

Finding the best places to eat in a city

1. INTRODUCTION

1.1 Background

As a frequent traveler and a food lover, I have often found it hard to land myself in a good restaurant, especially in a new city. Almost everyone travels, and hence it would be beneficial to them if there were an automated way of finding a good restaurant to eat in, depending on the area they are in and what cuisine they would like.

1.2 Business Problem

The problem here is to know which restaurant out of so many to choose when traveling in a new place. This project aims to provide a wide variety of eatery options to choose from, as well as recommend the best one.

1.3 Interest

Obviously, the interest would be to anyone that travels that is interested in eating out.

2. DATA ACQUISITION AND CLEANING:

2.1 Data Sources

All the data for this project will be scraped using the Foursquare API. The data includes a list of restaurants and eateries in the areas, and tips for each of these restaurants, the number of tips, and the agree and disagree counts of these tips.

2.2 Data cleaning

Data was downloaded in three steps. Each step specified a different location in Vancouver so as to get as many results as possible, since the Foursquare API limits number of results for a particular query to 50 (for the basic account). The data was then loaded into a Dataframe, and the data frames were merged.

Each of the three initial data frames had a similar format in terms of number of columns and column names. For each data set, columns with too many missing values as well as irrelevant data were removed.

After clustering and creating maps with these data frames, tips for each venue were extracted from the Foursquare API and appended to a list. This was then cleaned and only required values were kept and converted into a Dataframe.

3. METHODOLOGY:

3.1 Exploratory Data Analysis

```
Out[2]: [{"name": "Food & Drink", "requestId": "3d0300379e7692708ee2f0c7",  
    "response": {"name": ["Food"], "lat": 49.2821511000000000000000,  
        "name": "King Square Mall + Food Court",  
        "location": {"address": "555 W. 12th Ave",  
            "crossStreet": "Ibex Rd + Gable",  
            "lat": 49.289266618828,  
            "lng": -121.18139773387041,  
            "labelledLatLng": [49.289266618828,  
                "lng": -121.18139773387041],  
            "distance": 174,  
            "formattedAddress": "555 12th",  
            "id": "CA",  
            "city": "Vancouver",  
            "state": "BC",  
            "country": "Canada",  
            "formattedAddress": "555 W. 12th Ave Ibex Rd + Gable",  
            "vancouver BC V6E 1Z3",  
            "CANADA"}},  
        "category": [49.28] "labelledLatLng": [49.289266618828,  
            "name": "Food Court",  
            "placeName": "Food Courts",  
            "shortName": "Food Court",  
            "lang": { "prefix": "https://api.mapbox.com/geocoding/v2/mapbox.places/",  
                "suffix": ".json",  
                "primary": "en"},  
            "formattedLat": "49.289266618828",  
            "hasRich": false,  
            "id": "Sto7222229561edde4b3d957",  
            "name": "Food Court Mallway",  
            "location": {"address": "115 West 1st Ave",  
                "lat": 49.280744474709,  
                "lng": -121.12786385603,  
                "labelledLatLng": [49.280744474709,  
                    "lng": -121.12786385603]},  
                "distance": 106,  
                "id": "CA",  
                "city": "Vancouver",  
                "state": "BC",  
                "country": "Canada",  
                "formattedAddress": ["115 West 1st Ave", "VANCOUVER BC", "CANADA"]},  
                "category": [49.28] "labelledLatLng": [49.280744474709,  
                    "name": "Green Food Gathering",  
                    "placeName": "Green Food Gatherings",  
                    "shortName": "Green Food Gatherings",  
                    "lang": { "prefix": "https://api.mapbox.com/geocoding/v2/mapbox.places/",  
                        "suffix": ".json",  
                        "primary": "en"},  
                    "formattedLat": "49.280744474709",  
                    "hasRich": false,  
                    "id": "Sto7222229561edde4b3d957",  
                    "name": "Green Food Gathering",  
                    "location": {"address": "115 West 1st Ave",  
                        "lat": 49.280744474709,  
                        "lng": -121.12786385603,  
                        "labelledLatLng": [49.280744474709,  
                            "lng": -121.12786385603]},  
                            "distance": 106,  
                            "id": "CA",  
                            "city": "Vancouver",  
                            "state": "BC",  
                            "country": "Canada",  
                            "formattedAddress": ["115 West 1st Ave", "VANCOUVER BC", "CANADA"]},  
                            "category": [49.28] "labelledLatLng": [49.280744474709,
```

