DA2 Regression Foundations - Multi-Level Modeling

Multi-Level Modeling

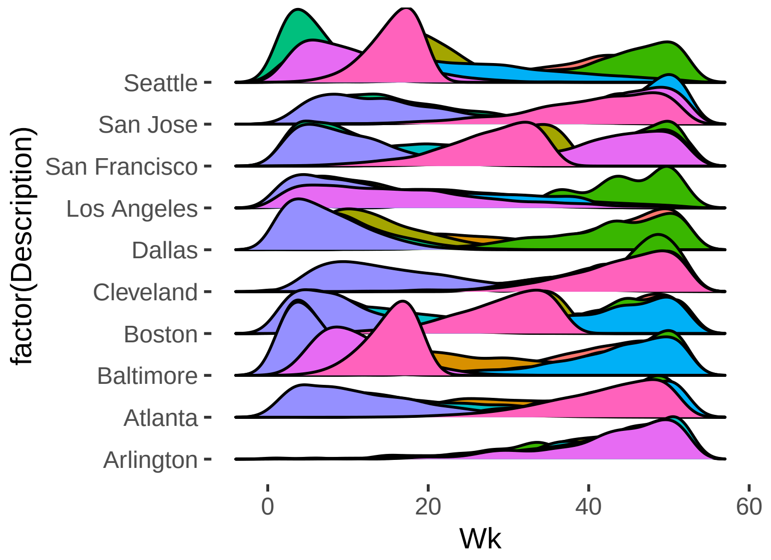
Introduction and Scope

As we finalize our review of regression and logistic regression in preparation for study of Bayesian analysis, this exercise introduces multilevel modeling, which is a core competency of professional analysts or auditors.

Recall that we began our study of Analytics with the dataset that represented a year of retail sales for 10 merchandise categories accross 10 national locations. Data acquisition and visualization is produced below:

```r
SalesTrans = dbGetQuery(con2,"
SELECT
      [MODELING].[SalesTrans].[TransID]
      ,[MODELING].[SalesTrans].[LocationID]
      ,[MODELING].[SalesTrans].[ProductID]
      ,[MODELING].[SalesTrans].[Tdate]
      ,[MODELING].[SalesTrans].[Amount]
  FROM [MODELING].[SalesTrans]
    ")

Product = dbGetQuery(con2,"
SELECT
  [MODELING].[Product].[ProductID]
  ,[MODELING].[Product].[MerGroup]
  ,[MODELING].[Product].[MfgPromo]
  FROM [MODELING].[Product]
    ")

Location = dbGetQuery(con2,"
SELECT
  [MODELING].[Location].[LocationID]
  ,[MODELING].[Location].[Description]
  ,[MODELING].[Location].[Population]
  ,[MODELING].[Location].[Income]
  FROM [MODELING].[Location]
      ")
# Transform Dates and add Weeks

SalesTrans$Tdate <- as_date(SalesTrans$Tdate)
SalesTrans$Wk = week(SalesTrans$Tdate)

SalesTrans = SalesTrans %>%
  inner_join(Product, by = "ProductID")  %>%
  inner_join (Location, by = "LocationID")
```

```r
ggplot(data.frame(SalesTrans),
       aes(y = factor(Description), x = Wk, fill = factor(ProductID))) +
  geom_density_ridges()  +
  guides(fill=FALSE) +
  theme(panel.background = element_rect(fill = "white"))
```



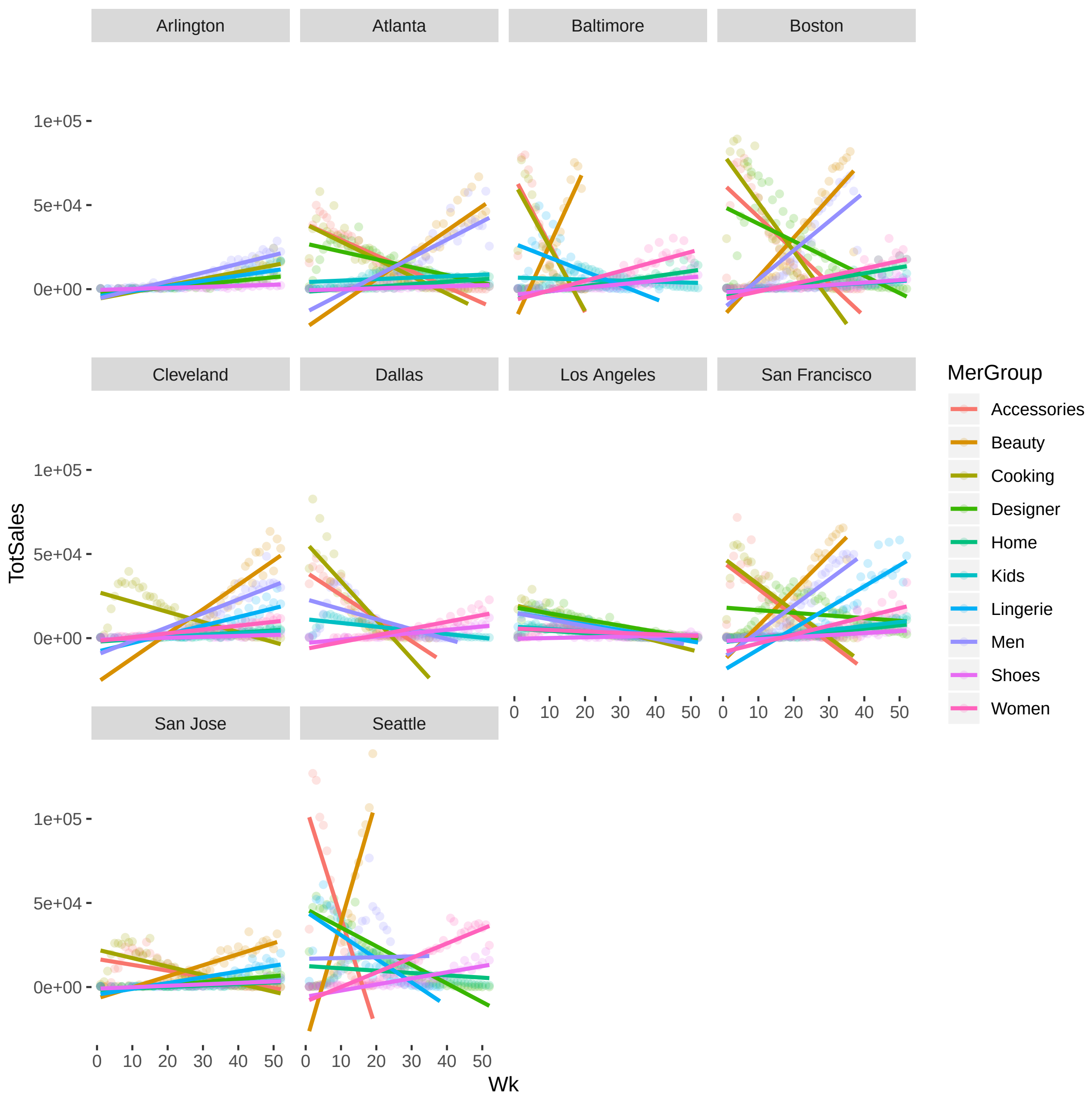
The distributions above reflect sales volume by location with fill for merchandising groups. Let's summarize the data first:

```r
SalesTransSummary = SalesTrans %>%
  group_by(Description, MerGroup, MfgPromo, Wk = week(Tdate)) %>%
  summarise(Volume = n(), TotSales = sum(Amount) )
```

Independent Linear Models

Now, lets create our first series of multilevel models. We'll use ggplot and create groups for MerGroup *(defining color will force ggplot to create a group)*, and Locations *(Description - defining a facet will also force ggplot to create groups)*:

```r
p =  ggplot(SalesTransSummary, aes(Wk, TotSales, color = MerGroup)) +
  geom_point(alpha = .2) +
  geom_smooth(method = "lm", se = F) +
  facet_wrap(~Description) +
  theme(panel.background = element_rect(fill = "white"))
p
```



These models are truly independent - ggplot creates a separate model for each Description and MerGroup. So, in this case, 100 sepearte models! You could eliminate all the bias error by creating 100 decision trees or suppor vector machines - but, will that really yield a useful model? Look at Accessories and Beauty in Seattle - are you willing to bet on those trends? Would you use them to forecast next year?

