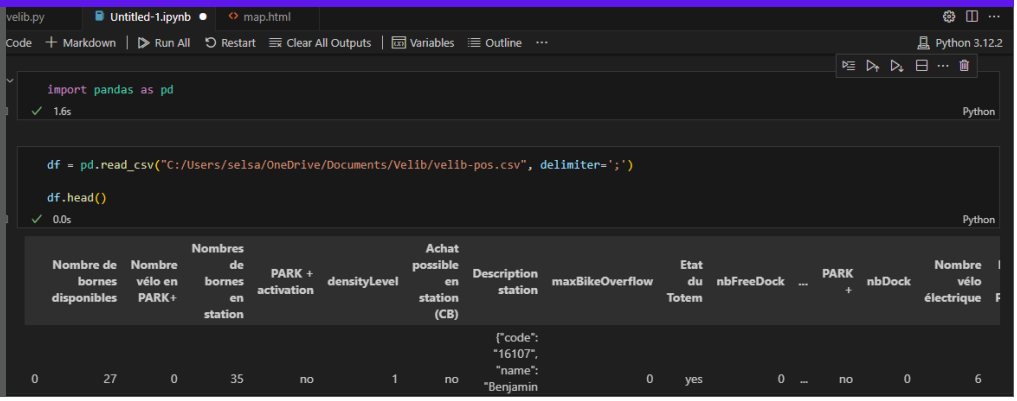
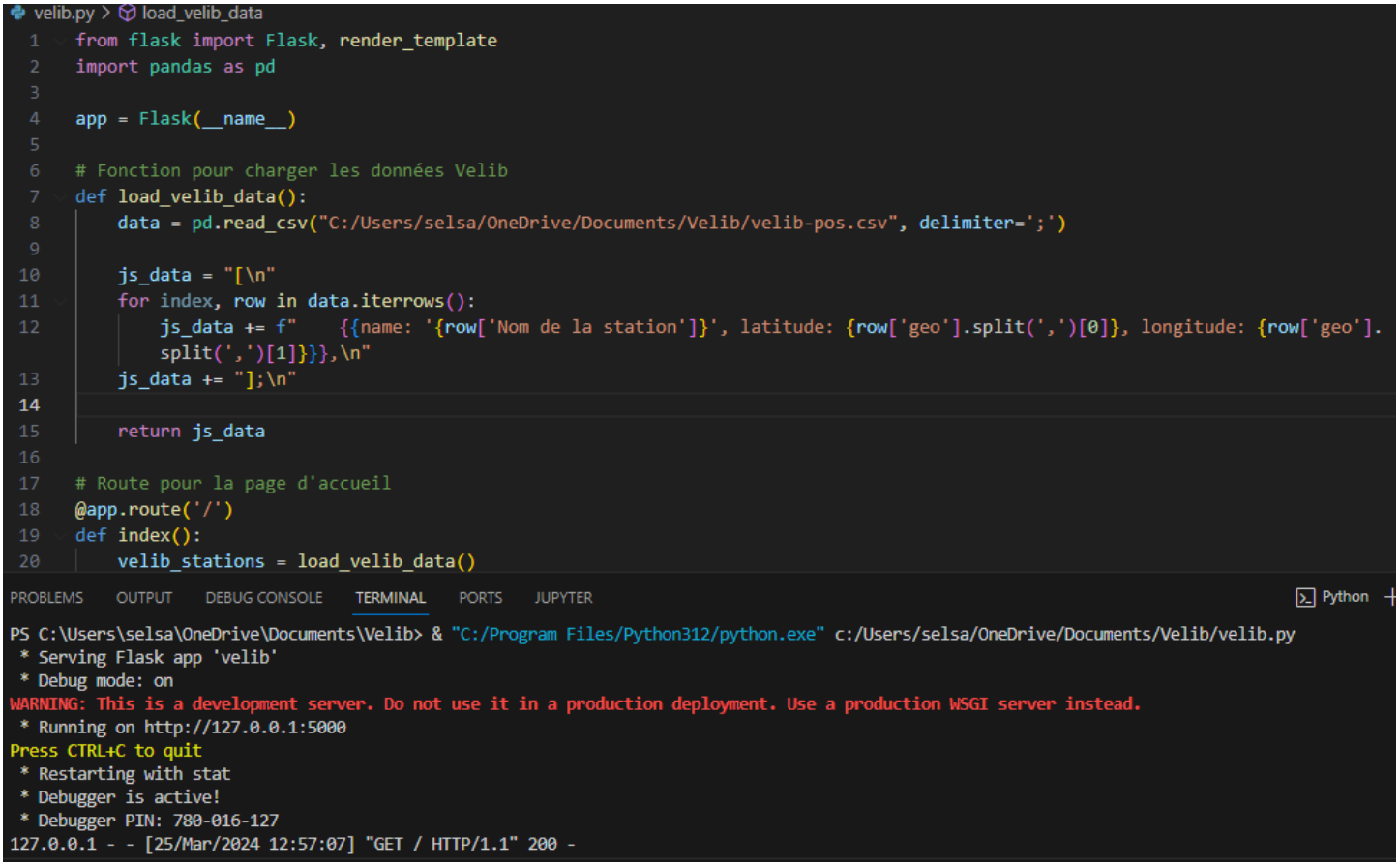
Première étape :

