Mathematically and Personally Optimal Itinerary Builder Application

ESE 498/499 Capstone Design Project

David Armstrong
B.S. Candidate, Systems Engineering
david.armstrong@wustl.edu

Kaan Dincer
B.S. Candidate, Systems Engineering
kaandincer@wustl.edu

Peter Jakiela
B.S. Candidate, Electrical Engineering
peterjakiela@wustl.edu

Advisor:

Ioannis (Yiannis) Kantaros
Assistant Professor, Electrical and Systems Engineering
Washington University in St. Louis
ioannisk@wustl.edu

Submitted to Professor Wang and the Department of Electrical and Systems Engineering

December 18, 2022

1 Abstract

Every day, billions of people try to make plans. Whether for future travel, or what to do locally that weekend, the same question comes up time and time – what should I do that day? When searching online for restaurants and activities, one is flooded with individual options that must be sifted through. And although some services, like Google, do have personalized recommendations, those recommendations are still overwhelming and don't take into account the day being planned holistically as a complete itinerary. This is where our product, PlanIt, comes in. PlanIt takes into account continuous (price, cost, average rating, number of reviews) and discrete (categorical) preferences to transform the internet's many suggestions into a personally optimal itinerary.

We accomplish this by first collecting data from Yelp on the top fifty activities, restaurants, and nightlife locations in the city of your choice, filtering out the categories the user is not interested in and using "preference curves" to obtain a unique profile of the user, creating a personalized score for every possible pairing of destinations, then running Dijkstra's algorithm on those scores to find your best possible route through the city. This results in a concise, mathematically optimal itinerary that users can trust will give them a satisfactory experience.

2 Introduction

This project tackles a time-consuming problem that many people face – planning their adventures around a city. Google and Yelp are fantastic databases with filters and reviews to inform your decision, but they don't look at the whole picture. Only an itinerary recommendation can give the user the best path through the city to optimize their enjoyment and their wallet. Inspirock creates an itinerary recommendation but has limited user input [1]. Our "preference curve" input method is completely novel and takes into account the intricacies of one's willingness to partake in certain activities as various parameters change.

The future of how you decide to spend your time is approaching. Already, people are more connected to the various events happening across their city than ever before. Working towards a perfect recommendation system is working to remove the inefficiencies of the clunky system of today. A world where we don't have to think about what to do this weekend is more of an inevitability than a solution to a problem.

While we cannot create the eventual all-powerful recommendation system (one that plans you and your friends' lives for you better than you ever could yourself) yet, we did make the following feasible steps towards this end in the scope of this semester: Build a (1) internet-connected (2) activity recommendation system that is (3) uniquely and highly personalized and (4) user-friendly. This is what our project accomplished – being capable of producing custom itineraries in the city of choice.

3 Methods

3.1 Overview and Objective

Our product aims to create a personalized, optimal set of events for users over the course of the day. To make this lofty goal attainable within the constraints of a senior capstone design, we make some fundamental assumptions about what constitutes a day of events. First, events are broken into three categories: activities, restaurants, and nightlife. Activities are events primarily designed around daytime recreation. Restaurants provide meals for users. Nightlife events are recreation events that are best enjoyed during the evening or later. With these three classes of events defined, we assume the following daily schedule for all users.

$$Morning\ Activity \rightarrow Lunch\ Restaurant \rightarrow Afternoon\ Activity \rightarrow Dinner\ Restaurant \rightarrow Nightlife$$

With this simple framework established, we are able to concretely categorize events as candidates for filling certain scheduled roles. Further, events that are candidates to fill the same roles can now be fairly compared. Throughout the methods section, we describe in detail the process of using a GUI to collect user data, using API calls to collect event data, scoring individual events, and optimizing combinations of ideal

event recommendations. Figure 1 provides an overview of the flow of information through our application program during the recommendation process.

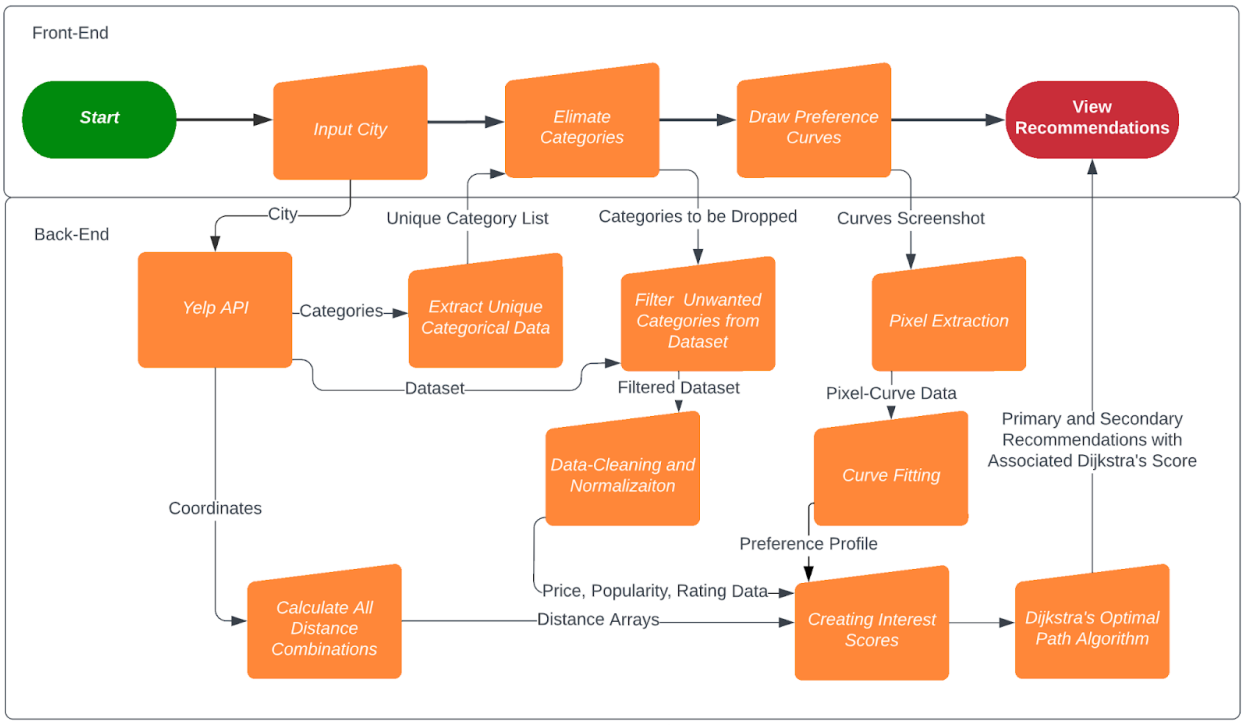


Figure 1: Recommendation Process Information Flow.

3.2 Start/UI

To build our Graphical User Interface (GUI), we used Python's Tkinter package. This allowed us to use the code we wrote for user input, optimization of recommendations, and Dijkstra's method with ease since the language we chose for coding those steps was Python. In addition, Python allowed us to create an appealing and easy-to-use GUI.

Using Tkinter, we created a canvas and added object such as labels, and buttons. By using and giving buttons certain commands we deleted existing objects on the screen and added new ones so that the program could advance to the next stages without opening a new window.

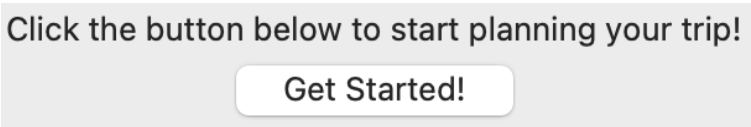


Figure 2: Start button.

3.3 Input City

For this step, we give the user a dropdown list of popular cities to choose to run the program on. Our platform technically works with any city input as the dataset is dynamically gathered from Yelp, but this input method (dropdown list) ensures that no erroneous city name is entered (as opposed to free response text).

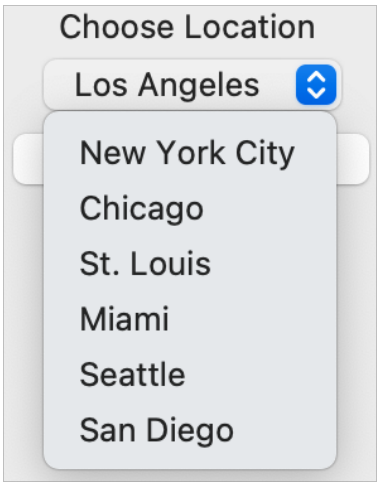


Figure 3: City selection GUI.

3.4 Eliminate Categories

For this step, we use the lists of unique categories to allow the user to scratch out any types of activities that do not interest them. We populate three side-by-side "listboxes" with these unique categories, representing one column for each activities, food, and nightlife to allow the user to select multiple unwanted categories and then eliminate them from the dataset.

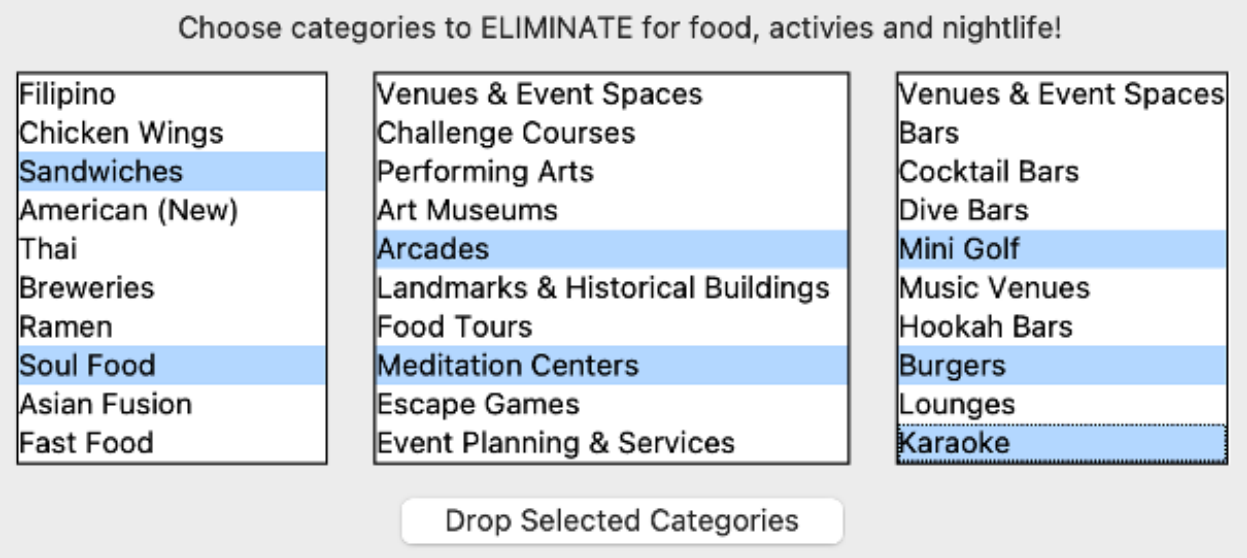


Figure 4: Event Elimination GUI.

