Effectiveness of SF311 Service

Utilizing a Geospatial Database with ArcGIS and PostgreSQL/PostGIS to Determine the Capability of the City Services Platform

Vlad Sliusar
GEOG 574 | UNIVERSITY OF WISCONSIN-MADISON
MAY 4TH, 2018

Introduction:

San Francisco is a highly populated metropolitan city and well-known tourist destination. Millions of people from around the globe visit the city every year. It is the heart of the Silicon Valley. Unfortunately, for the past several years San Francisco has been facing a prevailing issue – the cleanliness of the city.

The cleanliness of San Francisco streets has become a very apparent issue. Certainly, this matter affects many other cities in the country. A solution to this has yet been found.

With this research, I will explore the effectiveness of San Francisco's 311 service that is employed as a platform for all local government services. Additionally, the platform accepts requests from the community via web, mobile, and phone call sources. Some of the types of requests are:

- *Street or Sidewalk Cleaning*
- *Graffiti*
- *Streetlight Repair*
- *Illegal Posting*
- *Abandoned Vehicles, etc.*

There are currently about ninety thousand open requests and seven hundred and ninety thousand closed requests in the queue. Three hundred thousand of the closed requests are the *Street or Sidewalk Cleaning* type.

My research will focus on the *Street or Sidewalk Cleaning* type as it appears to be a matter of the utmost urgency. As of this writing, there are seventeen thousand outstanding requests. The following questions will be considered in this project:

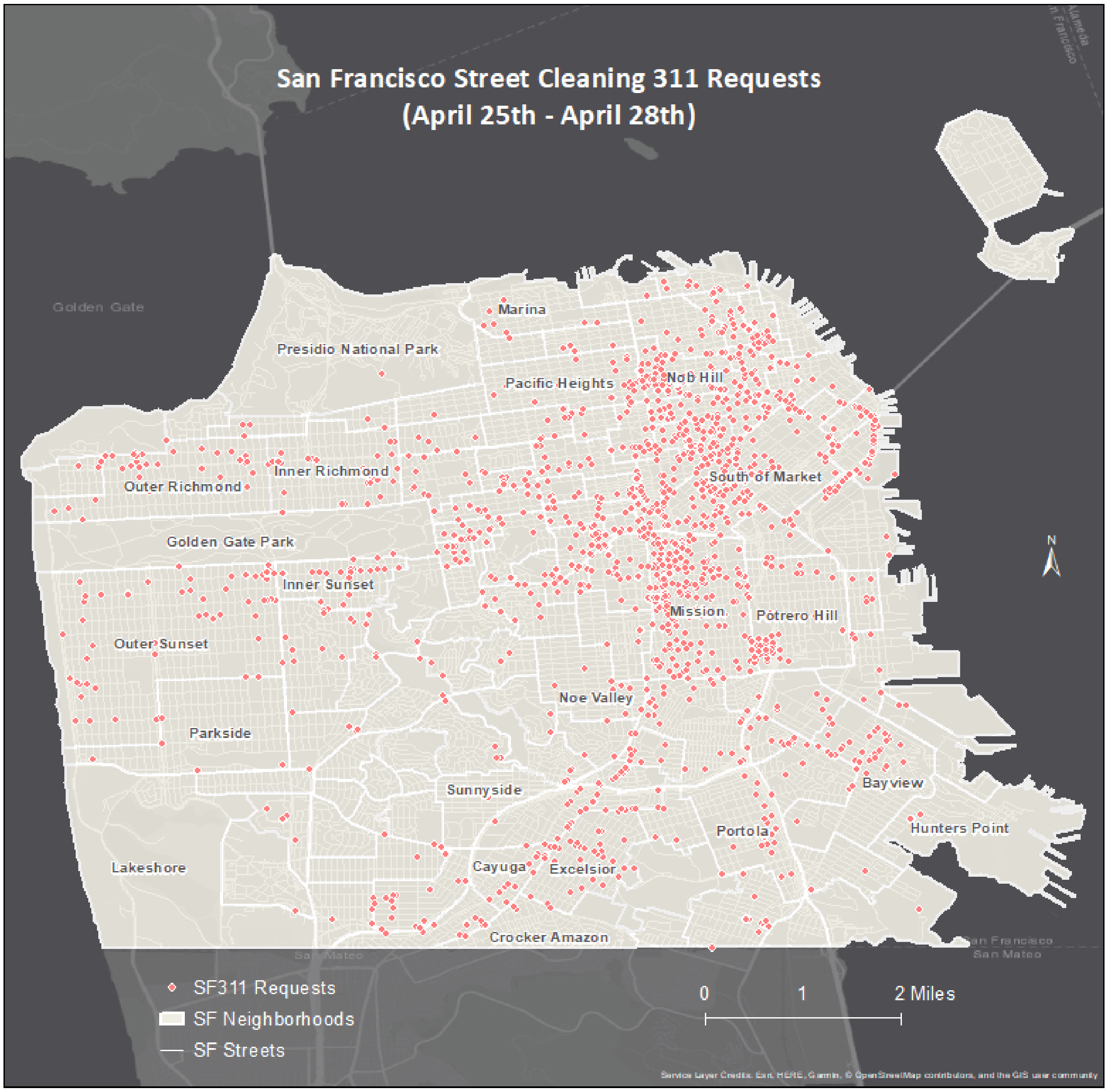
1. *Which city neighborhoods have the largest and smallest amount of street cleaning requests submitted?* This allows allocating additional resources to the areas with highest rates of requests.
2. *What is the most popular method (device) that is being utilized by the community to submit the requests?* Appropriate technical improvements can be made to the broadly used platform.
3. *How many of the requests submitted during the day are resolved within Day 1 and Day 2?* According to the SF311 service, their expected average response time is 24 hours for most of the Street or Sidewalk cleaning subtypes.

Data Collection and Storage - ArcGIS

The SF311 platform only allows to view two hundred of closed and opened recent requests. Alternatively, a one-gigabyte Excel file containing all the requests since the start of the service is available for download.

Instead, the main data for this research was collected via web scraping technic using Python programming language. I wrote a program that allowed to scrape data from www.sf311.org and ran it four consecutive days at midnight, starting from April 25th through April 28th (please see the program on **Page 12**)*. About two thousand of opened/closed requests for street cleaning were collected within the four-day period and saved in a .csv file. The shapefile *SF311_ requests* was created from the .csv file using the Display XY Data feature and exporting the data into a shapefile. Additionally, two shapefiles (*SF_streets and SF_neighborhoods*) were obtained from www.datasf.org. Lastly, the fourth shapefile was created to represent the location and other related attribute information of the SF311 services.

For storage purposes a File Geodatabase was created in ArcMap. A new Feature Dataset was created in the geodatabase with the coordinate system to be shared by feature classes. The four shapefiles (*SF311_requests, SF_neighborhoods, SF_streets, and SF311*) were converted into feature classes. To ensure data integrity, flags were set for all fields to allow/not allow the null/no values, and domains were added were necessary (e.g. street class field). A subtype was created for the status field (Opened or Closed) of the *SF_requests* feature class. The resulted map is shown below:



Database Design and Implementation – PostgreSQL/PostGIS

The conceptual design of the PostgreSQL spatial database for this project includes four entities: *requests, neighborhoods*, *streets*, and *organization*. The relationships between *requests*, *streets*, and *neighborhoods* entities are of great interest, as they will help answer the main questions of this research. Listed below is the Entity-Relationship Diagram, created via ERDPlus – the database modeling tool, representing the four entities with their associated attributes and relationships. Primary keys are shown as underlined attributes. The *organization* entity contains the composite attribute *address.* The cardinalities between the entities are many to one as *requests* must have at least one *organization* to be reported to, one *neighborhood* to be located in, and one *street* to be located on. While streets, neighborhoods, and organization may have non to many requests.

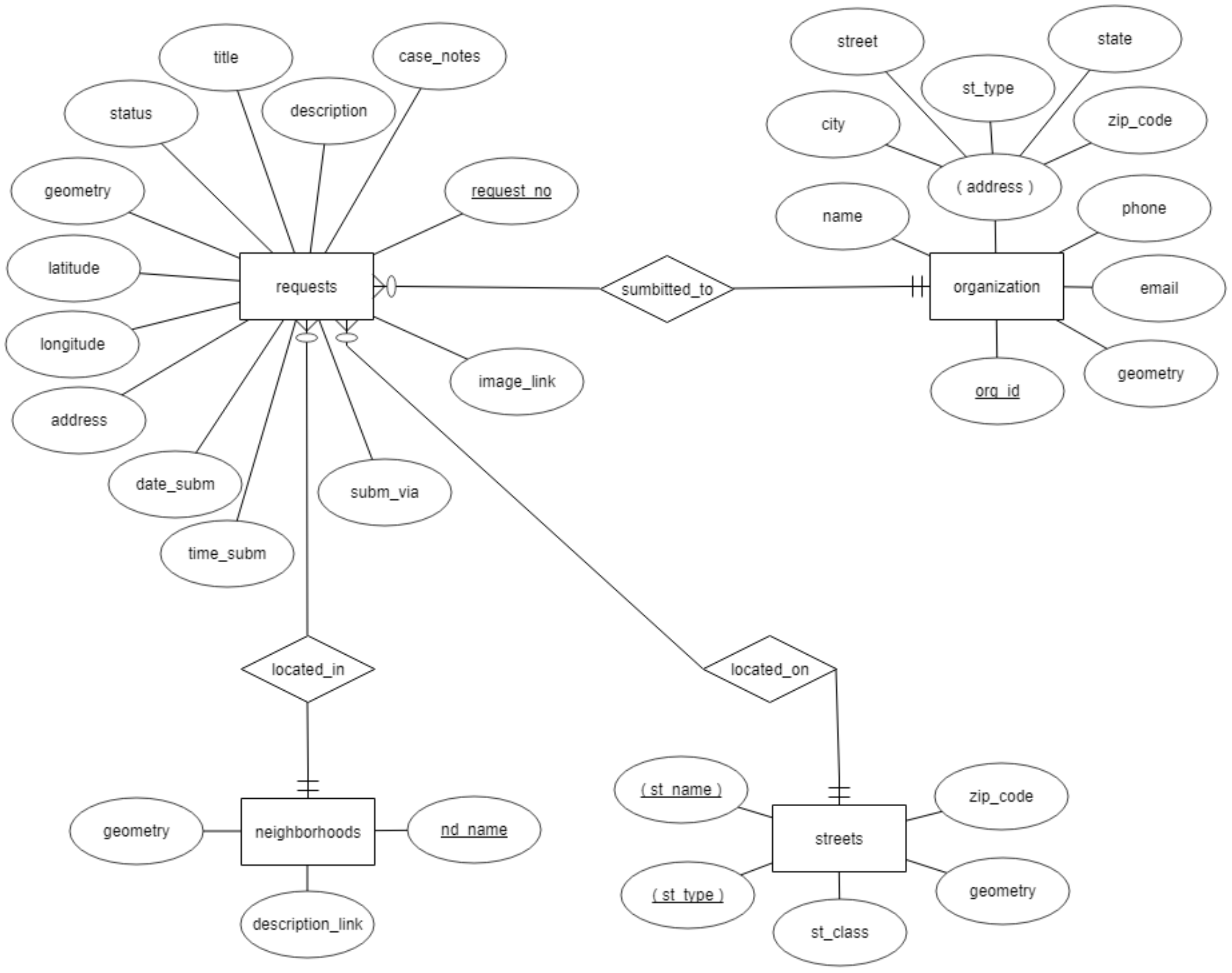


Figure 1: Spatial database conceptual design – E-R Model

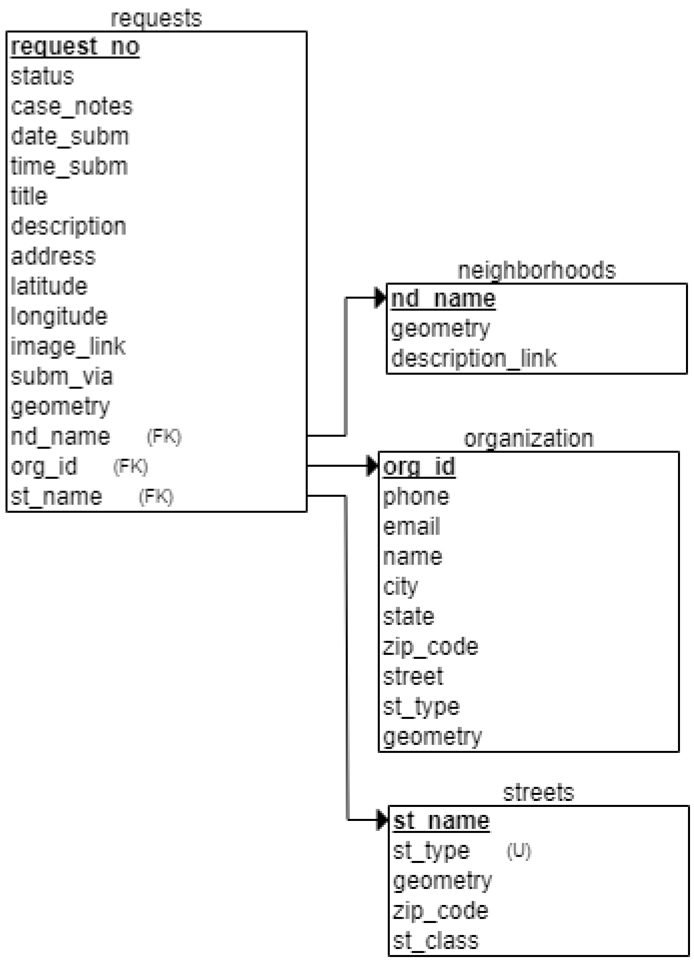


Figure 2: Spatial database logical design – Relational Model

Having finished the conceptual and logical modeling, the spatial database was created in PostgreSQL using pgAdmin III interface and PostGIS – the open-source software that adds support for geographic objects to the PostgreSQL ORDB. The database tables were populated utilizing the PostGIS shapefile loader and the appropriate SRID #4326 for WGS84(DD). Additionally, a geometry table was created for the database which allows to create spatial queries.

