Pedestrian Accessibility in Seattle - Dashboard Style Visualizations

Group Members: Harshitha Akkaraju, Xiaotong(Sherry) Chen, Zhiqi Lin, Logan Young

Abstract

The focus for this quarter was to build four different interactive visualizations that explore the pedestrian accessibility in Seattle. These visualizations include pedestrian accessibility explorer, dynamic isochrone generator based on a user profile, distribution of curb-ramps across different districts in Seattle, and a network centrality visualizer.

Larger Project Context

The current context of our project is that people in the community are asking all kinds of questions about the data that Access Map has right now. We hope to answer their questions by developing data visualizations to allow the users to have a better understanding of accessibility status in Seattle Area. For now, each of us created a visualization that resonated with our interests.

Group Objectives

- To build four interactive data visualizations that allow the users to explore accessibility related data.
- To Integrate multiple datasets together.
- To learn new skills while working on this project.

Pedestrian Accessibility Explorer

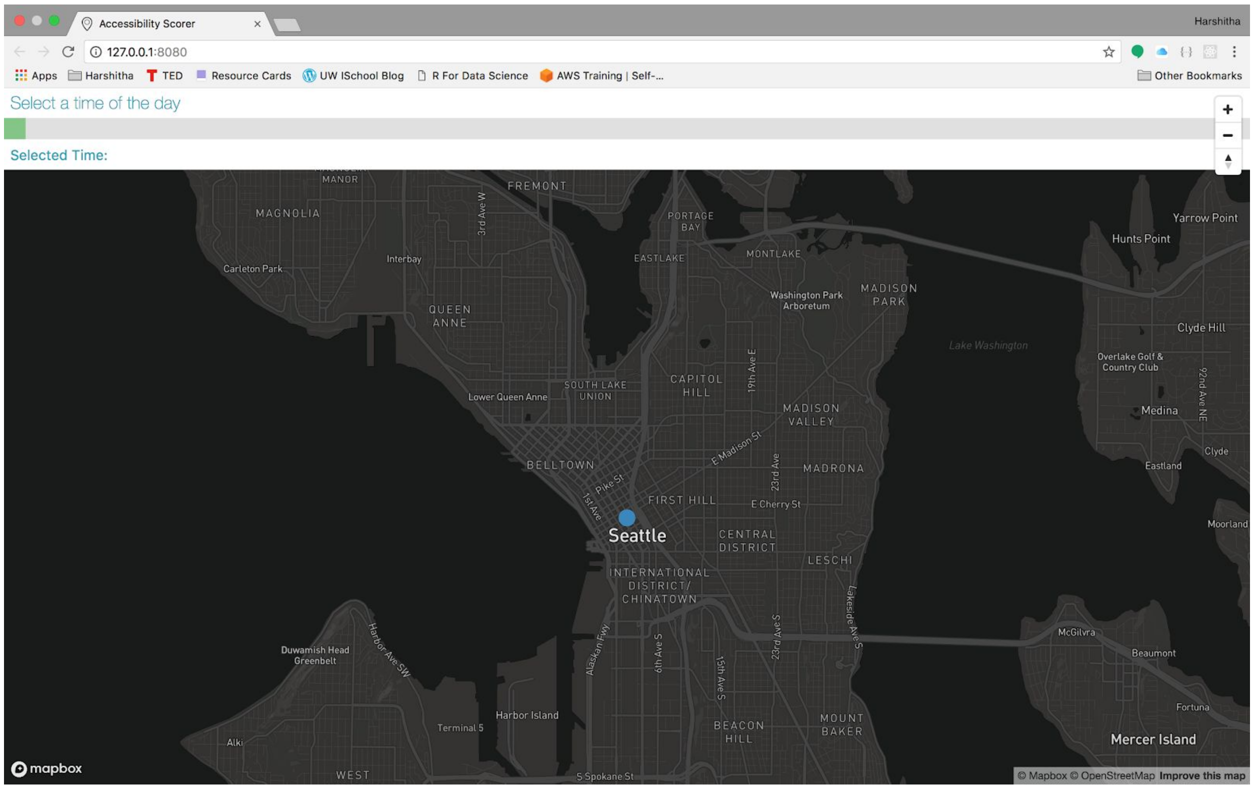
Harshitha Akkaraju

Objective

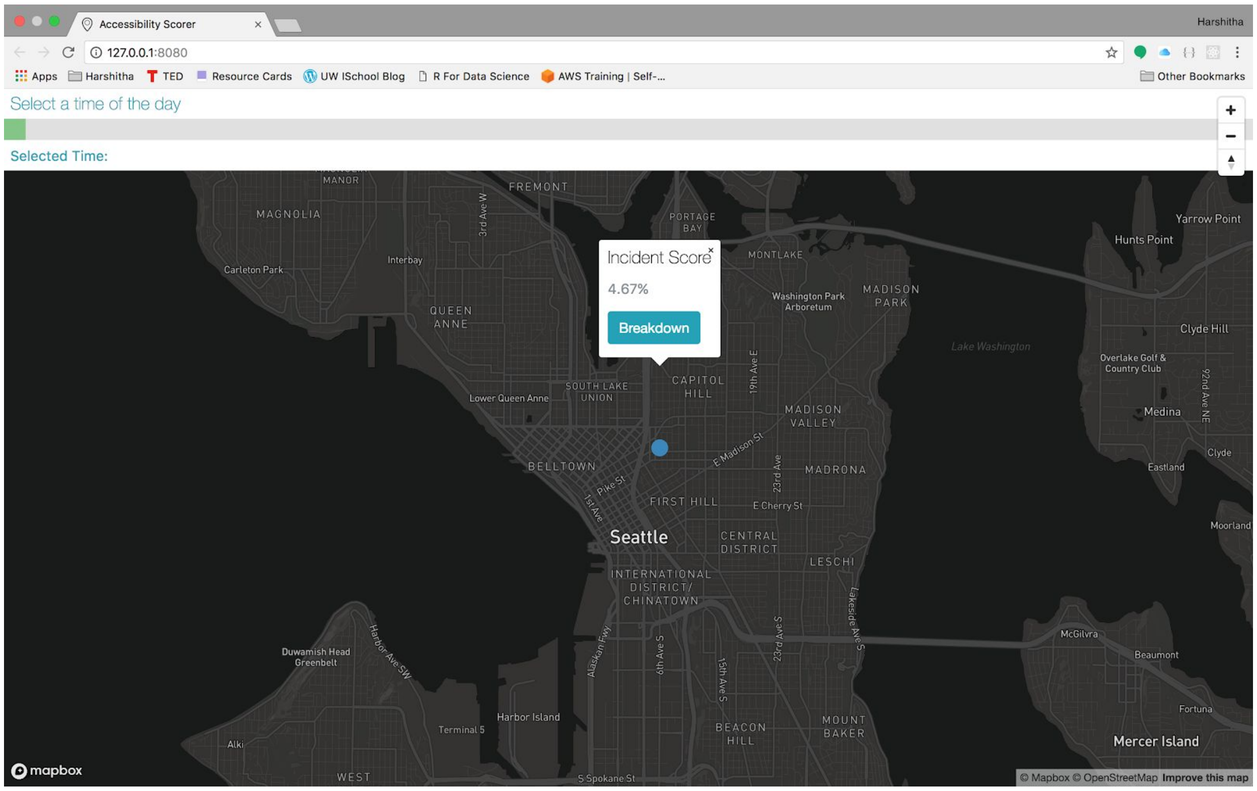