[illegible]

```

# clean address data by keeping only last 1000
df_addresses = df_addresses[0:1000]
df_addresses = df_addresses[0:1000]

df_addresses

```

	name	categories	address	city	country	crossStreet	distance	locationAddress	label	lat
0	215 West 1st Ave	Food Court	215 West 1st Ave	Canada	Canada	West 1st & Cambie	174	[55.121111, -123.111111]	215 West 1st Ave, Vancouver BC, Canada	49.25000000000000
1	215 West 1st Ave	Food Court	215 West 1st Ave	Canada	Canada	West 1st & Cambie	174	[55.121111, -123.111111]	215 West 1st Ave, Vancouver BC, Canada	49.25000000000000
2	3000 Cambie St	Chinese Restaurant	3000 Cambie St	Canada	Canada	West 1st & Cambie	352	[49.250000, -123.111111]	3000 Cambie St, Vancouver BC, Canada	49.25000000000000
3	Food Truck	Food Truck	Food Truck	Canada	Canada	West 1st & Cambie	352	[49.250000, -123.111111]	Food Truck, Vancouver BC, Canada	49.25000000000000
4	215 West 1st Ave	Grocery Store	215 West 1st Ave	Canada	Canada	West 1st & Cambie	352	[49.250000, -123.111111]	215 West 1st Ave, Vancouver BC, Canada	49.25000000000000

```

# address of dataframe two
df_addresses2 = df_addresses2[0:1000]
df_addresses2 = df_addresses2[0:1000]
df_addresses2

```

The data frames were merged and a total of 85 venues were left over.

```

Join 3 databases

df_new = pd.concat([df1, df2, df3], sort=False)
df_new.head()

```

	address	categories	city	country	crossStreet	distance	locationAddress	lat	label
0	215 West 1st Ave	Food Court	Canada	Canada	West 1st & Cambie	174	[55.121111, -123.111111]	49.25000000000000	215 West 1st Ave, Vancouver BC, Canada
1	215 West 1st Ave	Food Court	Canada	Canada	West 1st & Cambie	174	[55.121111, -123.111111]	49.25000000000000	215 West 1st Ave, Vancouver BC, Canada
2	3000 Cambie St	Chinese Restaurant	Canada	Canada	West 1st & Cambie	352	[49.250000, -123.111111]	49.25000000000000	3000 Cambie St, Vancouver BC, Canada
3	Food Truck	Food Truck	Canada	Canada	West 1st & Cambie	352	[49.250000, -123.111111]	49.25000000000000	Food Truck, Vancouver BC, Canada
4	215 West 1st Ave	Grocery Store	Canada	Canada	West 1st & Cambie	352	[49.250000, -123.111111]	49.25000000000000	215 West 1st Ave, Vancouver BC, Canada

