Problem statement

For the top 10 users (with more checkins) build: (Refer to section 4)

- A basket of recommendation: venues (places)
- A list of likely venues (places) the user will visit based on their friends.
- Where they will go next (with probability scores).

For the top 10 more “social” users (with more friends): (Refer to section 5)

- Draw the path (with map) of a week/month of users checkins.
- List your friends and how close they are in terms of “taste” (based on venues visited and ranked)

In [1]:
```
import numpy as np
import pandas as pd
import sqlite3
import implicit
```

Open database connection

In [2]:
```
conn = sqlite3.connect('fsdata.db')
print("Database opened successfully")
```

```
Database opened successfully
```

1. Perform data munging

Noticed in social graph there are users which are not registered in users table, and not having bi-direction relationship. Could be due to no foreign key constraint enabled on that SQLite table.

Following is to clean up these records.

Clean socialgraph table from invalid user id, to contain only bi-direction network

In [3]:
```
sql = "DELETE FROM socialgraph WHERE first_user_id not in (select id from users)"
sql += " or second_user_id not in (select id from users)"
conn.execute(sql)
conn.commit()
print("Record on socialgraph which doesn't exists in users table deleted successfully")
```

```
Record on socialgraph which doesn't exists in users table deleted successfully
```

In [4]:
```
sql = "DELETE FROM socialgraph WHERE first_user_id not in (select second_user_id from socialgraph)"
sql += " or second_user_id not in (select first_user_id from socialgraph)"
conn.execute(sql)
conn.commit()
print("Record on socialgraph which doesn't have bi-direction reference first & second user_id deleted successfully")
```

```
Record on socialgraph which doesn't have bi-direction reference first & second user_id deleted successfully
```

Clean ratings and checkins tables from invalid user_id and invalid venue_id

In [5]:
```
sql = "DELETE FROM ratings WHERE user_id not in (select id from users)"
conn.execute(sql)
conn.commit()
print("Record on ratings which doesn't exists in users table deleted successfully")
```

```
Record on ratings which doesn't exists in users table deleted successfully
```

In [6]:
```
sql = "DELETE FROM ratings WHERE venue_id not in (select id from venues)"
conn.execute(sql)
conn.commit()
print("Record on ratings which doesn't exists in venues table deleted successfully")
```

```
Record on ratings which doesn't exists in venues table deleted successfully
```

In [7]:
```
sql = "DELETE FROM checkins WHERE user_id not in (select id from users)"
conn.execute(sql)
conn.commit()
print("Record on checkins which doesn't exists in users table deleted su
ccessfully")
```

```
Record on checkins which doesn't exists in users table deleted successf
ully
```

In [8]:
```
sql = "DELETE FROM checkins WHERE venue_id not in (select id from venue
s)"
conn.execute(sql)
conn.commit()
print("Record on checkins which doesn't exists in venues table deleted s
uccessfully")
```

```
Record on checkins which doesn't exists in venues table deleted success
fully
```

We need to know if checkins without geo data are an abuse to the system. If yes, exclude them. If not, can use geo data from Venues

In [9]:
```
#sql = "DELETE FROM checkins WHERE longitude=0 and latitude=0"
#conn.execute(sql)
#conn.commit()
#print("Record on checkins with invalid geodata deleted successfully")
```

2. Load Data

Load Users

In [10]:
```
users = pd.read_sql_query("select * from users", conn)
users.head(5)
```

Out[10]:

	id	latitude	longitude
0	1	45.072464	-93.455788
1	2	30.669682	-81.462592
2	3	43.549975	-96.700327
3	4	44.840798	-93.298280
4	5	27.949436	-82.465144

In [11]:
```
users.shape
```

Out[11]:
```
(2153469, 3)
```

Load Users with hometown geo data

In [12]:
```
users_hometown = pd.read_sql_query("select * from users where longitude<
>0 and latitude <> 0", conn)
users_hometown.head(5)
```

Out[12]:

	id	latitude	longitude
0	1	45.072464	-93.455788
1	2	30.669682	-81.462592
2	3	43.549975	-96.700327
3	4	44.840798	-93.298280
4	5	27.949436	-82.465144

In [13]:
```
users.shape
```

Out[13]: (2153469, 3)

Load Venues

In [14]:
```
venues = pd.read_sql_query("select * from venues", conn)
venues.head(5)
```

Out[14]:

	id	latitude	longitude
0	1	44.882011	-93.212364
1	2	44.883169	-93.213687
2	3	44.883455	-93.214316
3	4	44.881387	-93.213801
4	5	44.882129	-93.214012

In [15]:
```
venues.shape
```

Out[15]: (1143090, 3)

Load Ratings

In [16]:
```
ratings = pd.read_sql_query("select * from ratings", conn)
ratings.head(5)
```

Out[16]:

	user_id	venue_id	rating
0	1	1	5
1	1	51	4
2	1	51	2
3	1	51	5
4	1	52	5

In [17]:
```
ratings.shape
```

Out[17]: (2809580, 3)

Load Checkins

In [18]:
```
checkins = pd.read_sql_query("select * from checkins", conn)
checkins.head(5)
```

Out[18]:

	id	user_id	venue_id	latitude	longitude	created_at
0	16	539270	1206	41.878114	-87.629798	2011-12-08 05:08:42
1	17	1330941	1206	0.000000	0.000000	2011-12-08 04:32:19
2	18	1330942	1206	0.000000	0.000000	2011-12-08 04:29:38
3	19	282798	1206	41.878114	-87.629798	2011-12-08 04:26:06
4	20	376793	1206	41.878114	-87.629798	2011-12-08 04:17:50

In [19]:
```
checkins.shape
```

Out[19]: (1021966, 6)

Load SocialGraph

In [20]:
```
socialgraph = pd.read_sql_query("select * from socialgraph", conn)
socialgraph.head(5)
```

Out[20]:

	first_user_id	second_user_id
0	1	10
1	10	1
2	1	11
3	11	1
4	1	12

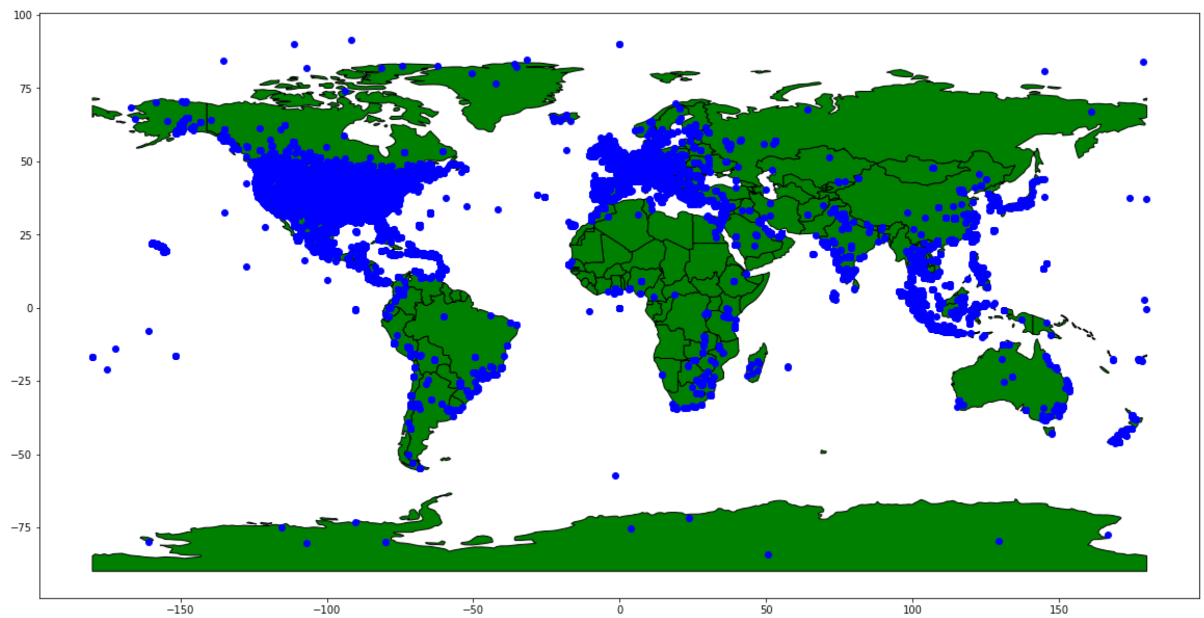
In [21]:
```
socialgraph.shape
```

Out[21]:
```
(27098469, 2)
```

3. Exploratory Data Analysis

To understand and get insight from available data provided.

