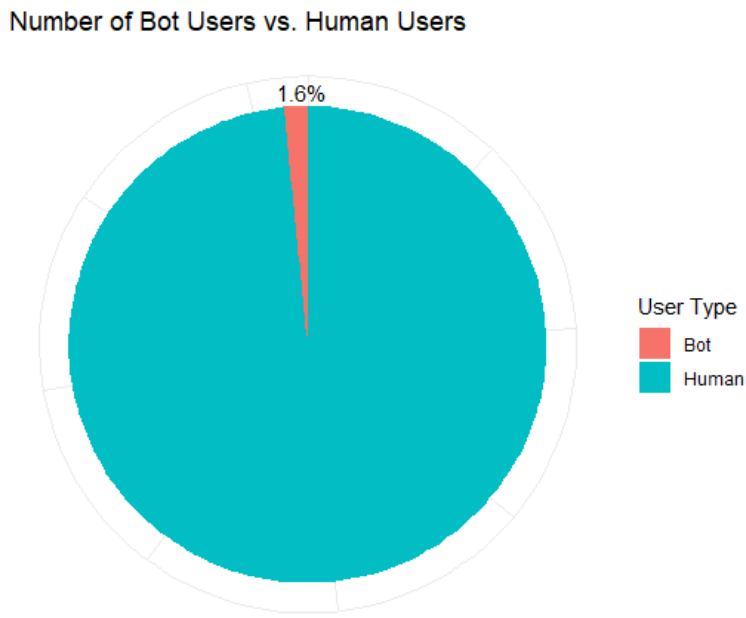
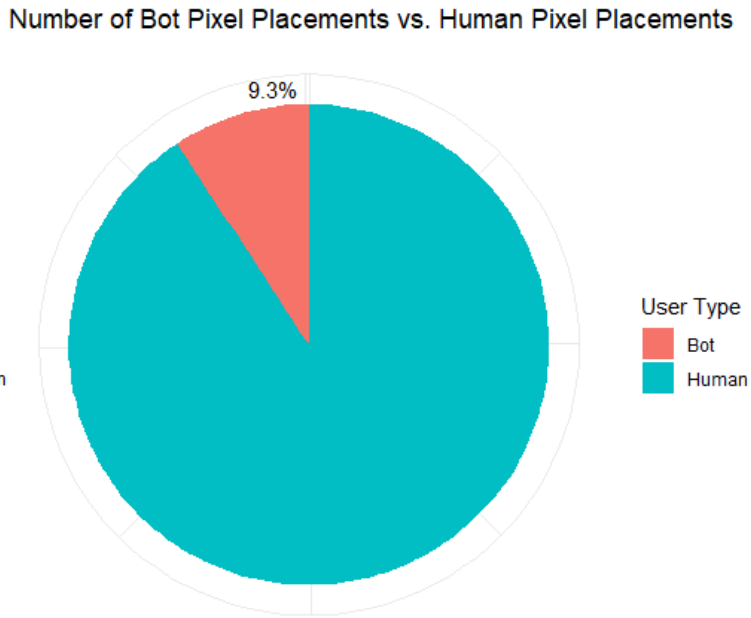
# The Effect of Bots in r/place

With communities clashing over pixel territories on r/place, the use of botting is a massive issue when it comes to pixel placements. Since a single user has to wait 5 minutes to place a pixel, many users opt to automate their contributions to the canvas to not only stay constantly active but to also leave a bigger presence of themselves and their community. The focus of this report is to explore how botting drives the different artistic creations in r/place.

Doing some research, many communities opted to present this specific youtube tutorial[1] for users to set up bots to automate their placements. The bot seems fairly straightforward, allowing users to upload images and multiple bot users would start mindlessly placing pixels to match a pixelated version of the uploaded image on the canvas. Upon further inspection of the bot structure, the pixel placement duration seems to be fixed, without any way for the user to add a variable duration rate between pixel placements. This was quite interesting, as I initially presumed that the best way for r/place moderators to detect bot behaviour was by precise pixel placement interval consistency. Using this knowledge, I decided that to detect which users were bots, I had gathered all the users who've placed more than 1 pixel, and tracked down which users have placed consecutive pixels down with the same duration in between. I was surprised to see that out of the 10,381,163 users, only 164,654 of them were bots (around 2%), which was honestly way less than expected. However, upon analysis on 160,353,104 placed pixels, those 164,654 bots actually had placed 14,854,384 pixels (around 10%).




This means that an average bot account has placed 89 more pixels than an average human account did.

