Demonstration

Johannes Keil

Demonstration for preregistration

This script is meant to illustrate and further elaborate the analysis approach discussed in the preregistration. Note that we do not at all intend to 'preregister' this code - the final code used in analysis may be very different. Think of the code provided more as a 'sketch' - For example, in many analyses crucial steps, e.g., checking data quality, are not included.

First, load in our data.

Reliability

Random responding

F-test:

Van Leeuwen & Mandabach (2002) point out that ANOVA is relatively robust to ipsative data (Greer & Dunlap, 1997). So, we could simply use an omnibus one-way F-test on ranks, which tests the null-hypothesis whether means in the dataset are different, or all equivalent to the grand mean (= 0). If this test is significant, this would indicate that there is a true difference in preference and respondents do not respond at random. However, this analysis applies to the entire dataset (i.e., overall, do participants respond at random?), not individual participants. So, it just yields a very rough yes/no answer.

Assess reliability using the trick from Van Leeuwen & Mandabach (2002).

```
# recompute wins & ranks within each participant, but separate by wave

  a <- dat %>%
    group_by(participant, wave, item1) %>%
    summarise(
      wins1 = sum(win1)
    ) %>%
```

```
    ungroup() %>%
    dplyr::rename(item = item1)
```

```
`summarise()` has grouped output by 'participant', 'wave'. You can override
using the `.groups` argument.
```

```
  # some entries are missing, because items only appear in the item1 or item2 column
  # for our merge operation below to include these, we need to
  # manually add those entries

  for (w in 1:2) {
    for (subj in unique(dat$participant)){

      item1_list <- filter(dat, participant == subj, wave == w)$item1 %>% unique()
      item2_list <- filter(dat, participant == subj, wave == w)$item2 %>% unique()

      for (i in item2_list[!item2_list %in% item1_list]) {
        a <- a %>% add_row(
          participant = subj,
          wave = w,
          item = i,
          wins1 = 0
        )
      }
    }
  }

  b <- dat %>%
    group_by(participant, wave, item2) %>%
    summarise(
      wins2 = sum(win2)
    ) %>%
    ungroup() %>%
    dplyr::rename(item = item2)
```

```
`summarise()` has grouped output by 'participant', 'wave'. You can override
using the `.groups` argument.
```

```
  for (w in 1:2) {
    for (subj in unique(dat$participant)){
```

```
      item1_list <- filter(dat, participant == subj, wave == w)$item1 %>% unique()
      item2_list <- filter(dat, participant == subj, wave == w)$item2 %>% unique()

      for (i in item1_list[!item1_list %in% item2_list]) {
        b <- b %>%
          add_row(
            participant = subj,
            wave = w,
            item = i,
            wins2 = 0
        )
      }
    }
  }

# compute ranks within each participant
  wins_participant <- merge(a, b) %>%
    mutate(wins = wins1 + wins2) %>%
    arrange(participant) %>%
    group_by(participant)


wins_participant_w1 <- wins_participant %>%
  filter(wave == 1) %>%
  mutate(
    rank = rank(wins)
  )


wins_participant_w2 <- wins_participant %>%
  filter(wave == 2) %>%
  mutate(
    rank = rank(wins)
  )
# overall not all items seem to be given equal rank.
# This means that (a) participants are not all random responders and (b) participants have sl
res1 <- oneway.test(rank ~ item, data = wins_participant_w1, var.equal = FALSE)

res2 <- oneway.test(rank ~ item, data = wins_participant_w2, var.equal = FALSE)
```

Determining a 'significance threshold' for random responding

The following two methods work by conducting a single significance test per participant, to see whether their response pattern is significantly different from 0. This test is computed for each participant. Thus, we are in need of a meaningful threshold for judging whether the proportion of participants whose test is significant (indicating non-random responding) is larger than would be expected at pure chance given our participant-level alpha significance threshold.

We can determine this threshold by constructing a binomial significance test. Let our null hypothesis be that all participants are answering at random. Thus, under the null hypothesis, the number of significant tests in our sample is binomially distributed as:

$$P(X = k) = \binom{n}{k} * \alpha^k * (1 - \alpha)^{n-k}$$

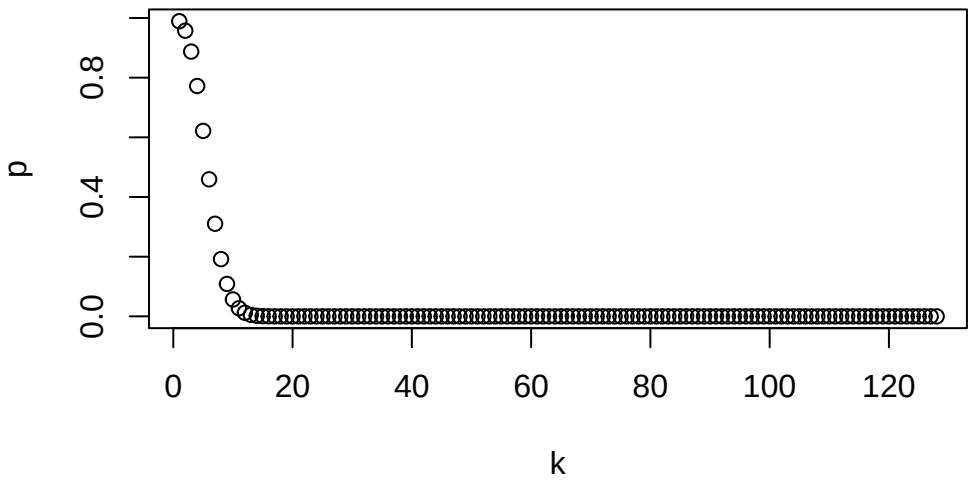
Where k is the number of significant tests we observe, n is the total number of participants, and alpha is our significance threshold. We now compute the cumulative binomial distribution:

$$P(X \geq k) = 1 - \sum_{i}^{k} P(X = i)$$

If this probability is very small given the actually observed number of significant tests, this would be grounds to reject the null hypothesis.

Given our sample size of n = 128 (barring potential exclusions), plot $P(X \geq k)$:

```
k = 1:128
n = 128
p = 1- pbinom(k, size = 128, prob = 0.05)

plot(k, p)
```



If we use a very stringent significance threshold of 0.001, this criterion would be met if more than 15 participants were found not to respond at random. However, clearly, only 15% of participants not responding at random would not be a satisfying outcome practically. For this reason, we arbitrarily set a higher threshold of 70%.

Kendall/David Method

Mazzuchi et al. (2008) applied a solution taken from the work of David (1963) and Kendall (1962) to the problem of reliability in aerospace safety expert judgments. The idea is that a perfectly deterministic rater should always rank items in a transitive fashion (i.e., if A is preferred over B, and B over C, than C cannot be preferred over A). Let item i and judge r, then N_r is the number of times that j ranked i as more severe than the other items (= the number of 'wins'). Then the number of 'intransitive' ratings can be calculated as:

$$c(r) = \frac{n(n^2-1)}{24} - \frac{1}{2} * \sum (N_r - \frac{1}{2}(n-1))^2$$

From this, we can derive an approximately chi-square distributed statistic for n > 7, with n(n-1) * (n-2) /(n-4)^2 degrees of freedom (Kendall, 1962):

$$c'(r) = d.f. + (\frac{8}{n-4}) * [\frac{1}{4} * \binom{n}{3} - c(r) + \frac{1}{2}]$$

This tests the null-hypothesis that the participant answered randomly - so a significant value would indicate that the participant's responses were not random.

```
############################
# Analysis with real data #
############################

# the list of false responders
false_responders <- list(
  participant = c(),      # participant id
  c_bar = c(),            # the false responding statistic
  p_val = c(),            # corresponding p_value
  is_false_responder = c()   # TRUE if participant was a false responder
)

# significance threshold
alpha = 0.05

# iterate over the number of participants
for (subj in unique(dat$participant)) {

  # select subset of data
  this_dat <- dat %>%
    filter(
      participant == subj,
      wave == 1
    )

  n_wins <- c()

  # the number of items
  k = unique(c(this_dat$item1, this_dat$item2)) %>% length()

  # now calculate degrees of freedom for our statistic
  df = k * (k - 1) * (k - 2) / (k - 4)^2

  for (i in 1:k) {
    n_wins[i] <- sum(this_dat$win1[this_dat$item1 == paste("i", i, sep = "")]) + sum(this_dat
  }

  # now apply David (1963)'s formula
  c = k * (k^2-1) / 24 - 0.5 * sum((n_wins - 0.5 * (k-1))^2)
```

```
  # and calculate our test statistic

  c_bar = df + 8 / (k - 4) * (0.25 * choose(k, 3) - c + 0.5)

  # get a p-value
  p_val = 1 - pchisq(c_bar, df)

  # was the participant a false responder
  is_false_responder = ifelse(p_val < alpha, 0, 1)

  false_responders$participant <- c(false_responders$participant, subj)
  false_responders$c_bar <- c(false_responders$c_bar, c_bar)
  false_responders$p_val <- c(false_responders$p_val, p_val)
  false_responders$is_false_responder <- c(false_responders$is_false_responder, is_false_resp

}

false_responders
```

```
$participant
 [1] 10542 10544 10548 10550 10556 10562 10563 10572 10581 10593 10602 10618
[13] 10619 10620 10631 10636 10637 10647 10655 10662

$c_bar
 [1]  590.8280  581.0888 1213.3149  738.2551 1039.9609 1707.9650  877.3600
 [8]  724.1893  852.1665 1607.4004 1164.3737  498.5950 1727.9650 1195.4325
[15] 1308.2482  509.1405 1435.6932  802.5514 1267.7910 1040.5411

$p_val
 [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

$is_false_responder
 [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Entropy

The entropy of a discrete probability distribution of n values tells you the sum across all outcomes i of the surprisal that would occur if the i-th value of the distribution were the outcome of a random experiment. It is maximised for a uniform distribution (since, in this, case we have no clue which outcome to expect), and minimised for a distribution in which one outcome always occurs and the others never occur. If participants have consistent, strong preferences, their entropies should be thus smaller than what would be expected under uniformity. Let p

be the probability that a randomly choosing actor would choose this item as the 'first choice' given the number of wins it has achieved. Then, the entropy H would be:

$$H = -\sum_i p_i * log_2(p_i)$$

We can compute the entropy of the distribution of wins for a single participant:

```
# recompute wins & ranks within each participant, but separate by wave

  a <- dat %>%
    group_by(participant, wave, item1) %>%
    summarise(
      wins1 = sum(win1)
    ) %>%
    ungroup() %>%
    dplyr::rename(item = item1)
```

```
`summarise()` has grouped output by 'participant', 'wave'. You can override
using the `.groups` argument.
```

```
  # some entries are missing, because items only appear in one row
  # for our merge operation below to work, we need to fix this
  # manually add those entries

  for (w in 1:2) {
    for (subj in unique(dat$participant)){

      item1_list <- filter(dat, participant == subj, wave == w)$item1 %>% unique()
      item2_list <- filter(dat, participant == subj, wave == w)$item2 %>% unique()

      for (i in item2_list[!item2_list %in% item1_list]) {
        a <- a %>% add_row(
          participant = subj,
          wave = w,
          item = i,
          wins1 = 0
        )
      }
    }
  }
```

