[Github]

Initial Concept and Inspiration

My project is a sort of self-portrait using my personal data obtained from various online platforms, including Instagram, Spotify, Pinterest, Google Takeout, Hinge, Apple, Tumblr, Reddit, Discord, Soundcloud, TikTok, Twitter, Snapchat, and Netflix. My project is to visually represent a digital persona and explore the intricate relationship between data, identity, and self-expression. My overarching goal was to promote investigation through the creation of a dynamic and interactive data visualization system.

Inspirations

***Everyone I Have Ever Slept With* by Tracey Emin**

- An iconic example of self-portraiture through data

Perpetual Provisional Selves: A conversation about authenticity and social media by Rob Horning and Amalia Ulman

Link: Rhizome Article

- Social media as an endless interview, exploring challenges in conveying sincerity and the delicate balance between self-expression and self-presentation.
- Hermeneutic Violence:
  - Examining how social media posts are interpreted and scrutinized, even when intended to be normal or superficial.
- Desire for Authenticity:
  - The fine line between presenting oneself authentically and the societal desire for genuine expression.
- Surveillance Capitalism:
  - Reflecting on the impact of surveillance capitalism on the original ideals of digital connection, considering it as a means to commercial ends.

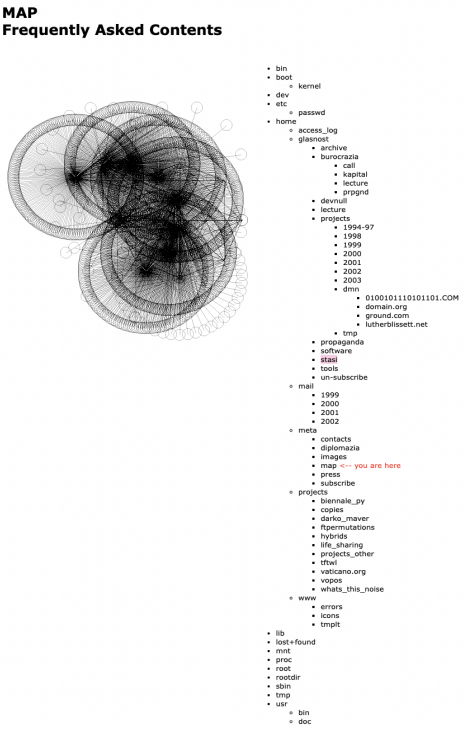
Data Portraits by Judith Donath

Link: MIT Press Article

Handmade data portraits emphasizing characteristic words in a conversation, exploring the potential automation of this process.

- "How we see people—ourselves as well as others—in the virtual world is perhaps the most challenging problem in the design of online spaces. Online, we have no inherent appearance. This is liberating, for we can rethink and recreate personal identity cues, or omit visual representation altogether. Yet a world in which it is difficult to perceive the inhabitants as distinct individuals can be dull and confusing… If we are to create more immersive and sensorial interfaces for social communication, one of the fundamental problems we must solve is how to represent the participants."

- “The goal of a visualization is often accuracy; it is a tool for scientific or sociological analysis. A portrait is an artistic production, shaped by the tension among the often-conflicting goals of the subject, artist, and audience. The subject wants to appear in the best possible light; the audience wants to gain insight about the subject; and the artist has his or her own aesthetic message to convey, as well as mediating between the subject and audience. The data portrait artist has an enormous number of choices to make in creating the portrait, beginning with deciding what data to show. Even given the decision to use, say, one’s history in a conversation, decisions remain about which patterns to show and how to depict them. This is not to deny that data portraits are visualizations; they are. The distinction is in the conceptual framing.”



Life sharing

Link: Net Art Anthology

- “…a radical gesture of self-surveillance. For three years, the couple made the contents of their home computer accessible to the public. All of the contents–including files, emails, bank statements, and so on–were available in real time to be read, copied, and downloaded.”
- “…a proto-typical meditation on living online. Made long before social media’s widespread influence, the work pointed towards the blurring of the public and private spheres that characterize our current moment. A performance, a provocation, and a site of exchange, the project was based on the idea that ‘file sharing=life sharing,’ suggesting that life had become inextricably enmeshed in digital culture and the network.”

Critical Interface Toolbox

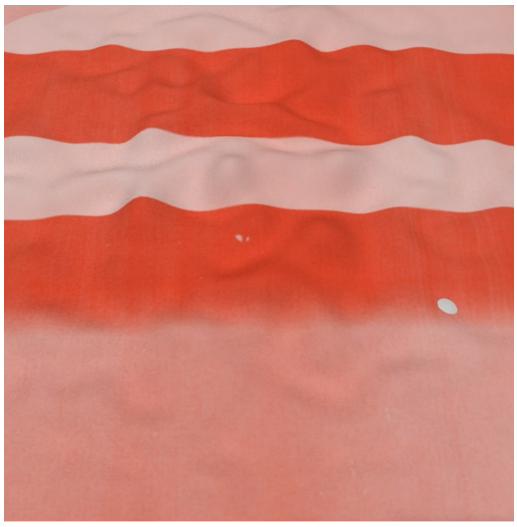
Link: Toolbox

- “…experimental methodologies, practices, and tools aimed at enhancing critical thought towards the actual configuration of the Interface.”

