In [794…]
```
sid = results['_scroll_id']
scroll_size = results['hits']['total']
```

In [795…]
```
print('sid = ', sid)
print('Scroll Size = ', scroll_size)
```

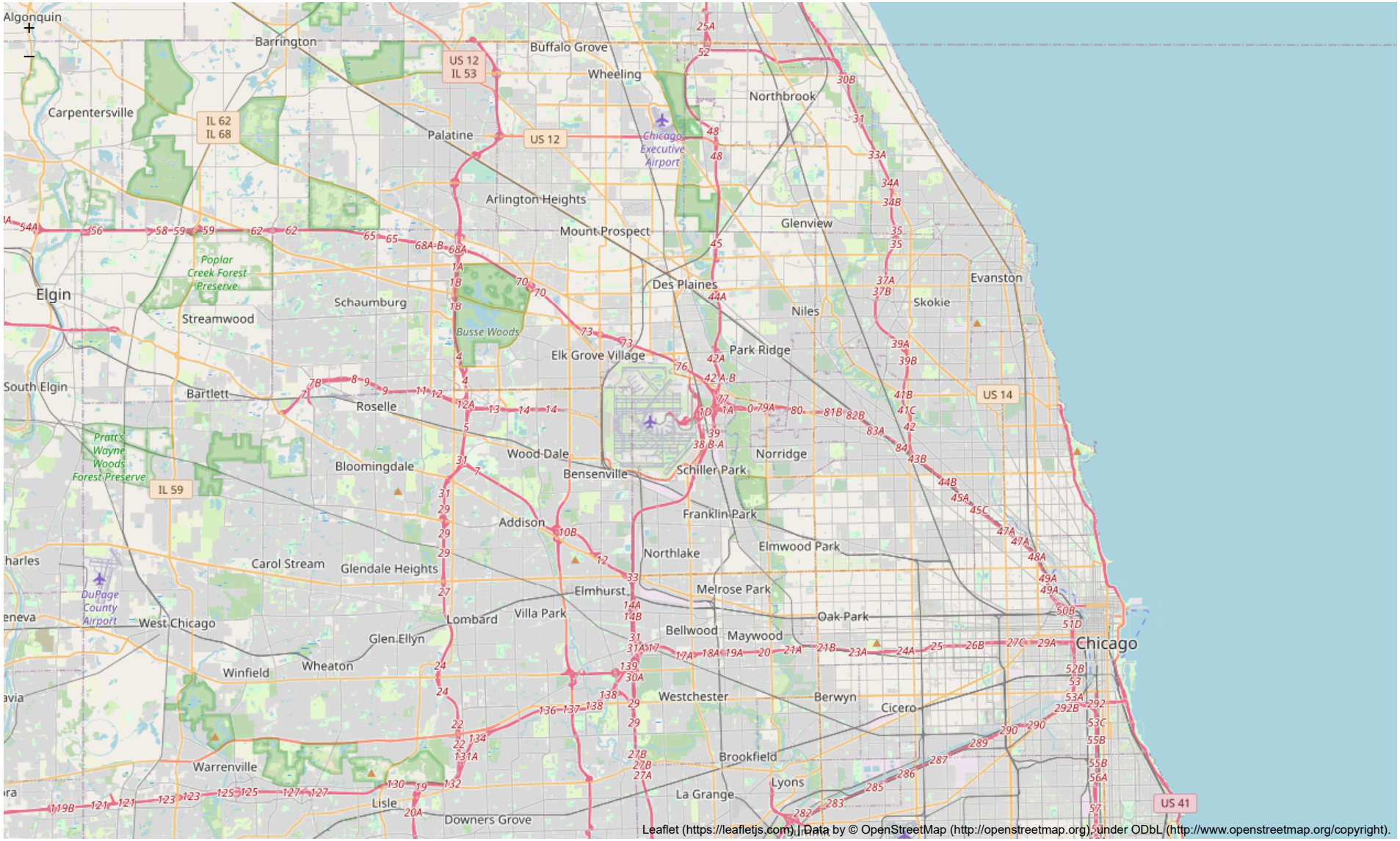
```
sid =  DnF1ZXJ5VGhlbkZldGNoCgAAAAADTeo9FmxSTjR3TFBsUXotQ2ZRQ0toYzlGQ0EAAAAAA03qQBZsUk40d0xQbFF6LUNmUUNLaGM5RkNBAAAAAANN6j4WbFJONHdMUGxRei1DZlFDS2hjOUZDQQAAAAADTeo_FmxSTjR3TFBsUXotQ2ZRQ0toYzlGQ0EAAAAAA03qPBZsUk40d0xQbFF6LUNmUUNLaGM5RkNBAAAAAANN6kWbFJONHdMUGxRei1DZlFDS2hjOUZDQQAAAAADTepCFmxSTjR3TFBsUXotQ2ZRQ0toYzlGQ0EAAAAAA03qQxZsUk40d0xQbFF6LUNmUUNLaGM5RkNBAAAAAANN6kQWbFJONHdMUGxRei1DZlFDS2hjOUZDQQAAAAADTepFFmxSTjR3TFBsUXotQ2ZRQ0toYzlGQ0E=
Scroll Size =  611
```

In [796…]
```
count = 0
list_of_lAT_LONG_pairs = []

while(scroll_size > 0):

    for inspection in results['hits']['hits']:
        current_location_lAT_LONG = []
        document = inspection['_source']
        count = count +1

        #defensive coding to ensure we have the fields in the inspection documents
        if 'Latitude' in document.keys():
            if 'Longitude' in document.keys():
                if 'Address' in document.keys():
                    if(document['Latitude'] != None and document['Longitude'] != None  and document['Address'] != None):
                        current_location_lAT_LONG.append(float(document['Latitude']))
                        current_location_lAT_LONG.append(float(document['Longitude']))
                        list_of_lAT_LONG_pairs.append(current_location_lAT_LONG)

    results = es.scroll(scroll_id = sid, scroll = '2m')
    sid = results['_scroll_id']
    scroll_size = len(results['hits']['hits'])

print("Total number of match with Children's using fuziness:",count)
```

```
Total number of match with Children's using fuziness: 611
```

In [797…]
```
chicago_map = folium.Map([41.90293279, -87.70769386], zoom_start=11)
chicago_map
```