```
  b <- dat %>%
    group_by(participant, wave, item2) %>%
    summarise(
      wins2 = sum(win2)
    ) %>%
    ungroup() %>%
    dplyr::rename(item = item2)
```

```
`summarise()` has grouped output by 'participant', 'wave'. You can override
using the `.groups` argument.
```

```
  for (w in 1:2) {
    for (subj in unique(dat$participant)){

      item1_list <- filter(dat, participant == subj, wave == w)$item1 %>% unique()
      item2_list <- filter(dat, participant == subj, wave == w)$item2 %>% unique()

      for (i in item1_list[!item1_list %in% item2_list]) {
        b <- b %>%
          add_row(
            participant = subj,
            wave = w,
            item = i,
            wins2 = 0
        )
      }
    }
  }

# compute ranks within each participant
  wins_participant_wave <- merge(a, b) %>%
    mutate(wins = wins1 + wins2) %>%
    arrange(participant) %>%
    group_by(participant)

this <- wins_participant_wave %>%
  filter(participant == 10542, wave == 1)

total_wins = this$wins %>% sum()

this <- this %>%
```

```
  mutate(
    p = wins / total_wins,
    p = ifelse(p == 0, 0.000000001, p)
  ) %>%
  mutate(
    h = p * log(p, base = 2)
  )
entropy <- -sum(this$h)
```

Now compare that participant's to a sample that is drawn from a uniform distribution:

```
total_wins = this$wins %>% sum()


these_wins <- data.frame( # list storing wins by item index
  item = unique(this$item),
  wins = rep(0, length(unique(this$item)))
)

for (i in 1:total_wins) {
  index <- runif(1, min = 0, max = length(unique(this$item))) %>% round()
  these_wins$wins[index] <- these_wins$wins[index] + 1
}

these_wins <- these_wins %>%
  mutate(
    p = wins / total_wins,
  ) %>%
  filter(p > 0) %>%
  mutate(
    h = p * log(p, base = 2)
  )
uniform_entropy <- -sum(these_wins$h)
uniform_entropy
```

```
[1] 4.580469
```

And the theoretical maximum:

```r
p <- 1 / length(unique(this$item))

max_entropy = -length(unique(this$item)) * p * log(p, base = 2)
max_entropy
```

```
[1] 4.754888
```

On this basis, we can construct a simulation/permutation test to see whether the entropy is smaller or larger than what would be expected under uniformity:

```r
n_permutations <- 1000

uniform_entropy <- c()

for (perm in 1:n_permutations) {

  these_wins <- data.frame( # list storing wins by item index
    item = unique(this$item),
    wins = rep(0, length(unique(this$item)))
  )

  for (i in 1:total_wins) {
    index <- runif(1, min = 0, max = length(unique(this$item))) %>% round()
    these_wins$wins[index] <- these_wins$wins[index] + 1
  }

  these_wins <- these_wins %>%
    mutate(
      p = wins / total_wins,
    ) %>%
    filter(p > 0) %>%
    mutate(
      h = p * log(p, base = 2)
    )
  uniform_entropy <- c(uniform_entropy, -sum(these_wins$h))
}

# get quantiles for a significance threshold
threshold <- quantile(uniform_entropy, probs = c(0.05))

# in our case, this is significant
```

```r
sig <- entropy < threshold

sig
```

```
  5%
TRUE
```

What does the entropy measure tell us?

- Comparable to the random responding test, we can see whether participants' responses significantly differ from what would be expected under random responding (uniformity). This essentially validates the c'-Analysis with a different method.
- In addition, entropy quantifies the amount of information that is already contained in our distribution. Less information = participants are more 'certain' in their choices. The fact that (for this participant at least), we get a value that is moderately smaller than what would be under uniformity is reassuring, because it tells us that participants do have pronounced preference profiles and this analysis is worth conducting.
- Unlike c', the entropy also gives an individual-difference measure of how 'strong' the preference profile is. Larger entropy = less strongly held preferences.

To get a version of this that is comparable across individuals, a few changes are needed (noting that H_max will differ if participants have different numbers of items in their questionnaire:

$$Score = 100 * (1 - \frac{H_{participant}}{H_{max}(k_{items})})$$

```r
# first, standardise by dividing by the maximum entropy
# this gives a number between 0 and 1
# flip, so larger numbers indicate a more pronounced preference profile.
entropy_score <- 100 * (1 - entropy / max_entropy)
entropy_score
```

```
[1] 6.908622
```

Finally, validate with simulation of random participants:

```r
n_participants <- 10
n_items = 36

dat_sim <- matrix(ncol = 3, nrow = 0) %>% as.data.frame()
colnames(dat_sim) <- c("participant", "item", "wins")

for (subj in 1:n_participants) {

    these_wins <- data.frame( # list storing wins by item index
      item = paste("i", 1:n_items, sep = ""),
      wins = rep(0, n_items)
    )

    for (i in 1:total_wins) {
      index <- runif(1, min = 0, max = n_items) %>% round()
      these_wins$wins[index] <- these_wins$wins[index] + 1
    }

    dat_sim <- rbind(dat_sim, data.frame(participant = rep(subj, n_items), item = paste("i",
  }



res_entropy <- matrix(ncol = 7, nrow = 0) %>% as.data.frame()
colnames(res_entropy) <- c("participant", "n_items", "h", "h_threshold", "h_max", "sig", "sc

n_permutations <- 200

counter <- 0

for (subj in unique(dat_sim$participant)) {

  counter <- counter + 1
  print(paste("Computing participant no. ", counter, sep = ""))

  this <- dat_sim %>%
    filter(participant == subj)

  total_wins = this$wins %>% sum()

  this <- this %>%
    mutate(
```

```
    p = wins / total_wins,
    p = ifelse(p == 0, 0.000000001, p)
  ) %>%
  mutate(
    h = p * log(p, base = 2)
  )
entropy <- -sum(this$h)

uniform_entropy <- c()

# create a permuted distribution
for (perm in 1:n_permutations) {

  these_wins <- data.frame( # list storing wins by item index
    item = unique(this$item),
    wins = rep(0, length(unique(this$item)))
  )

  for (i in 1:total_wins) {
    index <- runif(1, min = 0, max = length(unique(this$item))) %>% round()
    these_wins$wins[index] <- these_wins$wins[index] + 1
  }

  these_wins <- these_wins %>%
    mutate(
      p = wins / total_wins,
    ) %>%
    filter(p > 0) %>%
    mutate(
      h = p * log(p, base = 2)
    )
  uniform_entropy <- c(uniform_entropy, -sum(these_wins$h))
}

# get quantiles for a significance threshold
threshold <- quantile(uniform_entropy, probs = c(0.05))

#compute significance
sig <- entropy < threshold

# compute maximal entropy (uniform distribution)
p <- 1 / length(unique(this$item))
```

```
  h_max <- -length(unique(this$item)) * p * log(p, base = 2)

  # save results
  res_entropy <- res_entropy %>%
    add_row(
      participant = subj,
      n_items = length(unique(this$item)),
      h = entropy,
      h_threshold = threshold,
      sig = sig,
      h_max = h_max,
      score = 100 * (1 - entropy / h_max)
    )
}
```

```
[1] "Computing participant no. 1"
[1] "Computing participant no. 2"
[1] "Computing participant no. 3"
[1] "Computing participant no. 4"
[1] "Computing participant no. 5"
[1] "Computing participant no. 6"
[1] "Computing participant no. 7"
[1] "Computing participant no. 8"
[1] "Computing participant no. 9"
[1] "Computing participant no. 10"
```

```
# for how many participants did this return a significant result?
sum(res_entropy$sig) / nrow(res_entropy)
```

```
[1] 0
```

```
# looks like a rate of 0%. Even less than the 5% we would have expected to find. This shows t
```

Test-retest: Aitchison's Distance

This is an idea from van Eijnatten et al. (2015), concerning the use of Aitchison's distance. AD is a metric which allows us to compare how similar/different two profiles on an (ipsative) preference score are.

The formula is the following, where X and Y are separate preference profiles over n items:

$$d(X,Y) = \sqrt{\frac{1}{2n}\sum_{a}\sum_{b}(ln\frac{X_a}{X_b} - ln\frac{Y_a}{Y_b})^2}$$

In words, this is the square-root of the sum of squares of the difference in log-ratio preferences for each item pair. Here, X stands for wave 1 and Y for wave 2. So, we want to see whether the log-ratio of ranks between item a and item b is different in wave 1 and 2 (AD increases) or the same (AD does not increase). For a perfect fit, this value would approach 0. The idea is to run a permutation test, testing if the AD-value that is actually observed is smaller than the value that would be observed if we permute the values from the 2nd wave (equivalent to no true relationship = larger distance expected).

Get Distance for real data

```
# recompute wins & ranks within each participant, but separate by wave

  a <- dat %>%
    group_by(participant, wave, item1) %>%
    summarise(
      wins1 = sum(win1)
    ) %>%
    ungroup() %>%
    dplyr::rename(item = item1)
```

```
`summarise()` has grouped output by 'participant', 'wave'. You can override
using the `.groups` argument.
```

```
  # some entries are missing, because items only appear in the item1 or item2 column
  # for our merge operation below to include these, we need to
  # manually add those entries

  for (w in 1:2) {
    for (subj in unique(dat$participant)){

      item1_list <- filter(dat, participant == subj, wave == w)$item1 %>% unique()
      item2_list <- filter(dat, participant == subj, wave == w)$item2 %>% unique()

      for (i in item2_list[!item2_list %in% item1_list]) {
        a <- a %>% add_row(
          participant = subj,
          wave = w,
```

```
          item = i,
          wins1 = 0
        )
      }
    }
  }

  b <- dat %>%
    group_by(participant, wave, item2) %>%
    summarise(
      wins2 = sum(win2)
    ) %>%
    ungroup() %>%
    dplyr::rename(item = item2)
```

```
`summarise()` has grouped output by 'participant', 'wave'. You can override
using the `.groups` argument.
```

```
  for (w in 1:2) {
    for (subj in unique(dat$participant)){

      item1_list <- filter(dat, participant == subj, wave == w)$item1 %>% unique()
      item2_list <- filter(dat, participant == subj, wave == w)$item2 %>% unique()

      for (i in item1_list[!item1_list %in% item2_list]) {
        b <- b %>%
          add_row(
            participant = subj,
            wave = w,
            item = i,
            wins2 = 0
        )
      }
    }
  }

# compute ranks within each participant
  wins_participant_wave <- merge(a, b) %>%
    mutate(wins = wins1 + wins2) %>%
    arrange(participant) %>%
    group_by(participant)
```

```r
res_AD <- matrix(nrow = 0, ncol = 5) %>% as.data.frame()
colnames(res_AD) <- c(
  "participant", # which participant
  "shared",      # how many items are shared between the two waves (i.e., relevant for AD)
  "total_w1",    # how many items in wave 1
  "total_w2",    # how many items in wave 2
  "AD"           # measured Aitchison's Distance
)

n_participants <- 0

for (p in unique(dat$participant)){

  dat1<- wins_participant_wave %>%
    filter(
      wave == 1,
      participant == p
    )

  dat2 <- wins_participant_wave %>%
    filter(
      wave == 2,
      participant == p
    )

  # in both cases, the contests were based on the current selection of items participants enc
  # i.e., this list may change
  # thus, we cannot compute the aichison's distance over all those items. Instead, use only t

  # How much overlap was there in selected items?

  # Items from one that are also in 2
  shared_items <- unique(dat1$item)[unique(dat1$item) %in% unique(dat2$item)]

  shared <- length(shared_items)

  # how many items in total in 1?
  total_w1 <- unique(dat1$item) %>% length()

  # how many items in total in 2?
  total_w2 <- unique(dat2$item) %>% length()
```

```r
# select only shared items

dat1 <- dat1 %>% filter(item %in% shared_items) %>% mutate(rank = rank(wins))
dat2 <- dat2 %>% filter(item %in% shared_items) %>% mutate(rank = rank(wins))

log_ratio1 <- c()
log_ratio2 <- c()
distance <- c()

remaining_players <- shared_items[2:length(shared_items)]

# now get the log ratios of ranks

for (i in shared_items) {
  for (j in remaining_players) {

    # note, we only compute the distance for the upper triangle of the item-item matrix, th
    if (i != j) {

      rank1i <- filter(dat1, item == i)$rank
      rank2i <- filter(dat2, item == i)$rank
      rank1j <- filter(dat1, item == j)$rank
      rank2j <- filter(dat2, item == j)$rank

      l1 <- log(rank1i / rank1j)
      l2 <- log(rank2i / rank2j)

      log_ratio1 <- append(log_ratio1, l1)
      log_ratio2 <- append(log_ratio2, l2)
    }
  # don't repeat items that already fought every other item.
  remaining_players = remaining_players[remaining_players != i]
  }
}

# compute the aitchison distance over the log ratios.

AD = sqrt(1/shared * sum((log_ratio1 - log_ratio2)^2))

# save info

res_AD <- res_AD %>%
```

```
    add_row(
      participant = p,
      shared = shared,
      total_w1 = total_w1,
      total_w2 = total_w2,
      AD = AD
    )
   n_participants <- n_participants + 1
   print(paste(n_participants, " out of ", length(unique(dat$participant)), " done!", sep = "
}
```

```
[1] "1 out of 20 done!"
[1] "2 out of 20 done!"
[1] "3 out of 20 done!"
[1] "4 out of 20 done!"
[1] "5 out of 20 done!"
[1] "6 out of 20 done!"
[1] "7 out of 20 done!"
[1] "8 out of 20 done!"
[1] "9 out of 20 done!"
[1] "10 out of 20 done!"
[1] "11 out of 20 done!"
[1] "12 out of 20 done!"
[1] "13 out of 20 done!"
[1] "14 out of 20 done!"
[1] "15 out of 20 done!"
[1] "16 out of 20 done!"
[1] "17 out of 20 done!"
[1] "18 out of 20 done!"
[1] "19 out of 20 done!"
[1] "20 out of 20 done!"
```

```
res_AD
```

```
   participant shared total_w1 total_w2       AD
1        10542     21       27       25 2.750445
2        10544     24       27       25 3.251341
3        10548     20       38       20 2.323717
4        10550      0       31        0      NaN
5        10556     34       36       43 3.458202
6        10562     43       44       43 4.666815
```

```
7        10563     32       34       37 3.411711
8        10572     18       30       20 2.609925
9        10581      0       33        0      NaN
10       10593     43       43       49 4.482542
11       10602     38       38       43 3.539673
12       10618     19       26       21 2.083697
13       10619     43       44       43 4.255683
14       10620     36       38       42 2.924042
15       10631     39       39       40 3.652732
16       10636     23       26       26 3.114480
17       10637     38       41       40 4.443920
18       10647     22       31       30 4.042734
19       10655     38       39       43 4.238526
20       10662     31       35       35 3.071479
```

Construct a permutation test

```
  # next, for each participant, compute an individualised permuted version
  # our null hypothesis would be that there is no relationship between responses at time 1 a

# very few permutations so the script compiles in a reasonable timeframe
n_permutations = 5

res_AD_perm = matrix(ncol = n_permutations + 1, nrow = 0)

n_participants <- 0

# exclude participants who don't have any shared values to work with
for (p in filter(res_AD, shared != 0)$participant) {

    dat1<- wins_participant_wave %>%
    filter(
      wave == 1,
      participant == p
    )

    dat2 <- wins_participant_wave %>%
      filter(
        wave == 2,
        participant == p
      )
```

```r
# select only shared items
shared_items <- unique(dat1$item)[unique(dat1$item) %in% unique(dat2$item)]
dat1 <- dat1 %>% filter(item %in% shared_items) %>% mutate(rank = rank(wins))
dat2 <- dat2 %>% filter(item %in% shared_items) %>% mutate(rank = rank(wins))

AD <- c()

# now permute
for (perm in 1:n_permutations) {

  # randomly shuffle the order of rankings in both sets
  # this generates a null-distribution where the ranks in our data are completely unrelat
  dat1$rank <- sample(dat1$rank, nrow(dat1), replace = TRUE)
  dat2$rank <- sample(dat2$rank, nrow(dat2), replace = TRUE)


  log_ratio1 <- c()
  log_ratio2 <- c()
  distance <- c()

  remaining_players <- shared_items[2:length(shared_items)]

  # now get the log ratios of ranks

  for (i in shared_items) {
    for (j in remaining_players) {

      if (i != j) {

        rank1i <- filter(dat1, item == i)$rank
        rank2i <- filter(dat2, item == i)$rank
        rank1j <- filter(dat1, item == j)$rank
        rank2j <- filter(dat2, item == j)$rank

        l1 <- log(rank1i / rank1j)
        l2 <- log(rank2i / rank2j)

        log_ratio1 <- append(log_ratio1, l1)
        log_ratio2 <- append(log_ratio2, l2)
      }
    # don't repeat items that already fought every other item.
    remaining_players = remaining_players[remaining_players != i]
```

```
      }
    }

    # compute the aitchison distance over the log ratios.
    # note that we only computed the upper triangle of the matrix.
    # so no need to halve the value as in the formula above.

    this_AD = sqrt(1/(shared) * sum((log_ratio1 - log_ratio2)^2))

    # save results
    AD <- c(AD, this_AD)

  }

  # save results
  res_AD_perm <- rbind(res_AD_perm, c(p, AD))

  # print progress

  n_participants <- n_participants + 1
  print(paste(n_participants, " out of ", length(filter(res_AD, shared != 0)$participant),

}
```

```
[1] "1 out of 18 done!"
[1] "2 out of 18 done!"
[1] "3 out of 18 done!"
[1] "4 out of 18 done!"
[1] "5 out of 18 done!"
[1] "6 out of 18 done!"
[1] "7 out of 18 done!"
[1] "8 out of 18 done!"
[1] "9 out of 18 done!"
[1] "10 out of 18 done!"
[1] "11 out of 18 done!"
[1] "12 out of 18 done!"
[1] "13 out of 18 done!"
[1] "14 out of 18 done!"
[1] "15 out of 18 done!"
[1] "16 out of 18 done!"
[1] "17 out of 18 done!"
[1] "18 out of 18 done!"
```

```r
colnames(res_AD_perm) <- c("participant", paste("perm", 1:n_permutations, sep = ""))

# save, so we don't have to run it all the time
saveRDS(res_AD_perm, file = "distance_permuted_pilot.rds")

res_AD_perm
```

```
      participant    perm1    perm2    perm3    perm4    perm5
 [1,]       10542 4.221816 3.562020 3.819325 3.815194 3.385884
 [2,]       10544 6.750003 5.880072 5.858241 6.892687 4.603126
 [3,]       10548 5.482325 3.596456 3.561366 1.817948 2.472217
 [4,]       10556 5.881446 5.262153 4.568907 3.020778 3.442292
 [5,]       10562 6.948643 6.279619 8.279522 6.916551 5.771189
 [6,]       10563 6.672417 7.947116 5.796294 6.760911 6.384547
 [7,]       10572 3.473389 2.683620 3.187131 3.150752 3.651389
 [8,]       10593 8.853062 8.542351 8.523729 7.834403 8.825929
 [9,]       10602 8.873237 7.213120 6.677509 7.094899 7.994186
[10,]       10618 2.904213 3.564903 4.187885 3.891423 3.997888
[11,]       10619 7.869864 7.460954 6.574387 6.402368 7.447343
[12,]       10620 6.685057 6.715796 6.259688 7.123547 7.120057
[13,]       10631 7.383619 7.603279 6.816259 6.675497 5.849574
[14,]       10636 3.917150 4.157592 5.988588 5.452136 4.956625
[15,]       10637 8.988539 9.260795 8.394173 7.480907 8.002271
[16,]       10647 3.752225 3.611908 3.066380 3.141540 2.286977
[17,]       10655 7.602498 8.681045 9.708332 7.066137 7.858833
[18,]       10662 6.559445 5.887190 5.426387 6.576016 6.597672
```

Compare results

We want to get p-values for each participant, testing whether their ADs are significantly lower than what has been obtained under a full switch. So, for each participant, get the Aichison's Distance at the lower 5% quantile (one-sided test), and compare to the actually observed distance. Here, we see some 77% significant, which is good.

```r
# load in data, to save compiling time when needed
#res_AD_perm <- readRDS(file = paste(getwd(), "/distance_permuted_pilot.rds", sep = ""))

res_AD_total <- cbind(res_AD_perm, dplyr::select(filter(res_AD, shared != 0), -participant))
threshold <- c()
sig <- c()
```

```r
for (i in 1:nrow(res_AD_total)) {
  t <- quantile(res_AD_perm[i, 2:(n_permutations + 1)], probs = 0.05, na.rm = FALSE)
  print(t)
  threshold <- c(threshold, print(t))
}
```

```
      5%
3.421111
      5%
3.421111
      5%
4.854149
      5%
4.854149
      5%
1.948802
      5%
1.948802
      5%
3.105081
      5%
3.105081
      5%
5.872875
      5%
5.872875
      5%
5.913945
      5%
5.913945
      5%
2.777046
      5%
2.777046
      5%
7.972269
      5%
7.972269
      5%
6.760987
      5%
6.760987
```

```
      5%
3.036351
      5%
3.036351
      5%
6.436772
      5%
6.436772
      5%
6.344762
      5%
6.344762
      5%
6.014758
      5%
6.014758
      5%
3.965238
      5%
3.965238
     5%
7.58518
     5%
7.58518
      5%
2.442858
      5%
2.442858
      5%
7.173409
      5%
7.173409
      5%
5.518548
      5%
5.518548
```

```r
res_AD_total$threshold <- threshold

res_AD_total <- res_AD_total %>%
  mutate(
    sig = ifelse(AD > threshold, 0, 1)
```

```
  )

# get proportion of significant responses
sum(res_AD_total$sig) / nrow(res_AD_total)
```

```
[1] 0.8333333
```

Test-retest: Ranking of core symptoms

We can also conduct a test-retest analysis on core symptoms only. Since participants are free in the amount of wins they give to core symptoms versus other symptoms (and our analysis is blind to that information), the number of wins is no longer bound by a sum constraint and traditional correlation methods are appropriate.

```
core_items <- c("i5", "i14", "i32", "i33", "i34")

wins_core <- wins_participant_wave %>%
  filter(item %in% core_items) %>%
  dplyr::select(-wins1, -wins2) %>%
  pivot_wider(names_from = wave, values_from = wins, names_prefix = "wins_w")

res_core <- list(
  item = c(),
  r = c(),
  p = c()
)

for (i in core_items) {
  this <- filter(wins_core, item == i)
  a <- cor.test(this$wins_w1, this$wins_w2, method = "pearson")

  res_core$item <- c(res_core$item, i)
  res_core$r <- c(res_core$r, round(a$estimate, 3))
  res_core$p <- c(res_core$p, round(a$p.value, 3))
}
```

Validity

Agreement with Likert-Items

Not all items feature at both timepoints, or have been shown to every participant. To deal with this, focus only on core items, or use within-participant rank differences.

BTM: Agreement with Core-items

```
# for now only consider wave 1

# every value you put into BTM needs to be a factor
dat2 <- dat

dat2$participant <- as.factor(dat2$participant)

# create an array storing participant-level information

dat_subj <- dat_filter %>%
  filter(wave == 1) %>%
  dplyr::select(participant, item_id, severity_response, frequency_response, impact_response
  pivot_wider(names_from = item_id, values_from = c(severity_response, frequency_response, i

dat_BTM <- list(
  preferences = dat2,
  participants = dplyr::select(dat_subj, -participant),
  items = diag(52)
)

rownames(dat_BTM$item) <- c(paste("i", 1:52, sep = "")) %>% as.factor()
colnames(dat_BTM$item) <- c(paste("i", 1:52, sep = "")) %>% as.factor()

# run the model
# note that it CANNOT deal with missingness in the subject-level variables (i.e., filter res
this_model <- BTm(outcome = cbind(win1, win2), formula = ~ item + i5[item] * severity_respons

summary(this_model)
```

```
Call:
BTm(outcome = cbind(win1, win2), player1 = item1, player2 = item2,
```

```
    formula = ~item + i5[item] * severity_response_5[participant] +
        i14[item] * severity_response_14[participant] + i32[item] *
        severity_response_32[participant] + i33[item] * severity_response_33[participant] +
        i34[item] * severity_response_34[participant], id = "item",
    refcat = "i52", data = dat_BTM, na.action = na.omit)

Coefficients: (10 not defined because of singularities)
                                            Estimate Std. Error z value
itemi1                                     0.1899240  0.1595111   1.191
itemi2                                    -0.7058737  0.1920779  -3.675
itemi3                                     0.0005487  0.1643547   0.003
itemi4                                     0.1852324  0.1681884   1.101
itemi5                                    -0.9293242  0.3925089  -2.368
itemi6                                     0.1753336  0.1569457   1.117
itemi7                                    -0.0291817  0.1800493  -0.162
itemi8                                     0.1115331  0.1661167   0.671
itemi9                                    -0.5592381  0.1723228  -3.245
itemi10                                   -1.6978117  0.2153306  -7.885
itemi11                                   -0.0207835  0.1665330  -0.125
itemi12                                   -0.8574256  0.1810722  -4.735
itemi13                                   -0.5987375  0.2074723  -2.886
itemi14                                   -1.5262520  0.3287168  -4.643
itemi15                                   -0.6630981  0.1711445  -3.874
itemi16                                    0.2716075  0.1600118   1.697
itemi17                                   -0.7784712  0.1723113  -4.518
itemi18                                   -0.0106478  0.1743835  -0.061
itemi19                                   -1.3084977  0.1946461  -6.722
itemi20                                   -0.7715507  0.1860461  -4.147
itemi21                                    0.3608646  0.1845451   1.955
itemi22                                    0.0161897  0.1923668   0.084
itemi23                                   -0.0623545  0.1613178  -0.387
itemi24                                   -0.1441057  0.1741933  -0.827
itemi25                                    0.2247458  0.1676844   1.340
itemi26                                   -0.3530015  0.1763845  -2.001
itemi27                                   -0.6030203  0.2704555  -2.230
itemi28                                    0.3514013  0.1684777   2.086
itemi29                                   -0.6124726  0.1965125  -3.117
itemi30                                   -0.4278941  0.1963421  -2.179
itemi31                                   -0.0646597  0.1551798  -0.417
itemi32                                   -1.5308229  0.3043736  -5.029
itemi33                                   -1.8874771  0.2988894  -6.315
itemi34                                   -1.6926939  0.2957168  -5.724
itemi35                                    0.1461685  0.1593806   0.917
```

```
itemi36                                          -0.0201619  0.1581004  -0.128
itemi37                                          -0.2492746  0.1848444  -1.349
itemi38                                          -2.0715399  0.2672561  -7.751
itemi39                                          -0.3511248  0.1866737  -1.881
itemi40                                          -0.4171038  0.1700538  -2.453
itemi41                                          -1.5208131  0.2233970  -6.808
itemi42                                          -0.4142811  0.3157043  -1.312
itemi43                                          -0.9453848  0.2198335  -4.300
itemi44                                          -0.4285564  0.1630534  -2.628
itemi45                                           0.1061208  0.1613843   0.658
itemi46                                          -0.3143563  0.1962651  -1.602
itemi47                                           0.3946546  0.1786269   2.209
itemi48                                           0.0667595  0.1867753   0.357
itemi49                                           0.3630256  0.1633266   2.223
itemi50                                           0.4020561  0.1600799   2.512
itemi51                                           0.3196890  0.1626446   1.966
i5[item]                                                 NA         NA      NA
severity_response_5[participant]                         NA         NA      NA
i14[item]                                                NA         NA      NA
severity_response_14[participant]                        NA         NA      NA
i32[item]                                                NA         NA      NA
severity_response_32[participant]                        NA         NA      NA
i33[item]                                                NA         NA      NA
severity_response_33[participant]                        NA         NA      NA
i34[item]                                                NA         NA      NA
severity_response_34[participant]                        NA         NA      NA
i5[item]:severity_response_5[participant]         0.4533865  0.1091155   4.155
i14[item]:severity_response_14[participant]       0.4346946  0.0876458   4.960
i32[item]:severity_response_32[participant]       0.2677083  0.0938579   2.852
i33[item]:severity_response_33[participant]       0.6284580  0.0778645   8.071
i34[item]:severity_response_34[participant]       0.5902520  0.0765923   7.706
                                                 Pr(>|z|)
itemi1                                           0.233786
itemi2                                           0.000238 ***
itemi3                                           0.997336
itemi4                                           0.270749
itemi5                                           0.017901 *
itemi6                                           0.263926
itemi7                                           0.871246
itemi8                                           0.501957
itemi9                                           0.001173 **
itemi10                                          3.15e-15 ***
itemi11                                          0.900681
```

```
itemi12                                  2.19e-06 ***
itemi13                                  0.003903 **
itemi14                                  3.43e-06 ***
itemi15                                  0.000107 ***
itemi16                                  0.089617 .
itemi17                                  6.25e-06 ***
itemi18                                  0.951312
itemi19                                  1.79e-11 ***
itemi20                                  3.37e-05 ***
itemi21                                  0.050533 .
itemi22                                  0.932929
itemi23                                  0.699103
itemi24                                  0.408081
itemi25                                  0.180151
itemi26                                  0.045358 *
itemi27                                  0.025771 *
itemi28                                  0.037002 *
itemi29                                  0.001829 **
itemi30                                  0.029307 *
itemi31                                  0.676915
itemi32                                  4.92e-07 ***
itemi33                                  2.70e-10 ***
itemi34                                  1.04e-08 ***
itemi35                                  0.359088
itemi36                                  0.898524
itemi37                                  0.177477
itemi38                                  9.11e-15 ***
itemi39                                  0.059978 .
itemi40                                  0.014176 *
itemi41                                  9.92e-12 ***
itemi42                                  0.189438
itemi43                                  1.70e-05 ***
itemi44                                  0.008581 **
itemi45                                  0.510817
itemi46                                  0.109224
itemi47                                  0.027148 *
itemi48                                  0.720768
itemi49                                  0.026236 *
itemi50                                  0.012019 *
itemi51                                  0.049349 *
i5[item]                                       NA
severity_response_5[participant]               NA
i14[item]                                      NA
```

```
severity_response_14[participant]                  NA
i32[item]                                          NA
severity_response_32[participant]                  NA
i33[item]                                          NA
severity_response_33[participant]                  NA
i34[item]                                          NA
severity_response_34[participant]                  NA
i5[item]:severity_response_5[participant]   3.25e-05 ***
i14[item]:severity_response_14[participant] 7.06e-07 ***
i32[item]:severity_response_32[participant] 0.004341 **
i33[item]:severity_response_33[participant] 6.96e-16 ***
i34[item]:severity_response_34[participant] 1.29e-14 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 10009  on 7220  degrees of freedom
Residual deviance:  9037  on 7164  degrees of freedom
AIC: 9149

Number of Fisher Scoring iterations: 4
```

LME: Correlate core items and rank/number of wins.

Again, since the number of wins in core items is unbounded, traditional linear models are appropriate, assuming assumptions are met overall. Here, fit an LME to the number of wins, predicted from likert-item agreement.

Note a problem: Since there was a random selection of item pairings, the raw number of wins is biased - if item 1 has only been shown 5 times and item 15 100 times, item 1 can never have as many wins. The BTM (technically) overcomes this limitation and should be more trustworthy. However, run both analyses anyway.

```r
# recompute wins & ranks within each participant, but separate by wave

  a <- dat %>%
    group_by(participant, wave, item1) %>%
    summarise(
      wins1 = sum(win1)
    ) %>%
    ungroup() %>%
    dplyr::rename(item = item1)
```

`summarise()` has grouped output by 'participant', 'wave'. You can override using the `.groups` argument.

```
  # some entries are missing, because items only appear in one row
  # for our merge operation below to work, we need to fix this
  # manually add those entries

  for (w in 1:2) {
    for (subj in unique(dat$participant)){

      item1_list <- filter(dat, participant == subj, wave == w)$item1 %>% unique()
      item2_list <- filter(dat, participant == subj, wave == w)$item2 %>% unique()

      for (i in item2_list[!item2_list %in% item1_list]) {
        a <- a %>% add_row(
          participant = subj,
          wave = w,
          item = i,
          wins1 = 0
        )
      }
    }
  }

  b <- dat %>%
    group_by(participant, wave, item2) %>%
    summarise(
      wins2 = sum(win2)
    ) %>%
    ungroup() %>%
    dplyr::rename(item = item2)
```

`summarise()` has grouped output by 'participant', 'wave'. You can override using the `.groups` argument.

```
  for (w in 1:2) {
    for (subj in unique(dat$participant)){

      item1_list <- filter(dat, participant == subj, wave == w)$item1 %>% unique()
      item2_list <- filter(dat, participant == subj, wave == w)$item2 %>% unique()

      for (i in item1_list[!item1_list %in% item2_list]) {
```

```
        b <- b %>%
          add_row(
            participant = subj,
            wave = w,
            item = i,
            wins2 = 0
        )
      }
    }
  }

# compute ranks within each participant, considering only core items.
  this_dat <- merge(a, b) %>%
    filter(item %in% core_items) %>%
    mutate(wins = wins1 + wins2) %>%
    arrange(participant) %>%
    group_by(participant, wave) %>%
    mutate(
      rank = rank(wins)
    )

this_filter <- dat_filter %>%
  mutate(
    item = paste("i", item_id, sep = "")
  ) %>%
  filter(
    item %in% core_items,
    wave == 1
    ) %>%
  mutate(
    item = item %>% factor(levels = core_items),
  ) %>%
  dplyr::select(participant, item, severity_response) %>%
  dplyr::rename(
    filter_response = severity_response
  )

dat_agreement <- merge(this_dat, this_filter)

# now we see that the number of wins is predicted by the response to the filter item (for nov
model_agree <- lm(data = dat_agreement, formula = wins ~ filter_response)
```

```
# include random intercepts

model_agree_int <- lme(dat = dat_agreement, fixed = wins ~ filter_response, random = ~ 1|part

# cannot include random effects, since we only have 2 timepoints. Try an interaction of filte
model_agree_int <- lme(dat = dat_agreement, fixed = wins ~ filter_response, random = ~ 1|part

# compare models

# test inclusion of random intercepts
test <- lrtest(model_agree, model_agree_int)
```

```
Warning in modelUpdate(objects[[i - 1]], objects[[i]]): original model was of
class "lm", updated model is of class "lme"
```

```
# adjust, since we compare a lm to a lme object
log_likelihood <- test[2, 2] + test[1, 2]
chi_sq <- 2 * log_likelihood
p <- dchisq(chi_sq, 1)

# the random intercepts model is strongly preferred.
```

LME: Rank difference and Filter-Difference

First, compute rank difference (looking only at wave 1 for now.

```
# recompute wins & ranks within each participant, but separate by wave

  a <- dat %>%
    group_by(participant, wave, item1) %>%
    summarise(
      wins1 = sum(win1)
    ) %>%
    ungroup() %>%
    dplyr::rename(item = item1)
```

```
`summarise()` has grouped output by 'participant', 'wave'. You can override
using the `.groups` argument.
```

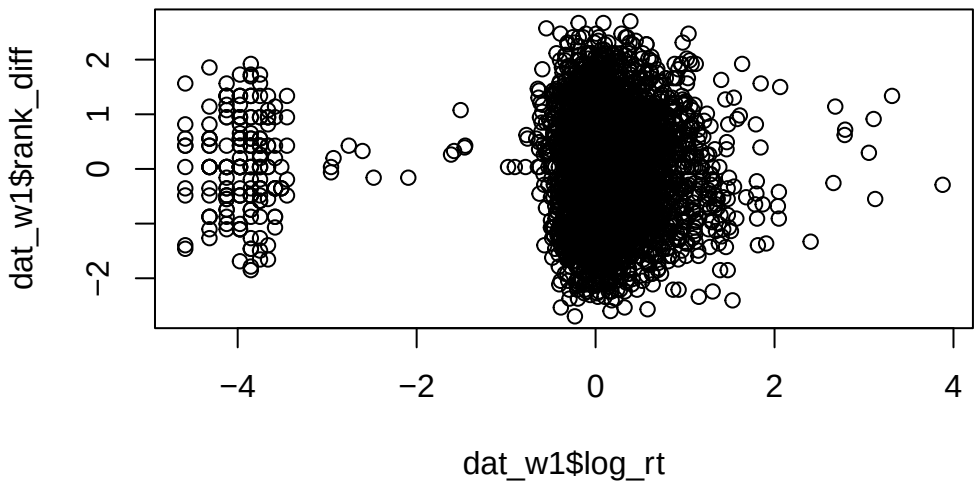
```
  # some entries are missing, because items only appear in one row
  # for our merge operation below to work, we need to fix this
  # manually add those entries

  for (w in 1:2) {
    for (subj in unique(dat$participant)){

      item1_list <- filter(dat, participant == subj, wave == w)$item1 %>% unique()
      item2_list <- filter(dat, participant == subj, wave == w)$item2 %>% unique()

      for (i in item2_list[!item2_list %in% item1_list]) {
        a <- a %>% add_row(
          participant = subj,
          wave = w,
          item = i,
          wins1 = 0
        )
      }
    }
  }

  b <- dat %>%
    group_by(participant, wave, item2) %>%
    summarise(
      wins2 = sum(win2)
    ) %>%
    ungroup() %>%
    dplyr::rename(item = item2)
```

```
`summarise()` has grouped output by 'participant', 'wave'. You can override
using the `.groups` argument.
```

```
  for (w in 1:2) {
    for (subj in unique(dat$participant)){

      item1_list <- filter(dat, participant == subj, wave == w)$item1 %>% unique()
      item2_list <- filter(dat, participant == subj, wave == w)$item2 %>% unique()

      for (i in item1_list[!item1_list %in% item2_list]) {
        b <- b %>%
          add_row(
```

```
            participant = subj,
            wave = w,
            item = i,
            wins2 = 0
        )
      }
    }
  }

# compute ranks within each participant
  wins_participant_wave <- merge(a, b) %>%
    mutate(wins = wins1 + wins2) %>%
    arrange(participant) %>%
    group_by(participant)

dat_w1 <- dat %>%
  filter(
    wave == 1
  ) %>%
  mutate(
    item1_rank = NA,
    item2_rank = NA
  )

params <- list(
  participant = c(),
  item = c(),
  alpha = c(),
  rank = c(),
  p = c(),
  se = c()
)

for (subj in unique(dat$participant)) {
  # pick a participant
  this_dat <- dat_w1 %>%
    filter(participant == subj)

  wins <- wins_participant_wave %>%
    filter(wave == 1,
           participant == subj)
  wins$rank <- rank(wins$wins)
```

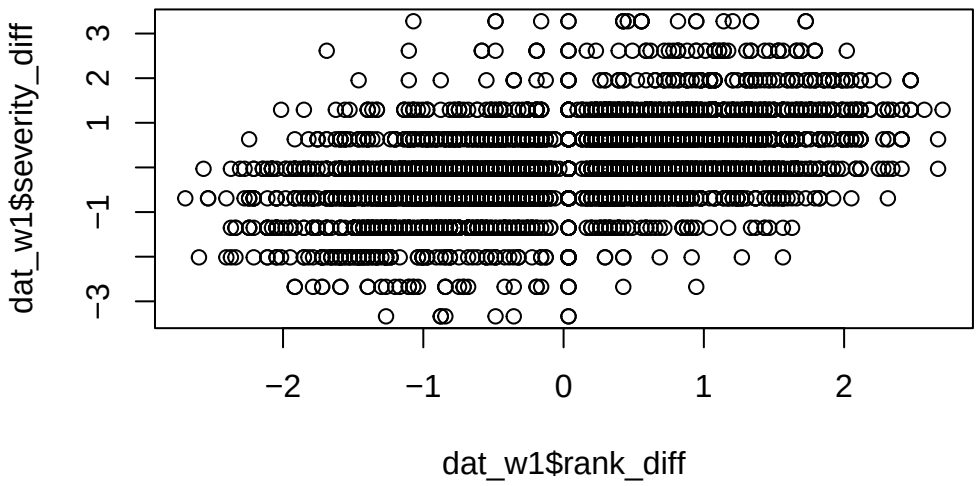
```
  wins <- wins %>% arrange(item)

  # save parameters in a list for checking
  params$rank <- c(params$rank, wins$rank)
  params$item <- c(params$item, wins$item)
  params$participant <- c(params$participant, rep(subj, nrow(wins)))
  params$wins <- c(params$wins, wins$wins)

  # record params in the actual dataset
  for (i in wins$item){
    dat_w1 <- dat_w1 %>%
      mutate(

        item1_rank = ifelse(item1 == i & participant == subj, wins[wins$item == i,]$rank, ite
        item2_rank = ifelse(item2 == i & participant == subj, wins[wins$item == i,]$rank, ite
      )
  }
}

    # Now, standardise values for ease of interpretation
    dat_w1 <- dat_w1 %>%
      mutate(
        rank_diff_raw = item1_rank - item2_rank,
        log_rt_raw = log_rt,
        log_rt = scale(log_rt_raw),
        rank_diff = scale(rank_diff_raw)
      )


  plot(dat_w1$log_rt, dat_w1$rank_diff)
```



Now, add in the filter items.

```
this_dat <- matrix(ncol = 6, nrow = 0) %>% as.data.frame()
colnames(this_dat) <- c("severity_response_1", "severity_response_2", "frequency_response_1"

dat_filter2 <- dat_filter %>%
  filter(wave == 1)

for (i in 1:nrow(dat_w1)) {

  p <- dat$participant[i]
  item1 <- sub("i", "", dat$item1[i])
  item2 <- sub("i", "", dat$item2[i])

  s1 <- dat_filter2 %>%
    filter(participant == p, item_id == item1) %>%
    dplyr::select(severity_response)
  s2 <- dat_filter2 %>%
    filter(participant == p, item_id == item2) %>%
    dplyr::select(severity_response)

  f1 <- dat_filter2 %>%
    filter(participant == p, item_id == item1) %>%
```

```
    dplyr::select(frequency_response)
  f2 <- dat_filter2 %>%
    filter(participant == p, item_id == item2) %>%
    dplyr::select(frequency_response)

  i1 <- dat_filter2 %>%
    filter(participant == p, item_id == item1) %>%
    dplyr::select(impact_response)
  i2 <- dat_filter2 %>%
    filter(participant == p, item_id == item2) %>%
    dplyr::select(impact_response)


  this_dat <- this_dat %>%
    add_row(
      severity_response_1 = s1[1, 1],
      severity_response_2 = s2[1, 1],
      frequency_response_1 = f1[1, 1],
      frequency_response_2 = f2[1, 1],
      impact_response_1 = i1[1, 1],
      impact_response_2 = i2[1, 1]
    )
}

this_dat <- this_dat %>%
  mutate(
    severity_diff = scale(severity_response_1 - severity_response_2),
    frequency_diff = scale(frequency_response_1 - frequency_response_2),
    impact_diff = scale(impact_response_1 - impact_response_2)
  )

dat_w1 <- cbind(dat_w1, this_dat)

plot(dat_w1$rank_diff, dat_w1$severity_diff)
```



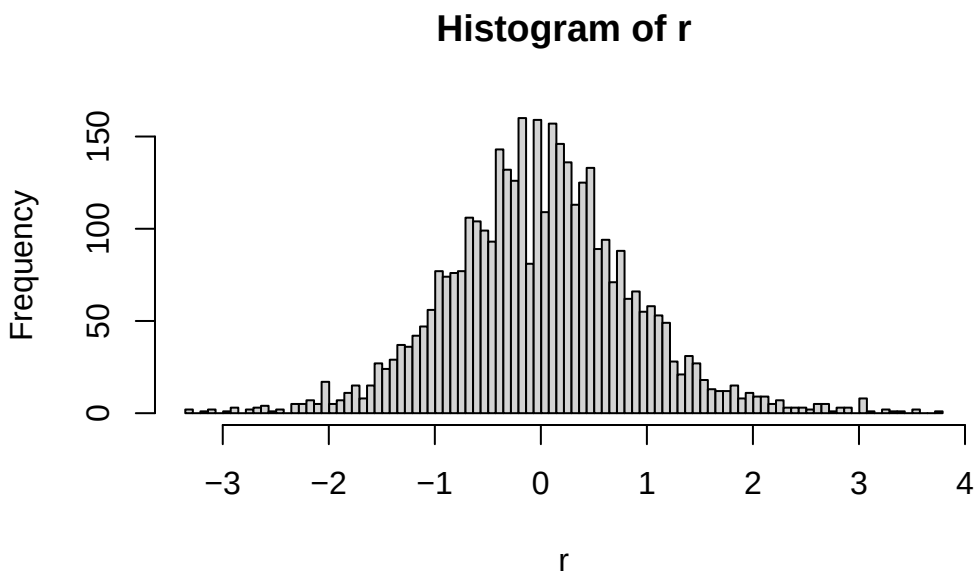
Finally, run the models:

```
# model without any random effects
model_filter <- lm(data = dat_w1, severity_diff ~ rank_diff)


# fit a model including only random intercepts

model_filter_int_only <- lme(data = dat_w1, log_rt ~ rank_diff, random = ~ 1 | participant, r

# fit a model including full random effects, and their correlation

model_filter_full_random <- lme(data = dat_w1, log_rt ~ severity_diff, random = ~ 1 + severit

# test inclusion of random intercepts
test <- lrtest(model_filter,model_filter_int_only)
```

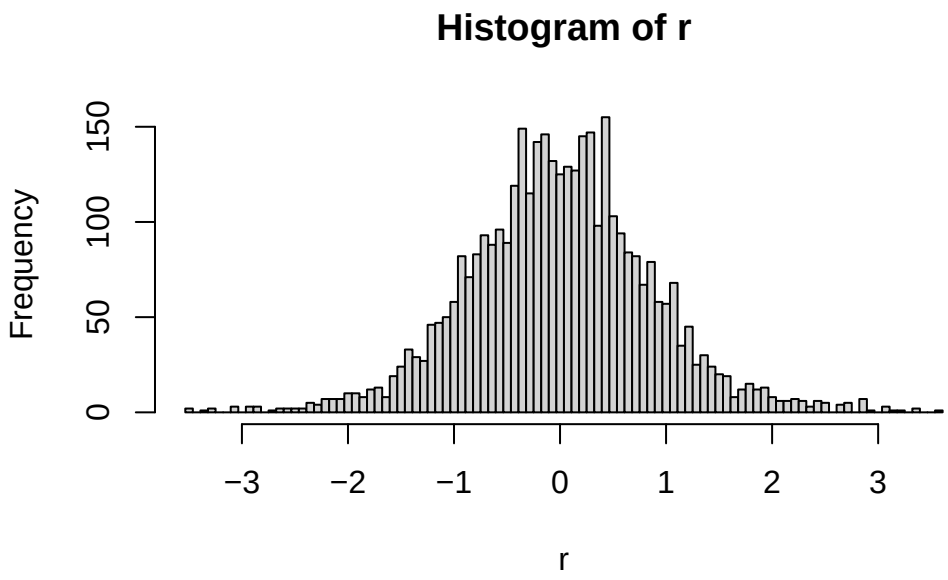
```
Warning in modelUpdate(objects[[i - 1]], objects[[i]]): original model was of
class "lm", updated model is of class "lme"
```

```
# adjust, since we compare a lm to a lme object
log_likelihood <- test[2, 2] + test[1, 2]
```

```r
chi_sq <- 2 * log_likelihood
p <- dchisq(chi_sq, 1)

#one d.f. difference
AIC_diff <- 2 * (1 - log_likelihood)

# test inclusion of random slopes
# adding the random slopes does not seem to improve fit
test2 <- anova(model_filter_full_random, model_filter_int_only)

AIC_diff2 <- summary(model_filter_full_random)$AIC - summary(model_filter_int_only)$AIC

# re-fit final model with REML
model_filter_final <- lme(data = dat_w1, severity_diff ~ rank_diff, random = ~ 1| participant

# plotted residuals at trial level look good.
r <-residuals(model_filter_final, level = 0)
hist(r, breaks = seq(min(r), max(r), length = 101))
```




```r
# plotted residuals at subject level look good, too
r <-residuals(model_filter_final, level = 1)
hist(r, breaks = seq(min(r), max(r), length = 101))
```




```
# look at the model
summary(model_filter_final)
```

```
Linear mixed-effects model fit by REML
  Data: dat_w1
       AIC      BIC    logLik
  9803.624 9828.593 -4897.812

Random effects:
 Formula: ~1 | participant
        (Intercept)  Residual
StdDev:  0.08280595 0.8745738

Fixed effects:  severity_diff ~ rank_diff
                Value  Std.Error   DF  t-value p-value
(Intercept) 0.0000000 0.02332649 3779  0.00000       1
rank_diff   0.4793894 0.01424511 3779 33.65292       0
 Correlation:
          (Intr)
rank_diff 0

Standardized Within-Group Residuals:
```

```
       Min          Q1         Med          Q3         Max
-4.0411259 -0.6190828 -0.0159360  0.5925934  4.1256192

Number of Observations: 3800
Number of Groups: 20
```

RT & Rank Difference

This is a simple LME, designed to estimate the relationship between reaction time and the strength difference between both items.

```
##############################
#####  Do analysis with Ranks#
##############################

# recompute wins & ranks within each participant, but separate by wave

  a <- dat %>%
    group_by(participant, wave, item1) %>%
    summarise(
      wins1 = sum(win1)
    ) %>%
    ungroup() %>%
    dplyr::rename(item = item1)
```

```
`summarise()` has grouped output by 'participant', 'wave'. You can override
using the `.groups` argument.
```

```
  # some entries are missing, because items only appear in one row
  # for our merge operation below to work, we need to fix this
  # manually add those entries

  for (w in 1:2) {
    for (subj in unique(dat$participant)){

      item1_list <- filter(dat, participant == subj, wave == w)$item1 %>% unique()
      item2_list <- filter(dat, participant == subj, wave == w)$item2 %>% unique()

      for (i in item2_list[!item2_list %in% item1_list]) {
        a <- a %>% add_row(
          participant = subj,
```

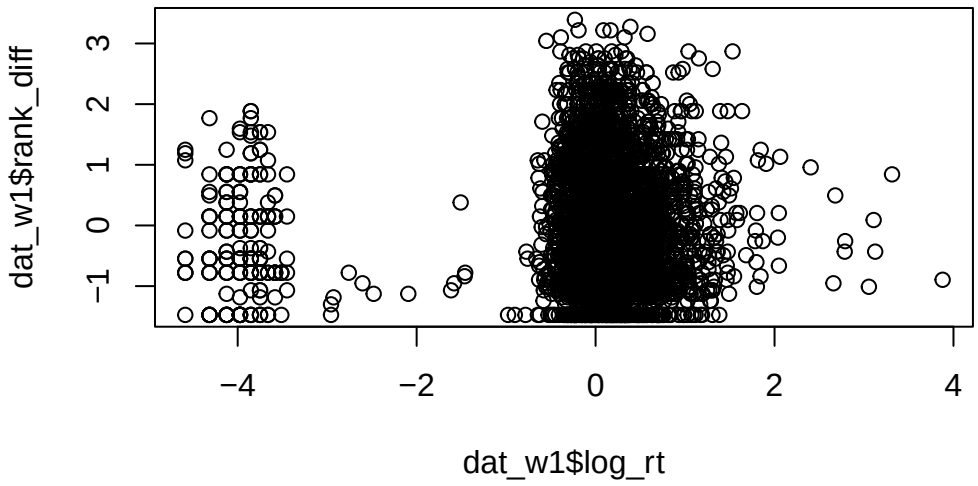
```
          wave = w,
          item = i,
          wins1 = 0
        )
      }
    }
  }

  b <- dat %>%
    group_by(participant, wave, item2) %>%
    summarise(
      wins2 = sum(win2)
    ) %>%
    ungroup() %>%
    dplyr::rename(item = item2)
```

```
`summarise()` has grouped output by 'participant', 'wave'. You can override
using the `.groups` argument.
```

```
  for (w in 1:2) {
    for (subj in unique(dat$participant)){

      item1_list <- filter(dat, participant == subj, wave == w)$item1 %>% unique()
      item2_list <- filter(dat, participant == subj, wave == w)$item2 %>% unique()

      for (i in item1_list[!item1_list %in% item2_list]) {
        b <- b %>%
          add_row(
            participant = subj,
            wave = w,
            item = i,
            wins2 = 0
        )
      }
    }
  }

# compute ranks within each participant
  wins_participant_wave <- merge(a, b) %>%
    mutate(wins = wins1 + wins2) %>%
    arrange(participant) %>%
    group_by(participant)
```

```r
dat_w1 <- dat %>%
  filter(
    wave == 1
  ) %>%
  mutate(
    item1_rank = NA,
    item2_rank = NA
  )

params <- list(
  participant = c(),
  item = c(),
  alpha = c(),
  rank = c(),
  p = c(),
  se = c()
)

for (subj in unique(dat$participant)) {
  # pick a participant
  this_dat <- dat_w1 %>%
    filter(participant == subj)

  wins <- wins_participant_wave %>%
    filter(wave == 1,
           participant == subj)
  wins$rank <- rank(wins$wins)
  wins <- wins %>% arrange(item)

  # save parameters in a list for checking
  params$rank <- c(params$rank, wins$rank)
  params$item <- c(params$item, wins$item)
  params$participant <- c(params$participant, rep(subj, nrow(wins)))
  params$wins <- c(params$wins, wins$wins)

  # record params in the actual dataset
  for (i in wins$item){
    dat_w1 <- dat_w1 %>%
      mutate(

        item1_rank = ifelse(item1 == i & participant == subj, wins[wins$item == i,]$rank, ite
        item2_rank = ifelse(item2 == i & participant == subj, wins[wins$item == i,]$rank, ite
```

```
      )
  }
}

    # Now, standardise values for ease of interpretation
    dat_w1 <- dat_w1 %>%
      mutate(
        rank_diff_raw = abs(item1_rank - item2_rank),
        log_rt_raw = log_rt,
        log_rt = scale(log_rt_raw),
        rank_diff = scale(rank_diff_raw)
      )


  plot(dat_w1$log_rt, dat_w1$rank_diff)
```



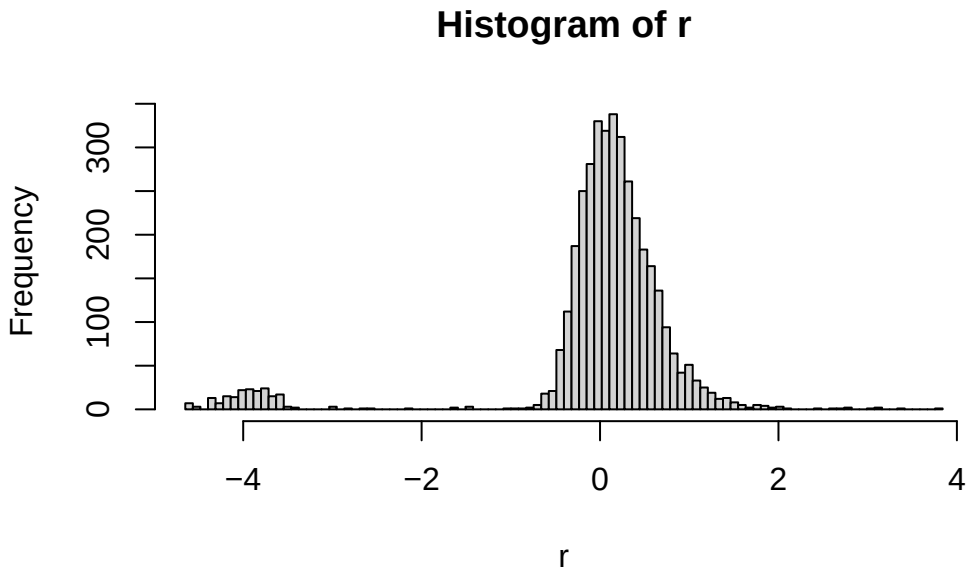
Now run and compare the models:

```
rt_model_rank <- lm(data = dat_w1, log_rt ~ rank_diff)


# fit a model including only random intercepts
```

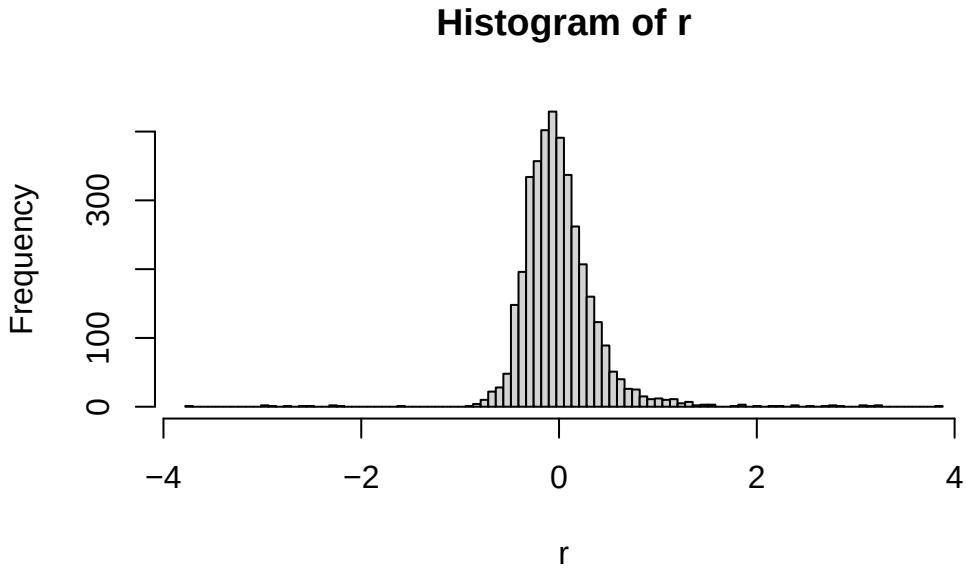
```
rt_model_rank_int_only <- lme(data = dat_w1, log_rt ~ rank_diff, random = ~ 1 | participant,

# fit a model including full random effects, and their correlation

rt_model_rank_random_effects <- lme(data = dat_w1, log_rt ~ rank_diff, random = ~ 1 + rank_di

# test inclusion of random intercepts
test <- lrtest(rt_model_rank,rt_model_rank_int_only)
```

```
Warning in modelUpdate(objects[[i - 1]], objects[[i]]): original model was of
class "lm", updated model is of class "lme"
```

```
# adjust, since we compare a lm to a lme object
log_likelihood <- test[2, 2] + test[1, 2]
chi_sq <- 2 * log_likelihood
p <- dchisq(chi_sq, 1)

#one d.f. difference
AIC_diff <- 2 * (1 - log_likelihood)

# test inclusion of random slopes
test2 <- anova(rt_model_rank_random_effects, rt_model_rank_int_only)

AIC_diff2 <- summary(rt_model_rank_random_effects)$AIC - summary(rt_model_rank_int_only)$AIC

# re-fit final model with REML
rt_model_rank_final <- lme(data = dat_w1, log_rt ~ rank_diff, random = ~ 1 + rank_diff| parti

# plotted residuals at trial level look good.
r <-residuals(rt_model_rank_final, level = 0)
hist(r, breaks = seq(min(r), max(r), length = 101))
```




```
# plotted residuals at subject level look good, too
r <-residuals(rt_model_rank_final, level = 1)
hist(r, breaks = seq(min(r), max(r), length = 101))
```



```r
# look at the model
summary(rt_model_rank_final)
```

```
Linear mixed-effects model fit by REML
  Data: dat_w1
       AIC      BIC    logLik
  4000.822 4038.275 -1994.411

Random effects:
 Formula: ~1 + rank_diff | participant
 Structure: General positive-definite, Log-Cholesky parametrization
            StdDev     Corr
(Intercept) 0.93958615 (Intr)
rank_diff   0.02733615 -0.283
Residual    0.40061920

Fixed effects:  log_rt ~ rank_diff
                  Value  Std.Error   DF   t-value p-value
(Intercept)  0.00021780 0.21020213 3779  0.001036  0.9992
rank_diff   -0.04279327 0.00922077 3779 -4.640967  0.0000
 Correlation:
          (Intr)
rank_diff -0.187

Standardized Within-Group Residuals:
       Min         Q1        Med         Q3        Max
-9.4338333 -0.5544375 -0.1128027  0.4123496  9.6826272

Number of Observations: 3800
Number of Groups: 20
```

Are preferences the same for everyone?

With plots

```r
# recompute wins & ranks within each participant, but separate by wave

  a <- dat %>%
    group_by(participant, wave, item1) %>%
    summarise(
      wins1 = sum(win1)
```

```
  ) %>%
  ungroup() %>%
  dplyr::rename(item = item1)
```

```
`summarise()` has grouped output by 'participant', 'wave'. You can override
using the `.groups` argument.
```

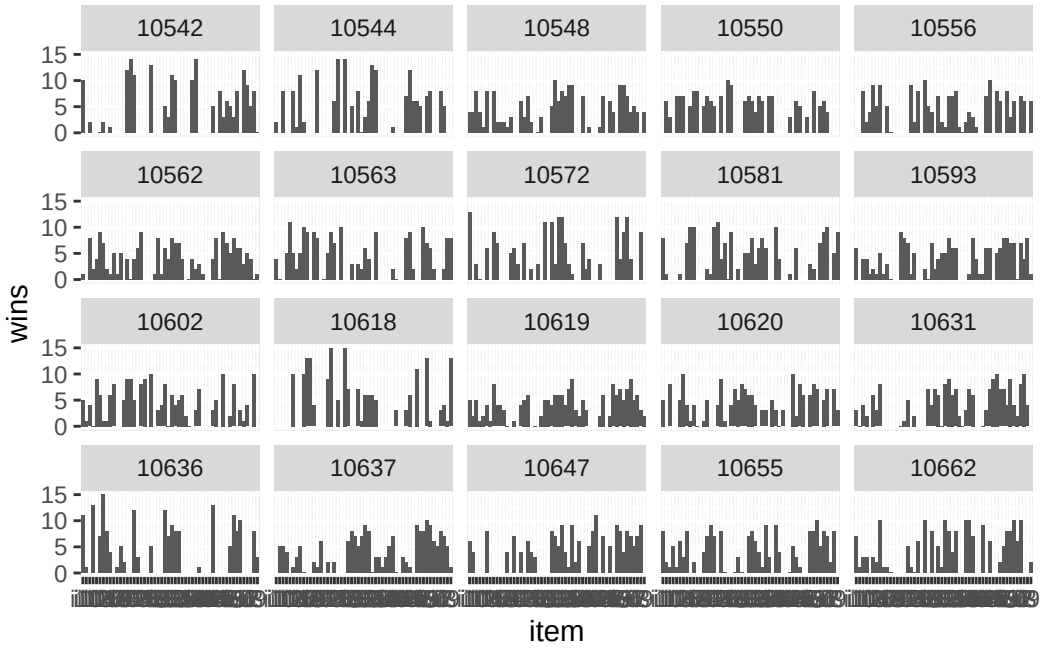
```
# some entries are missing, because items only appear in the item1 or item2 column
# for our merge operation below to include these, we need to
# manually add those entries

for (w in 1:2) {
  for (subj in unique(dat$participant)){

    item1_list <- filter(dat, participant == subj, wave == w)$item1 %>% unique()
    item2_list <- filter(dat, participant == subj, wave == w)$item2 %>% unique()

    for (i in item2_list[!item2_list %in% item1_list]) {
      a <- a %>% add_row(
        participant = subj,
        wave = w,
        item = i,
        wins1 = 0
      )
    }
  }
}

b <- dat %>%
  group_by(participant, wave, item2) %>%
  summarise(
    wins2 = sum(win2)
  ) %>%
  ungroup() %>%
  dplyr::rename(item = item2)
```

```
`summarise()` has grouped output by 'participant', 'wave'. You can override
using the `.groups` argument.
```

```
  for (w in 1:2) {
    for (subj in unique(dat$participant)){

      item1_list <- filter(dat, participant == subj, wave == w)$item1 %>% unique()
      item2_list <- filter(dat, participant == subj, wave == w)$item2 %>% unique()

      for (i in item1_list[!item1_list %in% item2_list]) {
        b <- b %>%
          add_row(
            participant = subj,
            wave = w,
            item = i,
            wins2 = 0
        )
      }
    }
  }

# compute ranks within each participant
  wins_participant <- merge(a, b) %>%
    mutate(wins = wins1 + wins2) %>%
    arrange(participant) %>%
    group_by(participant)


wins_participant_w1 <- wins_participant %>%
  filter(wave == 1) %>%
  mutate(
    rank = rank(wins)
  )

plot_w1 <- ggplot(data = wins_participant_w1) +
  facet_wrap(facets = vars(participant),, scales = "fixed") +
  geom_col(aes(x = item, y = wins), stat = "sum")
```

```
Warning in geom_col(aes(x = item, y = wins), stat = "sum"): Ignoring unknown
parameters: `stat`
```

```
plot_w1
```



With Aitchison's distance

Same idea as in the test-retest section. However, now we randomly pair up participants. If their preference profiles were actually the same, we would expect the overall Aitchison's distance to be close to 0.

```
# randomly group participants into pairs
# for now, only look at the first wave
p1 <- unique(dat$participant) %>% sample(round(length(unique(dat$participant))/2))
p2 <- unique(dat$participant)[!unique(dat$participant) %in% p1]

# prep data

# recompute wins & ranks within each participant, but separate by wave

  a <- dat %>%
    group_by(participant, wave, item1) %>%
    summarise(
      wins1 = sum(win1)
    ) %>%
    ungroup() %>%
    dplyr::rename(item = item1)
```

```
`summarise()` has grouped output by 'participant', 'wave'. You can override
using the `.groups` argument.
```

```r
  # some entries are missing, because items only appear in the item1 or item2 column
  # for our merge operation below to include these, we need to
  # manually add those entries

  for (w in 1:2) {
    for (subj in unique(dat$participant)){

      item1_list <- filter(dat, participant == subj, wave == w)$item1 %>% unique()
      item2_list <- filter(dat, participant == subj, wave == w)$item2 %>% unique()

      for (i in item2_list[!item2_list %in% item1_list]) {
        a <- a %>% add_row(
          participant = subj,
          wave = w,
          item = i,
          wins1 = 0
        )
      }
    }
  }

  b <- dat %>%
    group_by(participant, wave, item2) %>%
    summarise(
      wins2 = sum(win2)
    ) %>%
    ungroup() %>%
    dplyr::rename(item = item2)
```

```
`summarise()` has grouped output by 'participant', 'wave'. You can override
using the `.groups` argument.
```

```r
  for (w in 1:2) {
    for (subj in unique(dat$participant)){

      item1_list <- filter(dat, participant == subj, wave == w)$item1 %>% unique()
      item2_list <- filter(dat, participant == subj, wave == w)$item2 %>% unique()

      for (i in item1_list[!item1_list %in% item2_list]) {
```

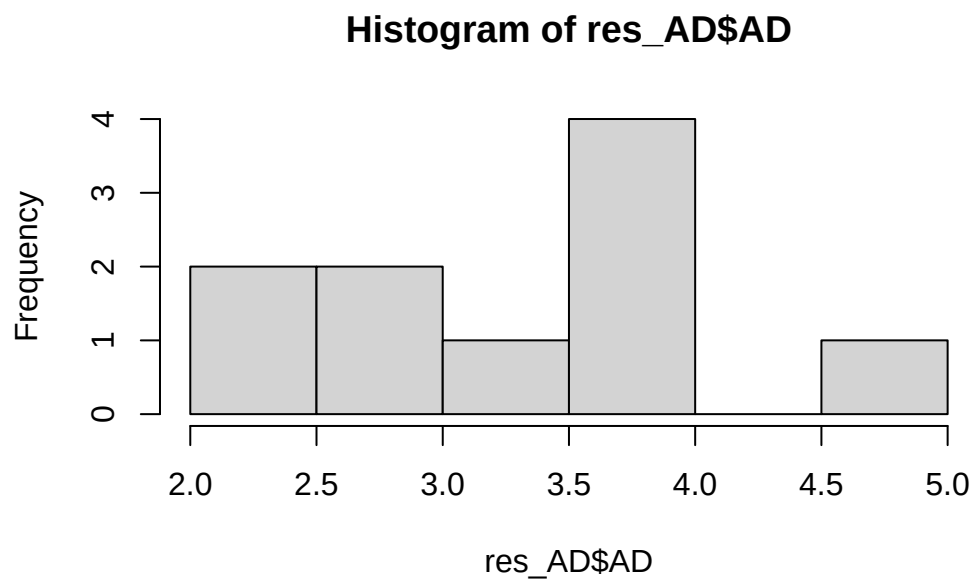
```
        b <- b %>%
          add_row(
            participant = subj,
            wave = w,
            item = i,
            wins2 = 0
        )
      }
    }
  }

# compute ranks within each participant
  wins_participant_wave <- merge(a, b) %>%
    mutate(wins = wins1 + wins2) %>%
    arrange(participant) %>%
    group_by(participant)


res_AD <- matrix(nrow = 0, ncol = 6) %>% as.data.frame()
colnames(res_AD) <- c(
  "participant1", # which participants
  "participant2",
  "shared",      # how many items are shared between the two waves (i.e., relevant for AD)
  "total_w1",    # how many items in wave 1
  "total_w2",    # how many items in wave 2
  "AD"           # measured Aitchison's Distance
)

n_participants <- 0

for (p in 1:length(p1)){

  dat1<- wins_participant_wave %>%
    filter(
      wave == 1,
      participant == p1[p]
    )

  dat2 <- wins_participant_wave %>%
    filter(
      wave == 1,
      participant == p2[p]
```

```
  )

# in both cases, the contests were based on the current selection of items participants enc
# i.e., this list may change
# thus, we cannot compute the aichison's distance over all those items. Instead, use only t

# How much overlap was there in selected items?

# Items from one that are also in 2
shared_items <- unique(dat1$item)[unique(dat1$item) %in% unique(dat2$item)]

shared <- length(shared_items)

# how many items in total in 1?
total_w1 <- unique(dat1$item) %>% length()

# how many items in total in 2?
total_w2 <- unique(dat2$item) %>% length()

# select only shared items

dat1 <- dat1 %>% filter(item %in% shared_items) %>% mutate(rank = rank(wins))
dat2 <- dat2 %>% filter(item %in% shared_items) %>% mutate(rank = rank(wins))

log_ratio1 <- c()
log_ratio2 <- c()
distance <- c()

remaining_players <- shared_items[2:length(shared_items)]

# now get the log ratios of ranks

for (i in shared_items) {
  for (j in remaining_players) {

    # note, we only compute the distance for the upper triangle of the item-item matrix, th
    if (i != j) {

      rank1i <- filter(dat1, item == i)$rank
      rank2i <- filter(dat2, item == i)$rank
      rank1j <- filter(dat1, item == j)$rank
      rank2j <- filter(dat2, item == j)$rank
```

```
        l1 <- log(rank1i / rank1j)
        l2 <- log(rank2i / rank2j)

        log_ratio1 <- append(log_ratio1, l1)
        log_ratio2 <- append(log_ratio2, l2)
      }
    # don't repeat items that already fought every other item.
    remaining_players = remaining_players[remaining_players != i]
    }
  }

  # compute the aitchison distance over the log ratios.

  AD = sqrt(1/(2*shared) * sum((log_ratio1 - log_ratio2)^2))

  # save info

  res_AD <- res_AD %>%
    add_row(
      participant1 = p1[p],
      participant2 = p2[p],
      shared = shared,
      total_w1 = total_w1,
      total_w2 = total_w2,
      AD = AD
    )
   n_participants <- n_participants + 1
   print(paste(n_participants, " out of ", length(p1), " done!", sep = ""))
}
```

```
[1] "1 out of 10 done!"
[1] "2 out of 10 done!"
[1] "3 out of 10 done!"
[1] "4 out of 10 done!"
[1] "5 out of 10 done!"
[1] "6 out of 10 done!"
[1] "7 out of 10 done!"
[1] "8 out of 10 done!"
[1] "9 out of 10 done!"
[1] "10 out of 10 done!"
```

```r
# have a look at the distribution of distances
hist(res_AD$AD)
```




```r
# looks reasonably normal so run a one-sample t-test
t.test(res_AD$AD, alternative = "greater")
```

```
    One Sample t-test

data:  res_AD$AD
t = 13.609, df = 9, p-value = 1.309e-07
alternative hypothesis: true mean is greater than 0
95 percent confidence interval:
 2.841781      Inf
sample estimates:
mean of x
 3.284149
```

Exploration

Is there more entropy in the preference profile of more depressed participants?

This will use the entropy score we computed in the section 'Entropy' above. We would then set up a correlation matrix, correlating entropy score to age, sex and depression status. Not illustrated here, since the code would be very straightforward.

Can we cluster participants based on their preferences?

k-means clustering is a clustering method that minimises variance within clusters and maximises variance between clusters. First, multi-dimensional scaling is used to project our data into a 2-dimensional space where distances between points represent dissimiliarity of scores (when doing this 'seriously, the number of dimensions will have to be determined through model comparison). Next, k-means clustering is applied. For illustrative purposes, I have extracted only 2 clusters (though a solution with 3 clusters looks reasonable as well). For the full dataset, one would have to use more sophisticated metrics to validate the number of clusters to be extracted.

```
library(magrittr)
```

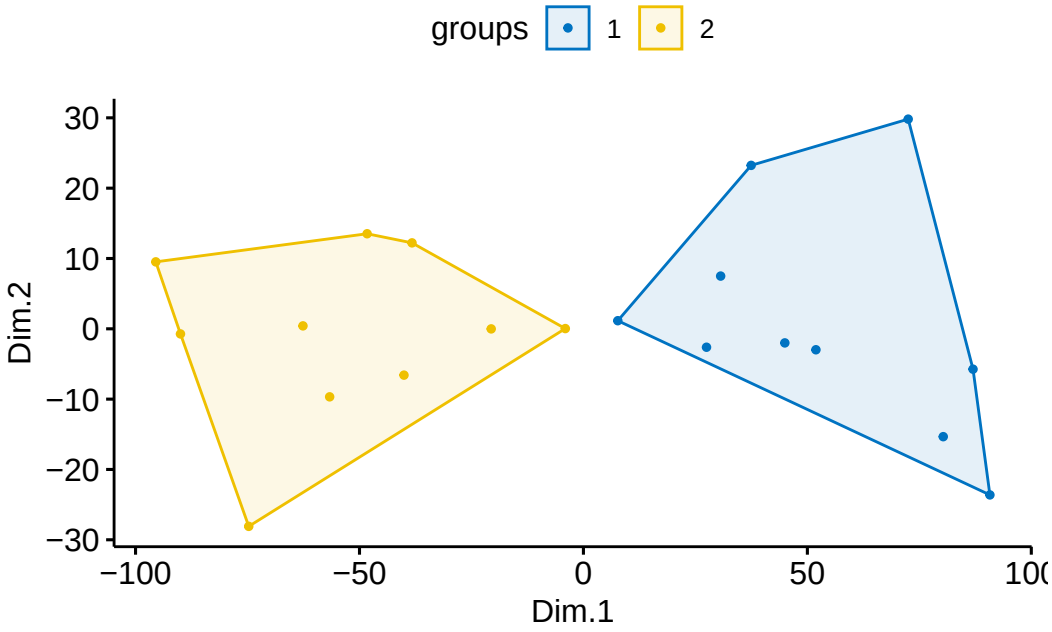
```
Attaching package: 'magrittr'

The following object is masked from 'package:purrr':

    set_names

The following object is masked from 'package:tidyr':

    extract
```

```
library(dplyr)
library(ggpubr)

# get data

dat_MDS <- dat_subj %>%
  dplyr::select(participant, starts_with("severity_response_"))
```

```
# Cmpute MDS
mds <- dat_MDS %>%
  dist() %>%
  cmdscale() %>%
  as_tibble()
```

```
Warning: The `x` argument of `as_tibble.matrix()` must have unique column names if
`.name_repair` is omitted as of tibble 2.0.0.
i Using compatibility `.name_repair`.
```

```
colnames(mds) <- c("Dim.1", "Dim.2")

# Plot MDS
my_plot <- ggscatter(mds, x = "Dim.1", y = "Dim.2",
          label = NULL,
          size = 1,
          repel = TRUE)

# now add in groups using clustering

clust <- kmeans(mds, 2)$cluster %>%
  as.factor()

mds <- mds %>%
  mutate(groups = clust)
# Plot and color by groups
this_plot <- ggscatter(mds, x = "Dim.1", y = "Dim.2",
          label = NULL,
          color = "groups",
          palette = "jco",
          size = 1,
          ellipse = TRUE,
          ellipse.type = "convex",
          repel = TRUE)

plot(this_plot)
```


groups
1
2
30
20
10
0
−10
−20
−30
Dim.2
−100
−50
0
50
100
Dim.1