Plant Your Own Produce Report

Medha Bulumulla, Hannah Rudt, Brett Schelsinger
INFO 4310/5311 Final Project
May 15, 2023

Project Goals, Motivations, and Intended Audience

Our project aims to provide interactive tools for audiences interested in gardening and cooking. We are using recipe and garden data to help users determine which of the vegetables and herbs they like to eat can be easily planted at home.

One use case is for a user interested in starting a garden. Before investing time, effort, and money into plants they cannot cook with or plants that do not grow well together, the user can explore which plants will fit their needs best. They can learn about the most popular plants, compatible plants, and recipes to make with the plants they become interested in. Another use case is for a user who already has a garden. With the compatible plant information, they may find additional plants to add to their garden. They can also find recipes that include the plants already in their garden.

Related Materials

The title page of our project was generally inspired by the New York Times data article style in which there is a captivating title page that fades away as the reader enters the article. We were specifically inspired by the INFO 4310 group with the F1 project; they used small animations to create a clean design that directed users to scroll through the article.

Data Cleaning

As mentioned above, we combine two datasets to have a more fruitful exploration. The recipe data contains information about common recipes, their ingredients, recipe instructions, and images. They were sourced from the cooking website Epicurious. The garden data contains information about 92 plants, with information about sowing, spacing, soil temperature, harvesting, compatible plants, incompatible plants, culinary purposes, preservation tips, and the link of where the information was sourced. This data was collected by Gardenate.

Both datasets were fairly messy and required significant string cleaning and function application to get the desired outcome. The garden data was compiled by a home gardening site; many of the features contained multiple pieces of information in one sentence which required parsing. We did our initial cleaning using simple Excel functions. The first step was to parse all of the possible names for each plant; some plants are known by five different names which all

had to be separated and factored into our recipe search later. All of the planting information with a minimum and maximum value (soil temperature, seed spacing, harvest time) was written in full sentences within one cell. We divided this information into multiple features (e.g., minTemp, maxTemp) and changed the strings to numeric values. Additionally, the compatible and incompatible plant lists were written in single cells as full sentences. We had to conduct more complex parsing to remove the extra words and divide the lists of plants into multiple cells (e.g., "Onions are compatible with garlic, peppers, and tomatoes" was divided amongst three new features of compat1, compat2, compat3). Finally, we gleaned extra information by summing the number of compatible edges each plant had.

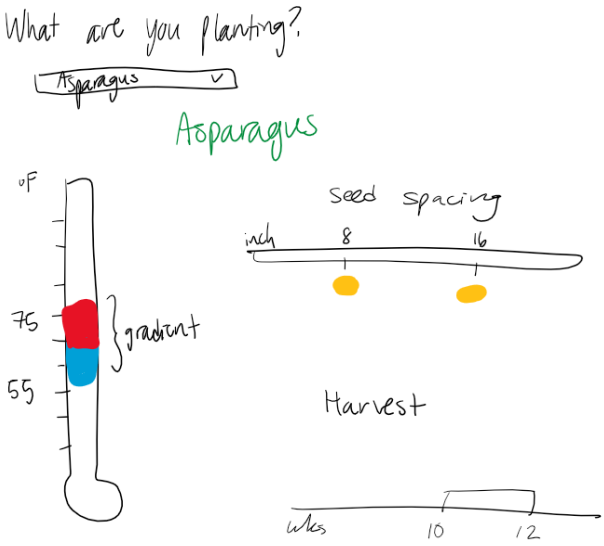
To clean the recipe data, we used Excel to parse the single-cell ingredient string into multiple cells (one ingredient per cell). This was simpler because the project relies less heavily on the direct data from the recipes and more on the interaction between the recipe data and the garden data. To combine the two datasets, we used an Excel search and sum function (=COUNTIF) to identify the number of recipes that contain any of the possible names for each plant.

In Python, additional cleaning to account for plurality (strawberry vs. strawberries), spaces, and multiple names had to be taken into consideration. A full adjacency matrix with the compatible plants was constructed. Using the compat1-compat11 variables, anytime a plant was compatible with another plant, the associated entry would be marked 1. To make a subset of the adjacency matrix, each row of the matrix was summed and the plants with a sum higher than 7 were kept in the adjacency subset matrix. This was exported as a csv and json to be used in the second graph, the network graph.

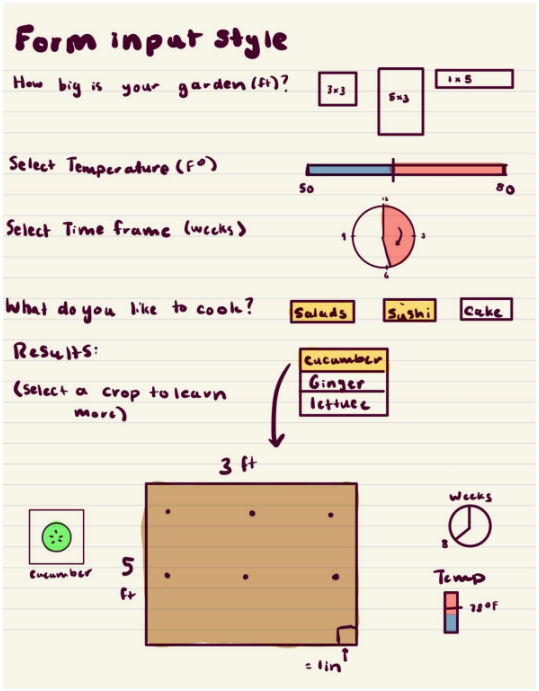
The second Python task was to create a dataset that identifies which plants are in the recipes. The recipe directions were converted into a list and iterated through to find commonalities between the plants we have information on (in the garden dataset). For each recipe, we identified plants that are in the recipes and created a column of this in the ingredients-and-crops.csv. This was used for the last graph to find intersections between plants.

Design Iterations

When sketching and developing the design for this project, many of our original designs focused on the specific gardening information for each plant. One idea was to create a dashboard for each plant that would change with a dropdown menu. Another idea was to make a form into which users put the specifications of their gardens. The page would then visualize the user's garden and provide growing recommendations.



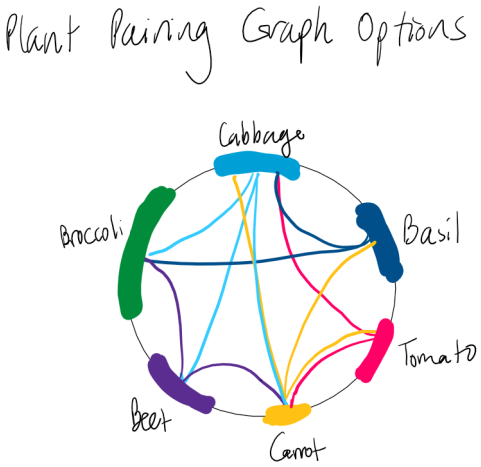
[Figure 1: Individual plant dashboard sketch]



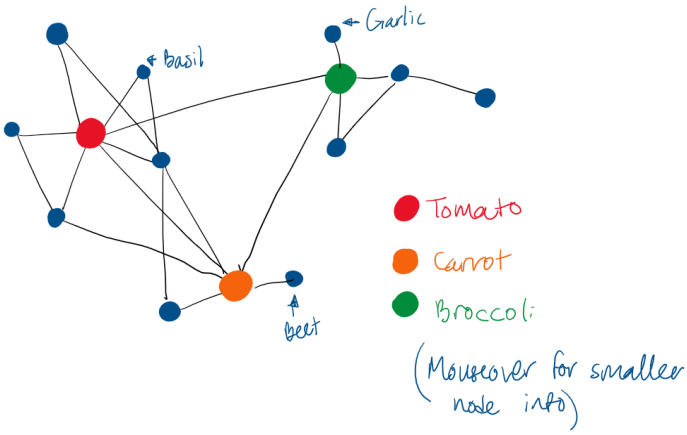
[Figure 2: Garden form sketch]

While these options were visually fun, they did not provide the context for a garden that we wanted users to have, because gardens typically include more than one plant. For the design option above, there was little room for comparison between plants and their growing conditions. Additionally, there was no connection to the recipes that use the plants, which is the ultimate goal of the project. We wanted to reconsider how to include this planting information while allowing for contextualization and comparison.

Our next design idea included more relation between the plants. We explored various edge and node graph options to show which plants grow well and taste good together.



[Figure 3: Plant compatibility chord diagram]

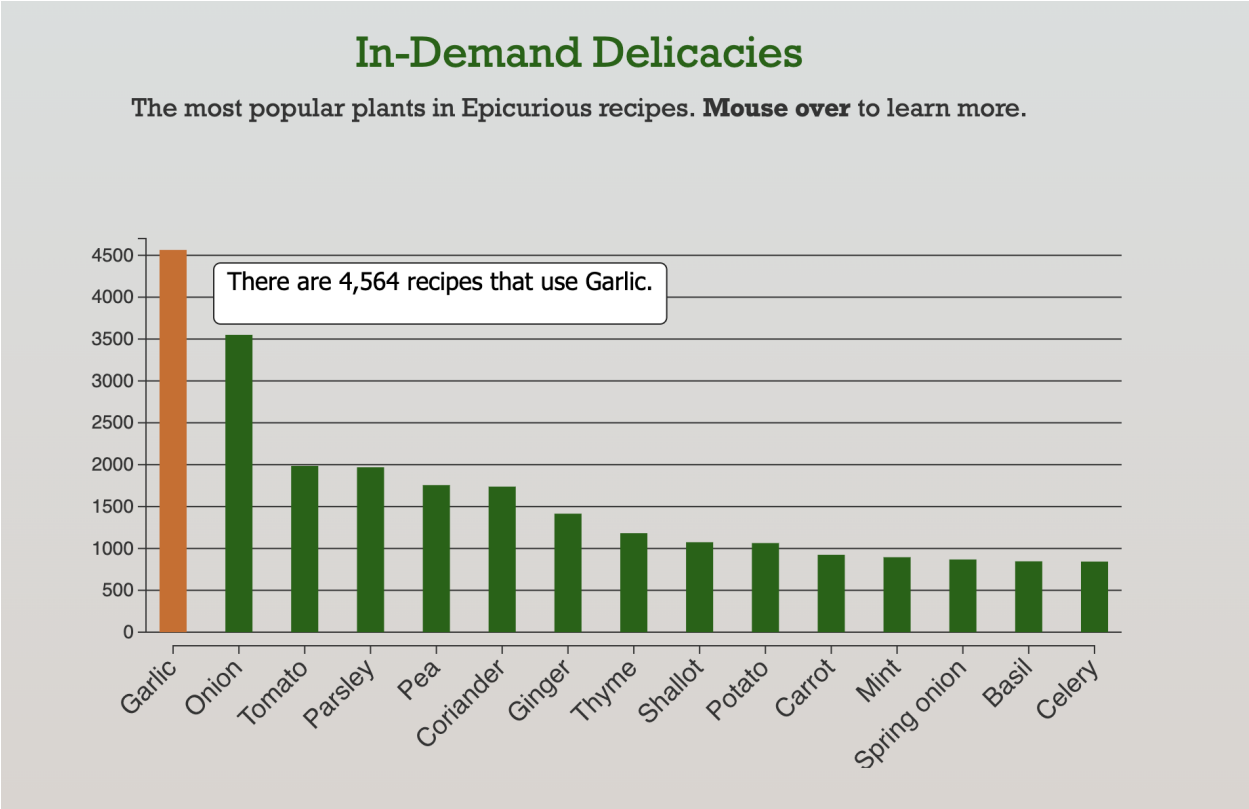


[Figure 4: Plant compatibility network graph]

The chord diagram on the left had potential to show the relationships between compatible plants, but became too convoluted when thinking about the details of implementation. One of the strengths of a chord diagram is the varying widths of the edges to convey an additional variable. For our data, the "weight" of the edges (the additional variable) could have either reflected the number of connections each plant has or the number of recipes each plant pairing has, but these differences would not have been easily perceptible to users. Another tradeoff with the chord diagram is the lack of interactivity and network visibility. The structured circular format could make it harder for users to see which nodes have the most connections. The network diagram on the right, however, allows users to move the diagram around and see more clearly which nodes are "hubs" as plants with many compatibilities.

Final Design Description & Implementation

Our final design features 3 separate visualizations that each tackle different parts of our story. We wanted to create a fun and playful story and tool for individuals to determine what to plant and cook.



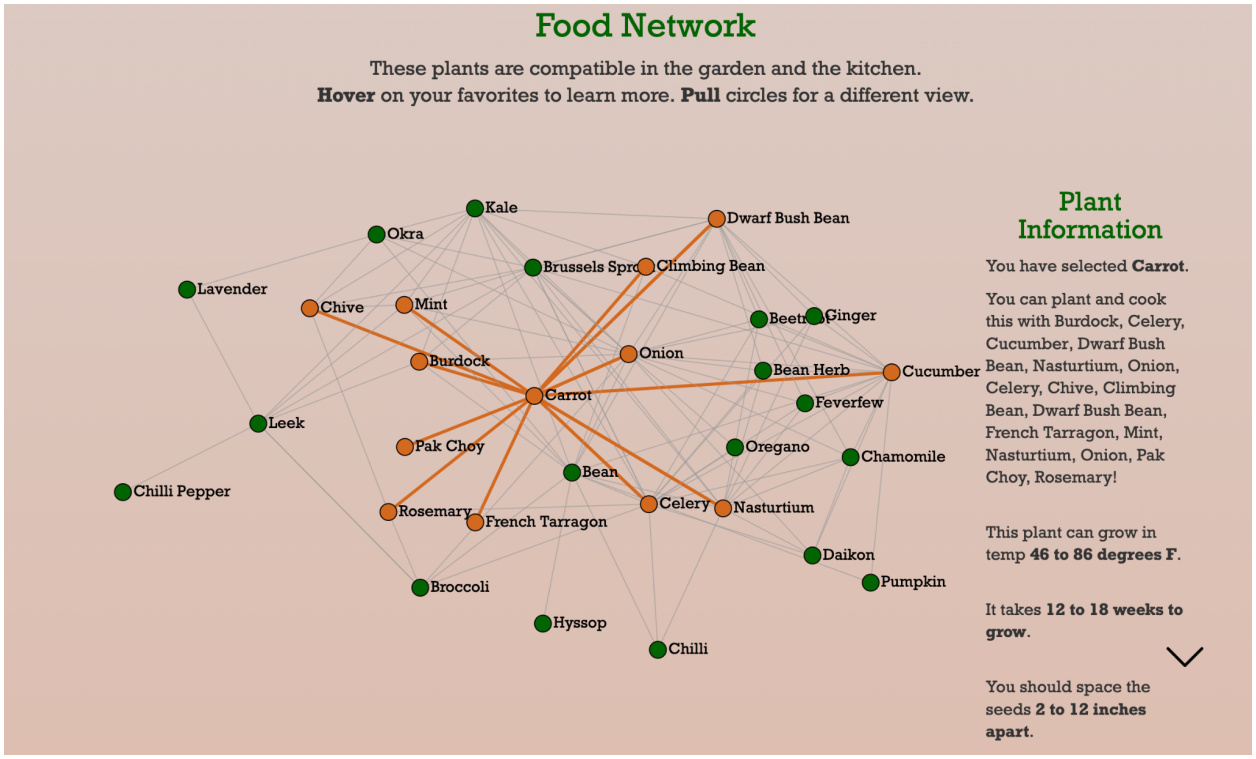
[Figure 5: Bar Graph]

The first visualization is a bar graph showing the top fifteen crops that are featured in the Epicurious recipe database. The goal of this visualization is to provide the user information on which plants are most popular in the kitchen if the user does not know which plant would be most helpful in their garden. It relies on aligned position and height as visual channels to communicate how many recipes each plant can be used for. The marks are the green rectangles or bars. The user can interact with the visualization by hovering on the bar, the color changes to signify that it has been selected. Once the bar is moused over, a text box appears that indicates the exact number of recipes to aid in the understanding of the visualization.

To get the data for this visualization, we created a subset of the top 15 plants using JavaScript. The actual bar graph and mouseover functionality is standard code, but the original design did not show the entire y-axis from 0 to 4500+ recipes. This squished the bars on the right side of the graph and prevented mouseover capability. To fix this, we extended the y-axis by adding space to the y-axis annotations and bars. The mouseover text box uses code from a 3300 lecture with "dummy text" to help determine the width of the text box. The function stringLen(str) calculates the length of the text of each mouseover (in this case varying by plant name) and creates an invisible text box until the bar is hovered over. This small detail allowed for a cleaner design than a static-size text box that would either have extra space or require two lines of text.

Another feature of this bar chart is the animation of the bars, which shoot up from the x-axis in succession. Because the user does not see this graph first, however, we used

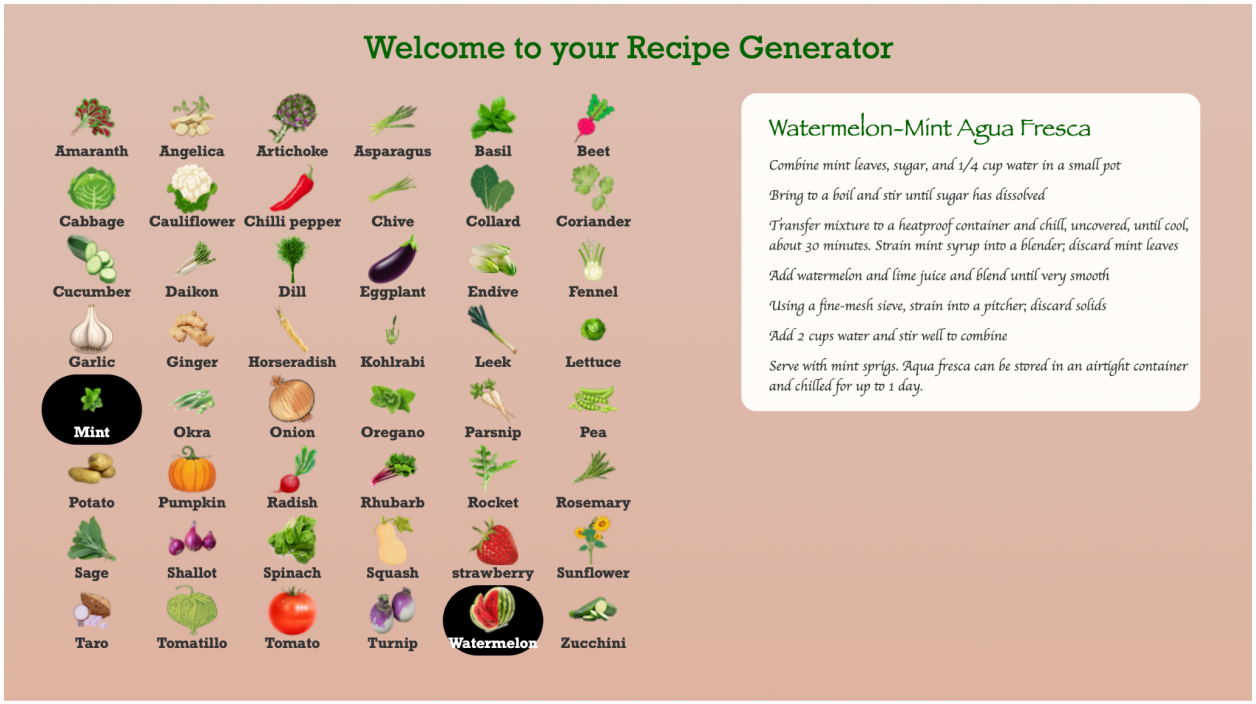
Scrollama.js to hold the animation until the user reached this part of the page. Unfortunately, the functionality is variable and works sometimes (the animation is triggering early and we don't know why), but if the user refreshes the page at this graph they will see the animation run.



[Figure 6: Network Graph]

The second visualization is a network graph that shows which plants can be grown together in the garden. The goal of this visualization is for the user to identify crops that are most versatile in the garden to help them make a decision on what they could plant together. The visualizations utilize edges of the marks, green dots that represent each plant, to communicate which plants can go together. Originally, we had all of the data in the network graph but through feedback we received, it was difficult to parse through so much data and the graph would always move to the left corner. We subsetted the data by focusing on plants with the most edges. When hovering on each dot, it highlights all plants that are connected to the selected plant. This was a suggestion we received multiple times in feedback to help the user identify which plants they are interested in. When hovering, there is also an information box with plant information. It provides a list of the compatible plants for easy reference, soil temperature, harvest time, and soil spacing. This information is pertinent when deciding a plant so it is important to include this to aid the user in making their decision on what the plant is. In order to implement this visualization, we utilized an adjacency matrix where "0" represented a lack of compatibility between plants, and "1" represented compatibility between them. We iterated through the matrix with a for loop, and then utilized the output to build out the network chart with the built in d3.forceSimulation function.

We created customized mouseover and mouseout functions so that the user can interact with the graph, and within the mouseover function connected nodes to the selected one were highlighted, and the color reverts back with mouseout. And lastly, we built out a sidebar providing information about the selected plant in the network chart and utilized regular expressions to convert the data present in the chart to readable information to present to the user.



[Figure 7: Recipe Generator]

The final visualization is a selection tool where the user can select multiple crops and find random recipes they can make using the selected crops. The goal of this visualization is to utilize what the user learned from the first visualization to select their ideal crops and find a recipe they can use. In this visualization, we wanted to mimic harvesting specific crops, bringing them to the kitchen when you decide something to create. We wanted to give a shop feel in the last visualization with a recipe generator added on. The user can interact with the visualization by selecting (or harvesting) one or more crops and receiving a random recipe with those crops if they exist. We wanted to make the recipe information on the right look like a written recipe card.

We utilized both text and vegetable imagery for the marks. Originally we only had text but we found this unappealing to work with and did not communicate the playful aspect. Then we only used images, but as it was the first time the user was interacting with the images, it could be difficult to determine which plants were which so we decided to use both text and imagery. We found clipart online for each of our interested plants and all the source links are located in /images/clip-art/original/clip-art-sources.rtf.

The recipe generator relies on the data we created in Python called ingredients-and-crops.csv that includes which plants in our garden dataset are in each recipe[1]. We choose plants that are in numerous recipes but some were manually dropped after interacting with the visualization and finding few common recipes with other crops. For all plants that we were interested in showing, we added the relevant image path into the garden data in addition to a binary variable, imgPresent, that allowed us to determine which variables to show. We used a subset of the plant data that had image information using imgPresent. Through this subset, we added the images and text that act as a button. When clicked on the image, the background color changes to signify that it has been clicked, we also add a hover element so the cursor changes to signify that something occurs. Once a plant is clicked on, the function purchaseSeeds(selectedCrops, crop, cropLower) finds a subset of the recipe data that contains all of the selectedCrops in the recipe. This function also removes the selected crop from the selectedCrops list if the crop is already there. After some string cleaning and conversion to a list, we find the subset of recipes that share the same selectedCrops. Then we randomly generate a number based on the amount of recipes to find a random recipe from the subset. The recipe directions are converted into a list split by each sentence, and each line is displayed on the recipe card. The user can select multiple plants that might not have a recipe in common, so we just inform the user to select something else.

Work Breakdown

Hannah

- Clean raw data in Excel (string parsing, recipe searching and summing)
- Interactive bar graph and animation
- Background research and photos for article aspect
- Styling - Next buttons, jumping text
- Report

Medha

- Create new datasets and clean data in Python for recipe and planting connections (appropriate formatting for different graphs)
- Seed catalog selection and recipe list functionality based on user selection
- Icon imagery
- Fonts, organization, styling
- Report

Brett

- Interactive network graph
- Sidebar
- Styling and project optimization

[1] Refer to Data Cleaning for information