# *Contact tracing using DBSCAN algorithm*

# Import the libraries that are required:

- We use Pandas for data manipulation like extracting and writing the data e.t.c
- Scikit-learn for machine-learning
- Matplotlib and Seaborn for data visualization

In [1]:
```
import numpy as np
import pandas as pd
from PIL import Image
from sklearn.cluster import DBSCAN
from datetime import datetime
import plotly.graph_objects as go
from sklearn.metrics import silhouette_score, calinski_harabasz_score, davies_bo
import plotly.express as px
import seaborn as sns
import matplotlib.pyplot as plt
```

Importing the data from livedata.json dataset

In [2]:
```
df = pd.read_json("livedata.json")
```

To get the concise summary of the DataFrame which prints information about the DataFrame including the index dtype, column dtypes, non-null values, and memory usage, we use:

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype         
---  ------     --------------  -----         
 0   id         100 non-null    object        
 1   timestamp  100 non-null    datetime64[ns]
 2   latitude   100 non-null    float64       
 3   longitude  100 non-null    float64       
dtypes: datetime64[ns](1), float64(2), object(1)
memory usage: 3.3+ KB
```

To get the statistics including those that summarize the Count(total samples in dataset),mean of latitude,mean of longitude,standard deviation(std),minimum,maximum that helps to restrict the plot size, we use:

In [4]:
```
df.describe()
```

Out[4]:

| | timestamp | latitude | longitude |
|---|---|---|---|
| **count** | 100 | 100.000000 | 100.000000 |
| **mean** | 2020-07-04 17:41:29.999999232 | 13.134709 | 77.639229 |
| **min** | 2020-07-04 12:35:30 | 13.010284 | 77.553381 |
| **25%** | 2020-07-04 14:35:30 | 13.081266 | 77.596577 |
| **50%** | 2020-07-04 17:05:30 | 13.133868 | 77.646240 |
| **75%** | 2020-07-04 20:35:30 | 13.195195 | 77.681645 |
| **max** | 2020-07-04 23:35:30 | 13.249645 | 77.705454 |
| **std** | NaN | 0.069591 | 0.046132 |

The first 5 rows to check if the object has the right type of data in it:

In [5]:
```
df.head()
```

Out[5]:

| | id | timestamp | latitude | longitude |
|---|---|---|---|---|
| **0** | David | 2020-07-04 15:35:30 | 13.148953 | 77.593651 |
| **1** | David | 2020-07-04 16:35:30 | 13.222397 | 77.652828 |
| **2** | Frank | 2020-07-04 14:35:30 | 13.236507 | 77.693792 |
| **3** | Carol | 2020-07-04 21:35:30 | 13.163716 | 77.562842 |
| **4** | Ivan | 2020-07-04 22:35:30 | 13.232095 | 77.580273 |

All the unique names that are present in the dataset

In [6]:
```
df['id'].unique()
```

Out[6]:
```
array(['David', 'Frank', 'Carol', 'Ivan', 'Erin', 'Bob', 'Grace', 'Alice',
       'Judy', 'Heidi'], dtype=object)
```

These minimum and maximum values of latitude and longitude are required to plot with exact size.

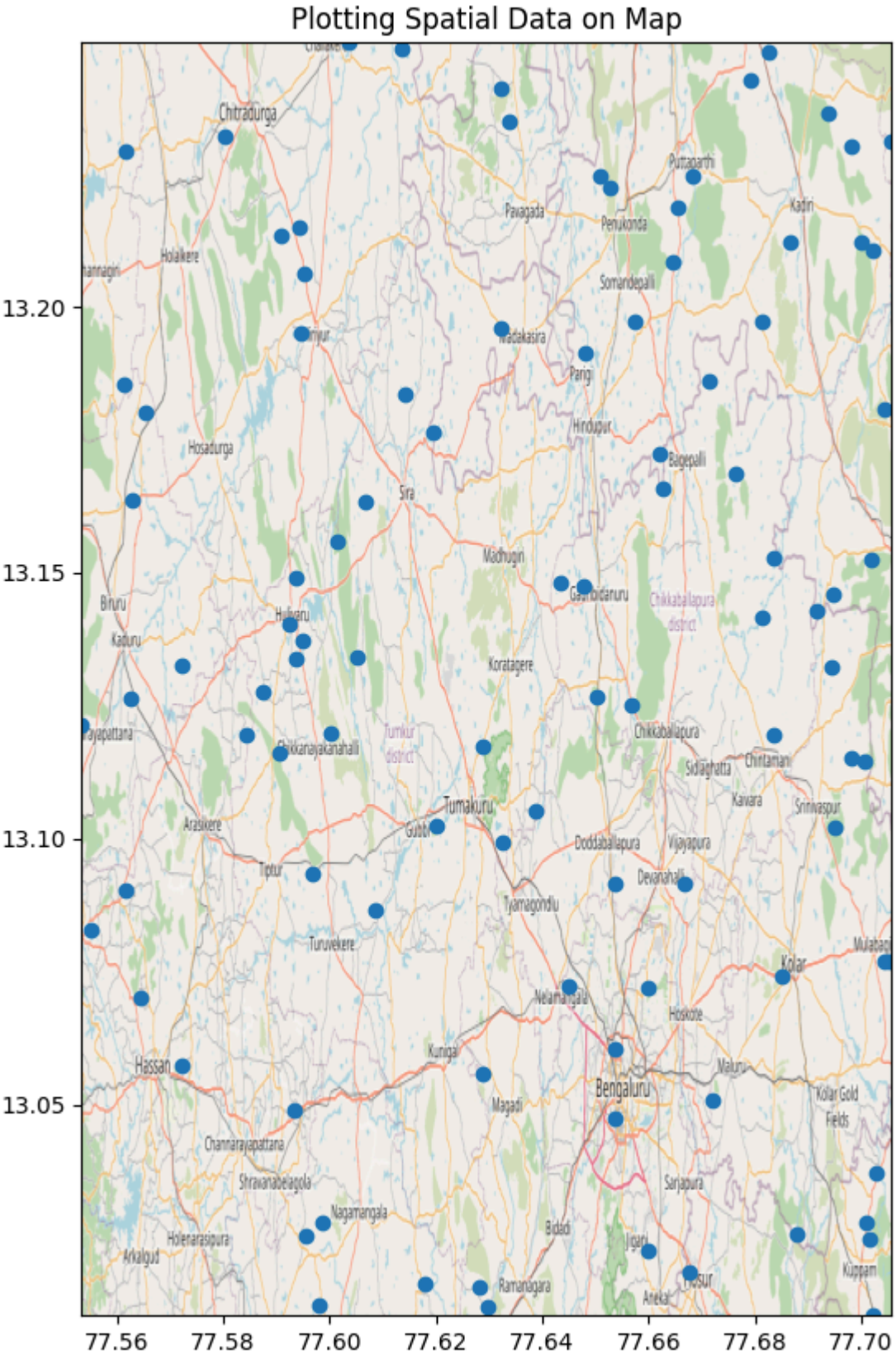
In [7]:
```
BBox = (df.longitude.min(), df.longitude.max(), df.latitude.min(), df.latitude.m
BBox
```

Out[7]:
```
(77.5533811, 77.7054541, 13.0102837, 13.2496455)
```

The dataset provided consists of data collected from Bangalore city. To display it against an accurate backdrop, I have utilized a map of Bangalore and plotted the real-time positions of the data points.

In [8]:
```
banglore_m = plt.imread('map.png')
fig, ax = plt.subplots(figsize = (10,10))
ax.scatter(df.longitude, df.latitude)
ax.set_title('Plotting Spatial Data on Map')
ax.set_xlim(BBox[0],BBox[1])
```

