Research Report 2 - Sample Code and Notes

Professor Noyce

2025-09-10

Model

Finding a target in a visual scene requires some degree of **search time**, the time cost of examining each item in the display. The search time varies depending on the properties of the stimuli, and the total response time depends on the search time, the number of items in the display, and some time cost of recognizing the target and making a click response. And, of course, an error term.

We can write this model as

$$T_{\text{response}} = \left(T_{\text{search}} * \frac{N(\text{items})}{2} \right) + T_{\text{motor}} + \epsilon$$

where the critical thing to estimate is T_{search}, the time it takes to check each additional distractor. Notice that this has the classic structure of a linear regression equation $Y = (\beta_1 * x_1) + \beta_0$.

In order to compare the T_{search} of the red distractors and the black distractors, we could fit two separate models and compare the two coefficients. That is,

$$\begin{aligned} \text{Red trials only:} \quad & T_{\text{response}} = \left(T_{\text{search}} * \frac{N(\text{items})}{2} \right) + T_{\text{motor}} + \epsilon \\ \text{Black trials only:} \quad & T_{\text{response}} = \left(T_{\text{search}} * \frac{N(\text{items})}{2} \right) + T_{\text{motor}} + \epsilon \end{aligned}$$

and then we could compare the two fitted T_{search} terms to understand how they differ.

This is a good start, but there is a critical problem with it. Because we are fitting two models independently, we are allowing every parameter to vary between the two models. Not just the T_{search} coefficient, but the T_{motor} constant, and also the subject-by-subject ϵ are allowed to be different between our models. This is a problem for two reasons. First, this gives us six free parameters, leading to possible *overfitting* of our model. Second, we don't have a good theoretical basis for expecting the time it takes to make a motor response, or the subject-by-subject variability in speed, to be different between the two distractor conditions. Instead, we want a model that *constrains* these parameters to be fixed for both colors.

We need to create a single linear model that includes both the $\frac{N(\text{items})}{2}$ term as well as terms capturing the **main effect** of condition, as well as any **interaction** between condition and search time.

Our one-color model already has a term for **main effect** of number of items:

$$\left(T_{\text{search\_n\_items}} * \frac{N(\text{items})}{2} \right)$$

We'll also add a term for the **main effect** of condition (See the ANOVA hw for review of dummy coding and categorical predictors.):

$$\left(T_{\text{search\_condition}} * \text{condition}\right)$$

Finally, we need to allow the two predictors to interact. We want to see whether letting the search time per item differ between the two conditions improves the fit of the model.

$$\left(T_{\text{search\_n:condition}} * \frac{N(\text{items})}{2} * \text{condition}\right)$$

This last term accommodates a slope shift between the reference level of the condition variable and the other level. Our final model to fit is:

$$\begin{aligned} T_{\text{response}} = & \left(T_{\text{search\_n\_items}} * \frac{N(\text{items})}{2}\right) + \\ & \left(T_{\text{search\_condition}} * \text{condition}\right) + \\ & \left(T_{\text{search\_n:condition}} * \frac{N(\text{items})}{2} * \text{condition}\right) + \\ & T_{\text{motor}} + \epsilon \end{aligned}$$

Get organized

Create a new directory a new R project for this lab. Load any necessary packages.

```
library(tidyverse) # tidy data
library(here) # easier file management
here() # check that the working dir is the project dir
```

```
## [1] "/Users/anoyce/Documents/2_Areas/Teaching/Cognitive RM 85-310 f25/lab-2"
```

```
library(performance) # model checks
```

Load data file and clean it up

Take a peek at the raw data from Gorilla, either by opening the .csv file in a spreadsheet application, or by loading it into R.

NOTE: this sample code uses the dataset from a prior semester, your numbers should look different from my numbers!

```
datafile <- here("1-data", "data_exp_207534-v1_task-ljfs.csv") # make sure to set your d
atafile name appropriately!

raw_data <- read.csv(datafile)

# head, glimpse, or summarize are all useful. Or you can look at it directly in RStudio!
```

Select the relevant columns

This dataframe has columns for every single thing we might possibly want from the online experiment. However, for our analysis, we only need the model terms above.

- Individual participant ID codes: `Participant.Private.ID`
- Reaction times on each trial: `Reaction.Time`
- Information about distractor colors on each trial: `Spreadsheet..distractor_color`
- Information about the number of distractors on each trial: `Spreadsheet..Distractor.count`

Sometimes platforms like Gorilla change the name of these columns; if these specific column names don't work, look at your data file and figure out what needs to change.

Because Gorilla creates a spreadsheet line for everything that happened in the experiment, we're going to want a few more columns to let us filter for only rows that actually include a trial response:

- Trial number (mostly for a sanity check): `Trial.Number`
- Whether the event was a response click or something else (like a fixation timeout): `Response.Type` or `Component.Name` both will work for this.

