Flask Installation

- Flask is a popular Python web framework, meaning it is a third-party Python library used for developing web applications.

```bash
$ pip install Flask
mkdir flaskDirectory

cd flaskDirectory
create app.py
Add data.csv

mkdir templates
cd templates
create index.html
```
from flask import Flask
app = Flask(__name__)  # creates the Flask instance

@app.route("/")  # creates a simple route so you can see the application working. It creates a connection between the URL /hello and a function that returns a response, the string 'Hello, World!' in this case.

def hello():
    return "Hello World!"

if __name__ == "__main__": app.run()
from flask import Flask
app = Flask(__name__)

@app.route("/")
def index():
    return "Index!"

@app.route("/hello")
def hello():
    return "Hello World!"

@app.route("/members")
def members():
    return "Members"

if __name__ == "__main__": app.run()
$ python app.py

Serving Flask app "dirName"

Environment: development

Debug mode: on

Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

Restarting with stat

Debugger is active!

Debugger PIN: 855-212-761
```python
@app.route("/")
def index():
    data = pd.read_csv('data2.csv')
    chart_data = data.to_dict(orient='records')
    chart_data = json.dumps(chart_data, indent=2)
    data = {'chart_data': chart_data}
    return render_template("index.html", data=data)

if __name__ == "__main__":
    app.run(debug=True)
```
Load Data to Front-End

<!-- Load the d3.js library -->
<script src="http://d3js.org/d3.v3.min.js"></script>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>

<script>

var data = {{ data.chart_data | safe }}

console.log(data);

// Set the dimensions of the canvas / graph
var margin = {top: 30, right: 20, bottom: 30, left: 50},
    width = 600 - margin.left - margin.right,
    height = 270 - margin.top - margin.bottom;

</script>
```python
import json

from flask import Flask, render_template, request, redirect, Response, jsonify
import pandas as pd

app = Flask(__name__)

@app.route('/', methods = ['POST', 'GET'])
def index():
    #df = pd.read_csv('data.csv').drop('Open', axis=1)
    global df

    data = df[['date', 'close']]
    chart_data = data.to_dict(orient='records')
    chart_data = json.dumps(chart_data, indent=2)
    data = {'chart_data': chart_data}
    return render_template('index.html', data=data)

if __name__ == '__main__':
df = pd.read_csv('data2.csv')
app.run(debug=True)
```
```python
@app.route("/", methods = ['POST', 'GET'])
def index():
    
    #df = pd.read_csv('data.csv').drop('Open', axis=1)
    global df
    
    if request.method == 'POST':
        data = df[['date', 'open']]  
        data = data.rename(columns={'open': 'close'})
        print(data)
        print("Hello World!")
        chart_data = data.to_dict(orient='records')
        chart_data = json.dumps(chart_data, indent=2)
        data = {'chart_data': chart_data}
        # data = {'chart_data': chart_data}
        return jsonify(data) # Should be a json string

    data = df[['date', 'close']]  
    chart_data = data.to_dict(orient='records')
    chart_data = json.dumps(chart_data, indent=2)
    data = {'chart_data': chart_data}
    return render_template("index.html", data=data)
```
// ** Update data section (Called from the onclick)
function updateData() {

    // Get the data again
    // Request the "" page and send some additional data along
    $.post('', {'data': 'received'}, function(data_infunc){
        // console.log({data_infunc})

        data2 = JSON.parse(data_infunc.chart_data)
        console.log(data2);
        data2.forEach(function(d) {
            d.date = parseDate(d.date);
            d.close = +d.close;
        });
    });
}
MDS is for irregular structures
- scattered points in high-dimensions (N-D)
- adjacency matrices

Maps the distances between observations from N-D into low-D (say 2D)
- attempts to ensure that differences between pairs of points in this reduced space match as closely as possible

The input to MDS is a distance (similarity) matrix
- actually, you use the *dissimilarity* matrix because you want similar points mapped closely
- dissimilar point pairs will have greater values and map farther apart
The Dissimilarity Matrix

Data Matrix

<table>
<thead>
<tr>
<th>point</th>
<th>attribute1</th>
<th>attribute2</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>x2</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>x3</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>x4</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

Dissimilarity Matrix (with Euclidean Distance)

<table>
<thead>
<tr>
<th></th>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>x4</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x2</td>
<td>3.61</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x3</td>
<td>2.24</td>
<td>5.1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>x4</td>
<td>4.24</td>
<td>1</td>
<td>5.39</td>
<td>0</td>
</tr>
</tbody>
</table>
MDS turns a distance matrix into a network or point cloud
- correlation, cosine, Euclidian, and so on

Suppose you know a matrix of distances among cities

<table>
<thead>
<tr>
<th></th>
<th>Chicago</th>
<th>Raleigh</th>
<th>Boston</th>
<th>Seattle</th>
<th>S.F.</th>
<th>Austin</th>
<th>Orlando</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chicago</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Raleigh</td>
<td>641</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boston</td>
<td>851</td>
<td>608</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Seattle</td>
<td>1733</td>
<td>2363</td>
<td>2488</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S.F.</td>
<td>1855</td>
<td>2406</td>
<td>2696</td>
<td>684</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Austin</td>
<td>972</td>
<td>1167</td>
<td>1691</td>
<td>1764</td>
<td>1495</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Orlando</td>
<td>994</td>
<td>520</td>
<td>1105</td>
<td>2565</td>
<td>2458</td>
<td>1015</td>
<td>0</td>
</tr>
</tbody>
</table>
RESULT OF MDS
COMPARE WITH REAL MAP
MDS Algorithm

- Task:
  - Find that configuration of image points whose pairwise distances are most similar to the original inter-point distances !!!

- Formally:
  - Define: $D_{ij} = \| x_i - x_j \|_D$, $d_{ij} = \| y_i - y_j \|_d$

- Claim: $D_{ij} \equiv d_{ij}$ $\forall i, j \in [1, n]$

- In general: an exact solution is not possible !!!

- Inter Point distances $\rightarrow$ invariance features
MDS Algorithm

**Strategy (of metric MDS):**

- iterative procedure to find a good configuration of image points
  
  1) Initialization
     → Begin with some (arbitrary) initial configuration

  2) Alter the image points and try to find a configuration of points that minimizes the following sum-of-squares error function:
MDS Algorithm

Strategy (of metric MDS):

- iterative procedure to find a good configuration of image points
  
  1) Initialization
  → Begin with some (arbitrary) initial configuration
  
  2) Alter the image points and try to find a configuration of points that minimizes the following sum-of-squares error function:

\[
E = \sum_{i<j}^N (D_{ij} - d_{ij})^2
\]
Spring-like system

- insert springs within each node
- the length of the spring encodes the desired node distance
- start at an initial configuration
- iteratively move nodes until an energy minimum is reached
Spring-like system

- insert springs within each node
- the length of the spring encodes the desired node distance
- start at an initial configuration
- iteratively move nodes until an energy minimum is reached

Vertex layout by correlations.
Uses of MDS

Distance (similarity) metric

- Euclidian distance (best for data)
- Cosine distance (best for data)
- |1-correlation| distance (best for attributes)
- use 1-correlation to move correlated attribute points closer
- use | | if you do not care about positive or negative correlations
MDS Examples
sklearn.manifold.MDS

class sklearn.manifold.MDS(n_components=2, metric=True, n_init=4, max_iter=300, verbose=0, eps=0.001, n_jobs=1, random_state=None, dissimilarity='euclidean')

sklearn.manifold.MDS(

  n_components=2,
  metric=True,

  n_init=4,   Number of time the smacof algorithm will be run with different initialisation.  
              The final results will be the best output of the n_init consecutive runs in terms of stress.
  
  max_iter=300, Maximum number of iterations of the SMACOF algorithm for a single run

  verbose=0,

  eps=0.001, relative tolerance w.r.t stress to declare converge

  n_jobs=1,

  random_state=None,

  dissimilarity='euclidean') Which dissimilarity measure to use. Supported are ‘euclidean’ and ‘precomputed’.

The SMACOF (Scaling by MAjorizing a COmplicated Function) algorithm is a multidimensional scaling algorithm which minimizes an objective function (the stress) using a majorization technique.