Bayesian Multilevel Modeling Exercise

Continuing with our exercise from last class. We want to build a Bayesian multi-level model. First, Load the following libraries:

```
library(tidyverse)
library(lme4)
library(rstan)
library(cowplot)
library(ggplot2)
library(lubridate)
library(ggridges)
library(kableExtra)
```

Then load the data from last class:

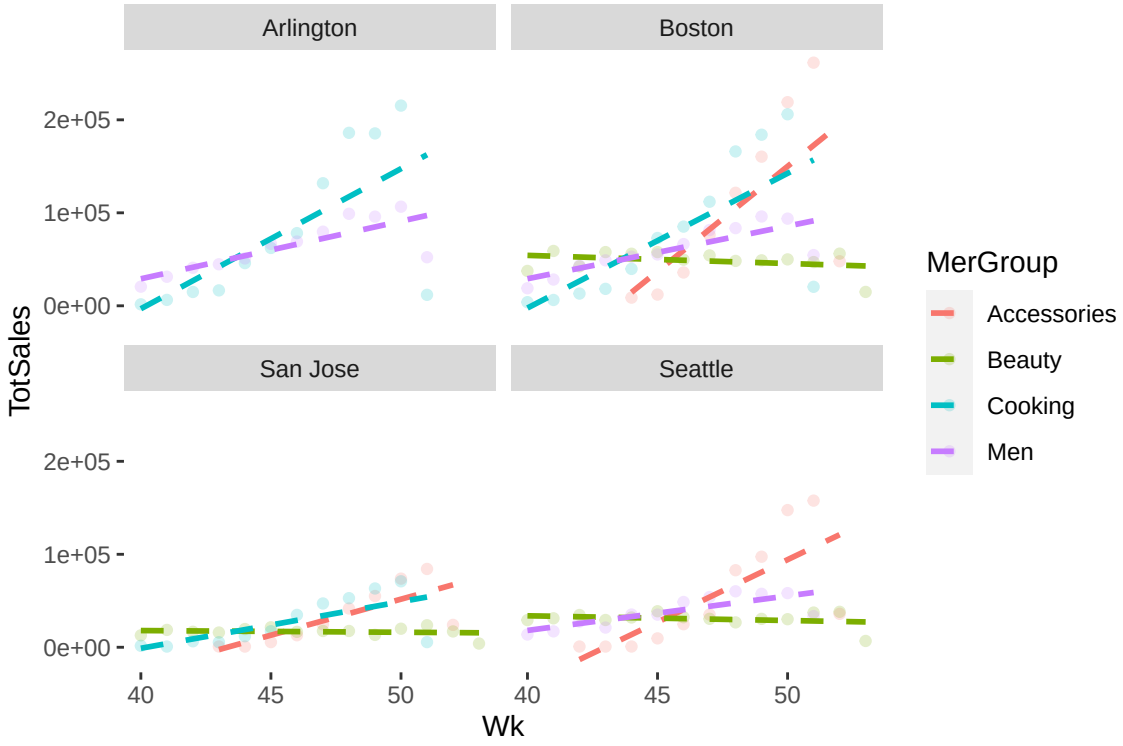
```
SalesTrans =
  read_csv("C:/Users/ellen/Documents/UH/Fall 2020/Class Materials/Section II/Class 1/Data/Sales.csv")
Location =
  read_csv("C:/Users/ellen/Documents/UH/Fall 2020/Class Materials/Section II/Class 1/Data/Location.csv")
MerGroup =
  read_csv("C:/Users/ellen/Documents/UH/Fall 2020/Class Materials/Section II/Class 1/Data/MerGroup.csv")
SalesTrans = SalesTrans %>% inner_join(Location, by = "LocationID")
SalesTrans = SalesTrans %>% inner_join(MerGroup, by = "MerGroup")
LocationID = as.factor(SalesTrans$LocationID)
SalesTrans$ProductID = as.factor(SalesTrans$ProductID)
SalesTrans$Description = as.factor(SalesTrans$Description)
SalesTrans$MerGroup = as.factor(SalesTrans$MerGroup)

# breaking out Q4 to simplify exercise

SalesTrans$Qtr = quarter(SalesTrans$Tdate)
SalesTrans = filter(SalesTrans, Qtr == 4)
SalesTransSummary = SalesTrans %>%
  group_by(Description, MerGroup, MfgPromo, Wk ) %>%
  summarise(Volume = n(), TotSales = sum(Amount) )
```

Now, let's filter it down to 3 Desciptions *(locations)* and 4 MerGroups so it's easier to see. We'll plot this out here - note that lm in ggplot does not generate models for level1/level2 groups where data does not exist, and it only generates independent models *(no pooling)*.

These are the locations and MerGroups we'll be analyzing the rest of the exercise:

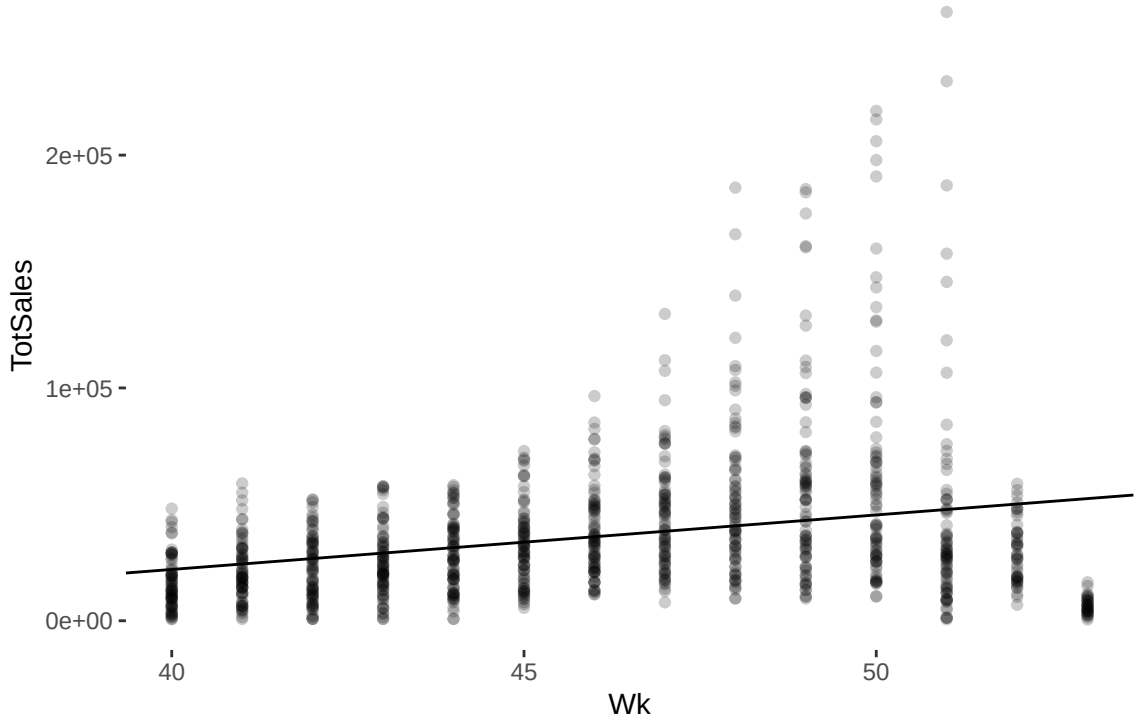
```
SalesSummarySub = filter(SalesTransSummary,
                      Description %in% c('Seattle', "Arlington",  "San Jose", "Boston"),
                      MerGroup %in% c("Accessories", "Beauty", "Cooking", "Men"))

p =  ggplot(SalesSummarySub, aes(Wk, TotSales, color = MerGroup)) +
  geom_point(alpha = .2) +
  geom_smooth(method = "lm", se = F, alpha = .05, linetype = "dashed", alpha = .5) +
  facet_wrap(~Description) +
```

```
  theme(panel.background = element_rect(fill = "white"))
p
```



But, we're going to build models based on the total population *(we need to understand how overall groups and effects behave first)*. So, let's back up, and to make this super obvious, we'll create a simple lm model *(full pooling)* for a baseline *(and we'll save the parameters for priors)*.

```
Priors = lm(TotSales ~ Wk , data = SalesTransSummary)
lmI = as.numeric(Priors$coefficients[1])
lmS = as.numeric(Priors$coefficients[2])

pt =  ggplot(SalesTransSummary, aes(Wk, TotSales)) +
  geom_point(alpha = .2) +
  geom_abline(aes(intercept=lmI, slope=lmS)) +
  theme(panel.background = element_rect(fill = "white"))
pt
```



Now, we'll take those priors and build a 2 level Bayesian model. First, we'll use a fully pooled lm model to force one intercept and one slope across all the levels and values *(we'll use rep to repeat the same value 10 times)*.

We're doing this just to see how the groups *(levels)* are constructed and controlled by priors and sigmas:

```
# build stan model
stanMod <- '

data {

  int<lower=0> N;
  vector[N] x; // Wk
  vector[N] y; // TotSales

  int<lower=0> J; // Description
  int Description[N];

  int<lower=0> K; // MerGroup
  int MerGroup[N];

  vector[J] p_alpha;
  vector[K] p_alpha2;

  real<lower=0> p_alphaSigma;
  real<lower=0> p_alphaSigma2;

  vector[J] p_beta;
  vector[K] p_beta2;
```

```
  real<lower=0> p_betaSigma;
  real<lower=0> p_betaSigma2;

}

parameters {

  // random effects we found
  vector[J] alpha; // intercept for Description
  vector[K] alpha2; // intercept for MerGroup

  vector[J] beta; // slope for Description
  vector[K] beta2; // slope for MerGroup

  real<lower=0> sigma; // to control sigma of y_hat

}

transformed parameters {

  vector[N] y_hat;
  for(i in 1:N)
  y_hat[i]=alpha[Description[i]]+alpha2[MerGroup[i]]+
  (beta[Description[i]]*x[i])+(beta2[MerGroup[i]]*x[i]);
}

model {

  target += normal_lpdf(alpha | p_alpha, p_alphaSigma);
  target += normal_lpdf(alpha2 | p_alpha2, p_alphaSigma2);

  target += normal_lpdf(beta | p_beta, p_betaSigma);
  target += normal_lpdf(beta2 | p_beta2, p_betaSigma2);

  target += normal_lpdf(sigma | 50, 50);

  // y_hat: our prediction
  target += normal_lpdf(y | y_hat, sigma);
}
'

fit <- stan(model_code = stanMod,  data = list(
  N = nrow(SalesTransSummary),
  y = SalesTransSummary$TotSales,
  x = SalesTransSummary$Wk,
  J = length(unique(SalesTransSummary$Description)),
  K = length(unique(SalesTransSummary$MerGroup)),
  Description = as.numeric(SalesTransSummary$Description),
  MerGroup = as.numeric(SalesTransSummary$MerGroup),

  p_alpha = rep(lmI,10), # Description intercept
  p_alpha2 = rep(lmI,10), # MerGroup intercept
```

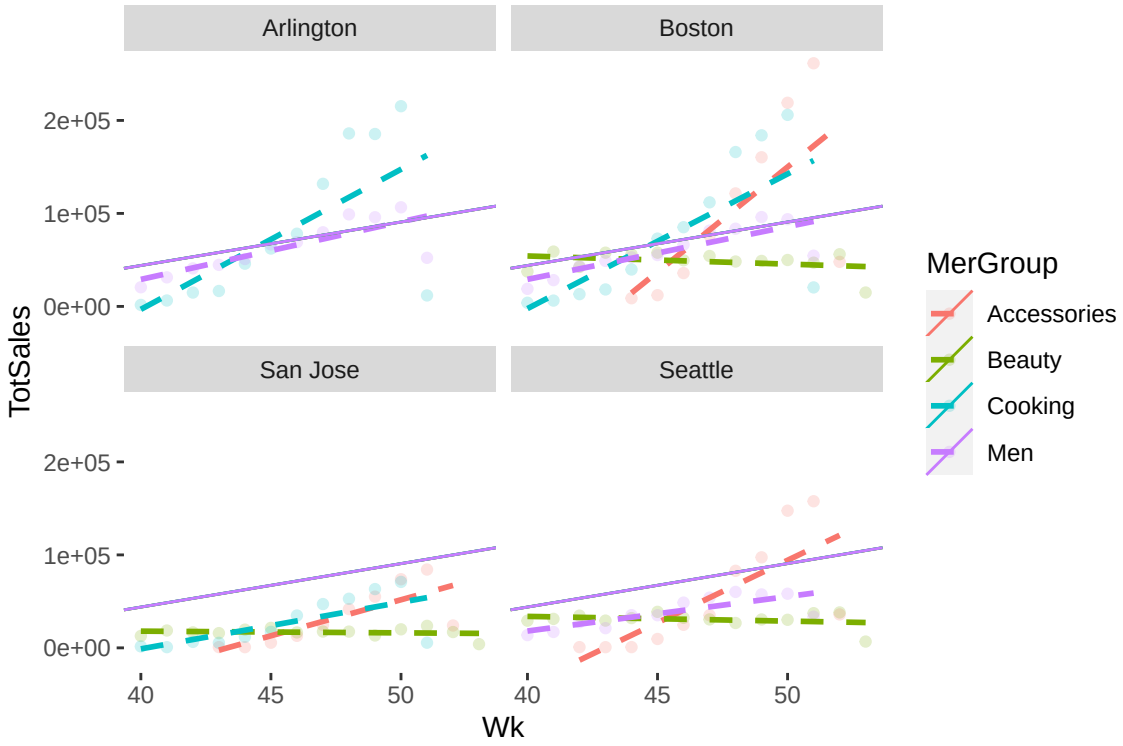
```
  #
  p_alphaSigma = 10,
  p_alphaSigma2 = 10,

  p_beta = rep(lmS,10), # Description slope (lmer model)
  p_beta2 = rep(lmS,10), # MerGroup slope (lmer model)

  p_betaSigma = 1,
  p_betaSigma2 = 1

), refresh = 0)
```

Now, we'll extract the parameters, put them in a dataframe that we can use for plotting, and narrow the visualization down to the subgroup we began with:

```
# Extract coefficients from stan model
sumFit <- data.frame(summary(fit))

# Description
Intercept1 <- summary(fit, pars = c("alpha"), probs = c(0.1, 0.9))$summary
# MerGroup
Intercept2 <- summary(fit, pars = c("alpha2"), probs = c(0.1, 0.9))$summary

# Description slope
Slope1 <- summary(fit, pars = c("beta"), probs = c(0.1, 0.9))$summary
# MerGroup slope
Slope2 <- summary(fit, pars = c("beta2"), probs = c(0.1, 0.9))$summary

# random effects for each Description
CoefMapD <- data.frame(Description = unique(SalesTransSummary$Description),
                       DIntercept = Intercept1[,1], DSlope = Slope1[,1])
# random effects for each MerGroup
CoefMapM <- data.frame(MerGroup = unique(SalesTransSummary$MerGroup), MIntercept = Intercept2[,1], MSlop

# add all intercepts and all slopes cross each row
CoefMapB = crossing(CoefMapD, CoefMapM) %>%
  mutate(Intercept = DIntercept + MIntercept, Slope = DSlope + MSlope)

CoefMapBSub <- filter(CoefMapB,
                      Description %in% c('Seattle', "Arlington",  "San Jose", "Boston"),
                      MerGroup %in% c("Accessories", "Beauty", "Cooking", "Men"))
```

Taking a look at the models on a plot:

```
p1 = p +
  geom_abline(data = CoefMapBSub,
                       aes(intercept=Intercept, slope=Slope, color=MerGroup))
p1
```



Note that our Bayesian model gives us parameter *effects* for every location and merch group *(see dataframe CoefMapBSub)*, even where no data exists *(e.g., Arlington data had not Accessories or Beauty products). (you can't see it on the plot here, because all the models overlap because we severely limited the priors and sigmas for learning purposes)* This is a BIG deal - how would you forecast Accessories for Arlington if Nordies was planning on carrying that group next year? We can do this becuase the model quantifies the *EFFECTS* of each location and the *EFFECTS* of each merch group. Then we just add the effect to the base to get the prosterior for a location, merch group of interest.