Critical Atlas of the Internet by Louise Drulhe

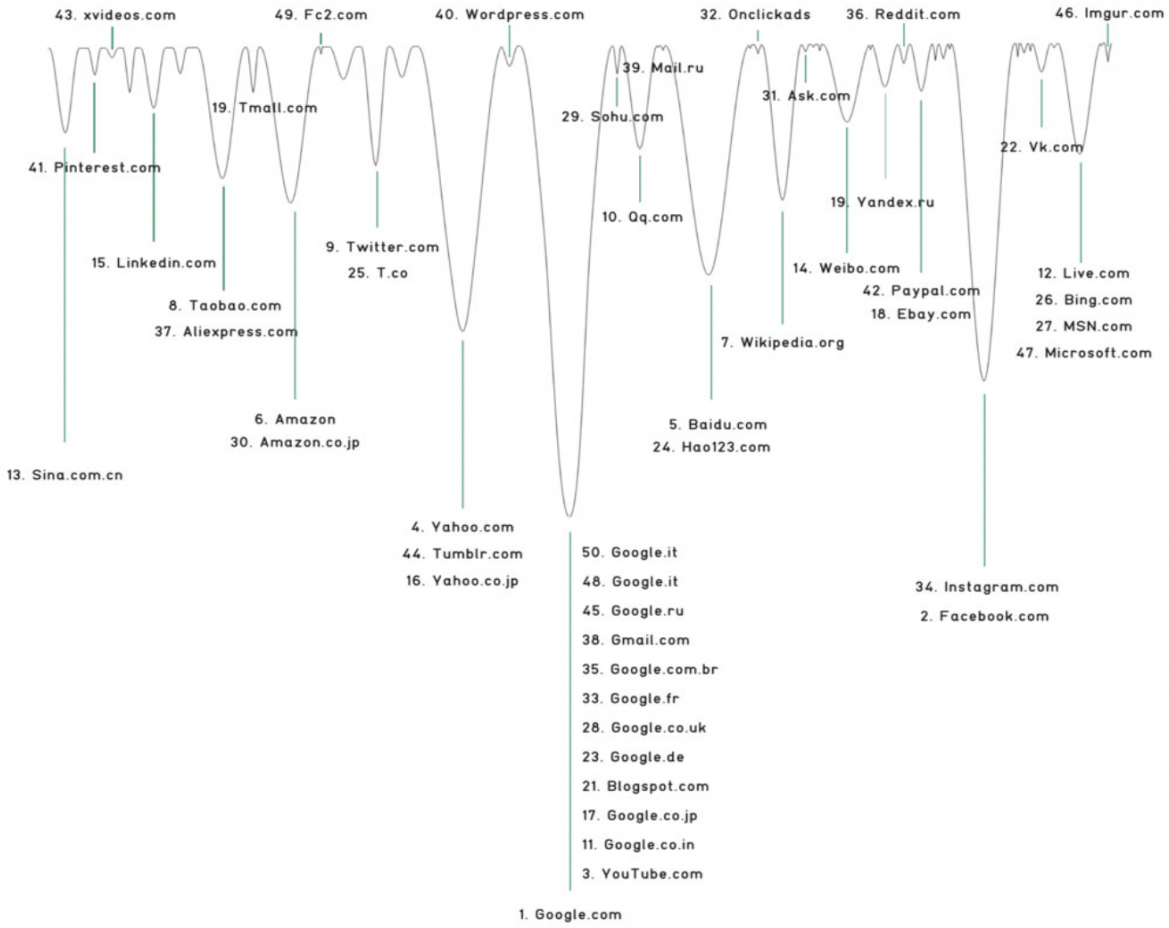
Link: Project

- Exploration of spatial analysis for socio-political purposes within the online realm.
- “In the early years of the Internet, it’s architecture was distributed, users published their own personal homepages and had a decentralized occupation of the space. Today, most of the activity is concentrated in the hands of a few players… These heavyweight players have dug into the Web surface, dragging activities down their slopes, activities that could have remained independent and decentralized.”



