**Coding and Data Skills**
Dr. Cindy Royal
Texas State University
School of Journalism and Mass Communication

## News Application

We've learned a lot of skills that could lend toward making a news application. You can use JavaScript to go through a JSON file and present data based on a user's input. You've learned how to use JavaScript-based charts that you can change, also based on a user's input. This tutorial will show you a more comprehensive approach to making a news application, one in which you have the ability to look at a full dataset and create pages to represent each line of data.

We'll be following this tutorial created by Ben Welsh of the LA Times. http://first-news-app.readthedocs.org/ . Ben did this tutorial at NICAR earlier this year. He used data from the Los Angeles area, deaths during the LA Riots in 1992. I decided to use a dataset that was related to our music focus, so we will be creating an application with Austin music venues.

To do this tutorial, you need:

- the Terminal
- a text editor like Text Wrangler
- Python
- the pip package manager
- the virtual environment

We will use Leaflet.js to make the maps. We will check to be sure that each is installed, working on the masscomm login (so we have permissions). We post everything to GitHub at the end. You can find everything in my GitHub, github.com/cindyroyal/newsapp

### Finding the Data

On the City of Austin's website, I found a page about Venues (https://www.austintexas.gov/department/venues). It included a link to a listing of Austin Venues including address, links, Twitter, phone. But this nice list was in pdf format. I looked around to see if I could find a better listing already in a spreadsheet or csv, but couldn't. So, I was stuck with the pdf. My first try was to just copy and paste. No luck. But, there is a free program called Tabula that helps you get data out of a pdf. I downloaded Tablula, opened the venues pdf and used the Lattice option to select and download the file.

- It was in pretty good shape, but I did have to do a few things to get it just right using Excel (or feel free to use Google Sheets).
- Some of the columns were out of line, so I had to move things around. And some of the text had moved to new lines within a cell. So, for those columns, I used the Excel function in a new column, then deleted the original column. I found that answer here: http://www.computing.net/answers/office/replace-with-line-break-excel-for-mac/14288.html

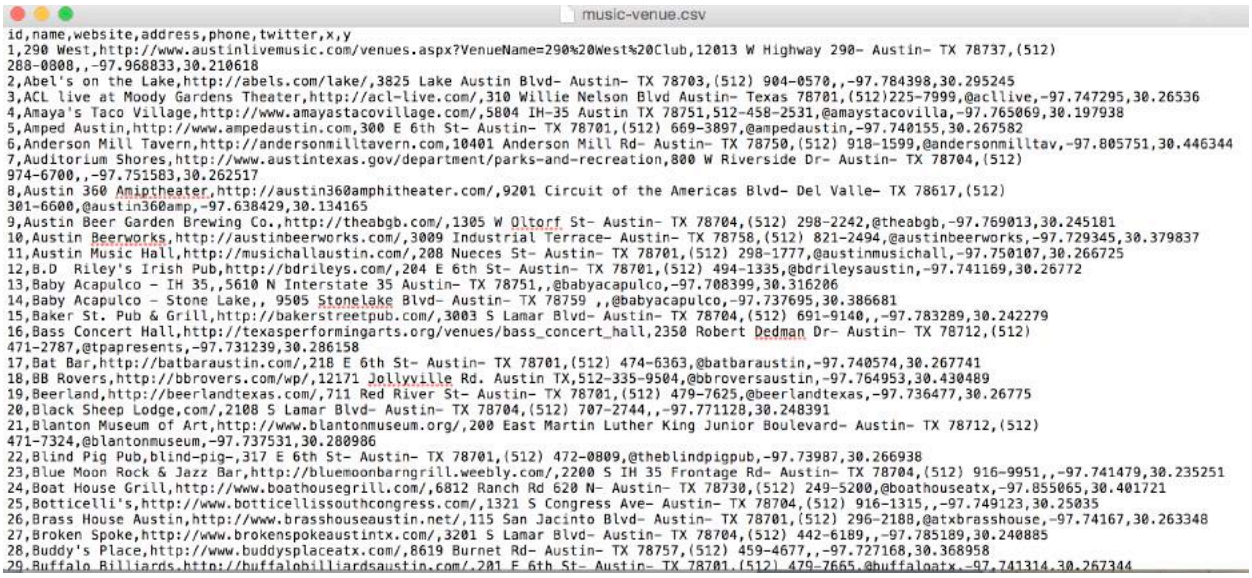
  =SUBSTITUTE(A1," | ",CHAR(13))

- I also found out later in the process that I needed to get the latitude and longitude for the addresses. Luckily, there is this awesome Batch GeoCode (http://www.findlatitudeandlongitude.com/batch-geocode/#.Vw5JUxMrKL8 ) site that did it for me. I inserted the addresses and had it output the lat and long, which I then copied and pasted into the appropriate columns called x and y in my spreadsheet.