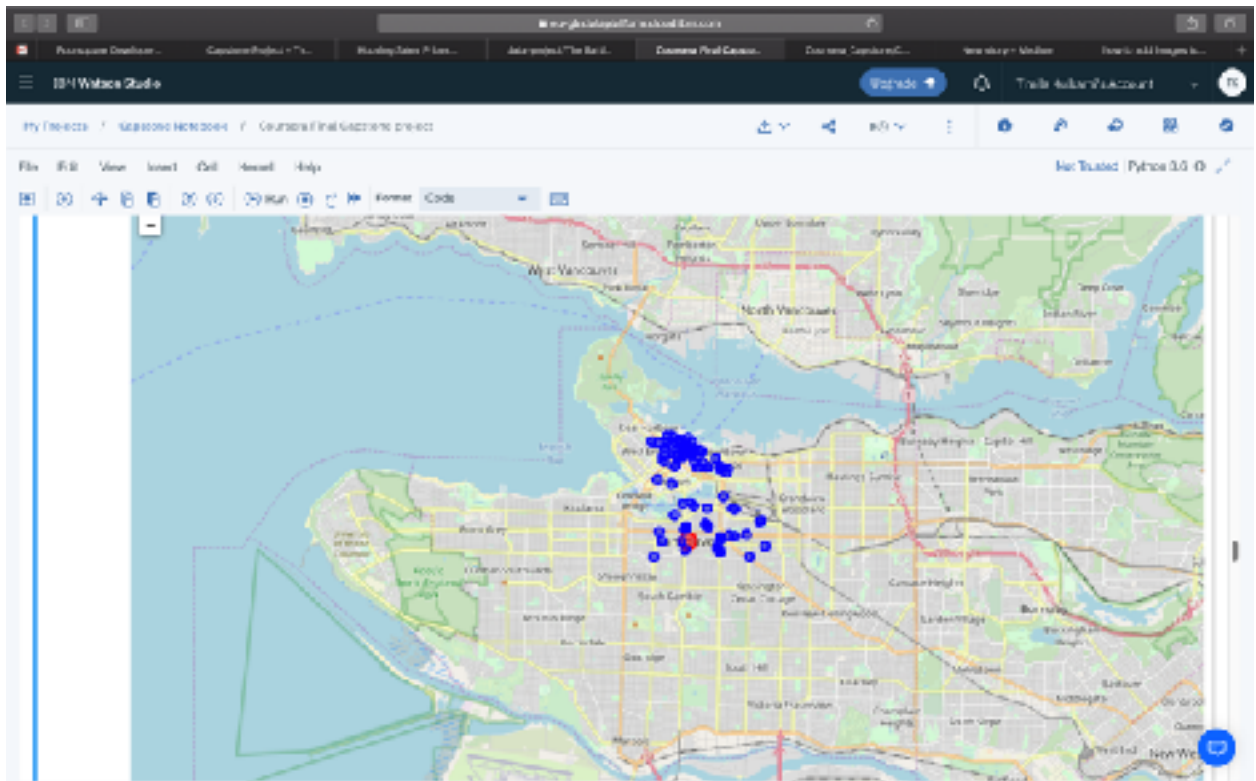
```

# map with all venues
venue_map = folium.Map(location=[49.25, -123.11], zoom_start=11) # generate map centered around the correct location

# add a red circle marker to represent the correct location
folium.Marker([49.25, -123.11], popup='Address', icon=folium.Icon(color='red')).add_to(venue_map)

```

The folium library was used to create a visualization (map) with all the venues superimposed on top.



