Predicting Local Experience after the Great Recession

Kyle Alden -- DAT8

1. Background

Localities experienced the Great Recession in vastly different ways. Some economies quickly returned to their pre-recession health, while others have yet to fully regain their footing 6 years after the official end of the recession.

As of 2014 only 15% of counties in the United States had returned their pre-recession unemployment rate, while only 56% percent had returned to within 1% of pre-recession unemployment. Over 30% of counties still saw unemployment over 2% higher than the average rate for 2007.

2. Problem Statement

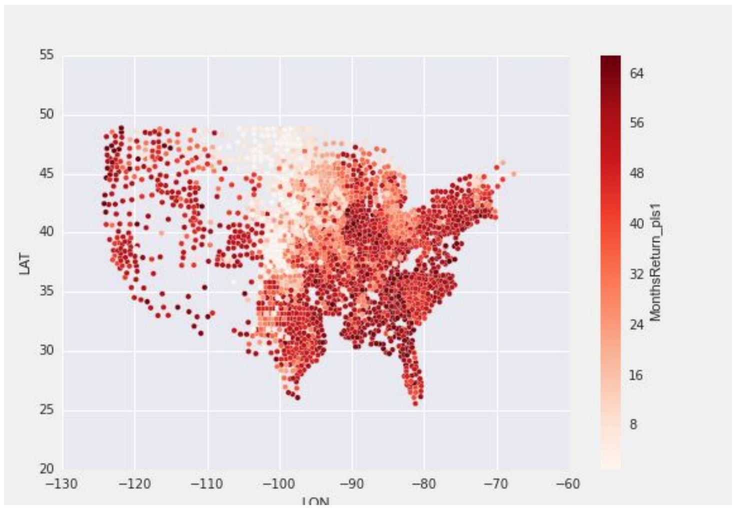
Can I predict whether any county in the US to return to within 1% of its pre-recession unemployment rate (average rate in 2007)?

Also, for counties that have returned can we predict how many months it will take?i.e. What types of counties are successful in America's post recession economy?

2.1 Initial Hypotheses:

1. Rural, suburban and urban counties will experience the recession differently
2. Local industry matters: counties that depend on farming and energy production probably fared better than those dependant on the service and government sectors.
3. Seasonality probably matters

Months until return within 1% of pre-recession unemployment



3. Response Variable Development

3.1 Data Source

Data was provided as a flat file of monthly unemployment rates from the Bureau of Labor statistics. The file was comprised of a series_id (every geographic area, with a code for data type), year, period(month), footnotes, and value--estimated unemployment rate, labor force, or number of unemployed individuals. Original data is available at http://download.bls.gov/pub/time.series/la/. The "la.data.64.County" file at the link contains the entire historical series for all counties it was not included in my github repository because of the 100mb limit.

3.2 Response Variable Engineering

The original data had a shape of (4272544, 6). We only needed data from after 2007 and that was at the county level. This was accomplished using boolean operations in a pandas dataframe where we selected out all features that were from after 2006 and their series_id ended with code "3". This left a dataframe with a shape of (357231, 6).

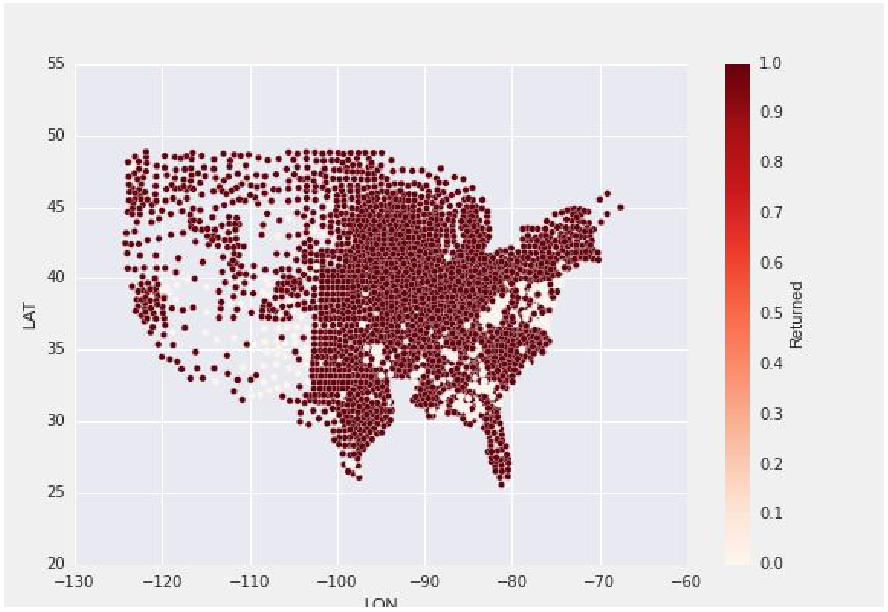
Next, in order to create a dataframe where I could easily calculate the average 2007 unemployment rate and when a county returned to that rate, I needed a dataframe where each row represents a county and each column represents the unemployment rate of that county during that month. I calculated a YearMonth value that would be the by concatenating in pandas. I had some issues creating a pivot table in pandas, so to expedite the process I saved the data as a csv and used google sheets to further develop my variables.

In Sheets, I created a pivot table, found the average unemployment rate for 2007 for each county by using =average(). I then calculated the month when a county's unemployment rate dipped below the average using, =INDEX(AM$1:DN$1,MATCH(TRUE,INDEX(AM4:DN4<C4),0)) to get the column name (YearMonth) of each month. I found several anomalies where a county's unemployment rate would drop and then jump back up, so I chose to repeat the process on a 3 month moving average of the rate.