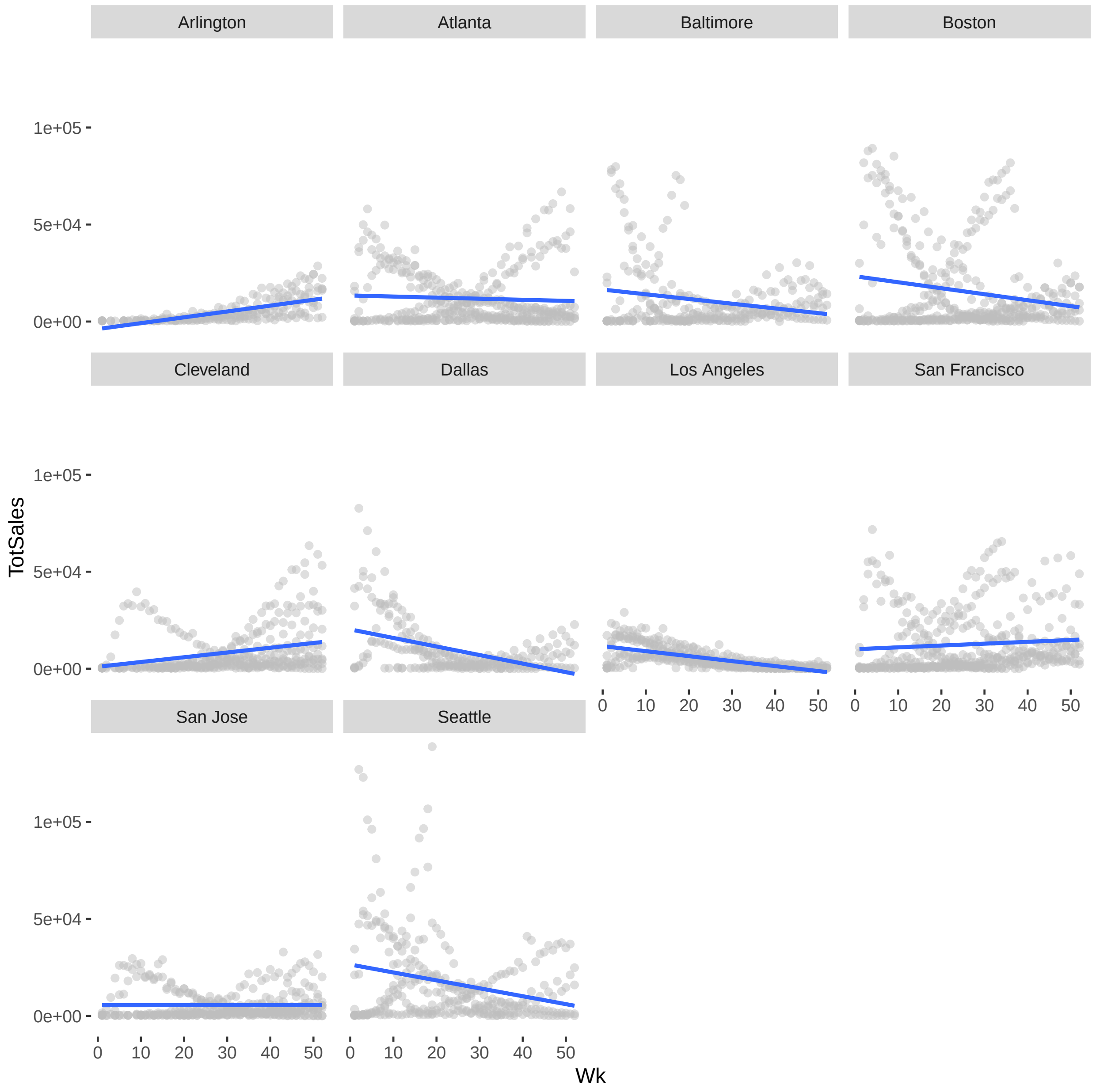
These models are**Not Pooled** - i.e., they do not share correlation between groups.

This should give you a feel for the scope and complexity of the problem space. Now, let's step back and look at how we could apply a simpler modeling approach and build from there.

Independent Single Level Models

We'll start with with ggplot models again - but this time with a single grouping level. let's generate a set of 10 models based on Description:

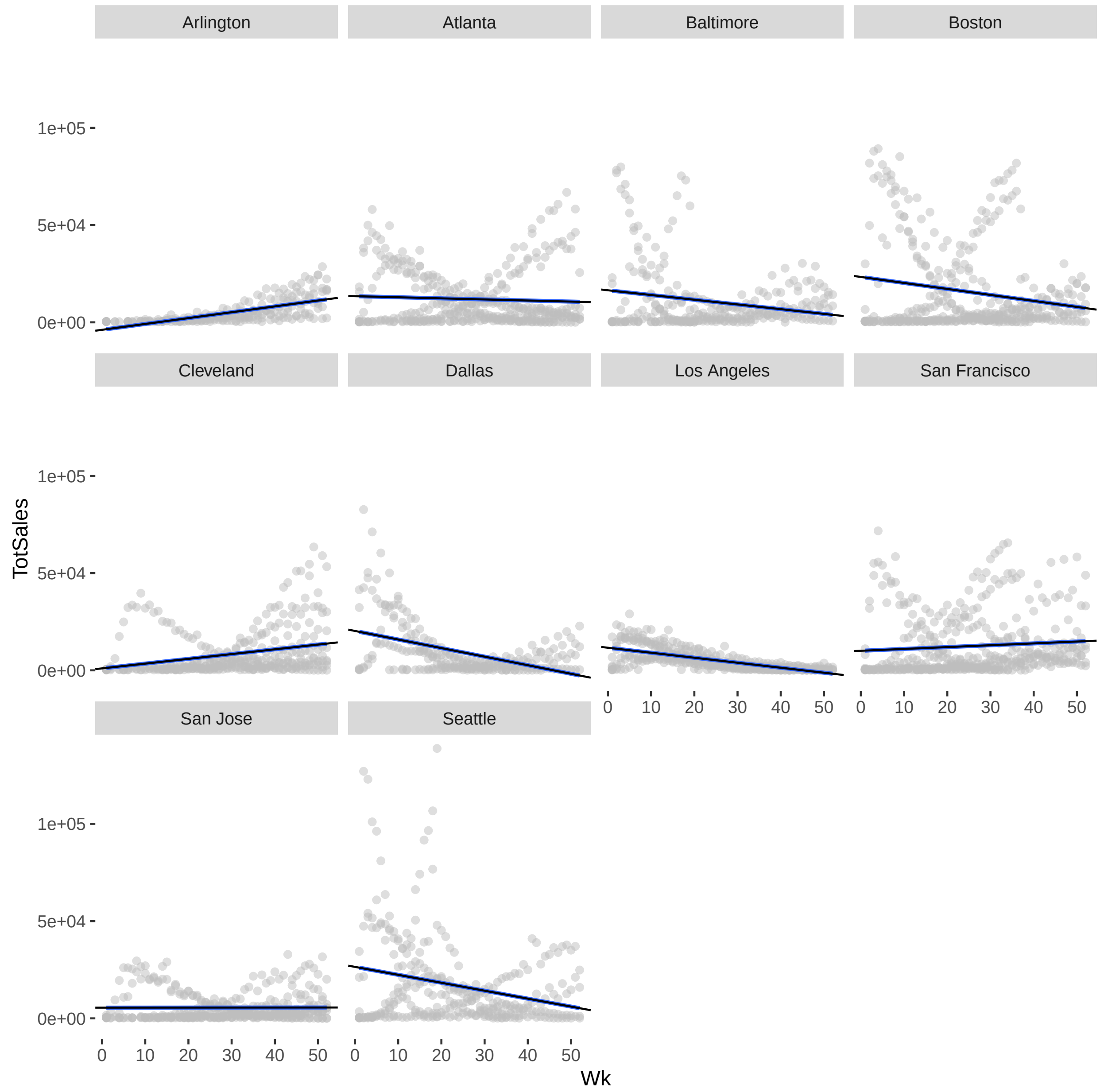
```r
p =  ggplot(SalesTransSummary, aes(Wk, TotSales)) +
  geom_point(color = "gray", alpha = .5) +
  geom_smooth(method = "lm", se = F) +
  facet_wrap(~Description) +
  theme(panel.background = element_rect(fill = "white"))
p
```



Notice again that these models still have different slopes and intercepts. And correlation is not shared *(so, there's no pooling)*.

