Technical Report (in progress)

Salsabil Arabi
PI: Professor Tanzima Hashem

Bangladesh University of Engineering and Technology— January 7, 2021

Motivation

Digital contact tracing has a great role to play in learning epidemiology. With that motivation, I decided to work on predicting a person's future meetup information using historical check-in data. So whenever a person is infected, we can infer possible suspects who might come within his contact within a certain period. Many research have been done for next location predicting using trajectory data. Predicting next social event is also quite explored. However, finding the next friends gathering location and time of an specific person is still quite unexplored. The work is still in progress.

1 Problem definition

Given the user-specific check-in data, $C =< c1, c2, c3...cn >$ of a person p and the friendship connection $f =< f1, f2, f3... >$ of p, this work is intended to predict the next gathering location, $Ln+1$, time $Tn+1$ and companion of p.

2 Dataset

To validate the proposed approach, I am using Foursquare global scale check-in dataset with user social networks. Each entry of the check-in data file contains the following info: User ID, Venue ID, UTC time, Timezone offset. The friendship network file contains the friendship information of the users. Each entry has two user id columns indicating the friendship between the two users.

3 Methodology

We can devide the problem into two parts. First, we need to find the friend co-location/meetup events of each user from random check-in data. Second, we need to perform predictive analysis. The events of step one will be the input of the model. I will discuss each step in the subsequent sections.

3.1 Finding friend meetup events

To find the friend meetup events, we need to run queries on three different files. To cope with the large size of check-in data file and varied sparsity of locations, I decided to divide the dataset into parts depending on country. To do so, I performed a merge on check-in event file and point of interest data file to add the corresponding country of each check-in location and divided the check-in data file of each country separately. Then, I performed inner join on two similar check-in data file to find pair of users who were in the same POI. I processed temporal query to extract the event when the both users were at the same POI within a close time. Finally, I merged the friendship network to extract the entries where two co-located users were friends. For this project, I have considered these entries as friend meetup events. The SQL queries were processed using AWS S3 and Athena. Some of the queries were also conducted on pandas dataframe in the local machine. The code snippets are added in the section 4.

3.2 Predictive analysis

In this step, I am basically trying to predict the next meetup location. In order to finalize solution approach, I initially preprocessed the dataset. I conducted exploratory analysis on some samples to see if there lies any pattern in each users meetup pattern.

3.2.1 Exploratpry Analysis

I conducted exploratory analysis on some samples to see if there lies any pattern in each users meetup pattern. Also, it will help me to extract features and reduce dimensions.
Initially I cleaned the dataset and checked if there lies any null or missing value. There were just a few missing value issues in the dataset. I also checked the mean, median, standard deviation and other percentile values of the dataset to know if there is any extreme values or outliers in the dataset. The values are shown in table 1 The values indicate that there are some issues in the longitude values.

Table 1: Statistics of the dataset

metric	userid1	time1	userid2	time2	latitude	longitude
count	3.811400e+04	3.811400e+04	3.811400e+04	3.811400e+04	38114.00	38114.00
mean	5.294193e+05	1.057909e+09	5.294193e+05	1.057909e+09	37.0510	-93.7475
std	5.723931e+05	2.127748e+05	5.723931e+05	2.127748e+05	5.9929	21.0138
min	1.900000e+01	1.057685e+09	1.900000e+01	1.057685e+09	18.4128	-159.523
25%	1.160190e+05	1.057740e+09	1.160190e+05	1.057740e+09	33.7891	-104.8902
50%	2.601610e+05	1.057816e+09	2.601610e+05	1.057816e+09	38.9167	-87.6217
75%	8.148350e+05	1.058056e+09	8.148350e+05	1.058056e+09	41.1938	-78.7605
max	2.181131e+06	1.058643e+09	2.181131e+06	1.058643e+09	63.7383	-66.2539

After plotting the events of some samples, I observed that the meetup patterns of some individual users have some recurrency. Two such users event patterns are shown in figure 1 and figure 2. So, I decided to use Long Short Term Memory (LSTM) Recurrent Neural Network model for the prediction of location. In order to observe the correlation among variables, I used pandas .corr() function and observed

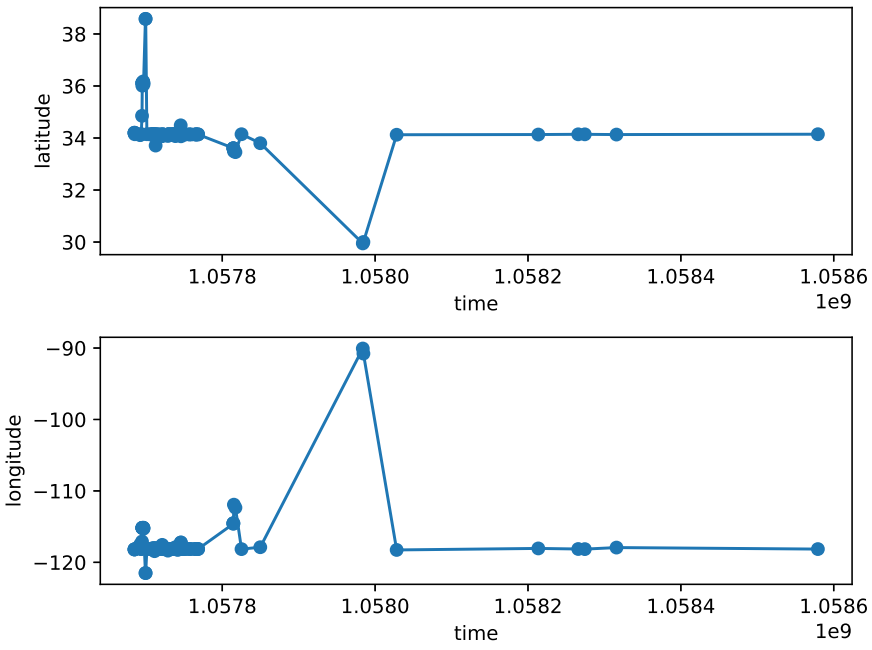

Figure 1: meetup events of user with id 5526

the correlation matrix using a heatmap in seaborn library. The figure 3, shows the correlation among the

