

# Bayesian Regression

## Priors and Pooling - Revisited with Last Weeks Data

### Introduction

The objective is to examine the impact of pooling on a Bayesian regression models. We will develop 2 models: first with no pooling and next with partial pooling, followed by a discussion of results.

The data comes from the *Automobile Price Prediction.csv* dataset (*which you should be familiar with*). The data are nested by model, and because our purpose is to examine the effect of pooling, we will only use 2 variables:

- model (*the grouping variable*)
- horsepower (*the independent variable*)

### Full Pooling, No Pooling and Partial Pooling (using lmer)

The following uses `lm` to generate a fully pooled model, `lmlist` to generate no pooled models, and `lmer` to generate partially pooled models:

```
library(tidyverse)
library(rstan)
library(shinystan)

rmse <- function(error)
{
  sqrt(mean(error^2))
}

# ----- build mixed model - No Pooling ----- #

set.seed(103)

Autos <- read.csv("C:/Users/ellen/Documents/UH/Fall 2020/Class Materials/Section II/Pooling/Automobile Price Prediction.csv")
Autos <- rowid_to_column(Autos, var="SampleID") # this creates a primary key (you have to be careful with this)
by_MakeStyle <- Autos %>% group_by(make) %>% dplyr::mutate(cnt = n()) %>% filter(cnt > 1)
train <- sample_frac(by_MakeStyle, .6) %>% ungroup()