We can create all these models usng lmList *(from the lme4 package which provides multilevel and random effects modeling)*. The code below will create a series of models based on one grouping variable. Then we can just pass the coefficients to abline and plot them out *(pretty handy)*:

```r
NoPoolCoef = lmList(TotSales ~ Wk | Description, data = SalesTransSummary, RMEL = FALSE) %>%
  coef() %>%
  rownames_to_column("Description")

p = p + geom_abline(data = NoPoolCoef, aes(intercept = `(Intercept)`, slope =  Wk))
p
```



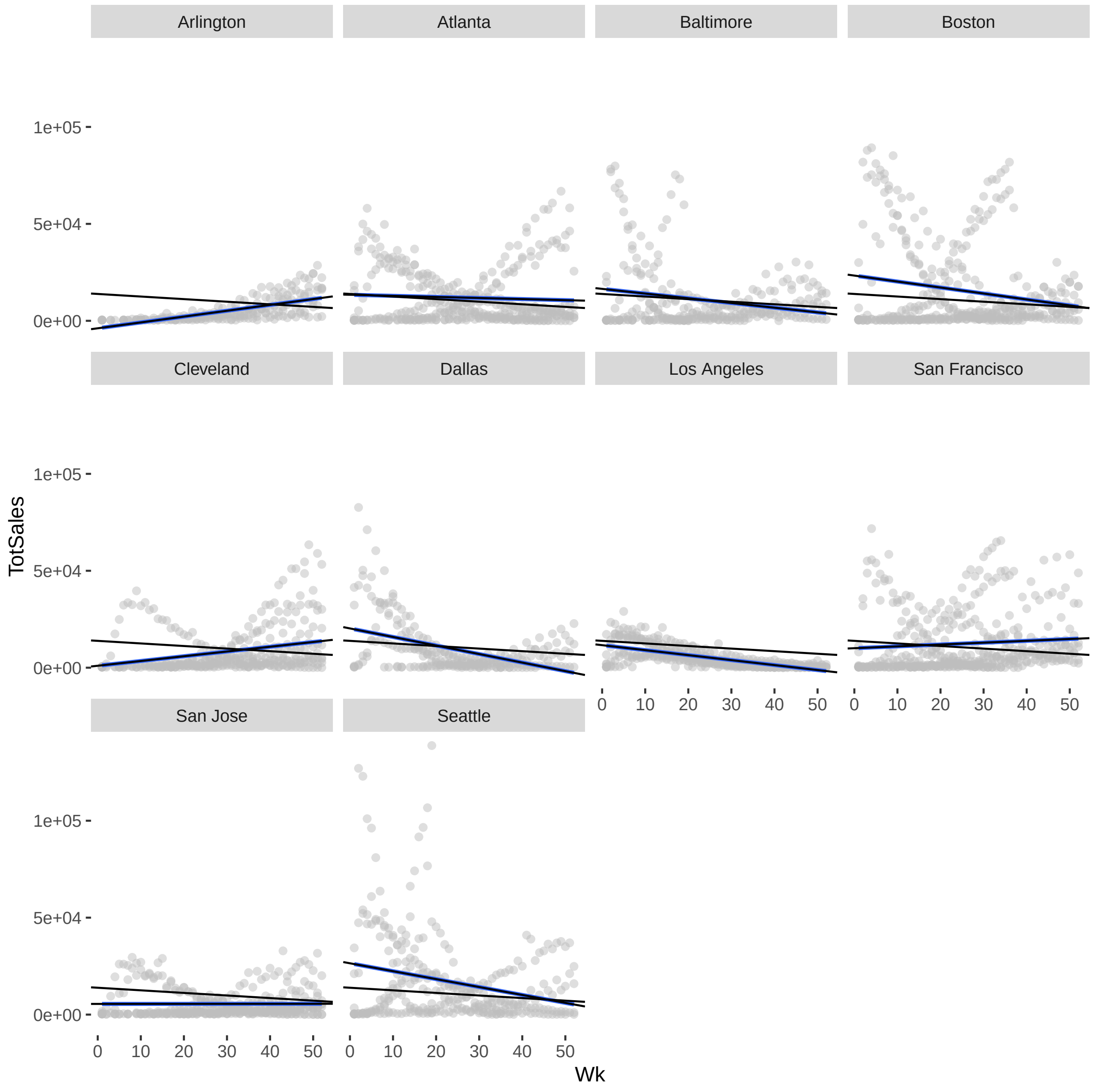
You don't see 2 sets of lines because lmer gets the same coefficients as lm here. *(lmer grouping syntax at the end)*.

Fully Pooled Linear Models

Fully Pooled models don't change with groups - they're fit using all the data and ignore correlation btween groups. Simple, but don't discount them - as you've learned, variability is low and explainablity is high *(i.e.,* *they're useful and maintainable)*. Variability is BIG DEAL in business - environments are hightly dynamic and volume / compleixty can be overwhelming, so Analysts tend to graviate to models with bias, or they spend a lot of time on parameter tuning and generalization *(as we will soon see, Bayesian analysis offers a very sophiticated generalization approach through priors)*

The code below creates a fully pooled model:

```
fpLMod = lm(TotSales ~ Wk, SalesTransSummary )
fpLMCoef = coef(fpLMod)

p = p + geom_abline(aes(intercept = fpLMCoef[1], slope = fpLMCoef[2]))
p
```

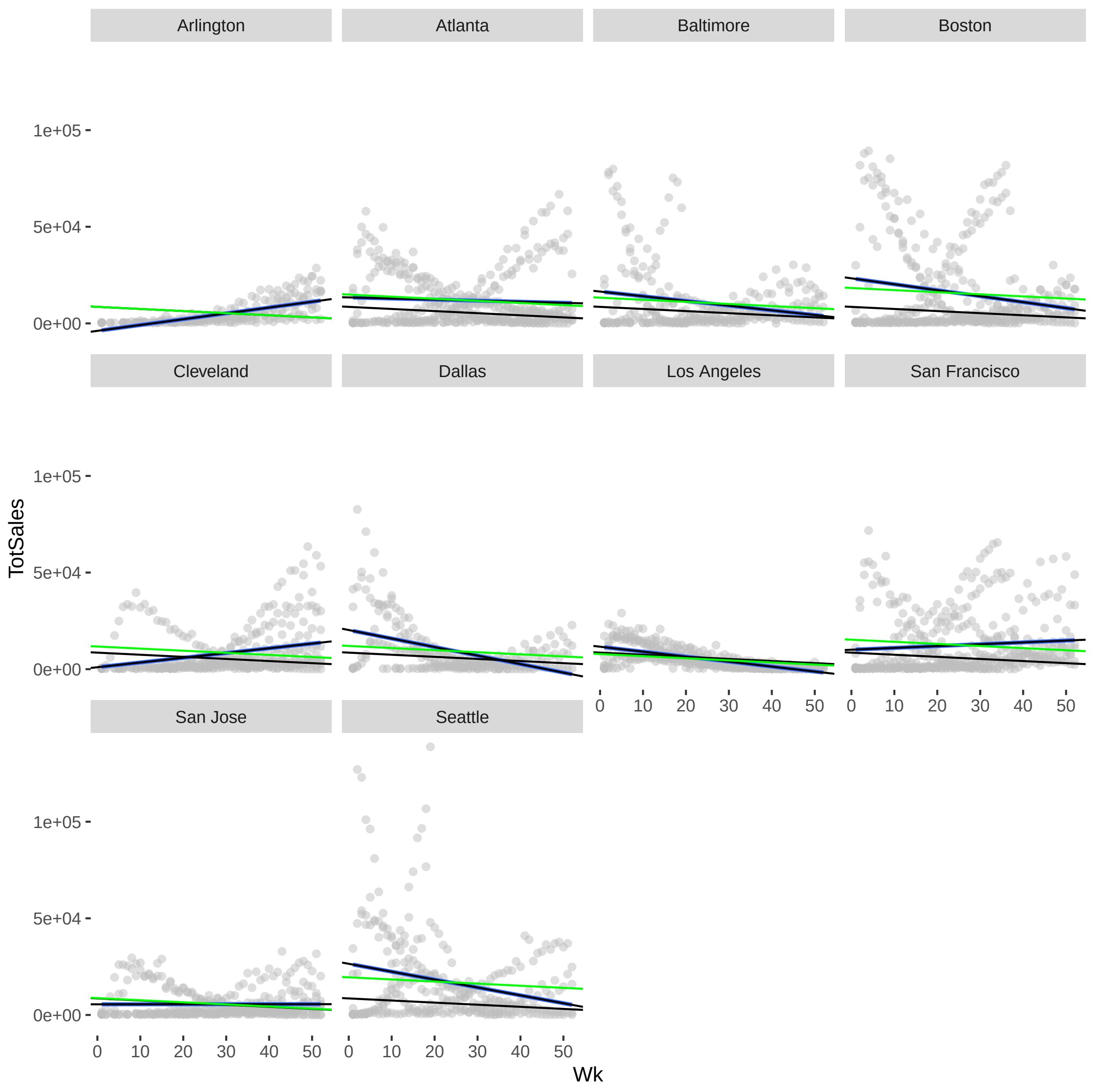


The plot above compares the fully pooled model with the independent, single group level model. Note how the fully pooled model doesn't change.