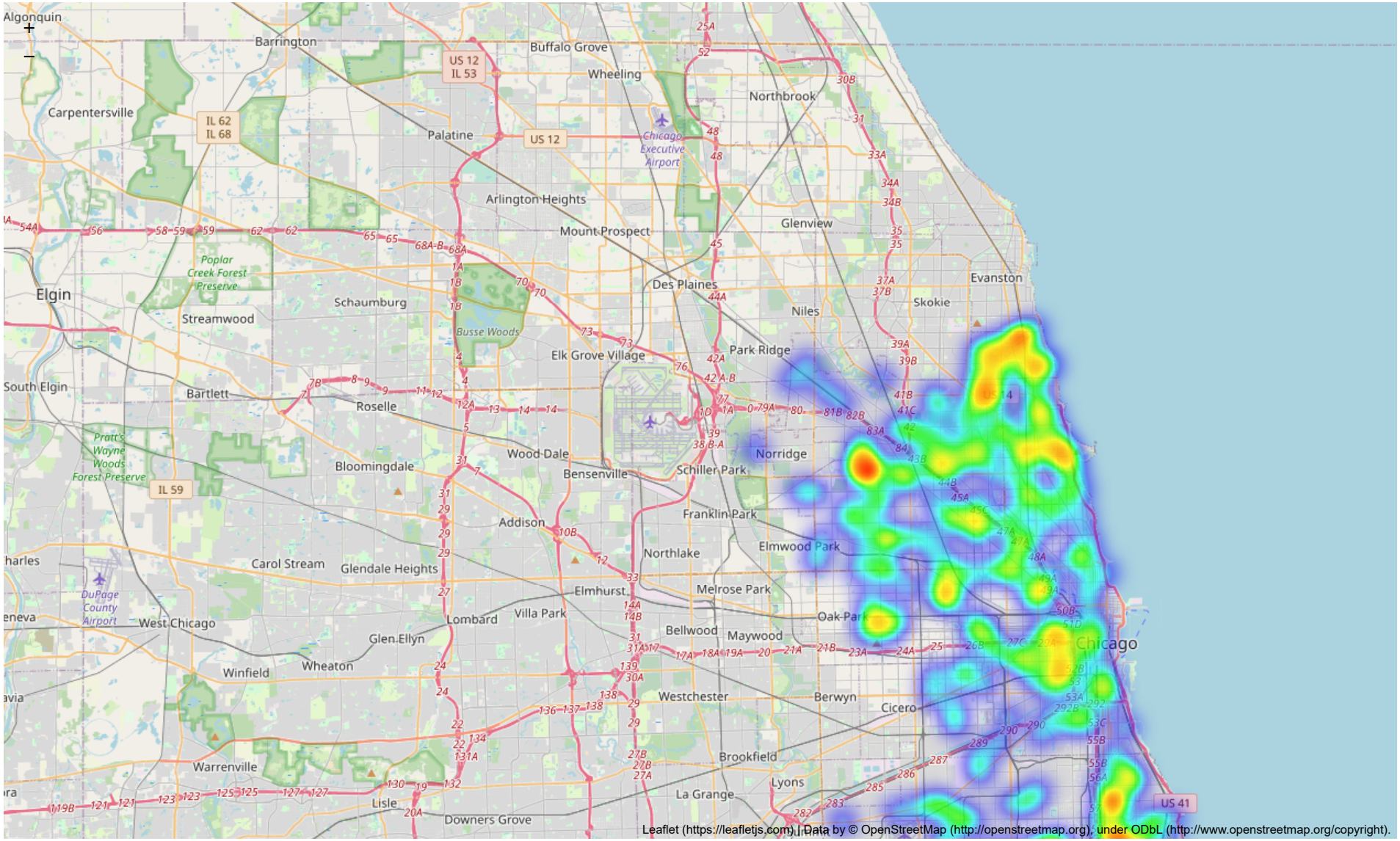
Out[797]:



In [798…

```
# Lets plot the  query matches for  "Children's" on Chicago HeatMap

chicago_map.add_child(plugins.HeatMap(list_of_lAT_LONG_pairs, radius=15))
chicago_map
```

Out[798]:



Requirement #1:

Provide your comparative analysis for the results obtained from 3 experiments you executed above

The first experiment uses the standard regex search, with asterisks before and after the search term "children". The asterisks allow for wildcards before and after the word. It is a way to include certain kinds of typos or plurals of the word, without needing to add additional search terms. This search returned 601 results.

The second and third experiments use fuzzy searches in ElasticSearch. The number after the tilde signifies the maximum edit distance from the queried term. This means that the query will search for words where up to two characters in the search term can be modified, added, or removed and will still be returned as part of the results. It is a powerful way to address issues in unstructured data like typos, where individuals are typing into free text fields and the probability of spelling or syntax errors is quite high. The main difference between experiment two and three is that three includes the "'s" at the end of children. The fuzzy search does not require the specific placement of the wildcard asterisk, as it will check through the entire search term and include results within the maximum edit distance.

Experiment two returned 602 results, only one more than the regex search from experiment one. The third resulted in 611 returned results. Of the three experiments, experiment 3 resulted in the largest number of returned results. This may be due to both the inclusion of the apostrophe and "s" at the end of children, and the flexibility of the fuzzy search to include differences within the term children as well. It seems, at least in this case and those similar to it, that the fuzzy search criteria has greater performance in returning results, especially in unstructured free text data. Fuzzy match tends to work well in longer words and search terms, where results include typos but are not inclusive of a large number of entirely different words. For shorter words and terms, it is likely a better option to either reduce the maximum edit distance or use a different kind of search, as many other words may be included within the fuzzy paramters that are not intended to be included in the search.

