

HW6

[HW4-HW5-HW6 Overview](#)

Table of Contents

Table of Contents	1
0 Overview	1
0.1 Donut chart	1
0.2 Scatter plot	2
0.3 Typos? Questions? Clarifications?	2
1 Extra files	2
1.1 Data Set	2
1.2 ajax.js	2
2 Requirements	3
2.1 main.py	3
2.2 index.html	4
2.2.1 The head element	4
2.2.2 The body element	4
2.2.2.1 The Donut chart	5
2.2.2.2 The Scatter plot	6
2.3 front_end.js	8

0 Overview

In this homework you will write code to tie everything together in a complete web application.

In this part you will write code to process visualize the data which your code in part 2 produced:

0.1 [Donut chart](#)

The donut chart will show the percentage of all service requests associated with each department (indicated in the "SUBJECT" field), within a given range of years.

0.2 [Scatter plot](#)

The scatter plot will show the number of requests (Y axis) by duration (X axis), per department (data series), in a given year.

0.3 Typos? Questions? Clarifications?

If you think you've found a typo, have questions, or would like clarifications on what is required, post in the class Piazza forum.

1 Extra files

There are some additional files you will need to complete the web application.

1.1 Data Set

Recall that your code will process data from [311 Service Requests](#) data set available via [OpenData Buffalo](#). The entire data set appears to be too big to load into repl.it, so here is a smaller dataset (though still substantial: it has ~80,000 records) that you can test your code with:

[311_Service_Requests_small.csv](#)

You can also download it as a zip file, which will be faster, but you will need to extract the contents to get the CSV file:

[311_Service_Requests_small.zip](#)

You should upload the CSV file to your repl.

Note that when you submit your work you need to make sure that this file is in your zip file, and make sure that at the top of your App.py file (after the imports but before your first function definition) you include:

```
ALL_DATA = readCSV("311_Service_Requests_small.csv")
```

1.2 ajax.js

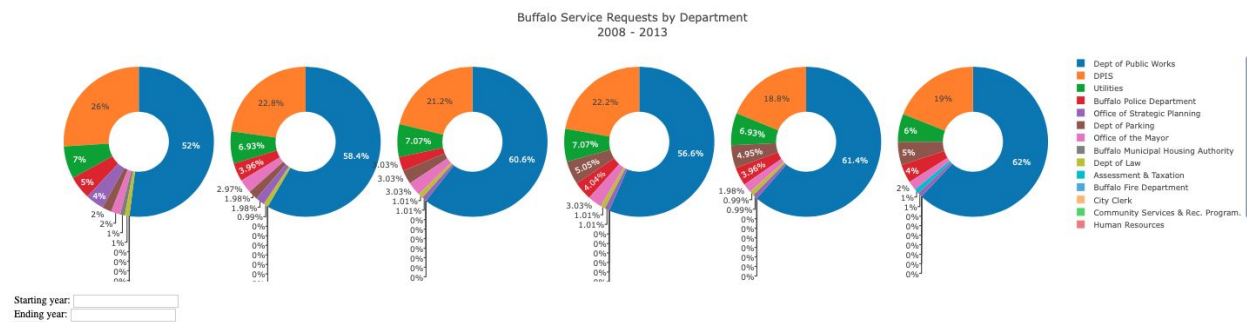
You can download [ajax.js](#); upload this file into your repl.

2 Requirements

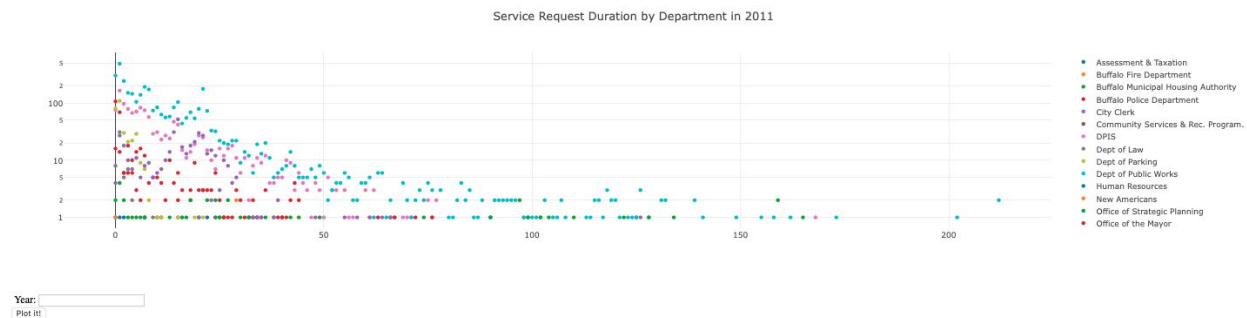
If you did not complete the requirements for parts 1 and 2, do that first. For part 3 you need to write two files (index.html and front_end.js), ensure that one file (ajax.js) is included in your repl (and in your submission), and make one small modification to your Server.py file.

In the end you will have a small web application that will be able to display graphs like this:

Donut



Scatter



2.1 main.py

Your main.py file should now be defined with the following content:

```
import bottle

import Server

bottle.run(host="0.0.0.0", port=8080)
```

If you still have testing code you wish to run to verify the part 1 and part 2 functionality you can comment leave that in the file and comment out the

```
bottle.run(host="0.0.0.0", port=8080)
```

line of code. When you are ready to try out your whole web application you should comment in the above line, and comment out any other testing code you may have in `main.py`.

2.2 index.html

The contents of `index.html` will define user interface (UI) for your web application. A UI has two primary functions: to present information to a user, and accept input from a user.

2.2.1 The *head* element

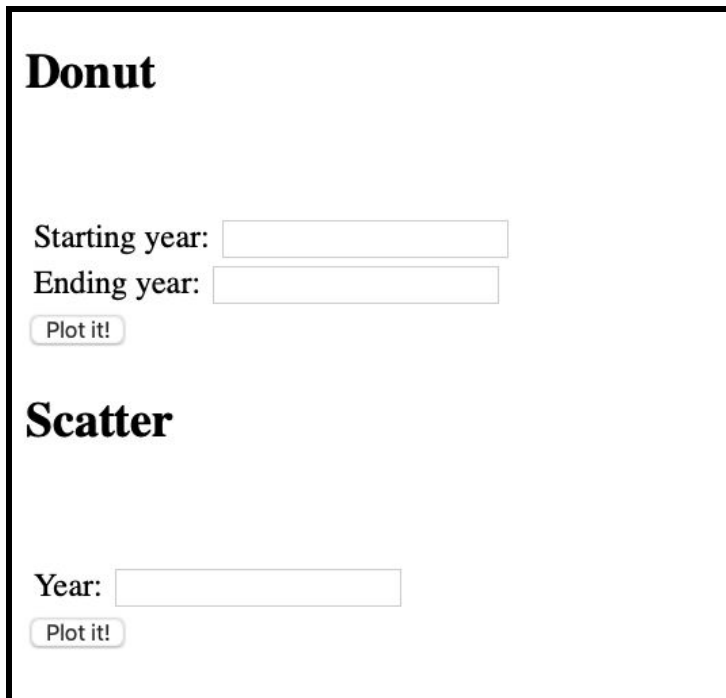
Of course our `index.html` file must also request various JavaScript code files. These files will be requested using `<script>` elements in the `<head>` of the HTML page. Javascript code must be requested from your web server along both the `ajax.js` and `front_end.js` routes.

The Plotly library must also be requested from `https://cdn.plot.ly/plotly-latest.min.js`

2.2.2 The *body* element

The two presentation elements will be two `<div>` elements, one for the donut chart and one for the scatter plot.

When the application starts up it should present a page that looks like this:



Donut

Starting year:

Ending year:

Scatter

Year:

2.2.2.1 The Donut chart

Looking at the Donut chart first, the user can enter a starting year and an ending year in the two input text fields. Clicking on the "Plot it!" button just after the two input text fields triggers an AJAX POST request to be sent to the back end server along the '/donut' route, along with a dictionary of the form

```
{"year_start":startYear , "year_end":endYear}
```

Where `startYear` is an integer corresponding to the year typed in the "Starting year:" text field, and `endYear` is an integer corresponding to the year typed in the "Ending year:" text field.

Note: For the purposes of this homework your code does not need to do any error checking on the values entered: if the user enters nonsensical values your application does not need to give an error message, though it must not crash either. After entering nonsensical values, entering valid values and clicking "Plot it!" should cause the application to behave correctly.

For example, entering the following for the donut plot,

Starting year:

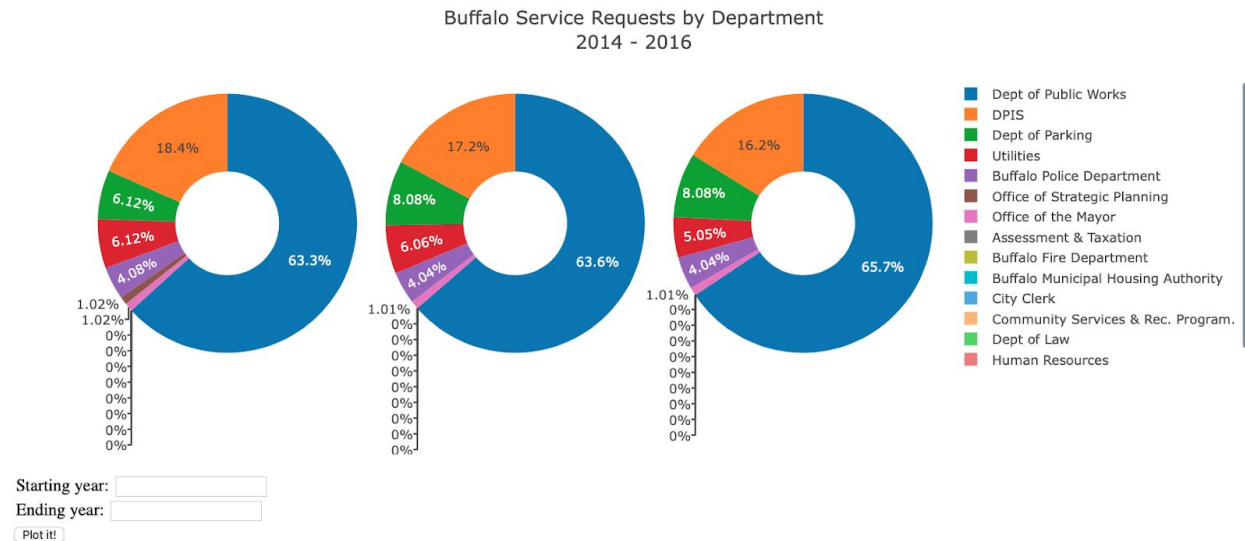
Ending year:

and clicking "Plot it!" should send this dictionary

```
{"year_start":2014 , "year_end":2016}
```

as part of the POST request to the server, which must respond with data to produce a graph that looks something like this:

Donut



Notice that the two text fields have been cleared.

The callback function for this POST request must call `Plotly.newPlot` with the data provided by the server, as well as some layout information. Figuring out a good layout with plotly can be tricky, so here's the layout we used to create the above, assuming that `data` is the data that the server sent:

```
let s = data[0]["name"]
let f = data[data.length-1]["name"]
let layout = {
  title: 'Buffalo Service Requests by Department<br>' + s + " - " + f,
  showlegend: true,
  grid: {rows: 1, columns: data.length}
};
```

2.2.2.2 The Scatter plot

The Scatter plot controls are a little simpler, because the user can enter just one year:

Year:

Clicking on the "Plot it!" button just after the single input text field triggers an AJAX POST request to be sent to the back end server along the '/scatter' route, along with a dictionary of the form

```
{"year":year}
```

Where `year` is an integer corresponding to the year typed in the "Year:" text field.

Note: For the purposes of this homework your code does not need to do any error checking on the value entered: if the user enters a nonsensical value your application does not need to give an error message, though it must not crash either. After entering a nonsensical value, entering a valid value and clicking "Plot it!" should cause the application to behave correctly.

For example, entering the following for the scatter plot,

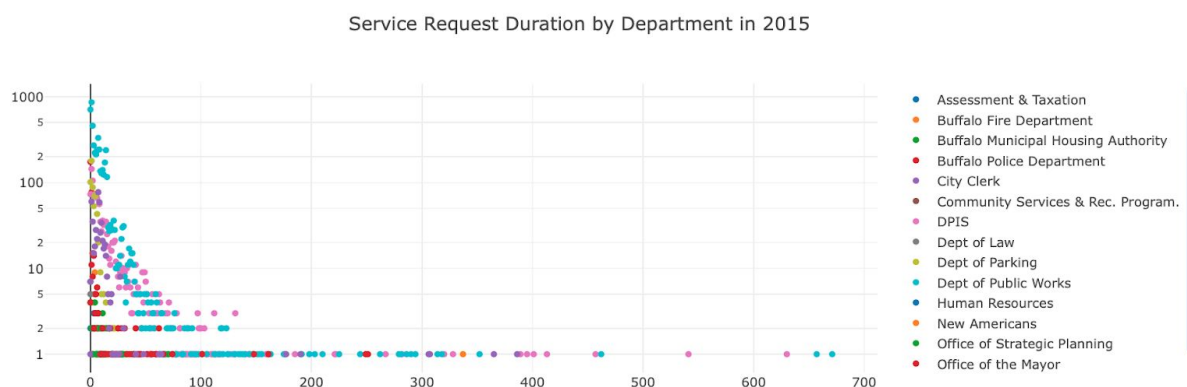
Year:

and clicking "Plot it!" should send this dictionary

```
{"year":2015}
```

as part of the POST request to the server, which must respond with data to produce a graph that looks something like this:

Scatter



Notice that the text field has been cleared.

The callback function for this POST request must call `Plotly.newPlot` with the data provided by the server, as well as some layout information. Figuring out a good layout with plotly can be

tricky, so here's the layout we used to create the above, assuming that `responseData` is the data that the server sent:

```
let year = responseData['year']
let layout = {
  xaxis: { autorange: true },
  yaxis: { type: 'log', autorange: true },
  title: 'Service Request Duration by Department in '+year
};
```

Here we see the effect of the small change that we're going to require in `Server.py`: rather than just send the data that our `data_by_server_duration` function returns, we'll arrange that the year for that data is included too. Assuming the variable `year` is assigned the year that was sent as part of the POST request, we can accomplish this with a few lines of code:

```
data = App.data_by_subject_duration(content)
result = {'year':year, 'data':data}
return json.dumps(result)
```

2.3 front_end.js

This file must define functions that provide the functionality necessary for the front end. Review the code of the Chat and Music Rater applications (posted on the resources page of the website) as well as relevant lecture recordings and slides for information on using AJAX, callback functions, and using the Plotly library.

You will need to define four functions, two to handle the Donut chart and two for the Scatter plot.

For each type of chart you need one function that will be called when the "Plot it!" button is clicked. This function will grab the user inputs from the input element(s), clear the input element(s), and formulate an appropriate AJAX POST request to the web server, including specifying the callback function.

Note: the body of each of these two functions is short, 3 to 4 lines. Of course if you like to space out your code and use more variables that we did it may be a little longer...but the basic message is don't overcomplicate things!

For each type of chart you will need a callback function. The callback function must unpack the data received from the server, formulate an appropriate layout specification (as described above), and invoke Plotly to draw the graph.

Note: the body of each of these two functions is a little longer, maybe 10 to 15 lines, depending on how you write it. About five of those lines are for the layout (given to you above). And again, if you like to space out your code and use more variables that we did it may be a little longer...but the basic message is don't overcomplicate things!