In [22]:
```
#### Plot all venues on world map

import geopandas as gpd
import geoplot as gplt
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings('ignore')

gdf = gpd.GeoDataFrame(venues, geometry=gpd.points_from_xy(venues.longit
ude, venues.latitude))

world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
ax = world.plot(figsize=(20,15),color='green', edgecolor='black')
gdf.plot(ax=ax, color='blue')

plt.show()
```



For each venues, how many users checkins?

In [23]:
```
sql = "select venue_id, count(*) as checkins from checkins group by venu
e_id order by count(*) desc"
venue_checkins = pd.read_sql_query(sql, conn)
venue_checkins
```

Out[23]:

	venue_id	checkins
0	5222	25366
1	7620	23622
2	2297	23415
3	11195	19463
4	11138	18088
...	...	...
83994	87	1
83995	79	1
83996	76	1
83997	4	1
83998	3	1

83999 rows × 2 columns

For each venues, how many times same user repeat checkins?

In [24]:
```
sql = "select venue_id, user_id, count(*) as x_times_checkin from checki
ns where longitude<>0 and latitude<>0 group by venue_id, user_id having
 count(*) > 1 order by count(*) desc"
venue_checkins_user_repeat = pd.read_sql_query(sql, conn)
venue_checkins_user_repeat
```

Out[24]:

	venue_id	user_id	x_times_checkin
0	11138	660409	41
1	64	517520	39
2	4432	304865	39
3	4432	439413	36
4	5222	8622	34
...	...	...	...
36009	1135586	386648	2
36010	1135757	263733	2
36011	1136241	318620	2
36012	1138491	863866	2
36013	1140238	356927	2

36014 rows × 3 columns

Is there same venue rated multiple times by same user?

In [25]:
```
sql = "select venue_id, user_id, count(*) as x_times_rating from ratings group by venue_id, user_id having count(*)>1 order by count(*) desc"
venue_rated_user_repeat = pd.read_sql_query(sql, conn)
venue_rated_user_repeat
```

Out[25]:

	venue_id	user_id	x_times_rating
0	407336	1001224	137
1	111995	107501	97
2	5732	2639	73
3	205294	67682	70
4	957067	1030348	68
...	...	...	...
298330	1142950	697097	2
298331	1142972	292419	2
298332	1142996	1016769	2
298333	1143024	401916	2
298334	1143072	401916	2

298335 rows × 3 columns

For each venues, what is each user's rating?

(Since same user can give multiple rating to same venue, average it)

In [26]:
```
sql = "select venue_id, user_id, avg(rating) as rating from ratings grou
p by venue_id, user_id order by avg(rating) desc"
venue_rating_by_user = pd.read_sql_query(sql, conn)
venue_rating_by_user
```

Out[26]:

	venue_id	user_id	rating
0	1	1	5.0
1	2	2	5.0
2	3	3	5.0
3	4	4	5.0
4	5	5	5.0
...	...	...	...
2436718	1142965	797976	2.0
2436719	1142966	1398671	2.0
2436720	1143011	83334	2.0
2436721	1143011	1626801	2.0
2436722	1143020	1407243	2.0

2436723 rows × 3 columns

For each venues what is universal rating?

(Use median as venue rating from multiple users)

In [27]:
```
venue_rating_universal = venue_rating_by_user.groupby('venue_id')['ratin
g'].mean().sort_values(ascending=False)
venue_rating_universal.sample(10)
```

Out[27]:
```
venue_id
8770       3.333333
856763     5.000000
901018     5.000000
544610     5.000000
230294     5.000000
817194     5.000000
372174     5.000000
66033      2.750000
674726     4.333333
1118935    5.000000
Name: rating, dtype: float64
```

Which are the top 10 users has more checkins?

In [28]:
```
sql = "select user_id, count(*) as x_times_checkin from checkins group b
y user_id order by count(*) desc"
users_checkins = pd.read_sql_query(sql, conn)
users_top_10_checkins = users_checkins.head(10).copy()
users_top_10_checkins
```

Out[28]:

	user_id	x_times_checkin
0	1348362	57
1	1900906	52
2	1326476	48
3	1365850	47
4	386648	47
5	467043	46
6	651415	45
7	439413	45
8	304865	45
9	8622	45

Which are the top 10 users has more friends?

In [29]:
```
sql = "select first_user_id, count(*) as x_friends from socialgraph grou
p by first_user_id order by count(*) desc"
users_social = pd.read_sql_query(sql, conn)
users_top_10_social = users_social.head(10).copy()
users_top_10_social
```

Out[29]:

	first_user_id	x_friends
0	754	107676
1	56	105091
2	4489	94991
3	50	82387
4	512	78277
5	59	73760
6	1334	70181
7	52	67931
8	47	67505
9	40	67093

4. Challenges for top 10 users with more checkins, build:

4.1. Problem: A basket of venue recommendations

Create global (universal) rating venues and use as base of recommendation. (see variable: ranking)

Excludes:

- Not fulfilling minimum number of reviews
- Places he ever visit before

Ranked by:

1. Highest rating
2. Highest number of people giving rating
3. Nearest house & places he has ever check in (not yet being used. data is not clean)
4. Most number of checkins

In [30]:
```
# Print top 10 users by checkins
users_top_10_checkins
```

Out[30]:

	user_id	x_times_checkin
0	1348362	57
1	1900906	52
2	1326476	48
3	1365850	47
4	386648	47
5	467043	46
6	651415	45
7	439413	45
8	304865	45
9	8622	45

Find out ratings

In [31]:
```
venue_rating = venue_rating_universal.to_frame().copy()
venue_rating
```

Out[31]:

	rating
venue_id	
570409	5.000000
591469	5.000000
591482	5.000000
161606	5.000000
161615	5.000000
...	...
352717	2.003030
758013	2.002933
1132898	2.002920
1110864	2.002740
306516	2.001883

1140494 rows × 1 columns

Find out number of reviewed

In [32]:
```
# Need to ask how data collected, to know if same user rating same venue multiple times is an abuse?
venue_times_reviewed = ratings.copy() # If want to avoid abuse, use: "venue_rating_by_user" variable instead of ratings
venue_times_reviewed['reviewed'] = 1
venue_times_reviewed = venue_times_reviewed[['venue_id','reviewed']].groupby(['venue_id']).count()
venue_times_reviewed
```

Out[32]:

	reviewed
venue_id	
1	20
2	14
3	6
4	2
5	1
...	...
1143044	1
1143045	1
1143046	1
1143072	2
1143073	1

