In [1]:
```
import numpy as np
import pandas as pd
from datetime import timedelta
from geopy.distance import geodesic
import geopandas as gpd
from random import shuffle
import giddy
```

Objective: quick mobility analysis in Yellowstone National Park

In [2]:
```
# Read in SHP file of OPTICS cluster
gdf = gpd.read_file('OPTICS\optics.shp')
gdf = gdf.drop(['REACHDIST','REACHORDER','COLOR_ID','geometry'], axis=1) # dr
gdf['CLUSTER_ID'] = gdf['CLUSTER_ID'].fillna(-1).astype(int) # fill NaN with
gdf = gdf[ gdf['CLUSTER_ID'] != -1 ] # keep all points that are part of an OP

# Read in EXCEL of 2018 YNP Flickr photos
raw_df = pd.read_excel('2018_Master.xlsx')

# Processes 2018 Photos
df = raw_df[['Longitude', 'Latitude','Owner Name', 'Photo ID', 'Date Taken',
df['SOURCE_ID'] = df.index
df['Taken'] = pd.to_datetime(df['Date Taken'] + ' ' + df['Time Taken'])
df = df.drop(['Time Taken','Date Taken'], axis=1)

# Merge OPTICS and raw on Photo ID
df = gdf.merge(df, on='SOURCE_ID')

# Sort results
df = df.sort_values(['Owner Name', 'Taken'], ascending=[True, True])
df = df.reset_index(drop=True)
```

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

Out[3]:

	SOURCE_ID	CLUSTER_ID	Longitude	Latitude	Owner Name	Photo ID	Taken
0	7682.0	4	-110.705848	45.033334	103084254@N04	29492059557	2018-07-30 20:11:03
1	8066.0	4	-110.705848	45.033334	103084254@N04	29492059557	2018-07-30 20:11:03
2	7681.0	4	-110.705848	45.033334	103084254@N04	30561527458	2018-07-30 20:15:55
3	8065.0	4	-110.705848	45.033334	103084254@N04	30561527458	2018-07-30 20:15:55
4	7680.0	4	-110.705848	45.033334	103084254@N04	43521270755	2018-07-30 20:21:23

In [4]:
```
def distance_between(latA, lonA, latB, lonB):
    return geodesic( (latA, lonA), (latB, lonB))
```

In [5]:
```python
"""
Drop photos taken by the same user within define time period
and spatial distance.

"""
DIST = .5 # km
TIME = timedelta(hours=2, minutes=30)

df['drop'] = False

for i, row in df.iloc[1:].iterrows():

    prev_row = df.iloc[i-1]

    temporal_dwell = abs(row['Taken'] - prev_row['Taken'])
    spatial_dwell = distance_between(row['Latitude'], row['Longitude'], prev_

    if row['Owner Name'] == prev_row['Owner Name'] and \
        temporal_dwell < TIME and \
        spatial_dwell < DIST:

        df['drop'][i] = True


print(f" Dropping {len(df[df['drop'] == True])} pictures taken within {TIME} a

df = df[df['drop'] != True]
df = df.drop('drop', axis=1)
df = df.reset_index(drop=True)

print(f"Leaving us with {df.shape}")
```

```
 Dropping 7407 pictures taken within 2:30:00 and within 0.5 km of each other
Leaving us with (2040, 7)
```

In [6]:
```python
"""
Drop photos with users who took only one photo

"""

print(f"Starting frame has shape {df.shape}")
print(f"Removing single posts...")

df = df[df.groupby('Owner Name')['Owner Name'].transform(len) > 1 ]
df = df.reset_index(drop=True)

print(f"Ending shape {df.shape}")
```

```
Starting frame has shape (2040, 7)
Removing single posts...
Ending shape (1959, 7)
```

In [7]:
```python
df.head()
```

Out[7]:

	SOURCE_ID	CLUSTER_ID	Longitude	Latitude	Owner Name	Photo ID	Taken
0	5043.0	6	-110.434988	44.810507	10393601@N08	43427477695	2018-07-11 08:42:01

	SOURCE_ID	CLUSTER_ID	Longitude	Latitude	Owner Name	Photo ID	Taken
1	5063.0	4	-110.701696	44.729997	10393601@N08	43278329464	2018-07-17 07:32:53
2	5342.0	7	-110.500075	44.713030	107115094@N05	43348243812	2018-07-13 13:46:32
3	5343.0	7	-110.479278	44.720847	107115094@N05	28526710077	2018-07-13 14:15:40

