Methods 3:

Multilevel Statistical Modeling and Machine Learning

13/12 - 2021

PORTFOLIO 1

Laura Wulff Paaby
202006161
Cognitive Science BA
Aarhus University AU

Portfolio 1, Methods 3, 2021, autumn semester

Sara Krejberg, Niels A. Krogsgaard, Sigurd F. Sørensen, Laura W. Paaby

29/9 - 2021

This assignment has been made in collaboration in Studygroup 9. At each exercise initials will indicate, who has made that particular exercise. However, we must emphasise that almost everything have been done as a group making it hard to distinguish, who has made what specific contributions. The members of the studygroup have the following initials (Name, initials, studynumber): - Laura Wulff Paaby (LWP), 202006161 - Niels Aalund Krogsgaard (NAK), 202008114 - Sara Engsig Krejberg (SEK), 202007949 - Sigurd Fyhn Sørensen (SFS), 202006317

Assignment 1: Using mixed effects modelling to model hierarchical data

In this assignment we will be investigating the *politeness* dataset of Winter and Grawunder (2012) and apply basic methods of multilevel modelling.

Dataset

The dataset has been shared on GitHub, so make sure that the csv-file is on your current path. Otherwise you can supply the full path.

```
politeness <- read.csv('/Users/laura/Desktop/GitHub methods 3/github_methods_3/week_02/politeness.csv') ## read in data
```

Exercises and objectives

The objectives of the exercises of this assignment are:
1) Learning to recognize hierarchical structures within datasets and describing them
2) Creating simple multilevel models and assessing their fitness
3) Write up a report about the findings of the study

REMEMBER: In your report, make sure to include code that can reproduce the answers requested in the exercises below
REMEMBER: This assignment will be part of your final portfolio

Exercise 1 - describing the dataset and making some initial plots

1, (LWP)) Describe the dataset, such that someone who happened upon this dataset could understand the variables and what they contain

The dataset is a result of a study, which investigated the properties of formal and informal speech register. To do so different variables were measured, to enlighten what might characterize the register. The variables are what we see in the dataset: ***f0mn****: the mean frequency of the pitch of the sentence uttered i Hz.* ***scenarios****: the number indicates what specific scenario the subject has been presented with in that observation, e.g. "You are in the professor's office and want to ask for a letter of recommendation" (Grawunder & Winter et al., 2011, p. 2) is an example of a scenario. I must add that this specific scenario was aimed at producing formal speech, while a scenario much the same was aimed at producing informal speech.* ***gender****: the gender of the participant (f = female, m = male)* ***total_duration****: duration of response in seconds ,* ***hiss_count****: the amount of loud hissing breath intake (hiss_count). T* ***attitude****: is either polite or informal, which are variables the scenarios are categorized by. The subjects are the participants of the study - F females whereas M is male.*

```
i. Also consider whether any of the variables in _politeness_ should be encoded as fa
ctors or have the factor encoding removed. Hint: ```?factor```
```

```
#investigating the data
#ls.str(politeness)
glimpse(politeness)
```

```
## Rows: 224
## Columns: 7
## $ subject        <chr> "F1", "F1", "F1", "F1", "F1", "F1", "F1", "F1", "F1", …
## $ gender         <chr> "F", "F", "F", "F", "F", "F", "F", "F", "F", "F", "F",…
## $ scenario       <int> 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 1, 1, 2, 2, …
## $ attitude       <chr> "pol", "inf", "pol", "inf", "pol", "inf", "pol", "inf"…
## $ total_duration <dbl> 18.392, 13.551, 5.217, 4.247, 6.791, 4.126, 6.244, 3.2…
## $ f0mn           <dbl> 214.6, 210.9, 284.7, 265.6, 210.6, 285.6, 251.5, 281.5…
## $ hiss_count     <int> 2, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 2, 0, 0, 0, 0, 0, …
```

```
#making gender, attitude and scenaruo into facter and adding them to the dataframe:
attitude.f = as.factor(politeness$attitude)
gender.f = as.factor(politeness$gender)
scenario.f = as.factor(politeness$scenario)

politeness <- politeness %>%
  mutate(attitude.f, gender.f, scenario.f)
```

2, (NAK)) Create a new data frame that just contains the subject *F1* and run two linear models; one that expresses *f0mn* as dependent on *scenario* as an integer; and one that expresses *f0mn* as dependent on *scenario* encoded as a factor

```
#making a dataframe only for the first subject (F1)
F1_df <- politeness %>%
  filter(subject == 'F1')


## Running the two linear models
#model 1 with scenario as integer:
F1_model1 <- lm(f0mn ~ scenario, data = F1_df)
#model 2 with scenario as factor
F1_model2 <- lm(f0mn ~ scenario.f, data = F1_df)

summary(F1_model1)
```

```
##
## Call:
## lm(formula = f0mn ~ scenario, data = F1_df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -44.836 -36.807   6.686  20.918  46.421
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  262.621     20.616  12.738 2.48e-08 ***
## scenario      -6.886      4.610  -1.494    0.161
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 34.5 on 12 degrees of freedom
## Multiple R-squared:  0.1568, Adjusted R-squared:  0.0865
## F-statistic: 2.231 on 1 and 12 DF,  p-value: 0.1611
```

```
summary(F1_model2)
```

```
##
## Call:
## lm(formula = f0mn ~ scenario.f, data = F1_df)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -37.50 -13.86   0.00  13.86  37.50
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   212.75      20.35  10.453  1.6e-05 ***
## scenario.f2    62.40      28.78   2.168   0.0668 .
## scenario.f3    35.35      28.78   1.228   0.2591
## scenario.f4    53.75      28.78   1.867   0.1041
## scenario.f5    27.30      28.78   0.948   0.3745
## scenario.f6    -7.55      28.78  -0.262   0.8006
## scenario.f7   -14.95      28.78  -0.519   0.6195
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 28.78 on 7 degrees of freedom
## Multiple R-squared:  0.6576, Adjusted R-squared:  0.364
## F-statistic:  2.24 on 6 and 7 DF,  p-value: 0.1576
```

```
i. Include the model matrices, $X$ from the General Linear Model, for these two model
s in your report and describe the different interpretations of _scenario_ that these
entail
```

```
#making a model matrix for each model:

X1 <- model.matrix(F1_model1) #integer model
X2 <- model.matrix(F1_model2) #factor model
```

The design matrix for the model with scenario as an integer take scenario as a continuous variable where going from 2 to 4 is some meaningful doubling. We therefore not only take the scenarios as having some kind of meaningful order, but also take scenario 6 is being double the amount of scenario 3, all in all treating it as a continuous variable (which is of course wrong, since we have no expectation that f0mn will change systematically with increasing scenario number).

The design matrix for the model with scenario as a factor take scenario to be a categorical variable. In the design matrix we can see all the different observations of scenario coded as dummy variables, so every factor level has its own beta-value connected to it. Scenario 1 is "excluded" since that will be the intercept.

Description of both models and matrixes: *the factored model:* The design matrix is a [14x7] matrix, so we will get the following β_{0-6}. This is also shown by the summary of a our linear regression model. *A simple regression f0mn ~ scenario was conducted. Scenario seemed to account for 36.4% of the variance in f0mn

following adjusted R^2. F(1,6) = 2.24, p >0.5) all beta values were insignificant. We only have 14 observations spread out over 7 different levels. So the high p-value is most likely due to sample-size. A further power-analysis could show the required sample size required.

the integer model: Now that scenario is encoded as an integer the design matrix will be a [14x2] matrix. Our model will therefore only give us β_{0-1} and not a β for each level of scenario as done in the previous model. This model assumes that there is a constant increment of f0mn following a "increase" in scenario (if you can even talk about a unit increase of scenario). This would only make sense if scenarios were ordered as getting harder and harder. The model is again f0mn ~ scenario F(1,12) = 2.231, p>0.5) with an adjusted R^2 = 0.0865 showing an explained variance of 8.65% (β_1 = -6.886, SE = 4.6, t = -1.5, p>0.16.) Again such a small sample size might be tricky to work with.

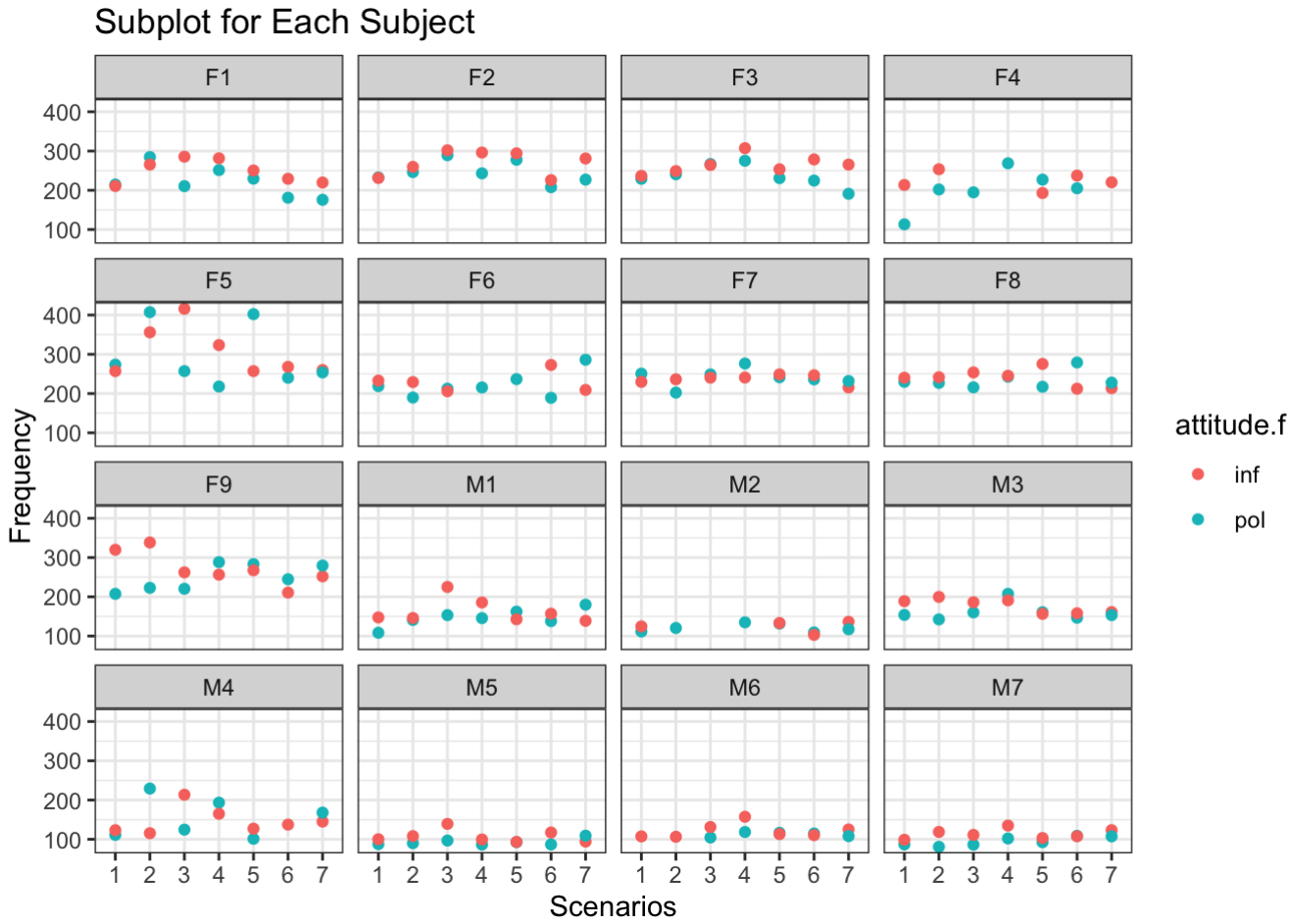
```
ii. Which coding of _scenario_, as a factor or not, is more fitting?
```

In this context it is only appropriate to code scenario as a factor. The reasons are given in the previous exercise.

3, (SEK)) Make a plot that includes a subplot for each subject that has *scenario* on the x-axis and *f0mn* on the y-axis and where points are colour coded according to *attitude* i. Describe the differences between subjects

```
politeness %>%
    ggplot(aes(scenario.f, f0mn, color = attitude.f)) + geom_point() +
    facet_wrap(~subject) +
  theme_bw() +
  xlab("Scenarios") +
  ylab("Frequency") +
  ggtitle("Subplot for Each Subject")
```

```
## Warning: Removed 12 rows containing missing values (geom_point).
```



There seem to be a lower baseline/intercept given that you're a male. Attitude doesn't seem to have an large effect on f0mn. So an idea could be to add Gender as a fixed effect and subject as a random intercept as there is also individual variance within the gender category.

Exercise 2 - comparison of models

1, (SFS)) Build four models and do some comparisons i. a single level model that models *f0mn* as dependent on *gender* ii. a two-level model that adds a second level on top of i. where unique intercepts are modelled for each *scenario* iii. a two-level model that only has *subject* as an intercept iv. a two-level model that models intercepts for both *scenario* and *subject* v. which of the models has the lowest residual standard deviation, also compare the Akaike Information Criterion `AIC` ? vi. which of the second-level effects explains the most variance?

```
#i
model1 <- lm(f0mn ~ gender.f, data = politeness)

#ii
model2 <- lmer(f0mn ~ gender.f + (1 | scenario.f), data = politeness, REML = FALSE)

#iii
model3 <- lmer(f0mn ~ gender.f + (1 | subject), data = politeness, REML = FALSE)

#iv
model4 <- lmer(f0mn ~ gender.f + (1 | scenario.f) + (1|subject), data = politeness, R
EML = FALSE)
```

Comparison of models by the Akaike Information Criterion:

```
# v
AIC(model1, model2, model3, model4)
```

	df <dbl>	**AIC** <dbl>
model1	3	2163.971
model2	4	2162.257
model3	4	2112.048
model4	5	2105.176

4 rows

Comparing the residual standard deviation of the models:

```
#v maybe gather everything in a table
sigma(model1)
```

```
## [1] 39.46268
```

```
sigma(model2)
```

```
## [1] 38.3546
```

```
sigma(model3)
```

```
## [1] 32.04227
```

```
sigma(model4)
```

```
## [1] 30.66355
```

Looking at both the standard deviation and the information criterion, we find that the model4 is the best performing model, since it has the smallest value both in AIC and RSD.

```
#vi the most variance explained by the effects (scenario or subject):
pacman::p_load(MuMIn)

r.squaredGLMM(model2)
```

```
## Warning: 'r.squaredGLMM' now calculates a revised statistic. See the help page.
```

```
##            R2m       R2c
## [1,] 0.6817304 0.6965456
```

```
r.squaredGLMM(model3)
```

```
##            R2m       R2c
## [1,] 0.6798832 0.7862932
```

```
r.squaredGLMM(model4)
```

```
##            R2m       R2c
## [1,] 0.6787423 0.8045921
```

model2 showed the best variance explained purely by fixed effects, 68,17%, with scenario as a random intercept. We can conclude in model3 that adding subject as random intercept rather than scenario explains more of the variance but also has more shared variance with our fixed effect gender. Model4 (f0mn ~ gender + (1|scenario) + (1|subject)) showed most explained variance with 80% of the variance being accounted for by both fixed and random effects.

2, (LWP)) Why is our single-level model bad? *(the single level model is bad, since it violates the most important assumption of independence)*

```
i. create a new data frame that has three variables, _subject_, _gender_ and _f0mn_,
where _f0mn_ is the average of all responses of each subject, i.e. averaging across _
attitude_ and_scenario_
```

```
#making a new dataframe with the selected variables:
politeness_sel <- politeness %>%
  filter(!is.na(f0mn)) %>% #making sure there is no NA in the new df
  select(f0mn,attitude,subject) %>%
  group_by(subject) %>%
  summarise(f0mn_mean = mean(f0mn))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

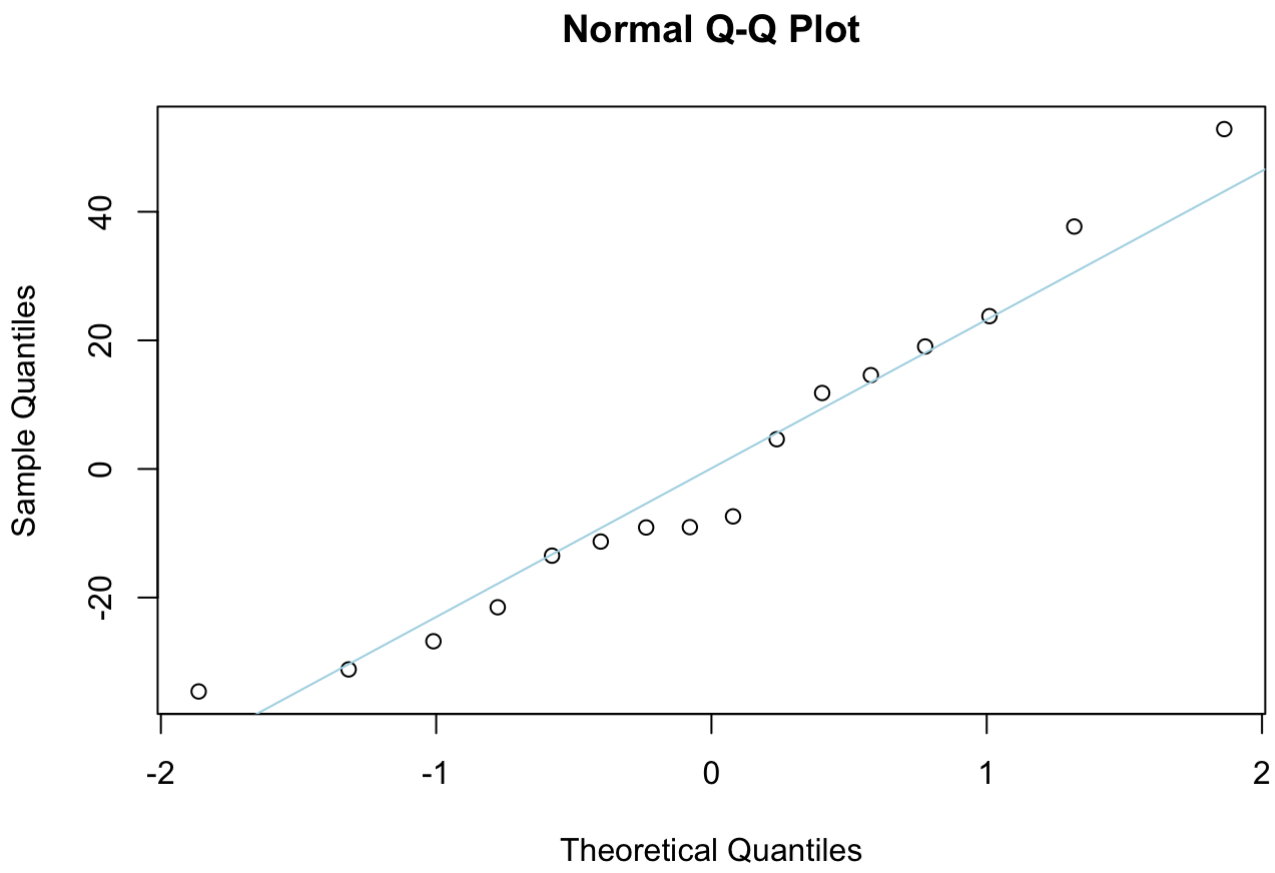
```
politeness_sel <- politeness_sel %>% #adding the gender to the dataframe
  mutate(gender = if_else(grepl("F", politeness_sel$subject, ignore.case = T),"F","M"
)) %>%
  mutate(gender = as.factor(gender))
```

ii. build a single-level model that models _f0mn_ as dependent on _gender_ using this new dataset

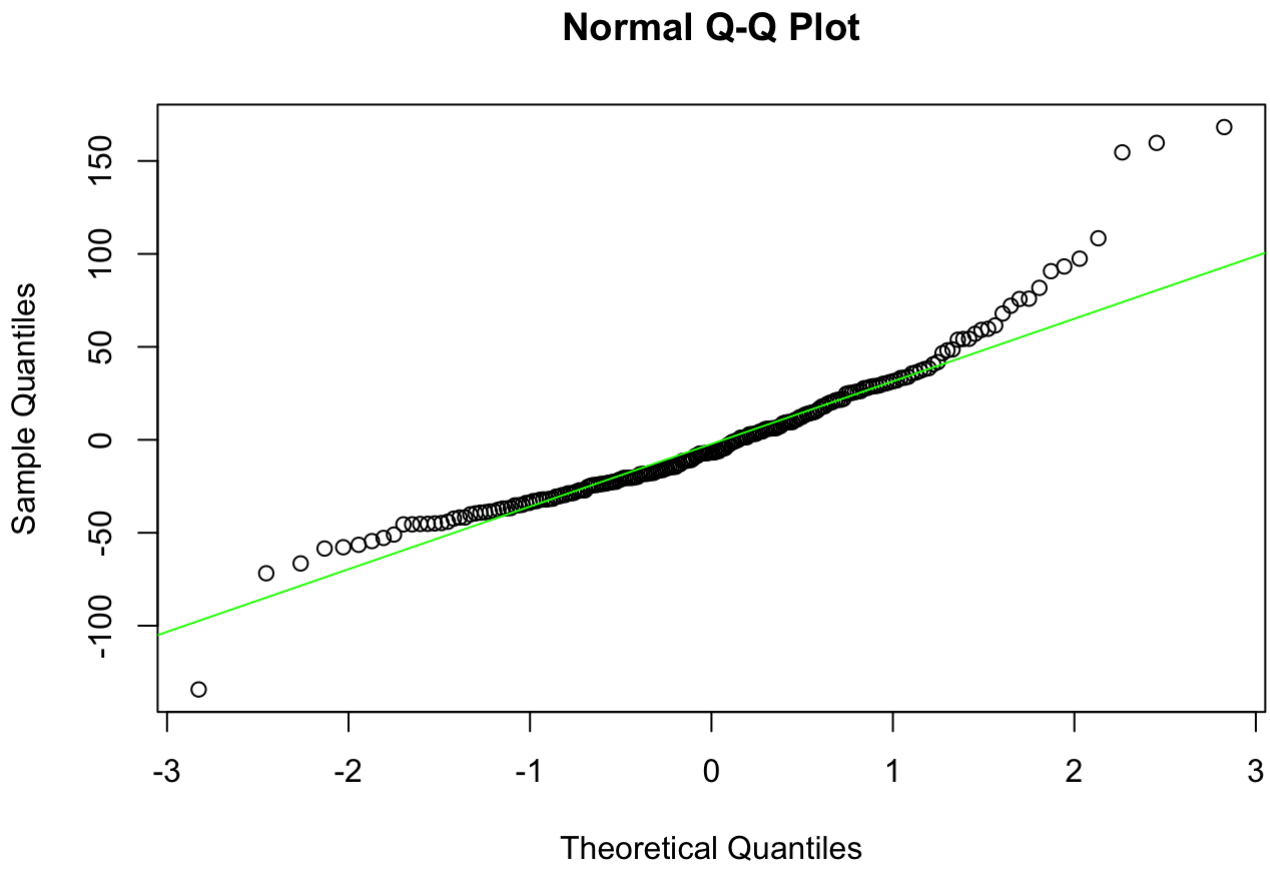
```
#builing single-level model
ms <- lm(f0mn_mean ~ gender, data = politeness_sel)
```

iii. make Quantile-Quantile plots, comparing theoretical quantiles to the sample quantiles) using `qqnorm` and `qqline` for the new single-level model and compare it to the old single-level model (from 1).i). Which model's residuals (ϵ) fulfil the assumptions of the General Linear Model better?)

```
#the new single model
qqnorm(resid(ms))
qqline(resid(ms), col = 'lightblue')
```



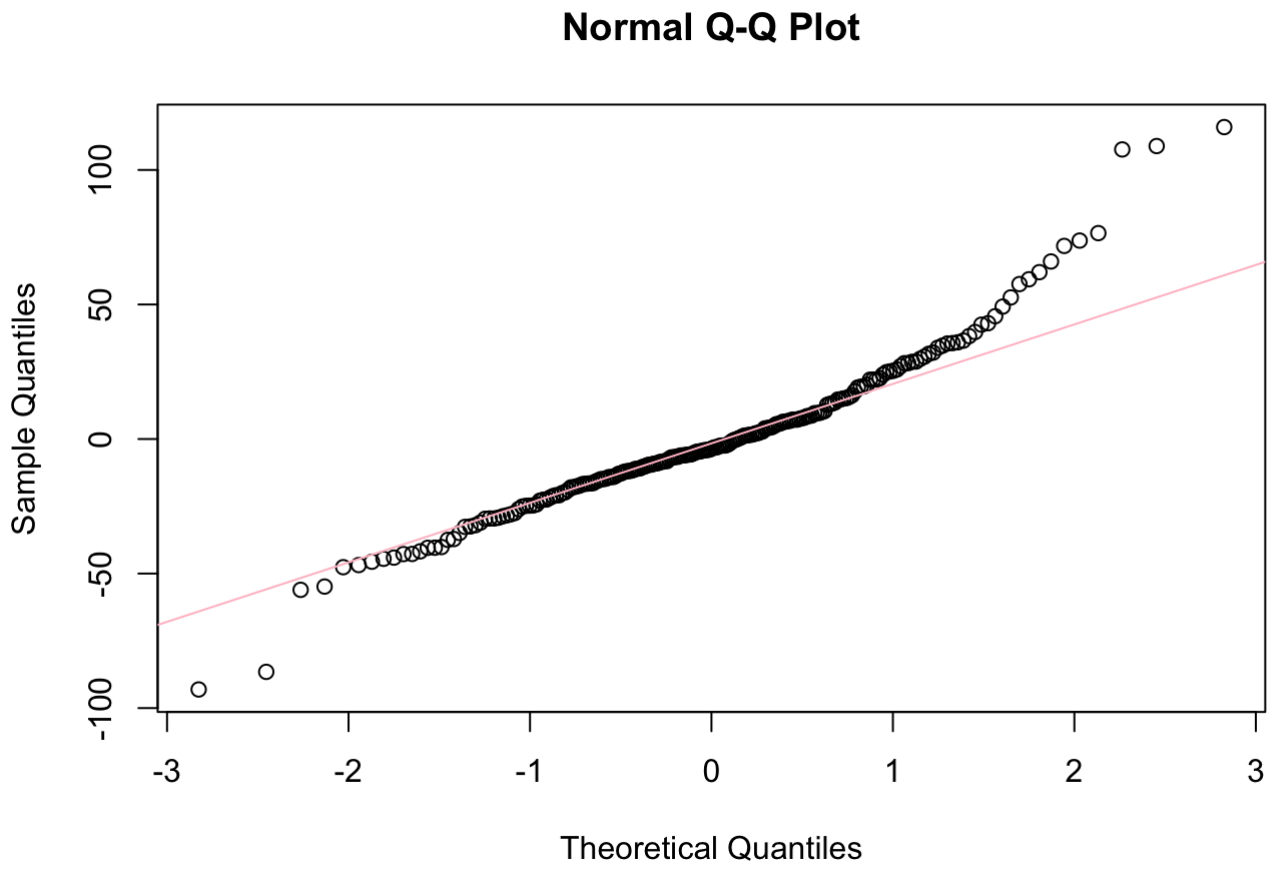

```
#The old single model
qqnorm(resid(model1))
qqline(resid(model1), col = 'green')
```



Looking at the data we how the ms model doesn't fit the line very well, however it does not seemed skewed. The model1 seems a bit skewed, and fits the line worse. This could properly have been fixed by trimming the data/remove outliers.

```
iv. Also make a quantile-quantile plot for the residuals of the  multilevel model wit
h two intercepts. Does it look alright?
```

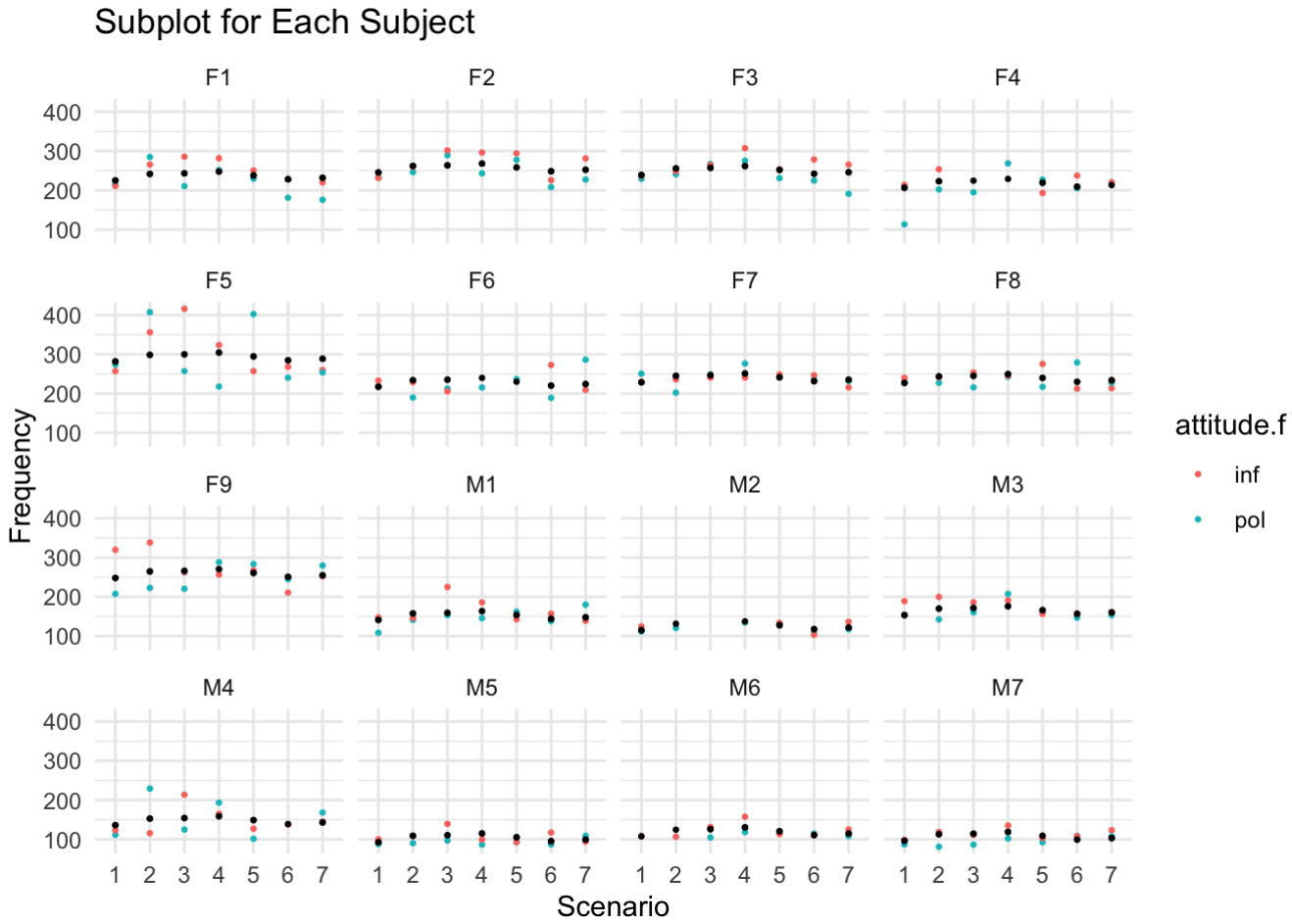
```
#The multilevel model (model 4)
qqnorm(resid(model4))
qqline(resid(model4), col = 'pink')
```



In a perfect world, this model would have made the datapoints fit the line better. This doesn't seem to be the case, and the residuals are still right skewed. They don't follow the normal distribution perfectly. However this is the least important of the assumptions, (normality of residuals).

3, (NAK)) Plotting the two-intercepts model i. Create a plot for each subject, (similar to part 3 in Exercise 1), this time also indicating the fitted value for each of the subjects for each for the scenarios (hint use `fixef` to get the "grand effects" for each gender and `ranef` to get the subject- and scenario-specific effects)

```
fitted <- fitted(model4) #making the fitted values

politeness_una <- politeness %>%
  filter(!is.na(f0mn)) %>%  #making sure we have no NA's
  mutate(fitted) #adding the fitted values to the dataset

politeness_una %>%
  ggplot(aes(scenario.f, f0mn, color = attitude.f))+
  geom_point(size = 0.5)+
  geom_point(aes(y = fitted), colour = 'black', size = 0.5)+
  facet_wrap(~subject) +
  theme_minimal()+
  xlab("Scenario")+
  ylab('Frequency') +
  ggtitle("Subplot for Each Subject")
```



Exercise 3 - now with attitude

1, (SEK)) Carry on with the model with the two unique intercepts fitted (*scenario* and *subject*). i. now build a model that has *attitude* as a main effect besides *gender*

```
# the model to carry on with: model4 <- lmer(f0mn ~ gender.f + (1 | scenario.f) + (1|subject), data = politeness)
#the new model with both gender and attitude:
model5 <- lmer(f0mn ~ gender.f + attitude.f + (1|scenario.f)+(1|subject), data = politeness, REML = FALSE)
```

```
ii. make a separate model that besides the main effects of _attitude_ and _gender_ also include their interaction
```

```
model6 <- lmer(f0mn ~ gender.f*attitude.f + (1|scenario.f)+(1|subject), data = politeness, REML = FALSE)
summary(model6)
```

```
## Linear mixed model fit by maximum likelihood . t-tests use Satterthwaite's
##   method [lmerModLmerTest]
## Formula: f0mn ~ gender.f * attitude.f + (1 | scenario.f) + (1 | subject)
##    Data: politeness
##
##      AIC      BIC   logLik deviance df.resid
##   2096.0   2119.5  -1041.0   2082.0      205
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -2.8460 -0.5893 -0.0685  0.3946  3.9518
##
## Random effects:
##  Groups     Name        Variance Std.Dev.
##  subject    (Intercept) 514.09   22.674
##  scenario.f (Intercept)  99.08    9.954
##  Residual               876.46   29.605
## Number of obs: 212, groups:  subject, 16; scenario.f, 7
##
## Fixed effects:
##                         Estimate Std. Error       df t value Pr(>|t|)
## (Intercept)              255.632      9.289   23.556  27.521  < 2e-16 ***
## gender.fM               -118.251     12.841   19.922  -9.209 1.28e-08 ***
## attitude.fpol            -17.198      5.395  190.331  -3.188  0.00168 **
## gender.fM:attitude.fpol    5.563      8.241  190.388   0.675  0.50049
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) gndr.M atttd.
## gender.fM   -0.605
## attitud.fpl -0.299  0.216
## gndr.fM:tt.  0.195 -0.323 -0.654
```

```
iii. describe what the interaction term in the model says about Korean men's pitch wh
en they are polite relative to Korean women's pitch when they are polite (you don't h
ave to judge whether it is interesting)
```

Understanding the output of the model:

When males are asked to be polite, their pitch will according to this be higher. The intercepts is for the female, when uttering the statement informal, where they here have the average pitch of 255 hz. GenderfM is then when we go from female to male on the x ax, we see how the average pitch decrease with 118 hz. attitudef.pol, when we go from informal to polite does the average (of both females and males) pitch decrease with 17 hz. This is why need the interaction, so we can consider more than just the average genderM:attitudepol: this is the interaction between gender and attitude, and it indicates that it decreases 5,5hz less for men than women. This means that the change in pitch for men are on -17.192+5.54 = -11.652, whereas the womens changes with -17.192 hz. Summarizingly, both men

and women decrease their pitch when going from informal to polite, but the male pitch does not decrease as much the women. (for the reader: the f just means that it is factors, a bit confusing considering the females - but this is not the case!)

Reporting the model: *The model f0mn ~ attitude:gender + (1|subject)+ (1|scenario)has an* R^2c *0.81 both attitude and gender showed a significant effect on f0mn (*β_1 *(attitude_pol) = -17.2, SE = 5.4, p>0.05) and (*β_2 *(genderM) = -119, SE = 12.8, p>0.05). Being polite and male lowers your frequency. Being both Male and Polite has an interaction effect of (*β_3 *= 5.5, SE = 8.24, p<0.05). Hereby concluding that there is a small positive insignificant interaction effect of being male and polite. The SE being proportional large compared to the effect size makes it very difficult to say anything meaningful.*

2, (SFS)) Compare the three models (1. gender as a main effect; 2. gender and attitude as main effects; 3. gender and attitude as main effects and the interaction between them. For all three models model unique intercepts for *subject* and *scenario*) using residual variance, residual standard deviation and AIC.

```
#model4: gender as main effect
summary(model4)
```

```
## Linear mixed model fit by maximum likelihood . t-tests use Satterthwaite's
##   method [lmerModLmerTest]
## Formula: f0mn ~ gender.f + (1 | scenario.f) + (1 | subject)
##    Data: politeness
##
##      AIC      BIC   logLik deviance df.resid
##   2105.2   2122.0  -1047.6   2095.2      207
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -3.0357 -0.5384 -0.1177  0.4346  3.7808
##
## Random effects:
##  Groups     Name        Variance Std.Dev.
##  subject    (Intercept) 516.19   22.720
##  scenario.f (Intercept)  89.36    9.453
##  Residual               940.25   30.664
## Number of obs: 212, groups:  subject, 16; scenario.f, 7
##
## Fixed effects:
##             Estimate Std. Error       df t value Pr(>|t|)
## (Intercept)  246.778      8.829   19.248  27.952  < 2e-16 ***
## gender.fM   -115.186     12.223   16.011  -9.424 6.19e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##           (Intr)
## gender.fM -0.604
```

```
#model5: gender and attitudes as main effects
summary(model5)
```

```
## Linear mixed model fit by maximum likelihood . t-tests use Satterthwaite's
##   method [lmerModLmerTest]
## Formula: f0mn ~ gender.f + attitude.f + (1 | scenario.f) + (1 | subject)
##    Data: politeness
##
##      AIC      BIC   logLik deviance df.resid
##   2094.5   2114.6  -1041.2   2082.5      206
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -2.8791 -0.5968 -0.0569  0.4260  3.9068
##
## Random effects:
##  Groups     Name        Variance Std.Dev.
##  subject    (Intercept) 514.92   22.692
##  scenario.f (Intercept)  99.22    9.961
##  Residual               878.39   29.638
## Number of obs: 212, groups:  subject, 16; scenario.f, 7
##
## Fixed effects:
##               Estimate Std. Error       df t value Pr(>|t|)
## (Intercept)    254.408      9.117   21.800  27.904  < 2e-16 ***
## gender.fM     -115.447     12.161   16.000  -9.494 5.63e-08 ***
## attitude.fpol  -14.817      4.086  190.559  -3.626 0.000369 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) gndr.M
## gender.fM   -0.583
## attitud.fpl -0.231  0.006
```

```
#model6: gender and attitude as main effects and with an interaction between them
summary(model6)
```

```
## Linear mixed model fit by maximum likelihood . t-tests use Satterthwaite's
##   method [lmerModLmerTest]
## Formula: f0mn ~ gender.f * attitude.f + (1 | scenario.f) + (1 | subject)
##    Data: politeness
##
##      AIC      BIC   logLik deviance df.resid
##   2096.0   2119.5  -1041.0   2082.0      205
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -2.8460 -0.5893 -0.0685  0.3946  3.9518
##
## Random effects:
##  Groups     Name        Variance Std.Dev.
##  subject    (Intercept) 514.09   22.674
##  scenario.f (Intercept)  99.08    9.954
##  Residual               876.46   29.605
## Number of obs: 212, groups:  subject, 16; scenario.f, 7
##
## Fixed effects:
##                         Estimate Std. Error       df t value Pr(>|t|)
## (Intercept)              255.632      9.289   23.556  27.521  < 2e-16 ***
## gender.fM               -118.251     12.841   19.922  -9.209 1.28e-08 ***
## attitude.fpol            -17.198      5.395  190.331  -3.188  0.00168 **
## gender.fM:attitude.fpol    5.563      8.241  190.388   0.675  0.50049
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) gndr.M atttd.
## gender.fM   -0.605
## attitud.fpl -0.299  0.216
## gndr.fM:tt.  0.195 -0.323 -0.654
```

```
#comparison by AIC:
AIC(model4, model5, model6)
```

	df	AIC
	<dbl>	<dbl>
model4	5	2105.176
model5	6	2094.489
model6	7	2096.034

3 rows

```
#comparing by standard deviation of residuals
sigma(model4)
```

```
## [1] 30.66355
```

```
sigma(model5)
```

```
## [1] 29.63771
```

```
sigma(model6)
```

```
## [1] 29.60505
```

```
#comparing by the residual variance:
sum(residuals(model4)^2)
```

```
## [1] 181913
```

```
sum(residuals(model5)^2)
```

```
## [1] 169681.1
```

```
sum(residuals(model6)^2)
```

```
## [1] 169305.6
```

Considering the output of the comparisons, we suggest model 5: it is the simpler model and adding the interaction effect (model 6) makes almost no explanatory power, while being more complex.

3, (LWP, NAK, SEK, SFS)) Choose the model that you think describe the data the best - and write a short report on the main findings based on this model. At least include the following: i. describe what the dataset consists of
The dataset used in this model consists of subject id, binary gender indication (F or M), scenario index (from 1 to 7 depending on what the scenario was), a variable indicating whether the text should be spoken in an formal/polite or informal tone, and a variable called f0mn basically stating the average frequency of the utterance in Hz. Besides these the data also consisted of total duration of utterances in seconds and count of hissing sounds but these are not relevant for the optimal model.

ii. what can you conclude about the effect of gender and attitude on pitch (if anything)?
f0mn was found to be significantly modulated by gender.
$\beta_2 = -115, SE = 12.16, p < 0.05$ *Attitude also showed a significant modulating of f0mn* $\beta_1 = -14.8, SE = 4, p < 0.05$

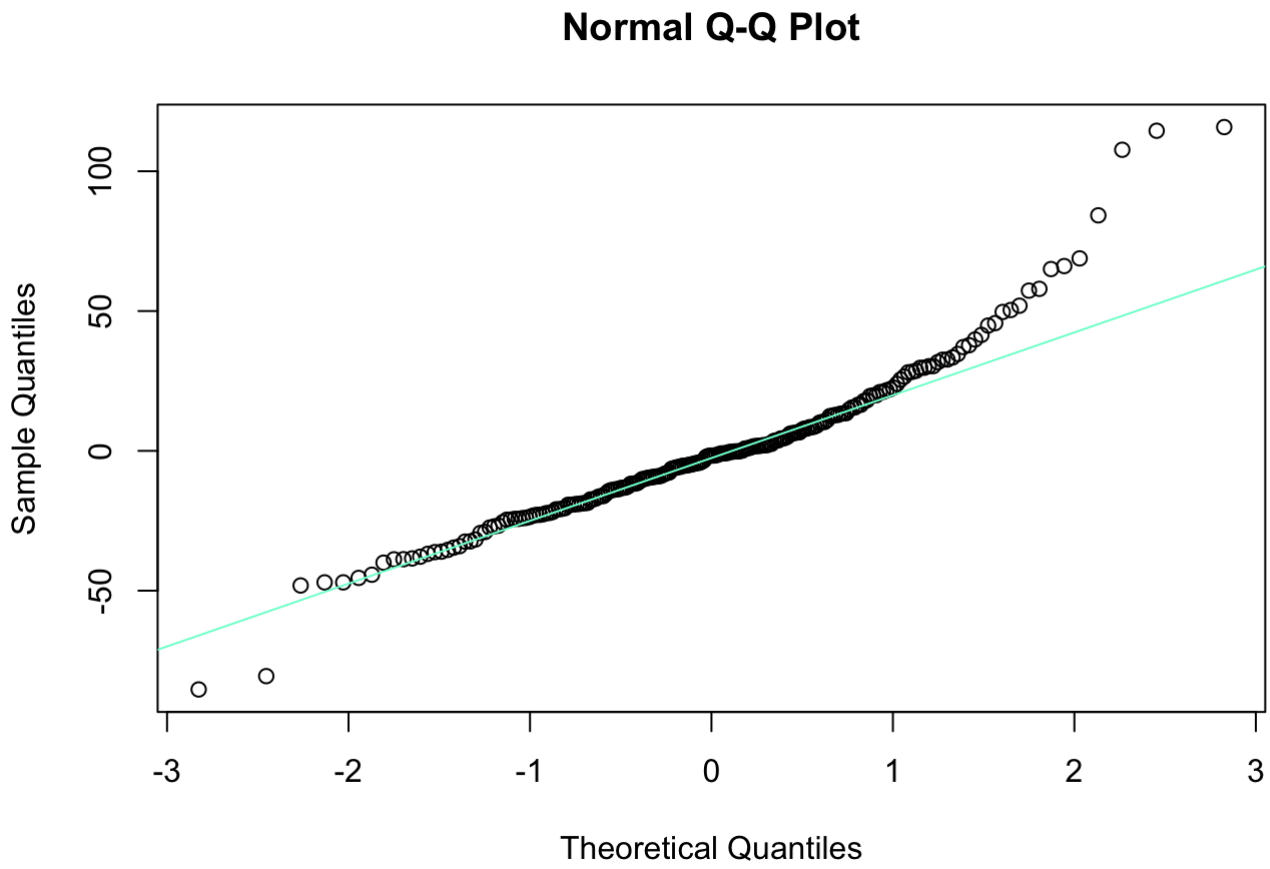
iii. motivate why you would include separate intercepts for subjects and scenarios (if you think they should be included)
Subjects: *these are only a sample of the total population. Because subject does not exhaust the population of interest (e.g. the whole Korean population) it should be modeled as a random effect. Also, each subject will express random variation caused by individual baselines and individual effects of formal vs. informal situation.*

Scenario: *Again, these scenarios does not exhaust the number of formal or informal scenarios that exist. It should be modeled as a random effect since we have no expectation of how the individual scenario will affect the pitch compared to the other scenarios. There are no preconceptions about any systematic differences between the scenarios, making them have idiosyncratic and random effects on pitch.*

iv. describe the variance components of the second level (if any)
Both fixed and random effects accounted for roughly 82% of the variance in the f0mn variable with random effects proportion being 12.7%. Visual inspection shows that both the qqplot and histogram violates the assumption of a mixed effect linear model. The more robust generalized mixed effect model with a link function would be preferred. But as it was not the task such model was not constructed.

v. include a Quantile-Quantile plot of your chosen model

```
qqnorm(resid(model5))
qqline(resid(model5), col = 'aquamarine')
```



We used R (R Core Team, 2019) and lmerTest (Kuznetsova, Brockhoff and Christensen, 2017) to perform a linear mixed effects analysis of the relationship between f0mn, gender and attitude. As random effects, we had intercepts for subjects, and scenario.

PORTFOLIO 2.1

Laura Wulff Paaby
202006161
Cognitive Science BA
Aarhus University AU

13/12 - 2021

Portfolio 2.1, Methods 3, 2021, autumn semester

Laura W. Paaby

4/9 - 2021

```
#loading libraries:
pacman::p_load(tidyverse, readbulk, patchwork, lmerTest, ggpubr, dfoptim)
```

added new line

Exercises and objectives

The objectives of the exercises of this assignment are:
1) Download and organise the data and model and plot staircase responses based on fits of logistic functions
2) Fit multilevel models for response times
3) Fit multilevel models for count data

REMEMBER: In your report, make sure to include code that can reproduce the answers requested in the exercises below (**MAKE A KNITTED VERSION**)
REMEMBER: This assignment will be part of your final portfolio

Exercise 1

Go to https://osf.io/ecxsj/files/ (https://osf.io/ecxsj/files/) and download the files associated with Experiment 2 (there should be 29).
The data is associated with Experiment 2 of the article at the following DOI https://doi.org/10.1016/j.concog.2019.03.007 (https://doi.org/10.1016/j.concog.2019.03.007)

1. Put the data from all subjects into a single data frame
2. Describe the data and construct extra variables from the existing variables
    i. add a variable to the data frame and call it *correct* (have it be a *logical* variable). Assign a 1 to each row where the subject indicated the correct answer and a 0 to each row where the subject indicated the incorrect answer (**Hint:** the variable *obj.resp* indicates whether the subject answered “even”, *e* or “odd”, *o*, and the variable *target_type* indicates what was actually presented.

```
#the target.type and the obj.response should match
data$correct <- ifelse((data$obj.resp == "o" & data$target.type =="odd") | (data$obj.
resp == "e" & data$target.type =="even"), 1,0)

#skimming the data to see the classes of the variables
ls.str(data)
```

```
## correct :  num [1:18131] 1 1 1 1 1 1 1 1 1 1 ...
## cue :  num [1:18131] 29 29 29 29 29 29 29 29 29 29 ...
## even.digit :  num [1:18131] 8 4 4 4 4 4 4 4 4 8 ...
## File :  chr [1:18131] "001.csv" "001.csv" "001.csv" "001.csv" "001.csv" "001.csv"
...
## jitter.x :  num [1:18131] -0.3425 0.0623 -0.4058 -0.3615 0.2891 ...
## jitter.y :  num [1:18131] 0.4489 0.0291 0.4995 -0.2224 0.4132 ...
## obj.resp :  chr [1:18131] "o" "e" "e" "o" "e" "e" "o" "o" "e" "o" "o" "e" "o" "o"
"o" ...
## odd.digit :  num [1:18131] 9 9 7 7 7 7 7 7 9 9 ...
## pas :  num [1:18131] 4 4 4 4 4 4 4 4 4 4 ...
## rt.obj :  num [1:18131] 1.158 2.456 0.951 0.62 0.95 ...
## rt.subj :  num [1:18131] 1.754 1.39 0.686 0.653 0.582 ...
## seed :  num [1:18131] 9817 9817 9817 9817 9817 ...
## subject :  chr [1:18131] "001" "001" "001" "001" "001" "001" "001" "001" "001" "00
1" ...
## target.contrast :  num [1:18131] 1 1 0.9 0.9 0.8 0.8 0.5 0.5 0.3 0.3 ...
## target.frames :  num [1:18131] 3 3 3 3 3 3 3 3 3 3 ...
## target.type :  chr [1:18131] "odd" "even" "even" "odd" "even" "even" "odd" "odd"
"even" ...
## task :  chr [1:18131] "pairs" "pairs" "pairs" "pairs" "pairs" "pairs" "pairs" ...
## trial :  num [1:18131] 0 1 2 3 4 5 6 7 8 9 ...
## trial.type :  chr [1:18131] "staircase" "staircase" "staircase" "staircase" "stair
case" ...
```

```
ii. describe what the following variables in the data frame contain, _trial.type_, _p
as_, _trial_, _target.contrast_, _cue_, _task_, _target_type_, _rt.subj_, _rt.obj_, _
obj.resp_, _subject_ and _correct_. (That means you can ignore the rest of the variab
les in your description). For each of them, indicate and argue for what `class` they
should be classified into, e.g. _factor_, _numeric_ etc.
```

TRIAL TYPE: *trial type is at the moment a character, but should be a factor, so that the performance can be modeled and compared by which trial was done*

```
#trial_type:
data$trial.type <- as.factor(data$trial.type)
```

PAS: *is a subjective rating and indicates whether you have seen the stimulus or not: 1 (no experience), 2 (weak glimpse), 3 (almost clear experience), 4 (clear experience). It is measures on the Perceptual Awareness Scale, and should be a factor since it is a categorical ordered variable.*

```
#pas:
data$pas <- as.factor(data$pas)
```

TARGET CONTRAST: *is the contrast in the stimulus shown relevant to the background and adjusted to each participants threshold. It is numeric, and should stay in that way since*

```
#target.contrast:
data$target.contrast <- as.numeric(data$target.contrast)
```

TRIAL: *number of trial, which reset as the experiment begins. It is nominal data, and should be encoded as a factor.*

```
#trial:
data$trial <- as.factor(data$trial)
```

CUE: *is 36 different combinations of numbers participants can be cued with, 3 types depending on the task setting. Again nominal, so factor.*

```
#cue:
data$cue <- as.factor(data$cue)
```

TASK: *is a character and should be a factor, since it is nominal. It is the task setting, and can be either singles, paired or quadruplets referring to the amount of letters showing in the cue.*

```
#task:
data$task <- as.factor(data$task)
```

TARGET TYPE: *Indicates whether the target was an even or odd number. It is binary data, and should be coded as a factor.*

```
#target_type:
data$target.type <- as.factor(data$target.type)
```

RT.SUBJ: *reaction time of the rating of the pass - their confidence in how clear they saw the target. This is numeric data and should stay that way, since we are dealing with reacting time, which here is a continuous variable.*

RT.OBJ: *reaction time of the answer on the task. Same as rt.subj*

OBJ.RESP: *the subject's to if the letter is odd or even, thus binomial and should be a factor*

```
#obj.resp:
data$obj.resp <- as.factor(data$obj.resp)
```

SUBJECT: *Participant index and should be a factor*

```
#subject:
data$subject <- as.factor(data$subject)
```

CORRECT: *indicates whether the subject answered correct 1 or wrong 0 when answering whether the letter was odd or even. It is binomial, thus it should be a factor.*

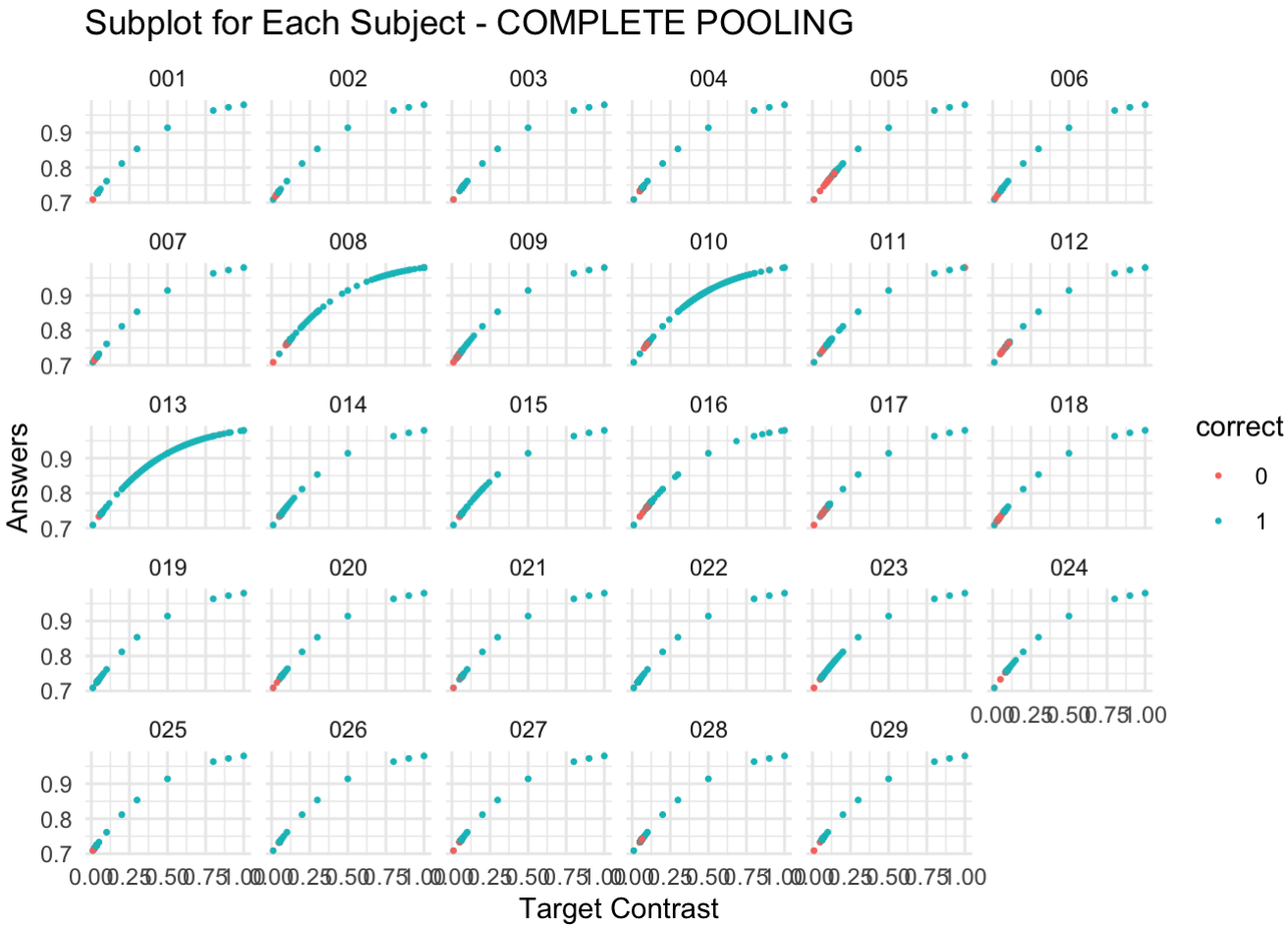
```
#correct:
data$correct <- as.factor(data$correct)
```

```
iii. for the staircasing part __only__, create a plot for each subject where you plot
the estimated function (on the _target.contrast_ range from 0-1) based on the fitted
values of a model (use `glm`) that models _correct_ as dependent on _target.contrast
_. These plots will be our _no-pooling_ model. Comment on the fits - do we have enoug
h data to plot the logistic functions?
```

```
df_m <- data %>%
  filter(trial.type == "staircase")


#model for COMPLETE pooling!!!!
trial_model<- glm(correct ~ target.contrast, data = df_m, family = "binomial")
fitted_y <- fitted(trial_model)

plot1 <- ggplot(df_m, aes(target.contrast, fitted_y, color = correct)) +
    geom_point(size =0.5)+
    facet_wrap(~subject) +
    theme_minimal()+
    xlab("Target Contrast")+
    ylab('Answers') +
    ggtitle("Subplot for Each Subject - COMPLETE POOLING")



#model for NO pooling:
##### meaning that we only take into account the individual and does not pool the dat
a together.
no.pool <- glm(correct ~ target.contrast + subject + subject:target.contrast, data =
 df_m, family = "binomial" )
```

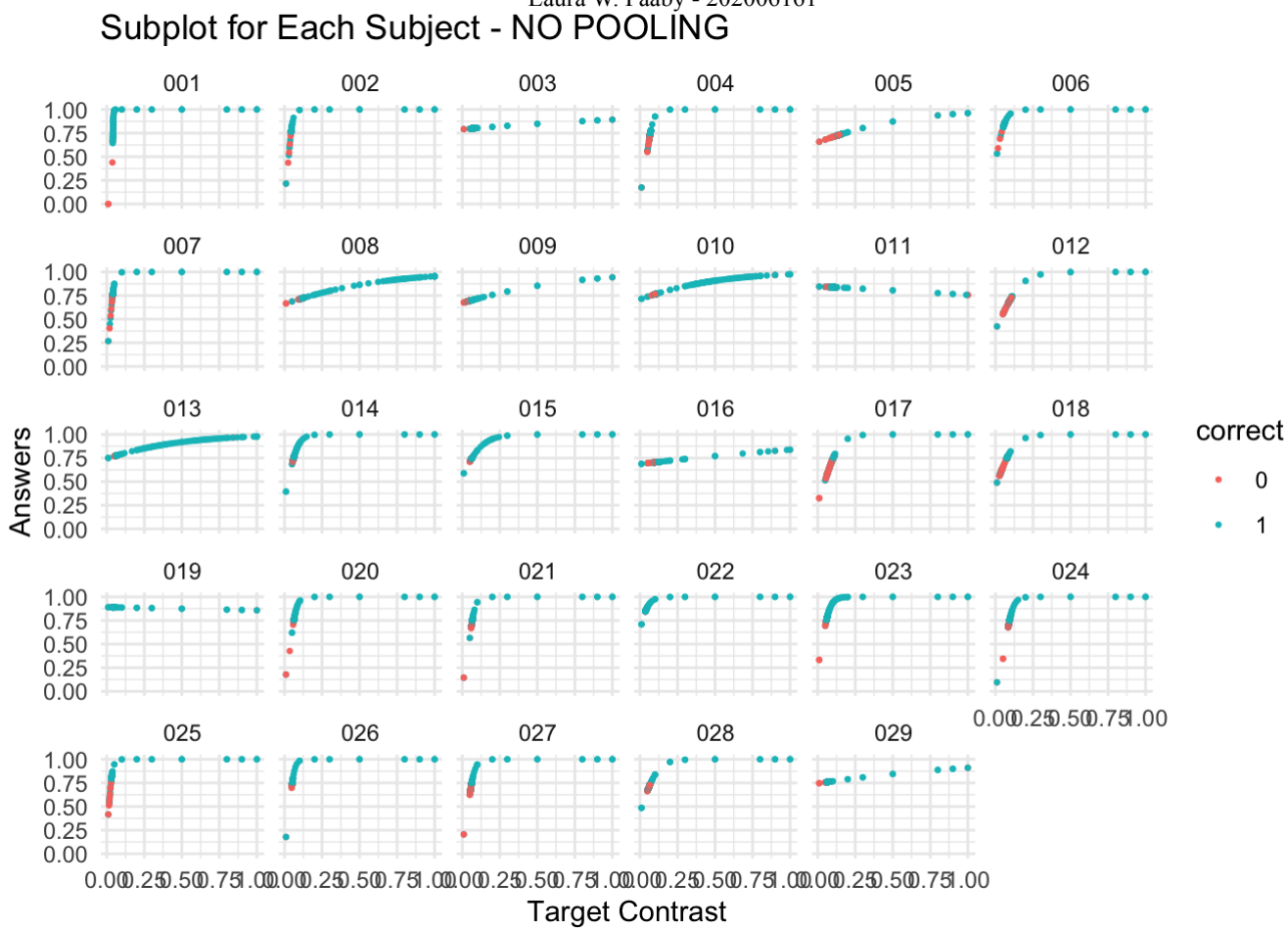
```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
fit_np <- fitted(no.pool)

plot2 <- ggplot(df_m, aes(target.contrast, fit_np, color = correct)) +
    geom_point(size = 0.5)+
    facet_wrap(~subject) +
    theme_minimal()+
    xlab("Target Contrast")+
    ylab('Answers') +
    ggtitle("Subplot for Each Subject - NO POOLING")

plot1
```



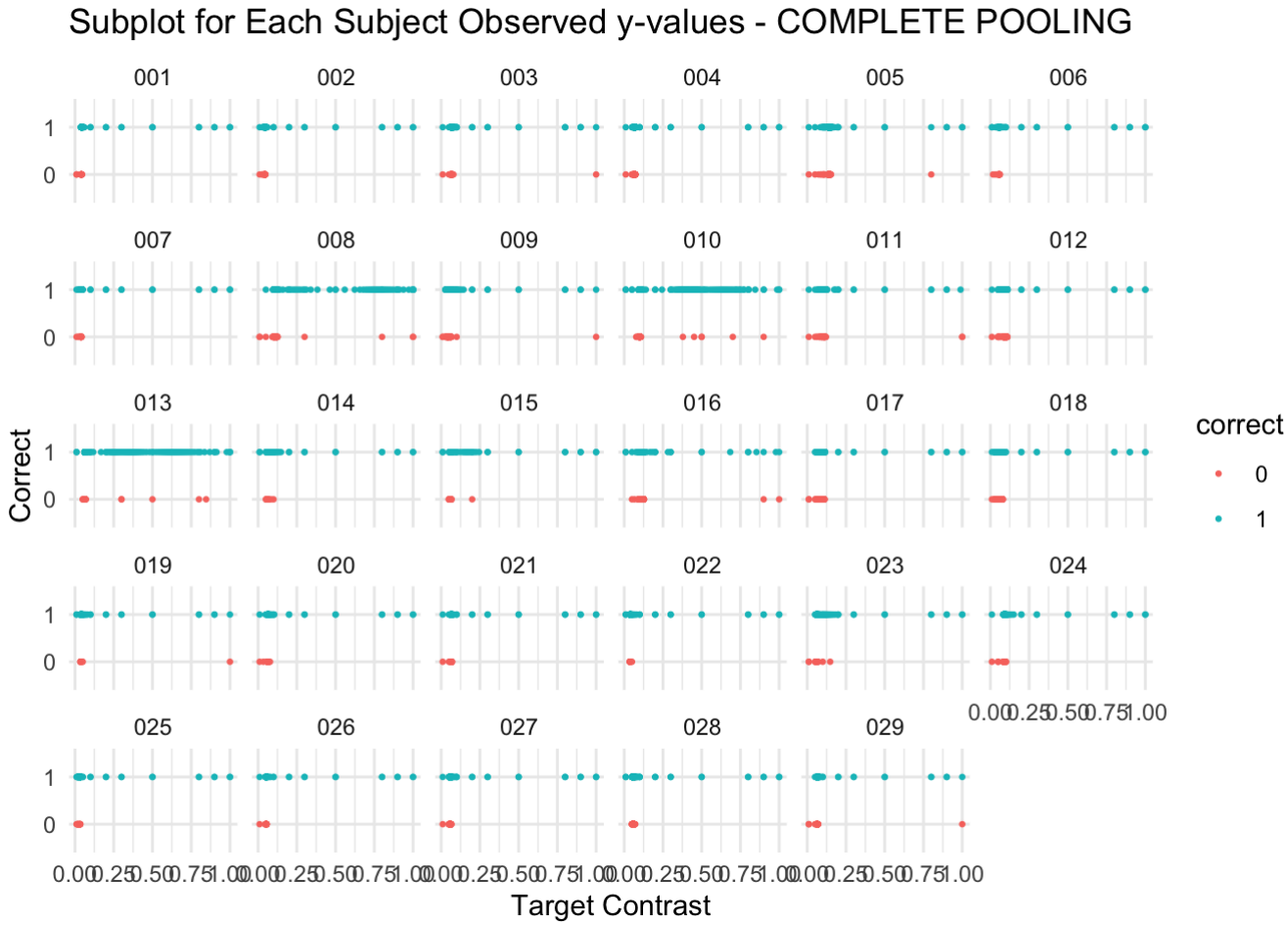

```
plot2
```

Laura W. Paaby - 202006161



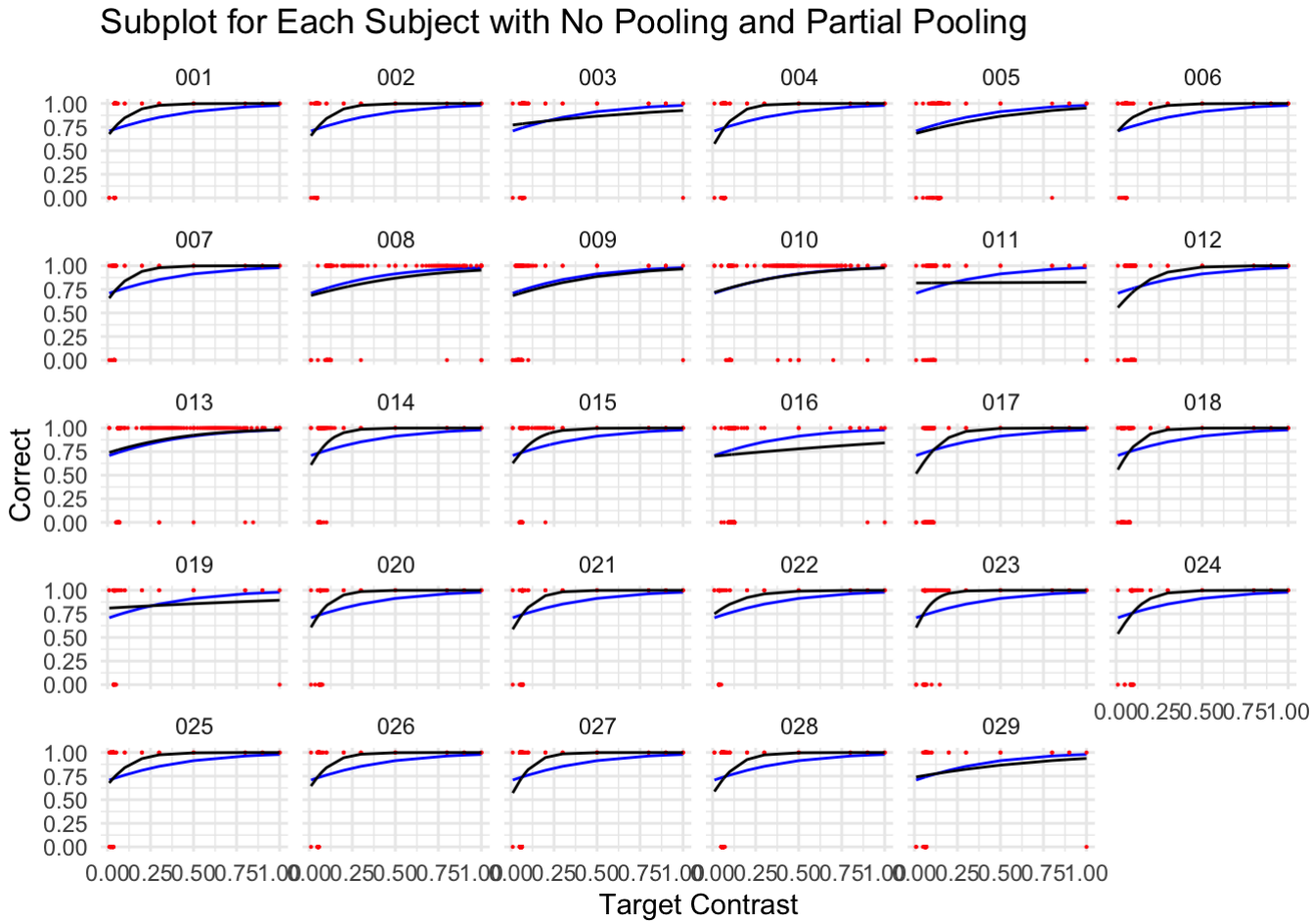
These plots are both supposed to follow a sigmoid function, assuming the relation is logistic, but this is not really the case. There appears to be a ceiling effect, since almost all subjects had way more correct answers than wrong, meaning almost all of the points are at y = 1. However, compared to the complete pooling plot, no pooling is a better fit for each subject. Moreover, it appears as if more data is needed to succesfully plot the logistic regression.

