Intro

In this project, I will be working with two datasets from two different research projects.

The first dataset contains eye-tracking files generated using Eye-Link Data Viewer. Unfortunately, I was unable to gather sufficient participants in time for hypothesis testing. However, the data required extensive preprocessing due to its complexity and richness, making the processing itself a challenge. Although I could not utilize it for the current project's analysis chapter, I developed preprocessing and visualization scripts that I will reuse in this project.

The second dataset contains behavioral data that I began analyzing earlier this semester. I had already extracted features from this dataset, making it nearly ready for hypothesis testing, though some preprocessing was still required since the data was not meant for analysis in R.

*If the R course team would like to grade this project by 1 part only - skip to part 2.

Part 1

Getting to know the eye movements dataset

This dataset is a standard eye tracking dataset, and has two inter-trial factors:

Trial type: Passive viewing / recall test

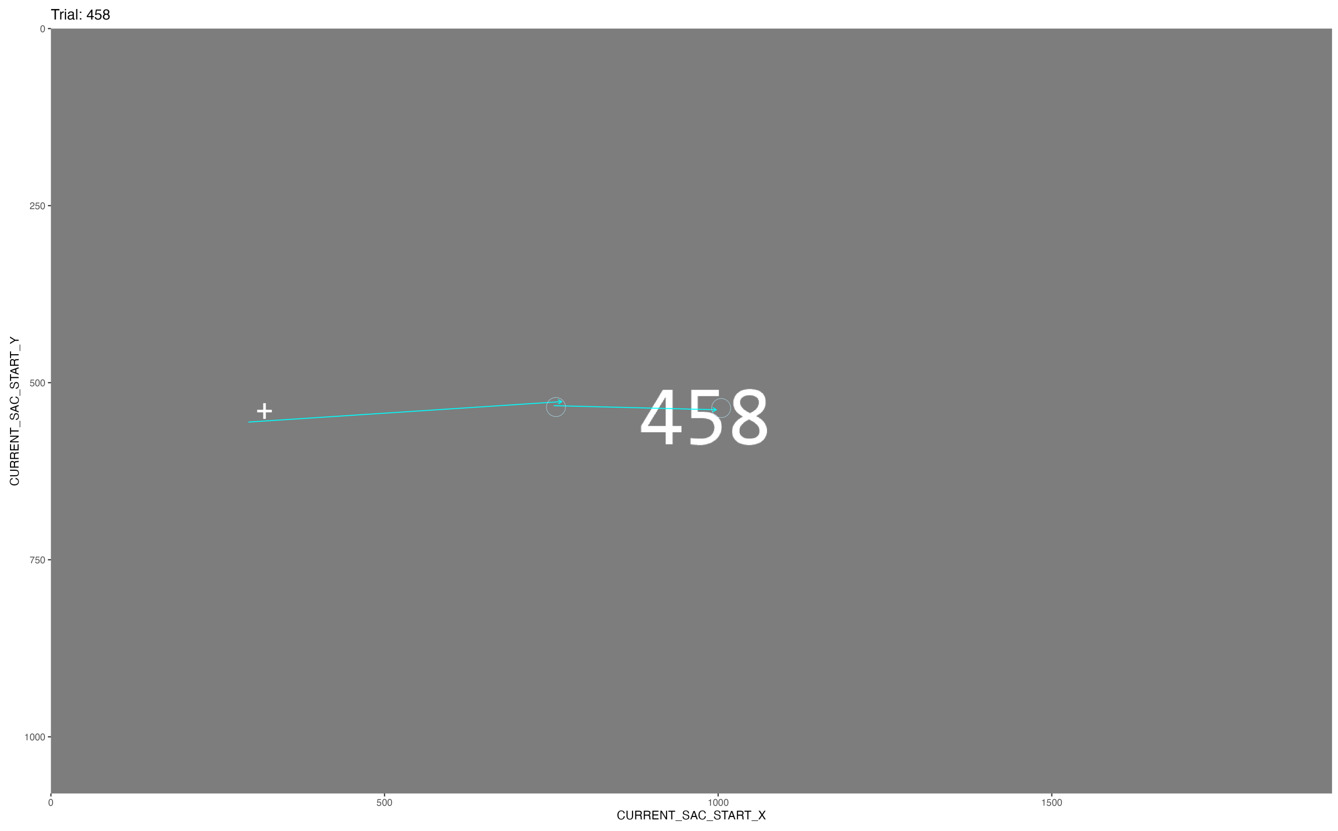
Number of digits: 3,4,5,6

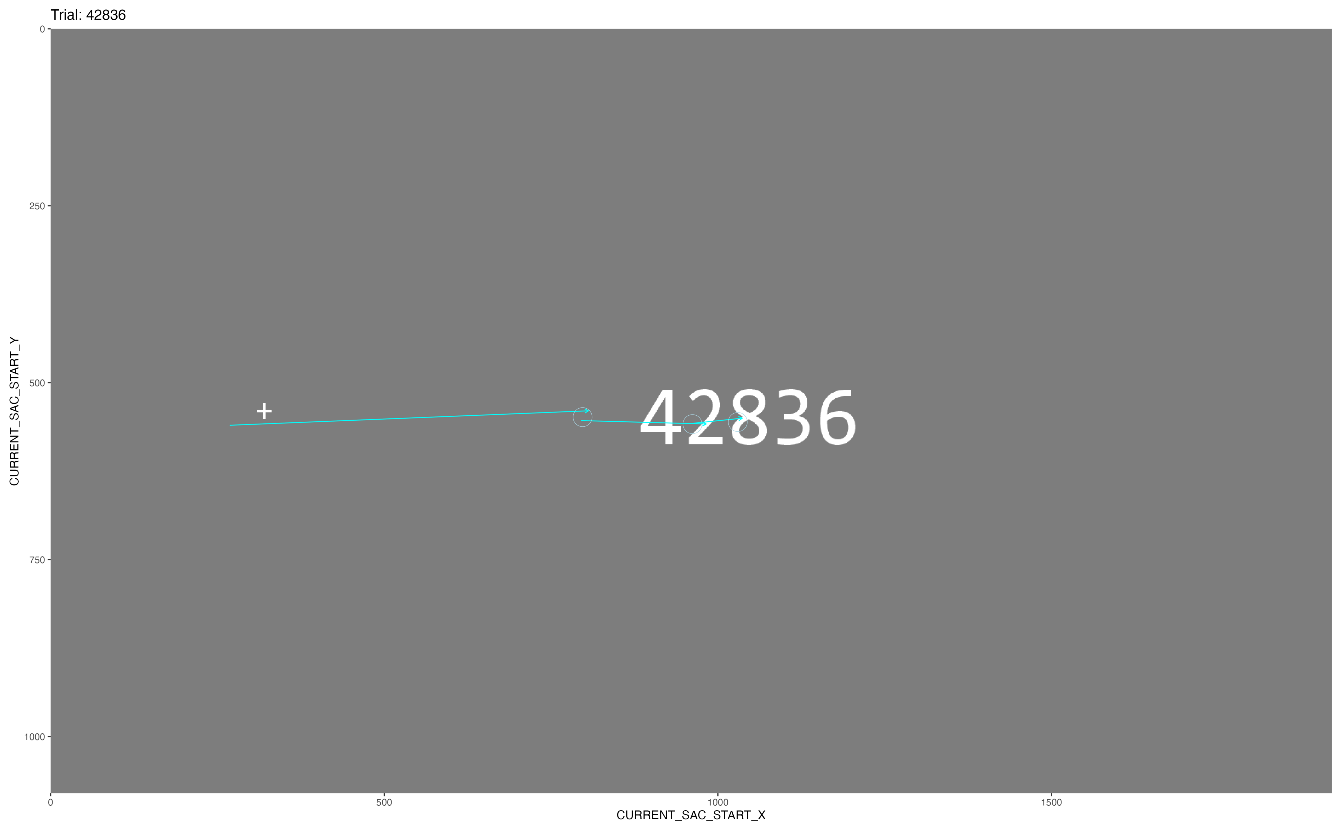
I would've liked to test how fixation index correlates with fixation location. And whether fixation location can be used to classify stimulus length. But I ended up not doing any analyses on this dataset.

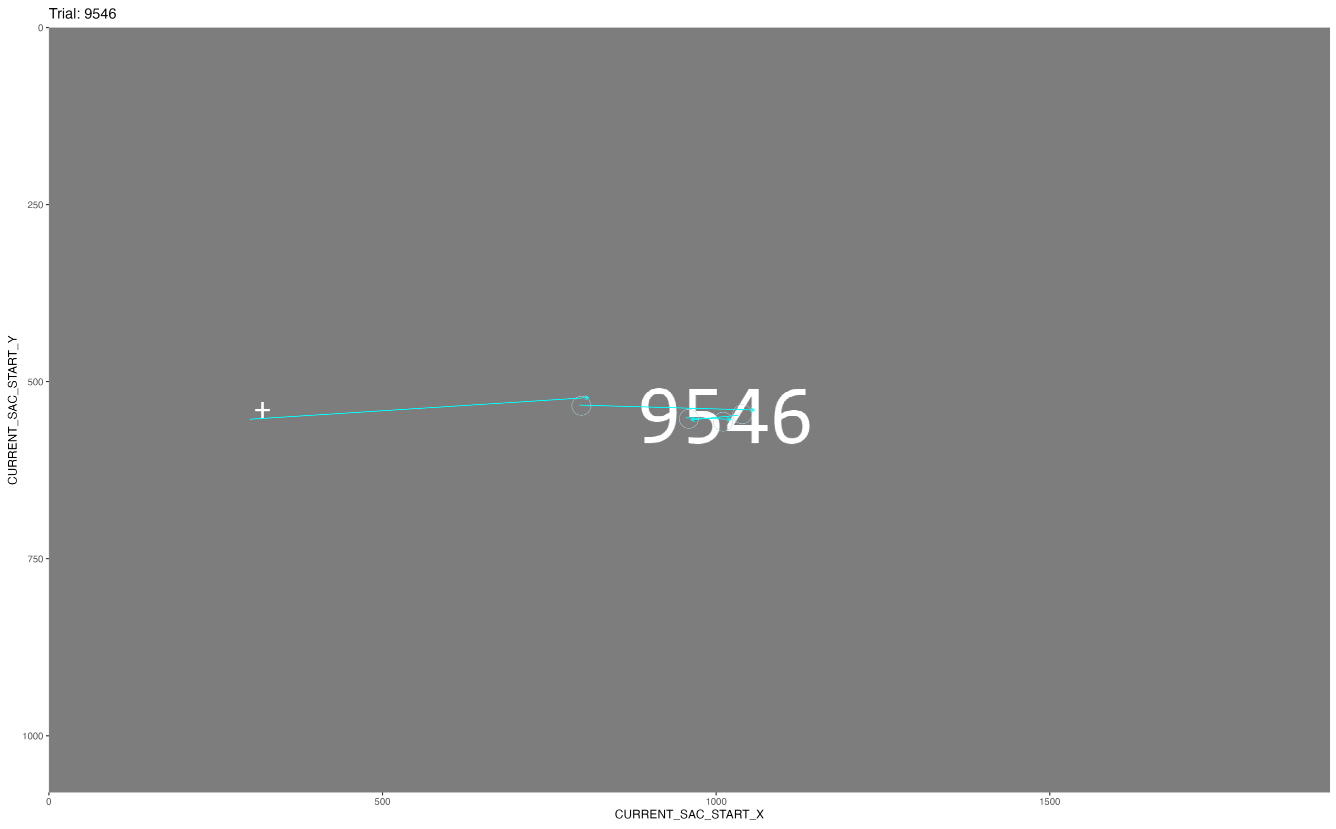
To better understand the dataset, my first goal was to get an intuition for the number of events a single trial may contain. In addition, I wanted to visualise the sequence of these events, so that the process of perceiving a single stimulus could be traced begining to end. Therefore, my first visualisation of the data is as **single trials**, saccades and fixations over time.

