The Ultimate Davis Coffee Guide

By: Priyal Patel & Venice Hodac

STA 141B Final Project

Contributions:

Priyal Patel: Obtained and preprocessed the data for Pachamama Coffee, Volt Coffee, and Temple Coffee. Preprocessed and visualized the time and location data for all coffee shops.

Venice Hodac: Obtained and preprocessed the data for Mishka's Coffee and Peet's Coffee. Consolidated all coffee shop data and developed the find_drinks() algorithm.

Abstract

Our objective is to find the optimal coffee shop that has our target drink at our specified time and location. We used web scraping tools and API calls to retrieve the menu, hours, and location data for five local coffee shops in downtown Davis. The final results for all stores were consolidated in a table that can be filtered by keywords and a series of interactive maps that display what coffee locations are open at the chosen time for each day of the week.

Introduction

Coffee is a vital part of a majority of college students' mental and physical well-being -- sometimes even a necessity to their everyday routine. As two coffee enthusiasts, we sought to find a comprehensive way for college students to compare drinks at different local coffee shops rather than pursuing multiple pages and apps to determine our choice for the day. The results gathered in this report can be beneficial for students to see if they can get their favorite drink somewhere else for cheaper, where to find a particular type of drink, or where to find a drink that contains specific ingredients. To make the process more practical, we chose to compare the times each store is open and map these open locations. This way, students can choose between the stores that are operating at their desired time. Overall, this report aims to simplify the process of choosing a coffee shop by providing all the information in one

convenient location and presenting this information in a format that is easy to facilitate in the decision-making process.

The five local coffee shops chosen were Misha's Coffee, Peet's Coffee, Volt Coffee, Pachamama Coffee, and Temple Coffee Roasters. All data was collected from various websites: Yelp, Uber Eats, or a coffee shop's official menu ordering site. Using this information, our report focuses on answering these central questions:

1. Where can you find your drink of choice by keywords?
2. When and where is your drink of choice available?
3. Where is your drink of choice the cheapest?

Our project uses the web scraping tool Selenium and API calls to retrieve all information. After, we process all data using xpath, regex, and string parsing. We then store all drink-related information in a pandas dataframe and using our filtering algorithm we find the entries that match the keywords provided. The time data is stored in a separate dataframe by weekday and can be visualized in 30 minute time intervals.

Process/Methodology

Since our objective primarily concerns the choices of coffee a Davis student may find most accessible, we chose to limit our web scraping to only five local shops downtown that they may be interested in. Although we will elaborate on this further, we also chose to gather this information from different websites, including Uber Eats, Yelp, and the store's official online menu. This is because of the unique names that each store may have for their drink selection; for instance, Mishka's Cafe calls their vanilla latte a "Mishka's Latte." To bypass this potential information discrepancy, we needed to gather information from sites that would also include item descriptions. With this, if we wanted to find which cafes sold a vanilla latte, then we would search both the item description and the name to find drinks that matched the target strings' keywords. Since we did not want to do two searches through the entire dataframe for those two columns, we chose to combine the name into the description, instead. Then, to

also avoid case sensitivity, we used lower() to ensure the description column only contained lowercase characters. Thus, identifying whether the target keywords were contained in the description became a much simpler, structured task.

1. Menu Extraction

Mishka's Cafe

Attempt 1:

To start, we attempted to retrieve the Mishka's menu from the Doordash website. However, the request returned a 403 error when we ran a 'raise_for_status,' so we knew that there was an error in the request function. To mitigate this issue, I attempted to add additional parameters and a header with cookie information, but this also failed to gather any information. Upon further research into their documented API, the Doordash website had limited public access to its API, which was primarily reserved for their business partners that use the platform.

Attempt 2:

Since I could not access the HTML of the Doordash page, we chose to divert our attention towards Selenium and hoped to use a web driver that could mimic the website's page. Once again, however, we encountered another issue. The web driver had a pop-up, warning us of an unsecure connection. At first, we attempted to solve this problem by adding arguments to the Chrome web driver, such as 'ignore-certificate-errors' and '--ignore-ssl-errors=yes'. Yet, the warning persisted, so we unfortunately decided to retrieve Mishka's menu from another website that had drink descriptions: Uber Eats.

Attempt 3:

Now, with this new URL, we used a 'request.get()' function that successfully retrieved the site's HTML. Although this menu had conflicting details compared to other websites, we ultimately settled with potentially having fewer menu items so that we could also have descriptions of said drinks. After close inspection of the website using the developer tools, we noticed the menu had an accessible structure under

the main body tag with the script type, 'application/ld+json'. Because of this, using the loads() function from the JSON package allowed us to turn the information gathered into an indexable dictionary. From there, it was a matter of indexing drink names, descriptions, and prices while removing items under the food menu sections since we only wanted the drink selections.