DataFrame

Pour commencer, j'ai téléchargé le fichier velib-pos.csv, qui contient les données sur les stations Velib. Ensuite, j'ai installer sur mon serveur. Une fois cela fait, j'ai utilisé la bibliothèque Pandas en Python . Cela m'a permis de visualiser facilement les données



Deuxième étape :



- from flask import Flask, render_template : J'ai importé la classe Flask de la bibliothèque Flask, ainsi que la fonction render_template qui me permet de rendre un template HTML.
- import pandas as pd : J'ai importé la bibliothèque Pandas sous l'alias pd. Je vais l'utiliser pour manipuler les données, pour lire le fichier CSV contenant les données Velib.

Création de l'application Flask :

- app = Flask(__name__) : J'ai initialisé une instance de l'application Flask.

Chargement des données Velib :

- load_velib_data() : J'ai défini cette fonction pour charger les données Velib à partir du fichier CSV. Elle lit le fichier velib-pos.csv à partir de l'emplacement spécifié, utilise Pandas pour le charger en tant que DataFrame, puis parcourt chaque ligne du DataFrame pour extraire le nom de la station, la latitude et la longitude, et les formater dans une chaîne JSON.

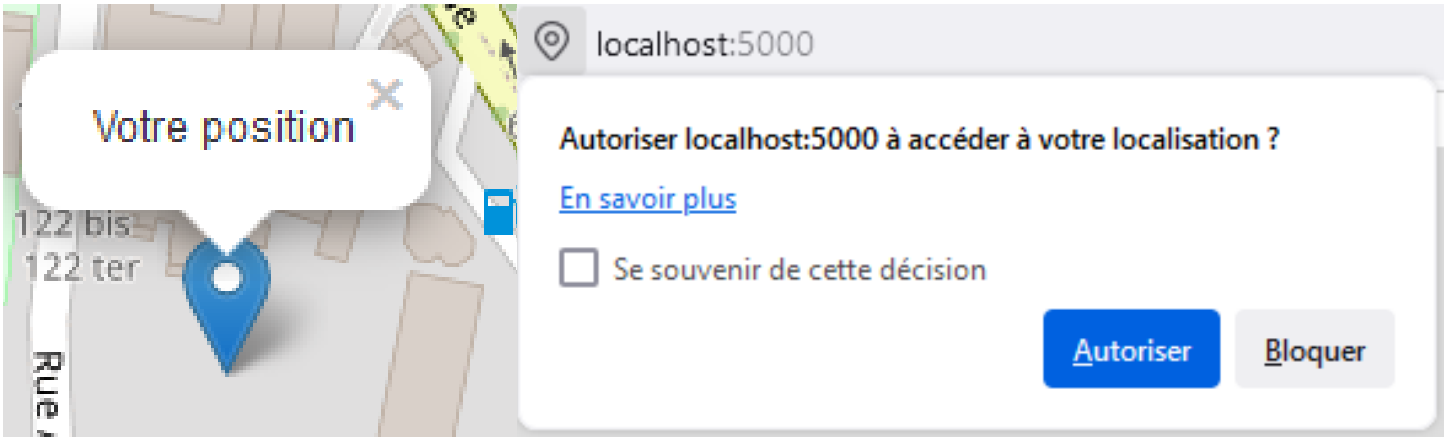
Définition de la route racine :

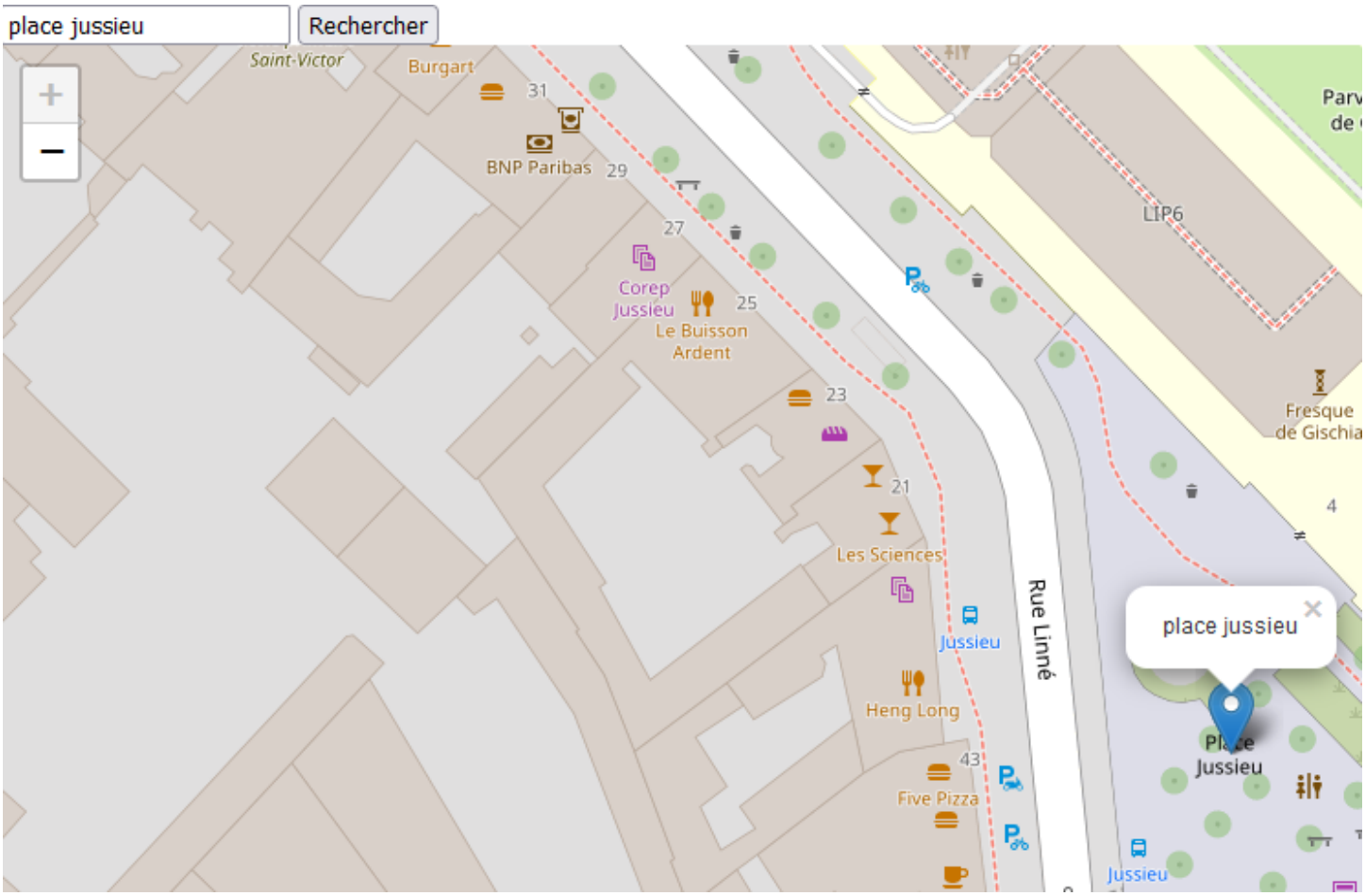
- @app.route('/') : J'ai défini une route pour la page d'accueil de mon application. Lorsque l'utilisateur accède à la racine de l'application, Flask appelle la fonction index().
- def index() : Dans cette fonction, j'ai chargé les données Velib en appelant load_velib_data(), puis j'ai rendu le template HTML map.html en passant les données Velib sous le nom v.

Exécution de l'application Flask :

- if __name__ == '__main__': : J'ai ajouté cette condition pour vérifier si le script est exécuté directement.
- app.run(debug=True) : Si c'est le cas, j'ai lancé l'application Flask en mode de débogage. Cela signifie que Flask affichera des informations de débogage en cas d'erreur. Enfin, l'application est exécutée sur http://127.0.0.1:5000.

Troisième étape :





Création de la structure HTML :

J'ai d'abord créé la structure HTML de base pour l'application en ajoutant un formulaire pour saisir une adresse et un conteneur pour afficher la carte Leaflet.

```
<form id="addressForm">
  <input type="text" id="addressInput" placeholder="Entrez une adresse sur Paris" />
  <button type="submit">Rechercher</button>
</form>
<div id="map"></div>
```

Initialisation de la carte et des données Velib :

Ensuite, j'ai initialisé la carte Leaflet avec var map = L.map('map'), et préchargé les données des stations Velib dans un tableau JavaScript.Les données des stations Velib sont préchargées dans un tableau JavaScript appelé velibData.

```
var map = L.map('map');
    var velibData = [
      { name: '1', latitude: 48.1234, longitude: 2.5678 },
```

```
        { name: '2', latitude: 48.2345, longitude: 2.6789 },
    ];
```

Gestion de la géolocalisation :

Pour gérer la géolocalisation, j'ai écrit une fonction onLocationFound(e) qui centre la carte sur la position de l'utilisateur, ajoute un marqueur à sa position et affiche les stations Velib à proximité. En cas d'erreur de géolocalisation, une autre fonction onLocationError(e) est appelée pour afficher un message d'erreur.

```
function onLocationFound(e) {
        var userCoords = e.latlng;
        map.setView(userCoords, 14);
// Ajout marqueur
        L.marker(userCoords).addTo(map)
          .bindPopup("Votre position");
```

Recherche d'adresse et affichage des stations Velib :

Lorsque l'utilisateur soumet le formulaire de recherche d'adresse, j'ai défini une fonction searchAddress(e) qui récupère l'adresse saisie, utilise un service de géocodage pour obtenir les coordonnées de l'adresse, puis centre la carte sur cette position. Elle ajoute également un marqueur pour l'adresse sur la carte et affiche les stations Velib à proximité de cette adresse.

```
function searchAddress(e) {
        e.preventDefault();
        var address = document.getElementById('addressInput').value;
        if (address.trim() === '') {
          alert("Veuillez entrer une adresse.");
          return;  }
```

Enfin, pour compléter le fonctionnement de l'application, j'ai ajouté des gestionnaires d'événements pour gérer la géolocalisation et la soumission du formulaire. De plus, afin d'enrichir la carte et d'afficher les données de manière visuelle, j'ai intégré une cartographie provenant d'OpenStreetMap.

```
map.on('locationfound', onLocationFound);
map.on('locationerror', onLocationError);
```

```
map.locate({setView: true, maxZoom: 16});

L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
    maxZoom: 19,
}).addTo(map);
```

En résumé, ce code crée une application web Flask qui charge les données Velib à partir d'un fichier CSV, les affiche sur une carte via un template HTML, et l'exécute localement sur le serveur de développement de Flask (sur http://127.0.0.1:5000).