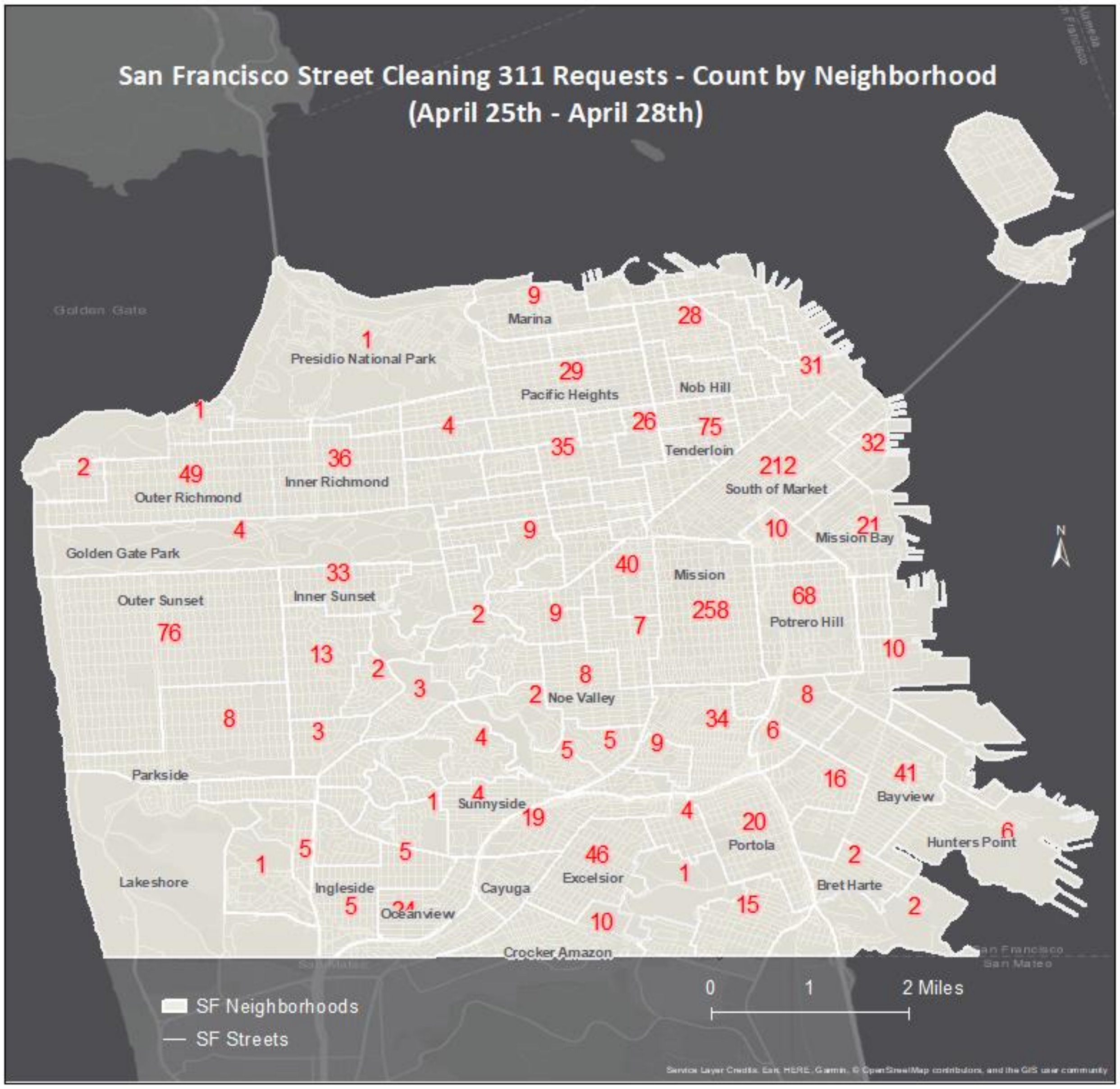
Database Manipulations and Results

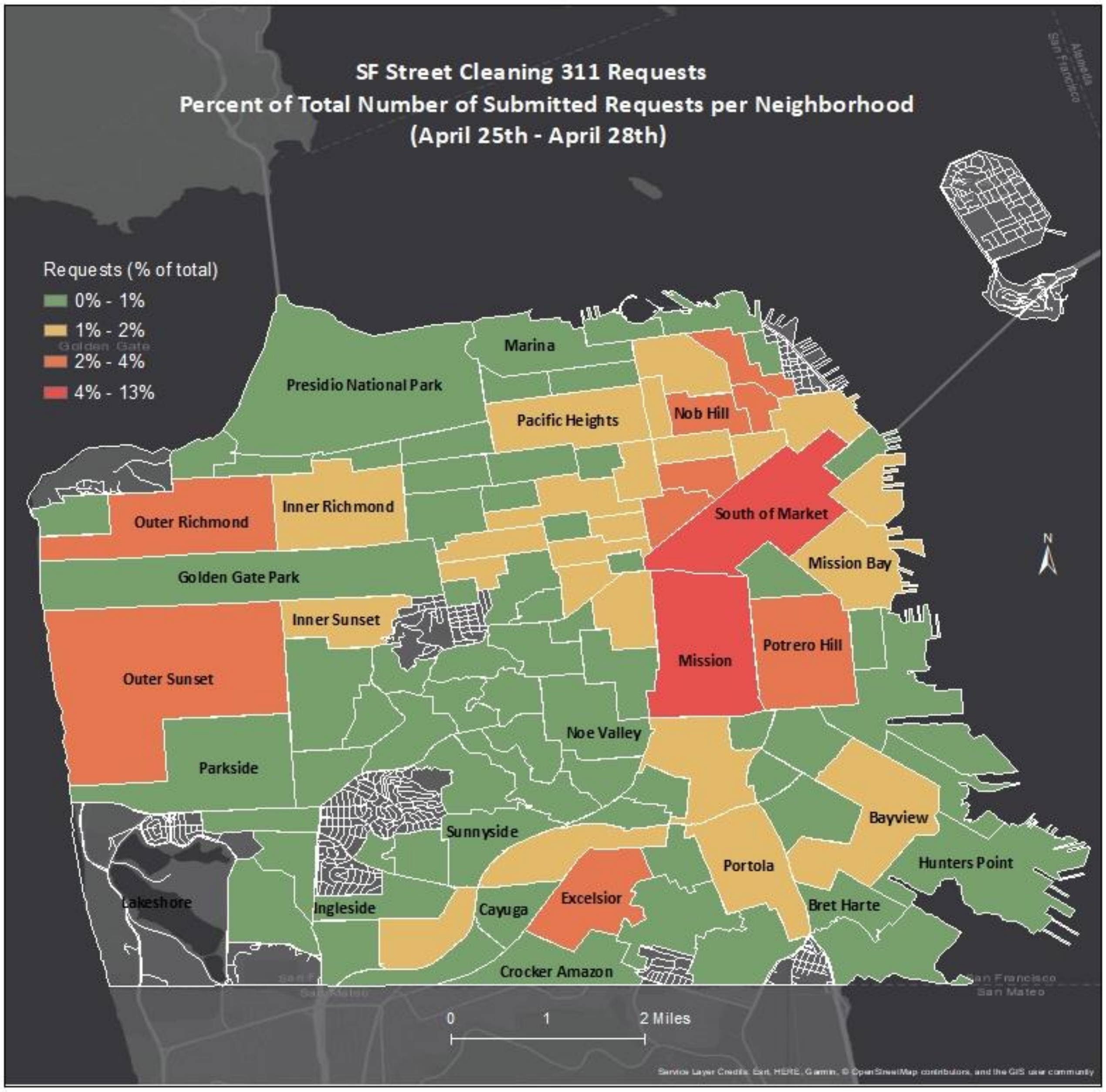
SQL is a great tool to aid in answering the standing research questions. The created PostgreSQL spatial database is manipulated to find the required information.

Answering the fist question: "*Which city neighborhoods have the largest and smallest amount of street cleaning requests submitted",* would be beneficial as it may help to allocate appropriate resources to the most affected areas. The following SQL query was utilized:

```
SELECT count(*), name
FROM requests as r, neighborhoods as n
WHERE ST_Within (r.geom, n.geom)
GROUP BY name
ORDER BY count DESC;
```

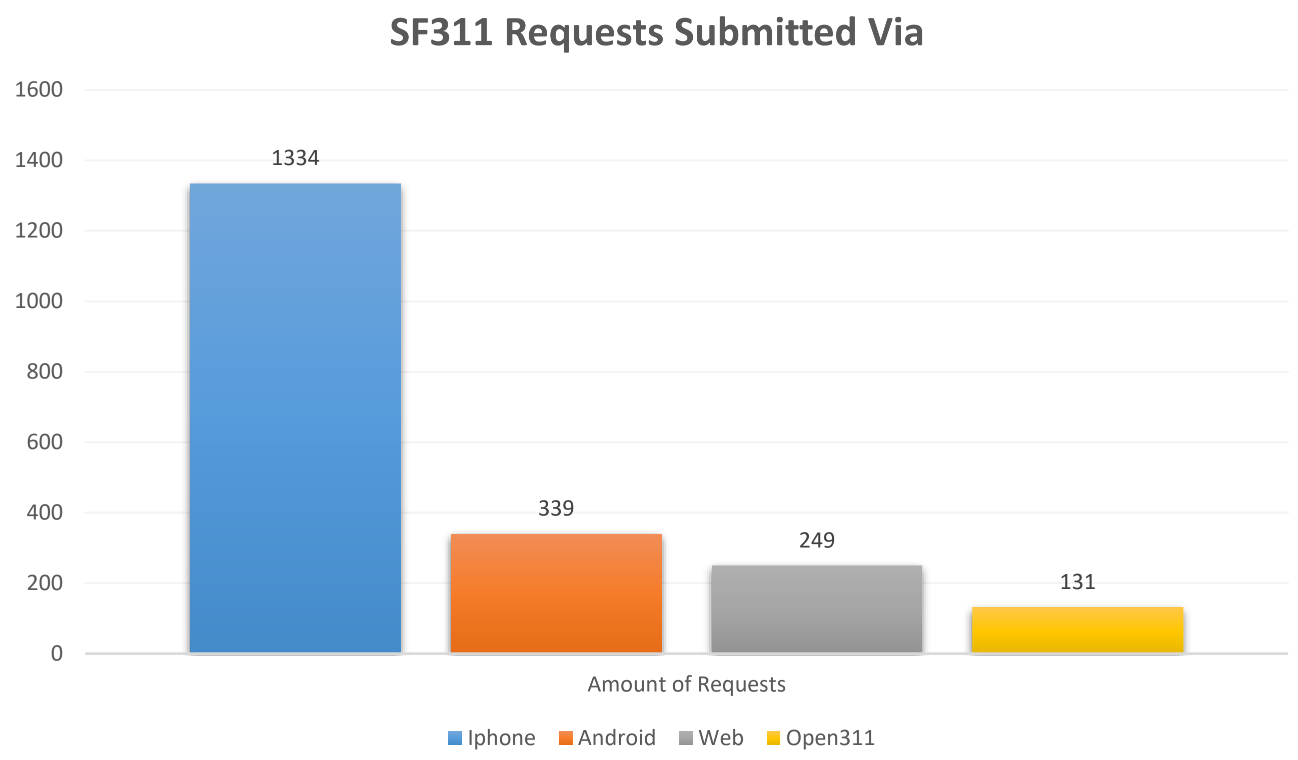
The resulting table was saved as a .csv file and joined to the *SF_neighborhoods* feature class to produce the following two maps (South of Market and Mission neighborhoods have the highest amounts of *Street or Sidewalk Cleaning* requests):





The second question that I am looking to consider is: *What is the most popular method (device) that is being utilized by the community to submit the requests?* Answering this question might direct additional resources towards improving technical aspects of the broadly used platform.

```
SELECT subm_via, count(*)
FROM requests
GROUP BY subm_via
ORDER BY count DESC;
```



The iPhone is the most used device. Therefore, the improvement of the iOS application should be of primary focus.

According to the SF311 service, the expected average response time for most of the *Street or Sidewalk Cleaning* requests is 24 hours. The third question of this research is directly related to the effectiveness of the platform:

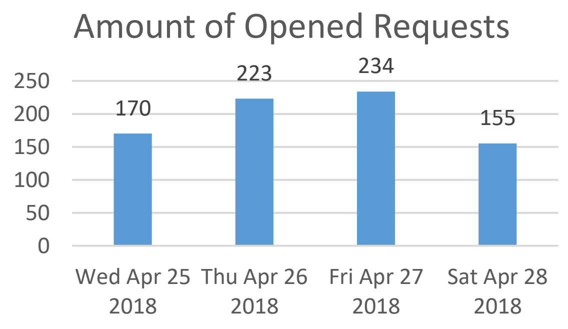
How many of the requests submitted during the day are resolved within Day 1 and Day 2?

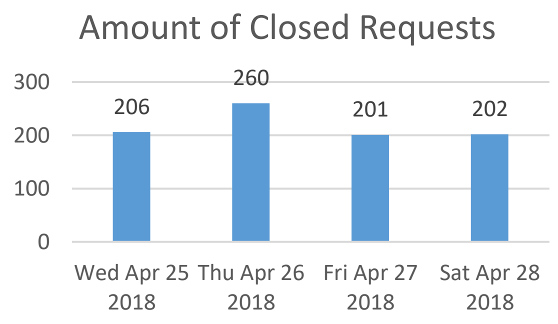
To answer this question multiple queries were run to find the data:

1. The first two queries find the amounts of Opened and Closed requests for April 25th through April 28th

a) SELECT date_subm, count(*)
FROM requests
WHERE status = 'OPENED'
GROUP BY date_subm;

b) SELECT date_subm, count(*)
FROM requests
WHERE status = 'CLOSED'
GROUP BY date_subm;





2. Knowing the amounts of opened and closed requests for April 25th through April 28th, now we can look at the number of requests that are addressed within the 24-hour and two-day periods:

    a) Day 1: number of requests submitted and closed on April 25th:

```
SELECT count(*) FROM
(SELECT request_id FROM requests
WHERE status = 'OPENED' AND date_subm = 'Wed Apr 25 2018') as a
JOIN
(SELECT request_id FROM requests
WHERE status = 'CLOSED' AND date_subm = 'Wed Apr 25 2018') as b
ON a.request_id = b.request_id
```

The result of this query is **1**. Only one request submitted on April 25th was closed on the same day.

    b) Day 2: number of requests submitted on April 25th and closed on April 26th:

```
SELECT count(*) FROM
(SELECT request_id FROM requests
WHERE status = 'OPENED' AND date_subm = 'Wed Apr 25 2018') as a
JOIN
(SELECT request_id FROM requests
WHERE status = 'CLOSED' AND date_subm = 'Thu Apr 26 2018') as b
ON a.request_id = b.request_id
```

The result of running the above query is **60** – which means that 60 *Street or Sidewalk Cleaning* requests were closed on the second day. Overall, **61** requests were completed within the two-day period, which amounts to **36%** of opened requests originated on **April 25th**.

    c) Day 1: number of requests submitted and closed on April 26th:

```
SELECT count(*) FROM
(SELECT request_id FROM requests
WHERE status = 'OPENED' AND date_subm = 'Thu Apr 26 2018') as a
JOIN
(SELECT request_id FROM requests
WHERE status = 'CLOSED' AND date_subm = 'Thu Apr 26 2018') as b
ON a.request_id = b.request_id
```

Result: **22**

d) Day 2: number of requests submitted on April 26th and closed on April 27th:

```
SELECT count(*) FROM
(SELECT request_id FROM requests
WHERE status = 'OPENED' AND date_subm = 'Thu Apr 26 2018') as a
JOIN
(SELECT request_id FROM requests
WHERE status = 'CLOSED' AND date_subm = 'Fri Apr 27 2018') as b
ON a.request_id = b.request_id
```

Result: **26**

A total of **48** street cleaning requests were closed within a two-day period starting on **April 26th**. This is only **25.5%** of the total number of opened requests submitted that day.

e) Day 1: number of requests submitted and closed on April 27th:

```
SELECT count(*) FROM
(SELECT request_id FROM requests
WHERE status = 'OPENED' AND date_subm = 'Fri Apr 27 2018') as a
JOIN
(SELECT request_id FROM requests
WHERE status = 'CLOSED' AND date_subm = 'Fri Apr 27 2018') as b
ON a.request_id = b.request_id
```

Result: **29**

f) Day 2: number of requests submitted on April 27th and closed on April 28th:

```
SELECT count(*) FROM
(SELECT request_id FROM requests
WHERE status = 'OPENED' AND date_subm = 'Fri Apr 27 2018') as a
JOIN
(SELECT request_id FROM requests
WHERE status = 'CLOSED' AND date_subm = 'Sat Apr 28 2018') as b
ON a.request_id = b.request_id
```

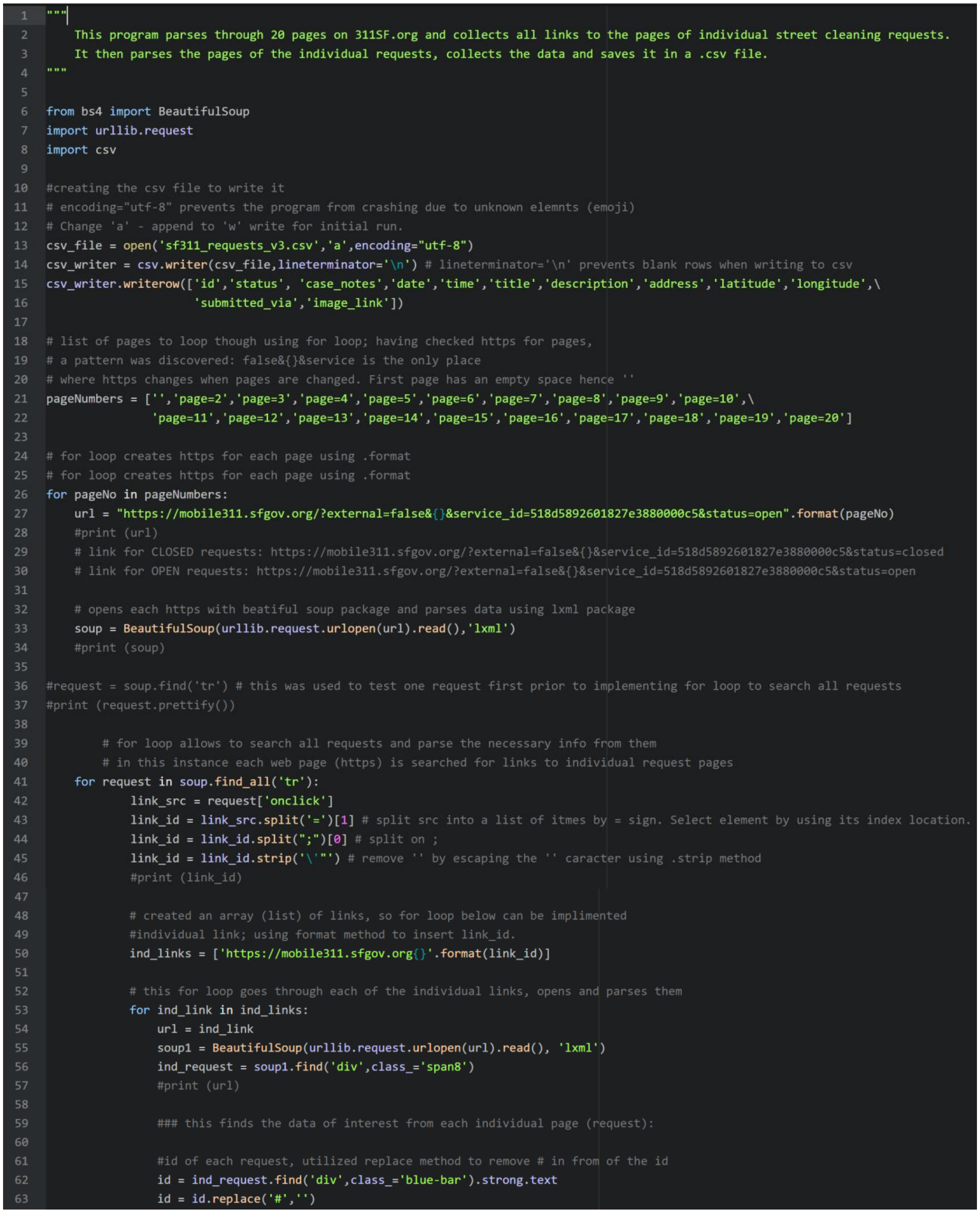
Result: **32**

A total of **61** street cleaning requests were closed within a two-day period starting on **April 27th**. This is only **26%** of the total number of opened requests submitted that day.

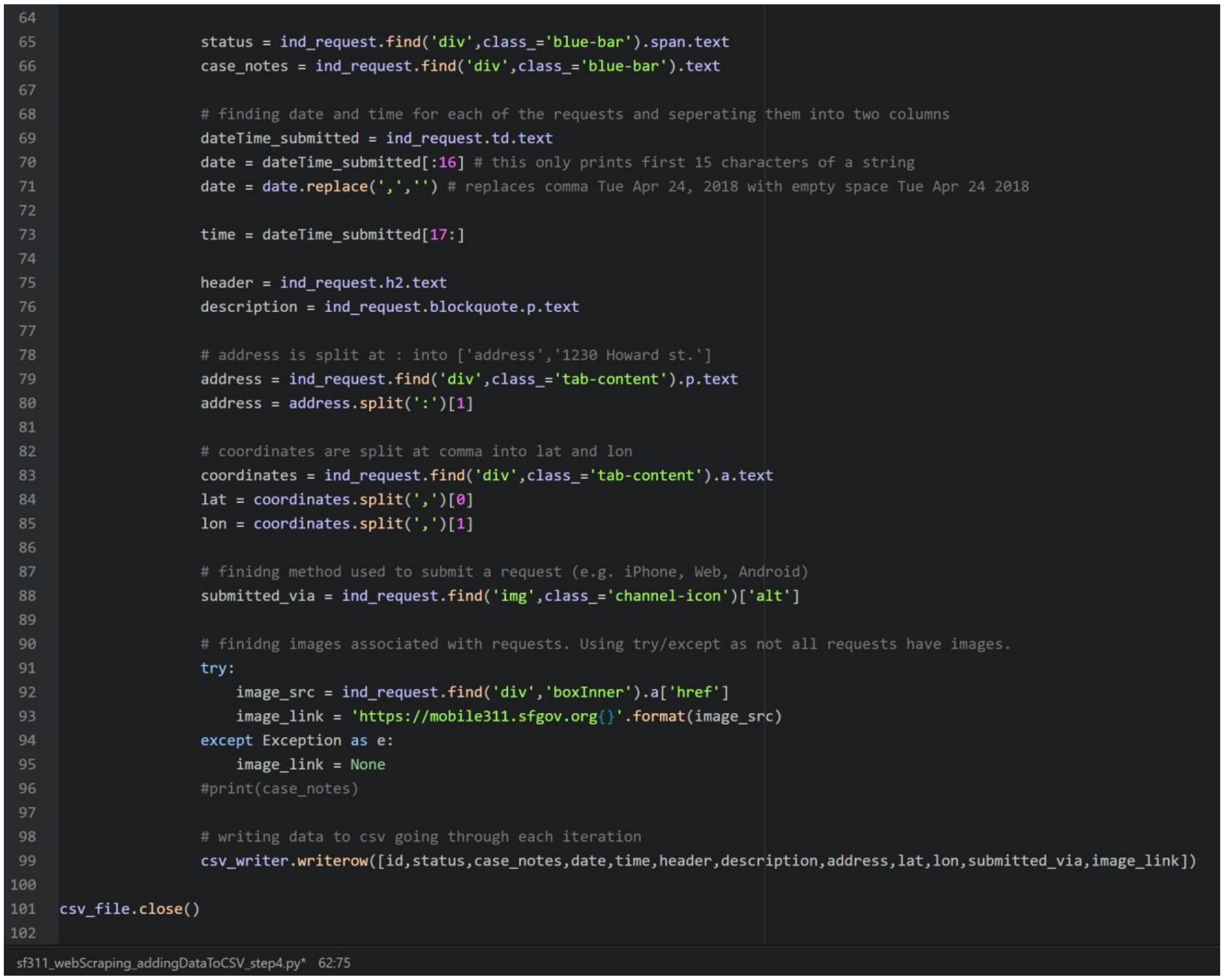
Considering the above results, on average about 29% of the *Street or Sidewalk Cleaning* requests submitted every day are addressed efficiently. This is a low number that shows that the 311SF service requires significant improvement to be able to sustain the high number of submitted requests for city cleanliness. The average number also explains the current backlog of seventeen thousand requests in the queue.

This research is only limited to four days, thus there is an expected margin of error. However, I do not anticipate the error to be significant.

Web Scraping Program Written and Utilized for this Project



```python
"""
    This program parses through 20 pages on 311SF.org and collects all links to the pages of individual street cleaning requests.
    It then parses the pages of the individual requests, collects the data and saves it in a .csv file.
"""

from bs4 import BeautifulSoup
import urllib.request
import csv

#creating the csv file to write it
# encoding="utf-8" prevents the program from crashing due to unknown elemnts (emoji)
# Change 'a' - append to 'w' write for initial run.
csv_file = open('sf311_requests_v3.csv','a',encoding="utf-8")
csv_writer = csv.writer(csv_file,lineterminator='\n') # lineterminator='\n' prevents blank rows when writing to csv
csv_writer.writerow(['id','status', 'case_notes','date','time','title','description','address','latitude','longitude',\
                     'submitted_via','image_link'])

# list of pages to loop though using for loop; having checked https for pages,
# a pattern was discovered: false&{}&service is the only place
# where https changes when pages are changed. First page has an empty space hence ''
pageNumbers = ['','page=2','page=3','page=4','page=5','page=6','page=7','page=8','page=9','page=10',\
               'page=11','page=12','page=13','page=14','page=15','page=16','page=17','page=18','page=19','page=20']

# for loop creates https for each page using .format
# for loop creates https for each page using .format
for pageNo in pageNumbers:
    url = "https://mobile311.sfgov.org/?external=false&{}&service_id=518d5892601827e3880000c5&status=open".format(pageNo)
    #print (url)
    # link for CLOSED requests: https://mobile311.sfgov.org/?external=false&{}&service_id=518d5892601827e3880000c5&status=closed
    # link for OPEN requests: https://mobile311.sfgov.org/?external=false&{}&service_id=518d5892601827e3880000c5&status=open

    # opens each https with beatiful soup package and parses data using lxml package
    soup = BeautifulSoup(urllib.request.urlopen(url).read(),'lxml')
    #print (soup)

#request = soup.find('tr') # this was used to test one request first prior to implementing for loop to search all requests
#print (request.prettify())

        # for loop allows to search all requests and parse the necessary info from them
        # in this instance each web page (https) is searched for links to individual request pages
    for request in soup.find_all('tr'):
            link_src = request['onclick']
            link_id = link_src.split('=')[1] # split src into a list of itmes by = sign. Select element by using its index location.
            link_id = link_id.split(";")[0] # split on ;
            link_id = link_id.strip('\'"') # remove '' by escaping the '' caracter using .strip method
            #print (link_id)

            # created an array (list) of links, so for loop below can be implimented
            #individual link; using format method to insert link_id.
            ind_links = ['https://mobile311.sfgov.org{}'.format(link_id)]

            # this for loop goes through each of the individual links, opens and parses them
            for ind_link in ind_links:
                url = ind_link
                soup1 = BeautifulSoup(urllib.request.urlopen(url).read(), 'lxml')
                ind_request = soup1.find('div',class_='span8')
                #print (url)

                ### this finds the data of interest from each individual page (request):

                #id of each request, utilized replace method to remove # in from of the id
                id = ind_request.find('div',class_='blue-bar').strong.text
                id = id.replace('#','')
```



```
                status = ind_request.find('div',class_='blue-bar').span.text
                case_notes = ind_request.find('div',class_='blue-bar').text

                # finding date and time for each of the requests and seperating them into two columns
                dateTime_submitted = ind_request.td.text
                date = dateTime_submitted[:16] # this only prints first 15 characters of a string
                date = date.replace(',','') # replaces comma Tue Apr 24, 2018 with empty space Tue Apr 24 2018

                time = dateTime_submitted[17:]

                header = ind_request.h2.text
                description = ind_request.blockquote.p.text

                # address is split at : into ['address','1230 Howard st.']
                address = ind_request.find('div',class_='tab-content').p.text
                address = address.split(':')[1]

                # coordinates are split at comma into lat and lon
                coordinates = ind_request.find('div',class_='tab-content').a.text
                lat = coordinates.split(',')[0]
                lon = coordinates.split(',')[1]

                # finidng method used to submit a request (e.g. iPhone, Web, Android)
                submitted_via = ind_request.find('img',class_='channel-icon')['alt']

                # finidng images associated with requests. Using try/except as not all requests have images.
                try:
                    image_src = ind_request.find('div','boxInner').a['href']
                    image_link = 'https://mobile311.sfgov.org{}'.format(image_src)
                except Exception as e:
                    image_link = None
                #print(case_notes)

                # writing data to csv going through each iteration
                csv_writer.writerow([id,status,case_notes,date,time,header,description,address,lat,lon,submitted_via,image_link])

csv_file.close()

```

sf311_webScraping_addingDataToCSV_step4.py* 62:75

*The collected data is used for education purposes and is similar to readily available data for download.

References

“Home - SF311.” *Home - SF311*, sf311.org/.