In [8]:

```
"""
Label consequative photos from the same user as
unique trips

"""


df['trip'] = ''

trip_number = 1

df['trip'][0] = trip_number

for i, row in df.iloc[1:].iterrows():
    prev_row = df.iloc[i-1]

    if prev_row['Owner Name'] == row['Owner Name']:
        df['trip'][i] = trip_number
    else:
        trip_number = trip_number + 1
        df['trip'][i] = trip_number
```

In [9]:

```
df.head()
```

Out[9]:

	SOURCE_ID	CLUSTER_ID	Longitude	Latitude	Owner Name	Photo ID	Taken	trip
0	5043.0	6	-110.434988	44.810507	10393601@N08	43427477695	2018-07-11 08:42:01	1
1	5063.0	4	-110.701696	44.729997	10393601@N08	43278329464	2018-07-17 07:32:53	1
2	5342.0	7	-110.500075	44.713030	107115094@N05	43348243812	2018-07-13 13:46:32	2
3	5343.0	7	-110.479278	44.720847	107115094@N05	28526710077	2018-07-13 14:15:40	2
4	51.0	3	-110.832462	44.460236	11090497@N06	48987339541	2018-09-08 09:40:54	3

In [10]:
```python
"""
Find cluster transitions between each photo
"""

transitions = []

for i, row in df.iloc[1:].iterrows():
    prev_row = df.iloc[i-1]
    transitions.append((prev_row['CLUSTER_ID'], row['CLUSTER_ID']))

print(f'Found {len(transitions)} transitions')
```

```
Found 1958 transitions
```

In [11]:
```python
"""
Create markov object from transitions list passed as np array

"""

m = giddy.markov.Markov(np.array(transitions))
```

```
The Markov Chain is irreducible and is composed by:
1 Recurrent class (indices):
[ 0  1  2  3  4  5  6  7  8  9 10 11]
0 Transient classes.
The Markov Chain has 0 absorbing states.
```

In [12]:
```python
print('Transitions:')
print(m.transitions)
```

```
Transitions:
[[ 29.   1.  26.   4.   4.   1.   9.   6.   3.   3.   0.   0.]
 [  2.   3.   2.   0.   0.   0.   3.   0.   0.   0.   0.   0.]
 [ 20.   5. 395.  58.  19.   1.  43.  12.   1.  16.   0.   0.]
 [  3.   0.  52. 246.  50.   2.  37.   8.   1.   6.   1.   0.]
 [  9.   0.  18.  45. 198.   6.  34.   7.   2.   7.   0.   1.]
 [  2.   0.   3.   3.   7.  13.   4.   1.   0.   0.   0.   0.]
 [ 11.   0.  39.  34.  37.   9. 175.  18.   1.   1.   4.   0.]
 [  8.   1.  12.   7.   9.   0.  19.  44.   3.   0.   0.   2.]
 [  1.   0.   1.   2.   1.   0.   1.   8.   5.   0.   0.   1.]
 [  1.   0.  20.   5.   2.   0.   4.   0.   1.  30.   0.   0.]
 [  0.   0.   2.   2.   0.   0.   0.   1.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.   0.   0.   3.   0.   0.   1.]]
```

In [13]:
```python
print('Probability')
print(m.p)
```

```
Probability
[[0.3372093  0.01162791 0.30232558 0.04651163 0.04651163 0.01162791
  0.10465116 0.06976744 0.03488372 0.03488372 0.         0.        ]
 [0.2        0.3        0.2        0.         0.         0.
  0.3        0.         0.         0.         0.         0.        ]
 [0.03508772 0.00877193 0.69298246 0.10175439 0.03333333 0.00175439
  0.0754386  0.02105263 0.00175439 0.02807018 0.         0.        ]
 [0.00738916 0.         0.12807882 0.60591133 0.12315271 0.00492611
  0.091133   0.01970443 0.00246305 0.01477833 0.00246305 0.        ]
 [0.02752294 0.         0.05504587 0.13761468 0.60550459 0.01834862
  0.10397554 0.02140673 0.00611621 0.02140673 0.         0.0030581 ]
 [0.06060606 0.         0.09090909 0.09090909 0.21212121 0.39393939
  0.12121212 0.03030303 0.         0.         0.         0.        ]
 [0.03343465 0.         0.11854103 0.10334347 0.11246201 0.02735562
  0.53191489 0.05471125 0.00303951 0.00303951 0.01215805 0.        ]
 [0.07619048 0.00952381 0.11428571 0.06666667 0.08571429 0.
```

```
 0.18095238 0.41904762 0.02857143 0.         0.         0.01904762]
 [0.05       0.         0.05       0.1        0.05       0.
  0.05       0.4        0.25       0.         0.         0.05      ]
 [0.01587302 0.         0.31746032 0.07936508 0.03174603 0.
  0.06349206 0.         0.01587302 0.47619048 0.         0.        ]
 [0.         0.         0.4        0.4        0.         0.
  0.         0.2        0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.75       0.         0.         0.25      ]]
```

In [ ]:

```
Visualize Results
```

```
Error: Timed out waiting to get a heartbeat from kernel process.
at m.waitForHeartbeat (c:\Users\mtral\.vscode\extensions\ms-toolsai.jupyter-20
20.12.414227025\out\client\extension.js:49:637256)
```

In [14]:

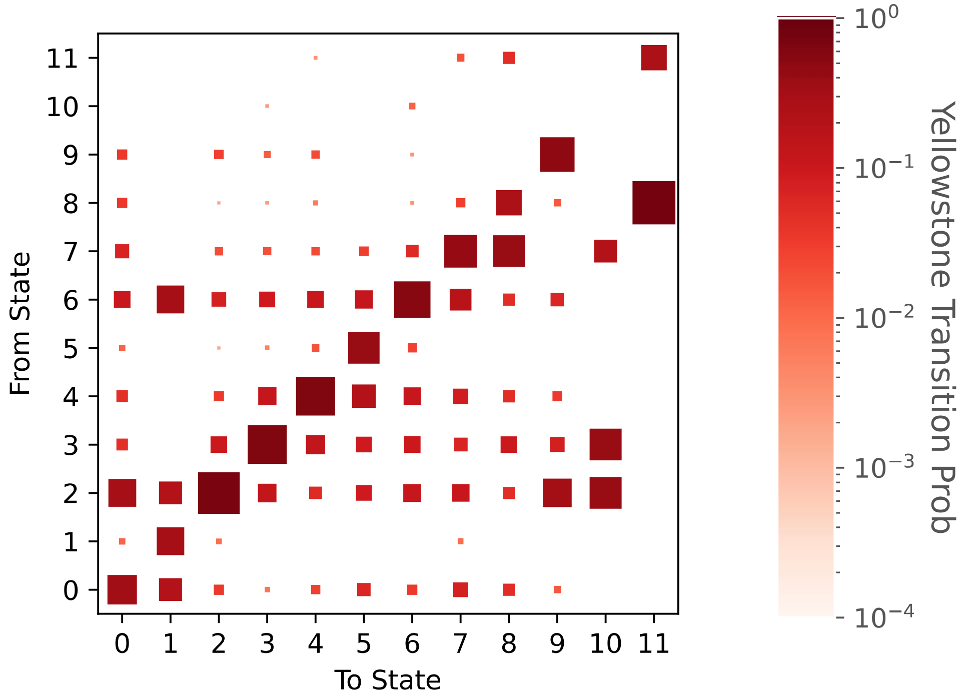
```
import matplotlib as mpl
import matplotlib.patches as patches
import matplotlib.pyplot as plt

import transitionMatrix as tm
```

In [15]:

```
prob = m.p
myMatrix = tm.TransitionMatrix(prob)
```

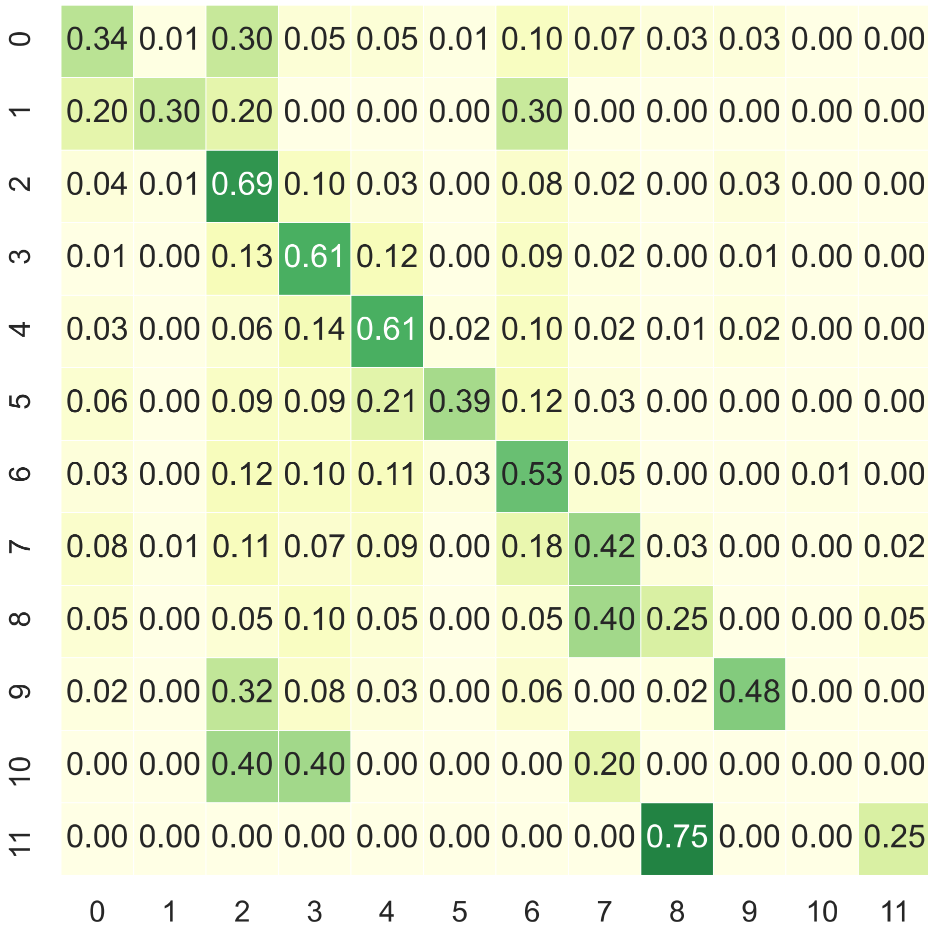
In [16]:

```
fig = plt.figure()
ax = fig.add_subplot(111, aspect='equal')

plt.style.use(['ggplot'])
plt.ylabel('From State')
plt.xlabel('To State')
mymap = plt.get_cmap("RdYlGn")
mymap = plt.get_cmap("Reds")
normalize = mpl.colors.LogNorm(vmin=0.0001, vmax=1)

matrix_size = myMatrix.shape[0]
square_size = 1.0 / matrix_size

diagonal = myMatrix.diagonal()

ax.set_xticklabels(range(0, matrix_size))
ax.set_yticklabels(range(0, matrix_size))
ax.xaxis.set_ticks(np.arange(0 + 0.5 * square_size, 1 + 0.5 * square_size, squ
ax.yaxis.set_ticks(np.arange(0 + 0.5 * square_size, 1 + 0.5 * square_size, squ

for i in range(0, matrix_size):
    for j in range(0, matrix_size):
        if myMatrix[i, j] > 0:
            rect_size = np.sqrt(myMatrix[i, j]) * square_size
        else:
            rect_size = 0

        dx = 0.5 * (square_size - rect_size)
        dy = 0.5 * (square_size - rect_size)
        p = patches.Rectangle(
            (i * square_size + dx, j * square_size + dy),
            rect_size,
            rect_size,
            fill=True,
            color=mymap(normalize(myMatrix[i, j]))
        )
        ax.add_patch(p)

cbax = fig.add_axes([0.85, 0.12, 0.05, 0.78])
cb = mpl.colorbar.ColorbarBase(cbax, cmap=mymap, norm=normalize, orientation=
cb.set_label("Yellowstone Transition Prob", rotation=270, labelpad=15)

plt.savefig('Tran_Matrix_Vis.png')
plt.show(block=True)
plt.interactive(False)
```



In [17]:
```
import seaborn as sns
```

In [18]:
```
sns.set()

fig, ax = plt.subplots(figsize = (7,6))
im = sns.heatmap(m.p, annot=True, linewidths=.4, ax=ax, cbar=False, vmin=0, vm
                     square=True,   cmap="YlGn",fmt='.2f')
```



In [20]:
```
fig.savefig("prob_matrix")
```