```
ax.set_ylim(BBox[2],BBox[3])
ax.imshow(banglore_m, extent = BBox, aspect= 'equal')
plt.show()
```



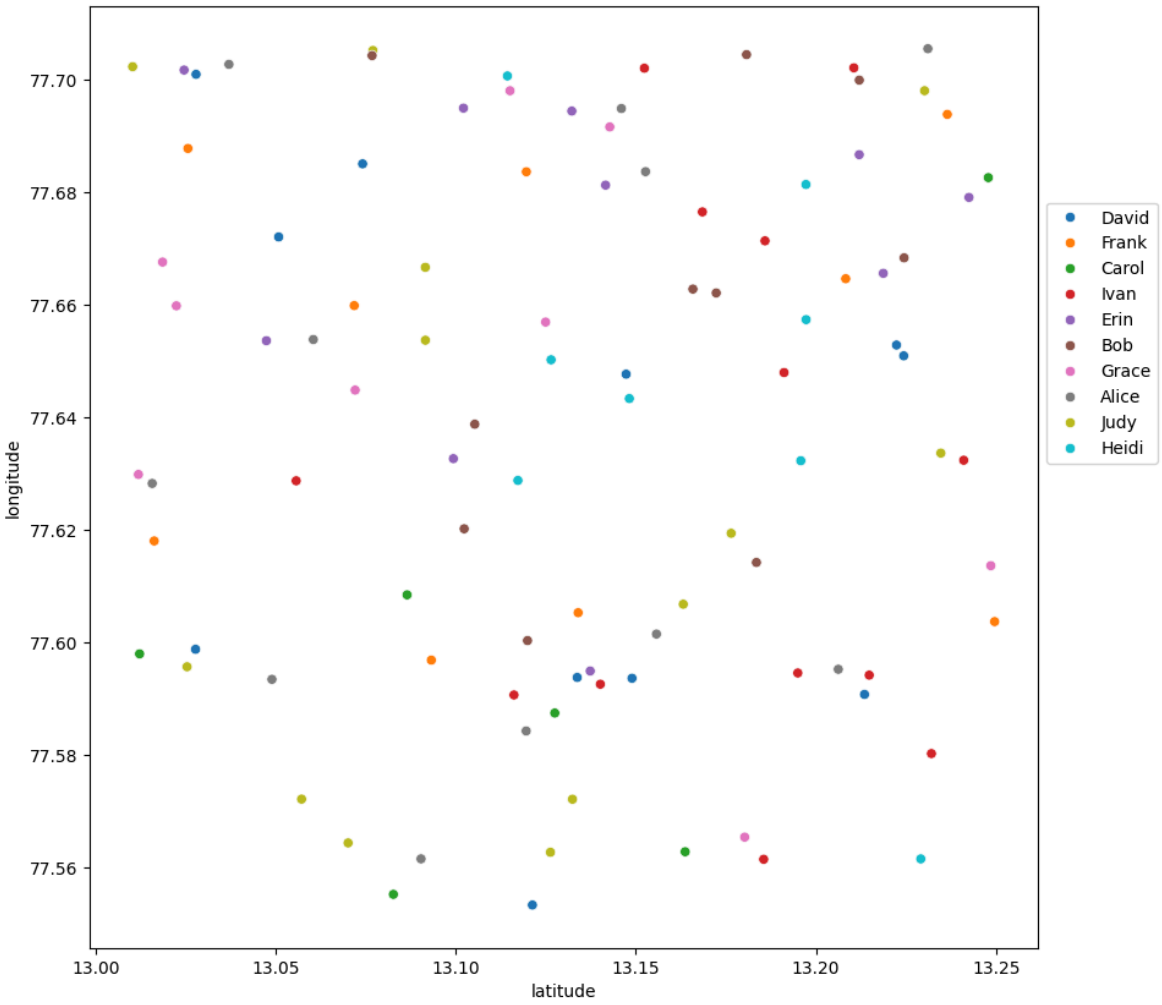

# EDA (Exploratory data analysis) :

Let us analyze the dataset using the Scatter plot showing the ids with their latitudes and longitudes across the x-axis and y-axis respectively.

**To understand the data more, we will plot it using matplotlib.pyplot the scatter plot. We can extract the different locations of each person.**

In [9]:
```
plt.figure(figsize=(10,10))
sns.scatterplot(x='latitude', y='longitude', data=df, hue='id')
plt.legend(bbox_to_anchor= [1, 0.8])
```

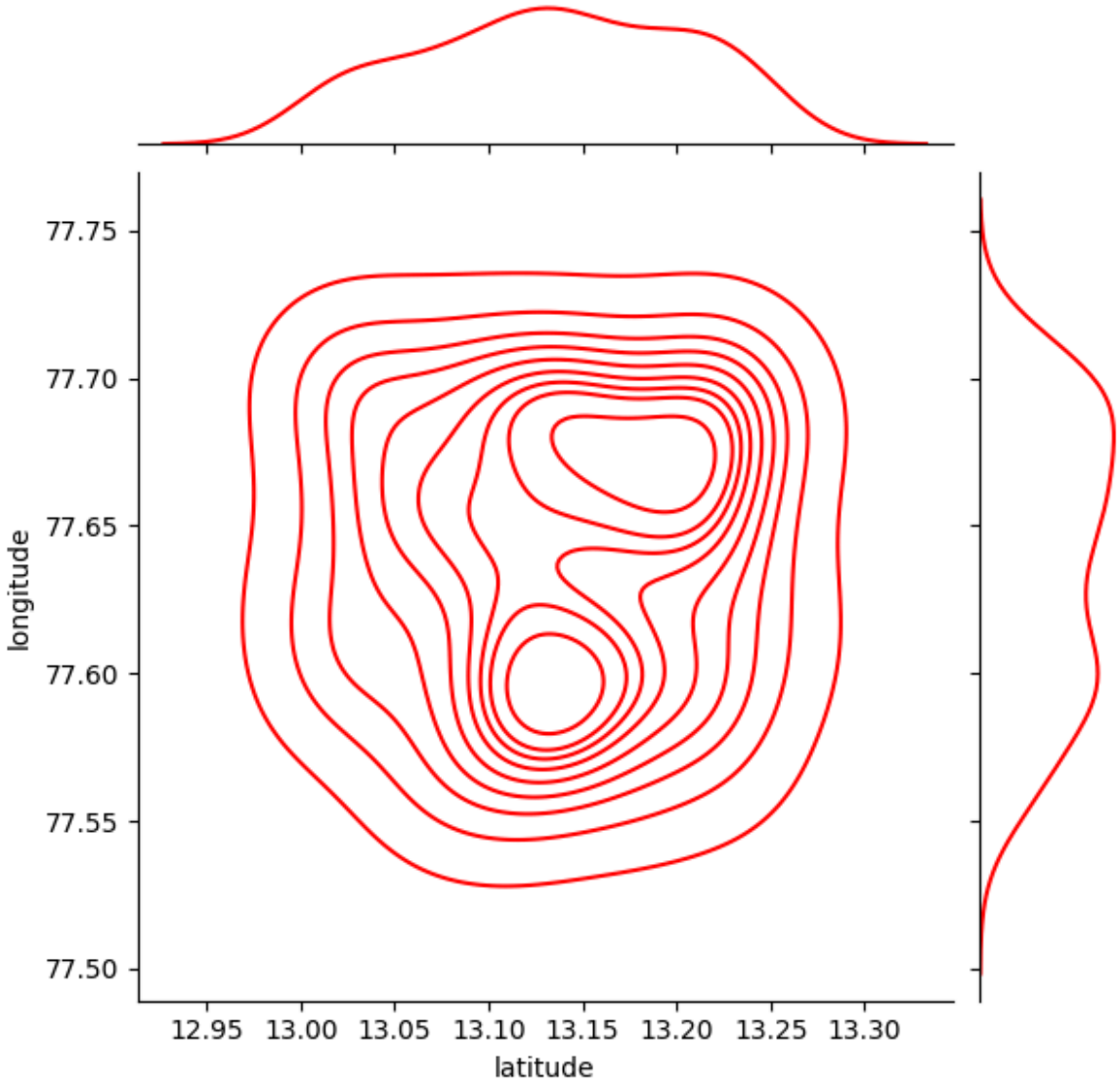
Out[9]:
```
<matplotlib.legend.Legend at 0x163db3e2f00>
```




Creating a joint plot (scatter plot and histograms) of latitude and longitude to analyze the same data using continuous plot, i.e it describes where most of the people are present and about their geographic location.