Peet's Coffee

Attempt 1:

Fortunately, the 'request.get()' function worked immediately, and we were able to access the extensive Yelp page for Peet's. This time, the menu was entirely complete, so we turned the retrieved page's data into HTML that we would be able to parse. From there, we used an xpath that would be able to access the menu items, the price, and the description separately. Initially, I had included the larger tags where the id="super-container" or the class="menu-sections", but I noticed that I could largely simplify my xpath by finding a more specific subtag that would still lead to each menu item. However, this is when I mistakenly chose to parse directly from the class that contained all of the desired data. It was only when I further processed the information that I realized I would not be able to remove non-drink items, since the menu section titles and the menu items were all contained in sibling tags, rather than ancestor tags. Afterwards, I chose to change my strategy to looking directly at the immediate siblings since it was another tag that held the menu section names. Once I used a condition that would ignore sections containing text like 'food', 'snack', 'brew', 'merchandise', 'loose leaf', or 'to-go', I then used '/following-sibling::div[1]' to look at the immediate section that contains the actual drinks that Peet's offers. Once I amended my overall xpath, I used more xpaths using './' that would continue parsing for each part of the information separately. Although I ran into additional problems, such as items containing a reference link rather than text or the returned list of strings containing excess white space and '\n', I was able to address these issues accordingly. For instance, my regex pattern substituted '[\n][\s]+' at the beginning of the string with '' to remove unnecessary characters and clean the data. Thus, the menu

from Peet's was successfully extracted with all 127 drink options, and I even found a new drink that I had not known they offered prior!

Volt Coffee

Attempt 1: API endpoint

We started by trying to retrieve the data directly from the main endpoint that returns the html code for the website. Upon attempting to query this endpoint, a 403 client error occurred. Even after adding additional headers such as cookies and user-agent, the error persisted, so we pivoted to web scraping the website.

Attempt 2: Selenium

We utilized selenium to capture a snapshot of the website's dynamic state and after increasing the timeout all drinks appeared in the html response. From here, we used xpath to find all drink names, descriptions, and prices. To find what path to input into xpath, we used dev tools to inspect specific elements. One problem we ran into was how our initial path did not capture the out-of-stock items. After inspecting the differences between how in stock and out of stock items were presented in the html using developer tools, we found out the class for the div element changed from "price" to "price outOfStock", so we added an additional or condition to capture this distinction. To clean up the data, we used regex, string parsing, and filled non-description data with the name of the drink.

Pachamama Coffee

Attempt 1: API endpoint

Upon inspecting all the requests in dev tools, we found an endpoint that returns all drink related information in a json format. We then queried this endpoint and noticed the data contained additional products such as coffee beans and variations in prices such as lowest prices and max price for each drink. We filtered out non-drink items and decided to include only the lowest price for each drink in our final dataset. These drinks also did not have descriptions, so we filled the column with the drink names.

Temple Coffee Roasters

Attempt 1: Yelp

Because the website did not contain a menu, we started by trying to retrieve the data from Yelp's menu. After querying the main website's endpoint, we were successful in retrieving the html data. Using xpath and paths found from inspecting the html in the developer tools, we were able to find the drink names and drink prices and fill the drink descriptions with the drink names. Since the menu also contained food items, we needed to modify the xpath paths to look specifically for data contained within sections of drinks.

2. Location & Hours Extraction

To extract the specific location coordinates for each coffee shop, we leverage the Nominatim API search endpoint and specifically include "Davis" in the query's input parameter to ensure we found the correct location. When trying to graph the data using these latitude and longitude values, we ran into a type error. After reading plotly's documentation for scatter_map(), we realized we needed to convert the values from strings to floats.

We obtained the hours of operation by querying the yelp website for each coffee shop. Upon noticing that the path in the HTML data was the same for all coffee shops, we made a function called append_times() to reduce repeating code and add to our code's readability. Since the data was in the format of a time interval, we needed to get all 30 minute sub-intervals in order to plot the data. We first used regex to extract the start and end times. Afterwards, we used the datetime package to convert these times to military time. This allowed us to easily calculate the sub-intervals and filter the graph using these increasing times. We collected time and location data from all coffee shops in a separate dataframe for each week day. Using plotly's scatter_map() function, we were able to create interactive maps where coffee shop locations would appear only at times the coffee shop is open. The user can move the time slider to see which shops are open at each 30 minute interval. By hovering over a point, the user can see the name of the coffee shop.

Results

After standardizing and calculating time intervals for hours opened, we were able to display in interactive maps which coffee shops were open based on day of the week and at a specific 30 minute interval timestamp. As shown in Figure 1, only two coffee shops are open at 6:00 AM compared to Figure 2 where all five coffee shops are open at 8:30 AM. This makes it easy for the user to check at what specific times they can purchase their target drink or see an alternative coffee shop open at their preferred time.