3.5 Draw Preference Curves

For this step, we use the Turtle python package to implement a drawing mechanism [2]. This package allows us to define x and y-axes and initialize the drawing turtle's position halfway up the y-axis to allow for purely positive or negative sloping curves to be drawn. These curves are individual elasticity curves in essence, describing one's willingness to purchase/participate given a changing variable.

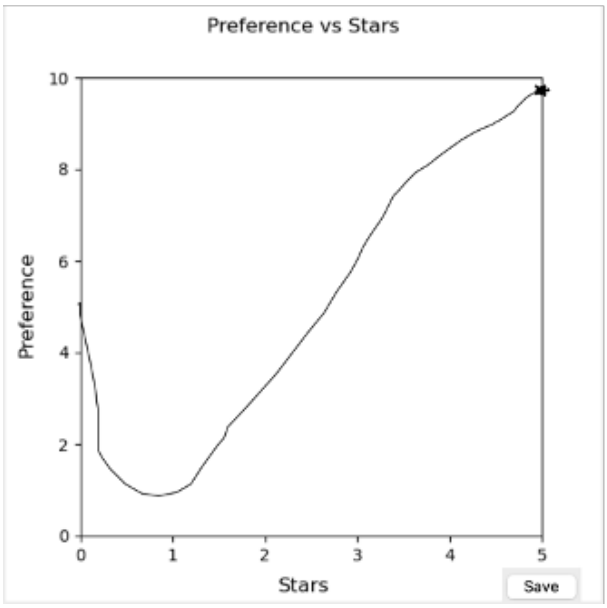


Figure 5: User Curve Drawing GUI.

3.6 Yelp API (Data Collected and Procedure for Acquiring)

For this step, we created a Yelp Fusion API account that provided us with an API key that unlocks their entire database [3]. We make three API calls (food, activities, and nightlife) at 50 locations per call in the user-specified location. This dataset contains everything we need to compare our options (coordinates, review count, stars, and price).

review_count	categories	rating	coordinates	transactions	price
84	[{'alias': 'galleries', 'title': 'Art Gallerie...	4.5	{'latitude': 38.6520275026572, 'longitude': -9...	[]	$$
23	[{'alias': 'parks', 'title': 'Parks'}]	5.0	{'latitude': 38.6505214880486, 'longitude': -9...	[]	NaN
16	[{'alias': 'parks', 'title': 'Parks'}, {'alias...	5.0	{'latitude': 38.6368020725721, 'longitude': -9...	[]	NaN

Figure 6: Relevant return parameters of the Yelp API call.

The search query parameters for the Yelp API include the following: `term`, `location`, `latitude`, `longitude`, `radius`, `categories`, `locale`, `limit`, `offset`, `sort-by`, `price`, `open-now`, `open-at`, and `attributes`. And the return parameters for the Yelp API are the following: `id`, `alias`, `name`, `image-url`, `is-closed`, `url`, `review-count`, `categories`, `rating`, `coordinates`, `transactions`, `price`, `location`, `phone`, `display-phone`, and `distance` We utilize the `review-count`, `categories`, `rating`, `coordinates`, and `price` parameters for our calculations and the `review-count`, `categories`, `rating`, `display-phone`, `name`, and `price` when presenting the results. Using this dynamic data collection technique allows us to run our program on any city in the world.


```
Enter your destination city: St. Louis
{
   "businesses": [
      {
         "id": "YYpYssZ8nToDVwT8f8Hd7Q",
         "alias": "platypus-st-louis",
         "name": "Platypus",
         "image_url": "https://s3-media4.fl.yelpcdn.com/bphoto/8sjQ9nhGB4ix4RChjksgow/o.jpg",
         "is_closed": false,
         "url": "https://www.yelp.com/biz/platypus-st-louis?adjust_creative=0T3oYYb-7_PGoE7letJXOg&utm_campaign=yelp_api_v3&utm_medium=api
_v3_business_search&utm_source=0T3oYYb-7_PGoE7letJXOg",
         "review_count": 15,
         "categories": [
            {
               "alias": "cocktailbars",
               "title": "Cocktail Bars"
            }
         ],
         "rating": 4.5,
         "coordinates": {
            "latitude": 38.6265043,
            "longitude": -90.2617607
         },
         "transactions": [
            "delivery"
         ],
         "location": {
            "address1": "4501 Manchester Ave",
            "address2": "",
            "address3": null,
            "city": "St. Louis",
            "zip_code": "63110",
            "country": "US",
            "state": "MO",
            "display_address": [
               "4501 Manchester Ave",
               "St. Louis, MO 63110"
            ]
         },
         "phone": "+13143592293",
         "display_phone": "(314) 359-2293",
         "distance": 5778.721229781712
      },
      {
         "id": "8Ux_iceWge-M6Dq3eCvgfg",
         "alias": "up-down-stl-saint-louis",
         "name": "Up-Down STL",
         "image_url": "https://s3-media3.fl.yelpcdn.com/bphoto/RIxpaTWboWTGu0U7EHxOXw/o.jpg",
         "is_closed": false,
         "url": "https://www.yelp.com/biz/up-down-stl-saint-louis?adjust_creative=0T3oYYb-7_PGoE7letJXOg&utm_campaign=yelp_api_v3&utm_medi
```


Figure 7: Original JSON format that the data returns as before conversion to a pandas dataframe.

See the below image for a full example of a Yelp call's return data:

	id	alias	name	image_url	is_closed	url	review_co	categories	rating	coordinate	transactior	price	location	phone	display_pl	distance
0	4vRKXeFjd	third-degree	Third Degre	https://s3-m	FALSE	https://www	84	[{'alias': 'ga	4.5	{'latitude': 3	[]	$$	{'address1':	1.31E+10	(314) 367-4	5882.392
1	Tf1shkER_	shaw-park-s	Shaw Park	https://s3-m	FALSE	https://www	23	[{'alias': 'pa	5	{'latitude': 3	[]	NaN	{'address1':	'Brentwood	Blvd At For	3142.975
2	7p79ZsAuV	worlds-fair-p	World's Fair	https://s3-m	FALSE	https://www	16	[{'alias': 'pa	5	{'latitude': 3	[]	NaN	{'address1':	1.31E+10	(314) 289-5	3921.457
3	JJBLzh26iiI	steinberg-s	Steinberg S	https://s3-m	FALSE	https://www	66	[{'alias': 'ska	3.5	{'latitude': 3	[]	NaN	{'address1':	1.31E+10	(314) 367-7	5385.479
4	wE-U4wEFI	grants-farm	Grant's Far	https://s3-m	FALSE	https://www	326	[{'alias': 'zo	4.5	{'latitude': 3	[]	NaN	{'address1':	1.31E+10	(314) 843-1	8780.646
5	ipybddMCb	taste-of-bla	Taste of Bl	https://s3-m	FALSE	https://www	1	[{'alias': 'fes	5	{'latitude': 3	[]	NaN	{'address1':	1.31E+10	(314) 564-2	9884.805
6	Ddu6G10Z	the-magic-h	The Magic	https://s3-m	FALSE	https://www	207	[{'alias': 'chi	4.5	{'latitude': 3	[]	NaN	{'address1':	1.31E+10	(314) 822-8	8763.195
7	aUZys3AlG	laumeier-sc	Laumeier S	https://s3-m	FALSE	https://www	144	[{'alias': 'pa	4.5	{'latitude': 3	[]	NaN	{'address1':	1.31E+10	(314) 615-5	11033.18
8	n5jsPRd2fII	sharpshoot	SharpShoo	https://s3-m	FALSE	https://www	62	[{'alias': 'gu	4.5	{'latitude': 3	[]	NaN	{'address1':	1.31E+10	(314) 353-2	7472.952
9	ANuiV4ZH0	lift-stl-saint-	Lift-STL	https://s3-m	FALSE	https://www	3	[{'alias': 'gy	5	{'latitude': 3	[]	NaN	{'address1':	1.31E+10	(314) 296-3	848.4573
10	LmKpHMX9	missouri-bo	Missouri Bo	https://s3-m	FALSE	https://www	506	[{'alias': 'ga	4.5	{'latitude': 3	[]	NaN	{'address1':	1.31E+10	(314) 577-5	6160.069
11	Wdt_Y8w1	cache-bayc	Cache Bay	https://s3-m	FALSE	https://www	9	[{'alias': 'bo	5	{'latitude': 3	[]	NaN	{'address1':	1.62E+10	(618) 201-4	184133.3
12	5U5W3k2JI	italia-americ	Italia Ameri	https://s3-m	FALSE	https://www	1	[{'alias': 'bo	5	{'latitude': 3	[]	NaN	{'address1':	1.31E+10	(314) 773-5	4926.723
13	60xFUHDm	saint-louis-s	Saint Louis	https://s3-m	FALSE	https://www	257	[{'alias': 'art	4.5	{'latitude': 3	[]	NaN	{'address1':	1.31E+10	(314) 721-0	3377.149
14	RBus8M0B	the-loop-st-	The Loop	https://s3-m	FALSE	https://www	29	[{'alias': 'sh	4	{'latitude': 3	[]	$$	{'address1':	1.31E+10	(314) 727-8	4445.593
15	H5z4Ndy5s	victory-race	Victory Rac	https://s3-m	FALSE	https://www	8	[{'alias': 'go	4	{'latitude': 3	[]	NaN	{'address1':	1.31E+10	(314) 925-7	7533.87
16	yl6PUs7nE	saint-louis-s	Saint Louis	https://s3-m	FALSE	https://www	228	[{'alias': 'pla	4	{'latitude': 3	[]	NaN	{'address1':	1.31E+10	(314) 289-4	5045.479
17	d7IG-3wtYI	historic-dan	Historic Dar	https://s3-m	FALSE	https://www	13	[{'alias': 'his	4.5	{'latitude': 3	[]	NaN	{'address1':	1.64E+10	(636) 798-2	45689.24
18	qMvPZKgY	bonne-terre	Bonne Terr	https://s3-m	FALSE	https://www	55	[{'alias': 'div	4	{'latitude': 3	[]	NaN	{'address1':	1.89E+10	(888) 843-3	80420.26
19	L34O6zjHn	st-louis-win	St. Louis W	https://s3-m	FALSE	https://www	3	[{'alias': 'fes	5	{'latitude': 3	[]	NaN	{'address1':	'Lagoon Dr',	'address2'	3728.42
20	rVKBp4D2r	hiit314-spo	HIIT314 Sp	https://s3-m	FALSE	https://www	1	[{'alias': 'int	5	{'latitude': 3	[]	NaN	{'address1':	1.31E+10	(314) 503-3	829.3885
21	XmpbEmS7	the-boatho	The Boatho	https://s3-m	FALSE	https://www	428	[{'alias': 'ne	3	{'latitude': 3	[delivery]	$$	{'address1':	1.31E+10	(314) 366-1	4158.124
22	9Kxm-HC9j	escape-fror	Escape Fro	https://s3-m	FALSE	https://www	14	[{'alias': 'es	5	{'latitude': 3	[]	NaN	{'address1':	1.31E+10	(314) 202-8	1441.138
23	SqotIdKH_	world-chess	World Ches	https://s3-m	FALSE	https://www	48	[{'alias': 'ga	4.5	{'latitude': 3	[]	$	{'address1':	1.31E+10	(314) 367-9	6239.193
24	mNdWqhw0	st-louis-ska	St. Louis S	https://s3-m	FALSE	https://www	8	[{'alias': 'ska	2.5	{'latitude': 3	[]	NaN	{'address1':	1.31E+10	(314) 631-3	10866.95
25	v7Gwhn4w	pure-hot-yo	Pure Hot Y	https://s3-m	FALSE	https://www	57	[{'alias': 'yo	3.5	{'latitude': 3	[]	NaN	{'address1':	1.31E+10	(314) 644-2	1458.564
26	Mb0NrOHG	eat-saint-lo	EAT Saint I	https://s3-m	FALSE	https://www	19	[{'alias': 'foo	5	{'latitude': 3	[]	NaN	{'address1':	1.31E+10	(314) 399-9	5071.005
27	OrgT05tCfL	tivoli-theatr	Tivoli Theat	https://s3-m	FALSE	https://www	89	[{'alias': 'mo	4.5	{'latitude': 3	[]	NaN	{'address1':	1.31E+10	(314) 727-7	4109.638
28	L5YPaW84	st-louis-veg	St. Louis V	https://s3-m	FALSE	https://www	1	[{'alias': 'fes	4	{'latitude': 3	[]	NaN	{'address1':	'1904 Concourse Dr',	'a	3952.688
29	b2PiLI_-Dm	man-on-a-h	Man On A I	https://s3-m	FALSE	https://www	3	[{'alias': 'lan	4.5	{'latitude': 3	[]	NaN	{'address1':	'Hanley And Wydown',		2177.725
30	X8hwEPRo	saint-louis-z	Saint Louis	https://s3-m	FALSE	https://www	963	[{'alias': 'zo	4.5	{'latitude': 3	[]	NaN	{'address1':	1.31E+10	(314) 781-0	3393.252
31	pVS9ekDY	kingsland-p	Kingsland F	https://s3-m	FALSE	https://www	1	[{'alias': 'pla	4	{'latitude': 3	[]	NaN	{'address1':	'6651 Chamberlain Ave		4828.474
32	flo_q4nCM	series-six-c	Series Six (	https://s3-m	FALSE	https://www	1	[{'alias': 'loc	5	{'latitude': 3	[]	NaN	{'address1':	'26 The Blvd Saint Lou		1857.329
33	5FZ8No8p8	hallowen-at	Hallowen a	https://s3-m	FALSE	https://www	1	[{'alias': 'ha	5	{'latitude': 3	[]	NaN	{'address1':	'10501 Gravois Rd', 'ac		8779.782
34	JPNjMRdFb	pedego-ele	Pedego Ele	https://s3-m	FALSE	https://www	1	[{'alias': 'bik	5	{'latitude': 3	[]	NaN	{'address1':	1.62E+10	(618) 791-1	38414.38
35	yd6drx8OzI	gateway-he	Gateway H	https://s3-m	FALSE	https://www	6	[{'alias': 'ae	4.5	{'latitude': 3	[]	NaN	{'address1':	1.31E+10	(314) 496-4	12628.61
36	_PRReGR0	tower-grove	Tower Grov	https://s3-m	FALSE	https://www	76	[{'alias': 'fai	4.5	{'latitude': 3	[]	$$	{'address1':	'4256 Magnolia Ave', 'a		6326.935
37	2c6z6zw1re	seascape-s	Seascape S	https://s3-m	FALSE	https://www	11	[{'alias': 'pe	4.5	{'latitude': 3	[]	NaN	{'address1':	1.31E+10	(314) 843-3	10674.8
38	ZJmRkyB6f	metropolita	Metropolitan Zoological		FALSE	https://www	1	[{'alias': 'zo	5	{'latitude': 3	[]	NaN	{'address1':	1.31E+10	(314) 862-4	2968.262
39	hXL3a3Dw	strength-wc	Strength W	https://s3-m	FALSE	https://www	4	[{'alias': 'gy	5	{'latitude': 3	[]	NaN	{'address1':	1.5E+10	(502) 905-2	1933.641
40	DDybAdOC	tower-tee-g	Tower Tee	https://s3-m	FALSE	https://www	2	[{'alias': 'go	4.5	{'latitude': 3	[]	NaN	{'address1':	1.31E+10	(314) 833-3	6259.489
41	M2BQVingI	wise-warrio	Wise Warric	https://s3-m	FALSE	https://www	2	[{'alias': 'kic	5	{'latitude': 3	[]	NaN	{'address1':	1.31E+10	(314) 960-7	1441.138
42	1vZK9-APS	loving-rose-	Loving Ros	https://s3-m	FALSE	https://www	3	[{'alias': 'rei	4.5	{'latitude': 3	[]	NaN	{'address1':	1.62E+10	(618) 236-5	1974.188
43	OiQuTy4dV	sharpshoot	SharpShoo	https://s3-m	FALSE	https://www	9	[{'alias': 'gu	4	{'latitude': 3	[]	NaN	{'address1':	1.31E+10	(314) 353-4	7463.55
44	vjXbNXmg0	forest-park-	Forest Park	https://s3-m	FALSE	https://www	1	[{'alias': 'pa	5	{'latitude': 3	[]	NaN	{'address1':	1.31E+10	(314) 367-7	4456.066
45	oQY950YC	oakland-ho	Oakland Ho	https://s3-m	FALSE	https://www	1	[{'alias': 'lan	5	{'latitude': 3	[]	NaN	{'address1':	1.31E+10	(314) 352-5	6382.087
46	T2bot2f9Zu	honeycomb	Honeycomb	https://s3-m	FALSE	https://www	4	[{'alias': 'toy	5	{'latitude': 3	[]	NaN	{'address1':	1.31E+10	(314) 202-8	6288.259
47	34mMhG2V	the-great-fc	The Great I	https://s3-m	FALSE	https://www	25	[{'alias': 'fes	4	{'latitude': 3	[]	NaN	{'address1':	'Central Field', 'address		5025.512
48	XgoxgzJh6	brewery-tou	Brewery To	https://s3-m	FALSE	https://www	27	[{'alias': 'be	4	{'latitude': 3	[]	NaN	{'address1':	1.31E+10	(314) 399-0	8620.944
49	ePLnAURT	max-effort-s	Max Effort	https://s3-m	FALSE	https://www	1	[{'alias': 'he	5	{'latitude': 3	[]	NaN	{'address1':	1.31E+10	(314) 537-1	1969.244

Figure 8: St. Louis Activities Data.

3.7 Extract Unique Categorical Data

In this section, we convert the JSON information into a pandas dataframe, isolate the category column, pick out the first category descriptor for each event (the logic here is that the first label Yelp gives it is most likely the most aptly descriptive – using all of them would create even more categories than the high number we already work with), narrowing in on the 'title' key within the categories column as opposed to 'alias' for clean formatting, and using `pd.unique()` to eliminate duplicate category labels.

3.8 Filter Unwanted Categories from Dataset

To filter the unwanted categories out of the dataset, we use the lists of unwanted categories from the listboxes the user fills out. We use np.where() to find the indices of where the unwanted categories are in the list of unique categories, then .drop() to remove the activities at those indices.

```
Dropping:  ['Chinese', 'Japanese']  food shape:  (44, 16)
Dropping:  ['Arcades', 'Yoga']  act shape:  (46, 16)
Dropping:  ['Bars', 'Venues & Event Spaces']  nl shape:  (39, 16)
```

Figure 9: Datasets of 50 each reducing down to 44, 46, and 39 each after category elimination.

3.9 Calculate All Distance Combinations

Because we use distance as one of our decision variables, it is important to know the distance between the events in our dataset. If we assume a specific current location and are choosing the next event to attend after finishing at this current location, we need to know not only information about all the prospective events, but also the distance between our current location and each prospective event. Therefore, we calculate the distances from activities to restaurants and restaurants to nightlife before running Dijkstra's algorithm.

We use our `distCalc.py` file to read in the filtered dataset, extract the coordinates, then calculate the distances between every possible restuarant/activity pairing. We use the Haversine formula which uses coordinates to approximate distances within a small area relative to the surface of the Earth.

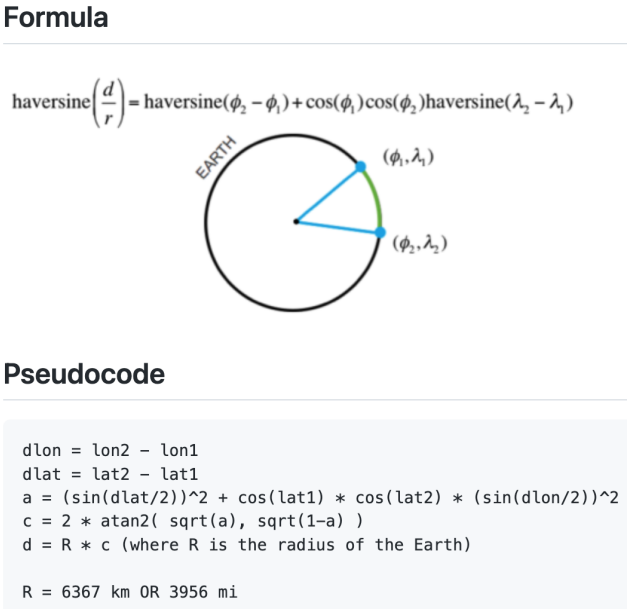


Figure 10: Distance calculation strategy [4].

3.10 Pixel Extraction

After capturing screenshots of the user-drawn curves, we extract the x and y coordinates of the drawn curve to be passed into the optimization algorithm for recommendations. We do this by checking each pixel in the image and saving its x-y coordinates if the RGB values all equal 0 since the RGB values of black are all 0, and the rest of the canvas is white. We use the following code to do this step.

```
    for x in range(im.size[0]):
    for y in range(im.size[1]):
        r, g, b, = rgb.getpixel((x, y))
        y = 771-y
        if counter%diveded==0:
            if r == 0 and g == 0 and b == 0:
                if len(x_values) ==0:
                    x_values.append(x)
                    y_values.append(y)
                elif len(x_values)>=1:
                    if x_values[-1] != x: # Ensures we use 1 y-value per x
                        x_values.append(x)
                        y_values.append(y)

```

Figure 11: Pixel extraction code snippet

3.11 Curve Fitting

While the curves given be the users contain relevant information about the user's preferences and willingness to attend a range of events, this information is not computer-interpretable. To make these curve drawings computer-interpretable, we fit the user-given curves to mathematical functions that describe their shapes. Originally, we considered using a nonlinear least squares method to fit curves to a wide variety of parent functions such as exponential, logarithmic, polynomials, square and cube root functions and more. We even considered using a weighted sum of multiple parent functions to fit the user curves. However, we eventually realized that we could fit a polynomial to each of the user curves with relatively high accuracy without the added complexity and bugs of the nonlinear least squares fitting.

Data points for polynomial curve fitting were extracted from the user drawings by processing the images pixel by pixel starting from the decision variable (price, distance, average rating, or number of reviews) axis and incrementing vertically through the images until a nonwhite pixel was discovered. This process was repeated along the decision variable axis, and it is illustrated for the "distance" decision variable in the figure below.

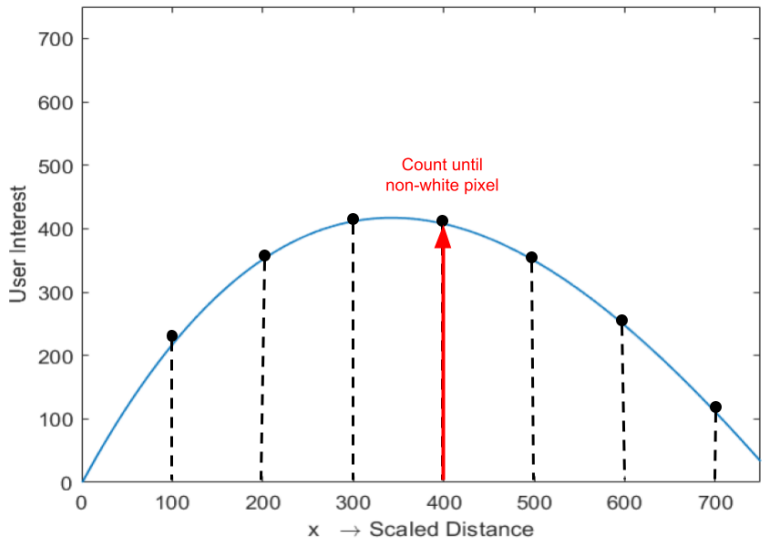


Figure 12: Pixel counting for polynomial curve fitting.

Once a set of points on each of the user-drawn curves were populated, we apply a moving average filter to smooth out any sharp corners that could interfere with curve fitting. At this point, we could fit a polynomial to these points using the NumPy polynomial polyfit function [5]. However, at a minimum, this polyfit function requires the points to fit the polynomial to and the degree of the polynomial. Because the degree of the polynomial is not specified by the user, we use k-fold cross-validation [6] to determine which polynomial degree best fits our data and would therefore lead to the smallest error when used to interpolate between sample points.

We use k-fold cross-validation by dividing our data set (ordered by x value) into k equally sized sets. We use $k-1$ of these sets to fit polynomials of degrees between zero and nine and reserve one set as a test set. We then find the sum of the square error between each of the points in the test set and the value of the polynomials at each of the x-values of the test set points. We then repeat the process with new testing and training sets until every set has functioned as the testing set. The polynomial degree that corresponds to the minimum sum of square error is considered to be the polynomial degree that yields the best fit to the user data. In equation form, for a test set of length N made up of points in (x, y) format and training-set-fitted polynomials of degree d in the form $f_d(x)$:

$$D = \min_{d \in \{\mathbb{Z} \cap [0,9]\}} \sum_{i=0}^{N-1} \left(y_i - f_d\left(x_i\right)\right)^2 \tag{1}$$

Once the ideal degree (D) is determined, the NumPy polyfit function is executed. It returns the coefficients of the best-fit polynomial. For our cross-validation model, we use $k = 3$ sets to ensure the testing set is sufficiently large and bring down computation time.

3.12 Creating a Combined Interest Function

Using the curve-fitting strategy outlined above, we fit a polynomial function to each of the four curves drawn by the user. Each of these interest curves exists in two-dimensional space with a single input variable and a single output variable. We would like four decision variables (price (p), distance (d), average rating (r), and number of reviews (n)) to serve as our inputs, and we would like our combined interest function to produce a single output. Therefore, one can envision our combined interest function as a surface in five-dimensional space. We create this combined interest function by summing up the four interest curves fit to the user drawings across different dimensions. In equation form, our combined interest function ($C(p, d, r, n)$) is:

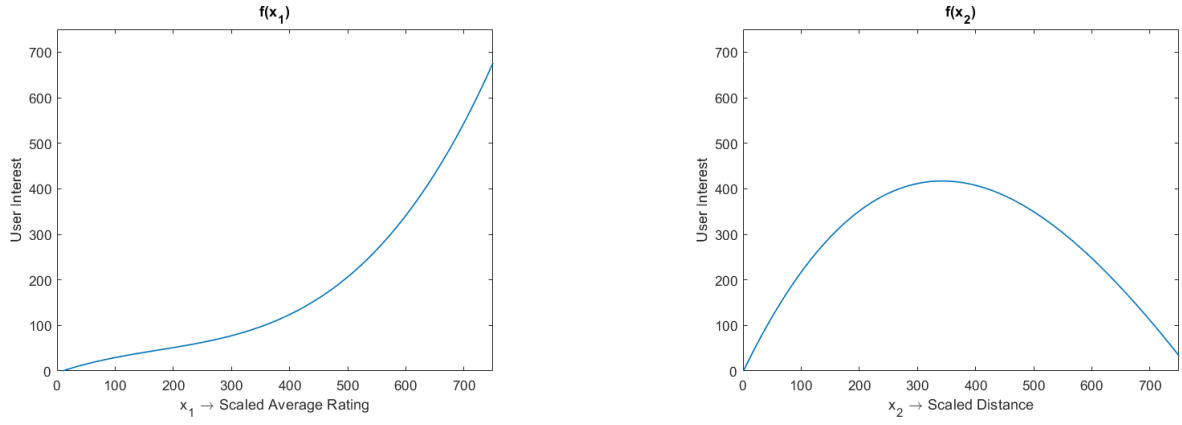
$$C(p, d, r, n) = f_1(p) + f_2(d) + f_3(r) + f_4(n) \tag{2}$$

This equation will serve as our objective function for shortest distance recommendations (SDRs) in Section 3.15. We use this combined interest function together with data from the Yelp API to evaluate various potential events to recommend to the user.

3.13 Evaluating Events

Our recommendation strategy is based on the premise that each possible candidate activity that can be considered for recommendation in a given city has available data for its price, distance, average rating, and number of reviews. This means that each candidate event specifies a concrete value for each of the decision variables we are considering, and therefore each candidate event maps to a point of the 5-D surface defined by the user's combined interest function. This point, $(p, d, r, n, C(p, d, r, n))$, provides the basis for numerically scoring events.

To simplify the visualization of scoring events, we can show in 3-D space how events are scored if we choose to only consider two decision variables instead of four. Instead of price, distance, average rating, and number of reviews suppose we have only the average rating and distance from the user as our decision variables. This would leave us with the following interest curves.

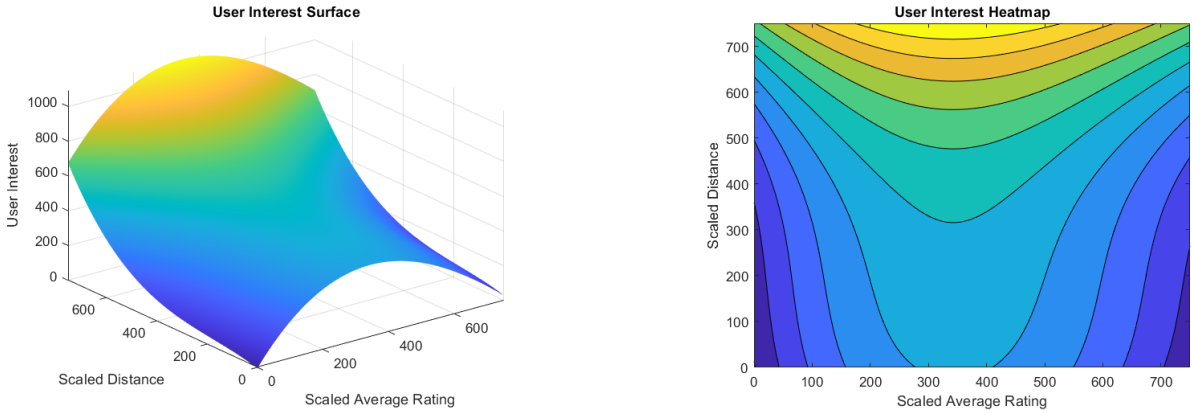


(a) Average rating polynomial

(b) Distance polynomial

Figure 13: Polynomials fitted to user drawings.

These curves can then be summed as described in Section 3.12 to create a combined interest function in 3-D space. The combined interest function can now be visualized as a surface which we can call the user interest surface, or as a heatmap, similarly called the user interest heatmap.



(a) Visualized as a user interest surface.

(b) Visualized as a user interest heatmap.

Figure 14: 3-D visualizations of two-variable combined interest function.

It is important to note here that, while the user drew curves with x-axes for average rating on a scale of zero to five stars and distance on a scale from zero to fifty kilometers, their drawings both share the same domain of zero to 775 pixels. Therefore, when placing each candidate activity on the user interest surface, the average rating will be scaled by $\frac{775\ pixels}{5\ stars}$ and the distance will be scaled by $\frac{775\ pixels}{50\ km}$, to ensure it is faithfully represented.

3.14 Highest Interest Recommendation (HIR)

Scaling events and placing them on the user interest surface, we might have a set of candidate events that appear as follows.

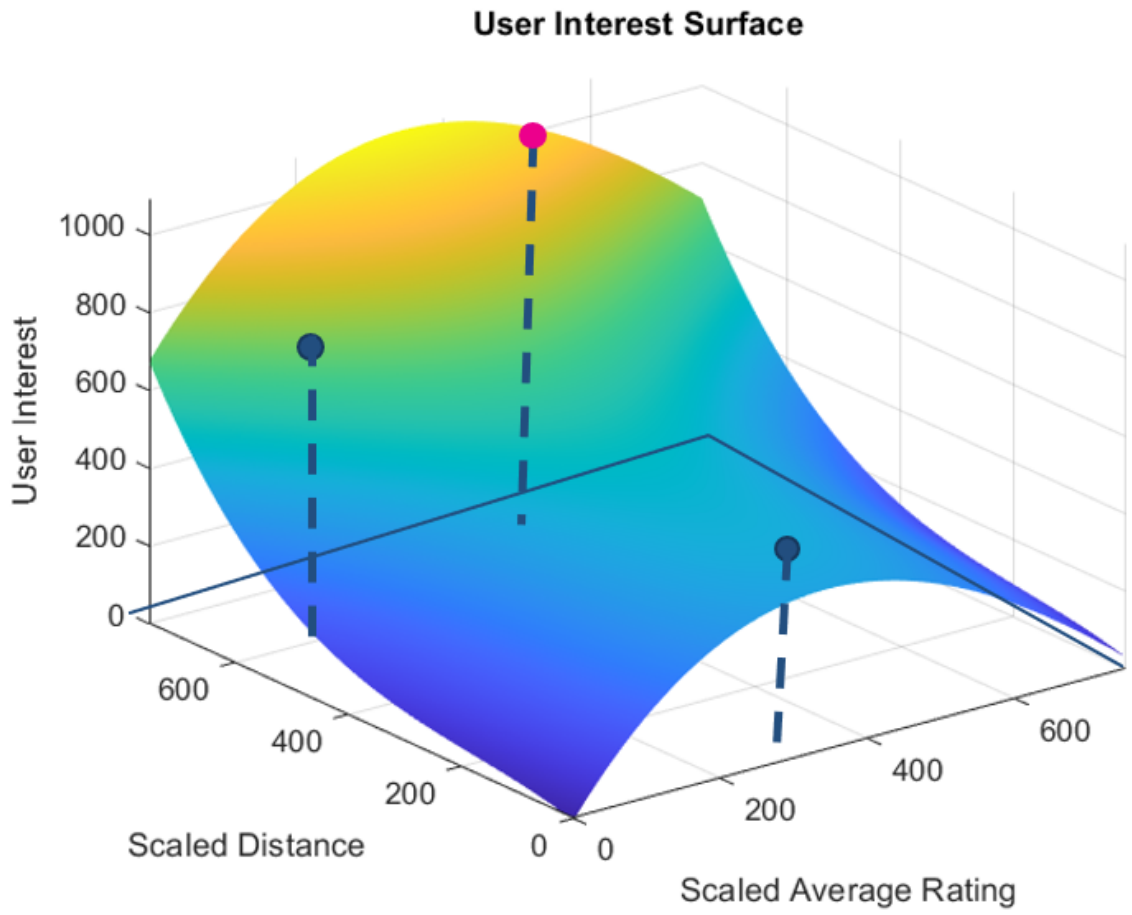


Figure 15: User interest surface with events plotted.

Given these three activities represented by dots on the user interest surface, we can already make one form of score determination. Since points on the surface that sit at a higher user interest level (a score of roughly 1000) correspond to higher interest, we can say that the pink dot event would be a better candidate than either of the blue dots. Taking the raw, unitless user interest score from this graph is one way of evaluating activities. We call this form of evaluation a highest interest recommendation (HIR). Since a higher interest is preferred to a lower interest, we aim to maximize the score when using a highest interest recommendation.

Since our real model uses four decision variables instead of two, for a given event:

$$HIR\ score = C\left(p_{event}, d_{event}, r_{event}, n_{event}\right) \tag{3}$$

3.15 Shortest Distance Recommendation (SDR)

With the HIR already considered, there is a second way to evaluate activities. We call this other way the Shortest Distance Recommendation (SDR). Instead of finding which points lie at the highest user interest value on the user interest surface, we find the highest (or optimal) point on the user interest surface, and calculate the Euclidian distance between the decision variable values associated with that point and the decision variable values associated with each candidate event. This optimal point is found by maximizing the combined interest function using the SciPy library's basinhopping method [7]. As mentioned earlier, there are four decision variables used in this maximization (price, distance, average reviews, and number of reviews). The optimization is constrained to the domain of the drawn image, meaning that each decision variable is limited to a range of [0,775] in the optimization process. This prevents the optimization from

selecting a point where the fitted polynomial fit may behave unexpectedly outside of the user's specified domain. Visualized in 3-D using two decision variables:

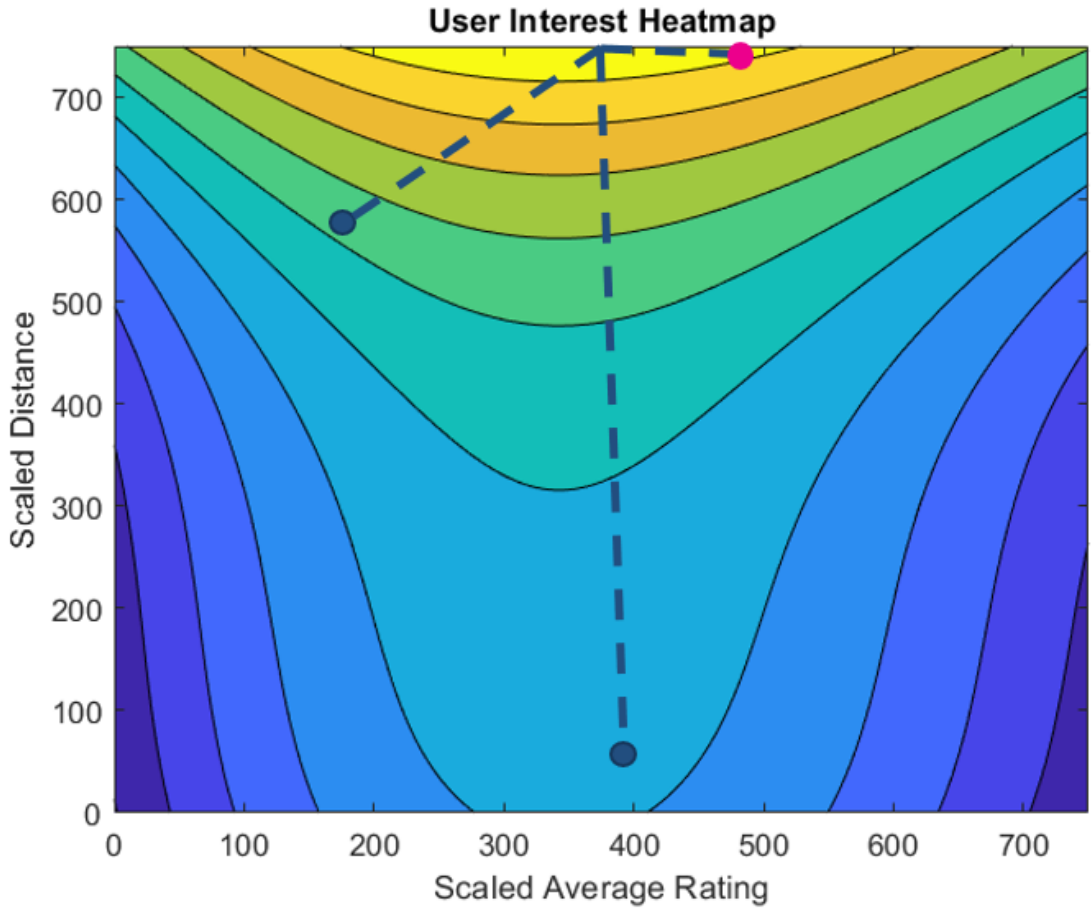


Figure 16: User interest heatmap with event distances from optimality.

Like in the case of the HIR, the event shown in pink would be the preferred activity for an SDR because it is the closest to optimality as measured by Euclidian distance. It follows that for a given event:

$$SDR\ score = (p_{optimal} - p_{event})^2 + (d_{optimal} - d_{event})^2 + (r_{optimal} - r_{event})^2 + (n_{optimal} - n_{event})^2 \quad (4)$$

Both HIR scoring and SDR scoring can be used to create weights for Dijkstra's algorithm (although HIR scores need to be transformed because Dijkstra's algorithm attempts to minimize path weight). This is covered in more depth in Section 3.16. The advantages and drawbacks of HIR and SDR are covered in Section 5.

3.16 Dijkstra's Optimal Path Algorithm

We use the Python package NetworkX to represent our possible routes through the city as a graph of nodes (destinations) and edges (HIR scores or SDR scores representing the interest of traveling from the current event to a destination). We first build the structure of the graph depending on how many activities are in each of the filtered datasets, with start and end nodes allowing us to run Dijkstra's from the start of the graph to the end.

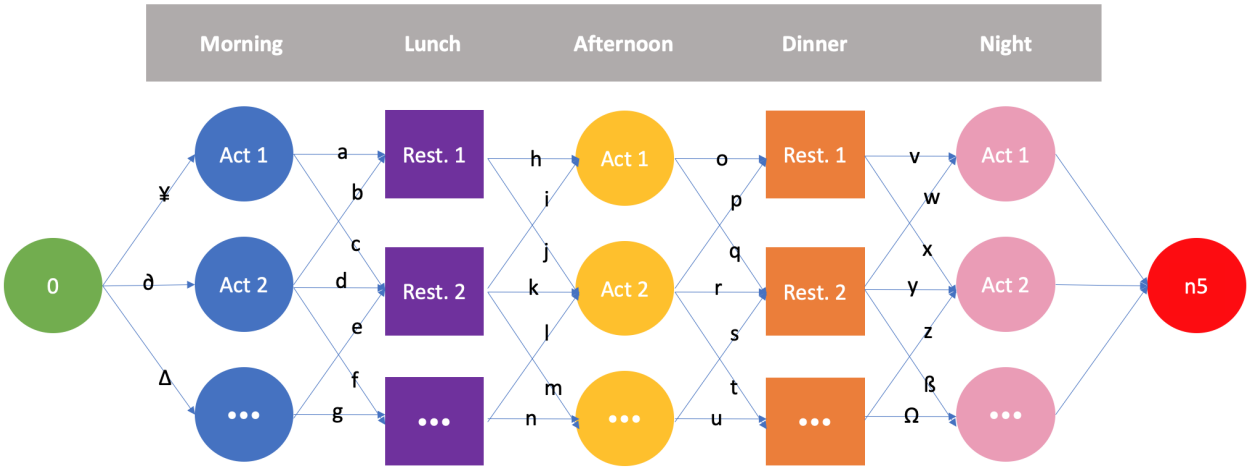


Figure 17: Graph with start and end nodes allowing for Dijkstra's algorithm to run.

Using the HIR interest scores we generated earlier, we create tuples in the form of `(start-node, end-node, interest-score)` to populate the edges between the nodes with proper weights describing the utility gained from moving between activities. The NetworkX representation of the above graph is seen below.

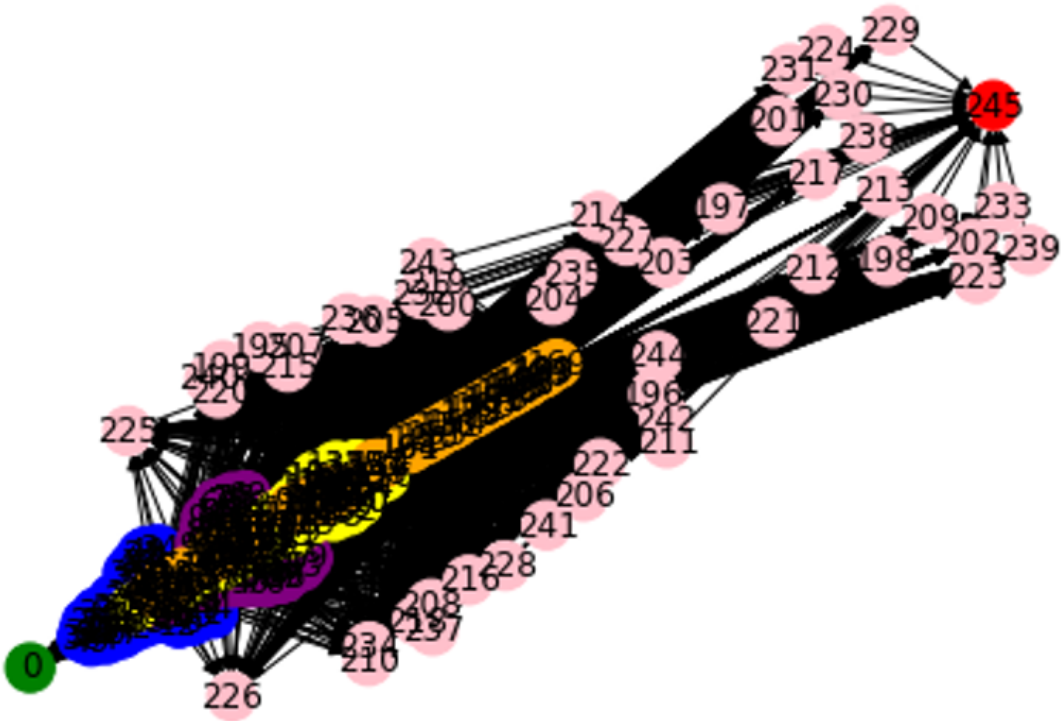


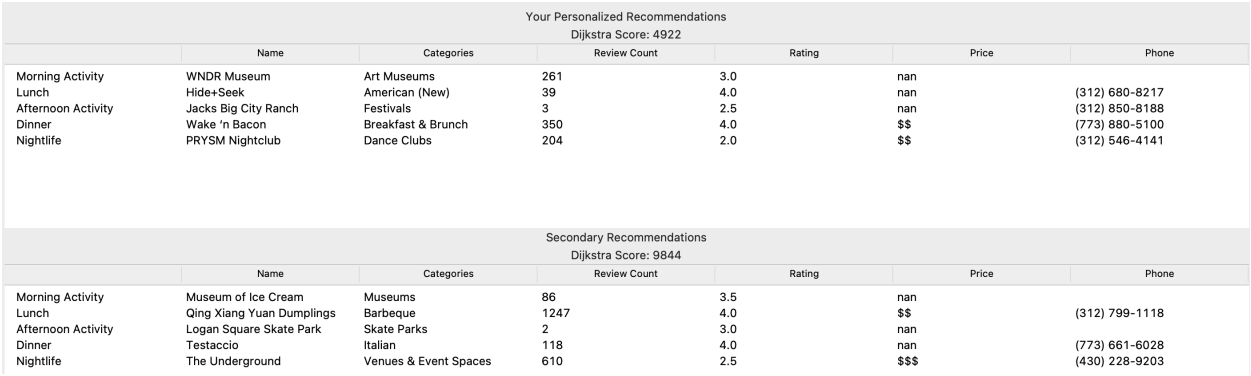
Figure 18: Traversal from node 0 to n5 (n5 is the sum of all necessary nodes before it).

Once the graph is prepared with nodes and weighted edges, we run `dijkstra-path(Graph, 0, n5)` to find the optimal path throughout the day. This returns a list of nodes (Ex: [0,24,53,83,132,207,n5]) which

we can then remove from the graph (except for 0 and n5) to run `dijkstra-path(Graph,0,n5)` again on the find the second-best recommendation for the user to compare.

3.17 View Recommendations

We relate the optimal path node values back to their relative indices in our filtered datasets, then save a simplified dataframe with only the recommendations and information relevant to the user visible.



Your Personalized Recommendations
Dijkstra Score: 4922

	Name	Categories	Review Count	Rating	Price	Phone
Morning Activity	WNDR Museum	Art Museums	261	3.0	nan	
Lunch	Hide+Seek	American (New)	39	4.0	nan	(312) 680-8217
Afternoon Activity	Jacks Big City Ranch	Festivals	3	2.5	nan	(312) 850-8188
Dinner	Wake 'n Bacon	Breakfast & Brunch	350	4.0	$$	(773) 880-5100
Nightlife	PRYSM Nightclub	Dance Clubs	204	2.0	$$	(312) 546-4141

Secondary Recommendations
Dijkstra Score: 9844

	Name	Categories	Review Count	Rating	Price	Phone
Morning Activity	Museum of Ice Cream	Museums	86	3.5	nan	
Lunch	Qing Xiang Yuan Dumplings	Barbeque	1247	4.0	$$	(312) 799-1118
Afternoon Activity	Logan Square Skate Park	Skate Parks	2	3.0	nan	
Dinner	Testaccio	Italian	118	4.0	nan	(773) 661-6028
Nightlife	The Underground	Venues & Event Spaces	610	2.5	$$$	(430) 228-9203

Figure 19: Example final recommendations display.

We also sum up the edges in the Dijkstra's path to give the user a numeric metric to compare the itineraries. We use the Treeview TKinter method to display the tabular data.

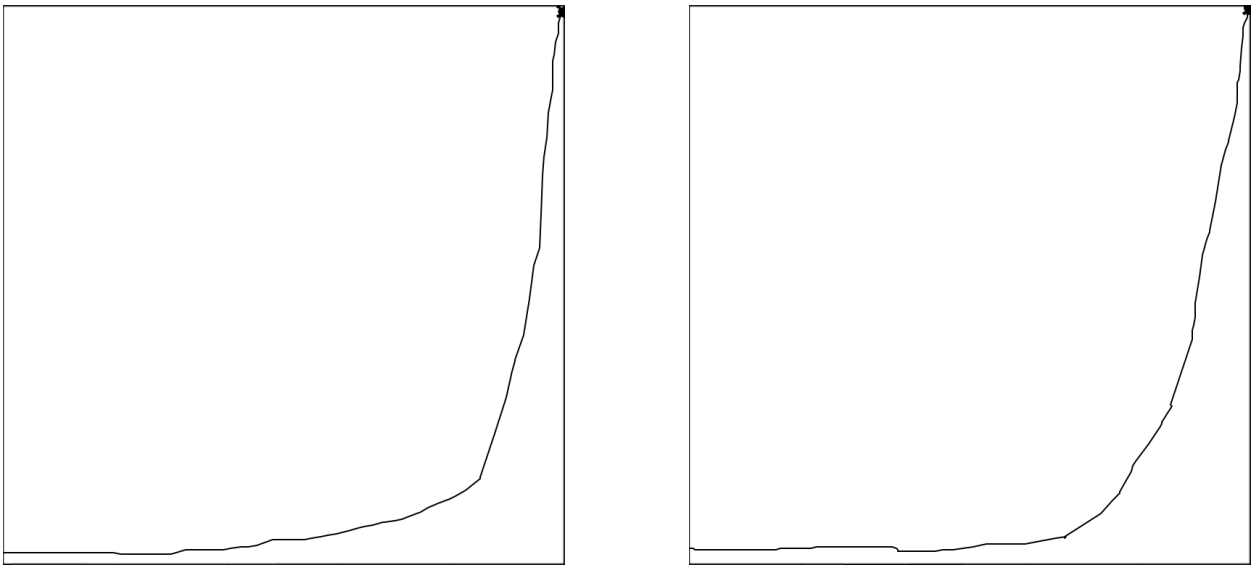
4 Results

The most important results of our project come from determining if our recommendations match the preferences inputted by the user. The main inputs are city, categories, and preference curves.

To first check that the city input is registered and applied properly, we can check in a variety of ways. Most intuitively, one can eyeball check that clicking 'St. Louis' results in the activities in St. Louis only and then clicking 'Los Angeles' after that update the dataset and recommendations to include locations such as 'LUME Los Angeles,' so we can be confident that the city input works properly.

We can then check that the category elimination step works properly. In the early testing phase, we noticed that some items that were originally meant to be eliminated were still being recommended. We fixed this issue by ensuring the *dropAll*() function runs at the right time and reindex the dataframe after removing rows to label and recommend the correct activities.

In order to test that the preference curves work correctly, we draw curves with extremely maximized preferences and insert fake locations ($MaximizedNightlife1$, etc.) with extremely maximized parameters into the dataset. If the extreme curves result in the extreme recommendation options, we can consider our program a success. We use the HIR scoring method for the following results. Below are the curves we use to represent extreme preferences.



(a) Extreme desire for well-traveled locations.

(b) Extremely expensive taste.

Figure 20: Extremely positive preference curves.

We use the following extreme fake data points to check if our program can narrow in on the right extrema:

name	review_count	rating	price
Maximized Activities 1	100000	5	$$$
Maximized Activities 2	90000	4.9	$$$
Maximized Activities 3	80000	4.8	$$$
Maximized Activities 4	70000	4.7	$$$
Minimized Activities 1	0	0	$
Minimized Activities 2	1	0.1	$
Minimized Activities 3	2	0.2	$
Minimized Activities 4	3	0.3	$

Figure 21: Fake activities with maxed-out parameters for testing purposes.

In the test results below, we see that the most extreme test data shows up in the primary recommendations and that the second-most extreme data shows up in the secondary recommendations.

Your Personalized Recommendations
Dijkstra Score: 14284

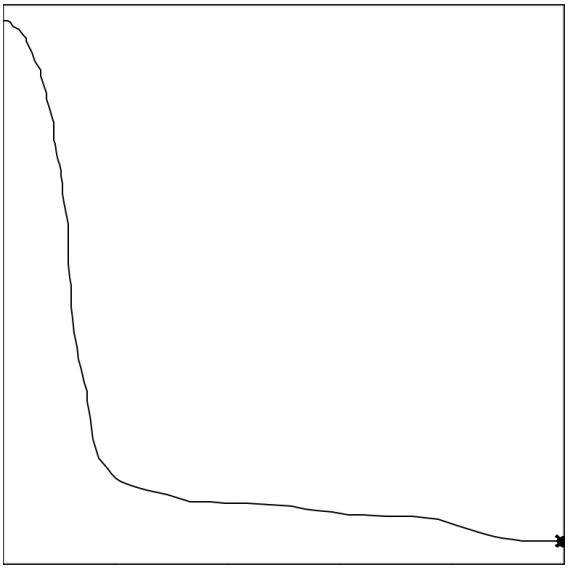
	Name	Categories	Review Count	Rating	Price	Phone
Morning Activity	Maximized Activities 2	Outdoor Movies	90000	4.9	$$$	
Lunch	Maximized Food 1	Hawaiian	100000	5.0	$$$	(213) 221-4070
Afternoon Activity	Maximized Activities 1	Escape Games	100000	5.0	$$$	(323) 497-9776
Dinner	Maximized Food 2	Chinese	90000	4.9	$$$	(323) 645-7272
Nightlife	Maximized Nightlife 1	Karaoke	100000	5.0	$$$	(213) 383-8686

Secondary Recommendations
Dijkstra Score: 28568

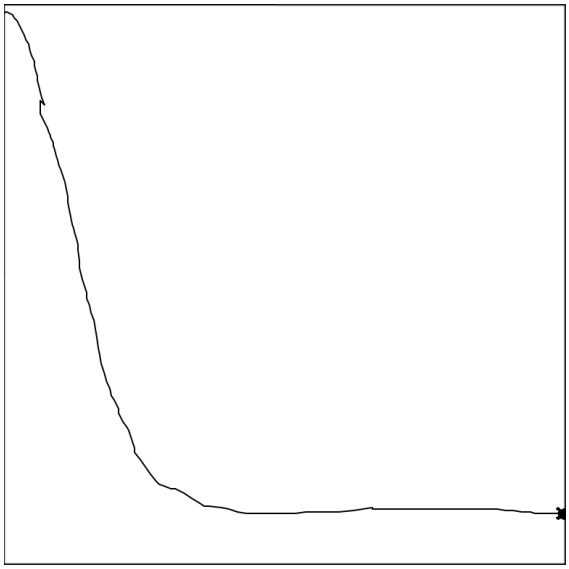
	Name	Categories	Review Count	Rating	Price	Phone
Morning Activity	Maximized Activities 4	Landmarks & Historical Building	70000	4.7	$$$	
Lunch	Maximized Food 3	Barbeque	80000	4.8	$$$	(323) 645-7366
Afternoon Activity	Maximized Activities 3	Trainers	80000	4.8	$$$	(310) 564-6508
Dinner	Maximized Food 4	Desserts	70000	4.7	$$$	
Nightlife	Maximized Nightlife 2	Cocktail Bars	90000	4.9	$$$	(213) 908-7037

Figure 22: Maximized final recommendations display.

We complete a similar test with extremely minimum preference curves and minimized test data. This ensures that both ends of someone's possible extreme preferences will be expressed in our recommendations. The following figures show the preference curves used and the following recommendations.



(a) Extreme desire for less-traveled locations.



(b) Extremely cheap taste.

Figure 23: Extremely negative preference curves.

Your Personalized Recommendations
Dijkstra Score: 8276

	Name	Categories	Review Count	Rating	Price	Phone
Morning Activity	Minimized Activities 2	Haunted Houses	1	0.1	$	
Lunch	Minimized Food 1	Korean	0	0.0	$	(213) 318-5250
Afternoon Activity	Minimized Activities 1	Art Galleries	0	0.0	$	(213) 608-3039
Dinner	Minimized Food 2	Chicken Wings	1	0.1	$	(213) 263-9492
Nightlife	Minimized Nightlife 1	Lounges	0	0.0	$	(213) 716-8919

Secondary Recommendations
Dijkstra Score: 16552

	Name	Categories	Review Count	Rating	Price	Phone
Morning Activity	Minimized Activities 4	Museums	3	0.3	$	(800) 593-2902
Lunch	Minimized Food 3	Desserts	2	0.2	$	
Afternoon Activity	Minimized Activities 3	Local Flavor	2	0.2	$	
Dinner	Minimized Food 4	Ramen	3	0.3	$	(213) 388-8607
Nightlife	Minimized Nightlife 2	Cocktail Bars	1	0.1	$	(213) 568-3224

Figure 24: Minimized final recommendations display.

5 Discussion

5.1 Objective Satisfied

As shown in Figure 19, our product delivers a final series of events for the user, successfully advancing through all of the information flow stages outlined in Figure 1. In advancing through these information flow stages, we demonstrate the function of several deliverables mentioned in Section 7: the GUI, the optimization algorithm, an implementation of Dijkstra's algorithm, and the generation and use of a dynamic dataset.

5.2 Interpretation of Results

Our testing in Section 4, though limited, demonstrates that our recommendation methods can match user curves heavily favoring maximal values for our decision variables with events with artificially maximized metrics in our data set. Similarly, our recommendation methods can match user curves heavily favoring minimal values for our decision variables with events with artificially minimized metrics in our data set. With these basic test cases behaving as expected and the application providing seemingly reasonable recommendations in informal tests, we are content with our application's function.

5.3 Method Strengths and Limitations

Many of the strengths of our methods come from the creative integration of existing ideas. Many of the mathematical strategies we use throughout the recommendation process such as polynomial curve fitting and multivariable optimization using basin hopping are well-established. Therefore, we were able to well-understood Python library functions from NumPy [5] and SciPy [7] to perform optimization that is theoretically reliable.

Limitations for our recommendation methods arise when we try to perform optimization processes such as curve fitting and interest function minimization on unusual user inputs. We have not yet found a way that users could draw a poorly-defined interest, but it is still likely possible that there possible interest curves that can be drawn that would create poorly fitted interest curves, and consequently poor or even malfunctioning recommendation outputs.

Further, Sections 3.14 and 3.15 outline our strategies for making event recommendations of two kinds: highest interest recommendations (HIRs) and shortest distance recommendations (SDRs). HIRs and SDRs both have advantages and disadvantages. Because HIRs score by the interest level of events when the event's characteristics are plugged into the combined interest function, an HIR will most faithfully represent a recommendation based on the curve the user drew.

In an ideal world, HIR recommendations would be strictly better than SDR recommendations. However, because people often draw rough or uneven curves, sometimes unintended curve features are made very pronounced when polynomial curve fitting occurs. Therefore, Shortest Distance Recommendations (SDRs), which score based on the Euclidian distance between a candidate event and the optimal event, may be more effective than HIRs for rough or unorthodox curve shapes.

One final limitation of our recommendation strategy is its difficulty to test. Verifying that a correct output is achieved is difficult because the data sets for events is large and externally sourced. Therefore, we may not catch errors in our event data. Further, it is difficult to determine whether a result from Dijkstra's algorithm is logical. Due to the nature of Dijkstra's algorithm, checking intermediate values during its execution does not provide a meaningful indication of whether it is functioning as expected.

5.4 Comparison with Product Alternatives

There are many online trip-planning services. For example, services such as those provided by Expedia and Tripadvisor unify information from various sources to simplify trip planning. While this is useful information, these massive amounts of information can be overwhelming, cumbersome, and lacking in personalization. Planning trips using Google features can provide better personalization; Google algorithms have large amounts of data collected on their customers. However, Google lacks complete itinerary recommendation features. None of these larger services seems to directly conflict with ours.

One potential competitor we found is a smaller venture called Inspirock [1]. Like our product, Inspirock allows users to plan trips to a set of destination cities and provide event category preferences. However, our

drawn curves strategy for quickly and accurately gathering user data is not used by Inspirock or any of the larger planning services mentioned in this section. The drawn curves feature, as far as we can tell, is truly novel in the travel planning market.

5.5 Discrepancies

Our recommendation process occasionally provides erroneous SDR results because our global optimization process to find the ideal event values sometimes finds (and becomes caught in) local maxima instead of the function's global maximum. We were unable to resolve this issue completely by fine-tuning optimization parameters. This is a topic of interest in future developments.

5.6 Potential Future Developments

Beyond fixing the discrepancy mentioned in Section 5.5, we would like to use a combination of HIR and SDR scores to weight network edges and make recommendations. Better HIR scores are higher, which would not make raw HIR scores a good candidate for implementation with Dijkstra's algorithm. One possible new way to weight a theoretical network edge leading from arbitrary node `0` to arbitrary node `1` could be:

$$weight_{0\to1} = SDR\ score_{0\to1} - C\,(HIR\ score_{0\to1}) \tag{5}$$

In this case, C is some scaling factor to weight the HIR score relative to the SDR score. Many schemes for potentially using the HIR score and SDR score together are possible; this is merely an example.

Additional future developments we have envisioned for this product relate to its reliability. Toward the end of the semester, we began developing test cases that could suggest that our recommendation strategy is working as expected, but it is difficult to prove that we are getting the correct results. Specifically, we would like to test Dijkstra's algorithm section of the recommendation more closely, maybe by using fewer candidate events, simplifying the network. Further, we could compare the results from our current implementation of Dijkstra's algorithm with other Dijkstra's algorithm implementations.

6 Conclusion

From testing our product ourselves and with our friends, we are confident the recommendations and schedule are appropriate. However, there are minor improvements we have thought about as a team. The most important of these improvements is to improve the user interface by transitioning it into a publishable website and by making it more intuitive and user-friendly. We also want to add photos of the restaurants and activities to display the user's results better and to enhance their experience with the product. In addition, we would like to improve the time efficiency of our product by saving fewer files and using graph weights, so that the user has a smoother experience.

While there are similar products in the market, we believe this idea has potential and by scaling it we can potentially turn it into a business or a profitable product. To do that, we need to host the program on a server and distribute it as a web application or mobile phone application. We also need to conduct more market research and ensure our product is ready to be used by paying customers.

7 Deliverables

Our deliverables consist of 5 categories:

1. Graphical User Interface

   The GUI allows the user to enter their preferred destination, categories to be eliminated, and preferences through curve drawings. Finally, after the optimization algorithm for recommendations and Dijkstra's algorithm runs in the back-end to produce an itinerary, the GUI displays the results (the travel schedule).

2. Optimization Algorithm

   This algorithm is used to optimize user recommendations. It takes in the x and y locations of pixels in the drawn preference curves to determine activities the user will enjoy.

3. Dijkstra's Method

   This method is used to produce an optimal travel schedule for the user.

4. Dynamic & Comprehensive Dataset

   This is a data set from a Yelp API, which includes locations and activities, restaurants and nightlife in those locations. Our algorithms use this dataset determine the desired recommendations and schedule for the user.

5. Final Report and Website

   We use these to portray everything we have done to complete the project and our results from it in an organized manner.

8 Timeline

Figure 25 shows our expected timeline when we initially proposed this project. While we settled on using Dijksta's algorithm instead of a more customized dynamic programming strategy, we completed milestones for the development of this project almost exactly on the schedule prescribed by this Gantt chart.

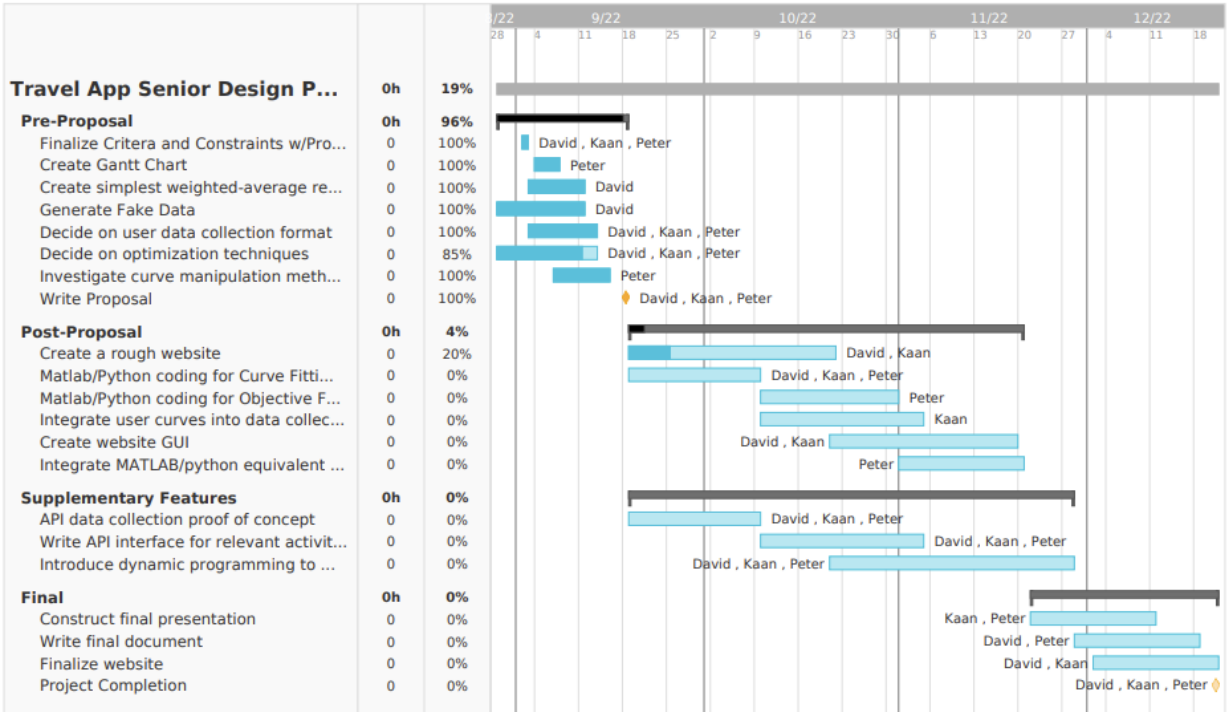


Figure 25: Gantt chart.

Specific responsibilities of each group member are listed on the timeline. Generally, Kaan worked GUI features, user drawing functionality, and pixel extraction. David worked with Kaan on GUI features, handed Yelp API calls and event dataset creation, and implemented Dijkstra's algorithm. Peter handled curve fitting, creation and optimization of the objective function, using event data from the dataset, and HIR/SDR scoring methods.

9 References

[1] Inspirock. [Online]. Available: https://www.inspirock.com/. [Accessed: 18-Dec-2022].

[2] "Turtle - Turtle Graphics," Python documentation. [Online]. Available: https://docs.python.org/3/library/turtle.html. [Accessed: 18-Dec-2022].

[3] "Getting started with yelp fusion API," Yelp Developers. [Online]. Available: https://docs.developer.yelp.com/docs/fusion-intro. [Accessed: 18-Dec-2022].

[4] DaniilSydorenko, "Daniilsydorenko/haversine-geolocation: Get distances between two points or get closest position to current point. based on the haversine formula," GitHub. [Online]. Available: https://github.com/DaniilSydorenko/haversine-geolocation. [Accessed: 18-Dec-2022].

[5] "Numpy.polyfit," numpy.polyfit - NumPy v1.23 Manual. [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.polyfit.html. [Accessed: 18-Dec-2022].

[6] P. Refaeilzadeh, L. Tang, and H. Liu, "Cross-validation," Encyclopedia of Database Systems, pp. 532–538, 2009.

[7] "Scipy.optimize.basinhopping," scipy.optimize.basinhopping - SciPy v1.9.3 Manual. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.basinhopping.htmlscipy.optimize.basinhopping. [Accessed: 18-Dec-2022].