3.3.1 Binary Classification Problem Response Variable Engineering

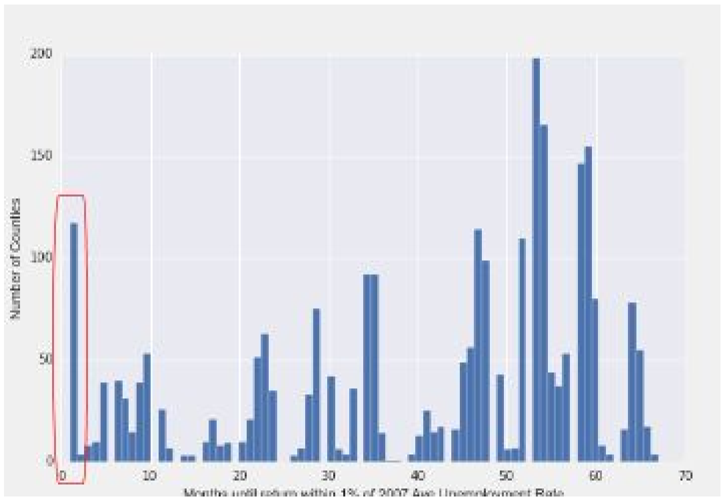
In order to answer the question: Has a county returned to within one percent of their pre-recession unemployment rate? I used the months to return to pre-recession unemployment variable to calculate a binary variable if a county returned to within one percent of its pre-recession unemployment rate or not. We gave a value of 1 to the counties that recovered and 0 to those that didn't.

Counties that returned to within 1% of 2007 unemployment rate



3.3.2 Multivariate Classification Problem Response Variable Engineering
Counties that returned to their pre-recession unemployment rate (or never really jumped during the recession) became a clear third set of counties that emerged during my data exploration. See:

Months until a county returned to pre-recession unemployment rate



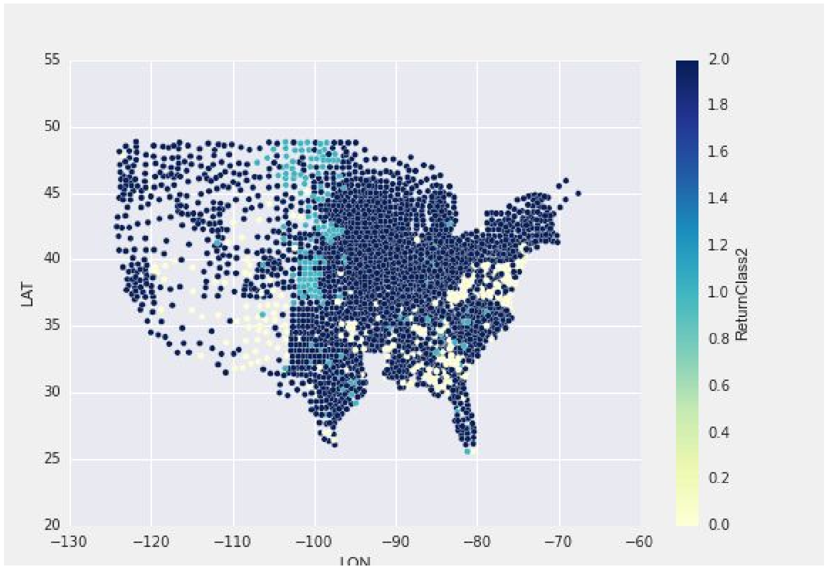
As such, I decided that a third class was worth attempting to model before I modeled the number of months it would take for any county to return to its pre-recession unemployment rate.

County Classes

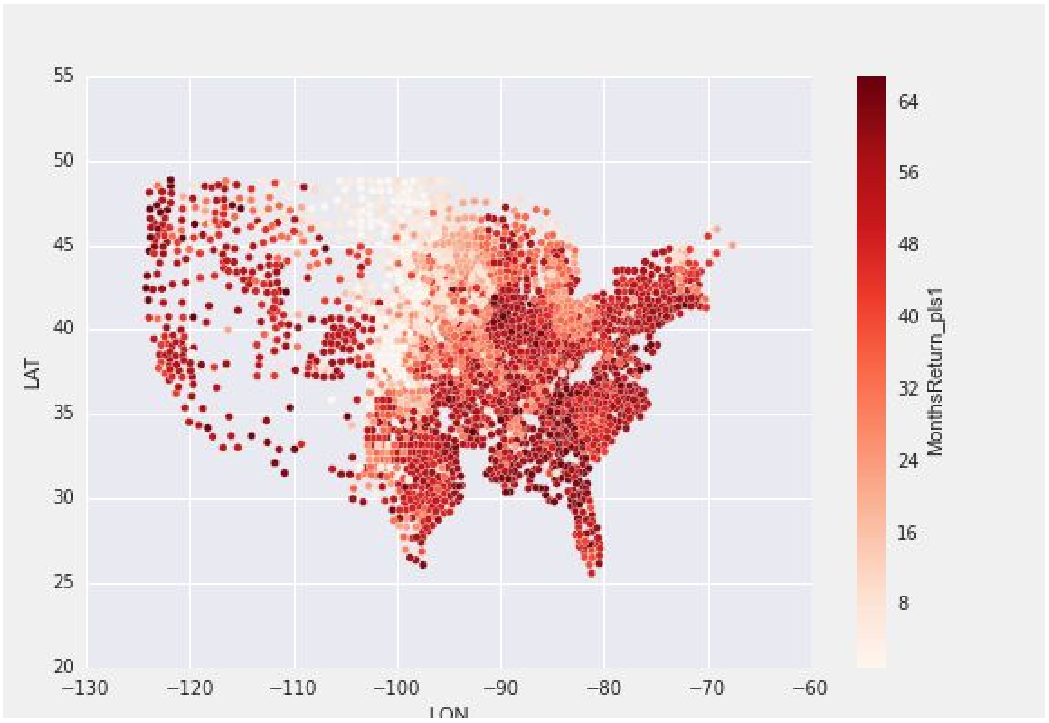
0 = Have yet to returned to within 1% of pre-recession unemployment

1 = returned to within 1% of pre-recession unemployment on or before Jan 2010

2 = returned to within 1% of pre-recession unemployment on or before Jul 2015



3.3.3 Multivariate Classification Problem Response Variable Engineering
In order to answer my core question--how many months will it take until a county returned to within 1% of its pre-recession unemployment rate-- I used the result of the INDEX operation--the column name representing the date--I then calculated the number of months past 1 January 2010 (the beginning of the recovery for the purposes of this study). This was then coded as 'MonthsReturn_pls1'.



4. Feature Variables: Engineering and Exploration

4.1 Feature Joining Process

The US Government uses a five-digit Federal Information Processing Standard (FIPS) code (FIPS 6-4) which uniquely identifies counties and county equivalents in the United State. These codes were available in our Series_id variable downloaded from the Bureau of Labor Statistics. In order to join each of the following variables to our dataset I used the VLOOKUP operation in google sheets. Using a GUI for this data helped me quickly check my data.