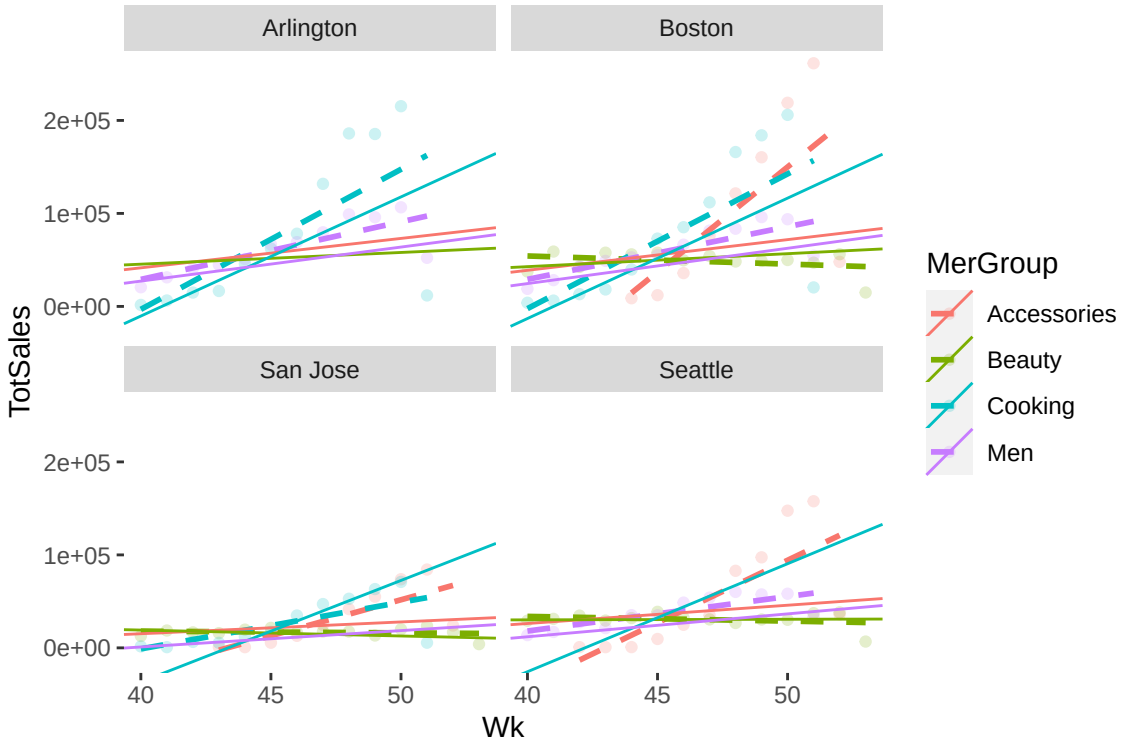
In this example however, we have unrealistically limited effects. So let's open this up a bit by using the last model parameters as priors and increasing sigma values, allowing the sampler to weight the liklihood data more:

```
fit <- stan(model_code = stanMod,  data = list(
  N = nrow(SalesTransSummary),
  y = SalesTransSummary$TotSales,
  x = SalesTransSummary$Wk,
  J = length(unique(SalesTransSummary$Description)),
  K = length(unique(SalesTransSummary$MerGroup)),
  Description = as.numeric(SalesTransSummary$Description),
  MerGroup = as.numeric(SalesTransSummary$MerGroup),

  p_alpha = as.numeric(CoefMapD[,2]),
  p_alpha2 = as.numeric(CoefMapM[,2]),

  #
  p_alphaSigma = 20000,
  p_alphaSigma2 = 20000,

  p_beta = as.numeric(CoefMapD[,3]),
  p_beta2 = as.numeric(CoefMapM[,3]),
```

```
  p_betaSigma = 1000,
  p_betaSigma2 = 1000

), refresh = 0)


# Extract coefficients from stan model
sumFit <- data.frame(summary(fit))

# Description
Intercept1 <- summary(fit, pars = c("alpha"), probs = c(0.1, 0.9))$summary
# MerGroup
Intercept2 <- summary(fit, pars = c("alpha2"), probs = c(0.1, 0.9))$summary

# Description slope
Slope1 <- summary(fit, pars = c("beta"), probs = c(0.1, 0.9))$summary
# MerGroup slope
Slope2 <- summary(fit, pars = c("beta2"), probs = c(0.1, 0.9))$summary

# random effects for each Description
CoefMapD2 <- data.frame(Description = unique(SalesTransSummary$Description),
                        DIntercept = Intercept1[,1], DSlope = Slope1[,1])
# random effects for each MerGroup
CoefMapM2 <- data.frame(MerGroup = unique(SalesTransSummary$MerGroup), MIntercept = Intercept2[,1], MSl

# add all intercepts and all slopes cross each row
CoefMapB2 = crossing(CoefMapD2, CoefMapM2) %>%
  mutate(Intercept = DIntercept + MIntercept, Slope = DSlope + MSlope)

CoefMapBSub2 <- filter(CoefMapB2,
                       Description %in% c('Seattle', "Arlington",  "San Jose", "Boston"),
                       MerGroup %in% c("Accessories", "Beauty", "Cooking", "Men"))
```

Notice how the models have extended to meet the likelihood data:

```
p2 <- p + geom_abline(data = CoefMapBSub2,
                      aes(intercept=Intercept, slope=Slope, color=MerGroup))
p2
```



Concept Review:

We created 100 models, each with different parameter combinations. These are not independent models - they share covariance with group members (i.e., Sales in Seattle move together, and Sales of Beauty move together). So, if I need to forecast Beauty sales in Arlington, I just add the Beauty parameter to the Arlington group. It won't be the same as adding the Beauty parameter to another location. This is very powerful for analysis and forecasting.

We control the models through priors and sigmas. Sigmas tell us how much confidence we have in our priors. If I had known that a major competitor would be opening a store across the street, I might reduce the location parameter and squeeze the sigma. This is how we integrate experience and judgment into models. Again, this is very powerful for analysis and forecasting.

Note how we managed parameters in dataframes. You can do this anyway you like, but the parameters is what you do all this work for, so you want to secure and manage the data. These parameters are why the models are descriptive, explainable, while also being flexible.