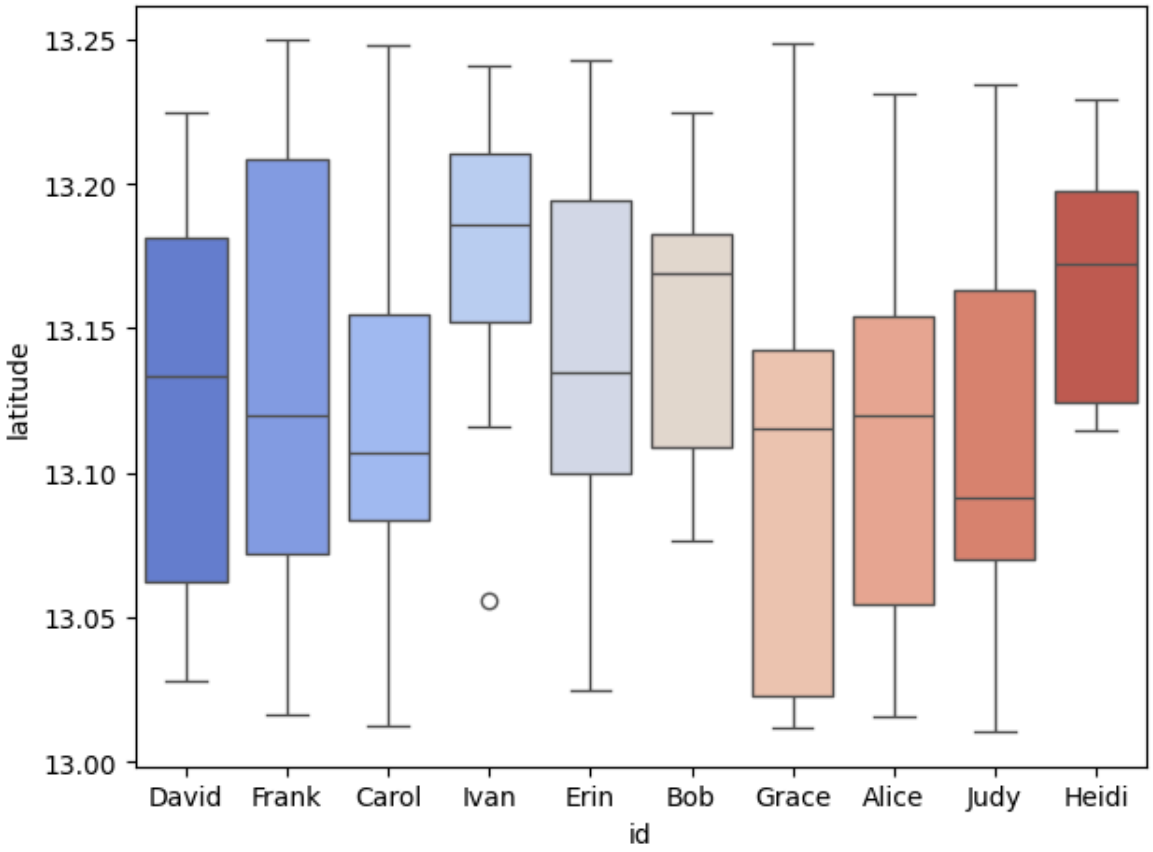
In [10]:
```
sns.jointplot(x='latitude', y='longitude', data=df, color='red', kind='kde')
```

Out[10]:
```
<seaborn.axisgrid.JointGrid at 0x163db2efb60>
```



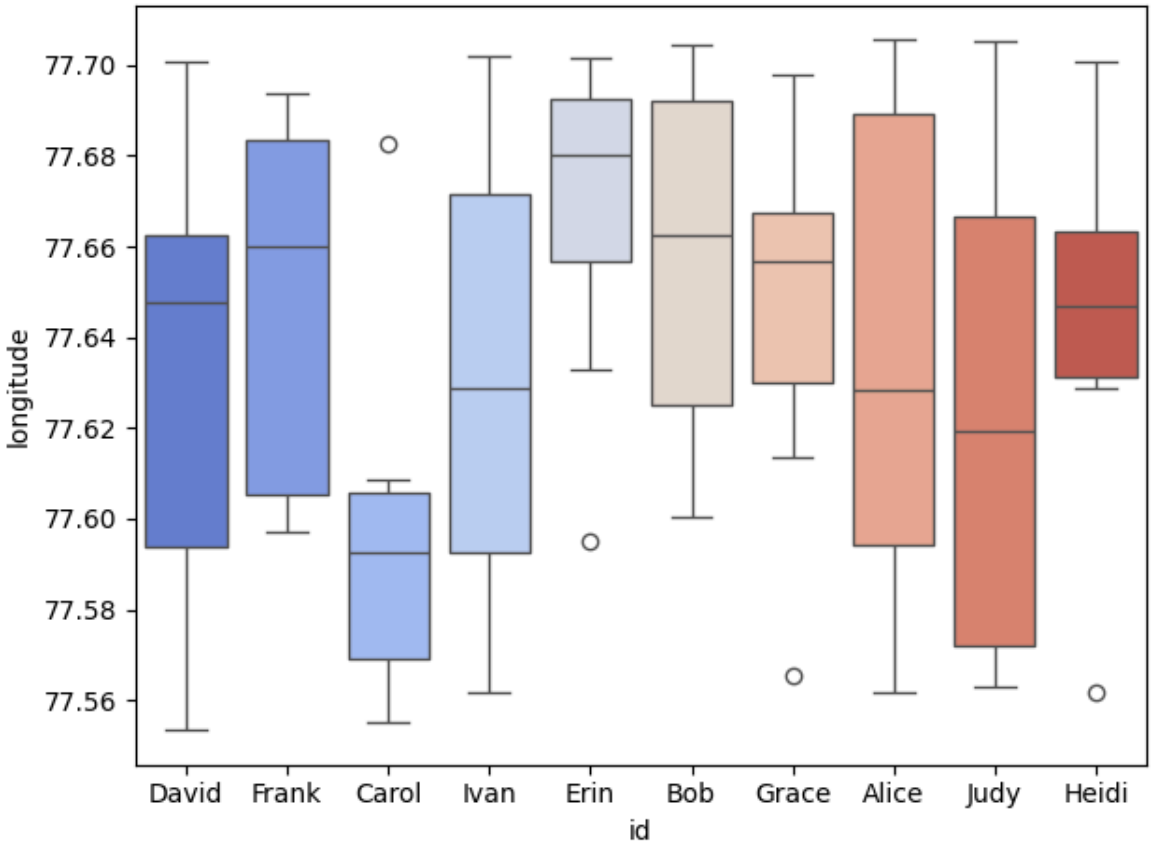

Now use the box plot to visualize the distribution of ids and latitudes in the x-axis and y-axis respectively.

In [11]:
```
sns.boxplot(data=df, x='id', y='latitude', hue='id', palette='coolwarm', legend=
plt.tight_layout()
```




Similarly, we'll plot to visualize the distribution of ids and longitudes in the x-axis and y-axis respectively.

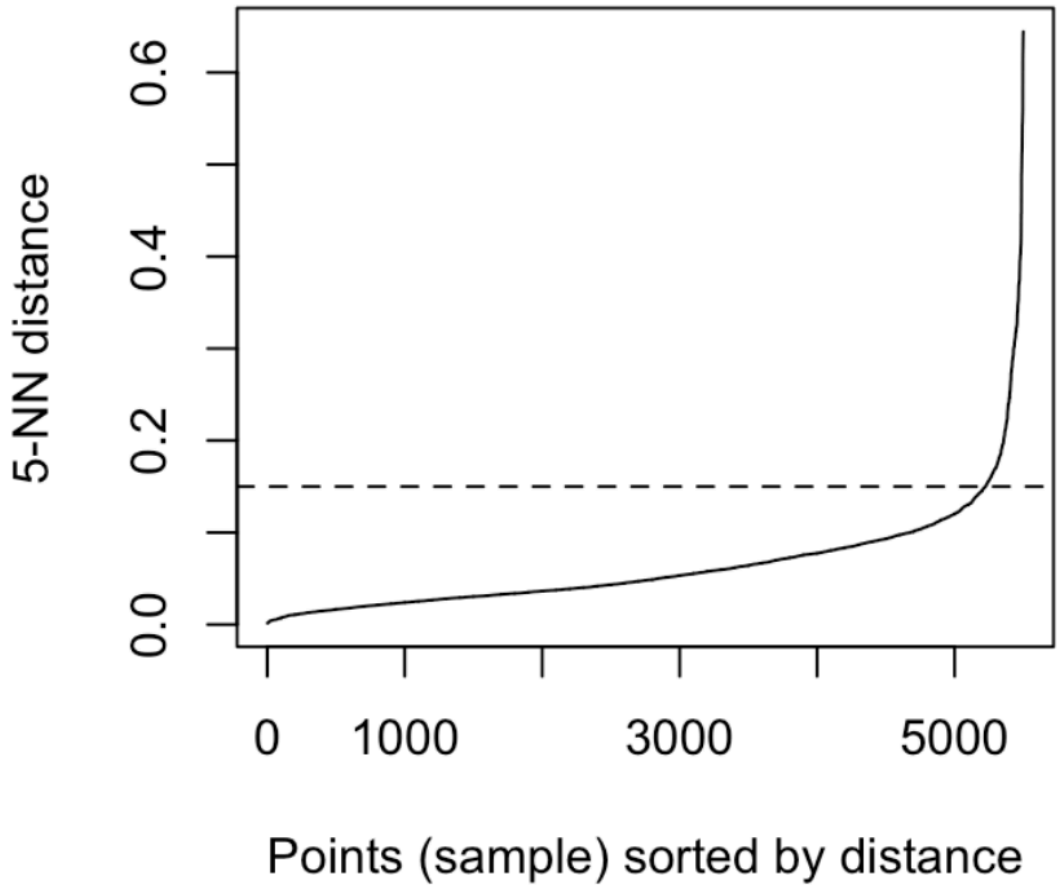
In [12]:
```
sns.boxplot(data=df, x='id', y='longitude', hue='id', palette='coolwarm', legend
plt.tight_layout()
```



# *Defining the model*

The subsequent code segment showcases the methodology for constructing a model, employing it to generate distinct clusters, and