In [8]:
```
import requests
import random



def get_error(a, b):
    url = f"http://ramcdougal.com/cgi-bin/error_function.py?a={a}&b={b}"
    headers = {"User-Agent": "MyScript"}
    response = requests.get(url, headers=headers)
    return float(response.text)

#Implement a two-dimensional version of the gradient descent algorithm to find optimal choices of a and b.

#identify any numerical choices
#justify why you think they were reasonable choices


#lr set to 0.1, as the learning rate should be small but not too small, lr = 0.1 is tested to be small enough to
#find the optimal but not too small to have too many query to make the server get error

# the delta is used to find the estimate of derivitive, like lr should be small enough and choose 0.001 as it is
# already small enough to get the correct estimate of derivitve

# the ending criteria is iteration more than 50 rounds or the gradient is already smaller than the tolerance. 0.001
# and the 50 round can already get an accurate enough answer and more accuracy just only add the time
def gradient_descent(a,b,lr=0.1, delta=0.001, max_iter=50, tolerance=0.001):

    for iteration in range(max_iter):

        current_error = get_error(a, b)

        # Explain how you estimate the gradient given that you cannot directly compute the derivative :

        # Cannot directly compute the derivative, so use f(x+delta)-f(x)/delta to estimate the derivative by
        # a very small delta, then we can have a estimate value of the gradient

        grada = (get_error(a + delta, b) - current_error) / delta
        gradb = (get_error(a, b + delta) - current_error) / delta

        anew = a - lr * grada
        bnew = b - lr * gradb

        grad_magnitude = (grada**2 + gradb**2)**0.5
        if grad_magnitude < tolerance:
            print(f"Converged in {iteration+1} iterations")
            break

        a, b = anew, bnew
    return a, b, current_error

# Run the gradient descent function
l = []
for i in range(10):
    optimal_a, optimal_b, optimal_error = gradient_descent(random.random(),random.random())
    l.append((optimal_a, optimal_b, optimal_error))
print(l)
```

```
Converged in 9 iterations
Converged in 32 iterations
Converged in 34 iterations
Converged in 32 iterations
Converged in 31 iterations
Converged in 9 iterations
Converged in 27 iterations
Converged in 30 iterations
Converged in 33 iterations
Converged in 34 iterations
[(0.7115884800404466, 0.16844794334339053, 1.00000142235), (0.21532354733998538, 0.6880529265178437, 1.10000135454), (0.2159864401506323,
0.6884253981649446, 1.10000033035), (0.21565328459503763, 0.6880315827814683, 1.10000105804), (0.2158890238379243, 0.6883086092421641, 1.1
0000049034), (0.7116038966984986, 0.1684556274020933, 1.00000135972), (0.21529567586768117, 0.6881145119523935, 1.10000128016), (0.2159419
7575266383, 0.688566275540969, 1.10000019148), (0.2158845869307605, 0.6886355542585675, 1.10000014614), (0.21593878305073577, 0.6884714491
445231, 1.10000028311)]
```

Find both locations (i.e. a, b values) querying the API as needed and identify which corresponds to which
Randomly select the starting points for several times(here use 10 times), from the result we can see there are two optimal, the local minimum is 1.1 at arround a=0.215 and b=0.688, the global minimum is 1.0 at arround a=0.711 and b = 0.168

Briefly discuss how you would have tested for local vs global minima if you had not known how many minima there were. The method to test whether it is local minima if we dont know there is how many minima is to generate many random starting points and see if there is smaller optimal we can get. If there are smaller results, then it is the local minima. If there is no smallller result, it is probably the global optimal.

Problem2

Data Source This project uses data from the World Cities Database, provided by SimpleMaps.

License The data is licensed under the Creative Commons Attribution 4.0 International (CC BY 4.0).