Metadata are absorbed through the porous surface of the Internet

❖ “When we browse the Internet, we automatically generate metadata. This type of data traces our online behavior patterns: what we have clicked on, looked at or looked for. These personal data, when dispersed, create a digital shadow of our online being. Once processed and gathered, the whole collection of data left implicitly by the user can reveal significant information enabling commercial targeting and/or mass surveillance. Metadata infiltrate the Internet’s surface where they are preciously collected. On the free Internet, we are not the clients but the products; we pay for online services by letting our digital personality be absorbed through the surface of the Internet.”



Web cross section
Cross section of the 50 top websites according to Alexa

Data Collection

My data collection journey began when I realized the potential to request my Spotify data, and from there, I endeavored to request as much data as possible from every major platform that I have created an account with, such as Instagram, Facebook, Twitter, Google, and more. Upon requesting data from these platforms, they compiled and provided me with a downloadable zipped file containing data associated with my account. While each platform's data package varied, the information generally included information pertaining to my profile, posts and other shared media, interactions and engagements on the platform, including likes, comments, and messages, ad interactions and custom audiences, login and account activity, and my account preferences and settings. It's worth noting that these platforms don't inherently *want* to provide users with their data; rather, the motivation is likely rooted in new privacy regulations and the growing user awareness about data collection practices. Regulations like the General Data Protection Regulation (GDPR) in Europe and the California Consumer Privacy Act (CCPA) in the United States have compelled companies to be more transparent about the data they collect.

The data collection process posed many challenges. Each platform determined its own delivery method. Some sent an email with a direct download link for the data package, while others required navigating deep into one's profile settings to find a request or download button. The waiting period for data package arrival varied, and each platform had its own timeframe. Notably, acquiring data from TikTok was particularly tedious, requiring users to request and download data on the platform without any email or notification, and only made available for a brief 3 day window before expiring.

Discovering datarequests.org proved instrumental while collecting data. The service lists dozens of companies, providing guidance on how to contact them. It automates the generation and sending of emails to selected companies requesting all user data, specifying crucial information to be included in the response. This includes the purposes of processing, categories of personal data, recipients of the data, storage period, data source, and more. The platform emphasizes the companies' obligation to provide requested information without undue delay and at no cost. My hope was to receive a more comprehensive set of data for more profound insights using this service. However, this method of data requesting fell short of my expectations. Facebook and Instagram blocked my request within minutes, citing unavailability in the U.S. and Canada due to the GDPR being a European Union regulation. Responses from the other companies varied; some claimed no account associated with my name/email, while others redirected me to their website for data download. Only a few companies processed my request, yielding data identical to that accessible through their platform. Nonetheless, datarequests.org emerged as a valuable resource. It helped me with identifying platforms I would have otherwise forgotten about, provided me with the language to identify myself to a company and make my requests clear, and importantly, informing me of my data rights.

Once received, the formats of all my data also posed challenges. Each platform determined the structure of the data package. Some platforms allowed me to choose the file format (e.g., JSON or TXT/HTML), and others offered options to select information for inclusion. Unzipping the data revealed a lack of consistency in file and folder organization, except for Instagram and Facebook, which share a parent company. Below is a list of all the platforms I requested my data from, with details such as when the request was made and the file format of the received data.

Platform-Specific Data Requests:

Instagram
JSON, multiple files organized into folders + HTML index.
Requested October 27th, received Oct. 26th.

Facebook
JSON, multiple files organized into folders.
Requested November 3rd.

Spotify (Account Data)
JSON, multiple files organized into folders.
Requested October 11th, received Oct. 14th.

Spotify (Extended Streaming History)
JSON, Multiple files.
Requested October 11th, received Oct. 30th.

Pinterest
HTML (only option).
Requested October 19th, received Oct. 19th.

Google Takeout
Multiple formats: JSON, HTML, and CSV, organized into folders.
Requested October 27th, received Oct. 27th.

Hinge
JSON, multiple files.
Requested October 27th, received Oct. 29th.

Apple
CSV, multiple files organized into folders.
Requested October 27th, received November 1st.

Tumblr
JSON, single file.
Requested November 4th, received Nov. 5th.

Reddit
CSV, multiple files.
Requested November 4th, received Nov. 6th.

Discord
JSON, multiple files organized into folders.
Requested November 4th, received Nov. 8th

Soundcloud
CSV, multiple files.
Requested November 8th, received Nov. 16th

TikTok
JSON, single file.
Requested October 27th.

Twitter
JavaScript, multiple files organized into folders + HTML index file.
Received November 11th.

Snapchat
JSON, multiple files organized into folders + HTML index file.
Requested November 29th, received Nov. 29th.

Netflix
CSV and TXT, multiple files organized into folders.
Requested November 29th, received Nov. 29th

Data Integration:
The current version of the project uses only a fraction of the acquired data. Nonetheless, the program is (theoretically) designed to seamlessly incorporate additional data (always easier said then done, of course). MongoDB serves as the database, handling the storage and management of the selected data as separate collections. Presently, the project only visualizes the following collections in the MongoDB database: my Hinge match data, Instagram messages, and media-related data. However, the database also includes the data utilized in my project prototype, which featured information from Instagram, Facebook, Twitter, and Pinterest concerning advertisers and inferences drawn from my interactions on those platforms. This dataset has undergone slight reformatting to better align with the structure of the final version as opposed to the prototype, and has been expanded to include additional data sourced from more

platforms such as Tumblr and Spotify. However, this extended dataset has yet to be fully integrated into the final project. The following provides a breakdown of each data collection currently residing in the project's MongoDB database, specifying the data source, interpretation methods, and any potential alterations applied to the data.

hinge_matches Collection:

This collection stores my data from the Hinge dating app, extracted from the **matches.json** file within my Hinge data package. The JSON file was inserted into the MongoDB collection in its original, unaltered state.

Briefly, some background on Hinge may be necessary to better understand the data. The platform is marketed as a dating app designed to foster so-called 'meaningful connections.' Users create profiles with photos and respond to prompts that serve as conversation starters. Users can express interest by "liking" specific photos or prompts on others' profiles. Upon receiving a like, users can view the sender's profile and choose to reciprocate interest, leading to a mutual match and unlocking the chat feature. Each document in the collection represents interactions such as sent likes, chats, matches, and blocks within the Hinge dating app.

Likes: This property records instances where I sent a 'like' to another user's photo or prompt. It includes details like the specific prompt or photo URL, timestamp of the like, and additional information like comments.

Matches: The "match" property seems to be instances of mutual interest, resulting in a match. It includes a timestamp and a type identifier.

Chats: The "chats" property includes an array of messages that I sent to a match within the app. It is a sort of one-way conversation, as the data package excludes messages sent by the other user. Each entry includes the message body, timestamp, and a type identifier.

Blocks: This property seems to be cases where one user blocks another, terminating communication. It incorporates a timestamp and a type identifier specifying the block event. Notably the nested "block_type" property consistently equals 'removed.' This property most often stands alone as the only property in the object, only occasionally does it appear alongside a 'like' or 'chats.' This raises several unanswered questions. Are these instances of a user manually removing a user from their matches? If so, was I the removed, or the remover? Or are these instances of matches getting automatically removed after a period of being inactive? Could these instances where I chose to not match with a user who expressed interest, thus 'removing' them? Or vice-versa?

We Met: data concerning the physical meetings between users. Following a substantial interaction history on the platform, users are queried by the app regarding whether they have met in person or not. The property includes a timestamp, and a meeting status (often "No").

The naming of the json file as 'matches' may be somewhat misleading, as the data seems to encompass various types of interaction beyond just mutual matches. Intriguing patterns, such as 'matches' without corresponding 'likes' or 'likes' without subsequent 'matches,' prompt nuanced interpretations. I've interpreted the latter as times where I have sent a like to another user but did not receive a like back, and the former as moments when another user showed interest through a like, and I opted to reciprocate with a match. Another recurring pattern emerged in the form of 'matches' unaccompanied by 'chats,' suggesting mutual interest but a lack of ensuing conversation. This pattern really got me thinking. Regardless of the

app’s goal of fostering connection, the dynamics of online dating often reduce other users to mere data points. We collect 'likes' and 'matches' not for meaningful connection but as a form of validation—another means to satisfy our online interactions.

instagramDM Collection:

This collection stores my direct messages on Instagram. The Instagram data package organized each chat thread into separate JSON files, residing in the path: **[data package for a specific instagram account] > messages > inbox > [individual chat thread folder] > messages_1.json.**

Over several years of interacting and messaging on Instagram across multiple accounts, I was met with hundreds of chat files to work with. However, many of these files included extremely long arrays which raised concerns about potential program slowdowns. MongoDB's limitations in adding multiple files at once also steered the decision-making process. Consequently, I decided to manually selected specific chat threads to add into the Mongo collection. My choices prioritized threads that were historically active, content-rich, and emotionally resonant—such as group chats with old friends or direct messages with significant connections.

Each document in this collection represents a chat thread and includes participant information, message history, and thread details. The "participants" array lists the individuals involved in the chat. The "messages" array contains message details, including sender, timestamp, content, and geoblocking status. Other then the manual curation of files, the data within this collection remains unaltered, preserving the original structure as sent by Instagram.