subsequently refining the data within these clusters to pinpoint potential infection occurrences. This process involves several key steps. Initially, the DBSCAN algorithm is utilized to define and establish the model's parameters. This algorithm excels at identifying clusters of varying shapes and densities within a dataset. Once the model is defined, it is applied to the data, resulting in the formation of clusters based on the proximity and density of data points. Finally, the data within each cluster is meticulously filtered to isolate and identify potential infection cases. This filtering process might involve considering factors like the number of infected individuals within a cluster or the duration of exposure. By combining clustering and filtering techniques, the code effectively highlights potential infection hotspots and aids in understanding the spread of a disease..

It has been take a reference from website to know the optimal value for the epsilon which is the radius around which it makes cluster. From that it has been found that optimal epsilon value lies between (0 - 0.2) as shown in the below graph:



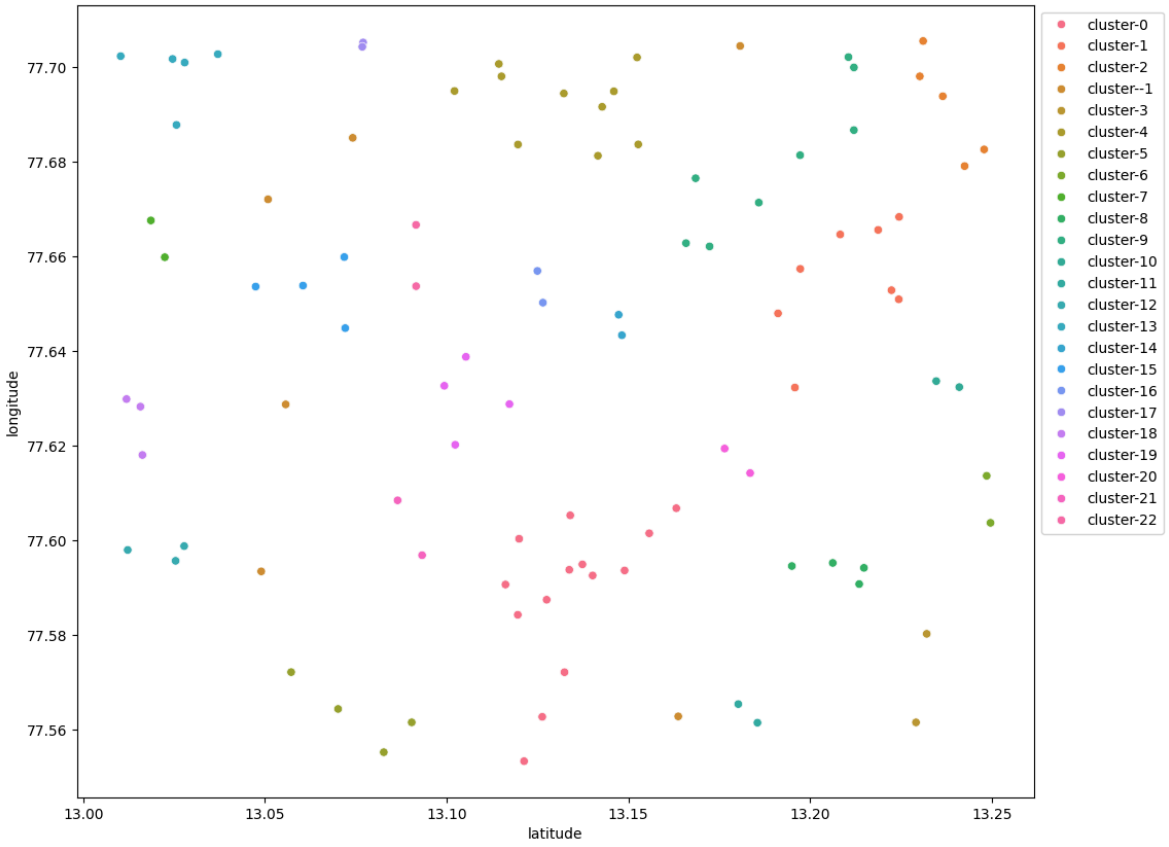

In [13]:
```
epsilon = 0.0158288
model = DBSCAN(eps = epsilon, min_samples =2 , metric = "haversine").fit(df[['la
df['cluster'] = model.labels_.tolist()
```

As we can see that we have choosen to check for every two people whether they are 1.5-feet(Km) apart or not. Here we can observe from the below plot that people were

clustered according to the geographic places they have travelled who might got affected by COVID-19 since geographical space between them is less than 1.5Km.

In [14]:
```
labels = model.labels_
fig = plt.figure(figsize=(12,10))
sns.scatterplot(data = df, x='latitude', y='longitude', hue = ['cluster-{}'.form
plt.legend(bbox_to_anchor = [1, 1])
```

Out[14]: <matplotlib.legend.Legend at 0x163e21c8710>




1)Creates and fits a DBSCAN model with:

- eps=epsilon: Maximum distance between points to be considered neighbors
- min_samples=2: Minimum points needed to form a cluster
- metric='haversine': Uses haversine distance for spherical coordinates (appropriate for GPS coordinates)

2)Adds cluster labels to the DataFrame. Each point gets assigned a cluster number (-1 means noise/no cluster).

3)Collects all other IDs that are in the same clusters as input_name, excluding noise points (cluster=-1).

4)Finally it returns the persons list and location of the persons to display on the app that is made for this

In [15]:
```
def get_infected_names_and_locations(input_name, df, epsilon=0.0158288, min_samp
    X = df[['latitude', 'longitude']].values
    model = DBSCAN(eps=epsilon, min_samples=min_samples, metric='haversine').fit
    df['cluster'] = model.labels_
```