1140494 rows × 1 columns

Find out number of checkins

In [33]:
```
venue_most_checkins = checkins.copy()
venue_most_checkins['checkins'] = 1
venue_most_checkins = venue_most_checkins[['venue_id','checkins']].group
by(['venue_id']).count()
venue_most_checkins
```

Out[33]:

	checkins
venue_id	
1	4
3	1
4	1
51	13
60	17777
...	...
1142955	3
1142965	1
1142966	1
1143011	2
1143020	1

83999 rows × 1 columns

Decide minimal # of reviews

In [34]:
```
venue_times_reviewed['reviewed'].median()
```

Out[34]: 1.0

In [35]:
```
venue_times_reviewed['reviewed'].mean()
```

Out[35]: 2.46347635322939

In [36]:
```
# As apparently median number of ratings on venues is 1 and better not t
o rely to single rating,
# we choose 2 as minimum number of review. In real life data, this shoul
d be higher like minimal 10 reviews needed.
min_num_of_reviews = 2
```

Finalise ranking - taking considerations: rating + number of reviews + checkins

In [37]:
```
ranking = pd.concat([venue_rating, venue_times_reviewed, venue_most_chec
kins], axis=1)
ranking['checkins'] = ranking['checkins'].fillna(0).astype(int)
ranking.shape
```

Out[37]:
```
(1140494, 3)
```

In [38]:
```
# Show to 20 Venues with highest ratings, number of reviews, number of c
heckins
ranking = ranking[ranking['reviewed']>=min_num_of_reviews].sort_values([
'rating','reviewed'], ascending=False)
ranking.head(20)
```

Out[38]:

venue_id	rating	reviewed	checkins
101862	4.5	10	0
463173	4.5	9	0
632158	4.5	9	0
570948	4.5	8	0
683881	4.5	8	0
52200	4.5	7	0
115267	4.5	7	0
632162	4.5	7	0
290211	4.5	6	0
360159	4.5	6	0
403000	4.5	6	0
567372	4.5	6	0
592272	4.5	6	0
716039	4.5	6	0
768431	4.5	6	0
917671	4.5	6	0
3614	4.5	5	0
52845	4.5	5	0
67632	4.5	5	0
71542	4.5	5	0

4.1 Answer for top 10 users with more checkins, a basket of venue recommendations:

- With Content Based Filtering

In [39]:
```
# Loop all top 10 users with more checkins
for index, row in users_top_10_checkins.iterrows():
    user_id = row[0]
    print("-------------------------------------")
    print('User ID:', user_id)
    print('Venue ID Recommentations:')

    # Print top 5 highest rank venues this user never check in befores
    print(ranking[~ranking.index.isin(checkins[checkins['user_id']==user
_id])].head(5).round(1))
```

```
------------------------------------
User ID: 1348362
Venue ID Recommentations:
          rating  reviewed  checkins
venue_id
101862       4.5        10         0
463173       4.5         9         0
632158       4.5         9         0
570948       4.5         8         0
683881       4.5         8         0
------------------------------------
User ID: 1900906
Venue ID Recommentations:
          rating  reviewed  checkins
venue_id
101862       4.5        10         0
463173       4.5         9         0
632158       4.5         9         0
570948       4.5         8         0
683881       4.5         8         0
------------------------------------
User ID: 1326476
Venue ID Recommentations:
          rating  reviewed  checkins
venue_id
101862       4.5        10         0
463173       4.5         9         0
632158       4.5         9         0
570948       4.5         8         0
683881       4.5         8         0
------------------------------------
User ID: 1365850
Venue ID Recommentations:
          rating  reviewed  checkins
venue_id
101862       4.5        10         0
463173       4.5         9         0
632158       4.5         9         0
570948       4.5         8         0
683881       4.5         8         0
------------------------------------
User ID: 386648
Venue ID Recommentations:
          rating  reviewed  checkins
venue_id
101862       4.5        10         0
463173       4.5         9         0
632158       4.5         9         0
570948       4.5         8         0
683881       4.5         8         0
------------------------------------
User ID: 467043
Venue ID Recommentations:
          rating  reviewed  checkins
venue_id
101862       4.5        10         0
463173       4.5         9         0
```

```
632158       4.5         9         0
570948       4.5         8         0
683881       4.5         8         0
------------------------------------
User ID: 651415
Venue ID Recommentations:
          rating  reviewed  checkins
venue_id
101862       4.5        10         0
463173       4.5         9         0
632158       4.5         9         0
570948       4.5         8         0
683881       4.5         8         0
------------------------------------
User ID: 439413
Venue ID Recommentations:
          rating  reviewed  checkins
venue_id
101862       4.5        10         0
463173       4.5         9         0
632158       4.5         9         0
570948       4.5         8         0
683881       4.5         8         0
------------------------------------
User ID: 304865
Venue ID Recommentations:
          rating  reviewed  checkins
venue_id
101862       4.5        10         0
463173       4.5         9         0
632158       4.5         9         0
570948       4.5         8         0
683881       4.5         8         0
------------------------------------
User ID: 8622
Venue ID Recommentations:
          rating  reviewed  checkins
venue_id
101862       4.5        10         0
463173       4.5         9         0
632158       4.5         9         0
570948       4.5         8         0
683881       4.5         8         0
```

- With collaboration filtering. eg. Alternate Least Square

In [40]:
```
import scipy.sparse as sparse
import implicit
import pandas as pd
```

Prepare 'collab_data' variable contains list of venues and ratings by each users

In [41]:
```
collab_data = venue_rating_by_user.copy()
collab_data
```

Out[41]:

	venue_id	user_id	rating
0	1	1	5.0
1	2	2	5.0
2	3	3	5.0
3	4	4	5.0
4	5	5	5.0
...	...	...	...
2436718	1142965	797976	2.0
2436719	1142966	1398671	2.0
2436720	1143011	83334	2.0
2436721	1143011	1626801	2.0
2436722	1143020	1407243	2.0

2436723 rows × 3 columns

In [42]:
```
sparse_venue_user = sparse.csr_matrix((collab_data['rating'].astype(floa
t), (collab_data['venue_id'], collab_data['user_id'])))
sparse_user_venue = sparse.csr_matrix((collab_data['rating'].astype(floa
t), (collab_data['user_id'], collab_data['venue_id'])))
```

In [43]:
```
model = implicit.als.AlternatingLeastSquares(factors=3, regularization=
0.1)
```

In [44]:
```
alpha_val = 40
data_conf = (sparse_venue_user * alpha_val).astype('double')
model.fit(data_conf)
```

Use ALS model to recommend one user_id as example from top 10 most checkins

In [45]:
```
user_id = 3
recommended = model.recommend(user_id, sparse_user_venue)
print(recommended)
```

```
[(7620, 0.20903055), (2297, 0.20314875), (29488, 0.18597801), (11138,
0.18367594), (64, 0.17045842), (9310, 0.16670139), (60, 0.1495703), (98
22, 0.14787117), (783, 0.14717545), (11492, 0.14549726)]
```

Recommended venue user id 3 based on latent factor on his rating is as this ranking:

In [46]:
```
rec = pd.DataFrame(recommended, columns=['venue_id', 'score'])
rec.sort_values(['score'], ascending=False)
```

Out[46]:

	venue_id	score
0	7620	0.209031
1	2297	0.203149
2	29488	0.185978
3	11138	0.183676
4	64	0.170458
5	9310	0.166701
6	60	0.149570
7	9822	0.147871
8	783	0.147175
9	11492	0.145497

4.2. Problem: A list of likely venues the user will visit based on their friends

Use previous global (universal) ranking. (see variable: ranking)

Only Include:

- Places his friends previously reviewed (give rating) or visited (checked-in)

Answer: For top 10 users with more checkins, a list of likely venues the user will visit based on their friends

In [47]:
```
for index, row in users_top_10_checkins.iterrows():
    user_id = row[0]
    print("------------------------------------")
    print('User ID:', user_id)

    # Get his friends
    friends = socialgraph[socialgraph['first_user_id']==user_id]['second
_user_id'].values

    if (len(friends)>0):
        # Get venues his friends rated before
        friend_ratings = ratings[ratings['user_id'].isin(friends)]['venu
e_id'].unique().tolist()

        # Get venues his friend checkin before
        friend_checkins = checkins[checkins['user_id'].isin(friends)]['v
enue_id'].unique().tolist()

        # Combine friend_ratings and friend_checkins as friend_venues
        friend_venues = friend_ratings + list(set(friend_checkins) - set
(friend_ratings))

        # Get ranking within friend_venues
        friend_recommendation = ranking[ranking.index.isin(friend_venues
)]

        # Print top 5 recommendation based onf
        print(friend_recommendation[:5].round(1))
    else:
        print('No Friend')
```

```
------------------------------------
User ID: 1348362
No Friend
------------------------------------
User ID: 1900906
No Friend
------------------------------------
User ID: 1326476
          rating  reviewed  checkins
venue_id
174555       4.2         3         0
26600        3.6         5         0
------------------------------------
User ID: 1365850
          rating  reviewed  checkins
venue_id
214953       4.5         2         0
642255       4.5         2         0
562          2.1      1925      1841
60           2.0     18061     17777
2297         2.0     23468     23415
------------------------------------
User ID: 386648
          rating  reviewed  checkins
venue_id
8622         4.5         2         0
24059        4.5         2         0
56729        4.5         2         0
57229        4.5         2         0
62033        4.5         2         0
------------------------------------
User ID: 467043
          rating  reviewed  checkins
venue_id
615282       4.5         3         0
126984       4.5         2         0
126986       4.5         2         0
126992       4.5         2         0
127001       4.5         2         0
------------------------------------
User ID: 651415
          rating  reviewed  checkins
venue_id
108931       4.5         2         0
176174       4.5         2         0
231798       4.5         2         0
486660       4.5         2         0
509328       4.5         2         0
------------------------------------
User ID: 439413
          rating  reviewed  checkins
venue_id
107600       4.5         2         0
119480       3.2         4         0
107602       2.8         7         4
107596       2.8         4         3
107603       2.8         4         3
```

```
------------------------------------
User ID: 304865
          rating  reviewed  checkins
venue_id
525749       4.5         2         0
525755       4.5         2         0
525757       4.5         2         0
525764       4.2         3         0
525759       3.5         2         1
------------------------------------
User ID: 8622
          rating  reviewed  checkins
venue_id
158014       4.5         3         0
368440       4.5         3         0
577519       4.5         3         0
679392       4.5         3         0
778753       4.5         3         0
```

4.3. Problem: Where they will go next (with probability scores)

1. Determine features such as day of week for each users
2. Assign venues as labels

In [48]:
```python
# Print top 10 users by checkins
users_top_10_checkins
```

Out[48]:

	user_id	x_times_checkin
0	1348362	57
1	1900906	52
2	1326476	48
3	1365850	47
4	386648	47
5	467043	46
6	651415	45
7	439413	45
8	304865	45
9	8622	45

Predict this user where they will go on certain day of week

In [49]:
```python
# Pick up one of user_id from above
user_id = 1348362
```

In [50]:
```
# Get user's checkin, drop unnecessary coluhttp://localhost:8888/noteboo
ks/ML-Engineer-Notebook.ipynb#Predict-this-user-where-they-will-go-on-ce
rtain-day-of-weekmn. Since latitude and longitude are same, looks like d
ummy, dropped.
checkins_data = checkins[checkins['user_id'] == user_id].copy()
checkins_data = checkins_data.drop(['id', 'user_id', 'latitude', 'longit
ude'], axis=1)
checkins_data['created_at'] = pd.to_datetime(checkins_data['created_at'
])
checkins_data['dayofweek'] = checkins_data['created_at'].dt.dayofweek
checkins_data['week'] = checkins_data['created_at'].dt.week
checkins_data = checkins_data.sort_values(['created_at'], ascending=True
)
checkins_data.reset_index(inplace=True, drop=True)
checkins_data.head(10)
```

Out[50]:

	venue_id	created_at	dayofweek	week
0	169552	2011-12-09 11:20:48	4	49
1	7491	2011-12-10 08:06:39	5	49
2	7491	2011-12-11 02:49:39	6	49
3	7491	2011-12-11 04:31:45	6	49
4	7491	2011-12-11 12:49:14	6	49
5	7491	2011-12-11 21:23:19	6	49
6	7491	2011-12-12 07:14:55	0	50
7	7491	2011-12-12 13:24:57	0	50
8	7491	2011-12-14 07:22:58	2	50
9	7491	2011-12-14 13:19:38	2	50

Label and Feature extraction

In [51]:
```
labels = checkins_data['venue_id']
labels.unique()
```

Out[51]:
```
array([169552,   7491,  39731])
```

In [52]:
```
features = checkins_data[['dayofweek']]
features.head(5)
```

Out[52]:

	dayofweek
0	4
1	5
2	6
3	6
4	6

Print training data shape

In [53]:
```
features = features.to_numpy()
labels = labels.to_numpy()
print(features.shape, labels.shape)
```

```
(57, 1) (57,)
```

Do preprocessing

In [54]:
```
from sklearn import preprocessing

le = preprocessing.LabelEncoder()
labels = le.fit_transform(labels)
le.classes_
```

Out[54]:
```
array([  7491,  39731, 169552])
```

Prepare training and test data

In [55]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(features, labels, te
st_size=0.20, random_state=10)
```

Build a model

In [56]:
```
import tensorflow as tf

model = tf.keras.models.Sequential([
  tf.keras.layers.Dense(100, activation=tf.nn.relu),
  tf.keras.layers.Dense(100, activation=tf.nn.relu),
  tf.keras.layers.Dense(len(le.classes_), activation=tf.nn.softmax)
])
```

In [57]:
```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Train model with training data

In [58]:
```
model.fit(X_train, y_train, epochs=10)
```

```
Epoch 1/10
2/2 [==============================] - 0s 1ms/step - loss: 1.6961 - acc
uracy: 0.2222
Epoch 2/10
2/2 [==============================] - 0s 3ms/step - loss: 1.4227 - acc
uracy: 0.2222
Epoch 3/10
2/2 [==============================] - 0s 3ms/step - loss: 1.2032 - acc
uracy: 0.2889
Epoch 4/10
2/2 [==============================] - 0s 5ms/step - loss: 1.0029 - acc
uracy: 0.9333
Epoch 5/10
2/2 [==============================] - 0s 4ms/step - loss: 0.8686 - acc
uracy: 0.9333
Epoch 6/10
2/2 [==============================] - 0s 2ms/step - loss: 0.7542 - acc
uracy: 0.9333
Epoch 7/10
2/2 [==============================] - 0s 2ms/step - loss: 0.6808 - acc
uracy: 0.9333
Epoch 8/10
2/2 [==============================] - 0s 2ms/step - loss: 0.6194 - acc
uracy: 0.9333
Epoch 9/10
2/2 [==============================] - 0s 2ms/step - loss: 0.5722 - acc
uracy: 0.9333
Epoch 10/10
2/2 [==============================] - 0s 1ms/step - loss: 0.5389 - acc
uracy: 0.9333
```