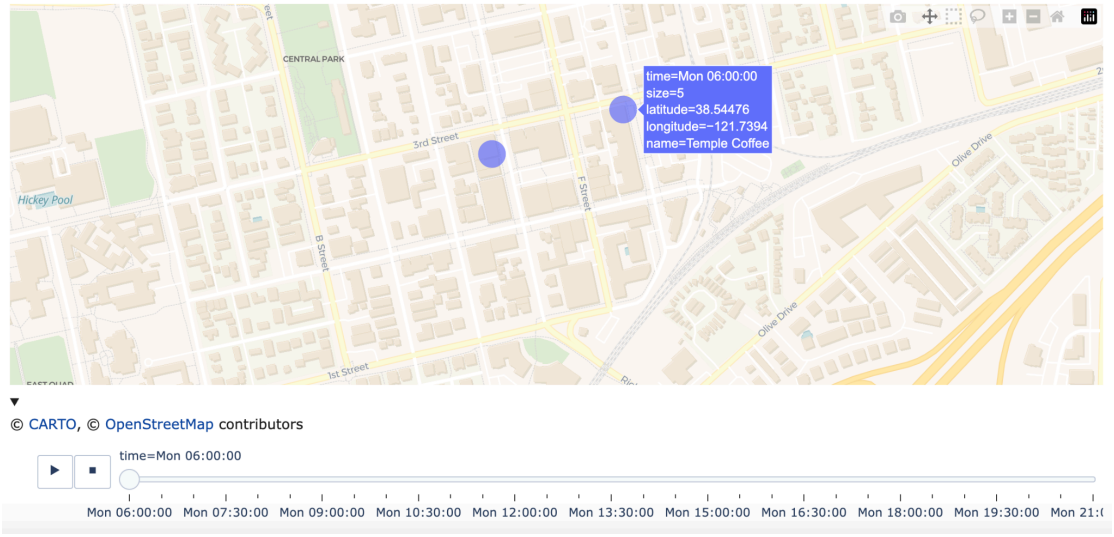

Figure 1: Map of coffee shops open on Monday at 6:00 AM

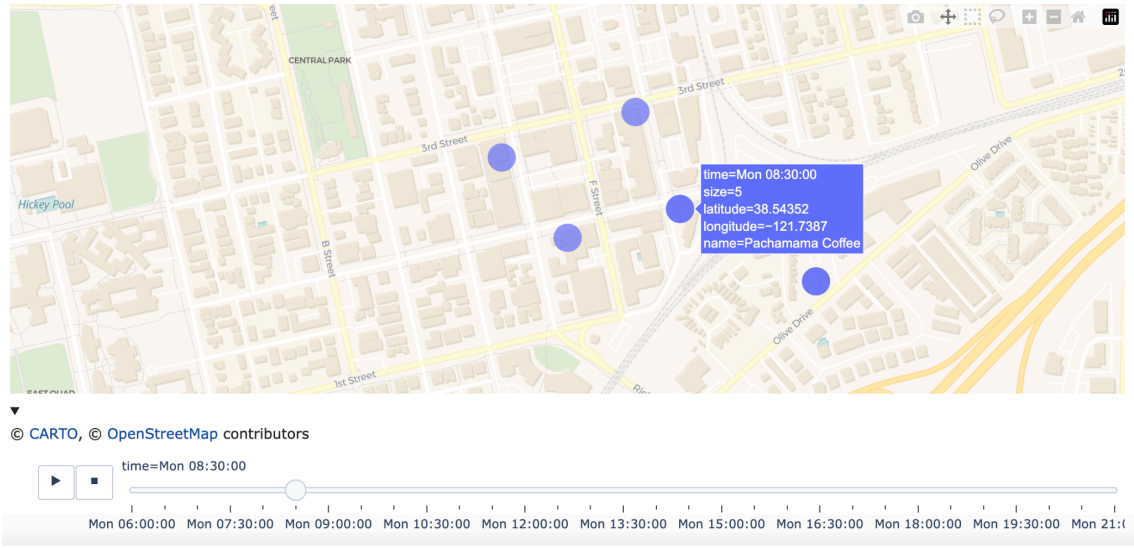

Figure 2: Map of coffee shops open on Monday at 8:30 AM

To find which stores carry a target drink, our find_drink() algorithm will detect all drinks that match the keywords inputted and return the matches in increasing order by price. The algorithm is agnostic of the ordering of keywords and will find matches from the name and description as seen by "Mishka's Latte" returned from keywords "latte vanilla" in Figure 3. The keyword can also just contain specific ingredients without drink type such as using the keyword 'honey' as seen in Figure 4.

```
find_drink('latte vanilla', df)
```

	item	description	price	store
9	mishka's latte	our signature drink....a vanilla flavored latt...	5.50	Mishka's
23	Hot Vanilla Latte	sweet vanilla syrup added to our freshly extra...	5.75	Volts
34	Iced Vanilla Latte	rich and creamy house espresso mixed with swee...	5.75	Volts
24	Hot SF Vanilla Latte	sf sweet vanilla syrup added to our freshly ex...	5.75	Volts
35	Iced SF Vanilla Latte	rich and creamy house espresso mixed with swee...	5.75	Volts
47	London Fog Latte	earl grey tea sweetened with vanilla syrup and...	6.00	Volts
50	Iced Volt Tea Latte	chamomile & earl grey tea sweetened with laven...	6.00	Volts
46	Volt Tea Latte	chamomile & earl grey tea sweetened with laven...	6.00	Volts
38	iced vanilla latte	madagascar vanilla takes a refreshing turn wit...	6.25	Peet's
23	sugar-free vanilla latte	satisfying without sacrifice. same ingredients...	6.25	Peet's
39	iced sugar-free vanilla latte	sugar-free vanilla syrup delivers the same bal...	6.25	Peet's
21	vanilla latte with protein	the sweetness of vanilla syrup, rich espresso ...	7.40	Peet's
2	vanilla latte with protein	the sweetness of vanilla syrup, rich espresso ...	7.40	Peet's
6	iced vanilla latte with protein	the sweetness of vanilla syrup, rich espresso ...	7.40	Peet's
35	iced vanilla latte with protein	the sweetness of vanilla syrup, rich espresso ...	7.40	Peet's
7	Vanilla Bean Latte	vanilla bean latte	7.50	Pachamama

Figure 3: Drinks found matching the keywords 'latte vanilla'

```
find_drink('honey', df)
```

	item	description	price	store
11	morning soul	caffeinated tea blend, steamed soy milk, a hin...	5.25	Mishka's
10	soul	herbal tea mix, steamed soy milk, hint of hone...	5.25	Mishka's
41	iced golden latte	our golden caff latte pairs espresso fort with...	6.25	Peet's
5	sparkling golden immunity	fan favorite turmeric-ginger-honey flavor meet...	6.30	Peet's
62	sparkling golden immunity	fan favorite turmeric-ginger-honey flavor meet...	6.30	Peet's
22	golden latte with protein	espresso forte, brightened with turmeric-ginge...	7.40	Peet's
3	golden latte with protein	espresso forte, brightened with turmeric-ginge...	7.40	Peet's
1	iced golden latte with protein	espresso forte, brightened with turmeric-ginge...	7.40	Peet's
36	iced golden latte with protein	espresso forte, brightened with turmeric-ginge...	7.40	Peet's
12	Honeybear Latte	honeybear latte	7.50	Pachamama
15	Honeybear Cold Brew	honeybear cold brew	7.50	Pachamama

Figure 4: Drinks found matching the keyword 'honey'

Conclusion

Returning to our objective questions, we successfully utilized various web tools to create a replicable algorithm that would be able to answer where you can find your drink of choice by keywords, when and where is your drink of choice available, and where is your drink of choice the cheapest. As shown in the 'Results' section of our report, we set our target drinks to be vanilla lattes and drinks containing honey. For a vanilla latte, you can choose between the cheapest being Mishka's Cafe, Volt being the second cheapest, Peet's, or Pachamama being the most expensive. You may also be particular about time, such as needing the aforementioned vanilla latte at 8:30 PM so you can only go to Volt. Or, you may be at Young Hall and you need a drink at 6AM on a Friday, so you choose Peet's over Temple, since you do not want to make the trek a few blocks further than you need to. All in all, our questions are all answered and displayed in a clear, succinct manner, which is an optimal solution for college students who need an immediate answer with one search. Our objective is complete and, although the answer has

always been online through each website, a few precious minutes have been shaved off of the process for Davis students.

Limitations

We faced some key challenges with our data, limiting us to analyzing the data from only five coffee shops. When searching through various websites, we noticed that menus were incomplete such as lacking the full list of items or not containing descriptions of a drink's contents. Further, the data was represented in various formats requiring individualized processing for each coffee shop's menu. This led us to tailor our main questions of focus to match the data that was available -- drink names, prices, store hours and locations. This also challenged us to be creative in how we processed and visualized the data, resulting in us being able to foster connections both in our data and the real world -- one target drink to a coffee shop at a time. It is also important to note that Yelp is a website that contains user input, so information can become outdated; but, having potential false information would exist regardless of our web scraping tool if a student were to search for the coffee shops' information online themselves.