After this, I ran a K-Means clustering algorithm on the data with number of clusters set to 7. Each cluster represents a different area, and depending on the location of the user of

```

In [25]: # all with only lat and long values
df_lat = df[['name', 'lat', 'lon', 'address', 'neighborhood', 'city', 'country', 'postalCode', 'state', 'city', 'postalCode', 'lat', 'lon']]
df_lat = df_lat[['lat', 'lon']]
df_lat = df_lat[['lat', 'lon']]

Out[25]:
lat    lon
0  49.282800 -123.118000
1  49.282800 -123.118000
2  49.282800 -123.118000
3  49.282800 -123.118000
4  49.282800 -123.118000

In [26]: # set number of clusters
kmeans = KMeans(n_clusters=7, random_state=0, init='k-means++')

# run K-means clustering
kmeans.fit(df_lat[['lat', 'lon']])

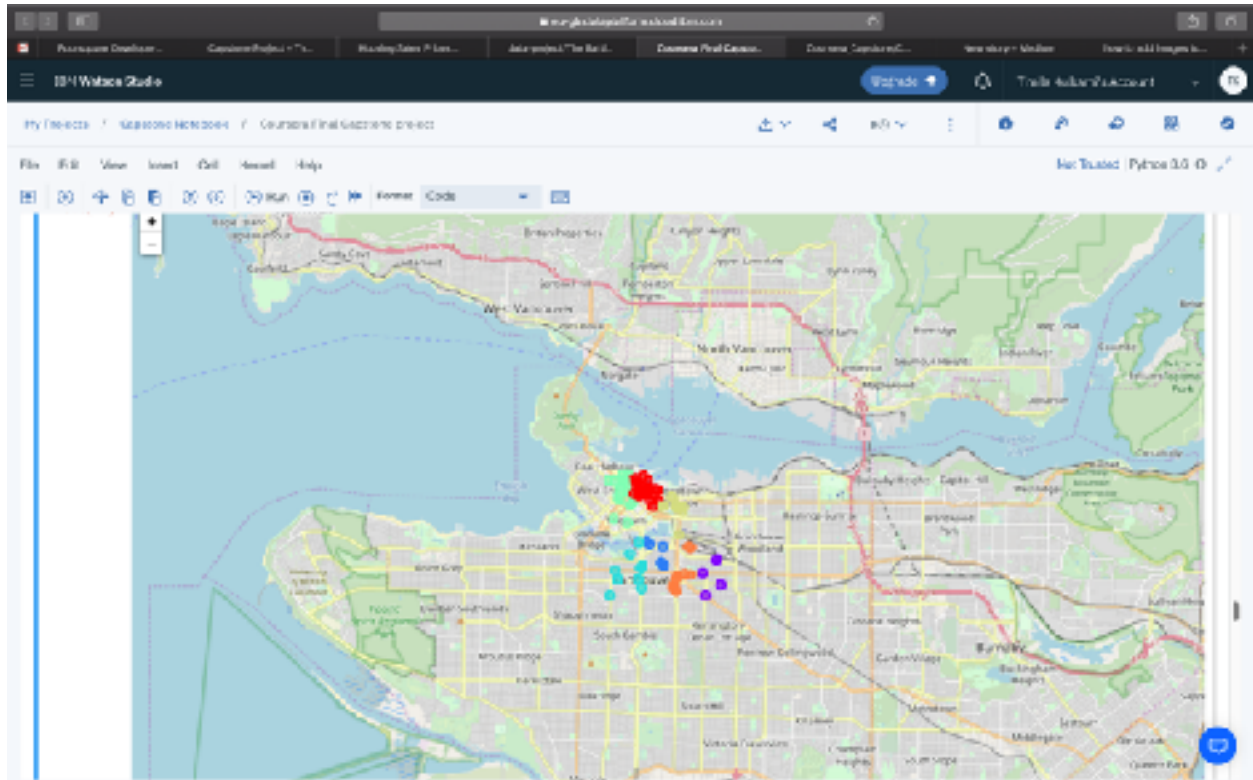
# check cluster labels generated for each row in the dataframe
df_lat['cluster'] = kmeans.labels_

Out[26]:
cluster
0  0
1  0
2  0
3  0
4  0
5  0
6  0
7  0
8  0
9  0
10 0
11 0
12 0
13 0
14 0
15 0
16 0
17 0
18 0
19 0
20 0
21 0
22 0
23 0
24 0
25 0
26 0
27 0
28 0
29 0
30 0
31 0
32 0
33 0
34 0
35 0
36 0
37 0
38 0
39 0
40 0
41 0
42 0
43 0
44 0
45 0
46 0
47 0
48 0
49 0
50 0
51 0
52 0
53 0
54 0
55 0
56 0
57 0
58 0
59 0
60 0
61 0
62 0
63 0
64 0
65 0
66 0
67 0
68 0
69 0
70 0
71 0
72 0
73 0
74 0
75 0
76 0
77 0
78 0
79 0
80 0
81 0
82 0
83 0
84 0
85 0
86 0
87 0
88 0
89 0
90 0
91 0
92 0
93 0
94 0
95 0
96 0
97 0
98 0
99 0
100 0
101 0
102 0
103 0
104 0
105 0
106 0
107 0
108 0
109 0
110 0
111 0
112 0
113 0
114 0
115 0
116 0
117 0
118 0
119 0
120 0
121 0
122 0
123 0
124 0
125 0
126 0
127 0
128 0
129 0
130 0
131 0
132 0
133 0
134 0
135 0
136 0
137 0
138 0
139 0
140 0
141 0
142 0
143 0
144 0
145 0
146 0
147 0
148 0
149 0
150 0
151 0
152 0
153 0
154 0
155 0
156 0
157 0
158 0
159 0
160 0
161 0
162 0
163 0
164 0
165 0
166 0
167 0
168 0
169 0
170 0
171 0
172 0
173 0
174 0
175 0
176 0
177 0
178 0
179 0
180 0
181 0
182 0
183 0
184 0
185 0
186 0
187 0
188 0
189 0
190 0
191 0
192 0
193 0
194 0
195 0
196 0
197 0
198 0
199 0
200 0
201 0
202 0
203 0
204 0
205 0
206 0
207 0
208 0
209 0
210 0
211 0
212 0
213 0
214 0
215 0
216 0
217 0
218 0
219 0
220 0
221 0
222 0
223 0
224 0
225 0
226 0
227 0
228 0
229 0
230 0
231 0
232 0
233 0
234 0
235 0
236 0
237 0
238 0
239 0
240 0
241 0
242 0
243 0