4.1.1 Demographic / Socio-economic Features

These variables were downloaded from the US Department of Agriculture at http://www.ers.usda.gov/data-products/county-level-data-sets/download-data.aspx

Rural_urban_continuum_code_2013	The 2013 Rural-Urban Continuum Codes form a classification scheme that distinguishes metropolitan counties by the population size of their metro area, and nonmetropolitan counties by degree of urbanization and adjacency to a metro area. The official Office of Management and Budget (OMB) metro and nonmetro categories have been subdivided into three metro and six nonmetro categories. Each county in the U.S. is assigned one of the 9 codes. This scheme allows researchers to break county data into finer residential groups, beyond metro and nonmetro, particularly for the analysis of trends in nonmetro areas that are related to population density and metro influence. The Rural-Urban Continuum Codes were originally developed in 1974. They have been updated each decennial since (1983, 1993, 2003, 2013), and slightly revised in 1988. Note that the 2013 Rural-Urban Continuum Codes are not directly comparable with the codes prior to 2000 because of the new methodology used in developing the 2000 metropolitan areas. See the Documentation for details and a map of the codes.

Urban_influence_code_2013	The 2013 Urban Influence Codes form a classification scheme that distinguishes metropolitan counties by population size of their metro area, and nonmetropolitan counties by size of the largest city or town and proximity to metro and micropolitan areas. The standard Office of Management and Budget (OMB) metro and nonmetro categories have been subdivided into two metro and 10 nonmetro categories, resulting in a 12-part county classification. This scheme was originally developed in 1993. This scheme allows researchers to break county data into finer residential groups, beyond metro and nonmetro, particularly for the analysis of trends in nonmetro areas that are related to population density and metro influence.
Civilian_labor_force_2011	2013 civilian labor force estimate from US census
Median_Household_Income_2013	2013 Median household income estimate US census
Median_Household_Income_Percent_of_State_Total_2013	2013 Percent of State total for Median household income estimate US census
Pct_less_than_hs_2009_2013	Percent of adults who did not graduate college 2009-2013 estimate from census
Pct_HS_Grad_only_2009_2013	Percent of adults who only graduated high school 2009-2013 estimate from census
Pct_SomeCollege_2009_2013	Percent of adults who attended some college 2009-2013 estimate from census
Pct_college_grad_2009_2013	Percent of adults graduated college or more 2009-2013 estimate from census
Net_Mig_2010_2014	Calculated from Census net migration data from 2010 to 2014
MigRate_2010_2014	Calculated from Census net migration data from 2010 to 2014
PctBlack	Downloaded from Census

4.1.2 Economic Variables

Data on the number and types of business establishments was downloaded from the US census and data for the amount of stimulus each county received was downloaded from a propublica research project at https://projects.propublica.org/recovery/.

Total_Business_Establishments_2011	2011 census of busines (SUSB) estimate of total number of establishments by county
Business_Estab_under_20_emp	2011 census of busines (SUSB) estimate of number of establishments with under 20employees by county
Business_Estab_20to99_emp	2011 census of busines (SUSB) estimate of number of establishments with 20 - 99 employees by county
Business_Estab_100to499_emp	2011 census of busines (SUSB) estimate of number of establishments with 100 - 499 employees by county
Business_Estab_over500_emp	2011 census of busines (SUSB) estimate of number of establishments with over 500 employees by county
Stimulus	- $$ in stimulus recieved by each county in US, pulled data from individual state pages on https://projects.propublica.org/recovery/
StimulusPerCapita	-- calculated from data pulled from state pages on https://projects.propublica.org/recovery/
Stim_PC_Limit	because of very large outliers in data I created a feature which limited the Stimulus Per Capita to $10,000 per person

4.1.3 Industry Typology Codes

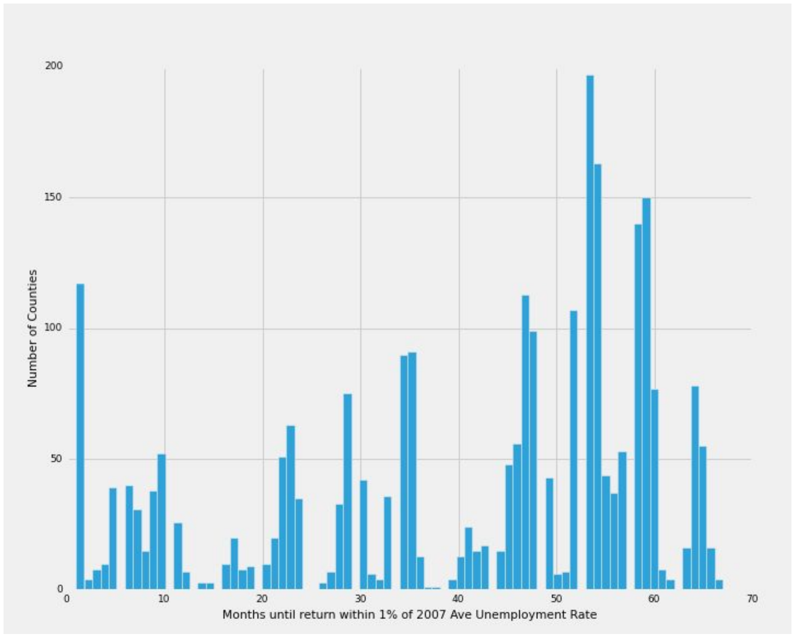
Industrial Type Codes:
http://www.ers.usda.gov/data-products/county-typology-codes.aspx

“The 2004 County Typology codes classify all U.S. counties according to six non-overlapping categories of economic dependence and seven overlapping categories of policy-relevant themes. The economic types include farming, mining, manufacturing, services, Federal/State government, and unspecialized counties. The policy types include housing stress, low education, low employment, persistent poverty, population loss, nonmetro recreation, and retirement destination. In addition, a code identifying counties with persistent child poverty is available.”

farm	Farm-dependent county indicator. 0=no 1=yes
mine	Mining-dependent county indicator. 0=no 1=yes
manf	Manufacturing-dependent county indicator. 0=no 1=yes
fsgov	Federal/State government-dependent county indicator. 0=no 1=yes
serv	Services-dependent county indicator. 0=no 1=yes
nonsp	Nonspecialized-dependent county indicator. 0=no 1=yes
house	Housing stress county indicator. 0=no 1=yes
loweduc	Low-education county indicator. 0=no 1=yes
lowemp	Low-employment county indicator. 0=no 1=yes
perpov	Persistent poverty county indicator. 0=no 1=yes
poploss	Population loss county indicator. 0=no 1=yes
rec	Nonmetro recreation county indicator. 0=no 1=yes
retire	Retirement destination county indicator. 0=no 1=yes
perchldpov	Persistent child poverty county indicator (0=no 1=yes). This code identifies counties in which the poverty rate for related children under 18 years old was 20% or more in 1970, 1980, 1990, and 2000. This indicator was not part of the original ERS county typology codes and was added to this file after several requests. There are a total of 734 persistent child poverty counties. Of the 734 persistent child poverty counties, 603 are nonmetro counties, 131 are metro. The persistent child poverty codes were added March 17, 2009.

4.1.4 Seasonal Feature Engineering

While exploring my data I noticed a seasonal trend in my data, with more counties returning to pre-recession rate during certain months. See graph:



Summer678	calculated a seasonal binary code for summer based on a county returning to within 1% of its prerecession unemployment rate in June, July, or August
Spring345	calculated a seasonal binary code for summer based on a county returning to within 1% of its prerecession unemployment rate in March, April, or May
Fall91011	calculated a seasonal binary code for summer based on a county returning to within 1% of its prerecession unemployment rate in September, October, or November
HolidaySeason	calculated a seasonal binary code for summer based on a county returning to within 1% of its prerecession unemployment rate in November or December

January	Counties that returned to within 1% in January
February	Counties that returned to within 1% in February
March	Counties that returned to within 1% in March
April	Counties that returned to within 1% in April
May	Counties that returned to within 1% in May
June	Counties that returned to within 1% in June
July	Counties that returned to within 1% in July
August	Counties that returned to within 1% in August

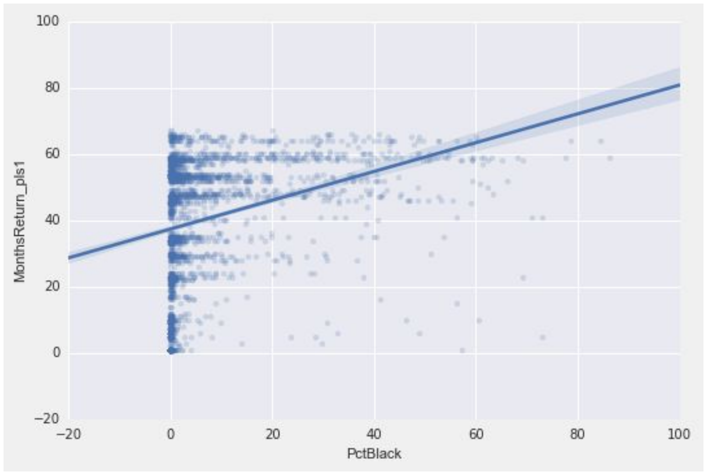
September	Counties that returned to within 1% in September
October	Counties that returned to within 1% in October
November	Counties that returned to within 1% in November
December	Counties that returned to within 1% in December

4.2 Feature Exploration

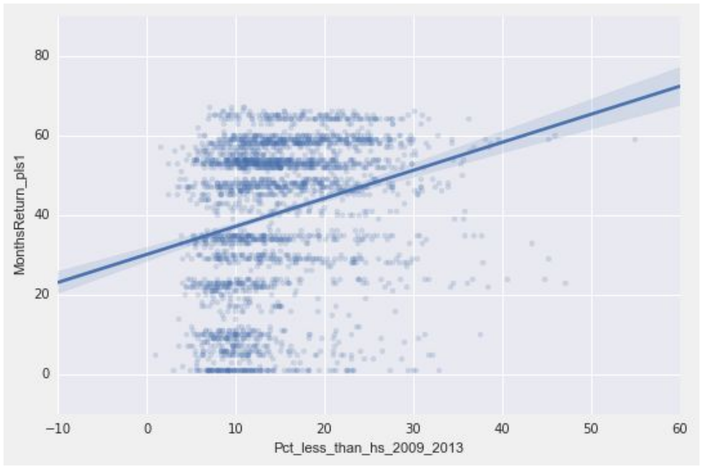
Using seaborn and matplotlib I created several scatterplots, map scatterplots, boxplots, histograms and a correlation matrix in order to look at the variables in question.

4.2.1 Scatterplots

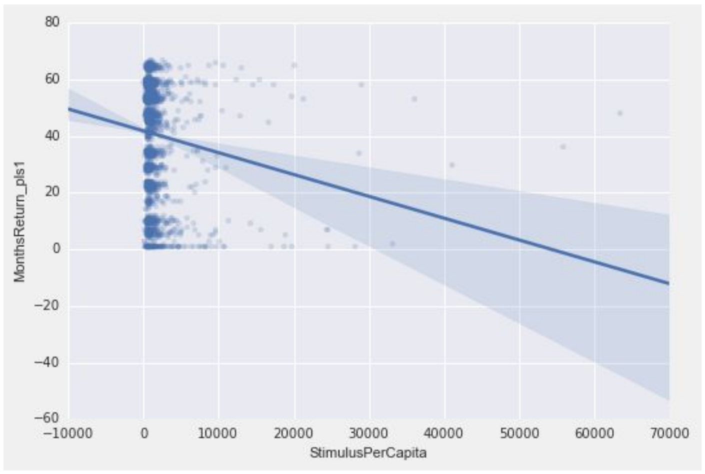
Percent Black -- Positive relationship between percent African American and the number of months until a county returned to its pre-recession unemployment rate.



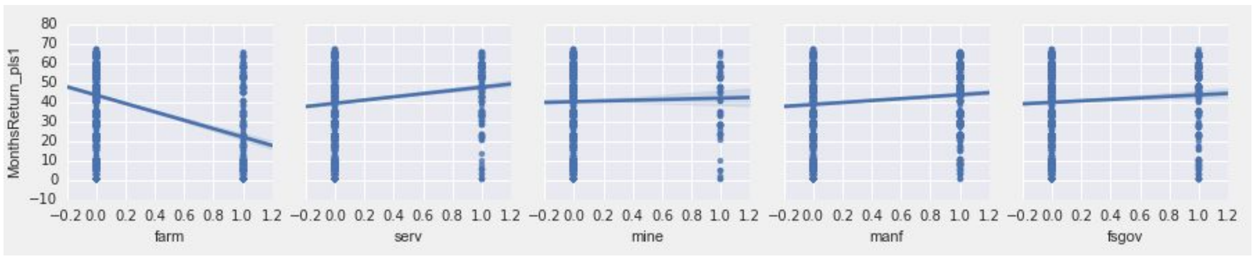
Percent that did not graduate from high school -- Positive relationship between percent that did not graduate from high school and the number of months until a county returned to its pre-recession unemployment rate.



Stimulus Per Capita -- Negative relationship between per capita stimulus and the number of months until a county returned to its pre-recession unemployment rate, but with significant outliers.

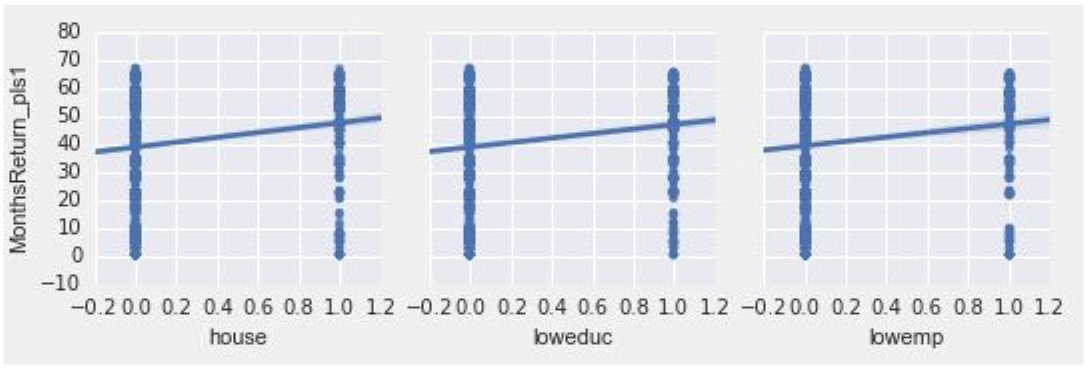


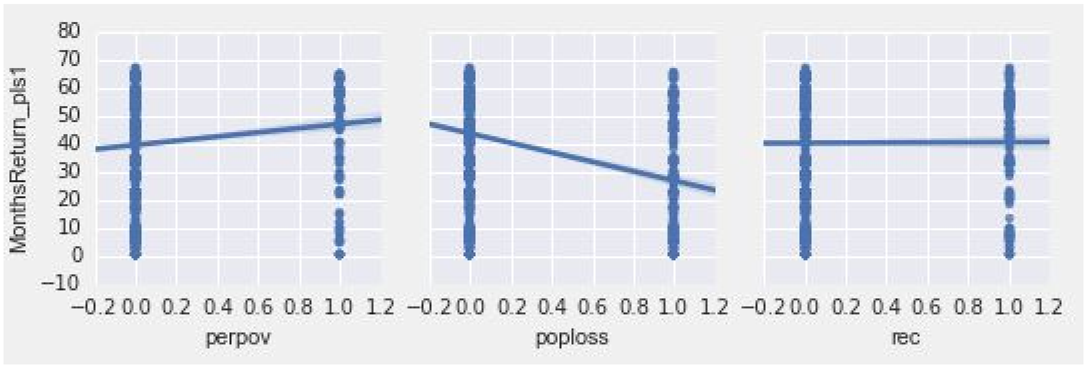
Industrial Type Codes -- Positive relationship between service based economies and the number of months until a county returned to its pre-recession unemployment rate, negative relationship between farm-based economies and

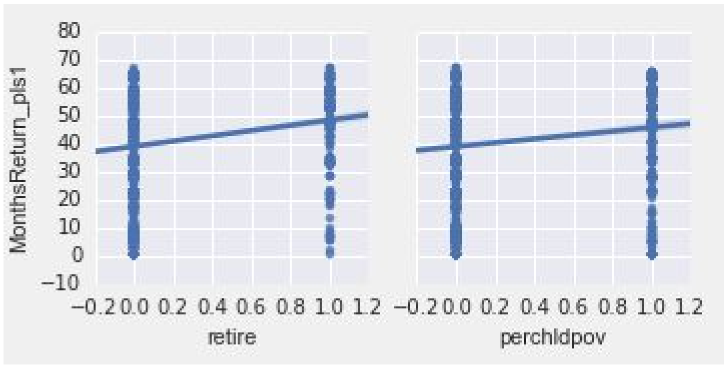


Policy County Codes --

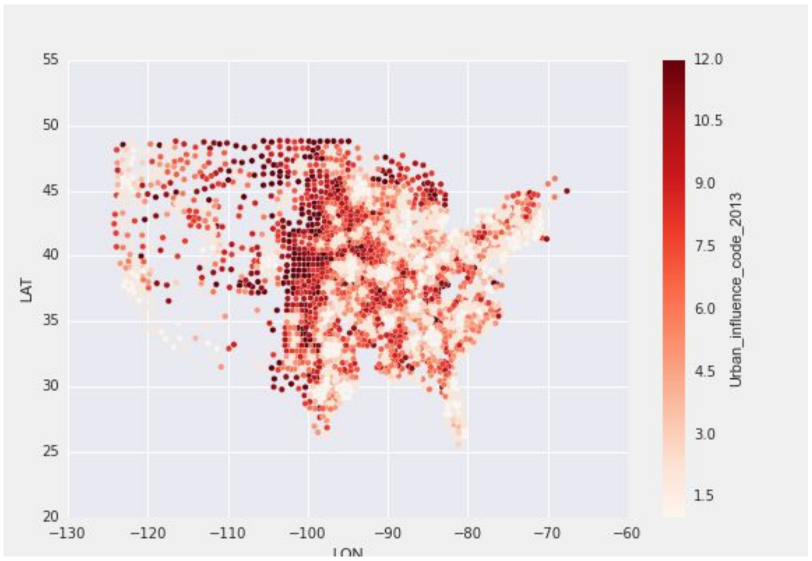
Many of the policy codes had useful differences.



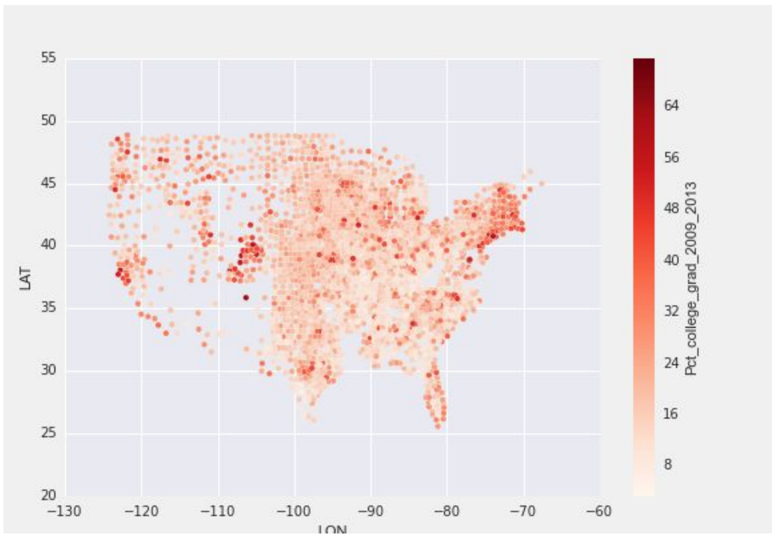




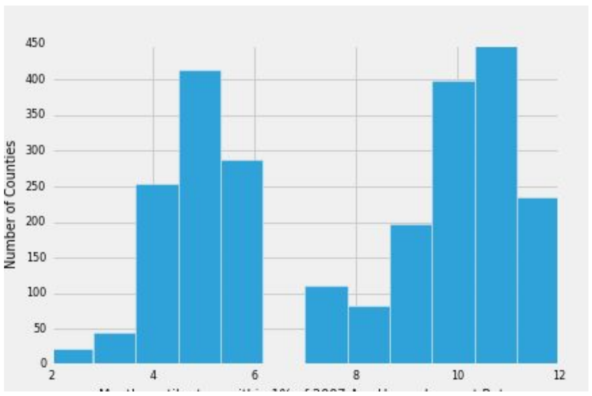
Urban Influence Code -- Urban influence code helps to capture the success that counties in the middle of the country had in returning to their pre-recession unemployment rates.



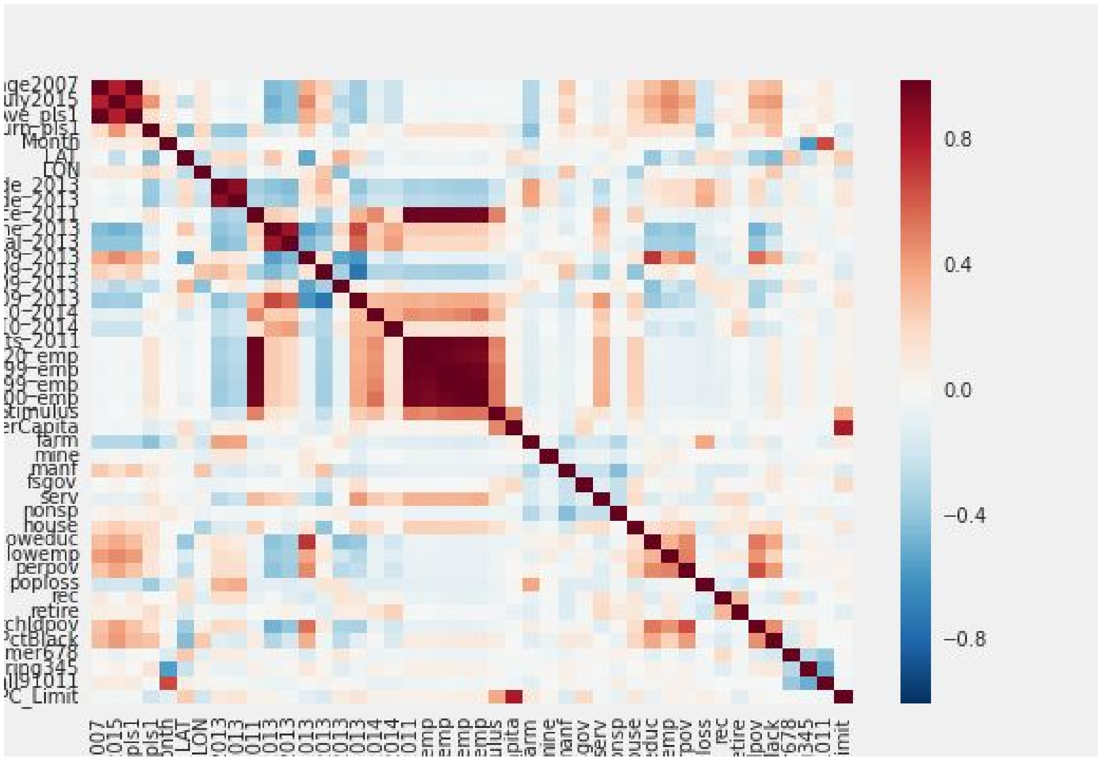
Pct College Grad -- Percent College Grad influence code helps to capture the success that counties in suburban counties.



Monthly Returns (without first January)



Example correlation matrix heatmap
The correlation matrix helped me the most in picking features, as it helped identify the variables that we're most correlated with our response variables.



5. Modeling

5.1 Classification Modeling
In an attempt to explore several options for trying to predict which counties have returned to within 1% of their pre-recession unemployment rate--and in order to practice the models we learned in class--I ran knn, logistic regression, decision trees, and Naive Bayes classification approaches against my dataset with different sets of features. I first ran them on my binary classification problem (returned or not) and then on my 3 class problem (returned before 2010, returned between 2010 and July 2015, and have yet to return)

5.2.1 Binary Classification Modeling
First I computed my null accuracy using Train Test Split and y_test.value_counts().head(1) / len(y_test)

Null Accuracy for this dataset = **.87878**

I produced 3 models for KNN, Logistic Regression, Decision Trees, and Gaussian Naive Bayes in order to explore d each of the below modeling approaches I tried three different sets of features

A. Includes a list of features that I believe through domain knowledge could be good predictors before = ['serv', 'farm', 'LAT', 'LON', 'poploss', 'PctBlack', 'July2015', 'Pct_SomeCollege_2009_2013', 'StimPC_Limit', 'Median_Household_Income_2013']
B. A smaller list of these features -- feature_cols = ['serv', 'farm', 'LAT', 'LON', 'OilRegion', 'coalBinay']
C. All Features -- feature_cols= ['LAT', 'LON', 'Rural_urban_continuum_code_2013', 'Urban_influence_code_2013', 'Civilian_labor_force_2011', 'Median_Household_Income_2013', 'Median_Household_Income_Percent_of_State_Total_2013', 'Pct_less_than_hs_2009_2013', 'Pct_HS_Grad_only_2009_2013', 'Pct_SomeCollege_2009_2013', 'Pct_college_grad_2009_2013', 'Net_Mig_2010_2014', 'MigRate_2010_2014', 'Business_Estab_under_20_emp', 'Business_Estab_20to99_emp', 'Business_Estab_100to499_emp', 'Business_Estab_over500_emp', 'Stimulus', 'StimulusPerCapita', 'farm', 'mine', 'manf', 'fsgov', 'serv', 'nonsp', 'house', 'loweduc', 'lowemp', 'perpov', 'poploss', 'rec', 'retire', 'perchldpov', 'PctBlack', 'StimPC_Limit', 'StimPC_Limit2', 'OilRegion', 'Bakken', 'EagleFord', 'Marcellus', 'CoalPts', 'coalBinay']

5.2.1.1 KNN

I computed the Accuracy, AUC and a confusion matrix for each set of features. My best attempt was the limited list of features (B) listed above using a value of 13 neighbors. The accuracy for that model was .899, about .02 higher than the null model. The confusion matrix was as follows.

True Positives: 648
True Negatives: 35
False Positives: 57
False Negatives: 1

Example Code:

```
X = cnty_data_notnull_cl[feature_cols]
y = cnty_data_notnull_cl.Returned
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
knn = KNeighborsClassifier(n_neighbors=13)

knn.fit(X_train, y_train)
y_pred_class = knn.predict(X_test)
```

```
# compute classification accuracy
metrics.accuracy_score(y_test, y_pred_class)
# compute AUC
metrics.roc_auc_score(y_test, y_pred_class)
# compute Confusion Matrix
metrics.confusion_matrix(y_test,y_pred_class)
```

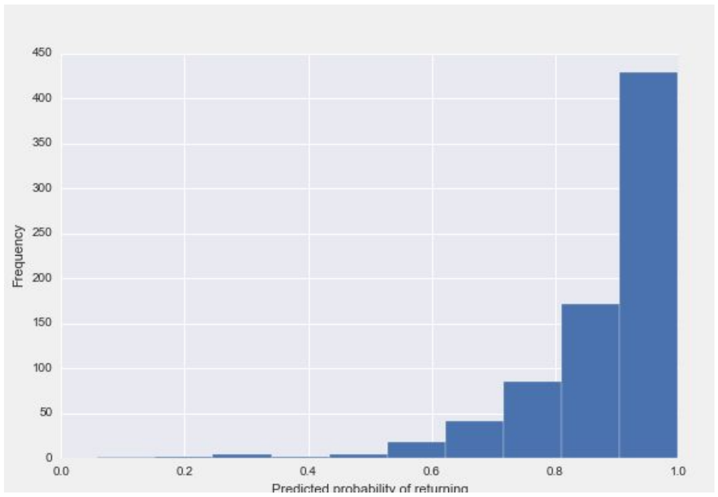
5.2.1.2 Logistic Regression

I computed the Accuracy, AUC and a confusion matrix for each set of features. My best attempt was the domain knowledge list of features (A) listed above. The accuracy for that model was .872, worse than the null model. The confusion matrix was as follows:

True Positives: 661
True Negatives: 1
False Positives: 91
False Negatives: 6

Each of the models was highly skewed toward positive results.

Predicted probability of returning to within 1% of pre-recession unemp rate



I was able to tweak the confusion matrix to make it a little less skewed toward positives by changing the threshold, but wasn't able to push it above the null accuracy.

Example Code:

```
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics

## Define AUC function
def train_test_auc_score(feature_cols):
    X = cnty_data_notnull_cl[feature_cols]
    y = cnty_data_notnull_cl.Returned
    X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
    logreg = LogisticRegression(C=1e9)
    logreg.fit(X_train, y_train)
    y_pred_prob = logreg.predict_proba(X_test)[:,1]
    return metrics.roc_auc_score(y_test, y_pred_prob)

## define confusion matrix function
def train_test_confu_matrix(feature_cols):
    X = cnty_data_notnull_cl[feature_cols]
    y = cnty_data_notnull_cl.Returned
    X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
    logreg = LogisticRegression(C=1e9)
    logreg.fit(X_train, y_train)
    y_pred_class = logreg.predict(X_test)
    return metrics.confusion_matrix(y_test, y_pred_class)

## define accuracy function
def train_test_accuracy_score(feature_cols):
    X = cnty_data_notnull_cl[feature_cols]
    y = cnty_data_notnull_cl.Returned
    X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
    logreg = LogisticRegression(C=1e9)
    logreg.fit(X_train, y_train)
    y_pred_class = logreg.predict(X_test)
    return metrics.accuracy_score(y_test, y_pred_class)

# Attempt 1
feature_cols = ['serv', 'farm', 'LAT', 'LON', 'Business_Estab_20to99_emp',
'poploss','PctBlack', 'OilRegion', 'coalBinay', 'Pct_SomeCollege_2009_2013',
'StimPC_Limit', 'Median_Household_Income_2013']
train_test_auc_score(feature_cols)
train_test_confu_matrix(feature_cols)
train_test_accuracy_score(feature_cols)

### increase the threshold
y_pred_class = np.where(y_pred_prob > 0.75, 1, 0)
metrics.confusion_matrix(y_test, y_pred_class)
metrics.roc_auc_score(y_test, y_pred_prob)
metrics.accuracy_score(y_test, y_pred_class)
#array([[ 29,  63],
#      [ 52, 615]]
```

5.2.1.3 Decision Trees

I computed the Accuracy, AUC and a confusion matrix for each set of features. My best attempt was the limited list of features (B) listed above using a max depth of 3. The accuracy for that model was .888, about .01 higher than the null model. The confusion matrix was as follows.