Varying Intercept Models

These are the models you're most familar with - as we spent a lot of time on categorical regression last semester *(recall the Auto price data, where each make of car was assigned a different intercept value while they all shared the same slope)*. So we won't spend a lot of time on this model other than to compare with the independent and fully pooled models:

```
LMVarIntMod = lm(TotSales ~ Wk + Description, SalesTransSummary )
fpLMCoef = coef(LMVarIntMod)



LMVarIntCoef = data.frame(Description = unique(SalesTransSummary$Description),
                          fpLMI = c( coef(LMVarIntMod)[1], coef(LMVarIntMod)[1] + c(coef(LMVarIntMod)[3
                          fpLMS = c( rep(coef(LMVarIntMod)[2], 10)))

p = p + geom_abline(data = LMVarIntCoef, aes(intercept = fpLMI, slope =  fpLMS), color = "green")
p
```

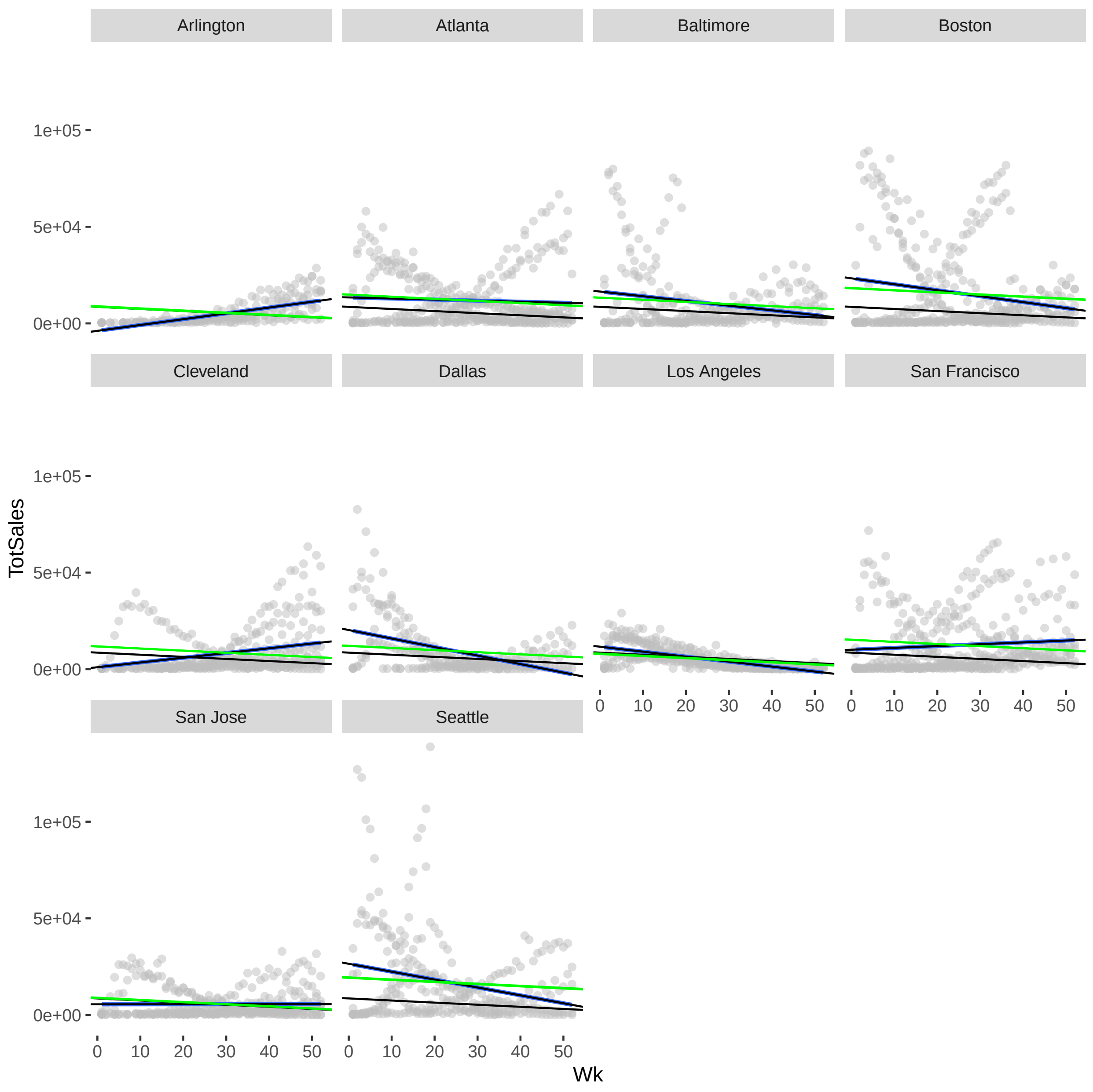


Also note that we can use glm to fit linear models instead of lm - using maximum likelihood for finding parameters where ols is insufficient *(lmer also uses maximum likelihood and derivative based parameter determination)*:

```
# BTW, you also can use glm without the family to fit a linear equation using ML instead of LS

GLMVarIntMod = glm(TotSales ~ Wk + Description, SalesTransSummary, family = gaussian() )
GLMVarIntCoef = coef(GLMVarIntMod)
GLMVarIntCoef = data.frame(Description = unique(SalesTransSummary$Description),
                           fpLMI = c( coef(LMVarIntMod)[1], coef(LMVarIntMod)[1] + c(coef(LMVarIntMod)[3
                           fpLMS = c( rep(coef(LMVarIntMod)[2], 10)))
```

And we can use lmer to do the same. This is how you create a varying intercept model in lmer:

```
LMERVarIntMod = lmer(TotSales ~ 1 + Wk + ( 1 | Description),
                     data = SalesTransSummary,
                     REML = FALSE)

#summary(LMERVarIntMod)
#fixef(LMERVarIntMod)
#ranef(LMERVarIntMod)$Description[,1]
# lmer breaks out the fixed effects and the "random effects"
# so the get the coefficients, we can add together fixed and random effects

#fixef(LMERVarIntMod)[1] + c(ranef(LMERVarIntMod)$Description[,1])
# or we can just ask for the coeffecients:

#coef(LMERVarIntMod)

# agrees to fixed + random above
# so

LMERVarIntCoef = data.frame(coef(LMERVarIntMod)$Description) %>%
  rownames_to_column("Description") %>%
  rename(Intercept = `X.Intercept.`, Slope = Wk)

p = p + geom_abline(data = LMERVarIntCoef,
                    aes(intercept = LMERVarIntCoef$Intercept, slope =  LMERVarIntCoef$Slope), color = "
p
```