Media Collection:

While exploring my various data packages, I would occasionally encounter images that triggered a sense of sentimentality or nostalgia. These moments inspired me to create a ‘media’ folder for storing and quickly accessing these images. Currently, the folder primarily consists of selected images discovered in my Blogger data from my Google Takeout package. These images were once uploaded to Blogger in 2012 when I was 10 years old. I only had the faintest memory of ever creating these ‘blogs,’ they had been long forgotten and are now mostly defunct. The images from the Blogger package were no longer accessible online nor on the family computer I used for ‘blogging’ back in the day, existing solely within Google’s database for nearly a decade.

vampire.png
PNG image - 30 KB

The decision to archive these images led to the conception of the Media Collection. MongoDB was chosen as the tool to manage this image archive, providing a structured approach to tracking image sources and associated data. Its purpose is to centralize the archiving process, keeping track of image details such as source and metadata provided by the package. Each document within the collection represents a media file and includes essential details like the source, filename, and creation timestamp. The filename property (e.g., "vampire.png") is used as the primary unique identifier, linking the data to the associated photo in the project’s ‘media’ folder. This collection serves as a repository for all media-related data, one that I can continually update as my exploration into my personal data continues.

Inferences Collection:

Utilized in the project prototype but not yet integrated into the final version, the Inferences Collection consists of documents representing assumptions about my interests based on activities across platforms like Instagram, Facebook, Twitter, and Pinterest. Subsequent to the prototype, I expanded this collection to encompass inferences made by Tumblr and Spotify.

Given the varied formatting of data from different sources, I manually standardized the data to maintain consistency. The "source" field designates the site and account from which the data is sourced. Meanwhile, the "topic" array contains inferred topics discovered within the data package.

Extracting these topics required manual extraction, reformatting into an array of strings, and insertion into the “topics” property of each document in the collection. For each platform, the data was located in the following paths:

- ➔ [Instagram package] > your_topics > **your_topics.json**
- ➔ [Twitter package] > **personalization.js** → extracted the "interests" property

Advertisers Collection:

This collection, akin to the Inferences collection, was created for the prototype and has yet to be integrated into the final version. It has undergone formatting alterations and expansion to include more data. Similar to the Inferences collection, the data underwent an extraction and formatting processes, necessitating conversion into an array of strings and manual insertion into the “advertisers” property.

- ➔ [Instagram package] > instagram_ads_and_businesses > **advertisers_using_your_activity_or_information.json**
- ➔ [Twitter package] > personalization.js → then extracted the "**audienceAndAdvertisers**" property, which had a nesed array called "advertisers"

Technology Overview and Program Logic

Server-Side:

The server-side logic, implemented in index.js, utilizes the Express.js framework for routing and serving data. MongoDB is employed as the database, collections are retrieved and assigned endpoints (e.g., '/hingeData', '/instagramData'). A single function is used for fetching the data from the database, and utilizes optional callbacks which allows for flexible filtering and sorting of data before sending it to the client-side application.

Function for Fetching Data from MongoDB:

- ➢ `const fetchDataFromCollection = async (res, collectionName, callback) => {...}`
- ➢ Connects to MongoDB, fetches data from a specified collection, and sends it as a JSON response.
- ➢ Accepts an optional callback function to filter or sort the data.

By preparing and processing the data server-side, I ensure that the client-side logic is exclusively dedicated to visualization of the data. This way I can focus on processing new data without knowing how it will be visualized yet or worrying about messing up what visuals are already implemented, and vice versa.

The script utilizes natural language processing tools from the Node Natural library. It uses tokenization, stemming, part-of-speech tagging, and Tf-idf analysis for processing Instagram direct

messages. The script also includes a word frequency analysis and part-of-speech tagging for better understanding the textual data.

Natural Language Toolkit (Natural): Natural is a natural language processing library for Node.js. It provides tools for tokenization, stemming, and part-of-speech tagging. Natural is used for analyzing and processing textual data from 'instagram'. The project is structured so that I could reuse the same word processing logic with other textual data. It aids in extracting meaningful insights from messages and performing tasks like word frequency analysis.

Tf-idf (Term Frequency-Inverse Document Frequency): Tf-idf is a numerical statistic that reflects the importance of a term in a document relative to a collection of documents. Tf-idf is used to perform word frequency analysis and identify terms in the 'Instagram' data. It helps in understanding the significance of terms within the context of the entire document collection.

Brill POS Tagger: Brill POS Tagger is a part-of-speech tagging tool provided by the Natural library. It assigns part-of-speech tags to words in a text. The POS tagging tool is utilized to analyze the grammatical structure of messages in the 'instagram' data. It aids in categorizing words based on their syntactic roles.

Client-Side Application

Primarily built with the p5.js library, the client-side application is initiated through the client.html file. The application runs on a localHost in my web browser, establishing a connection to the server-side logic implemented in Node.js. Upon loading, data is fetched from the server. The JavaScript is organized into several classes, each handling a specific aspect of visualization. Main.js manages data fetching, UI display, and calls functions from each class for display and mouse interaction.