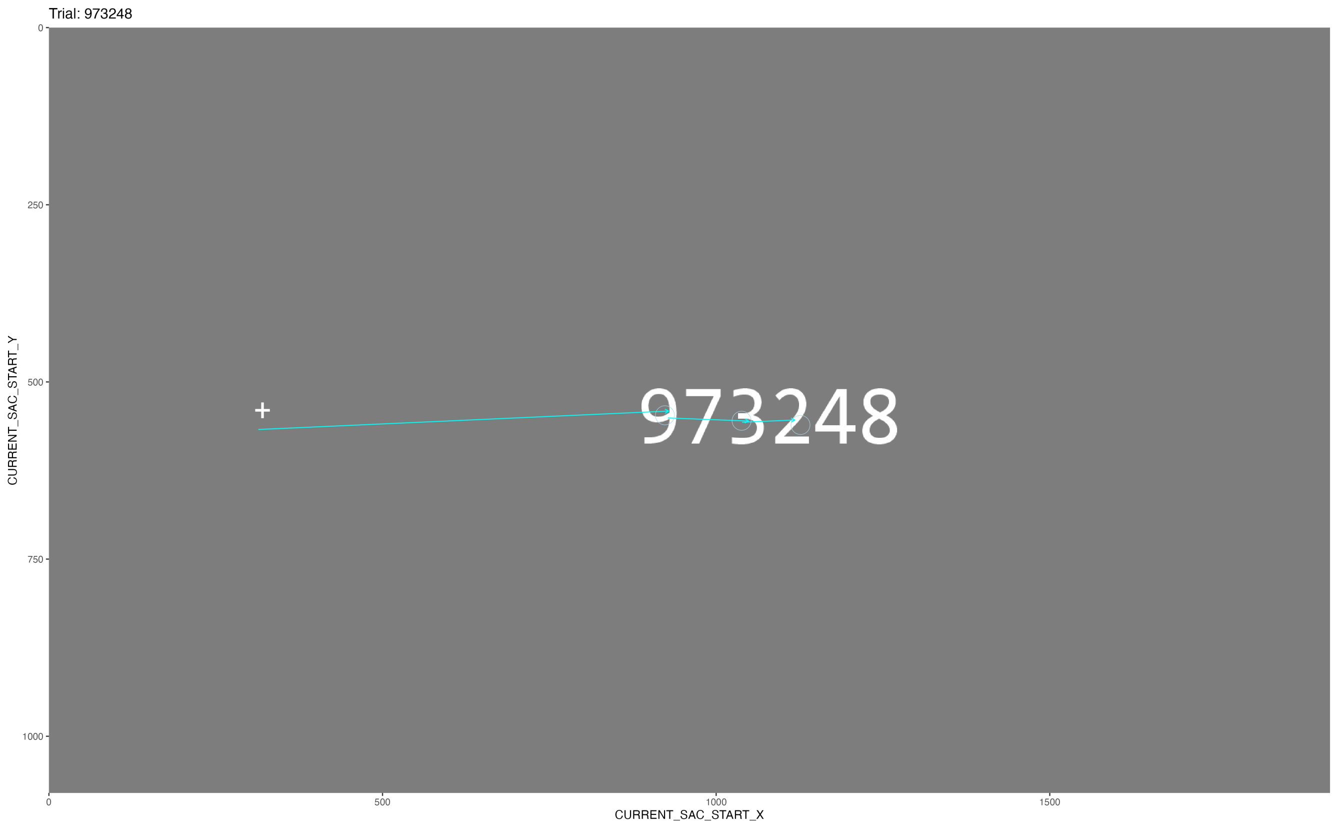
The following scripts: process the eye-movement files -> pair each trial index with a background image (stimulus) -> and displays saccades and fixations over the stimulus.

```
#code snippet from the single trial visualiser
    p <- img_gg +
      geom_segment(aes(x = CURRENT_SAC_START_X, y = CURRENT_SAC_START_Y,
                       xend = CURRENT_SAC_END_X, yend = CURRENT_SAC_END_Y),
                   data = df_trial,
                   arrow = arrow(length = unit(arrow_tip_size, "cm")),
                   color = arrow_color,
                   linewidth = arrow_width) +
      geom_point(aes(x = NEXT_FIX_X, y = NEXT_FIX_Y),
                 data = df_trial,
                 color = fixation_color,
                 fill = fixation_fill,
                 shape = 21,
                 stroke = fixation_line_width,
                 size = fixation_diameter) +
'''
Why manipulate point diameter?
In the eyelink algorithm precise (x,y) values are given for fixations.
However, fixations are highly-localized groups of micro-saccades,
to reflect this in the visualization fixations are drawn as large hollow circles.
'''
      scale_x_continuous(limits = c(0, 1920), expand = c(0, 0)) +
      scale_y_reverse(limits = c(1080, 0), expand = c(0, 0)) +
'''
In eye-tracking the y axis is usually flipped, such that 0 is on top.
'''
      ggtitle(paste("Trial:", image_num))
```


Trial: 458
0
250
500
750
1000
CURRENT_SAC_START_Y
458
0
500
1000
1500
CURRENT_SAC_START_X


Trial: 42836
0
250
500
750
1000
CURRENT_SAC_START_Y
42836
0
500
1000
1500
CURRENT_SAC_START_X





As expected, saccades seem to progress smoothly through the digits from left to right, occasionally revisiting previously fixated digits, a process known as a "Regression".

Because of the first visualizations I wanted to see how fixations differ across index (dive of appearance within a trial). The next visualizations are aggregate fixation maps.

To classify trials by fixation index and condition, I had to refine the dataset (this process is detailed below). With the refined datasets, I developed a function to segment the fixation data by digit length (3 to 5 digits) and fixation index (1 to 6), producing heatmaps for each subset across trials.

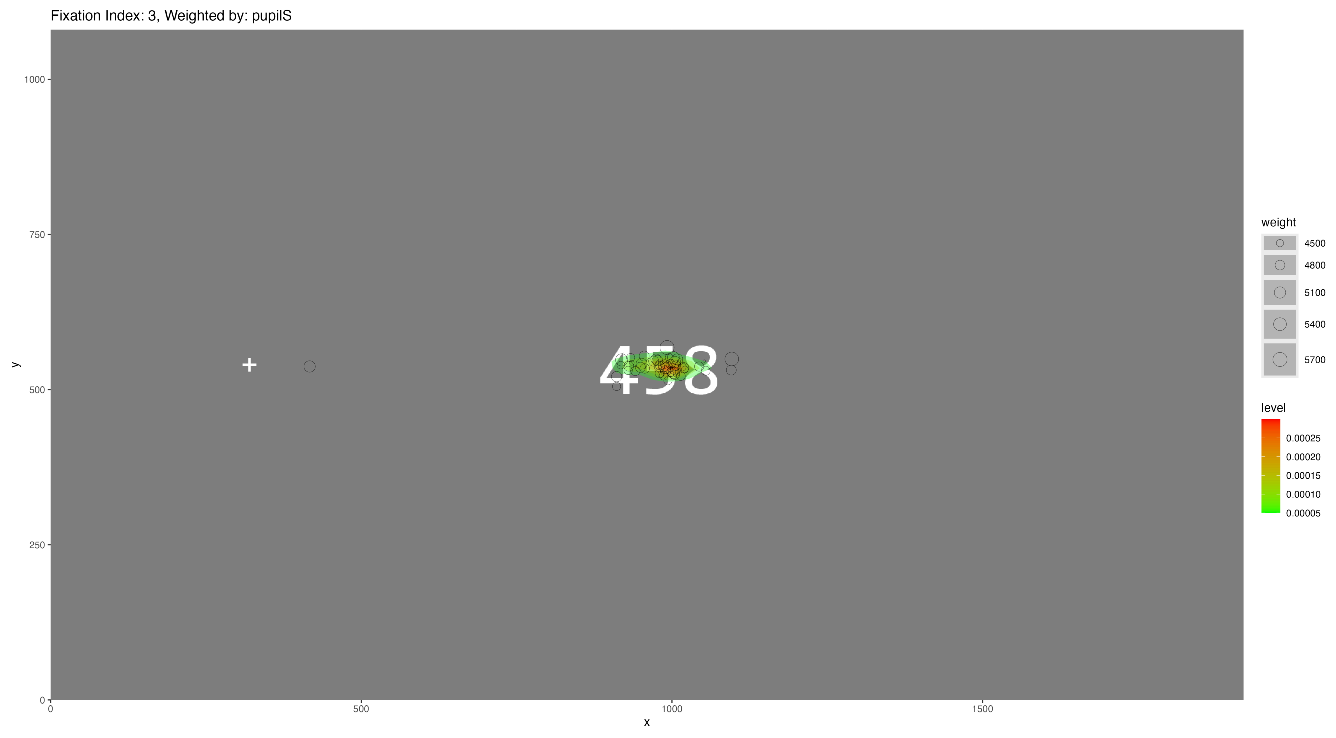
The visualization pipeline:

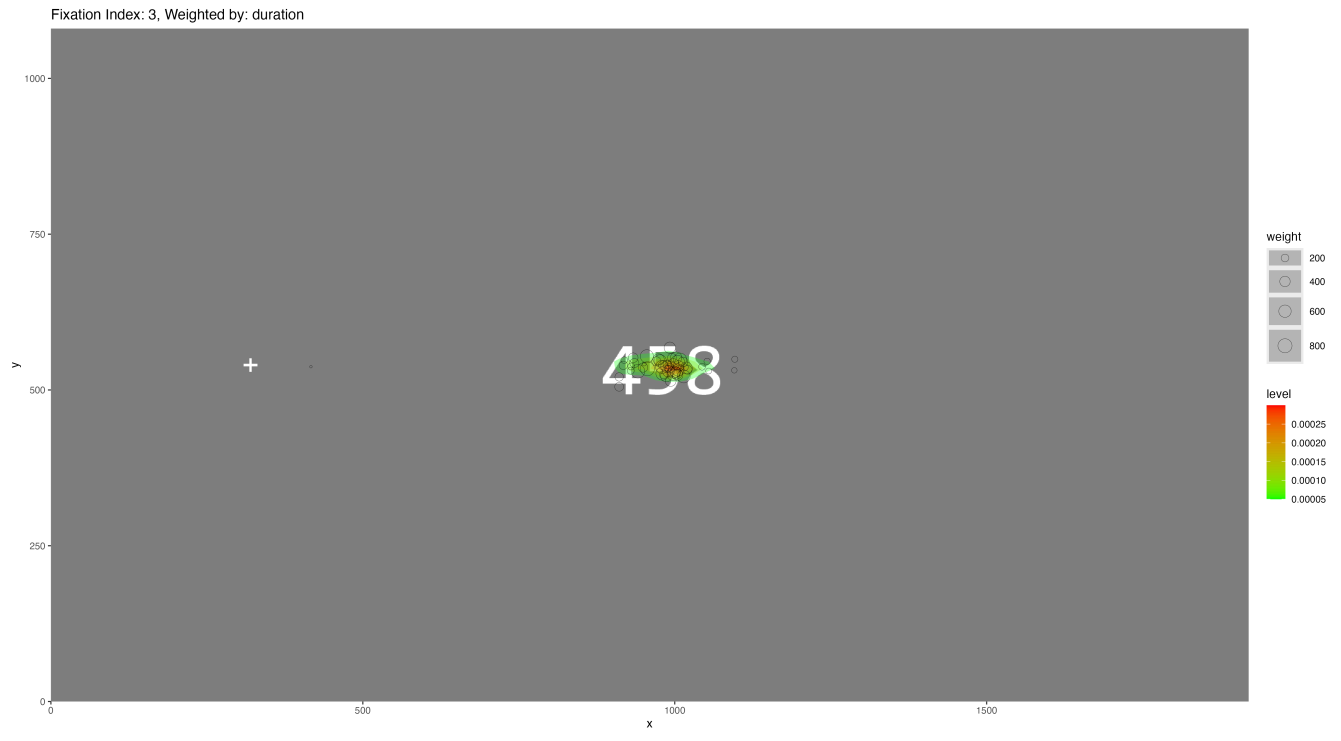
1. Grouping the dataset by condition and fixation index.
2. Loading the appropriate background image corresponding to each digit-length condition.

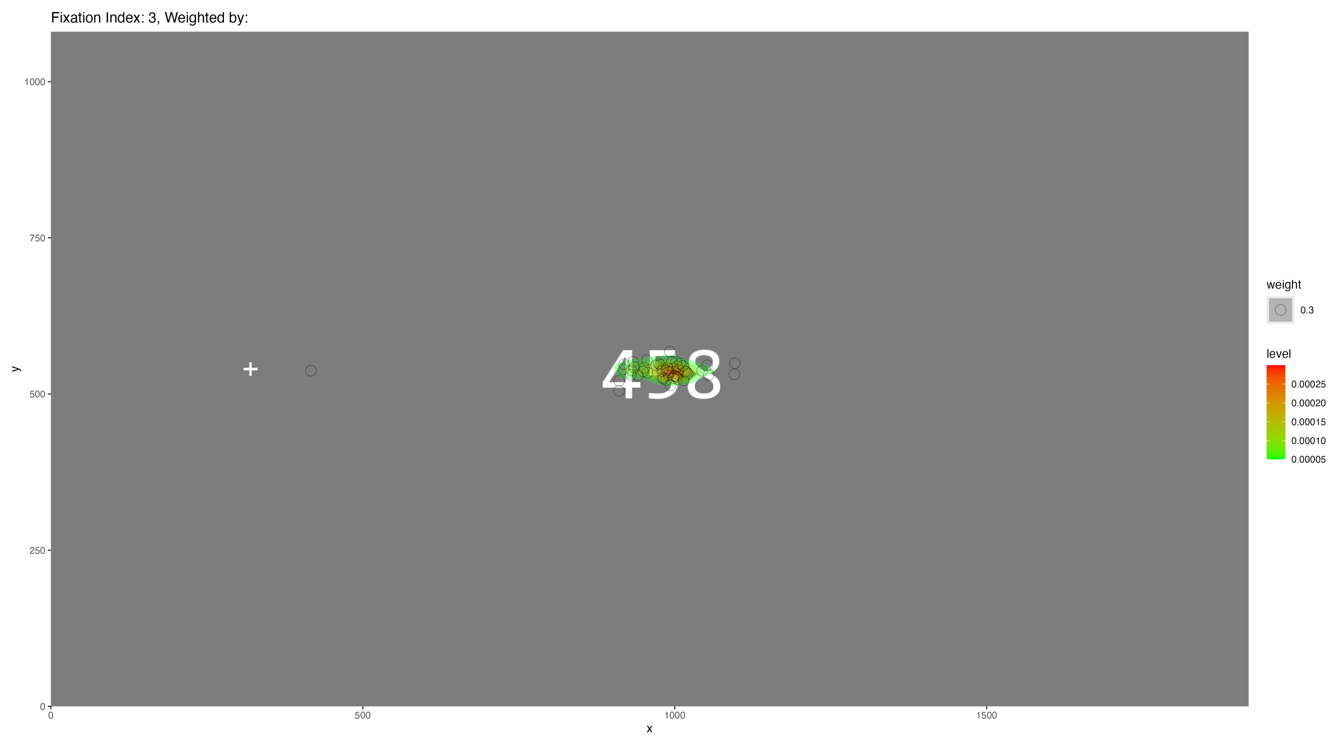
3. Overlaying individual fixations on the image and adding a density map.

The last step was incorporating an additional dimension that could represent attention. For the examples below I chose fixation duration and average pupil size. First I tried weighting the density map according to these additional dimensions, but at the end chose to represent them with fixation circle size. Importantly, circle sizes were scaled relative to one another (by the standard deviation of the subset).

The next 3 plots are density maps of the third fixation on 3-digit numbers, with fixation diameter weighted by pupil size, duration of fixation, and no weights (in that order).



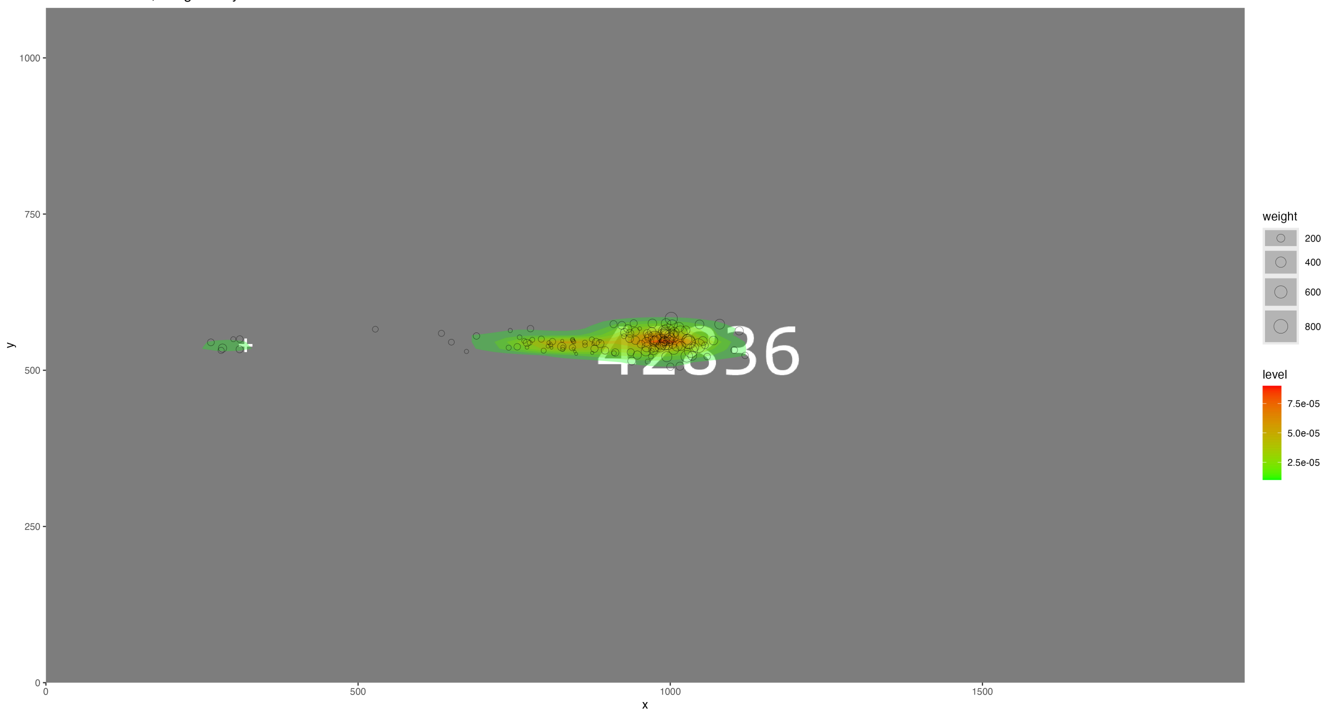




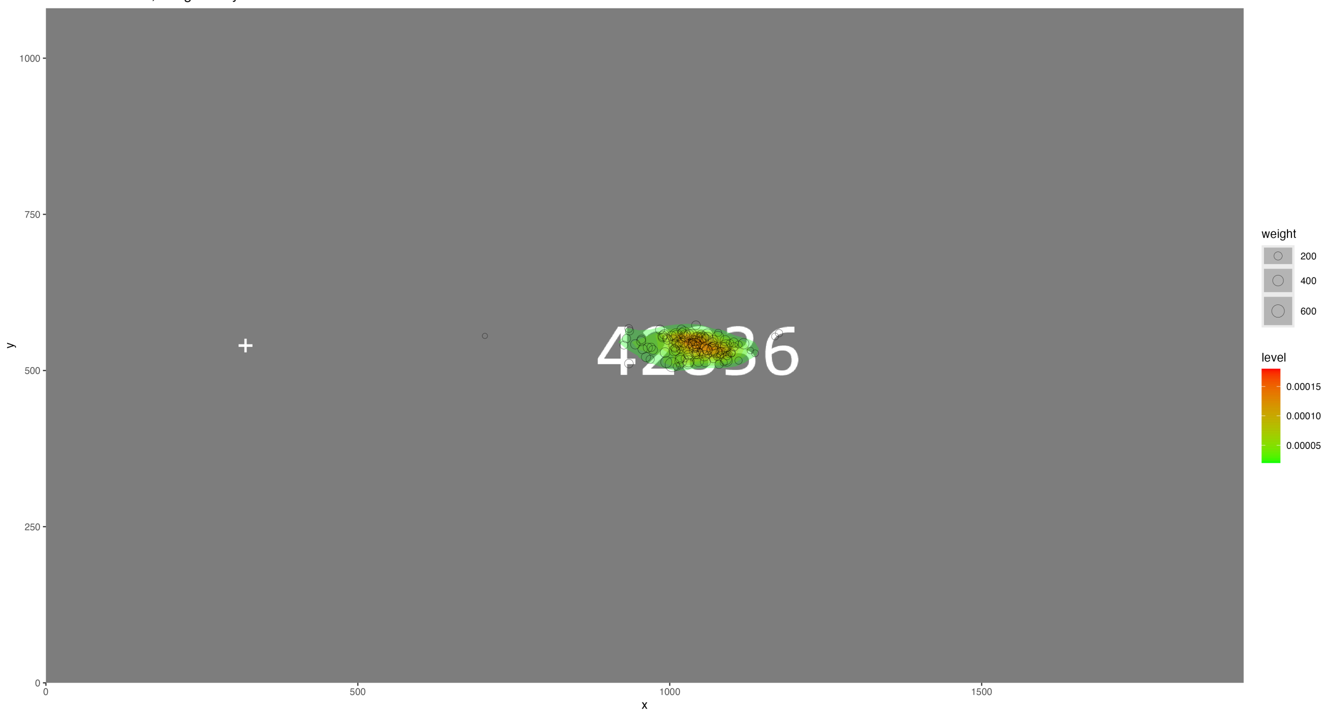
```
#Code snippet of a function I wrote that visuallizes trials with an interchangeable factor

visualize_fixation_heatmap <- function(df, image_list, weight_variable_name)
# df is the fixations.rdata with a few tranformations
# image_list are the example images for each length condition
# weight_variable_name is the 4th dimension (interchangable attention variable)

      heatmap_plot <- ggplot(df_fixation, aes(x = x, y = y)) +

        annotation_raster(img, xmin = 0, xmax = 1920, ymin = 0, ymax = 1080) + # background image
        stat_density2d(geom = "polygon", aes(fill = ..level..), alpha = 0.3,) +
        scale_fill_gradient(low = "green", high = "red") +
        scale_alpha(range = c(0.1, 0.9)) +

        geom_point(aes(size = weight), shape = 21, color = "black", fill = NA, stroke = 0.15)+ #4th dimension: interchangeabl

        scale_x_continuous(limits = c(0, 1920), expand = c(0, 0)) +
        scale_y_continuous(limits = c(0, 1080), expand = c(0, 0)) + coord_fixed() +
          # y coords are flipped before df is sent into this function because the image itself loads to
          #the "normal" orientation extending up and to the right of 0,0

        ggtitle(paste0("Fixation Index: ", unique(df_fixation$index_in_trial)[1], ", Weighted by: ", weight_variable_name)) #
                                                                                  # legend formatting with density ma
```

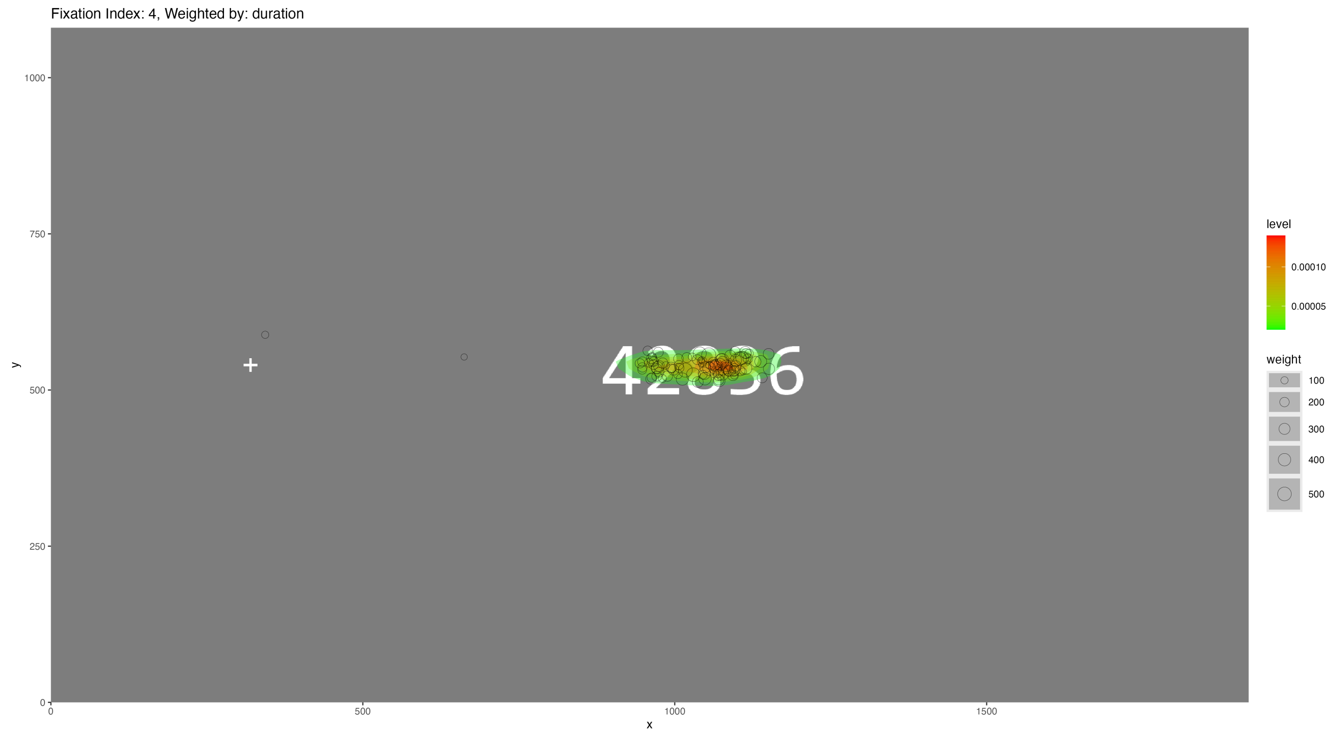
The next 4 plots are density maps of fixations in the 5-digit condition, by fixation index (2,3,4,5) in order. The circles are weighted by duration, as it seemed to me most informative.

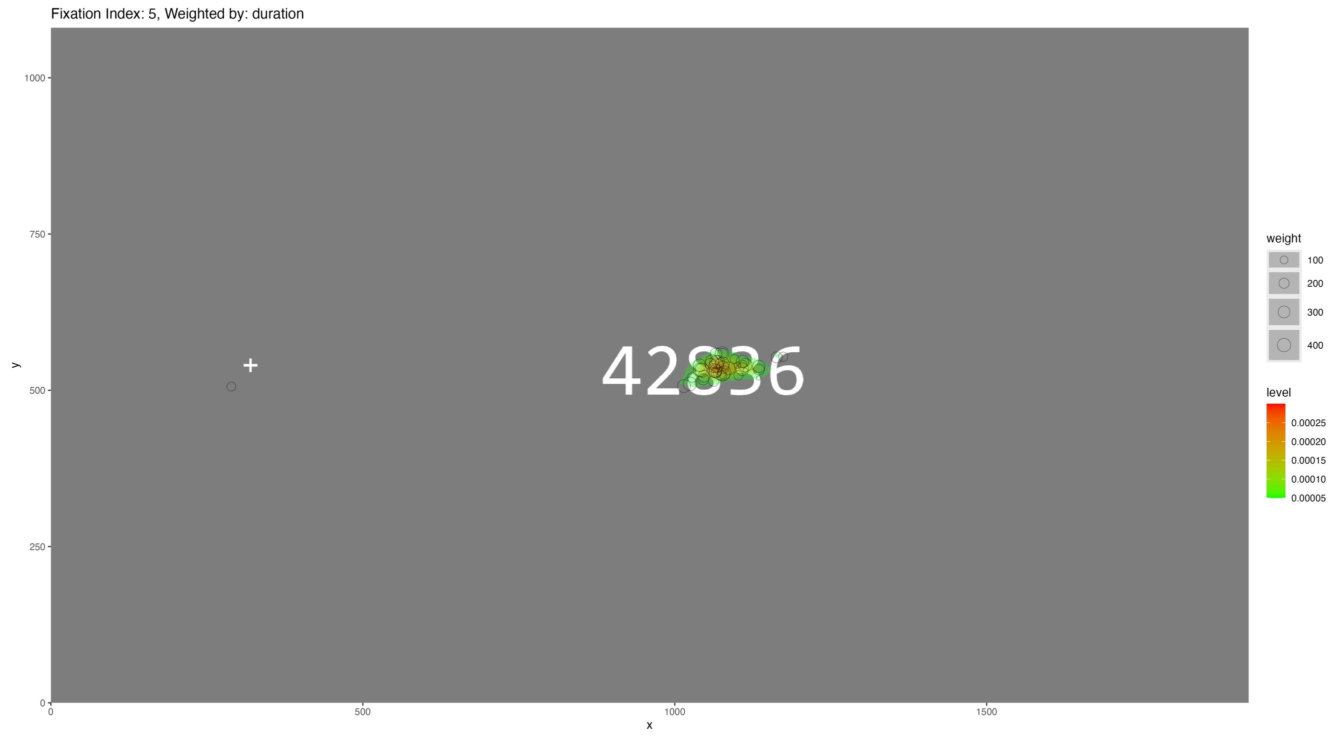
Fixation Index: 2, Weighted by: duration



Fixation Index: 3, Weighted by: duration







Refining the Eye Tracking Data

Scripts:

single_trial_visualizer

get_saccades

get_fixations

get_messages

indexed_saccades

Excerpt using piping:

```
saccades <- saccades |>
  filter(NAME != "UNDEFINED") |>
  mutate(NAME = gsub("\\.0$", "", NAME)) |>
  rowwise() |>
  mutate(nNum = trials$nNum[which(trials$index == NAME)[1]],
         IMAGE_PATH = ifelse(!is.na(nNum),
                             paste0("images/", nNum, ".png"),
                             paste0("images/", trials$attCheck[which(trials$index == NAME)[1]], ".png"))) |> ungroup()
```

```
Problems & Solutions:

> Too Much Data
•    Get rid of irrelevant data.
•    Rename the useful columns.

> Merging Multiple Files
•    Combine the trial messages dataset with the two eye-tracking datasets.


> Convert messages into features:

TRIALID:
    •    Standardize naming.
    •    Create a feature is_pattern, based on the trial ID (contains x or only numbers).
Trial Identification:
    •    Each trial begins with a TRIALID_{} message.
    •    The trial ends with a trial result message.
Stimulus Extraction:
    •    Extract the stimulus path to the photo.
    •    Store the stimulus as a feature.
Trial Result:
    •    Assign trial results as 0/1.
Early End Detection:
    •    Identify trials where the last fixation occurs before a predefined threshold (X ms).
```

Part 2

This data relates to a research project I am participating in this year, exploring abstract versus concrete representations in text. Previously, I worked on this dataset to extract vectorized representations of words from an LLM. The goal of this project is to use these embeddings to construct a model which could classify the concreteness of sentence embeddings without directly analyzing the text. This feature could be useful for the the larger research project, but is an interesting exploration on its own. More specifically, the the goal is to identify a direction in the embedding space that corresponds to concreteness levels, and use it in a simple classifier.

The First Dataset

This dataset provides human concreteness ratings for common English words, and sentence-transformer (LLM) embeddings of them.

The data files created and analyzed in this part of the peoject:

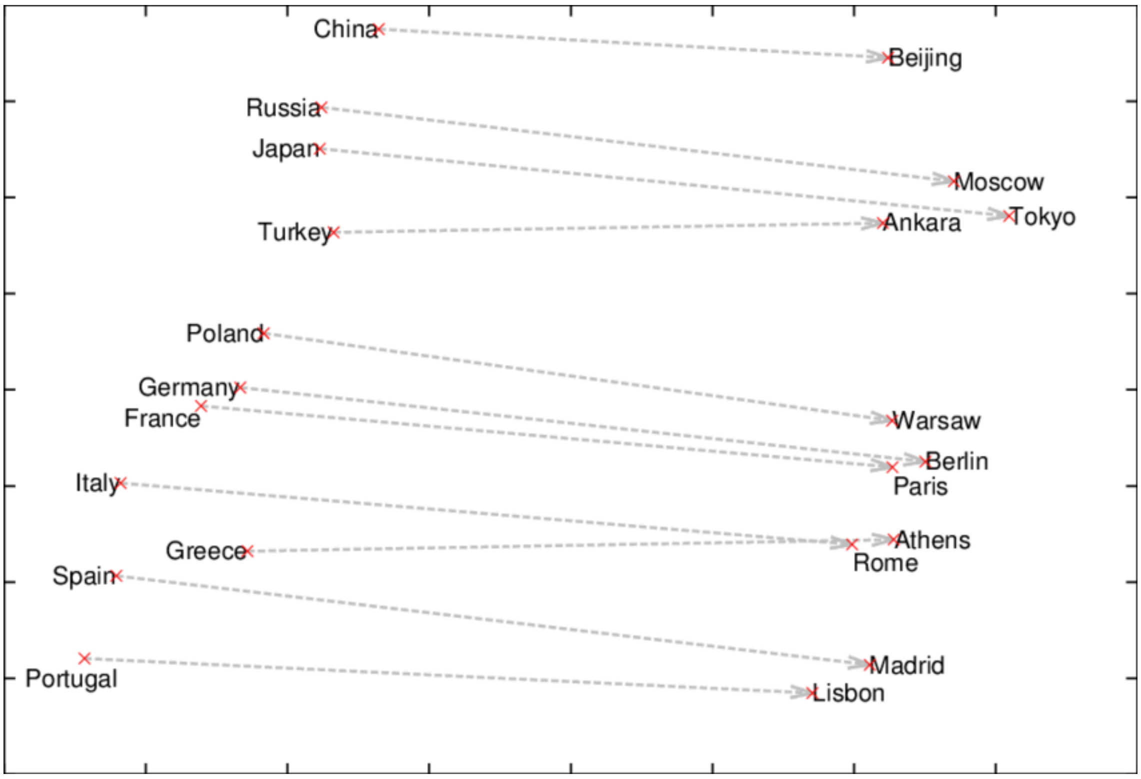
1. dictionary_df - contains behavioral data like word identities, concreteness ratings.
2. batch_{x} - these files are the embedding extractions for our behavioral data stimulus . Each batch has a name column for the word identity, and columns 1 through 2048 for the embedding features.
3. embeddings - these files are the test set, stimulus which we are using in the larger research project, which are goal is to classify.

Visualizing the Data:

Although visualizations were not initially planned for this dataset, representing low-level semantic relationships would be a useful proof of concept for my more complex hypothesis.

I wanted to see evidence of semantic clustering within the data, with categories known to exhibit geometric relationships in between embeddings such as Word2Vec, and Skip-gram.

Below is a graph of PCA projections of multiple countries and their capital cities from the Skip-gram paper linked below, this demonstates the goal of my following visualisation:



Graph from:

Distributed Representations of Words and Phrases and their Compositionality

https://arxiv.org/abs/1310.4546

My visualization was achieved similarly by projecting word embeddings onto the first two principal components derived from the embedding matrix.

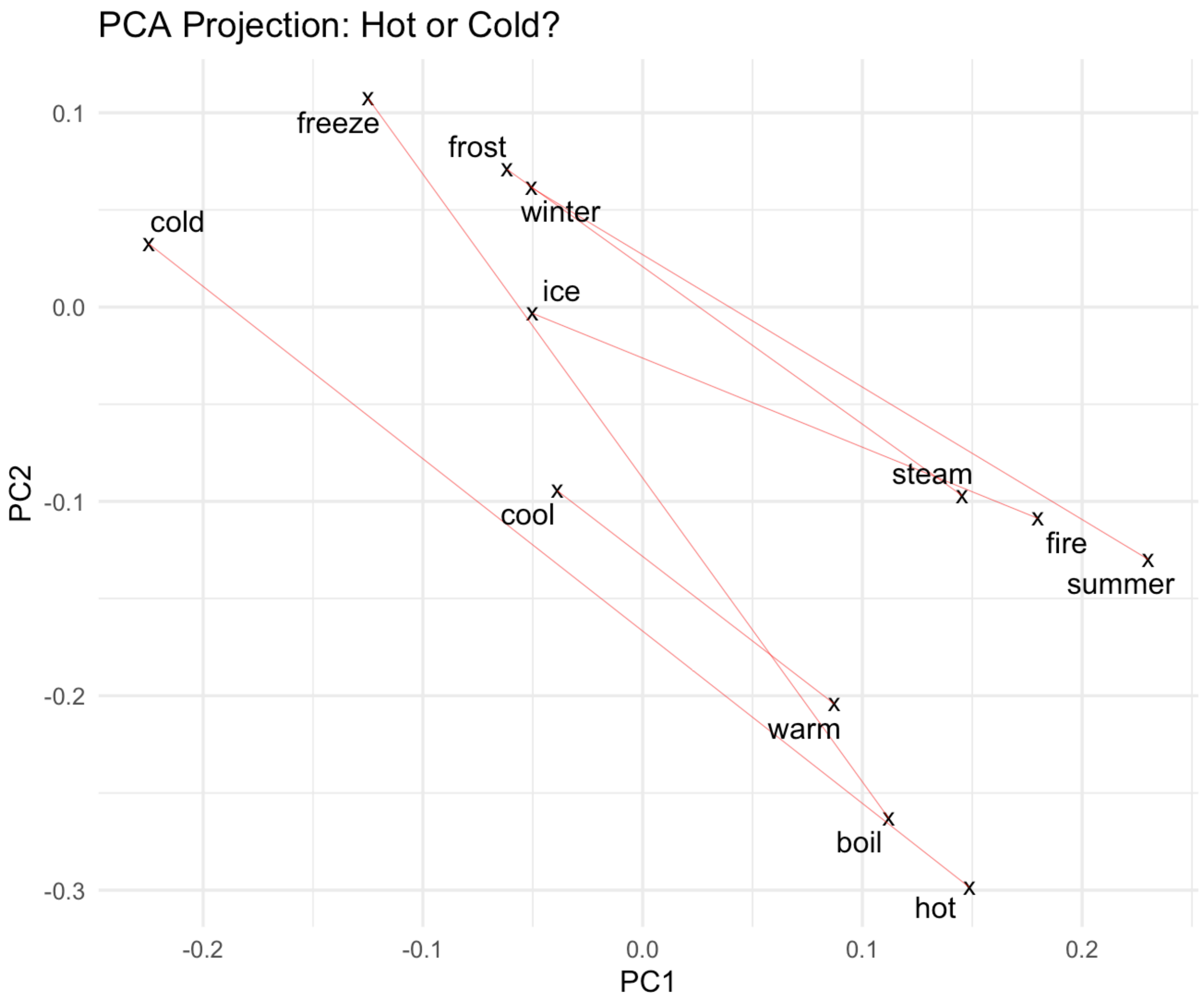
The script's visualization pipeline includes the following steps:

1. Extract the entire dictionary of word vectors into a single embedding matrix.
2. Perform Principal Component Analysis (PCA) and save the first two components (PC1 and PC2).
3. Select a meaningful subset of words to visualise.
4. Retrieve embeddings for these selected words.
5. Project these embeddings onto PC1 and PC2.
6. Plot the projected points onto the PC1 + PC2 plane.
7. Examine the resulting plot for scale-like geometry / clustering.

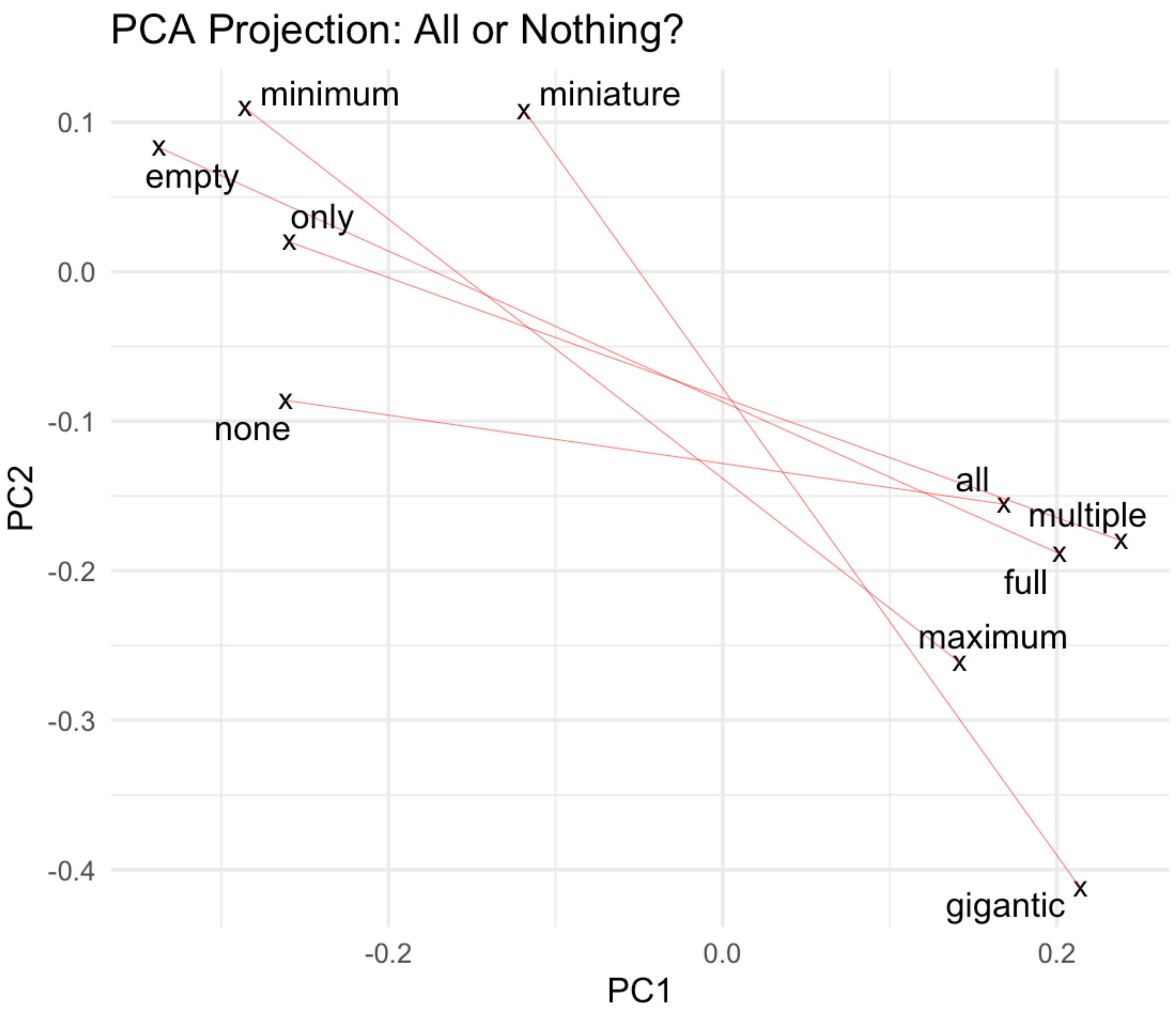
Excerpt using piping to map embeddings on to PC1/PC2 plane using a word list as input:

```
df_word_list <- tibble(name = word_list) |>
  left_join(dictionary_df, by = "name") |>
  mutate(embed_vec = map(embeddings, ~ if (is.null(.x)) NA_real_ else as.numeric(.x))) |>
  mutate(
    PC1 = map_dbl(embed_vec, ~ if (all(is.na(.x))) NA_real_ else sum(.x * pca_result$rotation[,1])),
    PC2 = map_dbl(embed_vec, ~ if (all(is.na(.x))) NA_real_ else sum(.x * pca_result$rotation[,2]))
  )|>
  mutate(idx = row_number()) |>
  group_by(grp = ceiling(idx / 2)) |>
  filter(!any(is.na(PC1) | is.na(PC2))) |>
  ungroup()
```

The first example shows scale like geometry, my data has few nouns (like place names) so I chose semantic categories:



This example shows nice clustering for size/quantity:



These projections seem to indicate that geometries do exist in the model's embedding space.

To make them I used a library with it and learn the course: library(ggrepel)

What will we test?

LLM embedding spaces have been shown to have interpretable geometries, such that human categories are present in the embedding space as relations between features, or sets of features.I will use a comprehensive database of human concreteness ratings of dictionary words to derive a vector that represents concreteness in a sentence-transformer's embedding space.

I will test the validity of this vector by fitting a linear regression which will predict mean concretness rating of a word by the cosine of the word's ebedding with the vector. Then i will test whether the cosine concretness scale (the cosine of different embeddings with the vector) is able to classify descriptions of images as concrete or abstract by their embeddings.

If the vector successfully generalizes the concreteness feature from individual words to entire sentences, it would suggest potential generalization to any text's embedding.

Theoretical Research Question

Do internal representations of text in a sentence tranformer contain a dimension encoding concreteness?

Operational Research Question

A) **Is there a vector in the embedding space that aligns with human concreteness ratings (of words)?**

If there is:

B) **Does the vector of single word concreteness generalize to concreteness of full sentences?**

Tests

OLS linear regression on the single word data

The model will test whether the embeddings' cosine similarity with the concreteness vector significantly predicts mean concreteness rating.

H0: b1 = 0

H1: b1≠0

Hypothesis test: F

*Explained variance really matter here, since we have a lot of data.

Logistic regression on real descriptions The model will test whether the embedding's cosine similarity with the concreteness vector is informative to classify descriptions as visual.

H0: b1 = 0

H1: b1≠0

Hypothesis test: X2

Extract Features

Compute the concreteness vector

```
train_df <- dictionary_df

train_df$Conc <- scale(train_df$Conc.M, center = TRUE, scale = TRUE) #normalize

train_df <- train_df |>
  mutate(embeddings= map(embeddings, ~ .x / sqrt(sum(.x^2)))) #normalize

compute_weighted_avg_embedding <- function(train_df) {  # there is a better way to this, but this is good enough for now :)

  embedding_matrix <- do.call(rbind, train_df$embeddings)

  scaled_conc <- scale(train_df$Conc.M, center = TRUE, scale = FALSE)[,1]

  #weighted_embeddings <- sweep(embedding_matrix,1 , scaled_conc,"*")
  weighted_embeddings <- embedding_matrix * scaled_conc

  avg_embedding <- colMeans(weighted_embeddings, na.rm = TRUE)
  avg_embedding <- avg_embedding / sqrt(sum(avg_embedding^2))
  return(avg_embedding)
}

weighted_avg_embedding <- compute_weighted_avg_embedding(train_df) #compute concreteness vector

train_df <- train_df |>
  mutate(
    dot_product = map_dbl(embeddings, ~ sum(.x %*% weighted_avg_embedding)))


min_dot_product <- min(train_df$dot_product, na.rm = TRUE) # should be minimum -1
max_dot_product <- max(train_df$dot_product, na.rm = TRUE) # should be maximum 1
```

The dot product is the cosine of the angle between the two vectors, this is leveraged to approx. similarity:

$$S_C(A, B) := \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

Because our vectors are normalized, the norms are always 1

V = concreteness vector

X = an embedding

$$Conc(X) = S_C(X, V) = \mathbf{X} \cdot \mathbf{V}$$

And thats why I double check the outputs:

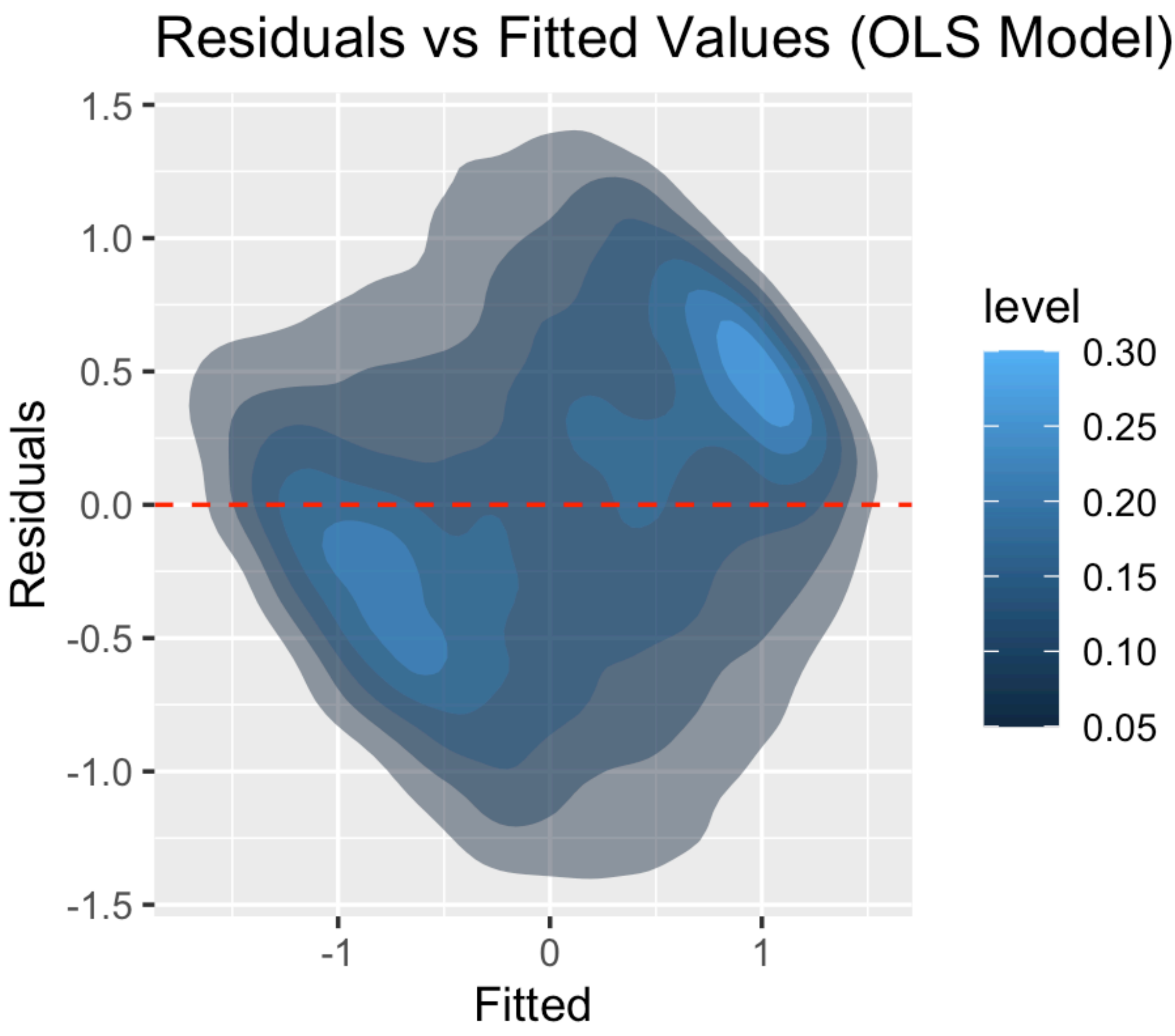
```
>print(min_dot_product)
[1] -0.5913344
> print(max_dot_product)
[1] 0.3966699
```

Linear Regression on Single-Word Data

We predict the response variable - the mean concreteness rating of Word_i by the cosine of its embedding wiht the concretness vector.

$$\text{Rating}_i = b_0 + b_1 * Conc(X_i) = +\epsilon_i$$

```
lm_model <- lm(Conc ~ dot_product, data = train_df)

ggplot(data.frame(Fitted = lm_model$fitted.values, Residuals = residuals(lm_model)),
       aes(x = Fitted, y = Residuals)) +
  stat_density_2d(aes(fill = ..level..), geom = "polygon", alpha = 0.5) +   # there are too many dots!
  geom_hline(yintercept = 0, linetype = "dashed", color = "red") +
  labs(title = "Residuals vs Fitted Values (OLS Model)")
```



The above graph plots the X values against the residuals, the color encodes density.

Although the hypothesis test is significant, the lack of fit suggests there is something wrong in the data. Because there are so many observations it is not due to outliers. Therefore, I teseted for heteroscedasticity, which is also evident in the graph:

```
ncvTest(lm_model)
```

```
 ** Non-constant Variance Score Test **

 Chisquare = 23.76713, Df = 1, p = 1.0872e-06
```

The test shows that the fit is significantly heteroscedastic.

I reanalyzed the data using weighted least squares:

```
weights <- 1 / lm(abs(lm_model$residuals) ~ lm_model$fitted.values)$fitted.values^2
wls_model <- lm(Conc ~ dot_product, data = train_df, weights = weights)

ncvTest(wls_model)

summ(wls_model)
```

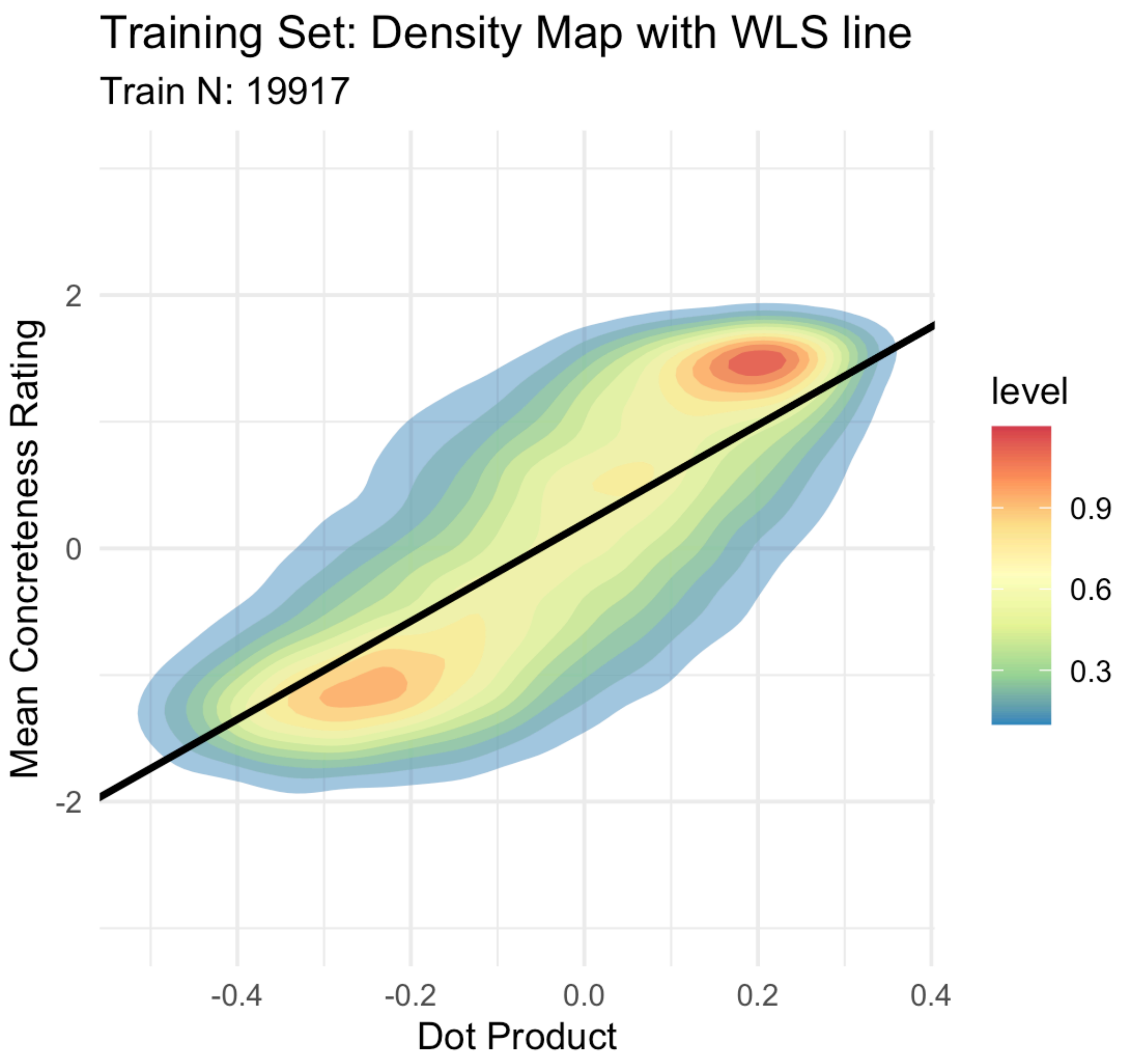
Output

```
Non-constant Variance Score Test
Variance formula: ~ fitted.values
Chisquare = 0.7356417, Df = 1, p = 0.39106

MODEL INFO:
Observations: 19917
Dependent Variable: Conc
Type: OLS linear regression

MODEL FIT:
F(1,19915) = 26301.12, p = 0.00
R² = 0.57
Adj. R² = 0.57

Standard errors:OLS
------------------------------------------------
                    Est.   S.E.   t val.      p
----------------- ------ ------ -------- ------
(Intercept)         0.20   0.00    40.62   0.00
dot_product         3.87   0.02   162.18   0.00
------------------------------------------------
```

Our first model used OLS to assses the linear relationship between the mean concreteness rating (Conc) and the cosine scale value (dot_ product). But the graph of the residuals showed a clear lack of fit, and a an increase in variance around x = 0.

Because my model did not meet the homoscedasticity assumption, I computed weights and analyzed the data using a weighted least squares model. The weighted model corrected for the heteroscedasticity. A possible cause for the inconsistent variance, is that the concreteness ratings of words that are not decisively concrete or abstract are less reliable, or not captured in the sentence transformer embedding space (or my model).

A note before the results: I considered splitting the data into a testing set and a training set, thinking it would improve the test's validity. However, after reviewing the results on the full dataset, I saw no differnce from the split analysis, specifically no indication of overfitting.



Results

A weighted least squares linear regression model F(1,19915) = 26301.12, p < 0.000) showed a correlation between the response variable and the cosine scale. The result indicates that the concreteness vector, to some degree, coincidence with participant ratings of word concreteness.

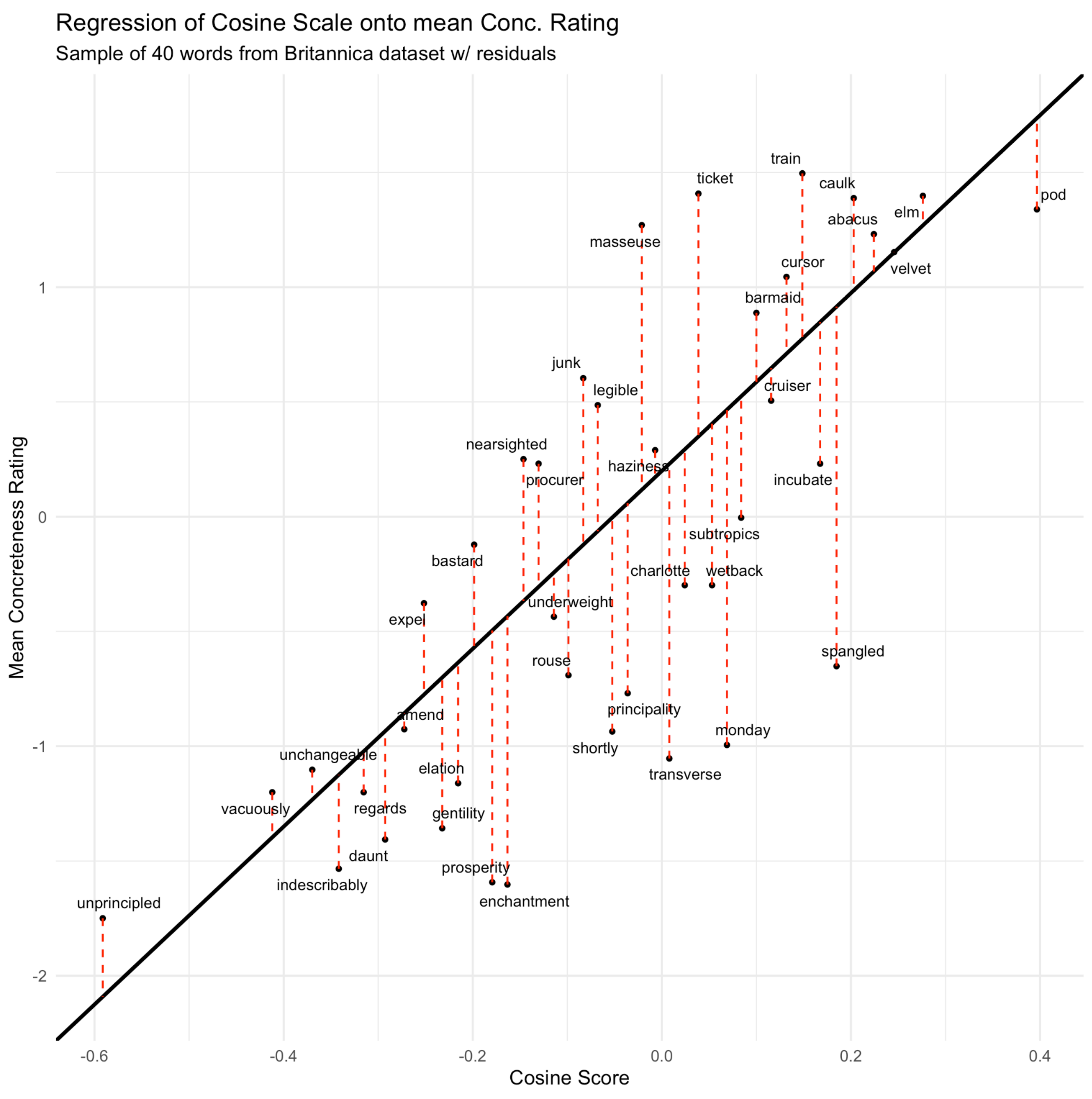
The intercept of 0.20 shows that the response variable is slightly positive, mean concreteness > 3.

The positive B1 value means that the cosine scale has a positive correlation with participants' conc. rating.

The R² value indicates that most (57%) of the variance in pariticipants' ratings is explained by this scale.I think this is a great result, this means that a more flexible model could explain even more of the relationship between the ratings dataset and the embedding space. A future model could consider many more factors like POS, rating SD, and word frequency A smaller angle between the word's embedding and the conc. vector -> The word's embedding is more like the Conc. Vec -> predicted to be a higher-concretness word.

In the sentence-transformer's embedding space - an embedding that is pointing in the opposite direction of the concreteness vector, is a maximally abstract word. An embedding that is orthogonal to both inhabits the midway between complete abstraction and complete concreteness. although I don't have the capability to search in reverse for these kinds of words, at the moment, I can look inside the data set and extract the maximally abstract and concrete words within it:

```
|name            | Conc.M| dot_product|
|:---------------|------:|-----------:|
|pod             |   4.63|      0.3967|
|colander        |   4.21|      0.3838|
|deck            |   4.77|      0.3819|
|spider          |   4.97|      0.3758|
|crustacean      |   4.66|      0.3735|
|silo            |   4.30|      0.3716|
|firepower       |   3.67|      0.0000|
|raggedy         |   2.72|      0.0000|
|wednesday       |   3.14|      0.0000|
|disembark       |   2.57|     -0.0001|
|birthday        |   3.20|     -0.0001|
|reapply         |   2.90|     -0.0001|
|semicolon       |   4.62|      0.0001|
|unprincipled    |   1.48|     -0.5913|
|unenlightened   |   1.85|     -0.5696|
|irrationally    |   2.12|     -0.5442|
|illogically     |   1.46|     -0.5423|
|demean          |   2.00|     -0.5395|
```



The Second Dataset

This dataset contains descriptions in two conditions:

Abstract: A long vegetable with dark green skin that is light green inside, usually eaten raw.

Visual: A fresh green cucumber with a smooth, slightly shiny surface and a small stem attached at one end.

The dataset contains extracted embeddings for all sentences, indexed by the stimulus name and condition.

Extract Features

Using tidyverse I parsed the input files (which I had planned to process in python):

Remove from all numbers the pytorch tensor wrappers

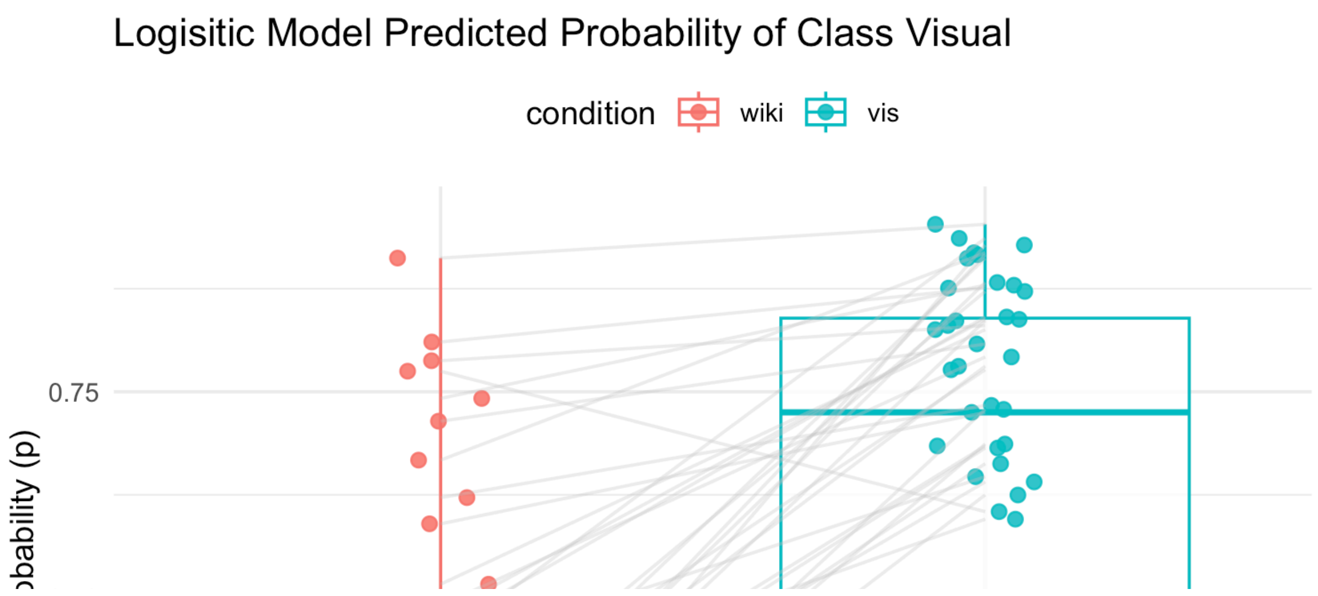
Normalize the embeddings

```
load("output/weighted_avg_embedding.RData")
#load data
folder_path <- "data/embeddings"
file_list <- list.files(path = folder_path, pattern = "\\.csv$", full.names = TRUE)

#cleanup func for the test data which was saved incorrectly
process_file_tidyverse <- function(file) {
  df <- read_csv(file, col_types = cols(.default = "c")) |>
    select(condition, name, '1':'2048') |>
    mutate(across(everything(), ~ str_replace_all(.x, "tensor\\(|, device='cuda:0'\\)", ""))) |>  # remove wrap
    mutate(across(-c(condition, name), as.numeric))
  return(df)
}

embeddings_df <- bind_rows(lapply(file_list, process_file_tidyverse))

embeddings_df <- embeddings_df |>
  mutate(embeddings = pmap(select(embeddings_df, `1`:`2048`), c))|>
  select(condition, name, embeddings)

for (i in 1:nrow(embeddings_df)) {  #make sure theyre all unit vecs
  embedding <- embeddings_df$embeddings[[i]]
  norm <- sqrt(sum(embedding^2))
  embeddings_df$embeddings[[i]] <- embedding / norm
}

#get cosine
embeddings_df <- embeddings_df |>
  mutate(dot_product = map_dbl(embeddings, ~ sum(.x * weighted_avg_embedding)))

#normalize output
embeddings_df <- embeddings_df |>
  mutate(scaled_conc = scale(dot_product, center = TRUE, scale = TRUE)[, 1])

embeddings_df$condition <- factor(embeddings_df$condition, levels = c("wiki", "vis"))
```

Logistic Regression On Sentence Data

$$\text{logit}(p_i) = \log\left(\frac{p_i}{1-p_i}\right) = b_0 + b_1 \cdot Cos()_{\,i} + \epsilon_i$$

```
embeddings_df$condition <- factor(embeddings_df$condition, levels = c("wiki", "vis"))

logit_model <- glm(condition ~ dot_product, data = embeddings_df, family = binomial)

summ(logit_model)
```


Logisitic Model Predicted Probability of Class Visual
condition
wiki
vis
obability (p)