POLITECNICO
MILANO 1863

# Software Release Document

An interactive application to support disaster management activities during the prevention and preparedness phase

|  |  |
|---|---|
| **Deliverable:** | SRD |
| **Title:** | Software Release Document |
| **Authors:** | Junjie Mu, Chaotung Wu, Jiayi Su, Jianwei Deng |
| **Version:** | 1.0 |
| **Date:** | June 28, 2024 |
| **Download page:** | https://github.com/Junjie-Mu/SE4GEO |
| **Copyright:** | Copyright © 2024, M.W.S.D – All rights reserved |

## Revision history

| **Version** | **Date** | **Change** |
|---|---|---|
| 1.0 | June 28, 2024 | First submitted version |

# Content

## List of Table

# 1. Introduction

## 1.1 Purpose of the Document

The purpose of this document is to provide a comprehensive overview of the web application designed for visualizing and analyzing environmental data from the Italian Institute for Environmental Protection and Research (ISPRA). This document details the system's architecture, functionality, and user interface, ensuring that stakeholders and developers have a clear understanding of the system's design and capabilities.

## 1.2 Overview of the Software

The software is an interactive web application that leverages spatial data to present users with dynamic and customizable visualizations. It integrates various tools and libraries to enable users to explore environmental data through maps, charts, and other graphical representations. The application is built on a client-server architecture, utilizing PostgreSQL with PostGIS for data storage, Flask for the backend web server, and Jupyter Notebooks for the frontend dashboard. This architecture ensures efficient data processing and an engaging user experience.

# 2. System Requirements

## 2.1 Hardware requirements

Building a Web GIS (Geographic Information System) requires a combination of server hardware, networking components, and client-side hardware to ensure efficient processing, storage, and delivery of geospatial data.

Here are the hardware requirements:

### 2.1.1 Server Hardware

- Processor (CPU):

Multi-core processors (e.g., Intel Xeon or AMD EPYC) to handle concurrent requests and geospatial computations efficiently.

- Memory (RAM):

At least 8GB of RAM for small to medium-sized applications.

16GB or more for large datasets and high user concurrency.

- Storage:

SSDs (Solid State Drives) for fast data access and retrieval.

High-capacity storage (256GB or more) depending on the size of geospatial data.

- Graphics Processing Unit (GPU):

Optional but beneficial for rendering complex maps and performing advanced spatial analysis.

- Network Interface Card (NIC):

High-speed network interfaces (1 Gbps or higher) to handle data transfer and communication between servers and clients.

## 2.2 Software dependencies

Overall, a Web GIS can efficiently manage and serve geospatial data, providing robust and interactive mapping capabilities to users on terms of integrating these software dependencies below.

### 2.2.1 Operating System

Server Operating Systems:

Linux or Windows.

### 2.2.2 GIS Server Software

Flask server:

An effective middleware that connects users to the data and services provided by the backend.

### 2.2.3 Databases

PostgreSQL with PostGIS:

An open-source Database Management System, which uses a client-server model where the server program manages the database files and accepts connections to the database from client applications.

SE4G is the specific database supports all of the information of the website.

### 2.2.4 GIS Libraries and APIs

Basemap:

To build an interactive map which enhances features and ongoing support such as Map Plotting, Geospatial Data Visualization and Handling Geographic Features.

Data and color map:

Using color maps to represent different data values visually, enhancing data interpretation.

Heatmap:

Displaying the concentration of data points in a given area, highlighting patterns or hotspots.

Idro_GEO API:

The PIR (Hazards and Risk Indicators) data that are retrieved from the well-defined The Italian Institute for Environmental Protection and Research (ISPRA) IdroGEO API.

## 2.3 Network requirements

A Web GIS system can provide reliable, high-performance access to geospatial data, ensuring a smooth and secure user experience by addressing these network requirements.

### 2.3.1 Internet Connectivity

High-Speed Internet Connection:

Ensure a reliable high-speed internet connection (minimum 1 Gbps) for both server and client connections to handle data transfers efficiently.

# 3. Installation and Running

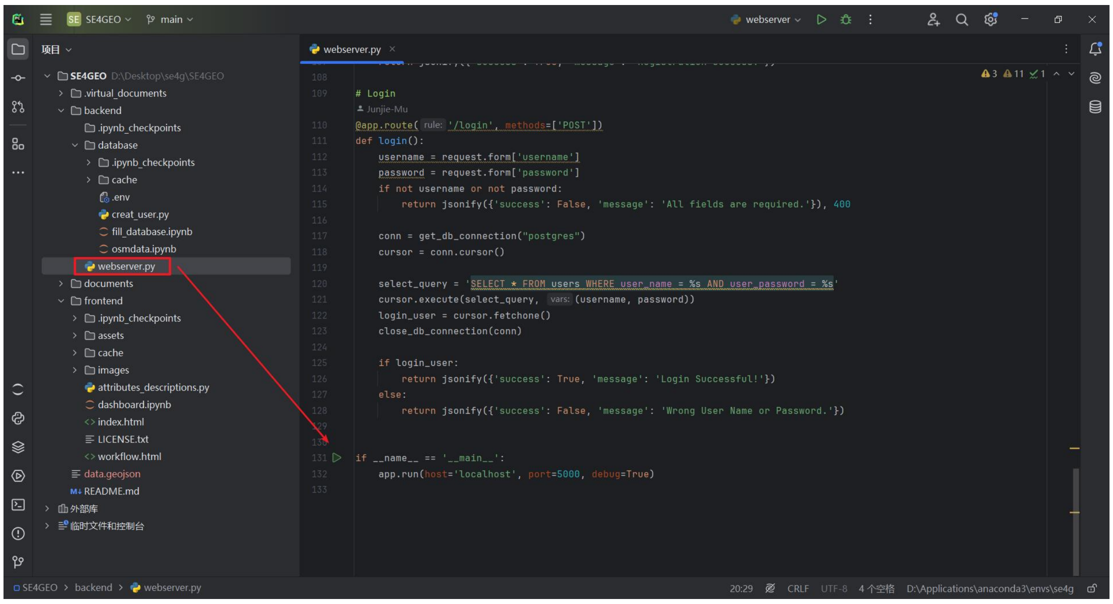
## 3.1 Step-by-step installation instructions

### Download:

Get the project from the GitHub repository(link).

### Open & Run:

Open the folder and find the file named " webserver.py ", run the programme.



```python

# Login
@app.route( rule: '/login', methods=['POST'])
def login():
    username = request.form['username']
    password = request.form['password']
    if not username or not password:
        return jsonify({'success': False, 'message': 'All fields are required.'}), 400

    conn = get_db_connection("postgres")
    cursor = conn.cursor()

    select_query = 'SELECT * FROM users WHERE user_name = %s AND user_password = %s'
    cursor.execute(select_query, vars: (username, password))
    login_user = cursor.fetchone()
    close_db_connection(conn)

    if login_user:
        return jsonify({'success': True, 'message': 'Login Successful!'})
    else:
        return jsonify({'success': False, 'message': 'Wrong User Name or Password.'})


if __name__ == '__main__':
    app.run(host='localhost', port=5000, debug=True)
```

Figure 1: webserver.py

### Open the Browser:

In your browser's address bar, type " http://localhost:5000 " and press enter to access the main page of the project.

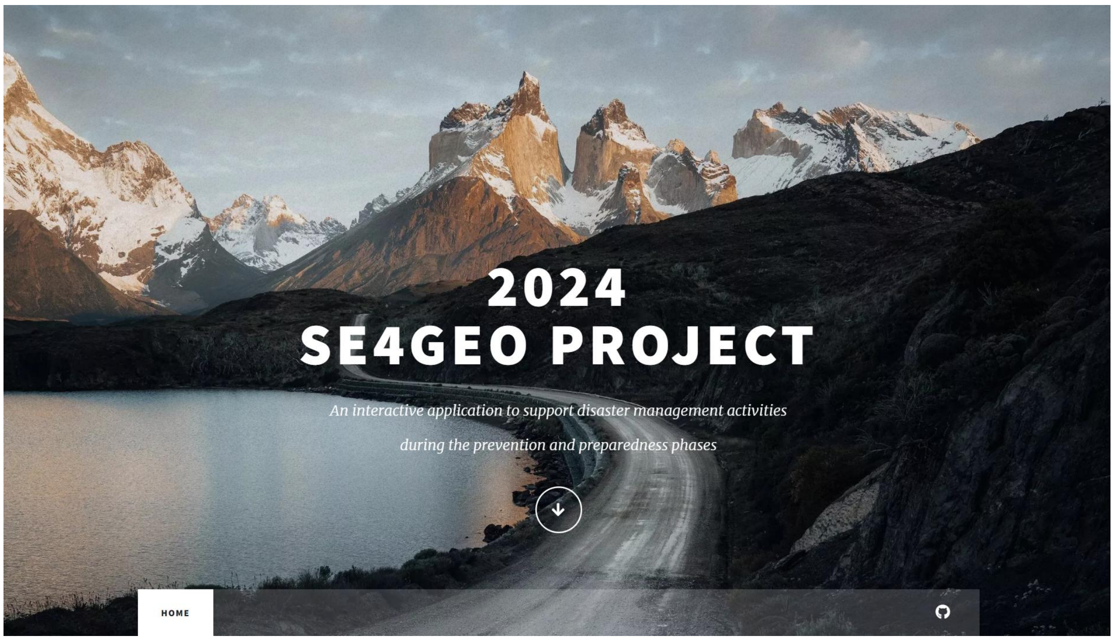
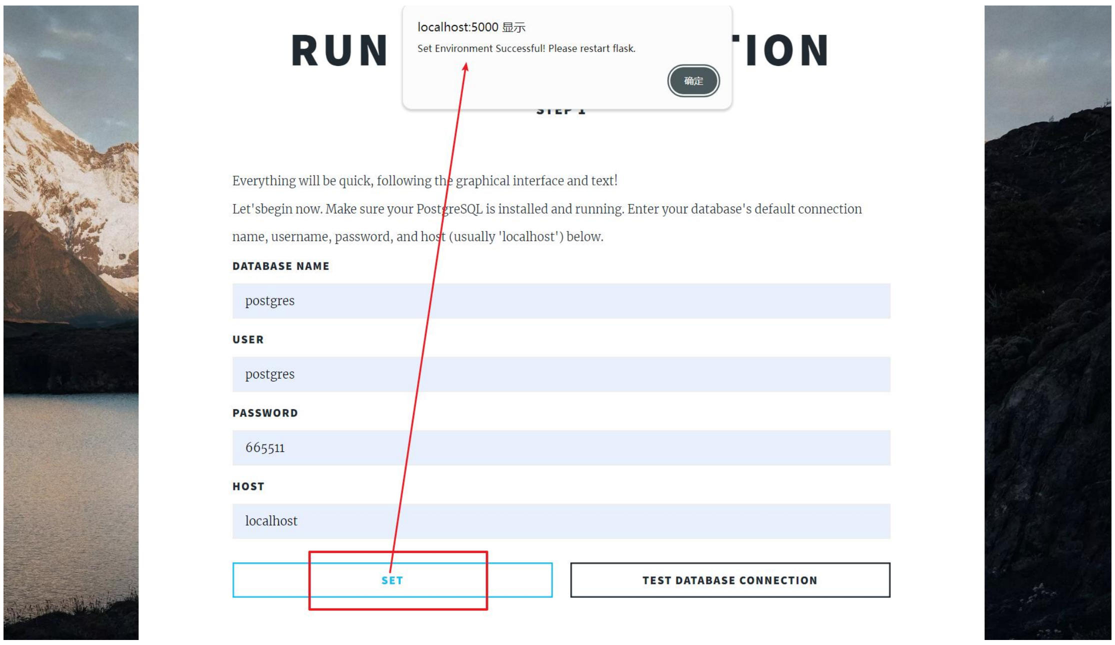