```
    metrics = calculate_cluster_metrics(X, model.labels_)

    input_name_clusters = []
    infected_data = {}

    for i in range(len(df)):
        if df['id'].iloc[i] == input_name:
            if df['cluster'].iloc[i] not in input_name_clusters:
                input_name_clusters.append(df['cluster'].iloc[i])

    for cluster in input_name_clusters:
        if cluster != -1:
            cluster_data = df[df['cluster'] == cluster]
            for _, row in cluster_data.iterrows():
                if row['id'] != input_name:
                    if row['id'] not in infected_data:
                        infected_data[row['id']] = {
                            'id': row['id'],
                            'latitude': row['latitude'],
                            'longitude': row['longitude'],
                            'cluster': cluster
                        }

    user_location = df[df['id'] == input_name][['latitude', 'longitude']].iloc[0
    infected_list = list(infected_data.values())

    return infected_list, user_location, metrics
```

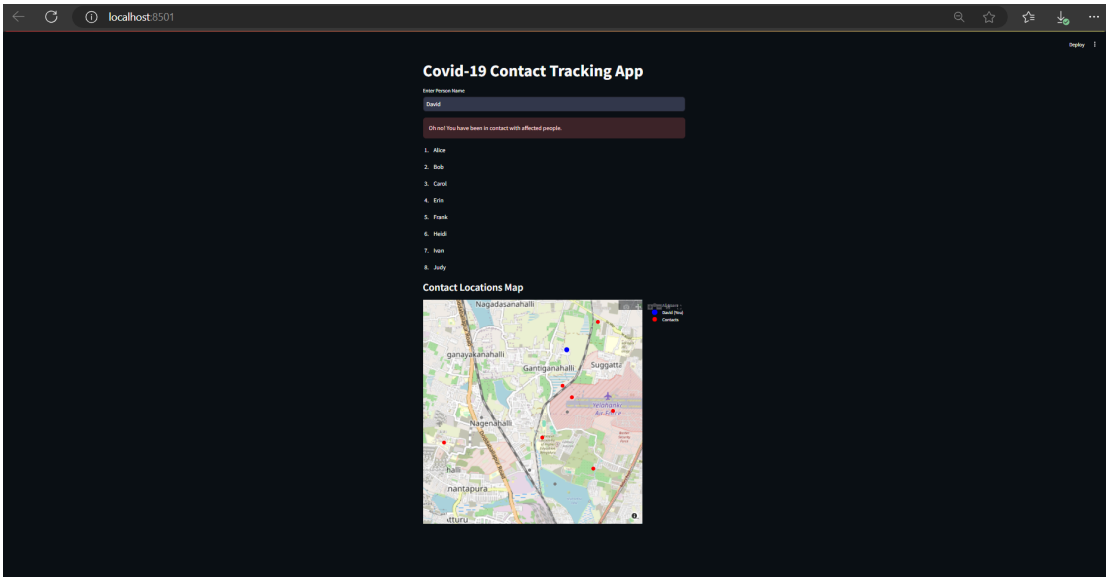
In [16]:
```
def calculate_cluster_metrics(X, labels):
    metrics = {}

    non_noise_mask = labels != -1
    X_clean = X[non_noise_mask]
    labels_clean = labels[non_noise_mask]

    if len(np.unique(labels_clean)) >= 2:
        metrics['silhouette_score'] = silhouette_score(X_clean, labels_clean, me
        metrics['calinski_harabasz_score'] = calinski_harabasz_score(X_clean, la
        metrics['davies_bouldin_score'] = davies_bouldin_score(X_clean, labels_c

    metrics['n_clusters'] = len(set(labels)) - (1 if -1 in labels else 0)
    metrics['n_noise_points'] = list(labels).count(-1)
    metrics['noise_ratio'] = metrics['n_noise_points'] / len(labels)

    return metrics
```

In [17]:
```
def load_data():
    try:
        return pd.read_json('livedata.json')
    except Exception as e:
        print(f"Error loading dataset: {e}")
```

In [18]:
```
df = load_data()
get_infected_names_and_locations('David',df, epsilon=0.0158288, min_samples=2)
```

```
Out[18]: ([{'id': 'Judy', 'latitude': 13.126296, 'longitude': 77.5627483, 'cluster': 0},
           {'id': 'Ivan',
            'latitude': 13.1401623,
            'longitude': 77.5925943,
            'cluster': 0},
           {'id': 'Alice',
            'latitude': 13.1557417,
            'longitude': 77.601504,
            'cluster': 0},
           {'id': 'Carol',
            'latitude': 13.1275096,
            'longitude': 77.5874724,
            'cluster': 0},
           {'id': 'Erin',
            'latitude': 13.1373459,
            'longitude': 77.5949414,
            'cluster': 0},
           {'id': 'Bob', 'latitude': 13.1199324, 'longitude': 77.6003416, 'cluster': 0},
           {'id': 'Frank',
            'latitude': 13.1339981,
            'longitude': 77.6052929,
            'cluster': 0},
           {'id': 'Heidi',
            'latitude': 13.1972994,
            'longitude': 77.6573416,
            'cluster': 1}],
          latitude     13.148953
          longitude    77.593651
          Name: 0, dtype: float64,
          {'silhouette_score': 0.43272153791293,
           'calinski_harabasz_score': 101.42869681543887,
           'davies_bouldin_score': 0.5450959971481147,
           'n_clusters': 23,
           'n_noise_points': 6,
           'noise_ratio': 0.06})
```

The below function takes the input from the above function and plot it in real time geographical map using plotly library, where other people are represented by "red" and person checking is represented by "blue" as shown below:



In [19]:

```
def visualize_contacts(input_name, infected_data, user_location):
    fig = go.Figure()

    fig.add_trace(go.Scattermapbox(
        lat=df['latitude'],
        lon=df['longitude'],
        mode='markers',
        marker=dict(size=10, color='grey'),
        name='All Users',
        hoverinfo='none'
    ))

    fig.add_trace(go.Scattermapbox(
        lat=[user_location['latitude']],
        lon=[user_location['longitude']],
        mode='markers',
        marker=dict(size=15, color='blue'),
        name=f'{input_name} (You)',
        text=[input_name]
    ))

    if infected_data:
        infected_lats = [p['latitude'] for p in infected_data]
        infected_lons = [p['longitude'] for p in infected_data]
        infected_names = [p['id'] for p in infected_data]

        fig.add_trace(go.Scattermapbox(
            lat=infected_lats,
            lon=infected_lons,
            mode='markers',
            marker=dict(size=12, color='red'),
            name='Contacts',
            text=infected_names
        ))

    fig.update_layout(
        mapbox=dict(
            style="open-street-map",
            center=dict(lat=user_location['latitude'], lon=user_location['longit
            zoom=13
        ),
        margin=dict(l=0, r=0, t=0, b=0),
        showlegend=True,
        height=600
    )

    return fig
```