Assignment 2, Part 1, Methods 3, 2021, autumn semester

Rikke Uldbæk

29/9 2021

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/ 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

1) Put the data from all subjects into a single data frame

```
# read the data
data <- read_bulk(directory = "experiment_2/", fun = read_csv)

attach(data)
```

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.

ii. describe what the following variables in the data frame contain, *trial.type*, *pas*, *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 variables in your description). For each of them, indicate and argue for what `class` they should be classified into, e.g. *factor*, *numeric* etc. For each of them, indicate and argue for what `class` they should be classified into, e.g. *factor*, *numeric* etc.

trial.type This variable contains two different values; "experiment" and "staircase" in which each represent two different trials. When eyeballing this variable, it is evident what we have 69% "experiment" values and 31% "staircase" variables. This variable is currently a character variable and will be classified into a factor for analytic purposes.

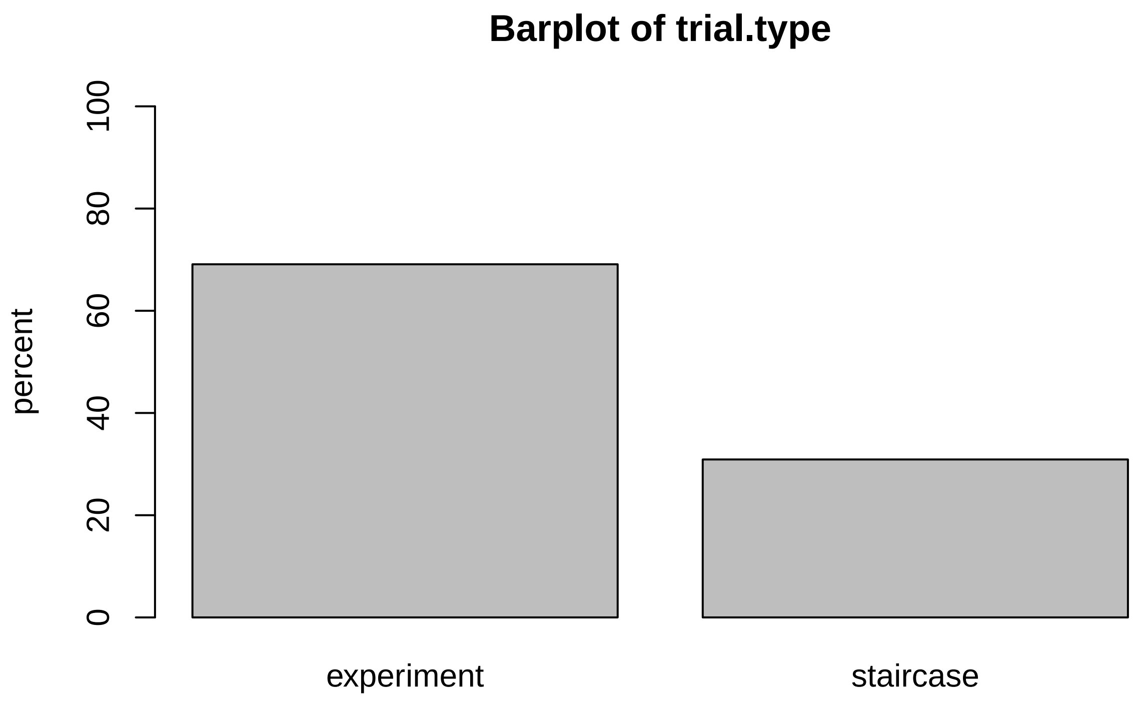
```
class(trial.type)
```

```
## [1] "character"
```

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

# plot
n <- nrow(data)  # Number of rows in total
(percent_trial.type <- table(data$trial.type)/n * 100)  #generating percentages of trial.type
```

```
##
## experiment  staircase
##   69.09713   30.90287
```

```
barplot(percent_trial.type, ylim = c(0, 100), ylab = "percent",
    main = "Barplot of trial.type")
```



pas The "pas" variable (an abbreviation of Perceptual Awareness Scale), contain 4 different values ranging from 1:4. 1 indicates No Experience, 2 indicates Weak Glimpse (WG), 3 indicates Almost Clear Experience

(ACE), and 4 indicates Clear Experience (CE). A histogram of the frequency, shows that the most frequent value is 1 and the least frequent value is 3. This variable is currently a numeric variable and will be classified into a factor for analytic purposes.

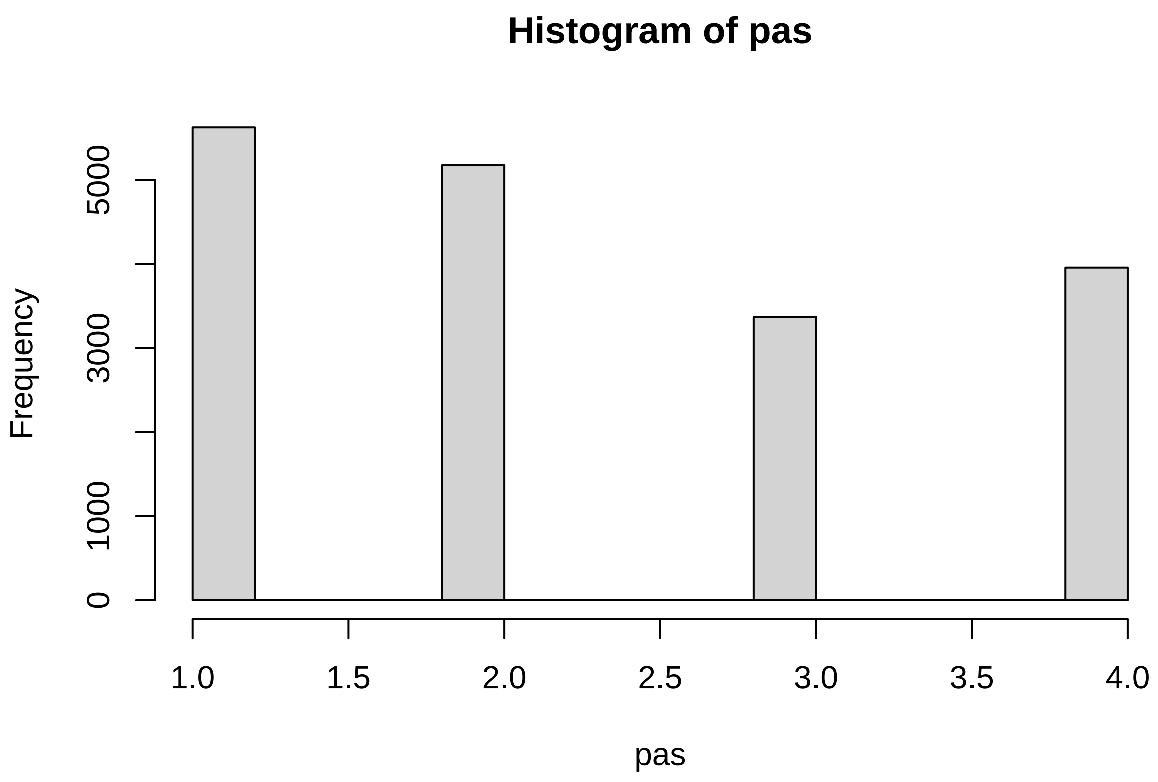
```
class(pas)
```

```
## [1] "numeric"
```

```
summary(pas)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##   1.000   1.000   2.000   2.312   3.000   4.000
```

```
data$pas <- as.factor(data$pas)
hist(pas)
```



trial The "trial" variable contains values ranging from 1:73. This variables indicate that there are 73 different trials in the data set. This variable is currently a numeric variable and will be classified into a factor for analytic purposes.

```
class(trial)
```

```
## [1] "numeric"
```

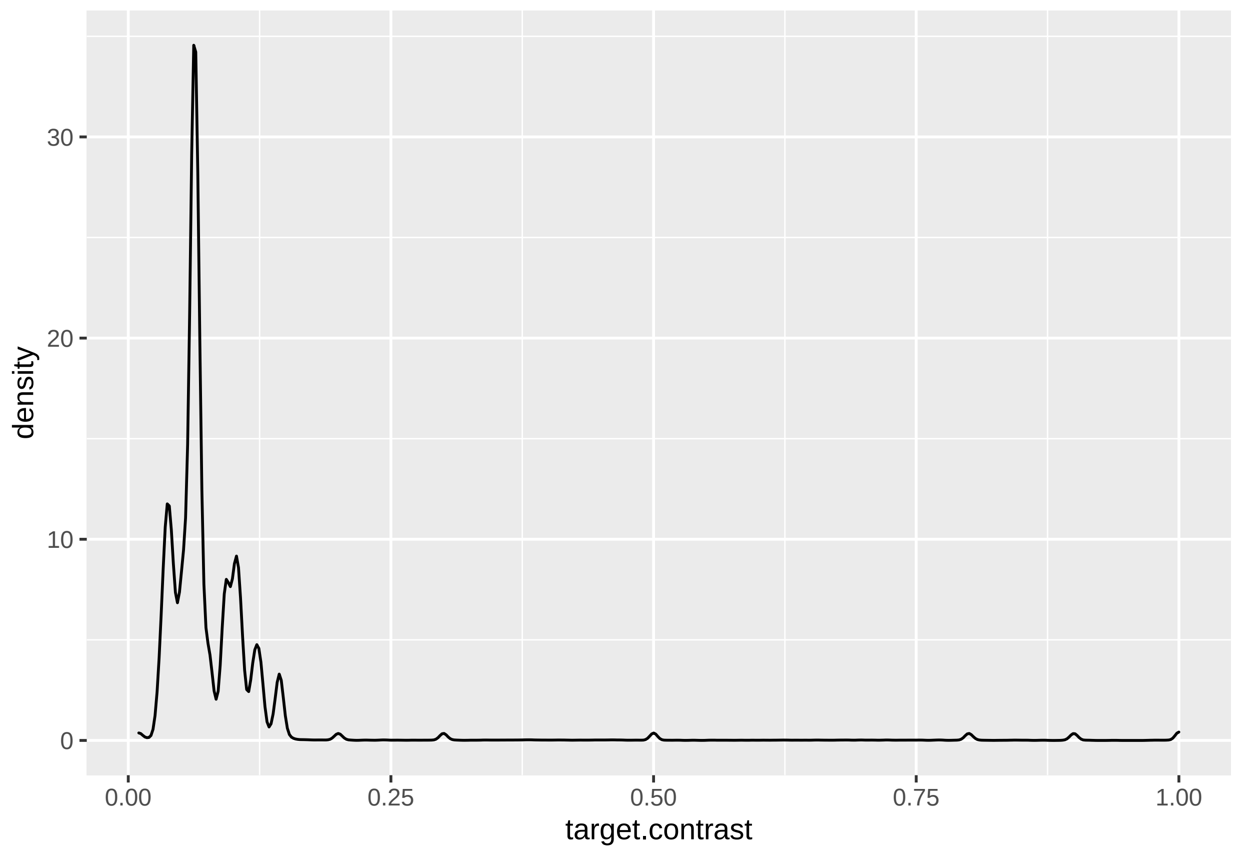
```
data$trial <- as.factor(data$trial)
```

target.contrast The "target.contrast" variable is a numeric variable ranging from 0-1, representing the grey-scale proportion of the target digit.