train$make <- factor(train$make)
train$makeID <- as.integer(train$make)
test <- anti_join(by_MakeStyle, train, by = "SampleID")

map <- unique(data.frame(val = train$make, makeID = as.integer(train$make)))

# convert make to Factor to work with Stan

library(lme4)
lmerMod <- lmer(price ~ 1 + horsepower + (1 + horsepower | makeID), train)
lmerCoef <- coef(lmerMod)$`make`
p_a <- lmerCoef[,1]
```

```

p_b <- lmerCoef[,2]

NoPoolCoef = lmList(price ~ horsepower | make, data = train, RMEL = FALSE) %>%
  coef() %>%
  rownames_to_column("Description")

NPCoef <- data.frame(Model = "NoPool",
  make = NoPoolCoef$Description,
  Intercept = NoPoolCoef$`(Intercept)`,
  Slope = NoPoolCoef$horsepower)

lmerCoef1 = data.frame(coef(lmerMod)$make) %>%
  rownames_to_column("Make")

BayesMapLM <- data.frame(Model = "lmer",
  make = unique(train$make),
  Intercept = lmerCoef$`(Intercept)`,
  Slope = lmerCoef$horsepower)

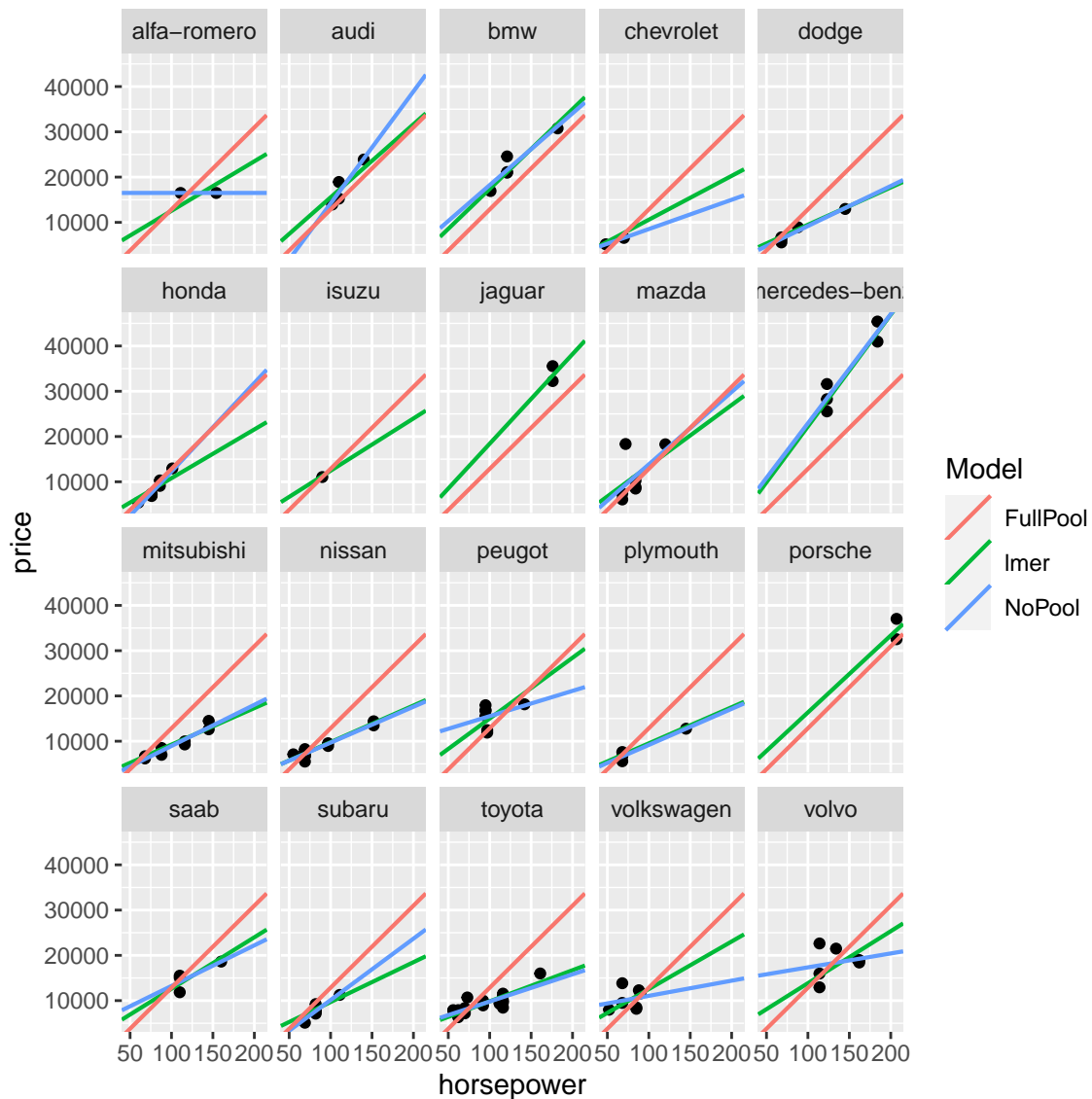
FPCoef = coef(lm(price ~ horsepower, data = train))

FPMaP = data.frame(Model = "FullPool",
  make = unique(train$make),
  Intercept = FPCoef[1],
  Slope = FPCoef[2])

BayesMap <- rbind(BayesMapLM, NPCoef, FPMaP)

p <- ggplot(data = train) +
  aes(x = horsepower, y = price) +
  geom_point() +
  geom_abline(data = BayesMap, aes(intercept = Intercept, slope = Slope, color = Model),
    size = .75) +
  facet_wrap("make")
p

```



Notice how many groups have no model for NoPool. Why?

## Model 2 - Partial Pooling

Now we'll use a Bayesian Model to tweak partial pooling

First let's use

```
p_aBU <- p_a
p_bBU <- p_b
#p_a <- rep(mean(p_a), length(p_a))
#p_b <- rep(mean(p_b), length(p_b))

p_a <- rep(FPcoef[1], length(p_a))
p_b <- rep(FPcoef[2], length(p_b))

stanModel12 <- '

```

```

data {
  int<lower=0> N;
  int<lower=0> J;
  vector[N] y;
  real x[N];
  int make[N];
  real p_a[J];
  real p_b[J];
  real p_aSigma;
  real p_bSigma;
}
parameters {
  real<lower = 0> sigma;
  vector[J] a;
  vector[J] b;
}
transformed parameters {
  vector[N] y_hat;
  for (i in 1:N)
    y_hat[i] = a[make[i]] + b[make[i]] * x[i];
}
model {
  target += normal_lpdf(y | y_hat, sigma);
  target += normal_lpdf(a | p_a, p_aSigma);
  target += normal_lpdf(b | p_b, p_bSigma);
}
'

stanData <- list(
  N=nrow(train),
  J=length(unique(train$make)),
  y=train$price,
  x=train$horsepower,
  make=train$makeID,
  p_a = p_a,
  p_b = p_b,
  p_aSigma = 1000,
  p_bSigma = 50
)

fit2 <- stan(model_code = stanModel2, data = stanData, refresh = 0)

sumFit2 <- data.frame(summary(fit2))

# build on this
Intercept2 <- summary(fit2, pars = c("a"), probs = c(0.1, 0.9))$summary
Slope2 <- summary(fit2, pars = c("b"), probs = c(0.1, 0.9))$summary

BayesMap2 <- data.frame(Model = "Bayes",
  make = unique(train$make),

```

```

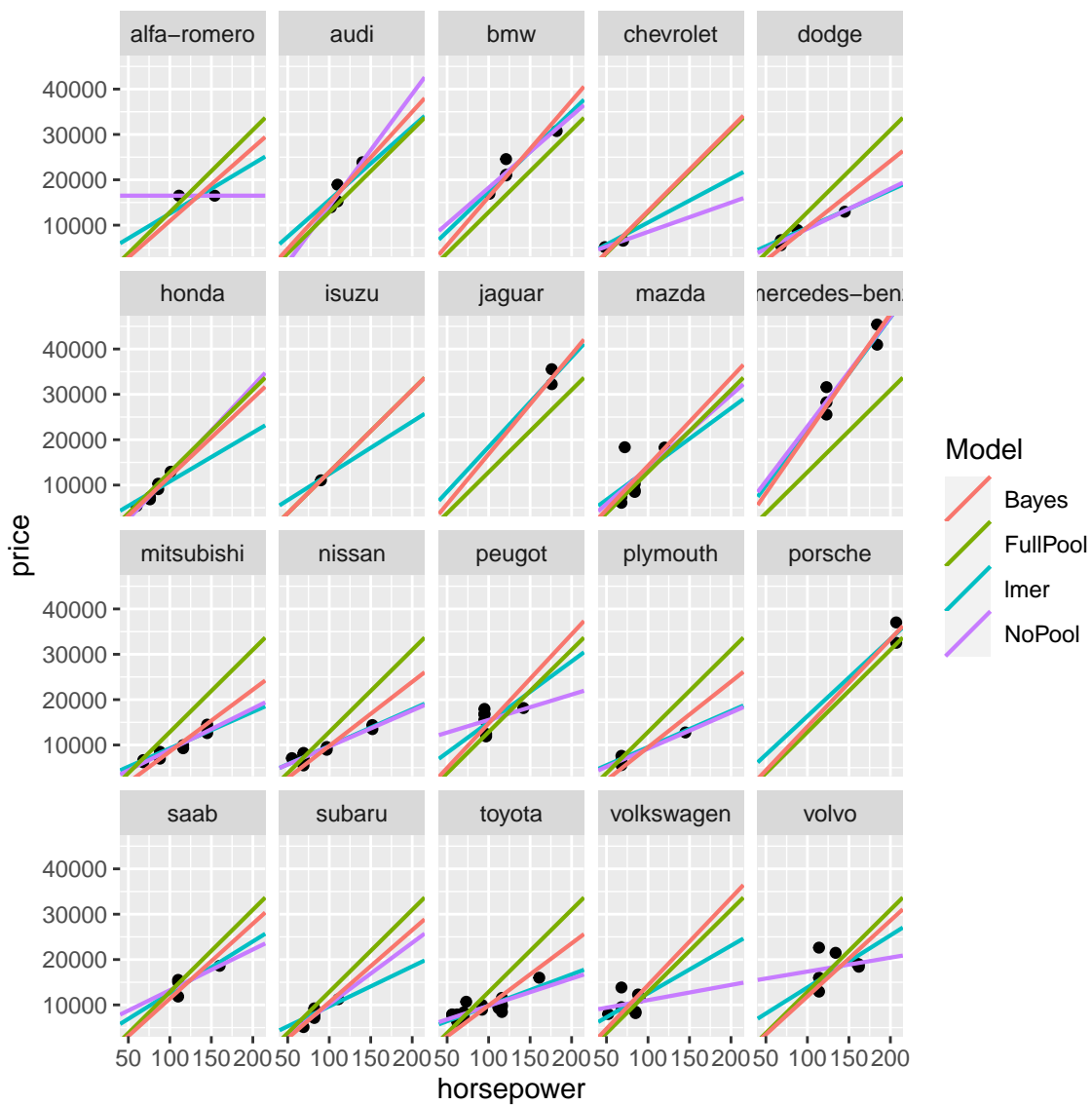
Intercept = Intercept2[,1],
Slope = Slope2[,1])

BayesMap <- rbind(BayesMap, BayesMap2)

#BayesMap = filter(BayesMap, Model != "Bayes")

p <- ggplot(data = train) +
  aes(x = horsepower, y = price) +
  geom_point() +
  geom_abline(data = BayesMap, aes(intercept = Intercept, slope = Slope, color = Model),
    size = .75) +
  facet_wrap("make")
p

```



Now, let's tighten up the priors a bit and see what that does to pooling:

```

stanData <- list(
  N=nrow(train),
  J=length(unique(train$make)),
  y=train$price,
  x=train$horsepower,
  make=train$makeID,
  p_a = p_a,
  p_b = p_b,
  p_aSigma = 10,
  p_bSigma = 1
)

fit3 <- stan(model_code = stanModel2, data = stanData, refresh = 0)

sumFit3 <- data.frame(summary(fit3))

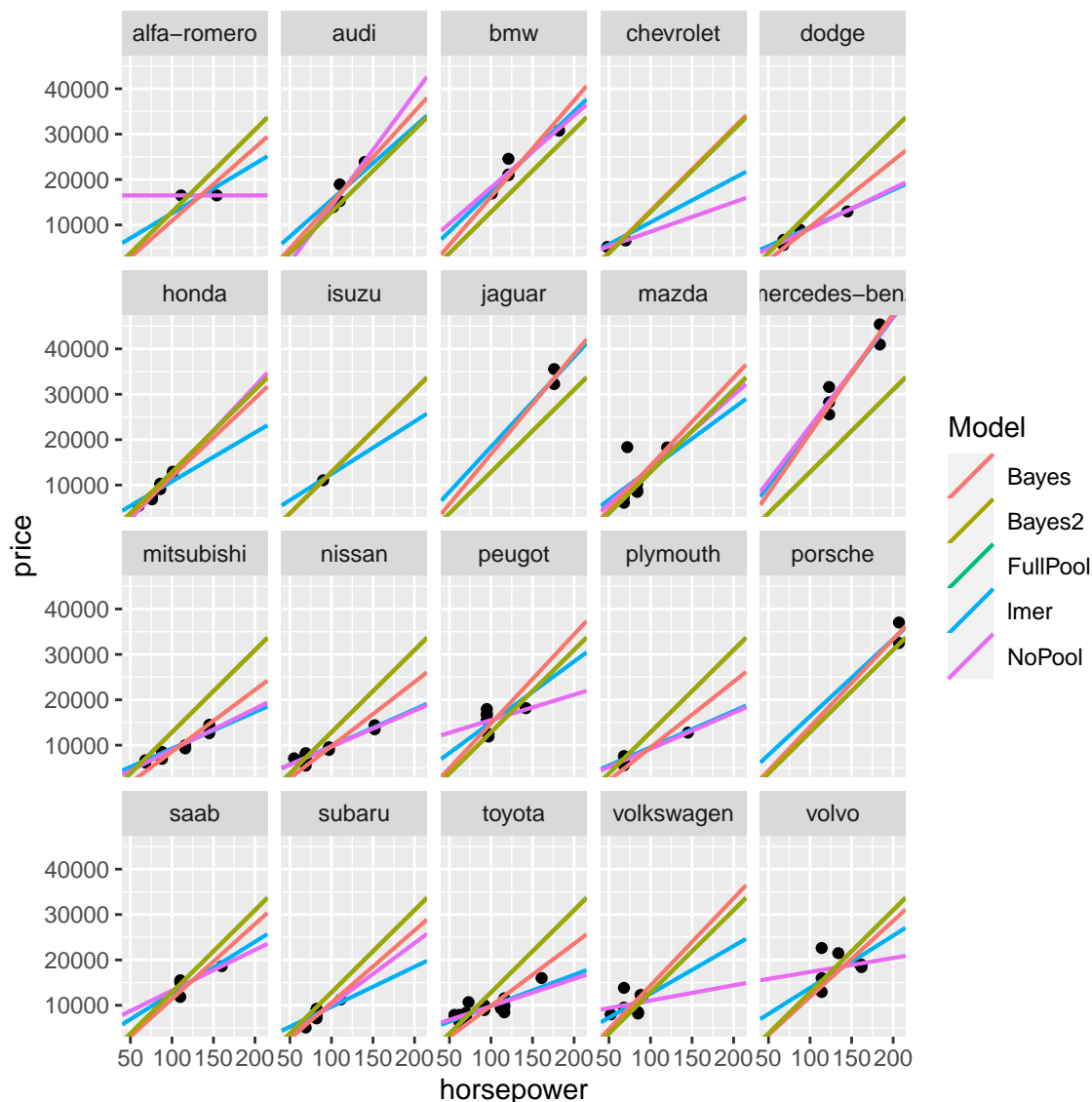
# build on this
Intercept3 <- summary(fit3, pars = c("a"), probs = c(0.1, 0.9))$summary
Slope3 <- summary(fit3, pars = c("b"), probs = c(0.1, 0.9))$summary

BayesMap3 <- data.frame(Model = "Bayes2",
  make = unique(train$make),
  Intercept = Intercept3[,1],
  Slope = Slope3[,1])

BayesMap <- rbind(BayesMap, BayesMap3)

p <- ggplot(data = train) +
  aes(x = horsepower, y = price) +
  geom_point() +
  geom_abline(data = BayesMap, aes(intercept = Intercept, slope = Slope, color = Model),
    size = .75) +
  facet_wrap("make")
p

```



Notice how the p-pool regression lines varies less across models (*notice how the slopes are more consistent*). Also notice how, in test data, the groups with less data has less impact on the partially pooled model.

## Analysis

There are many possible combinations of pooling and with most data, even a simple dataset like this one. The usage of Bayesian priors gives us great flexibility in controlling the effect of pools (*note that we can set a prior mean AND variance for EACH grouping*). To restate a few of the advantages:

- Crossed effects let us differentiate pricing between models (*a shopper expecting to buy a Mercedes based on an average of all models is going to be very disappointed*). So we have the ability to target expected values.
- Partial pooling lets us tune effects for each group - data tends to normalize inter-group, as well in intra-group and inter-group. In many cases, neither no-pooling nor complete pooling will be a good approach.
- Partial pooling lets us create predictions for groups that have little data (*a no pooled model will fail if there are few data points*)

- Generalization. Using nested models with priors gives us the ability to generalize models in a very targeted way - by group, by parameter. This level of control is just not possible with any other approach to modeling.