Main.js - Data Management and UI Display

- **Data Fetching & State Management:**
  - Handles data fetching and state management
  - Initiates the fetch process upon loading and sets the application state accordingly.
- **Class Object Creation:**
  - Utilizes the `createClassObj` function to create class objects from fetched data.
  - Facilitates better organization and manipulation of the data.

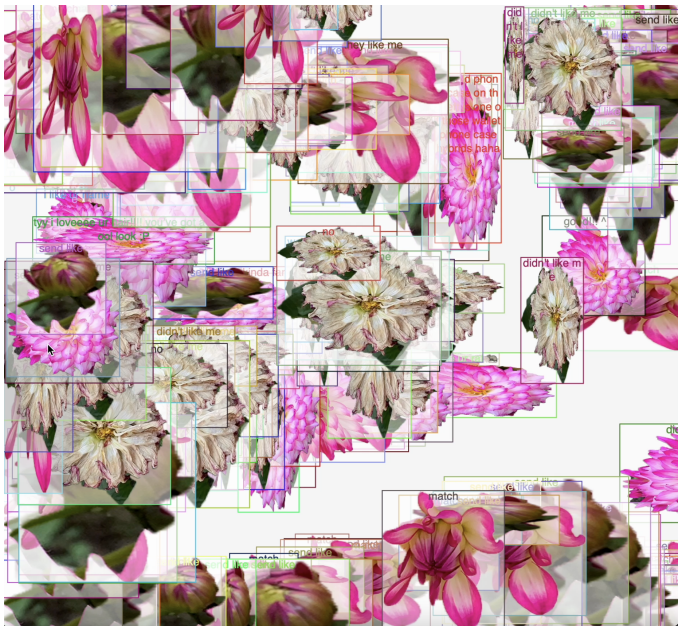
Hinge Visualization - Flower Garden

When the user clicks, they create a new object from a class called Match. Each instance uses a random object from the `hinge_matches` collection retrieved from Mongo. Each instance undergoes a lifecycle with different states. Here's an explanation of how it works: The constructor initializes various properties of a Match object; the X and Y positions are the user's mouse positions when they click on the canvas. The init method sets the initial state of the Match based on the hinge match data, determining if a like was sent or if there was a match. If neither a like nor a match exists, meaning that the only property is `Blocks,` a new random object in the collection is selected.

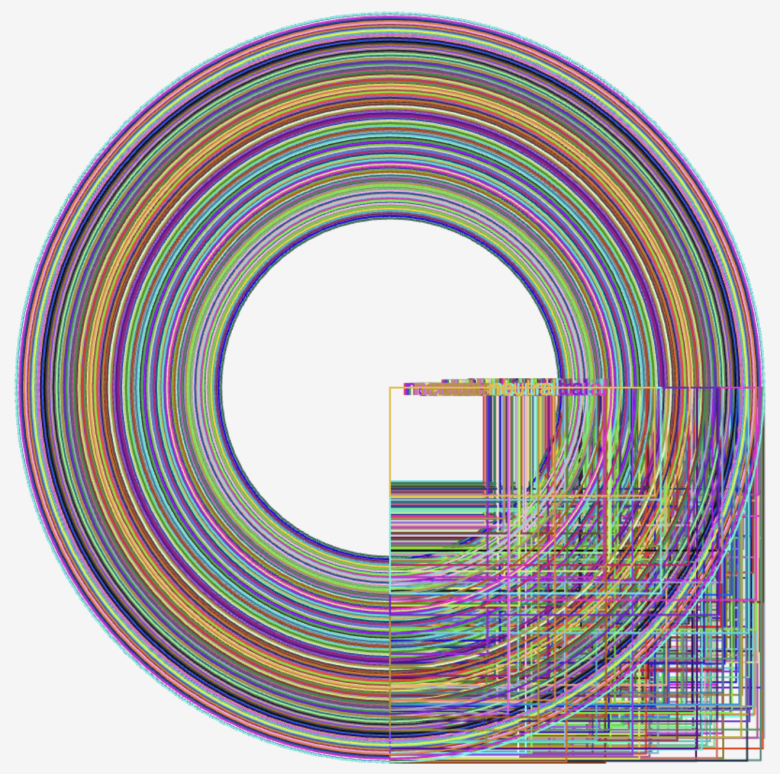
Lifecycle Methods:

The class includes methods that simulate the lifecycle of a hinge match. Matches spawn as a young flower bud, grow, and eventually wither away over time. The changeState method is triggered on mouse click and handles state transitions based on the current state and the properties of that particular Hinge match object. It

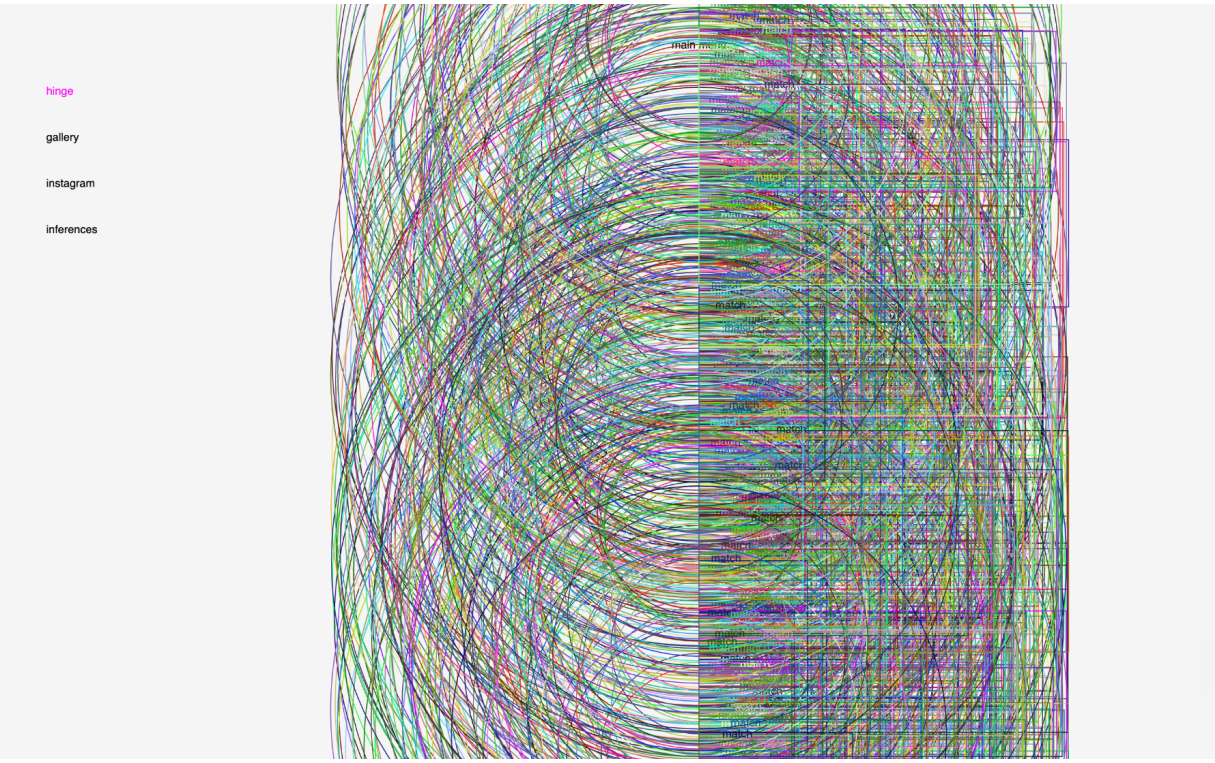
considers scenarios such as sending a like, matching, engaging in chats, and if the users have met. The flower image changes based on the state, representing the stage in the match lifecycle. The lifecycle begins either as a like sent by me, represented by a flower bud, or as a match, represented by a young budding flower. When the user clicks on the image, it progresses to the next stage, determined by the properties in that instance of a hinge match. For instance, if the data has both a like and match property, the instance starts as a flower bud and, when clicked, transforms into a budding flower with the text “they like me.” However, if the data has a like but no match, it starts as a bud image, and when clicked, the flower begins to decay, displaying the text “didn’t like me back”, shrinking until it disappears. Sometimes, the lifecycle can be very long. If the data includes chat records, meaning the fellow hinge user and I had conversations on the app, the flower iterates through each message I sent when clicked, eventually reaching the end of the conversation and before withering away.



While developing this visualization, an interesting phenomenon occured. I started the visual before opting for classes dedicated to each data collection that could manage themselves. The original hinge visual was made in a different script file named sketch.js, which still utilizing the Match class. However, much of the user interactions were managed in the sketch.js script rather than by the class itself. Later, I refined the class and merged it into the main.js alongside the Instagram and gallery visuals. An unresolved issue from this integration is the touchStart function not functioning correctly. In the original, users could drag the mouse, continuously spawning new flowers. When the mouse crossed paths with flowers, it triggered a lifecycle iteration, creating a busier and more dynamic visual, albeit less intimate. In the version presented during class, I couldn't get the touchStart function to work properly due to the way images were preloaded. In the original Hinge visual, all images were preloaded in the main script, but in the final version, the images had to be preloaded by the class, resulting in a delay when each flower is spawned. While this delay is less noticeable when flowers are spawned one at a time by clicking, it becomes obvious when many are generated at once. I plan to address this issue eventually because I loved the effect created by the touchStart function.



