## Acknowlegement

This notebook provides an example of PySAL in combination with Oracle Spatial for spatio-temporal analysis.
It is based on http://pysal.org/notebooks/explore/giddy/Markov_Based_Methods.html (http://pysal.org/notebooks/explore/giddy/Markov_Based_Methods.html) (Serge Rey sjsrey@gmail.com, Wei Kang weikang9009@gmail.com)

Unemployment data from https://www.bls.gov/lau/#tables (https://www.bls.gov/lau/#tables)

## Scenario

For a region of interest, how is unemployment over time related to location:

- Probabilities of changes based on 'regional' unemployment?
- Probabilities of transitions from hotspot to coldspot, visa versa?

Example results for a region of interest:

- High unemployment in a county has ~60% probablity of remaining high if neighboring unemployment is in the top 20%.
- This drops to ~50% if neighboring unemployment is in the 4th quintile.
- A transition from an unemployment hotspot to coldspot is expected to take ~13 years.

## Approach

- Oracle Spatial for spatial data management, pre-processing, preparation
- PySAL (Python library) for spatial data science
- Jupyter notebook for running Python code, viewing results, and commentary (like this)

## Required libraries

In [29]:
```
import cx_Oracle
from shapely.wkt import loads
import pandas as pd
import geopandas as gpd
import pysal.lib
from pysal.viz import mapclassify as mc
from pysal.explore import giddy
from pysal.explore.esda.moran import Moran
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
np.set_printoptions(precision=4)
```

## Data for analysis

**Establish Oracle connection**

In [2]:
```
file = open('/opt/dbconfig.txt', 'r')
user = file.readline().strip()
pwd = file.readline().strip()
host_port_service = file.readline().strip()
connection = cx_Oracle.connect(user, pwd, host_port_service)
cursor = connection.cursor()
```

In [3]:
```
def OutputTypeHandler(cursor, name, defaultType, size, precision, scale):
    if defaultType == cx_Oracle.CLOB:
        return cursor.var(cx_Oracle.LONG_STRING, arraysize = cursor.arraysize)
connection.outputtypehandler = OutputTypeHandler
```

**Preview the unemployment data**

In [4]:
```
cursor.execute("""
SELECT statefips, countyfips, year, unemp_pct
FROM bls_unemployment
where rownum<10
""")
pd.DataFrame(cursor.fetchall(), columns = ['STATEFIPS','COUNTYFIPS','YEAR', 'UNEM
P_PCT'])
```

Out[4]:

| | STATEFIPS | COUNTYFIPS | YEAR | UNEMP_PCT |
|---|---|---|---|---|
| **0** | 13 | 087 | 2000 | 4.6 |
| **1** | 13 | 089 | 2000 | 3.3 |
| **2** | 13 | 091 | 2000 | 4.8 |
| **3** | 13 | 093 | 2000 | 4.8 |
| **4** | 13 | 095 | 2000 | 5.3 |
| **5** | 13 | 097 | 2000 | 3.0 |
| **6** | 13 | 099 | 2000 | 5.1 |
| **7** | 13 | 101 | 2000 | 3.5 |
| **8** | 13 | 103 | 2000 | 3.2 |

**Model requires data as 1 row per county with a column per year.**

In [5]:
```
cursor.execute("""
SELECT * FROM
(select statefips, countyfips, unemp_pct, year from bls_unemployment)
PIVOT( avg(unemp_pct)
    FOR year IN (1996,1997,1998,1999,2000,2001,2002,2003,
                 2004,2005,2006,2007,2008,2009,2010,2011,
                 2012,2013,2014,2015,2016,2017,2018) )
WHERE rownum<10
""")
pd.DataFrame(cursor.fetchall(), columns = ['STATEFIPS','COUNTYFIPS','1996','1997'
,
                                           '1998','1999','2000','2001','2002','20
03',
                                           '2004','2005','2006','2010','2007','20
08',
                                           '2009','2011','2012','2013','2014','20
15',
                                           '2016','2017','2018'])
```

Out[5]:

| | STATEFIPS | COUNTYFIPS | 1996 | 1997 | 1998 | 1999 | 2000 | 2001 | 2002 | 2003 | ... | 2008 | 2009 | 2011 | 2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 01 | 001 | 3.9 | 3.9 | 3.1 | 3.8 | 4.0 | 4.1 | 4.8 | 5.1 | ... | 9.7 | 8.9 | 8.4 | 6 |
| **1** | 01 | 003 | 4.5 | 3.5 | 2.8 | 3.3 | 3.7 | 4.3 | 5.0 | 5.0 | ... | 9.8 | 10.0 | 9.0 | 7 |
| **2** | 01 | 005 | 9.6 | 7.4 | 4.9 | 4.8 | 5.5 | 7.4 | 7.7 | 7.1 | ... | 14.3 | 12.3 | 11.5 | 1 |
| **3** | 01 | 007 | 7.2 | 7.5 | 7.0 | 6.8 | 5.3 | 6.8 | 7.0 | 6.0 | ... | 13.3 | 11.4 | 10.5 | 8 |
| **4** | 01 | 009 | 3.3 | 3.0 | 3.3 | 2.8 | 3.5 | 3.6 | 5.4 | 4.6 | ... | 10.0 | 9.8 | 8.7 | 6 |
| **5** | 01 | 011 | 14.7 | 10.4 | 10.5 | 10.0 | 8.5 | 9.9 | 10.2 | 11.0 | ... | 15.6 | 11.8 | 11.6 | 1 |
| **6** | 01 | 013 | 9.2 | 9.9 | 11.0 | 13.7 | 7.8 | 7.5 | 7.9 | 8.7 | ... | 16.4 | 13.6 | 12.5 | 1 |
| **7** | 01 | 015 | 6.3 | 5.5 | 4.7 | 5.1 | 5.1 | 5.5 | 6.0 | 6.0 | ... | 11.1 | 11.4 | 10.3 | 8 |
| **8** | 01 | 017 | 5.2 | 4.6 | 4.2 | 4.7 | 4.4 | 5.4 | 6.7 | 6.4 | ... | 19.6 | 14.9 | 12.0 | 9 |

9 rows × 25 columns

**Create region of interest: 100 mi buffer around Kansas, Missouri, Oklahoma, Tennessee, Kentucky, Arkansas**

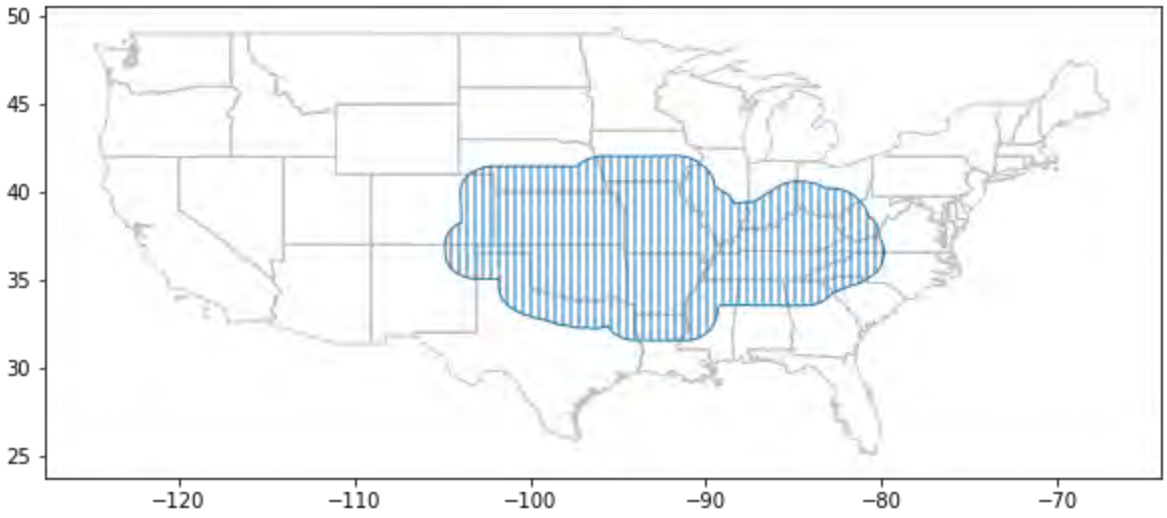
In [6]:
```
cursor.execute("""
select sdo_util.to_wktgeometry(
          sdo_geom.sdo_buffer(
             sdo_aggr_union(sdoaggrtype(c.geom, 0.05)), 100, 0.05, 'unit=MILE'))
 as geometry
FROM states c
WHERE state in ('Kansas','Missouri','Oklahoma','Tennessee','Kentucky','Arkansas')
"""
)
gdf = gpd.GeoDataFrame(cursor.fetchall(), columns = ['geometry'])
gdf['geometry'] = gpd.GeoSeries(gdf['geometry'].apply(lambda x: loads(x)))
gdf['geometry']
```

Out[6]:
```
0    POLYGON ((-79.8391639686706 36.4344685209189, ...
Name: geometry, dtype: object
```

**View the region**

In [7]:
```
fig, ax = plt.subplots(figsize=(10,5))
ax.set_facecolor("white")
us48 = gpd.read_file(pysal.lib.examples.get_path('us48.shp'))
us48.plot(ax=ax, facecolor='none', edgecolor='#c0c0c0')
gdf.plot(ax=ax, facecolor='none', edgecolor='#1f77b4', hatch='|||')
```