To create a web app that allows pedestrians to explore the accessibility of an area in various aspects. I would like to integrate datasets such as the SDOT's incident data, SDOT's Neighborhood dataset, and also the AccessMap's sidewalk data.

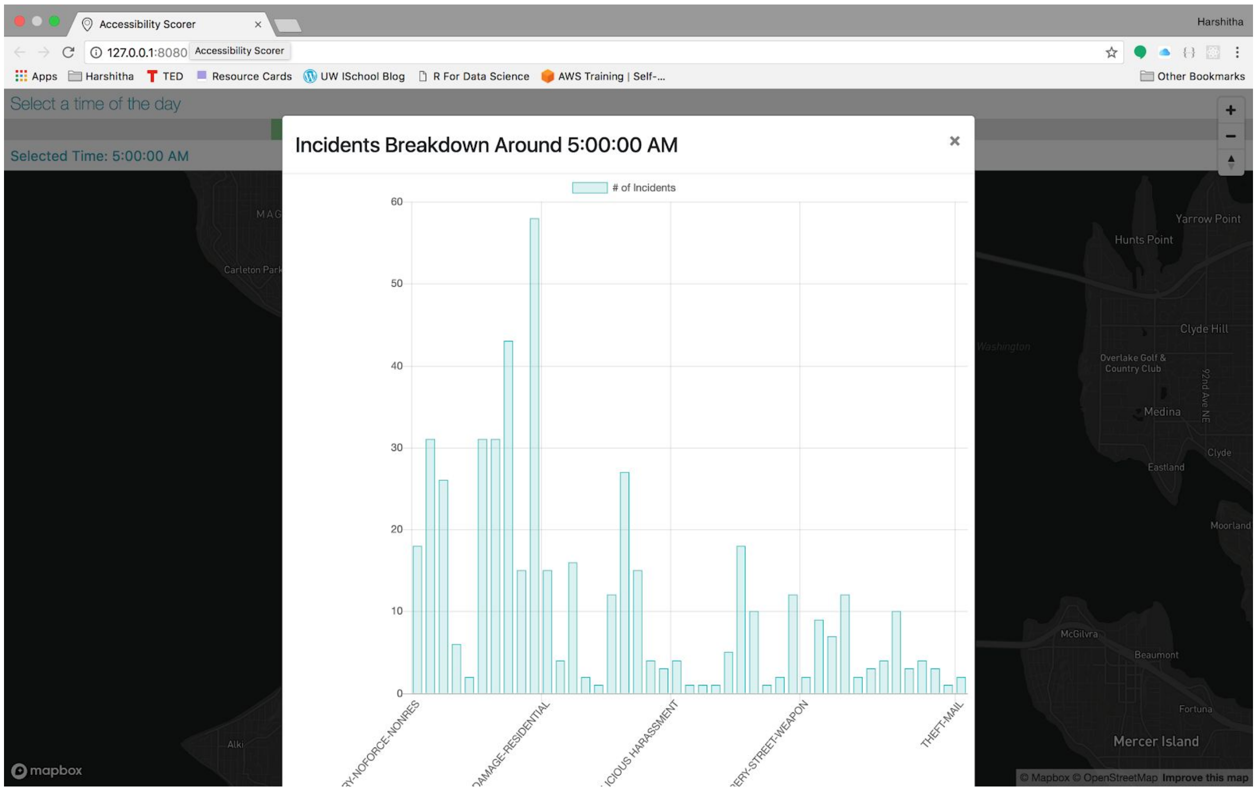
The webapp would generate an accessibility score based on user input. I think such scores based on different aspects of accessibility could be integrated with routing as well.

Webapp Screenshots



The user can drag and drop the blue circle anywhere in the Seattle area


Accessibility Scorer
127.0.0.1:8080
Select a time of the day
Selected Time:
Incident Score
4.67%
Breakdown
Seattle
mapbox
© Mapbox © OpenStreetMap Improve this map


Accessibility Scorer
127.0.0.1:8080
Select a time of the day
Selected Time: 5:00:00 AM
Incidents Breakdown Around 5:00:00 AM
# of Incidents
60
50
40
30
20
10
0
RY-NOFORCE-NONRES
DAMAGE-RESIDENTIAL
ICIOUS HARASSMENT
ERY-STREET-WEAPON
THEFT-MAIL
mapbox

Functionality

- Select a time of the day you want the data for (by default, the app uses current time in the day)
- Drop a pin in the area you want to explore
- This shows a popup with a percent score that indicates the percent of incidents that were reported in a given time of the day
- Click on the 'breakdown' button to further explore the data

Documentation

- Clone branch ‘akkarh’ in the GitHub Repo.
- Install live-server through npm.
- Run live-server in the project directory.

Process

Various factors affect the accessibility of pedestrian routes in Seattle. I wanted to create a webapp that allows users to explore these different factors.

This webapp is built using JavaScript, the MapBox GLJS API, and ChartsJS API. The app uses Sidewalk data from AccessMap and also the SDOT’s incidents API.

The incident data explorer is two-folded. First, the user is able to drop a pin and the app retrieves the incident data within a perimeter of approximately 2 miles. I then filter out the incident reports by either the current time or the time in the slider input. After the back-end computations, the user sees a pop up on the screen indicating a score of the number of incidents that have been reported in the 2-mile radius. The user also has the option to see a further breakdown of the data by clicking on the ‘Breakdown’ button.

Future Developments

- The current web app is somewhat slow because it is reading through a lot of data to eliminate irrelevant results; I am looking to see how I can make it faster.
- I am currently working with the AccessMap’s sidewalk network. I am finding that some of the sidewalk information is missing (because of poor connectivity) and through my visualization, I would like to identify where the missing information is.

- I also would like to look in the ORCA card transit data and see what inferences we could make from it.

Dynamic Isochrones Based On User Profiles
Sherry Chen

Description

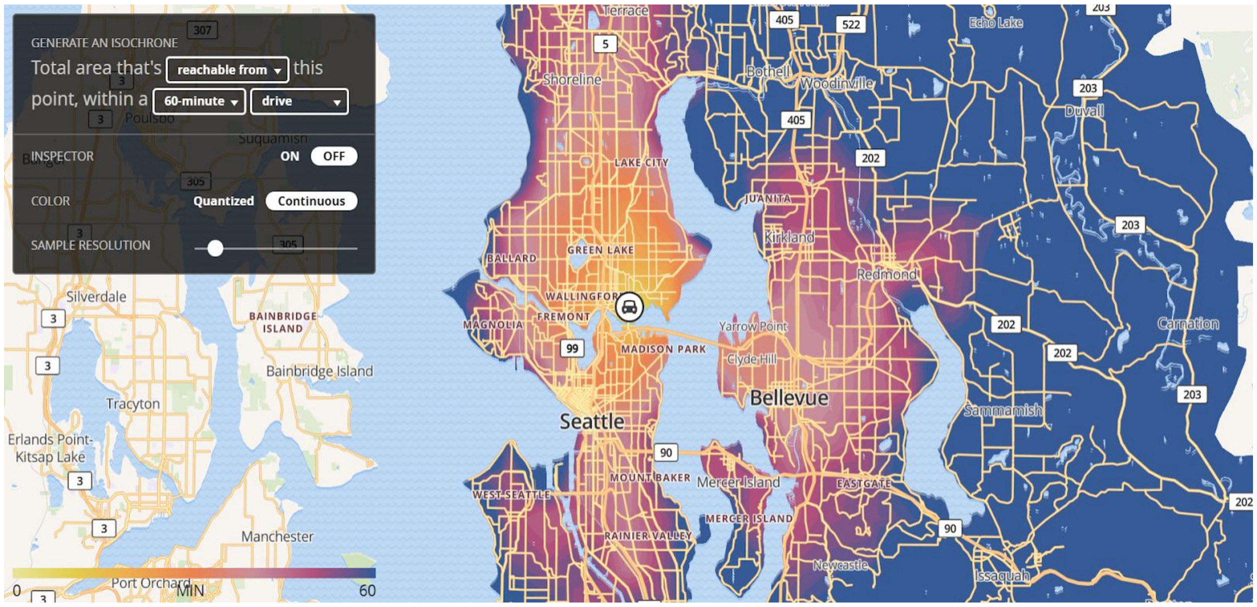
From daily experience, it is clear that the incline of Seattle streets can vary greatly from area to area. Big incline can potentially bring much inconvenient to people, especially among the populations with limited mobility, such as people using walker and wheelchair.

The purpose of this visualization is the show the user accessibility of Seattle streets based on the area they can travel in a given amount of time with a certain mode of transportation.

As more factors of streets such as street condition and furnishing are to be considered, this visualization takes street incline as the only factor that influences travel time. And the level of accessibility is directly shown in the form of isochrone.

Product Preview

http://htmlpreview.github.io/?https://github.com/ZKeiLin/AccessMap-Dashboard/blob/cxt/mapbox-isochrone/index.html


GENERATE AN ISOCHRONE

Total area that's reachable from this point, within a 60-minute drive

INSPECTOR ON OFF

COLOR Quantized Continuous

SAMPLE RESOLUTION

Code

Repo: https://github.com/ZKeiLin/AccessMap-Dashboard/tree/cxt/mapbox-isochrone

Documentation

- To use the product, download or clone the entire mapbox-isochrone folder into the local drive, and open index.html file with your browser of preference.
- Files in folders
    - Website development
        - index.html: the HTML file that includes the map visualization
        - index.js: the javascript file to create the map functionality
        - /dist/app.js: the javascript file with more fundamental built-in functions for index.js. Any modification on index.js needs to be applied to app.js as well.
    - Data
        - The database is based on Mapbox Matrix API
        - This isochrone generator is mainly based off https://github.com/mapbox/mapbox-isochrone

Process

The visualization is built by using Mapbox API, website development language javascript and HTML. Specifically, the algorithms are largely based off Mapbox Matric API.

I have been learning python since the start of the quarter. Consider the time constraint to present the product, I choose to switch javascript to process data and build visualization.

For building this visualization, the backend data processing is largely done using Mapbox Matrix API, of which supporting travel time computation in the mode of driving, bike, and walking. To emphasize street accessibility, I choose to add two additional modes of transportation: walker and wheelchair. The travel time computation is modified upon Mapbox walking mode, using incline as an independent variable. The weight of incline to changing in travel time is based off daily experience, and I am looking to do more research in that aspect.

This is the algorithm to calculate travel time using a walker. The walker mode is named as walking1 for the first modification of walking mode. Incline, both as steep uphill and downhill, is equally considered to decrease accessibility and increase travel time because pedestrians need to take extra work to control the walker.

```
if (parameter.mode === 'walking1') {
  parameter.mode = 'walking'
  for (t in state.timeTravel) {
    var incline = (abs(h/(x2 - x1)) + abs(h/(y2 - y1))) / 2 // calcuate the incline
    if (incline >= 0.06 || incline <= -0.06) {  // for steep slope, change the timeTravel
      t = t * 1.4
    } else { // default timeTravel for walker
      t = t * 1.1
    }
  }
}
```

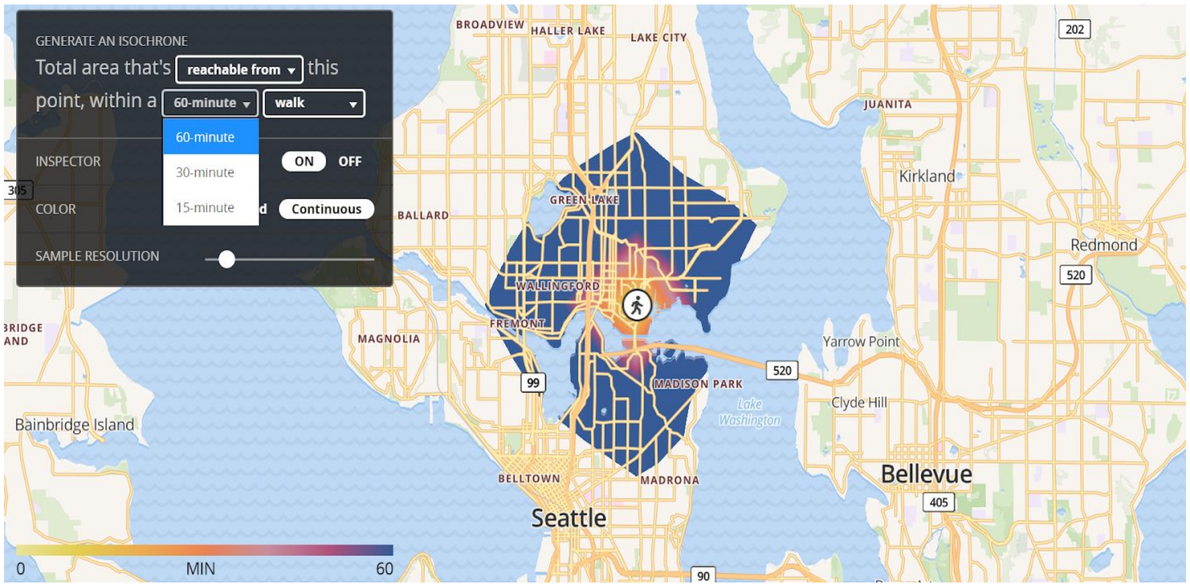
This is the algorithm to calculate travel time using a wheelchair. The wheelchair mode is named as walking2 for the second modification of walking mode.

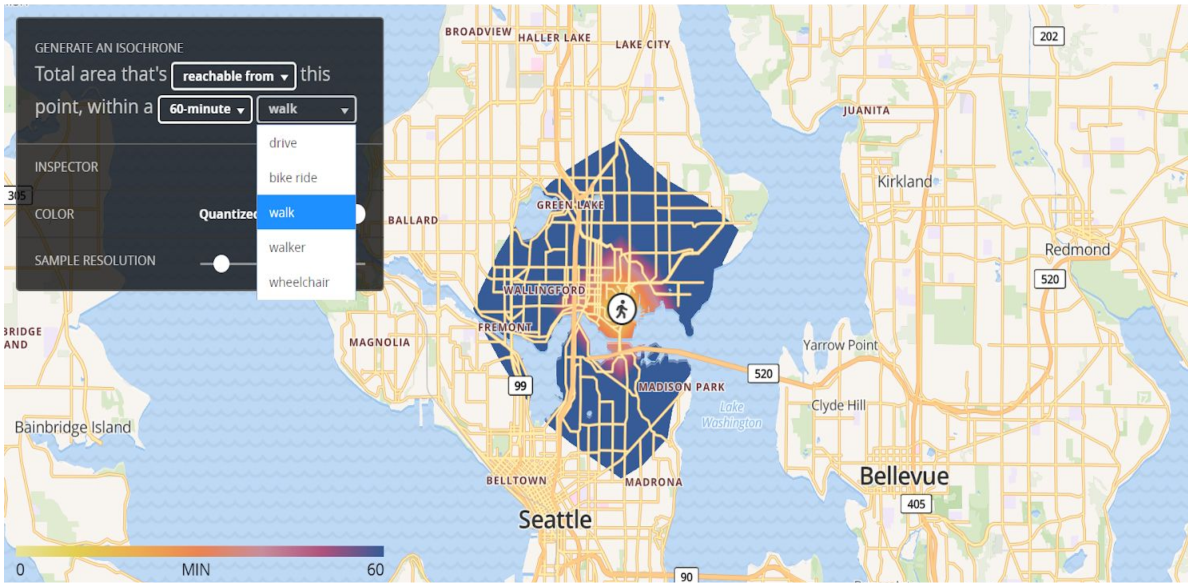
```
if (parameter.mode === 'walking2') {
  parameter.mode = 'walking'
  for (t in state.timeTravel) {
    var incline = (abs(h/(x2 - x1)) + abs(h/(y2 - y1))) / 2
    if (incline >= 0.06 || incline <= -0.06) {
      t = t * 1.3
    } // else t = t
  }
}
```

For the website development, it is built using javascript and Mapbox packages. I have implemented different Mapbox styles and changed the setting of the map for visual readability. Consider the current focus area is Seattle, the default drop-in location is also set to be Seattle, but the isochrone should work for any location on the map considering the database of Mapbox.

For the specific feature of this visualization, this map is interactive, and users can choose different locations, transportation mode, maximum time interval, etc.

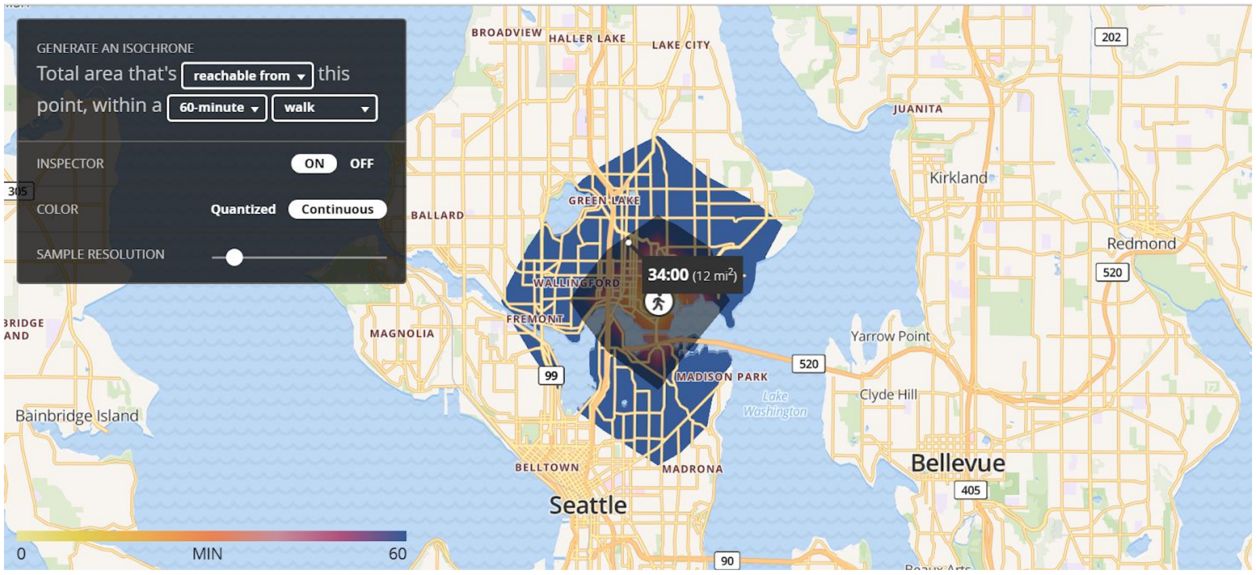
This is the dropdown for the maximum time interval and transportation mode.





In addition to all the options to customize the visualization, one feature that breaks the time interval limit is the inspector, which allows the user to choose a specific time and see the area can be reached from a certain location.

This is the inspector showing the transparent black area which can be reached within 34 min walk.



(These features are supported by Mapbox isochrone plugin. For more details, check https://github.com/mapbox/mapbox-isochrone)

Future Developments

- The mode.js file in the folder is currently in progress. The final goal to create two additional modes besides three default modes: drive, bike, walk that calculate travel time using better and more specific algorithms rather than modifying walking mode algorithms directly.
- The algorithms are only considering the incline of the street as the factor of accessibility now. However, the condition of the street and the appearance of street furnishing should be taken into consideration. This would require implementing other databases besides Mapbox API.

Distribution of Curb-ramps

Zhiqi Lin

Description

The number of crossings without curb ramps is as twice as the crossings with curb ramps. The purpose of the visualization is to let the users find out which area in Seattle need to build more curb ramps to make the crossings more accessible.

This visualization shows the distribution of the curb ramps in the Seattle area and how they vary across districts. Users are able to hover the map to gain the certain neighborhood's name.

The cost of each curb ramp at nearly $13,100 in Seattle because of Seattle's complex geography, so that the visualization can also which neighborhoods need more funds on developing curb ramps.

Product

- HTML file preview(final product preview):
  - http://htmlpreview.github.io/?https://github.com/ZKeiLin/AccessMap-Dashboard/blob/gh-pages/website/index.html

Code

Repo: https://github.com/ZKeiLin/AccessMap-Dashboard/tree/zk

Documentation

- To use the product, clone the specific branch zk from repository https://github.com/ZKeiLin/AccessMap-Dashboard/ and go into the website file.
- Highly recommended to use live-server to render the website file.
  - Use `npm install` to install all the needed packages such as mapbox for the javascript file.
- Files in website file:
  - Website Development
    - index.html: the HTML file that includes the map
    - js/index.js: the js file that is used to render data and create the map
  - Data
    - counties.geojson: dataset contains all the neighborhood in Seattle Area
    - crossings.geojson: the dataset contains all the crossings(with/without curb ramps in Seattle Area

Process

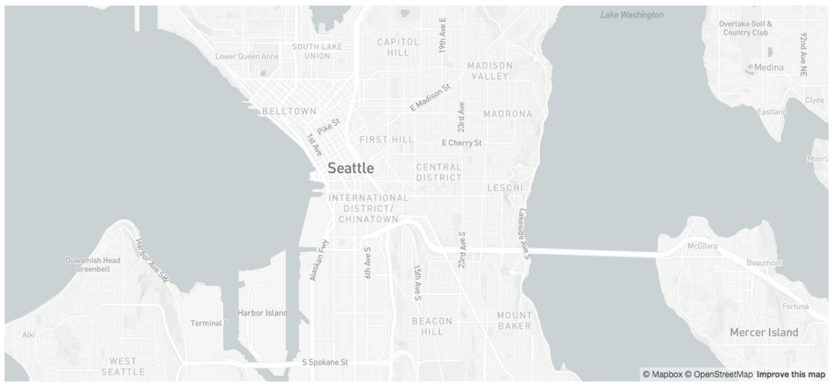
The visualization is built by using AccessMap API, Seattle neighborhood data, and website development languages including HTML, CSS, and javascript.

I have studied how to use javascript to import and process the data and how to use the mapbox packages since the projects started. There are two main parts of implements, the data processing, and web page development. I have processed the data and then started working on the visualization.

Data: I used the data from seattle.io API and AccessMap Crossing API. I process the data by using javascript. For the AccessMap Crossing API, I grouped the crossings according to whether they have curb ramps or not. For the Seattle Area neighborhood, I mainly focus on Seattle Area, so I filtered out some of the neighborhoods that offered by the API such as Bellevue.

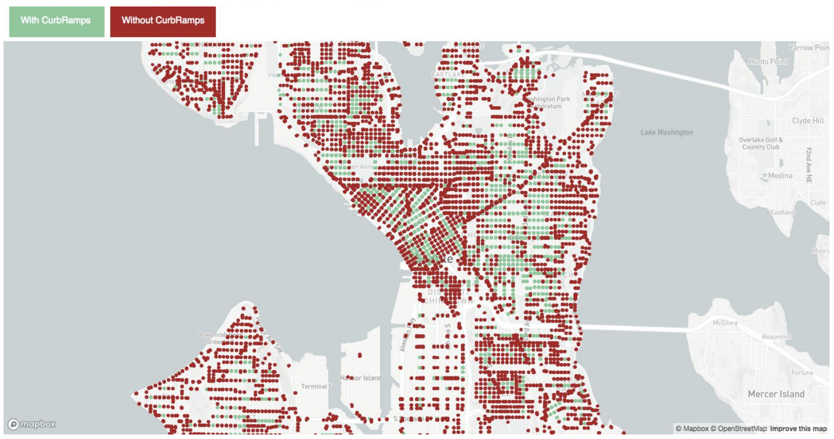
Webpage: I built the webpage by using HTML, CSS, and javascript with packages mapbox and Mapbox GL JS. I implemented the map of Seattle area by using mapbox.

This is the original map that I have on my webpage:



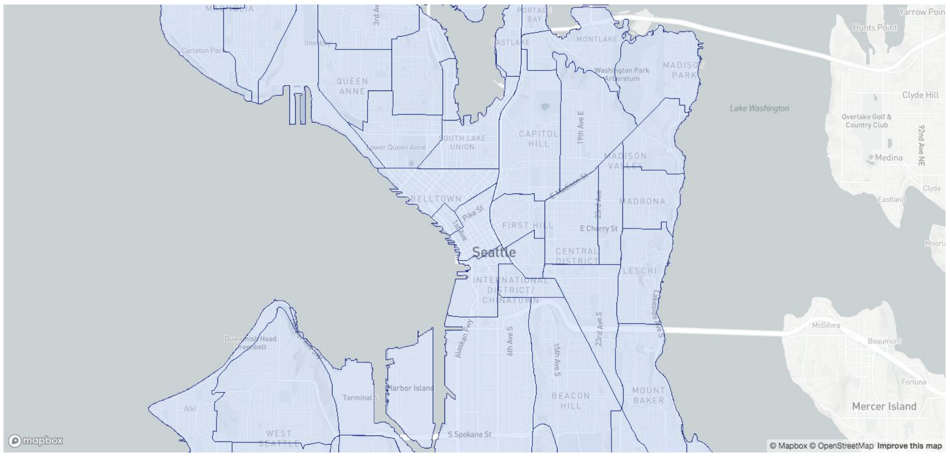
There are two layers on my map, the first layers are the dots layers representing whether the certain crossing with curb ramps or not. The green dots are representing the crossings with curb ramps, and the red dots are representing the crossings without curb ramps.

This is my first layer:

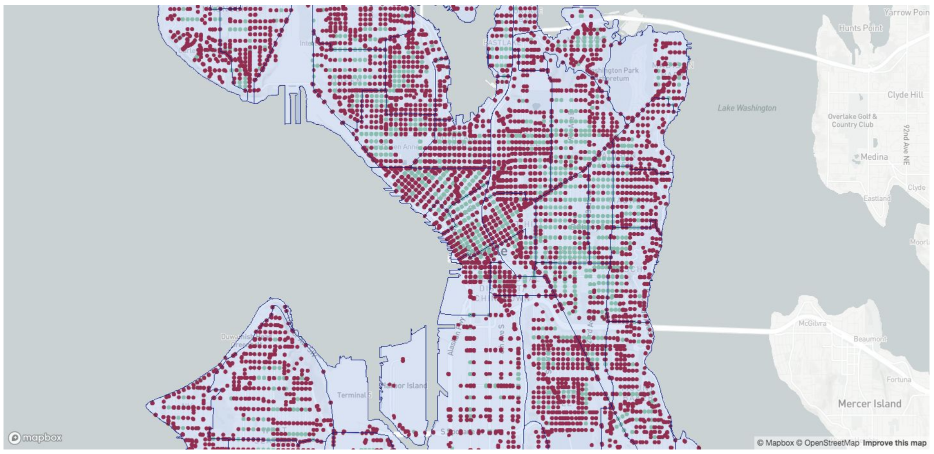


The second layer is the polygons representing the certain neighborhood areas.

This is my second layer:



After combining two of them:



However, at this time, the graph still is not able to show which area with crossings that have the curb ramps. Therefore, I developed the popups for the area, wherever the users use the mouse to move over a certain area, the name of the neighborhood will pop up.

Future Developments

- Still in the process: Summarize and visualize the exact number of curb ramps in different neighborhoods and shows up the data on the pops up (by using Turf-within), but there are some crossings are right on the boundaries.
- Calculating how much each neighborhood might cost to build all the curb ramps
- Add more interactivities on the map such as only shows a layer with/without the curb ramps and more details(more research needed) the population of disability in different neighborhoods.

Network Centrality Visualization
Logan Young

Objectives
The main objective of my visualization was to give a dynamic way to see the connectivity of local pedestrian networks, shown through edge betweenness centrality. I chose this goal because of its high potential for extensibility. There are many ways to expand on this visualization, such as adding walkways to the network, customizing the criteria for walkways included in the network, and how to calculate the edge centrality.

For now, however, I limited my goal to getting just the basics up and running: A working web page that displays a map of Seattle, where the user can select a bounding box of walkways to use, then display the selected walkways, with each linestring colored based on its centrality.

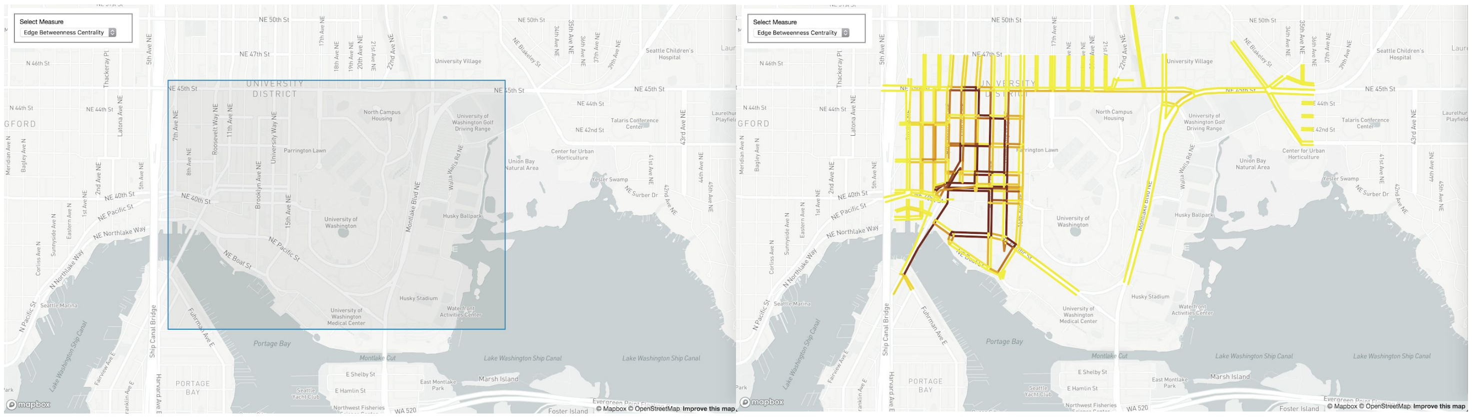
While I was able to get a barebones implementation of this working, I faced issues in mainly two area: Speed, and accuracy. Currently, the interaction between the map and the API is almost excruciatingly slow. Additionally, I did not have time to investigate fully how accurately the graph is being constructed, and whether or not there are potential disconnects between sidewalks that should be connected. Similarly, the centrality measure currently does not take into account the clustering of nodes, so the calculation will be skewed towards areas with highly concentrated groupings of nodes.

Process
I implemented this visualization with a simple, self-contained framework. There are two main components: An API, which handles the pedestrian walkways data and centrality

calculations, and the web page, which handles all of the user-facing information, displaying a map and allowing the user to input selections.

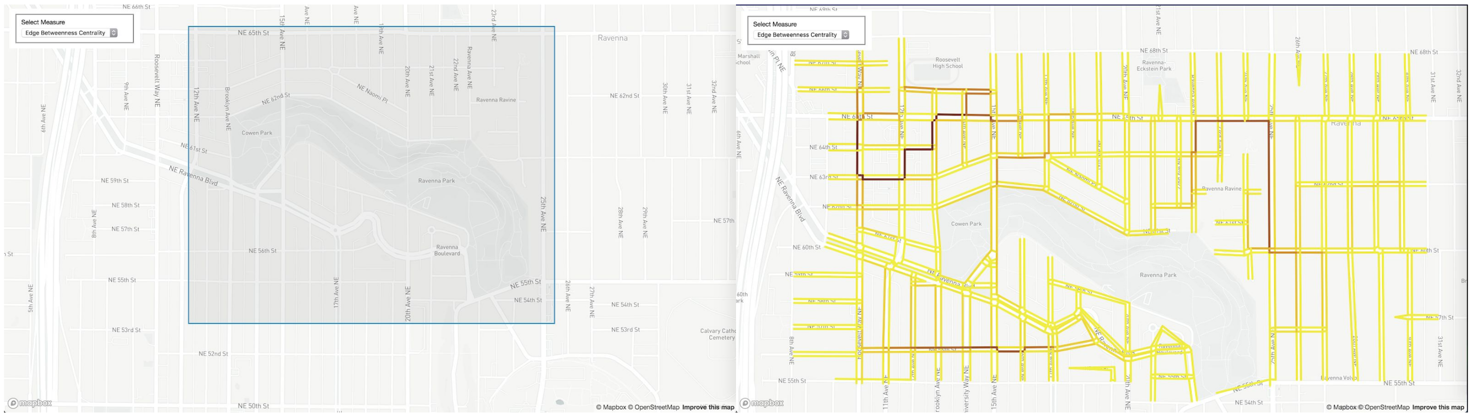
API: The API (api.py) is built using Python's Flask, providing only one endpoint used for inputting two points and returning the linestrings of the pedestrian walkways contained in the bounding box defined by those two points, with the returned walkways containing the edge-betweenness-centrality of each walkway within a graph/network made up of those walkways. The API uses NetworkX to construct the graph and calculate the centrality. The graph building portion is offloaded to a small python module (measure_graph.py).

Web Page: The web page (centrality_map.html) is built on HTML with Mapbox GL JS, similar to the AccessMap. It currently only has two pieces of functionality besides that of base MapBox, bounding box selection and pedestrian walkway rendering. Unfortunately, the response time of the API is currently very slow, so the delay between the bounding box selection and the rendering of the pedestrian walkways is significant.


Select Measure: Edge Betweenness Centrality

Select Measure: Edge Betweenness Centrality
UNIVERSITY DISTRICT
University of Washington
Portage Bay
Lake Washington Ship Canal
© Mapbox © OpenStreetMap Improve this map

The coloring is done on a static scale, low to high from yellow to dark brown. While the scale currently does not adjust to fit the max centrality of the selected network, the values chosen for each color I have found to match most selection ranges.


Select Measure: Edge Betweenness Centrality

Select Measure: Edge Betweenness Centrality

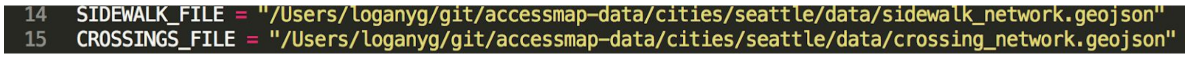
Install and Deployment—
All of the files necessary can be found on the "centrality_vis" branch of the AccessMap-Dashboard repository (https://github.com/ZKeiLin/AccessMap-Dashboard/tree/centrality_vis).

NOTE: The following instructions are solely for MAC OSX, these steps are not guaranteed to work on windows, though they should be largely the same for many Linux distributions.

To install and deploy the visualization, you will need the following:

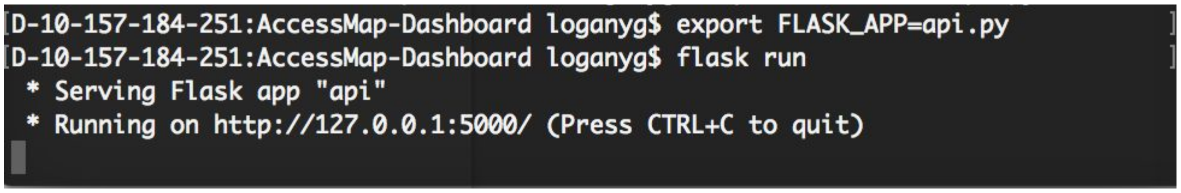
- Files
  - Code
    - centrality_map.html
    - api.py
    - measure_graph.py
  - Data
    - sidewalk_network.geojson
    - crossing_network.geojson
- Packages
  - MapBox GL JS
  - Python Version 3.6+
    - geojson
    - geopandas
    - pandas
    - json
    - networkx
    - flask
    - Numpy

First, it is necessary to start the flask server. You will need to modify the SIDEWALK_FILE and CROSSINGS_FILE variables in api.py and point them to your local copies of the sidewalk network geojson data and the crossings network geojson data respectively. These can be found in the accessmap-data repository of the AccessMaps project (https://github.com/AccessMap/accessmap-data/tree/master/cities/seattle/data).



```
SIDEWALK_FILE = "/Users/loganyg/git/accessmap-data/cities/seattle/data/sidewalk_network.geojson"
CROSSINGS_FILE = "/Users/loganyg/git/accessmap-data/cities/seattle/data/crossing_network.geojson"
```

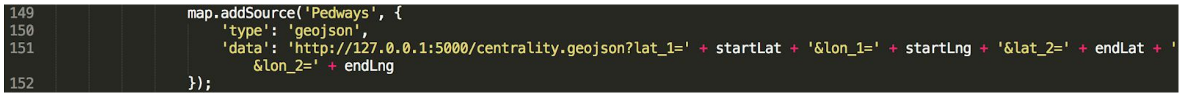
This can be done by first running, from your terminal in the main directory, “export FLASK_APP=api.py”, and then “flask run”.



```
[D-10-157-184-251:AccessMap-Dashboard loganyg$ export FLASK_APP=api.py
[D-10-157-184-251:AccessMap-Dashboard loganyg$ flask run
 * Serving Flask app "api"
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Once the API is running, you can then launch a local copy of centrality_map.html, and use the visualization. Use SHIFT + LEFT MOUSE BUTTON and drag to select a bounding box, then (after a sizeable delay) the pedestrian walkways should appear.

If the API is running on anywhere other than http://127.0.0.1:5000/, then it will be necessary to change the source location for the data in the web page code (centrality_map.html).



```
                map.addSource('Pedways', {
                    'type': 'geojson',
                    'data': 'http://127.0.0.1:5000/centrality.geojson?lat_1=' + startLat + '&lon_1=' + startLng + '&lat_2=' + endLat + '
                        &lon_2=' + endLng
                });
```

Future Developments

Though I got a barebones version of the visualization working, there are a few issues with the current implementation, as well as many possible expansions of the current work.

Most notably, the current implementation is very slow. This is mainly due to the time the API takes to search through the pedestrian walkways to find all included walkways, as well as the subsequent time to build the graph and calculate edge betweenness centrality. Unfortunately, I am unsure how to alleviate such bottlenecks except for running the API on a better machine than my laptop. Additionally, there seem to be some small visual bugs on the frontend side, with the map occasionally disappearing briefly.

There are also small visual tweaks and modifications that would improve the style of the current visualization. An important addition would be to have the color scale be based on the actual maximum value in the currently rendered network, instead of a static scale. Additionally, I have not played around with other palettes so far, and it might be more visually appealing to choose a different color scheme instead of brown and yellow. It would also be useful to try and mediate the biases in the current calculation, as, though the least path calculation for the centrality calculation factors in distance, it weighs each node evenly, so areas with many short interconnected walkways are favored heavily. A potential remedy for this would be to calculate edge betweenness centrality using only a subset of more evenly spaced nodes, with a limit to how many nodes can be included in a certain area.

Finally, there are a few larger extensions I could see making the visualization better and more interactive. Firstly, it could integrate the preferences options in the AccessMap application, allowing people to filter out walkways from the network based on the incline or curb ramps, showing the change in the centrality of each walkway based on this. Furthermore, an interesting addition could be to allow the user to drop in new walkways, choosing a start and end point for new walkways, then seeing how the centrality in the new network changes with the addition of this new walkway. Though this would be a much more significant and probably difficult addition, it would allow the user to not only understand their surrounding pedestrian network but test how it could be improved.