```
#complete pooling with observed y-values.
plot3 <- ggplot(df_m, aes(target.contrast, correct, color = correct)) +
    geom_point(size =0.5)+
    facet_wrap(~subject) +
    theme_minimal()+
    xlab("Target Contrast")+
    ylab('Correct') +
    ggtitle("Subplot for Each Subject Observed y-values - COMPLETE POOLING")

plot3
```



iv. on top of those plots, add the estimated functions (on the _target.contrast_ range from 0-1) for each subject based on partial pooling model (use `glmer` from the package `lme4`) where unique intercepts and slopes for _target.contrast_ are modelled for each _subject_

```
#partial pooled model:
library(lme4)
part.pool.model <- glmer(correct ~ target.contrast + (target.contrast|subject), family = "binomial", data = df_m)

part_fit <- fitted(part.pool.model)


plot3 <- ggplot(df_m, aes(target.contrast, fitted_y)) +
    geom_line(colour = 'blue', size = 0.5)+
    geom_point(aes(x = target.contrast, y = as.numeric(as.character(correct))), colour = 'red', size = 0.07)+
    geom_line(aes(y=part_fit), colour = 'black', size = 0.5) +
    facet_wrap(~subject) +
    theme_minimal()+
    xlab("Target Contrast")+
    ylab('Correct') +
    ggtitle("Subplot for Each Subject with No Pooling and Partial Pooling")

plot3
```



the black line are here the partial pooling and the blue the no-pooling. The red points are the observed data points.

```
v. in your own words, describe how the partial pooling model allows for a better fit
for each subject
```

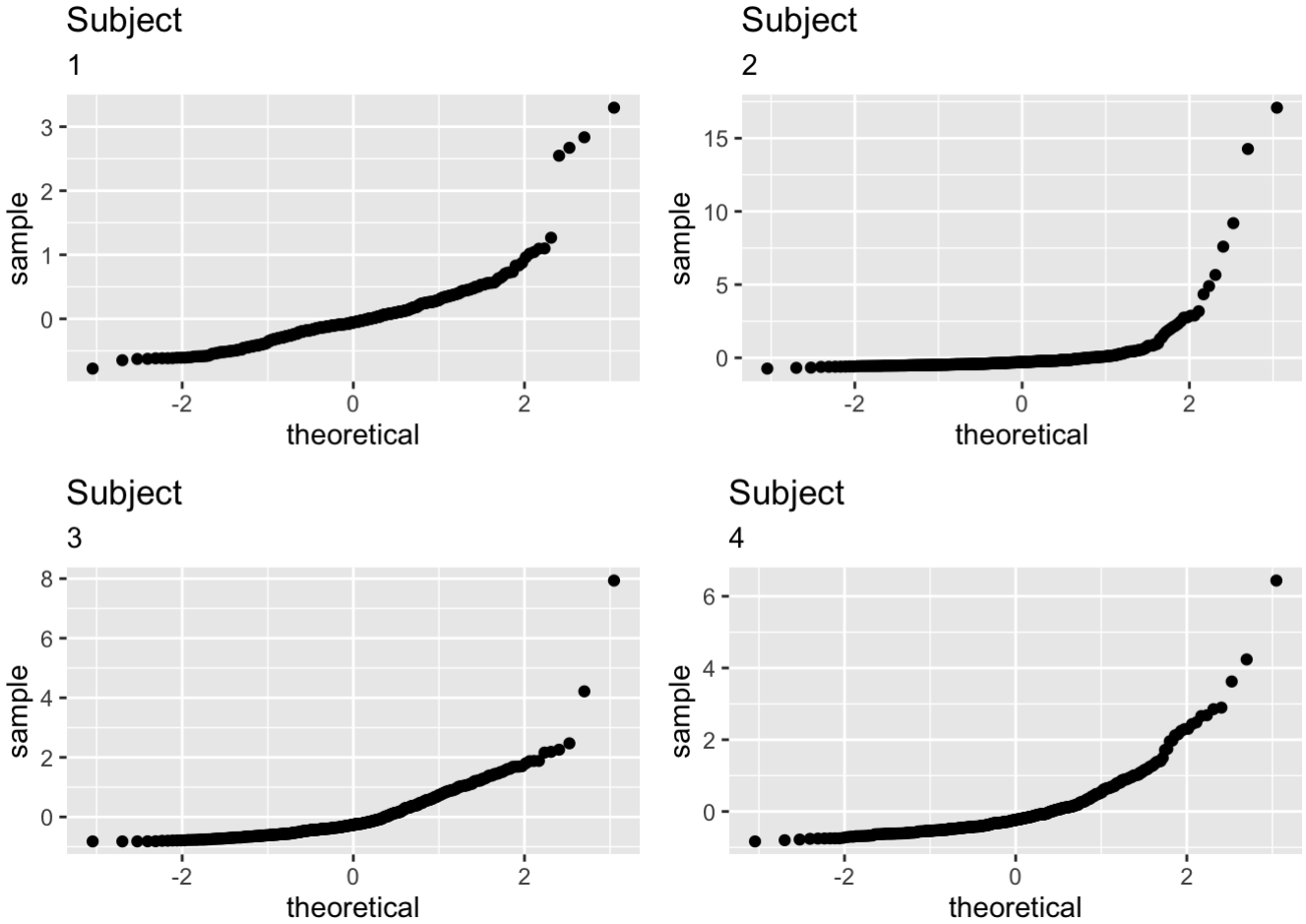
When partial pooling we are able to take into account both the average and each level of the categorical predictor. Hereby the model are more generalizable. The partial pooled points also appears to follow a sigmoid function a bit better than the not-pooled. It appears how the no pooled line actually fit the datapoints better, since it models different baselines and slopes for the subjects. However, this is a very hard model to generalise, thus the partial pooling model must be said to allow fot the better fit.

Exercise 2

Now we **only** look at the *experiment* trials (*trial.type*)

1. Pick four subjects and plot their Quantile-Quantile (Q-Q) plots for the residuals of their objective response times (*rt.obj*) based on a model where only intercept is modelled

```
#making a dataframe only containing the experimental trials.
experi.df <- data %>% filter(trial.type == "experiment")

df_1 <- experi.df %>%
    filter(subject == "001")
df_2 <- experi.df %>%
    filter(subject == "002")
df_3 <- experi.df %>%
    filter(subject == "003")
df_4 <- experi.df %>%
    filter(subject == "004")

#making the models for each subject's response time, with only the intercept modeled
m1 <- lm(rt.obj ~ 1, data = df_1)
m2 <- lm(rt.obj ~ 1, data = df_2)
m3 <- lm(rt.obj ~ 1, data = df_3)
m4 <- lm(rt.obj ~ 1, data = df_4)


#a function that plots a qq plot for the residuals of the given model
qq_f <- function(data.i, model.i, subject.i){
  plot <- data.i %>%
    ggplot(aes(sample = resid(model.i)))+
    geom_point(stat ="qq")+
    ggtitle("Subject", as.character(subject.i))

}

q1 <- qq_f(df_1, m1, 001)
q2 <- qq_f(df_2, m2, 002)
q3 <- qq_f(df_3, m3, 003)
q4 <- qq_f(df_4, m4, 004)

ggarrange(q1, q2, q3, q4)
```



i.

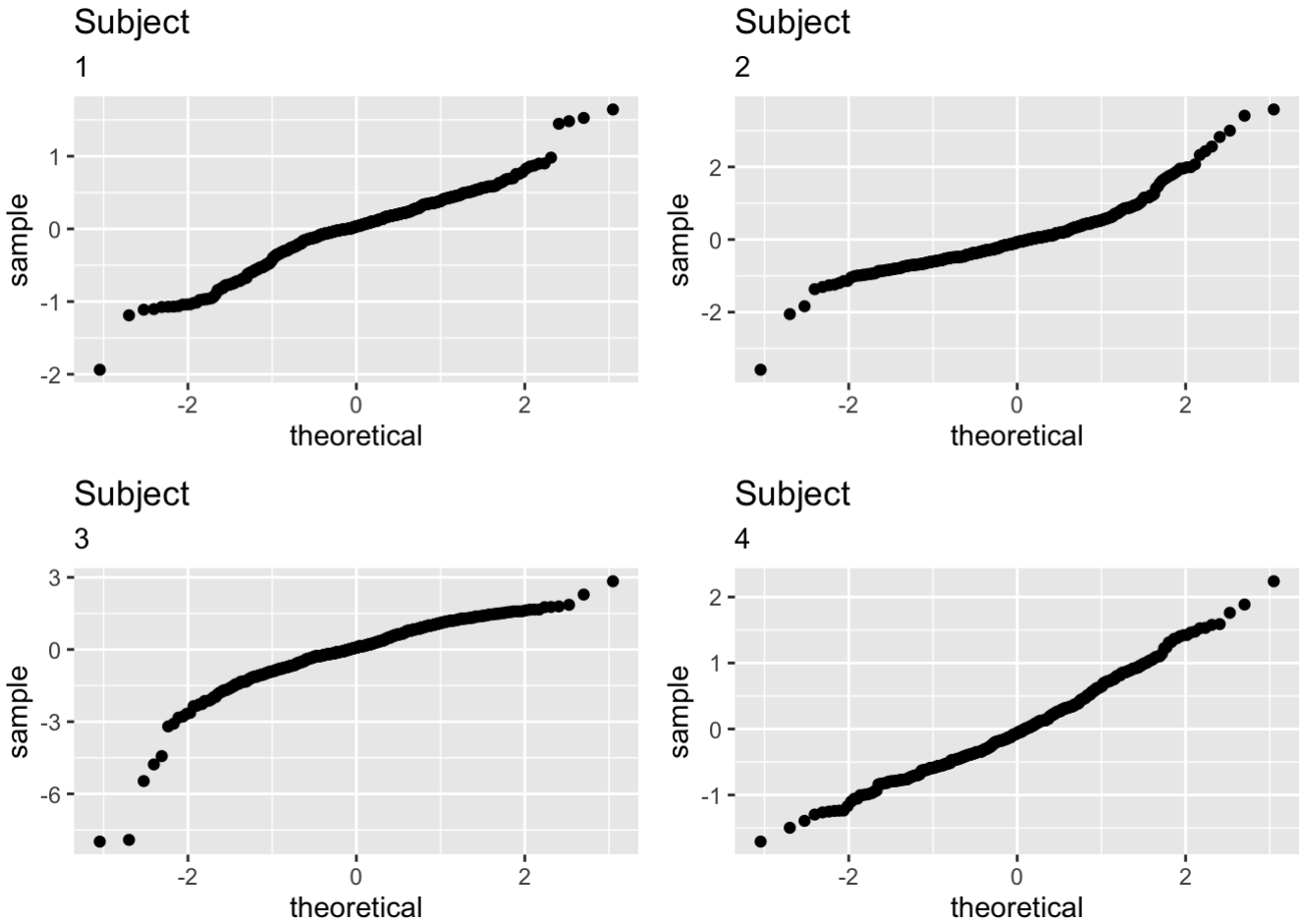
comment on these

It appears as if the values needs a transformation, since it looks rather skewed and doesn't look normally distributed.

```
ii. does a log-transformation of the response time data improve the Q-Q-plots?
```

make the exact same thing, but with log around the rt.obj in each model

```
#making the logged models
m1_log <- lm(log(rt.obj) ~ 1, data = df_1)
m2_log <- lm(log(rt.obj) ~ 1, data = df_2)
m3_log <- lm(log(rt.obj) ~ 1, data = df_3)
m4_log <- lm(log(rt.obj) ~ 1, data = df_4)

#using the functions to plot the qqplot over residuals now over the logged models
p1log <- qq_f(df_1, m1_log, 001)
p2log <- qq_f(df_2, m2_log, 002)
p3log <- qq_f(df_3, m3_log, 003)
p4log <- qq_f(df_4, m4_log, 004)

ggarrange(p1log, p2log, p3log, p4log)
```



2. Now do a partial pooling model modelling objective response times as dependent on *task*? (set `REML=FALSE` in your `lmer` -specification)
this is done to the entire data set - I'm not sure however if what it asked is to only make the model on the four chosen participants. A sample size that small just doesn't seem that generalizable to me, so I went with all the participants.

    i. which would you include among your random effects and why? (support your choices with relevant measures, taking into account variance explained and number of parameters going into the modelling) *Both subject (explained variance of 0.1241) and trial (explained variance of 0.1308) are chosen as random intercepts, since I'd argue that each subject must have a uniqe baseline when entering the task. This could be both their mental state, memory, IQ etc., which means that they should be compared to their own performance between tasks to take such factors into account, as well as the average across the group. I have chosen to make trials a random intercept as well, since the type and difficulty of trial might be affecting the performance. Looking at the summary, I find that despite these arguments, the variance for trial and subject is low compared to the variance explained by residuals (6.4924) ….. see the summary plot:*

```
#partial pooling model:
part_model <- lmer(rt.obj ~ task + (1|subject) + (1|trial), data = data, REML = FALSE
)
summary(part_model)
```

```
## Linear mixed model fit by maximum likelihood . t-tests use Satterthwaite's
##   method [lmerModLmerTest]
## Formula: rt.obj ~ task + (1 | subject) + (1 | trial)
##    Data: data
##
##      AIC      BIC   logLik deviance df.resid
##  85716.7  85763.5 -42852.3  85704.7    18125
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
##  -1.859  -0.199  -0.088   0.073 112.414
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  trial    (Intercept) 0.1308   0.3617
##  subject  (Intercept) 0.1241   0.3523
##  Residual             6.4924   2.5480
## Number of obs: 18131, groups:  trial, 432; subject, 29
##
## Fixed effects:
##                  Estimate Std. Error         df t value Pr(>|t|)
## (Intercept)     1.233e+00  7.544e-02  4.350e+01  16.344  < 2e-16 ***
## taskquadruplet -9.607e-02  4.666e-02  1.803e+04  -2.059   0.0395 *
## tasksingles    -1.900e-01  4.692e-02  1.810e+04  -4.049 5.16e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) tskqdr
## taskqudrplt -0.310
## tasksingles -0.311  0.500
```

```
MuMIn::r.squaredGLMM(part_model)
```

```
## Warning: 'r.squaredGLMM' now calculates a revised statistic. See the help page.
```

```
##               R2m        R2c
## [1,] 0.0008899448 0.03864005
```

```
ii. explain in your own words what your chosen models says about response times betwe
en the different tasks
```

A significant difference in reaction time across all three types of tasks appears p <.05. i.e. when the cue changes (can be either singles, pairs or quadruplets). Going from pairs to single or quadruplets, the reaction times decreases significantly. The

R^2 is small, and not much of the variance is explained. Likewise, none of the chosen intercepts explains that much of the variance either: subject = 0.12 and trial = 0.13, compared to the residual variance 6.49

3. Now add *pas* and its interaction with *task* to the fixed effects
    i. how many types of group intercepts (random effects) can you add without ending up with convergence issues or singular fits?

```
inter_model <- lmer(rt.obj ~ pas*task + (1|subject) + (1 | trial) + (1|task) + (1|pa
s), data = data, REML = FALSE)
```

```
## boundary (singular) fit: see ?isSingular
```

```
inter_model2 <- lmer(rt.obj ~ pas*task + (1|subject) + (1|trial) + (1|pas), data = da
ta, REML = FALSE)
```

```
## boundary (singular) fit: see ?isSingular
```

```
inter_model3 <- lmer(rt.obj ~ pas*task + (1|subject) + (1|trial), data = data, REML =
FALSE)
```

the latter model is the only one of these I can make without running into convergence issues.

```
ii. create a model by adding random intercepts (without modelling slopes) that result
s in a singular fit - then use `print(VarCorr(<your.model>), comp='Variance')` to ins
pect the variance vector - explain why the fit is singular (Hint: read the first para
graph under details in the help for `isSingular`)
```

```
?isSingular
m_sing <- lmer(rt.obj ~ pas*task + (1|subject) + (1|trial) + (1|task), data = data, R
EML = FALSE)
```

```
## boundary (singular) fit: see ?isSingular
```

*This gives the message:*vboundary (singular) fit: see ?isSingular *meaning that this model now results in a singular fit due to the random intercepts added.*

```
print(VarCorr(m_sing), comp='Variance')
```

```
##  Groups   Name        Variance
##  trial    (Intercept) 0.13127
##  subject  (Intercept) 0.11273
##  task     (Intercept) 0.00000
##  Residual             6.46032
```

```
iii. in your own words - how could you explain why your model would result in a singu
lar fit?
```

I assume the singularity occurs as an effect of overfitted data, which occurs when a model is too complex, i.e. it has too many predictors - both random intercept and slopes.

Exercise 3

1. Initialise a new data frame, `data.count`. *count* should indicate the number of times they categorized their experience as *pas* 1-4 for each *task*. I.e. the data frame would have for subject 1: for task:singles, pas1 was used # times, pas2 was used # times, pas3 was used # times and pas4 was used # times. You would then do the same for task:pairs and task:quadruplet

```
data.count <- data %>%
  group_by(subject, pas, task) %>%
  summarise(count = n()) #using n to give the current size of the group, which gives
 the number of times eaxh subject categorized their experience as pas for each task,
 since these are the one called in group by.
```

```
## `summarise()` regrouping output by 'subject', 'pas' (override with `.groups` argum
ent)
```

2. Now fit a multilevel model that models a unique “slope” for *pas* for each *subject* with the interaction between *pas* and *task* and their main effects being modelled

```
#count modelled with slope for pas and subject and an interaction between pas and tas
k.
m_count <- glmer(count ~ pas*task + (1+pas|subject), family = poisson, data = data.co
unt)
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00273537 (tol = 0.002, component 1)
```

it says that my model fails to converge…

```
i. which family should be used?
```

since we work with count data a Poisson distribution should be used

```
ii. why is a slope for _pas_ not really being modelled?
```

Since it is a factor, so we can’t make a general slope as we know them from continuous variables

```
iii. if you get a convergence error, try another algorithm (the default is the _Nelde
r_Mead_) - try (_bobyqa_) for which the `dfoptim` package is needed. In `glmer`, you
can add the following for the `control` argument: `glmerControl(optimizer="bobyqa")`
(if you are interested, also have a look at the function `allFit`)
```

```
m_count_new <- glmer(count ~ pas*task + (1+pas|subject), family = poisson, data = dat
a.count, glmerControl(optimizer="bobyqa"))

summary(m_count_new)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: poisson  ( log )
## Formula: count ~ pas * task + (1 + pas | subject)
##    Data: data.count
## Control: glmerControl(optimizer = "bobyqa")
##
##      AIC      BIC   logLik deviance df.resid
##   3148.4   3232.7  -1552.2   3104.4      318
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -4.3871 -0.7853 -0.0469  0.7550  6.5438
##
## Random effects:
##  Groups  Name        Variance Std.Dev. Corr
##  subject (Intercept) 0.3324   0.5765
##          pas2        0.3803   0.6167   -0.75
##          pas3        1.1960   1.0936   -0.84  0.63
##          pas4        2.3736   1.5407   -0.86  0.42  0.72
## Number of obs: 340, groups:  subject, 29
##
## Fixed effects:
##                     Estimate Std. Error z value Pr(>|z|)
## (Intercept)          4.03570    0.10976  36.769  < 2e-16 ***
## pas2                -0.02378    0.11963  -0.199 0.842461
## pas3                -0.51365    0.20718  -2.479 0.013165 *
## pas4                -0.77292    0.29076  -2.658 0.007853 **
## taskquadruplet       0.11490    0.03127   3.674 0.000239 ***
## tasksingles         -0.23095    0.03418  -6.756 1.42e-11 ***
## pas2:taskquadruplet -0.11376    0.04605  -2.470 0.013508 *
## pas3:taskquadruplet -0.20902    0.05287  -3.954 7.69e-05 ***
## pas4:taskquadruplet -0.21500    0.05230  -4.111 3.94e-05 ***
## pas2:tasksingles     0.19536    0.04830   4.045 5.23e-05 ***
## pas3:tasksingles     0.24299    0.05369   4.526 6.02e-06 ***
## pas4:tasksingles     0.56346    0.05101  11.045  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) pas2   pas3   pas4   tskqdr tsksng ps2:tskq ps3:tskq
## pas2        -0.742
## pas3        -0.829  0.613
## pas4        -0.847  0.412  0.703
## taskqudrplt -0.151  0.138  0.080  0.057
## tasksingles -0.138  0.126  0.073  0.052  0.484
## ps2:tskqdrp  0.102 -0.198 -0.054 -0.039 -0.679 -0.328
## ps3:tskqdrp  0.089 -0.082 -0.125 -0.034 -0.592 -0.286  0.402
## ps4:tskqdrp  0.090 -0.083 -0.048 -0.093 -0.598 -0.289  0.406    0.354
## ps2:tsksngl  0.098 -0.188 -0.052 -0.037 -0.342 -0.708  0.490    0.203
## ps3:tsksngl  0.088 -0.080 -0.124 -0.033 -0.308 -0.637  0.209    0.486
## ps4:tsksngl  0.092 -0.085 -0.049 -0.091 -0.324 -0.670  0.220    0.192
##             ps4:tskq ps2:tsks ps3:tsks
## pas2
## pas3
```

```
## pas4
## taskqudrplt
## tasksingles
## ps2:tskqdrp
## ps3:tskqdrp
## ps4:tskqdrp
## ps2:tsksngl  0.205
## ps3:tsksngl  0.184    0.451
## ps4:tsksngl  0.507    0.474    0.427
```

iv. when you have a converging fit - fit a model with only the main effects of _pas_ and _task_. Compare this with the model that also includes the interaction

```
#now making a model of count with no interaction, but pas and task as fixed effects:
m_count2 <- glmer(count~ task + pas +(1+pas|subject), family = poisson, data = data.c
ount, glmerControl(optimizer = "bobyqa"))

summary(m_count2)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: poisson  ( log )
## Formula: count ~ task + pas + (1 + pas | subject)
##    Data: data.count
## Control: glmerControl(optimizer = "bobyqa")
##
##      AIC      BIC   logLik deviance df.resid
##   3398.5   3459.8  -1683.3   3366.5      324
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -5.5885 -0.9001 -0.0477  0.8253  6.5100
##
## Random effects:
##  Groups  Name        Variance Std.Dev. Corr
##  subject (Intercept) 0.3325   0.5766
##          pas2        0.3805   0.6169   -0.75
##          pas3        1.1895   1.0906   -0.84  0.63
##          pas4        2.4221   1.5563   -0.86  0.42  0.73
## Number of obs: 340, groups:  subject, 29
##
## Fixed effects:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)     4.004593   0.108722  36.833   <2e-16 ***
## taskquadruplet  0.003294   0.018188   0.181   0.8563
## tasksingles     0.004307   0.018177   0.237   0.8127
## pas2           -0.006532   0.116615  -0.056   0.9553
## pas3           -0.509923   0.204449  -2.494   0.0126 *
## pas4           -0.663839   0.291958  -2.274   0.0230 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) tskqdr tsksng pas2   pas3
## taskqudrplt -0.084
## tasksingles -0.084  0.501
## pas2        -0.742  0.000  0.000
## pas3        -0.832  0.001  0.000  0.623
## pas4        -0.850 -0.002  0.000  0.412  0.723
```

```
#comparison of the two models:

#by AIC:
AIC <- AIC(m_count_new, m_count2)

#residuals for each model
resid1 <- residuals(m_count_new)
resid2 <- residuals(m_count2)

#comparing by standard deviation of residuals
sd_model1<- sqrt((sum(resid1)^2/length(resid1)-2))
sd_model2<- sqrt((sum(resid2)^2/length(resid2)-2))

#comparing by the residual variance:
resid.var1 <- sum(resid1^2)
resid.var2 <- sum(resid2^2)


residuals_com <- tibble("model" =c("With Interaction", "Without Interaction"), "Res V
ar" = c(resid.var1, resid.var2), "Res SD" = c(sd_model1, sd_model2), "AIC" = c(AIC[1,
2], AIC[2,2]))

residuals_com
```

```
## # A tibble: 2 x 4
##   model               `Res Var` `Res SD`   AIC
##   <chr>                   <dbl>    <dbl> <dbl>
## 1 With Interaction         699.    0.847 3148.
## 2 Without Interaction      962.    1.32  3399.
```

```
v. indicate which of the two models, you would choose and why
```

*Comparing the models by AIC, the standard deviation of residuals, and the residual variance, I would choose the first model with interaction (glmer(count ~ pas * task + (1+pas|subject)) since it in both cases has the smaller score. This model does also include the interaction, which could be an argument for the choice of model as well. One could imagine a relationship between the type of cue given and the confidence of the subject in the pas.*

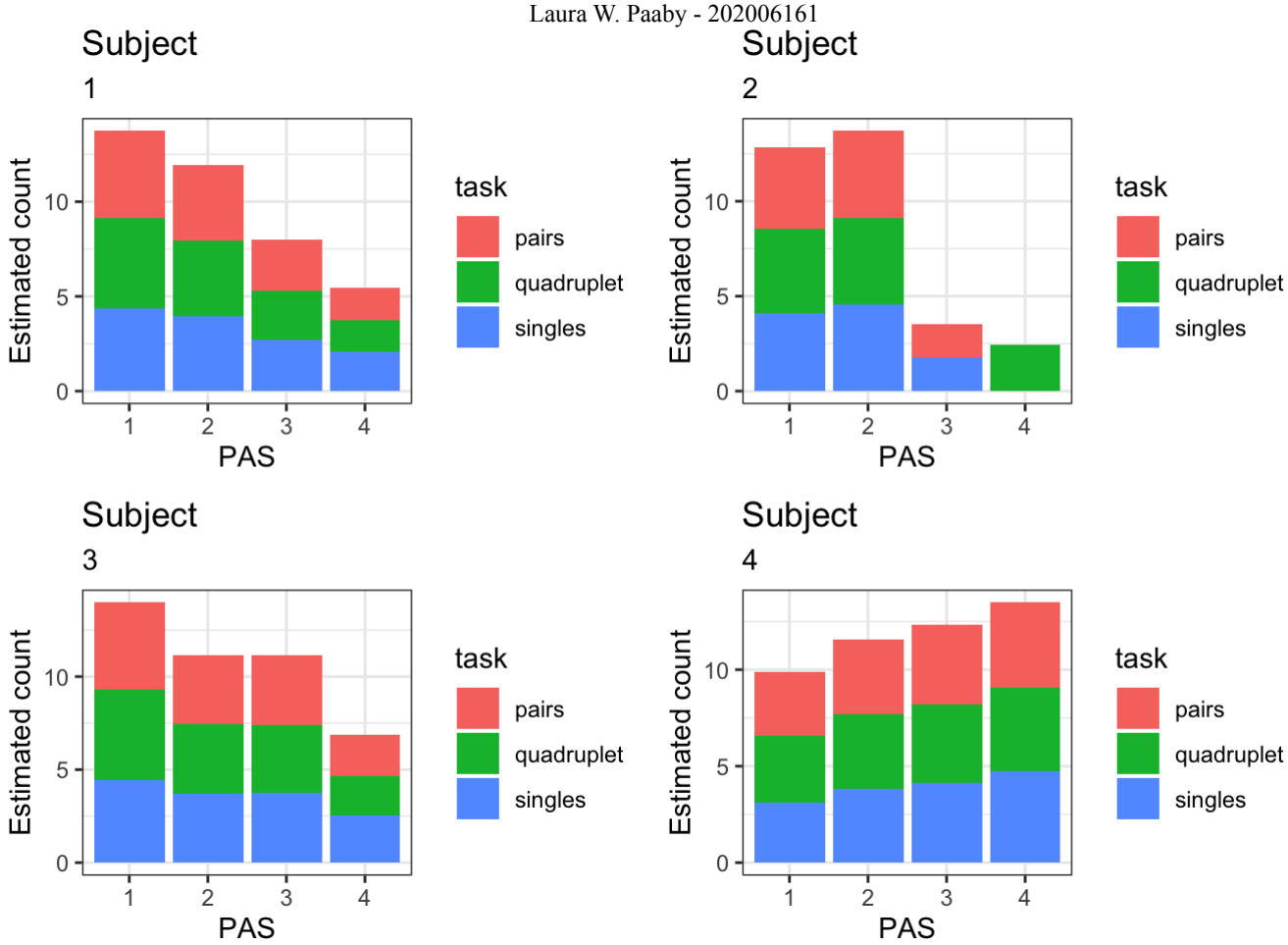
vi. based on your chosen model - write a short report on what this says about the distribution of ratings as dependent on *pas* and *task*

The analysis was conducted to investigate the effect of pas and task on the distribution of ratings (count). To test this, I fitted a general linear mixed effect model, with a link function of the Poisson family. This was done in the assumption that count was distributed as a Poisson distribution. My models has the fixed effects pas and task, and the outcome count. The fixed effects are also included as interactions. Moreover, pas is modelled as a random slope for each subject,

meaning that random intercepts are modelled for each subject Moreover, based on the significant estimates of the model output, ratings does not appear to be equally distributed across task and pas, though pas does not solely predict count.

```
vii. include a plot that shows the estimated amount of ratings for four subjects of y
our choosing
```

```
#choosing subject 1, 2, 3 and 4 again :)
subject1 <- data.count %>% filter(subject == "001")
subject2 <- data.count %>% filter(subject == "002")
subject3 <- data.count %>% filter(subject == "003")
subject4 <- data.count %>% filter(subject == "004")


f <- function(i, model){
  sub.pred <- predict(model, newdata = i)
  i$est.count <- sub.pred

  p <- i %>%
    ggplot()+
    geom_bar(aes(x = pas, y = est.count, fill = task), stat = "identity")+
    theme_bw() +
    ggtitle("Subject", as.character(i[1,1])) +
    xlab("PAS")+
    ylab("Estimated count")

}

p1 <- f(subject1, m_count_new)
p2 <- f(subject2, m_count_new)
p3 <- f(subject3, m_count_new)
p4 <- f(subject4, m_count_new)

ggarrange(p1, p2, p3, p4)
```



3. Finally, fit a multilevel model that models *correct* as dependent on *task* with a unique intercept for each *subject*

```
model3.1 <- glmer(correct ~ task + (1|subject), family = binomial, data = data)
summary(model3.1)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: correct ~ task + (1 | subject)
##    Data: data
##
##      AIC      BIC   logLik deviance df.resid
##  19927.2  19958.4  -9959.6  19919.2    18127
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -2.7426 -1.0976  0.5098  0.6101  0.9111
##
## Random effects:
##  Groups  Name        Variance Std.Dev.
##  subject (Intercept) 0.1775   0.4214
## Number of obs: 18131, groups:  subject, 29
##
## Fixed effects:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)     1.10071    0.08387  13.124  < 2e-16 ***
## taskquadruplet -0.09825    0.04190  -2.345    0.019 *
## tasksingles     0.18542    0.04336   4.276  1.9e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) tskqdr
## taskqudrplt -0.256
## tasksingles -0.247  0.495
```

```
i. does _task_ explain performance?
```

to test if it does we must take the log of the model values, since the model is logistic

```
logit <- function(x) log(x / (1 - x))
inv.logit <- function(x) exp(x) / (1 + exp(x))

taskpair.log <- inv.logit(1.10071)
tasksingles.log <- inv.logit(1.10071 + 0.18542)
taskquad.log <- inv.logit(1.10071 - 0.09825)

tibble("Prob. of Correct in paired task" = taskpair.log, "Prob. of Correct in Singles
Task" = tasksingles.log, "Prob. of Correct in Quadruplets Task" = taskquad.log)
```

```
## # A tibble: 1 x 3
##   `Prob. of Correct in pai… `Prob. of Correct in Sin… `Prob. of Correct in Quad…
##                       <dbl>                     <dbl>                       <dbl>
## 1                     0.750                     0.783                       0.732
```

There is a probability between 70-80% chance to get a correct answer in all three trials, thus I'd argue that the task does not explain performance very well.

ii. add _pas_ as a main effect on top of _task_ - what are the consequences of that?

```
model3.2 <- glmer(correct ~ task + pas + (1|subject), family = binomial, data = data)
summary(model3.2)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: correct ~ task + pas + (1 | subject)
##    Data: data
##
##      AIC      BIC   logLik deviance df.resid
##  17424.9  17479.5  -8705.5  17410.9    18124
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -8.4872 -0.6225  0.3240  0.5767  1.6144
##
## Random effects:
##  Groups  Name        Variance Std.Dev.
##  subject (Intercept) 0.1979   0.4449
## Number of obs: 18131, groups:  subject, 29
##
## Fixed effects:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)     0.08530    0.09149   0.932    0.351
## taskquadruplet -0.03055    0.04498  -0.679    0.497
## tasksingles    -0.01059    0.04688  -0.226    0.821
## pas2            0.95477    0.04420  21.599   <2e-16 ***
## pas3            1.97709    0.06221  31.780   <2e-16 ***
## pas4            3.12732    0.08626  36.254   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) tskqdr tsksng pas2   pas3
## taskqudrplt -0.260
## tasksingles -0.226  0.489
## pas2        -0.213  0.021 -0.040
## pas3        -0.166  0.030 -0.045  0.356
## pas4        -0.123  0.016 -0.079  0.257  0.236
```

Now that pas is added as a main effect along with task, task does not significantly predict correct anymore. Instead pas now do. This could be explained by pas1 now being included in our baseline along with tasksingles, giving that the taskpairs nad taskquadruplet estimates is now only compared with pas 1 .. maybe.

iii. now fit a multilevel model that models _correct_ as dependent on _pas_ with a unique intercept for each _subject_

```
model3.3<- glmer(correct ~ pas + (1|subject), data = data, family = "binomial")
model3.3
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: correct ~ pas + (1 | subject)
##    Data: data
##       AIC       BIC    logLik  deviance  df.resid
## 17421.387 17460.414 -8705.694 17411.387     18126
## Random effects:
##  Groups  Name        Std.Dev.
##  subject (Intercept) 0.4451
## Number of obs: 18131, groups:  subject, 29
## Fixed Effects:
## (Intercept)         pas2         pas3         pas4
##     0.07044      0.95575      1.97892      3.12940
```

iv. finally, fit a model that models the interaction between _task_ and _pas_ and their main effects

```
model3.4 <- glmer(correct ~ pas*task + (1|subject), data = data, family = "binomial")
model3.4
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: correct ~ pas * task + (1 | subject)
##    Data: data
##       AIC       BIC    logLik  deviance  df.resid
## 17430.974 17532.444 -8702.487 17404.974     18118
## Random effects:
##  Groups  Name        Std.Dev.
##  subject (Intercept) 0.4458
## Number of obs: 18131, groups:  subject, 29
## Fixed Effects:
##         (Intercept)                 pas2                 pas3
##            0.083740             0.963210             1.999968
##                pas4       taskquadruplet          tasksingles
##            3.049586             0.006492            -0.058967
## pas2:taskquadruplet  pas3:taskquadruplet  pas4:taskquadruplet
##           -0.049301            -0.134143            -0.095786
##    pas2:tasksingles     pas3:tasksingles     pas4:tasksingles
##            0.040476             0.079941             0.296579
```

v. describe in your words which model is the best in explaining the variance in accuracy

```
#comparison of the  models:
#by AIC:
AIC1 <- AIC(model3.1, model3.2, model3.3, model3.4)

#residuals for each model
resid3.1 <- residuals(model3.1)
resid3.2 <- residuals(model3.2)
resid3.3 <- residuals(model3.3)
resid3.4 <- residuals(model3.4)

#comparing by standard deviation of residuals
sd_model3.1<- sqrt((sum(resid3.1)^2/length(resid3.1)-2))
sd_model3.2<- sqrt((sum(resid3.2)^2/length(resid3.2)-2))
sd_model3.3<- sqrt((sum(resid3.3)^2/length(resid3.3)-2))
sd_model3.4<- sqrt((sum(resid3.4)^2/length(resid3.4)-2))

#comparing by the residual variance:
resid.var3.1 <- sum(resid3.1^2)
resid.var3.2 <- sum(resid3.2^2)
resid.var3.3 <- sum(resid3.3^2)
resid.var3.4 <- sum(resid3.4^2)

library(tidyverse)

residuals_comparison <- tibble("model" =c("Model 3.1", "Model 3.2", "Model 3.3", "Mod
el 3.4"), "Res Var" = c(resid.var3.1, resid.var3.2, resid.var3.3, resid.var3.4), "Res
SD" = c(sd_model3.1, sd_model3.2, sd_model3.3, sd_model3.4), "AIC" = c(AIC1[1,2], AIC
1[2,2], AIC1[3,2], AIC1[4,2]))

residuals_comparison
```

```
## # A tibble: 4 x 4
##   model     `Res Var` `Res SD`    AIC
##   <chr>         <dbl>    <dbl>  <dbl>
## 1 Model 3.1    19804.     19.9 19927.
## 2 Model 3.2    17297.     16.6 17425.
## 3 Model 3.3    17297.     16.6 17421.
## 4 Model 3.4    17291.     16.6 17431.
```

*I would suggest model 3.4: (correct ~ pas***task + (1|subject), family = “binomial”), since this has both the lowest residual variance (17294.14) and standard deviation of residuals (16.74). Thus the model must explain much of the variance it self. Intuitively, this makes rather good sense since one could imagine how subjects experience of the PAS interact with the type of task.*

PORTFOLIO 2.2

Laura Wulff Paaby
202006161
Cognitive Science BA
Aarhus University AU

13/12 - 2021

Portfolio 2.2, Methods 3, 2021, autumn semester

Laura W. Paaby, studygroup 9

13/10 - 2021

Exercises and objectives

The objectives of the exercises of this assignment are based on: https://doi.org/10.1016/j.concog.2019.03.007 (https://doi.org/10.1016/j.concog.2019.03.007)

4. Download and organise the data from experiment 1
5. Use log-likelihood ratio tests to evaluate logistic regression models
6. Test linear hypotheses
7. Estimate psychometric functions for the Perceptual Awareness Scale and evaluate them

REMEMBER: In your report, make sure to include code that can reproduce the answers requested in the exercises below (**MAKE A KNITTED VERSION**)
REMEMBER: This is part 2 of Assignment 2 and will be part of your final portfolio

EXERCISE 4 - Download and organise the data from experiment 1

Go to https://osf.io/ecxsj/files/ (https://osf.io/ecxsj/files/) and download the files associated with Experiment 1 (there should be 29).
The data is associated with Experiment 1 of the article at the following DOI
https://doi.org/10.1016/j.concog.2019.03.007 (https://doi.org/10.1016/j.concog.2019.03.007)

```
#loading libraries:
pacman::p_load(tidyverse, readbulk, patchwork, lmerTest, ggpubr, dfoptim, multcomp, d
plyr)
```

1. Put the data from all subjects into a single data frame - note that some of the subjects do not have the *seed* variable. For these subjects, add this variable and make in *NA* for all observations. (The *seed* variable will not be part of the analysis and is not an experimental variable)

since we are using read_bulk the missing values are already noted as NA:

```
sum(is.na(data$seed))
```

```
## [1] 2646
```

```
i. Factorise the variables that need factorising
```

```
#checking out the data:
ls.str(data)
```

```
## cue :  num [1:25602] 0 0 0 0 0 0 0 0 0 0 ...
## even.digit :  num [1:25602] 8 4 2 8 6 6 6 4 4 8 ...
## File :  chr [1:25602] "001.csv" "001.csv" "001.csv" "001.csv" "001.csv" "001.csv"
...
## jitter.x :  num [1:25602] -0.431 -0.292 0.308 0.238 -0.427 ...
## jitter.y :  num [1:25602] -0.335 -0.182 0.406 0.175 -0.221 ...
## obj.resp :  chr [1:25602] "e" "o" "e" "o" "e" "o" "o" "e" "e" "o" "e" "o" "e" "o"
"e" ...
## odd.digit :  num [1:25602] 9 5 5 9 3 9 3 5 7 9 ...
## pas :  num [1:25602] 4 4 3 2 3 2 1 1 1 4 ...
## rt.obj :  num [1:25602] 1.418 0.86 0.746 1.676 0.847 ...
## rt.subj :  num [1:25602] 4.93 8.67 8.41 2.13 1.82 ...
## seed :  num [1:25602] 93764 93764 93764 93764 93764 ...
## subject :  chr [1:25602] "001" "001" "001" "001" "001" "001" "001" "001" "001" "00
1" ...
## target.contrast :  num [1:25602] 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 ...
## target.frames :  num [1:25602] 9 8 7 6 5 4 3 2 1 9 ...
## target.type :  chr [1:25602] "even" "odd" "even" "odd" "even" "odd" "even" "odd"
"even" ...
## task :  chr [1:25602] "quadruplet" "quadruplet" "quadruplet" "quadruplet" ...
## trial :  num [1:25602] 0 1 2 3 4 5 6 7 8 9 ...
## trial.type :  chr [1:25602] "practice" "practice" "practice" "practice" "practice"
...
```

```
data$trial.type <- as.factor(data$trial.type)
data$pas <- as.factor(data$pas)
data$trial <- as.factor(data$trial)
data$cue <- as.factor(data$cue)
data$task <- as.factor(data$task)
data$target.type <- as.factor(data$target.type)
data$obj.resp <- as.factor(data$obj.resp)
data$subject <- as.factor(data$subject)
```

The factorisation of variables have been made on the basis of Assignment 2 Part 1, in which arguments were presented for the factorisation as well.

```
ii. Remove the practice trials from the dataset (see the _trial.type_ variable)
```

```
data_exp <- data %>%
  filter(trial.type == "experiment")
```

```
iii. Create a _correct_ variable
```

```
data_exp$correct <- ifelse((data_exp$obj.resp == "o" & data_exp$target.type =="odd")
 | (data_exp$obj.resp == "e" & data_exp$target.type =="even"), 1,0)
```

```
iv. Describe how the _target.contrast_ and _target.frames_ variables differ compared
to the data from part 1 of this assignment
```

target contrast is now only one value: 0.1. Which is the contrast on the screen between the target and the background. This is now a constant, opposite the previous experiment.

```
summary(data_exp$target.contrast)
```

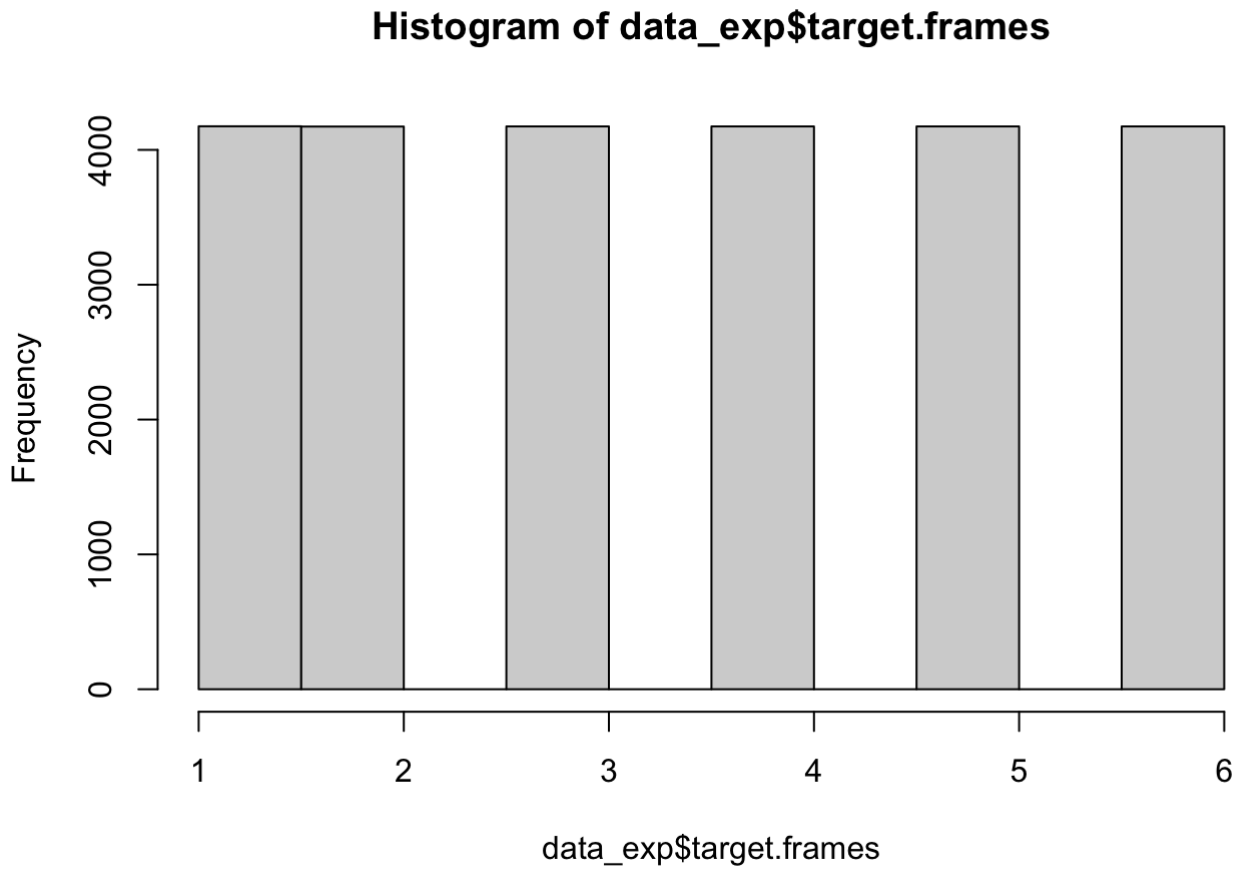
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     0.1     0.1     0.1     0.1     0.1     0.1
```

target frame are now a ordinal, non-continuous variable, as seen in the summary and the histogram, which also indicates that there is the same amount of counts in each target frame. This is opposite the old dataset, where the frame was a constant. It is the amound of frames in which the target is shown.

```
summary(data_exp$target.frames)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     1.0     2.0     3.5     3.5     5.0     6.0
```

```
hist(data_exp$target.frames)
```



EXERCISE 5 - Use log-likelihood ratio tests to evaluate logistic regression models

1. Do logistic regression - *correct* as the dependent variable and *target.frames* as the independent variable. (Make sure that you understand what *target.frames* encode). Create two models - a pooled model (**m5.1.pook**) and a partial-pooling model. The partial-pooling (**m5.1.partpool**) model should include a subject-specific intercept.

```
m5.1.pool <- glm(correct ~ target.frames, family = binomial, data = data_exp)
m5.1.partpool <- glmer(correct ~ target.frames + (1 | subject), family = binomial, da
ta = data_exp)
```

```
i. the likelihood-function for logistic regression is:
```

$L(p) = \prod_{i=1}^{N} p^{y_i}(1-p)^{(1-y_i)}$ (Remember the probability mass function for the Bernoulli Distribution). Create a function that calculates the likelihood.

```
# Likelihood-Function for logistic regression based on the prob-mass-func.
like_funct <- function(model, df, y){
  prob <- fitted(model)

  return(prod(prob^y*(1-prob)^(1-y)))
}
```

```
ii. the log-likelihood-function for logistic regression is: $l(p) = {\displaystyle\su
m_{i=1}^N}[y_i\ln{p}+(1-y_i)\ln{(1-p)}$. Create a function that calculates the log-li
kelihood
```

```
# Log-Likelihood-Function for logistic regression
log_like_funct <- function(model, df, y){
  prob <- fitted(model)

  sum((y*(log(prob)) + (1-y)*log(1-prob)))

}
```

```
iii. apply both functions to the pooling model you just created. Make sure that the l
og-likelihood matches what is returned from the _logLik_ function for the pooled mode
l.
```

```
#using the functions on the pooled model:
pool_prob_like <- like_funct(m5.1.pool, data_exp, data_exp$correct)
pool_prob_likelog <- log_like_funct(m5.1.pool, data_exp, data_exp$correct)

#the outcome:
pool_prob_like
```

```
## [1] 0
```

```
pool_prob_likelog
```

```
## [1] -10865.25
```

Does the likelihood-function return a value that is surprising? *It seems a bit odd that the likelihood estimate is a 0, however considering the huge size of the data and how small the values are, it isn't that suprising after all that it ends on a 0 (also considering the limited precision of the likelihood function).*

```
log_like_r <- logLik(m5.1.pool)

compare_prob <- tibble("Estimated Likelihood Log" =c(pool_prob_likelog), "logLik" = c
(log_like_r))
compare_prob
```

Estimated Likelihood Log	logLik
<dbl>	<dbl>
-10865.25	-10865.25

1 row

Why is the log-likelihood preferable when working with computers with limited precision? *When calculating the likelihood numbers are multiplied together - are these small the numbers will be to small in the end for the computer to comprehend it. Additionally a 0 in this line of numbers will result in it all becoming 0, and not interpretable. Therefore the log-likelihood is preferred, since we then avoid super small numbers and 0's. It simply does not require as much computational precision.*

```
iv. now show that the log-likelihood is a little off when applied to the partial pool
ing model - (the likelihood function is different for the multilevel function - see s
ection 2.1 of https://www.researchgate.net/profile/Douglas-Bates/publication/2753537_
Computational_Methods_for_Multilevel_Modelling/links/00b4953b4108d73427000000/Computa
tional-Methods-for-Multilevel-Modelling.pdf if you are interested)
```

```
log_like_r1 <- logLik(m5.1.partpool)
part_pool1 <- log_like_funct(m5.1.partpool, data_exp, data_exp$correct)

compare_prob1 <- tibble("Estimated Likelihood Log Part Pool" =c(part_pool1), "logLik
 Part Pool" = c(log_like_r1))
compare_prob1
```

Estimated Likelihood Log Part Pool	logLik Part Pool
<dbl>	<dbl>
-10565.53	-10622.03

1 row

As showed, there is a slight difference between the two log-likelihoods

2. Use log-likelihood ratio tests to argue for the addition of predictor variables, start from the null model, `glm(correct ~ 1, 'binomial', data)` (*m2.1*), then **add** subject-level intercepts (*m2.2*), then add a group-level effect of *target.frames* (*m2.3*) and finally add subject-level slopes for *target.frames*.(*m2.4*)

Also assess whether or not a correlation between the subject-level slopes and the subject-level intercepts should be included. (*m2.5*)

```
#the null model
m2.1 <- glm(correct ~ 1, family = binomial, data = data_exp)
llrt2.1 <- log_like_funct(m2.1, data_exp, data_exp$correct)

#added sub level intercepts
m2.2 <- glmer(correct ~ 1 + (1|subject), family = binomial, data = data_exp)
llrt2.2 <- log_like_funct(m2.2, data_exp, data_exp$correct)

#added group-level effect of target frames
m2.3 <- glmer(correct ~ target.frames + (1|subject), family = binomial, data = data_e
xp)
llrt2.3 <- log_like_funct(m2.3, data_exp, data_exp$correct)

#without correlation between slope and intercept (2 times |) and with subject-level e
ffect of target.frames
m2.4 <- glmer(correct ~ target.frames + (1 + target.frames || subject), family = bino
mial, data = data_exp)
llrt2.4 <- log_like_funct(m2.4, data_exp, data_exp$correct)

#equal to the previous, but with correlation between slope and intercept (only one |)
m2.5 <- glmer(correct ~ target.frames + (1 + target.frames | subject), family = binom
ial, data = data_exp)
llrt2.5 <- log_like_funct(m2.5, data_exp, data_exp$correct)

#anova comparison
anov <- anova(m2.2, m2.3, m2.4, m2.5, m2.1)
anov
```

	n... <dbl>	AIC <dbl>	BIC <dbl>	logLik <dbl>	deviance <dbl>	Chisq <dbl>	Df <dbl>	Pr(>Chisq) <dbl>
m2.1	1	26685.08	26693.21	-13341.54	26683.08	*NA*	*NA*	*NA*
m2.2	2	26319.10	26335.35	-13157.55	26315.10	367.97976	1	5.153061e-82
m2.3	3	21250.06	21274.45	-10622.03	21244.06	5071.03518	1	0.000000e+00
m2.4	4	20928.58	20961.09	-10460.29	20920.58	323.48672	1	2.520170e-72
m2.5	5	20907.65	20948.29	-10448.83	20897.65	22.92598	1	1.683609e-06

5 rows

```
#comparison by log likelihood and anova summary
compare_prob2 <- tibble("Model" = c("Null Model m2.1", "Sub Intercept Model m2.2", "T
arget Frames Effect m2.3", "No Correlation between Random Effects m2.4", "Correlation
Between Random Effects m2.5"), "Log Likelihood RT" =c(llrt2.1, llrt2.2, llrt2.3, llrt
2.4, llrt2.5))
compare_prob2
```

Model <chr>	Log Likelihood RT <dbl>
Null Model m2.1	-13341.54
Sub Intercept Model m2.2	-13105.03

Model <chr>	Log Likelihood RT <dbl>
Target Frames Effect m2.3	-10565.53
No Correlation between Random Effects m2.4	-10378.96
Correlation Between Random Effects m2.5	-10375.14

5 rows

```
i. write a short methods section and a results section where you indicate which model you chose and the statistics relevant for that choice.
```

When comparing models by their log-likelihood ratio, one should look for the higher value, since one wants it to be maximized. In this case I would therefore suggest model 2.5: glmer(correct ~ target.frames + (1 + target frames|subject), family = binomial) which has the target frames as the fixed effects, subject as the random intercept and a correlation between the subject level slopes and intercepts: which has a log likelihood ratio value of -10375.14.

Additionally … it can be seen how the results of an anova comparison indicates the same finding - the model perform best in the anova as well, leading me to suggest that a correlation should be included in our model

Summed upe: **The model with the highest log-likelihood score (-10375.14) was found to be the mixed effect model which included a correlation between random slope and intercept. Due to this finding, the m2.4 model is chosen as the final model with: beta_0 = 1.09 (SE = 0.059, p < .001) and beta_1 = 0.83 (SE = 0.044, p < .001).**

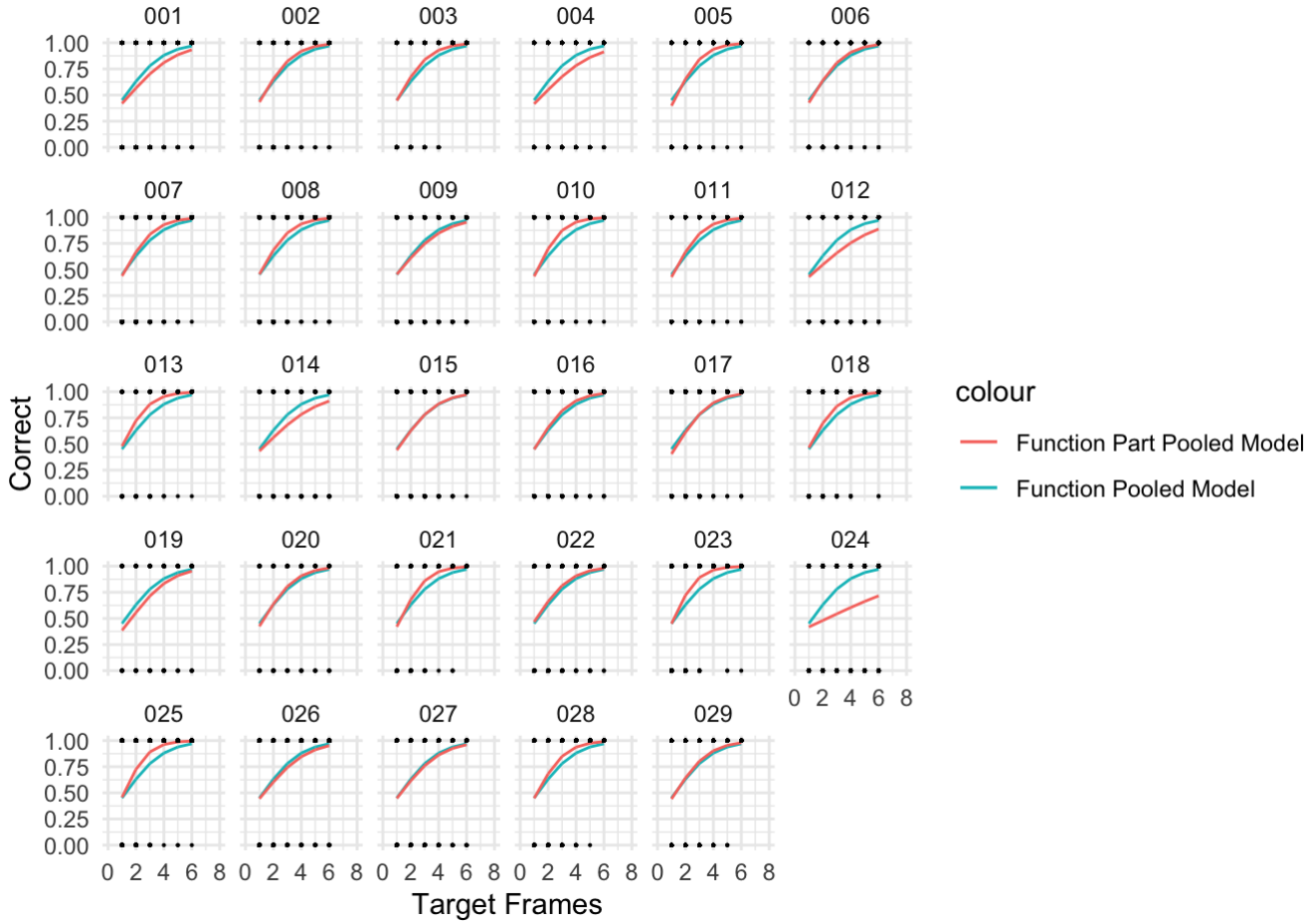
as found in:

```
summary(m2.4)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: correct ~ target.frames + (1 + target.frames || subject)
##    Data: data_exp
##
##      AIC      BIC   logLik deviance df.resid
##  20928.6  20961.1 -10460.3  20920.6    25040
##
## Scaled residuals:
##      Min       1Q   Median       3Q      Max
## -14.1977   0.0898   0.2470   0.4894   1.3292
##
## Random effects:
##  Groups    Name          Variance Std.Dev.
##  subject   (Intercept)   0.0360   0.1897
##  subject.1 target.frames 0.0366   0.1913
## Number of obs: 25044, groups:  subject, 29
##
## Fixed effects:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -1.06926    0.05083  -21.04   <2e-16 ***
## target.frames  0.82207    0.03817   21.54   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr)
## target.frms -0.230
```

Include a plot of the estimated group-level function with `xlim=c(0, 8)` that includes the estimated subject-specific functions.

```
#fitted values from the pool model to compare
pool.fit <- fitted(m5.1.pool)

#fitted values over the chosen model
m5.fit <- fitted(m2.5)


plot2 <- ggplot(data_exp, aes(target.frames, as.numeric(as.character(correct)))) +
    geom_line(aes(x = target.frames, y = pool.fit, color = "Function Pooled Model"))+
    geom_line(aes(x = target.frames, y = m5.fit, color = "Function Part Pooled Model"
))+
    geom_point(aes(y = as.numeric(as.character(correct), color = "Response")), size =
0.07) +
   facet_wrap(~subject)+
  xlim(min = 0, max = 8)+
    theme_minimal()+
    xlab("Target Frames")+
    ylab('Correct')


plot2
```



The dots: Are the responses given by each participant (0 being wrong, 1 correct)

By eyeballing the plot it appears as if the function for subject 24 looks rather different that the group-specific function. One could additionally compare the subject specific function with the group specific, and find that in many cases there is a bit of a difference.

```
ii. also include in the results section whether the fit didn't look good for any of t
he subjects. If so, identify those subjects in the report, and judge (no statistical
test) whether their performance (accuracy) differed from that of the other subjects.
Was their performance better than chance? (Use a statistical test this time) (50 %)
```

```
data_24 <- data_exp %>%
  filter(subject == "024")

t.test(x = data_24$correct, mu = 0.5)
```

```
##
##  One Sample t-test
##
## data:  data_24$correct
## t = 4.026, df = 873, p-value = 6.167e-05
## alternative hypothesis: true mean is not equal to 0.5
## 95 percent confidence interval:
##  0.5345964 0.6004150
## sample estimates:
## mean of x
## 0.5675057
```

Inspecting the one-sampled t-test, we see whether the performance of subject 24 was better than if the outcomes were due to pure chance (hence the 0.5, which is the value it should be compared by) - this indicates there is a significant difference ($p < .001$) between the subject 24 performance and a performance achieved by chance.

3. Now add *pas* to the group-level effects - if a log-likelihood ratio test justifies this, also add the interaction between *pas* and *target.frames* and check whether a log-likelihood ratio test justifies this

```
#pas as group level effect
m3.1 <- glmer(correct ~ target.frames + pas + (1 + target.frames | subject), family =
binomial, data = data_exp)

#pas as interaction with target.frames
m3.2 <- glmer(correct ~ target.frames * pas + (1 + target.frames | subject), family =
binomial, data = data_exp)
```

```
i. if your model doesn't converge, try a different optimizer
```

it appears to be working :D

```
#does log-like justify????
log_compare <- tibble("Model" = c("m2.5", "m.3.1", "m3.2"), "Type" = c("Correlation M
odel without Pas", "With Pas as Group Level Effect", "With Pas As Interaction"), "Log
-Like Value" = c(log_like_funct(m2.5, data_exp, data_exp$correct), log_like_funct(m3.
1, data_exp, data_exp$correct), log_like_funct(m3.2, data_exp, data_exp$correct)))

log_compare
```

Model <chr>	**Type** <chr>	**Log-Like Value** <dbl>
m2.5	Correlation Model without Pas	-10375.136
m.3.1	With Pas as Group Level Effect	-9860.225
m3.2	With Pas As Interaction	-9684.128

3 rows

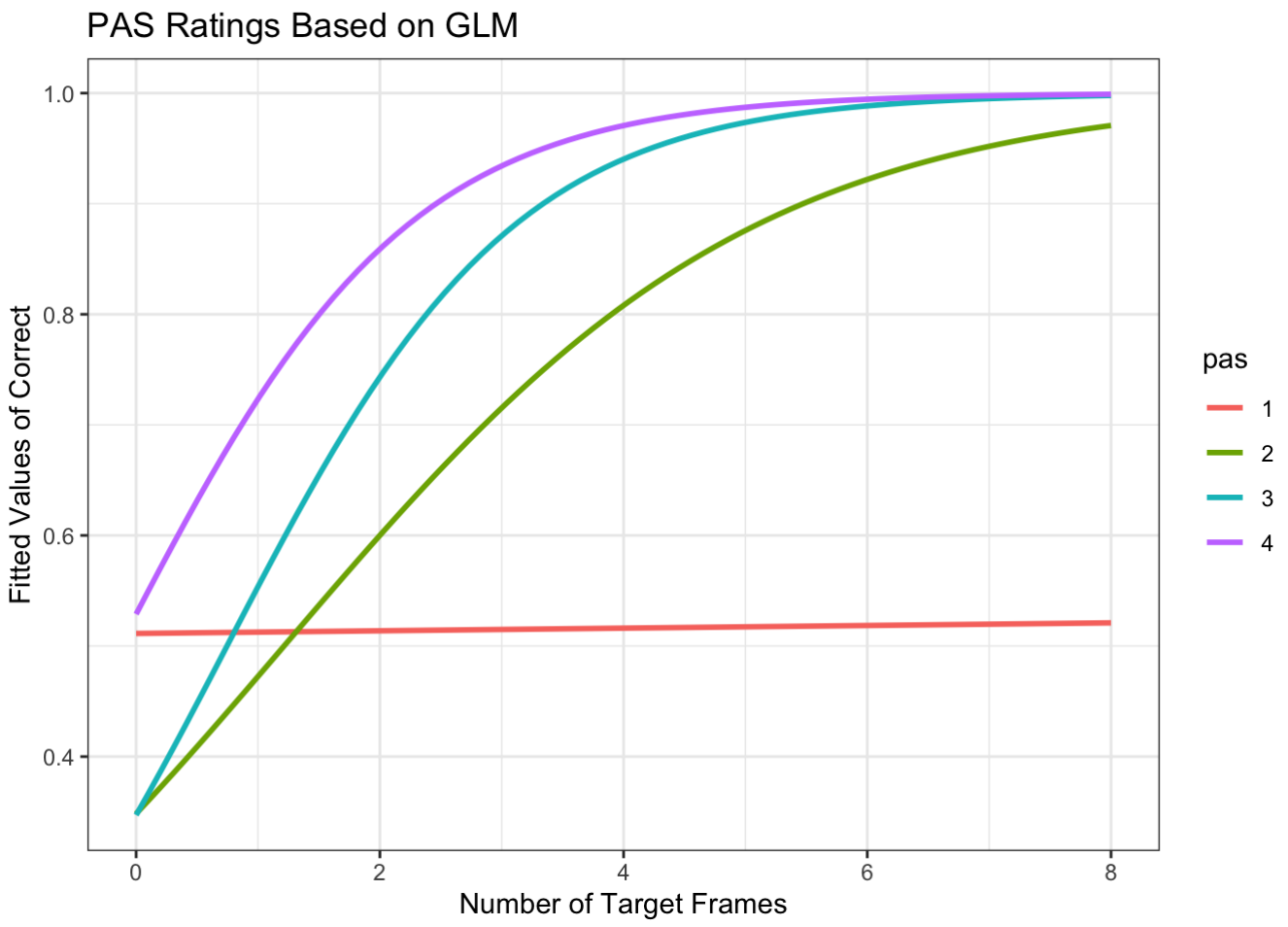
*Seeing how the Log-Likelihood Value increases by adding PAS as interaction, it can be justified to add this to our model. Our final model is now m3.2: glmer(correct ~ target.frames * pas + (1 + target.frames | subject), family = binomial).*

```
ii. plot the estimated group-level functions over `xlim=c(0, 8)` for each of the four
PAS-ratings - add this plot to your report (see: 5.2.i) and add a description of your
chosen model. Describe how _pas_ affects accuracy together with target duration if at
all.
```

```
data_exp$fit_mod <- fitted.values(m3.2)

plot_5.3 <- data_exp %>%
  ggplot(aes(target.frames, fit_mod)) +
  geom_smooth(aes(colour = pas), method = "glm", method.args = list(family = "binomia
l"), se = FALSE, fullrange = TRUE) +
  xlim(0,8) +
  ylab("Fitted Values of Correct") +
  xlab("Number of Target Frames") +
  ggtitle("PAS Ratings Based on GLM")+
  theme_bw()

plot_5.3
```

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

```
## Warning in eval(family$initialize): non-integer #successes in a binomial glm!

## Warning in eval(family$initialize): non-integer #successes in a binomial glm!

## Warning in eval(family$initialize): non-integer #successes in a binomial glm!

## Warning in eval(family$initialize): non-integer #successes in a binomial glm!
```



This visualize how subjects performs better the higher the PAS value. Additionally the amount of target.frames appears to affect the performance: the more target frames, the better the performance - apart from when pas = 1, where it seems as if most subjects have given the wrong answer. However this is ONLY by eyeballing the

plot. The chosen model m3.2 includes an interaction between pas and target.frames, and their individual effect. Looking at the plot, this choice of model is supported: at pas 1 target frames barely affects correctness, while it in the other pas affects it a lot.

Also comment on the estimated functions' behaviour at target.frame = 0 - is that behaviour reasonable?
since the lowest amount of targetframes in the experiment is 0, Id argue that it would make sense to look at the behaviour to that targetframe. Should we inspect it however, we see how it appears to have values under due to chance, which again is not reasonable. Unless you are subject 24 :). We should additionally remember that these values are estimated and not observed in the actual experiment.

```
#summary of the chosen model:
summary(m3.2)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: correct ~ target.frames * pas + (1 + target.frames | subject)
##    Data: data_exp
##
##      AIC      BIC   logLik deviance df.resid
##  19506.1  19595.5  -9742.0  19484.1    25033
##
## Scaled residuals:
##      Min       1Q   Median       3Q      Max
## -19.0116   0.0537   0.1607   0.4849   1.4465
##
## Random effects:
##  Groups  Name          Variance Std.Dev. Corr
##  subject (Intercept)   0.03697  0.1923
##          target.frames 0.02057  0.1434   -0.76
## Number of obs: 25044, groups:  subject, 29
##
## Fixed effects:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)        -0.12164    0.06419  -1.895 0.058081 .
## target.frames       0.11480    0.03710   3.095 0.001971 **
## pas2               -0.57138    0.08948  -6.386 1.71e-10 ***
## pas3               -0.53844    0.13980  -3.852 0.000117 ***
## pas4                0.20147    0.25132   0.802 0.422758
## target.frames:pas2  0.44718    0.03477  12.863  < 2e-16 ***
## target.frames:pas3  0.74867    0.04602  16.270  < 2e-16 ***
## target.frames:pas4  0.75930    0.06867  11.057  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) trgt.f pas2   pas3   pas4   trg.:2 trg.:3
## target.frms -0.811
## pas2        -0.462  0.306
## pas3        -0.308  0.208  0.249
## pas4        -0.175  0.124  0.123  0.091
## trgt.frms:2  0.482 -0.429 -0.874 -0.246 -0.125
## trgt.frms:3  0.393 -0.359 -0.279 -0.891 -0.111  0.371
## trgt.frms:4  0.276 -0.260 -0.164 -0.121 -0.919  0.226  0.200
```

The log-likelihoods from this can now be used to find the estimated behaviour when target.frame = 0

```
inv.logit <- function(x) exp(x) / (1 + exp(x))

#intercept - so when targetframe equals 0
inv.logit(-0.12164)
```

```
## [1] 0.4696274
```

```
#pas 2. here i can see that the probability of answering correct when going from pas
 1 to pas 2 is increased with 36%
inv.logit(-0.57138)
```

```
## [1] 0.3609185
```

```
#pas 3: Here i can see that the probability of answering correct when going from pas
 2 to pas 3 is increased with 36%
inv.logit(-0.53844)
```

```
## [1] 0.3685506
```

```
#pas 4:  here i can see that the probability of answering correct when going from pas
1 to pas 2 is increased with 55%
inv.logit(0.20147)
```

```
## [1] 0.5501978
```

EXERCISE 6 - Test linear hypotheses

In this section we are going to test different hypotheses. We assume that we have already proved that more objective evidence (longer duration of stimuli) is sufficient to increase accuracy in and of itself and that more subjective evidence (higher PAS ratings) is also sufficient to increase accuracy in and of itself.

We want to test a hypothesis for each of the three neighboring differences in PAS, i.e. the difference between 2 and 1, the difference between 3 and 2 and the difference between 4 and 3. More specifically, we want to test the hypothesis that accuracy increases faster with objective evidence if subjective evidence is higher at the same time, i.e. we want to test for an interaction.