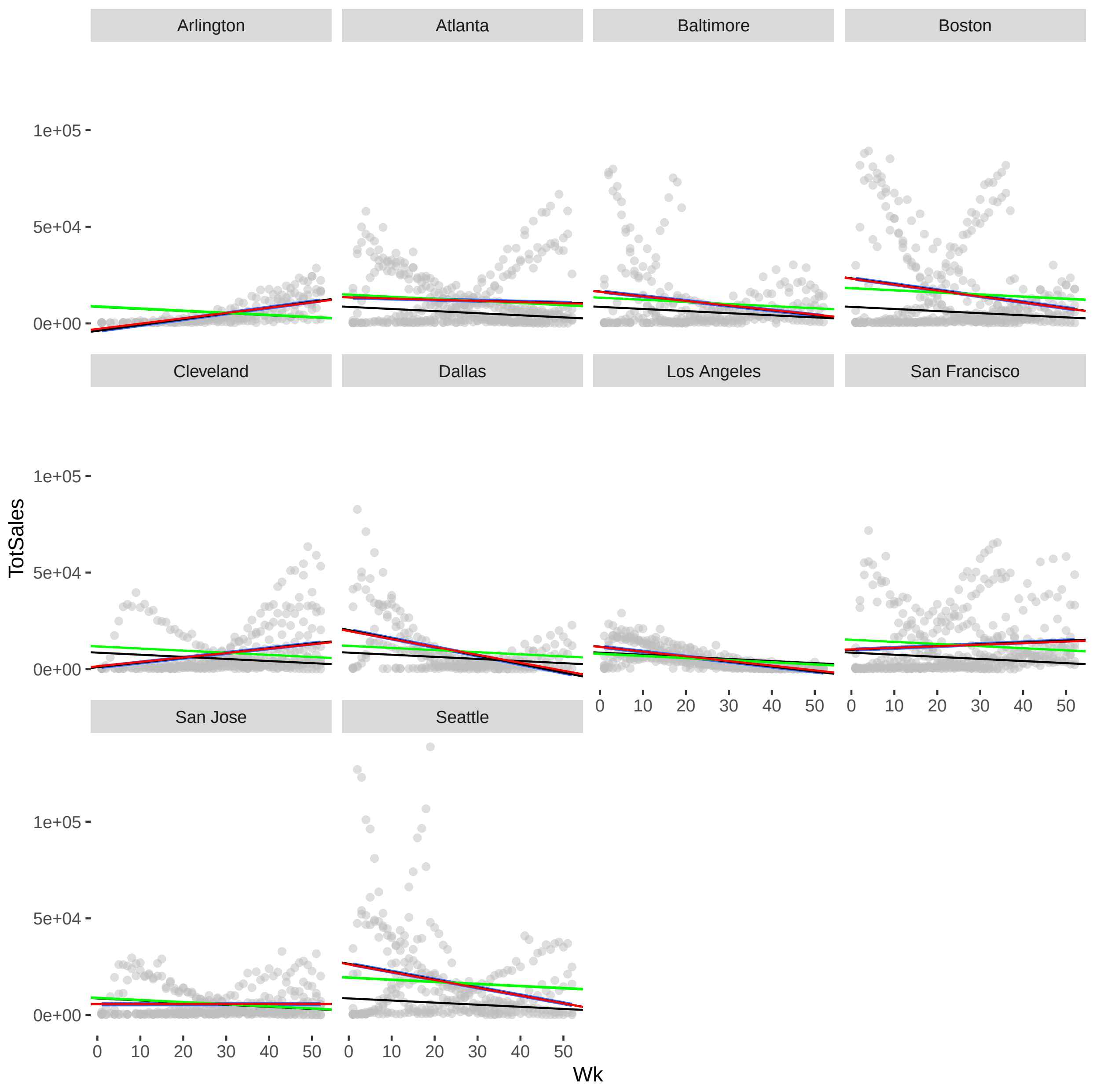


Note that the green lines from the lm varying intercept model have been overridden - the coefficients are the same. These models are partially pooled, they share correlation with other groups.

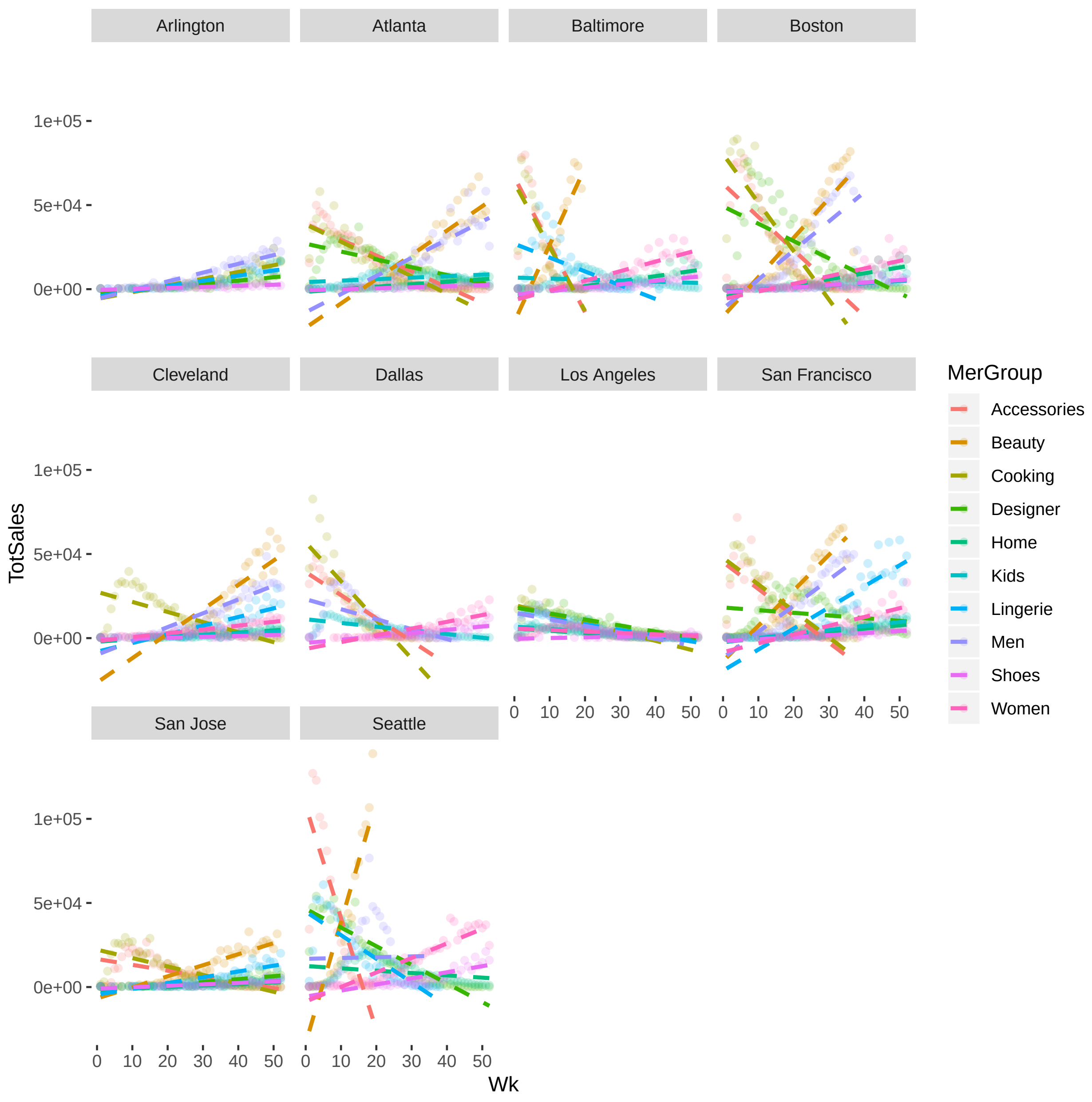
Varying Intercept and Slope Models

We can also let both Intercept and slope vary. The following lmer creates this type of model *(you might get convergence errors but it should get close enought for our purposes here - lmer struggles with complex, skewed data - its based on a gaussian assumption and takes a frequentist approach to determining effects)*. We could debug and correct by tweaking hyperparmeters - e.g., optimization algorithm, threshold, etc. but we're not going to be using lmer beyond estimating priors for Bayesian models, so just be aware that this is work to be done if you're using lmer for real. If we were going to use this model for deliverables in projects, then we'd have to resolve errors, which is why we spent some time on maximum likelihood, optimization and gradient descent. In the real world, data monkeys *(ah, that's you)* spend most of their time on ETL. Data scientists spend most of their time on parameter tuning, optimization and generalization.

```r
LMERVarIntSlpMod = lmer(TotSales ~ Wk + ( Wk | Description),
                     data = SalesTransSummary)

# tolerance is  set to 0.001 and optimizer is set to (Bound Optimization BY Quadratic Approximation)
# change optimizer

LMERVarIntSlpMod = update(LMERVarIntSlpMod, control = lmerControl(optimizer="Nelder_Mead"))

#summary(LMERVarIntSlpMod)
#fixef(LMERVarIntSlpMod)
#ranef(LMERVarIntSlpMod)$Description[,1]
# lmer breaks out the fixed effects and the "random effects"
# so the get the coefficients, we can add together fixed and random effects

#fixef(LMERVarIntSlpMod)[1] + c(ranef(LMERVarIntSlpMod)$Description[,1])
# or we can just ask for the coeffecients:

#coef(LMERVarIntSlpMod)
# agrees to above fixed + variable

LMERVarIntSlpModCoef = data.frame(coef(LMERVarIntSlpMod)$Description) %>%
  rownames_to_column("Description") %>%
  rename(Intercept = `X.Intercept.`, Slope = Wk)

p = p + geom_abline(data = LMERVarIntCoef,
                    aes(intercept = LMERVarIntSlpModCoef$Intercept, slope =  LMERVarIntSlpModCoef$Slope
p
```



```r
p =  ggplot(SalesTransSummary, aes(Wk, TotSales, color = MerGroup)) +
  geom_point(alpha = .2) +
  geom_smooth(method = "lm", se = F, alpha = .05, linetype = "dashed") +
  facet_wrap(~Description) +
  theme(panel.background = element_rect(fill = "white"))
p
```

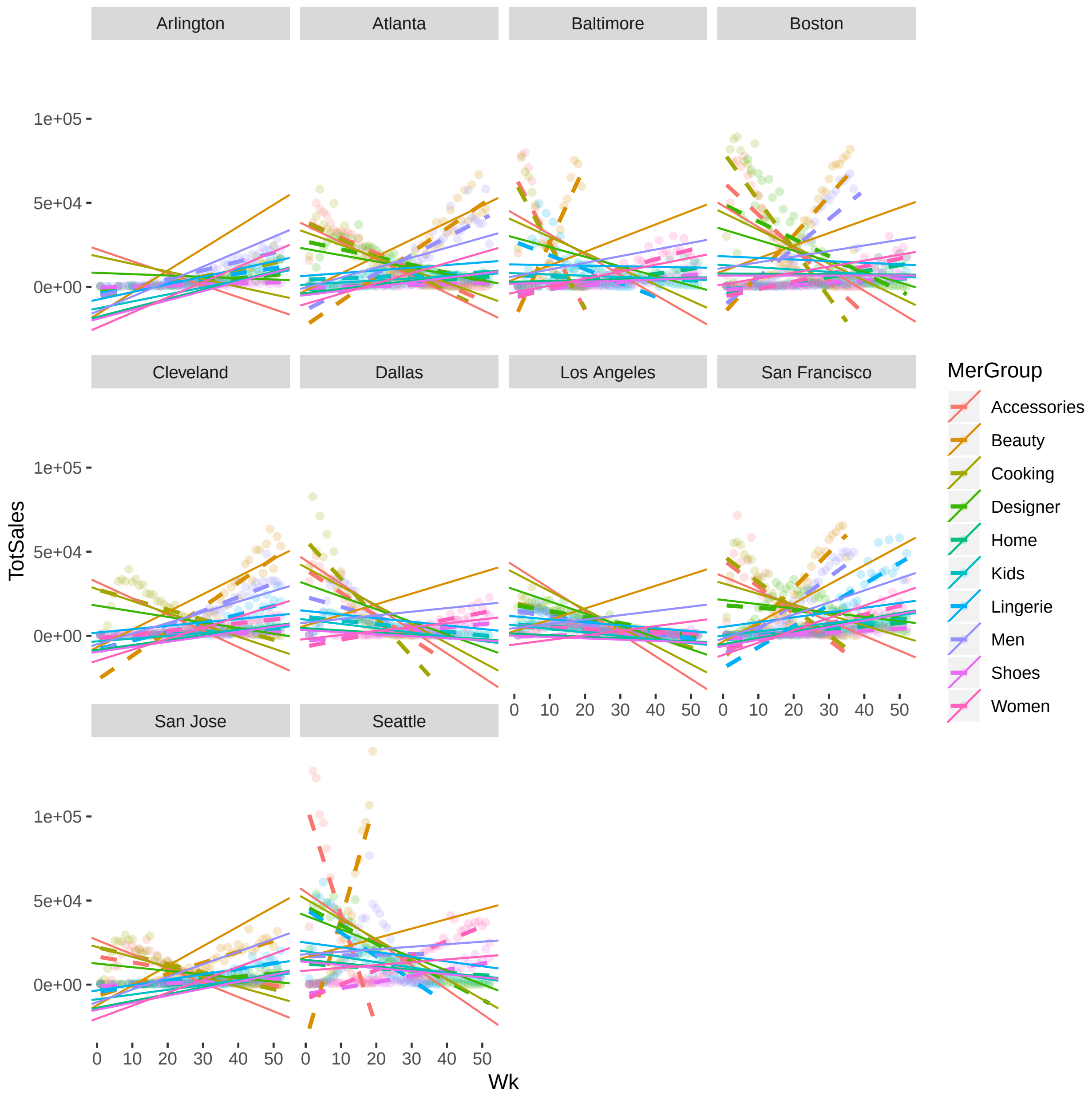


Two Levels - Varying Intercept and Slope

Finally, lets take a look at a varying intercept and slope with 2 grouping levels. The syntax is shown below:

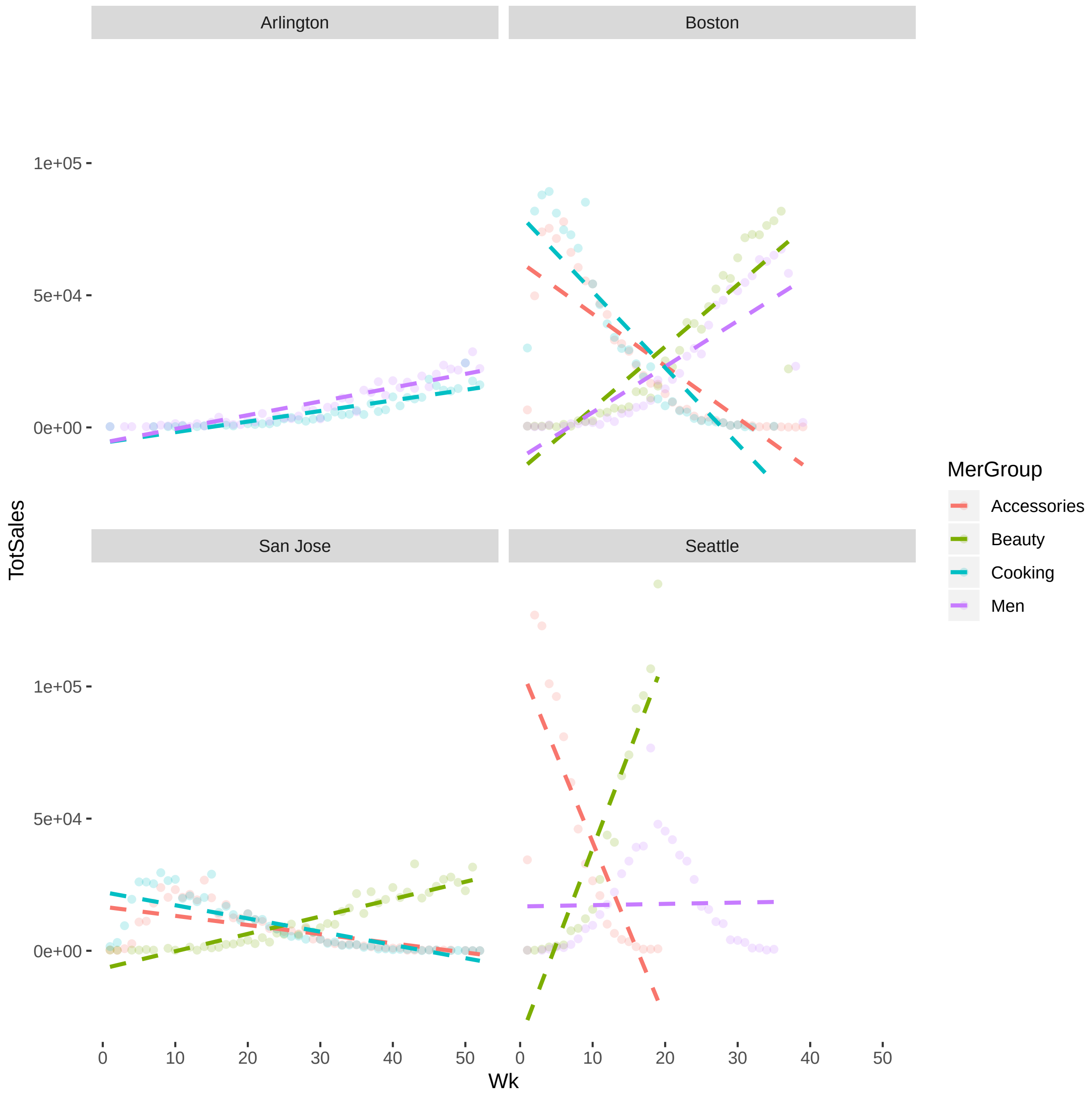
```
LMERVarIntSlpMod2L = lmer(TotSales ~ Wk + ( Wk | Description) + (Wk | MerGroup),
                          data = SalesTransSummary)
```

```
#summary(LMERVarIntSlpMod2L)
#fixef(LMERVarIntSlpMod2L)
#ranef(LMERVarIntSlpMod2L)$Description[,1]
#ranef(LMERVarIntSlpMod2L)$MerGroup[,1]
#coef(LMERVarIntSlpMod2L)

d1 = ranef(LMERVarIntSlpMod2L)$Description %>% rownames_to_column("Description") %>% select(Description
d2 = ranef(LMERVarIntSlpMod2L)$MerGroup %>% rownames_to_column("MerGroup") %>% select(MerGroup, "I2" =
I = fixef(LMERVarIntSlpMod2L)[1]
S = fixef(LMERVarIntSlpMod2L)[2]

Coef = crossing(d1, d2) %>% mutate(Intercept = I + I1 + I2, Slope = S + S1 + S2)

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



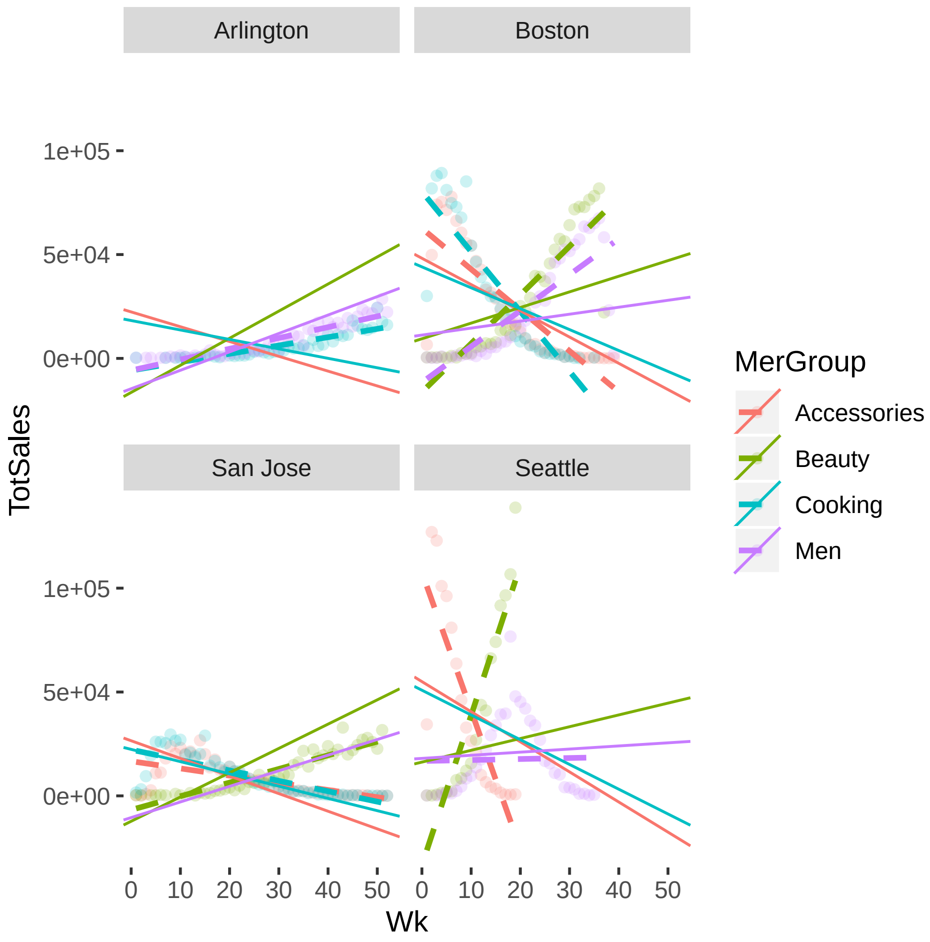
This is really getting crowded, so let's filter it down to 3 locations and 4 Mergroups. First we'll show the independent models for comparison:

```r
SalesSummarySub = filter(SalesTransSummary,
                         Description %in% c("Arlington", "Boston", "San Jose", "Seattle"),
                         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") +
  facet_wrap(~Description) +
  theme(panel.background = element_rect(fill = "white"))
p
```



And adding the lmer models:

```r
CoefSub = filter(Coef,
                        Description %in% c("Arlington", "Boston", "San Jose", "Seattle"),
                        MerGroup %in% c("Accessories", "Beauty", "Cooking", "Men" ))


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



A few things to notice here:

1. The lmer models are more grouped - they **pooled**, sharing correlation. One could say they're more generalized. And we use pooling for generalization, it gives us more control than L1 and L2 methods in machine learning.
2. Notice how Accessories and Beauty are modeled for Arlington, even though they didn't sell any. That's because the coefficients are group effects which can be applied for any combination. So if Arlington wanted to add Accesories or Beauty, we would already have a good start on predicting that effect. That's a BIG DEAL in business planning.