Out[58]: <tensorflow.python.keras.callbacks.History at 0x146013208>

Test model if it has low loss and high accuracy

In [59]:
```
metrics = model.evaluate(X_test, y_test)
```

```
1/1 [==============================] - 0s 866us/step - loss: 0.4039 - a
ccuracy: 0.9167
```

Make venue_id class probability prediction where this user will go next. If next day is Monday = 0

In [60]:
```
next_is_which_day = 0
pred = model.predict([[next_is_which_day]])
pred
```

Out[60]:
```
array([[0.37909272, 0.31756896, 0.30333826]], dtype=float32)
```

Show prediction result in probability

In [61]:
```
pd.DataFrame({'Labels': le.classes_, 'Probabilities': pred.reshape(-1)})
```

Out[61]:

	Labels	Probabilities
0	7491	0.379093
1	39731	0.317569
2	169552	0.303338

In [62]:
```
print('Predicted venue_id:', le.classes_[pred.argmax()])
```

```
Predicted venue_id: 7491
```

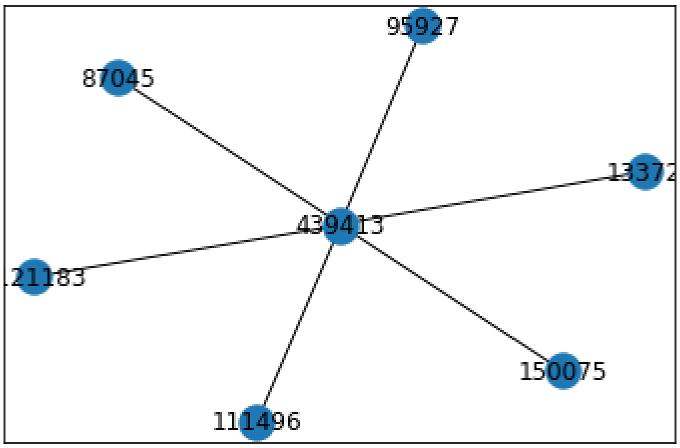
5. Challenges for top 10 more "social" users (with more friends):

In [63]:
```
users_top_10_social
```

Out[63]:

	first_user_id	x_friends
0	754	107676
1	56	105091
2	4489	94991
3	50	82387
4	512	78277
5	59	73760
6	1334	70181
7	52	67931
8	47	67505
9	40	67093

In [64]:
```
import networkx as nx

G_symmetric = nx.Graph()

for index, row in socialgraph[socialgraph['first_user_id']==439413].iter
rows():
    G_symmetric.add_edge(row['first_user_id'],row['second_user_id'])

nx.spring_layout(G_symmetric)
nx.draw_networkx(G_symmetric)
```



Problem 5.1: Draw path (with map) of week/month of users checkins.

It is strange that this being asked for top 10 users with most social instead of users with most checkins

In [65]:
```
import geopandas as gpd
import geoplot as gplt
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings('ignore')
```

In [66]:
```
users_top_10_checkins
```

Out[66]:

	user_id	x_times_checkin
0	1348362	57
1	1900906	52
2	1326476	48
3	1365850	47
4	386648	47
5	467043	46
6	651415	45
7	439413	45
8	304865	45
9	8622	45

Use user with more checkins as sample

In [67]:
```
sample_user_id = 1326476
checkins_week_month = checkins.copy()
checkins_week_month['created_at'] = pd.to_datetime(checkins_week_month[
'created_at'])
checkins_week_month['week'] = checkins_week_month['created_at'].dt.week
checkins_week_month['month'] = checkins_week_month['created_at'].dt.mont
h
sample_user_checkins = checkins_week_month[checkins_week_month['user_id'
]==sample_user_id]
sample_user_checkins.head(10)
```

Out[67]:

	id	user_id	venue_id	latitude	longitude	created_at	week	month
19399	19415	1326476	61002	33.058106	-112.047642	2011-12-09 02:26:28	49	12
22919	22935	1326476	4432	33.058106	-112.047642	2011-12-09 04:45:24	49	12
190900	190916	1326476	11138	33.058106	-112.047642	2011-12-12 23:31:39	50	12
199607	199623	1326476	2964	33.058106	-112.047642	2011-12-13 04:25:57	50	12
302114	302130	1326476	1011459	33.058106	-112.047642	2011-12-24 12:29:01	51	12
312016	312032	1326476	68691	33.058106	-112.047642	2011-12-24 16:12:12	51	12
314680	314696	1326476	11138	33.058106	-112.047642	2011-12-24 19:08:44	51	12
340270	340286	1326476	28304	33.058106	-112.047642	2011-12-25 15:27:45	51	12
347842	347858	1326476	25610	33.058106	-112.047642	2011-12-25 19:56:54	51	12
376508	376524	1326476	4202	33.058106	-112.047642	2011-12-26 16:20:13	52	12

Provided data does not seems have valid checkin longitude and latitude. Get from Venues instead...

In [68]:
```
sample_user_checkins = sample_user_checkins.drop(['id','longitude', 'lat
itude'],axis=1)
sample_user_checkins = sample_user_checkins.merge(venues, left_on='venue
_id', right_on='id', how='left')
sample_user_checkins = sample_user_checkins.drop(['id'],axis=1)
sample_user_checkins
```

Out[68]:

	user_id	venue_id	created_at	week	month	latitude	longitude	geometry
0	1326476	61002	2011-12-09 02:26:28	49	12	38.693650	-121.590199	POINT (-121.59020 38.69365)
1	1326476	4432	2011-12-09 04:45:24	49	12	33.436527	-112.002182	POINT (-112.00218 33.43653)
2	1326476	11138	2011-12-12 23:31:39	50	12	39.848349	-104.674988	POINT (-104.67499 39.84835)
3	1326476	2964	2011-12-13 04:25:57	50	12	40.690596	-74.178100	POINT (-74.17810 40.69060)
4	1326476	1011459	2011-12-24 12:29:01	51	12	33.435484	-111.985859	POINT (-111.98586 33.43548)
5	1326476	68691	2011-12-24 16:12:12	51	12	29.653113	-95.276656	POINT (-95.27666 29.65311)
6	1326476	11138	2011-12-24 19:08:44	51	12	39.848349	-104.674988	POINT (-104.67499 39.84835)
7	1326476	28304	2011-12-25 15:27:45	51	12	41.787905	-87.740700	POINT (-87.74070 41.78790)
8	1326476	25610	2011-12-25 19:56:54	51	12	38.742402	-90.365896	POINT (-90.36590 38.74240)
9	1326476	4202	2011-12-26 16:20:13	52	12	30.202577	-97.667148	POINT (-97.66715 30.20258)
10	1326476	4432	2011-12-26 19:36:41	52	12	33.436527	-112.002182	POINT (-112.00218 33.43653)
11	1326476	11492	2012-01-23 15:44:09	4	1	40.750476	-73.993607	POINT (-73.99361 40.75048)
12	1326476	2964	2012-01-23 16:48:18	4	1	40.690596	-74.178100	POINT (-74.17810 40.69060)
13	1326476	4432	2012-01-24 07:25:38	4	1	33.436527	-112.002182	POINT (-112.00218 33.43653)
14	1326476	4432	2012-01-25 22:51:14	4	1	33.436527	-112.002182	POINT (-112.00218 33.43653)
15	1326476	12004	2012-01-26 01:38:14	4	1	32.732346	-117.197299	POINT (-117.19730 32.73235)
16	1326476	5222	2012-01-26 03:41:42	4	1	37.616407	-122.386236	POINT (-122.38624 37.61641)

	user_id	venue_id	created_at	week	month	latitude	longitude	geometry
17	1326476	4432	2012-01-26 06:19:11	4	1	33.436527	-112.002182	POINT (-112.00218 33.43653)
18	1326476	4432	2012-01-27 16:39:08	4	1	33.436527	-112.002182	POINT (-112.00218 33.43653)
19	1326476	4432	2012-03-05 20:51:08	10	3	33.436527	-112.002182	POINT (-112.00218 33.43653)
20	1326476	28304	2012-03-16 12:58:17	11	3	41.787905	-87.740700	POINT (-87.74070 41.78790)
21	1326476	4432	2012-03-16 20:26:24	11	3	33.436527	-112.002182	POINT (-112.00218 33.43653)
22	1326476	4432	2012-03-16 20:26:24	11	3	33.436527	-112.002182	POINT (-112.00218 33.43653)
23	1326476	4432	2012-03-19 19:59:15	12	3	33.436527	-112.002182	POINT (-112.00218 33.43653)
24	1326476	21948	2012-03-19 23:28:33	12	3	40.786683	-111.981926	POINT (-111.98193 40.78668)
25	1326476	16642	2012-03-20 05:06:56	12	3	29.986721	-90.255175	POINT (-90.25517 29.98672)
26	1326476	4432	2012-04-01 20:06:54	13	4	33.436527	-112.002182	POINT (-112.00218 33.43653)
27	1326476	12004	2012-04-01 22:41:38	13	4	32.732346	-117.197299	POINT (-117.19730 32.73235)
28	1326476	61002	2012-04-02 01:10:15	14	4	38.693650	-121.590199	POINT (-121.59020 38.69365)
29	1326476	4432	2012-04-02 03:25:44	14	4	33.436527	-112.002182	POINT (-112.00218 33.43653)
30	1326476	11138	2012-04-02 06:01:45	14	4	39.848349	-104.674988	POINT (-104.67499 39.84835)
31	1326476	11138	2012-04-02 22:09:03	14	4	39.848349	-104.674988	POINT (-104.67499 39.84835)
32	1326476	7620	2012-04-03 00:29:21	14	4	33.943894	-118.405023	POINT (-118.40502 33.94389)
33	1326476	39814	2012-04-03 02:24:19	14	4	37.366625	-121.926304	POINT (-121.92630 37.36662)

	user_id	venue_id	created_at	week	month	latitude	longitude	geometry
34	1326476	12004	2012-04-03 04:03:27	14	4	32.732346	-117.197299	POINT (-117.19730 32.73235)
35	1326476	575803	2012-04-04 18:24:43	14	4	33.319779	-111.974808	POINT (-111.97481 33.31978)
36	1326476	4432	2012-04-07 18:48:57	14	4	33.436527	-112.002182	POINT (-112.00218 33.43653)
37	1326476	4432	2012-04-10 17:40:46	15	4	33.436527	-112.002182	POINT (-112.00218 33.43653)
38	1326476	4432	2012-04-11 00:42:58	15	4	33.436527	-112.002182	POINT (-112.00218 33.43653)
39	1326476	60	2012-04-11 22:13:57	15	4	36.085055	-115.149937	POINT (-115.14994 36.08505)
40	1326476	5222	2012-04-12 03:03:35	15	4	37.616407	-122.386236	POINT (-122.38624 37.61641)
41	1326476	4432	2012-04-12 05:14:46	15	4	33.436527	-112.002182	POINT (-112.00218 33.43653)
42	1326476	4432	2012-04-13 00:22:30	15	4	33.436527	-112.002182	POINT (-112.00218 33.43653)
43	1326476	64	2012-04-13 04:00:09	15	4	44.883546	-93.211484	POINT (-93.21148 44.88355)
44	1326476	4432	2012-04-17 19:05:09	16	4	33.436527	-112.002182	POINT (-112.00218 33.43653)
45	1326476	12004	2012-04-18 01:30:15	16	4	32.732346	-117.197299	POINT (-117.19730 32.73235)
46	1326476	4432	2012-04-18 03:16:31	16	4	33.436527	-112.002182	POINT (-112.00218 33.43653)
47	1326476	1031600	2012-04-21 01:51:57	16	4	33.368739	-111.938347	POINT (-111.93835 33.36874)

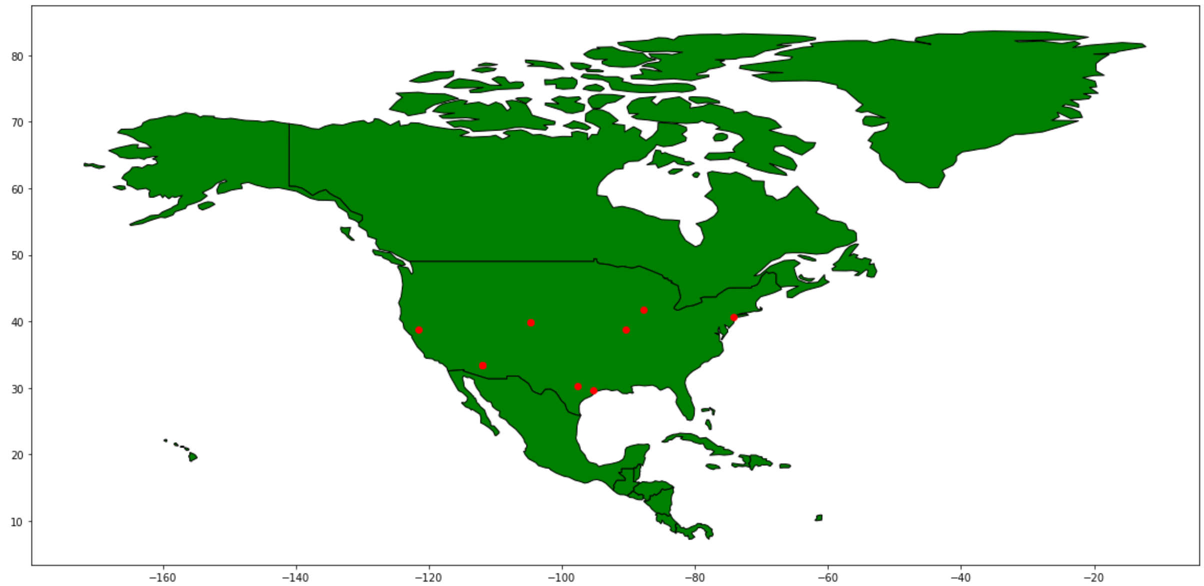
Plot this sample user month 12 checkins

In [69]:
```
sample_user_checkins_month_12 = sample_user_checkins[sample_user_checkin
s['month']==12]
geo_df_month_12 = gpd.GeoDataFrame(sample_user_checkins_month_12, geomet
ry=gpd.points_from_xy(sample_user_checkins_month_12.longitude, sample_us
er_checkins_month_12.latitude))
geo_df_month_12
```

Out[69]:

	user_id	venue_id	created_at	week	month	latitude	longitude	geometry
0	1326476	61002	2011-12-09 02:26:28	49	12	38.693650	-121.590199	POINT (-121.59020 38.69365)
1	1326476	4432	2011-12-09 04:45:24	49	12	33.436527	-112.002182	POINT (-112.00218 33.43653)
2	1326476	11138	2011-12-12 23:31:39	50	12	39.848349	-104.674988	POINT (-104.67499 39.84835)
3	1326476	2964	2011-12-13 04:25:57	50	12	40.690596	-74.178100	POINT (-74.17810 40.69060)
4	1326476	1011459	2011-12-24 12:29:01	51	12	33.435484	-111.985859	POINT (-111.98586 33.43548)
5	1326476	68691	2011-12-24 16:12:12	51	12	29.653113	-95.276656	POINT (-95.27666 29.65311)
6	1326476	11138	2011-12-24 19:08:44	51	12	39.848349	-104.674988	POINT (-104.67499 39.84835)
7	1326476	28304	2011-12-25 15:27:45	51	12	41.787905	-87.740700	POINT (-87.74070 41.78790)
8	1326476	25610	2011-12-25 19:56:54	51	12	38.742402	-90.365896	POINT (-90.36590 38.74240)
9	1326476	4202	2011-12-26 16:20:13	52	12	30.202577	-97.667148	POINT (-97.66715 30.20258)
10	1326476	4432	2011-12-26 19:36:41	52	12	33.436527	-112.002182	POINT (-112.00218 33.43653)

In [70]:
```
# As this user checkins are in North America, filter by this continent w
hen showing
world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
ax = world[world.continent == 'North America'].plot(figsize=(20,15),colo
r='green', edgecolor='black')
geo_df_month_12.plot(ax=ax, color='red')

plt.show()
```



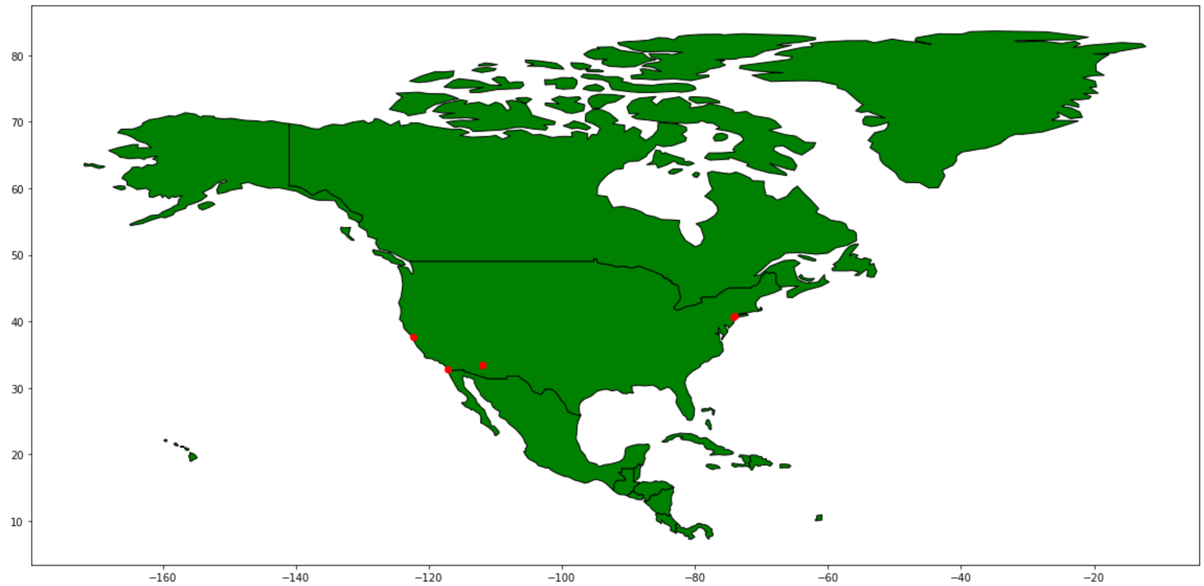
Plot this sample user week 4 checkins

In [71]:
```
sample_user_checkins_week_4 = sample_user_checkins[sample_user_checkins[
'week']==4]
geo_df_week_4 = gpd.GeoDataFrame(sample_user_checkins_week_4, geometry=g
pd.points_from_xy(sample_user_checkins_week_4.longitude, sample_user_che
ckins_week_4.latitude))
geo_df_week_4
```

Out[71]:

	user_id	venue_id	created_at	week	month	latitude	longitude	geometry
11	1326476	11492	2012-01-23 15:44:09	4	1	40.750476	-73.993607	POINT (-73.99361 40.75048)
12	1326476	2964	2012-01-23 16:48:18	4	1	40.690596	-74.178100	POINT (-74.17810 40.69060)
13	1326476	4432	2012-01-24 07:25:38	4	1	33.436527	-112.002182	POINT (-112.00218 33.43653)
14	1326476	4432	2012-01-25 22:51:14	4	1	33.436527	-112.002182	POINT (-112.00218 33.43653)
15	1326476	12004	2012-01-26 01:38:14	4	1	32.732346	-117.197299	POINT (-117.19730 32.73235)
16	1326476	5222	2012-01-26 03:41:42	4	1	37.616407	-122.386236	POINT (-122.38624 37.61641)
17	1326476	4432	2012-01-26 06:19:11	4	1	33.436527	-112.002182	POINT (-112.00218 33.43653)
18	1326476	4432	2012-01-27 16:39:08	4	1	33.436527	-112.002182	POINT (-112.00218 33.43653)

In [72]:
```
# As this user checkins are in North America, filter by this continent w
hen showing
world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
ax = world[world.continent == 'North America'].plot(figsize=(20,15),colo
r='green', edgecolor='black')
geo_df_week_4.plot(ax=ax, color='red')

plt.show()
```



Problem 5.2: List your friends and how close they are in terms of "taste" (based on venues visited and ranked)

In [73]:
```
# Users most social and has ever check in before
users_social_checkins = users_social.merge(users_checkins, left_on='firs
t_user_id', right_on='user_id')
users_social_checkins = users_social_checkins[['user_id', 'x_friends',
'x_times_checkin']]
users_social_checkins
```

Out[73]:

	user_id	x_friends	x_times_checkin
0	101	36532	2
1	3561	25062	2
2	48	19565	1
3	110131	14734	4
4	68741	14127	17
...	...	...	...
227413	10015	1	1
227414	9872	1	1
227415	4653	1	3
227416	4201	1	1
227417	3924	1	1

227418 rows × 3 columns

Cluster ratings

In [74]:
```
venue_rating_by_user
```

Out[74]:

	venue_id	user_id	rating
0	1	1	5.0
1	2	2	5.0
2	3	3	5.0
3	4	4	5.0
4	5	5	5.0
...	...	...	...
2436718	1142965	797976	2.0
2436719	1142966	1398671	2.0
2436720	1143011	83334	2.0
2436721	1143011	1626801	2.0
2436722	1143020	1407243	2.0

2436723 rows × 3 columns

Create variable "my_network" contains user and his friends user_id

In [75]:
```
user = 49857
```

In [76]:
```
friends = socialgraph[socialgraph['first_user_id']==user]['second_user_i
d'].unique()
friends
```

Out[76]:
```
array([    47,     58, 148026,   4489,  19418,   9839, 855492, 406230,
           40, 855504,  10570, 672838, 855523, 855525, 855529, 822963,
       289412, 855538, 855540, 543445, 169657, 855574,  72121])
```

In [77]:
```
self = socialgraph[socialgraph['first_user_id']==user]['first_user_id'].
unique()
self
```

Out[77]:
```
array([49857])
```

In [78]:
```
my_network = np.concatenate((self, friends), axis=0)
my_network
```

Out[78]:
```
array([ 49857,     47,     58, 148026,   4489,  19418,   9839, 855492,
       406230,     40, 855504,  10570, 672838, 855523, 855525, 855529,
       822963, 289412, 855538, 855540, 543445, 169657, 855574,  72121])
```

Create variable "venue_rating_my_network" contains venue ratings in "my_network"

In [79]:
```
venue_rating_my_network = venue_rating_by_user[venue_rating_by_user['user_id'].isin(my_network)].copy()
venue_rating_my_network
```

Out[79]:

	venue_id	user_id	rating
129230	190115	72121	5.0
285123	409649	19418	5.0
285124	409650	19418	5.0
285125	409651	19418	5.0
310480	445087	9839	5.0
...	...	...	...
1558438	409652	49857	2.5
1573139	64	10570	2.0
1709390	7284	9839	2.0
2178352	151331	9839	2.0
2186898	157565	9839	2.0

633 rows × 3 columns

Create variable "venue_rating_my_network_pivot" contains venue id as columns and rating as value

In [80]:
```
venue_rating_my_network_pivot = venue_rating_my_network.pivot(index='user_id', columns='venue_id', values='rating')
```

In [81]:
```
venue_rating_my_network_pivot = venue_rating_my_network_pivot.fillna(0)
venue_rating_my_network_pivot
```

Out[81]:

venue_id user_id	64	386	388	397	406	407	415	416	417	418	...	878870	878871	878872	87887
40	0.0	4.0	4.0	4.0	0.0	0.0	4.0	4.0	4.0	4.0	...	0.0	0.0	0.0	0
47	0.0	0.0	0.0	0.0	0.0	4.0	0.0	4.0	0.0	0.0	...	0.0	0.0	0.0	0
58	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0
4489	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0
9839	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0
10570	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0
19418	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0
49857	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0
72121	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	4.0	4.0	4.0	4
148026	0.0	0.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0
289412	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0
672838	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0
855523	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0
855525	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0

14 rows × 616 columns

In [82]:
```
labels = venue_rating_my_network_pivot.index.values
features = venue_rating_my_network_pivot.values.astype('int')
```

In [83]:
```
from sklearn.cluster import KMeans
from sklearn import metrics
import numpy as np
```

In [84]:
```
kmeans_model = KMeans(n_clusters=7, max_iter=10).fit(features)
```

In [85]:
```
print('Silhouette_score: ', metrics.silhouette_score(features, kmeans_mo
del.labels_))
```

```
Silhouette_score:  0.2880649018757592
```

In [86]:
```
kmeans_model.labels_
```

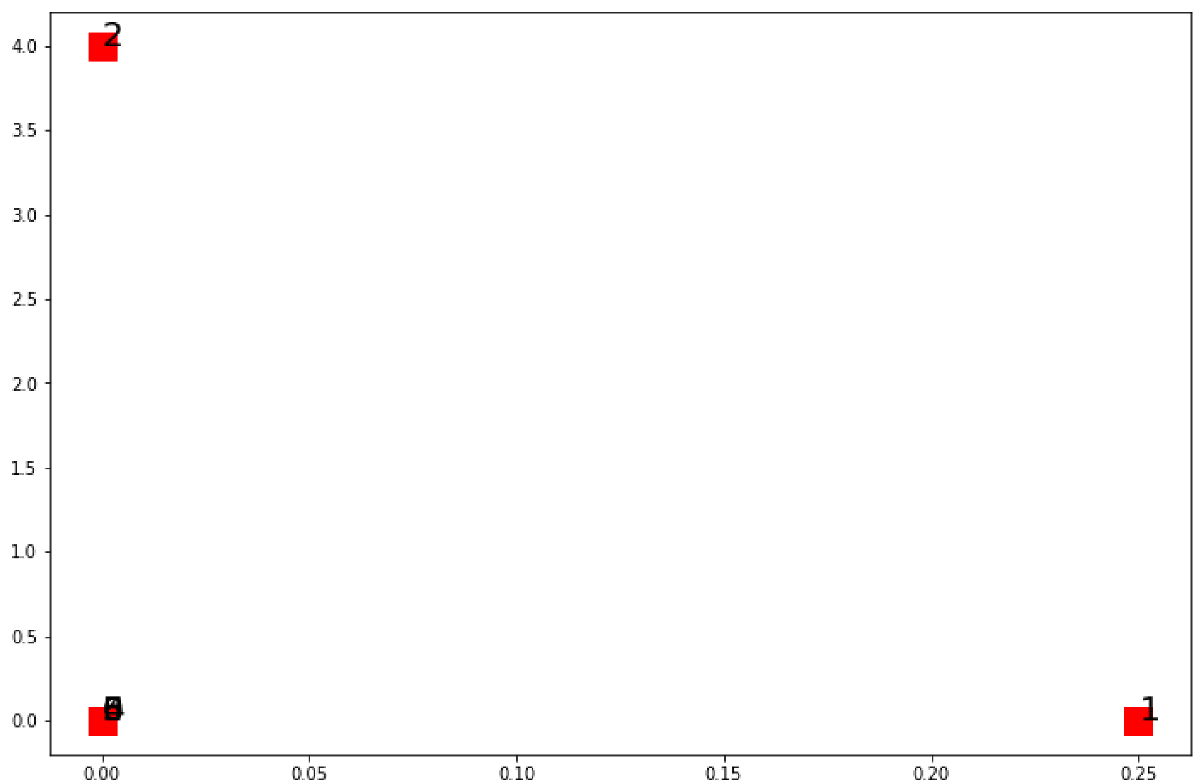
Out[86]:
```
array([2, 3, 5, 6, 1, 1, 1, 1, 0, 4, 1, 1, 1, 1], dtype=int32)
```

In [87]:
```
centroids = kmeans_model.cluster_centers_
centroids
```

Out[87]:
```
array([[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
        4.00000000e+00, 4.00000000e+00, 4.00000000e+00],
       [2.50000000e-01, 5.55111512e-17, 5.55111512e-17, ...,
        5.55111512e-17, 5.55111512e-17, 5.55111512e-17],
       [0.00000000e+00, 4.00000000e+00, 4.00000000e+00, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       ...,
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00]])
```

Show cluster centroids

In [88]:
```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(12,8))

plt.scatter(centroids[:,0], centroids[:,1], c='r', s=250, marker='s')

for i in range(len(centroids)):
    plt.annotate(i , (centroids[i][0], centroids[i][1]), fontsize=20)
```



In [89]:
```
category = pd.Series(kmeans_model.labels_)
```

In [90]:
```
user_ids = pd.Series(labels)
```

In [91]:
```
categorized = pd.DataFrame({'user_id': user_ids, 'category': category})
categorized
```

Out[91]:

	user_id	category
0	40	2
1	47	3
2	58	5
3	4489	6
4	9839	1
5	10570	1
6	19418	1
7	49857	1
8	72121	0
9	148026	4
10	289412	1
11	672838	1
12	855523	1
13	855525	1

Find out what category is this user

In [92]:
```
my_category = categorized[categorized['user_id']==user]
my_category = my_category['category'].values[0]
print('User ID:',user,'Category:',my_category)
```

```
User ID: 49857 Category: 1
```

Find his friends belongs to same cluster category

In [93]:
```
my_category_cluster_filter = categorized[categorized['category']==my_cat
egory]['user_id']
my_category_cluster_filter
```

Out[93]:
```
4       9839
5      10570
6      19418
7      49857
10    289412
11    672838
12    855523
13    855525
Name: user_id, dtype: int64
```

Close database connection

In [94]:
```
conn.close()
print("Database closed successfully")
```

```
Database closed successfully
```

In [ ]: