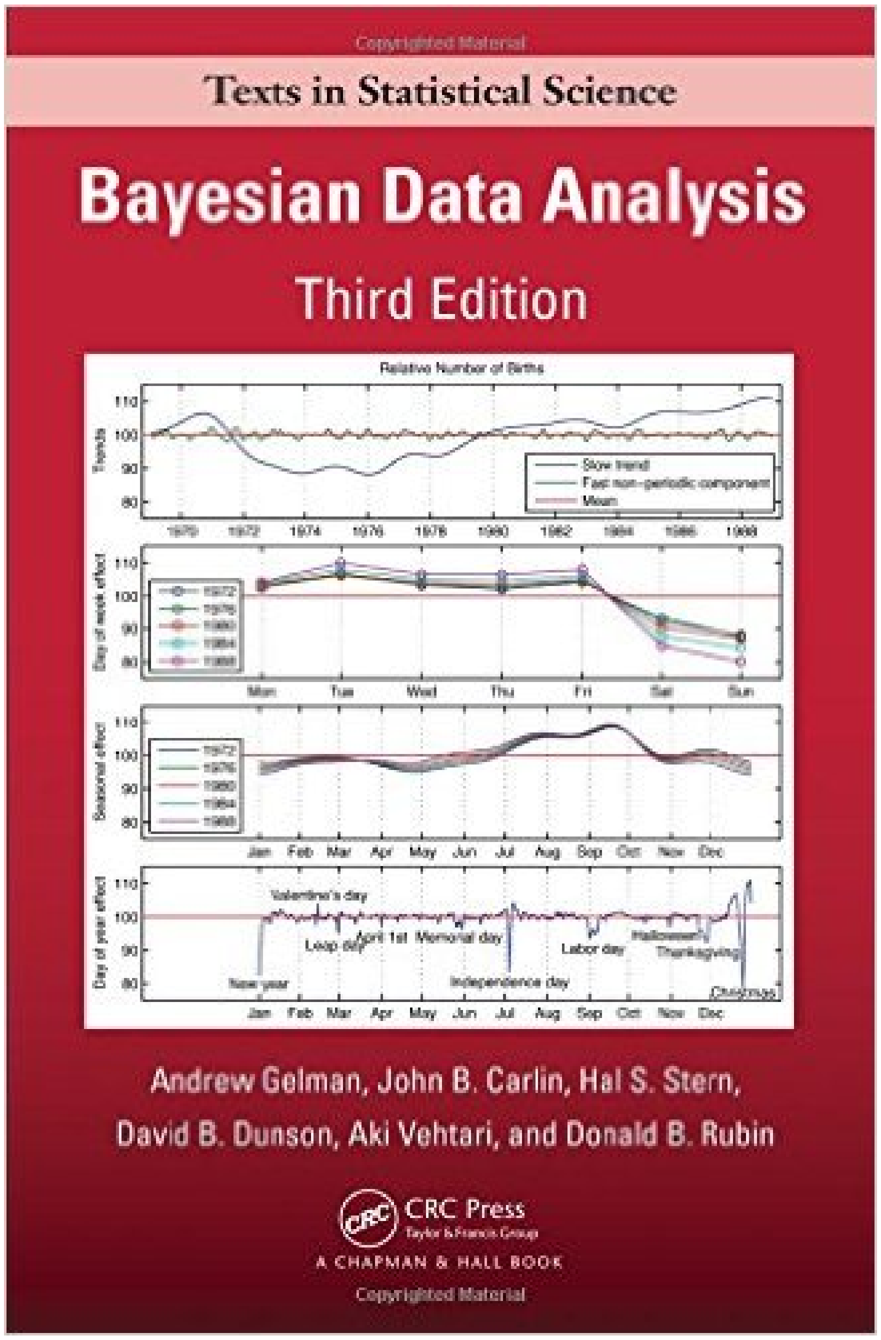


Figure 14: Doing Bayesian Data Analysis, by Kruschke



Texts in Statistical Science

Bayesian Data Analysis

Third Edition

Andrew Gelman, John B. Carlin, Hal S. Stern, David B. Dunson, Aki Vehtari, and Donald B. Rubin

CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK

Figure 15: Bayesian Data Analysis, by Gelman