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x7f7b993ec0d0>




**Prepare unemployment data for analysis (pivot, add geometry, spatial filter)**

In [8]:
```
cursor.execute("""
WITH
-- pivot the unemployment data
unemp_data as (
 SELECT * FROM
 (select statefips, countyfips, unemp_pct, year from bls_unemployment)
 PIVOT( avg(unemp_pct)
    FOR year IN (1996,1997,1998,1999,2000,2001,2002,2003,
                 2004,2005,2006,2007,2008,2009,2010,2011,
                 2012,2013,2014,2015,2016,2017,2018) )
),
-- define region of interest
aoi as (
select     sdo_geom.sdo_buffer(
             sdo_aggr_union(sdoaggrtype(c.geom, 0.05)), 100, 0.05, 'unit=MILE') a
s geom
FROM states c
WHERE state in ('Kansas','Missouri','Oklahoma','Tennessee','Kentucky','Arkansas')
)
-- add geometry, county/state names, and filter for counties in the region of int
erest
SELECT c.state, c.county, a.*, sdo_util.to_wktgeometry(b.geom) as geometry
FROM unemp_data a, cb_2018_us_county_500k b, fips_county c, aoi
WHERE a.statefips=b.statefp   and a.countyfips=b.countyfp
AND a.statefips=c.state_fips   and a.countyfips=c.county_fips
AND sdo_anyinteract(b.geom,aoi.geom)='TRUE'
""")

gdf = gpd.GeoDataFrame(cursor.fetchall(), columns = ['STATE','COUNTY','STATEFIPS'
,'COUNTYFIPS','1996','1997',
                                                     '1998','1999','2000','2001',
'2002','2003','2004','2005',
                                                     '2006','2010','2007','2008',
'2009','2011','2012','2013',
                                                     '2014','2015','2016','2017',
'2018','geometry'])
gdf['geometry'] = gpd.GeoSeries(gdf['geometry'].apply(lambda x: loads(x)))
gdf.head()
```

Out[8]:

| | STATE | COUNTY | STATEFIPS | COUNTYFIPS | 1996 | 1997 | 1998 | 1999 | 2000 | 2001 | ... | 2009 | 2011 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | Alabama | Blount | 01 | 009 | 3.3 | 3.0 | 3.3 | 2.8 | 3.5 | 3.6 | ... | 9.8 | 8.7 |
| **1** | Alabama | Calhoun | 01 | 015 | 6.3 | 5.5 | 4.7 | 5.1 | 5.1 | 5.5 | ... | 11.4 | 10.3 |
| **2** | Alabama | Cherokee | 01 | 019 | 4.4 | 4.1 | 4.3 | 4.9 | 4.5 | 4.5 | ... | 10.6 | 9.7 |
| **3** | Alabama | Cleburne | 01 | 029 | 4.4 | 3.5 | 3.7 | 4.3 | 4.1 | 5.2 | ... | 10.0 | 9.7 |
| **4** | Alabama | Colbert | 01 | 033 | 7.0 | 7.2 | 8.0 | 7.2 | 5.5 | 6.8 | ... | 11.5 | 10.2 |

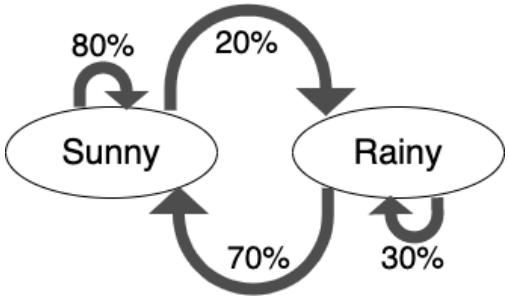
5 rows × 28 columns

# Temporal analysis with Discrete Markov Chains (DMC)

- DMC is only suitable for certain scenarios (https://en.wikipedia.org/wiki/Markov_chain (https://en.wikipedia.org/wiki/Markov_chain))
- Input: Set of observation time series
- Result: Probabilities and predictions of future observation transitions
- PySAL supports **Classic Markov** and 2 spatial variants **Spatial Markov** and **LISA Markov**

**Overview**

- Input: Multiple time series of observations
- Result: Probabilities of subsequent observations
- Classic Hello World Example:




- PySAL supports **Classic Markov** and 2 spatial variants **Spatial Markov** and **LISA Markov**

## Classic Markov

- Input is arrays for each county with unemployment % classified into bins for each year
- PySAL provides classification to bin the data

In [9]:
```
print(np.array(gdf["2004"][0:20]))
print()
print(mc.Quantiles(gdf["2004"],k=5))
print()
print((mc.Quantiles(gdf["2004"],k=5).yb)[0:20])
```

```
[4.2 5.5 4.8 5.3 7.1 5.2 5.8 6.4 6.9 7.  6.9 5.3 8.2 6.4 6.4 5.2 4.7 6.4
 4.9 6.1]

               Quantiles

Lower            Upper              Count
=========================================
         x[i] <=  4.300                255
 4.300 < x[i] <=  5.200                287
 5.200 < x[i] <=  5.900                234
 5.900 < x[i] <=  6.800                241
 6.800 < x[i] <= 13.700                254

[0 2 1 2 4 1 2 3 4 4 4 2 4 3 3 1 1 3 1 3]
```

Generate arrays of binned values, where each array is a year with values for each county

In [10]:
```
binnedData = np.array([mc.Quantiles(gdf[str(y)],k=5).yb for y in range(2004,2018
)])
print(binnedData.shape)
print()
print(binnedData)
```

```
(14, 1271)

[[0 2 1 ... 4 2 3]
 [0 1 1 ... 4 2 2]
 [0 1 1 ... 3 2 3]
 ...
 [2 4 2 ... 4 3 4]
 [3 4 2 ... 4 3 4]
 [1 3 2 ... 4 4 4]]
```

For input to Classic Markov, transpose so that each array is a county with values for each year (time series)

In [11]:
```
binnedDataT = binnedData.transpose()
print(binnedDataT.shape)
print()
print(binnedDataT)
```

```
(1271, 14)

[[0 0 0 ... 2 3 1]
 [2 1 1 ... 4 4 3]
 [1 1 1 ... 2 2 2]
 ...
 [4 4 3 ... 4 4 4]
 [2 2 2 ... 3 3 4]
 [3 2 3 ... 4 4 4]]
```

Create Markov instance and view transition counts

In [12]:
```
m5 = giddy.markov.Markov(binnedDataT)
m5.transitions
```

Out[12]:
```
array([[2946.,  463.,   44.,   10.,    5.],
       [ 483., 2057.,  673.,  123.,   24.],
       [  42.,  718., 1663.,  686.,  117.],
       [  14.,   98.,  740., 1813.,  584.],
       [   3.,   15.,   87.,  638., 2477.]])
```

View transition probabilities

In [13]:
```
print(m5.p)
```

```
[[0.8495 0.1335 0.0127 0.0029 0.0014]
 [0.1438 0.6122 0.2003 0.0366 0.0071]
 [0.013  0.2226 0.5155 0.2126 0.0363]
 [0.0043 0.0302 0.2278 0.558  0.1797]
 [0.0009 0.0047 0.027  0.1981 0.7693]]
```

View steady state distribution

```
In [14]: m5.steady_state
```

```
Out[14]: array([0.2172, 0.2029, 0.1926, 0.1965, 0.1909])
```

View first mean passage time

```
In [15]: print(giddy.ergodic.fmpt(m5.p))
```

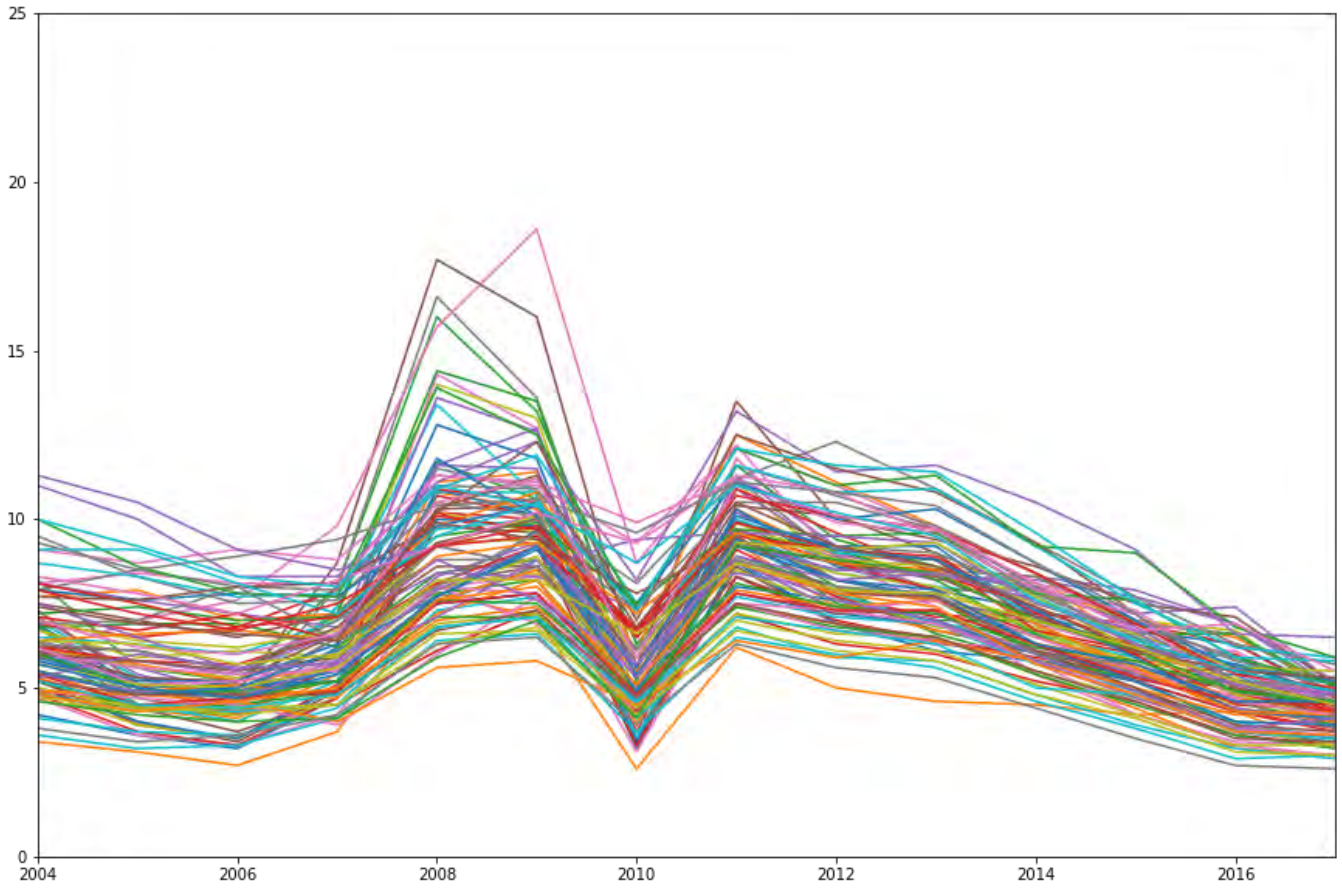
```
[[ 4.6041  7.9945 14.8811 22.1314 35.6067]
 [22.8275  4.9295  9.0061 16.3859 29.9697]
 [31.2434 10.4107  5.1934 10.635  24.3351]
 [35.8927 15.3302  7.5034  5.0902 17.1953]
 [39.2734 18.7488 11.0187  5.9992  5.2373]]
```

## Visualize the data

Absolute values

In [16]:
```
npData = np.array([gdf[str(y)] for y in range(2004,2018)])
years = range(2004,2018)
from pylab import rcParams
rcParams['figure.figsize'] = 15,10
plt.plot(years,npData[:,0:100])
plt.xlim((years[0], years[-1]))
plt.ylim((0, 25))
```

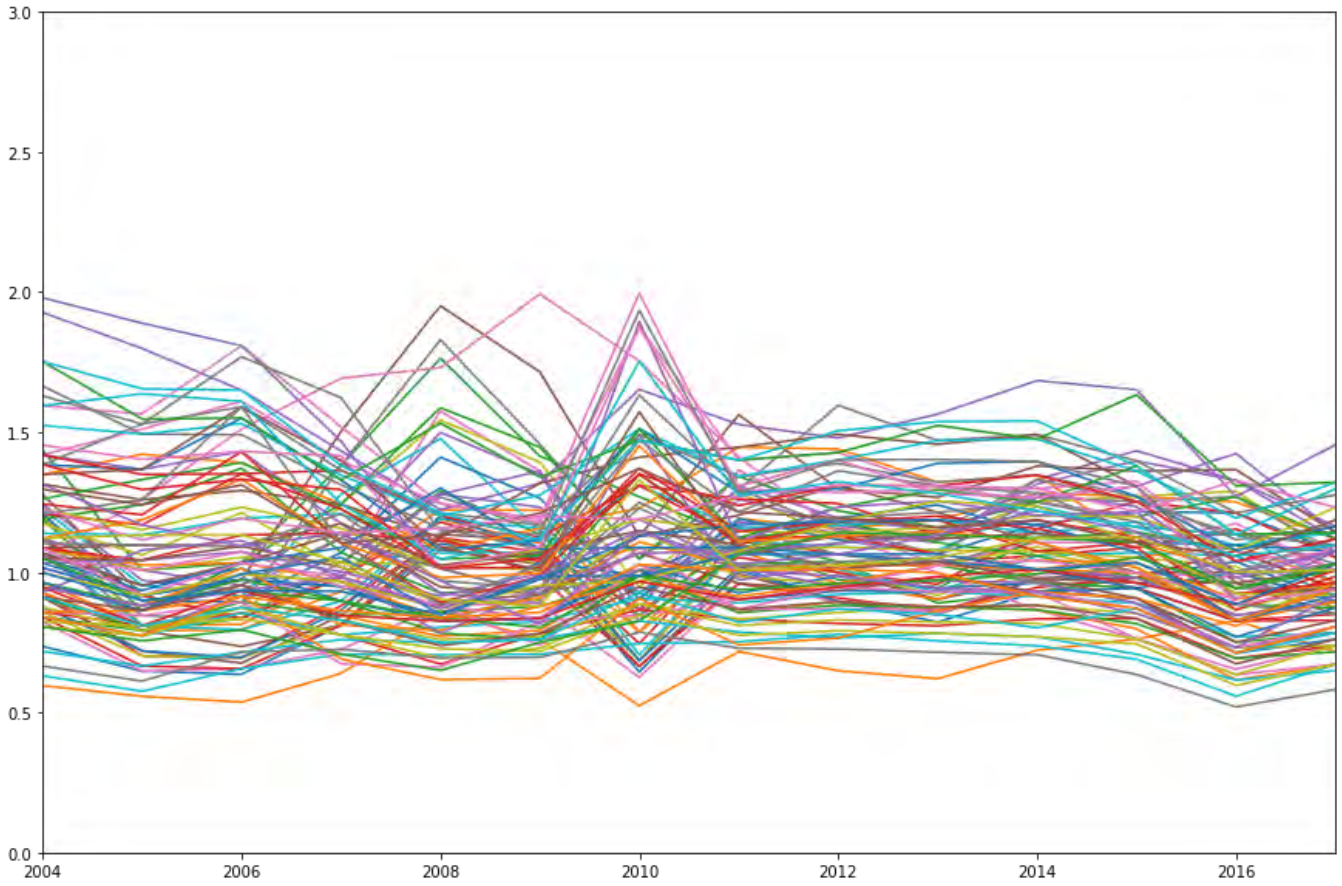
Out[16]: (0, 25)




Relative values

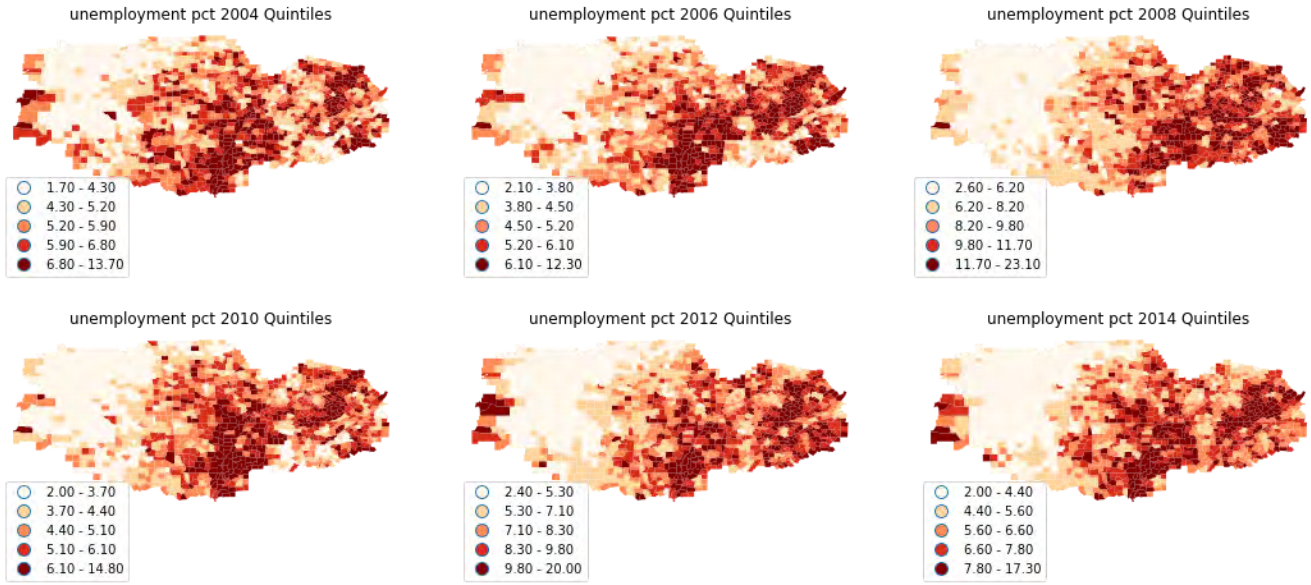
In [17]:
```
years = range(2004,2018)
from pylab import rcParams
rcParams['figure.figsize'] = 15,10
plt.plot(years,(npData.T/npData.mean(axis=1)).T[:,0:100])
plt.xlim((years[0], years[-1]))
plt.ylim((0, 3))
```

Out[17]: (0, 3)




Map viz

In [18]:
```
index_year = range(2004,2018,2)
fig, axes = plt.subplots(nrows=2, ncols=3,figsize = (15,7))
for i in range(2):
    for j in range(3):
        ax = axes[i,j]
        gdf.plot(ax=ax, column=str(index_year[i*3+j]), cmap='OrRd', scheme='quant
iles', legend=True)
        ax.set_title('unemployment pct %s Quintiles'%str(index_year[i*3+j]))
        ax.axis('off')
        leg = ax.get_legend()
        leg.set_bbox_to_anchor((0.18, 0.0, 0.16, 0.2))
plt.tight_layout()
```




## Test for Global Spatial Autocorrelation

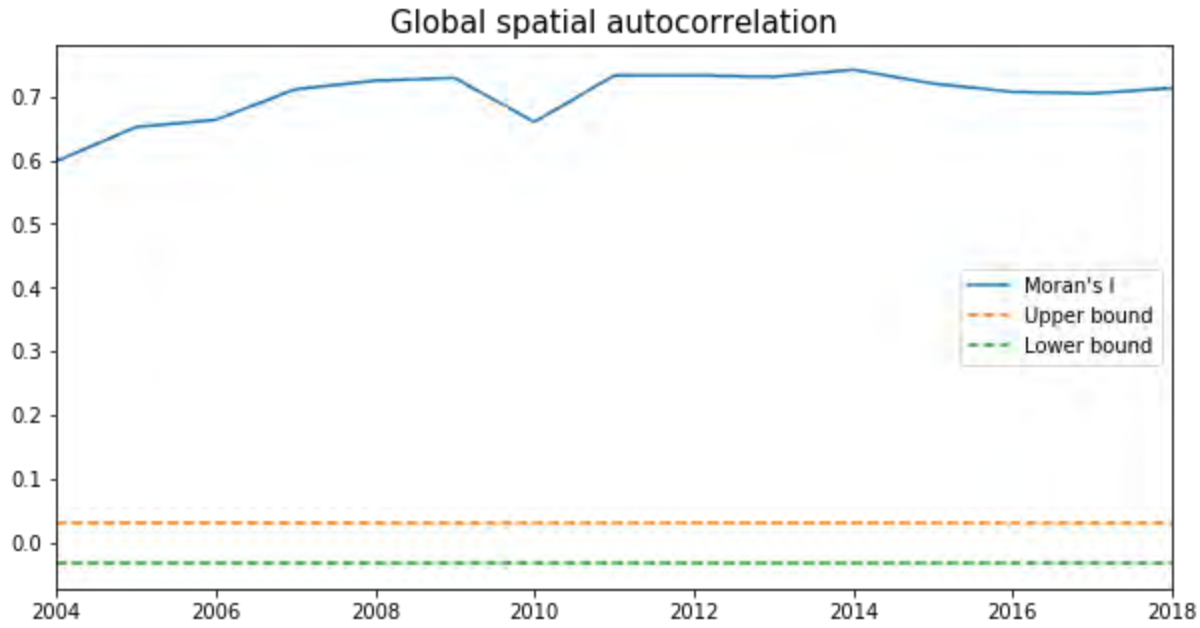
- **spatial weights:** for each county, the set of neighbor counties
- **spatial lag:** for each county, the weighted avg unemployment of it's neighbors
- **Moran I:** measure of spatial autocorrelation (similarity with neighbors)

In [19]:
```
wq = pysal.lib.weights.Queen.from_dataframe(gdf)
wq.transform = 'r'
wq[7]
```

Out[19]:
```
{0: 0.16666666666666666,
 1: 0.16666666666666666,
 2: 0.16666666666666666,
 18: 0.16666666666666666,
 20: 0.16666666666666666,
 6: 0.16666666666666666}
```

In [20]:
```
years = np.arange(2004,2019)
mitest = [Moran(gdf[str(x)], wq) for x in years]
res = np.array([(mi.I, mi.EI, mi.seI_norm, mi.sim[974]) for mi in mitest])
fig, ax = plt.subplots(nrows=1, ncols=1,figsize = (10,5) )
ax.plot(years, res[:,0], label='Moran\'s I')
ax.plot(years, res[:,1]+1.96*res[:,2], label='Upper bound',linestyle='dashed')
ax.plot(years, res[:,1]-1.96*res[:,2], label='Lower bound',linestyle='dashed')
ax.set_title("Global spatial autocorrelation",fontdict={'fontsize':15})
ax.set_xlim([2004, 2018])
ax.legend()
```

Out[20]:
```
<matplotlib.legend.Legend at 0x7f7b9916e2d0>
```




## Spatial Markov

- Markov for regional context, using classification (bins) of Spatial lag
- Spatial Markov classifies the lag and observations for us
- i.e. Markov for counties with high neighboring unemployment

In [21]:
```
sm = giddy.markov.Spatial_Markov(npData.transpose(), wq, fixed = True, k = 5,m=5)
print(npData.transpose().shape)
print()
print(sm.cutoffs)
print()
print(sm.lag_cutoffs)
print()
print(sm.classes)
```

```
(1271, 14)

[4.2 5.3 6.6 8.6]

[4.35   5.3667 6.58   8.58  ]

[0 1 2 3 4]
```

Transition probabilities per spatial lag bin

In [22]:
```
sm.summary()
```

```
------------------------------------------------------------
                    Spatial Markov Test
------------------------------------------------------------
Number of classes: 5
Number of transitions: 16523
Number of regimes: 5
Regime names: LAG0, LAG1, LAG2, LAG3, LAG4
------------------------------------------------------------
   Test                  LR               Chi-2
  Stat.             651.129             673.265
    DOF                  72                  72
p-value                0.000               0.000
------------------------------------------------------------
P(H0)          C0         C1         C2         C3         C4
     C0      0.726      0.127      0.077      0.054      0.016
     C1      0.291      0.390      0.117      0.123      0.079
     C2      0.042      0.329      0.359      0.114      0.155
     C3      0.039      0.053      0.303      0.417      0.188
     C4      0.023      0.078      0.078      0.244      0.576
------------------------------------------------------------
P(LAG0)        C0         C1         C2         C3         C4
     C0      0.770      0.120      0.066      0.034      0.011
     C1      0.447      0.280      0.096      0.120      0.057
     C2      0.110      0.329      0.315      0.178      0.068
     C3      0.077      0.154      0.462      0.154      0.154
     C4      0.000      0.000      0.000      0.000      0.000
------------------------------------------------------------
P(LAG1)        C0         C1         C2         C3         C4
     C0      0.599      0.148      0.104      0.117      0.033
     C1      0.270      0.388      0.127      0.131      0.084
     C2      0.035      0.393      0.258      0.118      0.197
     C3      0.051      0.082      0.378      0.306      0.184
     C4      0.000      0.000      0.111      0.444      0.444
------------------------------------------------------------
P(LAG2)        C0         C1         C2         C3         C4
     C0      0.609      0.155      0.136      0.091      0.009
     C1      0.257      0.442      0.112      0.102      0.087
     C2      0.040      0.349      0.363      0.099      0.149
     C3      0.007      0.055      0.361      0.347      0.230
     C4      0.016      0.065      0.048      0.387      0.484
------------------------------------------------------------
P(LAG3)        C0         C1         C2         C3         C4
     C0      0.500      0.333      0.167      0.000      0.000
     C1      0.170      0.491      0.104      0.179      0.057
     C2      0.042      0.257      0.434      0.125      0.143
     C3      0.032      0.040      0.313      0.424      0.191
     C4      0.016      0.034      0.030      0.423      0.496
------------------------------------------------------------
P(LAG4)        C0         C1         C2         C3         C4
     C0      0.000      0.000      0.000      0.000      0.000
     C1      0.000      1.000      0.000      0.000      0.000
     C2      0.105      0.158      0.395      0.211      0.132
     C3      0.088      0.082      0.201      0.491      0.139
     C4      0.025      0.089      0.090      0.197      0.598
------------------------------------------------------------
```

From the summary above, the probability of a high unemployment county remaining high is ~60% if its neighbiors are in the 5th quintile, and ~50% if its neighbors are in the 4th quintile.

Spatially conditional first mean passage times per spatial lag bin

In [23]:
```
print(sm.F)
```

```
[[[ 1.8164  6.6329 10.5272 15.5179 25.0656]
  [ 3.0172  5.427   9.6416 13.7857 22.9675]
  [ 4.4597  4.5931  7.017  12.0374 21.4539]
  [ 4.7812  5.51    5.7342 12.1919 19.2924]
  [ 3.6541  6.9632 10.3868 15.3447 24.5934]]

 [[ 5.4579  6.369   6.1909  6.4034 10.3668]
  [ 8.7361  4.9012  5.8694  6.0232  9.5586]
  [11.7555  4.8698  4.9459  5.6106  8.1261]
  [12.6411  6.7157  3.8864  4.4734  7.7496]
  [14.264   8.1465  4.9091  2.9221  5.3472]]

 [[ 5.6254  6.1034  6.2361  8.0982 11.6294]
  [ 9.3706  4.532   6.4071  7.5312 10.3762]
  [12.5385  4.9063  4.8276  7.0931  9.1739]
  [14.6174  7.0942  4.3067  4.907   7.606 ]
  [15.2473  7.9089  6.1633  3.7969  5.2452]]

 [[ 8.6109  4.171   6.3057  8.1284 12.5284]
  [14.2024  4.52    6.4586  6.2197 11.0282]
  [17.2607  6.513   4.4549  5.9459  9.5289]
  [18.6021  8.7914  4.7202  3.9457  8.3786]
  [19.6032  9.8932  6.596   3.0248  5.4136]]

 [[ 1.      2.      2.3216  3.0276  3.4811]
  [ 1.      1.      2.3216  3.0276  3.4811]
  [ 0.5972  5.8334  1.      1.5809  2.2215]
  [ 0.5696  6.5573  1.0982  1.      2.0354]
  [ 0.6357  7.142   1.1997  1.2172  1.    ]]]
```

This tells us that a county in the 1st quintile with neighbors in the 5th quintile (ie 1st row in 5th matrix) will enter the 5th quintile after 3.5 years. If that county has neighbors in the 4th quintile (ie 1st row in 4th matrix) it will enter the 4th quintile after 8.1 years.

## LISA Markov

- LISA = Local Indicators of Spatial Autocorrelation
- LISA Markov analyzes joint changes in observations and their spatial lag (neighboring observations)
- instead of unemplyment % bins, the classes are Moran quadrants:
  - 1 = HH-high values surrounded by high values (hotspot)
  - 2 = LH-low values surrounded by high values
  - 3 = LL-low values surrounded by low values (coldspot)
  - 4 = HL-high values surrounded by low values

In [24]:
```
lm = giddy.markov.LISA_Markov(npData.transpose(), wq)
print(lm.classes)
```

```
[1 2 3 4]
```

In [25]:
```
print(lm.transitions)
```

```
[[5635.  369.  267.  314.]
 [ 380. 1040.  375.    8.]
 [ 269.  329. 6128.  276.]
 [ 283.    9.  308.  533.]]
```

In [26]:
```
print(lm.p)
```

```
[[0.8557 0.056  0.0405 0.0477]
 [0.2108 0.5768 0.208  0.0044]
 [0.0384 0.047  0.8752 0.0394]
 [0.2498 0.0079 0.2718 0.4704]]
```

In [27]:
```
print(lm.steady_state)
```

```
[0.3856 0.1017 0.4441 0.0686]
```

In [28]:
```
print(giddy.ergodic.fmpt(lm.p))
```

```
[[ 2.5934 20.3638 13.1432 25.0193]
 [ 9.6756  9.8369  8.995  27.6636]
 [14.6746 21.3732  2.2517 26.1253]
 [ 9.5664 22.4648  8.2224 14.5711]]
```