Tutorial-MEM-Pooling-Comparison-SleepStudyData

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.3     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.4.4     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```
library(lme4)
```

```
## Loading required package: Matrix
##
## Attaching package: 'Matrix'
##
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
library(ggplot2)
library(tibble)
```

Dataset Generation

```
# GLOBAL VARIABLES
N_OBSERVATIONS <- 100

# Local group variables
mean_A <- 25
sd_A <- 3
mean_B <- 50
sd_B <- 8

A <- rnorm(N_OBSERVATIONS, mean = mean_A, sd = sd_A)
B <- rnorm(N_OBSERVATIONS, mean = mean_B, sd = sd_B)
```

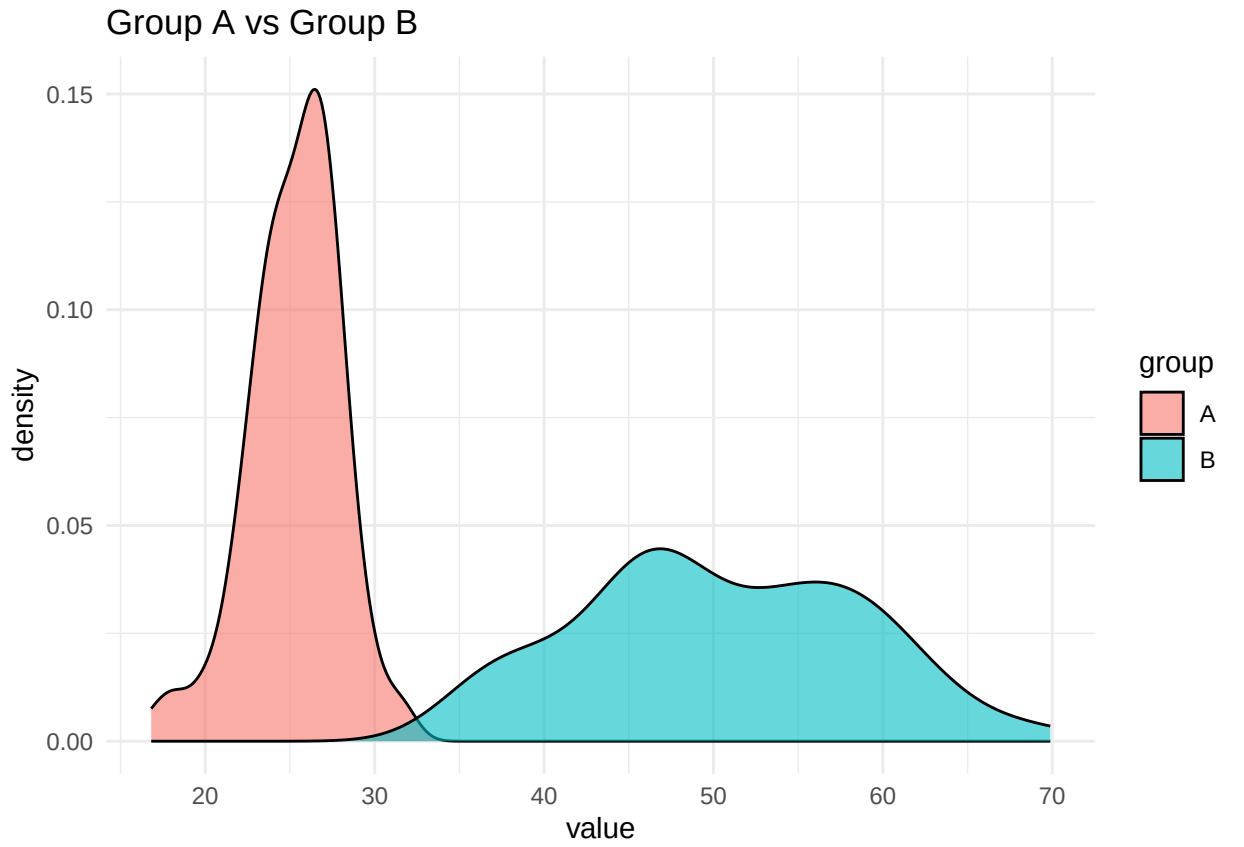
Dataset investigation

- Gather pivots the two columns A and B to be the Key column related to the Value column that is collecting the values of the observations of A and B. Basically is a hstack of A and B.
- Mutate generates a new column called group that is equal to the values present in key.

```r
df <- data.frame(A = A, B = B) %>%
      gather() %>%
      mutate(group = key)
print(head(df))
```

```
##   key    value group
## 1   A 24.36934     A
## 2   A 23.43448     A
## 3   A 28.50294     A
## 4   A 28.14485     A
## 5   A 27.00078     A
## 6   A 25.69198     A
```

```r
df %>%
  ggplot(aes(x= value, fill=group)) +
    geom_density(alpha=.6) +
    labs(title="Group A vs Group B") +
    theme_minimal()
```



Explanation of Ordinary Least square regression

The two, looks pretty different. We could quantify their differences by using *ordinary least square regression*, predicting *value* from *group*

Y $X + \epsilon$ X:= predictor or fixed effect ϵ:= error term that encompasses everything that the model does not know about where the variance of Y might come from. Basically it's the variance of Y that is not correlated with X. Y $\beta_0 + \beta_1 X + \epsilon$

```r
simple_model <- lm(data = df, value ~ group)
summary(simple_model)
```

```
##
## Call:
## lm(formula = value ~ group, data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.6014  -3.3931   0.0408   2.8815  19.5327
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  25.2911     0.6082   41.59   <2e-16 ***
## groupB       25.0608     0.8601   29.14   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.082 on 198 degrees of freedom
## Multiple R-squared:  0.8109, Adjusted R-squared:  0.8099
## F-statistic:   849 on 1 and 198 DF,  p-value: < 2.2e-16
```

Lot's of results but our main concern is in the estimate of the coefficient of our predictor variables. In this case the β_1:

$\beta_1 = 24.9669 \pm 0.7559$ with a significant pvalue <2e-16

```r
print(simple_model$coefficients["(Intercept)"])
```

```
## (Intercept)
##    25.29107
```

```r
cat("This is the mean of group A: ", mean(A),"\n")
```

```
## This is the mean of group A:  25.29107
```

```r
print(simple_model$coefficient["groupB"])
```

```
##   groupB
## 25.06076
```

In other words the group B is on average 24.96691 units higher than the observations in A. We can verify it by evaluating the difference in the mean of the two groups:

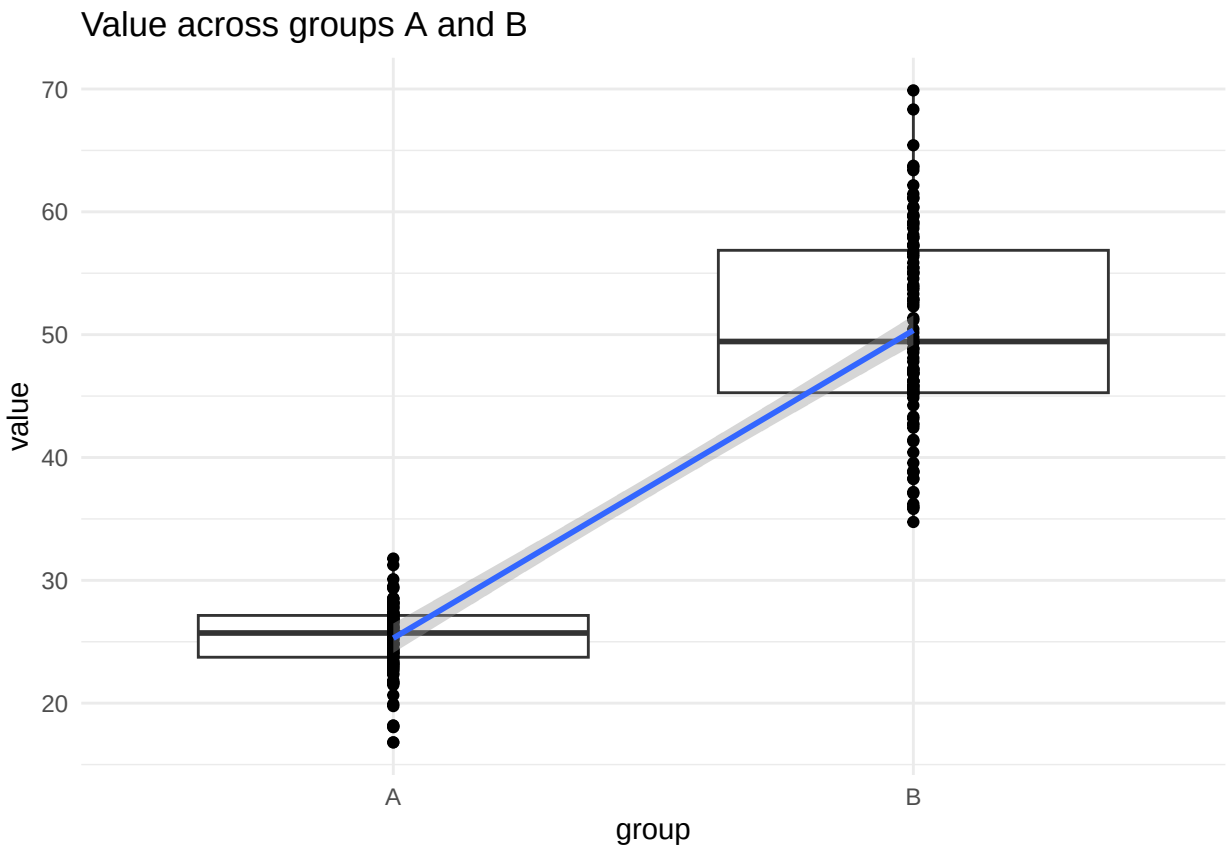
```r
print(mean(B) - mean(A))
```

```
## [1] 25.06076
```

We can also express this visually. The mean of group A is equivalent to the intercept β_0 and the best-fit line β_1 connects this intercept to the mean of group B

```r
df %>%
  ggplot() +
  geom_boxplot(aes(x=group, y=value)) +
  geom_point(aes(x=as.integer(factor(group)), y = value)) +
  geom_smooth(aes(x=as.integer(factor(group)), y=value), method="lm") +
  labs(title="Value across groups A and B") +
  theme_minimal()
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



When indepencence is not met

Unfortunately, this independence assumption is not always met. This is problematic for traditional OLS approaches, regardless of what your dependent variable may be.

Let's load a new dataset from lme4 package

```
sleepstudy <- lme4::sleepstudy
print(head(sleepstudy))
```

```
##   Reaction Days Subject
## 1 249.5600    0     308
## 2 258.7047    1     308
## 3 250.8006    2     308
## 4 321.4398    3     308
## 5 356.8519    4     308
## 6 414.6901    5     308
```

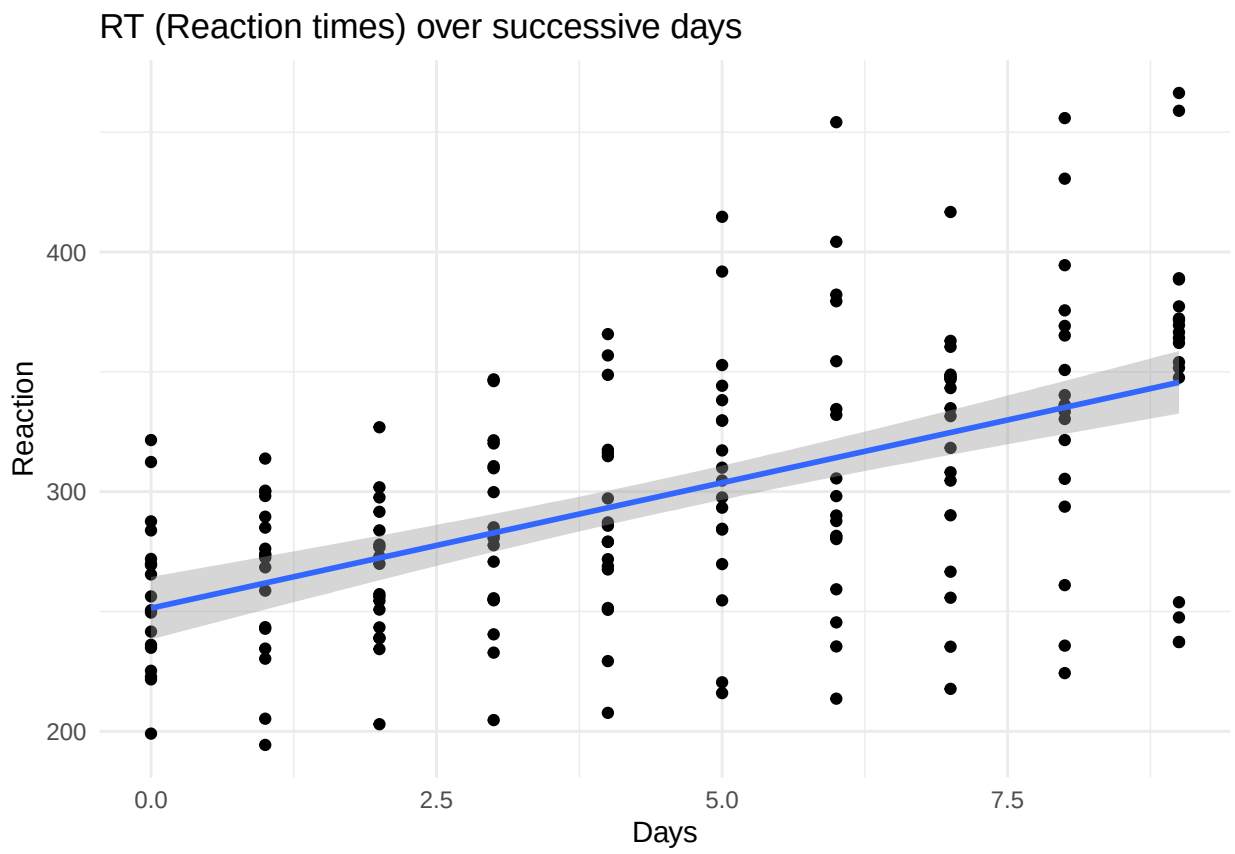
In this dataset i understand three features: - Reaction - Days - Subject Reaction may be the value that i'm measuring, while Days and Subject are clearly some metadata that i can exploit to understand if data are independent from each other or not. For example, measures coming from the same subject may be correlated between each other. Same thing for observaitons coming from the same day. Let's think that whoever collected the data may be a source of variability, therefore, observations collected the same date may be correlated to each other.

Naive analysis

Let's pretend that we did not know about the source of non-independence

```
sleepstudy %>%
  ggplot(aes(x=Days,y=Reaction)) +
  geom_point() +
  geom_smooth(method="lm") +
  labs(title= "RT (Reaction times) over successive days") +
  theme_minimal()
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



Looks like there is a positive relationship between Days and Reaction time. As we would expect, more sleep deprivation leads to slower reaction times. Let's try to model this relationship with OLS.

```
naive_model <- lm(data = sleepstudy,
                  Reaction ~ Days)
summary(naive_model)
```

```
##
## Call:
## lm(formula = Reaction ~ Days, data = sleepstudy)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -110.848  -27.483    1.546   26.142  139.953
##
```

```
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  251.405      6.610  38.033  < 2e-16 ***
## Days          10.467      1.238   8.454 9.89e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 47.71 on 178 degrees of freedom
## Multiple R-squared:  0.2865, Adjusted R-squared:  0.2825
## F-statistic: 71.46 on 1 and 178 DF,  p-value: 9.894e-15
```

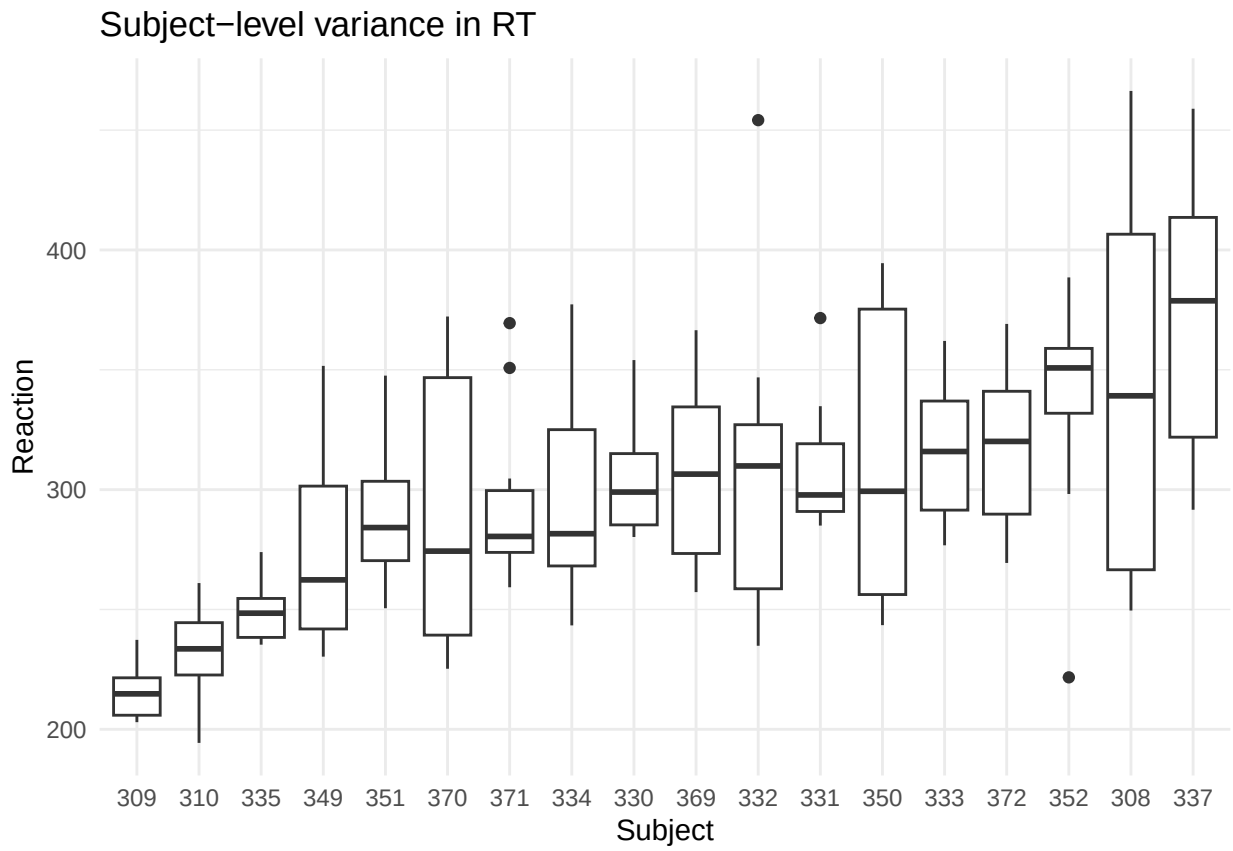
The naive model shows that for eahc continued day without sleep the reaction times increase by an average of 10.47. The average starting reaction times is 251.405

However, we recall that we have observaitons from 18 different subjects, that may have some variability in their overall reaciton time. They may also show variability in how resilient they are to sleep deprivation.

Non-Independence analysis

reorder: =

```
sleepstudy %>%
  ggplot(aes(x=reorder(Subject,Reaction),
             y=Reaction)) +
  geom_boxplot() +
  labs(title="Subject-level variance in RT", x = "Subject") +
  theme_minimal()
```



Another example of usage of sleepstudy dataset

Pooling

This will be the first usage of tibble. I tibble sono strutture di dati definite da una classe, costruita come sottoclasse di data.frame. Tutti i tibble sono dataframe ma non tutti i dataframe sono tibble.

Un tibble, a differenza di un dataframe, può contenere anche delle liste, purchè il numero degli elementi corrisponda al numero delle righe

```
sleepstudy <- sleepstudy %>%
  as_tibble() %>%
  mutate(Subject = as.character(Subject))
head(sleepstudy)
```

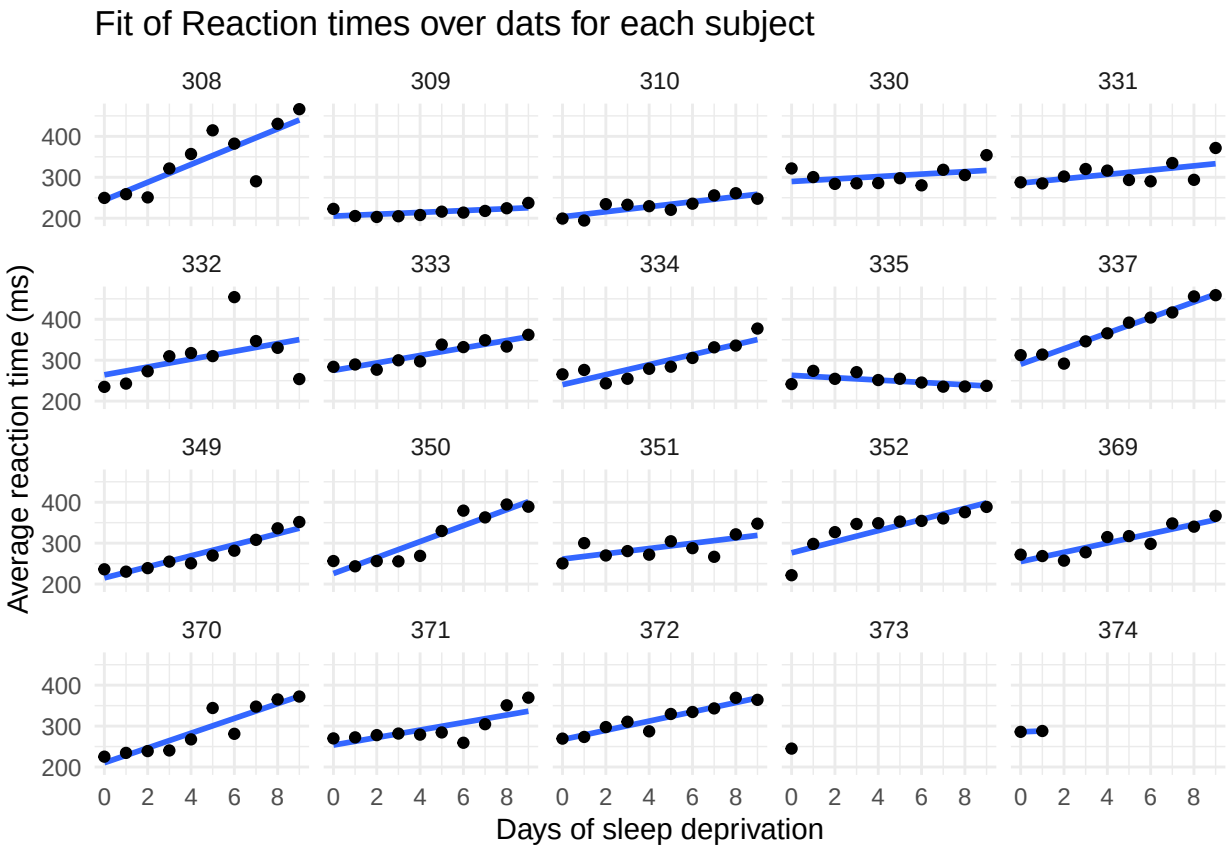
```
## # A tibble: 6 x 3
##   Reaction  Days Subject
##      <dbl> <dbl> <chr>
## 1     250.     0 308
## 2     259.     1 308
## 3     251.     2 308
## 4     321.     3 308
## 5     357.     4 308
## 6     415.     5 308
```

```
# Let's add two new fake subjects
df_sleep <- bind_rows(sleepstudy,
                      tibble(Reaction = c(286,288), Days = 0:1, Subject = "374"),
                      tibble(Reaction = 245, Days = 0, Subject = "373"))
```

Let's visualize all the data by dividing each group.

```
df_sleep %>%
  ggplot(aes(x=Days, y=Reaction)) +
  geom_smooth(method="lm", se=FALSE) +
  geom_point() +
  facet_wrap("Subject") +
  labs(title="Fit of Reaction times over dats for each subject",
       x="Days of sleep deprivation",
       y="Average reaction time (ms)") +
  scale_x_continuous(breaks = 0:4 * 2) +
  theme_minimal()
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



Complete pooling and no Pooling models

Each of these panels plotted above shows an independently estimated regression line. This approach to fit a separate line for each participant is sometimes called the **no pooling** model because none of the information from different participants is combined or pooled together.

We fit a separate line for each cluster of data, unaware that any of the other participants exist.

The lmList() function in lme4 automates this process.

The following code should fit in df_sleep a different line: Y = X where Y is Reaction and X is Days by grouping for each different Subject. "|" is the operator to group by a column values.

```
df_no_pooling <- lmList(Reaction ~ Days | Subject, df_sleep) %>%
  coef() %>%
  rownames_to_column("Subject") %>%
  rename(Intercept = `(Intercept)`, Slope_Days = Days) %>%
  add_column(Model = "No pooling") %>%
  filter(Subject != "373")
head(df_no_pooling)
```

```
##   Subject Intercept Slope_Days      Model
## 1     308  244.1927  21.764702 No pooling
## 2     309  205.0549   2.261785 No pooling
## 3     310  203.4842   6.114899 No pooling
## 4     330  289.6851   3.008073 No pooling
## 5     331  285.7390   5.266019 No pooling
## 6     332  264.2516   9.566768 No pooling
```

In contrast, we might consider **a complete pooling** model where all the information from the participants is

combined together. We fit a single line for the combined data set, unaware that the data came from different participants.

```
# Fit a model on all the data pooled together
m_pooled <- lm(Reaction ~ Days, df_sleep)

# Repeat the intercept and slope terms for each participant
df_pooled <- tibble(
  Model = "Complete pooling",
  Subject = unique(df_sleep$Subject),
  Intercept = coef(m_pooled)[1],
  Slope_Days = coef(m_pooled)[2]
)
head(df_pooled)
```

```
## # A tibble: 6 x 4
##   Model            Subject Intercept Slope_Days
##   <chr>            <chr>       <dbl>      <dbl>
## 1 Complete pooling 308          252.       10.3
## 2 Complete pooling 309          252.       10.3
## 3 Complete pooling 310          252.       10.3
## 4 Complete pooling 330          252.       10.3
## 5 Complete pooling 331          252.       10.3
## 6 Complete pooling 332          252.       10.3
```

Let's compare by not fitting again the line, but by drawing the lines for the no_pooling model and pooling_model

```
# Join the raw data so we can use plot the points and the lines.
df_models <- bind_rows(df_pooled, df_no_pooling) %>%
  left_join(df_sleep, by = "Subject")
```

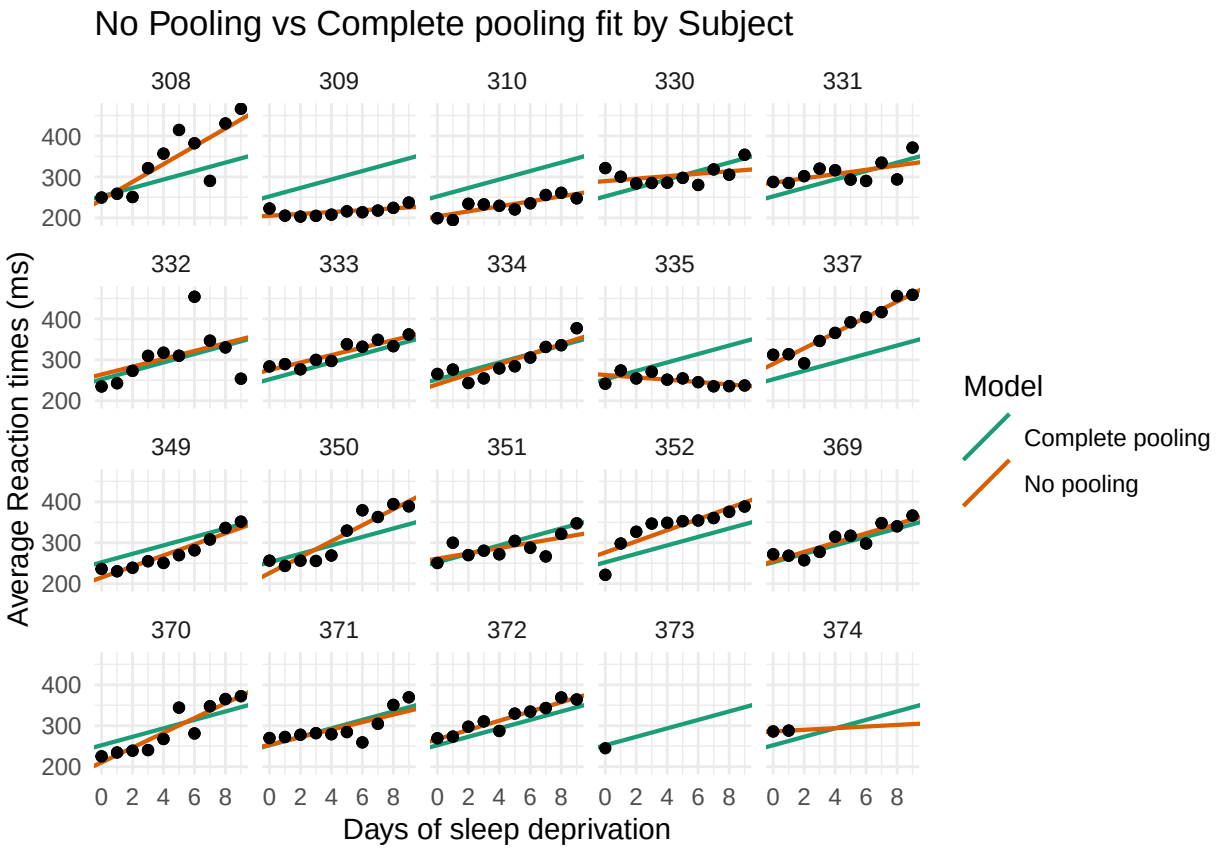
```
## Warning in left_join(., df_sleep, by = "Subject"): Detected an unexpected many-to-many relationship b
## i Row 1 of `x` matches multiple rows in `y`.
## i Row 1 of `y` matches multiple rows in `x`.
## i If a many-to-many relationship is expected, set `relationship =
##   "many-to-many"` to silence this warning.
```

```
p_pooling_comparison <- df_models %>%
  ggplot(aes(x=Days,
             y=Reaction)) +
  geom_abline(aes(intercept=Intercept,
                  slope=Slope_Days,
                  color=Model),
              size=.75) +
  geom_point() +
  facet_wrap("Subject") +
  labs(title = "No Pooling vs Complete pooling fit by Subject",
       x="Days of sleep deprivation",
       y="Average Reaction times (ms)") +
  scale_x_continuous(breaks = 0:4*2) +
  scale_color_brewer(palette="Dark2") +
  theme(legend.position = "bottom", legend.justification = "right") +
  theme_minimal()
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
```

```
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
p_pooling_comparison
```



The complete pooling model estimates a single line, and we see that same line drawn on every facet. One advantage is that the model can make a guess about the line for 373, who only has one observation. The model looks terrible elsewhere, 309, 310, etc... In contrast the no pooling data approach can follow the data, without being able to make a guess on 373 though.

Mixed Models: a Walkthrough

The sleepstudy dataset is an example of data with **nested structure**. In this case are manifested as multiple observations coming from the same subject,

1) Random intercepts: Control for subject-level variance in Y
2) Random slopes: Control for subject-level variance in the effect of X on Y

Random Intercepts: accounting for nested variance in Y

Let's say that different people differ in their average reaciton time, regardless of how sleep deprived they are. We can incorporate it into our model as something called **random effect** by saying that the Reaction time is modeled as:

Reaction ~ Days + (1 | Subject)

Some time expetiments will involve other kinds of nested structure, such as item-level variance. Our current

dataset averages across all observations for a subject in a given date. But if it did, and these items exhibited nested strucutre, we could model it as follows:

This is an example of an alternative: Reaction ~ Days + (1 | Subject) + (1 | Item)

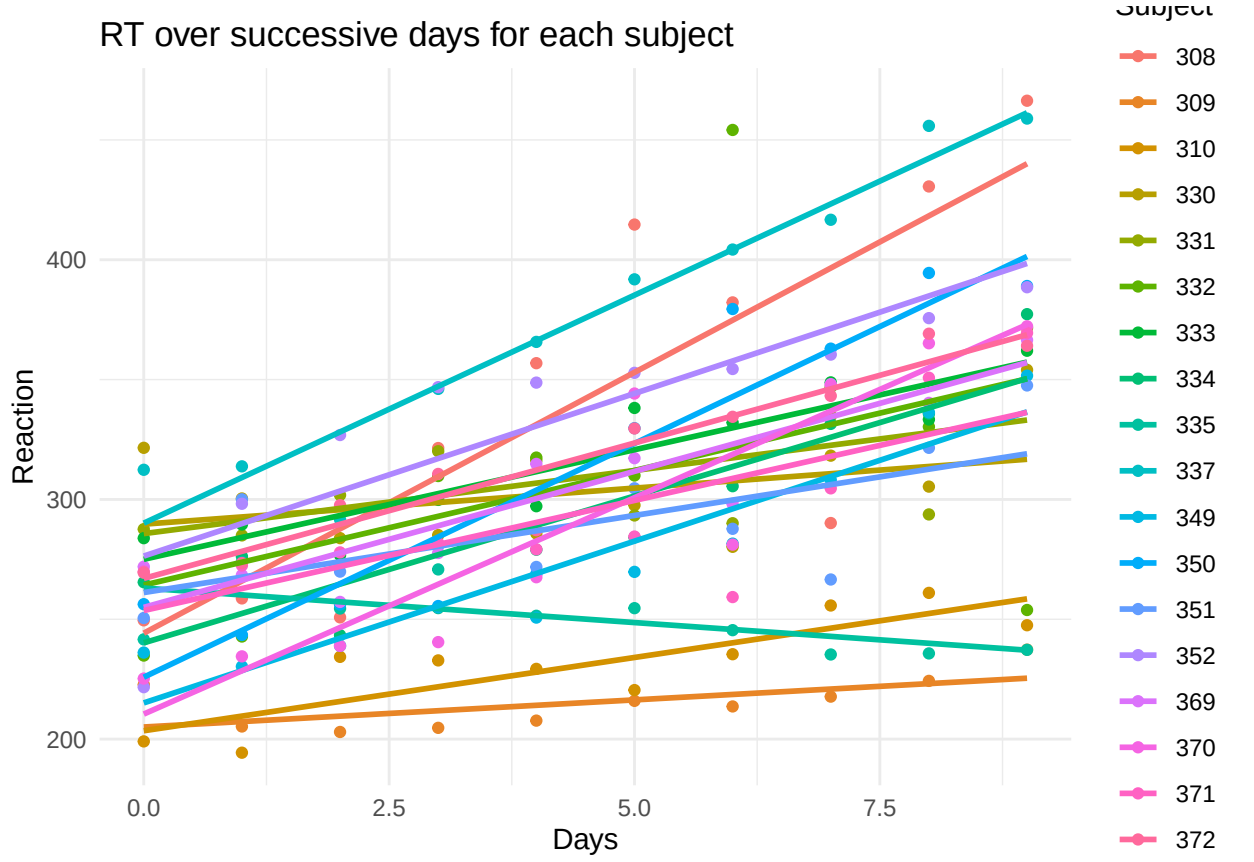
Random Slopes: accounting for nested variance in the effect of X on Y

Subjects may vary in their effect of sleep deprivation on reaction time. THis is actually not unreasonable to assume. Some may act perfectly even with only 2-3 hours of sleep, while others won't.

We can actually visualize this variability by drawin separate best-fit lines for each subject.

```
sleepstudy %>%
  ggplot(aes(x=Days,y=Reaction,color=Subject)) +
  geom_point() +
  geom_smooth(method="lm",se=FALSE) +
  labs(title="RT over successive days for each subject") +
  theme_minimal()
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



Our ultimate goal may be to come up with a coefficient estimate for the effect of Days. To do so, we need to account for the fact that not all subjects show this effect on equal degrees.

The actual model would then also have a random effect on the effect of the predictory variable Days over the Reaction times for each subject.

Reaction ~ Days + (1 + Days | Subject)

```r
model_full = lmer(data = sleepstudy,
                  Reaction ~ Days + (1 + Days | Subject),
                  REML = FALSE)
summary(model_full)
```

```
## Linear mixed model fit by maximum likelihood  ['lmerMod']
## Formula: Reaction ~ Days + (1 + Days | Subject)
##    Data: sleepstudy
##
##      AIC      BIC   logLik deviance df.resid
##   1763.9   1783.1   -876.0   1751.9      174
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -3.9416 -0.4656  0.0289  0.4636  5.1793
##
## Random effects:
##  Groups   Name        Variance Std.Dev. Corr
##  Subject  (Intercept) 565.48   23.780
##           Days         32.68    5.717   0.08
##  Residual             654.95   25.592
## Number of obs: 180, groups:  Subject, 18
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept)  251.405      6.632  37.907
## Days          10.467      1.502   6.968
##
## Correlation of Fixed Effects:
##      (Intr)
## Days -0.138
```

Differences with OLS: - There is no *p-value* associated information with the coefficients.

This is an intentional design by the author of lme4, and has to do with the way lme4 calculates its t- and F-statitics. If you really want to estimate the p-values for each coefficient, you can import the package lmerTest and rerun the model. The standard approach to select mixem models is not by significance of the coefficients but by model comparison.

Model comparison

The core idea behind the model comparison approach is to compare the **explanatory power** of two or more models. Of course a model with more parameters will always explain more variance than a model with less. However, this can result in overfitting. A model whose parameters are too tuned to the variance specific to a particular dataset, and thus exhibits poor generalizability across datasets. So reducing variance in Y is not out only goals. We also want a parsimonious model.

1) Comparing the amount of variance explained of two models
2) Enforcing model parsimony. We prefer a simpler model.

We'll be using the so called likelihood ratio test.

```r
# We omit the fixed effect of Days.
model_reduced <- lmer(data = sleepstudy,
                      Reaction ~ (1 + Days | Subject),
                      REML = FALSE)
```

```
comparison <- anova(model_full, model_reduced)
comparison
```

```
## Data: sleepstudy
## Models:
## model_reduced: Reaction ~ (1 + Days | Subject)
## model_full: Reaction ~ Days + (1 + Days | Subject)
##               npar    AIC    BIC  logLik deviance  Chisq Df Pr(>Chisq)
## model_reduced    5 1785.5 1801.4 -887.74   1775.5
## model_full       6 1763.9 1783.1 -875.97   1751.9 23.537  1  1.226e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
chisq = round(comparison$Chisq[2],2)
chisq
```

```
## [1] 23.54
```

```
df <- comparison$Df[2]
df
```

```
## [1] 1
```

```
pvalue <- comparison$`Pr(>Chisq)`[2]
pvalue
```

```
## [1] 1.22564e-06
```

There is a fair amount of output here: - loglik: this refers to the log-likelihood values for each of the models. model_reduced has more negative value, indicaitng a lower likelihood of the full model. Likelihood Ratio: LR = -2 * (LLreduce - LLfull) - Chisq: this is the chi-squared statistic that we'll be looking up to obtain our p-value. Note that is identical to the LR - DF: this is the difference in degrees of freedom between our models. - Pr(>Chisq): this is the probability of obtaining a chi-squared statistic that high under the null hypothesis, i.e., our p-value

The p-value indicates that the difference in model likelihoods is quite significant, i.e., this Chisq statistic is quite unlikely under the corresponding null distribution.

We interpret the results as follow: adding a fixed effect of Days significantly increases the explanatory power of our model.

A model including a fixed effect of Days, as well as by-subject random slopes for the effect of Days (and by-subject random intercepts), explained significantly more variance than a reduced moedl omitting only the fixed effect of Days [X^2(1)=23.54, p=1.2256e^-6]

We can also inspect the direction of the effect of Days.

```
coef_days = fixef(model_full)["Days"]
coef_days
```

```
##     Days
## 10.46729
```

As in the OLS regression, the coefficient is positive and for every day of sleep deprivation we expect participants to slow by an average of 10.47 ms. The difference is that this parameter now accounts for nested variance in the data.

Best practices

Sometimes the distinciton between fixed and random effect is defined in terms of "things you care about modeling directly" (fixed) vs "effects related to the population you're interested in modeling" (random).

Some rules of thumb: - Should i Include random factor X in the model? If X demonstrates nested strucutre in my dataset, then i include it. seems preferable to me to try to account for potential nested variance, and perhaps be overly cautious, than to miss something important. - Should i enter variable X as a random of fixed effect? This is trickier. It comes down to my a priori assumptions about the relationship between this variable and Y. If i think that X covariates in important, systematic ways with Y, then I enter it as a fixed effect. If i think that each cluster of nested structure in X is essentially random with respect to Y, that is each group in X varies, but not systematically such that we can assume each group has been pulled from some overarching distribution, then i enter it as a random effect.

How do i know how many random effects to include?

Should you also include random effects for both Item and Subject ID? Further, should you model them as random intercepts random slopes or both??

Standard advice, "Keep it Maximal". Using simulation based analyses, the author demonstrate that the best way to minimize false positive rates and improve generalizability is to begin with a maximally specified model: include both slopes and intercepts for all random factors: this can sometimes result in pretty complicated models, such as the one below:

Y = X1*X2 + (X1*X2 | Subject) + (X1*X2 / Item) + (X1* X2 | Group)

Which is basically the same as saying: Calculate by-subject, by-item, and by-group random slopes for the effect of X1, X2 and their interaction as well as random intercepts for all those variables.

A common issue is that the model won't converge. Then do the following: 1) Rescale and center continous parameters. Differences in the scales of the parameters can lead to issues in estimating std of fixed effects. 2) Use a different optimizer, for example bobyqa optimizer. lmerControl(optimizer="bobyqa") 3) Simplify the model.

If 1) and 2) both fail, then you can take a principled approach to 3). You can use the same model comparison procedure described earlier using the LRtests and reduce the model iteratively.

How are estimates for random factor computed?

Estimates for random factors are calculated using something called shrinkage (sometimes called partial pooling or regularizaiton).

Theoretical background

When we compute estimates for each group, we assume that they are pulled from a normal distribution around the overall mean for those values across all groups. Hence the term shrinkage. Rather than being allowed to vary freely, the estimates are constrained in terms of the values they can take on.

We can compare this to two extreme cases: 1) Consider the case in which we have no random factors. Typical OLS. We are not accounting for any nested structure and computing a single intercept for the entire dataset. This intercept is "pooled" over all observations. $Y \sim \beta_0 + \beta_1 X + \epsilon$ 2) Consider the extreme case in whic we compute a different intercept for each group adn this intercept is allowed to vary freely for each participant: $Y\ \beta_{0i} + \beta_1 X + \epsilon$ In this case, any given beta0i will be equivalent that Participant's mean Y. 3) Consider the case of shrinkage, where we allow each person's intercept to vary, but we model these intercepts as normally distributed around the group mean.

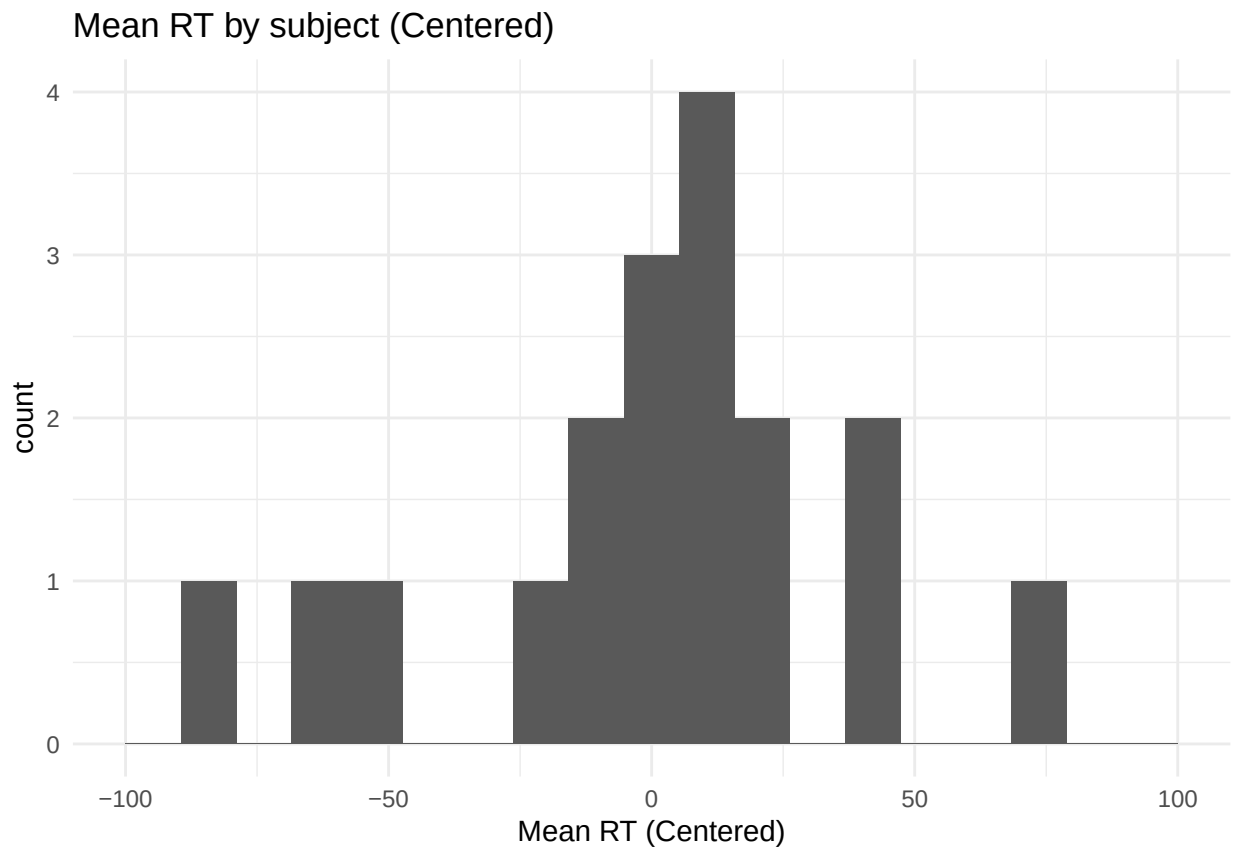
Example

```
# We are basically evaluating the overall mean of the population reaction times.
# Then, we are grouping by subject and creating a we new column mean_rt that will contain the average v
# Then we are mutating by adding a new column that is the average value of the Reaction times for eeach

group_mean <- mean(sleepstudy$Reaction)

by_subject <- sleepstudy %>%
  group_by(Subject) %>%
  summarise(mean_rt = mean(Reaction)) %>%
  mutate(mean_rt_centered = mean_rt - group_mean)
```

```
by_subject %>%
  ggplot(aes(x=mean_rt_centered)) +
  geom_histogram(bins=20) +
  labs(title="Mean RT by subject (Centered)", x="Mean RT (Centered)") +
  scale_x_continuous(limits=c(-100,100)) +
  theme_minimal()
```

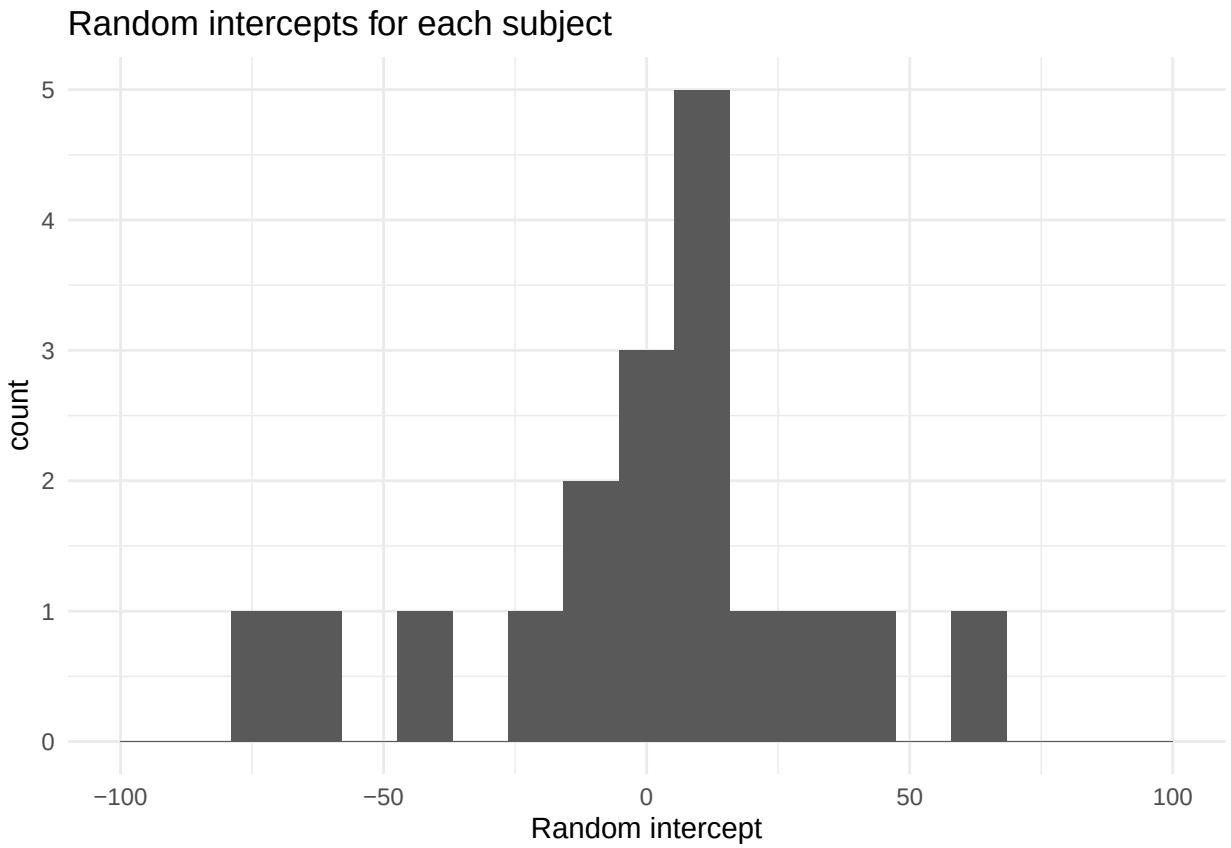


Now let's extract the random intercepts from a model only icluding our random intercepts:

```
model_null <- lmer(data =sleepstudy,
                   Reaction ~ (1 | Subject),
                   REML = FALSE)
by_subject$random_intercepts = ranef(model_null)$Subject$`(Intercept)`

by_subject %>%
```

```
  ggplot(aes(x=random_intercepts)) +
  geom_histogram(bins=20) +
  labs(title = "Random intercepts for each subject",
      x = "Random intercept") +
  scale_x_continuous(limits=c(-100,100)) +
  theme_minimal()
```



This plot looks very similar to the one showing each subject's mean RT. This is exactly what we would expect. Recall that the intercepts are meant to account for subject level variance in Reaction.

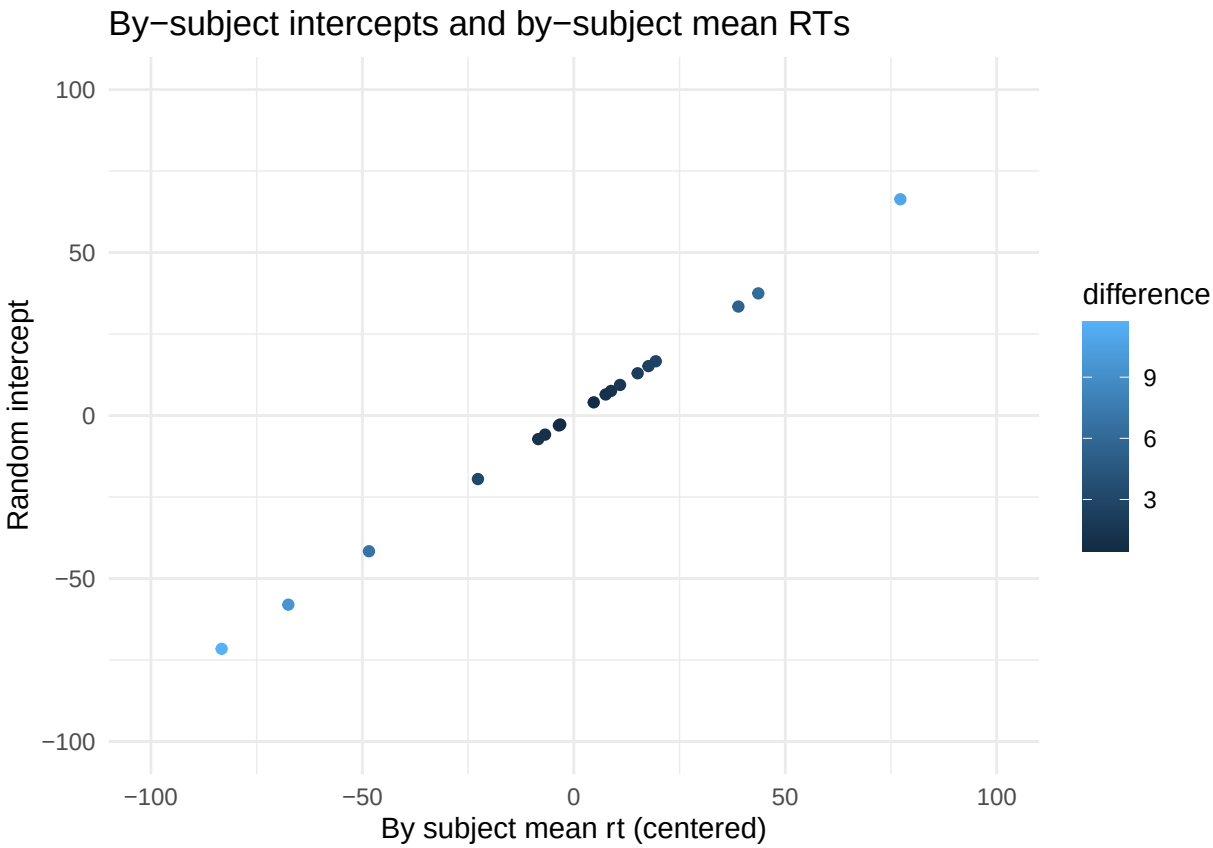
However, the random intercepts are also subject to shrinkage, meaning they're assumed to be drawn from a normal distribution around the group mean. We can illustrate exactly what this looks like by plotting each subjects mean Reaction against that subject's random intercept. Furthermore, we can shade each point in terms of the difference between the subject's random intercept and their mean RT:

```
by_subject$difference <- abs(by_subject$random_intercepts - by_subject$mean_rt_centered)

#this shows:
#1) perfectly linear (as expected)
#2) slight shrinkage (max/min real RT are slightly larger magnitude than random intercept estimates)
#3) the differences between real RT and random intercept grow larger as your approach the ends of the d

by_subject %>%
  ggplot(aes(x=mean_rt_centered,
             y=random_intercepts,
             color=difference)) +
  geom_point() +
```

```
  scale_x_continuous(limits = c(-100,100)) +
  scale_y_continuous(limits = c(-100,100)) +
  labs(title="By-subject intercepts and by-subject mean RTs",
       x="By subject mean rt (centered)",
       y="Random intercept") +
  theme_minimal()
```



Partial pooling with MEM We saw before that no pooling and complete pooling are two ways to approach the dataset. However, they have pros and cons. We can do better with MEM models, since we can pool information from all the groups to improve the estimates of each individual group. This is called **partial pooling**.

```
model <- lmer(Reaction ~ 1 + Days + (1 + Days | Subject), df_sleep)
arm::display(model)
```

```
## lmer(formula = Reaction ~ 1 + Days + (1 + Days | Subject), data = df_sleep)
##             coef.est coef.se
## (Intercept) 252.54     6.43
## Days         10.45     1.54
##
## Error terms:
##  Groups   Name        Std.Dev. Corr
##  Subject  (Intercept) 24.14
##           Days         5.92    0.07
##  Residual             25.48
## ---
## number of obs: 183, groups: Subject, 20
```

```
## AIC = 1783.4, DIC = 1787.8
## deviance = 1779.6
```

The first two coef.est items are the "fixed effects" estimates.

The model also assumes that each participant's individual intercept and slope are deviation from this average, and these random deviations drawn from a distribution of possible intercept and slope parameters. These are randomly varying or random effects. The information in the *Error terms* area describes the distribution of the effects.

Let's visualize these estimates:

```
# Make a dataframe with the fitted effects
df_partial_pooling <- coef(model)[["Subject"]] %>%
  rownames_to_column("Subject") %>%
  as_tibble() %>%
  rename(Intercept = `(Intercept)`, Slope_Days = Days) %>%
  add_column(Model = "Partial pooling")
head(df_partial_pooling)
```

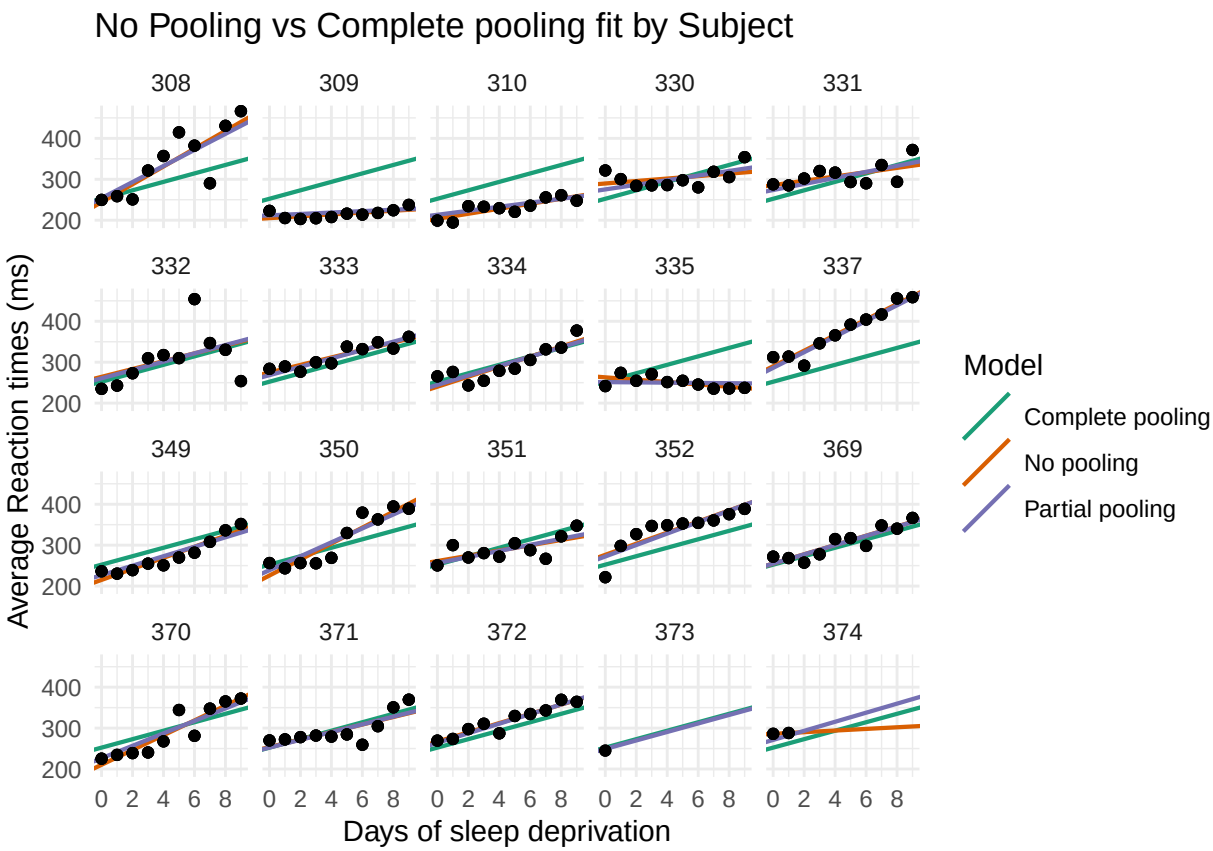
```
## # A tibble: 6 x 4
##   Subject Intercept Slope_Days Model
##   <chr>       <dbl>      <dbl> <chr>
## 1 308          254.      19.6  Partial pooling
## 2 309          212.       1.73 Partial pooling
## 3 310          213.       4.91 Partial pooling
## 4 330          275.       5.64 Partial pooling
## 5 331          274.       7.39 Partial pooling
## 6 332          261.      10.2  Partial pooling
```

Let's update the plot presented in `Chunk 18` of Complete pooling and no pooling models.

```
df_models <- bind_rows(df_pooled, df_no_pooling, df_partial_pooling) %>%
  left_join(df_sleep, by="Subject")
```

```
## Warning in left_join(., df_sleep, by = "Subject"): Detected an unexpected many-to-many relationship b
## i Row 1 of `x` matches multiple rows in `y`.
## i Row 1 of `y` matches multiple rows in `x`.
## i If a many-to-many relationship is expected, set `relationship =
##   "many-to-many"` to silence this warning.
```

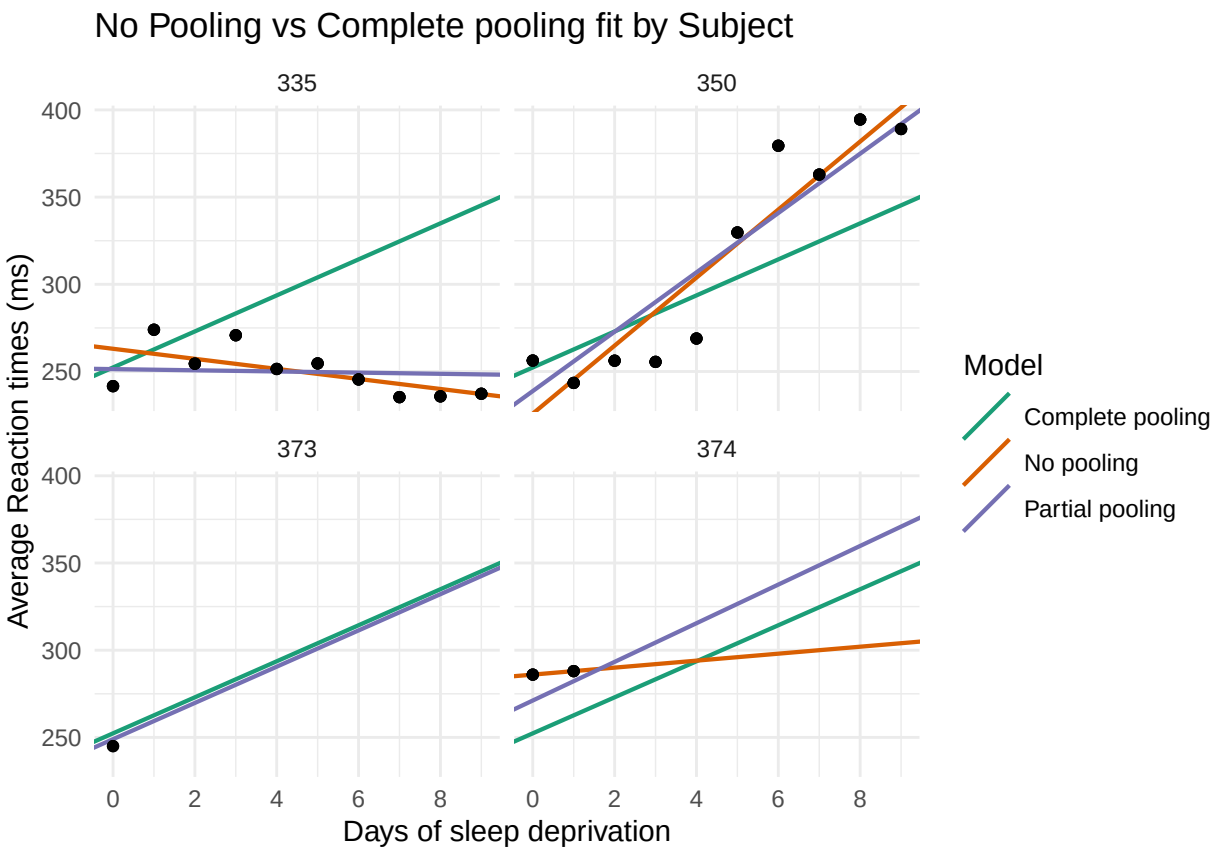
```
p_pooling_comparison %+% df_models
```



Most of the time, the no pooling and partial pooling lines are on top of each other. But when the two differ, it's because the partial pooling model's line is pulled slightly towards the complete-pooling line.

```
df_zoom <- df_models %>%
  filter(Subject %in% c("335", "350", "373", "374"))

p_pooling_comparison %+% df_zoom
```



Shrinkage The partial pooling model pulls more extreme estimates towards an overall average. We can visualize this effect by plotting a scatterplot of intercept and slope parameters from each model and cnnecting estimates for the same partecipant.

First usage of Distinct: - Unique of combination of columns. In this case The complete pooling has a single average value for the intercept and slope_days for every subject

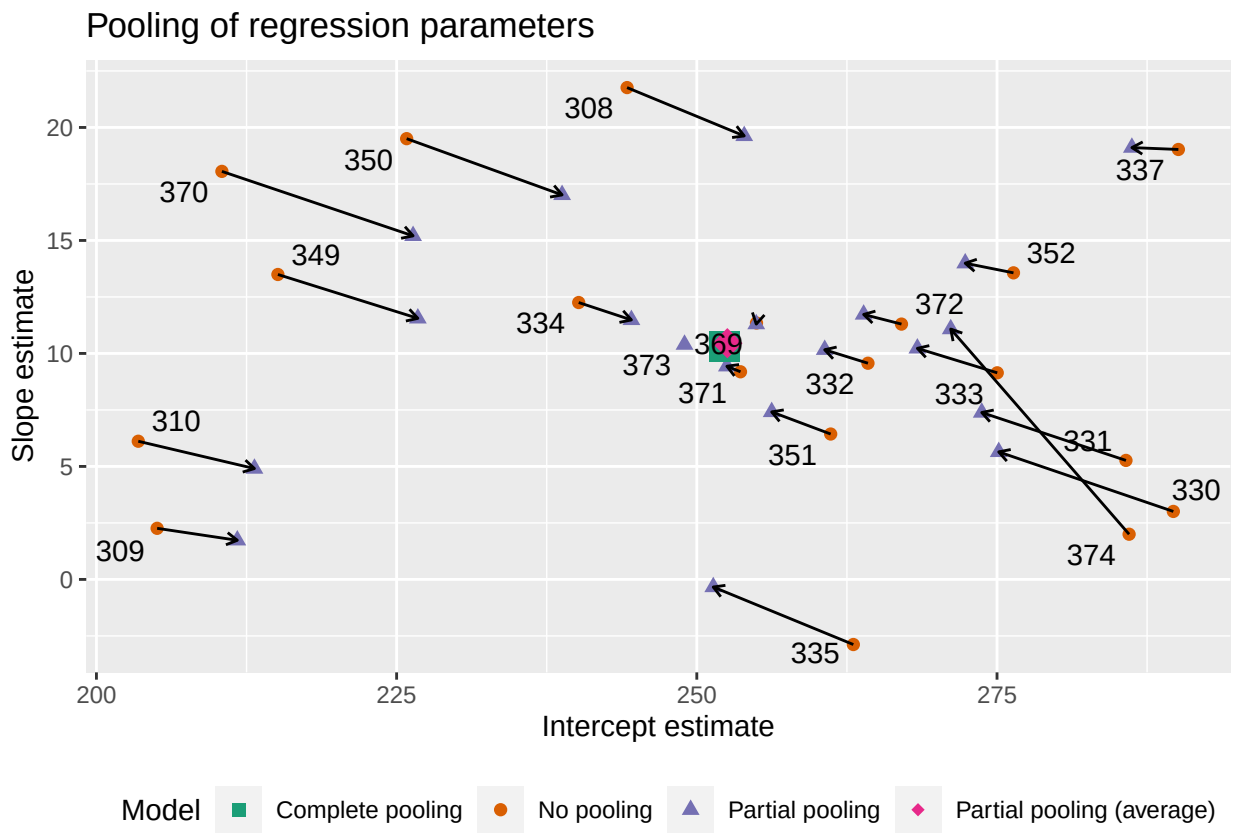
```
df_fixedf <- tibble(
  Model = "Partial pooling (average)",
  Intercept = fixef(model)[1],
  Slope_Days = fixef(model)[2]
)

# Complete pooling / fixed effects are center of gravity in the plot
df_gravity <- df_pooled %>%
  distinct(Model, Intercept, Slope_Days) %>%
  bind_rows(df_fixedf)
df_gravity
```

```
## # A tibble: 2 x 3
##   Model                      Intercept Slope_Days
##   <chr>                          <dbl>      <dbl>
## 1 Complete pooling               252.        10.3
## 2 Partial pooling (average)      253.        10.5
```

```
df_pulled <- bind_rows(df_no_pooling, df_partial_pooling)
df_pulled %>%
  ggplot(aes(x=Intercept,
```

```
            y=Slope_Days,
            color=Model,
            shape=Model)) +
# Draw all points of no pooling and partial pooling
geom_point(size=2) +
# Draw intercept and slope of complete pooling vs partial pooling
geom_point(data = df_gravity,
            size= 5,
            show.legend = FALSE) +
# Draw an arrow connecting the observations between models
geom_path(aes(group=Subject,
              color=NULL),
          arrow=arrow(length=unit(.02,"npc")),
          show.legend=FALSE) +
# Use ggrepel to jitter the labels away from the points for visibility
ggrepel::geom_text_repel(aes(label=Subject,
                             color=NULL),
                         data=df_no_pooling,
                         show.legend = FALSE) +
# Don't forget subject 373, which is not collected by no_pooling
ggrepel::geom_text_repel(aes(label=Subject,
                             color=NULL),
                         data=filter(df_partial_pooling,
                                     Subject == "373"),
                         show.legend = FALSE) +
theme(
  legend.position="bottom",
  legend.justification="right"
) +
labs(title="Pooling of regression parameters",
     x="Intercept estimate",
     y="Slope estimate") +
scale_shape_manual(values = c(15:18)) +
scale_color_brewer(palette = "Dark2")
```



The pull is greater for more extreme values. The lines near that center point are very short. The lines in general get longer as we move away from the complete pooling estimate. The fewer the observations in a cluster, the more information is borrowed from other clusters, and the greater the pull towards the average estimate.

Topographic map of Parameters

We want to visualize the distribution of the randomly varying effects. It's not a routine visualization, but it reveals a little more about where estimates are being pulled towards.

Imagine that the laast plot is a landscape, and fixed effects point is the peak of a hill. What we are going to do is draw a topographic map with contour lines to show different elevation regions on that hill.

First, let's extract the covariance matrix estimated by the model.

```
# Extract the matrix
cov_mat <- VarCorr(model)[["Subject"]]
# Stripp off some details so that just the useful part is printed
attr(cov_mat, "stddev") <- NULL
attr(cov_mat, "correlation") <- NULL
cov_mat
```

```
##             (Intercept)      Days
## (Intercept)  582.717345  9.897673
## Days           9.897673 35.033088
```

This means that the Subject, which is the both for a random intercept and a random slope the grouping variable for the random effects, has a covariance matrix that describes the covariance between the intercept effect and the slope on the Days explanatory variable.

To do so, we draw count lines of confidence regions of the two parameters.

Confidence region: In statistic, a confidence region typically refers to a region in the parameters space where the parameters are likely to lie with a certain level of confidence. The confidence level associated with a pairwise confidence region represents the probability that the region will contain the true values of the parameters of interest in repeated sampling.

```
# Helper function to make a data-frame of ellipse points that includes the levels as a column
make_ellipse <- function(cov_mat, center, level){
  ellipse::ellipse(cov_mat, centre=center, level=level) %>%
    as.data.frame() %>%
    add_column(level=level) %>%
    as_tibble()
}

center <- fixef(model)
# Level:
# The confidence level of a pairwise confidence region. The default is 0.95, for a 95% region. This is
levels <- c(.1, .3, .5, .7, .9)

# Create an ellipse dataframe for each of the levels defined above and combine them
df_ellipse <- levels %>%
  lapply(
    function (x) make_ellipse(cov_mat, center, level=x)
  ) %>%
  bind_rows() %>%
  rename(Intercept = `(Intercept)`, Slope_Days = Days)
head(df_ellipse)
```

```
## # A tibble: 6 x 3
##   Intercept Slope_Days level
##       <dbl>      <dbl> <dbl>
## 1      261.       12.4   0.1
## 2      260.       12.6   0.1
## 3      260.       12.7   0.1
## 4      259.       12.8   0.1
## 5      258.       12.8   0.1
## 6      258.       12.9   0.1
```

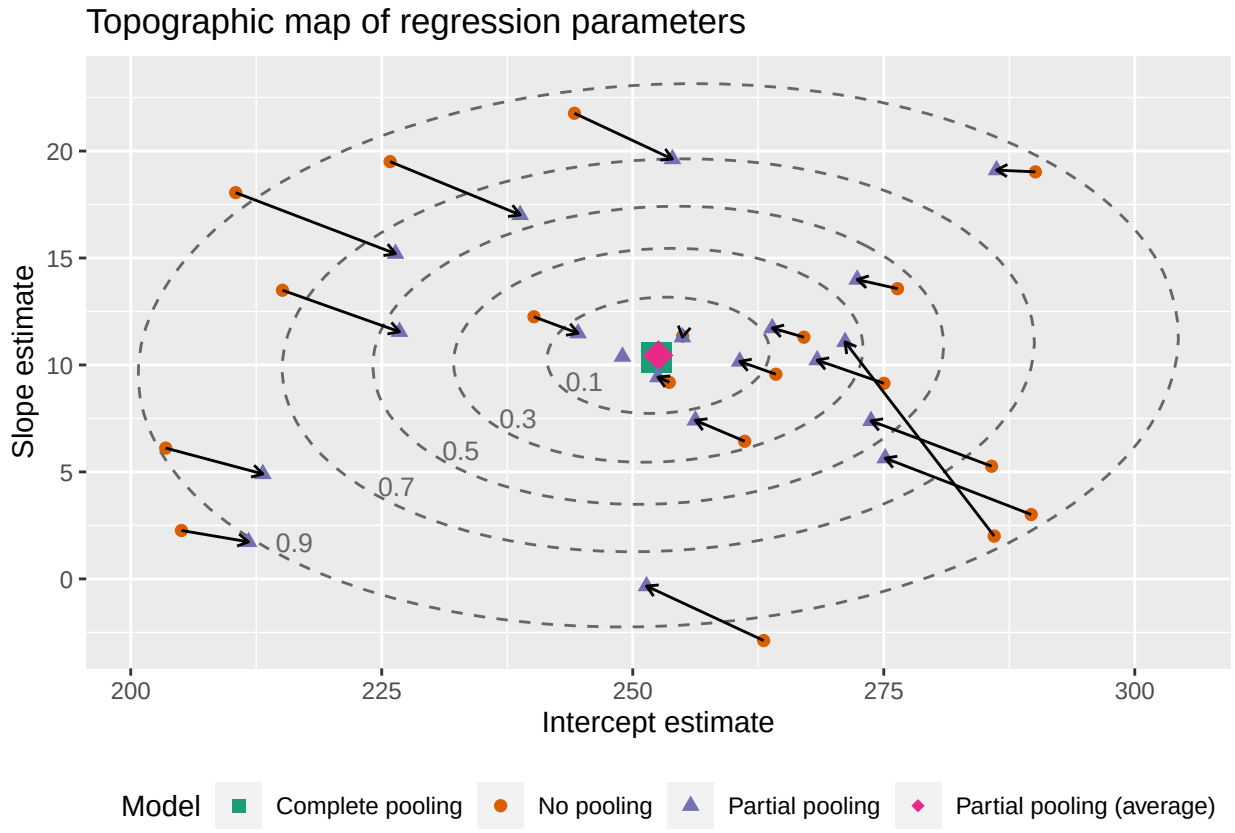
```
# Euclidean distance
contour_dist <- function(xs, ys, center_x, center_y) {
  x_diff <- (center_x - xs) ^ 2
  y_diff <- (center_y - ys) ^ 2
  sqrt(x_diff + y_diff)
}

# Find the point to label in each ellipse.
df_label_locations <- df_ellipse %>%
  group_by(level) %>%
  filter(
    Intercept < quantile(Intercept, .25),
    Slope_Days < quantile(Slope_Days, .25)
  ) %>%
  # Compute distance from center.
  mutate(
    dist = contour_dist(Intercept, Slope_Days, fixef(model)[1], fixef(model)[2])
```

```
  ) %>%
  # Keep smallest values.
  top_n(-1, wt = dist) %>%
  ungroup()

# I also want to add the number of the topological contours evaluated in df_label_locations

ggplot(df_pulled) +
  aes(x = Intercept, y = Slope_Days, color = Model, shape = Model) +
  # Draw contour lines from the distribution of effects
  geom_path(
    aes(group = level, color = NULL, shape = NULL),
    data = df_ellipse,
    linetype = "dashed",
    color = "grey40"
  ) +
  geom_point(
    aes(shape = Model),
    data = df_gravity,
    size = 5,
    show.legend = FALSE
  ) +
  geom_point(size = 2) +
  geom_path(
    aes(group = Subject, color = NULL),
    arrow = arrow(length = unit(.02, "npc")),
    show.legend = FALSE
  ) +
  geom_text(aes(label=level,
                color=NULL,
                shape=NULL),
            data=df_label_locations,
            nudge_x=.5,
            nudge_y=.8,
            size=3.5,
            color="grey40") +
  theme(
    legend.position = "bottom",
    legend.justification = "right"
  ) +
  ggtitle("Topographic map of regression parameters") +
  xlab("Intercept estimate") +
  ylab("Slope estimate") +
  scale_color_brewer(palette = "Dark2") +
  scale_shape_manual(values = c(15:18))
```


Topographic map of regression parameters
Slope estimate
Intercept estimate
20
15
10
5
0
200
225
250
275
300
0.1
0.3
0.5
0.7
0.9
Model
Complete pooling
No pooling
Partial pooling
Partial pooling (average)