2024
SE4GEO PROJECT
An interactive application to support disaster management activities
during the prevention and preparedness phases
HOME

Figure 2: main page

## Step 1 : Set Database Parameters

On this page, follow the prompts to enter the database information and click the "SET" button.



localhost:5000 显示
Set Environment Successful! Please restart flask.

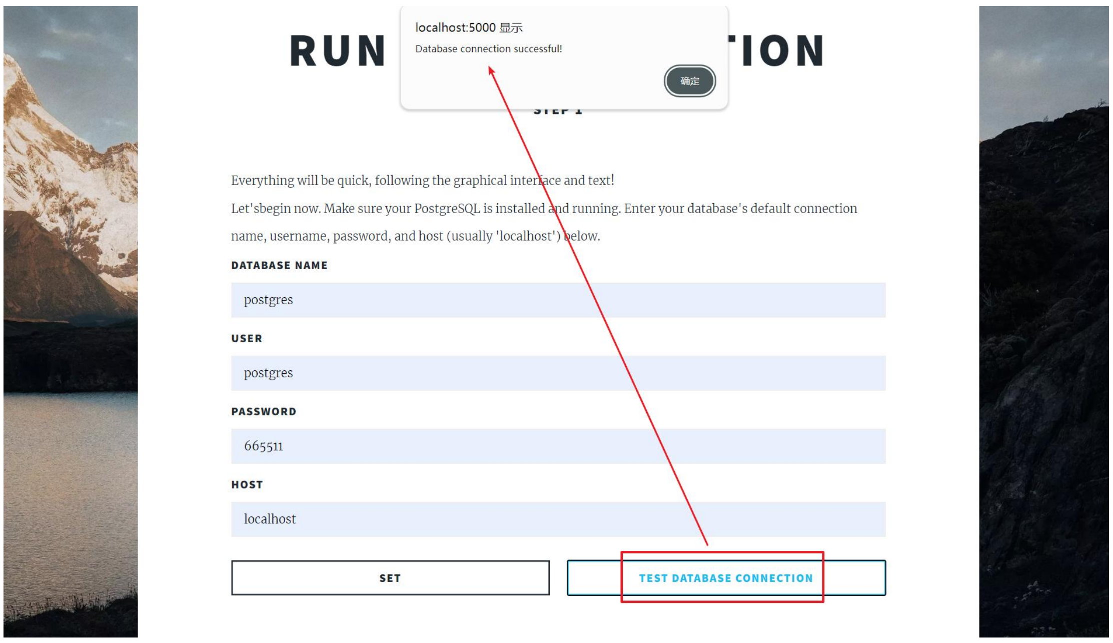
Everything will be quick, following the graphical interface and text!
Let'sbegin now. Make sure your PostgreSQL is installed and running. Enter your database's default connection name, username, password, and host (usually 'localhost') below.

DATABASE NAME
postgres
USER
postgres
PASSWORD
665511
HOST
localhost

Figure 3:Set database

Restart the Flask server and click the "Test Dtabase Connection" button.

If the connection is successful, proceed to the next step.


localhost:5000 显示

Database connection successful!

Everything will be quick, following the graphical interface and text!

Let'sbegin now. Make sure your PostgreSQL is installed and running. Enter your database's default connection name, username, password, and host (usually 'localhost') below.

DATABASE NAME: postgres

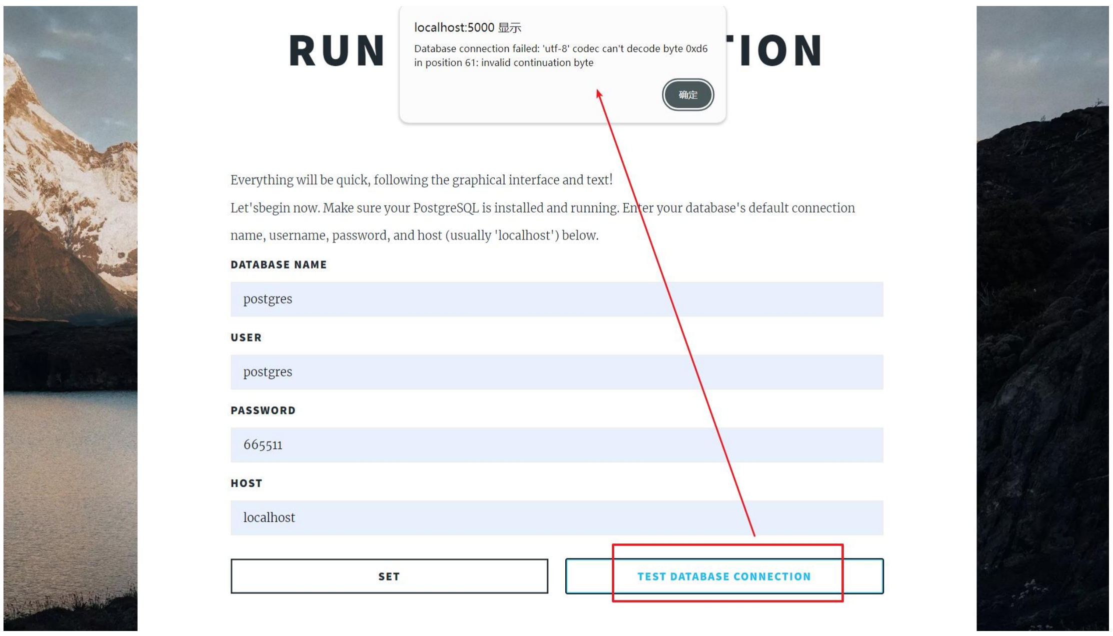
USER: postgres

PASSWORD: 665511

HOST: localhost

Figure 4:connection is successful

If the connection fails, check the database parameters and try again.


localhost:5000 显示

Database connection failed: 'utf-8' codec can't decode byte 0xd6 in position 61: invalid continuation byte

Everything will be quick, following the graphical interface and text!

Let'sbegin now. Make sure your PostgreSQL is installed and running. Enter your database's default connection name, username, password, and host (usually 'localhost') below.

DATABASE NAME: postgres

USER: postgres

PASSWORD: 665511

HOST: localhost

Figure 5:connection fails

## Step 2. Create Database Table

Click the "Create" button on the main page. The system will automatically create the necessary database tables.

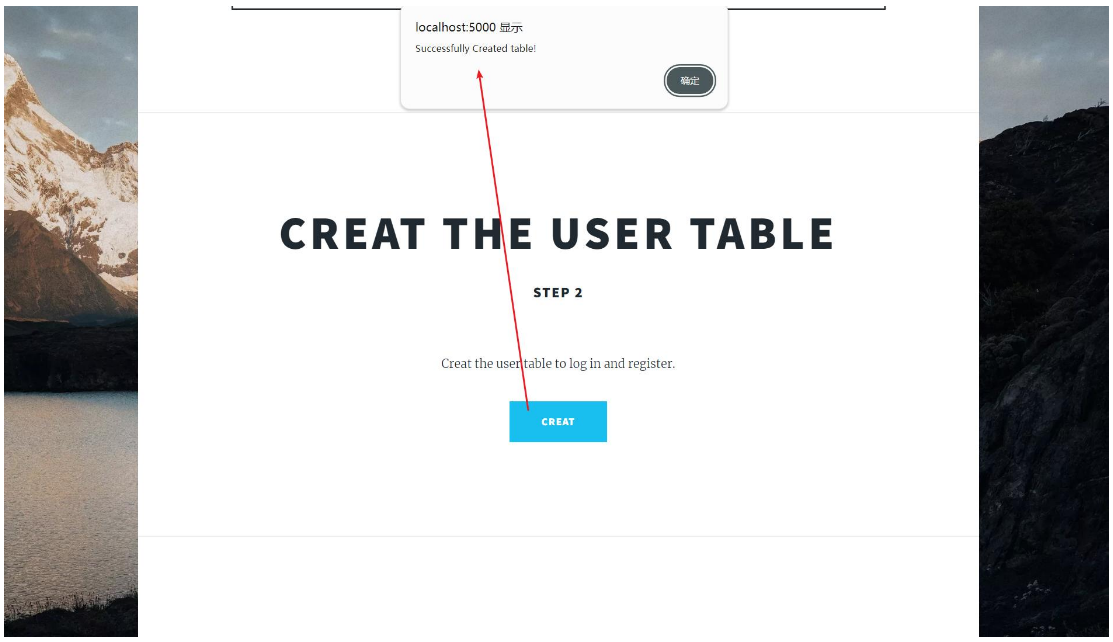

localhost:5000 显示
Successfully Created table!
确定
CREAT THE USER TABLE
STEP 2
Creat the user table to log in and register.
CREAT

Figure 6:create the necessary database tables

## Step 3. User Login

Click the "Login Now" button to open the login interface.

If this is your first time using the software, click the "Register Now" button at the bottom of the login interface.

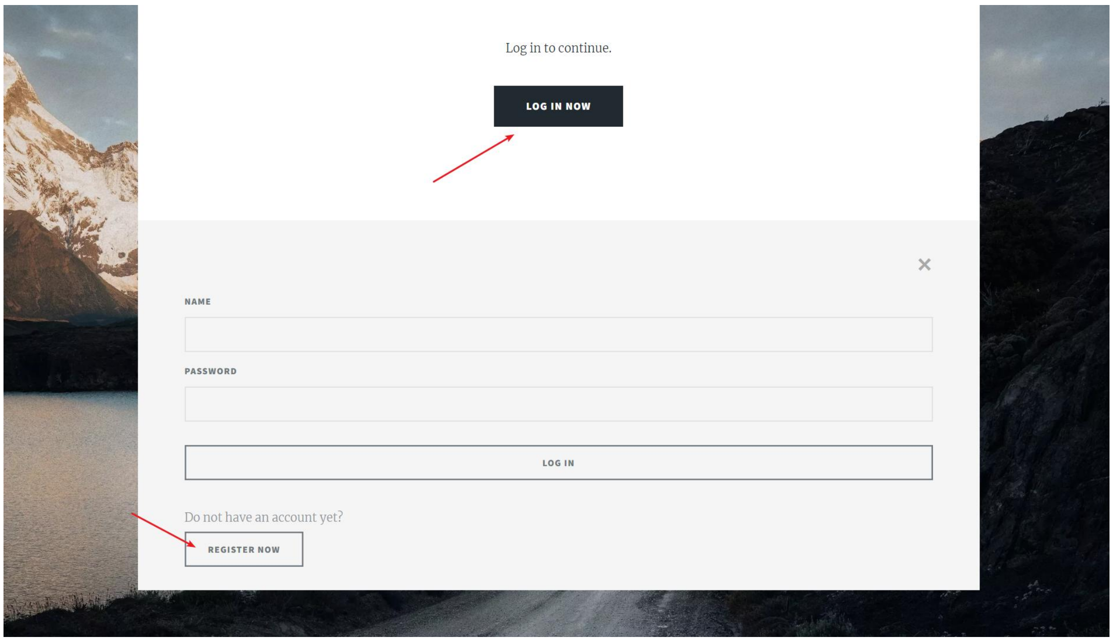


Figure 7:Register Now

Follow the prompts to complete the registration.

Log in using the username and password you just set.

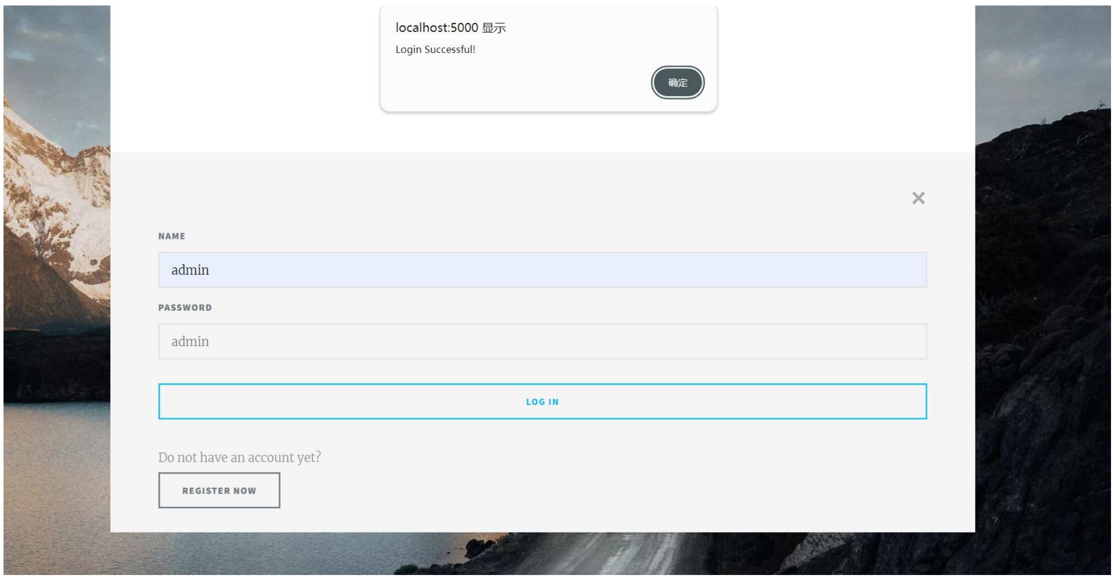

localhost:5000 显示
Login Successful!

Figure 8:Log in

If you enter the wrong credentials, the system will display an error message.

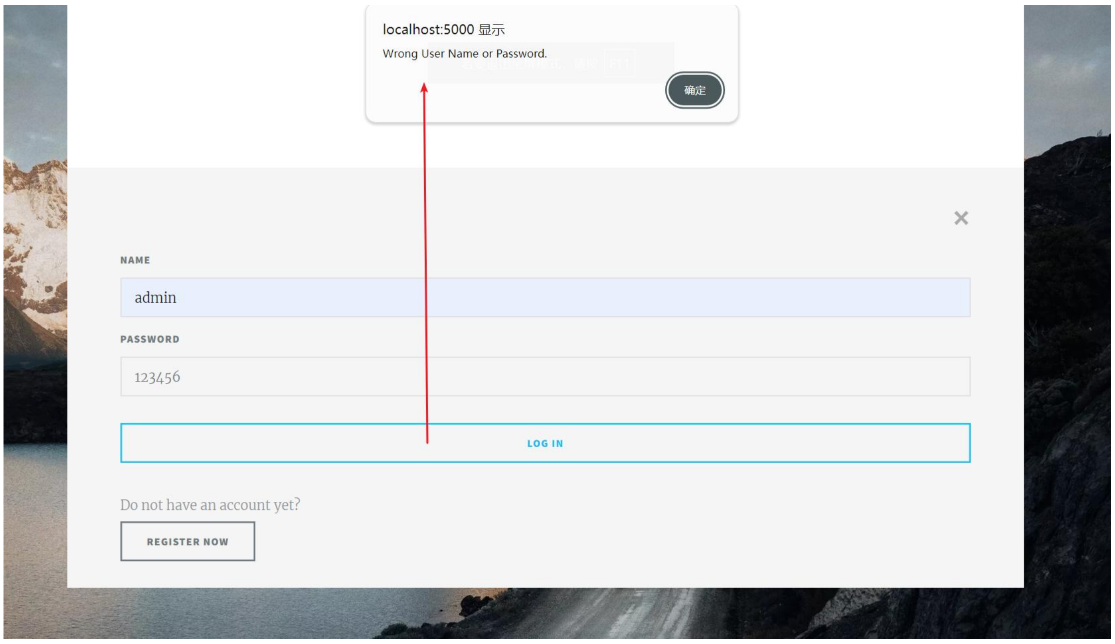

localhost:5000 显示
Wrong User Name or Password.

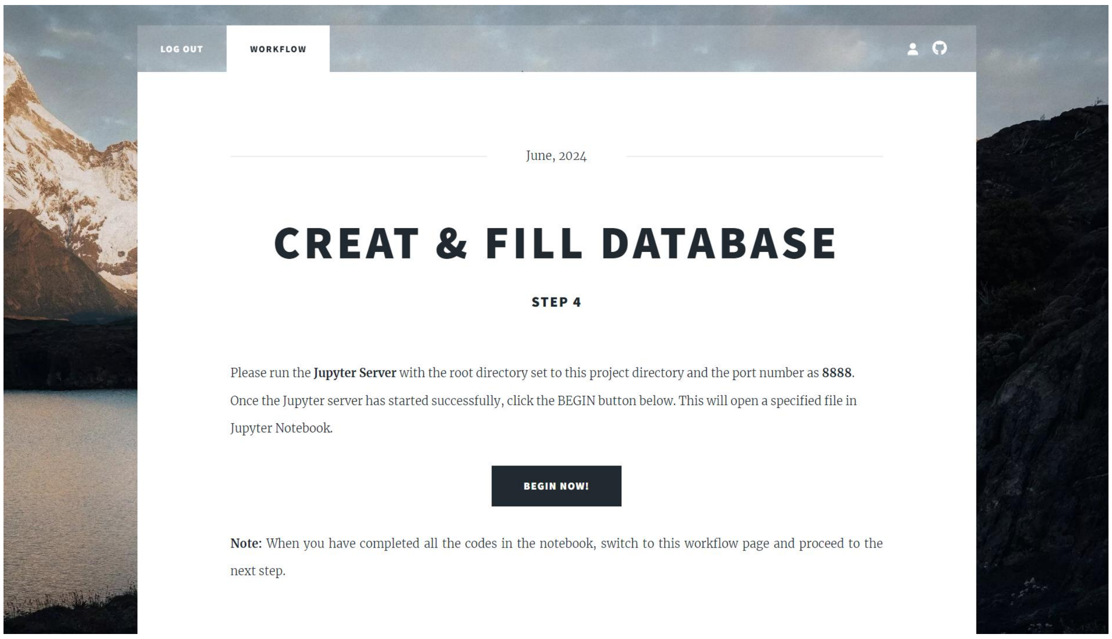
Figure 9:wrong credentials

Upon successful login, the system will automatically redirect you to the workflow page, indicating that you are logged in.

Click the "Logout" button to log out and return to the main page.

## Step 4. Initialize Database and Configuration

After logging in, on the workflow page, click the "Begin Now" button to open Jupyter Notebook.



June, 2024

CREAT & FILL DATABASE

STEP 4

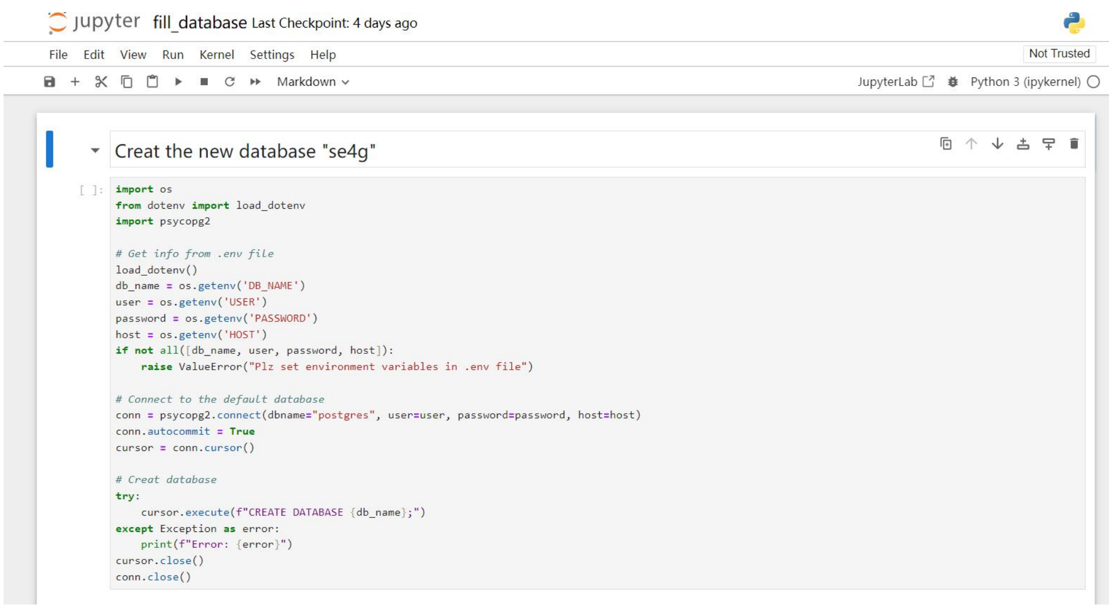
Please run the **Jupyter Server** with the root directory set to this project directory and the port number as **8888**. Once the Jupyter server has started successfully, click the BEGIN button below. This will open a specified file in Jupyter Notebook.

**Note:** When you have completed all the codes in the notebook, switch to this workflow page and proceed to the next step.

Figure 10: Begin Now

In Jupyter Notebook, execute the following steps:

Create a new database "SE4G" .

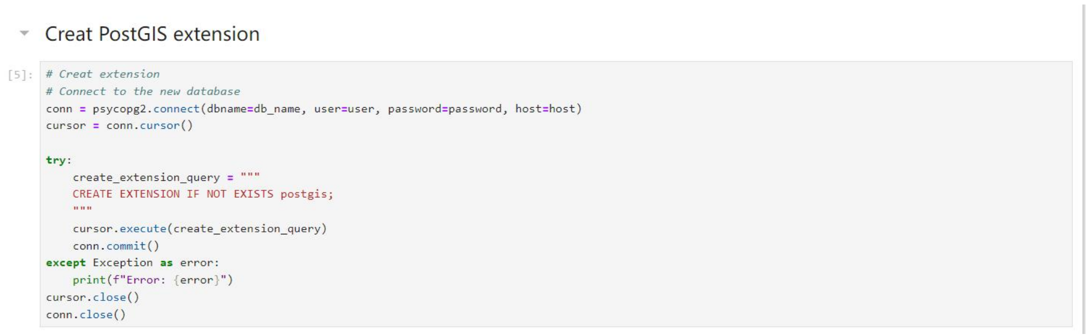

jupyter fill_database Last Checkpoint: 4 days ago
File Edit View Run Kernel Settings Help
Not Trusted
Markdown
JupyterLab
Python 3 (ipykernel)

Creat the new database "se4g"

```
import os
from dotenv import load_dotenv
import psycopg2

# Get info from .env file
load_dotenv()
db_name = os.getenv('DB_NAME')
user = os.getenv('USER')
password = os.getenv('PASSWORD')
host = os.getenv('HOST')
if not all([db_name, user, password, host]):
    raise ValueError("Plz set environment variables in .env file")

# Connect to the default database
conn = psycopg2.connect(dbname="postgres", user=user, password=password, host=host)
conn.autocommit = True
cursor = conn.cursor()

# Creat database
try:
    cursor.execute(f"CREATE DATABASE {db_name};")
except Exception as error:
    print(f"Error: {error}")
cursor.close()
conn.close()
```

Figure 11: Create a new database

Create the PostGIS extension.

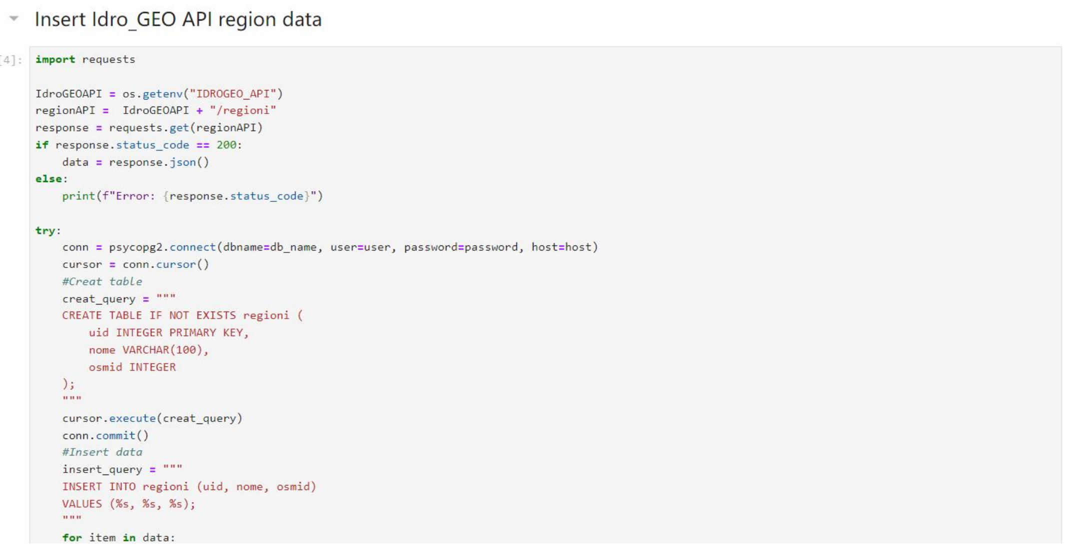


Creat PostGIS extension

```
# Creat extension
# Connect to the new database
conn = psycopg2.connect(dbname=db_name, user=user, password=password, host=host)
cursor = conn.cursor()

try:
    create_extension_query = """
    CREATE EXTENSION IF NOT EXISTS postgis;
    """
    cursor.execute(create_extension_query)
    conn.commit()
except Exception as error:
    print(f"Error: {error}")
cursor.close()
conn.close()
```

Figure 12: PostGIS extension

Insert Idro_GEO API region data.

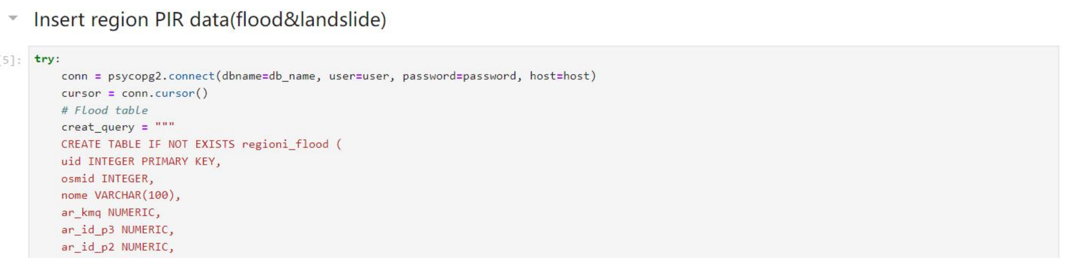


Insert Idro_GEO API region data

```
[4]: import requests

     IdroGEOAPI = os.getenv("IDROGEO_API")
     regionAPI =  IdroGEOAPI + "/regioni"
     response = requests.get(regionAPI)
     if response.status_code == 200:
         data = response.json()
     else:
         print(f"Error: {response.status_code}")

     try:
         conn = psycopg2.connect(dbname=db_name, user=user, password=password, host=host)
         cursor = conn.cursor()
         #Creat table
         creat_query = """
         CREATE TABLE IF NOT EXISTS regioni (
             uid INTEGER PRIMARY KEY,
             nome VARCHAR(100),
             osmid INTEGER
         );
         """
         cursor.execute(creat_query)
         conn.commit()
         #Insert data
         insert_query = """
         INSERT INTO regioni (uid, nome, osmid)
         VALUES (%s, %s, %s);
         """
         for item in data:
```

Figure 13: region data

## Insert region PIR data( flood&landslide).

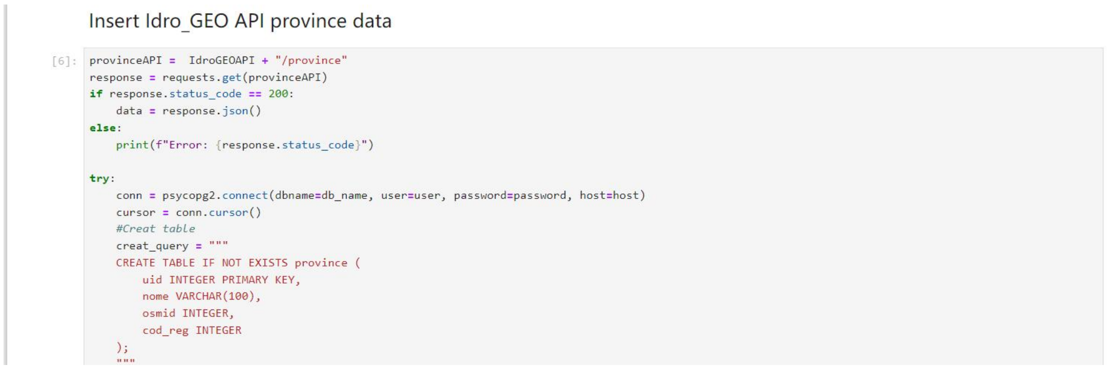


Insert region PIR data(flood&landslide)

```
[5]: try:
         conn = psycopg2.connect(dbname=db_name, user=user, password=password, host=host)
         cursor = conn.cursor()
         # Flood table
         creat_query = """
         CREATE TABLE IF NOT EXISTS regioni_flood (
         uid INTEGER PRIMARY KEY,
         osmid INTEGER,
         nome VARCHAR(100),
         ar_kmq NUMERIC,
         ar_id_p3 NUMERIC,
         ar_id_p2 NUMERIC,
```

Figure 14: PIR data

## Insert Idro_GEO API province data.



Insert Idro_GEO API province data

```
[6]: provinceAPI =  IdroGEOAPI + "/province"
     response = requests.get(provinceAPI)
     if response.status_code == 200:
         data = response.json()
     else:
         print(f"Error: {response.status_code}")

     try:
         conn = psycopg2.connect(dbname=db_name, user=user, password=password, host=host)
         cursor = conn.cursor()
         #Creat table
         creat_query = """
         CREATE TABLE IF NOT EXISTS province (
             uid INTEGER PRIMARY KEY,
             nome VARCHAR(100),
             osmid INTEGER,
             cod_reg INTEGER
         );
         """
```

Figure 15: province data

## Step 5. Import OSM Map Data

Return to the workflow page and click the "Begin Now" button again to execute the following tasks:

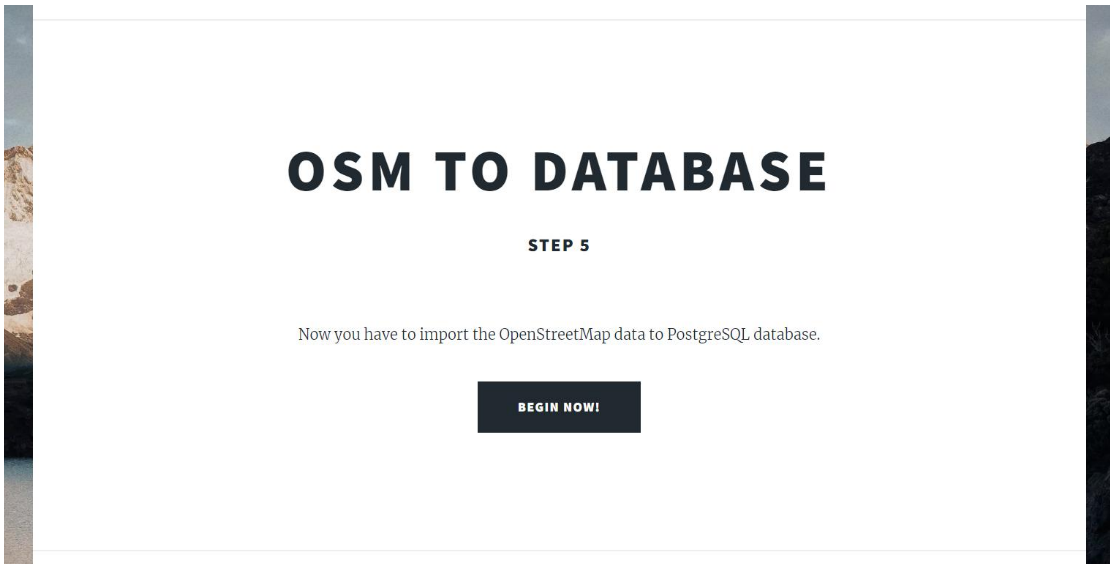
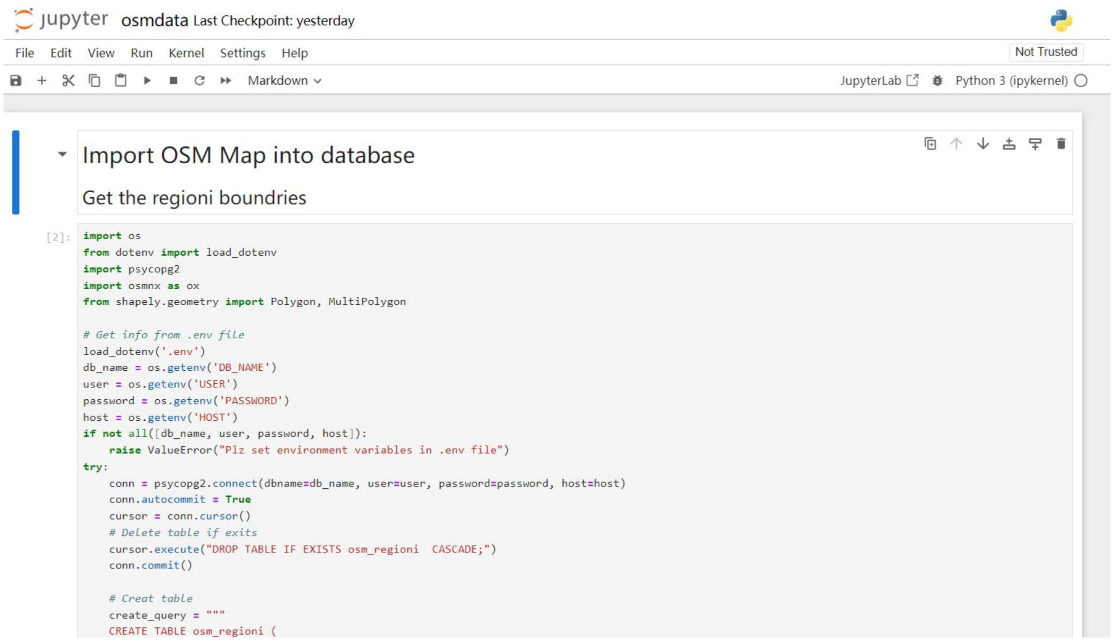

OSM TO DATABASE
STEP 5
Now you have to import the OpenStreetMap data to PostgreSQL database.
BEGIN NOW!

Figure 16: Import OSM Map Data

Import OSM map data into the database.

get the region boundaries.


jupyter osmdata Last Checkpoint: yesterday
File Edit View Run Kernel Settings Help
Not Trusted
Markdown
JupyterLab
Python 3 (ipykernel)

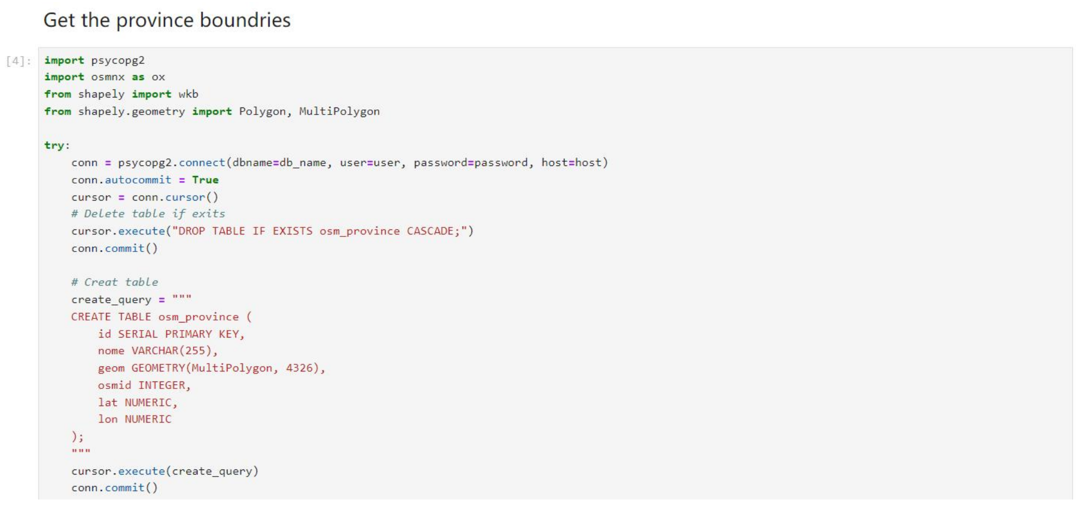
Import OSM Map into database

Get the regioni boundries

```
[2]: import os
from dotenv import load_dotenv
import psycopg2
import osmnx as ox
from shapely.geometry import Polygon, MultiPolygon

# Get info from .env file
load_dotenv('.env')
db_name = os.getenv('DB_NAME')
user = os.getenv('USER')
password = os.getenv('PASSWORD')
host = os.getenv('HOST')
if not all([db_name, user, password, host]):
    raise ValueError("Plz set environment variables in .env file")
try:
    conn = psycopg2.connect(dbname=db_name, user=user, password=password, host=host)
    conn.autocommit = True
    cursor = conn.cursor()
    # Delete table if exits
    cursor.execute("DROP TABLE IF EXISTS osm_regioni  CASCADE;")
    conn.commit()

    # Creat table
    create_query = """
    CREATE TABLE osm_regioni (
```

Figure 17: region boundaries

get the province boundaries.


Get the province boundries

```
[4]: import psycopg2
     import osmnx as ox
     from shapely import wkb
     from shapely.geometry import Polygon, MultiPolygon

     try:
         conn = psycopg2.connect(dbname=db_name, user=user, password=password, host=host)
         conn.autocommit = True
         cursor = conn.cursor()
         # Delete table if exits
         cursor.execute("DROP TABLE IF EXISTS osm_province CASCADE;")
         conn.commit()

         # Creat table
         create_query = """
         CREATE TABLE osm_province (
             id SERIAL PRIMARY KEY,
             nome VARCHAR(255),
             geom GEOMETRY(MultiPolygon, 4326),
             osmid INTEGER,
             lat NUMERIC,
             lon NUMERIC
         );
         """
         cursor.execute(create_query)
         conn.commit()
```

Figure 18: province boundaries

**Step 6**. After completing these tasks, click the "Go to Dashboard" button to end the configuration process.

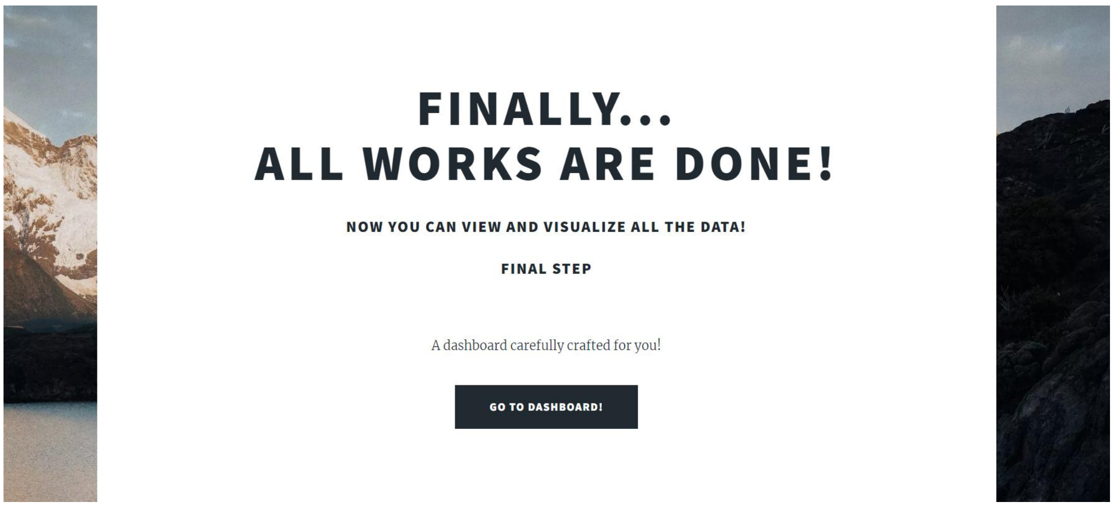

FINALLY...
ALL WORKS ARE DONE!
NOW YOU CAN VIEW AND VISUALIZE ALL THE DATA!
FINAL STEP
A dashboard carefully crafted for you!
GO TO DASHBOARD!

Figure 19: Go to Dashboard

After configuration, you can start using the software's various features for data analysis and management.

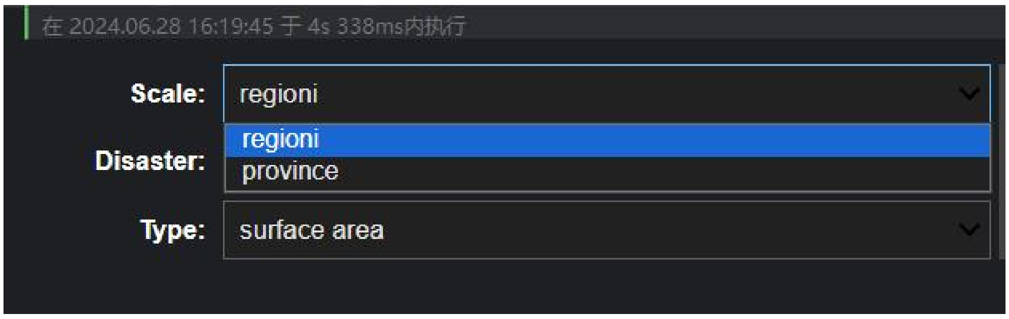
## 3.2 Basic usage instructions

This section provides a detailed guide on how to use the core functionalities of the software.

**Functionality 1:** Specified Condition Data Query

This functionality allows users to retrieve and visualize data based on specific conditions.

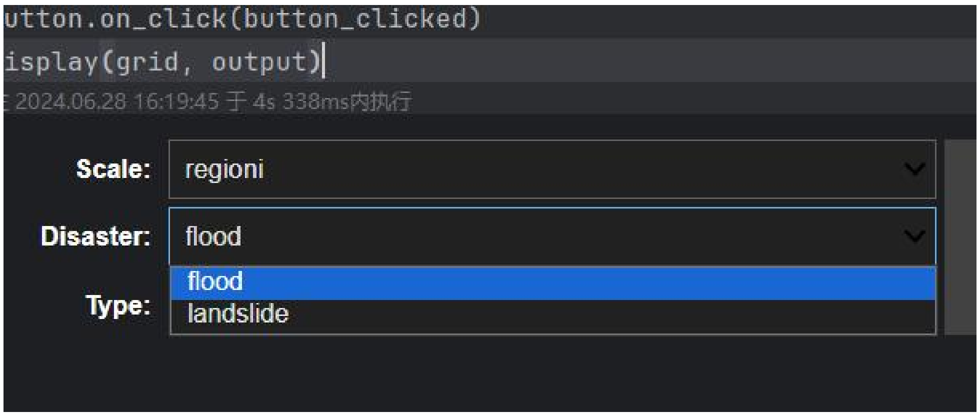
Select Query Scale : In the "scale" dropdown menu, select the desired query scale (regioni OR province).



```
在 2024.06.28 16:19:45 于 4s 338ms内执行
```

Figure 20: scale

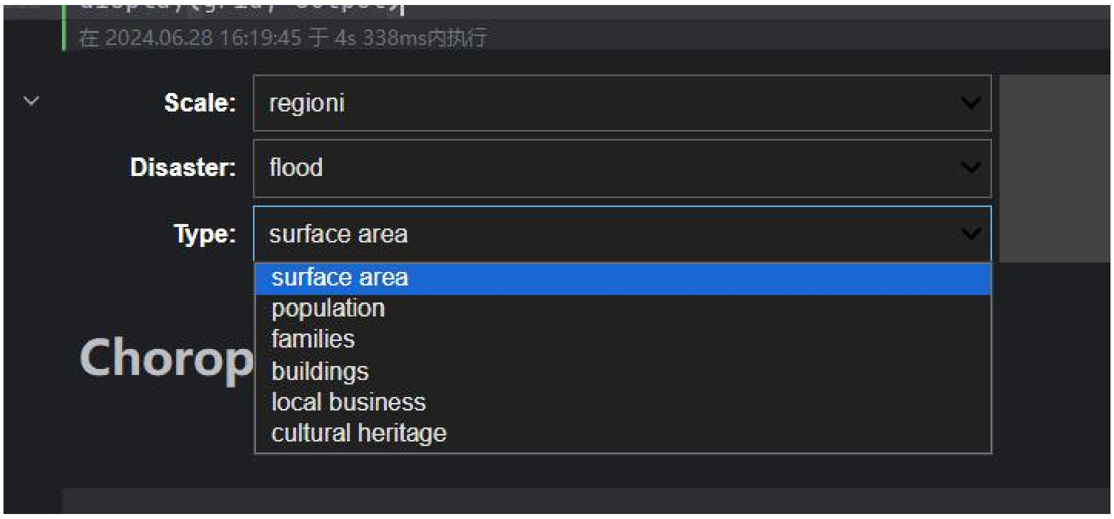
Select Disaster Type : In the "disaster" dropdown menu, choose the type of disaster (flood OR landslide ).



```
utton.on_click(button_clicked)
isplay(grid, output)
: 2024.06.28 16:19:45 于 4s 338ms内执行
```

Figure 21: disaster

Select Affected Object Type: In the "type" dropdown menu, select the type of affected objects (surface area, population, families, buildings,local business, cultural heritage ).



在 2024.06.28 16:19:45 于 4s 338ms内执行

Scale: regioni

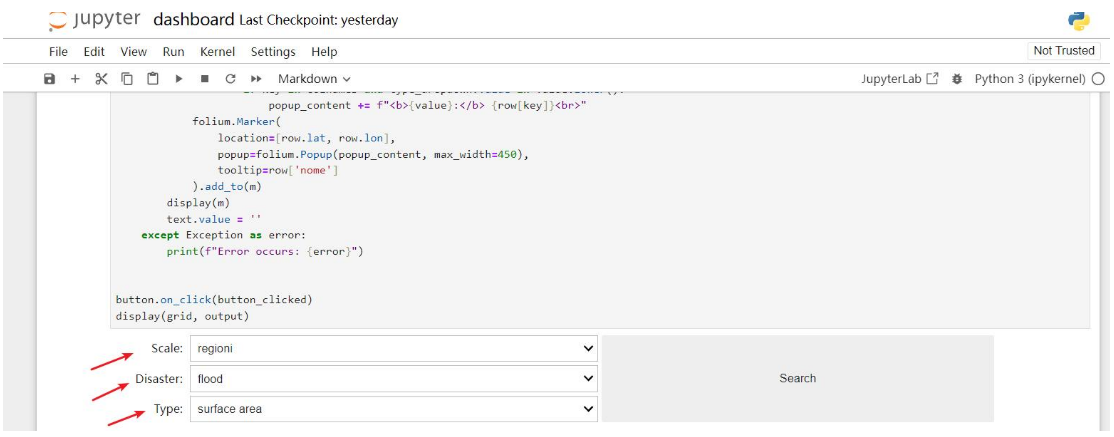
Disaster: flood

Type: surface area

Figure 22: type

Search for Data: Click the "search" button.

The filtered map data will appear below the button, providing a visual representation of the data.



jupyter dashboard Last Checkpoint: yesterday
File Edit View Run Kernel Settings Help
Not Trusted
Markdown
JupyterLab
Python 3 (ipykernel)
Search

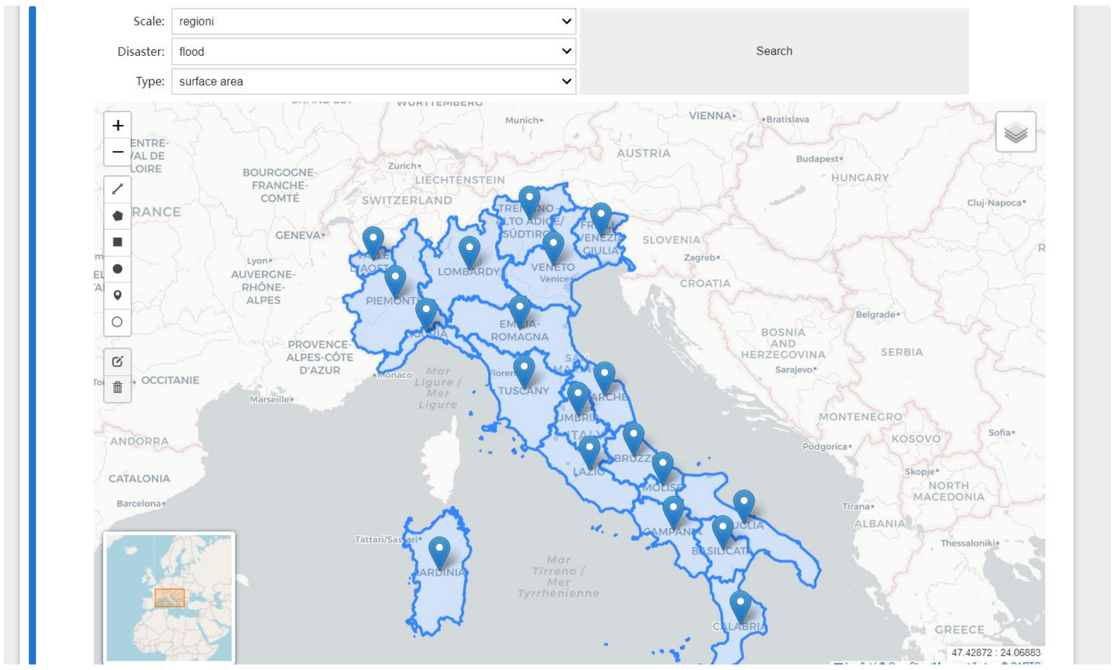
```
                popup_content += f"<b>{value}:</b> {row[key]}<br>"
            folium.Marker(
                location=[row.lat, row.lon],
                popup=folium.Popup(popup_content, max_width=450),
                tooltip=row['nome']
            ).add_to(m)
        display(m)
        text.value = ''
    except Exception as error:
        print(f"Error occurs: {error}")


button.on_click(button_clicked)
display(grid, output)
```

Scale: regioni

Disaster: flood

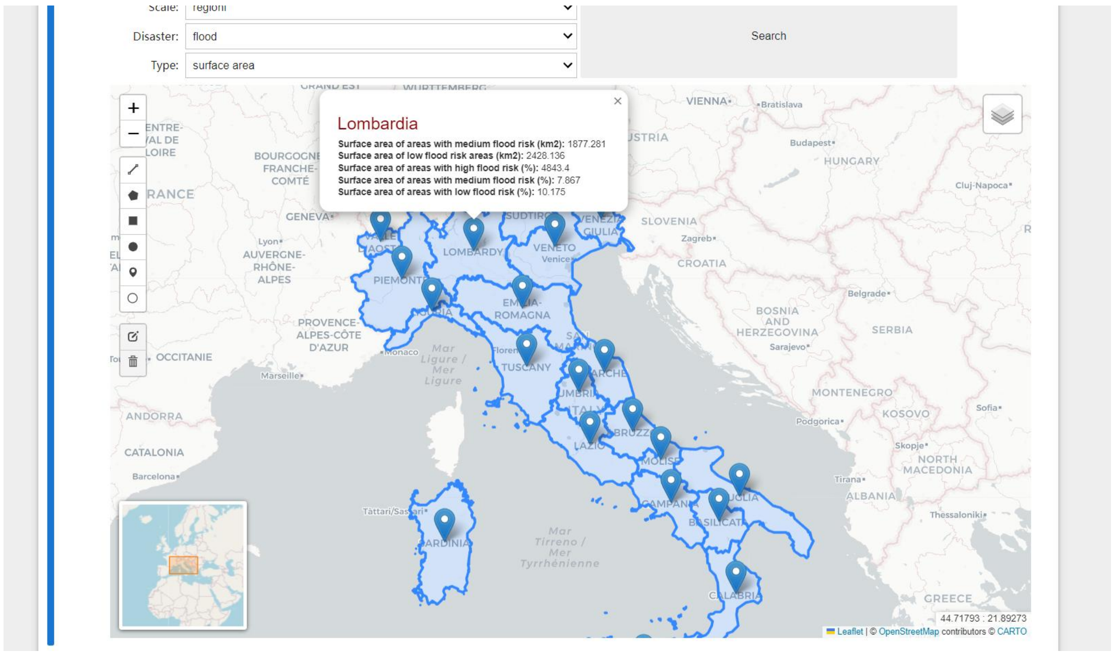
Type: surface area

Figure 23: Click the "search" button


Scale: regioni

Disaster: flood

Type: surface area


Scale: regioni

Disaster: flood

Type: surface area

Lombardia

**Surface area of areas with medium flood risk (km2):** 1877.281
**Surface area of low flood risk areas (km2):** 2428.136
**Surface area of areas with high flood risk (%):** 4843.4
**Surface area of areas with medium flood risk (%):** 7.867
**Surface area of areas with low flood risk (%):** 10.175

Figure 24: Visualization

Base Map Change: The map layer offers a "base map change" functionality. You can switch between the following map layers:

"cartodbpositron"

"openstreetmap"

"cartodbdark_matter"

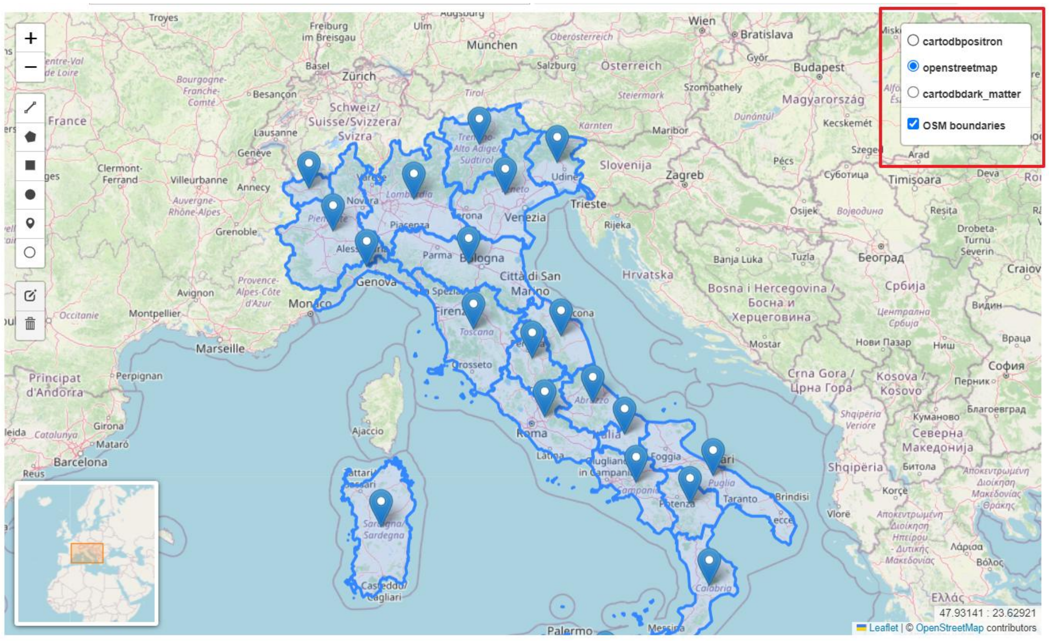


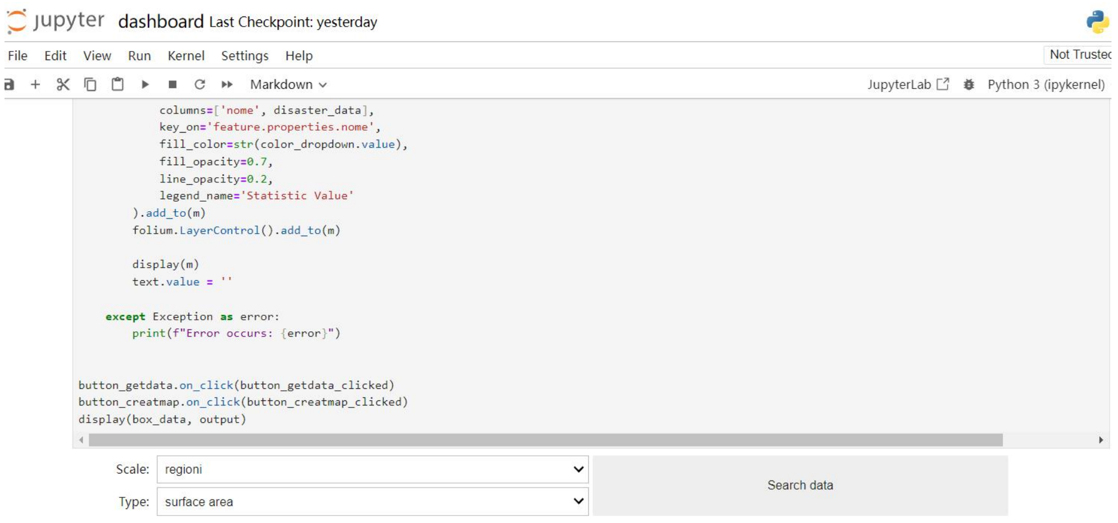
Figure 25: base map change

**Functionality 2:** Hierarchical Statistical Map

This functionality allows users to create a statistical map with hierarchical data representation.

Select Data Parameters: In the "scale" and "type" dropdown menus, select the desired parameters for your data query.

Search for Data: Click the "search data" button. A new interface will pop up.


jupyter dashboard Last Checkpoint: yesterday
File Edit View Run Kernel Settings Help
Not Trusted
Markdown
JupyterLab
Python 3 (ipykernel)

```
            columns=['nome', disaster_data],
            key_on='feature.properties.nome',
            fill_color=str(color_dropdown.value),
            fill_opacity=0.7,
            line_opacity=0.2,
            legend_name='Statistic Value'
        ).add_to(m)
        folium.LayerControl().add_to(m)

        display(m)
        text.value = ''

    except Exception as error:
        print(f"Error occurs: {error}")


button_getdata.on_click(button_getdata_clicked)
button_creatmap.on_click(button_creatmap_clicked)
display(box_data, output)
```

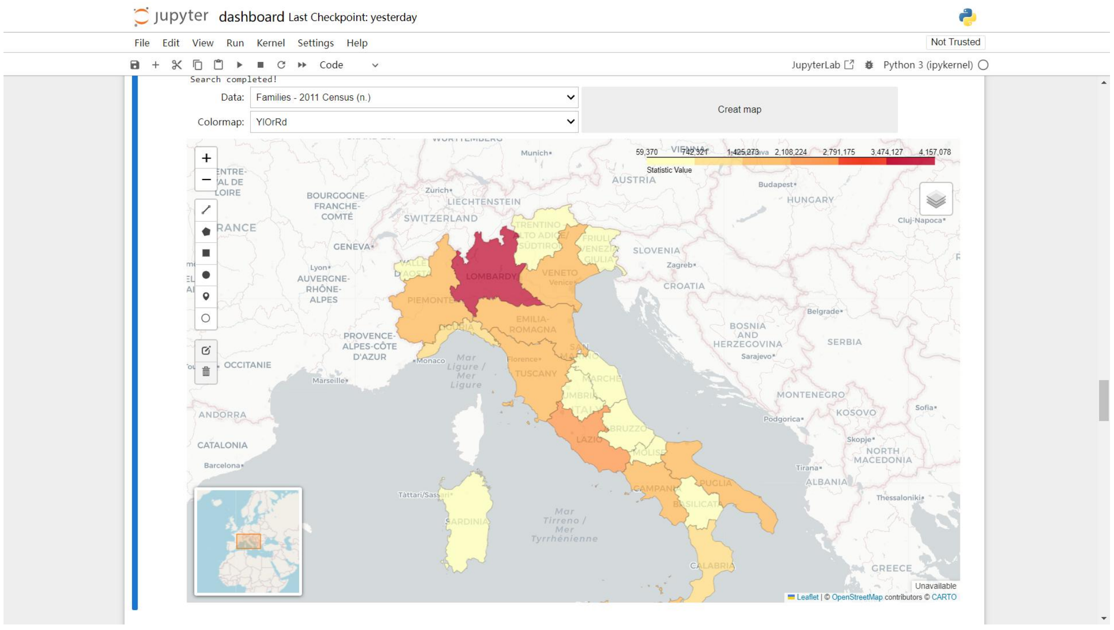
Scale: regioni

Type: surface area

Figure 26: Hierarchical Statistical Map

Select Data and Colormap: In the popup interface, choose the desired "data" and "colormap".

Create Map: Click the "create map" button. The map layer will be displayed using the selected color scheme.



```
Search completed!
```

Data: Families - 2011 Census (n.)

Colormap: YlOrRd

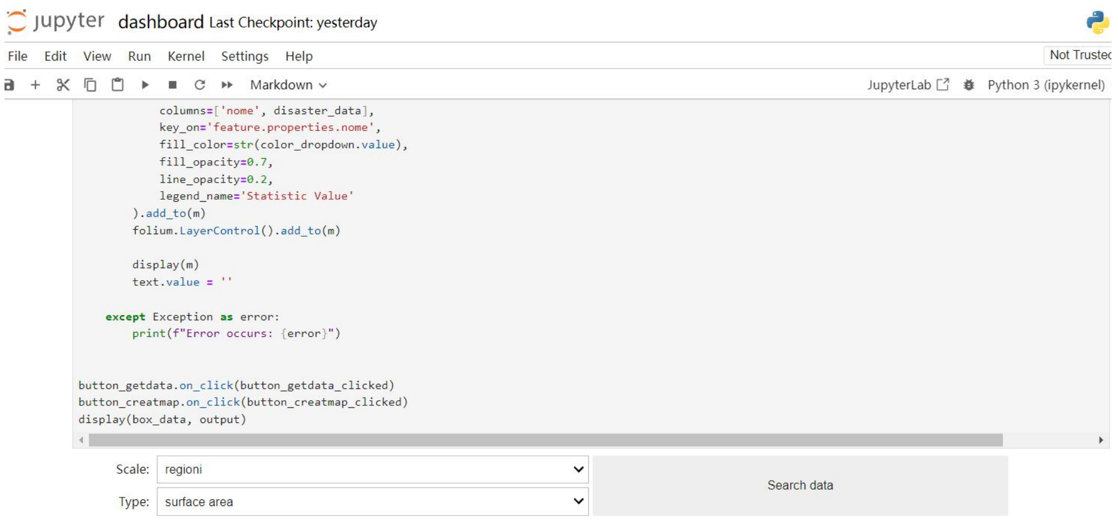
Figure 27: Colormap

**Functionality 3:** Heatmap Data Visualization

This functionality allows users to create a heatmap based on specified conditions.

Select Query Scale:In the "scale" dropdown menu, select the desired query scale.

Select Disaster Type: In the "type" dropdown menu, choose the disaster type.


jupyter dashboard Last Checkpoint: yesterday
File Edit View Run Kernel Settings Help
Not Trusted
Markdown
JupyterLab
Python 3 (ipykernel)

```
            columns=['nome', disaster_data],
            key_on='feature.properties.nome',
            fill_color=str(color_dropdown.value),
            fill_opacity=0.7,
            line_opacity=0.2,
            legend_name='Statistic Value'
        ).add_to(m)
        folium.LayerControl().add_to(m)

        display(m)
        text.value = ''

    except Exception as error:
        print(f"Error occurs: {error}")


button_getdata.on_click(button_getdata_clicked)
button_creatmap.on_click(button_creatmap_clicked)
display(box_data, output)
```

Scale: regioni

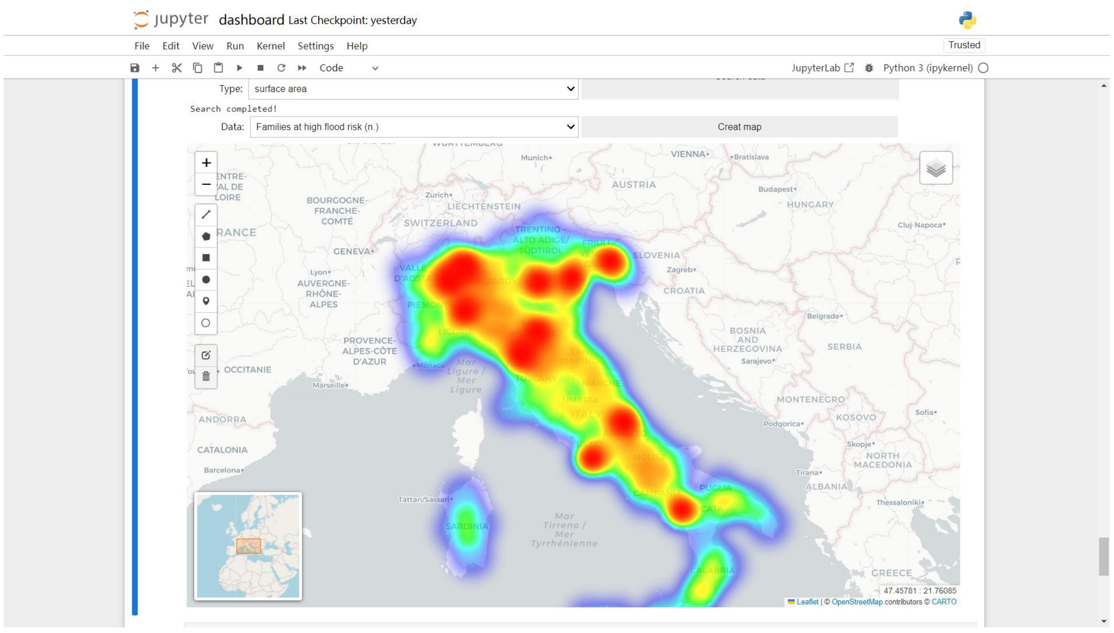
Type: surface area

Figure 28: scale and type

Search for Data: After performing the query, a new dropdown menu "data" will appear.

Select Heatmap Data: In the "data" dropdown menu, select the input data for the heatmap.

Visualize Heatmap: The software will generate and display the heatmap based on the selected data.


jupyter dashboard Last Checkpoint: yesterday
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
Creat map
Leaflet | © OpenStreetMap contributors © CARTO

Type: surface area

```
Search completed!
```

Data: Families at high flood risk (n.)

Figure 29: Heatmap Data Visualization

## 4. Known Limitations

### 4.1 List of known issues or limitations

#### 4.1.1 Performance Issues with Large Datasets:

The system may experience latency or slow performance when handling very large datasets due to the complexity of spatial queries and data processing.

#### 4.1.2 Browser Compatibility:

Some features may not be fully supported on all web browsers, potentially causing inconsistencies in user experience.

#### 4.1.3 Limited Mobile Support:

The user interface is optimized for desktop use, and certain visualizations and interactions may not function properly on mobile devices.

### 4.1.4 Real-time Data Processing:

While the system requests and processes data in real time, there may be delays due to server load or network latency, affecting the immediacy of data updates.

### 4.1.5 User Authentication and Authorization:

The current implementation of user authentication may have vulnerabilities or limitations that could affect the security of user data.

### 4.1.6 Geospatial Analysis Limitations:

The geospatial analysis capabilities, while robust, may not cover all possible use cases or complex spatial operations required by advanced users.