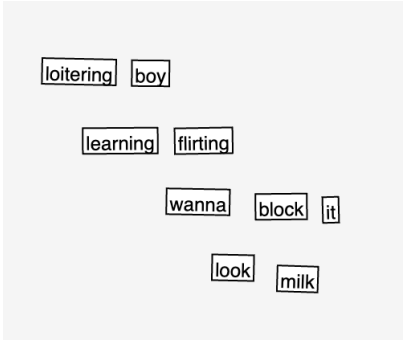
A more serendipitous side-effect of this integration was a visually striking glitchy display that occurred while modifying the Match class. It appeared that every match in the array was being displayed simultaneously, before images were reintegrated and a with a leftover ellipse existing originally for troubleshooting being displayed. These happy accidents were too visually appealing to ‘fix,’ so I decided to create a copy of the Match class, naming it ‘AllMatches,’ that produced these visuals before continuing to modify the original class for integration into the project. I experimented with some display parameters of the AllMatches class to see what else might happen, resulting in another captivating visual. I wanted these to be incorporated into the project somehow. Hence, I configured an instance of the AllMatches class to appear when the user hovers the mouse over the hinge text button, offering a glitchy sneak-peek into the Hinge visualization.



Instagram Visualization: Fridge Magnet Poetry

This visualization transforms my Instagram direct message (DM) data into an interactive representation of magnetic fridge words. Inspired by bryant4882's p5.js sketch, my script, magnets.js, undergoes significant adaptations to better suit my project's client-side logic.

Script Adaptation



In terms of script adaptation, my version uses an array (`Words`) derived from InstagramDM data, in contrast to the original script that relies on an external file. My script adopts a more modular structure with encapsulated classes, including a new class called Magnets to represent the entire group of words. These changes were done to allow my Main script to focus only on calling display and click functions, as well as data fetching. In terms of visualization, only words with the highest frequency (using Tf-idf) are displayed, with POS-specific thresholds for importance. My script introduces these thresholds for word filtering based on frequency and importance, leveraging POS tags obtained during server-side filtering and tagging of the Words array. Adjustments to the container display method allow for clustering of words with the same POS.

In the future, I would like to implement a method for live editing of these thresholds. Additionally, the script's is versatile, and could potentially be reused with different textual data sources such as comments, captions, or tweets.

Media Visualization - Media Gallery

This visual is the simplest of them all, providing a straightforward way to display images from the Media collection. Currently, the images are sourced from my Google Takeout Blogger folder, but I plan to add more images eventually. In the main.js script, the media collection is fetched from the server, and then each item in the media array is iterated through, creating an instance of the Image class. The Image class serves as a blueprint for creating instances of images and handling their display. The Image class constructor takes the media data as a parameter, extracting different properties associated with the image (e.g., filename, creation timestamp). It uses the loadImage function to load the image file from the specified path and assigns it to the image property. The displayImage function calculates a scaling factor based on the maximum size allowed for the image and uses the p5 image function to draw the image on the canvas at the specified position with the calculated dimensions. There is also a function called formatTimestamp, which takes the image's timestamp in milliseconds and returns it as a JavaScript Date object. Lastly, the class handles user clicks on the image for navigation. It checks the mouse position concerning the canvas width and increments or decrements the imgIndex accordingly. If the end of the image array is reached, it resets imgIndex to 0.

jan1320010 053.JPG
Sun Mar 18 2012 10:59:52 GMT-0400 (Eastern Daylight Time)

webcam-toy-photo5.jpeg
Fri Jan 06 2012 18:06:44 GMT-0500 (Eastern Standard Time)

Future Steps

There are many possibilities for the future of this project. I've already mentioned a few things: fixing the Hinge touchStart by altering the logic of the Match class and how it handles preloading images. For the fridge magnets, I would like to implement a method for live editing of these thresholds. Additionally, adding more chat threads to the InstagramDM collection and likely changing the words in the magnet visual to make it more 'accurate' to the words I use over DMs. Regarding the media visual, I plan to keep adding more images to the gallery. I also want to display which data package the images were sourced from once I have images from more sources beyond Google. Some feedback I received during the in-class demo was to include some sort of explanation of what the data means and where it comes from so that the site can stand alone. I think this is a good idea, even if the site is not something that becomes publicly

browsable. It would be beneficial to include this information for my future self, for documentation purposes. Furthermore, I really want to integrate the data used in the prototype, the inferences and advertiser data. I didn't integrate them immediately because I was on the fence about the visual I had created for them. I wanted to think of a different way to visualize it, but nothing came to mind. Hopefully, I will think of something one day. If not, I may just integrate them exactly as they were visualized in the prototype. Moreover, I aim to create more visuals for different data. There is so much material to work with in all my different data packages; this could be a lifelong project.