URL https://simplemaps.com/data/world-cities

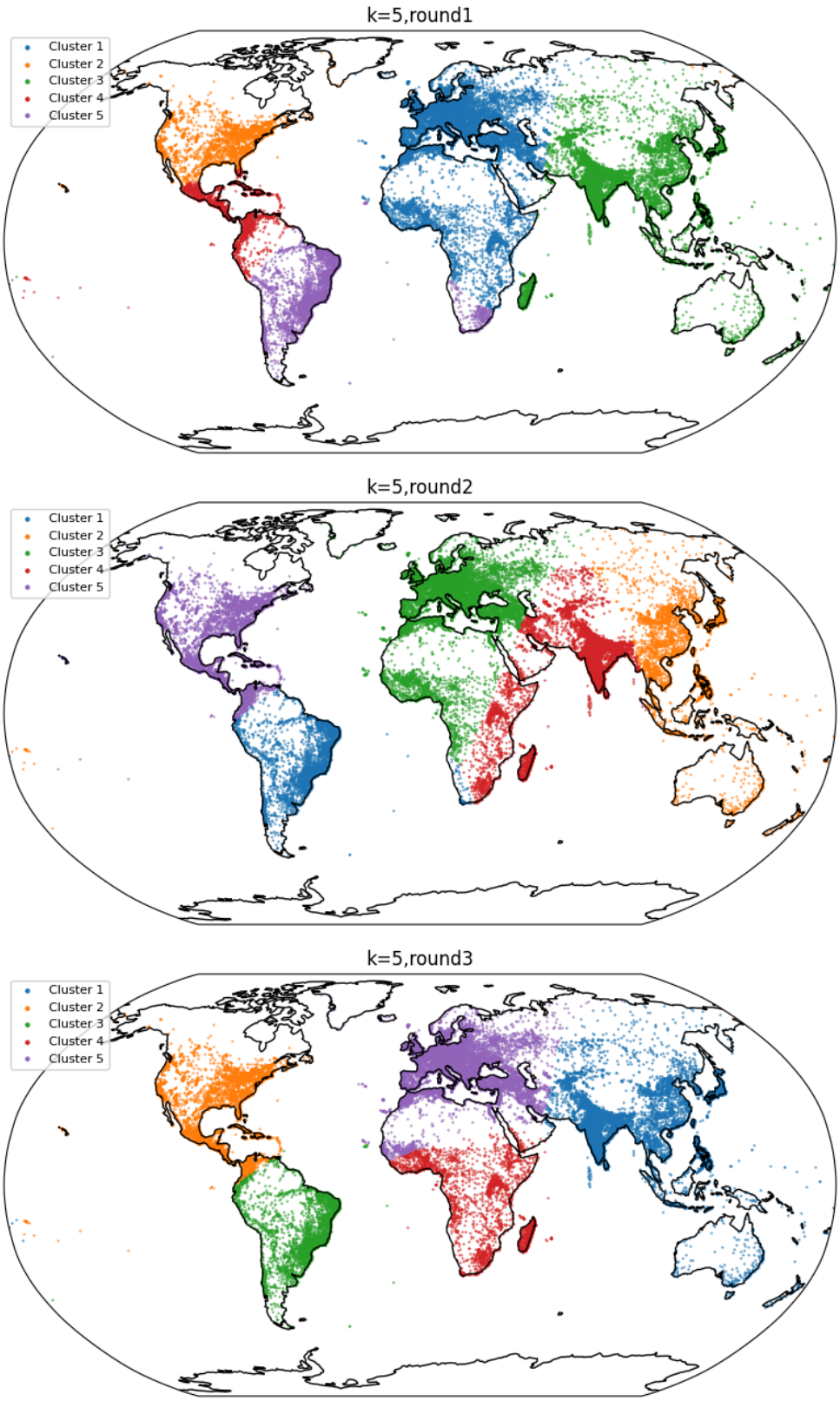
In [2]:
```
import pandas as pd
world_cities = pd.read_csv('worldcities.csv')
```

In [3]:
```
import numpy as np
def lat_lon_to_cartesian(lat, lon):
    """Convert latitude and longitude to Cartesian coordinates."""
    lat_rad = np.radians(lat)
    lon_rad = np.radians(lon)
    x = np.cos(lat_rad) * np.cos(lon_rad)
    y = np.cos(lat_rad) * np.sin(lon_rad)
    z = np.sin(lat_rad)
    return np.array([x, y, z])


def cartesian_to_lat_lon(cartesian_coords):
    """Convert Cartesian coordinates to latitude and longitude."""
    x, y, z = cartesian_coords
    lon = np.arctan2(y, x)
    hyp = np.sqrt(x**2 + y**2)
    lat = np.arctan2(z, hyp)
    return np.degrees(lat), np.degrees(lon)
```

In [4]:
```
world_cities['cartesian_coords'] = world_cities.apply(lambda row: lat_lon_to_cartesian(row['lat'], row['lng']), axis=1)
```

In [42]:
```
def cosine_similarity(vec1, vec2):
    """Calculate cosine similarity between two vectors."""
    return np.dot(vec1, vec2) / (np.linalg.norm(vec1) * np.linalg.norm(vec2))

# K-means clustering algorithm using cosine similarity
def k_means_cosine(data, k, max_iters=100):
    centroids = data[np.random.choice(len(data), k, replace=False)]

    for _ in range(max_iters):
        clusters = [[] for _ in range(k)]
        for point in data:
            similarities = [cosine_similarity(point, centroid) for centroid in centroids]
            closest_centroid = np.argmax(similarities)
            clusters[closest_centroid].append(point)

        new_centroids = []
        for cluster in clusters:
            if cluster:  # avoid division by zero
                new_centroid = np.mean(cluster, axis=0)
                new_centroids.append(new_centroid)
            else:
                new_centroids.append(data[np.random.randint(len(data))])
        if np.allclose(new_centroids, centroids, atol=1e-5):
            break

        centroids = new_centroids

    return clusters, centroids

data_points = np.array(world_cities['cartesian_coords'].tolist())

clusters_5_set = []
for _ in range(3):
    clusters_5, centroids_5 = k_means_cosine(data_points, k=5)
    clusters_5_set.append(clusters_5)
```

In [43]:
```
import cartopy.crs as ccrs
import matplotlib.pyplot as plt
for j in range(3):
    cluster_points_5 = [[cartesian_to_lat_lon(point) for point in cluster] for cluster in clusters_5_set[j]]

    # Plotting the clusters on a Robinson projection
    fig, ax = plt.subplots(figsize=(10, 5), subplot_kw={"projection": ccrs.Robinson()})
    ax.set_global()
    ax.coastlines()

    # Plot each cluster in a different color

    for i in range(5):
        lons, lats = zip(*cluster_points_5[i])
        ax.scatter(
            lats, lons, s=0.2, transform=ccrs.PlateCarree(), label=f"Cluster {i+1}"
        )

    # Adding legend and title
    plt.legend(loc="upper left", markerscale=5, fontsize=8)
    plt.title(f"k=5,round{j+1}")
    plt.show()
```

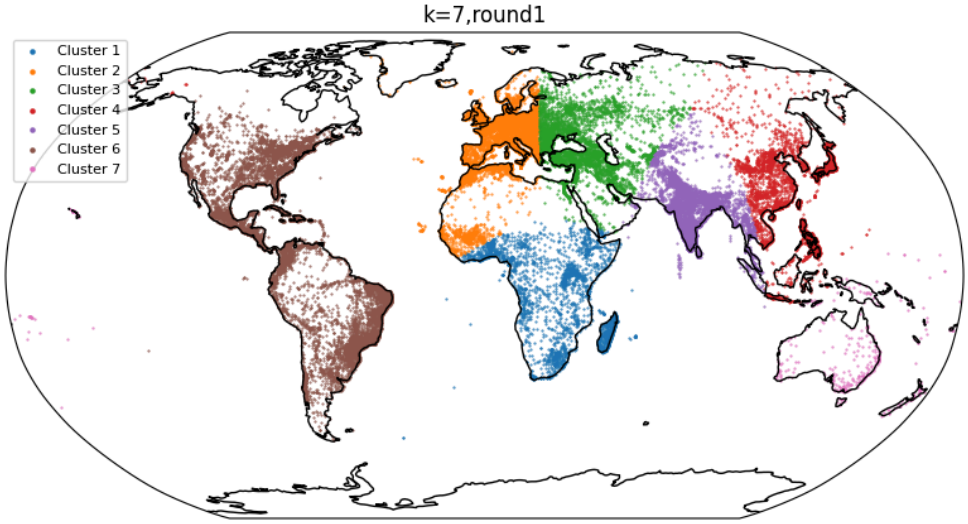


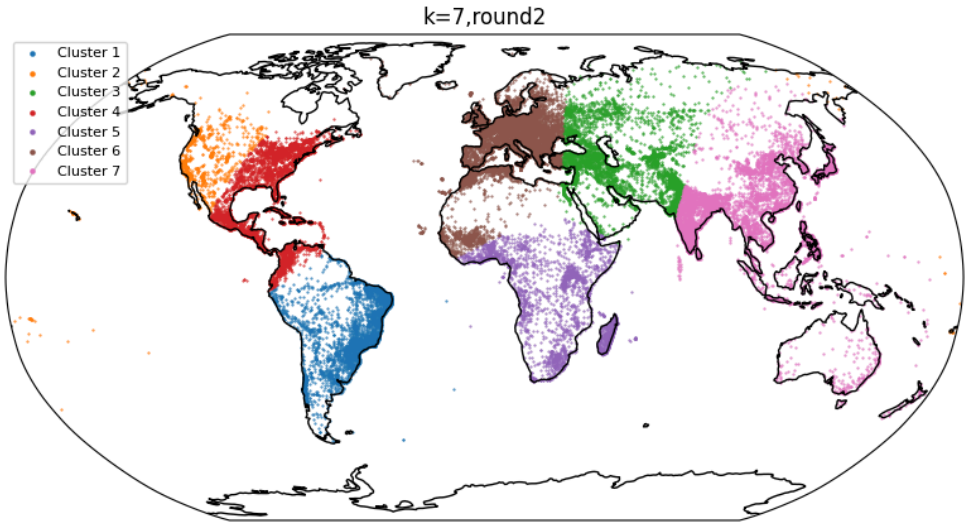
For cluster equals 5, the different rounds illustrate that cluster boundaries are sensitive to the initialization of centroids and the iterative process of k-means. We can see that North and South American might having the problem clustering together, the asia and austrilia also might be cluster together.

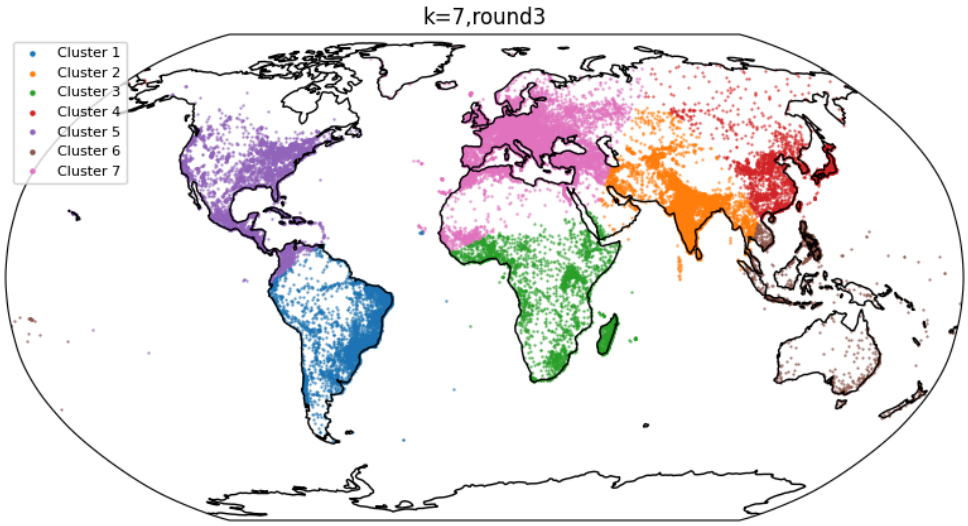
In [ ]:
```
clusters_7_set = []
for _ in range(3):
    clusters_7, centroids_7 = k_means_cosine(data_points, k=7)
    clusters_7_set.append(clusters_7)
```

In [ ]:
```
for j in range(3):
    cluster_points_7 = [[cartesian_to_lat_lon(point) for point in cluster] for cluster in clusters_7_set[j]]

    # Plotting the clusters on a Robinson projection
    fig, ax = plt.subplots(figsize=(10, 5), subplot_kw={"projection": ccrs.Robinson()})
    ax.set_global()
    ax.coastlines()
```

```
    # Plot each cluster in a different color

    for i in range(7):
        lons, lats = zip(*cluster_points_7[i])
        ax.scatter(
            lats, lons, s=0.2, transform=ccrs.PlateCarree(), label=f"Cluster {i+1}"
        )

    # Adding legend and title
    plt.legend(loc="upper left", markerscale=5, fontsize=8)
    plt.title(f"k=7,round{j+1}")
    plt.show()
```



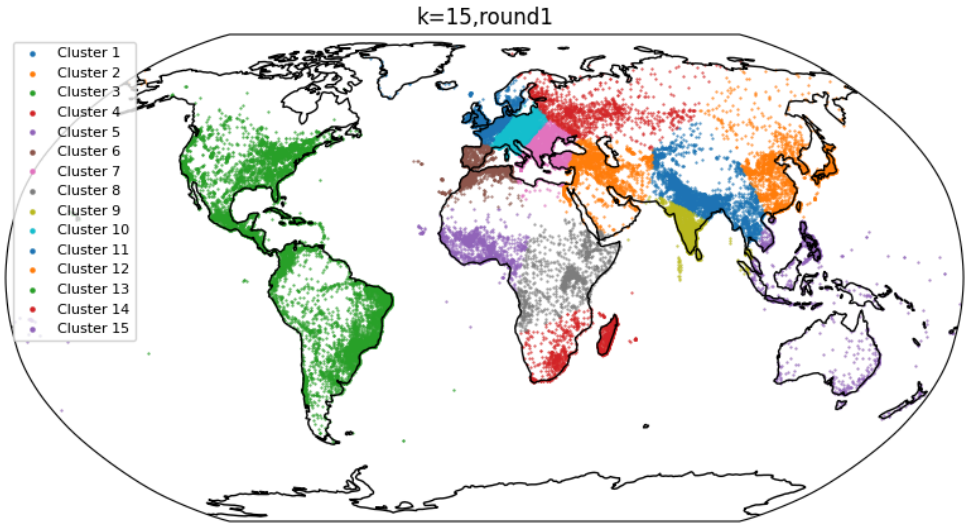


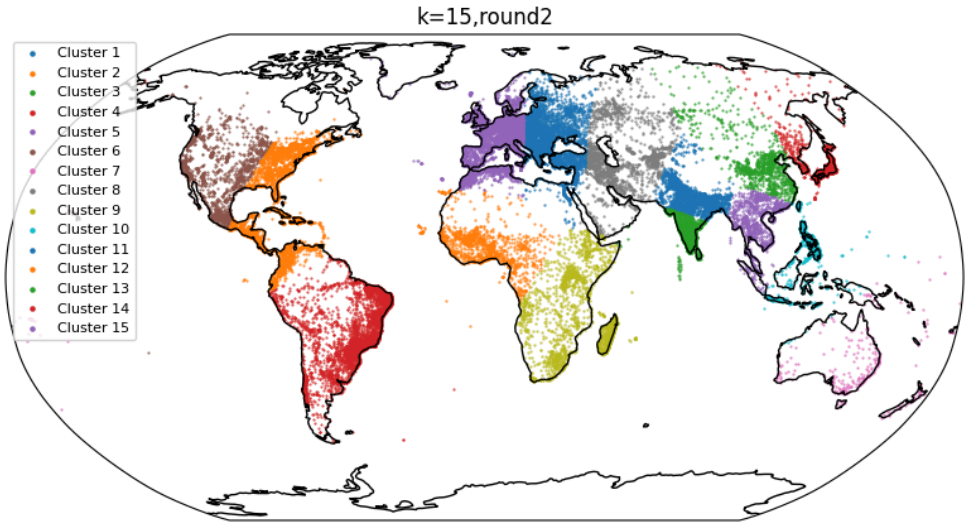


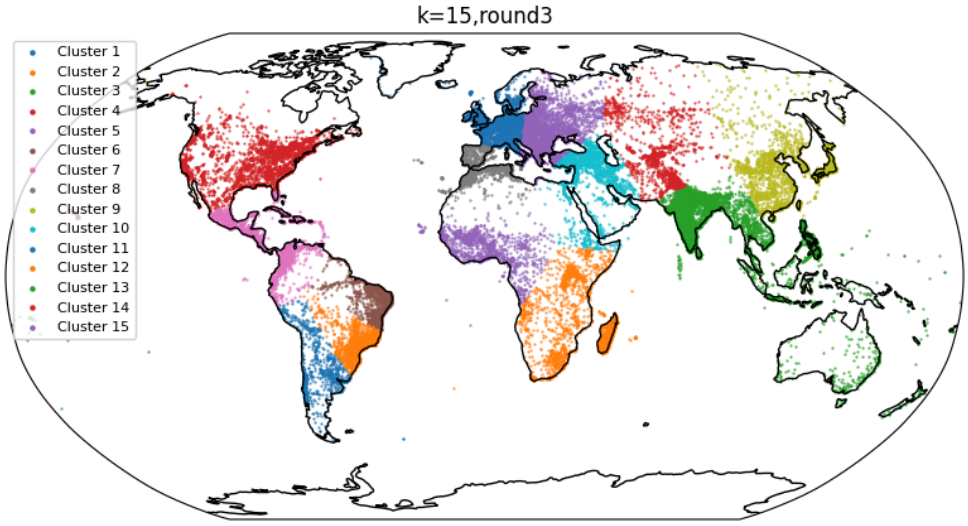
For cluster equals 7, it is still sensitive to the initial points, however there are potential to have the right cluster that can classify by the continents. It still might have some problems on dividing the asia and europe continent and australia. Overall the cluster is better than k equals 5.

In [47]:
```
clusters_15_set = []
for _ in range(3):
    clusters_15, centroids_15 = k_means_cosine(data_points, k=15)
    clusters_15_set.append(clusters_15)
```

In [48]:
```
for j in range(3):
    cluster_points_15 = [[cartesian_to_lat_lon(point) for point in cluster] for cluster in clusters_15_set[j]]

    # Plotting the clusters on a Robinson projection
    fig, ax = plt.subplots(figsize=(10, 5), subplot_kw={"projection": ccrs.Robinson()})
    ax.set_global()
    ax.coastlines()

    # Plot each cluster in a different color

    for i in range(15):
        lons, lats = zip(*cluster_points_15[i])
        ax.scatter(
            lats, lons, s=0.2, transform=ccrs.PlateCarree(), label=f"Cluster {i+1}"
        )

    # Adding legend and title
    plt.legend(loc="upper left", markerscale=5, fontsize=8)
    plt.title(f"k=15,round{j+1}")
    plt.show()
```





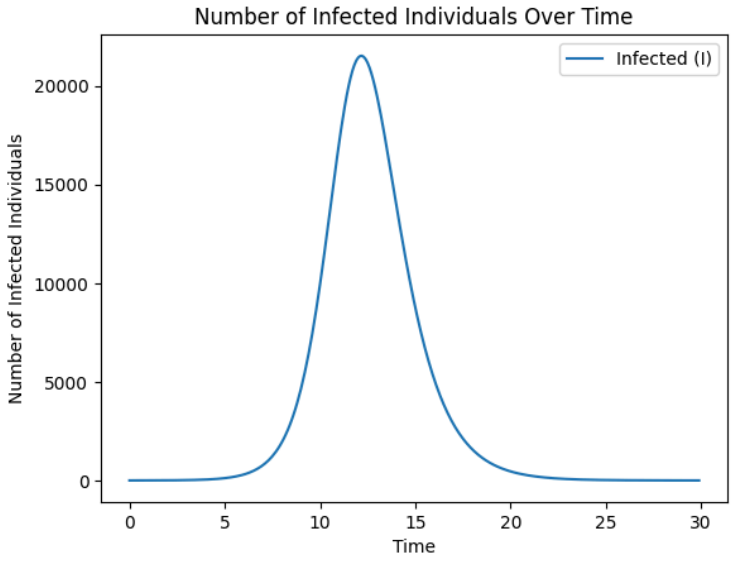


For cluster number equals 15, each cluster is much more inside in one continent. However sometimes it still cannot divide properly, crossing two continents, maybe for those cities they are more close to each other. The cluster is still initial sensitive.

Problem 3

In [1]:
```
# provide your own implementation of SIR
def sir_model(S0, I0, R0, beta, gamma, Tmax, dt=0.1):
    N = S0 + I0 + R0
    svalue,ivalue,rvalue,tvalue = [S0],[I0],[R0],[0]
    S, I, R, T = S0, I0, R0, 0
    for i in range(int(Tmax/dt)):
        dS = -beta * S * I / N
        dI = beta * S * I / N - gamma * I
        dR = gamma * I
        S += dS * dt
        I += dI * dt
        R += dR * dt
        T += dt
        svalue.append(S)
        ivalue.append(I)
        rvalue.append(R)
        tvalue.append(T)

        if I < 1:
            break
    return svalue,ivalue,rvalue,tvalue
```

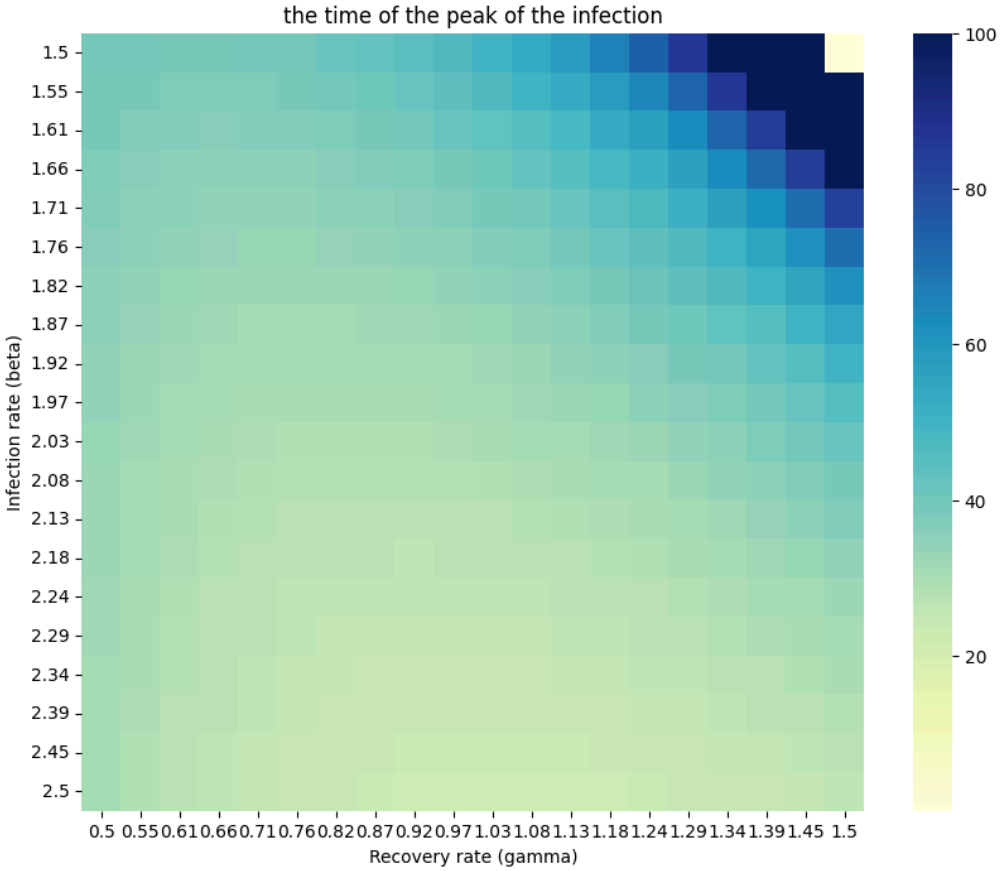
In [2]:
```
# Plot the time course of the number of infected individuals until that number drops below 1
import matplotlib.pyplot as plt
import numpy as np
svalue, ivalue, rvalue,tvalue = sir_model(136999, 1, 0, 2, 1, 100)
plt.plot(tvalue, ivalue, label='Infected (I)')
plt.xlabel('Time')
plt.ylabel('Number of Infected Individuals')
plt.title('Number of Infected Individuals Over Time')
plt.legend()
plt.show()
```

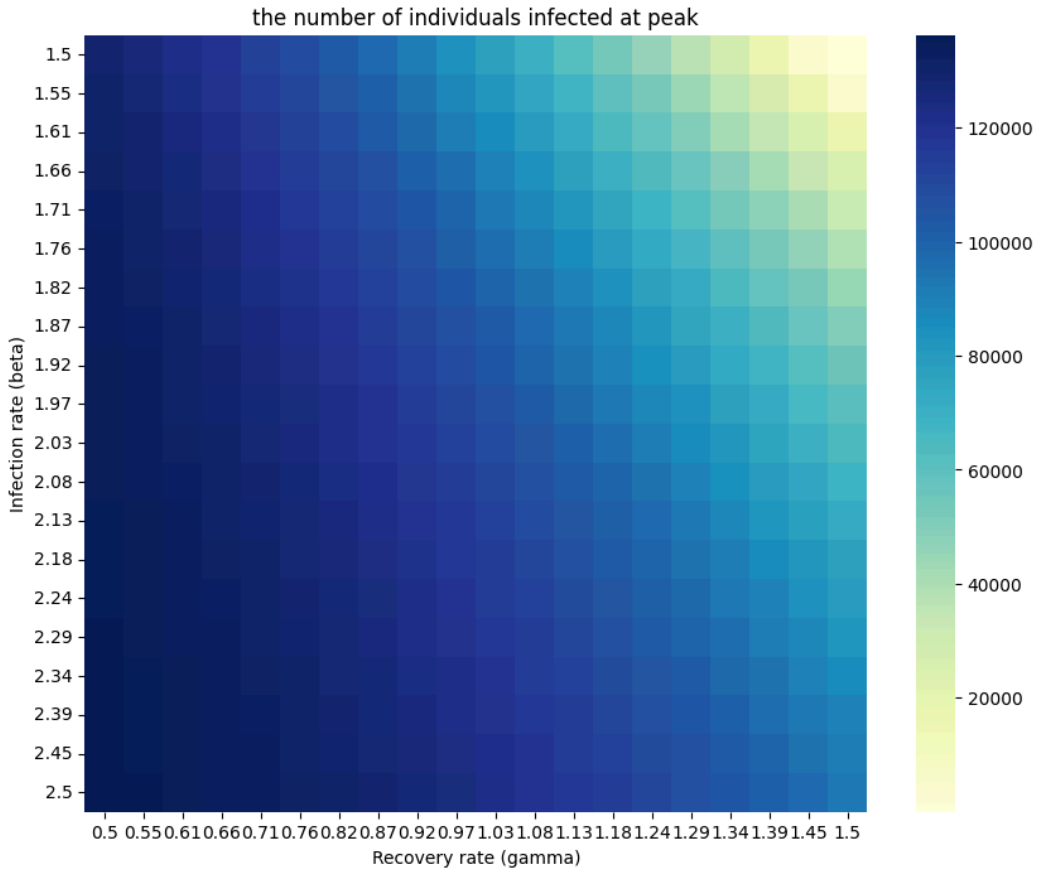


In [3]:
```
#  when does the number of infected people peak?  How many people are infected at the peak?
peak_infected = max(ivalue)
peak_time = tvalue[np.argmax(ivalue)]
print(f"Peak number of infected individuals: {peak_infected}")
print(f"Time at peak infection: {peak_time}")
```

```
Peak number of infected individuals: 21525.699125405023
Time at peak infection: 12.199999999999973
```

In [ ]:
```
import seaborn as sns
beta_values = np.linspace(1.5, 2.5, 20)
gamma_values = np.linspace(0.5, 1.5, 20)
peak_times = np.zeros((len(beta_values), len(gamma_values)))
peak_infections = np.zeros((len(beta_values), len(gamma_values)))


for i, beta in enumerate(beta_values):
    for j, gamma in enumerate(gamma_values):
        _, _, ivalue, tvalue = sir_model(136999, 1, 0, beta, gamma, 100)
        peak_infections[i, j] = max(ivalue)
        peak_times[i, j] = tvalue[np.argmax(ivalue)]


plt.figure(figsize=(10, 8))
sns.heatmap(peak_times, xticklabels=np.round(gamma_values, 2), yticklabels=np.round(beta_values, 2), cmap="YlGnBu")
plt.xlabel("Recovery rate (gamma)")
plt.ylabel("Infection rate (beta)")
plt.title("the time of the peak of the infection")
plt.show()
plt.figure(figsize=(10, 8))
sns.heatmap(peak_infections, xticklabels=np.round(gamma_values, 2), yticklabels=np.round(beta_values, 2), cmap="YlGnBu")
plt.xlabel("Recovery rate (gamma)")
plt.ylabel("Infection rate (beta)")
plt.title("the number of individuals infected at peak")
plt.show()
```





Problem 4

What does each file do? How are they interconnected? Are there any key parts of the files for making the server do something?

server.py: sets up the Flask server and allows to debug and open the website to operate it.
the index route renders the main page at index.html, The /analyze route processes the text input, performs a simple character count using Counter, and passes the result to the analyze.html template.

index.html: This HTML file presents the main form where users can input text. It is where the first interface user opens.

analyze.html: This HTML file displays the analyzed results. It receives and shows the user's input and the analysis

the input text that it takes, the output that it returns, and how you can determine the response from the data.

the input text it takes is a number from 0 - 87, the output is the number of how many this type of interaction does the whole data contains, I can use the count method to my dataset to determine the response from the data

create an interactive website that asks for input from the user, has Python perform the relevant data lookup in your dataset, and displays the results in a new webpage.

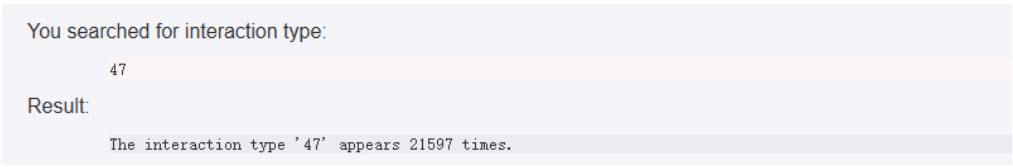
the code connot process in ipynb so the code is just shows below but not for process

In [ ]:
```python
from flask import Flask, render_template, request
from collections import Counter
import pandas as pd
import matplotlib.pyplot as plt

app = Flask(__name__)

data = pd.read_csv(r"C:\Users\jackchang\Desktop\regular_python_work\Yale\BIS634\problem_set_3\ddi_training.csv")

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


@app.route("/analyze", methods=["POST"])
def analyze():
    interaction_type = request.form["interaction_type"]
    analysis_type = request.form["analysis_type"]
    #  error handling
    if analysis_type == "count":
        try:
            count = data['type'].value_counts().get(int(interaction_type), 0)
        except:
            result = 'invalid number'
            return render_template("analyze.html", analysis=result, interaction_type=interaction_type)
```

```
        if count == 0:
            result = f"The interaction type '{interaction_type}' was not found in the dataset."
        else:
            result = f"The interaction type '{interaction_type}' appears {count} times."

    elif analysis_type == "list_top":
        top_interactions = data['type'].value_counts().head(5)
        result = "Top 5 interaction types:\n" + "\n".join([f"{item}: {count}" for item, count in top_interactions.items()])

    counts = data['type'].value_counts()
    colors = ['blue' if item != int(interaction_type) else 'red' for item in counts.index]
    print(colors)
    plt.figure(figsize=(10, 6))
    counts.plot(kind='bar', color=colors)
    plt.title('Interaction Type Counts')
    plt.xlabel('Interaction Type')
    plt.ylabel('Count')
    plt.savefig(r'C:\Users\jackchang\Desktop\regular_python_work\Yale\BIS634\problem_set_3\static\interaction_plot.png')
    plt.close()
    return render_template("analyze.html", analysis=result, interaction_type=interaction_type)


if __name__ == "__main__":
    app.run(debug=True)
```

asks for input from the user:

Python perform the relevant data lookup in your dataset, and displays the results in a new webpage

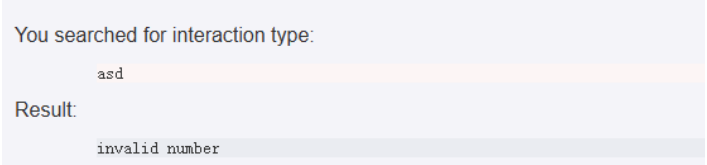


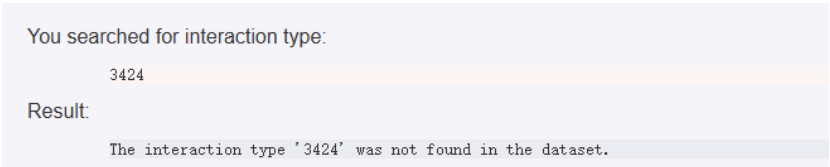
Extra credits

(1) error handling

```
#  error handling
if analysis_type == "count":
    try:
        count = data['type'].value_counts().get(int(interaction_type), 0)
    except:
        result = 'invalid number'
        return render_template("analyze.html", analysis=result, interaction_type=interaction_type)

    if count == 0:
        result = f"The interaction type '{interaction_type}' was not found in the dataset."
    else:
        result = f"The interaction type '{interaction_type}' appears {count} times."
```

can handling if the input is not a number or the number of interaction does not exist



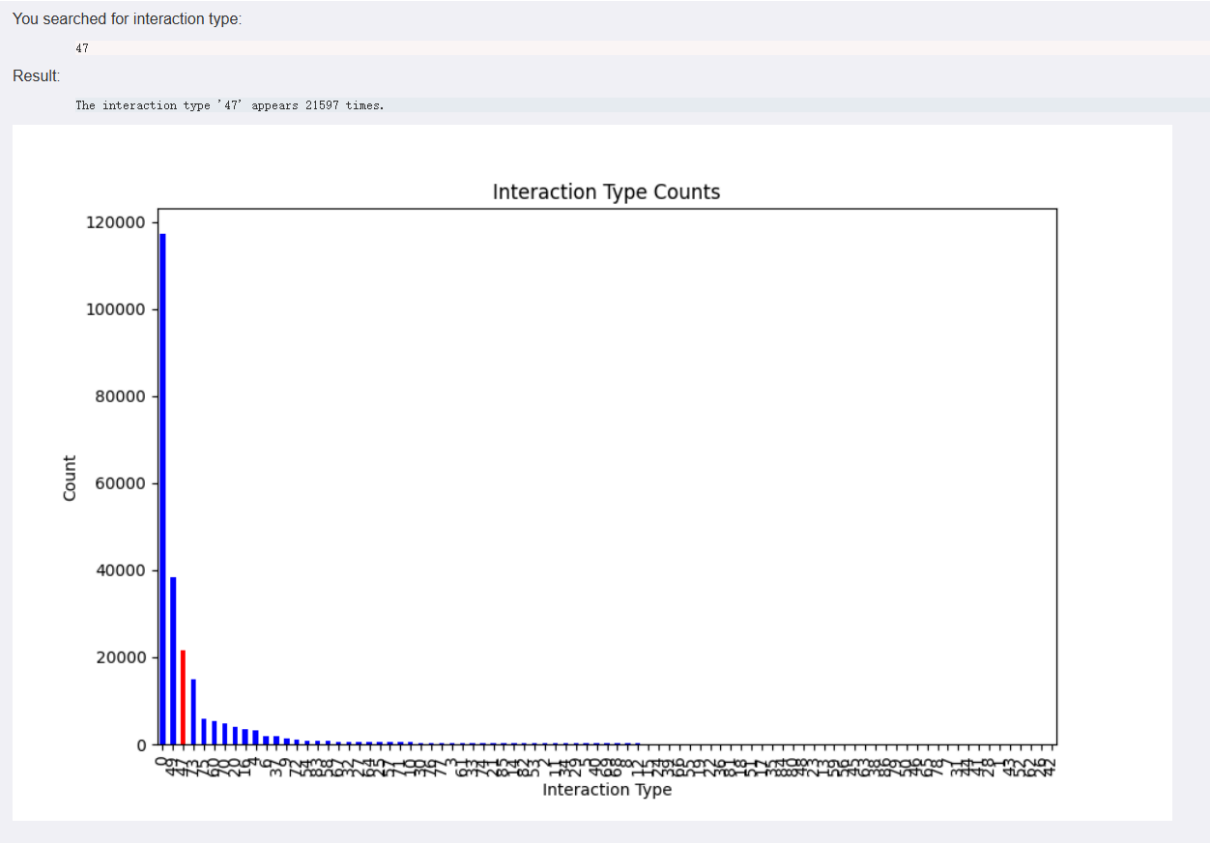


(2) styling with CSS

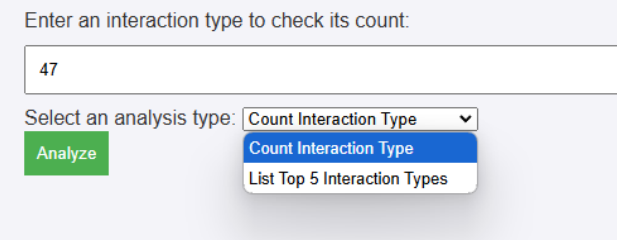
```css
body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f9;
    color: #333;
    margin: 0;
    padding: 20px;
}

h1 {
    color: #4CAF50;
    text-align: center;
}

textarea, input[type="text"] {
    width: 100%;
    padding: 10px;
    margin: 10px 0;
}

input[type="submit"] {
    background-color: #4CAF50;
    color: white;
    padding: 10px;
```

include in analyze.html and index.html by

(3) relevant static image,(4) an image generated in response to the input



(6) multiple analyses.



You searched for interaction type:

```
47
```

Result:

```
Top 5 interaction types:
0: 117219
49: 38339
47: 21597
73: 14908
75: 5906
```