- And then each row needed an id, which was easy to add by inserting a column at the beginning of the doc and adding sequential numbers starting at 1.

I then saved it as a csv from Excel, so I could use it in the Tutorial in the same way as in the First News App tutorial

There were a few other things I had to figure out along the way, because some of the characters had funny encoding and when Excel creates a csv, it puts a newline break after each line. The python script didn't like that. But I found a simple tweak that is represented in the code that handled both those issues, although it took me a long time to figure out that was what was happening (problem solving, right?). For this exercise, however, you get the completed csv with all the changes made.




## Starting our News App

So with our edited csv of the venues, we are ready to make an application. We are going to create an application that lists all the venues from the csv, maps them and then creates a separate page for each venue. Since there are 230 on the list, it would take a really long time to do each of these individually. So, we are going to use the Flask framework to help us. Flask is a Python micro-framework http://flask.pocoo.org/. It differs from a framework like Ruby on Rails in that it is more lightweight, simpler and doesn't provide a database for you. But that's fine for this project. We are going to use our csv for the data. We are also going to ultimately "freeze" our site to make a static site – generating all the html pages, so we don't have to host the site on a Python/Flask-enabled server. This will make sense as we go.

The documentation for the First News App explains what you need and how to get it. I have already installed Python, PIP and virtualenv on our computers. PIP lets you install things and the virtualenv sets up an environment that lets us run all the required things we need, basically hosting a lightweight server on our computer while we create the app.

You can do in the Terminal, to check if you have each. Remember that the $ just represents the prompt, so you don't type it.

```
$ python --version
$ pip --version
$ virtualenv --version
```

## Creating our Environment

Start by creating the Virtual Environment. You can navigate in Terminal to anywhere you want to start this app. I am just going to use the root of the user. If you ever need to get to that root in the Finder, click Cmd-Shift-H and it will bring you there.

$ virtualenv music-venues-app

cd to that folder and turn on the virtual environment with the . bin/activate command.

$ cd music-venues-app
$ . bin/activate

We are going to create a git repository for all this. Ben's tutorial pushes to GitHub throughout, but we will do this at the end. Create the repo and cd into it.

$ git init repo
$ cd repo

Ben's tutorial uses the touch command to create new files. You can do that, but you can also just create a new file in Text Wrangler when it is necessary. We will be creating two python scripts - app.py and later freeze.py. And we'll have two html templates - index.html and detail.html.

## Installing Flask

For your application, you will need to install Flask.

$ pip install Flask

Go to Text Wrangler and create a new file. Save it in the repo directory as app.py. It is VERY important that you save things in the right place. This is the nature of using a framework.

Put this code in the app.py file and save it. This file will do all the routing for us in the application. We'll be adding to it as we go. We are importing Flask, rendering a template and creating an index page. We are also including the app/route to connect to the root of our site.

from flask import Flask

```
from flask import render_template
app = Flask(__name__)

@app.route("/")
def index():
    template = 'index.html'
    return render_template(template)
```

Now create a templates folder. You can do this through the finder or the Terminal.

```
$ mkdir templates
```

Create a file called index.html in the templates folder and put in some text like: Hello World. Make sure to save it. We are just testing to see if it works.

Add the last two lines to your app.py to instruct it to run when called.

```
from flask import Flask
from flask import render_template
app = Flask(__name__)

@app.route("/")
def index():
    template = 'index.html'
    return render_template(template)

if __name__ == '__main__':
    app.run(debug=True, use_reloader=True)
```

Save and go to the Terminal, run app.py. When we are running the app, we are simulating a python server environment that will allow the framework functions to execute.

```
$ python app.py
```

You'll see some code to indicate the app is running. Go to a browser and open:

localhost:5000

Localhost is the root of your server. This only works when you have started the virtual environment.

**Adding the Data**

You should see "Hello World" come up in your index page. Congrats. Your app is on its way. You can do this in the Finder. Remember, if you need to get to the user directory on a Mac, use Cmd-Shift-H

Make a directory called static and include your csv in there. The static directory should be below repo.

Now make the following changes to app.py. This imports the csv module. The def get_csv() function allows you to open a csv and create objects from the items.

Notice the line that has csv_file = open(csv_path, 'rU'). I added the U to take care of Unicode and newline characters in the data.

Also notice the reference to music-venue.csv. You would change this if you used a different csv.

```
import csv
from flask import Flask
from flask import render_template
app = Flask(__name__)

def get_csv():
    csv_path = './static/music-venue.csv'
    csv_file = open(csv_path, 'rU')
    csv_obj = csv.DictReader(csv_file)
    csv_list = list(csv_obj)
    return csv_list

@app.route("/")
def index():
    template = 'index.html'
    object_list = get_csv()
    return render_template(template, object_list=object_list)

if __name__ == '__main__':
    app.run(debug=True, use_reloader=True)
```

Save it.

Then go to the index.html file and remove “Hello World”.

Add this to it. This will check to see if any of the data is working.

```
<!doctype html>
<html lang="en">
   <head></head>
   <body>
      <h1>Austin Music Venues</h1>
      {{ object_list }}
   </body>
</html>
```

The code in the curly braces indicates a call to the csv. Reference that in def index() function in app.py

If the app is still running, go to the browser at localhost:5000 and refresh. You should see the data all jumbled on the screen.

**Formatting the Data on the Page**

Now we will work to format it better. Change index.html to look like this.

```
<!doctype html>
<html lang="en">
   <head>

   </head>
   <body>

      <h1>Austin Music Venues</h1>
      <table border=1 cellpadding=7>
         <tr>
            <th>Name</th>
            <th>Website</th>
            <th>Address</th>
            <th>Phone</th>
            <th>Twitter</th>
         </tr>
         {% for obj in object_list %}
         <tr>
```

```
            <td><a href="{{ obj.id }}/">{{ obj.name }}</a></td>
            <td><a href="{{ obj.website }}" target="_blank">{{
obj.website }}</a></td>
            <td>{{ obj.address }}</td>
            <td>{{ obj.phone }}</td>
            <td><a href="http://www.twitter.com/{{ obj.twitter }}"
target="_blank">{{ obj.twitter }}</a></td>
          </tr>
          {% endfor %}
        </table>


    </body>
</html>
```

We are creating a table that includes all the items in the csv. The items in the double curly braces represent reading the elements from the csv data. Python/Flask provides the loop right in the html to go through the csv, writing each element as items in the table. I have also added the ability to include links for the obj.id that will take you to another page (we haven't made yet) and hrefs for the website and twitter.

You should be able to reload the localhost:5000 page and see the complete table!

**Austin Music Venues**

| Name | Website | Address | Phone | Twitter |
|---|---|---|---|---|
| 290 West | http://www.austinlivemusic.com /venues.aspx?VenueName=290%20West%20Club | 12013 W Highway 290- Austin- TX 78737 | (512) 288-0808 | |
| Abel's on the Lake | http://abels.com/lake/ | 3825 Lake Austin Blvd- Austin- TX 78703 | (512) 904-0570 | |
| ACL live at Moody Gardens Theater | http://acl-live.com/ | 310 Willie Nelson Blvd Austin- Texas 78701 | (512)225-7999 | @acllive |
| Amaya's Taco Village | http://www.amayastacovillage.com/ | 5804 IH-35 Austin TX 78751 | 512-458-2531 | @amaystacovilla |
| Amped Austin | http://www.ampedaustin.com | 300 E 6th St- Austin- TX 78701 | (512) 669-3897 | @ampedaustin |
| Anderson Mill Tavern | http://andersonmilltavern.com | 10401 Anderson Mill Rd- Austin- TX 78750 | (512) 918-1599 | @andersonmilltav |
| Auditorium Shores | http://www.austintexas.gov/department/parks-and-recreation | 800 W Riverside Dr- Austin- TX 78704 | (512) 974-6700 | |
| Austin 360 Amiptheater | http://austin360amphitheater.com/ | 9201 Circuit of the Americas Blvd- Del Valle- TX 78617 | (512) 301-6600 | @austin360amp |
| Austin Beer Garden Brewing Co. | http://theabgb.com/ | 1305 W Oltorf St- Austin- TX 78704 | (512) 298-2242 | @theabgb |
| Austin Beerworks | http://austinbeerworks.com/ | 3009 Industrial Terrace- Austin- TX 78758 | (512) 821-2494 | @austinbeerworks |

**Adding the Detail Pages**

Next we want those links in the name column to work. They will go to a special page for each venue.

In app.py, include this function for the detail section. Do this below the index function but before the final if statement.

```
@app.route('/<row_id>/')
def detail(row_id):
    template = 'detail.html'
    object_list = get_csv()
    for row in object_list:
        if row['id'] == row_id:
            return render_template(template, object=row)
```

We are creating a row id to correspond to the url, loop through the csv to match the url and pass that matching row to the detail template.

Create a file in your templates directory called detail.html and include this in it.

```
<!doctype html>
<html lang="en">
    <head>

    </head>
    <body>
            <h1>{{ object.name }}</h1>
        <p>Web: <a href="{{ object.website }}" target="_blank">{{
object.website }}</a></p>
        <p>Address: {{ object.address }}</p>
        <p>Phone: {{ object.phone }}</p>
        <p>Twitter: <a href="http://twitter.com/{{ object.twitter }}"
target="_blank">{{ object.twitter }}</a></p>


    </body>
```

</html>

Save everything. Restart the server (ctrl-C if it is still running) with app.py. Go back to localhost:5000 and refresh. You should see the full table there and be able to click on one of the names to get to its page. The url for the details pages is localhost:5000/1/ … etc. You should see venue pages that look something like this.


Of course, you can, and should add better styling and layout for these pages. This tutorial just goes through the functionality.

Ben's tutorial also includes the ability to handle errors in the url. The entire app.py should look like this now. See the abort import and statement in the detail function.

```
import csv
from flask import Flask
from flask import abort
from flask import render_template
app = Flask(__name__)

def get_csv():
    csv_path = './static/music-venue.csv'
    csv_file = open(csv_path, 'rU')
    csv_obj = csv.DictReader(csv_file)
    csv_list = list(csv_obj)
    return csv_list

@app.route("/")
def index():
    template = 'index.html'
    object_list = get_csv()
    return render_template(template, object_list=object_list)

@app.route('/<row_id>/')
```

```
def detail(row_id):
    template = 'detail.html'
    object_list = get_csv()
    for row in object_list:
        if row['id'] == row_id:
            return render_template(template, object=row)
    abort(404)

if __name__ == '__main__':
    app.run(debug=True, use_reloader=True)
```

Try choosing another number for a detail page that doesn't exist, like localhost:5000/9999/

**Adding the Maps**

This is a great app. But it could be even better if we could map those items. We are going to create a map for the index page that maps the location for all the venues and then individual maps for each page for each venue. Bet you're glad you aren't coding each one.

For this, we are going to include a nice JavaScript library for mapping called Leaflet.js.

Modify index.html to look like this. It includes the leaflet libraries in the head, the place where the map is drawn in the page (div) and the script to create it. The script relies on the x and y coordinates in the data and it even includes the name and link to the individual page in the pin popup.

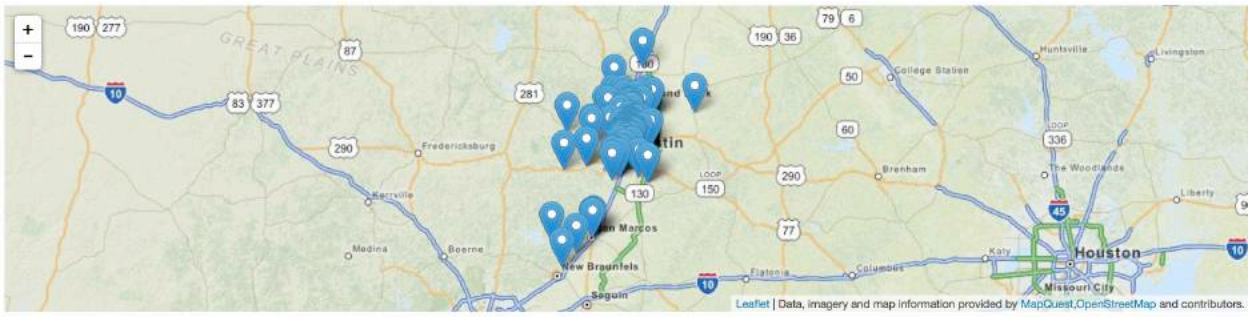
```
<!doctype html>
<html lang="en">
    <head>
    <link rel="stylesheet" href="http://cdn.leafletjs.com/leaflet-
0.7.2/leaflet.css" />
    <script type="text/javascript" src="http://cdn.leafletjs.com/leaflet-
0.7.2/leaflet.js?2"></script>
    </head>
    <body>
        <div id="map" style="width:100%; height:300px;"></div>
        <h1>Austin Music Venues</h1>
        <table border=1 cellpadding=7>
            <tr>
```

```
            <th>Name</th>
            <th>Website</th>
            <th>Address</th>
            <th>Phone</th>
            <th>Twitter</th>
        </tr>
        {% for obj in object_list %}
        <tr>
            <td><a href="{{ obj.id }}/">{{ obj.name }}</a></td>
            <td><a href="{{ obj.website }}" target="_blank">{{
obj.website }}</a></td>
            <td>{{ obj.address }}</td>
            <td>{{ obj.phone }}</td>
            <td><a href="http://www.twitter.com/{{ obj.twitter }}"
target="_blank">{{ obj.twitter }}</a></td>
        </tr>
        {% endfor %}
    </table>

    <script type="text/javascript">
        var map = L.map('map').setView([30.25, -97.75], 8);
        var mapquestLayer = new
L.TileLayer('http://{s}.mqcdn.com/tiles/1.0.0/map/{z}/{x}/{y}.png', {
            maxZoom: 18,
            attribution: 'Data, imagery and map information provided by <a
href="http://open.mapquest.co.uk" target="_blank">MapQuest</a>,<a
href="http://www.openstreetmap.org/"
target="_blank">OpenStreetMap</a> and contributors.',
            subdomains: ['otile1','otile2','otile3','otile4']
        });
        map.addLayer(mapquestLayer);
        var data = {
          "type": "FeatureCollection",
          "features": [
            {% for obj in object_list %}
            {
              "type": "Feature",
              "properties": {
                "name": "{{ obj.animalid }}",
                "id": "{{ obj.id }}"
              },
              "geometry": {
                "type": "Point",
                "coordinates": [{{ obj.x }}, {{ obj.y }}]
```

```
            }
          }{% if not loop.last %},{% endif %}
          {% endfor %}
        ]
      };
      var dataLayer = L.geoJson(data, {
        onEachFeature: function(feature, layer) {
          layer.bindPopup(
            '<a href="' + feature.properties.id + '/">' +
              feature.properties.animalid +
            '</a>'
          );
        }
      });
      map.addLayer(dataLayer);
    </script>
  </body>
</html>
```

Reload the page and click a pin.




## Austin Music Venues

| Name | Website | Address | Phone | Twitter |
|---|---|---|---|---|
| 290 West | http://www.austinlivemusic.com /venues.aspx?VenueName=290%20West%20Club | 12013 W Highway 290- Austin- TX 78737 | (512) 288-0808 | |
| Abel's on the Lake | http://abels.com/lake/ | 3825 Lake Austin Blvd- Austin- TX 78703 | (512) 904-0570 | |
| ACL live at Moody Gardens Theater | http://acl-live.com/ | 310 Willie Nelson Blvd Austin- Texas 78701 | (512)225-7999 | @acllive |
| Amaya's Taco Village | http://www.amayastacovillage.com/ | 5804 IH-35 Austin TX 78751 | 512-458-2531 | @amaystacovilla |
| Amped Austin | http://www.ampedaustin.com | 300 E 6th St- Austin- TX 78701 | (512) 669-3897 | @ampedaustin |

### Adding the Map to the Detail Page

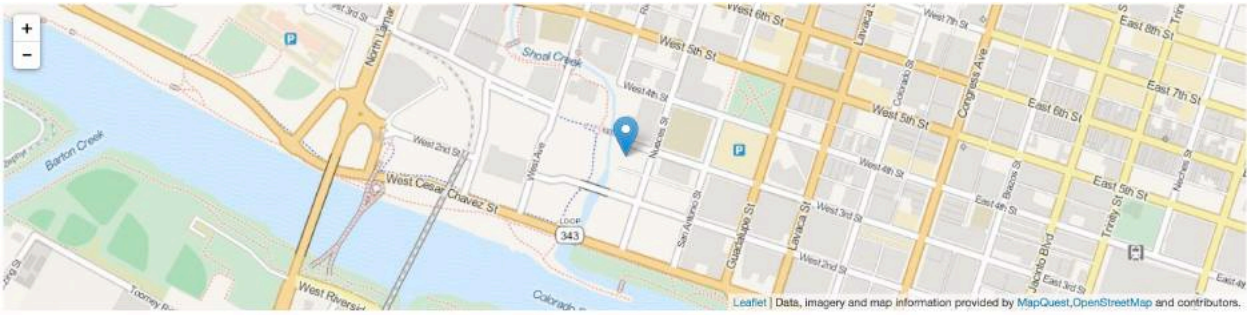
To add the map to the detail page, do roughly the same thing to detail.html as we did to index.html.

```
<!doctype html>
<html lang="en">
    <head>
        <link rel="stylesheet" href="http://cdn.leafletjs.com/leaflet-
0.7.2/leaflet.css" />
        <script type="text/javascript" src="http://cdn.leafletjs.com/leaflet-
0.7.2/leaflet.js"></script>
    </head>
    <body>
    <div id="map" style="width:100%; height:300px;"></div>
        <h1>{{ object.name }}</h1>
        <p>Web: <a href="{{ object.website }}" target="_blank">{{
object.website }}</a></p>
        <p>Address: {{ object.address }}</p>
        <p>Phone: {{ object.phone }}</p>
        <p>Twitter: <a href="http://twitter.com/{{ object.twitter }}"
target="_blank">{{ object.twitter }}</a></p>
    <script type="text/javascript">
            var map = L.map('map').setView([{{ object.y }}, {{ object.x }}],
16);
            var mapquestLayer = new
L.TileLayer('http://{s}.mqcdn.com/tiles/1.0.0/map/{z}/{x}/{y}.png', {
                maxZoom: 18,
                attribution: 'Data, imagery and map information provided by <a
href="http://open.mapquest.co.uk" target="_blank">MapQuest</a>,<a
href="http://www.openstreetmap.org/"
target="_blank">OpenStreetMap</a> and contributors.',
                subdomains: ['otile1','otile2','otile3','otile4']
            });
            map.addLayer(mapquestLayer);
            var marker = L.marker([{{ object.y }}, {{ object.x
}}]).addTo(map);
        </script>


    </body>
</html>
```

Reload the page and you should be able to go to one of the detail pages and see a map. For both files, you can modify the script to identify the center

latitude and longitude and to adjust the initial zoom level.




**Austin Music Hall**

Web: http://musichallaustin.com/

Address: 208 Nueces St- Austin- TX 78701

Phone: (512) 298-1777

Twitter: @austinmusichall

## Creating the Static Site

This site is awesome, but we want to host it on the Web. We could try to find a host that could support a Python/Flask environment. We need to do that if we are creating a site that users actively contribute to. But this site is static. We might update venues occasionally, but we'd update the csv and then rerun the app to do that. For this app, we can generate a static site that creates all our pages, and then we can host it on any server (like our Bluehost accounts) that supports HTML, CSS and JavaScript.

To do this, we will use Frozen Flask, a library that saves every page in your app as a flat file. In the Terminal (use Ctrl-C to exit your app, if necessary), run:

$ pip install Frozen-Flask
(you might need to sudo this if you get a permission error)

Create a new file called freezy.py in repo directory. Put this code in it.

```
from flask_frozen import Freezer
from app import app, get_csv
freezer = Freezer(app)

@freezer.register_generator
```

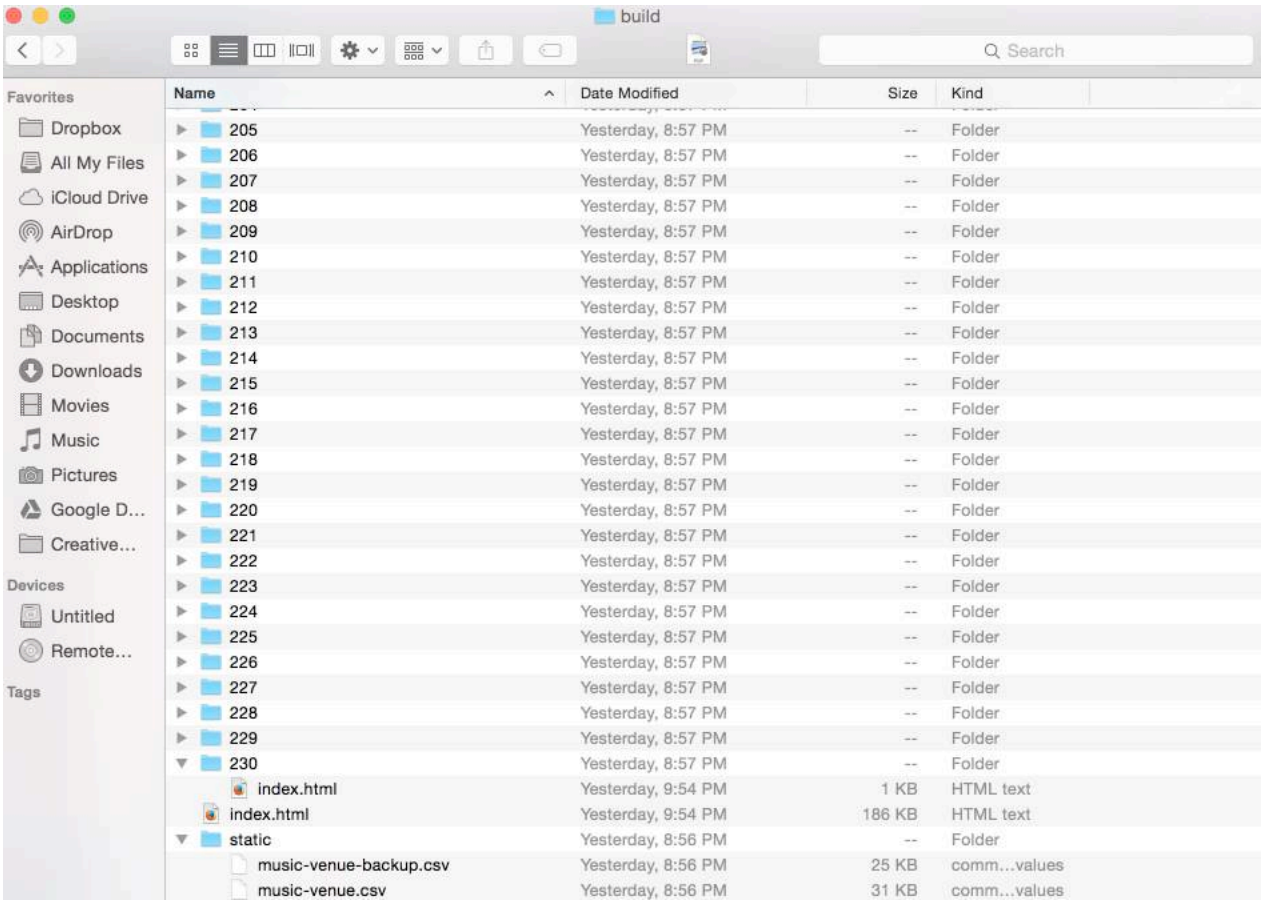
```
def detail():
    for row in get_csv():
        yield {'row_id': row['id']}

if __name__ == '__main__':
    freezer.freeze()
```

Run freeze.py on the command line.

```
$ python freeze.py
```

This creates a build folder in which all your flat files are located. Look for the build folder in your application in the Finder. The folder has index.html and folders with an index.html for each venue. It also includes the static folder with the csv in it. You can fetch these folders/files to a folder on your website to host the application there!



Of course, you should now go and commit this entire repo to a new GitHub repository, so you can share it with the world!

Now you can see how all the different things we covered this semester come together. Think about how you could use this process for other types of data, other types of storytelling. Completed files at various stages of this process can be found at github.com/cindyroyal/music-venue-app.