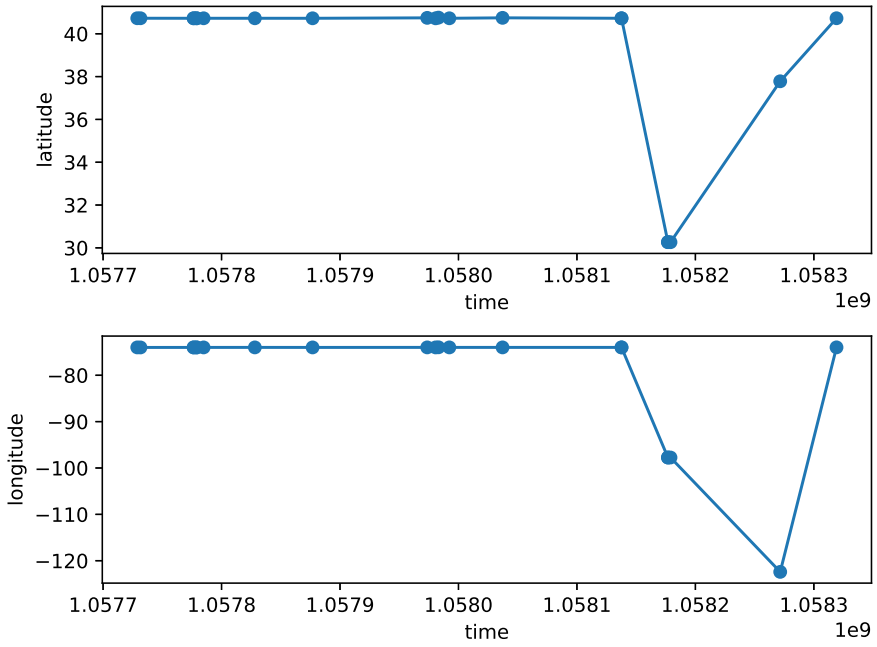

Figure 2: meetup events of user with id 19

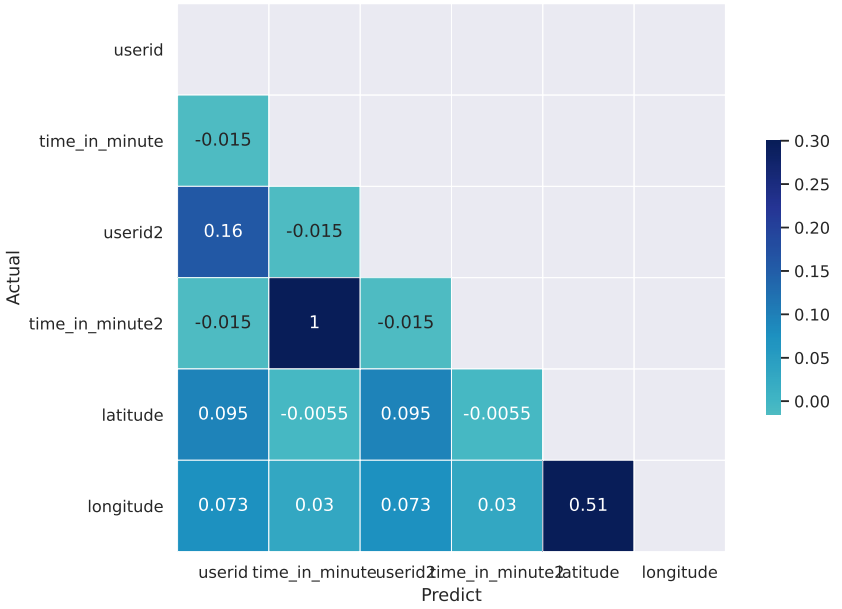

Figure 3: correlation matrix among features

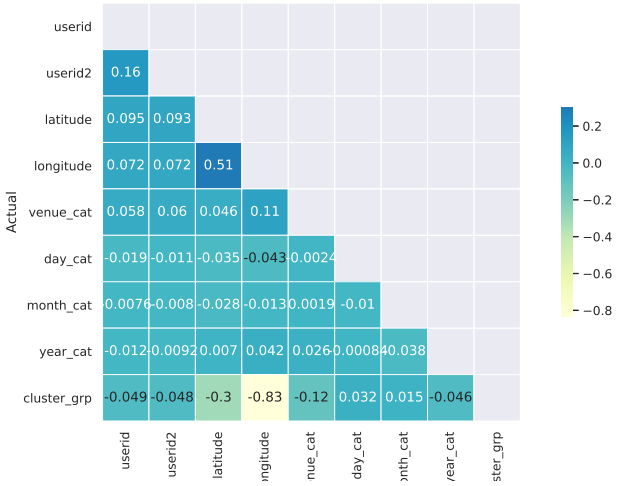

Figure 4: correlation matrix after removing closely connected features and adding new relevant features

features. The darker blocks indicate strong correlation among those features. The darker shades represent positive correlation and the lighter shades represent negative correlation. As we are not applying linear regression model, we do not need to bother much regarding correlation among features. However, while extracting features, we can choose to remove correlated variables to reduce dimension. The figure 4 shows the correlation matrix after removing the strongly correlated features.

3.2.2 Clustering

I used meanshift clustering algorithm of scikit-learn library in order to divide the input check-in locations into clusters. At the second output layer, I will get the probabilities of being in each cluster. To get the actual longitude and latitude, I have multiplied the probabilities with the cluster centers at the final output layer. In figure 5, the green locations are cluster centers whereas the purple dots are actual locations.

3.2.3 Preprocess Dataset

Before feeding the model, I needed to process the input dataset. I converted the feature values into categorical values using labelencoder function from scikit learn library. Also, I removed some correlated features to reduce feature dimension. The final set of feature is shown in table 2. Other than latitude and longitude, all the other features are converted into categorical values.

Table 2: Extracted Features

latitude	longitude	date	hour	holiday	venue	day_type	month	year	venue_type	cluster
float	float	int	int	int	int	int	int	int	int	float

3.2.4 Building the model

I have used the LSTM model from keras library to perform predictive analysis. The model is comprosed of an input layer, a hidden layer of 100 neurons and RELU activation function, a softmax layer for finding probability of each cluster and an output layer. The softmax activation layer is followed by the linear layer which calculates the actual longitude latitude from the probabilities of softmax layer. Also, I have used haversine distance as the loss function to calculate the shortest distance between two points on earth's surface.

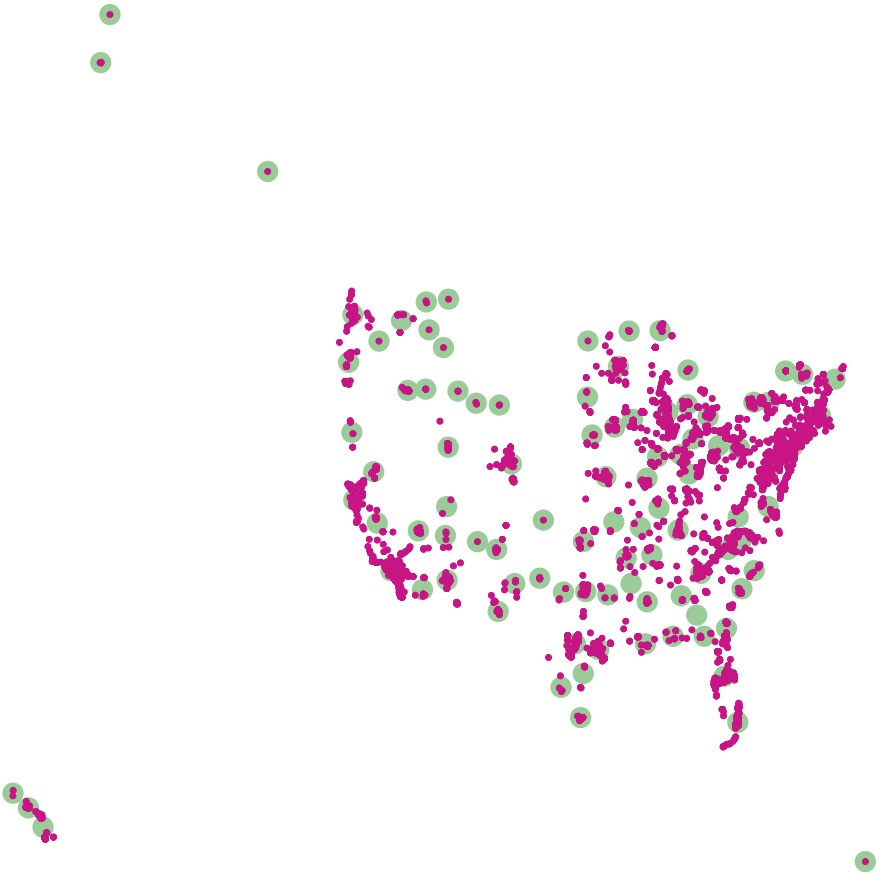

Figure 5: Clustering of US based events when quantile 0.02. Green dots resemble cluster centers and Purple dots resemble actual location of user

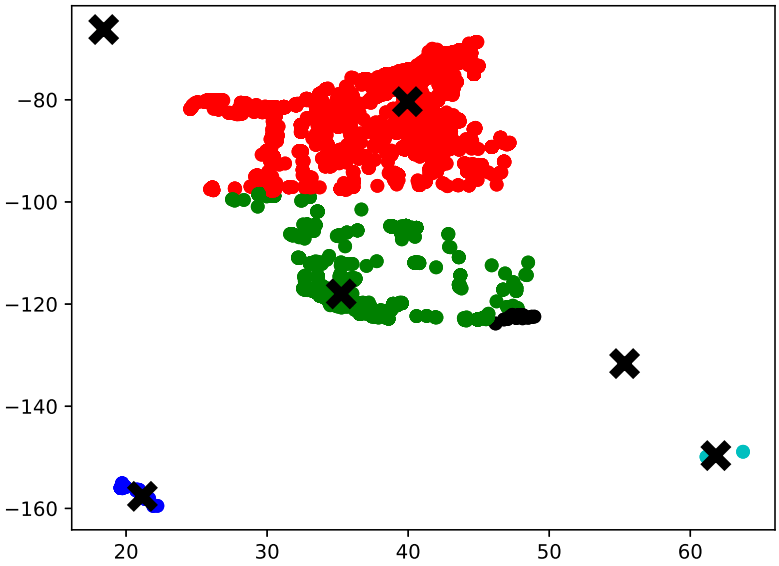

Figure 6: Clustering of US based events when quantile 0.2

The LSTM architecture used in our experiments is given by the following equations:

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$$

$$\tilde{c_t} = \tanh(W_c[h_{t-1}, x_t] + b_c)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c_t}$$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t \odot \tanh(c_t)$$

3.2.5 Training the model

I have trained the model over different number of epochs. Using 10 epochs and 0.2 quantile value in clustering algorithm during training, from a total of 2780 test data, the model accurately identified the cluster groups of 2123 data. I still need to perform other optimization to get better accuracy of clusters and locations. To calculate the loss of the model, I am using haversine distance between the actual and predicted location. However, currently the loss is not satisfactory. The model still needs much more optimization.

```
tf.Tensor([[ 38.761288 -92.80686 ]], shape=(1, 2), dtype=float32) long  40.096253000000004 lat  -88.235879
tf.Tensor([[ 38.761288 -92.80686 ]], shape=(1, 2), dtype=float32) long  39.914274 lat  -86.104488
tf.Tensor([[ 38.761288 -92.80686 ]], shape=(1, 2), dtype=float32) long  32.498435 lat  -92.060069
tf.Tensor([[ 38.761288 -92.80686 ]], shape=(1, 2), dtype=float32) long  33.785740000000004 lat  -84.384351
tf.Tensor([[ 38.761288 -92.80686 ]], shape=(1, 2), dtype=float32) long  36.088964000000004 lat  -115.177982
tf.Tensor([[ 38.760616 -92.841156]], shape=(1, 2), dtype=float32) long  42.35947 lat  -71.053726
tf.Tensor([[ 38.761288 -92.80704 ]], shape=(1, 2), dtype=float32) long  45.141501 lat  -93.23420300000001
tf.Tensor([[ 38.761288 -92.80686 ]], shape=(1, 2), dtype=float32) long  32.507384 lat  -84.963313
tf.Tensor([[ 38.761288 -92.80686 ]], shape=(1, 2), dtype=float32) long  42.351544 lat  -71.064875
tf.Tensor([[ 38.761288 -92.80686 ]], shape=(1, 2), dtype=float32) long  45.534715999999996 lat  -122.65554499999999
tf.Tensor([[ 38.761288 -92.80686 ]], shape=(1, 2), dtype=float32) long  39.958562 lat  -75.136658
tf.Tensor([[ 38.761288 -92.80686 ]], shape=(1, 2), dtype=float32) long  35.227677 lat  -80.839084
total  2780  correct  2123
```

Figure 7: Snippet of test data prediction

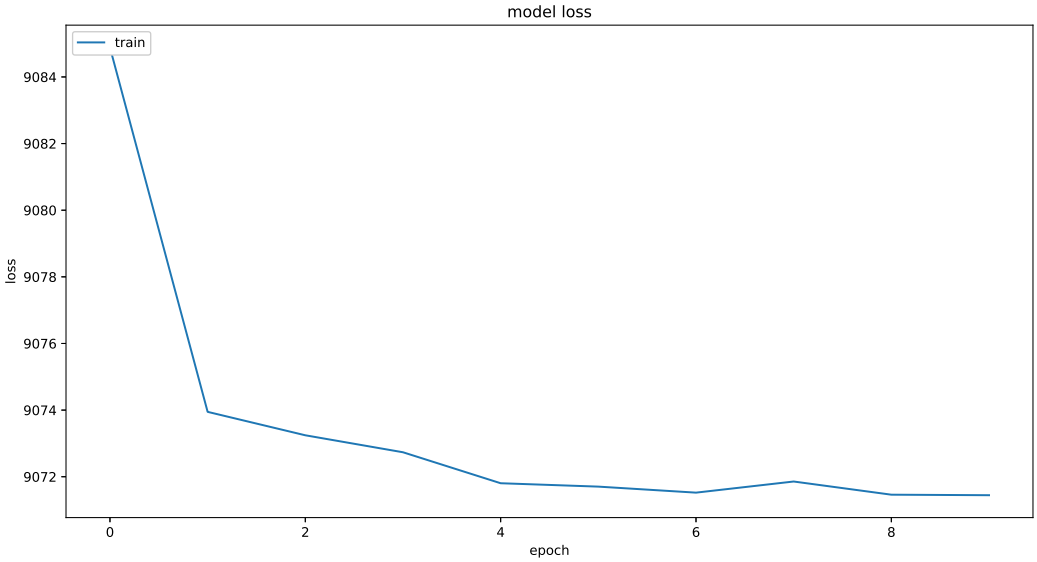

Figure 8: training loss with epoch 5

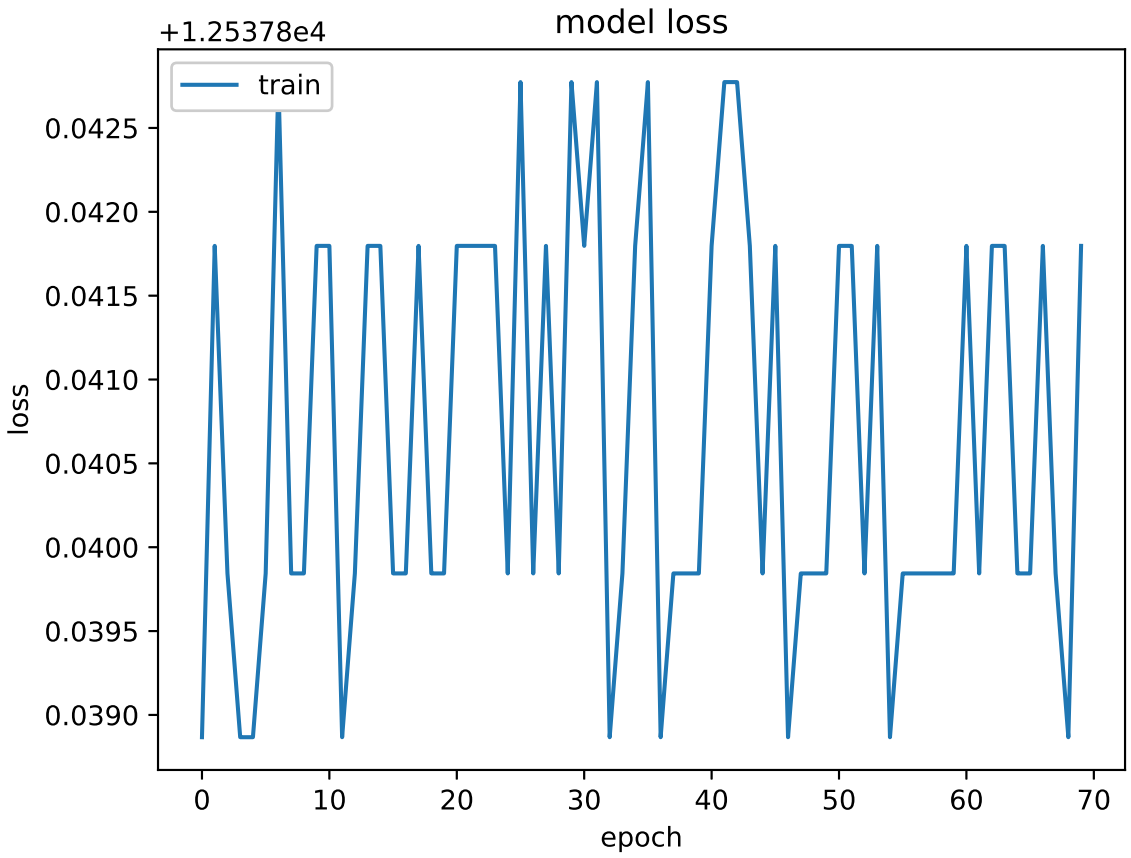

Figure 9: training loss with epoch 70 (not converged yet)

4 Appendix

```
import pandas as pd
#merging dataset on venueid to find the check-in at the same place
result = pd.merge(data1,data2,how='inner',on='venueId')
# excluding self merged events
result = result[result.userId != result.userId2]
# Keeping only those event check-in time difference is less than 120 min
result = result[result.time_diff <= 120]
```

Listing 1: finding co-located spatiotemporal events

```
is_friend = []
data = pd.read_csv("friendship2.csv")
with open('test_US.csv', 'r') as file:
        i = -1
        for line in file:
            i += 1
            if i == 0:
                continue
            p = 0
            for cnt,val in enumerate(line.split(',')):
                p += 1
                if p == 1:
                    id1 = int(val)
                elif p == 5:
                    id2 = int(val)
                    data1 = data[(data.userid1 == id1) & (data.userid2
    == id2)]
                    data2 = data[(data.userid1 == id2) & (data.userid2
    == id1)]
                    if len(data1.index) == 0 and len(data2.index) == 0:
                        is_friend.append("NO")
                    else:
                        is_friend.append("YES")

                else:
                    continue

data = pd.read_csv("test_US.csv")
data['is_friend'] = is_friend
data = data[data.is_friend == "YES"]
```

Listing 2: finding spatiotemporal co-located friend gathering event