

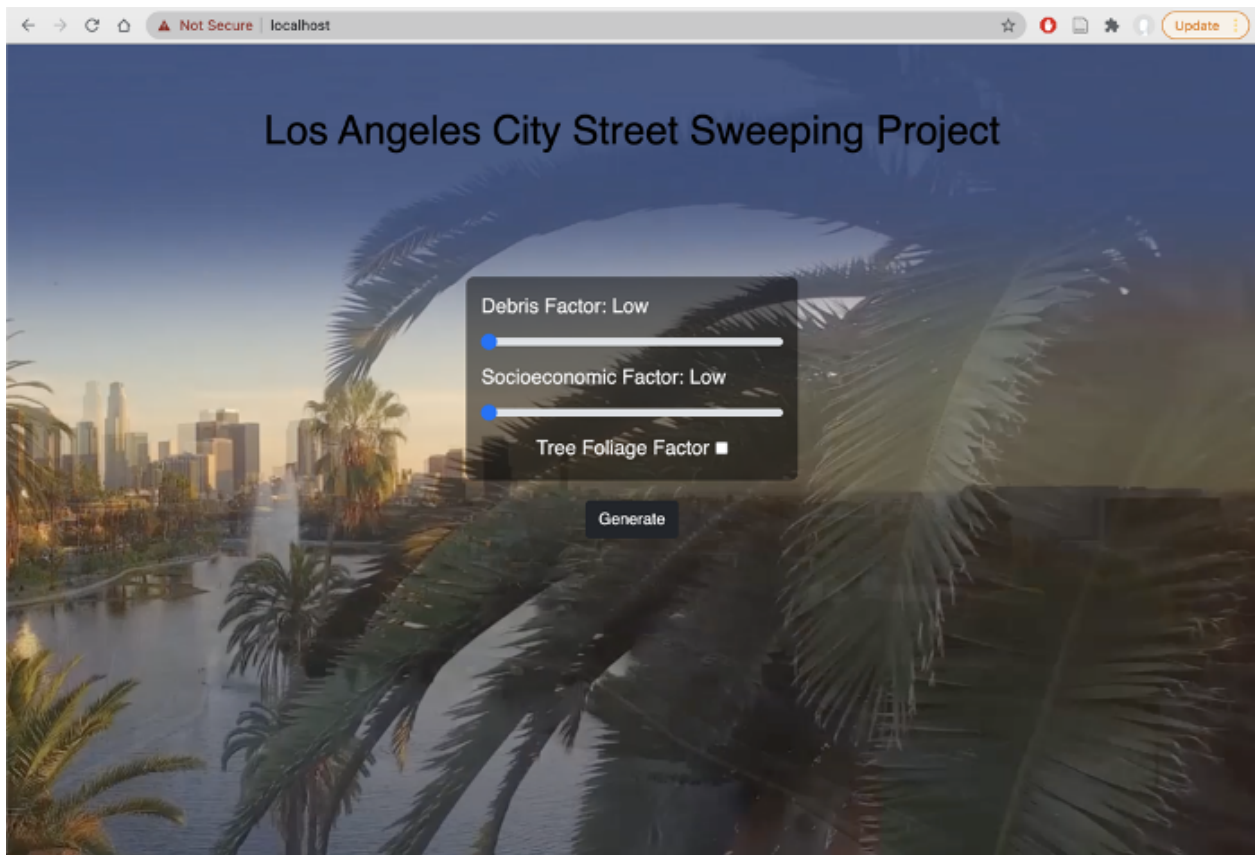
LA Equitable Street Sweeping Final Documentation