[1] https://www.youtube.com/watch?v=LMeagfqTrSo

To make sure that my rules were picking up bots properly, I analyzed the distributions of pixel placement intervals. For bots:

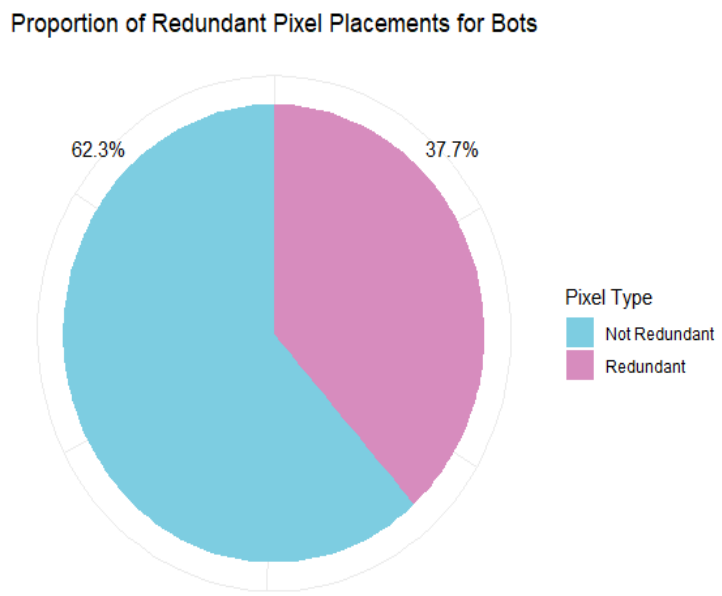
| 10th percentile | 25th percentile | 50th percentile | 75th percentile | 90th percentile |
|---|---|---|---|---|
| 00:05:03 | 00:05:07 | 00:05:24 | 00:07:58 | 00:22:54 |

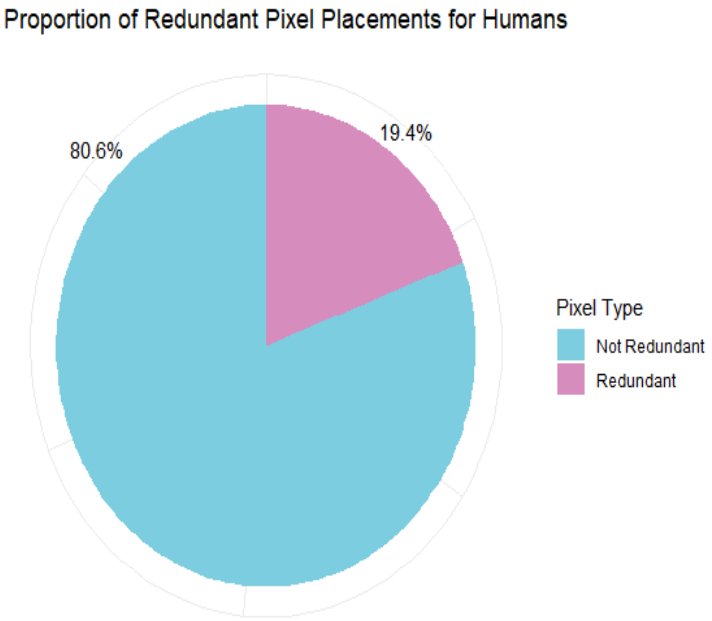
And for humans:

| 10th percentile | 25th percentile | 50th percentile | 75th percentile | 90th percentile |
|---|---|---|---|---|
| 00:05:09 | 00:05:26 | 00:07:58 | 00:31:19 | 3:27:57 |

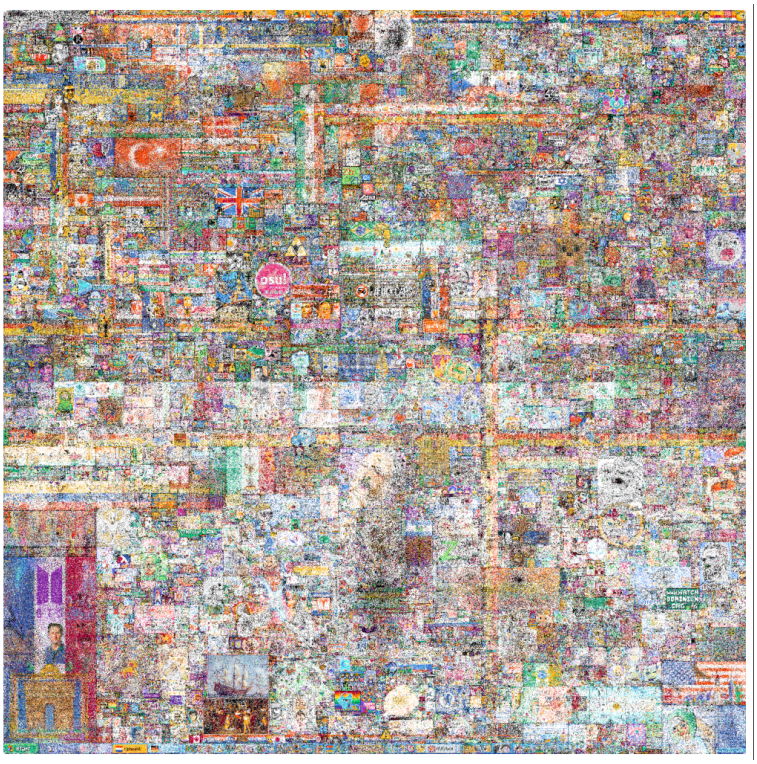
It seems like it worked pretty well considering how closely knit the bot's pixel placement interval distribution was compared to the humans.

Additionally, if the bot seemed to be naive in its approach in placing redundant pixels (placing a pixel that doesn't change the color, we could also assure that this was the case. For bots, out of 14,854,384 pixel placements, 5,701,301 of them were redundant (around 38%). For humans, out of 145,498,720 pixel placements, 28,160,050 of them were redundant (around 19%). Meaning that bots were associated with twice as many redundant pixel placements.

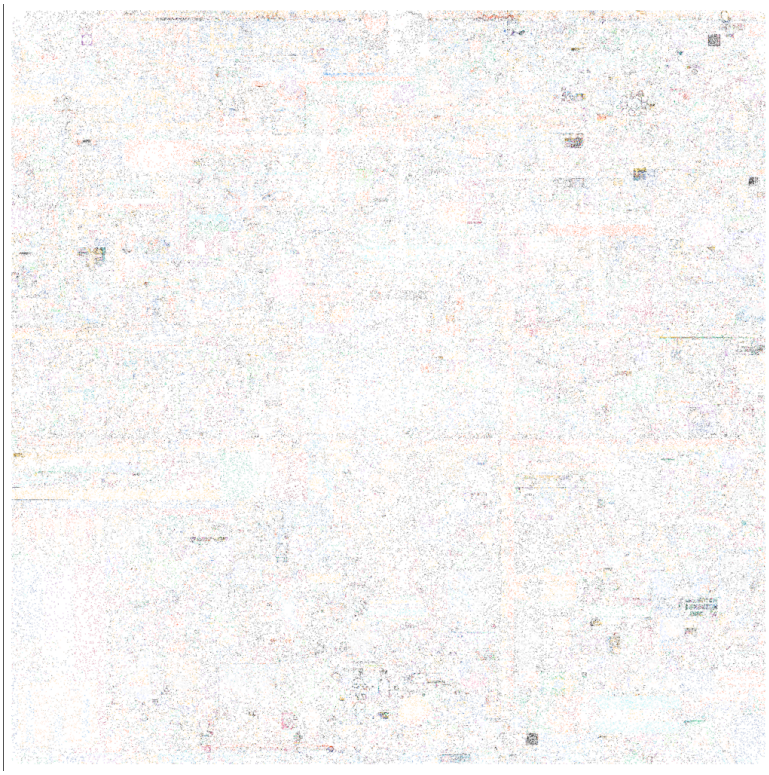



Finally, I also wanted to see what r/place would look like if we separated the bots from the humans. If we reran the r/place canvas but only used the bot behaviour, this is what the canvas would look like:



Meanwhile if we only analyzed the pixels humans placed, we would get this canvas:



Given, a lot of humans probably placed their pixels cohesively with bots to make an image, but it's surprising how sparse the canvas is.

# Spark vs. Polars Analysis

(I didn't include any visualizations in the spark task because that's not reliant on a specific data wrangling package)

Spark has insanely fast data manipulation. Queries that involved mutating columns, joining and creating window functions for partitioning were almost instantaneous when using spark. However, spark seemed to be a lot slower when it came to having to access individual values. Queries such as filtering and getting summary statistics seemed to be significantly slower. Polars is definitely optimized for those types of queries, due to its columnar processing. It also might be because Polars is optimized for multi-threading? I couldn't find anything on how distributed computing frameworks on a single machine get impacted.

I also didn't get the same values from spark vs. polars, I'm not too sure why.