244 0
245 0
246 0
247 0
248 0
249 0
250 0
251 0
252 0
253 0
254 0
255 0
256 0
257 0
258 0
259 0
260 0
261 0
262 0
263 0
264 0
265 0
266 0
267 0
268 0
269 0
270 0
271 0
272 0
273 0
274 0
275 0
276 0
277 0
278 0
279 0
280 0
281 0
282 0
283 0
284 0
285 0
286 0
287 0
288 0
289 0
290 0
291 0
292 0
293 0
294 0
295 0
296 0
297 0
298 0
299 0
300 0
301 0
302 0
303 0
304 0
305 0
306 0
307 0
308 0
309 0
310 0
311 0
312 0
313 0
314 0
315 0
316 0
317 0
318 0
319 0
320 0
321 0
322 0
323 0
324 0
325 0
326 0
327 0
328 0
329 0
330 0
331 0
332 0
333 0
334 0
335 0
336 0
337 0
338 0
339 0
340 0
341 0
342 0
343 0
344 0
345 0
346 0
347 0
348 0
349 0
350 0
351 0
352 0
353 0
354 0
355 0
356 0
357 0
358 0
359 0
360 0
361 0
362 0
363 0
364 0
365 0
366 0
367 0
368 0
369 0
370 0
371 0
372 0
373 0
374 0
375 0
376 0
377 0
378 0
379 0
380 0
381 0
382 0
383 0
384 0
385 0
386 0
387 0
388 0
389 0
390 0
391 0
392 0
393 0
394 0
395 0
396 0
397 0
398 0
399 0
400 0
401 0
402 0
403 0
404 0
405 0
406 0
407 0
408 0
409 0
410 0
411 0
412 0
413 0
414 0
415 0
416 0
417 0
418 0
419 0
420 0
421 0
422 0
423 0
424 0
425 0
426 0
427 0
428 0
429 0
430 0
431 0
432 0
433 0
434 0
435 0
436 0
437 0
438 0
439 0
440 0
441 0
442 0
443 0
444 0
445 0
446 0
447 0
448 0
449 0
450 0
451 0
452 0
453 0
454 0
455 0
456 0
457 0
458 0
459 0
460 0
461 0
462 0
463 0
464 0
465 0
466 0
467 0
468 0
469 0
470 0
471 0
472 0
473 0
474 0
475 0
476 0
477 0
478 0
479 0
480 0
481 0
482 0
483 0
484 0
485 0
486 0
487 0
488 0
489 0
490 0
491 0
492 0
493 0
494 0
495 0
496 0
497 0
498 0
499 0
500 0
501 0
502 0
503 0
504 0
505 0
506 0
507 0
508 0
509 0
510 0
511 0
512 0
513 0
514 0
515 0
516 0
517 0
518 0
519 0
520 0
521 0
522 0
523 0
524 0
525 0
526 0
527 0
528 0
529 0
530 0
531 0
532 0
533 0
534 0
535 0
536 0
537 0
538 0
539 0
540 0
541 0
542 0
543 0
544 0
545 0
546 0
547 0
548 0
549 0
550 0
551 0
552 0
553 0
554 0
555 0
556 0
557 0
558 0
559 0
560 0
561 0
562 0
563 0
564 0
565 0
566 0
567 0
568 0
569 0
570 0
571 0
572 0
573 0
574 0
575 0
576 0
577 0
578 0
579 0
580 0
581 0
582 0
583 0
584 0
585 0
586 0
587 0
588 0
589 0
590 0
591 0
592 0
593 0
594 0
595 0
596 0
597 0
598 0
599 0
600 0
601 0
602 0
603 0
604 0
605 0
606 0
607 0
608 0
609 0
610 0
611 0
612 0
613 0
614 0
615 0
616 0
617 0
618 0
619 0
620 0
621 0
622 0
623 0
624 0
625 0
626 0
627 0
628 0
629 0
630 0
631 0
632 0
633 0
634 0
635 0
636 0
637 0
638 0
639 0
640 0
641 0
642 0
643 0
644 0
645 0
646 0
647 0
648 0
649 0
650 0
651 0
652 0
653 0
654 0
655 0
656 0
657 0
658 0
659 0
660 0
661 0
662 0
663 0
664 0
665 0
666 0
667 0
668 0
669 0
670 0
671 0
672 0
673 0
674 0
675 0
676 0
677 0
678 0
679 0
680 0
681 0
682 0
683 0
684 0
685 0
686 0
687 0
688 0
689 0
690 0
691 0
692 0
693 0
694 0
695 0
696 0
697 0
698 0
699 0
700 0
701 0
702 0
703 0
704 0
705 0
706 0
707 0
708 0
709 0
710 0
711 0
712 0
713 0
714 0
715 0
716 0
717 0
718 0
719 0
720 0
721 0
722 0
723 0
724 0
725 0
726 0
727 0
728 0
729 0
730 0
731 0
732 0
733 0
734 0
735 0
736 0
737 0
738 0
739 0
740 0
741 0
742 0
743 0
744 0
745 0
746 0
747 0
748 0
749 0
750 0
751 0
752 0
753 0
754 0
755 0
756 0
757 0
758 0
759 0
760 0
761 0
762 0
763 0
764 0
765 0
766 0
767 0
768 0
769 0
770 0
771 0
772 0
773 0
774 0
775 0
776 0
777 0
778 0
779 0
780 0
781 0
782 0
783 0
784 0
785 0
786 0
787 0
788 0
789 0
790 0
791 0
792 0
793 0
794 0
795 0
796 0
797 0
798 0
799 0
800 0
801 0
802 0
803 0
804 0
805 0
806 0
807 0
808 0
809 0
810 0
811 0
812 0
813 0
814 0
815 0
816 0
817 0
818 0
819 0
820 0
821 0
822 0
823 0
824 0
825 0
826 0
827 0
828 0
829 0
830 0
831 0
832 0
833 0
834 0
835 0
836 0
837 0
838 0
839 0
840 0
841 0
842 0
843 0
844 0
845 0
846 0
847 0
848 0
849 0
850 0
851 0
852 0
853 0
854 0
855 0
856 0
857 0
858 0
859 0
860 0
861 0
862 0
863 0
864 0
865 0
866 0
867 0
868 0
869 0
870 0
871 0
872 0
873 0
874 0
875 0
876 0
877 0
878 0
879 0
880 0
881 0
882 0
883 0
884 0
885 0
886 0
887 0
888 0
889 0
890 0
891 0
892 0
893 0
894 0
895 0
896 0
897 0
898 0
899 0
900 0
901 0
902 0
903 0
904 0
905 0
906 0
907 0
908 0
909 0
910 0
911 0
912 0
913 0
914 0
915 0
916 0
917 0
918 0
919 0
920 0
921 0
922 0
923 0
924 0
925 0
926 0
927 0
928 0
929 0
930 0
931 0
932 0
933 0
934 0
935 0
936 0
937 0
938 0
939 0
940 0
941 0
942 0
943 0
944 0
945 0
946 0
947 0
948 0
949 0
950 0
951 0
952 0
953 0
954 0
955 0
956 0
957 0
958 0
959 0
960 0
961 0
962 0
963 0
964 0
965 0
966 0
967 0
968 0
969 0
970 0
971 0
972 0
973 0
974 0
975 0
976 0
977 0
978 0
979 0
980 0
981 0
982 0
983 0
984 0
985 0
986 0
987 0
988 0
989 0
990 0
991 0
992 0
993 0
994 0
995 0
996 0
997 0
998 0
999 0

```

this program, the restaurants can be chosen from the closest cluster.

These clusters were then visualized using folium with venues superimposed. On clicking, we can see the venue name as well as the cluster it belongs to. Each cluster has a different



colour.

Next, I created a list of all the venue ids and looped through that list to get tips for each of the venues. I appended the results I got back from the API to a list after converting it to json.

```

for i in range(0, len(posts)):
    post = posts[i]
    url = post['url']
    title = post['title']
    content = post['content']
    author = post['author']
    date = post['date']
    image = post['image']

    # Create a new post object
    post_obj = Post(title, content, author, date, image)

    # Add the post to the list
    posts_list.append(post_obj)

# Create a new DataFrame
posts_df = pd.DataFrame(posts_list)

# Print the DataFrame
print(posts_df)

```

From this list, I created another list with only the required values. Then I looped through this list and created a DataFrame out of it.

```

# Create a new DataFrame
posts_df = pd.DataFrame(posts_list)

# Print the DataFrame
print(posts_df)

```

id	author	author_id	author_type	content	date	image	url
1	John Doe	1	user	My first post!	2023-01-01	https://i.imgur.com/1234567890.jpg	https://example.com/post/1
2	Jane Smith	2	user	My second post!	2023-01-02	https://i.imgur.com/9876543210.jpg	https://example.com/post/2
3	Bob Johnson	3	user	My third post!	2023-01-03	https://i.imgur.com/0987654321.jpg	https://example.com/post/3
4	Alice Brown	4	user	My fourth post!	2023-01-04	https://i.imgur.com/1098765432.jpg	https://example.com/post/4
5	Charlie Davis	5	user	My fifth post!	2023-01-05	https://i.imgur.com/2109876543.jpg	https://example.com/post/5
6	Diana Prince	6	user	My sixth post!	2023-01-06	https://i.imgur.com/3210987654.jpg	https://example.com/post/6
7	Edward Nigma	7	user	My seventh post!	2023-01-07	https://i.imgur.com/4321098765.jpg	https://example.com/post/7
8	Fiona Glenanne	8	user	My eighth post!	2023-01-08	https://i.imgur.com/5432109876.jpg	https://example.com/post/8
9	George Costanza	9	user	My ninth post!	2023-01-09	https://i.imgur.com/6543210987.jpg	https://example.com/post/9
10	Helen Partridge	10	user	My tenth post!	2023-01-10	https://i.imgur.com/7654321098.jpg	https://example.com/post/10

Then I applied the same cleaning methods to the Dataframe, i.e dropped unwanted columns, missing values, etc.

```

In [34]: # keep only columns that include likes and anything that is associated with tips
df_ratings.columns = ['text', 'agreedCount', 'count', 'group', 'summary', 'id']
df_ratings = df_ratings[df_ratings.columns]

# clean column names by keeping only last term
df_ratings.columns = [column.split('.')[-1] if column in df_ratings.columns
                      else column for column in df_ratings.columns]

df_ratings.head()

Out[34]:
  text                                     agreedCount  count  group  summary  id
0  Go to Yara's Quince and order number 10 Lemon Gl...  12      1  1 like  48555e4e4b2623c1555d4d...
1  Derna with friends, Ovidio and coque...  0      0  NaN  52e780541102c0a0e0a040...
2  Everything is fresh, I can always count on W...  4      0  NaN  5d9d78d0d7125d4f3e17...
3  Jose Peris is delicious...  0      0  NaN  5250a03d11d50d60e0a040...
4  Staff is friendly and helpful, but what sup...  0      0  NaN  5d9d78d0d7125d4f3e17...

In [35]: df_ratings = df_ratings.drop(['group', 'id', 'text'])
df_ratings.head()

Out[35]:
  agreedCount  count  summary  id
0           12      1  1 like  48555e4e4b2623c1555d4d...
1            0      0  NaN  52e780541102c0a0e0a040...
2            4      0  NaN  5d9d78d0d7125d4f3e17...
3            0      0  NaN  5250a03d11d50d60e0a040...
4            0      0  NaN  5d9d78d0d7125d4f3e17...

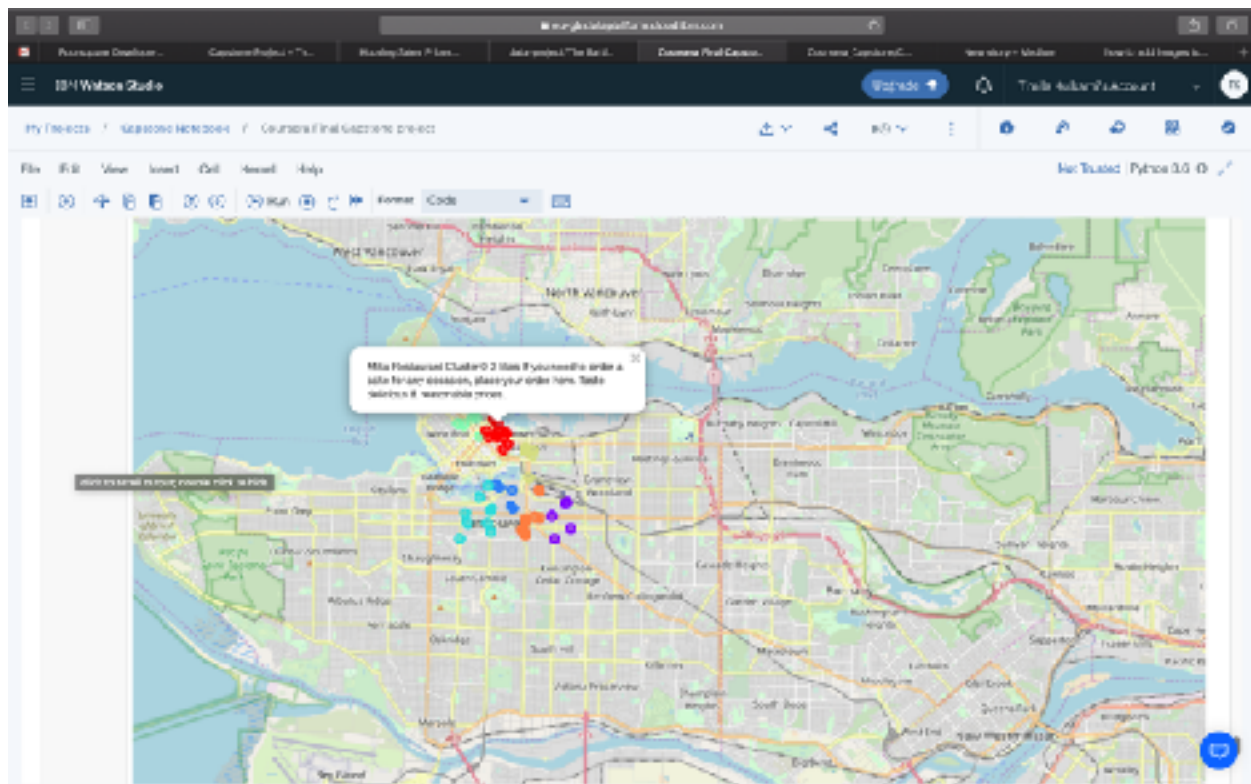
In [36]: # map with clusters and tips

# create map
map_clusters = folium.Map(location=[latitude, longitude], zoom_start=11)

# add color scheme for the clusters

```

I then visualized this using a folium map with the same clusters, venues superimposed on top. When clicked, we can see name of the place, cluster it belongs to, number of likes, and one review.



4. RESULTS:

In the end, I was able to get 88 different venues with all the data in them and tips for 62 of the venues. From this, I was able to create a visualization in the form of the map to display the various restaurants all over the city, and make sure that when clicked on, name, cluster, likes, and tips were displayed. From the map we can also conclude the best restaurants for each cluster, based on tips and likes as follows:

Cluster 0: Yagger's Downtown Restaurant & Sports

Cluster 1: Harry's (Best Foods Grocery)

Cluster 2: Dougie Dog Truck

Cluster 3: City Square Mall - Food Court

Cluster 4: Chongqing Chinese Restaurant & Hawksworth Restaurant

Cluster 5: New Town Bakery & Restaurant
Cluster 6: One-O-One Bulk Food & Deli

5. DISCUSIONS:

There is much scope for improvement in this project. For example, the best restaurant for each cluster could be generated with a bit more code. Also, this project does not include all the venues in and around Vancouver city.

This project was mainly focused on features in the dataset returned by the API. It did not take into account the likes and dislikes of the person, as a perfect recommendation system would.

6. CONCLUSION:

This project is limited to Vancouver city, however, the same technique can be applied to any other city. The locations just need to be replaced with those of the required city. In this aspect, I feel the project has achieved what it was originally meant to do: improve the choice of restaurants for tourists in a new city.