```
class(target.contrast)
```

```
## [1] "numeric"
```

```
ggplot(data, aes(target.contrast))+geom_density()
```



```
summary(target.contrast)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.01000 0.05683 0.06329 0.08771 0.09392 1.00000
```

cue The "cue" variable is a numeric variable, representing a number code for cue.

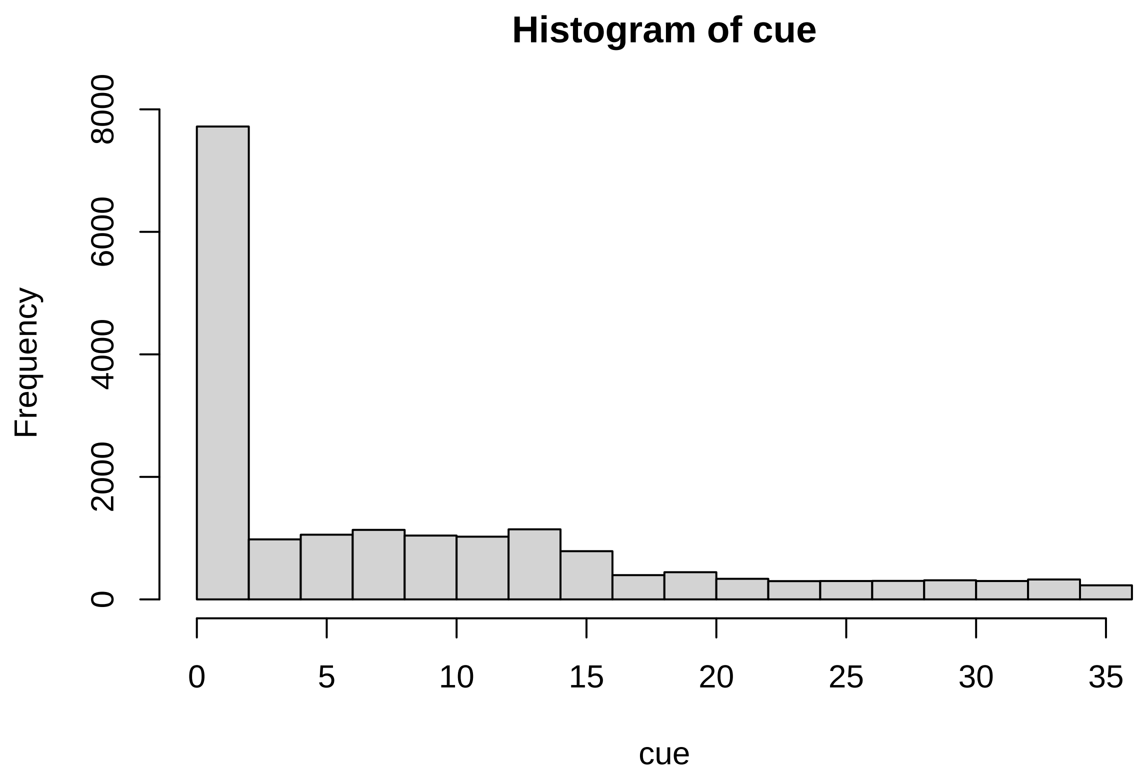
```
class(cue)
```

```
## [1] "numeric"
```

```
summary(cue)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   0.000   5.000   8.378  14.000  35.000
```

```
hist(cue)
```



task The "task" variable consists of 3 different values: "singles", "pairs" and "quadruplets". These 3 different values each represent the number of digits that would be presented to the participant. An example of "singles" being 2:3, an example of "pairs" being 12:44, and an example of quadruplets being 5432:2345.This variable is currently a character variable and will be classified into a factor for analytic purposes.

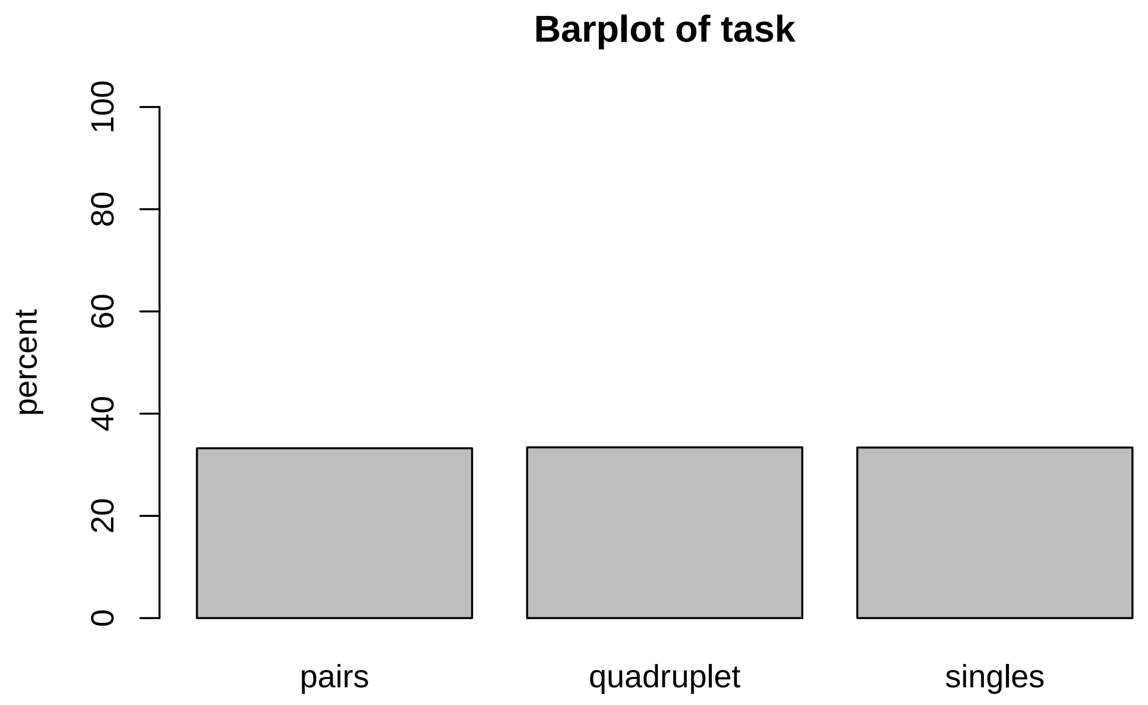
```
class(task)
```

```
## [1] "character"
```

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

# plot
n <- nrow(data)  # Number of rows in total
(percent_task <- table(data$task)/n * 100)  #generating percentages of trial.type
```

```
##
##      pairs quadruplet    singles
##   33.22486   33.40687   33.36826
```

```
barplot(percent_task, ylim = c(0, 100), ylab = "percent", main = "Barplot of task")
```



target.type The "target.type" variable represents if the digit (target) was either an even or odd number. This variable is currently a character variable and will be classified into a factor for analytic purposes.

```
class(target.type)
```

```
## [1] "character"
```

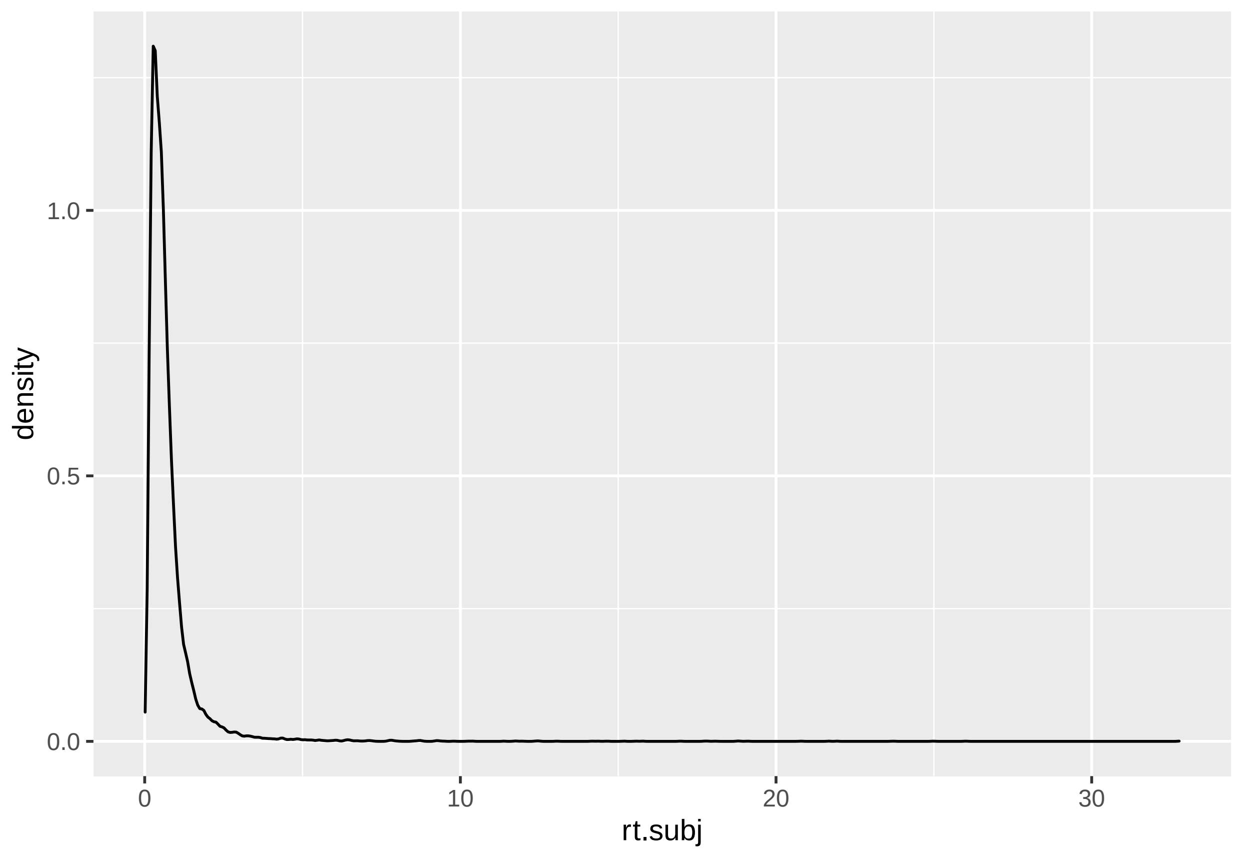
```
data$target.type <- as.factor(data$target.type)
```

rt.subj The "rt.subj" variable represents the reaction time (seconds) on the PAS response. The variable is numeric and highly right skewed, wwith a minimum value of 0.01369, a maximum value of 32.77161, and a mean of 0.74128.

```
class(rt.subj)
```

```
## [1] "numeric"
```

```
ggplot(data, aes(rt.subj)) + geom_density()
```



```
summary(rt.subj)
```

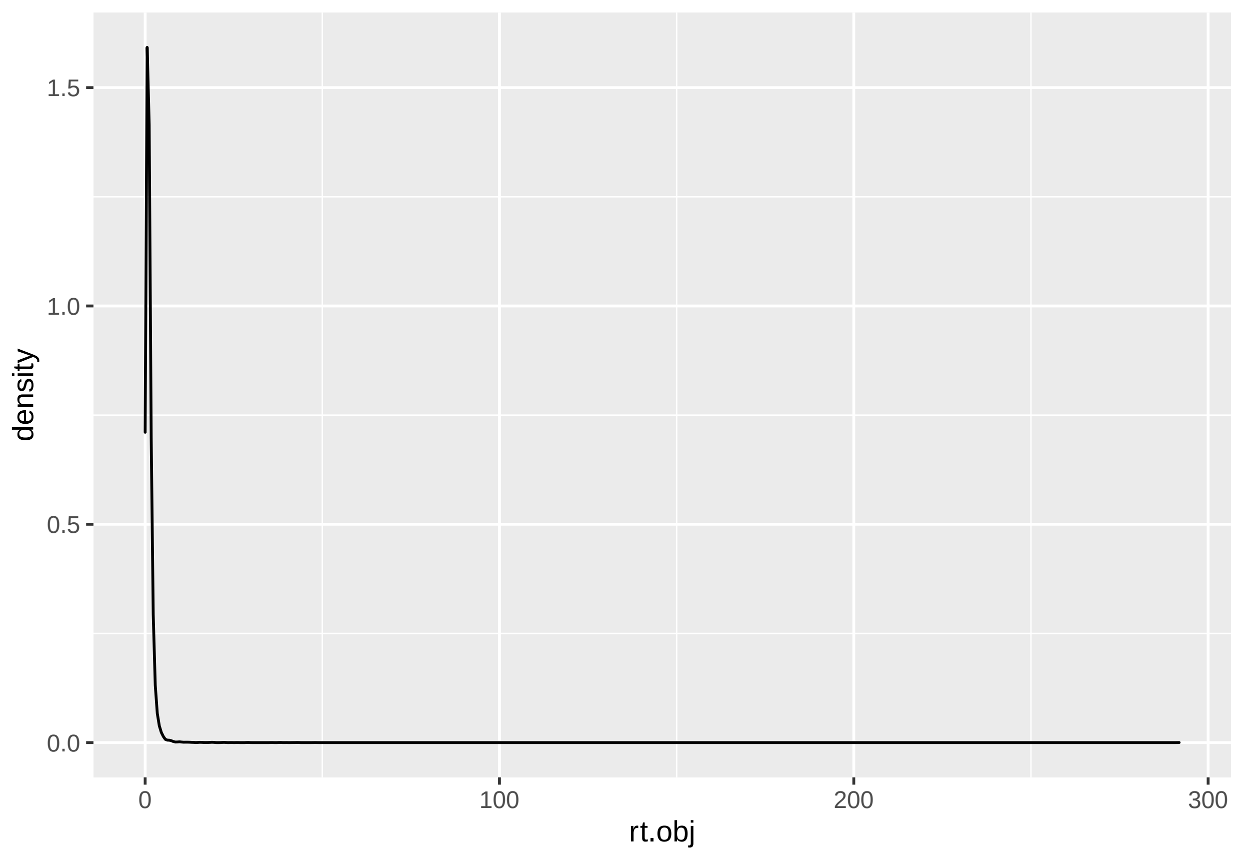
```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##  0.01369  0.31844  0.52811  0.74128  0.82874 32.77161
```

rt.obj The variable "rt.obj" represents the reaction time (seconds) on the target digit. This variable is numeric, and highly right skewed, with a minimum value of 0.00004, a maximum value of 291.83119 , and a mean of 1.16186.

```
class(rt.obj)
```

```
## [1] "numeric"
```

```
ggplot(data, aes(rt.obj)) + geom_density()
```



```
summary(rt.obj)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
##   0.00004   0.53662   0.88568   1.16186   1.37276 291.83119
```

obj.resp The character variable "obj.resp" represents the key actually pressed e for even and o for odd.This variable is currently a character variable and will be classified into a factor for analytic purposes.

```
class(obj.resp)
```

```
## [1] "character"
```