Mary Abgarian, Avery Botansky, Yvette Lopez, Jason LaRue

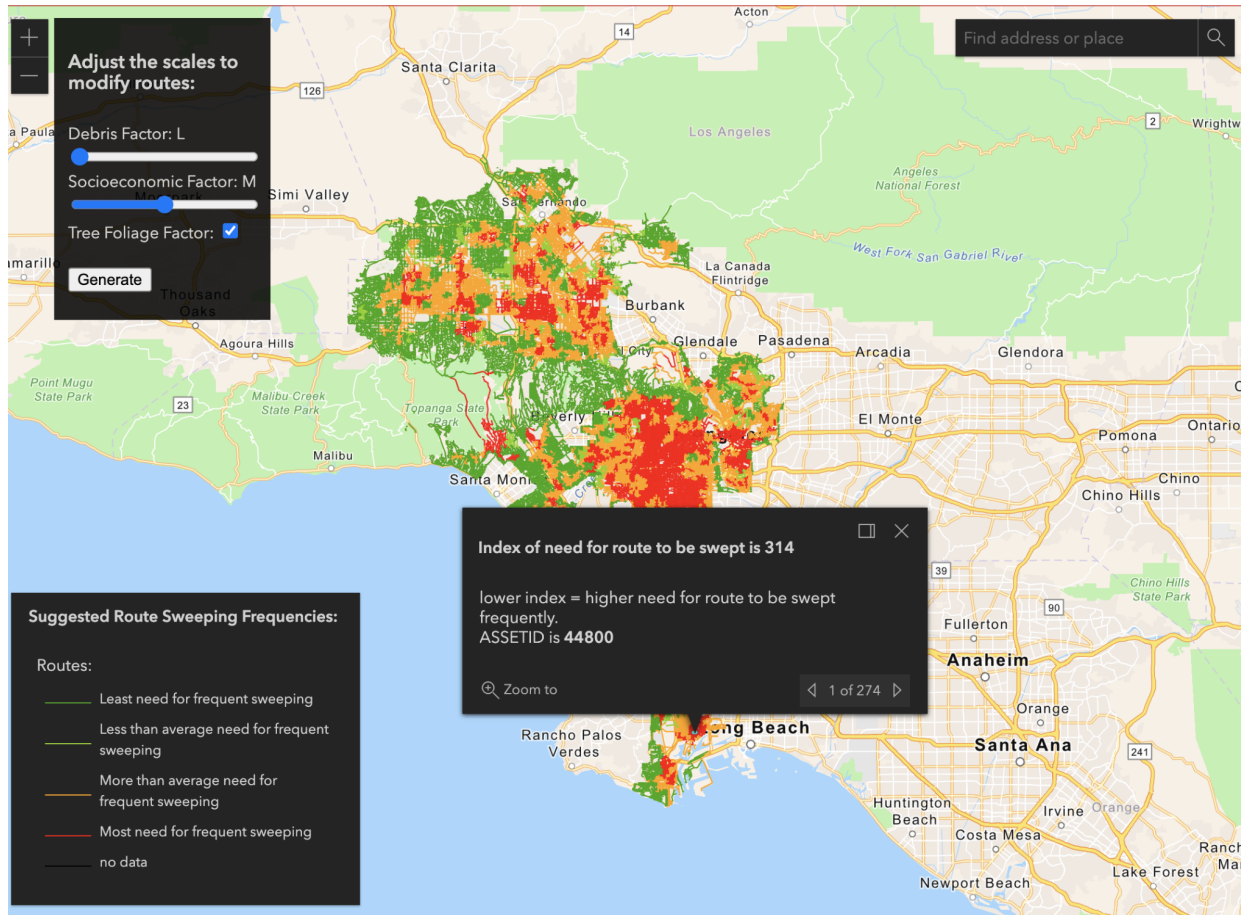
Project Github: https://github.com/MMariam/401_street_sweeping

Usage Documentation

1. To load the front page of the web app, type “<https://localhost>” into the browser url box.



2. Choose the values for Debris, Socioeconomic, and Tree Foliage factors (“low”, “medium”, “high”, and checked/unchecked)
3. Click “Generate”
4. You will be redirected to <https://localhost/sweepingneed.php> shown below:



A map and colored routes will slowly load over the next 5-6 seconds. A legend on the bottom-left corner will explain the significance of the colors chosen. 25% of the lowest values (representing highest need for sweeping) are colored in red, followed by the next 25% (more than average need for sweeping) colored orange, followed by next 25% (less than average need for frequent sweeping) colored by light green, followed by the next 25% (least need for frequent sweeping) colored dark green.

The search box in the top-right corner will allow for the user to enter any location by its name, street address or zip code to zoom in closer to that area. The zoom in/out +/- button will allow for the user to focus in/out of an area.

Should the user need to generate another recommendation based on different factor values, they can do so by adjusting the scales of the factors and clicking “Generate” in the upper right corner box. A new layer will load with different color values.

Likewise, should the user need to display individual numeric values per route, they can click on a segment, which will reveal it’s numeric standing in the overall need for more frequent sweeping.

Data Analysis

***This analysis was done using the standard output from the output files*

***The csv containing the analysis is in the github titled DataAnalysis.csv*

Of the 88,903 segments that we used in the algorithm, 18,783 segments were deemed highest priority (RED) and 20,743 were the next highest priority (ORANGE). Of the 18,783 high priority segments, 3610 of them were not originally in routes. That is around **20%** of the high priority (RED) segments. Of the 20,743 next priority segments, 8558 were not originally in routes. This is around **40%** of second highest priority (ORANGE) segments.

Technical Documentation

Route Generation

Running the C++ Program:

The user must have the following files to run the C++ program:

- RouteGenerator.cpp
- CSVReader.h

There are no additional command line arguments needed other than compiling then running the program using a C++ compiler.

Necessary Files:

- DebrisData.csv
- CleanStat.csv
- TreeScore.csv
- MHI.csv (MHI stands for median household income)
- cleanstreetrouteslatlong.csv

The route generator program is very particular about the files that are provided to it and the format in which those files are stored. Any miscellaneous data provided in the CSV files must **not** include commas, because this will cause the CSV reader to read in the provided data incorrectly. Be especially careful when street names are included as data, because the names occasionally include commas and that will break the CSV reader.

There are 5 major CSV files that need to be provided to the program. The column names provided must exactly match the column names in the CSV in order for the program to recognize them.

File 1: DebrisData.csv

Adapted from: [SweepingRouteData](#)

Column Name	Data
StrippedRoute	Contains all of the route numbers without the day of the week
miles	The length of the route, in miles
iMonth	The month in which a route is swept
Sum	The total amount of debris swept in the 5 weeks of a month
NumWeeks	The number of weeks that contributed to the sum

File 2: CleanStat.csv

Adapted from: [Clean Streets Index Segments 2020 Q4](#)

Column Name	Data
SegmentID	The AssetID of a segment
CS_RoundScore	The RoundScore of the CleanStat data

File 3: TreeScore.csv

Adapted from: [Aggregation of Street Trees & Street Sweeping Service Requests](#)

Column Name	Data
ASSETID	The AssetID of a segment
COUNT_T	The number of trees on the segment

File 4: MHI.csv

Adapted from: [Centerlines wTractID Cleaned](#)

Column Name	Data
ASSETID	The AssetID of a segment
AvgMHI	The average median household income of houses on the segment

File 5: cleanstreetrouteslatlong.csvAdapted from: [Centerlines Centroid Routes v2](#)

Column Name	Data
ASSETID	The AssetID of a segment
Shape_Length	The length of a segment relative to other segments
ZIP_R	The zip code of the right side of the segment
StrippedRoute	The route a segment belongs to, without the day of the week
Lat	The latitude of a segment's top right corner
Long	The longitude of a segment's top right corner

Input Values:

If you scroll to the bottom of “RouteGenerator.cpp” to the “main” function, you will see a section of code that stores the values for the user to modify. Included in these values are as follows:

Filenames:

For input files, please include the full file path of the CSV files being used as an input for the program. Sample file paths are provided as a reference.

Differentiators:

These are percentages used to compare segments to each other. The default values are .45 for “low” and .8 for “high”. This means that a segment's value that is in the lowest 45% of all values will be given a normalized score of 1, a value that is above 45% but below 80% will be given a normalized score of 2, and values above 80% will be given a score of 3. Modifying these values will change what the program determines as “low” and “high” when calculating the need scores.

Bias Values:

A segment's need score is the sum of its normalized scores multiplied by bias values. Mathematically, the equation is as follows:

$$\text{Segment Normalized Score} = (\text{debrisScore} * \text{debrisBias}) + (\text{incomeScore} * \text{incomeBias}) + (\text{treeScore} * \text{treeBias}) + \text{cleanStatScore}$$

The default bias scores are .5 and 1.5 which respectively decrease or increase a normalized score. The user has the option to add custom bias scores in the main function.

Output:

The program outputs 18 csv files containing routes with all possible permutations of bias scores and some corresponding segment information. However, if the user so chooses, the program can also output a single CSV file with a route that uses custom bias scores. These CSV files then need to be converted to GeoJSON files to be used with the UI.

The Algorithm for Route Generation

Step 1: Load in Street Segments and All Relevant Values

Assuming that the user input all filenames correctly and the files follow the correct formatting, the program will load all of the files and store all 88,000 segments and the relevant data. Segments will include the amount of debris found, the median household income, the cleanstat score, and the amount of trees on the segment when this data is available. These scores are then normalized into values of 1 through 3, with 1 being the lowest priority and 3 being the highest. How these values are normalized is determined by the percentages the user provides to act as differentiators (see input values for more info).

Step 2: Partition Los Angeles into Equal-Sized Boxes

Creating routes directly from segments proved to be difficult and computationally costly. For this reason, all segments in Los Angeles were partitioned into “boxes” holding 10 or 11 segments each. In order to partition these boxes, the following algorithm was used:

While there exists a box with more than 12 segments:

- Split that box in half

- Half of the segments are stored in the original box

- The other half are stored in the new box

Repeat until all boxes have less than 12 segments

A visual representation of this algorithm is below:

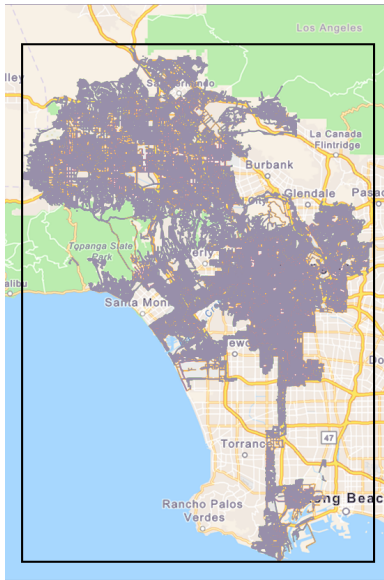


Figure 1: First box around Los Angeles is created using the highest and lowest latitudes and longitudes of segments

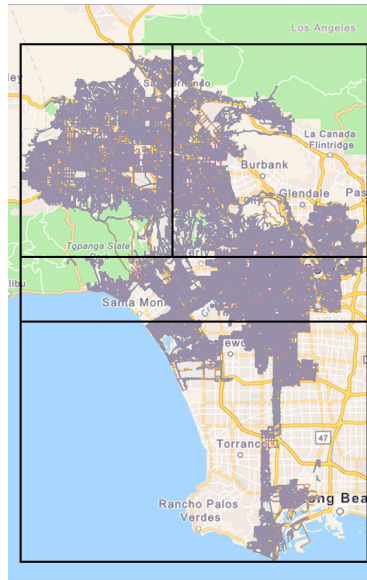


Figure 2: The program does its first three splits, creating 4 boxes containing roughly equal numbers of segments

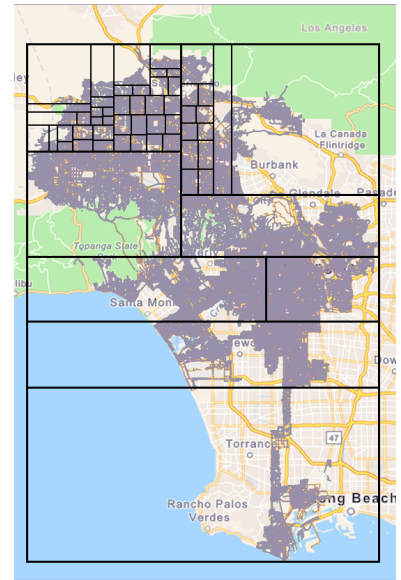


Figure 3: The program continues to split boxes into smaller sections until each box contains less than 12 segments each

Step 3: Determine the Need Score of each Box

Once boxes are partitioned, a box's "need score" is calculated by taking the average need score of all of the segments in the box. This allows the route creation algorithm to be run on approximately 8,100 boxes as opposed to 88,000 segments.

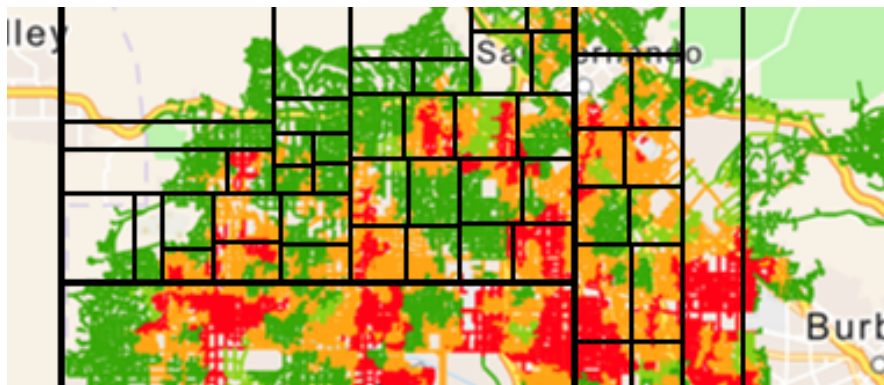


Figure 4: A rough diagram of need scores being applied to specific boxes. Note that the boxes that the actual algorithm uses are much smaller and more plentiful.

Step 4: Create Routes from Boxes

Started with the box that has the highest need value, the program does the following:

1. Find the neighboring boxes
2. Add the neighbor with the highest need value to the route
3. Add any new neighbors to the list of possible neighbors
4. Continue until max route size reached or no possible neighbors exist

After the initial routes are created, there will be some “islands” that do not contain many boxes. These “island” routes, routes containing less than 5 boxes, are then added to the closest routes that have the most similar need score. The resulting routes have at least 5 boxes with all similar need scores.

Converting the CSV Files to a GEOJSON

The algorithm generates a CSV as an output, however this output is unable to be plotted by the web application as it is missing the geolocation data (latitude and longitude). Additionally, the library used for plotting in the web application can only plot points in a CSV format. Because we need to plot street segments (lines and occasionally more complex shapes) we need to use the geojson format instead.

The first step in the conversion process is creating a dictionary, where the keys are the AssetID for every street in LA, and the value is the geographic coordinates that make up the shape of that street (a pair of latitude and longitude for straight streets, with additional latitudes and longitudes for more complex street shapes). A dictionary is used as it provides the fastest lookup times, which is crucial as there are 80,000 street segments in LA which need to be converted 18 times for all possible input parameter combinations. The python script `createdict` is used, which parses through `data.kml` which is obtained from the City of LA's geohub website, by downloading the Streets (Centerline) dataset. It will generate `dict.txt`, which is fed into the next python script.

`Parselatlong.py` does the majority of the work in the conversion. As inputs, it requires `dict.txt` (explained in previous paragraph) and also requires the 18 CSV files that are generated by the routing algorithm. All of these required files should be placed in the same directory as the python script. The script then parses through each CSV, and for every street segment it retrieves the geo data from `dict.txt`, then appends it to the existing data in the CSV (route, asset ID, need to be swept, etc.) All of this is then exported in a geojson format, which is then passed to the web application.

Instructions for use of python algorithm:

1. Download the Streets (Centerline) dataset as a kml, rename it data.kml and place it in the same directory as createdict.py
2. Run createdict.py, you should see dict.txt as a result.
3. Place dict.txt along with the 18 CSVs generated by the routing algorithm in the same directory as parselatlong.py, then run.
4. You should see 18 files with the same names as the CSVs, but with a .geojson extension instead. Provide these files to the web application to see them plotted.

Instructions for Setting Up the Web Application

Download files from github repository

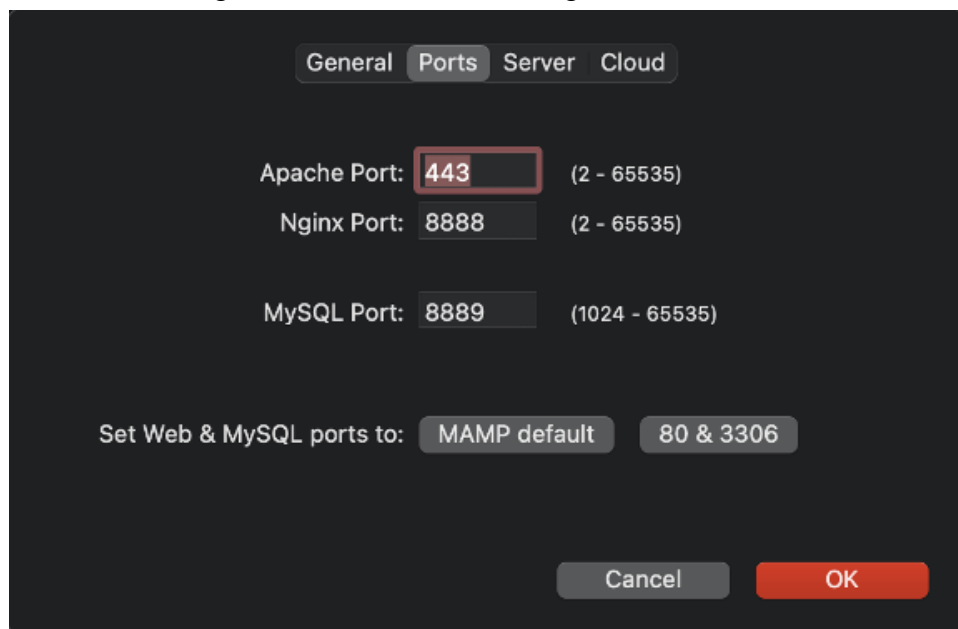
- Clone [github repository](#)
- Navigate to src/resources/data directory and follow the directions in the “download.txt” file to download necessary files

Download mamp 6.3 to set up the local web server Apache

- [Download here](#)

In order to set up an Apache local server to host the webapp, download and install Mamp, and change the root folder to the folder containing the “index.php”. The php version used was 7.4.12.

The Port settings should be the following:



The screenshot shows the 'Ports' tab of the MAMP configuration window. It features four input fields for port numbers: 'Apache Port' (443), 'Nginx Port' (8888), and 'MySQL Port' (8889). Each field has a range of available ports shown to its right: (2 - 65535) for Apache and Nginx, and (1024 - 65535) for MySQL. At the bottom, there is a section 'Set Web & MySQL ports to:' with two buttons: 'MAMP default' and '80 & 3306'. The 'OK' button is highlighted in red, and the 'Cancel' button is in gray.

Service	Port	Range
Apache	443	(2 - 65535)
Nginx	8888	(2 - 65535)
MySQL	8889	(1024 - 65535)

Set Web & MySQL ports to: MAMP default 80 & 3306

Cancel OK

Set up SSL in order to use local files

In order to be able to use downloaded geojson files without CORS (Cross-Origin Resource Sharing) error, set up SSL on the Apache server by following below steps:

source: [Add SSL on localhost \(MacOS\)](#)

1. Generate a RSA-2048 key and save it to a file rootCA.key (note the pass phrase which you set on prompt)

```
openssl genrsa -des3 -out rootCA.key 2048
```

2. Now use the above key to generate the root SSL certificate. (In snippet below, certificate will expire after 100 days. You can change it as per requirement.)

```
openssl req -x509 -new -nodes -key rootCA.key -sha256 -days 100 -out rootCA.pem
```

3. Open Keychain Access. Click on System => Certificates, now click "Shift + Cmd + I" and import the rootCA.pem (which we created in above step). After that double click on imported certificate and expand the Trust section, and then set "When using this certificate" to "Always Trust". Refer screenshot below

4. Create a certificate key (server.key)

```
openssl req -new -sha256 -nodes -out server.csr -newkey rsa:2048 -keyout server.key
```

And enter all the data as prompted. For example:

```
Generating a 2048 bit RSA private key
writing new private key to 'server.key'
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
Country Name (2 letter code) []:US
State or Province Name (full name) []:CA
Locality Name (eg, city) []:Los Angeles
Organization Name (eg, company) []:USC
Organizational Unit Name (eg, section) []:Viterbi
Common Name (eg, fully qualified host name) []:localhost
```

Email Address []: abgarian@usc.com

5. Create a certificate file (server.crt)

```
openssl x509 -req -in server.csr -CA rootCA.pem -CAkey rootCA.key -CAcreateserial  
-out server.crt -days 500 -sha256
```

Output would be,

```
Signature ok  
subject=/C=IN/ST=MH/L=Localhost Mac World/O=Local Mac/OU=Web  
Department/CN=localhost/emailAddress=example@example.com  
Getting CA Private Key  
Enter pass phrase for rootCA.key:
```

6. Search for httpd.conf file from Mamp folder and add following code at the end of file. (Configure the correct paths for DocumentRoot, SSLCertificateFile and SSLCertificateKeyFile)

Custom code for SSL

Listen 443

```
<VirtualHost *:443>  
    DocumentRoot <document_root_location>  
    ServerName localhost  
    SSLEngine on  
    SSLCertificateFile "<document_root_location>/server.crt"  
    SSLCertificateKeyFile "<document_root_location>/server.key"  
</VirtualHost>
```

Descriptions of Web Application Files:

index.php:

This page should be accessed by typing <https://localhost> into the browser url box. In this front page of the webapp, there are options to select Debris Factor (low, medium, high), Socioeconomic Factor (low, medium, high), and the Tree Foliage Factor (on/off). It is not required to change any of these inputs in order to generate a map, as these settings are valid options to choose. The values of the options range from 0-2 and each step is interpreted as either a “low”, “medium”, or “high” value. The Foliage factor value is either “on” or 0. These values are then sent to the next page via POST form submission.

The video in the background is one publicly available from lacity.org and can be changed in the <video> tag of index.php. Additionally, the web page is made responsive for various screen sizes.

resources/js/index.js:

This javascript code makes sure that the webpage begins with Debris and Socioeconomic factors selected as “Low”, and all values initialized as 0. This can be changed by replacing the txt variable to “Medium” or “High” and the value to 1 or 2 accordingly.

UpdateTextInput function updates the labels of the abovementioned factors to display the value selected.

sweepingneed.php:

This should be taken to <https://localhost/sweepingneed.php> after choosing settings on the index page. This page retrieves the values that were sent through the form in the index.php page from the \$_POST variable. The same form is available on this page with the retrieved variables as the new adjusted values on the form. The options can be changed and a new map can be generated after clicking “Generate.”

updateTextInput function, as in index.php, updates the labels of the inputs. refreshInput function updates the values of the inputs after new input values are selected and sent to the same page to generate a new map layer via POST.

resources/js/sweepingneed.js:

A “routes” map data structure is generated with a key-value pair of options selected from the previous page and file names to be used as layers (file names will likely need to be updated if a set of geojson files with different names are generated.)

Then, Arcgis function is started where colors per segment are identified based on the value of the “route” field. The “route” field, as designed by the algorithm team, represents the highest-lowest need for sweeping. Therefore, about 25% of the lowest values (representing highest need for sweeping) are colored in red, followed by the next 25% (more than average need for sweeping) with color orange, followed by next 25% (less than average need for frequent sweeping) colored by light green, followed by the next 25% (least need for frequent sweeping) with the color dark green. This is shown in a legend on the bottom-left corner of the screen.

A search widget is similarly added to the upper right corner of the screen, for users to be able to go to a certain location and examine the areas nearby. A zoom in/out button is also added at the upper left corner of the screen.

Finally, a popup template is created every time a user selects a given segment. It displays the number which determines it's standing for sweeping need out of all the rest of the streets that require sweeping. Just as in the previous page (index.php), this page is made responsive as well for various screen sizes.