We'll use `select()` (just like in the data wrangling hw) to extract just those columns. Below, I specified the first two variables; fill in the others.

```
data_selected <- raw_data %>%

  select(Participant.Private.ID,
         Reaction.Time,

         # Fill in the rest of the variables yourself

         )
```

Filter the relevant rows

Looking at the data, we see that we have multiple rows for each trial number! One of them has `Response.Type` "continue" (and `Component.Name` "Fixation Timing"); the other has `Response.Type` "response" (and `Component.Name` "Click Response"). We want to `filter()` for only the "response" / "Click Response" rows.

After running this block, check that you have one line of data per trial number per participant ID.

```
data_filtered <- data_selected %>%

  filter(____ == "_____") # remember that filter() uses 'variable-name = "value"' syntax
```

Rename variables (for quality of life)

We will be happier if we `rename()` these variables to be something less annoying to type out a zillion times. Rename's syntax is very simple: newname = oldname.

```
data_renamed <- data_filtered %>%

  rename(id = Participant.Private.ID,
         # fill in any others you'd like
  )
```

Transform *distractor count* to *items searched*

In a serial search process, in order to find the target, you have to look at 50% of the items on the screen on average. (Some trials, you will find the target very early; others you will find it very late. But it works out to half, on average.) The number of items on the screen is the number of distractors plus the target. We will use `mutate()` to create a new variable called `items_searched` .

```
data_new_var <- data_renamed %>%

  mutate(items_searched = (___ + 1) / 2 # newvariable = formula, fill in the column with
the count of distractors
         )
```

Set factors

Finally, we want to tell R that the participant ID and distractor color are categorical variables, aka factors. Model interpretation will be easier later if we set the first level of distractor color to be “Red” (rather than R’s default of setting levels in alphabetical order).[1]

```
data <- data_new_var %>%

  # note that if you renamed this above, you'll need to change the column name here too.
  mutate(participant_id = as.factor(participant_id)
    Spreadsheet..distractor_color = as.factor(Spreadsheet..distractor_color),
         Spreadsheet..distractor_color = fct_relevel(Spreadsheet..distractor_color, "Re
d")
         )
```

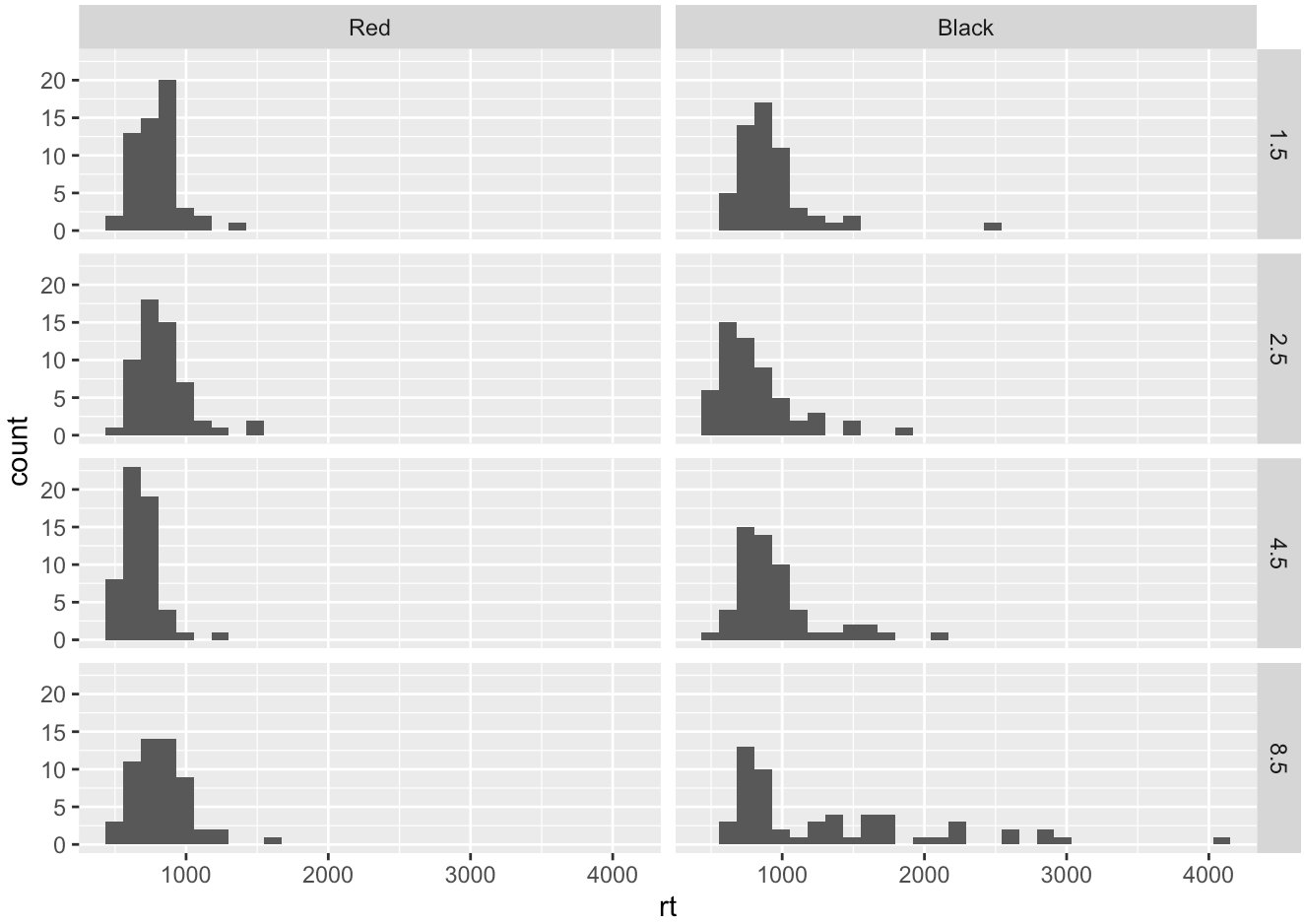
In real life, I’d combine all of these steps into a single pipe (and do them in a slightly different order):

```
data <- raw_data %>%

  filter(___ == "____") %>%

  rename(id = Participant.Private.ID,
         # any other renaming
  ) %>%

  select(id, # Select only model-relevant columns
         # also RT, distractor count, distractor color
         ) %>%

  # Specify factors
  mutate(participant_id = as.factor(participant_id),
         Spreadsheet..distractor_color = as.factor(Spreadsheet..distractor_color),
         Spreadsheet..distractor_color = fct_relevel(Spreadsheet..distractor_color,"Re
d"),
         items_searched = (Spreadsheet..Distractor.count + 1) / 2
  )
```

Initial data visualization

We always always always want to look at our data first thing. Let's make a quick histogram for each condition. Remember, `aes` maps data dimensions onto visualization dimensions. We want to set the reaction time on the X axis, and use `facet_grid` to break up the results by different values of distractor count and color.

```
ggplot(data=data, aes(x = _____)) + # fill in your RT variable
  facet_grid(_____ ~ ______) + # fill in your items_searched and distractor_color column
names
  geom_histogram()
```



You might notice:

- RT distributions are often skewed, with long positive tails from a few slow trials.
- Most trials are under 2000 ms to respond, but a few are longer, and 1 or 2 may be much longer.
- Those slower trials tend to be when the distractors are Black, and the number of distractors is large.

Summarize across trials for each participant

Right now, each trial for each participant is a single datapoint. However, we want to summarize across trials and get each participant's median RT in each condition.

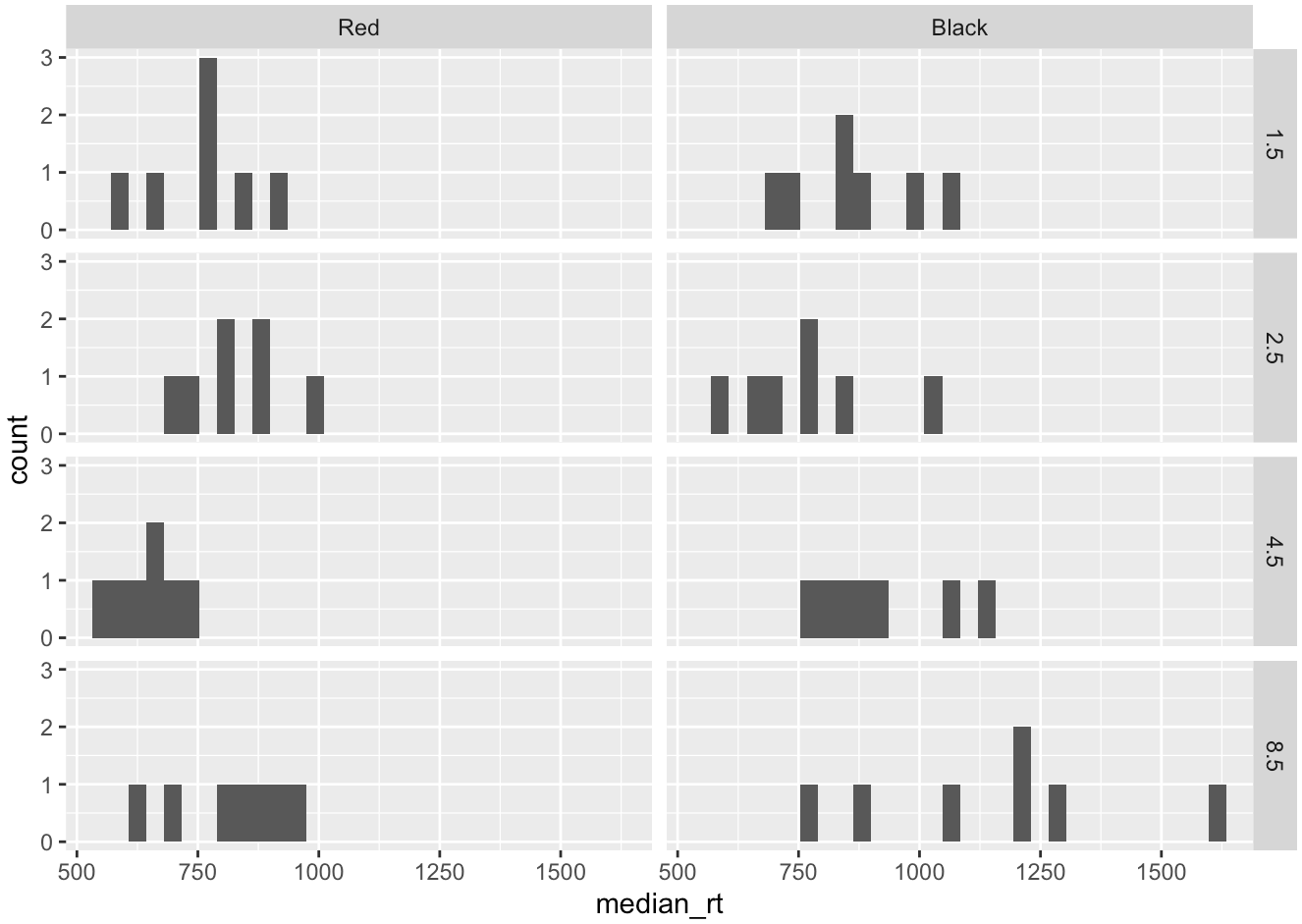
(We'll call this clean_data since it's the point at which we can start running our analyses.)

```
clean_data <- data %>%

  group_by(___, # participant ID
           ___, # distractor color
           ___  # items searched
           ) %>%

  summarize(median_rt = median(___)) %>% # reaction time

  ungroup()
```

Our `clean_data` data frame should have 1/8 as many rows as `data`, or 1/16 as many as our initial `raw_data` input. Now we have one estimate per subject per condition, and we can do some analyses.

Interim data visualization

Let's look at the `clean_data` histograms. We can use a code snippet very similar to the above, with 2 changes in the `ggplot()` call: (1) we need to use our updated data with `data = clean_data`, and (2) we need to use our updated RT variable with `aes(x = median_rt)`.



Hopefully, any wild outliers have been mitigated by taking the median, and these data look plausible. Take a minute to think about our upcoming analysis. We're hoping to estimate the **search time**, the amount that the RT changes with the number of distractors. How much does the search time appear to change with red distractors? With black distractors? Do you anticipate observing the interaction we hypothesized?

Descriptive stats

To get the mean and standard deviations that we would want for a Results section, we first `group_by()` distractor color (our categorical predictor), then `summarize()` with the mean, standard deviation, and n.

```
descriptives <- clean_data %>%

  group_by(____ # distractor color
           ) %>%

  summarize(mean_rt = mean(median_rt),
            sd_rt = sd(median_rt),
            n = n()
            ) %>%

  ungroup()

# Print out the resulting descriptive stats table
descriptives
```

distractor_color <fct>	**mean_rt** <dbl>	**sd_rt** <dbl>	**n** <int>
Red	764.9947	116.1579	28
Black	926.9714	224.2365	28

2 rows

Fitting linear models

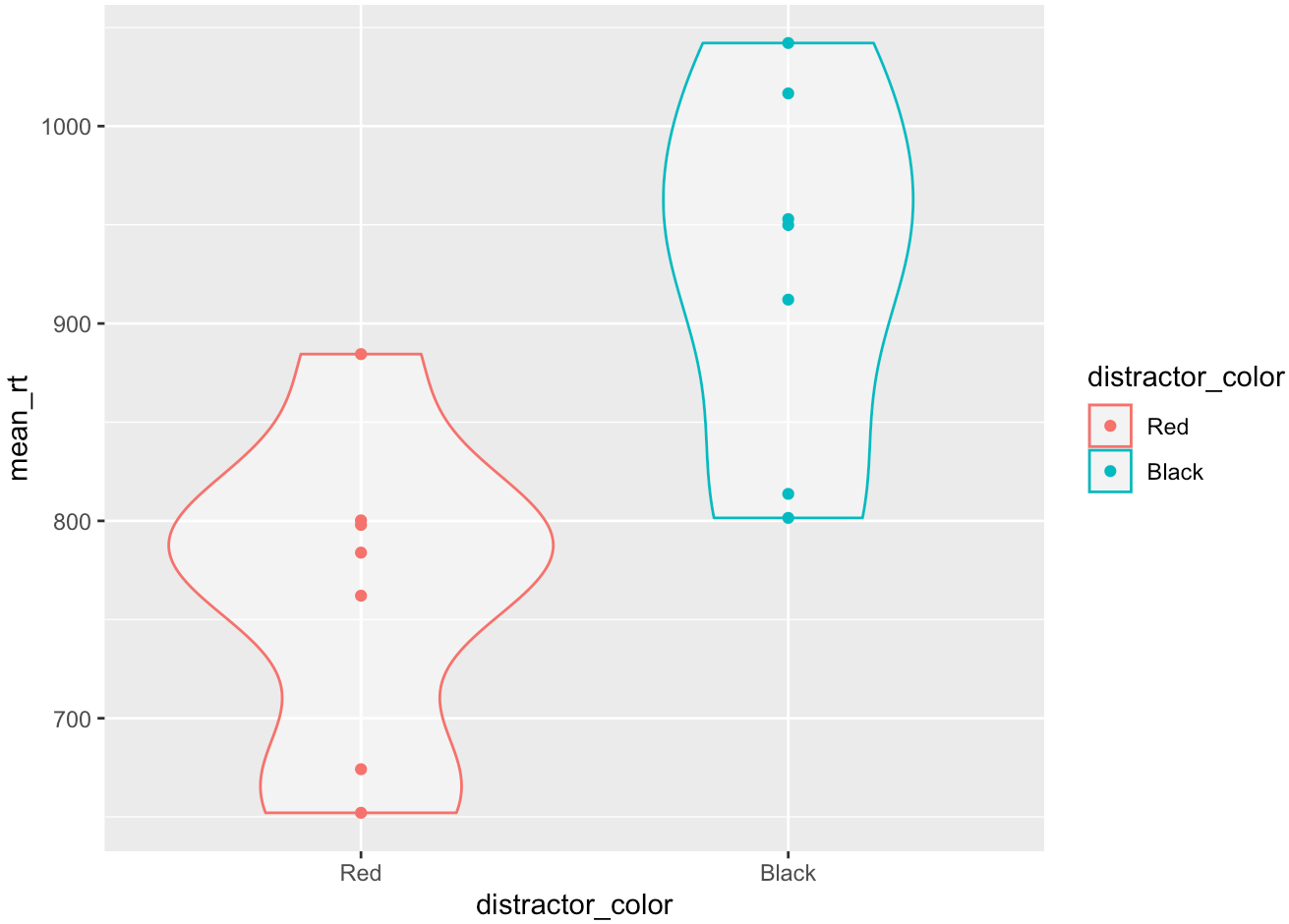
Before we fit the full model, we'll step through the componenets individually. Not all of these may make it into your final research report, but they are useful to help understand the data.

Simplified: one-way effect of color (t-test)

To ask whether, overall, Red and Black distractors lead to different reaction times, we need to first get the average reaction time for each participant and each color. We'll use `group_by()` and `summarize()`.

```
clean_data_colortest <- clean_data %>%

  group_by(___, # keep participant ID
           ___. # keep color
           ) %>%

  summarize(mean_rt = mean(___)) # the RT column in your clean_data
```

Here's a look at the summarized data.



Now we run the t-test, just like we did with the data HW.

```
t.test(formula = ___ ~ ___, # outcome ~ predictor
       data = clean_data_colortest
       )
```

```
##
##  Welch Two Sample t-test
##
## data:  mean_rt by distractor_color
## t = -3.5136, df = 11.739, p-value = 0.004412
## alternative hypothesis: true difference in means between group Red and group Black is
not equal to 0
## 95 percent confidence interval:
##  -262.66802  -61.28548
## sample estimates:
##   mean in group Red mean in group Black
##            764.9947            926.9714
```

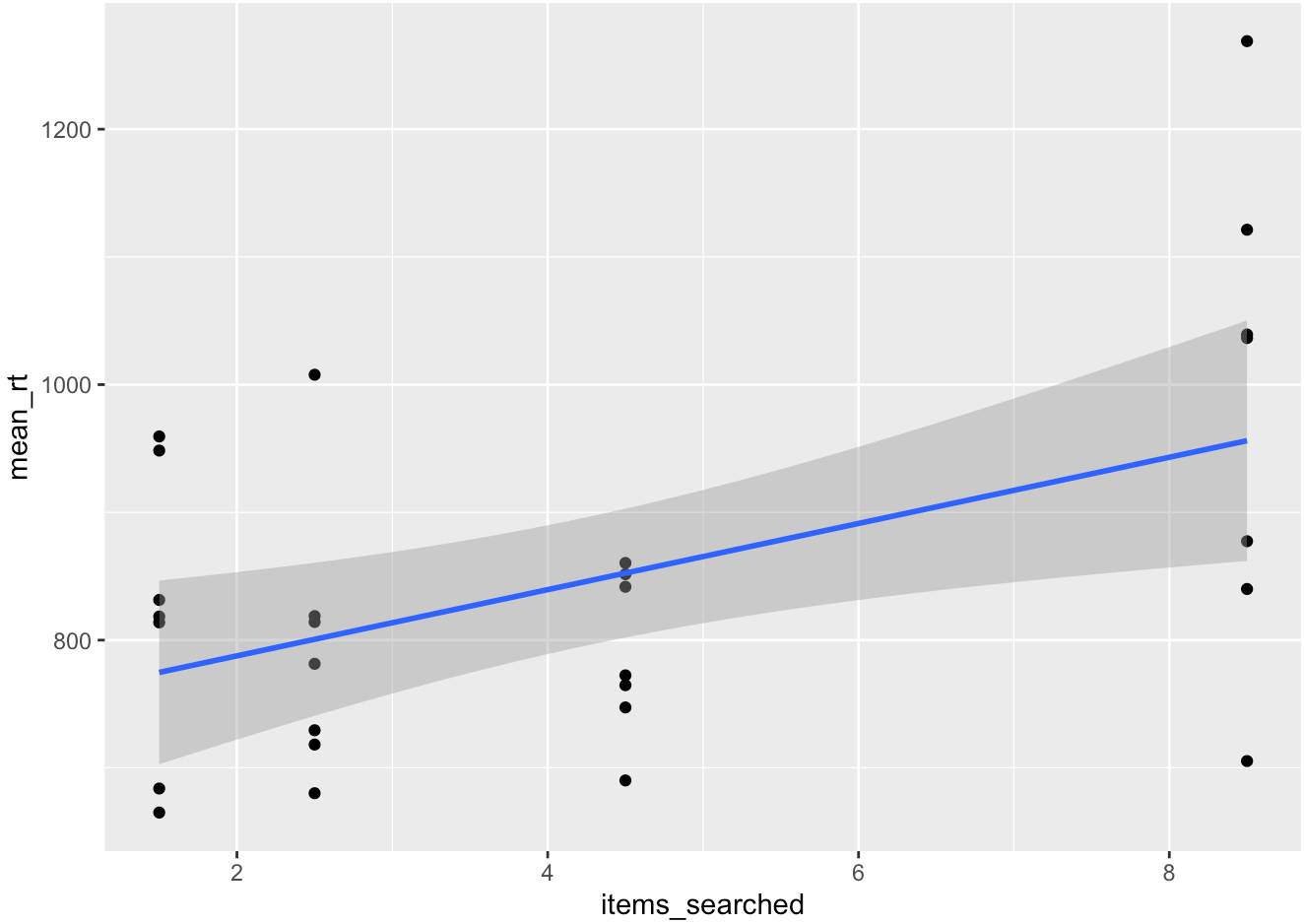
Make sure you understand this output! Is there a significant difference between colors? Which one has faster reaction times? How much faster? (Looking at the graph above can help with interpreting the direction of the effect.)

Simplified: one-way effect of item count (regression)

To ask whether, overall, the number of distractors changes the reaction times, we need to first get the average reaction time for each participant and each distractor count. We'll again use `group_by()` and `summarize()`.

```
clean_data_counttest <- clean_data %>%

  group_by(___, # keep participant id
           ___  # keep search count
           ) %>%

  summarize(mean_rt = mean(___) # take the mean reaction time
            ) %>%

  ungroup()
```

Again, here's a quick look at the summarized data. (GGplot can show a linear fit directly on the plot, that's handy!)



To ask R for the stats on that fitted linear model, we use the same syntax as was in the reading:

```
model_count <- lm(data = clean_data_counttest,
                     formula = ___ ~ ___ # output variable ~ predictor variable
                            )

summary(model_count)
```

```
##
## Call:
## lm(formula = mean_rt ~ items_searched, data = clean_data_counttest)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -250.792  -88.542   -5.787   62.637  312.808
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)     735.849     45.773  16.076 5.04e-15 ***
## items_searched   25.914      9.109   2.845  0.00855 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 129.2 on 26 degrees of freedom
## Multiple R-squared:  0.2374, Adjusted R-squared:  0.208
## F-statistic: 8.093 on 1 and 26 DF,  p-value: 0.008549
```

Make sure you understand this output! What is the effect of the items searched (in ms per additional item)? Is it significant? What does the intercept represent? (Looking at the graph above can help with interpreting these effects.)

Simplified: simple one-way effects of item count (regression)

So far, we've seen that color and distractor count both affect the reaction time. Our final goal is to model them simultaneously, but in order to help you make sense of that, we will first look at the model with JUST red, or JUST black trials.

This means the model call will look a lot like the one above, but instead of feeding it the *average* reaction time *across* colors, we will feed it the *single* reaction time from *one* color.

This is a **simple main effect**, the effect of one variable at a single level of the other variable(s).

Simple effect of item count at Red level

We'll use `filter()` to get just the Red trials.

```
clean_data_onlyred <- clean_data %>%
  filter(___ == "___") # variable-name == "value"

model_onlyred <- lm(data = clean_data_onlyred,
                    formula = ___ ~ ___ # outcome ~ predictor
                    )

summary(model_onlyred)
```

```
##
## Call:
## lm(formula = median_rt ~ items_searched, data = clean_data_onlyred)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -207.92  -92.76   14.70   79.86  225.73
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)     752.195     41.823  17.985 3.44e-16 ***
## items_searched    3.012      8.323   0.362     0.72
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 118.1 on 26 degrees of freedom
## Multiple R-squared:  0.005011,   Adjusted R-squared:  -0.03326
## F-statistic: 0.1309 on 1 and 26 DF,  p-value: 0.7204
```

Remember back to our model specification above:

$$T_{\text{response}} = \left(T_{\text{search}} * \frac{N(\text{items})}{2} \right) + T_{\text{motor}} + \epsilon$$

Make sure you understand how the values in the Coefficients table map to the T_{search} and T_{motor} terms. What is the estimated search time, and the estimated motor response time?

Simple effect of item count at Black level

Same thing, but for just the black trials.

```
clean_data_onlyblack <- clean_data %>%
  filter(___ == "___") # variable-name == "value"

model_onlyblack <- lm(data = clean_data_onlyblack,
                    formula = ___ ~ ___ # outcome ~ predictor
                    )

summary(model_onlyblack)
```

```
##
## Call:
## lm(formula = median_rt ~ items_searched, data = clean_data_onlyblack)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -358.79 -101.68  -37.75  106.33  488.66
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)      719.50      65.09  11.054 2.53e-11 ***
## items_searched    48.82      12.95   3.768 0.000853 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 183.8 on 26 degrees of freedom
## Multiple R-squared:  0.3532, Adjusted R-squared:  0.3284
## F-statistic:  14.2 on 1 and 26 DF,  p-value: 0.0008527
```

Looking between the “onlyred” and “onlyblack” models, is the estimate for T_{motor} similar between them? How about the estimate for T_{search}?

Full model

Remember our model specification is

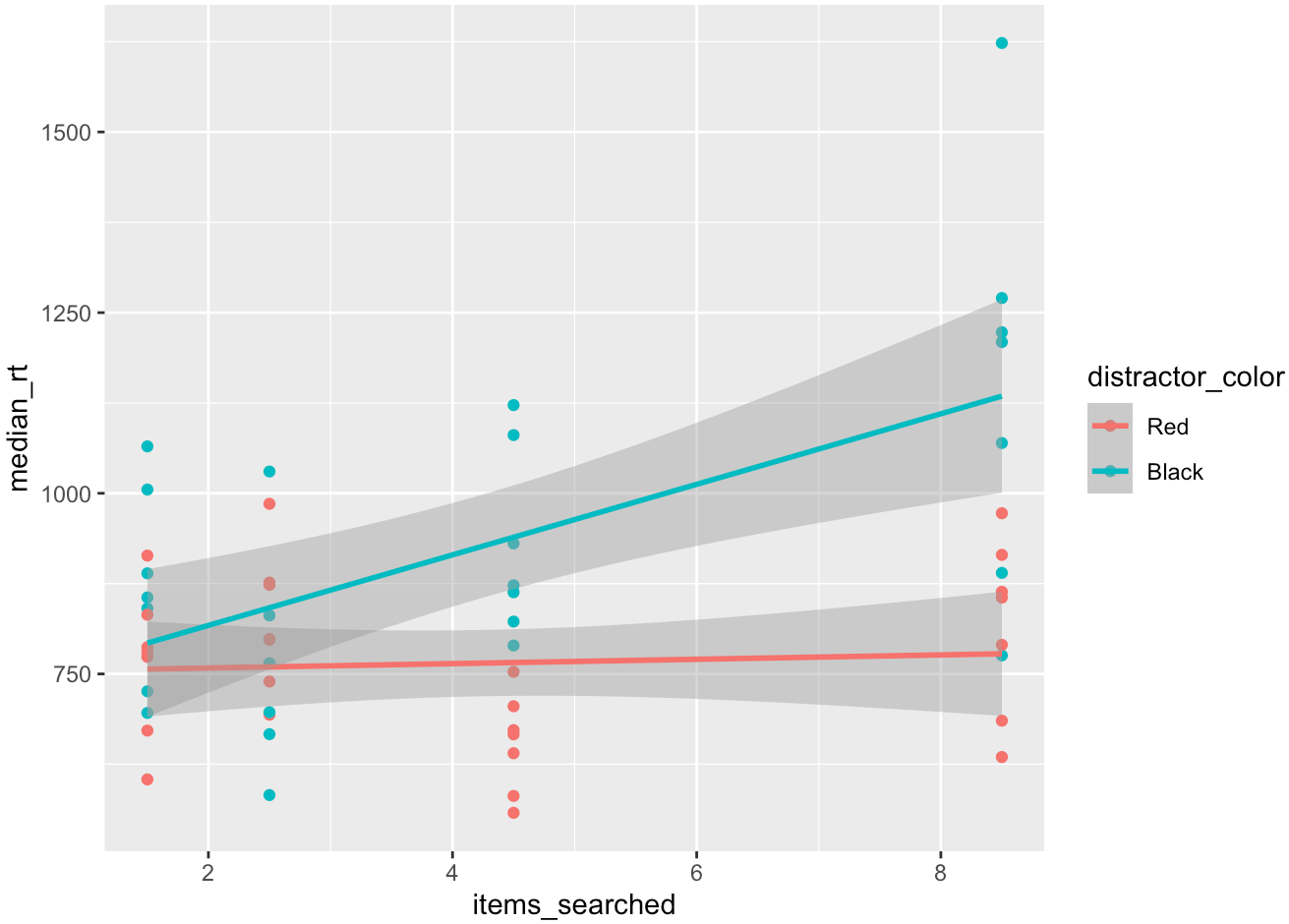
$$
\begin{aligned}
T_{\text{response}} = & \left(T_{\text{search\_n\_items}} * \frac{N(\text{items})}{2} \right) + \\
& \left(T_{\text{search\_condition}} * \text{condition} \right) + \\
& \left(T_{\text{search\_n:condition}} * \frac{N(\text{items})}{2} * \text{condition} \right) + \\
& T_{\text{motor}} + \epsilon
\end{aligned}
$$

Data visualization

As we did above, let’s first look at these data using ggplot’s built-in linear model tools, so that we know what to expect.

```
# Remember that you will likely have different column names if you try this yourself!
ggplot(data = clean_data, aes(x = items_searched, y = median_rt, color = distractor_colo
r)) +
  geom_point() +
  geom_smooth(method = "lm")
```

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



Model fitting

We've got a continuous predictor (our `items_searched` variable) and a categorical predictor (our `distractor_color` variable).

CAUTION: By default, R codes categorical predictors by what's called *treatment coding*. One level is set to be the baseline (way back at the top, we set Red as the baseline), and the model reports *simple main effects* of the other predictors *at that level*. We need to pay attention to detail when interpreting the output.

```
model_full <- lm(data = clean_data,
                 # formula = outcome ~ predictor1 + predictor2 + predictor1:predictor2
                 formula =  _____
)

summary(model_full)
```

Interpretting output

```
##
## Call:
## lm(formula = median_rt ~ distractor_color + items_searched +
##     distractor_color:items_searched, data = clean_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -358.79  -94.33   -9.58   90.44  488.66
##
## Coefficients:
##                                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)                        752.195     54.709  13.749  < 2e-16 ***
## distractor_colorBlack              -32.691     77.371  -0.423  0.67439
## items_searched                       3.012     10.888   0.277  0.78317
## distractor_colorBlack:items_searched  45.804     15.397   2.975  0.00444 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 154.5 on 52 degrees of freedom
## Multiple R-squared:  0.4062, Adjusted R-squared:  0.372
## F-statistic: 11.86 on 3 and 52 DF,  p-value: 4.964e-06
```

There's a lot going on in this output! By default, R reports these fits as the values when all other predictors are set to zero. Above, with only one predictor variable, that wasn't too bad, but here we will need to be careful.

`(Intercept)` gives the estimated response time when both predictors are "zero" (that is, our categorical predictor is at its baseline value, in this case Red, and the items searched is also 0).

`distractor_colorBlack` gives the estimated *increase in response time* when the color is Black and number of items searched is 0. That is, it's the difference in intercepts between the Red and Back distractors.

`items_searched` gives the estimated *search time per item* when `distractor_color` is baseline (Red).

`distractor_colorBlack:items_searched` gives the estimated *change in search time per item* when the distractor_color becomes Black.

That is, when distractors are Red, the estimated search cost is in the `items_searched` row, and when distractors are Black, the estimated search cost is the SUM of the `items_searched` row and the `distractor_colorBlack:items_searched` row.

These values should be very close to those that we saw when fitting the Red and Black models separately, above.

Constrained model

Wait a second. The full model allows for different intercepts between the Red and the Black colors, and the intercept corresponds to the motor response time, and above we decided to *constrain* the intercept to be identical between the two colors. That is, we need to remove one term from the model:

```
model_constrained <- lm(data = clean_data,
                        # formula = output ~ predictor1 + predictor1:predictor2
                  formula =  ____
)

summary(model_constrained)
```

```
##
## Call:
## lm(formula = median_rt ~ items_searched + distractor_color:items_searched,
##     data = clean_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -351.75  -96.42   -9.84   89.42  495.70
##
## Coefficients:
##                                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)                          735.849     38.384  19.171  < 2e-16 ***
## items_searched                         5.763      8.658   0.666    0.509
## items_searched:distractor_colorBlack  40.302      8.151   4.944 8.07e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 153.3 on 53 degrees of freedom
## Multiple R-squared:  0.4042, Adjusted R-squared:  0.3817
## F-statistic: 17.98 on 2 and 53 DF,  p-value: 1.097e-06
```

In this output table:

`(Intercept)` again gives the estimated response time when `items_searched` is zero, for any `distractor_color`.

`items_searched` again gives the estimated *search time per item* when `distractor_color` is "zero" (Red).

`distractor_colorBlack:items_searched` again gives the estimated *change in search time per item* when the distractor_color becomes Black.

By looking at the p value associated with each line, we can see whether this effect in our data is statistically significant.

Model comparisons

By design, `model_constrained` is a better fit to our *theoretical* specification of this system. But is it a better fit to the data? Or are we throwing out an important source of variability when we drop that predictor? To answer this question, there are formal model comparison tools that we will explore in our next lab. For now, we'll just look at the R^2 and adjusted R^2 values from the model output.

The bottom of each models summary includes "Multiple R-squared," which is an estimate of the proportion of the variance in the dataset that is explained by the model. It also gives "Adjusted R-squared," which scales that value down somewhat for each free parameter in the model. This helps penalize models that have a high potential for overfitting.

A quick and dirty approach to comparing models is to see which model has a higher Adjusted R-squared value. That model is accounting for more of the variance per fitted free parameter.

---

1. Why? Because I have a strong belief that the fitted slope for Red trials will be flatter, *less steep*, than the fitted slope for Black trials, and most humans have an easier time thinking about “adding steepness” than “subtracting steepness”.↩