1. Fit a model based on the following formula:
   `correct ~ pas * target.frames + (target.frames | subject))`
   i. First, use `summary` (yes, you are allowed to!) to argue that accuracy increases faster with objective evidence for PAS 2 than for PAS 1.

```
#making the model
m6.1 <- glmer(correct ~ pas * target.frames + (1+target.frames | subject), family = b
inomial(link = "logit"), data = data_exp)

summary(m6.1)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: correct ~ pas * target.frames + (1 + target.frames | subject)
##    Data: data_exp
##
##      AIC      BIC   logLik deviance df.resid
##  19506.1  19595.5  -9742.0  19484.1    25033
##
## Scaled residuals:
##      Min       1Q   Median       3Q      Max
## -19.0110   0.0537   0.1606   0.4849   1.4465
##
## Random effects:
##  Groups  Name          Variance Std.Dev. Corr
##  subject (Intercept)   0.03698  0.1923
##          target.frames 0.02057  0.1434   -0.76
## Number of obs: 25044, groups:  subject, 29
##
## Fixed effects:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)        -0.12163    0.06416  -1.896 0.058008 .
## pas2               -0.57140    0.08943  -6.389 1.67e-10 ***
## pas3               -0.53848    0.13956  -3.859 0.000114 ***
## pas4                0.20156    0.25059   0.804 0.421216
## target.frames       0.11480    0.03709   3.095 0.001966 **
## pas2:target.frames  0.44719    0.03475  12.869  < 2e-16 ***
## pas3:target.frames  0.74869    0.04595  16.293  < 2e-16 ***
## pas4:target.frames  0.75928    0.06850  11.084  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) pas2   pas3   pas4   trgt.f ps2:t. ps3:t.
## pas2        -0.462
## pas3        -0.308  0.248
## pas4        -0.174  0.121  0.092
## target.frms -0.811  0.306  0.208  0.124
## ps2:trgt.fr  0.482 -0.874 -0.245 -0.124 -0.428
## ps3:trgt.fr  0.393 -0.279 -0.891 -0.112 -0.359  0.371
## ps4:trgt.fr  0.276 -0.163 -0.121 -0.918 -0.260  0.225  0.200
```

For pas = 1, the accuracy increases with 0.1148 on the logit scale per increase in target.frames, this is an increase with a probability of almost 53%. For pas = 2 the increase is 0.1148+0.4472 = 0.562 on the logit scale per increase in target.frames, which is probability is around 53,5%. This indicates how the accuracy increases faster for pas 2 compared to pas 1 per increase in target frames.

```
######from logit to prob
#for pas 1:
est_1<- (coef(summary(m6.1))[5])
est.inv <- inv.logit(est_1)
est.inv
```

```
## [1] 0.5286684
```

```
# for pas 2:
est_2<- (coef(summary(m6.1))[5+6])
est_2_inv <- inv.logit(est_2)
est_2_inv
```

```
## [1] 0.5348322
```

2. `summary` won't allow you to test whether accuracy increases faster with objective evidence for PAS 3 than for PAS 2 (unless you use `relevel`, which you are not allowed to in this exercise). Instead, we'll be using the function `glht` from the `multcomp` package

    i. To redo the test in 6.1.i, you can create a *contrast* vector. This vector will have the length of the number of estimated group-level effects and any specific contrast you can think of can be specified using this. For redoing the test from 6.1.i, the code snippet below will do:

```
summary(m6.1)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: correct ~ pas * target.frames + (1 + target.frames | subject)
##    Data: data_exp
##
##      AIC      BIC   logLik deviance df.resid
##  19506.1  19595.5  -9742.0  19484.1    25033
##
## Scaled residuals:
##      Min       1Q   Median       3Q      Max
## -19.0110   0.0537   0.1606   0.4849   1.4465
##
## Random effects:
##  Groups  Name          Variance Std.Dev. Corr
##  subject (Intercept)   0.03698  0.1923
##          target.frames 0.02057  0.1434   -0.76
## Number of obs: 25044, groups:  subject, 29
##
## Fixed effects:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)        -0.12163    0.06416  -1.896 0.058008 .
## pas2               -0.57140    0.08943  -6.389 1.67e-10 ***
## pas3               -0.53848    0.13956  -3.859 0.000114 ***
## pas4                0.20156    0.25059   0.804 0.421216
## target.frames       0.11480    0.03709   3.095 0.001966 **
## pas2:target.frames  0.44719    0.03475  12.869  < 2e-16 ***
## pas3:target.frames  0.74869    0.04595  16.293  < 2e-16 ***
## pas4:target.frames  0.75928    0.06850  11.084  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) pas2   pas3   pas4   trgt.f ps2:t. ps3:t.
## pas2        -0.462
## pas3        -0.308  0.248
## pas4        -0.174  0.121  0.092
## target.frms -0.811  0.306  0.208  0.124
## ps2:trgt.fr  0.482 -0.874 -0.245 -0.124 -0.428
## ps3:trgt.fr  0.393 -0.279 -0.891 -0.112 -0.359  0.371
## ps4:trgt.fr  0.276 -0.163 -0.121 -0.918 -0.260  0.225  0.200
```

Snippet for 6.2.i

```
## testing whether PAS 2 is different from PAS 1
contrast.vector <- matrix(c(0, 0, 0, 0, 0, 1, 0, 0), nrow=1)

gh <- glht(m6.1, contrast.vector)
print(summary(gh))

inv.logit(0.44719)
```

this now give us an outcome who shows how there is a significant difference in how fast the accuracy increases when going from PAS 1 to PAS 2 (p < .001) Where the shift should be made are given by the 1 in the contrast vector. By taking the inverse logit of the estimate we find that the probability of an increase from PAS 1 till 2 to be: 60,997%

```
ii. Now test the hypothesis that accuracy increases faster with objective evidence fo
r PAS 3 than for PAS 2.
```

as another example, we could also test whether there is a difference in the increment of the accuracy between PAS 2 and PAS 3. This is now done by changing the place of the 1's in the contrast vector

```
contrast.vector1 <- matrix(c(0, -1, 1, 0, 0, 0, 0, 0), nrow=1)
gh1 <- glht(m6.1, contrast.vector1)
print(summary(gh1))
```

```
##
##   Simultaneous Tests for General Linear Hypotheses
##
## Fit: glmer(formula = correct ~ pas * target.frames + (1 + target.frames |
##     subject), data = data_exp, family = binomial(link = "logit"))
##
## Linear Hypotheses:
##        Estimate Std. Error z value Pr(>|z|)
## 1 == 0  0.03292    0.14587   0.226    0.821
## (Adjusted p values reported -- single-step method)
```

```
inv.logit(0.03292)
```

```
## [1] 0.5082293
```

It can here be concluded that there is no significant difference between PAS 2 and 3 (p > .05) By taking the inverse logit of the estimate we find that the probability of an increase from PAS 2 till 3 to be: 50,823%

this can now be checked for the slope as well - before it was only intercept:

```
contrast.vector11 <- matrix(c(0, 0, 0, 0, 0, -1, 1, 0), nrow=1)
gh11 <- glht(m6.1, contrast.vector11)
print(summary(gh11))
```

```
##
##   Simultaneous Tests for General Linear Hypotheses
##
## Fit: glmer(formula = correct ~ pas * target.frames + (1 + target.frames |
##     subject), data = data_exp, family = binomial(link = "logit"))
##
## Linear Hypotheses:
##        Estimate Std. Error z value Pr(>|z|)
## 1 == 0  0.30150    0.04621   6.524 6.85e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## (Adjusted p values reported -- single-step method)
```

Here we see how the difference in slope are significant (p < .001)

```
iii. Also test the hypothesis that accuracy increases faster with objective evidence
for PAS 4 than for PAS 3
```

```
#between pas 3 and 4
contrast.vector2 <- matrix(c(0, 0, 0, 0, 0, 0, -1, 1), nrow=1)
gh2 <- glht(m6.1, contrast.vector2)
print(summary(gh2))
```

```
##
##   Simultaneous Tests for General Linear Hypotheses
##
## Fit: glmer(formula = correct ~ pas * target.frames + (1 + target.frames |
##     subject), data = data_exp, family = binomial(link = "logit"))
##
## Linear Hypotheses:
##        Estimate Std. Error z value Pr(>|z|)
## 1 == 0  0.01060    0.07445   0.142    0.887
## (Adjusted p values reported -- single-step method)
```

```
inv.logit(0.01060)
```

```
## [1] 0.50265
```

This is not significant, indicating that the perception in both pass 3 and 4 is enough for the subject to have the same precision in their accuracy of the answers (correct vs. wrong)

By taking the inverse logit of the estimate we find that the probability of an increase from PAS 3 till 4 to be: 50,26%

3. Finally, test that whether the difference between PAS 2 and 1 (tested in 6.1.i) is greater than the difference between PAS 4 and 3 (tested in 6.2.iii)

```
#taking the log of the gh values to make them comparable:
logit <- function(x) log(x / (1 - x))
log_gh <- inv.logit(0.44719)
log_gh2 <- inv.logit(0.01060)

#so if there is a difference between gh and gh2:
compare_gh <- tibble("Pas" = c("Pas 2 and 1", "Pas 4 and 3"), "GH Value" = c("0.4363"
, 0.002838), "Logged Value" = c(log_gh, log_gh2))
compare_gh
```

Pas	**GH Value**	**Logged Value**
<chr>	<chr>	<dbl>
Pas 2 and 1	0.4363	0.6099709
Pas 4 and 3	0.002838	0.5026500

2 rows

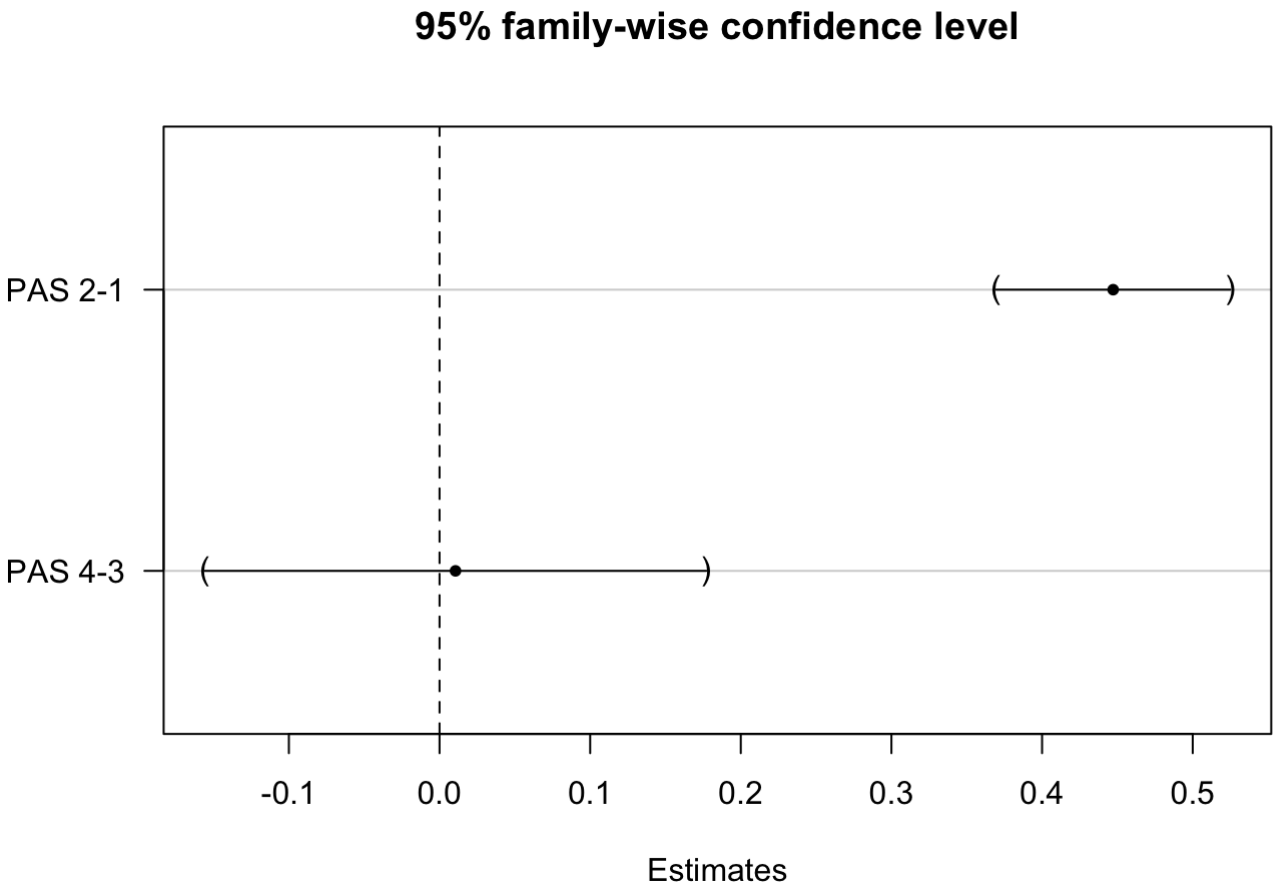
*There is clearly is a difference, but to test if they are significally different more should be done …

```
#binding the two pas matrixes into one
contrast.matrix <- rbind(c(0, 0, 0, 0, 0, 1, 0, 0), c(0, 0, 0, 0, 0, 0, -1, 1))
rownames(contrast.matrix) <- c("PAS 2-1", "PAS 4-3")
gh <- glht(m6.1, contrast.matrix)

print(summary(gh))
```

```
##
##   Simultaneous Tests for General Linear Hypotheses
##
## Fit: glmer(formula = correct ~ pas * target.frames + (1 + target.frames |
##     subject), data = data_exp, family = binomial(link = "logit"))
##
## Linear Hypotheses:
##               Estimate Std. Error z value Pr(>|z|)
## PAS 2-1 == 0  0.44719    0.03475  12.869   <1e-10 ***
## PAS 4-3 == 0  0.01060    0.07445   0.142    0.987
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## (Adjusted p values reported -- single-step method)
```

```
#now we see that the estimates clearly are different which can be plottet:
```

```
# a vizualization of the difference:
plot(gh, xlab= "Estimates")
```



Eyeballing the plot we see no overlap in the two 95% confidence intervals, which indicates a 'true' difference between the differences of PAS2-PAS1 (0.44718) and PAS4-PAS3 (0.01060).

EXERCISE 7 - Estimate psychometric functions for the Perceptual Awareness Scale and evaluate them

We saw in 5.3 that the estimated functions went below chance at a target duration of 0 frames (0 ms). This does not seem reasonable, so we will be trying a different approach for fitting here.
We will fit the following function that results in a sigmoid, $f(x) = a + \frac{b-a}{1+e^{\frac{c-x}{d}}}$

It has four parameters: *a*, which can be interpreted as the *minimum accuracy level*, *b*, which can be interpreted as the *maximum accuracy level*, *c*, which can be interpreted as the so-called *inflexion point*, i.e. where the derivative of the sigmoid reaches its maximum and *d*, which can be interpreted as the *steepness at the inflexion point*. (When *d* goes towards infinity, the slope goes towards a straight line, and when it goes towards 0, the slope goes towards a step function).

We can define a function of a residual sum of squares as below

```
RSS <- function(dataset, par)
{
    ## "dataset" should be a data.frame containing the variables x (target.frames)
    ## and y (correct)

    ## "par" are our four parameters (a numeric vector)
    a <- par[1]
    b<- par[2]
    c <- par[3]
    d <- par[4]


    x <- dataset$x
    y <- dataset$y
    ## you fill in the estimate of y.hat
    y.hat <-  a + ((b-a)/(1+exp(1)^((c-x)/d)))
    RSS <- sum((y - y.hat)^2)
    return(RSS)
}
```

1. Now, we will fit the sigmoid for the four PAS ratings for Subject 7

```
#making a new dataframe only for participant 7
data_7.1 <- data_exp %>%
  dplyr::select(subject, pas, target.frames, correct) %>%
  rename(x = target.frames, y = correct)

dataset7 <- data_7.1 %>%
  filter(subject == "007")

### making a new data set for each pas for subject 7

sub_7_pas_1 <- dataset7 %>%
  filter(subject == "007" & pas == 1)

sub_7_pas_2 <- dataset7 %>%
  filter(subject == "007" & pas == 2)

sub_7_pas_3 <- dataset7 %>%
  filter(subject == "007" & pas == 3)

sub_7_pas_4 <- dataset7 %>%
  filter(subject == "007" & pas == 4)
```

i. use the function `optim`. It returns a list that among other things contains the f
our estimated parameters. You should set the following arguments:
`par`: you can set _c_ and _d_ as 1. Find good choices for _a_ and _b_ yourself (and
argue why they are appropriate)
`fn`: which function to minimise?
`data`: the data frame with _x_, _target.frames_, and _y_, _correct_ in it
`method`: 'L-BFGS-B'
`lower`: lower bounds for the four parameters, (the lowest value they can take), you
can set _c_ and _d_ as `-Inf`. Find good choices for _a_ and _b_ yourself (and argue
why they are appropriate)
`upper`: upper bounds for the four parameters, (the highest value they can take) can
set _c_ and _d_ as `Inf`. Find good choices for _a_ and _b_ yourself (and argue why t
hey are appropriate)

```
##
optim1 <- optim(c(0.5, 1.00, 1.00, 1.00), fn = RSS, data = sub_7_pas_1, method = 'L-B
FGS-B', lower = c(0.5, 0.5, -Inf, -Inf), upper = c(1, 1, Inf, Inf))

optim2 <- optim(c(0.5, 1.00, 1.00, 1.00), fn = RSS, data = sub_7_pas_2, method = 'L-B
FGS-B', lower = c(0.5, 0.5, -Inf, -Inf), upper = c(1, 1, Inf, Inf))

optim3 <- optim(c(0.5, 1.00, 1.00, 1.00), fn = RSS, data = sub_7_pas_3, method = 'L-B
FGS-B', lower = c(0.5, 0.5, -Inf, -Inf), upper = c(1, 1, Inf, Inf))

optim4 <- optim(c(0.5, 1.00, 1.00, 1.00), fn = RSS, data = sub_7_pas_4, method = 'L-B
FGS-B', lower = c(0.5, 0.5, -Inf, -Inf), upper = c(1, 1, Inf, Inf))


## printing the optimal values for each:
print(c(optim1, optim2, optim3, optim4))
```

```
## $par
## [1] 0.500000 0.500000 2.218342 1.449114
##
## $value
## [1] 45.75
##
## $counts
## function gradient
##        5        5
##
## $convergence
## [1] 0
##
## $message
## [1] "CONVERGENCE: NORM OF PROJECTED GRADIENT <= PGTOL"
##
## $par
## [1] 0.53333057 0.61111627 2.00265034 0.06488018
##
## $value
## [1] 31.75556
##
## $counts
## function gradient
##       66       66
##
## $convergence
## [1] 0
##
## $message
## [1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
##
## $par
## [1] 0.5000000 0.9259250 1.9182875 0.0754563
##
## $value
## [1] 7.476435
##
## $counts
## function gradient
##       37       37
##
## $convergence
## [1] 0
##
## $message
## [1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
##
## $par
## [1] 0.5000000 0.9900976 1.1927144 0.4163145
##
## $value
## [1] 5.871986
##
## $counts
```

```
## function gradient
##       45       45
##
## $convergence
## [1] 0
##
## $message
## [1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

```
#the upper and lower in here are set to 0 and 1, since it is a sigmoid function.
```

Argument: the minimum accuracy level is set to 0,5 since it is then what would be due to chance (50%). The maximum is set to 1 since I find i reasonable that a few might get an accuracy of 100%, and no one will for sure be above.

This now give us new suggested parameters, that will be the most optimal to use for each pas

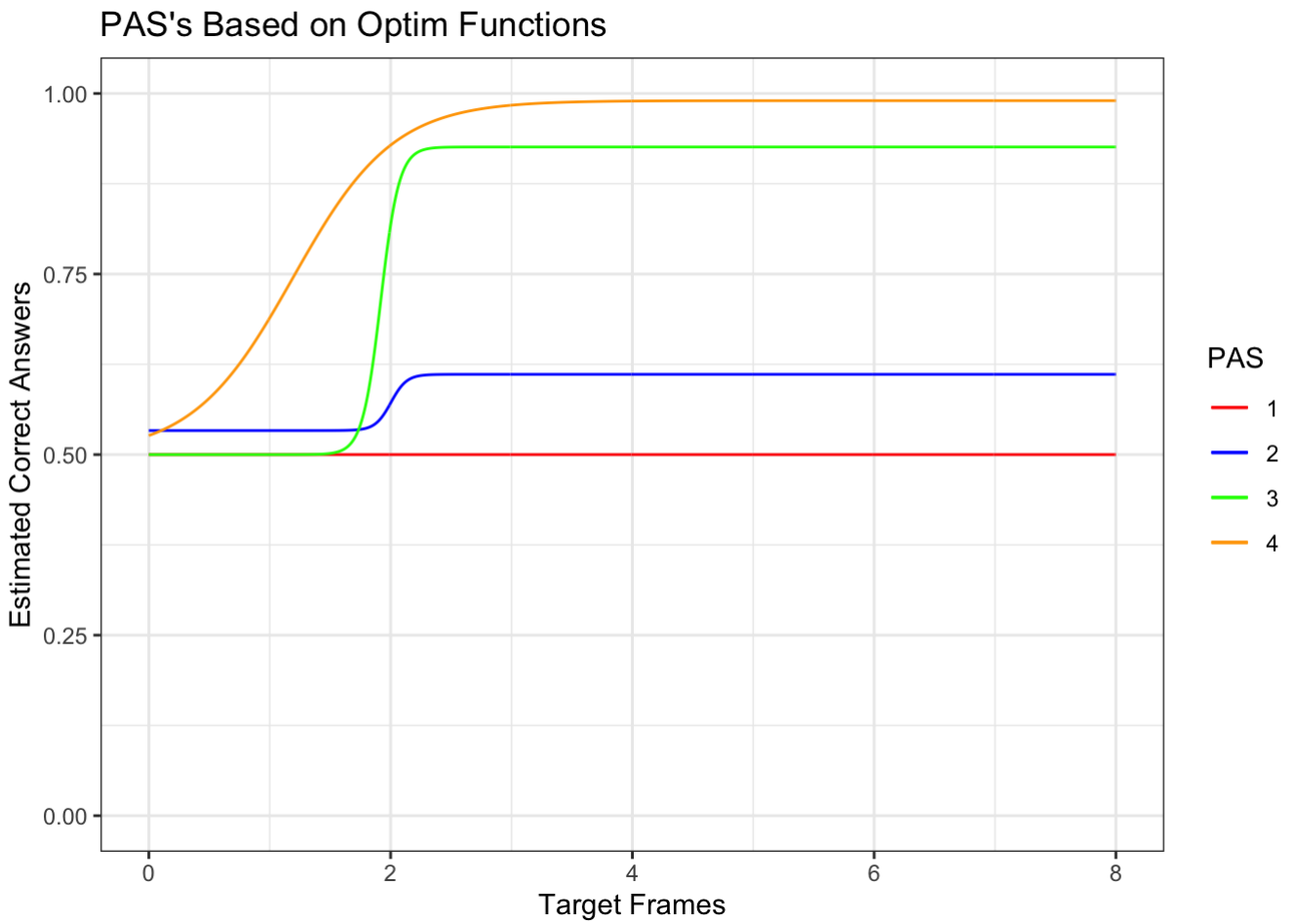
```
ii. Plot the fits for the PAS ratings on a single plot (for subject 7) `xlim=c(0, 8)`
```

to do so we must first calculate the y hats:

```
y_hat_func <- function(a, b, c, d, x) {
  y.hat <- a + ((b-a)/(1+exp(1)^((c-x)/d)))
  return(y.hat)
}

#making and empty frame for the yhats:
optim_data <- data.frame(cbind("x" = seq(0, 8, by = 0.01)))

#for pas1
optim_data$yhat1 <- y_hat_func(optim1$par[1], optim1$par[2], optim1$par[3],optim1$par
[4], optim_data$x)
#for pas2
optim_data$yhat2 <- y_hat_func(optim2$par[1], optim2$par[2], optim2$par[3],optim2$par
[4], optim_data$x)
#for pas3
optim_data$yhat3 <- y_hat_func(optim3$par[1], optim3$par[2], optim3$par[3],optim3$par
[4], optim_data$x)
#for pas4
optim_data$yhat4 <- y_hat_func(optim4$par[1], optim4$par[2], optim4$par[3],optim4$par
[4], optim_data$x)
```

this can now be plottet ….

```
plot_optim <- ggplot(optim_data) +
  geom_line(aes(x=x, y=yhat1, color = "1"))+
  geom_line(aes(x=x, y=yhat2, color = "2"))+
  geom_line(aes(x=x, y=yhat3, color = "3"))+
  geom_line(aes(x=x, y=yhat4, color = "4"))+
  scale_color_manual(name = "PAS", values = c("1" = "red", "2" = "blue", "3" = "gree
n", "4" = "orange"))+
  xlim(c(0,8))+
  ylim(c(0,1))+
  labs(y = "Estimated Correct Answers", x = "Target Frames", title = "PAS's Based on
 Optim Functions")+
  theme_bw()


plot_optim
```

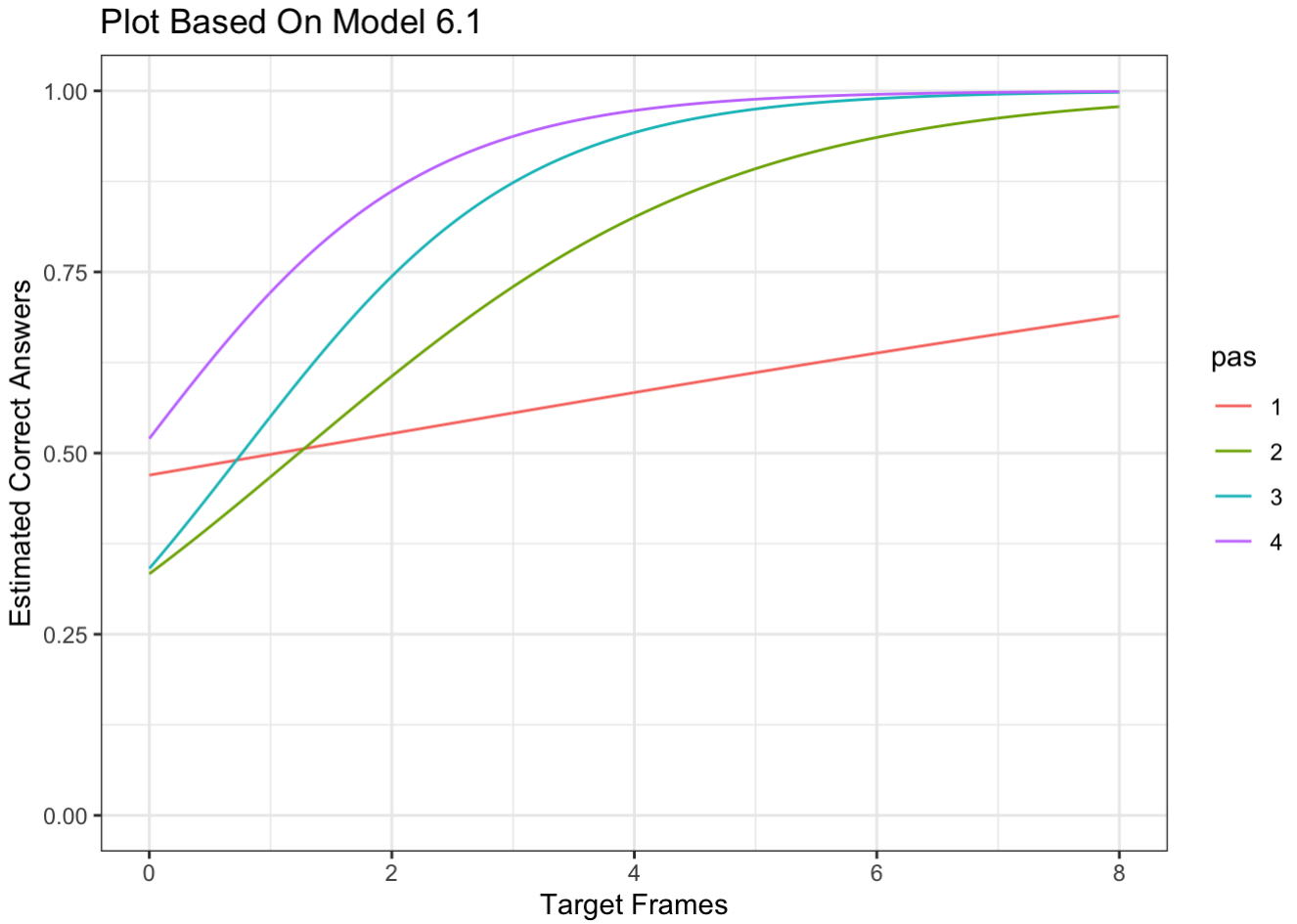


iii. Create a similar plot for the PAS ratings on a single plot (for subject 7), but this time based on the model from 6.1 `xlim=c(0, 8)`

to do so (getting an xlim 0-8) we must make predictions for the y values (correct) for each pas per subject 7, across all for pass. I have chosen to use a model fitted on all the data (so all subjects) and then using it to predict new y values, specific for subject 7. This is done to avoid fitting the model to the same data twice and making the prediction within sample - if I fit on subject 7 only and then afterwards try to predict on subject 7 I get the same values. Additionally, subjects are taken into account in the big data the model is fitted on, thus I find this to be the solution.

```
#to get all the points we neew to make the lines go from 0-8 on the x axe we generate
the data:
model6_data_new <- data.frame(cbind("x" = seq(0, 8, by = 0.01),"pas1" = rep(1,801),"p
as2" = rep(2,801), "pas3" = rep(3,801), "pas4" = rep(4,801), "subject" = rep(7,801)))

##combining all the passes to make them work with the model:
model6_pivot <- model6_data_new %>%
  pivot_longer(cols = pas1:pas4, values_to = "pas")

## model:
m6.1_new <- glmer(y ~ pas * x + (1+x | subject), family = binomial(link = "logit"), d
ata = data_7.1)

## factorising:
model6_pivot$pas <- as.factor(model6_pivot$pas)
model6_pivot$subject <- as.factor(model6_pivot$subject)


model6_pivot$yhat_model6 <- predict(m6.1_new, re.form = ~(1+x| subject), newdata = mo
del6_pivot, type = "response", allow.new.levels = TRUE)
```

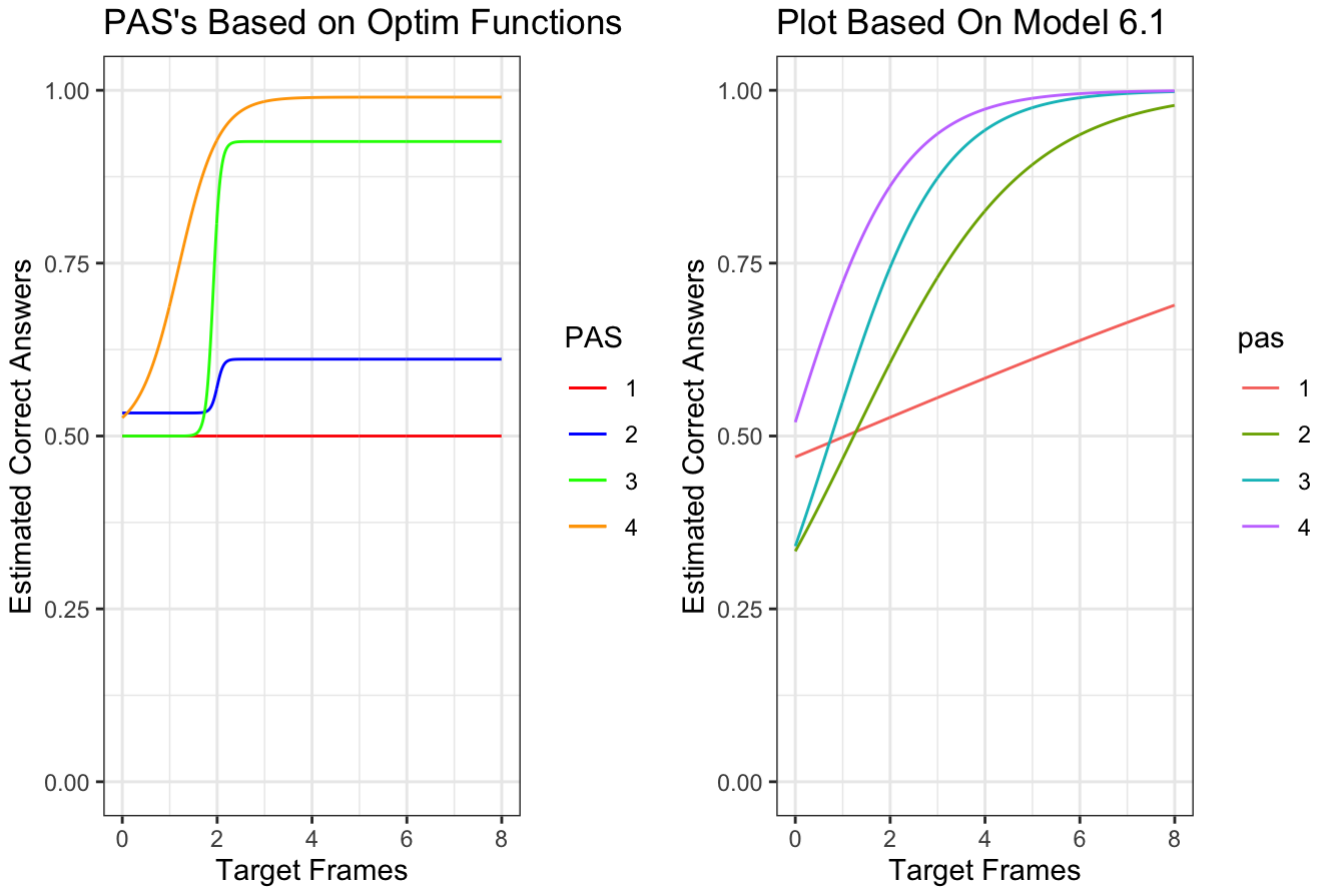
```
#plot
plot6.1 <- model6_pivot %>%
  ggplot(aes(x = x, y = yhat_model6, colour = pas)) +
  xlim(c(0,8))+
  ylim(c(0,1))+
  geom_line(aes(x = x, y = yhat_model6))+
  labs(title = "Plot Based On Model 6.1",
       x = "Target Frames",
       y = "Estimated Correct Answers")+
  theme_bw()

plot6.1
```



This can now be compared to the sigmoid function plot over pas ratings.

```
cool_com <- ggarrange(plot_optim, plot6.1)
annotate_figure(cool_com, top = text_grob("PAS fits for Subject 7", color = "darkblu
e", face = "italic", size = 18))
```

PAS fits for Subject 7



iv. Comment on the differences between the fits - mention some advantages and disadvantages of each way

The plot appears to have quite the same tendencies, at first glance. However, by a closer look especially pas 2 differs a lot in its relation to the other lines in the two plots. In the optim function the correctness is quite low, whereas it is almost as good as in pas 3 in the model 6.1 plot.

By looking at the Optim plot, we see how the optim function has fixed the problem of having less correct answers then if due to chance, which I'd say is a big advantage! This stems from us setting the accepted minimum at a=0.5. This however can't be done when the correctness of answers are estimated on the general mixed model m6.1, thus we see the slopes being below 0.5 in the model 6.1 plot.

2. Finally, estimate the parameters for all subjects and each of their four PAS ratings. Then plot the estimated function at the group-level by taking the mean for each of the four parameters, *a*, *b*, *c* and *d* across subjects. A function should be estimated for each PAS-rating (it should look somewhat similar to Fig. 3 from the article: https://doi.org/10.1016/j.concog.2019.03.007 (https://doi.org/10.1016/j.concog.2019.03.007))

```
#making a loop that creates pas ratings estimated by the optim function per subject
cool_loop_function <- function(){
  data_frame_parameters <- data.frame(subject = NA, pas = NA , a = NA, b = NA, c = NA
, d = NA)


  for (i in 1:4){
    df_temp1 <- data_exp %>%
      filter(pas == i) %>%
      mutate(subject = as.numeric(subject))

    for (ii in 1:length(unique(data_exp$subject))){
      data_temp <- df_temp1 %>%
        filter(subject == ii) %>%
        dplyr::select(target.frames, correct, pas) %>%
        rename(x = target.frames, y = correct)

      op_temp = optim(par = c(0.5,0.5,1,1), fn = RSS, data = data_temp, method = "L-B
FGS-B", lower = c(0.5,0.5,-Inf,-Inf),
      upper = c(1,1,Inf,Inf))
      data_frame_parameters <- rbind(data_frame_parameters , c(ii,i,op_temp$par))

    }
  }
  return(data_frame_parameters)
}

df_loop_pasratings <- cool_loop_function() %>%
  na.omit() #removing na's



df_loop_pasratings
```

	subject	pas	a	b	c	d
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
2	1	1	0.5000000	0.5000000	1.0000000	1.0000000000
3	2	1	0.6072665	0.5000000	0.9206736	0.7154031594
4	3	1	0.6248181	0.5374096	0.8602455	0.4300550152
5	4	1	0.7593121	0.5000000	0.9151872	0.0932777361
6	5	1	0.5000000	0.5000000	1.0000000	1.0000000000
7	6	1	0.6143188	0.5000000	1.0058600	0.9558662745
8	7	1	0.5000000	0.5000000	1.0000000	1.0000000000
9	8	1	0.6165075	0.5000000	0.9436198	0.8557934710
10	9	1	0.5000000	0.5391376	1.7117410	0.0203760864
11	10	1	0.5000000	0.5000000	1.0000000	1.0000000000

1-10 of 116 rows Previous **1** 2 3 4 5 6 … 12 Next

this now gives us a dataframe containing the parameters for each subject in each pas, which we then can take the mean of:

```
#taking the average of all subjects ratings per pas:
mean_rating <- df_loop_pasratings %>%
  group_by(pas) %>%
  summarise(a = mean(a), b = mean(b), c = mean(c), d = mean(d))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
mean_rating
```

pas <dbl>	a <dbl>	b <dbl>	c <dbl>	d <dbl>
1	0.5796431	0.5514403	1.638810	0.4571415
2	0.5274088	0.8775388	2.832406	0.1749703
3	0.5768916	0.9556266	2.281409	0.2259202
4	0.7198609	0.9658502	2.190136	0.5012006

4 rows

Having the mean of each parameter in each class we can now estimate the y values for all subjects, which is originally estimated by the optim.

```
#making y values per pass:
y_for_pas <- function() {
  x <- seq(1, 8, 0.1)
  y_df = data.frame(x)

  for (i in 1:4){
  df_temp <- mean_rating %>%
      filter(pas == i)

   y_df[,i+1] <- y_hat_func(df_temp$a, df_temp$b, df_temp$c, df_temp$d, x)

  }
y_df <- y_df %>%
  rename("1" = V2, "2" = V3, "3" = V4, "4" = V5)

return(y_df)

}

df_est_pas <- y_for_pas()
df_est_pas
```

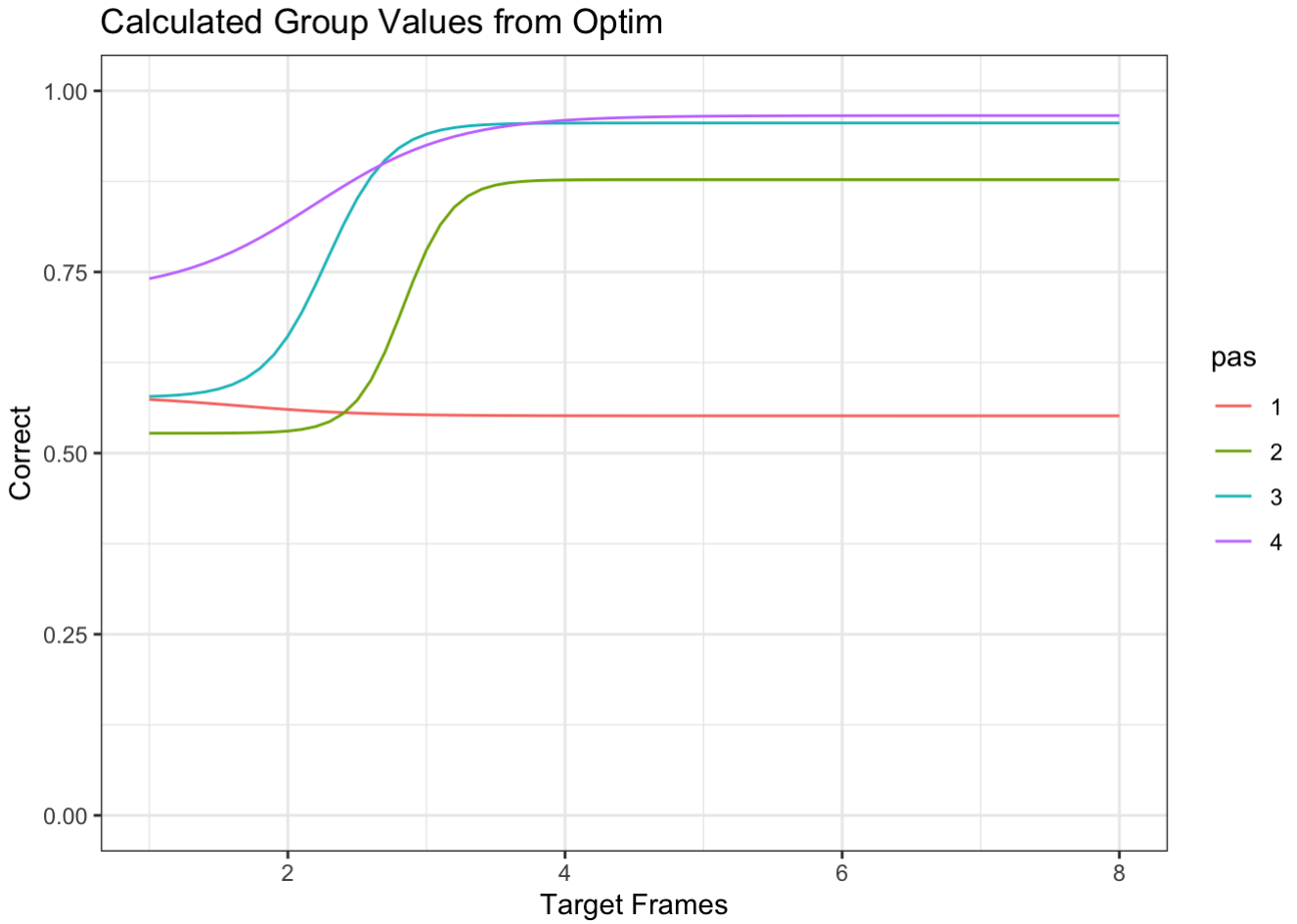
x <dbl>	1 <dbl>	2 <dbl>	3 <dbl>	4 <dbl>
1.0	0.5740525	0.5274187	0.5781904	0.7408026

x <dbl>	1 <dbl>	2 <dbl>	3 <dbl>	4 <dbl>
1.1	0.5730071	0.5274264	0.5789097	0.7449551
1.2	0.5718338	0.5274399	0.5800241	0.7498215
1.3	0.5705406	0.5274638	0.5817460	0.7554794
1.4	0.5691434	0.5275062	0.5843954	0.7619973
1.5	0.5676663	0.5275813	0.5884460	0.7694269
1.6	0.5661399	0.5277142	0.5945791	0.7777943
1.7	0.5645993	0.5279493	0.6037297	0.7870907
1.8	0.5630810	0.5283649	0.6170872	0.7972649
1.9	0.5616194	0.5290985	0.6359767	0.8082177

1-10 of 71 rows Previous **1** 2 3 4 5 6 … 8 Next

Plotting the estimated function at the group-level by taking the mean for each of the four parameters, a, b, c and d across subjects:

```
df_final <- df_est_pas %>%
  pivot_longer(cols = c("1", "2","3", "4") , names_to = "pas", values_to = "y_hat_mer
ged")

plot_final <- df_final %>%
  ggplot(aes(x, y_hat_merged, color = pas))+
  geom_line() +
  ylim(0,1)+
  labs(title = "Calculated Group Values from Optim",
       x = "Target Frames",
       y = "Correct") +
  theme_bw()

plot_final
```



i. compare with the figure you made in 5.3.ii and comment on the differences between the fits - mention some advantages and disadvantages of both.

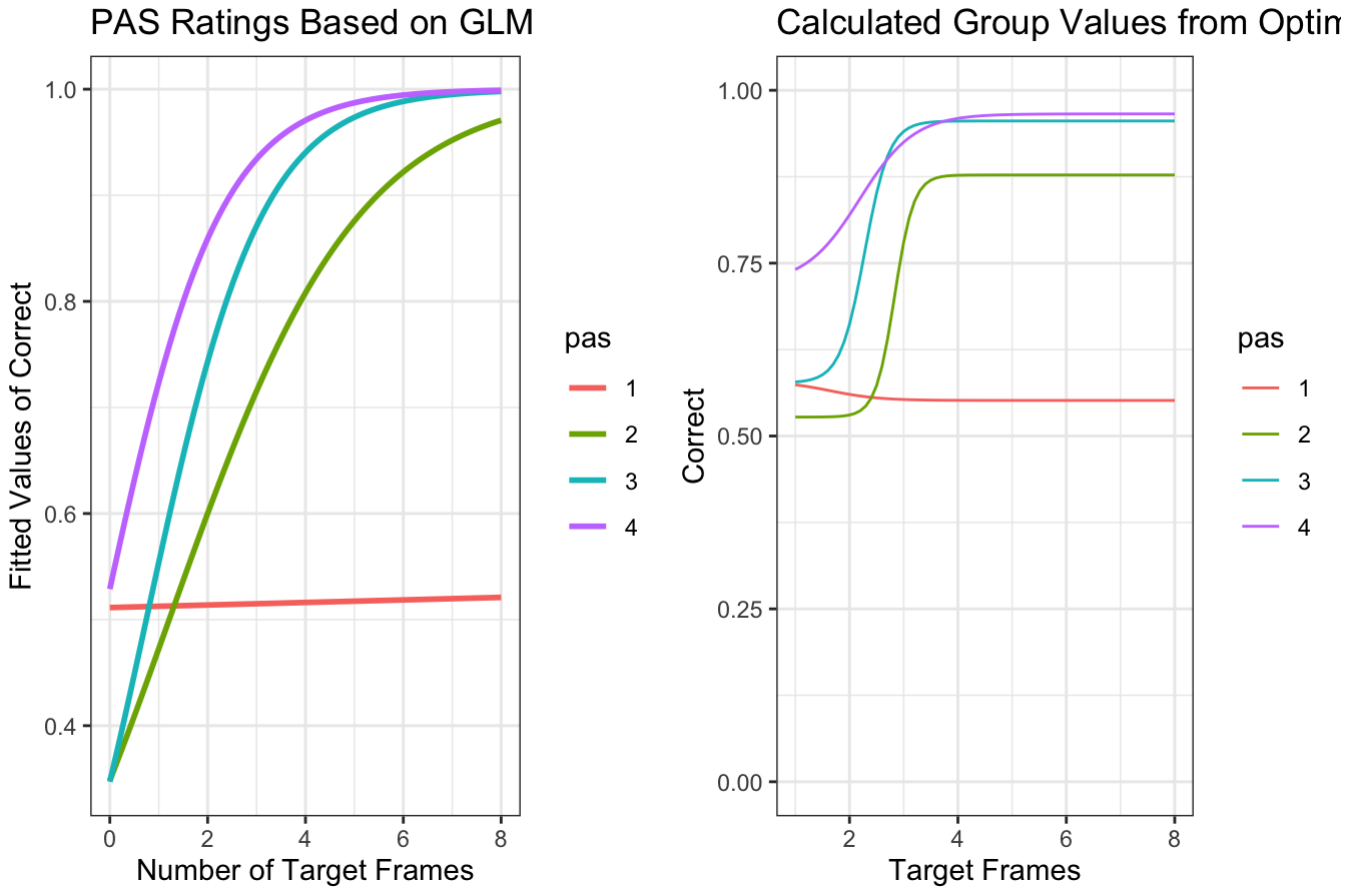
```
last_com <- ggarrange(plot_5.3 , plot_final)
```

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

```
## Warning in eval(family$initialize): non-integer #successes in a binomial glm!

## Warning in eval(family$initialize): non-integer #successes in a binomial glm!

## Warning in eval(family$initialize): non-integer #successes in a binomial glm!

## Warning in eval(family$initialize): non-integer #successes in a binomial glm!
```

```
annotate_figure(last_com, top = text_grob("Comparison of PAS Plots", color = "darkblu
e", face = "italic", size = 18))
```

Comparison of PAS Plots



Just by eyeballing the plots, the lines for each pas is quite similar in relationship to one another. Looking at the optim group plot at the right the line for pas 1 is weird looking, since it between target frame 1-3 it declines, since a is higher than b (as found in df_loop_pasratings). However in this plot (group optim) the lines are not at any point below 0.5 on the y axes, which is an advantages the GLM plot does not have - this means that the estimated accuracy of correct answers never gets worse than by due to chance. Moreover, The GLM plot doesn't seem to act as an entire sigmoid function either, which I intuitively would say it should, to reflect the data properly - an ability the group optim plot possesses.

PORTFOLIO 3

Laura Wulff Paaby
202006161
Cognitive Science BA
Aarhus University AU

13/12 - 2021

Portfolio 3 , Methods 3, 2021, autumn semester

Laura W. Paaby

1/12 - 2021

Exercises and objectives

1) Load the magnetoencephalographic recordings and do some initial plots to understand the data 2) Do logistic regression to classify pairs of PAS-ratings 3) Do a Support Vector Machine Classification on all four PAS-ratings REMEMBER: In your report, make sure to include code that can reproduce the answers requested in the exercises below (MAKE A KNITTED VERSION) REMEMBER: This is Assignment 3 and will be part of your final portfolio

EXERCISE 1 - Load the magnetoencephalographic recordings and do some initial plots to understand the data

1) Load megmag_data.npy and call it data using np.load. You can use join, which can be imported from os.path, to create paths from different string segments i. The data is a 3-dimensional array. The first dimension is number of repetitions of a visual stimulus, the second dimension is the number of sensors that record magnetic fields (in Tesla) that stem from neurons activating in the brain, and the third dimension is the number of time samples. How many repetitions, sensors and time samples are there?

In [ ]:
```
## importing libraries
import statistics
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import RANSACRegressor
from sklearn.model_selection import train_test_split
import scipy as sp
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.linear_model import ElasticNet
from sklearn.preprocessing import PolynomialFeatures
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
import os
```

In [ ]:
```
#data = np.load("/Users/laura/Desktop/megmag_data.npy")

import requests
import io

response = requests.get('http://laumollerandersen.org/data_methods_3/megmag_d
response.raise_for_status()
data = np.load(io.BytesIO(response.content))

#numbers of repitions of visual stimuli: 682
#numbers of sensors (spots in the cap that record magnetic fields): 102
#numbers of time samples: 251
#### this is found by:
print(data.shape)
```

```
(682, 102, 251)
```

ii. The time range is from (and including) -200 ms to (and including) 800 ms with a sample recorded every 4 ms. At time 0, the visual stimulus was briefly presented. Create a 1-dimensional array called `times` that represents this.

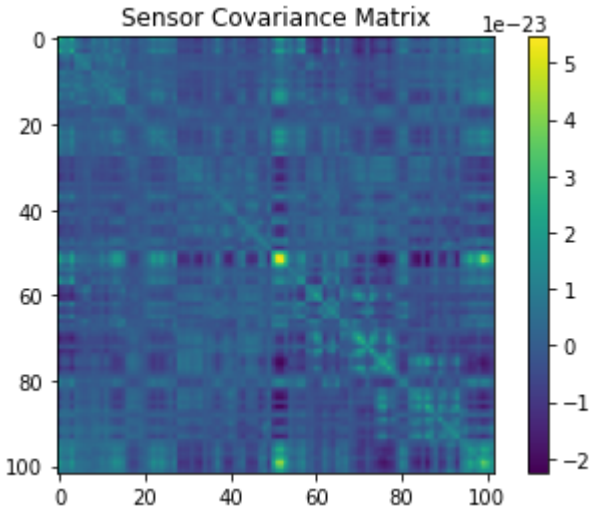
In [ ]:
```
#creating array of time samples:
time_array = np.arange(-200, 804, 4)
```

iii. Create the sensor covariance matrix Σ_{XX}:

$$\Sigma_{XX} = \frac{1}{N}\sum_{i=1}^{N} XX^T$$

N is the number of repetitions and X has s rows and t columns (sensors and time), thus the shape is $X_{s \times t}$. Do the sensors pick up independent signals? (Use `plt.imshow` to plot the sensor covariance matrix)

In [ ]:
```
n = 682
cov_mat = []

#calculating the dot product for all rows i using all datapoints in the dimen
for i in range(n):
    cov_mat.append(data[i,:,:] @ data[i,:,:].T)

#out of the loop the dot product of the the matrices for each i is summed and
cov_mat = sum(cov_mat)/n

#plotting the covariance matrix
plt.imshow(cov_mat)
plt.title("Sensor Covariance Matrix")
plt.colorbar()
plt.show()
```



The plot here is hard to interpret, due to the amount of pixels. However, some areas seem to be troubeling having a rather high covariance (the colours are not around 0 according to the colourscale).

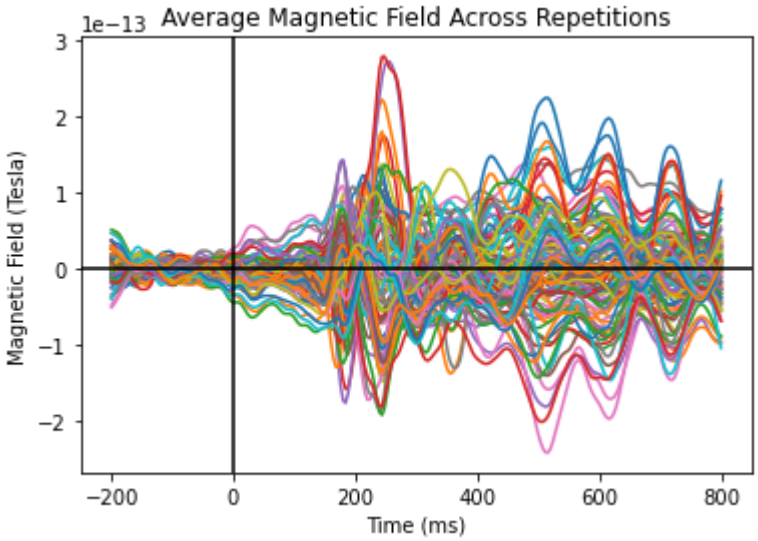
iv. Make an average over the repetition dimension using `np.mean` - use the `axis` argument. (The resulting array should have two dimensions with time as the first and magnetic field as the second)

In [ ]:
```
### we take the mean of all the repetions, the first axis = 0 - this now give
rep_mean = np.mean(data, axis=0)
###this should be transposed ?????
print(rep_mean.shape)
```

```
(102, 251)
```

v. Plot the magnetic field (based on the average) as it evolves over time for each of the sensors (a line for each) (time on the x-axis and magnetic field on the y-axis). Add a horizontal line at $y = 0$ and a vertical line at $x = 0$ using `plt.axvline` and `plt.axhline`

In [ ]:
```
plt.figure()
plt.plot(time_array, rep_mean.T) #taking the transposed gives the same as if
plt.title("Average Magnetic Field Across Repetitions")
plt.xlabel("Time (ms)")
plt.ylabel("Magnetic Field (Tesla)")
plt.axvline(0,0, color = "black")
plt.axhline(0,0, color = "black")
plt.show()
```



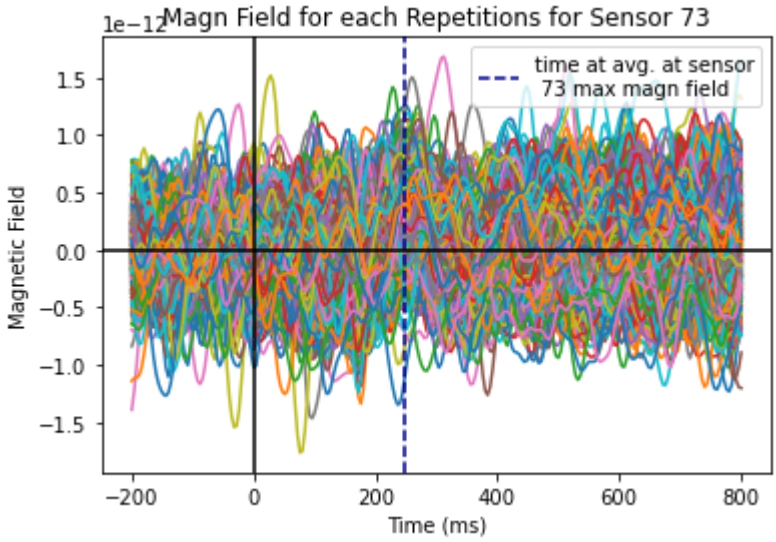
vi. Find the maximal magnetic field in the average. Then use `np.argmax` and `np.unravel_index` to find the sensor that has the maximal magnetic field.

In [ ]:
```
#finding the max - but this gives a coordinate in the dataframe of mean times
maxi_mag = np.unravel_index(np.argmax(rep_mean), rep_mean.shape)
print(maxi_mag) #sensor 73, repition 112
#the coordinates can be printed as a number:
print("the value of the maximal of magnetic field:",rep_mean[73,112])
# = 2.7886216843591933e-13 which is a super small number, yet it is our maxim
```

```
(73, 112)
the value of the maximal of magnetic field: 2.7886216843591933e-13
```

vii. Plot the magnetic field for each of the repetitions (a line for each) for the sensor that has the maximal magnetic field. Highlight the time point with the maximal magnetic field in the average (as found in 1.1.v) using `plt.axvline`

In [ ]:
```
#first I save the sensor found to have the max mag field avg
sensor73 = data[:,73,:]

#now the times and repitions for this can be plottet (Taking one sensor only
plt.figure()
plt.plot(time_array, sensor73.T)
plt.axvline(time_array[112], color = "darkblue", linestyle = "--", label = "t
plt.legend(loc = "upper right")
plt.axvline(color = "black")
plt.axhline(color = "black")
plt.title("Magn Field for each Repetitions for Sensor 73")
plt.xlabel("Time (ms)")
plt.ylabel("Magnetic Field")
plt.show()

#### we have plottet the 112 time *point*, which is not corresponding to the
```



viii. Describe in your own words how the response found in the average is represented in the single repetitions. But do make sure to use the concepts *signal* and *noise* and comment on any differences on the range of values on the y-axis:

in this plot we no longer plot means, but all repition for sensor 73 (in the first we have all sensors but the mean repitions). In this plot it is very hard to interpret visually, since we cant distinct one line from another in this plot. It therefore does not really say much about the indiviudal repitions for sensor 73. In other words, the signal is rather hard to interpret due to the amount of noise this plot contains. However a few spikes in the magnetic fiels can still be spotted.

2) Now load `pas_vector.npy` (call it `y`).

PAS is the same as in Assignment 2, describing the clarity of the subjective experience the subject reported after seeing the briefly presented stimulus

In [ ]:
```
#vector_y = np.load("/Users/laura/Desktop/pas_vector.npy")

response = requests.get('http://laumollerandersen.org/data_methods_3/pas_vect
response.raise_for_status()
vector_y = np.load(io.BytesIO(response.content))
```

i. Which dimension in the `data` array does it have the same length as?

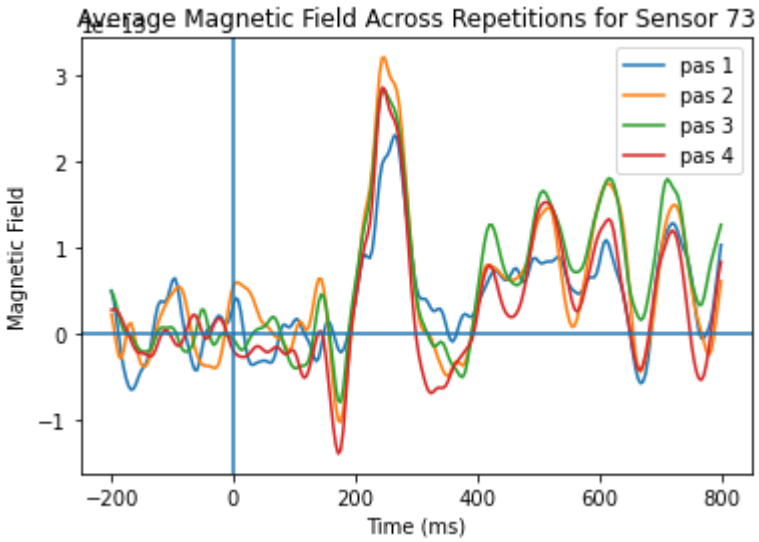
In [ ]:
```
print(vector_y.shape)
```

```
(682,)
```

ii. Now make four averages (As in Exercise 1.1.iii), one for each PAS rating, and plot the four time courses (one for each PAS rating) for the sensor found in Exercise 1.1.v

In [ ]:
```
idx_1 = np.argwhere(vector_y == 1)
idx_2 = np.argwhere(vector_y == 2)
idx_3 = np.argwhere(vector_y == 3)
idx_4 = np.argwhere(vector_y == 4)

#making new df only for73
sensor73 = data[:,73,:]
```

```
avgpas1 = np.mean(np.squeeze(sensor73[idx_1]), axis = 0)
avgpas2 = np.mean(np.squeeze(sensor73[idx_2]), axis = 0)
avgpas3 = np.mean(np.squeeze(sensor73[idx_3]), axis = 0)
avgpas4 = np.mean(np.squeeze(sensor73[idx_4]), axis = 0)

#plotting it
plt.figure()
plt.plot(time_array, avgpas1)
plt.plot(time_array, avgpas2)
plt.plot(time_array, avgpas3)
plt.plot(time_array, avgpas4)
plt.axvline()
plt.axhline()
plt.legend(['pas 1', 'pas 2', 'pas 3', 'pas 4'])
plt.title("Average Magnetic Field Across Repetitions for Sensor 73")
plt.xlabel("Time (ms)")
plt.ylabel("Magnetic Field")
plt.show()

## this now show the mean of all magnitudes in the respective pass
```



iii. Notice that there are two early peaks (measuring visual activity from the brain), one before 200 ms and one around 250 ms. Describe how the amplitudes of responses are related to the four PAS-scores. Does PAS 2 behave differently than expected?

first we notice how the pas1 curve fits the expectations: no perceptual experience leads to least amount of activity. In allignemnt with this logic we should expect the pas ratings to gradually increase in activation. However, PAS 2 appears to spike the most and have the highest magnitude. This is a bit odd, since you don't see the stimuli that clear. An idea could be that you therefore use a lot of brain capacity to perceive what is actually shown. This idea could have been investigated further, if we knew where the exact sensor where placed. Additionally we see how the PAS 4 line dives just before 200. This could might be explained by the activation just before perfectly perceiving the stimuli.

what is PAS you ask?!?! - recap from our assignment 2 part 2: 1. No Experience, 2. Weak Glimpse, 3. Almost Clear Experience, 4. Clear Experience, are the ratings of the perceptual awareness scale.

EXERCISE 2 - Do logistic regression to classify

pairs of PAS-ratings

1) Now, we are going to do Logistic Regression with the aim of classifying the PAS-rating given by the subject

i. We'll start with a binary problem - create a new array called `data_1_2` that only contains PAS responses 1 and 2. Similarly, create a `y_1_2` for the target vector

In [ ]:
```
list_y = np.argwhere((vector_y ==1) | (vector_y == 2))
y_1_2 = vector_y[list_y]


#finding pas one and two of the data:
pas_one = [i for i, x in enumerate(vector_y) if x ==1 ]
#this list can now be used index the matching numbers out of the data - we sh
print(data[pas_one,:,:].shape)

## pas two
pas_two = [i for i, x in enumerate(vector_y) if x == 2 ]
print(data[pas_two,:,:].shape)


#taking the target values for each pass and saved them into data_1_2
data_one = data[pas_one,:,:]
data_two = data[pas_two,:,:]
data_1_2 = np.squeeze(np.concatenate((data_one,data_two), axis = 0))


##looking at the length to make sure they are the same
print(data_1_2.shape)
```

```
(99, 102, 251)
(115, 102, 251)
(214, 102, 251)
```

ii. Scikit-learn expects our observations (`data_1_2`) to be in a 2d-array, which has samples (repetitions) on dimension 1 and features (predictor variables) on dimension 2. Our `data_1_2` is a three-dimensional array. Our strategy will be to collapse our two last dimensions (sensors and time) into one dimension, while keeping the first dimension as it is (repetitions). Use `np.reshape` to create a variable `X_1_2` that fulfils these criteria.

In [ ]:
```
X_1_2 = data_1_2.transpose(0,1,2).reshape(-1,data_1_2.shape[0])
print(X_1_2.shape)
#this appears to be right since the shape now is the collapsed data in the 25
```

```
(25602, 214)
```

iii. Import the `StandardScaler` and scale `X_1_2`

In [ ]:
```
#reshapingen
X_1_2_reshaped = X_1_2.reshape(214, 25602)


## splitting the data so we can test how well it classifies the test set:
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_1_2_reshaped, y_1_2, te

y_train = y_train.squeeze()
```

```
y_test = y_test.squeeze()

#making everything on the same scale
sc = StandardScaler()
sc.fit(X_1_2_reshaped)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
```

iv. Do a standard `LogisticRegression` - can be imported from `sklearn.linear_model` - make sure there is no `penalty` applied

In [ ]:
```
logR = LogisticRegression(penalty='none') # no regularisation
logR.fit(X_train_std, y_train)
print(logR.score(X_train_std, y_train))
```

```
1.0
```

v. Use the `score` method of `LogisticRegression` to find out how many labels were classified correctly. Are we overfitting? Besides the score, what would make you suspect that we are overfitting?

In [ ]:
```
l_sc = (logR.score(X_test_std, y_test))
print("this means that", l_sc*100, "% of the classifications are correct. thi
```

```
this means that 55.81395348837209 % of the classifications are correct. this i
s not very much, which might could be explained by overfitting of the test dat
a. The model are therefore not generalisable, and classifies the test data qui
te badly. if the model was not overfitted to the train data, we would expect i
t to be a better classifier - maybe we should penalizzzzze it
```

vi. Now apply the *L1* penalty instead - how many of the coefficients (`.coef_`) are non-zero after this?

In [ ]:
```
log_l1 = LogisticRegression(penalty = 'l1', solver = 'liblinear') # set
log_l1.fit(X_train_std, y_train)
print("the classification accuracy on the train set:",(log_l1.score(X_train_s

print("the classification accuracy on the test set:", (log_l1.score(X_test_st
print("now that we have fitted the new penalized model we can check how many
```

```
the classification accuracy on the train set: 100.0 %
the classification accuracy on the test set: 48.837209302325576 %
now that we have fitted the new penalized model we can check how many of the c
oefficients are not equal to zero, this actually has a lower accuracy of class
ification
```

we here see how it is clearly overfitting, since it in the test scores a 100% classification accuracy and only around due to chance in the test

In [ ]:
```
#finding the zeros
l1_coef = log_l1.coef_
zero = np.count_nonzero(l1_coef == 0)
print("the zeros:", zero) #this is the amounts of 0's
non_zero = np.count_nonzero(l1_coef != 0)
print("the non zeros:", non_zero) #this is the amounts of non 0's
```

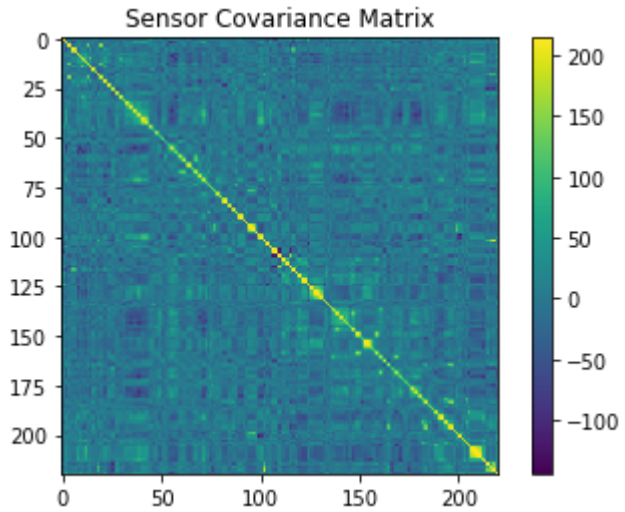
```
the zeros: 25382
the non zeros: 220
```

vii. Create a new reduced X that only includes the non-zero coefficients - show the

covariance of the non-zero features (two covariance matrices can be made; $X_{reduced}X_{reduced}^T$ or $X_{reduced}^TX_{reduced}$ (you choose the right one)) . Plot the covariance of the features using `plt.imshow` . Compared to the plot from 1.1.iii, do we see less covariance?

In [ ]:
```
log1_coef_0 = np.where(l1_coef!= 0)[1]
log1_coef_0.shape

reduced_X=X_1_2_reshaped[:,log1_coef_0]
reduced_X.shape

#standardizing all the data, not only test/train
x_stan = sc.fit_transform(reduced_X)
#transposing the matrix, which must be done to get the covariance matrix of o
cov = x_stan.T @ x_stan
plt.imshow(cov)
plt.title("Sensor Covariance Matrix")
plt.colorbar()