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

subject The subject variable represents each participant's individual ID, which makes it easier for us to distinguish between the participants and account for possible individual differences. This variable is a character string containing 29 different subject ID's, and each subject ID has 506 rows (observations). This variable is currently a character variable and will be classified into a factor for analytic purposes.

```
class(subject)
```

```
## [1] "character"
```

```
length(unique(subject))  #amount of different participants
```

```
## [1] 29
```

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

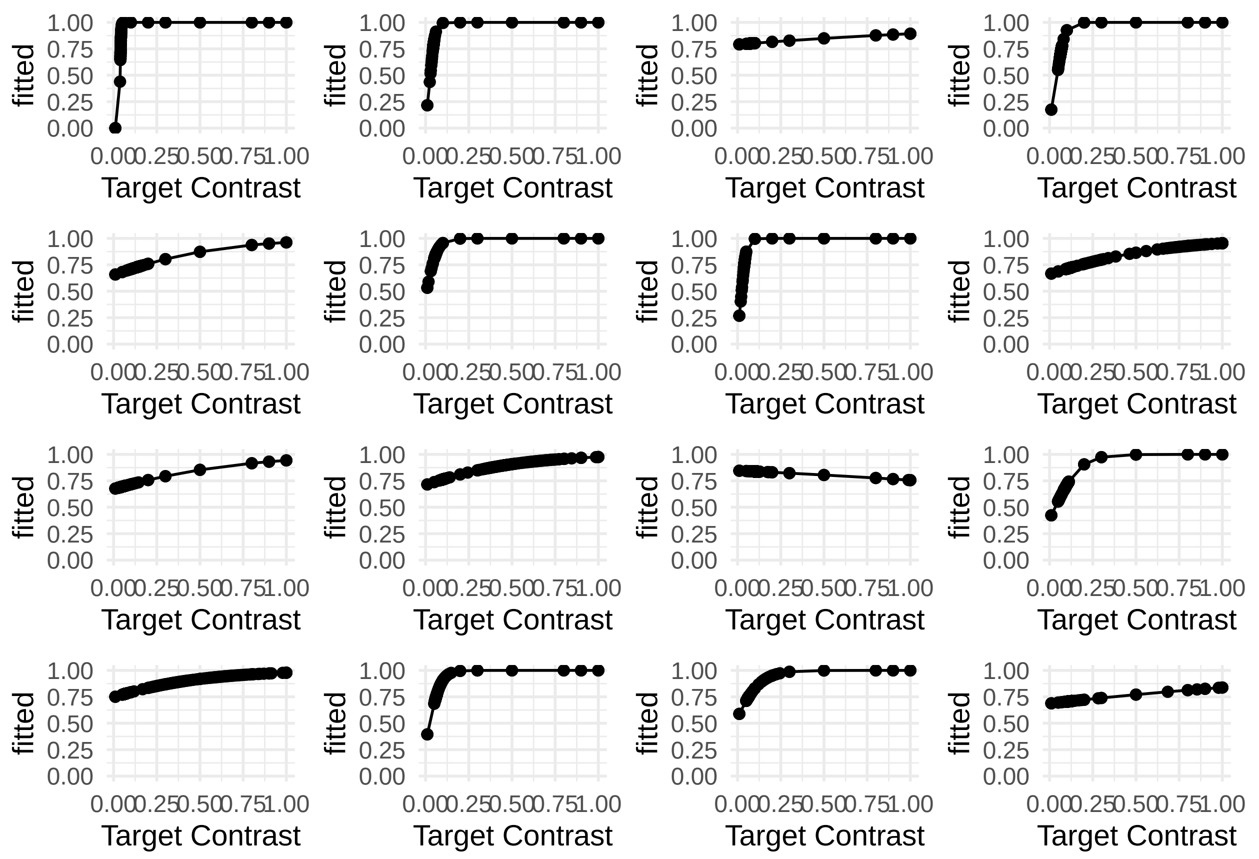
correct Add "correct" variable

The "correct" variable is a logical binary variable containing 1's and 0's. 1 demonstrates when the subject indicated a correct answer (i.e. target_type = "even" and obj.resp = "e", and target_type = "odd" and obj.resp = "o"), and 0 demonstrates when the subject indicated an incorrect answer.

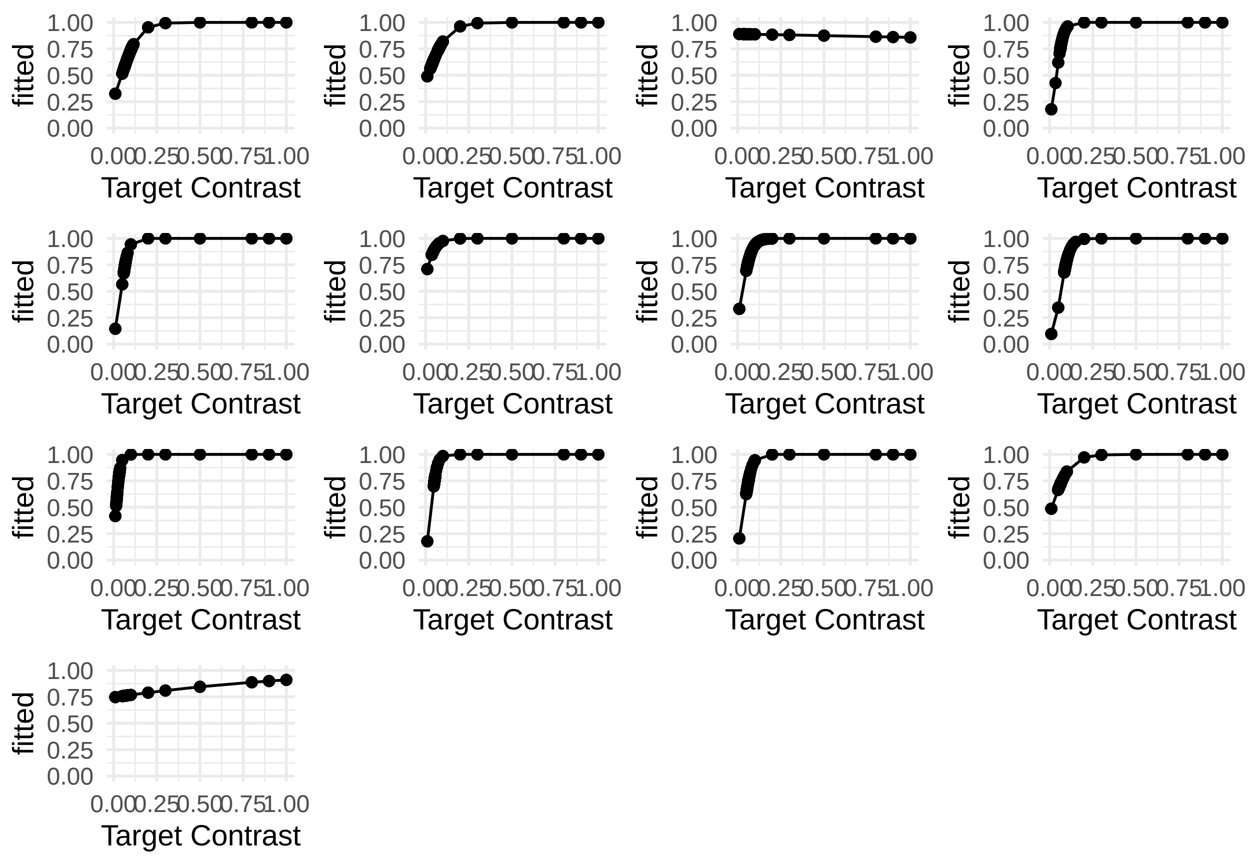
```
# add a variable and call it 'correct'
data$correct <- (ifelse(obj.resp == "o" & target.type == "odd",
    "1", ifelse(obj.resp == "e" & target.type == "even", "1",
        0)))
# making it as a factor
data$correct <- as.factor(data$correct)
```

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 enough data to plot the logistic functions? I'm not sure whether you mean complete pooling or no pooling, so I only did no pooling. The fits looks quite poor, because the fitted values does not resemble the sigmoid function. In order to get better fits (looking more sigmoid-like) we might need more data.

```
## no pooling, glm for each subject

dfStair = data %>%
    subset(trial.type == "staircase")
dfStair$subject = gsub("(?<![0-9])0+", "", dfStair$subject, perl = TRUE)
dfStair$subject = as.integer(dfStair$subject)

nopoolfun <- function(i) {
    dat <- dfStair[which(dfStair$subject == i), ]
    model <- glm(correct ~ target.contrast, family = "binomial",
        data = dat)
    fitted <- model$fitted.values
    plot_dat <- data.frame(cbind(fitted, target.contrast = dat$target.contrast))
    plot <- ggplot(plot_dat, aes(x = target.contrast, y = fitted)) +
        geom_point() + geom_line(aes(x = target.contrast, y = fitted)) +
        xlab("Target Contrast") + ylim(c(0, 1)) + theme_minimal()
    return(plot)
}

library(gridExtra)

subjects <- c(1:16)
plots <- lapply(subjects, FUN = nopoolfun)
do.call(grid.arrange, plots)
```

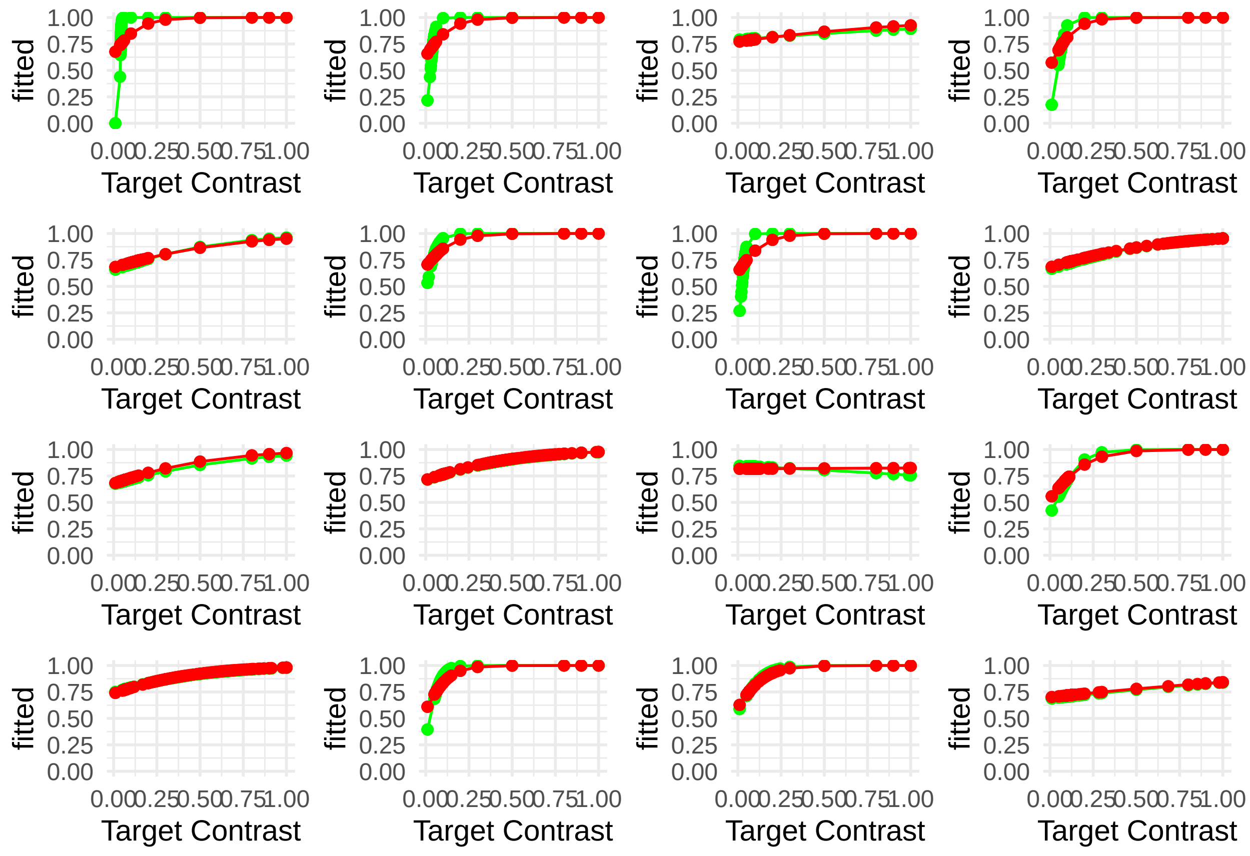


```
subjects <- c(17:29)
plots <- lapply(subjects, FUN = nopoolfun)
do.call(grid.arrange, plots)
```

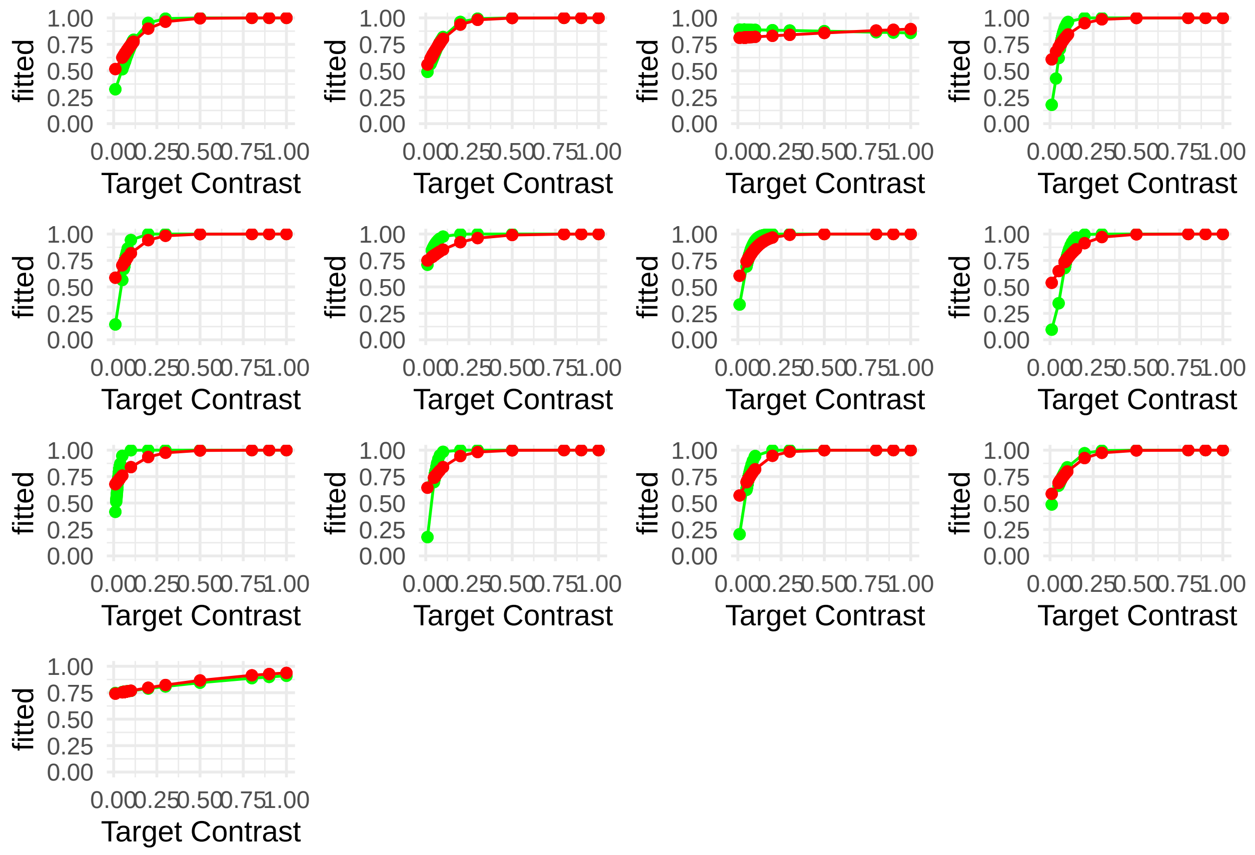


```
# partial pool
partialPoolModel = glmer(correct ~ target.contrast + (target.contrast |
    subject), family = "binomial", dfStair)
dfStair$yEstPartialPooling = fitted(partialPoolModel)
poolNoPoolFun <- function(i) {
    dat <- dfStair[which(dfStair$subject == i), ]
    model <- glm(correct ~ target.contrast, family = "binomial",
        data = dat)
    fitted <- model$fitted.values
    plot_dat <- data.frame(cbind(fitted, target.contrast = dat$target.contrast,
        yEstPartialPooling = dat$yEstPartialPooling))
    plot <- ggplot(plot_dat) + geom_point(aes(x = target.contrast,
        y = fitted), color = "green") + geom_line(aes(x = target.contrast,
        y = fitted), color = "green") + geom_point(aes(x = target.contrast,
        y = yEstPartialPooling), color = "red") + geom_line(aes(x = target.contrast,
        y = yEstPartialPooling), color = "red") + xlab("Target Contrast") +
        ylim(c(0, 1)) + theme_minimal()
    return(plot)
}
subjects <- c(1:16)
plots <- lapply(subjects, FUN = poolNoPoolFun)
do.call(grid.arrange, plots)
```

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*



```r
subjects <- c(17:29)
plots <- lapply(subjects, FUN = poolNoPoolFun)
do.call(grid.arrange, plots)
```



v. in your own words, describe how the partial pooling model allows for a better fit for each subject. In the plot I've compared both partial pooling and no pooling, where the preditcted probabilities are plotted as the following: red is partial pooling and green is no pooling. Compared to no pooling, partial pooling provides better insight about the population (general tendencies), while allowing the model to take individual differences into account (individual baselines), this is visually evident in the plots, where the red graphs (partial pooling) does not vary much in between subject as the green graphs (no pooling). With partial pooling we can both make a great fit for each subject and make it generalizable (in contrast to no pooling).

Exercise 2

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

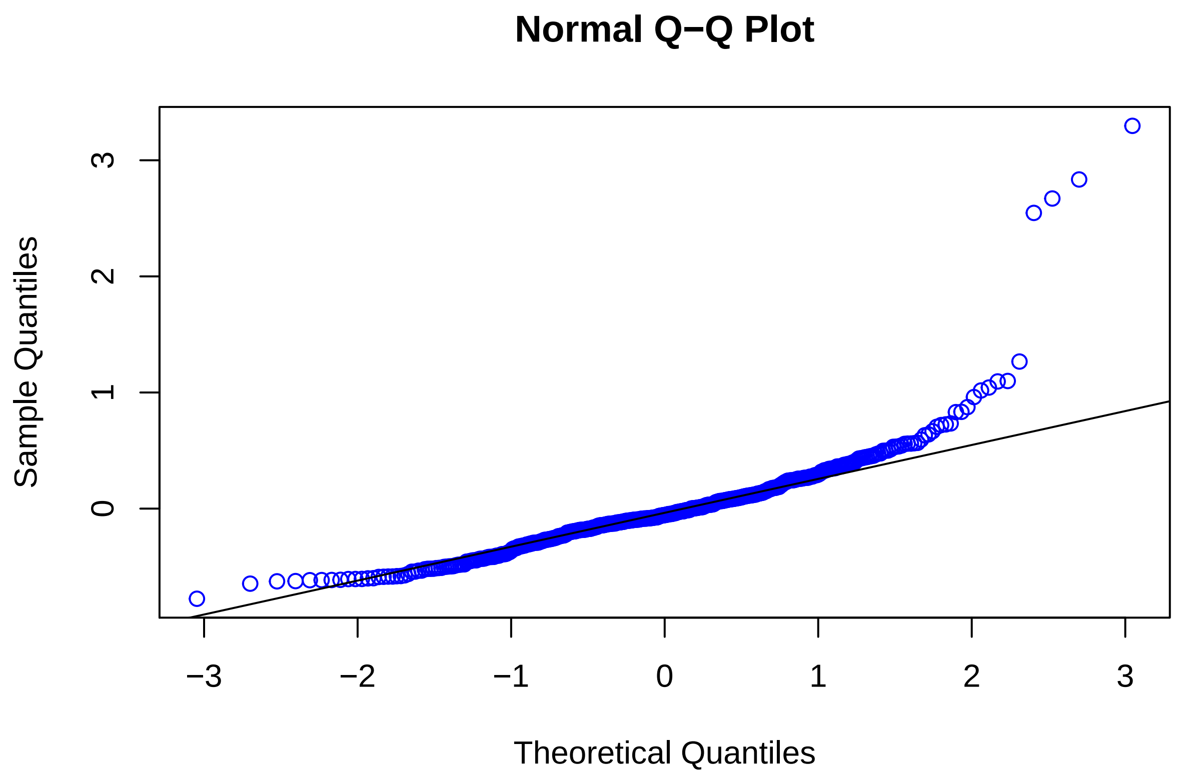
1) Pick four subjects and plot their Quantile-Quantile (Q-Q) plots for the residuals of their objective response times (*rt.obj*) based on a model where only intercept is modelled

i. comment on these

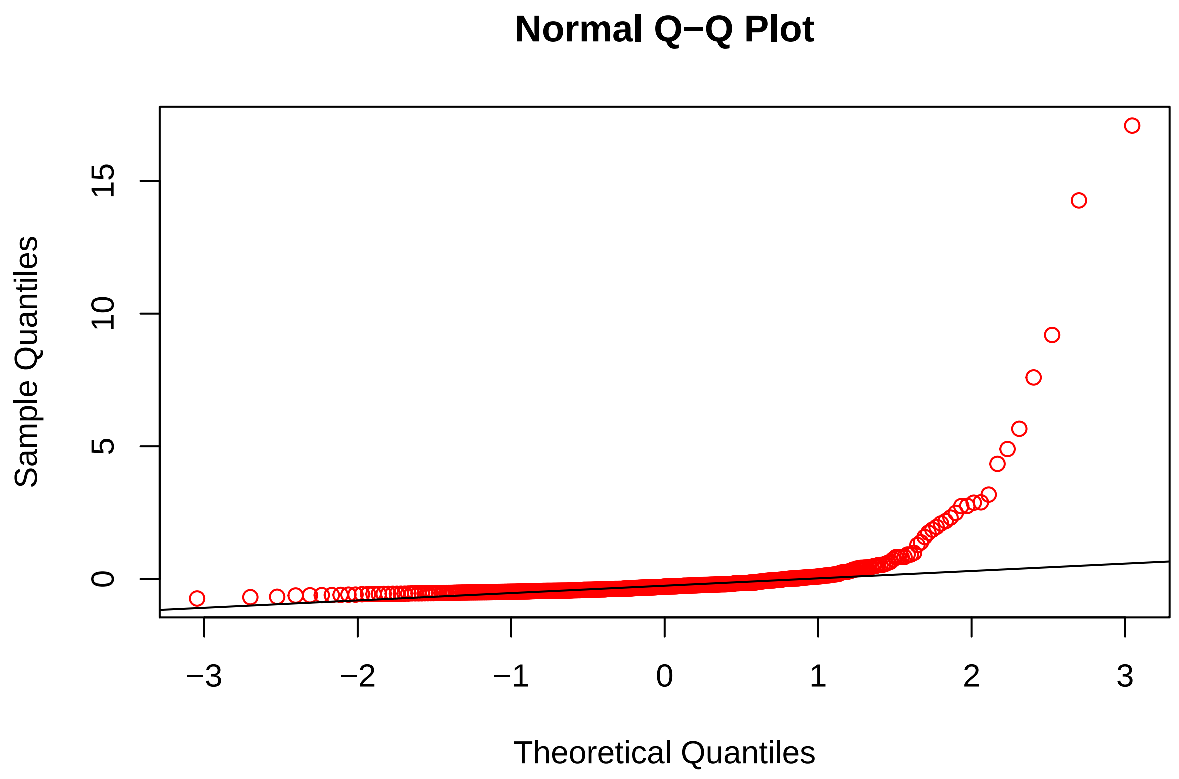
ii. does a log-transformation of the response time data improve the Q-Q-plots? Before log transformation of the 4 subjects objective response time based on the model with only an intercept, the residuals indicate a right skewed tail, as they deviate from the line, i.e deviate from normality.

After log transformation of the 4 subjects objective response time based on the model with only an intercept, the residuals indicate both a left skewed tail and a tiny right skewed tail. The log transformation of the objective response time data (variable= “rt.obj”) improves the Q-Q-plot, as the dots generally falls better on the line , i.e the residuals look more normally distributed.

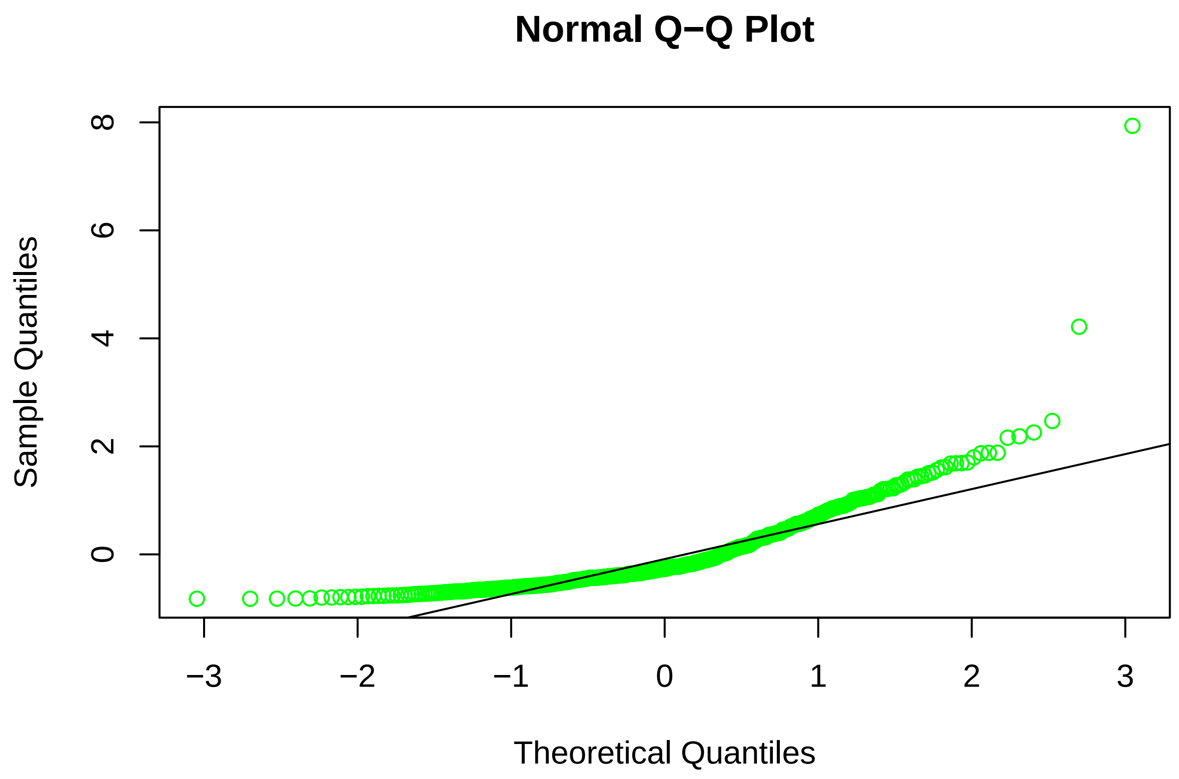
```r
# experiment dataframe
experiment <- data %>%
    subset(data$trial.type == "experiment")

# df + model for each subject
subject1 <- experiment %>%
    filter(subject == "001") %>%
    mutate(log_rt.obj = log(rt.obj))

subjects_model1 <- lm(rt.obj ~ 1, data = subject1)
subjects_model_log1 <- lm(log_rt.obj ~ 1, data = subject1)  #log trans

# df + model for each subject
subject2 <- experiment %>%
    filter(subject == "002") %>%
    mutate(log_rt.obj = log(rt.obj))

subjects_model2 <- lm(rt.obj ~ 1, data = subject2)
subjects_model_log2 <- lm(log_rt.obj ~ 1, data = subject2)  #log trans

# df + model for each subject
subject3 <- experiment %>%
    filter(subject == "003") %>%
    mutate(log_rt.obj = log(rt.obj))

subjects_model3 <- lm(rt.obj ~ 1, data = subject3)
subjects_model_log3 <- lm(log_rt.obj ~ 1, data = subject3)  #log trans

# df + model for each subject
subject4 <- experiment %>%
    filter(subject == "004") %>%
    mutate(log_rt.obj = log(rt.obj))

subjects_model4 <- lm(rt.obj ~ 1, data = subject4)
subjects_model_log4 <- lm(log_rt.obj ~ 1, data = subject4)  #log trans


# plotting the residuals in a QQ-plot
qqnorm(resid(subjects_model1), col = "blue")
qqline(resid(subjects_model1))
```



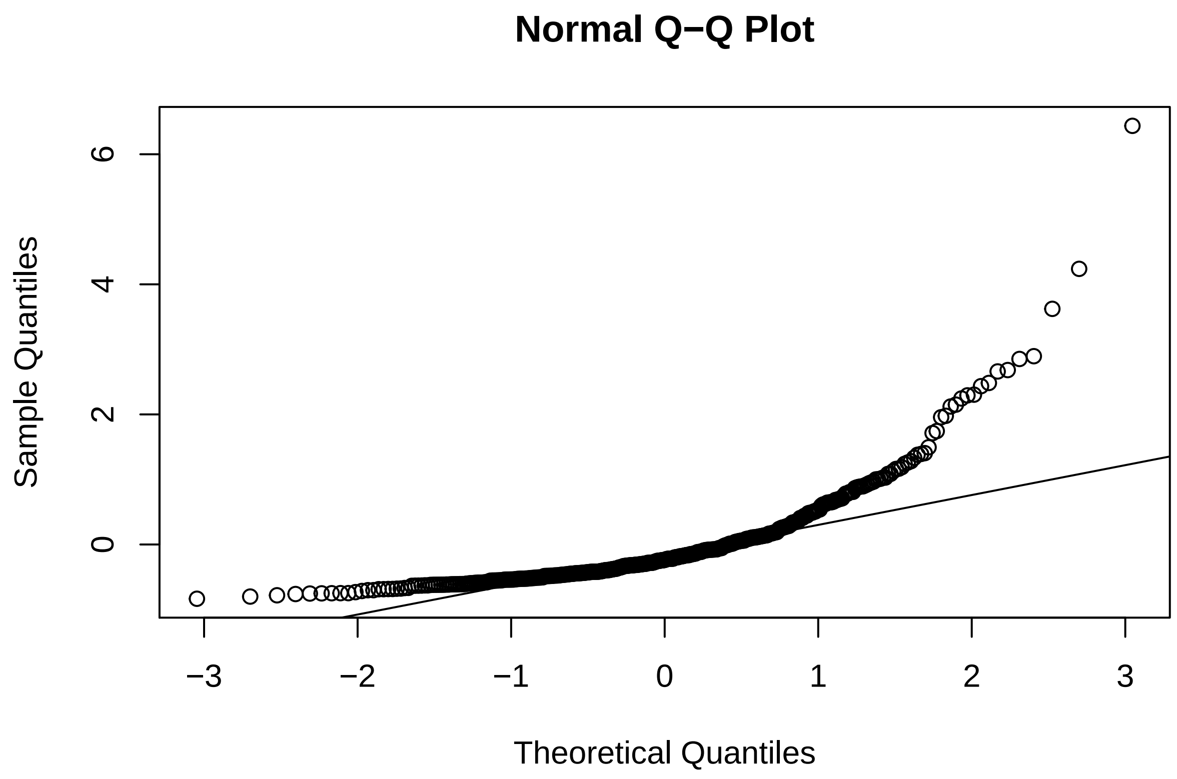
```
qqnorm(resid(subjects_model2), col = "red")
qqline(resid(subjects_model2))
```



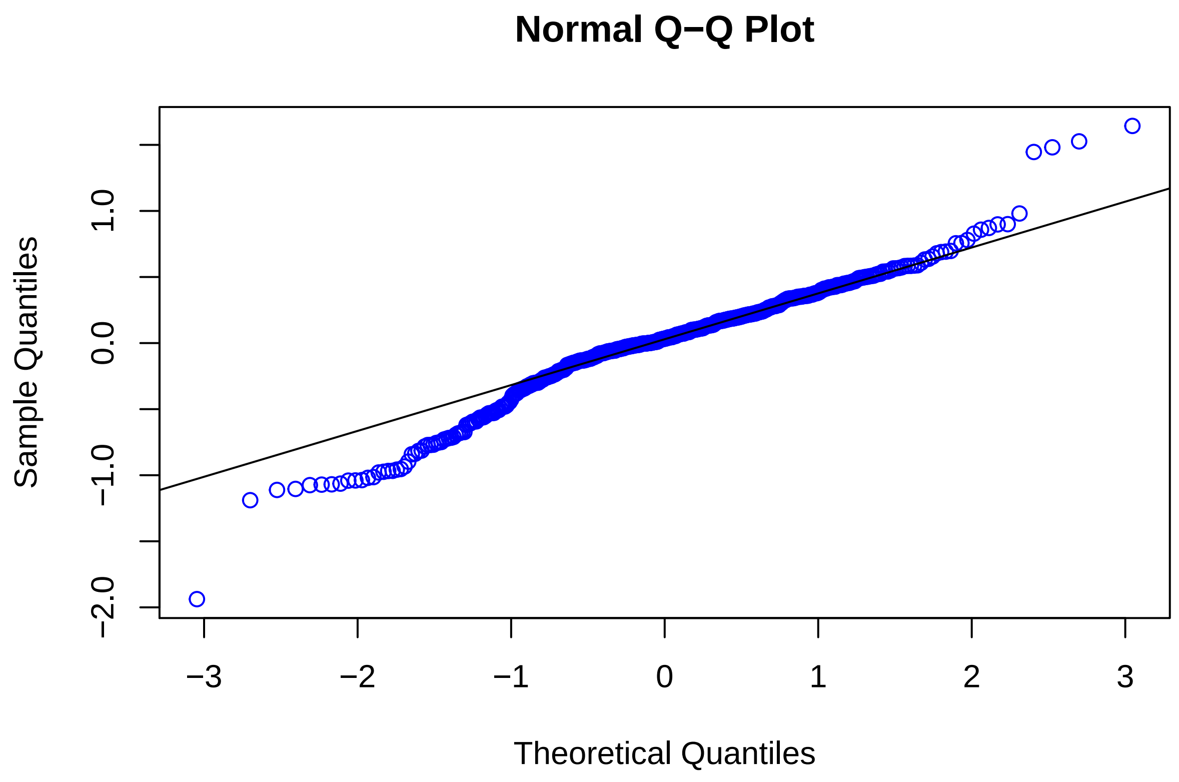
```
qqnorm(resid(subjects_model3), col = "green")
qqline(resid(subjects_model3))
```