Requirement #2:

Rerun Experiments #1, #2, #3 but searching for "Child" matches

Requirement 2 Experiment 1 "Child" (Regex)

In [807…
```
query = {
        'size' : 10000,
        'query': {
            'bool': {
                    'must' : [{'match' : {'Results': 'Fail'}}, {"match" : {'Risk': {"query": 'Risk 1 (High)', "operator": "and"}} }, # same as where clasue in SQL


                                {"query_string": {
                                             "query": "*Child*",  #using regex of children  to match all posssible combinations of "Children"
                                             "fields": ["Facility Type","Violations","DBA Name"] #Multi-field matching query
                                              }
                                }


                                ]
                    }
                }
        }
results = es.search(index='food_inspections', body=query, scroll='1h')
```

In [808…
```
sid = results['_scroll_id']
scroll_size = results['hits']['total']

print('sid = ', sid)
print('Scroll Size = ', scroll_size)
```

```
sid =  DnF1ZXJ5VGhlbkZldGNoCgAAAAADTepQFmxSTjR3TFBsUXotQ2ZRQ0toYzlGQ0EAAAAAA03qUxZsUk40d0xQbFF6LUNmUUNLaGM5RkNBAAAAAANN6lIWbFJONHdMUGxRei1DZlFDS2hjOUZDQQAAAAADTepRFmxSTjR3TFBsUXot
Q2ZRQ0toYzlGQ0EAAAAAA03qVxZsUk40d0xQbFF6LUNmUUNLaGM5RkNBAAAAAANN6lQWbFJONHdMUGxRei1DZlFDS2hjOUZDQQAAAAADTepVFmxSTjR3TFBsUXotQ2ZRQ0toYzlGQ0EAAAAAA03qVhZsUk40d0xQbFF6LUNmUUNLaGM5RkN
BAAAAAANN6lgWbFJONHdMUGxRei1DZlFDS2hjOUZDQQAAAAADTepZFmxSTjR3TFBsUXotQ2ZRQ0toYzlGQ0E=
Scroll Size =  774
```

In [809…
```
len(results['hits']['hits'])
```

Out[809]:
```
774
```

In [810…
```
count = 0
list_of_lAT_LONG_pairs = []
while(scroll_size > 0):

    for inspection in results['hits']['hits']:                  #Iterating each  results of  the qurey
        current_location_lAT_LONG = []
        document = inspection['_source']
        count = count +1

        #defensive coding to ensure we have the fields in the inspection documents
        if 'Latitude' in document.keys():
            if 'Longitude' in document.keys():
```

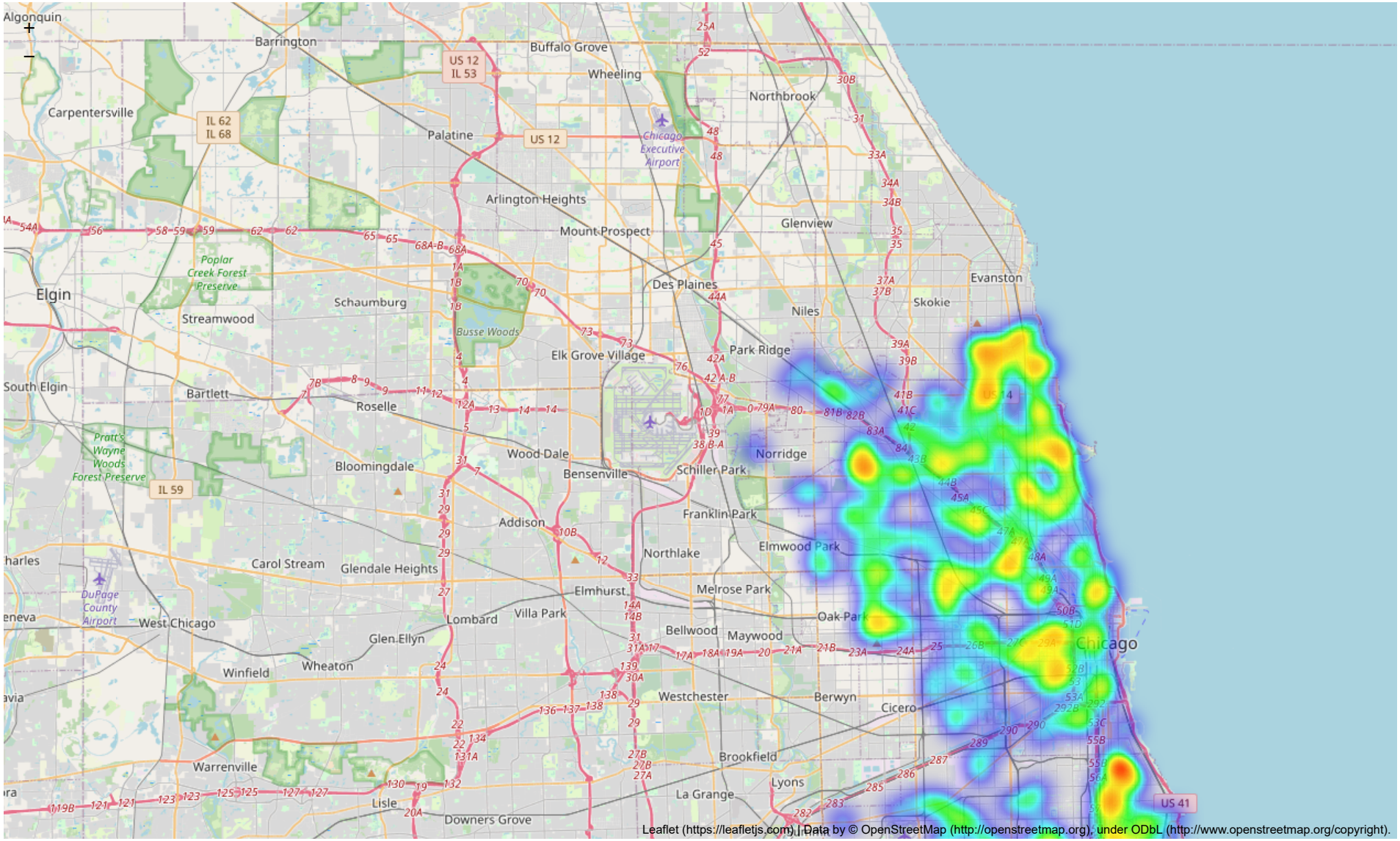
```
                if 'Address' in document.keys():
                    if(document['Latitude'] != None and document['Longitude'] != None  and document['Address'] != None):
                        current_location_lAT_LONG.append(float(document['Latitude']))    #Appending Latitude and Longitude into the list
                        current_location_lAT_LONG.append(float(document['Longitude']))
                        list_of_lAT_LONG_pairs.append(current_location_lAT_LONG)

    results = es.scroll(scroll_id = sid, scroll = '2m')
    sid = results['_scroll_id']                                 #Changing the scroll-id
    scroll_size = len(results['hits']['hits'])

print("the total number of match with child using wild card:",count)
```

```
the total number of match with child using wild card: 774
```

In [811…

```
chicago_map = folium.Map([41.90293279, -87.70769386], zoom_start=11)
chicago_map.add_child(plugins.HeatMap(list_of_lAT_LONG_pairs, radius=15))
chicago_map
```

Out[811]:



Requirement 2 Experiment 2 "Child~2" (Fuzzy)

In [812…

```
query = {
        'size' : 10000,
        'query': {
            'bool': {

                    'must' : [{'match' : {'Results': 'Fail'}}, {"match" : {'Risk': {"query": 'Risk 1 (High)', "operator": "and"}} }, # same as where clasue in SQL

                            {"query_string": {
                                        "query": "Child~2",
                                        "fields": ["Facility Type","Violations","DBA Name"]
                                         }
                            }
```

```
                            ]
                }
            }
        }
results = es.search(index='food_inspections', body=query,scroll='1h')
```

In [813…

```
sid = results['_scroll_id']
scroll_size = results['hits']['total']
```

In [814…

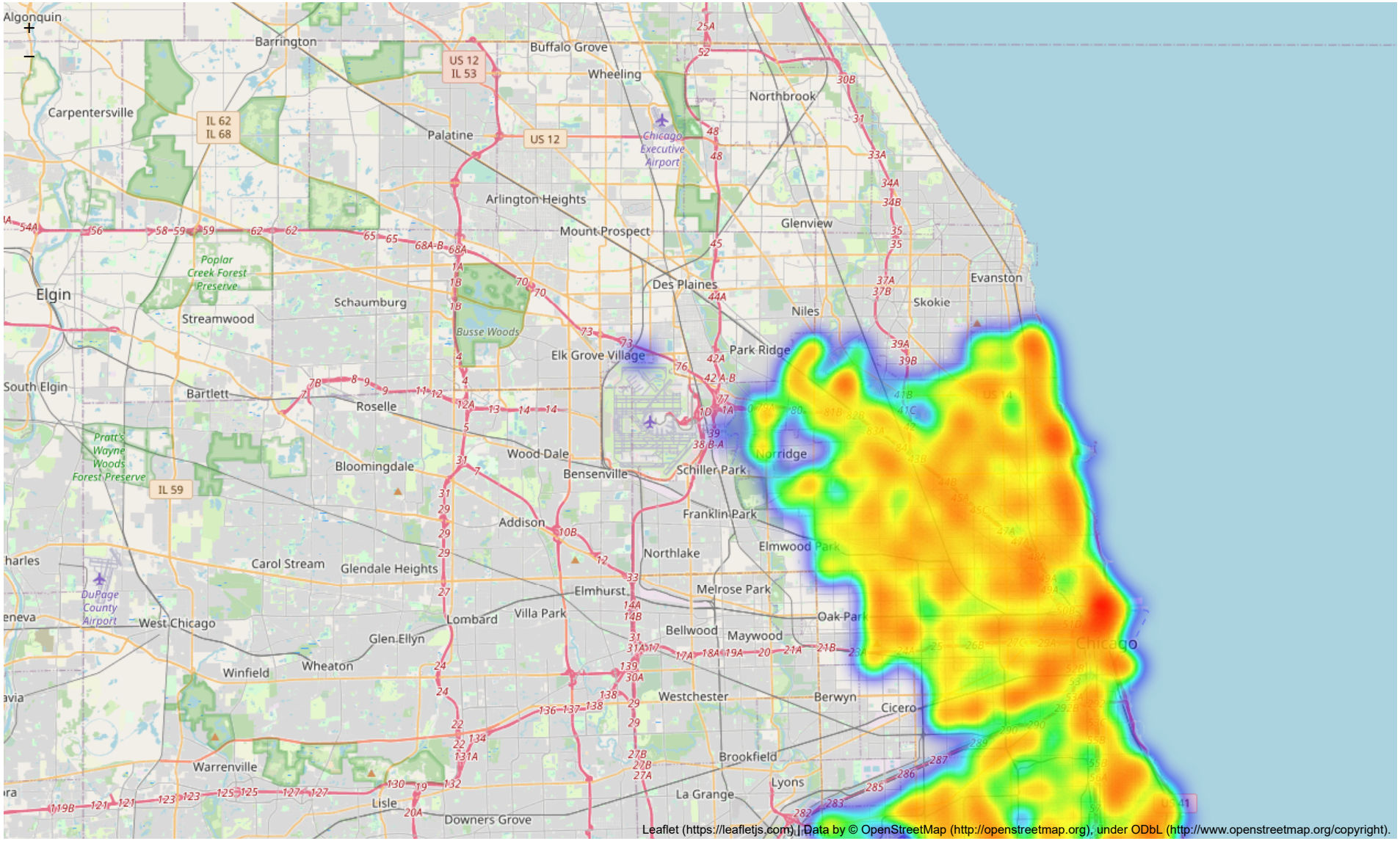
```
count = 0
list_of_lAT_LONG_pairs = []

while(scroll_size > 0):

    for inspection in results['hits']['hits']:
        current_location_lAT_LONG = []
        document = inspection['_source']
        count = count +1

        #defensive coding to ensure we have the fields in the inspection documents
        if 'Latitude' in document.keys():
            if 'Longitude' in document.keys():
                if 'Address' in document.keys():
                    if(document['Latitude'] != None and document['Longitude'] != None  and document['Address'] != None):
                        current_location_lAT_LONG.append(float(document['Latitude']))
                        current_location_lAT_LONG.append(float(document['Longitude']))
                        list_of_lAT_LONG_pairs.append(current_location_lAT_LONG)

    results = es.scroll(scroll_id = sid, scroll = '2m')
    sid = results['_scroll_id']
    scroll_size = len(results['hits']['hits'])

print("Total number of query matches with child using fuziness:",count)
```

```
Total number of query matches with child using fuziness: 10970
```

In [815…

```
chicago_map = folium.Map([41.90293279, -87.70769386], zoom_start=11)
chicago_map.add_child(plugins.HeatMap(list_of_lAT_LONG_pairs, radius=15))
chicago_map
```

Out[815]:



Requirement 2 Experiment 3 "Child's~2" (Fuzzy)

In [816…

```
query = {
        'size' : 10000,
        'query': {
            'bool': {
                    'must' : [{'match' : {'Results': 'Fail'}}, {"match" : {'Risk': {"query": 'Risk 1 (High)', "operator": "and"}} }, # same as where clasue in SQL

                            {"query_string": {
                                        "query": "Child's~2",
                                        "fields": ["Facility Type","Violations","DBA Name"]
                                         }
                            }

                            ]
```

```
                    }
                }
        }
results = es.search(index='food_inspections', body=query,scroll='1h')
```

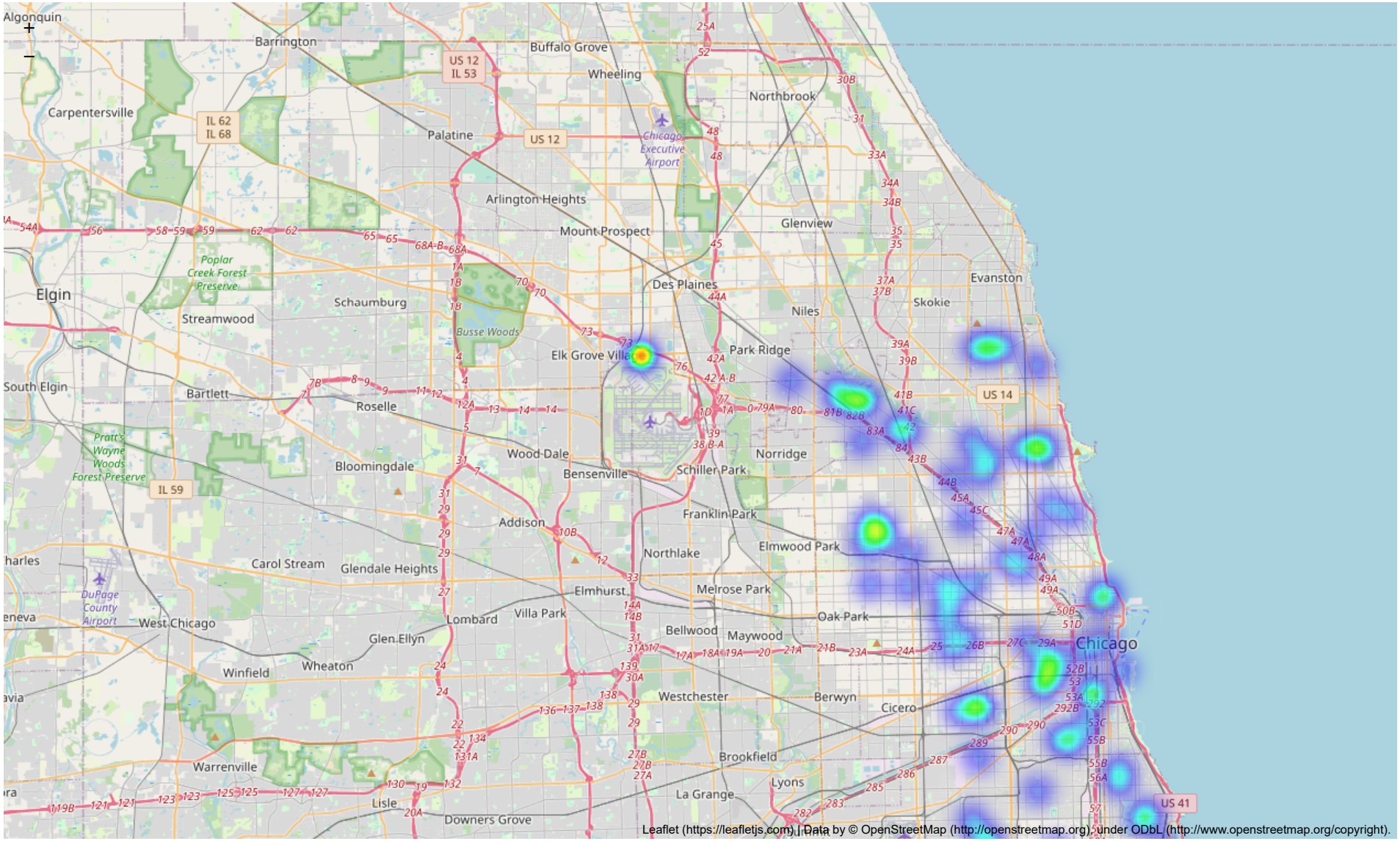
In [817…
```
sid = results['_scroll_id']
scroll_size = results['hits']['total']
```

In [818…
```
count = 0
list_of_lAT_LONG_pairs = []

while(scroll_size > 0):

    for inspection in results['hits']['hits']:
        current_location_lAT_LONG = []
        document = inspection['_source']
        count = count +1

        #defensive coding to ensure we have the fields in the inspection documents
        if 'Latitude' in document.keys():
            if 'Longitude' in document.keys():
                if 'Address' in document.keys():
                    if(document['Latitude'] != None and document['Longitude'] != None  and document['Address'] != None):
                        current_location_lAT_LONG.append(float(document['Latitude']))
                        current_location_lAT_LONG.append(float(document['Longitude']))
                        list_of_lAT_LONG_pairs.append(current_location_lAT_LONG)

    results = es.scroll(scroll_id = sid, scroll = '2m')
    sid = results['_scroll_id']
    scroll_size = len(results['hits']['hits'])

print("Total number of match with Child's using fuziness:",count)
```

```
Total number of match with Child's using fuziness: 212
```

In [819…
```
chicago_map = folium.Map([41.90293279, -87.70769386], zoom_start=11)
chicago_map.add_child(plugins.HeatMap(list_of_lAT_LONG_pairs, radius=15))
chicago_map
```

Out[819]:



Requirement #3:

In Experiment #4 we have obtained the list of frequent vilators, produce a table that shows DBA Name, number of violations and number of licenses issued for every DBA Name

In [820…

```
# Initial query from Experiment #4
query ={
      'size' : 10000,
      'query': {
          "bool" : {
              "should":[    {'match' : {'Facility Type': {"query" : 'Daycare (2 - 6 Years)',"operator":"and"}}},
                            {'match' : {'Facility Type':{"query" : 'Daycare Above and Under 2 Years',"operator": "and"}}},
                            {'match' : {'Facility Type':{"query" : 'CHILDRENS SERVICES FACILITY',"operator" : "and"}}},
                        ],
                        "minimum_should_match" : 1,
```

```
                    "filter" : [{"match" : {'Results': {"query": 'Fail', "operator": "and"}}},
                                {"match" : {'Risk': {"query": 'Risk 1 (High)', "operator": "and"}}}
                               ]

            }
      },
      "aggs" : {
          "selected_dbas" :{
                        "terms" : {
                                    "field" : "DBA Name.keyword",
                                    "min_doc_count": 5,
                                    "size" :10000

                                  },
                        "aggs": {
                            "top_dba_hits": {
                                "top_hits": {
                                "size": 10
                                }
                            }
                  }


        }
    }
}

results = es.search(index='food_inspections', body=query,scroll='1h')

row_index =0
df_top_frequent_violators = pd.DataFrame()
for dba_bucket in results["aggregations"]["selected_dbas"]["buckets"]:
    if "top_dba_hits" in dba_bucket and "hits" in dba_bucket["top_dba_hits"] and "hits" in dba_bucket["top_dba_hits"]["hits"]:
        doc_count = dba_bucket['doc_count']
        for hit in dba_bucket["top_dba_hits"]["hits"]["hits"]:
            score = hit['_score']
            if "_source" in hit:
                row_index += 1
                df_frequent_violator = pd.DataFrame(hit['_source'],index =[row_index])
                df_frequent_violator['doc_count'] = doc_count
                df_frequent_violator['score'] = score
                df_top_frequent_violators = pd.concat([df_top_frequent_violators, df_frequent_violator])   # Replace append with concat due to deprecation
                # df_top_frequent_violators = df_top_frequent_violators.append(df_frequent_violator)       # Keep to show what was replaced with concat in line above.
```

In [821…

```
# Count number of licenses per DBA Name
license_df = df_top_frequent_violators[['DBA Name','License #']]
license_df = license_df.groupby(by=['DBA Name'], as_index=False).nunique()
license_df.rename(columns={'License #':'License # Count'})

# Count number of violations per DBA Name
DBAcount_df = pd.DataFrame((df_top_frequent_violators['DBA Name'].value_counts()))
DBAcount_df = DBAcount_df.rename(columns={'DBA Name':'Violations Count'})
DBAcount_df.reset_index(inplace=True)
DBAcount_df = DBAcount_df.rename(columns={'index':'DBA Name'})

# Join DBAcount_df and license_df
dfjoin = DBAcount_df.merge(license_df, on='DBA Name', how='inner')
dfjoin
```

Out[821]:

	DBA Name	Violations Count	License #
0	BUSY BUMBLE BEE ACADEMY DAYCARE	9	3
1	BOTTLES TO BOOKS LEARNING CENTER	8	2
2	A CHILD'S WORLD EARLY LEARNING CENTER	7	2
3	AMAZING GRACE DAYCARE CENTER	7	2
4	KIDS R FIRST LEARNING ACADEMY	6	2
5	Little People's Day Care & Kindergarten, Inc.	6	2
6	LITTLE KIDS VILLAGE LEARNING	6	2
7	LINCOLN KING DAY CARE	6	1
8	THE WORLD IS YOUR'S CHILD CARE & LEARNING CENT...	6	2
9	JELLYBEAN LEARNING CENTER	6	4
10	EARLY CHILDHOOD EDUCARE CENTER	6	3
11	DISCOVERY LEARNING ACADEMY, INC.	6	2
12	COMMONWEALTH DAYCARE CENTER	6	1
13	FIRMAN COMMUNITY SERVICES	6	4
14	KENYATTA'S DAYCARE	5	3
15	THE CRYSTAL PALACE EARLY LITERACY ZONE	5	2
16	MONTESSORI ACDY. INFT/TOD. CNT	5	3
17	MOLADE' CHILD DEVELOPMENT CENTER	5	1
18	LAKE & PULASKI CHILD DEVELOPMENT CENTER	5	2
19	EZZARD CHARLES DAYCARE CENTER	5	2
20	GREATER INSTITUTE AME CHURCH	5	2
21	GRANT DAY CARE INC	5	1
22	CENTRO INFANTIL	5	3
23	ANGELS	5	2
24	ADA S MCKINLEY MAGGIE DRUMMON	5	2
25	THE EDSEL ALBERT AMMONS NURSER	5	1

Requirement #4:

Use the results of Experiment #4 to plot on the Heatmap those frequent violators who have obtained 3 business licenses or more under the same DBA Name through out the liftime of their business

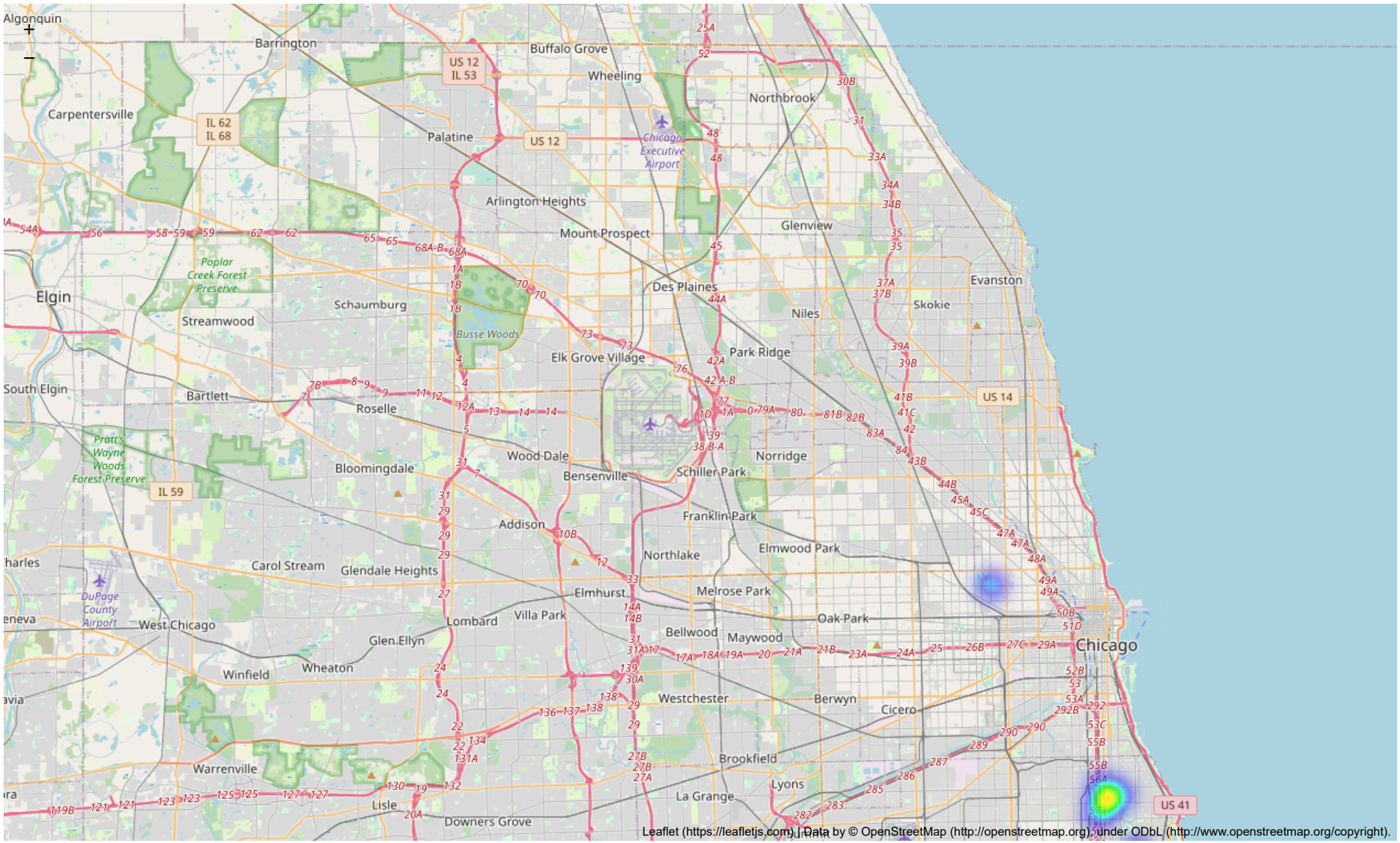
In [822…

```
# Find number of unique licenses per DBA Name.
# as_index must be False, otherwise it will be the index and cause problems down the line.
dba_license_df = df_top_frequent_violators[['DBA Name','License #']]
repeatlicense_df = dba_license_df.groupby(by=['DBA Name'], as_index=False).nunique()

# Filter df down to each DBA Name with > 3 licenses.
repeatlicense3_df = repeatlicense_df[repeatlicense_df['License #'] >= 3]

# Filter down df_top_frequent_violators to only those that are listed in repeatlicense3_df
topviolators_repeatlicense3_df = df_top_frequent_violators[df_top_frequent_violators['DBA Name'].isin(repeatlicense3_df['DBA Name'].tolist())]
```

In [823…
```
# Convert Latitude and Longitude to paired python list, so that it can be fed into Folium heatmap.
lat_long = topviolators_repeatlicense3_df[['Latitude', 'Longitude']].values.tolist()
```

In [824…
```
# Initalize map
chicago_map = folium.Map([41.90293279, -87.70769386], zoom_start=11)

chicago_map.add_child(plugins.HeatMap(lat_long , radius=15))
```

Out[824]:



Requirement #5:

Plot on the Heatmap those facilites that serve children but failed inspections with high risk, and **MICE DROPPINGS were OBSERVED** in the Violations; you have to **execlude** violations that stated **NO MICE DROPPINGS were OBSERVED**

In [825…

```
query ={
      'size' : 10000,
      'query': {
          "bool" : {
              "should":[    {'match' : {'Facility Type': {"query" : 'Daycare (2 - 6 Years)',"operator":"and"}}},
                            {'match' : {'Facility Type':{"query" : 'Daycare Above and Under 2 Years',"operator": "and"}}},
                            {'match' : {'Facility Type':{"query" : 'CHILDRENS SERVICES FACILITY',"operator" : "and"}}},
                        ],
                        "minimum_should_match" : 1,
                        "filter" : [{"match" : {'Results': {"query": 'Fail', "operator": "and"}}},
                                    {"match" : {'Risk': {"query": 'Risk 1 (High)', "operator": "and"}}}
                                   ],
              "must":[
                  {
                      "match": {
                          "Violations": {
                              "query": 'MICE DROPPINGS were OBSERVED', "operator":"and"}

                          }

                  }

              ],

              "must_not":[
                  {
                      "match": {
                          "Violations": {
                              "query": 'NO MICE DROPPINGS were OBSERVED', "operator":"and"}
                      }
                  }
              ]

             }
      },
      "aggs" : {
          "selected_dbas" :{
                          "terms" : {
                                       "field" : "DBA Name.keyword",
                                       "min_doc_count": 5,
                                       "size" :10000

                                    },
                          "aggs": {
                              "top_dba_hits": {
                                  "top_hits": {
                                  "size": 10
                                  }
                              }
                  }


         }
     }
}

results = es.search(index='food_inspections', body=query,scroll='1h')
```

In [826…

```
len(results['hits']['hits'])
```

Out[826]:

```
1
```

In [827…
```
# Lets dump the hits per bucket into a datframe object for all buckets

row_index =0
df_top_frequent_violators = pd.DataFrame()
for dba_bucket in results["aggregations"]["selected_dbas"]["buckets"]:
    if "top_dba_hits" in dba_bucket and "hits" in dba_bucket["top_dba_hits"] and "hits" in dba_bucket["top_dba_hits"]["hits"]:
        doc_count = dba_bucket['doc_count']
        for hit in dba_bucket["top_dba_hits"]["hits"]["hits"]:
            score = hit['_score']
            if "_source" in hit:
                row_index += 1
                df_frequent_violator = pd.DataFrame(hit['_source'],index =[row_index])
                df_frequent_violator['doc_count'] = doc_count
                df_frequent_violator['score'] = score
                df_top_frequent_violators = pd.concat([df_top_frequent_violators, df_frequent_violator])
                # df_top_frequent_violators = df_top_frequent_violators.append(df_frequent_violator)       # Keep to show what was replaced with concat in line above.
```

In [828…
```
df_frequent_violator
```

Out[828]:

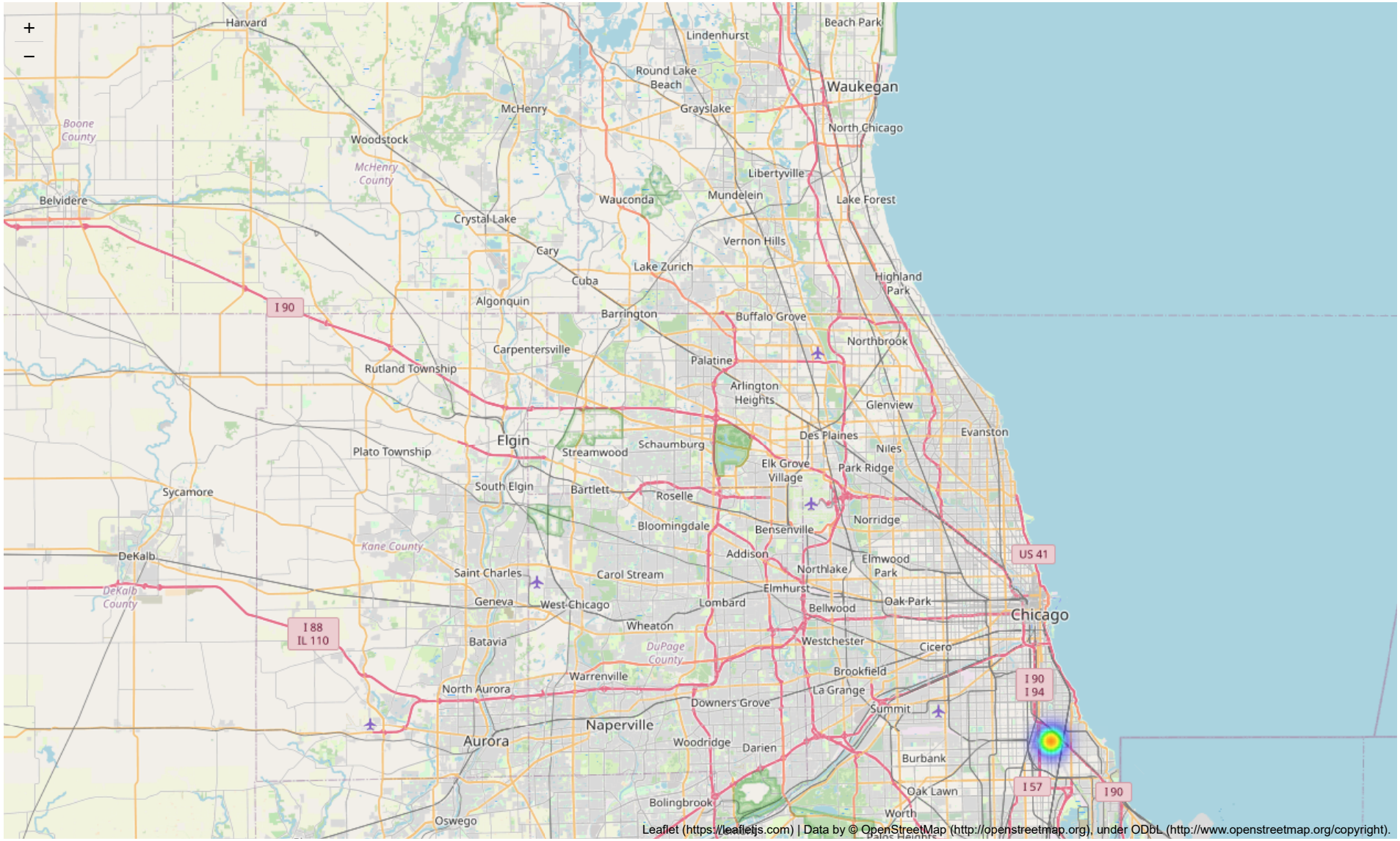
	Inspection ID	DBA Name	AKA Name	License #	Facility Type	Risk	Address	City	State	Zip	Inspection Date	Inspection Type	Results	Violations	Latitude	Longitude	Location	doc_count	score
151	1234922	THE EDSEL ALBERT AMMONS NURSER	THE EDSEL ALBERT AMMONS NURSER	15803.0	Daycare (2 - 6 Years)	Risk 1 (High)	549 E 76TH ST	CHICAGO	IL	60619.0	08/28/2012	License	Fail	18. NO EVIDENCE OF RODENT OR INSECT OUTER OPEN...	41.756551	-87.61069	(41.75655095611123, -87.61068980246957)	5	8.0246

In [829…
```
# Convert Latitude and Longitude to paired python list, so that it can be fed into Folium heatmap.
lat_long = df_frequent_violator[['Latitude', 'Longitude']].values.tolist()
lat_long
```

Out[829]:
```
[[41.7565509561, -87.6106898025]]
```

In [830…
```
# Initalize map
chicago_map = folium.Map([41.90293279, -87.70769386], zoom_start=10)

chicago_map.add_child(plugins.HeatMap(lat_long , radius=15))
```

Out[830]:


Harvard
Lindenhurst
Beach Park
Round Lake Beach
Waukegan
McHenry
Grayslake
North Chicago
Boone County
Woodstock
McHenry County
Libertyville
Belvidere
Wauconda
Mundelein
Lake Forest
Crystal Lake
Vernon Hills
Cary
Lake Zurich
Highland Park
Cuba
Algonquin
I 90
Barrington
Buffalo Grove
Northbrook
Carpentersville
Palatine
Rutland Township
Arlington Heights
Glenview
Elgin
Des Plaines
Evanston
Schaumburg
Niles
Plato Township
Streamwood
Elk Grove Village
Park Ridge
South Elgin
Bartlett
Roselle
Sycamore
Norridge
Bloomingdale
Bensenville
Kane County
DeKalb
Addison
US 41
Elmwood Park
Saint Charles
Carol Stream
Northlake
DeKalb County
Elmhurst
Geneva
West Chicago
Lombard
Bellwood
Oak Park
Chicago
I 88
IL 110
Wheaton
Batavia
Westchester
Cicero
DuPage County
Brookfield
Warrenville
I 90
I 94
North Aurora
La Grange
Downers Grove
Summit
Naperville
Aurora
Woodridge
Darien
Burbank
I 57
I 90
Oak Lawn
Bolingbrook
Worth
Oswego
Leaflet (https://leafletjs.com) | Data by © OpenStreetMap (http://openstreetmap.org), under ODbL (http://www.openstreetmap.org/copyright).

In [ ]: