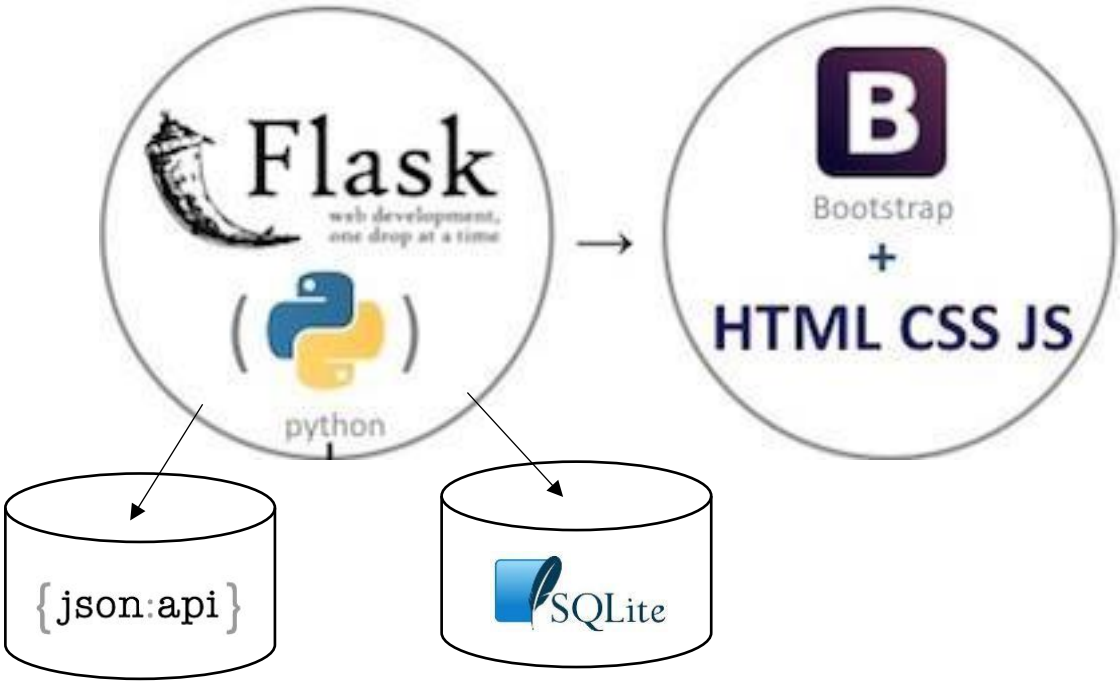
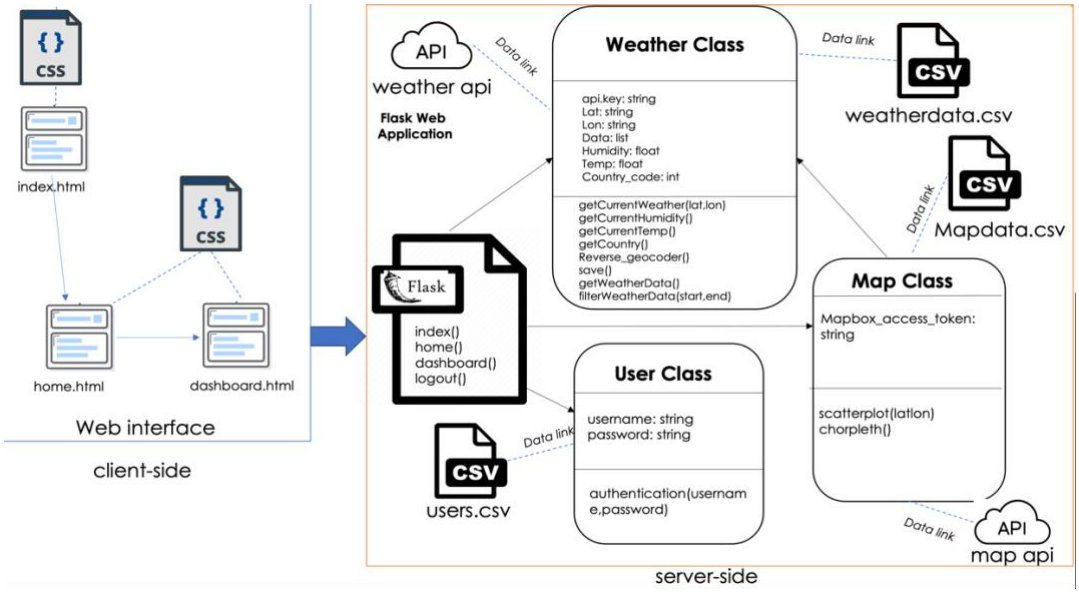
Flask Web Application



- Allows you to use python server-side to manage the content displayed on a web page.
- It allows you to display content based on data stored in text files, csv or a database (SQLlite).
- Access data from api's and display this data in a web page.
- Generate maps and graphs.
- Create web-based forms and use python to process the data.
- You can create your own classes for managing different features, for example users.py to manage login authentication.
- Use HTML, CSS and JavaScript to create interactive web-based interface.

Class Diagram



File Structure

Flask web applications will have a file structure as follows:

Flaskapp

<templates> base.html index.html home.html
dashboard.html

<static>

<CSS>

signin.css
style.css

<images>

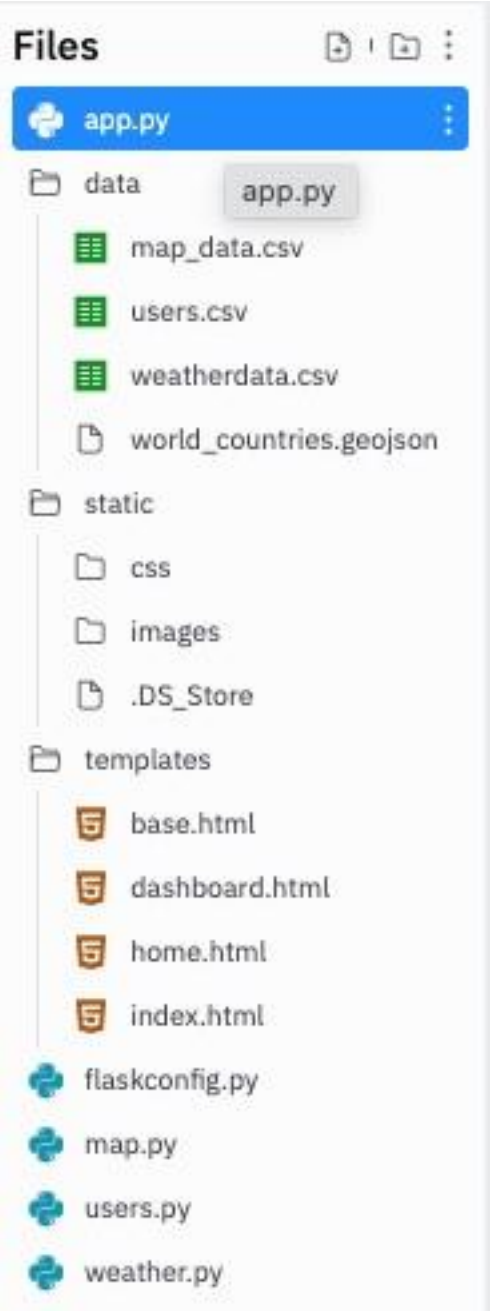
logo.png

<data>

users.csv
weatherdata.csv
world_countries.geojson
map_data.csv

<venv>

app.py (Flask python file)
users.py (class)
map.py (class)
weather.py (class)



You can code a solution on replit or use pycharm. However you will need to configure pycharm to run Flask applications. How to set up pycharm to run Flask applications
https://medium.com/@mushtaque87/flask-in-pycharm-community-editionc0f68400d91e

Flask Web Application

Flask app.py

Final code:

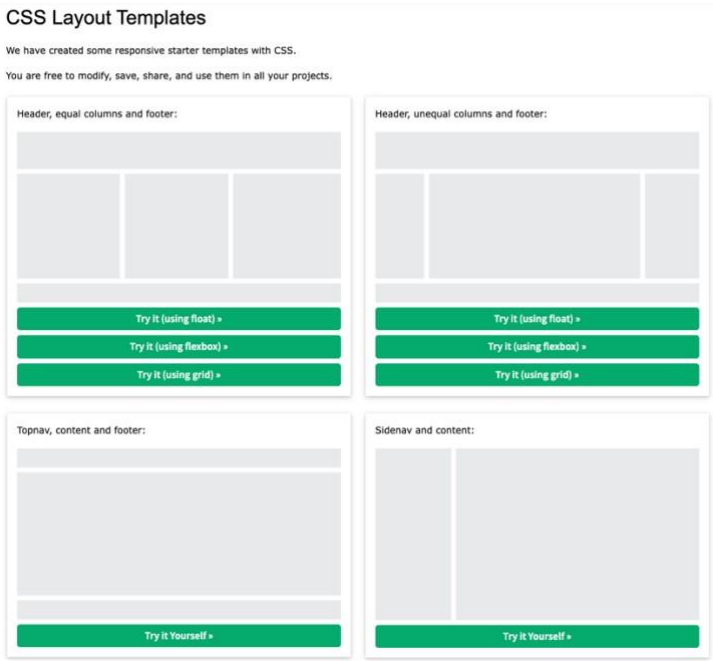
```
from flask import Flask, render_template, request, url_for, redirect, session
from weather import Weather from users import Users
from map import Map

app = Flask(__name__)

app.secret_key = 'any random string'

#Login interface
@app.route('/',methods=('GET', 'POST'))
def index():
    ##Initalise user class object
            user_login = Users()
            if request.method == 'POST':
                    #store username as session variable
                    session["username"] =
                    request.form.get("username")
                    password = request.form['password']

            if user_login.authenticate(session["username"],password):
                    return redirect(url_for('home'))
            else:
                    #Remove session variable
                    session.pop('username', None)
                    return redirect(url_for('index'))

    return render_template('index.html')

#Home page interface
@app.route('/home')
def home():
    #Authenticates session[username]
    if 'username' in session:
            return render_template('home.html')
    return redirect(url_for('index'))

#Dashboard interface
@app.route('/dashboard',methods=('GET', 'POST'))
def dashboard():
            #Initalise class objects
            data = Weather()
            my_map = Map()
            #Set default settings
            session["start"] = '00:00:00'
            session["end"] ='00:00:00'
```

```
        # Authenticates session[username]
        if 'username' in session:

                if request.method == 'POST':          #Check clicked submit button

                        if request.form['btn'] == 'Update':
                                session["lat"] = request.form.get("lat")
                                session["lon"] = request.form.get("lat")

                        else:

                                session["start"] = request.form.get("start")
                                session["end"] = request.form.get("end")

                        # Weather data
                        data.getCurrentWeather(session["lat"], session["lon"])
                        data.save()
                        humidity = data.getCurrentHumidity()
                        temperature = data.getCurrentTemp()

                        # Interactive weather map
                        latlon = [(session["lat"], session["lon"] )]
                        html_string                                 =
                        my_map.scatterplot_map(latlon)
                        # html_string = my_map.choropleth()

                        #Filter data and display in table
                        filter = data.filterWeatherData(session["start"], session["end"])

                        # weather graph
                        legend = 'Monthly Data'
                        labels = ["January", "February", "March", "April", "May", "June", "July",
                        "August"]           values = [10, 9, 8, 7, 6, 4, 7, 8]

                        return render_template('dashboard.html', humidity=humidity,
                temp=temperature, values=values, labels=labels, legend=legend,
                div_placeholder=html_string, filter=[filter.to_html()], titles=[''])
        return render_template('dashboard.html')
    return redirect(url_for('index'))

#The application also contains a logout () view function that pops up the 'username'
session variable.Therefore, the ' /' URL displays the start page again.
@app.route('/logout') def
logout():
        # remove the username from the session if it is there
        session.pop('username', None)
        return redirect(url_for('index'))

if __name__ == "__main__":
    app.run(host='0.0.0.0', port='8080', debug=True)
```

Creating Templates

Now creating a folder **templates** and inside it creating three HTML files, *login.html*, *home.html* and dashboard

A good place to start for designing a layout use a CSS template.
Go to https://www.w3schools.com/css/css_templates.asp

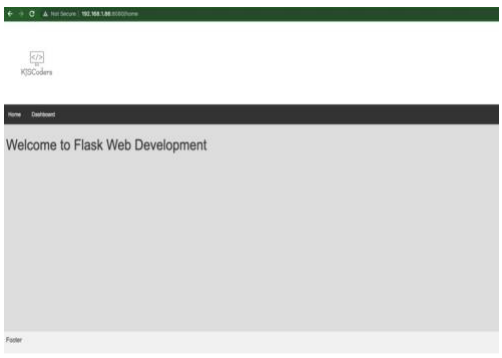


It will show the <style> required and the structure required for the layout.

home.html

home.html

```
<!doctype html>
    <head>
    <!-- Custom styles for this template -->
        <link href="{{ url_for('static', filename='css/style.css') }}" rel="stylesheet">
        <!-- Latest compiled and minified CSS -->
        <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
        <!-- Latest compiled JavaScript -->
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
        <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>
        <script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.5.0/Chart.min.js"></script>

        <title>home</title>
    </head>

<body>
    <img src="{{ url_for('static', filename='images/logo.png') }}"/>

    <div class="topnav">
        <a class="nav-link active" href="{{ url_for('home')}}">Home</a>
        <a class="nav-link active" href="{{ url_for('dashboard')}}">Dashboard</a>
        <a class="nav-link active" href = '{{ url_for('logout')}}'>Log out</a>
    </div>

    <div class="content">
       <h1> Welcome to Flask Web Development</h1>
    </div>

    <div class="footer">
      <p>Footer</p>
    </div>


</body>
</html>
```

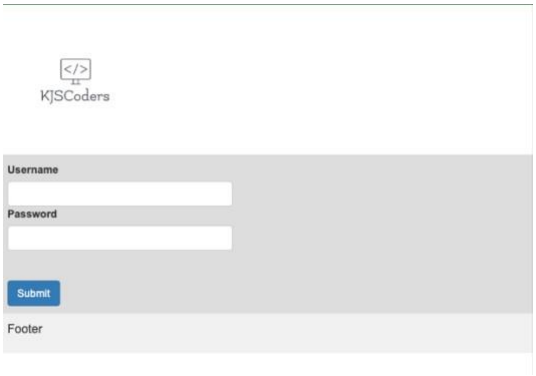


style.css

```
* {
  box-sizing: border-box;
  font-family: Arial, Helvetica, sans-serif;
}

body {
  margin: 0;
  font-family: Arial, Helvetica, sans-serif;
}

/* Style the top navigation bar */
.topnav {
  overflow: hidden;
  background-color: #333;
}

/* Style the topnav links */
.topnav a {
  float: left;
  display: block;
  color: #f2f2f2;
  text-align: center;
  padding: 14px 16px;
  text-decoration: none;
}

/* Change color on hover */
.topnav a:hover {
  background-color: #ddd;
  color: black;
}

/* Style the content */
.content {
  background-color: #ddd;
  padding: 10px;
  min-height:500px;
}

p {font-size: 16px;}

.footer {
  background-color: #f1f1f1;
  padding: 10px;
```

index.html template

```
<!doctype html>

  <head>
    <!-- Custom styles for this template -->
    <link href="{{ url_for('static', filename='css/signin.css') }}" rel="stylesheet">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
</head>


<body>

<img src="{{ url_for('static', filename='images/logo.png') }}"/>

<div class="content">
    <div class="login">

      <form method="post">
              <div class="mb-3">
                <label for="InputUsername" class="form-label">Username</label>
                <input type="text" class="form-control" id="InputUsername" name="username" value="{{ request.form['username'] }}">
                </div>

              <div class="mb-3">
                <label for="InputPassword" class="form-label">Password</label>
                <input type="password" class="form-control" id="InputPassword" name="password" value="{{ request.form['password'] }}">
              </div>
              <br>
              <button type="submit" class="btn btn-primary">Submit</button>
            </form>
    </div>
</div>

<div class="footer">
  <p>Footer</p>
</div>

  </body>
</html>
```



To allow user input you will need to use a <form> tag and use the method=”post” so that when the submit button is pressed the data is posted to the server.

```
<form method="post">
```

The label tag is used for a form label:

```
<label for="InputUsername" class="form-label">Username</label>
```

The input tag is used for a user input.

value="{{ request.form['username'] }}" is used to get the data for username.
<input type="text"> defines a **single-line text input field**:

```
<input type="text" class="form-control" id="InputUsername" name="username"
value="{{ request.form['username'] }}">
```

```
<label for="InputPassword" class="form-label">Password</label>
```

value="{{ request.form['password'] }}" is used to get the data for password. <input type="password"> defines a **password field** and characters in a password field are masked (shown as asterisks or circles).

```
<input type="password" class="form-control" id="InputPassword"
name="password" value="{{ request.form['password'] }}">
```

```
<button type="submit" class="btn btn-primary">Submit</button>
```

signin.css

```css
* {
  box-sizing: border-box;
  font-family: Arial, Helvetica, sans-serif;
}

body {
  margin: 0;
  font-family: Arial, Helvetica, sans-serif;
}

/* Style the content */
.content {
  background-color: #ddd;
  padding: 10px;
}

.login{
width: 300px;
}

p {font-size: 16px;}

.footer {
  background-color: #f1f1f1;
  padding: 10px;

```

Flask app.py

Session data in Python Flask

Unlike cookies, Session (session) data is stored on the server. The session is the interval at which the client logs on to the server and logs out the server. The data that is required to be saved in the session is stored in a temporary directory on the server.

Assign session IDs to sessions for each client. Session data is stored at the top of the cookie, and the server signs it in encrypted mode. For this encryption, the Flask application requires a defined SECRET_KEY.

A Session object is also a dictionary object that contains key value pairs for session variables and associated values.
For example, to set a 'username' session variable, use the following statement:

```
session["username"] = request.form.get("username")
```

```python
from flask import Flask, render_template, request, url_for, redirect, session
from weather import Weather
from users import Users
from map import Map

app = Flask(__name__)

app.secret_key = 'any random string'

#Login interface
@app.route('/',methods=('GET', 'POST'))
def index():
    ##Initalise user class object
    user_login = Users()
    if request.method == 'POST':
        #store username as session variable
        session["username"] = request.form.get("username")
        password = request.form['password']

        if user_login.authenticate(session["username"],password):
            return redirect(url_for('home'))
        else:
            #Remove session variable
            session.pop('username', None)
            return redirect(url_for('index'))

    return render_template('index.html')

#Home page interface
@app.route('/home')
def home():
    #Authenticates session[username]
    if 'username' in session:
        return render_template('home.html')
    return redirect(url_for('index'))
```

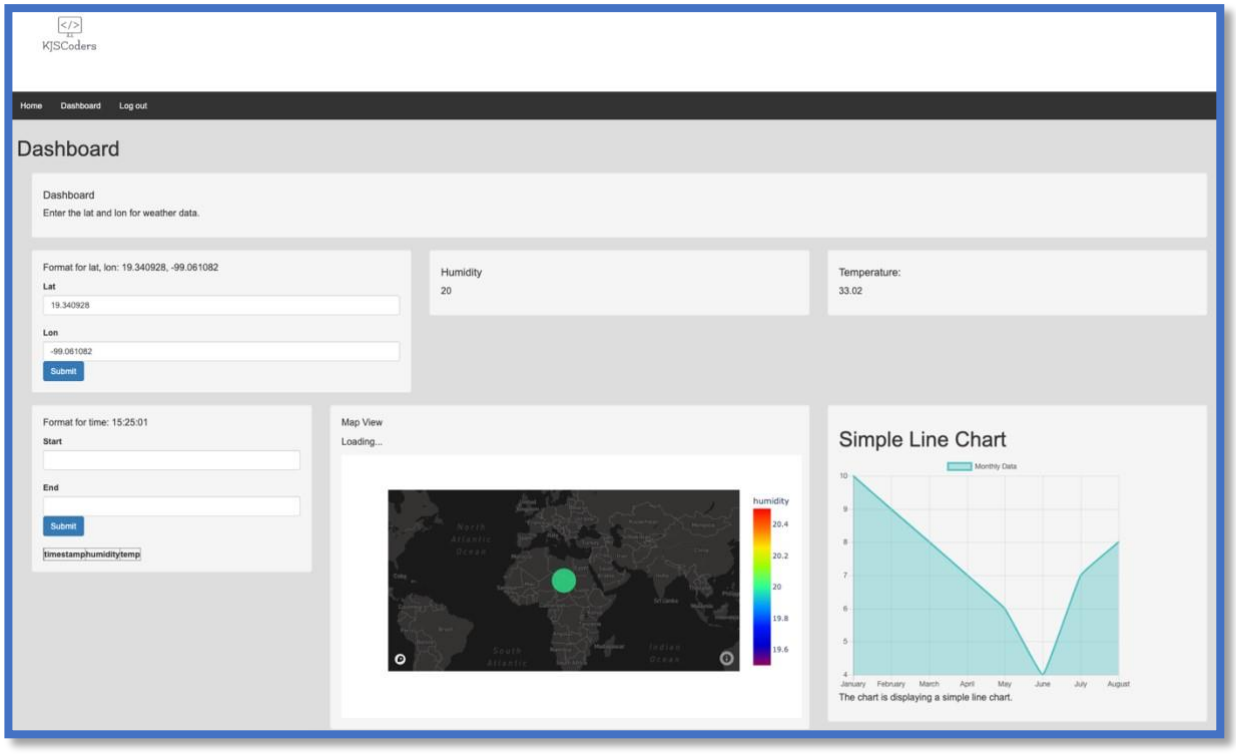
users.py class

```python
class Users():
    """Class for user login authentication"""
    def __init__(self):
        """Constructor Method"""
        self.file = open('data/users.csv', 'r')


    def authenticate(self, username, password):
        """Method to complete user login authentication"""
        all_lines = self.file.readlines()
        found = False
        for line in all_lines:
            if username in line and password in line:
                found = True
                break  # no need to check other lines
        self.file.close()
        return found
```

users.csv

```
username,password
test,test
paddington,password123
scoobydo,12345678
```

Dashboard Template



This template is adapted from:
https://www.w3schools.com/bootstrap/tryit.asp?filename=trybs_temp_analytics&stacked=h

The dashboard template requires using imported bootstrap css, jquery and JavaScript libraries.

```
<!doctype html>
    <head>
    <!-- Custom styles for this template -->
        <link href="{{ url_for('static', filename='css/style.css') }}" rel="stylesheet">
        <!-- Latest compiled and minified CSS -->
        <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
        <!-- Latest compiled JavaScript -->
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
        <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>
        <script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.5.0/Chart.min.js"></script>
        <title></title>
    </head>
```

dashboard.html template

The data that is sent via the render_template() can be displaying in the web page, for example this will display the humidity data:

{{humidity}}

We can also display data that is sent as a list using:

```
[{% for item in values %}
        {{item}},
{% endfor %}],
```

Another Example:

```
{% for i in current %}
   data - {{i}} <br>
{%endfor%}

<p> Humidity Data: {{current[0]}}</p>
<p>Temp Data: {{current[1]}}</p>
```

To display the map requires:

```
<script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
{{ div_placeholder|safe }}
```

To display a chart requires further JavaScript code and this will be explained later.

HTML code for entering the lat and lon data

```
<div class="col-sm-12">
  <div class="well">
    <h4>Dashboard</h4>
    <p>Enter the lat and lon for weather data.</p>
  </div>

  <div class="row">
    <div class="col-sm-4">
      <div class="well">
        <p>Format for lat, lon: 19.340928, -99.061082 </p>
        <form method="post">
          <div class="mb-3">
            <label for="InputLat" class="form-label">Lat</label>
            <input type="text" class="form-control" id="InputLat" name="lat" value="{{ request.form['lat'] }}">
          </div>
          <br>
          <div class="mb-3">
            <label for="InputLon" class="form-label">Lon</label>
            <input type="text" class="form-control" id="InputLon" name="lon" value="{{ request.form['lon'] }}">
          </div>
          <button type="submit" name="btn" value="Update" class="btn btn-primary">Submit</button>
        </form>
      </div>
    </div>
```

Working with data and displaying the data on a web page

Display the Humidity and Temperature data

Weather data from an api

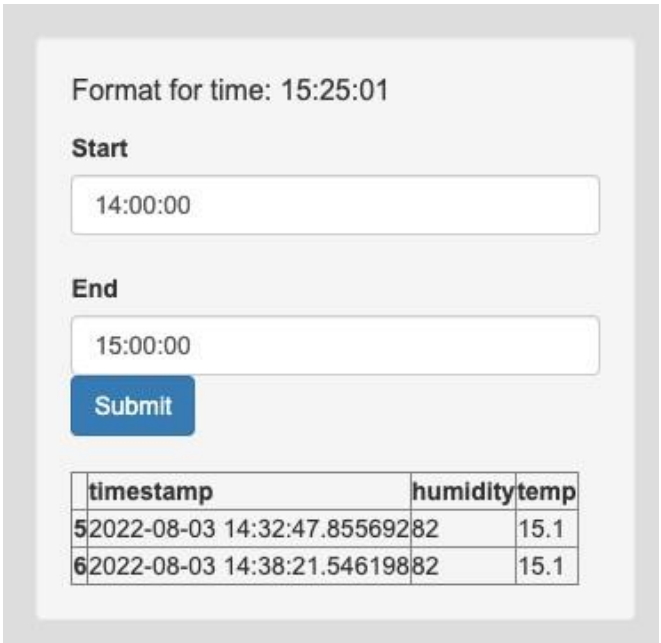
1. The **weather class** connects to openweather api
2. Gets the json file
3. Extracts the data required
4. Returns the data as a list

```
55 ▼        <div class="col-sm-4">
56 ▼          <div class="well">
57              <h4>Humidity</h4>
58              <p> {{humidity}}</p>
59            </div>
60          </div>
61 ▼        <div class="col-sm-4">
62 ▼          <div class="well">
63              <h4>Temperature: </h4>
64              <p>{{temp}}</p>
65            </div>
66          </div>
67        </div>
68
```

Form for entering range of date/times and display a list of data in a html table:

Filtering Data

This filters the data stored in the weatherdata.csv by time. This could be changed to filtering by dates.



	timestamp	humidity	temp
5	2022-08-03 14:32:47.855692	82	15.1
6	2022-08-03 14:38:21.546198	82	15.1

```
timestamp,humidity,temp,country_code
2022-08-03 12:34:03.393293,67,24.96,GB
2022-08-03 12:35:24.203859,67,24.87,GB
2022-08-03 12:37:48.337536,69,24.6,GB
2022-08-03 13:32:27.622093,66,25.68,GB
2022-08-03 13:32:37.166265,66,25.68,GB
2022-08-03 14:32:47.855692,82,15.1,MX
2022-08-03 14:38:21.546198,82,15.1,MX
2022-08-03 15:00:33.849628,79,15.27,MX
2022-08-03 15:04:04.714330,77,15.71,MX
2022-08-03 15:04:09.274849,77,15.71,MX
2022-08-03 15:22:12.161557,77,15.78,MX
2022-08-03 15:23:13.555523,77,15.78,MX
2022-08-03 15:23:18.459674,77,15.78,MX
2022-08-03 15:25:23.026511,77,15.78,MX
2022-08-03 15:26:38.934373,77,15.78,MX
2022-08-03 15:30:13.883799,77,15.78,MX
2022-08-03 15:34:07.845295,77,15.78,MX
2022-08-03 15:34:12.325609,77,15.78,MX
2022-08-03 15:35:17.305873,77,15.78,MX
2022-08-03 15:36:34.539742,77,15.78,MX
2022-08-03 15:38:32.524081,77,15.78,MX
2022-08-03 15:43:12.397120,77,15.78,MX
2022-08-03 15:44:48.982808,77,15.78,MX
2022-08-03 16:48:41.440603,64,18.57,MX
2022-08-03 16:48:46.536461,64,18.57,MX
2022-08-03 16:56:05.951841,64,18.57,MX
```

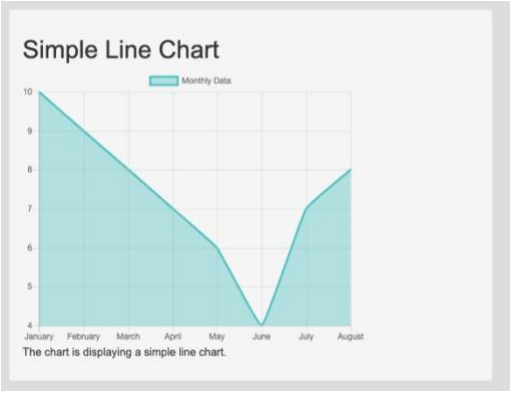
weatherdata.csv

```
<div class="row">

  <div class="col-sm-3">
    <div class="well">
      <p>Format for time: 15:25:01</p>
      <form method="post">
        <div class="mb-3">
          <label for="InputStart" class="form-label">Start</label>
          <input type="text" class="form-control" id="InputStart" name="start" value="{{ request.form['start'] }}">
        </div>
        <br>
        <div class="mb-3">
          <label for="InputEnd" class="form-label">End</label>
          <input type="text" class="form-control" id="InputEnd" name="end" value="{{ request.form['end'] }}">
        </div>

        <button type="submit" class="btn btn-primary" name="btn" value="filter_update">Submit</button>

      </form>

      <table>
      <h1>
        {%for i in filter %}
      <h2>{{titles[loop.index]}}</h2>
        {{i|safe}} {%endfor %}
      </h1>
      </table>

    </div>
  </div>
```

Display a Map:

```
    <div class="col-sm-5">
      <div class="well">
        <p>Map View</p>
        <p>Loading...</p>
         <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
        {{ div_placeholder|safe }}
      </div>
    </div>

```

Inserting a chart:



The example illustrates utilizing a simple Flask application that originates the data to be displayed by Chart.js as a simple line chart. be focusing on the Javascript side of this application

The template file (chart.html) is a combination of a number of languages:

- HTML
- Jinja2 template scripts
- Javascript

In order to use the Chart.js library, the 'Chart.min.js' file needs to be specified in the 'head' section:

<script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.5.0/Chart.min.js"></script>

The chart is then defined as a HTML5 canvas element:

```
<h1>Simple Line Chart</h1>
<!-- bar chart canvas element -->
<canvas id="myChart" width="600" height="400"></canvas>
<p id="caption">The chart is displaying a simple line chart.</p>
```

Finally, the Javascript section of this file does the following in order:

1. Defines the global parameters that apply to all charts
2. Defines the chart data for this specific chart
3. Gets the HTML canvas element
4. Creates the chart to be displayed in the canvas element

Here are the contents of the 'script' section containing the Javascript that utilizes the Chart.js library.

Most of the parameters that are set in this section are straight from the Chart.js documentation, so please refer to the documentation for descriptions of each parameter being set for this chart.

The key parameter that I changed from the default value is the only parameter defined in the global settings section ('responsive').

I recommend setting this parameter to 'false' to ensure that the size of the chart is not resized, but is maintained at 600x400px (good size for viewing on a laptop/desktop). This parameter can be easily changed, but it does provide an example of setting a global parameter that will be applied to all Chart objects in your application.

```
109 ▼      <div class="col-sm-4">
110 ▼        <div class="well">
111 ▼          <div class="chart-area">
112            <h1>Simple Line Chart</h1>
113            <!-- bar chart canvas element -->
114            <canvas id="myChart" width="500" height="400"></canvas>
115            <p id="caption">The chart is displaying a simple line chart.</p>
116            </div>
117 ▼          <script>
118            // Global parameters:
119            // do not resize the chart canvas when its container does (keep at 600x400px)
120            Chart.defaults.global.responsive = false;
121
122            // define the chart data
123 ▼          var chartData = {
124 ▼          labels : [{% for item in labels %}
125            "{{item}}",
126            {% endfor %}],
127
```

```
datasets : [{
label: '{{ legend }}',
fill: true,
lineTension: 0.1,
backgroundColor: "rgba(75,192,192,0.4)",
borderColor: "rgba(75,192,192,1)",
borderCapStyle: 'butt',
borderDash: [],
borderDashOffset: 0.0,
borderJoinStyle: 'miter',
pointBorderColor: "rgba(75,192,192,1)",
pointBackgroundColor: "#fff",
pointBorderWidth: 1,
pointHoverRadius: 5,
pointHoverBackgroundColor: "rgba(75,192,192,1)",
pointHoverBorderColor: "rgba(220,220,220,1)",
pointHoverBorderWidth: 2,
pointRadius: 1,
pointHitRadius: 10,

data : [{% for item in values %}
{{item}},
{% endfor %}],
spanGaps: false
}]
}
// get chart canvas
var ctx = document.getElementById("myChart").getContext("2d");
// create the chart using the chart canvas
var myChart = new Chart(ctx, {type: 'line',data: chartData,});
</script>
```

Flask app.py

Session variables are used again to store lat, lon and start, end data input from the forms. When the page is refreshed by clicking on a submit buttons data can be updated successfully. To send data back to the web page is performed when using **render_template()**

```
return render_template('dashboard.html',
        humidity=humidity,
        temp=temperature,
        values=values,
        labels=labels, legend=legend,
        div_placeholder=html_string,
        filter=[filter.to_html()], titles=[''])
```

Final Code for Dashboard in Flask.py

#Dashboard interface

```
#Dashboard interface
@app.route('/dashboard',methods=('GET', 'POST'))
def dashboard():
    #Initalise class objects
    data = Weather()
    my_map = Map()

    #Set default settings
    session["start"] = '00:00:00'
    session["end"] ='00:00:00'

    # Authenticates session[username]
    if 'username' in session:

        if request.method == 'POST':
            #Check clicked submit button
            if request.form['btn'] == 'Update':
                session["lat"] = request.form.get("lat")
                session["lon"] = request.form.get("lon")

            else:
                session["start"] = request.form.get("start")
                session["end"] = request.form.get("end")

            # Weather data
            data.getCurrentWeather(session["lat"], session["lon"])
            data.save()
            humidity = data.getCurrentHumidity()
            temperature = data.getCurrentTemp()
```

```
            # Interactive weather map
            latlon = [(session["lat"], session["lon"] )]
            html_string = my_map.scatterplot_map(latlon)
            # html_string = my_map.choropleth()

            #Filter data and display in table
            filter = data.filterWeatherData(session["start"], session["end"])

            # weather graph
            legend = 'Monthly Data'
            labels = ["January", "February", "March", "April", "May", "June", "July", "August"]
            values = [10, 9, 8, 7, 6, 4, 7, 8]

            return render_template('dashboard.html', humidity=humidity, temp=temperature, values=values,
                                   labels=labels, legend=legend, div_placeholder=html_string,
                                   filter=[filter.to_html()], titles=[''])
        return render_template('dashboard.html')
    return redirect(url_for('index'))
```

weather class

```
import json
import requests
from datetime import datetime
from csv import writer,reader
import reverse_geocoder as rg
import pandas as pd
from datetime import datetime

class Weather():
  """Class to access weather data"""
  def __init__(self):
    """Constructor Method"""
    self.api_key = "efb67b5073bab84b1dc7575bedd41c27"

  def getCurrentWeather(self, lat, lon):
    """Method to get current weather data based on lat and lon"""
    self.lat, self.lon = lat, lon
    # Current Weather
    url = "https://api.openweathermap.org/data/2.5/onecall?lat=%s&lon=%s&appid=%s&units=metric" % (
      self.lat, self.lon, self.api_key)
    response = requests.get(url)
    self.data = json.loads(response.text)

    #get specific data from the api json file
    weather = self.data.get("current")
    self.humidity_data = weather.get('humidity')
    self.temp_data = weather.get('temp')

    #get cc: ISO 3166-1 alpha-2 country code
    self.reverse_geocoder()
```

```
  def getCurrentHumidity(self):
    """Get Method to return current humidity"""
    return self.humidity_data

  def getCurrentTemp(self):
    """Get Method to return current temp"""
    return self.temp_data

  def getCountry(self):
    """Get Method to return current country"""
    return self.country_code


  def reverse_geocoder(self):
    """Get Method get counrty based on lat and lon"""
    coordinates = (self.lat, self.lon)
    dict = rg.search(coordinates)
    # cc: ISO 3166-1 alpha-2 country code
    print(dict)
    self.country_code = dict[0]['cc']
```

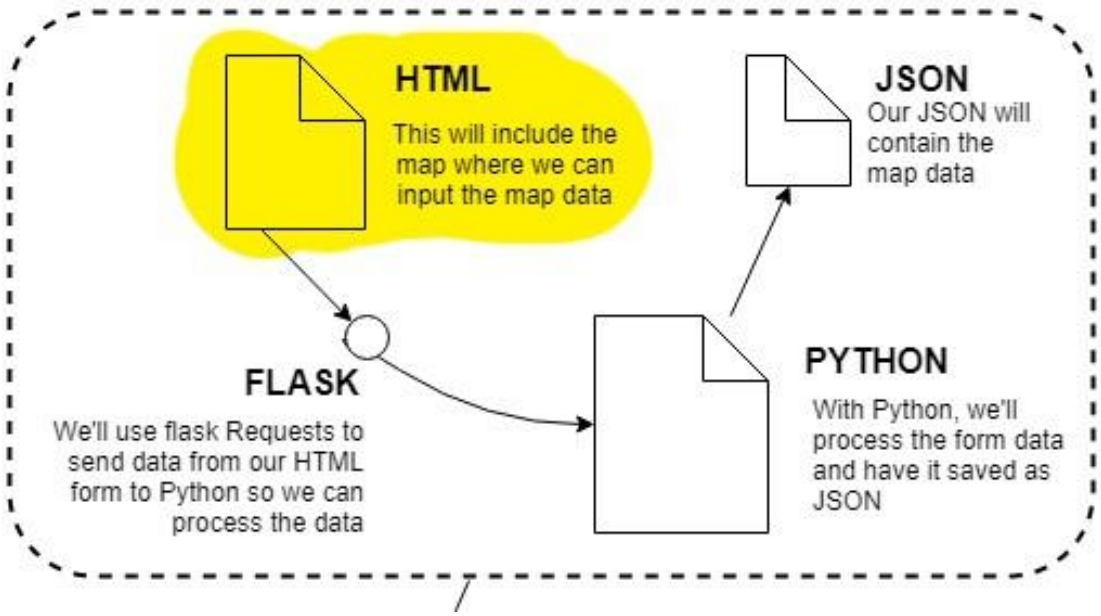
```
55 ▼  def save(self):
56      """Get Method to save current weather to csv file"""
57      #Generate a timestamp using the time module
58      timestamp = datetime.now()
59
60      # The data collected will should be stored like this temperature timestamp
61      data = [timestamp,self.getCurrentHumidity(),self.getCurrentTemp(),self.getCountry()]
62
63 ▼    with open('data/weatherdata.csv', 'a', newline='') as f_object:
64        writer_object = writer(f_object)
65        writer_object.writerow(data)
66        f_object.close()
67
68
69 ▼  def getWeatherData(self):
70      """Get Method to view csv file for testing purposes """
71 ▼    with open('data/weatherdata.csv', 'r') as my_file:
72        file_reader = reader(my_file)
73 ▼      for i in file_reader:
74          print (i)
75
```

Weather class - filterWeatherData method

This method is used to filter the data. To be able to filter rows in a csv file by times or dates, the data has to be converted into a pandas data frame (df).

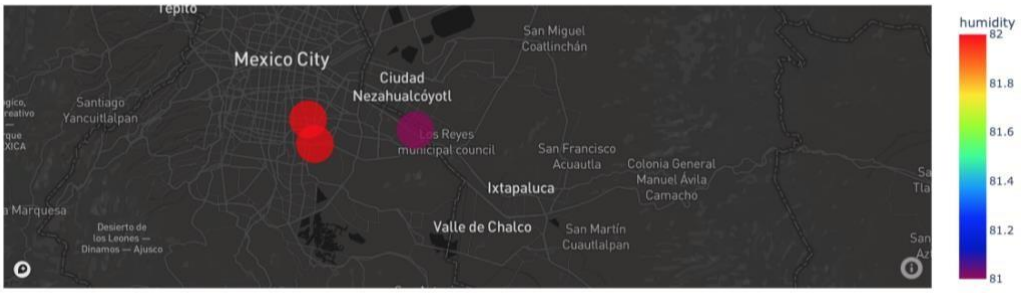
```
78 ▼  def filterWeatherData(self, start, end):
79      """Get Method to filter csv data by start and end time"""
80
81      df = pd.read_csv("data/weatherdata.csv")
82      df['Date'] = pd.to_datetime(df['timestamp']).dt.date
83      df['Time'] = pd.to_datetime(df['timestamp']).dt.time
84
85      # Filter data by date
86      #date = datetime.strptime('2022-07-24', '%Y-%m-%d').date()
87      #filtered_dates = self.df.loc[self.df["Date"] <= date]
88      #Filter by time
89      start_time = datetime.strptime(start, '%H:%M:%S').time()
90      end_time = datetime.strptime(end, '%H:%M:%S').time()
91
92      after_start_time = df["Time"] >= start_time
93      before_end_time = df["Time"] <= end_time
94      between_two_times = after_start_time & before_end_time
95      filtered_times = df.loc[between_two_times,["timestamp","humidity","temp"]]
96
97      return filtered_times
98
```

Displaying maps in a Flask web application



What are Geomaps?

Geomaps are fantastic visual representation tools for interpreting and presenting data which includes location.



Data at hand that has some kind of location information attached to it can come in many forms, subjects and domains. Some examples are:

And the list goes on as we count more and more niche subjects. This type of data sometimes comes fully ready with coordinates but sometimes only location data included will be a **city name, county name, country name**.

Global events	Nature related	Socio-economic	Tourism and culture:
Festivals War Conflict Pandemic Global warming	Earthquake Tsunami Bird, fish, mammals	Trade data Housing and Real Estate	Voyage Celebrations Holidays

Discovering the Map Service and Getting a Free Account

First of all we'll use Mapbox API key for map services. So it might be useful to take care of that as first step. There are many different map services out there and it might make sense to invest some time and discover different benefits and costs when a commercial application is at hand.

But, Mapbox has a generous free account and it will be perfect for this tutorial. https://account.mapbox.com/auth/signup/

You can sign up here and get a free account and API key.

We will read the API key from a file you can save as following: (named .mapbox_token with no file extension)

```
px.set_mapbox_access_token(open(".mapbox_token").read())
```

Thankfully, Mapbox has a generous free plan so you shouldn't have to worry about billing unless you have a huge traffic.

Creating the map chart w/Plotly (Step-by-step)

Let's split up the code first and investigate each part for a better understanding.

First you'll need the libraries:

```
Prep & Reading Data (libraries, data file, API token
etc.) import plotly import plotly.express as px import
pandas as pd
API Access Token will be needed as explained above.
```

Map.class

```python
import plotly.express as px
import pandas as pd
import json
from weather import Weather

class Map():
    """A class for generating different types of maps"""
    def __init__(self):
        """Constructor Method"""
        px.set_mapbox_access_token(
            'pk.eyJ1IjoiZ2ZlbGl4IiwiYSI6ImNrZTNsbnYzMTBraG0zMnFuZXNjOWZhdDgifQ.5sMKH7NQ6_oVyU4oJlcBUw')

    def scatterplot_map(self,latlon):
        """A method to create a scatterplot map"""
        #areas of interest using lat,lon
        humidity = []
        lat = []
        lon = []
        for i in latlon:
          clat = i[0]
          clon = i[1]
          #Current Weather
          current_weather = Weather()
          current_weather.getCurrentWeather(clat,clon)
          humidity_data = current_weather.getCurrentHumidity()
          humidity.append(humidity_data)
          lat.append(clat)
          lon.append(clon)

        dict_map = {'humidity': humidity, 'lat': lat, 'lon': lon}

        df = pd.DataFrame.from_dict(dict_map)  # transforms the dictionary to a pandas dataframe

        fig = px.scatter_mapbox(df, lat="lat", lon="lon", size="humidity",
                                color="humidity",
                                color_continuous_scale=px.colors.sequential.Rainbow, size_max=30, zoom=1)
        fig.update_layout(mapbox_style="dark")
        html_string = fig.to_html(full_html=False)
        return html_string

```

```
43 ▼    def choropleth(self):
44          """A method to create a choropleth map"""
45
46          #get a choropleth map using Country Polygons as GeoJSON
47          countries = json.load(open('data/world_countries.geojson', 'r'))
48          #purpose of this code is to add a column to the df that includes unique locations id
49          country_id_map = {}
50 ▼        for feature in  countries['features']:
51              feature['id'] = feature['properties']['cartodb_id']
52              country_id_map[feature['properties']['name']] = feature['id']
53
54          #link to csv file that contains your data to be plotted on the map
55          df = pd.read_csv("data/map_data.csv")
56
57          #adds new column to df - applies correct id based on name
58          df['id'] = df['country'].apply(lambda x:country_id_map[x])
59
60          fig = px.choropleth_mapbox(df, locations='id', geojson=countries, color='data',
61                                     color_continuous_scale="Viridis",
62                                     mapbox_style="carto-positron",
63                                     zoom=2,
64                                     opacity=0.5,
65                                     labels={'data': 'levels'}
66                                     )
67          fig.update_layout(margin={"r": 0, "t": 0, "l": 0, "b": 0})
68          html_string = fig.to_html(full_html=False)
69          return html_string
```

Heart of the operation (*Creating figure object*)

At this point, we are ready to create the figure object that will represent the map chart using express module of plotly library.

Here are the main parameters that's used in the creation of map chart:

- **df** is the first argument representing dataframe containing input data
- **lat**: latitude data for each data point
- **lon**: longitude data for each data point

Here are some optional parameters that are used that make the map much prettier and functional:

- **hover_name**: Defines the data to be shown when mouse pointer is hovered over each data point.
- **size**: Points to data that sizes are based on
- **color_continuous_scale**: Applies an appropriate color scale of choice
- **size_max**: limits the maximum size of each data
- **zoom**: defines map zoom level upon loading. 1 is for most zoomed out map

```
fig = px.scatter_mapbox(df, lat="lat", lon="lon", size="humidity",
                        color="humidity",
                        color_continuous_scale=px.colors.sequential.Rainbow,
size_max=30, zoom=1)
```

Map Styles & Layers (*mapbox_style and mapbox_layer*)

After that, there are different map styles you can apply to your map. Using **mapbox_style** is a great way to quickly give style to the map and it will dictate the overall map color scheme (this part is also known as lowest layer or base map). Here are some of the options you can try:

dark open-street-map white-bg carto-positron carto-darkmatter stamen-terrain stamen-toner stamen-watercolor	basic streets outdoors light dark satellite satellite-streets

```
fig.update_layout(mapbox_style="dark")
```

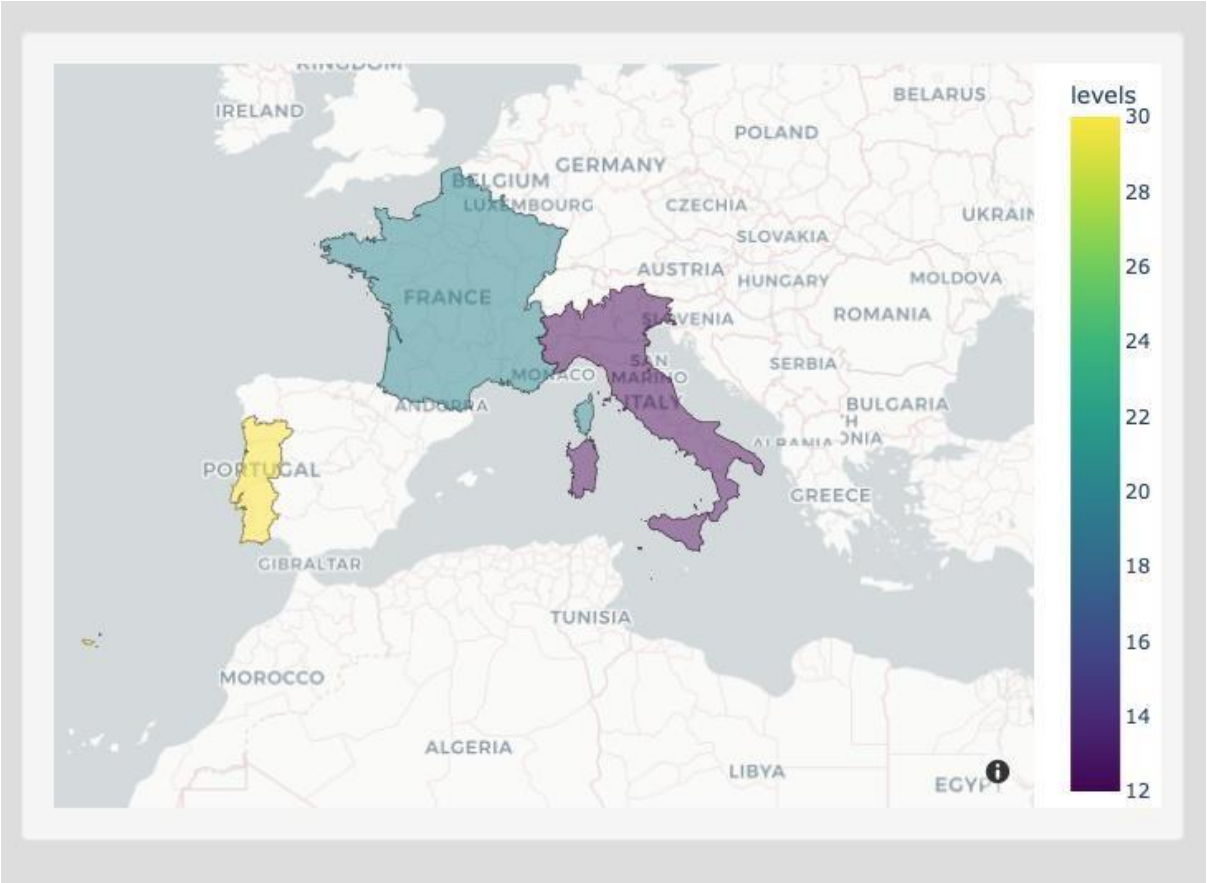
Flask app.py

```
@app.route('/dashboard')
def dashboard():

    #Interactive weather map
        my_map = Map()
        latlon = [(lat, lon)]

        html_string = my_map.scatterplot_map(latlon)

        return render_template('dashboard.html',div_placeholder=html_string)
```

dashboard.html

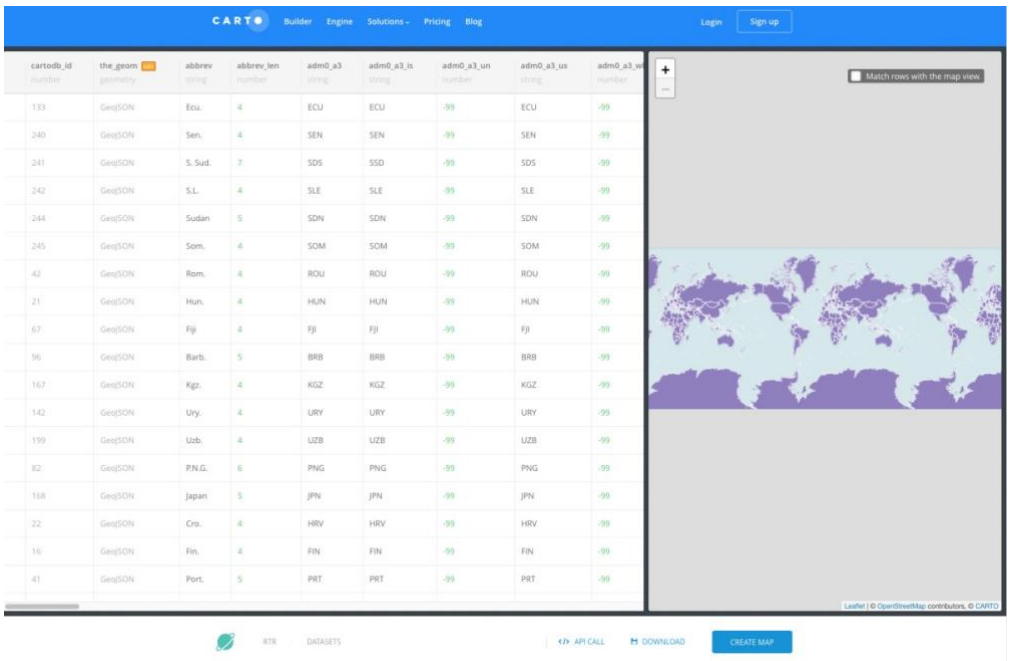
```
<script src="https://cdn.plot.ly/plotly-latest.min.js"></script>

{{ div_placeholder|safe }}
```

Creating a Choropleth map



You will need;

geojson file - GeoJSON is an open standard format designed for representing simple geographical features. A useful website for geojson data is CARTO - https://rtr.carto.com/datasets



Step 1: Get geojson data

In this tutorial I am using a choropleth map for all world countries.

I used this resource for the geojson data
https://rtr.carto.com/tables/world_countries_geojson/public/map

Download the geojon file. I saved it as data>world_countries.geojson

Step 2: Create a map_data.csv that contains your data of interest to plot on the map. This data can be accessed from an api but it must contain a column that links with data in the GeoJSON.

Example of a data set:

```
country,data
France,20
Italy,12
Portugal,30
```

You be using **import plotly.express as px** and using **choropleth_mapbox()** to create the map.

Flask app.py

```
@app.route('/dashboard')
def dashboard():


    #Interactive weather map
    my_map = Map()

    html_string = my_map.choropleth()


    return render_template('dashboard.html',div_placeholder=html_string)
```

Project Libraries:

requirements.txt

```
#Usage: pip install -r requirements.txt

#Flask framework

Flask==2.0.2
Jinja2==3.0.0
MarkupSafe==2.0.0rc2
Werkzeug==2.0.0

pandas~=1.4.3
requests~=2.28.1
reverse_geocoder>=1.5.1
plotly~=5.9.0 numpy>=1.11.0
scipy>=0.17.1
```

Further development:

Storing Data in a database using SQLite

```
import sqlite3
from flask import Flask, render_template

app = Flask(  name  )

def get_db_connection():
    conn = sqlite3.connect('database.db')
    conn.row_factory = sqlite3.Row
    return conn


@app.route('/')
def index():
    conn = get_db_connection()
    posts = conn.execute('SELECT * FROM posts').fetchall()
    conn.close()
    return render_template('index.html', posts=posts)
```

Tutorial: https://www.digitalocean.com/community/tutorials/how-to-use-ansqlite-database-in-a-flask-application