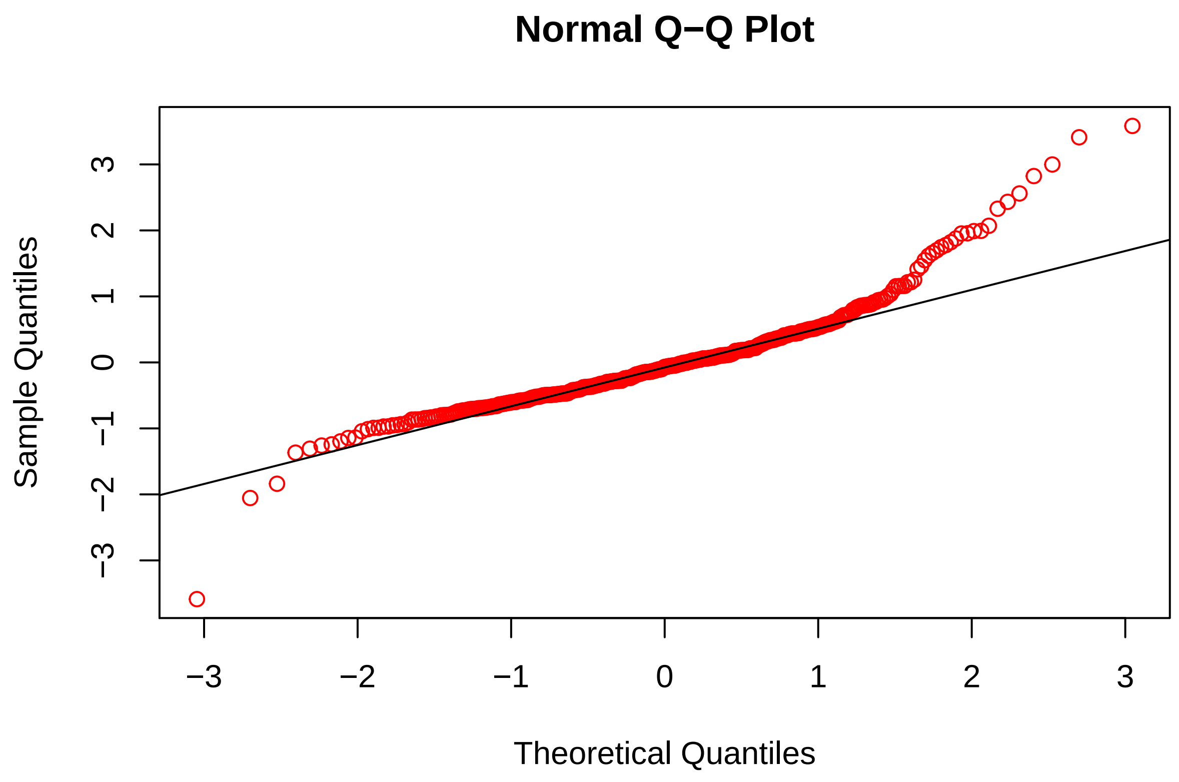
```
qqnorm(resid(subjects_model4), col = "black")
qqline(resid(subjects_model4))
```



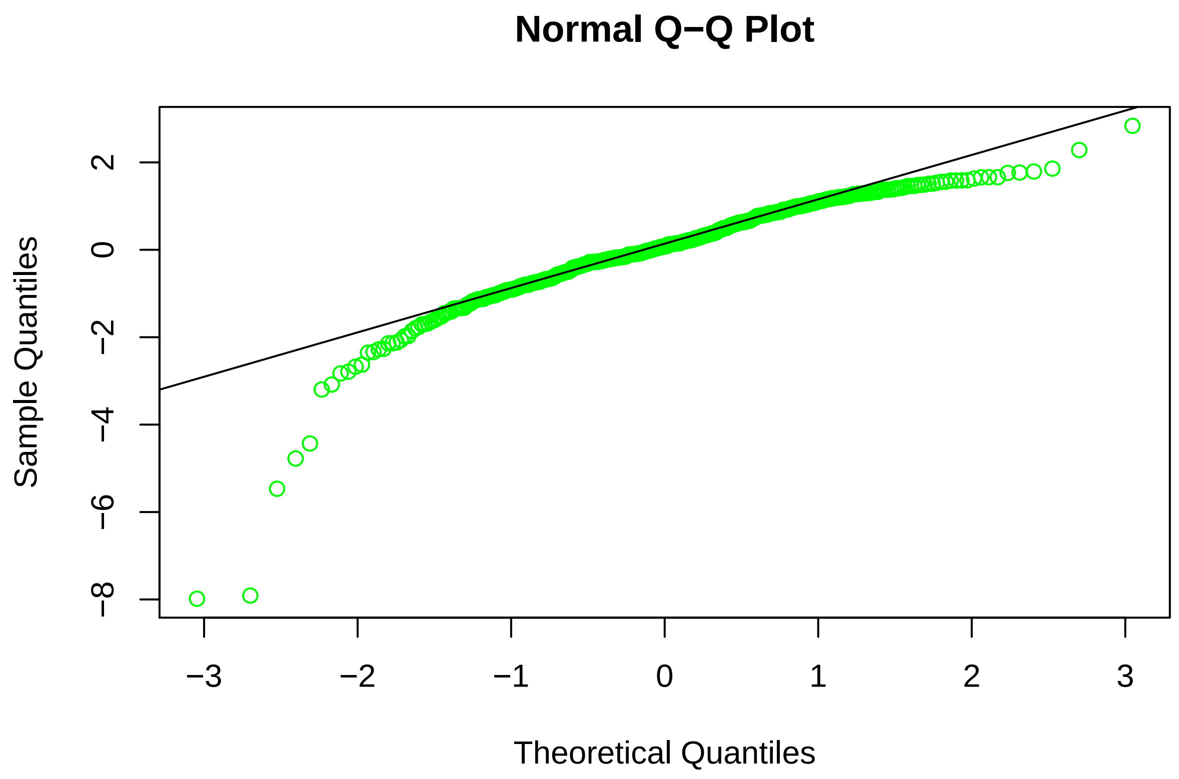
```
# plotting the residuals after log-transformation
qqnorm(resid(subjects_model_log1), col = "blue")
qqline(resid(subjects_model_log1))
```



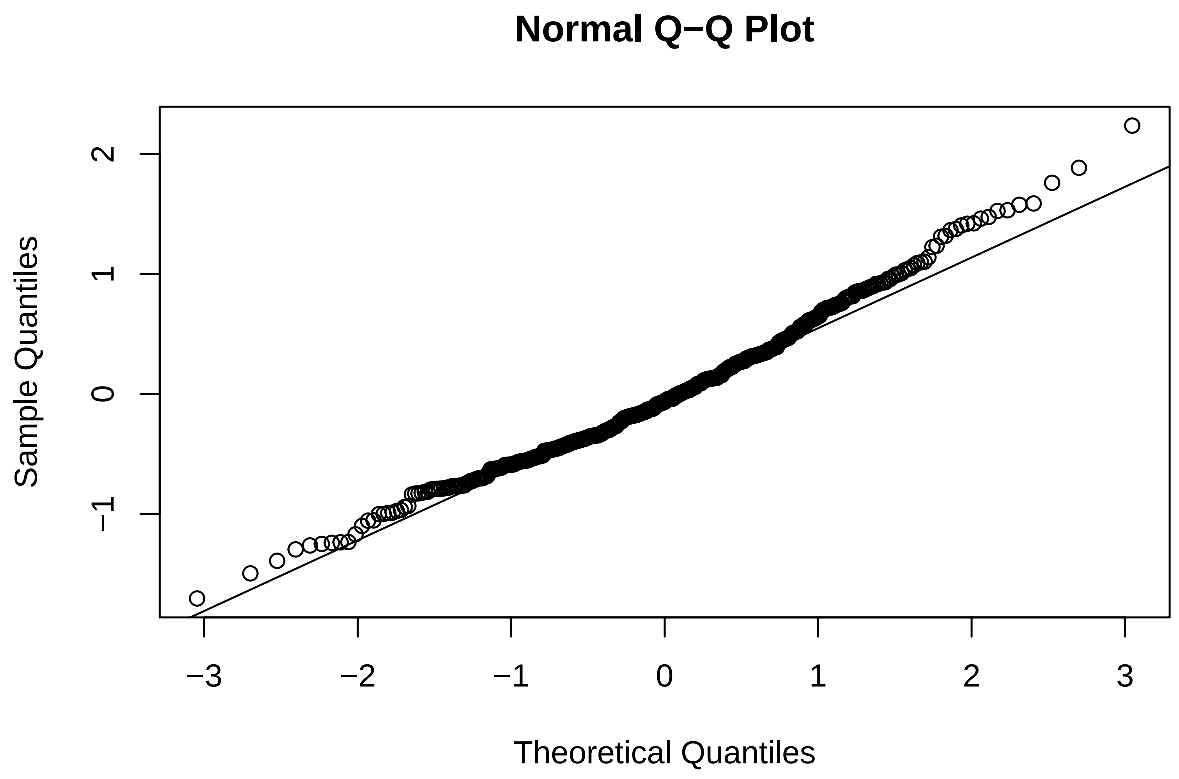
```
qqnorm(resid(subjects_model_log2), col = "red")
qqline(resid(subjects_model_log2))
```



```
qqnorm(resid(subjects_model_log3), col = "green")
qqline(resid(subjects_model_log3))
```



```
qqnorm(resid(subjects_model_log4), col = "black")
qqline(resid(subjects_model_log4))
```



2) Now do a partial pooling model modelling objective response times as dependent on *task*? (set `REML=FALSE` in your `lmer`-specification)

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) First, I built different models, in order to argue which effects to include. The partial pooling model 4 (log(rt.obj) ~ task + (1 + task | subject) + (1 | trial)), has the lowest sigma as well as AIC value. This is the model that predicts objective response times (rt.obj) dependent on task (fixed effects), modelling a random intercepts for subjects and trial as well as random slopes for task. Random intercepts were modeled for subject as one would expect individual baseline performance differences among different participants. Trial as also been modelled as random intercepts as you would expect different trials to vary in difficulty.

ii. explain in your own words what your chosen models says about response times between the different tasks The estimates for quadruplet and singles are negative, which indicates that participants have a faster objective response time compared to the intercept (pair). When comparing the tasks "quadruplet" and "singles" the people are generally faster to the "single" task. The effect is, however, strongest from changing from "pairs" to "singles" (the respondents generally complete the singles tasks faster, relative to the quadruplet tasks). I logged the rt.obj, due to the result in the previous exercise, where we found the logged version of rt.obj better for use (more normalized residuals).

```
# partial pooling models
partial1 <- lmerTest::lmer(log(rt.obj) ~ task + (1 | subject) +
    (1 | trial), data = experiment, REML = FALSE)
```

```
partial2 <- lmerTest::lmer(log(rt.obj) ~ task + (1 | trial),
    data = experiment, REML = FALSE)

partial3 <- lmerTest::lmer(log(rt.obj) ~ task + (1 | subject),
    data = experiment, REML = FALSE)

partial4 <- lmerTest::lmer(log(rt.obj) ~ task + (1 + task | subject) +
    (1 | trial), data = experiment, REML = F)

partial5 <- lmerTest::lmer(log(rt.obj) ~ task + (1 + task | subject),
    data = experiment, REML = FALSE)

# Akaike information criterion
AIC(partial1, partial2, partial3, partial4, partial5)
```

```
##          df      AIC
## partial1  6 29459.94
## partial2  5 32560.15
## partial3  5 29685.31
## partial4 11 29327.68
## partial5 10 29560.34
```

```
# sigma
partial_sigma <- c(sigma(partial1), sigma(partial2), sigma(partial3),
    sigma(partial4), sigma(partial5))
print(partial_sigma)
```

```
## [1] 0.7673537 0.8774439 0.7865108 0.7610302 0.7805267
```

```
# pesudo R2
r.squaredGLMM(partial1)  # R2c = .2629
```

```
##              R2m       R2c
## [1,] 0.006697664 0.2628567
```

```
r.squaredGLMM(partial2)  # R2c = .0360
```

```
##              R2m        R2c
## [1,] 0.006663833 0.03597408
```

```
r.squaredGLMM(partial3)  # R2c = .2253
```

```
##              R2m       R2c
## [1,] 0.006538142 0.2253338
```

```
r.squaredGLMM(partial4)  # R2c = .2750
```

```
##              R2m       R2c
## [1,] 0.006742102 0.2750811
```

```r
r.squaredGLMM(partial5)  # R2c = .2371
```

```
##               R2m       R2c
## [1,] 0.006538065 0.2370859
```

```r
pseudo_r2 <- c(0.2629, 0.036, 0.2253, 0.275, 0.2371)
print(pseudo_r2)
```

```
## [1] 0.2629 0.0360 0.2253 0.2750 0.2371
```

```r
# printing the best model
summary(partial4)
```

```
## Linear mixed model fit by maximum likelihood . t-tests use Satterthwaite's
##   method [lmerModLmerTest]
## Formula: log(rt.obj) ~ task + (1 + task | subject) + (1 | trial)
##    Data: experiment
##
##      AIC      BIC   logLik deviance df.resid
##  29327.7  29409.5 -14652.8  29305.7    12517
##
## Scaled residuals:
##      Min       1Q   Median       3Q      Max
## -10.5363  -0.4958  -0.0279   0.5119   8.0799
##
## Random effects:
##  Groups   Name           Variance Std.Dev. Corr
##  trial    (Intercept)    0.02995  0.17307
##  subject  (Intercept)    0.14920  0.38626
##           taskquadruplet 0.00400  0.06325   0.34
##           tasksingles    0.03227  0.17963   0.38 -0.58
##  Residual                0.57917  0.76103
## Number of obs: 12528, groups:  trial, 432; subject, 29
##
## Fixed effects:
##                 Estimate Std. Error       df t value Pr(>|t|)
## (Intercept)    -0.26407    0.07318 29.74697  -3.609  0.00111 **
## taskquadruplet -0.07225    0.02054 28.55869  -3.518  0.00148 **
## tasksingles    -0.17868    0.03739 29.11864  -4.779 4.65e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) tskqdr
## taskqudrplt  0.099
## tasksingles  0.280 -0.114
## optimizer (nloptwrap) convergence code: 0 (OK)
## Model failed to converge with max|grad| = 0.00250746 (tol = 0.002, component 1)
```

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 i
```

When adding different types of group intercepts (random effect), I can only add two intercepts: (1|subject)+(1|trial), before the model meets singular fit issues.

ii. create a model by adding random intercepts (without modelling slopes) that results in a singular fit - then use `print(VarCorr(<your.model>), comp='Variance')` to inspect the variance vector - explain why the fit is singular (Hint: read the first paragraph under details in the help for `isSingular`) By adding (1|subject)+(1|trial)+(1|task) to the existing model with "pas" and "task" as fixed effects interaction, the model met singular fits issues. When inspecting the variance vector of the model with singular fits (np_model_4), it shows that the reason for the singular fit, is that the model contains a random-effect variance estimate of zero, this random-effect is (1|task).

iii. in your own words - how could you explain why your model would result in a singular fit? I suppose the singular fit is a result of overfitting. The model and its random effects are to complex to be supported by the data. To overcome this, we could remove some random effects one by one.

```
# original model
np_model_1 <- lm(rt.obj ~ task + pas + pas * task, data = experiment,
    REML = FALSE)

# adding random effects until convergence issues and singular
# fit
np_model_2 <- lmer(rt.obj ~ task + pas + pas * task + (1 | subject),
    data = experiment, REML = FALSE)
np_model_3 <- lmer(rt.obj ~ task + pas + pas * task + (1 | subject) +
    (1 | trial), data = experiment, REML = FALSE)
np_model_4 <- lmer(rt.obj ~ task + pas + pas * task + (1 | subject) +
    (1 | trial) + (1 | task), data = experiment, REML = FALSE)  ### boundary (singular fit)

# cheking out why np_model_4 has a singular fit
print(VarCorr(np_model_4), comp = "Variance")  # random effect task variance = 0
```

```
##  Groups   Name        Variance
##  trial    (Intercept) 0.0025289
##  subject  (Intercept) 0.0974986
##  task     (Intercept) 0.0000000
##  Residual             8.1472913
```

```
# when a random effect task variance = 0, this random effect
# does not explain anything
```

Exercise 3

1) Initialize 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

```
# new dataframe counting number of the 4 pas in each task
data_count <- data %>%
    group_by(subject, task, pas) %>%
    summarise(count = n())
```

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

```
# multilevel model
countmodel1 <- glmer(count ~ pas * task + (1 + pas | subject),
    data = data_count, family = poisson, control = glmerControl(optimizer = "bobyqa"))
```

i. which family should be used? Poisson distribution because we are dealing with count data.

ii. why is a slope for *pas* not really being modelled? The "pas" variable is encoded as a factor, meaning that it computes the analysis for each level separately. Thus, we'll get a slope for each separate level of "pas", where each slope is relative to the reference level.

```
# check why PAS is not really being modelled
summary(countmodel1)
```

```
## 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 0.842458
## pas3                  -0.51365    0.20718  -2.479 0.013166 *
## pas4                  -0.77292    0.29075  -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 -0.11375    0.04605  -2.470 0.013508 *
## pas3:taskquadruplet -0.20901    0.05287  -3.954 7.70e-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
```

iii. if you get a convergence error, try another algorithm (the default is the *Nelder_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`) No convergence error occur if the aforementioned algorithm is used.

```r
# Model with only the main effects of pas and task
countmodel2 <- glmer(count ~ pas + task + (1 + pas | subject),
    data = data_count, family = "poisson", control = glmerControl(optimizer = "bobyqa"))

# residuals
resids_countmodel1 <- sum(residuals(countmodel1)^2)
print(resids_countmodel1)
```

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

```
## [1] 699.4866
```

```r
resids_countmodel2 <- sum(residuals(countmodel2)^2)
print(resids_countmodel2)
```

```
## [1] 962.4628
```

```
# akaike information criterion
AIC(countmodel1, countmodel2)
```

```
##             df      AIC
## countmodel1 22 3148.441
## countmodel2 16 3398.549
```

```
# comparing by R^2
MuMIn::r.squaredGLMM(countmodel1)
```

```
##                 R2m       R2c
## delta     0.1577062 0.9750530
## lognormal 0.1577375 0.9752462
## trigamma  0.1576745 0.9748567
```

```
MuMIn::r.squaredGLMM(countmodel2)
```

```
##                 R2m       R2c
## delta     0.1375957 0.9747420
## lognormal 0.1376233 0.9749376
## trigamma  0.1375677 0.9745434
```

v. indicate which of the two models, you would choose and why I would choose countmodel1 as it has the lowest AIC, lowest residual standard deviation, and the highest pseudo R2. I guess, that it conceptually also makes more sense to look at the interaction between "task" and "pas" and not how they individually predict "count".

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* We have used lmerTest (Kuznetsova, Brockhoff and Christensen, 2017) to perform a linear mixed effects analysis of the relationship between 'count', 'task' and 'pas'. Count indicates the number of times the subject categorized their experience as PAS 1-4 for each task. PAS indicates perceptual awareness scale ranging from 1-4, and task being 3 different versions (singles, pairs and quadruplets). The mixed effect model had the following syntax:

count ~ pas * task + (1 + pas | subject)

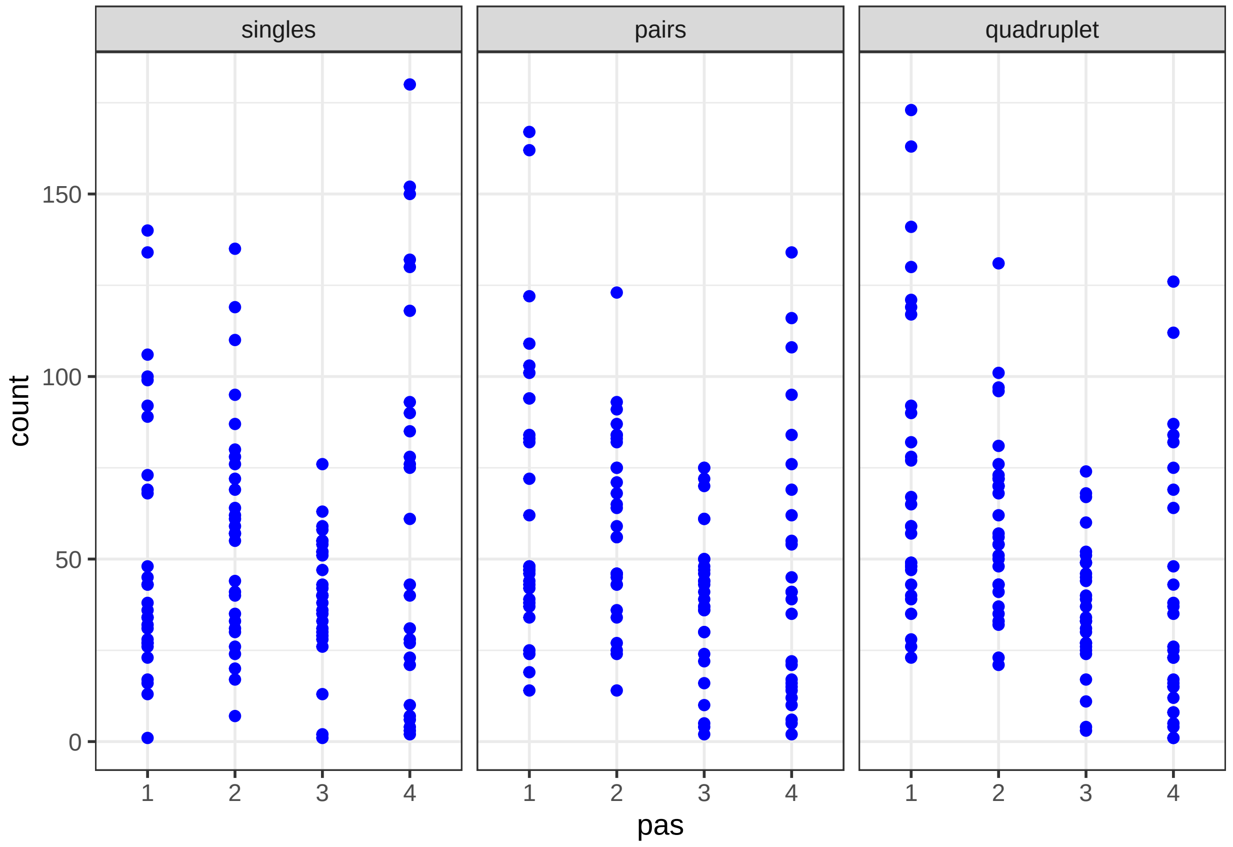
The mixed effect model (countmodel1 = count~pas*task + (1+pas|subject)) was constructed to predict "count" from the fixed effect (both main and interaction) between "pas" and "task" and the random slope "pas" given random intercept "subject". The direction of the slope (whether it is positive or negative) is highly dependent on the interaction between pas and task.

Both fixed and random effects accounted for roughly 98% of variance in the pitch variable. The observed interactions between pas and task were significant with p-values = < 0.05.

The model has a RSS = 699.5, an AIC = 3148.4, and an R2c = 97.5. The model has a lower AIC value than the other model (AIC= 3398.5), and a slightly better R2c and RSS.

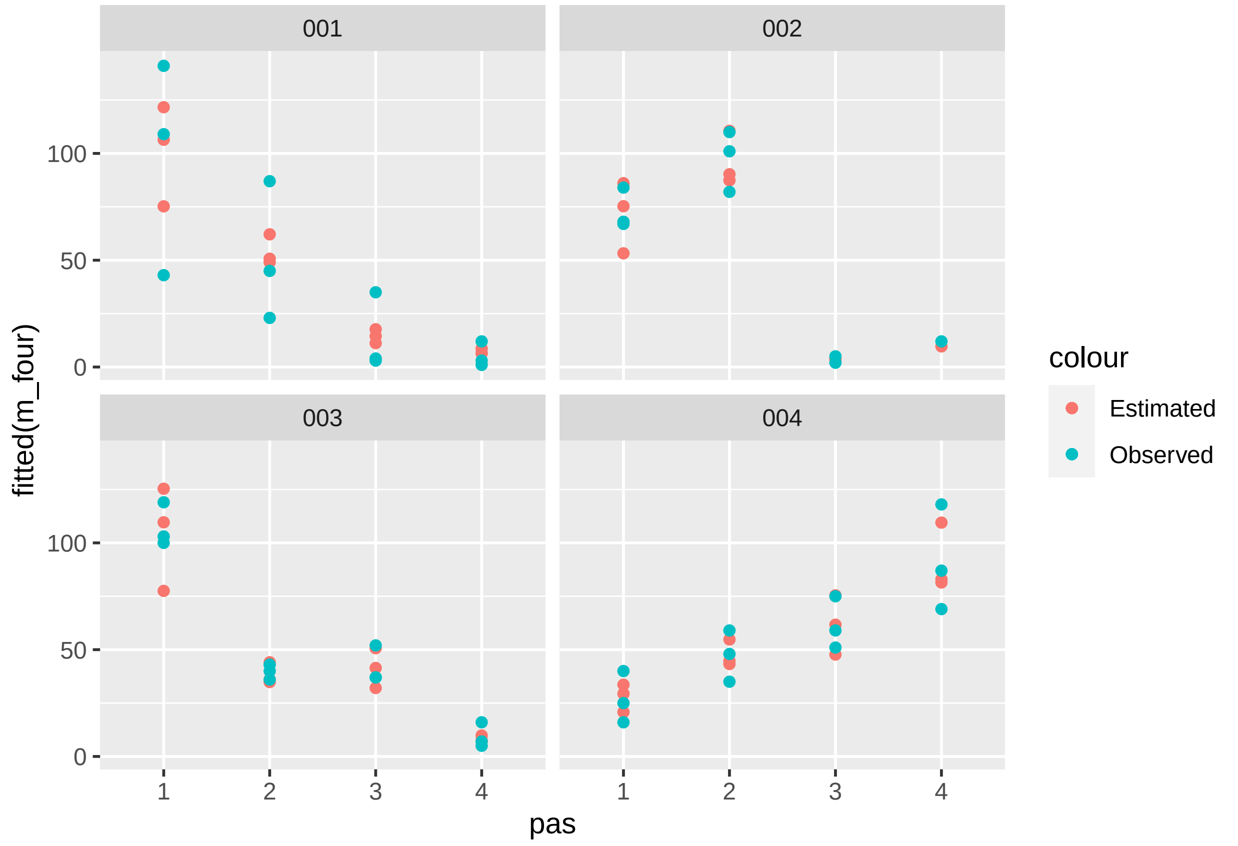
Furthermore, the two plots show, how the the rating of "pas" are to a certain extent similar across the 3 tasks. Though, "pas 3" is does not have as many ratings as the other ones.

```
# plot for report
data_count$task <- relevel(data_count$task, ref = "singles")

ggplot(data_count, aes(x = pas, y = count)) + geom_point(aes(pas,
    count), color = "blue") + facet_wrap(~task) + theme_bw()
```



```
# plot for 4 subjects: 001, 002, 003, 004,
data.count_four <- data_count %>%
    filter(subject == "001" | subject == "002" | subject == "003" |
        subject == "004")
m_four <- glmer(count ~ pas * task + (pas | subject), data = data.count_four,
    family = poisson)
data.count_four %>%
    ggplot() + geom_point(aes(x = pas, y = fitted(m_four), color = "Estimated")) +
    geom_point(aes(x = pas, y = count, color = "Observed")) +
    facet_wrap(~subject)
```

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



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

i. does *task* explain performance? The output of the model indicates that "task" is significant for all levels but "quaduplet", therefore task does explain performance for "singles" and "pairs". Though, inspecting the coefficients of the model, it makes great sense that "quadruplet" is negative, which indicates more a decrease in "correct" answers, i.e when complexity increases the amount correct answer decreases. Calculated probabilities state: 75% probability of getting 1 (correct) in pair task, 73% probability of getting 1 (correct) in quadruplet task, 78% probability of getting 1 (correct) in single task. Meaning, there is less probability of being correct in the quadruplet task. Though note, these probabilities are still quite similar, so drawing any conclusion from them would probably be faulty.

```
# model with 'correct'
model_correct1 <- glmer(correct ~ task + (1 | subject), data = experiment,
    family = binomial)
# see summary again to see log odds
summary(model_correct1)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: correct ~ task + (1 | subject)
##    Data: experiment
##
```

```
##       AIC      BIC   logLik deviance df.resid
##   13636.3  13666.0  -6814.1  13628.3    12524
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -3.7450 -1.0599  0.4791  0.6483  0.9795
##
## Random effects:
##  Groups  Name        Variance Std.Dev.
##  subject (Intercept) 0.3622   0.6018
## Number of obs: 12528, groups:  subject, 29
##
## Fixed effects:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)      1.11896    0.11770   9.507  < 2e-16 ***
## taskquadruplet -0.07496    0.05082  -1.475  0.14019
## tasksingles      0.16603    0.05218   3.182  0.00146 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) tskqdr
## taskqudrplt -0.220
## tasksingles -0.213  0.494
```

```
# probability getting each task
pair <- invlogit(1.11896)
print(pair)
```

```
## [1] 0.7537958
```

```
quar <- invlogit(1.11896 - 0.07496)
print(quar)
```

```
## [1] 0.7396211
```

```
single <- invlogit(1.118961 + 0.16603)
print(single)
```

```
## [1] 0.7832982
```

ii. add *pas* as a main effect on top of *task* - what are the consequences of that? When adding "pas" as a main effect on top of "task", "task" becomes insignificant and "pas" becomes significant, indicating "pas" could be a better predictor.

```
# add _pas_ as a main effect on top of _task_ - what are the
# consequences of that?
model_correct2 <- glmer(correct ~ task + pas + (1 | subject),
    data = experiment, family = binomial)
summary(model_correct2)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: correct ~ task + pas + (1 | subject)
##    Data: experiment
## 
##      AIC      BIC   logLik deviance df.resid 
##  12260.4  12312.5  -6123.2  12246.4    12521 
## 
## Scaled residuals: 
##     Min      1Q  Median      3Q     Max 
## -5.9663 -0.7416  0.3198  0.6051  1.6044 
## 
## Random effects:
##  Groups  Name        Variance Std.Dev.
##  subject (Intercept) 0.2388   0.4886  
## Number of obs: 12528, groups:  subject, 29
## 
## Fixed effects:
##                Estimate Std. Error z value Pr(>|z|)    
## (Intercept)     0.14963    0.10255   1.459    0.145    
## taskquadruplet -0.02659    0.05368  -0.495    0.620    
## tasksingles    -0.03642    0.05569  -0.654    0.513    
## pas2            0.88735    0.05415  16.388   <2e-16 ***
## pas3            1.87054    0.07482  24.999   <2e-16 ***
## pas4            2.88685    0.10243  28.183   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Correlation of Fixed Effects:
##             (Intr) tskqdr tsksng pas2   pas3  
## taskqudrplt -0.271                            
## tasksingles -0.236  0.488                     
## pas2        -0.235  0.011 -0.040              
## pas3        -0.190  0.020 -0.052  0.381       
## pas4        -0.152  0.011 -0.095  0.273  0.263
```

```r
# now fit a multilevel model that models _correct_ as
# dependent on _pas_ with a unique intercept for each
# _subject_
model_correct3 <- glmer(correct ~ pas + (1 | subject), data = experiment,
    family = binomial)
summary(model_correct3)
```

iii. now fit a multilevel model that models *correct* as dependent on *pas* with a unique intercept for each *subject*

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: correct ~ pas + (1 | subject)
##    Data: experiment
##
##      AIC      BIC   logLik deviance df.resid
##  12256.9  12294.1  -6123.4  12246.9    12523
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -6.0023 -0.7364  0.3214  0.6111  1.5907
##
## Random effects:
##  Groups  Name        Variance Std.Dev.
##  subject (Intercept) 0.2385   0.4883
## Number of obs: 12528, groups:  subject, 29
##
## Fixed effects:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.13014    0.09788    1.33    0.184
## pas2         0.88638    0.05407   16.39   <2e-16 ***
## pas3         1.86891    0.07461   25.05   <2e-16 ***
## pas4         2.88198    0.10170   28.34   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##      (Intr) pas2   pas3
## pas2 -0.249
## pas3 -0.202  0.379
## pas4 -0.172  0.269  0.257
```

iv. finally, fit a model that models the interaction between *task* and *pas* and their main effects
See chunk. I'm not sure whether you mean if we should still keep (1|subject) in the model, but I just do.

```
# finally, fit a model that models the interaction between
# _task_ and _pas_ and their main effects
model_correct4 <- glmer(correct ~ pas + task + pas * task + (1 |
    subject), data = experiment, family = binomial)
summary(model_correct4)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: correct ~ pas + task + pas * task + (1 | subject)
##    Data: experiment
##
##      AIC      BIC   logLik deviance df.resid
##  12265.8  12362.4  -6119.9  12239.8    12515
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -6.4528 -0.7529  0.3231  0.6026  1.6685
##
## Random effects:
##  Groups  Name        Variance Std.Dev.
##  subject (Intercept) 0.2403   0.4902
## Number of obs: 12528, groups:  subject, 29
##
## Fixed effects:
##                      Estimate Std. Error z value Pr(>|z|)
## (Intercept)           0.14959    0.10796   1.386    0.166
## pas2                  0.91504    0.08894  10.289   <2e-16 ***
## pas3                  1.82975    0.12090  15.135   <2e-16 ***
## pas4                  2.83307    0.17360  16.320   <2e-16 ***
## taskquadruplet        0.03068    0.07763   0.395    0.693
## tasksingles          -0.11076    0.08405  -1.318    0.188
## pas2:taskquadruplet -0.11916    0.12081  -0.986    0.324
## pas3:taskquadruplet -0.06631    0.16769  -0.395    0.693
## pas4:taskquadruplet -0.17375    0.23515  -0.739    0.460
## pas2:tasksingles      0.05846    0.12725   0.459    0.646
## pas3:tasksingles      0.20714    0.17127   1.209    0.226
## pas4:tasksingles      0.28173    0.22752   1.238    0.216
## ---
## 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        -0.350
## pas3        -0.269  0.335
## pas4        -0.198  0.228  0.189
## taskqudrplt -0.371  0.451  0.332  0.231
## tasksingles -0.340  0.415  0.304  0.207  0.475
## ps2:tskqdrp  0.237 -0.695 -0.212 -0.147 -0.644 -0.304
## ps3:tskqdrp  0.171 -0.209 -0.680 -0.105 -0.464 -0.221  0.299
## ps4:tskqdrp  0.122 -0.149 -0.110 -0.694 -0.332 -0.158  0.213    0.152
## ps2:tsksngl  0.232 -0.671 -0.213 -0.146 -0.313 -0.662  0.484    0.146
## ps3:tsksngl  0.170 -0.204 -0.671 -0.108 -0.233 -0.491  0.149    0.481
## ps4:tsksngl  0.127 -0.151 -0.110 -0.720 -0.176 -0.371  0.113    0.082
##             ps4:tskq ps2:tsks ps3:tsks
## pas2
## pas3
## pas4
## taskqudrplt
## tasksingles
## ps2:tskqdrp
## ps3:tskqdrp
## ps4:tskqdrp
## ps2:tsksngl  0.103
## ps3:tsksngl  0.077    0.327
## ps4:tsksngl  0.532    0.244    0.181
```

v. describe in your words which model is the best in explaining the variance in accuracy The model that explains the most variance is model_correct4 (correct~pas+task+pas*task+ (1|subject)), since it has the lowest R2 of 0.30. Though, model_correct3 has the lowest AIC value, but model_correct4's AIC is almost just as small. In fact model_correct2, model_correct3, and model_correct4 are quite good all of them.

```
# comparing by AIR and R^2
MuMIn::r.squaredGLMM(model_correct1)  # R2c = 0.10167098
```

```
##                     R2m        R2c
## theoretical 0.002769013 0.10167098
## delta       0.001807539 0.06636812
```

```
MuMIn::r.squaredGLMM(model_correct2)  # R2c = 0.3018983
```

```
##                   R2m       R2c
## theoretical 0.2512343 0.3018983
## delta       0.1777562 0.2136025
```

```
MuMIn::r.squaredGLMM(model_correct3)  # R2c = 0.3015883
```

```
##                   R2m       R2c
## theoretical 0.2509652 0.3015883
## delta       0.1775427 0.2133554
```

```
MuMIn::r.squaredGLMM(model_correct4)  # R2c = 0.3031205 #best
```

```
##                   R2m       R2c
## theoretical 0.2522223 0.3031205
## delta       0.1785466 0.2145771
```

```
aic <- AIC(model_correct1, model_correct2, model_correct3, model_correct4)
print(aic)
```

```
##                df      AIC
## model_correct1  4 13636.29
## model_correct2  7 12260.41
## model_correct3  5 12256.88
## model_correct4 13 12265.77
```