Data Wrangling and Visualization 101

for BCS 206

Fall 2019

Contents

1 Goals for next two weeks

- Thinking about workflow in R:
  - Version control
  - R Markdown
- Data wrangling: Turning the data into the form you need (*dplyr*)
- Data visualization:
  - General principles
  - How to plot in R (*ggplot*, *plotly*)

We only have a relatively short time, so we will focus on learning what tools are available and on *examples* of use (rather than an in-depth tutorial). There are great online tutorials and cheatsheets that contain further information.

2 Preliminaries

2.1 Version control

RStudio makes version control, data backup, and data sharing easy (e.g., via Github.com). To use it, download and install git on your computer. Get a free github.com or bitbucket.com account. You only have to do this once.

Then, for each project, create a new project in RStudio and link it to the remote repository (select "Create project" > "Version control"). You will have to enter a URL for the remote repository, which you get, for example, at github.com under the repository's main page by clicking the "Clone or download button".

For step by step instructions, see:

- Setting up RStudio for version control
- RStudio help on version control
- Reverting a file to an earlier version

2.2 Reproducibility and literate coding

R and RStudio support reproducibility oriented literate coding via Sweave and Knitr: lab books, presentations, and papers can weave/knit together data, code, and text. The document you share contains the code needed to create its outputs (figures, tables, etc.). This is achieved by combining latex or R markdown with R code (or, for that matter, code from other programming languages). For an excellent video-based introduction, see this tutorial on R markdown. *This document is R markdown compiled with RStudio's knitr.

3 Data wrangling

The *R* libraries *dplyr* provide us with efficient ways to transform ('wrangle') our data tables. The library *magrittr* let's us concatenate these operations in transparent and easy to read code.

3.1 An example data set

We will illustrate the use of *dplyr* with the following data from an experiment with a 2AFC task in three within-subject conditions (A, B, C), for which we have extracted correctness (1 = correct; 0 = incorrect) and reaction times (RT):

```
summary(d)
```

```
##  condition     trial           subject        correct             RT
##  A:2688    Min.   : 1.00   1      : 192   Min.   :0.0000   Min.   : 180.1
##  B:2688    1st Qu.:16.75   2      : 192   1st Qu.:0.0000   1st Qu.: 394.4
##  C:2688    Median :32.50   3      : 192   Median :1.0000   Median : 555.4
##            Mean   :32.50   4      : 192   Mean   :0.6308   Mean   : 748.9
##            3rd Qu.:48.25   5      : 192   3rd Qu.:1.0000   3rd Qu.: 986.5
##            Max.   :64.00   6      : 192   Max.   :1.0000   Max.   :3162.5
##                            (Other):6912
```

```
glimpse(d)
```

```
## Observations: 8,064
## Variables: 5
## $ condition <fct> A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, ...
## $ trial     <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ subject   <fct> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17...
## $ correct   <int> 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, ...
## $ RT        <dbl> 568.801, 357.301, 476.414, 352.467, 522.844, 328.874, 381...
```

3.2 Dplyr's verbs

Dplyr has 'verbs' like filter, select, summarize, mutate, transmute, etc. to let use conduct operations on our data, and reshape the data frame into the format we need. We can use dplyr, for example, to calculate the proportion correct answers in our experiment by using *summarise*.

```
summarise(d, meanCorrect = mean(correct))
```

```
## # A tibble: 1 x 1
##   meanCorrect
##         <dbl>
## 1       0.631
```

Or just for condition A:

```
d.A = filter(d, condition == "A")
summarise(d.A, meanCorrect = mean(correct))
```

```
## # A tibble: 1 x 1
##   meanCorrect
##         <dbl>
## 1       0.496
```

3.3 Maggritr's pipes

Here we will use only of the 'pipes' magrittr provides:

- x %>% f: takes x and hands it to the function f on the right, as f's first argument

- x %<>% f1 %>% f2 %>% etc.: takes x hands it to f1, takes the output of f1 and hands it to f2, etc. And since the first pipe was %<>% (rather than just %>%), the final result will be written back into x.

Figure 1: Magritt's pipe

Figure 2: Magrittr's pipe

3.4 Putting it together: Wrangling through pipes

Remember how we got the mean proportion correct for just Condition A?

```
d.A = filter(d, condition == "A")
summarise(d.A, meanCorrect = mean(correct))
```

```
## # A tibble: 1 x 1
##   meanCorrect
##         <dbl>
## 1       0.496
```

This is inelegant and hard to read. Pipes let us make this more transparent:

```
d %>%
  filter(condition == "A") %>%
  summarise(meanCorrect = mean(correct))
```

```
## # A tibble: 1 x 1
##   meanCorrect
##         <dbl>
## 1       0.496
```

And this advantage becomes even clearer, the more operationgs we concatenate. For example, *group_by* is an elegant operator that tells the pipes to conduct all subsequent operations for each of the groups (and then put all the separate outcomes back together into a single data frame). So if we want the proportion correct for all groups:

```r
d %>%
  group_by(condition) %>%
  summarise(meanCorrect = mean(correct))
```

```
## # A tibble: 3 x 2
##   condition meanCorrect
##   <fct>           <dbl>
## 1 A               0.496
## 2 B               0.583
## 3 C               0.814
```

3.5 Exercises

How can we:

- View the entire data set? (*View*)
- Calculate the by-subject averages for all three conditions? (*group_by*, *summarise*)
- Calculate the by-subject standard deviations around those averages? (*group_by*, *summarise*)
- Attach this information (the averages and SDs) to each row of the present data.frame? (*group_by*, *mutate*)
- Determine whether RTs were on average faster for correct, as compared to incorrect, trials?
- Add a column for log-transformed RTs to the data set?
- Remove the old column for raw RTs? (*select*)
- Sort the data by log-transformed reaction times? (*arrange*)

Say we further have an additional data frame with information about our subjects:

```
## Source: local data frame [42 x 3]
## Groups: <by row>
##
## # A tibble: 42 x 3
##    subject gender   age
##    <fct>   <chr>  <dbl>
##  1 1       female    21
##  2 2       female    21
##  3 3       male      21
##  4 4       male      18
##  5 5       female    19
##  6 6       female    19
##  7 7       male      20
##  8 8       female    20
##  9 9       male      19
## 10 10      female    20
## # ... with 32 more rows
```

- How can we join the information from the two data sources together? (*left_join*)

```
## Source: local data frame [42 x 3]
## Groups: <by row>
##
## # A tibble: 42 x 3
##    subject gender   age
```

```
##    <fct>   <chr>  <dbl>
##  1 1       female    21
##  2 2       female    21
##  3 3       male      21
##  4 4       male      18
##  5 5       female    19
##  6 6       female    19
##  7 7       male      20
##  8 8       female    20
##  9 9       male      19
## 10 10      female    20
## # ... with 32 more rows
```

```
## Joining, by = "subject"
```

```
## # A tibble: 8,064 x 7
##    condition trial subject correct    RT gender   age
##    <fct>     <int> <fct>     <int> <dbl> <chr>  <dbl>
##  1 A             1 1             1  569. female    21
##  2 A             1 2             0  357. female    21
##  3 A             1 3             1  476. male      21
##  4 A             1 4             0  352. male      18
##  5 A             1 5             0  523. female    19
##  6 A             1 6             1  329. female    19
##  7 A             1 7             1  381. male      20
##  8 A             1 8             1  480. female    20
##  9 A             1 9             0  380. male      19
## 10 A             1 10            1  409. female    20
## # ... with 8,054 more rows
```

4 Data visualization

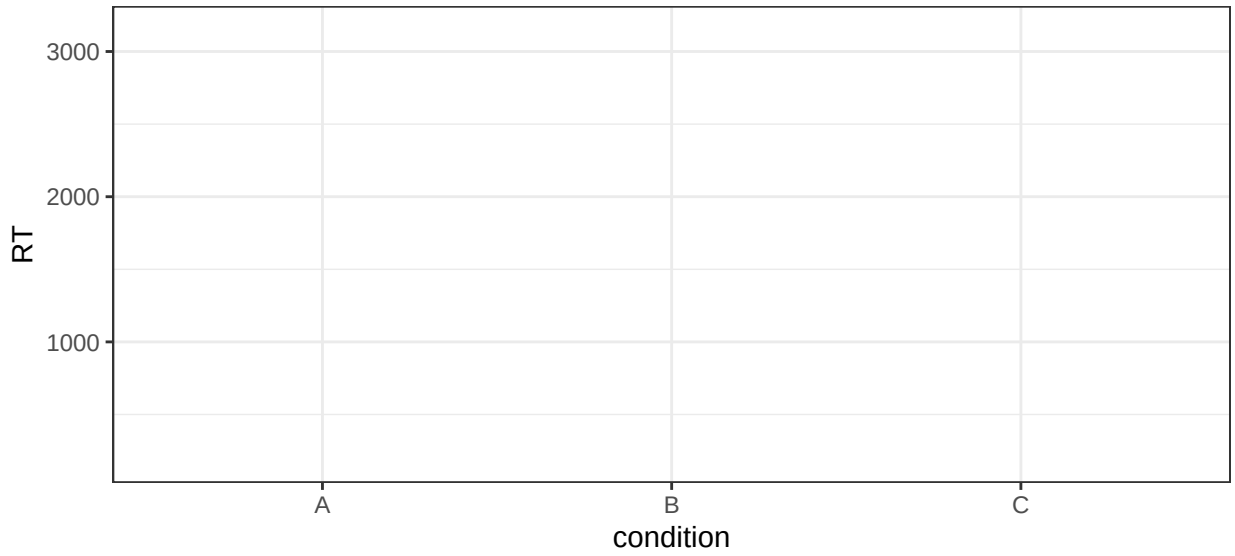
The two main libraries in R we will be using for visualization are *ggplot2* and *plotly*. Ggplot2 provides a grammar of graphics approach to plotting. Plotly let's us interact with our data. In particular, *ggplotly()* wrapped around a ggplot2 figure let's us interact with that figure.

4.1 Ggplot2's components (aesthetic mappings)

In order to plot in ggplot2, we need to understand the way it thinks about visualization. There are excellent online course that explain all of this, so I focus on the basics.

At the heart of a plot is a mapping between properties of your data (i.e., column in your data frame) and abstract properties of the plot (such as x- or y-coordinates, color, fill, transparency (alpha), linetype, shape, or label information). If we call the function ggplot() in order to create a figure, we specify two arguments: the name of the data frame we want to work with, and the mapping. The latter is done through a helpful function called *aes()*—for aesthetics:

```
ggplot(
  data = d,
  mapping =
    aes(
      x = condition,
      y = RT))
```



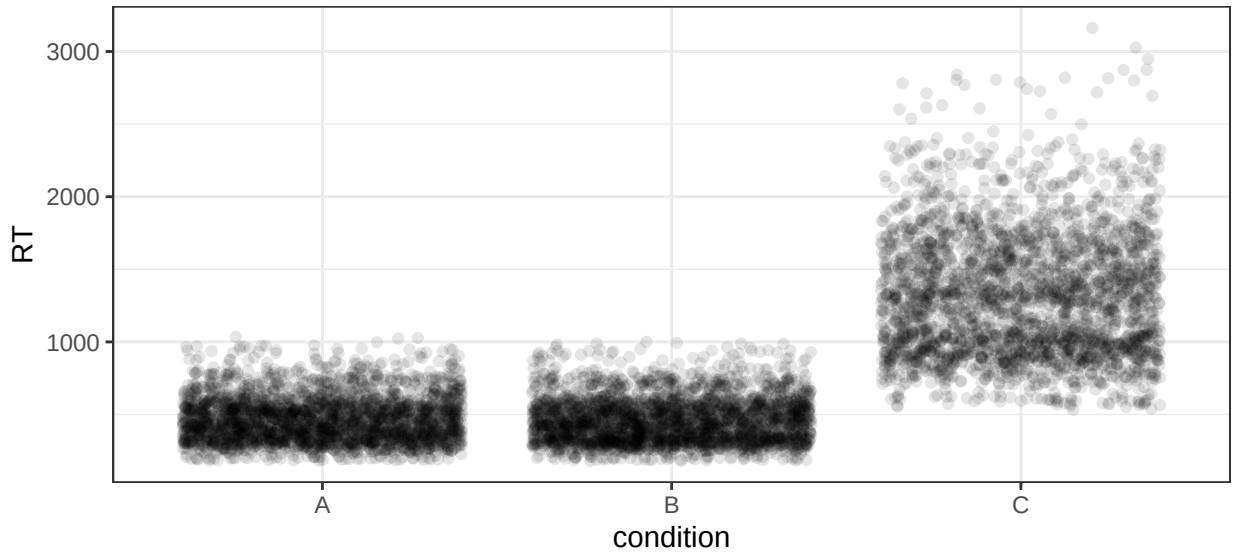

```
# or equivalently and shorter:
# ggplot(
#   d,
#   aes(
#       x = condition,
#       y = RT))
```

4.1.1 Adding geometric components (geoms)

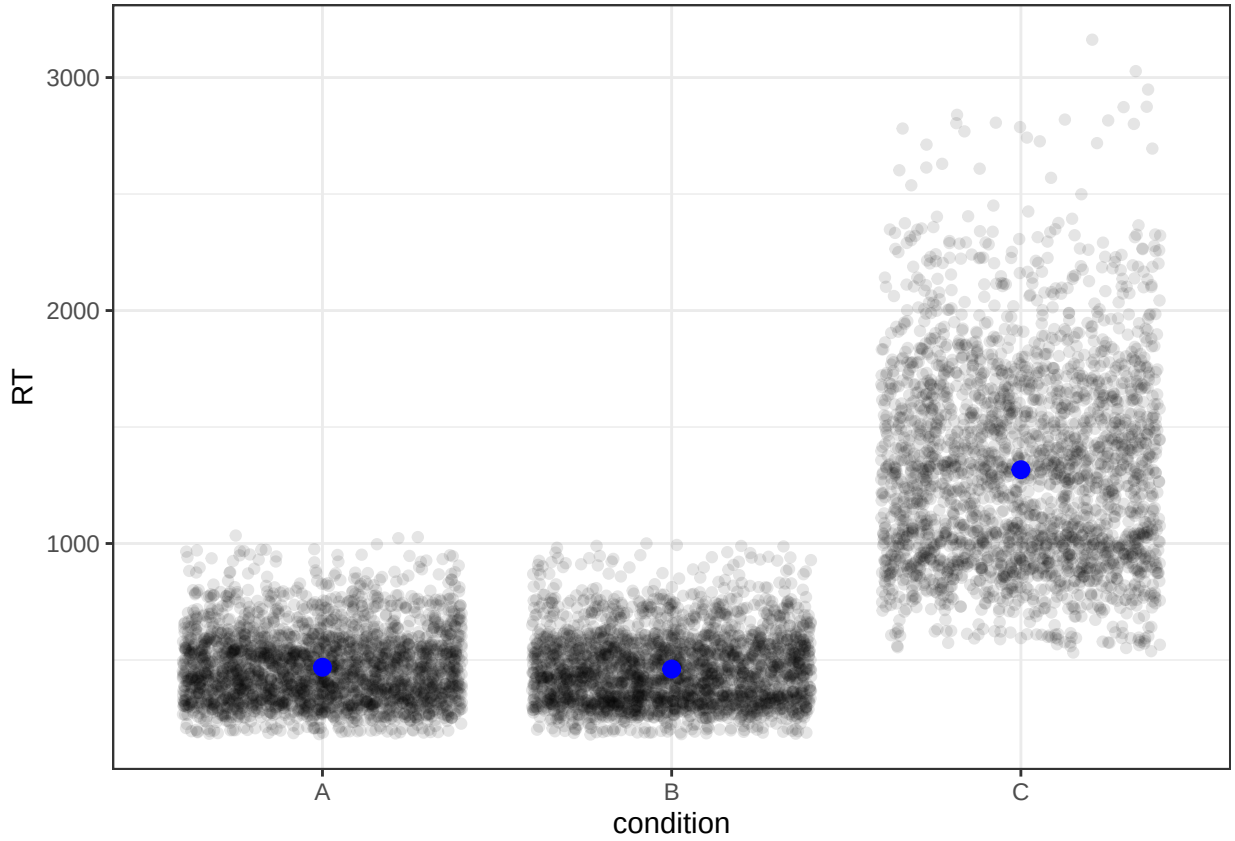
Notice that this by itself only returns an empty plot. That's the case because we have not yet specified how we want the abstract properties of the graph to be expressed visually. That's achieved by specifying *geom*s (for geometrics), such as points (*geom_point*), lines (*geom_line*), histograms (*geom_histogram*), lineranges (*geom_linerange*), and many similar functions (I take it you're getting th hang for the naming scheme …). You can find all of them on the ggplot2 cheatsheet.

We add such components to a plot with "+". We can also further explicitly specify any unused aesthetical properties of any geom. For example, to plot all the RTs for all three conditions with some transparency so that we see whether points cluster, plus some jittering along the x-axis (for the same reason):

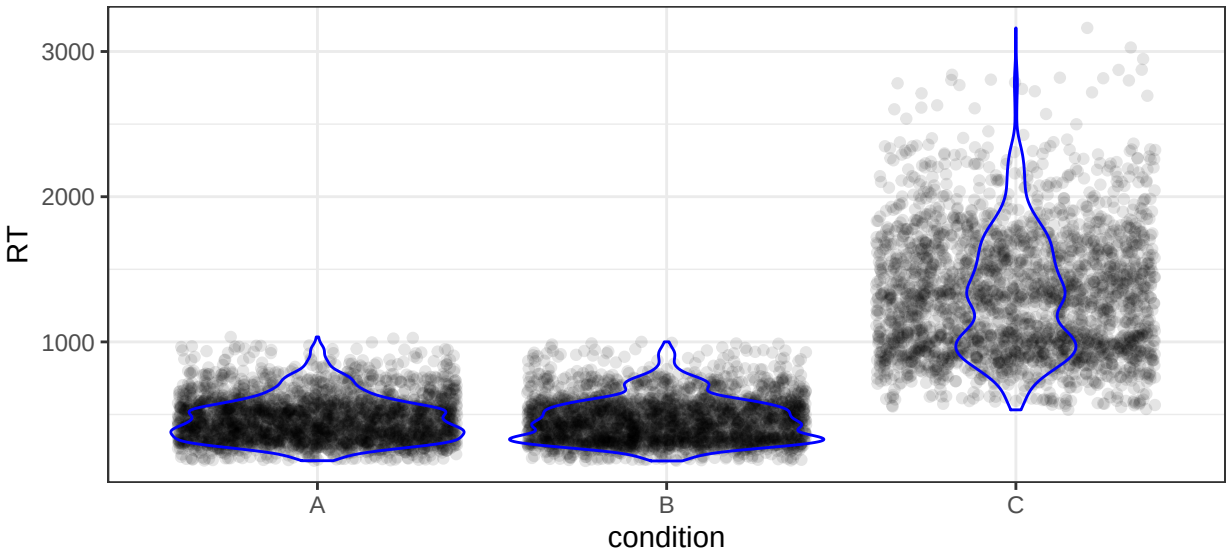
```
p = ggplot(
  data = d,
  mapping =
    aes(
      x = condition,
      y = RT)) +
  geom_point(alpha = .1, position = position_jitter())

plot(p)
```



We could also summarize the data and plot a bootstrapped 95% confidence interval as a pointrange. In this case, we're specifying a statistical summary of the data and, as part of that, specify through which type of geom we would like it to be expressed:

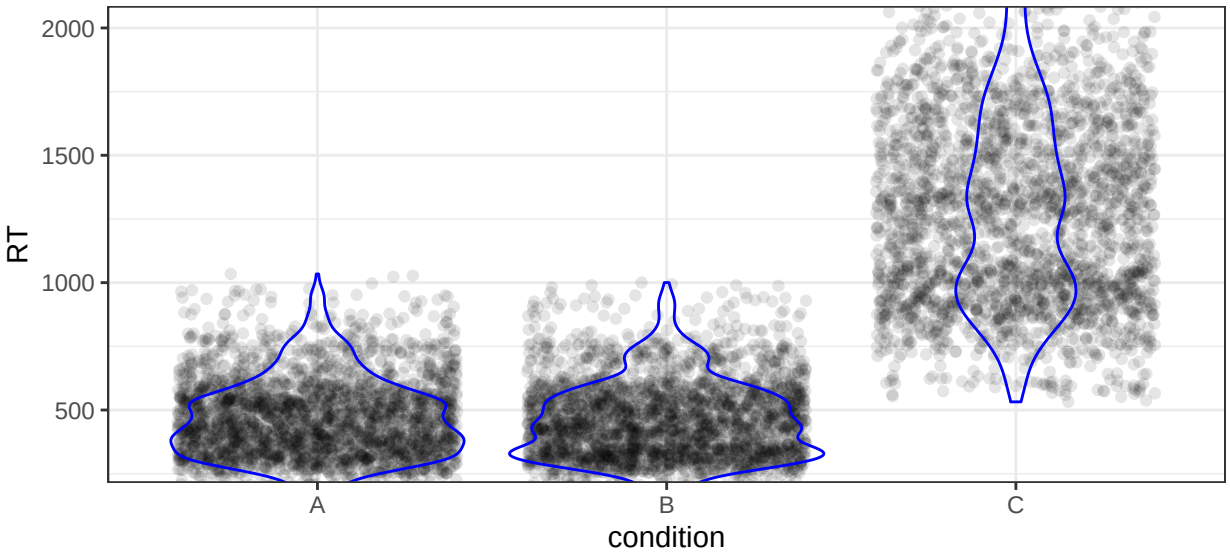


Alternatively, we could add a violin plot (essentially a mirrored density distribution, in this case displayed vertically on top of the points):

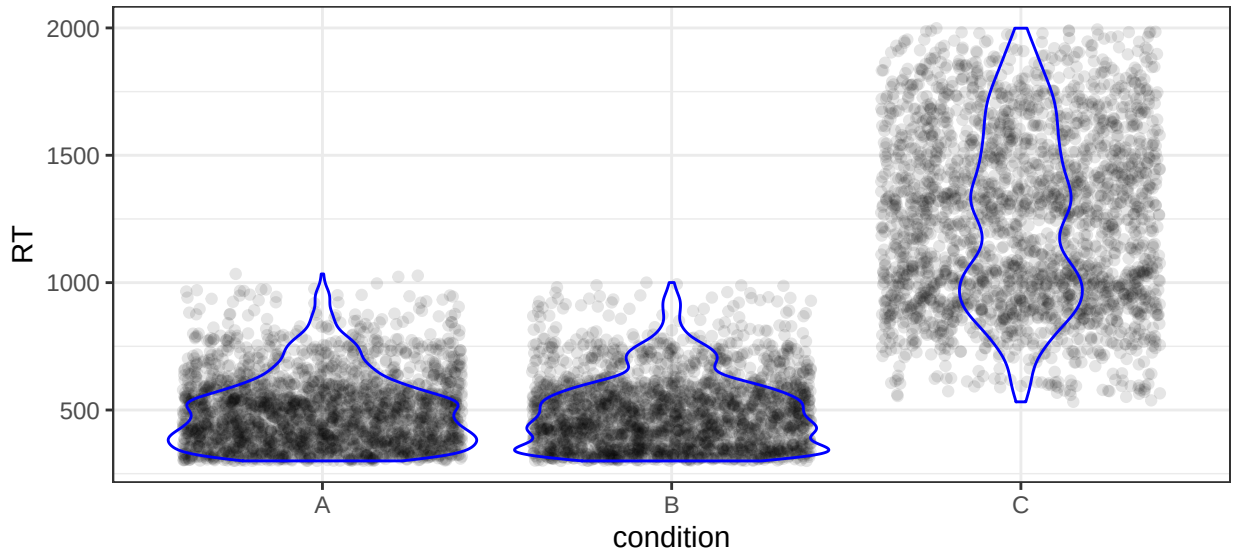


4.1.2 Scales and coordinate systems

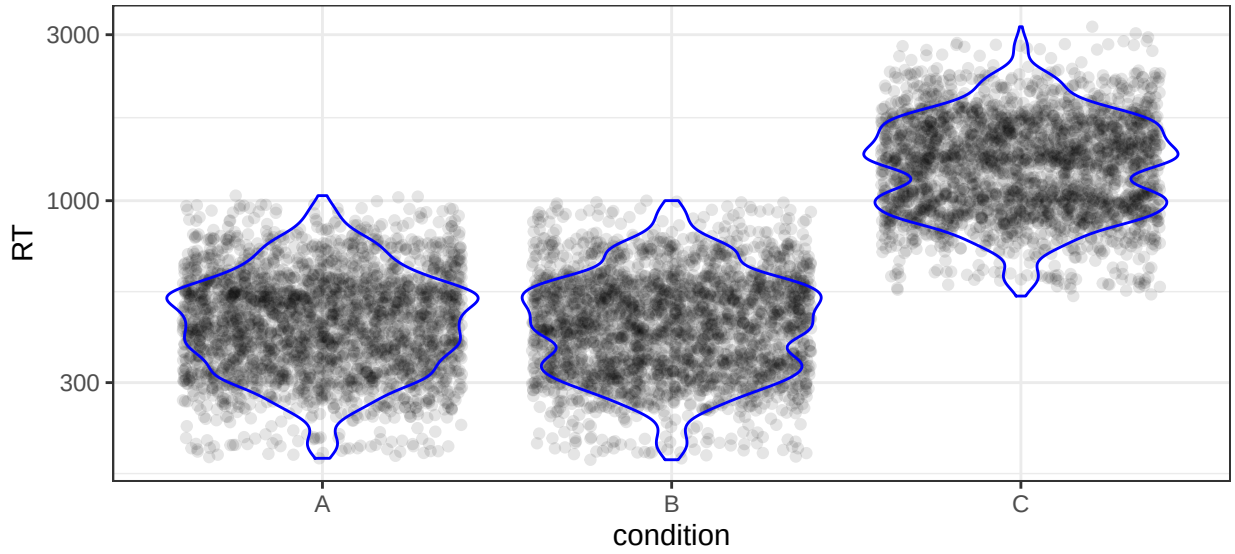
Sometimes we don't want to see all of the data, or we want to zoom into some ranges of our data. We can do so by explicitly specifying the x- and y-limits of our coordinate system:



Note that this zooms into parts of our data without excluding any data (e.g., from the calculation of the violins, which have the same shape as above). If we want to exclude data, transform data or in other ways change the way the aesthetical mappings are interpreted, this is achieved through *scales*. For example, the following *ex*cludes all RTs below 300 and above 2000. Note how that changes the violin plots (as it should: they estimate the ditribution of RTs):

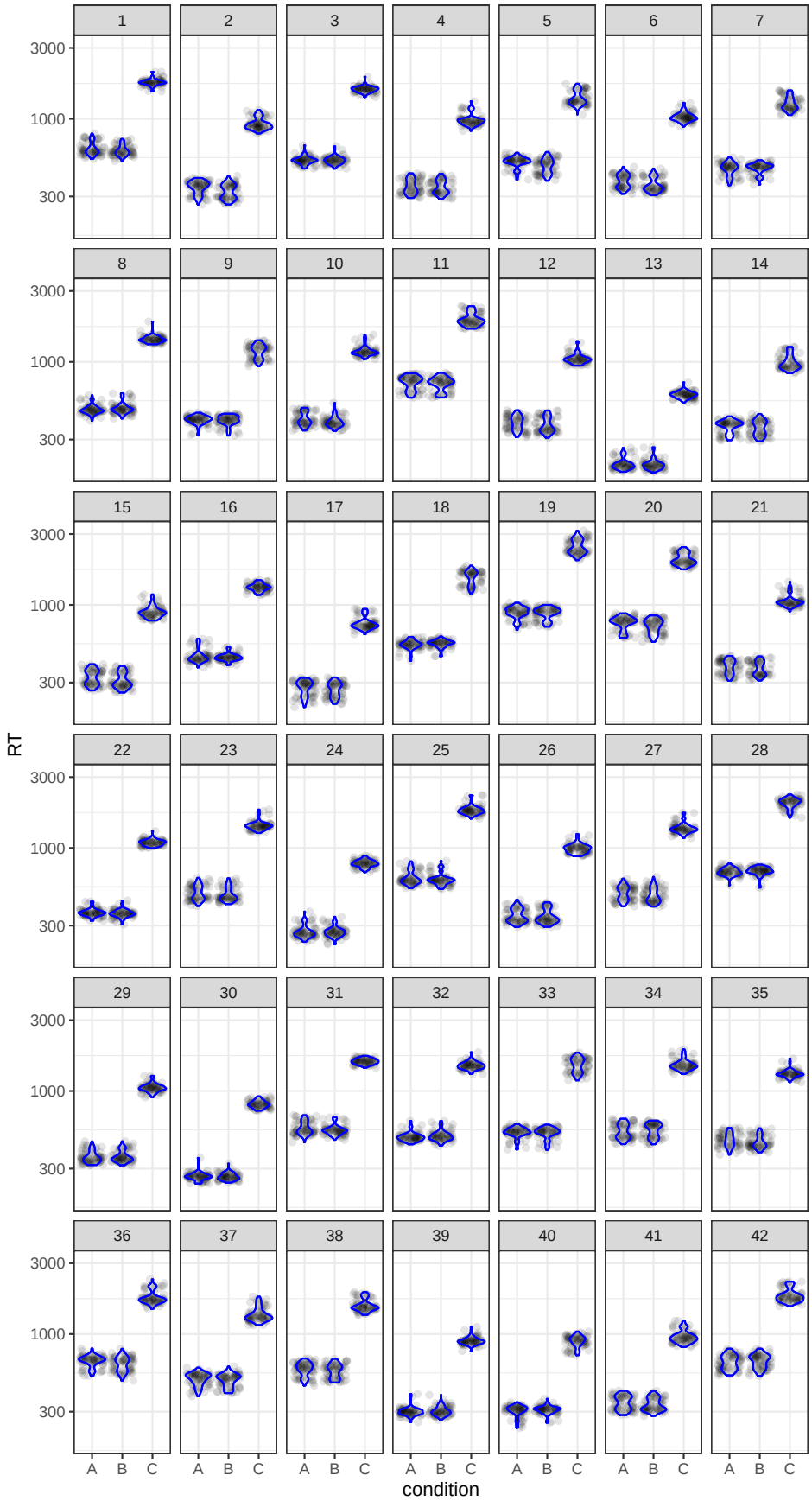


Since reaction times often have distributions that are more lognormal, rather than normal, let's update our original plot to use a log-transformed y-axis:

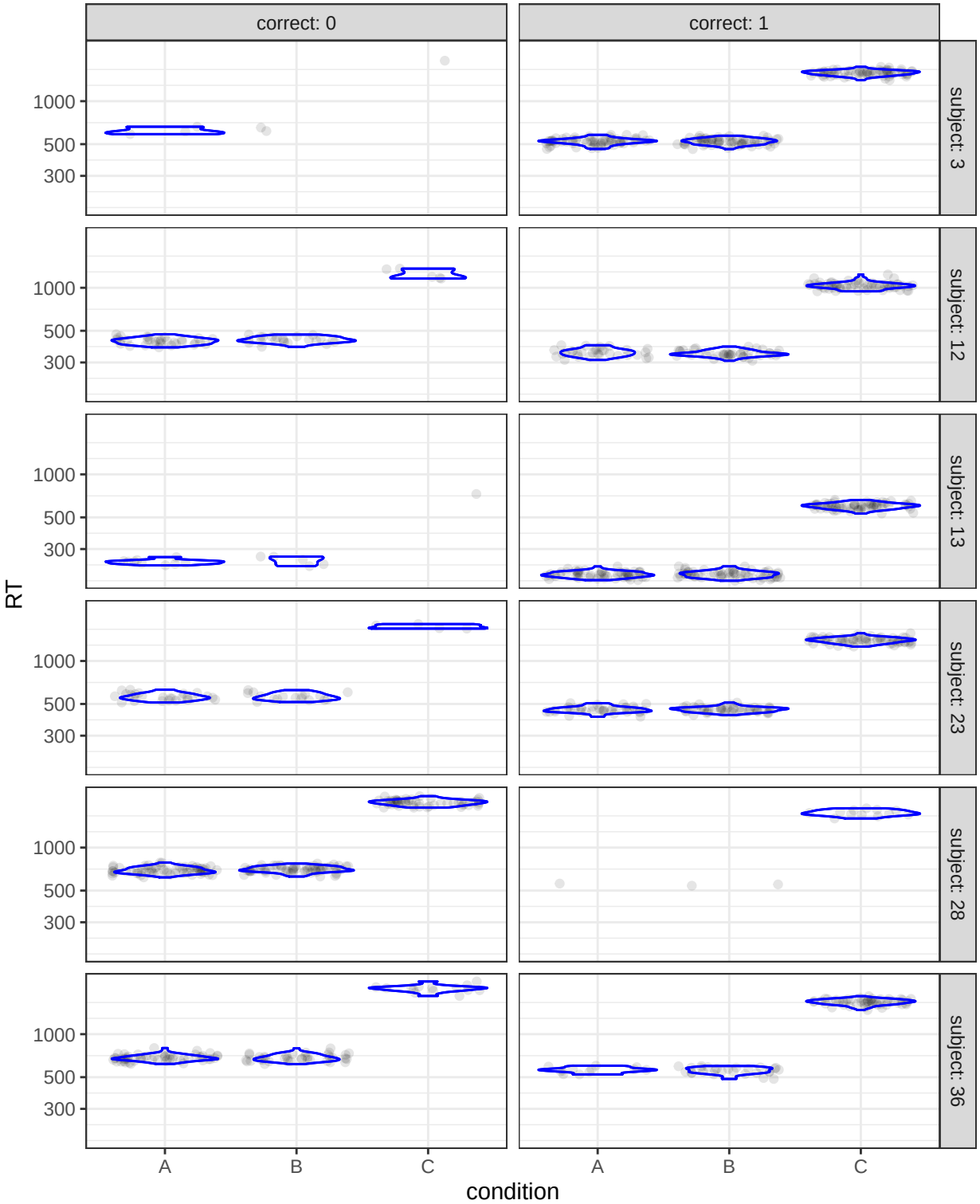


4.1.3 Facets

If we want to have separate panels conditional on another variable, we can do so through *facet*s. There are two major facet functions, *facet_wrap* (to have panels conditional on one variable) and *facet_grid* (conditional on two variables). For example, we can have separate panels for each subject:


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
3000
1000
300
RT
A
B
C
condition

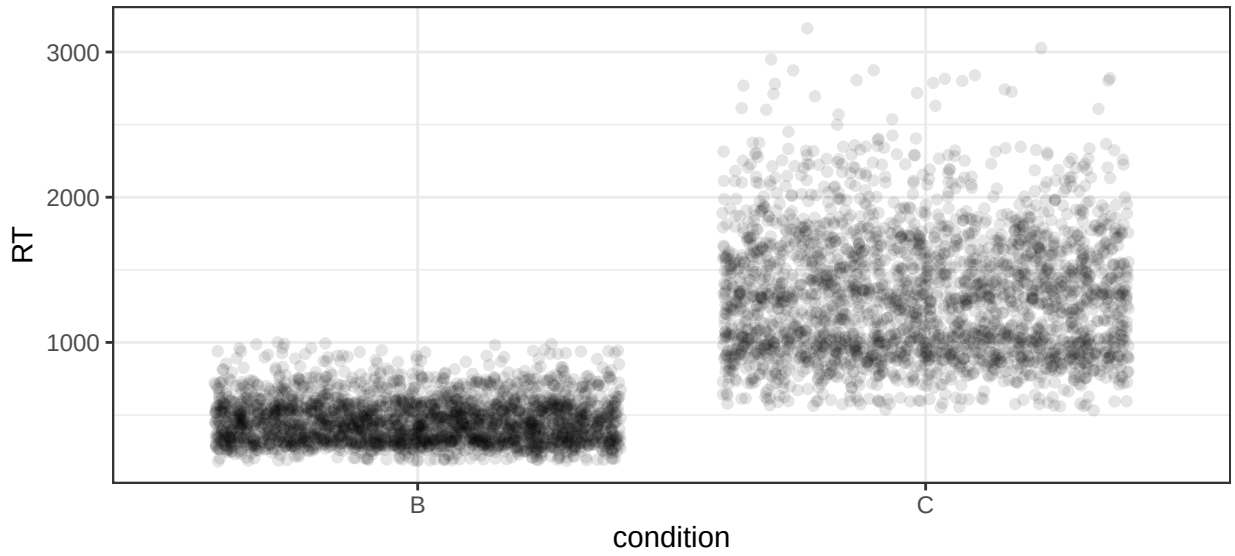
Or we could show by-subject RTs in two columns, separately for false and correct answers. Here, we do so after first sampling 6 random subjects (since the plot would otherwise be rather large).



4.2 Pipes (again)

Of course, we can use pipes to pipe the data frame into the plotting function, optionally after first piping the data through some additional *dplyr* operations (since the output of that entire pipe is again a data frame):

```
d %>%
  filter(condition != "A") %>%
  ggplot(
    aes(
      x = condition,
      y = RT)) +
  geom_point(alpha = .1, position = position_jitter())
```



4.3 Exercises

- Plot a histogram of the RTs by condition. Make on version where you plots the histograms in different facets, and another version where you have only one facet and use fill color to distinguish between conditions. (*geom_histogram*)
- Plot the average proportion of correct answers by condition as a pointrange.
- Do the same, but first average by subject and condition, and then plot the average (and confidence interval) of those by-subject averages of correct responses.
- Try to make a pie chart that shows the proportion correct for the three conditions. (*coord_polar*)

5 Case Study I: (Rucci group)

This study seeks to determine whether myopia affects fundamental properties of eye-movments. To this end, we compare typical and myopic subjects in a eye-movement task.

Subjects moved their eyes horizontally to target fixation points. We are interested in three dependent variables for each trial:

- How long did it take the subject to move their eyes to the target? (reaction time)
- How fast were their eye-movements during the fastest point of the trial? (peak velocity)
- How long did it take to reach this peak velocity? (time to peak velocity)

None of these variables are available in the raw data, and we will have to infer / create them from the raw data.

5.1 Design

Targets are presented at seven locations (*stimulus_deg* is one of -30, -20, -10, 0, 10, 20, 30). The total degree of horizontal movement depends on the stimulus on the previous trial, and the new stimulus on the current trial. For example, if the previous trial had a stimulus degree of -20 and the present trial has a stimulus degree of 20, that corresponds to a movement of positive 40 degrees.

5.2 Loading data from .csv file

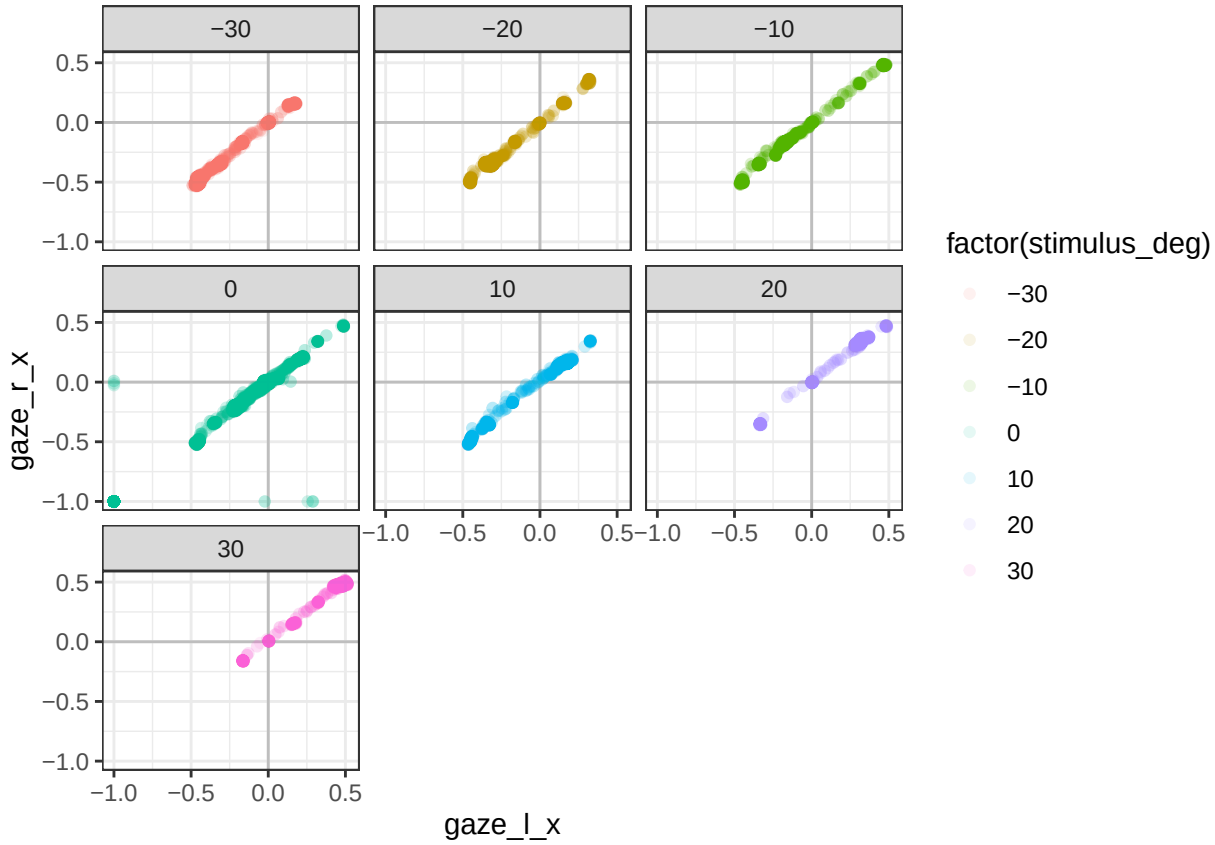
```
## Parsed with column specification:
## cols(
##   SMI_timestamp = col_double(),
##   Unity_timestamp = col_double(),
##   head_pos_x = col_double(),
##   head_pos_y = col_double(),
##   head_pos_z = col_double(),
##   head_rot_x = col_double(),
##   head_rot_y = col_double(),
##   head_rot_z = col_double(),
##   head_rot_w = col_double(),
##   gaze_l_x = col_double(),
##   gaze_l_y = col_double(),
##   gaze_l_z = col_double(),
##   gaze_r_x = col_double(),
##   gaze_r_y = col_double(),
##   gaze_r_z = col_double(),
##   session_idx = col_double(),
##   stimulus_deg = col_double()
## )
```

```
##  SMI_timestamp       Unity_timestamp      gaze_l_x            gaze_l_y
##  Min.   :0.000e+00   Min.   :  1.515   Min.   :-1.000000   Min.   :-1.000000
##  1st Qu.:4.022e+10   1st Qu.: 44.985   1st Qu.:-0.342505   1st Qu.:-0.015963
##  Median :7.076e+10   Median : 75.524   Median :-0.159412   Median :-0.005040
##  Mean   :7.041e+10   Mean   : 75.146   Mean   :-0.126632   Mean   :-0.033892
##  3rd Qu.:1.013e+11   3rd Qu.:106.064   3rd Qu.: 0.007257   3rd Qu.: 0.004665
##  Max.   :1.319e+11   Max.   :136.625   Max.   : 0.514591   Max.   : 0.096892
##     gaze_l_z          gaze_r_x            gaze_r_y             gaze_r_z
##  Min.   :-1.0000   Min.   :-1.000000   Min.   :-1.0000000   Min.   :-1.0000
##  1st Qu.: 0.8959   1st Qu.:-0.355706   1st Qu.:-0.0200058   1st Qu.: 0.8845
##  Median : 0.9719   Median :-0.157496   Median :-0.0081834   Median : 0.9708
##  Mean   : 0.9016   Mean   :-0.135263   Mean   :-0.0360727   Mean   : 0.8959
##  3rd Qu.: 0.9947   3rd Qu.: 0.009525   3rd Qu.: 0.0000424   3rd Qu.: 0.9955
##  Max.   : 1.0000   Max.   : 0.516798   Max.   : 0.1120100   Max.   : 1.0000
##      trial        stimulus_deg
##  Min.   : 0.00   Min.   :-30.000
##  1st Qu.: 9.00   1st Qu.:-20.000
##  Median :39.00   Median :  0.000
##  Mean   :41.24   Mean   : -6.309
##  3rd Qu.:70.00   3rd Qu.:  0.000
```

```
##  Max.   :96.00   Max.   : 30.000
```

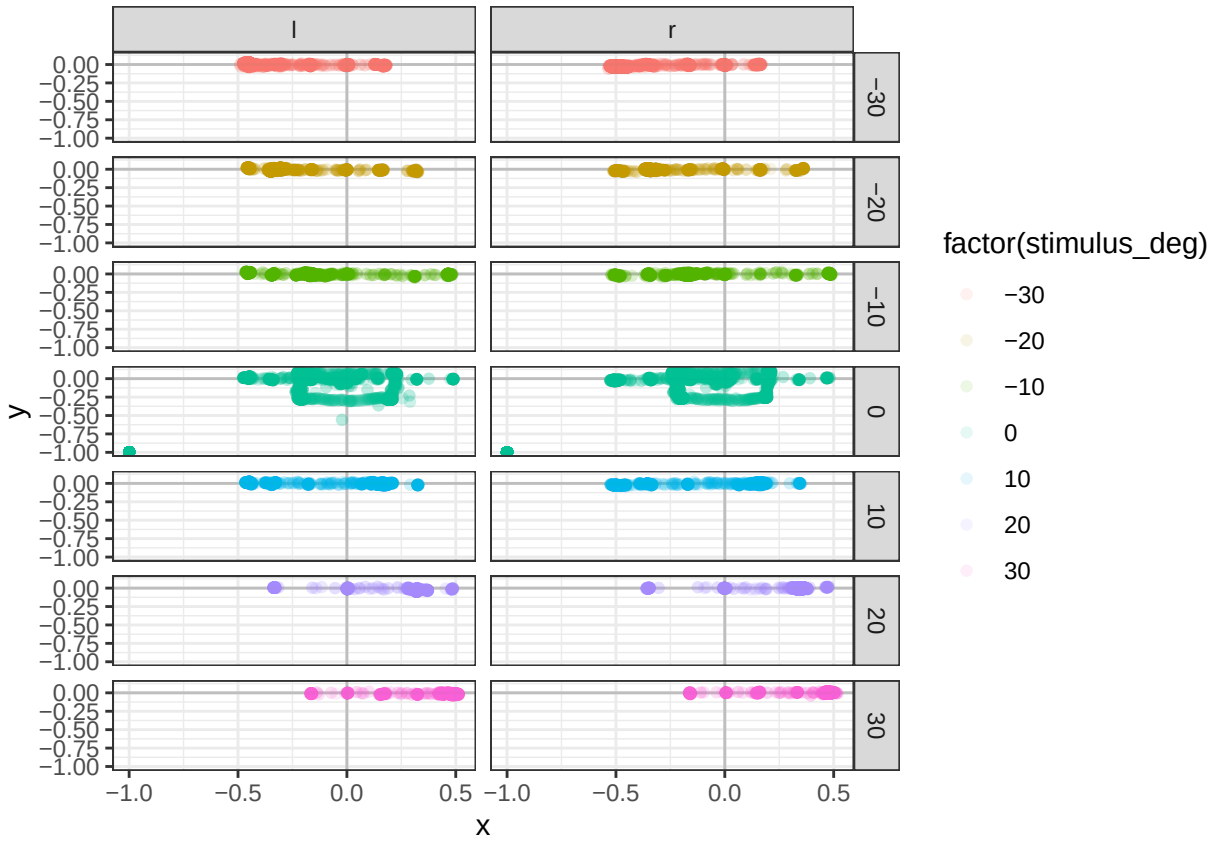
5.2.1 Sanity check: Plotting left and right eye's x coordinate by stimulus degree

If the left and right eye are generally tracked well, then we should see that most of the data points fall onto the 45 degree diagonal when we plot the left and right eye's x-position on the x-axis and y-axis, respectively:



5.2.2 Sanity check: Plotting left and right eye's x and y coordinates by stimulus degree

If the subject did the task, we should see that eyes primarily move along the x-axis, rather than, for example, the y-axis. For this we first need to transform that data so that we have separate rows for gaze information about the left and right eye. Then we can plot the data in a way very similar to the plot in the previous section.



5.3 Getting information about previous trial's stimulus

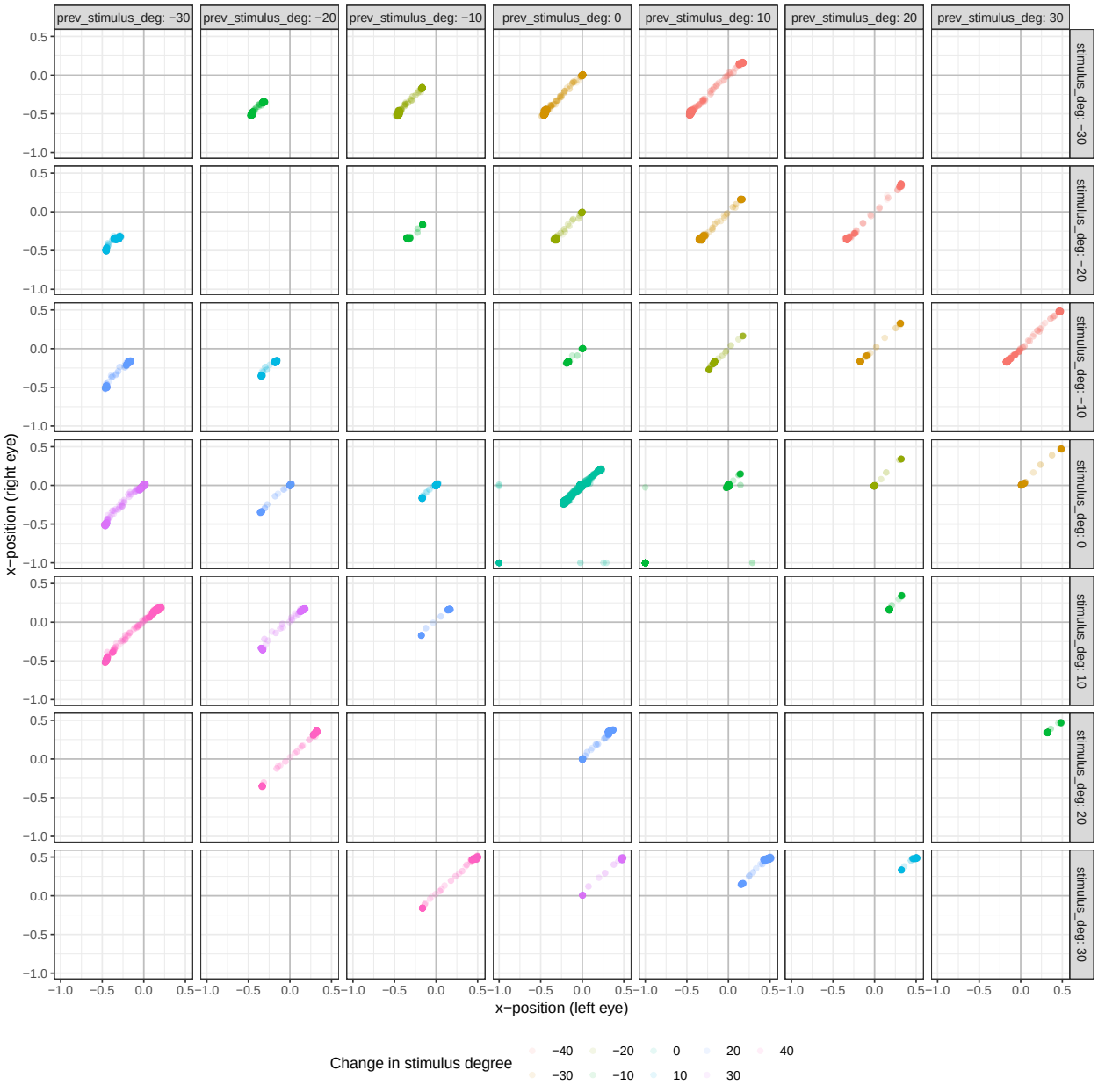
Let's calculate the change in degrees from the previous stimulus to the present stimulus. We will call this variable *stimulus_deg_delta*. This variable tells us how much and in which direction subjects had to move their eyes on the present trial.

```
## Joining, by = c("trial", "stimulus_deg")
```

```
## # A tibble: 27,980 x 12
##    SMI_timestamp Unity_timestamp gaze_l_x gaze_l_y gaze_l_z gaze_r_x gaze_r_y
##            <dbl>           <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
##  1             0            1.51        0        0        0        0        0
##  2             0            1.54        0        0        0        0        0
##  3             0            1.56        0        0        0        0        0
##  4             0            1.56        0        0        0        0        0
##  5             0            2.01        0        0        0        0        0
##  6             0            2.01        0        0        0        0        0
##  7             0            2.01        0        0        0        0        0
##  8             0            2.01        0        0        0        0        0
##  9             0            2.01        0        0        0        0        0
## 10             0            2.01        0        0        0        0        0
## # ... with 27,970 more rows, and 5 more variables: gaze_r_z <dbl>, trial <dbl>,
## #   stimulus_deg <dbl>, prev_stimulus_deg <dbl>, stimulus_deg_delta <dbl>
```

5.3.1 Plot current eye position based on stimulus degree the previous stimulus degree

This plot let's us check whether subjects were doing the task. Each row shows one target (*stimulus_deg*) and each column shows where the eye-movement started off (*prev_stimulus_deg*). Color indicates the total change in stimulus degree. Take for example the first row: we see that the subject's fixation seems to always end in the same point (-.5), corresponding to the stimulus degree of the present trial (-30). Where the eye-movements start differs from column to column, depending on the stimulus degree of the previous trial.



5.4 Getting information about reaction time, velocity, peak velocity, and time to peak velocity

We can calculate the speed of the eye movement at each point in time by deviding the distance traveled by the time that has passed. We can do so either along just one dimension (say x) or along any combination of dimensions. Here, we calculate the distance traveled, and speed, along the x-axis.

TO DO: determine the appropriate definitions for these three measures. For example, the RT measure seems to contain a lot of zero values, suggesting that we're doing something wrong. Perhaps one should simply take the highest time within each trial?

```
## # A tibble: 11,603 x 15
## # Groups:   trial [97]
##    SMI_timestamp gaze_l_x gaze_l_y gaze_l_z gaze_r_x gaze_r_y gaze_r_z trial
##            <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl> <dbl>
##  1             0   0        0        0      0        0        0        0
##  2    8602181300  -0.166    0.0704   0.984 -0.194    0.0730   0.978    0
##  3    8610204400  -0.165    0.0710   0.984 -0.199    0.0769   0.977    0
##  4    8622168700  -0.151    0.0657   0.986 -0.199    0.0770   0.977    0
##  5    8634196000  -0.162    0.0676   0.985 -0.198    0.0767   0.977    0
##  6    8646198700  -0.161    0.0678   0.985 -0.201    0.0740   0.977    0
##  7    8658193400  -0.138    0.0624   0.988 -0.126    0.0661   0.990    0
##  8    8670181700  -0.0390   0.0638   0.997 -0.0486   0.0753   0.996    0
##  9    8682180400   0.0232   0.0663   0.998  0.00883  0.0702   0.997    0
## 10    8690199800   0.0676   0.0622   0.996  0.0335   0.0769   0.996    0
## # ... with 11,593 more rows, and 7 more variables: stimulus_deg <dbl>,
## #   prev_stimulus_deg <dbl>, stimulus_deg_delta <dbl>, gaze_l_x_distance <dbl>,
## #   gaze_r_x_distance <dbl>, gaze_l_x_velocity <dbl>, gaze_r_x_velocity <dbl>
```

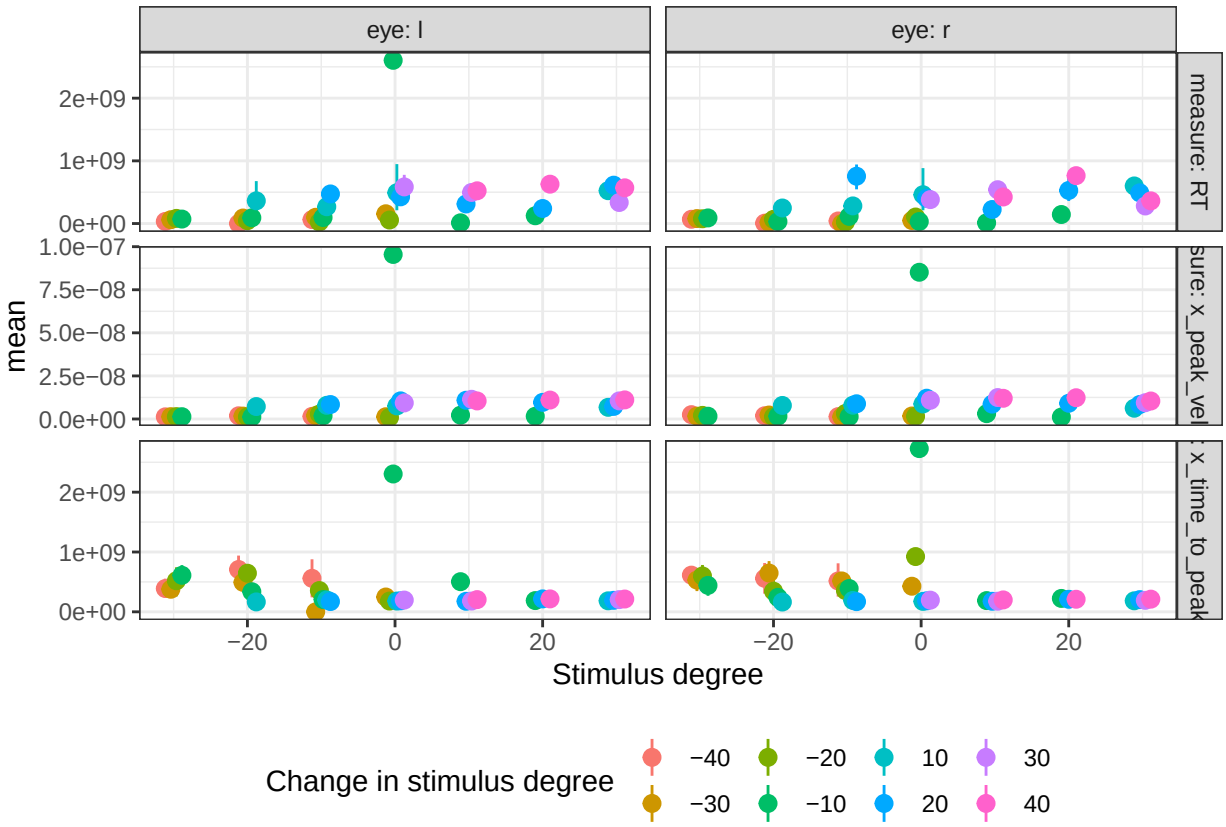
Next, we also determine the peak velocity, time to peak velocity, and reaction time within each trial. We can summarize the data to one row per trial. Notice how we now only have 97 rows—one for each trial (plus one for the 0th trial).

If we wanted to keep around all the within-trial msec-by-msec information, we would simply use mutate() instead of summarise().

```
## # A tibble: 97 x 9
## # Groups:   trial, stimulus_deg [97]
##    trial stimulus_deg stimulus_deg_de~ gaze_l_x_peak_v~ gaze_r_x_peak_v~
##    <dbl>        <dbl>            <dbl>            <dbl>            <dbl>
##  1     0            0                0    0.000000105      0.0000000853
##  2     1           20               20    0.0000000117     0.00000000999
##  3     2          -10              -30    0.00000000197    0.00000000128
##  4     3            0               10    0.00000000663    0.00000000832
##  5     4          -30              -30    0.00000000123    0.00000000114
##  6     5            0               30    0.0000000103     0.0000000120
##  7     6           20               20    0.00000000859    0.00000000891
##  8     7          -20              -40    0.00000000131    0.00000000131
##  9     8          -30              -10    0.00000000188    0.00000000179
## 10     9           10               40    0.0000000115     0.0000000127
## # ... with 87 more rows, and 4 more variables:
## #   gaze_l_x_time_to_peak_velocity <dbl>, gaze_r_x_time_to_peak_velocity <dbl>,
## #   gaze_l_RT <dbl>, gaze_r_RT <dbl>
```

5.4.1 Plotting these trial-level properties by trial type (and, in the future, subject type)

Ultimately, we would want to analyze the effect of between-subject variables (in pariculuar, myopic vs. non-myopic) on the trial-level variables we've derived above (peak velocity, time to peak velocity, and reaction time). Since we so far only have the data from one subject, we can instead plot these trial-level variables as a function of the different trial types:



6 Case Study II: visual decision-making (Haefner group)

This group seeks to replicate Herce Castañón et al. (2019).

6.1 Design

The design of the present study crossed two levels of contrast (Low = 15%, High = 60%), 3 levels of variance (0, 4, 10), and how the trials in the block were cued (L = left, R = right, N = uncued), for a total of 2 x 3 x 3 = 18 within-subject conditions.

6.2 Loading data from MatLab

The data are stored in a MatLab (.mat) file. The file contains one matrix with fields: participant, exp(eriment), stimuli and response. Within each field, there is further information. The important information seems to be in the response field. Some of the important parts include:

- responseRight: the response of the subject (0 for CCW, 1 for CW, w.r.t horizontal)
- correct: what the correct answer is (0 for CCW, 1 for CW, w.r.t horizontal)
- accuracy: whether subject got the correct answer (1) or not (0)
- reaction time: time in seconds the subject took to answer
- confidence: whether the subject was confident in their answer (1) or not (-1)
- cue: whether the cue on that trials is left (-1), right (1), or no cue (0)
- contrast: the contrast of the gabor patch on that trial
- variance: variability in the orientation of gratings of gabor patches on that trial

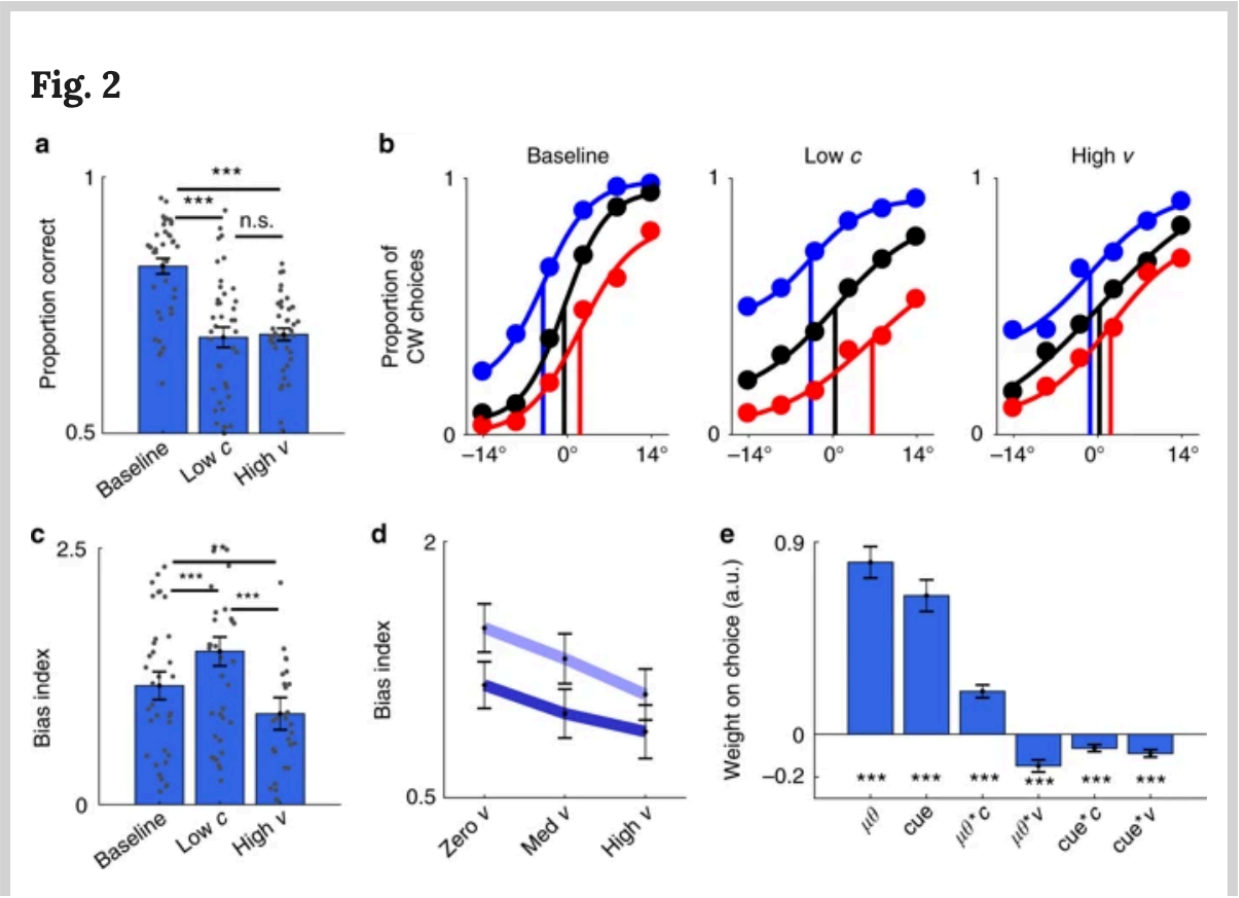


Figure 3: Figure 2 from Herce Castañón et al. (2019)

- isCuedBlock: whether a block (of trials) will have cues (1) or no cues (0)

```
# Load a matlab file and extract the "data" matrix out of it
d.haefner = readMat("./data/Haefner/uncertaintyV1-subject18-1-EarlyQuit.mat")
d.haefner = d.haefner[["data"]][,,1][["response"]][,,1]
d.haefner[["trueOrientaions"]] <- NULL

# Look at what we've imported.
# NB: str() gives your the structure of an R object
str(d.haefner)

d.haefner %<>%
  map(.f = function(x) c(x)) %>%
  as_tibble()

# The data we have are preliminary pilot data from one of the
# experimenters, and that run did contain all trials. We omit
# all the trials with missing information.
d.haefner %<>%
  na.omit()

# Add the definition of the three conditions of interest in the
# original paper
d.haefner %<>%
  mutate(
    condition = case_when(
        variance == min(variance) & contrast == max(contrast) ~ "baseline",
        variance == max(variance) & contrast == max(contrast) ~ "high variance",
        variance == min(variance) & contrast == min(contrast) ~ "low contrast",
        T ~ ""
    )
  )
```

Now that we've imported the data into an R data frame (or *tibble*), let's have a look at it. First, we can get a general idea of the data by using str() (for structure) or print():

```
## # A tibble: 864 x 12
##    randSeed responseRight correct accuracy reactionTime confidence isCuedBlock
##       <dbl>         <dbl>   <dbl>    <dbl>        <dbl>      <dbl>       <dbl>
##  1   2.20e8             0       1        0        0.708         -1           1
##  2   2.20e8             1       0        0        0.609         -1           1
##  3   2.20e8             0       1        0        1.73           0           1
##  4   2.20e8             1       0        0        0.684          0           1
##  5   2.20e8             0       1        0        0.550         -1           1
##  6   2.20e8             1       1        1        0.565         -1           1
##  7   2.20e8             0       1        0        0.492         -1           1
##  8   2.20e8             0       0        1        0.994         -1           1
##  9   2.20e8             0       1        0        0.872          0           1
## 10   2.20e8             0       1        0        0.782          1           1
## # ... with 854 more rows, and 5 more variables: cue <dbl>,
## #   orientationMean <dbl>, contrast <dbl>, variance <dbl>, condition <chr>
```

To instead get a summary of the data:

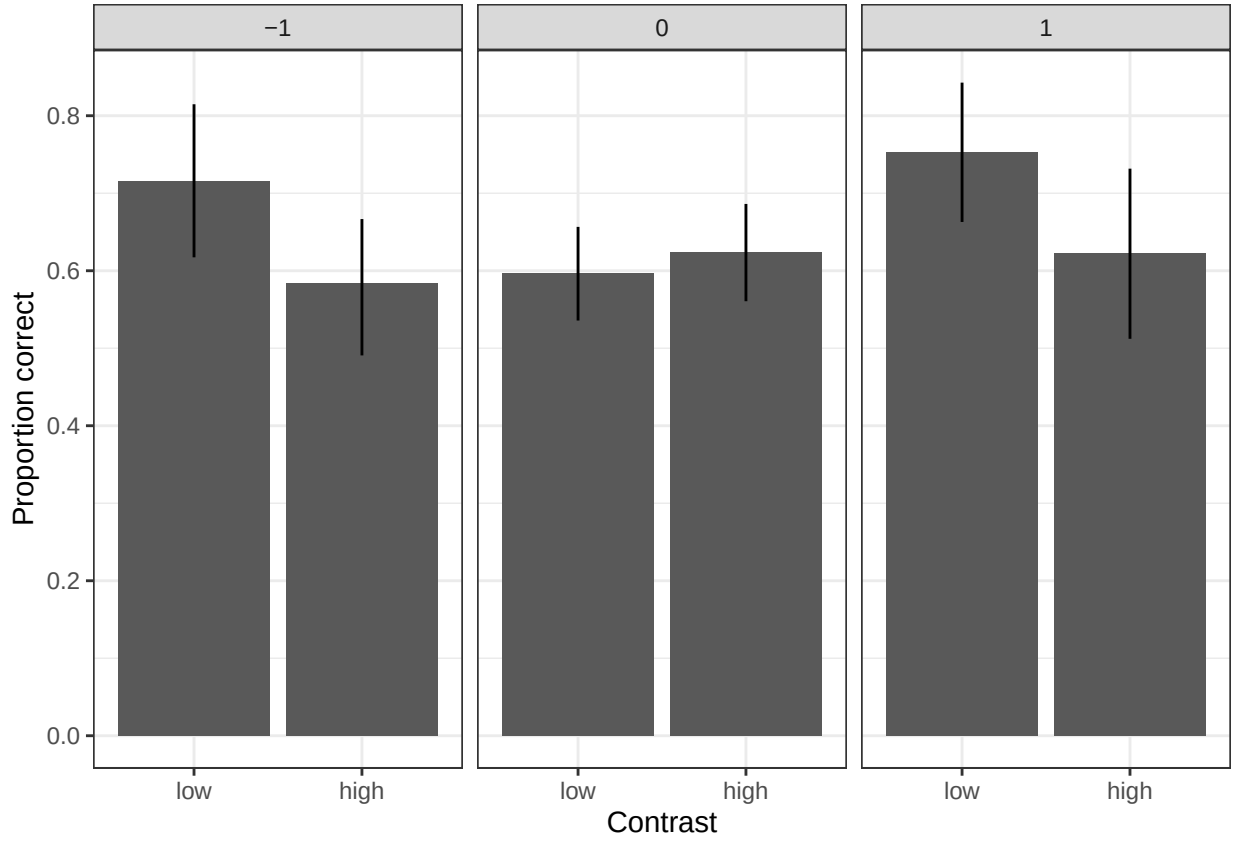
```
##     randSeed         responseRight      correct          accuracy
##  Min.   :220286057   Min.   :0.000   Min.   :0.0000   Min.   :0.0000
```

```
##  1st Qu.:220377520   1st Qu.:0.000   1st Qu.:0.0000   1st Qu.:0.0000
##  Median :220475950   Median :1.000   Median :1.0000   Median :1.0000
##  Mean   :220480775   Mean   :0.559   Mean   :0.5081   Mean   :0.6319
##  3rd Qu.:220587872   3rd Qu.:1.000   3rd Qu.:1.0000   3rd Qu.:1.0000
##  Max.   :220674325   Max.   :1.000   Max.   :1.0000   Max.   :1.0000
##   reactionTime      confidence       isCuedBlock          cue
##  Min.   :0.1983   Min.   :-1.0000   Min.   :0.0000   Min.   :-1.00000
##  1st Qu.:0.4322   1st Qu.:-1.0000   1st Qu.:0.0000   1st Qu.: 0.00000
##  Median :0.5677   Median :-1.0000   Median :0.0000   Median : 0.00000
##  Mean   :0.6970   Mean   :-0.5289   Mean   :0.4167   Mean   :-0.02083
##  3rd Qu.:0.8143   3rd Qu.: 0.0000   3rd Qu.:1.0000   3rd Qu.: 0.00000
##  Max.   :2.9974   Max.   : 1.0000   Max.   :1.0000   Max.   : 1.00000
##  orientationMean       contrast         variance      condition
##  Min.   :-26.67646   Min.   :0.1500   Min.   : 0.00   Length:864
##  1st Qu.: -5.95720   1st Qu.:0.1500   1st Qu.: 4.00   Class :character
##  Median :  0.15682   Median :0.1500   Median : 4.00   Mode  :character
##  Mean   :  0.08092   Mean   :0.3734   Mean   : 4.66
##  3rd Qu.:  5.95273   3rd Qu.:0.6000   3rd Qu.:10.00
##  Max.   : 25.98183   Max.   :0.6000   Max.   :10.00
```

6.3 Figure 2 from Herce Castañón et al.

6.3.1 Panel A

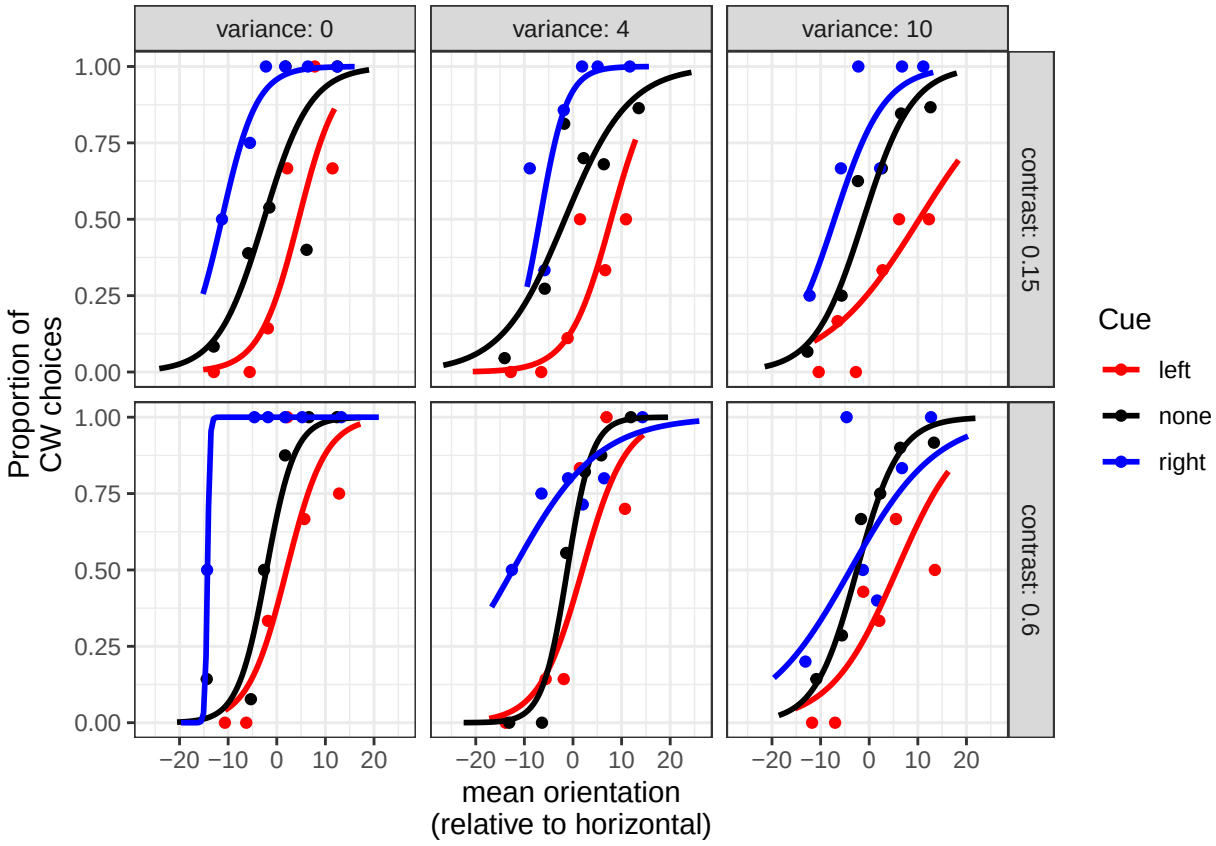
We begin by plotting the proportion of correct choices for *all* conditions:



6.3.2 Panel B

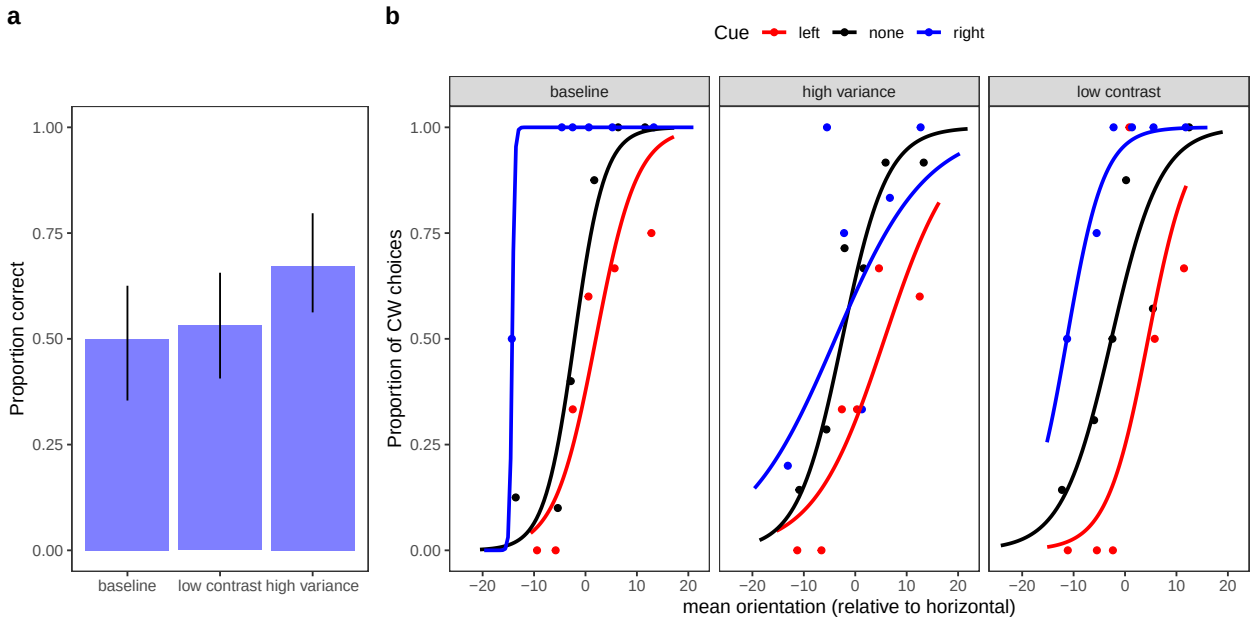
We begin by plotting the proportion of CW choices for *all* conditions:

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



6.3.3 Panel A and B together

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



7 Case Study III: improvements in visual perception after training (Huxlin group)

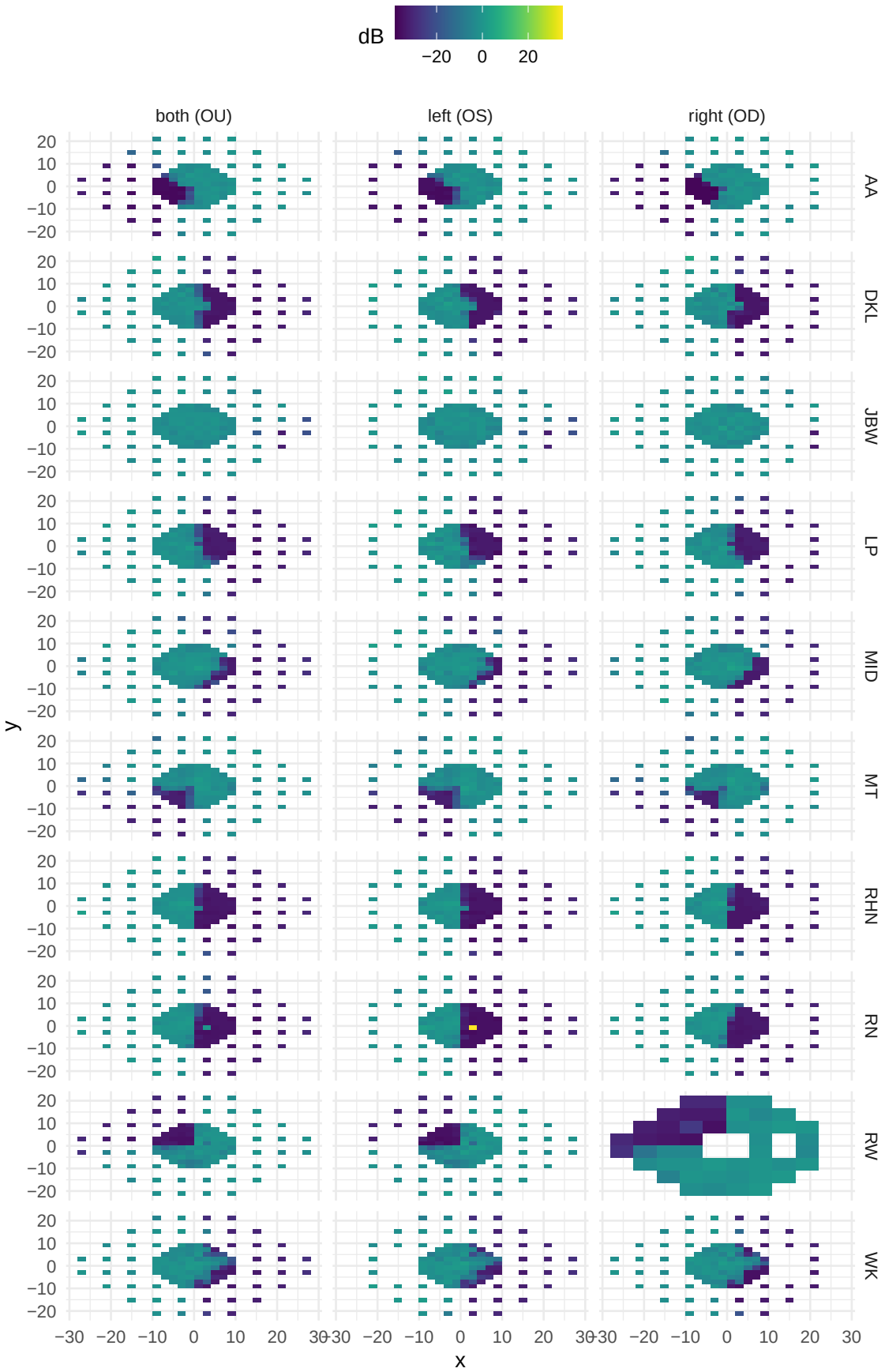
This study seeks to assess the improvement due to visual training after stroke-induced blindness.

7.1 Load data from Excel files

```
##     subject           test              eye             x
##  BN     : 1408   Min.   :1.00   both (OU) :6358   Min.   :-2.70e+01
##  JM     : 1056   1st Qu.:1.00   left (OS) :6084   1st Qu.:-5.00e+00
##  MID    : 1056   Median :2.00   right (OD):6080   Median :-1.00e+00
##  MS     : 1056   Mean   :1.83                     Mean   : 1.08e-04
##  RW     :  928   3rd Qu.:2.00                     3rd Qu.: 5.00e+00
##  AA     :  704   Max.   :8.00                     Max.   : 2.70e+01
##  (Other):12314
##        y                  dB
##  Min.   :-2.10e+01   Min.   :-44.00
##  1st Qu.:-5.00e+00   1st Qu.:-25.50
##  Median :-1.00e+00   Median : -2.50
##  Mean   :-6.48e-04   Mean   :-10.29
##  3rd Qu.: 5.00e+00   3rd Qu.: -1.00
##  Max.   : 2.10e+01   Max.   : 35.00
##
```

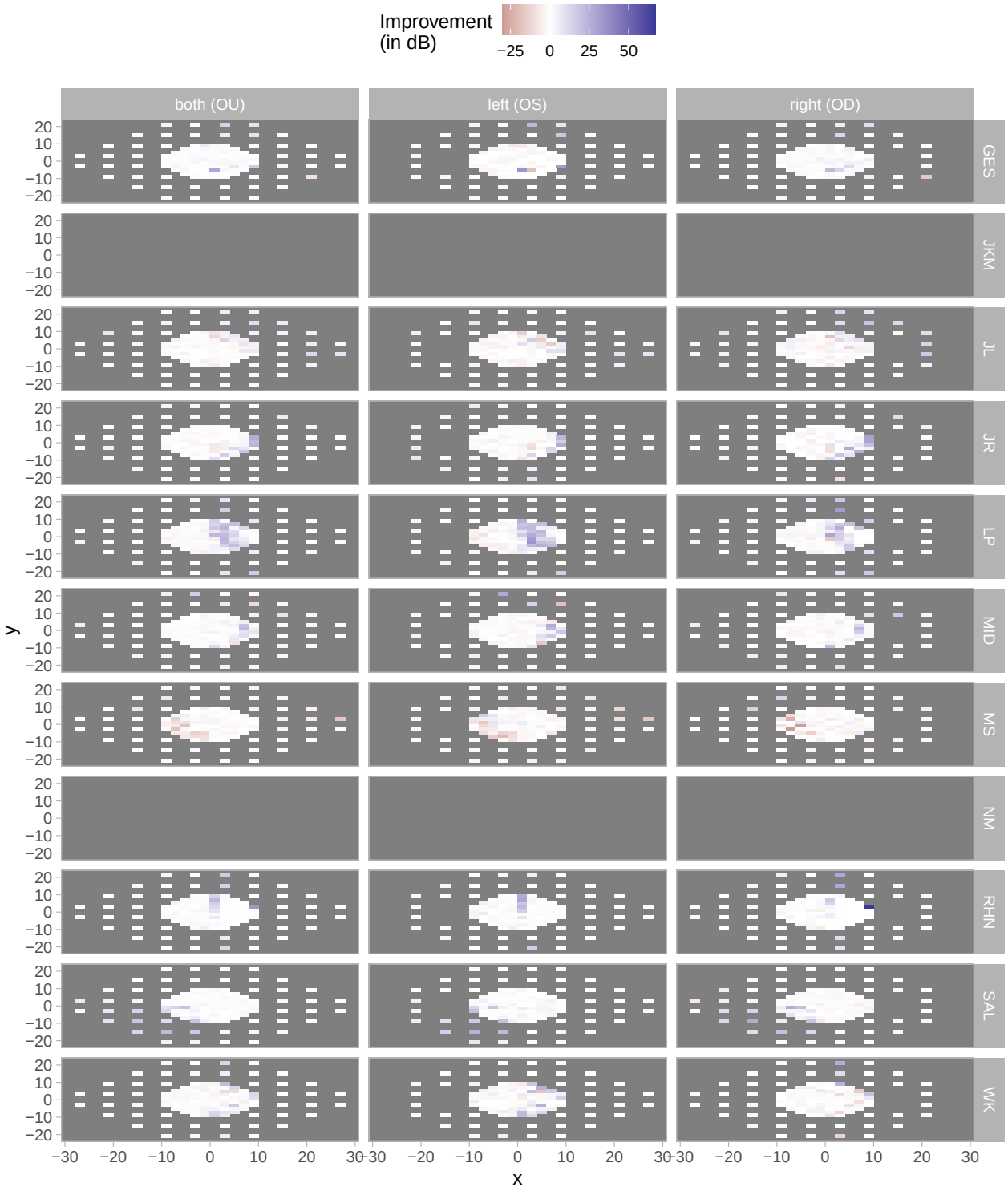
7.2 Visualizing visual fields

Let's visualize the first test sesion of a few example subjects. For this purpose, I randomly sample 10 subjects:


dB
−20
0
20
both (OU)
left (OS)
right (OD)
AA
DKL
JBW
LP
MID
MT
RHN
RN
RW
WK
y
x
20
10
0
−10
−20
−30
−20
−10
0
10
20
30

7.3 Visualizing improvements in visual fields over tests

Now let's look at the first and last test of each subject and subtract them from each other. For that, we first need to turn the data into wide format with the first and last test next to each other.



8 Session info

```
## - Session info ---------------------------------------------------------------
##  setting  value
##  version  R version 3.6.2 (2019-12-12)
##  os       macOS High Sierra 10.13.6
##  system   x86_64, darwin15.6.0
##  ui       X11
##  language (EN)
##  collate  en_US.UTF-8
##  ctype    en_US.UTF-8
##  tz       America/New_York
##  date     2020-03-24
##
## - Packages -------------------------------------------------------------------
##  package      * version date       lib source
##  acepack        1.4.1   2016-10-29 [1] CRAN (R 3.6.0)
##  assertthat     0.2.1   2019-03-21 [1] CRAN (R 3.6.0)
##  backports      1.1.5   2019-10-02 [1] CRAN (R 3.6.0)
##  base64enc      0.1-3   2015-07-28 [1] CRAN (R 3.6.0)
##  broom          0.5.5   2020-02-29 [1] CRAN (R 3.6.2)
##  callr          3.4.2   2020-02-12 [1] CRAN (R 3.6.0)
##  cellranger     1.1.0   2016-07-27 [1] CRAN (R 3.6.0)
##  checkmate      2.0.0   2020-02-06 [1] CRAN (R 3.6.0)
##  cli            2.0.2   2020-02-28 [1] CRAN (R 3.6.0)
##  cluster        2.1.0   2019-06-19 [1] CRAN (R 3.6.2)
##  codetools      0.2-16  2018-12-24 [1] CRAN (R 3.6.2)
##  colorspace     1.4-1   2019-03-18 [1] CRAN (R 3.6.0)
##  cowplot      * 1.0.0   2019-07-11 [1] CRAN (R 3.6.0)
##  crayon         1.3.4   2017-09-16 [1] CRAN (R 3.6.0)
##  data.table     1.12.8  2019-12-09 [1] CRAN (R 3.6.0)
##  DBI            1.1.0   2019-12-15 [1] CRAN (R 3.6.0)
##  dbplyr         1.4.2   2019-06-17 [1] CRAN (R 3.6.0)
##  desc           1.2.0   2018-05-01 [1] CRAN (R 3.6.0)
##  devtools       2.2.2   2020-02-17 [1] CRAN (R 3.6.0)
##  digest         0.6.25  2020-02-23 [1] CRAN (R 3.6.2)
##  dplyr        * 0.8.5   2020-03-07 [1] CRAN (R 3.6.0)
##  ellipsis       0.3.0   2019-09-20 [1] CRAN (R 3.6.0)
##  evaluate       0.14    2019-05-28 [1] CRAN (R 3.6.0)
##  fansi          0.4.1   2020-01-08 [1] CRAN (R 3.6.0)
##  farver         2.0.3   2020-01-16 [1] CRAN (R 3.6.0)
##  forcats      * 0.5.0   2020-03-01 [1] CRAN (R 3.6.2)
##  foreign        0.8-76  2020-03-03 [1] CRAN (R 3.6.0)
##  Formula        1.2-3   2018-05-03 [1] CRAN (R 3.6.0)
##  fs             1.3.2   2020-03-05 [1] CRAN (R 3.6.0)
##  generics       0.0.2   2018-11-29 [1] CRAN (R 3.6.0)
##  ggplot2      * 3.3.0   2020-03-05 [1] CRAN (R 3.6.0)
##  glue           1.3.2   2020-03-12 [1] CRAN (R 3.6.0)
##  gridExtra      2.3     2017-09-09 [1] CRAN (R 3.6.0)
##  gtable         0.3.0   2019-03-25 [1] CRAN (R 3.6.0)
##  haven          2.2.0   2019-11-08 [1] CRAN (R 3.6.0)
##  Hmisc          4.3-1   2020-02-07 [1] CRAN (R 3.6.0)
##  hms            0.5.3   2020-01-08 [1] CRAN (R 3.6.0)
##  htmlTable      1.13.3  2019-12-04 [1] CRAN (R 3.6.0)
```

```
##  htmltools      0.4.0   2019-10-04 [1] CRAN (R 3.6.0)
##  htmlwidgets    1.5.1   2019-10-08 [1] CRAN (R 3.6.0)
##  httr           1.4.1   2019-08-05 [1] CRAN (R 3.6.0)
##  jpeg           0.1-8.1 2019-10-24 [1] CRAN (R 3.6.0)
##  jsonlite       1.6.1   2020-02-02 [1] CRAN (R 3.6.0)
##  knitr          1.28    2020-02-06 [1] CRAN (R 3.6.0)
##  labeling       0.3     2014-08-23 [1] CRAN (R 3.6.0)
##  lattice        0.20-40 2020-02-19 [1] CRAN (R 3.6.0)
##  latticeExtra   0.6-29  2019-12-19 [1] CRAN (R 3.6.0)
##  lazyeval       0.2.2   2019-03-15 [1] CRAN (R 3.6.0)
##  lifecycle      0.2.0   2020-03-06 [1] CRAN (R 3.6.0)
##  lubridate      1.7.4   2018-04-11 [1] CRAN (R 3.6.0)
##  magrittr     * 1.5     2014-11-22 [1] CRAN (R 3.6.0)
##  Matrix         1.2-18  2019-11-27 [1] CRAN (R 3.6.2)
##  memoise        1.1.0   2017-04-21 [1] CRAN (R 3.6.0)
##  mgcv           1.8-31  2019-11-09 [1] CRAN (R 3.6.2)
##  modelr         0.1.6   2020-02-22 [1] CRAN (R 3.6.2)
##  munsell        0.5.0   2018-06-12 [1] CRAN (R 3.6.0)
##  nlme           3.1-145 2020-03-04 [1] CRAN (R 3.6.0)
##  nnet           7.3-13  2020-02-25 [1] CRAN (R 3.6.0)
##  openxlsx     * 4.1.4   2019-12-06 [1] CRAN (R 3.6.0)
##  pillar         1.4.3   2019-12-20 [1] CRAN (R 3.6.0)
##  pkgbuild       1.0.6   2019-10-09 [1] CRAN (R 3.6.0)
##  pkgconfig      2.0.3   2019-09-22 [1] CRAN (R 3.6.0)
##  pkgload        1.0.2   2018-10-29 [1] CRAN (R 3.6.0)
##  plotly       * 4.9.2   2020-02-12 [1] CRAN (R 3.6.0)
##  png            0.1-7   2013-12-03 [1] CRAN (R 3.6.0)
##  prettyunits    1.1.1   2020-01-24 [1] CRAN (R 3.6.0)
##  processx       3.4.2   2020-02-09 [1] CRAN (R 3.6.0)
##  ps             1.3.2   2020-02-13 [1] CRAN (R 3.6.0)
##  purrr        * 0.3.3   2019-10-18 [1] CRAN (R 3.6.0)
##  R.matlab     * 3.6.2   2018-09-27 [1] CRAN (R 3.6.0)
##  R.methodsS3    1.8.0   2020-02-14 [1] CRAN (R 3.6.0)
##  R.oo           1.23.0  2019-11-03 [1] CRAN (R 3.6.0)
##  R.utils        2.9.2   2019-12-08 [1] CRAN (R 3.6.0)
##  R6             2.4.1   2019-11-12 [1] CRAN (R 3.6.0)
##  RColorBrewer   1.1-2   2014-12-07 [1] CRAN (R 3.6.0)
##  Rcpp           1.0.4   2020-03-17 [1] CRAN (R 3.6.0)
##  readr        * 1.3.1   2018-12-21 [1] CRAN (R 3.6.0)
##  readxl       * 1.3.1   2019-03-13 [1] CRAN (R 3.6.0)
##  rematch        1.0.1   2016-04-21 [1] CRAN (R 3.6.0)
##  remotes        2.1.1   2020-02-15 [1] CRAN (R 3.6.0)
##  reprex         0.3.0   2019-05-16 [1] CRAN (R 3.6.0)
##  rlang          0.4.5   2020-03-01 [1] CRAN (R 3.6.0)
##  rmarkdown      2.1     2020-01-20 [1] CRAN (R 3.6.0)
##  rpart          4.1-15  2019-04-12 [1] CRAN (R 3.6.2)
##  rprojroot      1.3-2   2018-01-03 [1] CRAN (R 3.6.0)
##  rstudioapi     0.11    2020-02-07 [1] CRAN (R 3.6.0)
##  rvest          0.3.5   2019-11-08 [1] CRAN (R 3.6.0)
##  scales         1.1.0   2019-11-18 [1] CRAN (R 3.6.0)
##  sessioninfo    1.1.1   2018-11-05 [1] CRAN (R 3.6.0)
##  stringi        1.4.6   2020-02-17 [1] CRAN (R 3.6.0)
##  stringr      * 1.4.0   2019-02-10 [1] CRAN (R 3.6.0)
##  survival       3.1-11  2020-03-07 [1] CRAN (R 3.6.0)
```

```
##  testthat       2.3.2   2020-03-02 [1] CRAN (R 3.6.0)
##  tibble       * 2.1.3   2019-06-06 [1] CRAN (R 3.6.0)
##  tidyr        * 1.0.2   2020-01-24 [1] CRAN (R 3.6.0)
##  tidyselect     1.0.0   2020-01-27 [1] CRAN (R 3.6.0)
##  tidyverse    * 1.3.0   2019-11-21 [1] CRAN (R 3.6.0)
##  usethis        1.5.1   2019-07-04 [1] CRAN (R 3.6.0)
##  utf8           1.1.4   2018-05-24 [1] CRAN (R 3.6.0)
##  vctrs          0.2.4   2020-03-10 [1] CRAN (R 3.6.0)
##  viridis      * 0.5.1   2018-03-29 [1] CRAN (R 3.6.0)
##  viridisLite  * 0.3.0   2018-02-01 [1] CRAN (R 3.6.0)
##  withr          2.1.2   2018-03-15 [1] CRAN (R 3.6.0)
##  xfun           0.12    2020-01-13 [1] CRAN (R 3.6.0)
##  xml2           1.2.5   2020-03-11 [1] CRAN (R 3.6.0)
##  yaml           2.2.1   2020-02-01 [1] CRAN (R 3.6.0)
##  zip            2.0.4   2019-09-01 [1] CRAN (R 3.6.0)
##
## [1] /Library/Frameworks/R.framework/Versions/3.6/Resources/library
```