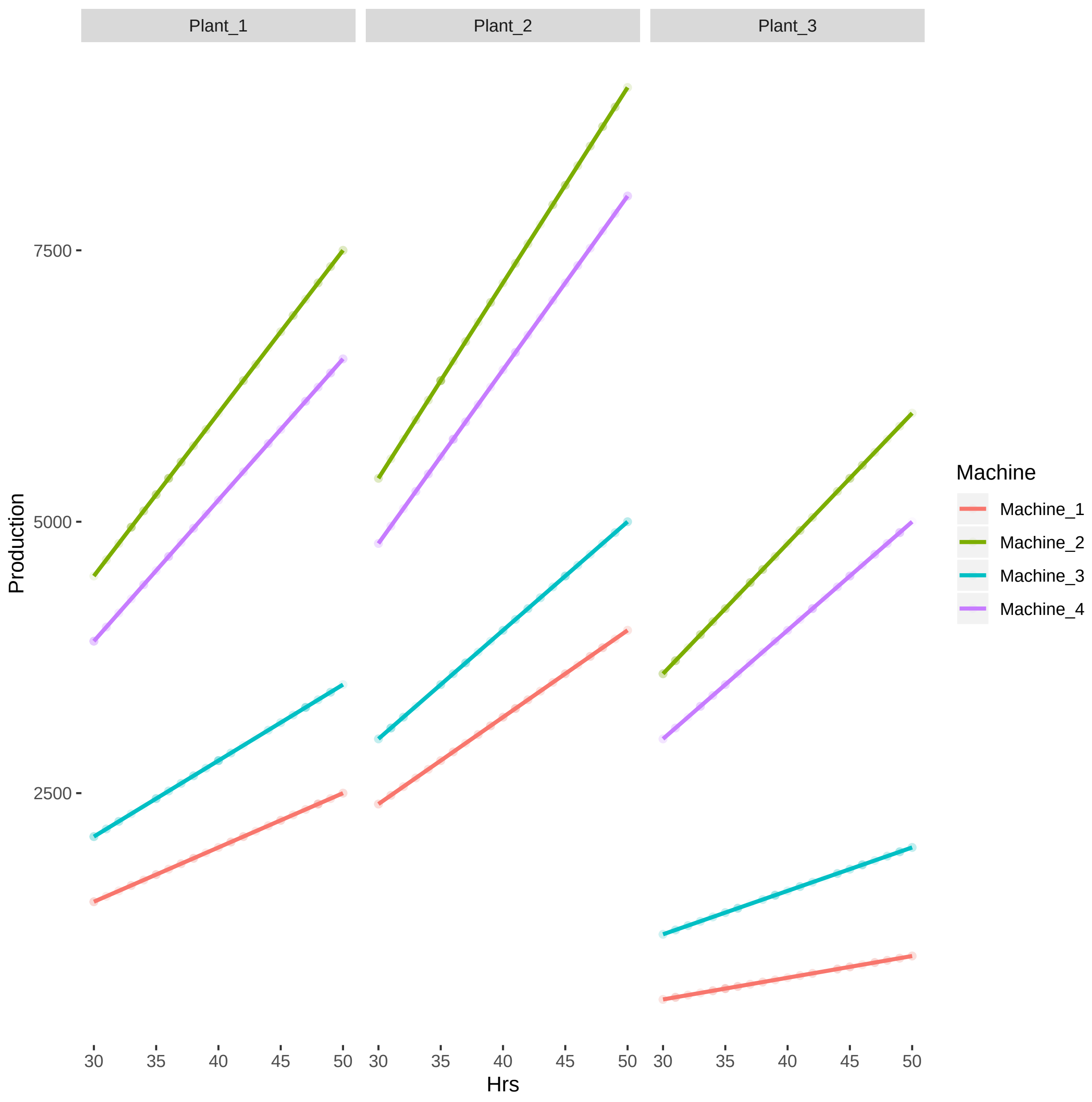
HOMEWORK

Use the following script to generate a 2 level dataset

```r
library(tidyverse)
library(lme4)

# Generate Multilevel Data

Mu = 100
N = 100
Plant = data.frame(Plant = c("Plant_1", "Plant_2", "Plant_3"), PEffect = c(0, 30, -30))
Machine = data.frame(Machine = c("Machine_1", "Machine_2", "Machine_3", "Machine_4"), MEffect = c(-50, 5
MHrs = 10
WOrders = 10
Data = crossing(Plant, Machine, Hrs = seq(1:MHrs), Production = seq(1:WOrders))
#placeholders (crossing needs unique keys)
Data = Data %>% mutate(Hrs = sample(seq(from = 30, to =50, by = 1), nrow(Data), replace = TRUE),
                       Production = Hrs*(Mu+PEffect+MEffect))
```

Which can be visualized *(there are data points there - it's just very tight - we're trying to keep variance to a minimum so you can see how the coefficients and effects reconcile easily)*:

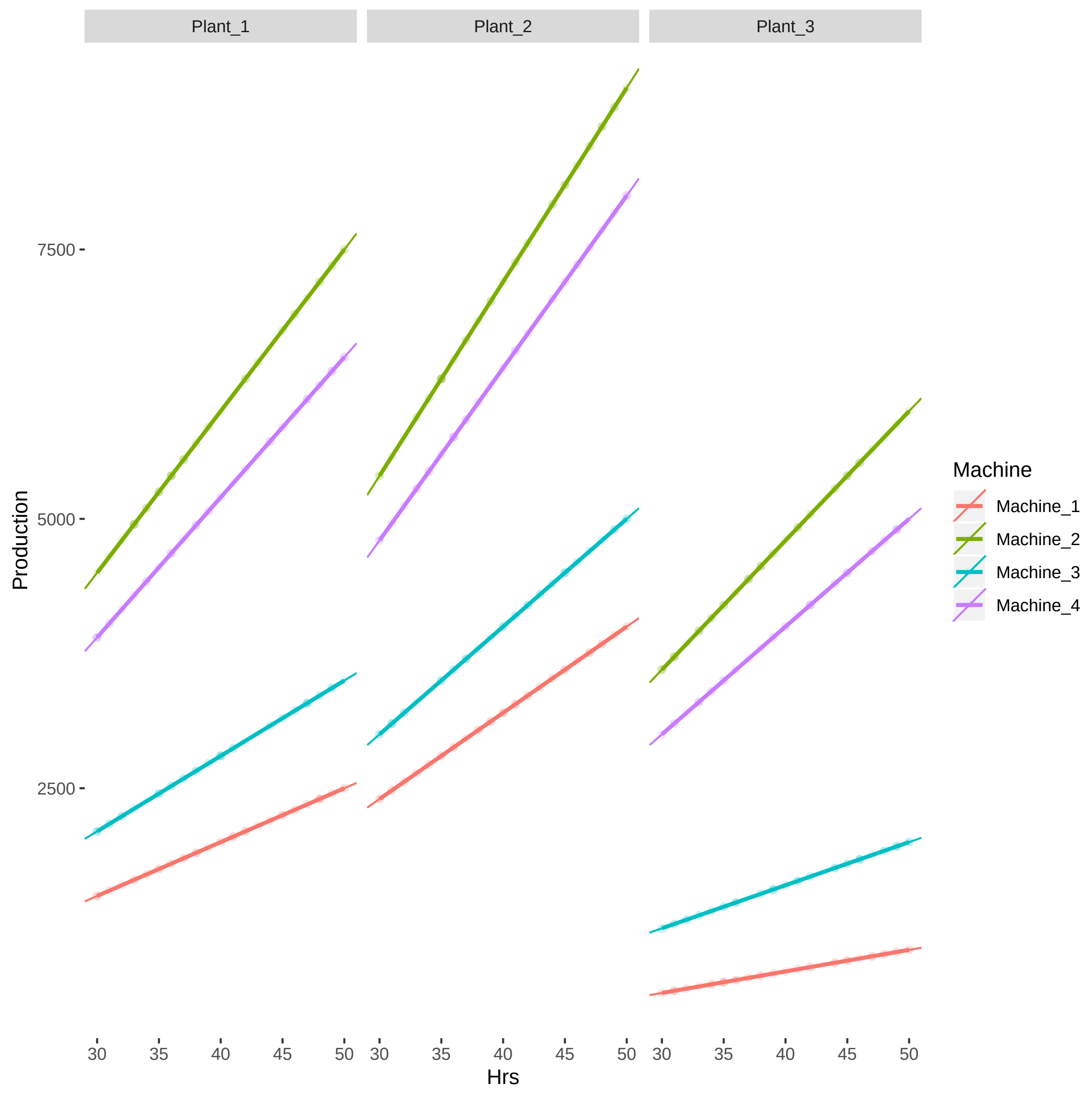
```r
p =  ggplot(Data, aes(Hrs, Production, color = Machine)) +
  geom_point(alpha = .05) +
  geom_smooth(method = "lm", se = F) +
  facet_wrap(~Plant) +
  theme(panel.background = element_rect(fill = "white"))
p
```



Create a mixed effects model, pull the coefficients out and print the model's regression lines:

```
HomeMod = lmer(Production ~ Hrs + ( Hrs | Plant) + (Hrs | Machine),
                             data = Data)

# this will result in sigular fit bc the data are generated with perfectly normal distributions

d1 = ranef(HomeMod)$Plant %>% rownames_to_column("Plant") %>% select(Plant, "I1" = "(Intercept)", "S1" =
d2 = ranef(HomeMod)$Machine %>% rownames_to_column("Machine") %>% select(Machine, "I2" = "(Intercept)",

I = fixef(HomeMod)[1]
S = fixef(HomeMod)[2]

Coef = crossing(d1, d2) %>% mutate(Intercept = I + I1 + I2, Slope = S + S1 + S2)
```

```
p = p + geom_abline(data = Coef,
                    aes(intercept = Coef$Intercept, slope =  Coef$Slope, color = Machine))
p
```



Now, just narrow the model estimate down to Slope, Pull the coefficients and join to the orignial effects data. Then add the effects together and compare to the model estimate, summing the total.

```
Data$Plant = as.character(Data$Plant)
Data$Machine = as.character(Data$Machine)
Data = inner_join(Data, Coef, by = c("Plant", "Machine"))
Data$EstSlope = Mu+Data$PEffect+Data$MEffect
sum(Data$EstSlope-Data$Slope)
```

```
## [1] -4.629337e-06
```

Now, Adjust the effect and observe the changes in the plot, and changes in the values of slope.

Lme4 Random Effects Cheat Sheet ***ref: Eshin Jolly)***

###Random intercepts only (1 | Group)

Random slopes only

(0 + Variable | Group)

Random intercepts and slopes (and their correlation)

(Variable | Group)

Random intercepts and slopes (without their correlation)

(1 | Group) + (0 + Variable | Group)

Same as above only if Variable is *continuous* and not a factor

(Variable || Group)

Random intercept and slope for more than one variable (and their correlations)

(Variable_1 + Variable_2 | Group)