#the shape of features
print(cov.shape)
```

```
(220, 220)
```



Do we see less covariance?

Looking at the scale of colour and the colours of the covariance plot, I have a hard time comparing. The colours should be around 0 on the scale, which it appears to be more than in the first. I'd therefore say the covariance has been reduced, but I find this a hard way to compare it by. So by removing fetaures as here we have reduced the covariance. However the first covariance plot plots the features being only sensors sensors, where as this is features (sensors x timesamples), making them hard to compare.

2) Now, we are going to build better (more predictive) models by using cross-validation as an outcome measure

i. Import `cross_val_score` and `StratifiedKFold` from `sklearn.model_selection`

In [ ]:
```
from sklearn.model_selection import cross_val_score, StratifiedKFold
```

ii. To make sure that our training data sets are not biased to one target (PAS) or the other, create `y_1_2_equal` , which should have an equal number of each target. Create a similar

`X_1_2_equal` . The function `equalize_targets_binary` in the code chunk associated with Exercise 2.2.ii can be used. Remember to scale `X_1_2_equal` !

In [ ]:
```
def equalize_targets_binary(data, y):
    np.random.seed(7)
    targets = np.unique(y) ## find the number of targets
    if len(targets) > 2:
        raise NameError ("can't have more than two targets") #may only be bin
    counts = list()
    indices = list()
    for target in targets:
        counts.append(np.sum(y == target)) ## find the number of each target
        indices.append(np.where(y == target)[0]) ## find their indices
    min_count = np.min(counts)
    # randomly choose trials
    first_choice = np.random.choice(indices[0], size=min_count, replace=False
    second_choice = np.random.choice(indices[1], size=min_count,replace=False
    # create the new data sets
    new_indices = np.concatenate((first_choice, second_choice))
    new_y = y[new_indices]

    new_data = data[new_indices, :, :]
    return new_data, new_y


#creating the new x and y
X_1_2_equal_new_o, y_1_2_equal_new = equalize_targets_binary(data_1_2, y_1_2)

#to be able to standardice, we must reshape it to be 2 dim
X_1_2_equal_new = X_1_2_equal_new_o.reshape(198,-1)


# standardizing
sc = StandardScaler()
X_1_2_equal_std = sc.fit_transform(X_1_2_equal_new)
```

iii. Do cross-validation with 5 stratified folds doing standard `LogisticRegression` (See Exercise 2.1.iv)

In [ ]:
```
k_fold = StratifiedKFold(n_splits = 5, random_state = 2, shuffle = True).spli
```

In [ ]:
```
#loop finding the cross validation scores:
lr = LogisticRegression(penalty = 'l1', solver = 'liblinear')
scores = cross_val_score(lr, X_1_2_equal_std, y_1_2_equal_new[:,0], scoring='
print("the score for each fold:", scores)

##this is across all 5 folds, therefore the mean is taken to summarize the sc
mean_score = np.mean(scores)
print("while the mean score,", mean_score,",is the mean accuracy of the class
```

```
the score for each fold: [0.55       0.6        0.55       0.46153846 0.384615
38]
while the mean score, 0.5092307692307692 ,is the mean accuracy of the classifi
cation of the leaved out k-folds across all 5 folds of the data.
```

iv. Do L2-regularisation with the following `Cs = [1e5, 1e1, 1e-5]` . Use the same kind of cross-validation as in Exercise 2.2.iii. In the best-scoring of these models, how many

more/fewer predictions are correct (on average)?

In [ ]:
```
## C = 1e5
lr_5 = LogisticRegression(C = 1e5, penalty = 'l2', solver = 'liblinear', rand
scores_5 = cross_val_score(lr_5, X_1_2_equal_std, y_1_2_equal_new[:,0], scori
print("Cross Val Score when C = 1e5:",(np.mean(scores_5))*100, "%")

## C = 1e1
lr_1 = LogisticRegression(C = 1e1, penalty = 'l2', solver = 'liblinear', rand
scores_1 = cross_val_score(lr_1, X_1_2_equal_std, y_1_2_equal_new[:,0], scori
print("Cross Val Score  when C = 1e1:",(np.mean(scores_1))*100, "%")

## C = 1e-5
lr__5 = LogisticRegression(C = 1e-5, penalty = 'l2', solver = 'liblinear', ra
scores__5 = cross_val_score(lr__5, X_1_2_equal_std, y_1_2_equal_new[:,0], sco
print("Cross Val Score when C = 1e-5:",(np.mean(scores__5))*100, "%")
```

```
Cross Val Score when C = 1e5: 46.98717948717949 %
Cross Val Score  when C = 1e1: 47.0 %
Cross Val Score when C = 1e-5: 48.05128205128205 %
```

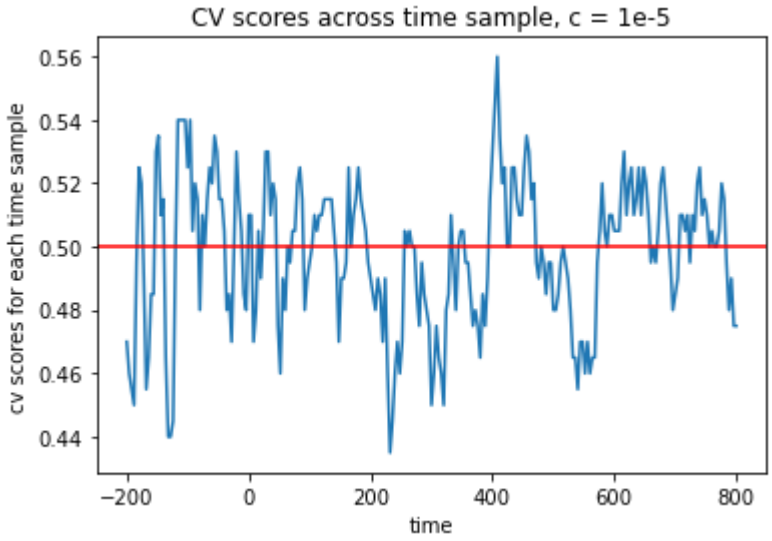
but why is this under due to chance??????? maybe seet, maybe a mistake .... we can conclude though that C = 1e-5 makes the best model, but still it is really not good.

v. Instead of fitting a model on all `n_sensors * n_samples` features, fit a logistic regression (same kind as in Exercise 2.2.iv (use the `C` that resulted in the best prediction)) for **each** time sample and use the same cross-validation as in Exercise 2.2.iii. What are the time points where classification is best? Make a plot with time on the x-axis and classification score on the y-axis with a horizontal line at the chance level (what is the chance level for this analysis?)

In [ ]:
```
time_scores = []
lr_new = LogisticRegression(penalty='l2', C=1e-5)
X_1_2_equal, y_1_2_equal = equalize_targets_binary(data_1_2, y_1_2)

for time_ofs in range(data_1_2.shape[2]):
  X_1_2_equal_time = X_1_2_equal[:, :, time_ofs].squeeze()
  sc.fit(X_1_2_equal_time)
  X_1_2_equal_std = sc.transform(X_1_2_equal_time)
  time_s = cross_val_score(lr_new, X_1_2_equal_std, y_1_2_equal.squeeze(), cv
  score = np.mean(time_s)
  time_scores.append(score)


time_scores = np.array(time_scores) #this is the mean of accuracy across all
```

In [ ]:
```
#finding the position of the max scores
max_score_pos = np.unravel_index(np.argmax(time_scores), time_scores.shape)

print("the time with the highest score is:", time_array[max_score_pos])

#plotting
plt.plot(time_array, time_scores)
plt.axhline(0.5, color = 'r')
plt.xlabel("time")
plt.ylabel("cv scores for each time sample")
plt.title("CV scores across time sample, c = 1e-5")
plt.show()
```

```
the time with the highest score is: 408
```

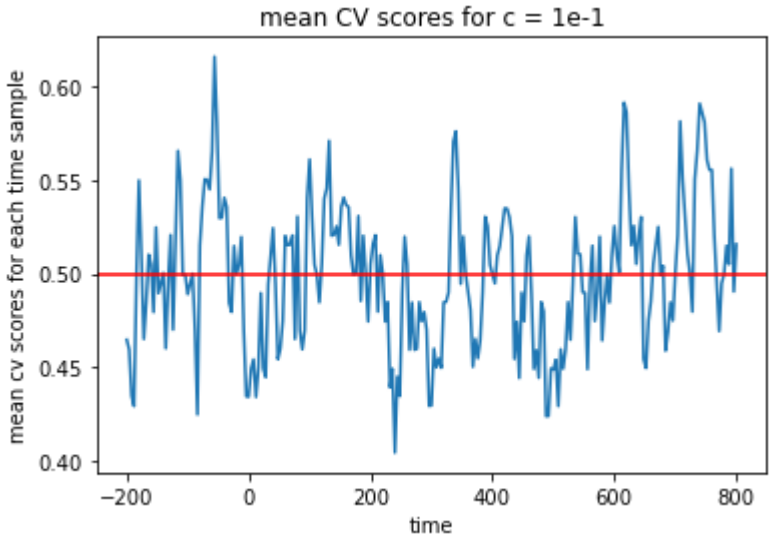


vi. Now do the same, but with L1 regression - set C=1e−1 - what are the time points when classification is best? (make a plot)?

In [ ]:

```
time_scores_c1 = []
lr_new_c1 = LogisticRegression(penalty='l1', solver='liblinear', multi_class=
X_1_2_equal_c1, y_1_2_equal_c1 = equalize_targets_binary(data_1_2, y_1_2)

for time_ofs in range(data_1_2.shape[2]):
  X_1_2_equal_time = X_1_2_equal_c1[:, :, time_ofs].squeeze()
  sc.fit(X_1_2_equal_time)
  X_1_2_equal_std = sc.transform(X_1_2_equal_time)
  time_s = cross_val_score(lr_new_c1, X_1_2_equal_std, y_1_2_equal.squeeze(),
  score = np.mean(time_s)
  time_scores_c1.append(score)


time_scores_c1 = np.array(time_scores_c1) #this is the mean of accuracy acros



#finding the position of the max scores
max_score_pos_c1 = np.unravel_index(np.argmax(time_scores_c1), time_scores_c1
print("the time with the highest score is:", time_array[max_score_pos_c1])



#plotting
plt.plot(time_array, time_scores_c1)
plt.axhline(0.5, color = 'r')
plt.xlabel("time")
plt.ylabel("mean cv scores for each time sample")
plt.title("mean CV scores for c = 1e-1")
plt.show()
```

```
the time with the highest score is: -56
```



vii. Finally, fit the same models as in Exercise 2.2.vi but now for `data_1_4` and `y_1_4`

(create a data set and a target vector that only contains PAS responses 1 and 4). What are the time points when classification is best? Make a plot with time on the x-axis and classification score on the y-axis with a horizontal line at the chance level (what is the chance level for this analysis?)

In [ ]:
```
#making new data:
list1_4 = np.argwhere((vector_y ==1) | (vector_y == 4))
y_1_4 = vector_y[list1_4,]


#taking the target values for each pass and saved them into data_1_4
data_1_4 = data[list1_4,:,:].squeeze()
```

In [ ]:
```
### making the exact same thing but now for the new pas1/4 data.
##### fitting the model with c = 1e-1

lr_new_c4 = LogisticRegression(penalty='l1', solver='liblinear', multi_class=
X_equal_1_4, y_equal_1_4 = equalize_targets_binary(data_1_4, y_1_4)
```

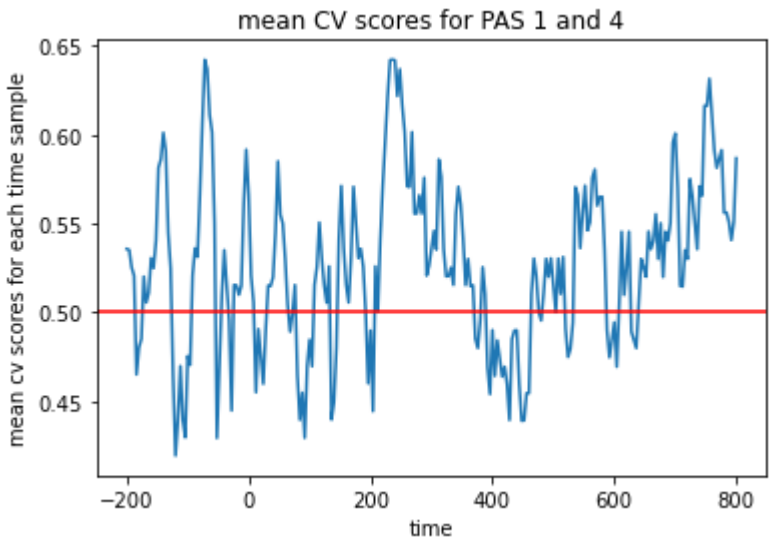
In [ ]:
```
time_scores_4 = []

for time_ofs in range(data_1_4.shape[2]):
  X_1_4_equal_time = X_equal_1_4[:, :, time_ofs].squeeze()
  sc.fit(X_1_4_equal_time)
  X_1_4_equal_std = sc.transform(X_1_4_equal_time)
  time_s = cross_val_score(lr_new_c4, X_1_4_equal_std, y_equal_1_4.squeeze(),
  score = np.mean(time_s)
  time_scores_4.append(score)
```

```
time_scores_4 = np.array(time_scores_4) #this is the mean of accuracy across

#finding the position of the max scores
max_score_pos_c4 = np.unravel_index(np.argmax(time_scores_4), time_scores_4.s
print("the time with the highest score is:", time_array[max_score_pos_c4])

print(time_scores_4.shape)
```

```
the time with the highest score is: 236
(251,)
```

In [ ]:

```
#plotting
plt.plot(time_array, time_scores_4)
plt.axhline(0.5, color = 'r')
plt.xlabel("time")
plt.ylabel("mean cv scores for each time sample")
plt.title("mean CV scores for PAS 1 and 4")
plt.show()
```



3) Is pairwise classification of subjective experience possible?

Any surprises in the classification accuracies, i.e. how does the classification score fore PAS 1 vs 4 compare to the classification score for PAS 1 vs 2?

comment:

```
the more similiar the data are the worse the accuracy in the
classification
the greater the difference the better the accuracy
- but in this case the difference are greater between pass 1 and
2, which is quite unsual.
```

EXERCISE 3 - Do a Support Vector Machine Classification on all four PAS-ratings

1) Do a Support Vector Machine Classification

i. First equalize the number of targets using the function associated with each PAS-rating using the function associated with Exercise 3.1.i

In [ ]:
```
def equalize_targets(data, y):
    np.random.seed(7)
    targets = np.unique(y)
    counts = list()
    indices = list()
    for target in targets:
        counts.append(np.sum(y == target))
        indices.append(np.where(y == target)[0])
    min_count = np.min(counts)
    first_choice = np.random.choice(indices[0], size=min_count, replace=False
    second_choice = np.random.choice(indices[1], size=min_count, replace=Fals
    third_choice = np.random.choice(indices[2], size=min_count, replace=False
    fourth_choice = np.random.choice(indices[3], size=min_count, replace=Fals

    new_indices = np.concatenate((first_choice, second_choice,
                                  third_choice, fourth_choice))
    new_y = y[new_indices]
    new_data = data[new_indices, :, :]

    return new_data, new_y

## making new data using this
data_equal, y_equal = equalize_targets(data, vector_y)

#checking the shape:
print(data_equal.shape)
#reshaping to 2d:
data_equal_2d = data_equal.reshape(data_equal.shape[0],-1) #so the time sampl
print(data_equal_2d.shape)
```

```
(396, 102, 251)
(396, 25602)
```

ii. Run two classifiers, one with a linear kernel and one with a radial basis (other options should be left at their defaults) - the number of features is the number of sensors multiplied the number of samples. Which one is better predicting the category?

- to be able to test which on of the classifiers perform better, we split up the data so we can both train and test the classifiers:

In [ ]:
```
X_tr, X_te, y_tr, y_te = train_test_split(data_equal_2d, y_equal, test_size=0
                                                    random_state=0)
## standardizing to make it comparable
sc.fit(data_equal_2d)
X_train_std = sc.transform(X_tr)
X_test_std = sc.transform(X_te)

#importing SVC class to make the different svm
from sklearn.svm import SVC
svml_linear = SVC(kernel = 'linear')### linear kernel
svml_rbf = SVC(kernel = 'rbf')### radial = rbf


##using the vector functions to find the better prediction:
lin_fit = svml_linear.fit(X_tr, y_tr)
rbf_fit = svml_rbf.fit(X_tr, y_tr)
```

```
## finding the accuracy score:
print("the accuracy score of the linear kernel is",(lin_fit.score(X_te, y_te)
print("the accuracy score of the linear kernel is", (rbf_fit.score(X_te, y_te
```

```
the accuracy score of the linear kernel is 21.25 %
the accuracy score of the linear kernel is 28.749999999999996 %
```

this is for the rbf better than due to chance (now 25% since 4 passes), while the linear is worse - one could imaging a lot of overfitting going on

iii. Run the sample-by-sample analysis (similar to Exercise 2.2.v) with the best kernel (from Exercise 3.1.ii). Make a plot with time on the x-axis and classification score on the y-axis with a horizontal line at the chance level (what is the chance level for this analysis?)

In [ ]:

```
time_scores_kernel = []
## here we use svml_rbf = SVC(kernel = 'rbf')
## our data is equalized, so we just use the X and y directly

#since we are to plot it and visualize the entire sample, we will use all dat
for i in range(data_equal.shape[2]):
    x_ker = data_equal[:,:,i]
    sc.fit(x_ker)
    X_ker_std = sc.transform(x_ker)
    cv_score_kernel = cross_val_score(svml_rbf, X_ker_std, y_equal, cv=5)
    mean_score = np.mean(cv_score_kernel)
    time_scores_kernel.append(mean_score)



time_scores_kernel = np.array(time_scores_kernel) #this is the mean of accura
```

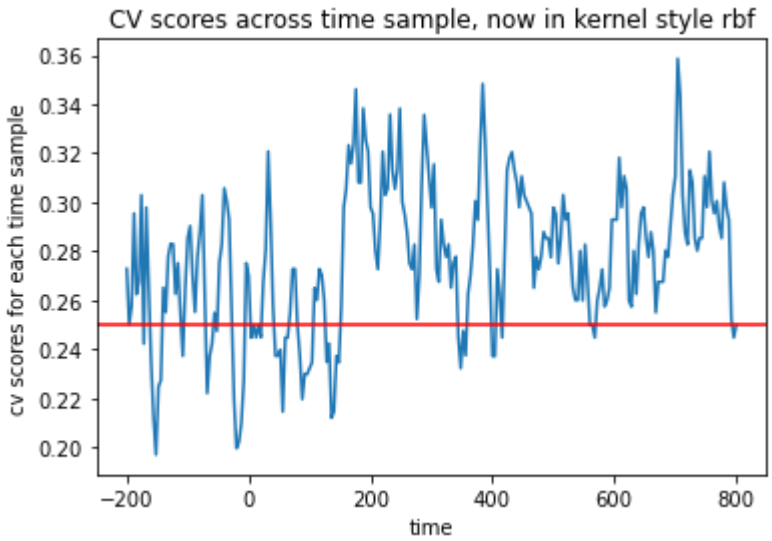
In [ ]:

```
#finding the position of the max scores
####### making the scores into an array first, and not a list:
time_scores_kernel = np.array(time_scores_kernel)

pos_max_kernel = np.unravel_index(np.argmax(time_scores_kernel), time_scores_

print("the time with the highest accuracy score is:", time_array[pos_max_kern
```

```
the time with the highest accuracy score is: 704
```

In [ ]:

```
#plotting
plt.plot(time_array, time_scores_kernel)
plt.axhline(0.25, color = 'r')
plt.xlabel("time")
plt.ylabel("cv scores for each time sample")
plt.title("CV scores across time sample, now in kernel style rbf")
plt.show()
```



iv. Is classification of subjective experience possible at around 200-250 ms?

It is possible to predict classification better then by chance in this timeinterval, since it is above the red line. It is not way above, but still quite alright, leading us to suggest: yes it is possible to classify the PAS rating at aroung 200-250.

2) Finally, split the equalized data set (with all four ratings) into a training part and test part, where the test part if 30 % of the trials. Use `train_test_split` from `sklearn.model_selection`

i. Use the kernel that resulted in the best classification in Exercise 3.1.ii and `fit` the training set and `predict` on the test set. This time your features are the number of sensors multiplied by the number of samples.

In [ ]:
```
### since we are now to compress sensors and time stamps a new data must be m
data_equal_rep = data_equal.reshape(396, -1)

### we actually did this in 3.1 as well, but will do it again, now with a tes
X_tr2, X_te2, y_tr2, y_te2 = train_test_split(data_equal_rep, y_equal, test_s
                                              random_state=0)

## standardizing to make it comparable
sc.fit(data_equal_rep)
X_tr2_std = sc.transform(X_tr2)
X_te2_std = sc.transform(X_te2)
```

In [ ]:
```
# comparing the accuracy of the classifier, when the test set is 30% of the d
##using the vector functions to find the better prediction:
rbf_fit = svml_rbf.fit(X_tr2_std, y_tr2)

#printing the accuracy score
print("the accuracy score of the linear kernel is",(rbf_fit.score(X_te2_std,
```

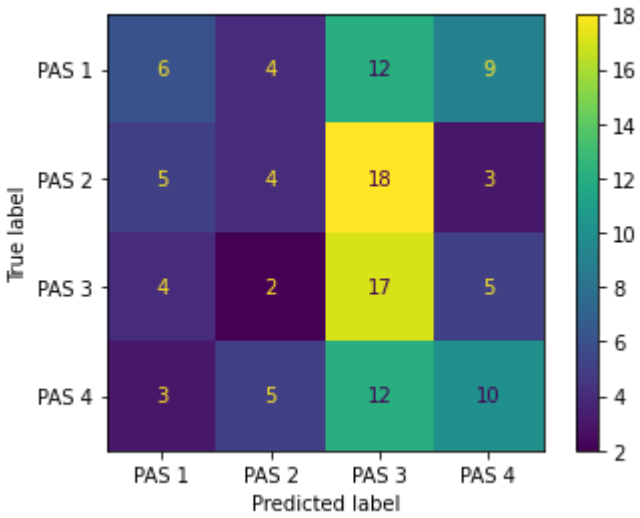
```
the accuracy score of the linear kernel is 31.092436974789916 %, which is bett
er than chance, just like previously
```

ii. Create a *confusion matrix*. It is a 4x4 matrix. The row names and the column names are

the PAS-scores. There will thus be 16 entries. The PAS1xPAS1 entry will be the number of actual PAS1, y_{pas1} that were predicted as PAS1, $\hat{y}_{pas1}$. The PAS1xPAS2 entry will be the number of actual PAS1, y_{pas1} that were predicted as PAS2, $\hat{y}_{pas2}$ and so on for the remaining 14 entries. Plot the matrix

In [ ]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
predictions = rbf_fit.predict(X_te2_std)

cm = confusion_matrix(y_te2, predictions, labels=rbf_fit.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['PAS 1', '
disp.plot()
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x15a3a94c0>
```



iii. Based on the confusion matrix, describe how ratings are misclassified and if that makes sense given that ratings should measure the strength/quality of the subjective experience. Is the classifier biased towards specific ratings?

Looking at the plot from left to right we see that there are truly 31 sample in each pas, but this is not what is predicted by the kernel. The best prediction is in pass 3, where 17 is rightfully classified. However, it predicts 18 samples of pas 2 to be pas 3, which really is off. It appears as if the kernel are biased towards PAS 3, since this is the what the kernel predicts most often.

pas 1 = 6/31 correct predictions pas 2 = 4/31 correct predictions pas 3 = 17/31 correct predictions pas 4 = 10/31 correct predictions

PORTFOLIO 4

Laura Wulff Paaby
202006161
Cognitive Science BA
Aarhus University AU

13/12 - 2021

Portfolio 4, Methods 3, 2021, autumn semester

LAURA W. PAABY

1/12 - 2021

Exercises and objectives

1) Use principal component analysis to improve the classification of subjective experience

2) Use logistic regression with cross-validation to find the optimal number of principal components

REMEMBER: In your report, make sure to include code that can reproduce the answers requested in the exercises below (MAKE A KNITTED VERSION) REMEMBER: This is Assignment 4 and will be part of your final portfolio

EXERCISE 1 - Use principal component analysis to improve the

classification of subjective experience

We will use the same files as we did in Assignment 3 The files megmag_data.npy and pas_vector.npy can be downloaded here (http://laumollerandersen.org/data_methods_3/megmag_data.npy) and here (http://laumollerandersen.org/data_methods_3/pas_vector.npy)

The function equalize_targets is supplied - this time, we will only work with an equalized data set. One motivation for this is that we have a well-defined chance level that we can compare against. Furthermore, wewill look at a single time point to decrease the dimensionality of the problem

In [ ]:

```
###LOADING LIBRARIES
## importing libraries
import statistics
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import RANSACRegressor
from sklearn.model_selection import train_test_split
import scipy as sp
from sklearn.metrics import r2_score
```

```
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.linear_model import ElasticNet
from sklearn.preprocessing import PolynomialFeatures
from sklearn.ensemble import RandomForestRegressor
import os
```

1) Create a covariance matrix, find the eigenvectors and the eigenvalues

i. Load megmag_data.npy and call it data using np.load. You can use join, which can be imported from os.path, to create paths from different string segments

In [ ]:

```
# IMPORTING DATA
import requests
import io

# THE MEG DATA
response = requests.get('http://laumollerandersen.org/data_methods_3/megmag_d
response.raise_for_status()
data = np.load(io.BytesIO(response.content))

#numbers of repitions of visual stimuli: 682
#numbers of sensors (spots in the cap that record magnetic fields): 102
#numbers of time samples: 251



# THE PAS VECTOR
response = requests.get('http://laumollerandersen.org/data_methods_3/pas_vect
response.raise_for_status()
y = np.load(io.BytesIO(response.content))
```

ii. Equalize the number of targets in y and data using equalize_targets

In [ ]:

```
### DEFINING THE EQUIALIZER FUNCTION
def equalize_targets(data, y):
    np.random.seed(7)
    targets = np.unique(y)
    counts = list()
    indices = list()
    for target in targets:
        counts.append(np.sum(y == target))
        indices.append(np.where(y == target)[0])
    min_count = np.min(counts)
    first_choice = np.random.choice(indices[0], size=min_count, replace=False
    second_choice = np.random.choice(indices[1], size=min_count, replace=Fals
    third_choice = np.random.choice(indices[2], size=min_count, replace=False
    fourth_choice = np.random.choice(indices[3], size=min_count, replace=Fals
    new_indices = np.concatenate((first_choice, second_choice,
    third_choice, fourth_choice))
    new_y = y[new_indices]
    new_data = data[new_indices, :, :]
    return new_data, new_y
```

In [ ]:

```
## EQUALIZING THE DATA AND Y:
```

```
data_equal, y_equal = equalize_targets(data, y)
```

iii. Construct times=np.arange(-200, 804, 4) and find the index corresponding to 248 ms - then reduce the dimensionality of data from three to two dimensions by only choosing the time index corresponding to 248 ms (248 ms was where we found the maximal average response in Assignment 3)

In [ ]:
```
#making the time array
times=np.arange(-200, 804, 4)
```

In [ ]:
```
## finding the time
time248 = np.argwhere(times == 248)
print("the time index corresponding to 248 ms:",time248)
```

```
the time index corresponding to 248 ms: [[112]]
```

In [ ]:
```
## squeezing the data to two dim with the time point in the data:
data_time = data_equal[:,:,time248].squeeze()
print("the data is now two dimensional:", data_time.shape)
```

```
the data is now two dimensional: (396, 102)
```

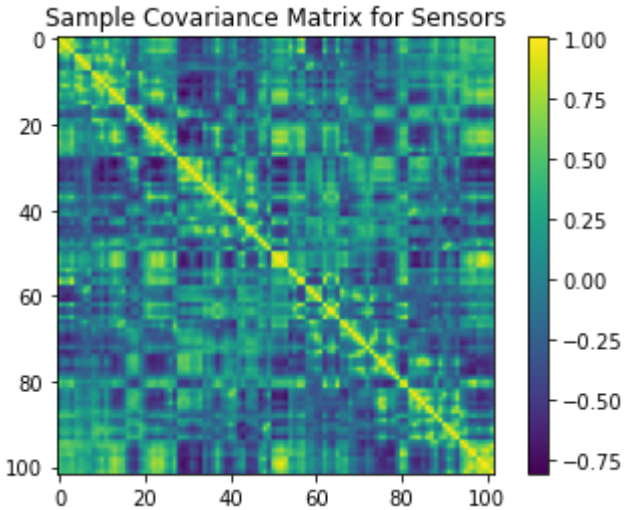
iv. Scale the data using StandardScaler

In [ ]:
```
# standardizing
sc = StandardScaler()
data_time_std = sc.fit_transform(data_time)
```

v. Calculate the sample covariance matrix for the sensors (you can use np.cov) and plot it (either using plt.imshow or sns.heatmap (import seaborn as sns))

In [ ]:
```
cov_mat = np.cov(data_time_std.T)
plt.imshow(cov_mat)
plt.title("Sample Covariance Matrix for Sensors")
plt.colorbar()

#the shape of features
print(cov_mat.shape)
```

```
(102, 102)
```



vi. What does the off-diagonal activation imply about the independence of the signals measured by the 102 sensors?

There appears to be a rather high covariance in the off-diagonal activation, looking at the colors. We see how many matches the colours on the scale around -0.75 and 1, indicating high covariance. The independece signals are therefore quite low, we'd argued

vii. Run np.linalg.matrix_rank on the covariance matrix - what integer value do you get? (we'll use this later)

In [ ]:
```
np_lin_matrix = np.linalg.matrix_rank(cov_mat)
print("the rank of the cov matrix:", np_lin_matrix)
```

```
the rank of the cov matrix: 97
```

viii. Find the eigenvalues and eigenvectors of the covariance matrix using np.linalg.eig - note that some of the numbers returned are complex numbers, consisting of a real and an imaginary part (they have a j next to them). We are going to ignore this by only looking at the real parts of the eigenvectors and -values. Use np.real to retrieve only the real parts

In [ ]:
```
### finding eigen values and vectors
eigen_values_mat, eigen_vectors_mat = np.linalg.eig(cov_mat)
print(eigen_values_mat, eigen_vectors_mat)

eigen_values_mat = np.real(eigen_values_mat)
eigen_vectors_mat = np.real(eigen_vectors_mat)
```

```
[ 2.86956421e+01  1.61532317e+01  1.19667828e+01  7.11946468e+00
  6.53758473e+00  3.56503758e+00  3.26728576e+00  2.74694300e+00
  2.58301935e+00  2.05934337e+00  1.76840286e+00  1.32700594e+00
  1.10080407e+00  1.04399299e+00  9.43037791e-01  8.43761487e-01
  7.56054528e-01  6.50829775e-01  6.06503267e-01  5.36007750e-01
  4.94678836e-01  4.00053880e-01  3.55520231e-01  3.29622537e-01
  3.09444655e-01  2.91850233e-01  2.65190752e-01  2.56099076e-01
  2.46024142e-01  2.35190093e-01  2.11523920e-01  2.15787340e-01
  2.02570026e-01  1.78607257e-01  1.83107840e-01  1.63718456e-01
  1.47542517e-01  1.43183916e-01  1.42175684e-01  1.35217523e-01
  1.30115449e-01  1.24802958e-01  1.22744036e-01  1.13727476e-01
  1.10608291e-01  1.03376043e-01  1.00246339e-01  9.66121945e-02
  9.22382747e-02  8.91661528e-02  8.80352165e-02  8.38123252e-02
  7.95106934e-02  7.56708457e-02  7.50049795e-02  7.17532589e-02
  7.29421553e-02  6.84257809e-02  6.52993774e-02  6.40611140e-02
  1.05260202e-02  5.98903487e-02  1.34204174e-02  1.40479340e-02
  5.71240196e-02  1.56792301e-02  5.63581476e-02  1.61114679e-02
  1.75314752e-02  1.79642111e-02  1.89773602e-02  5.52056614e-02
  5.38232974e-02  5.30717201e-02  5.17885667e-02  2.09186404e-02
  4.90053356e-02  4.77963583e-02  4.67514994e-02  2.21312089e-02
  2.33476326e-02  2.39863924e-02  4.27736047e-02  2.71567213e-02
  3.17721994e-02  3.08786078e-02  3.72739518e-02  3.90017448e-02
  2.84125782e-02  2.47132649e-02  3.36438792e-02  4.07416727e-02
  2.95657187e-02  4.14389338e-02  3.99926065e-02  3.44327716e-02
  2.49751605e-02  4.21331203e-16  1.02517949e-17 -4.82979671e-17
 -2.90168273e-16 -4.52040614e-16] [[ 0.12771427 -0.06908204 -0.08059849 ... -
0.05359027 -0.18106345
   0.0759772 ]
 [ 0.09676535 -0.14564561 -0.07476254 ... -0.02035155 -0.10215784
   0.02548895]
 [ 0.10334695 -0.12256363 -0.12387986 ... -0.02430483 -0.12870615
   0.04402212]
 ...
```

```
[ 0.16293631  0.04934011  0.06496482 ...  0.03952017  0.15032851
  0.15720407]
[ 0.15957395  0.03865037  0.04133159 ...  0.13440067  0.16404444
  0.08148025]
[ 0.14719124 -0.03554908  0.0728808  ...  0.0327546   0.08479141
  0.11853622]]
```

2) Create the weighting matrix W and the projected data, Z

i. We need to sort the eigenvectors and eigenvalues according to the absolute values of the eigenvalues (use np.abs on the eigenvalues).

In [ ]:
```
abs_eigen_val = abs(eigen_values_mat)
```

ii. Then, we will find the correct ordering of the indices and create an array, e.g. sorted_indices that contains these indices. We want to sort the values from highest to lowest. For that, use np.argsort, which will find the indices that correspond to sorting the values from lowest to highest. Subsequently, use np.flip, which will reverse the order of the indices.

In [ ]:
```
sorted_indices = np.flip(np.argsort(abs_eigen_val))
print(sorted_indices)
```

```
[  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  31  30  32  34  33  35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
  54  56  55  57  58  59  61  64  66  71  72  73  74  76  77  78  82  93
  91  94  87  86  95  90  84  85  92  88  83  96  89  81  80  79  75  70
  69  68  67  65  63  62  60 101  97 100  99  98]
```

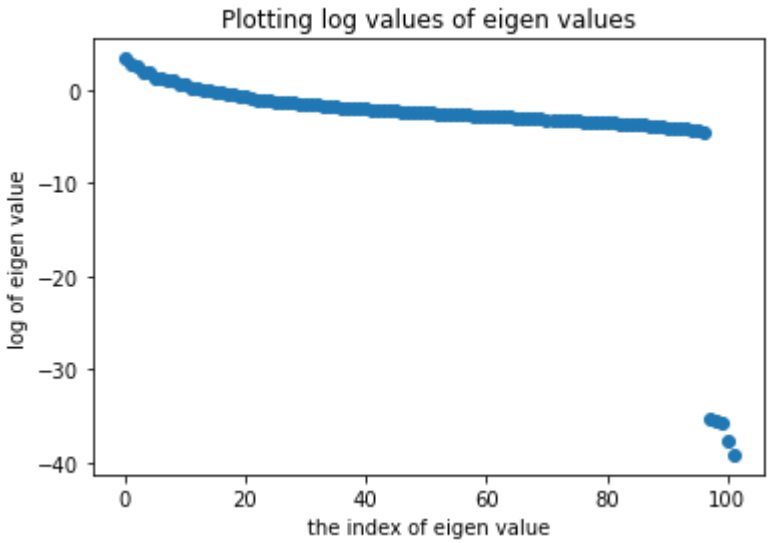
iii. Finally, create arrays of sorted eigenvalues and eigenvectors using the sorted_indices array just created. For the eigenvalues, it should like this eigenvalues = eigenvalues[sorted_indices] and for the eigenvectors: eigenvectors = eigenvectors[:, sorted_indices]

In [ ]:
```
eigen_val_sorted = abs_eigen_val[sorted_indices]
eigen_vec_sorted = eigen_vectors_mat[:,sorted_indices]
```

iv. Plot the log, np.log, of the eigenvalues, plt.plot(np.log(eigenvalues), 'o') - are there some values that stand out from the rest? In fact, 5 (noise) dimensions have already been projected out of the data - how does that relate to the matrix rank (Exercise 1.1.vii)

In [ ]:
```
log_eigen = np.log(eigen_val_sorted)
plt.plot((log_eigen), 'o')
plt.title("Plotting log values of eigen values")
plt.xlabel("the index of eigen value")
plt.ylabel("log of eigen value")
```

Out[ ]:
```
Text(0, 0.5, 'log of eigen value')
```



There appears to be some values around index 100, where the last 5 values stands out (they are below -30 on the log scale, while the rest is above -10), maybe because the variance information these contains are already explained by the 97 other dimensions. The matrix rank it self was 97, meaning that 97 columns are linearly independent, and can explain all the variance, leaving the 5 (from 97 to 102) dimensions with a very low log value.

v. Create the weighting matrix, W (it is the sorted eigenvectors)

In [ ]:
```
W = eigen_vec_sorted
```

vi. Create the projected data, Z , $Z = XW$ - (you can check you did everything right by checking whether the X you get from $X = ZW^T$ is equal to your original X, `np.isclose` may be of help)

In [ ]:
```
X = data_time_std #saving the data in matrix X

#creating the projected data z:
Z = X @ W
```

In [ ]:
```
## checking if its equal to the original:
check_X = Z @ W.T
np.isclose(check_X, X)
```

Out[ ]:
```
array([[ True,  True,  True, ...,  True,  True,  True],
       [ True,  True,  True, ...,  True,  True,  True],
       [ True,  True,  True, ...,  True,  True,  True],
       ...,
       [ True,  True,  True, ...,  True,  True,  True],
       [ True,  True,  True, ...,  True,  True,  True],
       [ True,  True,  True, ...,  True,  True,  True]])
```

vii. Create a new covariance matrix of the principal components (n=102) - plot it! What has happened off-diagonal and why?

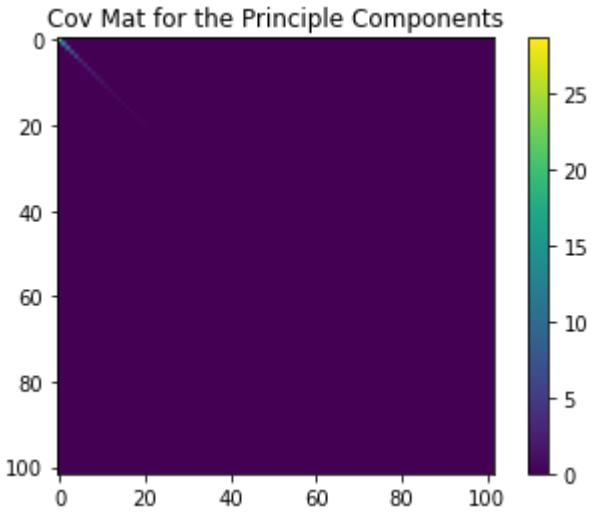
In [ ]:
```
print(W.shape)
```

```
(102, 102)
```

In [ ]:
```
cov_mat_pc = np.cov(Z.T)
```

```
plt.imshow(cov_mat_pc)
plt.colorbar()
plt.title("Cov Mat for the Principle Components")

#the shape of features
print(cov_mat_pc.shape)
```

```
(102, 102)
```



There does not seem to be any covariance overall by inspecting the plot, which makes sense since the principle components are orthogonal. However, the few first principle components seem to covary with themselves, which could be explained by the squared eigenvalues that express variance.

EXERCISE 2 - Use logistic regression with cross-validation to find the optimal number of principal components

1) We are going to run logistic regression with in-sample validation

i. First, run standard logistic regression (no regularization) based on Zn×k and y (the target vector). Fit (.fit) 102 models based on: k = [1, 2, ..., 101, 102] and n = 102. For each fit get the classification accuracy, (.score), when applied to Zn×k and y. This is an in-sample validation. Use the solver newton-cg if the default solver doesn't converge

In [ ]:

```
k_score = []
for i in range(101):
    log_r = LogisticRegression(penalty= 'none', solver= "newton-cg")
    log_r.fit(Z[:,0:i+1], y_equal)
    score = log_r.score(Z[:,0:i+1], y_equal)
    k_score.append(score)
```

In [ ]:

```
k_score = np.array(k_score)
print("the classification accuracy scores:", k_score)
print(k_score.shape)
```

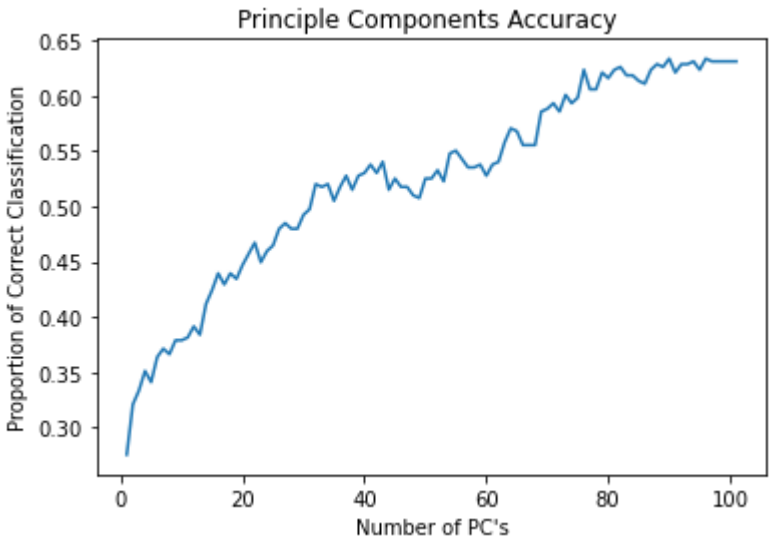
```
the classification accuracy scores: [0.27525253 0.32070707 0.33333333 0.351010
1  0.34090909 0.36363636
 0.37121212 0.36616162 0.37878788 0.37878788 0.38131313 0.39141414
```

```
 0.38383838 0.41161616 0.42424242 0.43939394 0.42929293 0.43939394
 0.43434343 0.4469697  0.45707071 0.46717172 0.44949495 0.45959596
 0.46464646 0.47979798 0.48484848 0.47979798 0.47979798 0.49242424
 0.49747475 0.52020202 0.51767677 0.52020202 0.50505051 0.51767677
 0.52777778 0.51515152 0.52777778 0.53030303 0.53787879 0.53030303
 0.54040404 0.51515152 0.52525253 0.51767677 0.51767677 0.51010101
 0.50757576 0.52525253 0.52525253 0.53282828 0.52272727 0.5479798
 0.55050505 0.54292929 0.53535354 0.53535354 0.53787879 0.52777778
 0.53787879 0.54040404 0.55808081 0.57070707 0.56818182 0.55555556
 0.55555556 0.55555556 0.58585859 0.58838384 0.59343434 0.58585859
 0.6010101  0.59343434 0.59848485 0.62373737 0.60606061 0.60606061
 0.62121212 0.61616162 0.62373737 0.62626263 0.61868687 0.61868687
 0.61363636 0.61111111 0.62373737 0.62878788 0.62626263 0.63383838
 0.62121212 0.62878788 0.62878788 0.63131313 0.62373737 0.63383838
 0.63131313 0.63131313 0.63131313 0.63131313 0.63131313]
(101,)
```

ii. Make a plot with the number of principal components on the *x*-axis and classification accuracy on the *y*-axis - what is the general trend and why is this so?

In [ ]:

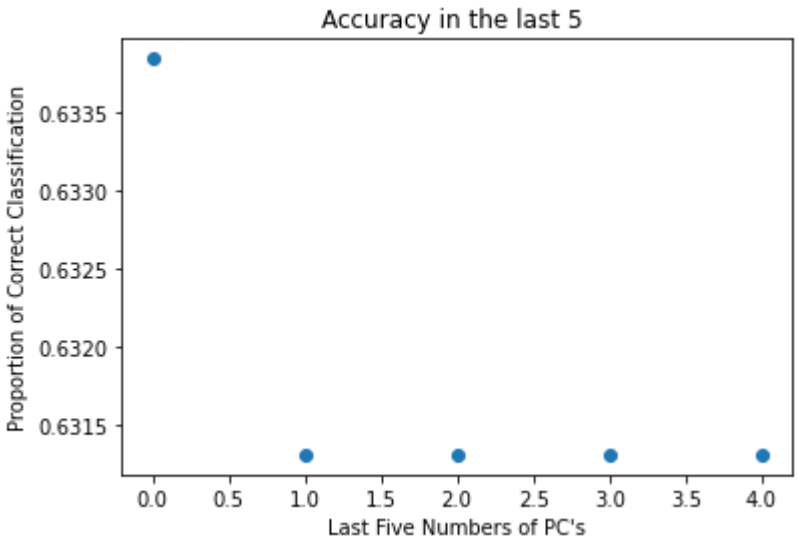
```
## remember the eigenvectors/principle components is saved in W:
#plotting
pc_list = np.arange(1, 102, 1)


plt.plot(pc_list, k_score)
plt.xlabel("Number of PC's")
plt.ylabel("Proportion of Correct Classification")
plt.title("Principle Components Accuracy ")
plt.show()
```



The trend appears to be that the accucary of the classification increases with the value of the principle components. However, adding the five last components have very little effect on the accuracy of classification. This could be due to the fact that the regression gets better to fit the components is added, which eventually leads to overfitting (since we don't cross validate). Around the 100th principle components, we see how the line flattens. This might be the case as we previously noticed how these five dimensions explain very little variance, if any at all. Adding these to the regression does therefore not making the regression better in classifying. This is visualised in the next plot;

iii. In terms of classification accuracy, what is the effect of adding the five last components? Why do you think this is so?

In [ ]:
```
plt.plot(k_score[-6:-1], 'o')
plt.xlabel("Last Five Numbers of PC's")
plt.ylabel("Proportion of Correct Classification")
plt.title("Accuracy in the last 5 ")
plt.show()
```



We see how the accuracy decreases over the last five principal components, which makes sense since these PC's not contains any variance useful to the classification.

2) Now, we are going to use cross-validation - we are using `cross_val_score` and `StratifiedKFold` from `sklearn.model_selection`

i. Define the variable: `cv = StratifiedKFold()` and run `cross_val_score` (remember to set the `cv` argument to your created `cv` variable). Use the same `estimator` in `cross_val_score` as in Exercise 2.1.i. Find the mean score over the 5 folds (the default of `StratifiedKFold`) for each k, $k = [1, 2, \ldots, 101, 102]$

In [ ]:
```
from sklearn.model_selection import cross_val_score, StratifiedKFold
```

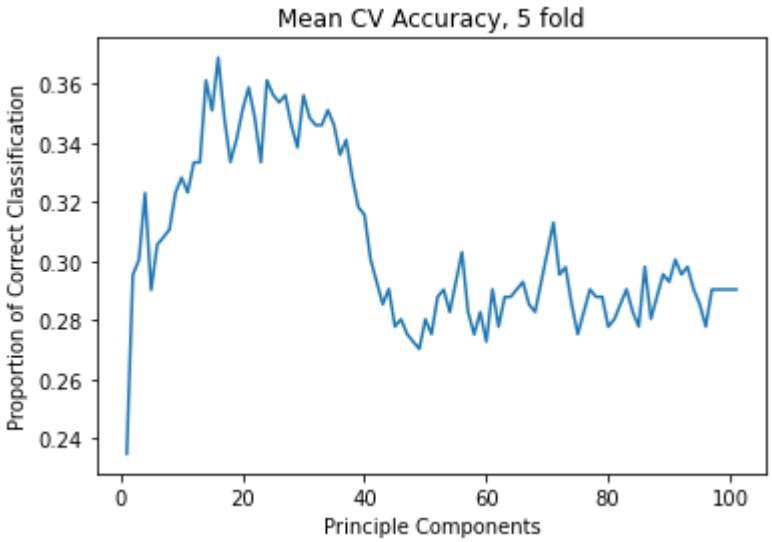
In [ ]:
```
#getting the cross val scores
cv_score = []
cv = StratifiedKFold()
for i in range(101):
  logR = LogisticRegression(penalty='none', solver = 'newton-cg', random_stat
  logR.fit(Z[:,0:i+1],y_equal)
  scores = cross_val_score(logR,Z[:,0:i+1],y_equal, cv = 5 )
  mean_cv_score = np.mean(scores)
  cv_score.append(mean_cv_score)

cv_score_array = np.array(cv_score)
```

ii. Make a plot with the number of principal components on the *x*-axis and classification accuracy on the *y*-axis - how is this plot different from the one in Exercise 2.1.ii?

In [ ]:
```
plt.plot(pc_list, cv_score_array)
```

```
plt.title("Mean CV Accuracy, 5 fold")
plt.xlabel("Principle Components")
plt.ylabel("Proportion of Correct Classification")
```

Out[ ]:
```
Text(0, 0.5, 'Proportion of Correct Classification')
```



In the first plot it appears as if the classification accuracy score gradually becomes better for each principle component at each fit. When we then use cross validation when fitting the logistic regression, the second plot shows a more accurate distribution of the classificatio accuracy score for each principle components. However we can't refuse that both classifiers might overfit, which is also indicated by the decrease of accuracy in the cross validation plot. So around principle component 50 the decrease of accuracy could indicate a overfit. We can also see how the y axe changes radically in the two blots: plot 1 ranges from around 0.27 to 0.63, while in plot 2 it ranges from 0.23 to 0.37.

iii. What is the number of principal components, $k_{max\_accuracy}$, that results in the greatest classification accuracy when cross-validated?

In [ ]:
```
max_score_pc = np.unravel_index(np.argmax(cv_score_array), cv_score_array.sha
print("the number of principle component with the highest classification accu
```

```
the number of principle component with the highest classification accuracy sco
re is: 16 where the value is: 36.88291139240506 %
```

iv. How many percentage points is the classification accuracy increased with relative to the to the full-dimensional, d, dataset

In [ ]:
```
## to work with the full data, we should prepare it. It is already equalized:
print(data_equal.shape)
```

```
(396, 102, 251)
```

In [ ]:
```
#this should then be reshaped, to be worked with:
data_equal_reshape = data_equal.reshape(396, -1)
print(data_equal_reshape.shape)
```

```
(396, 25602)
```

In [ ]:
```
logR_d = LogisticRegression(penalty='none', solver = 'newton-cg', random_stat
logR_d.fit(data_equal_reshape[:,max_score_pc],y_equal)
```

```
scores_d = cross_val_score(logR_d, data_equal_reshape[:,max_score_pc],y_equal
mean_cv_score_d = np.mean(scores_d)

print("the accuracy score of the principle component with the highest previou
```

```
the accuracy score of the principle component with the highest previously (num
ber 16), has in the full data the accuracy score: 24.240506329113927 %
```

In [ ]:

```
#the difference between these two in percent:
difference_percent = ((np.max(cv_score_array)-(mean_cv_score_d))/(mean_cv_sco
print("the increase in percent is:", difference_percent, "%")

#the difference in percent point:
dif_point = ((np.max(cv_score_array))-(mean_cv_score_d))*100
print("the increase in percent point:", dif_point,"points")
```

```
the increase in percent is: 52.15404699738902 %
the increase in percent point: 12.642405063291134 points
```

v. How do the analyses in Exercises 2.1 and 2.2 differ from one another? Make sure to comment on the differences in optimization criteria.

In the first exercise (2.1) logistic regression with in-sample validation is used, leaving both our train and test data unchanged throughout the modelling. This gives a model that only gets really good at predicting on that specific test set, which is what we see in the first plot, and this will then perform worse when predicting unseen new data. The criteria is here the best accuracy score. In comparison, we use cross-validation with 5 folds in exercise 2.2, which means the whole data set is split into train and test sets through 5 iterations and get a performance measure (classification accuracy score) for each fold. The mean performance is taken to evaluate our model, and in this way we will become better at predicting unseen data. The criteria is in this case the cross validation score. I'd argue that a good model is a generalisable model, which has a high classification accuracy on unseen data - which makes the cross validated the better choice in this case. This model is also opposite the logistic regression with no cross validation more proun not to overfit, which is an advantage.

3) We now make the assumption that $k_{max\_accuracy}$ is representative for each time sample (we only tested for 248 ms). We will use the PCA implementation from *scikit-learn*, i.e. import `PCA` from `sklearn.decomposition`.

In [ ]:

```
from sklearn.decomposition import PCA
#fit finding eigenvector values, and then transform
#used like this:

#pca = PCA(n_components=2)
#principalComponents = pca.fit_transform(x)
```

i. For **each** of the 251 time samples, use the same estimator and cross-validation as in Exercises 2.1.i and 2.2.i. Run two analyses - one where you reduce the dimensionality to $k_{max\_accuracy}$ dimensions using `PCA` and one where you use the full data. Remember to

scale the data (for now, ignore if you get some convergence warnings - you can try to increase the number of iterations, but this is not obligatory)

In [ ]:

```
#RUNNING THE ANALYSIS PCA STYLE:
cv_score_pca = []
cv = StratifiedKFold()
n_dim = pc_list[max_score_pc]

for i in range(251):
  logR_pca = LogisticRegression(penalty='none', solver = 'newton-cg', random_
  data_prep = data_equal[:,:,i] #making sure we take the data for all 251 tim
  data_prep = sc.fit_transform(data_prep) #standardizing the data
  pca = PCA(n_components=n_dim) #making the principle components on my best P
  X_pca = pca.fit_transform(data_prep) #fitting it - this is now the X matrix
  logR_pca.fit(X_pca, y_equal) #fitting the new X to the log regression
  scores_pca = cross_val_score(logR_pca, X_pca ,y_equal, cv = 5 ) #finding th
  mean_cv_score_pca = np.mean(scores_pca)
  cv_score_pca.append(mean_cv_score_pca)

cv_score_array_pca = np.array(cv_score_pca)
```

In [ ]:

```
#max_pca = np.unravel_index(np.argmax(cv_score_array_pca), cv_score_array_pca
print("the accuracy score over time for the PCA:", cv_score_array_pca)
```

```
the accuracy score over time for the PCA: [0.27025316 0.28287975 0.24990506 0.
27259494 0.26       0.26756329
 0.25759494 0.2525     0.24993671 0.26746835 0.23734177 0.2171519
 0.23231013 0.26756329 0.28037975 0.27022152 0.29044304 0.26012658
 0.26278481 0.24243671 0.24487342 0.27512658 0.2625     0.23984177
 0.2525     0.26265823 0.29291139 0.29560127 0.29044304 0.30822785
 0.29813291 0.29541139 0.2625     0.23731013 0.23743671 0.22724684
 0.23227848 0.26756329 0.27015823 0.27025316 0.30063291 0.30822785
 0.29047468 0.25006329 0.25268987 0.24762658 0.25262658 0.245
 0.23996835 0.24487342 0.25737342 0.25987342 0.27746835 0.26753165
 0.27768987 0.26512658 0.2828481  0.26765823 0.24493671 0.29291139
 0.2928481  0.23974684 0.20693038 0.22455696 0.26234177 0.24981013
 0.24246835 0.27515823 0.29272152 0.30044304 0.25762658 0.245
 0.25256329 0.245      0.25246835 0.24737342 0.23990506 0.26256329
 0.24987342 0.23471519 0.27512658 0.25240506 0.29281646 0.30297468
 0.30047468 0.28781646 0.28525316 0.28522152 0.26493671 0.28781646
 0.28547468 0.2981962  0.29800633 0.32072785 0.31325949 0.30060127
 0.30050633 0.29547468 0.29794304 0.31306962 0.31797468 0.28515823
 0.2978481  0.29294304 0.28044304 0.30560127 0.33085443 0.33591772
 0.33091772 0.33585443 0.34848101 0.36370253 0.36882911 0.33844937
 0.34107595 0.33088608 0.29303797 0.24993671 0.27259494 0.26506329
 0.24237342 0.27265823 0.26753165 0.28512658 0.26240506 0.25987342
 0.27262658 0.29028481 0.28528481 0.31306962 0.30041139 0.2928481
 0.29022152 0.31041139 0.29037975 0.26509494 0.28015823 0.27765823
 0.27509494 0.27259494 0.28009494 0.25487342 0.29544304 0.31313291
 0.30291139 0.29537975 0.29287975 0.28528481 0.26246835 0.25490506
 0.24981013 0.27003165 0.27259494 0.25993671 0.29278481 0.28787975
 0.32579114 0.29553797 0.28541139 0.26506329 0.26253165 0.28528481
 0.30294304 0.29791139 0.27765823 0.27759494 0.25737342 0.27253165
 0.27509494 0.26509494 0.26246835 0.25493671 0.27515823 0.30291139
 0.30275316 0.28262658 0.29791139 0.28281646 0.29310127 0.30056962
 0.29797468 0.28025316 0.28525316 0.29810127 0.27012658 0.27765823
 0.27015823 0.26503165 0.27012658 0.27265823 0.25996835 0.27265823
 0.245      0.22731013 0.24746835 0.28028481 0.27787975 0.28541139
 0.27528481 0.28028481 0.28797468 0.30294304 0.29794304 0.30056962
 0.29813291 0.29800633 0.27272152 0.27272152 0.29294304 0.32060127
 0.31810127 0.30300633 0.2903481  0.28031646 0.27775316 0.24987342
```

```
0.24487342 0.24243671 0.25259494 0.25506329 0.24243671 0.27012658
0.26759494 0.29037975 0.29291139 0.29050633 0.30572785 0.31832278
0.32588608 0.30053797 0.32316456 0.30556962 0.2803481  0.28787975
0.29044304 0.31060127 0.3256962  0.30550633 0.31060127 0.31803797
0.33063291 0.32547468 0.31037975 0.29528481 0.29281646 0.27259494
0.26006329 0.24993671 0.27772152 0.29531646 0.29525316]
```

In [ ]:

```
import warnings
warnings.filterwarnings(action='once')

# THE SECOND ANALYSIS NO PCA, ALL DATA:
all_data_score = []
cv = StratifiedKFold()
for i in range(251):
  logR_= LogisticRegression(penalty='none', solver = 'newton-cg', random_stat
  data_prep_ = data_equal[:,:,i] #making sure we take the data for all 251 ti
  data_prep_ = sc.fit_transform(data_prep_) #standardizing the data
  logR_.fit(data_prep_, y_equal) #fitting the new X to the log regression
  scores_ = cross_val_score(logR_, data_prep_ ,y_equal, cv = cv ) #finding th
  all_scores = np.mean(scores_)
  all_data_score.append(all_scores)
```

```
/Users/laura/opt/miniconda3/envs/methods3/lib/python3.9/site-packages/sklearn/
utils/optimize.py:210: ConvergenceWarning: newton-cg failed to converge. Incre
ase the number of iterations.
  warnings.warn(
/Users/laura/opt/miniconda3/envs/methods3/lib/python3.9/site-packages/sklearn/
utils/optimize.py:210: ConvergenceWarning: newton-cg failed to converge. Incre
ase the number of iterations.
  warnings.warn(
/Users/laura/opt/miniconda3/envs/methods3/lib/python3.9/site-packages/sklearn/
utils/optimize.py:210: ConvergenceWarning: newton-cg failed to converge. Incre
ase the number of iterations.
  warnings.warn(
/Users/laura/opt/miniconda3/envs/methods3/lib/python3.9/site-packages/sklearn/
utils/optimize.py:210: ConvergenceWarning: newton-cg failed to converge. Incre
ase the number of iterations.
  warnings.warn(
/Users/laura/opt/miniconda3/envs/methods3/lib/python3.9/site-packages/sklearn/
utils/optimize.py:210: ConvergenceWarning: newton-cg failed to converge. Incre
ase the number of iterations.
  warnings.warn(
/Users/laura/opt/miniconda3/envs/methods3/lib/python3.9/site-packages/sklearn/
utils/optimize.py:210: ConvergenceWarning: newton-cg failed to converge. Incre
ase the number of iterations.
  warnings.warn(
```

In [ ]:

```
print("the accuracy score over time for the data without pca:", all_data_scor
```

```
the accuracy score over time for the data without pca: [0.23977848101265825,
0.2575, 0.22974683544303795, 0.2195253164556962, 0.22202531645569618, 0.224556
96202531644, 0.22199367088607597, 0.2473101265822785, 0.24740506329113923, 0.2
3221518987341772, 0.2727215189873418, 0.24996835443037976, 0.2499050632911392,
0.23218354430379745, 0.23981012658227846, 0.2652531645569621, 0.28791139240506
325, 0.2880379746835443, 0.22993670886075948, 0.22224683544303797, 0.209556962
02531646, 0.23740506329113925, 0.22740506329113921, 0.21458860759493673, 0.249
74683544303797, 0.23218354430379745, 0.2725316455696203, 0.24740506329113923,
0.22465189873417724, 0.2273417721518987, 0.24737341772151894, 0.24237341772151
896, 0.27262658227848097, 0.3029746835443038, 0.2929113924050633, 0.2524050632
911392, 0.20455696202531642, 0.23474683544303793, 0.272879746835443, 0.2829113
924050633, 0.2954430379746835, 0.298006329113924, 0.27281645569620255, 0.25765
82278481013, 0.24246835443037976, 0.2802848101265823, 0.26772151898734176, 0.2
```

```
7018987341772155, 0.25999999999999995, 0.26231012658227854, 0.257246835443038,
0.2598101265822785, 0.2651582278481013, 0.31835443037974687, 0.270189873417721
55, 0.24, 0.2601582278481013, 0.2905379746835443, 0.28284810126582277, 0.28018
98734177215, 0.290379746835443, 0.2828481012658277, 0.27022151898734176, 0.28
53481012658228, 0.2775316455696203, 0.30291139240506326, 0.3435443037974684,
0.31585443037974686, 0.3031645569620253, 0.29060126582278484, 0.28268987341772
156, 0.28268987341772156, 0.2752215189873418, 0.24240506329113926, 0.219683544
3037975, 0.25240506329113926, 0.2650632911392405, 0.27281645569620255, 0.25284
81012658228, 0.2830696202531645, 0.2751898734177215, 0.24496835443037973, 0.26
272151898734175, 0.27778481012658224, 0.2372784810126582, 0.28025316455696203,
0.28287974683544304, 0.255126582278481, 0.22727848101265824, 0.244905063291139
23, 0.2827848101265823, 0.2854113924050633, 0.2650316455696203, 0.247341772151
89873, 0.2702848101265823, 0.2222151898734177, 0.24496835443037973, 0.26003164
55696202, 0.26006329113924054, 0.2827215189873418, 0.2600316455696202, 0.26253
16455696202, 0.2524367088607595, 0.25506329113924053, 0.2224367088607595, 0.22
224683544303797, 0.25756329113924054, 0.30550632911392406, 0.2877848101265822
5, 0.2574367088607595, 0.27259493670886076, 0.2802215189873417, 0.290316455696
2025, 0.3081012658227848, 0.28034810126582277, 0.26765822784810134, 0.25237341
7721519, 0.2399050632911392, 0.232373417721519, 0.2626582278481013, 0.26768987
341772155, 0.29287974683544304, 0.2802848101265823, 0.2804430379746835, 0.2854
746835443038, 0.2676898734177215, 0.2626898734177215, 0.2677848101265823, 0.27
772151898734176, 0.2649683544303797, 0.2651582278481012, 0.25775316455696207,
0.2625632911392405, 0.275126582278481, 0.27275316455696197, 0.2624683544303797
5, 0.26746835443037975, 0.2599367088607595, 0.23981012658227846, 0.21971518987
34177, 0.2195886075949367, 0.21202531645569622, 0.2222151898734177, 0.22727848
101265824, 0.23734177215189875, 0.2273417721518987, 0.2601582278481013, 0.3004
7468354430384, 0.260126582278481, 0.2424050632911392, 0.25259493670886074, 0.2
575, 0.24224683544303796, 0.2952848101265823, 0.29287974683544304, 0.270253164
556962, 0.2877848101265823, 0.2853164556962025, 0.22727848101265824, 0.2802848
101265823, 0.2979430379746835, 0.3233227848101266, 0.2829746835443038, 0.28034
810126582277, 0.29787974683544305, 0.267626582278481, 0.2774683544303797, 0.25
477848101265826, 0.28522151898734177, 0.30287974683544305, 0.2903797468354430
4, 0.2500316455696202, 0.2675632911392405, 0.2624367088607594, 0.2599050632911
392, 0.2651582278481013, 0.2700316455696202, 0.2954746835443038, 0.30303797468
35443, 0.26778481012658223, 0.2651898734177215, 0.28018987341772156, 0.2979113
9240506326, 0.24237341772151896, 0.21468354430379746, 0.23227848101265822, 0.2
39873417721519, 0.25509493670886074, 0.23224683544303798, 0.2651582278481012,
0.2575632911392405, 0.239873417721519, 0.265, 0.2574050632911392, 0.2095569620
2531646, 0.23987341772151902, 0.23481012658227848, 0.2246835443037975, 0.21977
848101265823, 0.23468354430379748, 0.23974683544303796, 0.27531645569620256,
0.2752215189873418, 0.2904113924050633, 0.3081012658227848, 0.282689873417721
5, 0.2601582278481013, 0.2475, 0.2877215189873418, 0.2876898734177215, 0.27259
493670886076, 0.2902848101265823, 0.29287974683544304, 0.275253164556962, 0.23
99367088607595, 0.22712025316455695, 0.24981012658227847, 0.2650316455696202,
0.2550632911392405, 0.27281645569620255, 0.2751898734177215, 0.257310126582278
5, 0.24227848101265823, 0.255, 0.249873417721519, 0.285253164556962, 0.2802531
6455696203, 0.27512658227848097, 0.28525316455696204, 0.275253164556962, 0.275
1898734177215, 0.3031645569620253, 0.28537974683544304, 0.2778797468354431, 0.
2830379746835443, 0.275253164556962, 0.29287974683544304, 0.30306962025316453,
0.27537974683544303, 0.28300632911392404, 0.28800632911392404, 0.2904430379746
835, 0.2778164556962025, 0.2700316455696202, 0.2776898734177215, 0.23734177215
189875, 0.24243670886075952, 0.24493670886075952, 0.24000000000000005, 0.23234
177215189872, 0.2576898734177215]
```

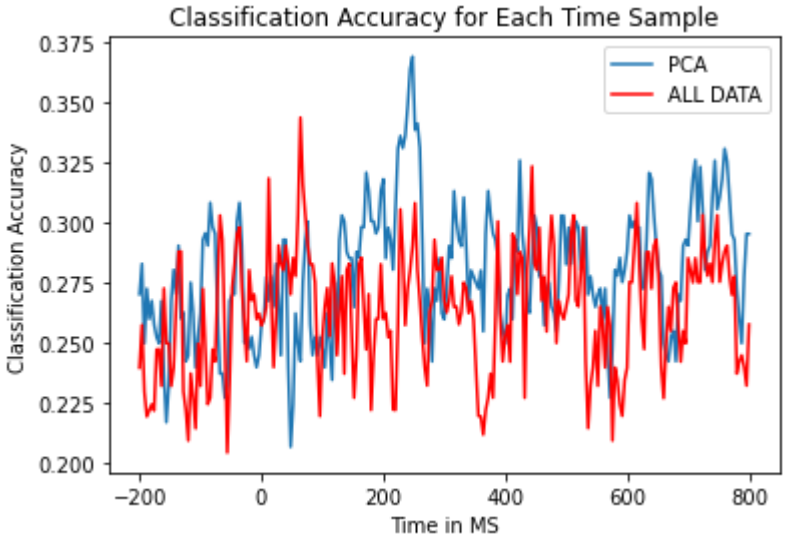
ii. Plot the classification accuracies for each time sample for the analysis with PCA and for the one without in the same plot. Have time (ms) on the *x*-axis and classification accuracy on the *y*-axis

In [ ]:

```
plt.plot(times, cv_score_array_pca)
plt.plot(times, all_data_score, color = 'red')
plt.title("Classification Accuracy for Each Time Sample")
plt.xlabel("Time in MS")
```

```
plt.ylabel("Classification Accuracy")
plt.legend(['PCA', 'ALL DATA'])
```

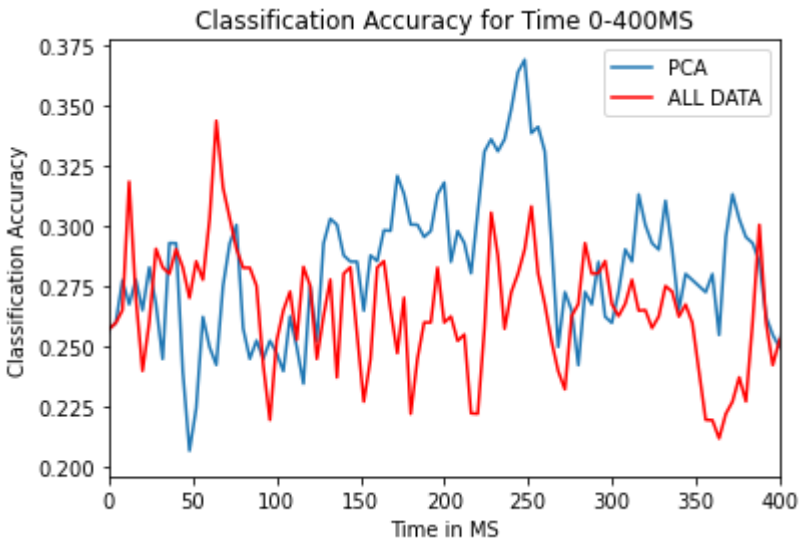
Out[ ]:
```
<matplotlib.legend.Legend at 0x159abcf70>
```



iii. Describe the differences between the two analyses - focus on the time interval between 0 ms and 400 ms - describe in your own words why the logistic regression performs better on the PCA-reduced dataset around the peak magnetic activity

In [ ]:
```
plt.plot(times, cv_score_array_pca)
plt.plot(times, all_data_score, color = 'red')
plt.xlim((0, 400))
plt.title("Classification Accuracy for Time 0-400MS")
plt.xlabel("Time in MS")
plt.ylabel("Classification Accuracy")
plt.legend(['PCA', 'ALL DATA'])
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x159b8cc40>
```



The plot shows the average of all the accuracy scores of all times classified by either PCA or the regression on all data. We see a peak in the accuracy of the PCA around 250 and a general tendency of it performing better when classifying. This makes sense explained by the sensors must have a very high activety in this. The higher the sensor activation the higher the covariance, which leads to the better performance of the PCA. When the

covariane isn't that high, the difference of the classifiers aren't that big, because the new fitted PCA's doesn't provide any new information.

The difference between the two analysis

The main difference between the two analysis is the number of principle components used when classifying. Inspecting the third plot the first 16 components results in the greatest classification accuracy, this being 36%. The total number of principle components (102) performs a bit worse, its maximum being 34%. Even though the 16 principle components seems to performe better, there is only a small differnence. The performance of the model is better on this PCA reduced dataset (16 principle components), which can be explained by the reduction of covariated variables, that did not provide information to the classification. Hence type of analysis ensures that the data does not contain any dirsturbing noise (remaining from the princple components), but only the information that contributes to the classification.