True Positives: 659
True Negatives: 15
False Positives: 77
False Negatives: 8

Example Code:

```
X = cnty_data_notnull_cl[feature_cols]
y = cnty_data_notnull_cl.Returned
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)

treeclf = DecisionTreeClassifier(max_depth=5, random_state=1)
treeclf.fit(X_train, y_train)
y_pred_class = treeclf.predict(X_test)
metrics.accuracy_score(y_test, y_pred_class)
```

5.2.1.4 Gaussian Naive Bayes

I computed the Accuracy, AUC and a confusion matrix for each set of features. My best attempt was the limited list of features (B) listed above using a max depth of 3. The accuracy for that model was .86, about .01 below than the null model. The confusion matrix was as follows.

'''True Positives: 653
True Negatives: 13
False Positives: 79
False Negatives: 14'''

Example Code:

```
X = cnty_data_notnull_cl[feature_cols]
y = cnty_data_notnull_cl.Returned

# split into training and testing sets
```

```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)


# testing accuracy of Gaussian Naive Bayes
gnb = GaussianNB()
gnb.fit(X_train, y_train)
y_pred_class = gnb.predict(X_test)
metrics.accuracy_score(y_test, y_pred_class)
```

5.2.2.1 3-Class Classification Modeling

I repeated the steps from 5.2.1 with three classes for KNN, NB and Decision Trees. The null accuracy was .814. Again in this case, I was unable to do much better than the null model. In the interest of brevity in this paper, see code for examples and results.

5.3 Regression Modeling

I chose several regression modeling approaches to understand how many months it will take for a county will return to within 1% of its pre-recession unemployment rate.

I cleaned my dataset because several datasets were not available in all counties. I dropped any county that had null values in the returned to within 1%, industry types, or demographic features. I used notnull() to drop out any features and created a new data frame called cnty_data_notnull_cl. I ended up with 2631 counties to research on.

For each of the following modeling efforts I tried many different sets of features, I am only including the sets that were most effective at modeling in this paper. In my provided code you can explore other less successful model sets.

5.3.1 Calculate the null RMSE

I calculated the null RMSE by using the average months it took until a county recovered and it was **18.773.**

Example code:

```
# Calculate Null RMSE
# split X and y into training and testing sets
X = cnty_data_notnull[feature_cols]
y = cnty_data_notnull.MonthsReturn_pls1
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
```

```
# create a NumPy array with the same shape as y_test
y_null = np.zeros_like(y_test, dtype=float)

# fill the array with the mean value of y_test
y_null.fill(y_test.mean())
y_null
# compute null RMSE
np.sqrt(metrics.mean_squared_error(y_test, y_null))
# define a function that accepts a list of features and returns testing RMSE

### NULL RMSE = 18.773
```

5.3.2 Linear Regression

My best linear model used all of my features and returned an RMSE of **14.27**, well below the null accuracy of 18.63.

Example Code:

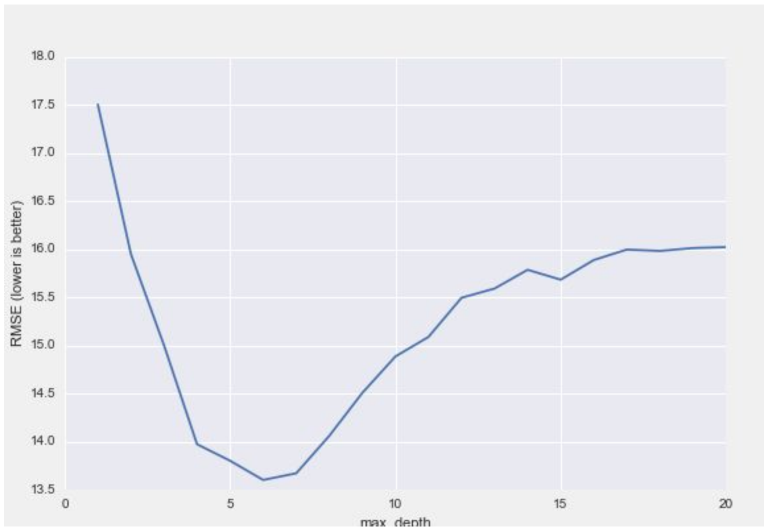
```
# define a function that accepts a list of features and returns testing RMSE
def train_test_rmse(feature_cols):
    X = cnty_data_notnull[feature_cols]
    y = cnty_data_notnull.MonthsReturn_pls1
    X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
    linreg = LinearRegression()
    linreg.fit(X_train, y_train)
    y_pred = linreg.predict(X_test)
    return np.sqrt(metrics.mean_squared_error(y_test, y_pred))
#Attempt 5 (ALL!)
feature_cols= ['LAT', 'LON', 'Rural_urban_continuum_code_2013',
'Urban_influence_code_2013', 'Civilian_labor_force_2011',
'Median_Household_Income_2013',
'Median_Household_Income_Percent_of_State_Total_2013',
'Pct_less_than_hs_2009_2013', 'Pct_HS_Grad_only_2009_2013',
'Pct_SomeCollege_2009_2013', 'Pct_college_grad_2009_2013', 'Net_Mig_2010_2014',
'MigRate_2010_2014', 'Business_Estab_under_20_emp',
'Business_Estab_20to99_emp', 'Business_Estab_100to499_emp',
'Business_Estab_over500_emp', 'Stimulus', 'StimulusPerCapita', 'farm', 'mine', 'manf',
'fsgov', 'serv', 'nonsp', 'house', 'loweduc', 'lowemp', 'perpov', 'poploss', 'rec', 'retire',
'perchldpov', 'PctBlack', 'Summer678', 'Spring345', 'Fall91011', 'StimPC_Limit',
'StimPC_Limit2', 'HolidaySeason', 'January', 'February', 'March', 'April', 'May', 'June',
'July', 'August', 'September', 'October', 'November', 'December', 'OilRegion', 'Bakken',
'EagleFord', 'Marcellus', 'CoalPts', 'coalBinay']
train_test_rmse(feature_cols)
```

## RMSE = 14.27

5.3.3 Regression Trees

My best Regression tree model used all of my features and returned an RMSE of **13.6**, well below the null accuracy of 18.63, but not as successful and better than the best linear regression model. For each mode I used 10-fold cross-validation and computed every model depth from 1 to 21. The ideal model depth of my best model was 7, see chart below as an example.

Exploration of ideal model depth for best Regression Tree Model



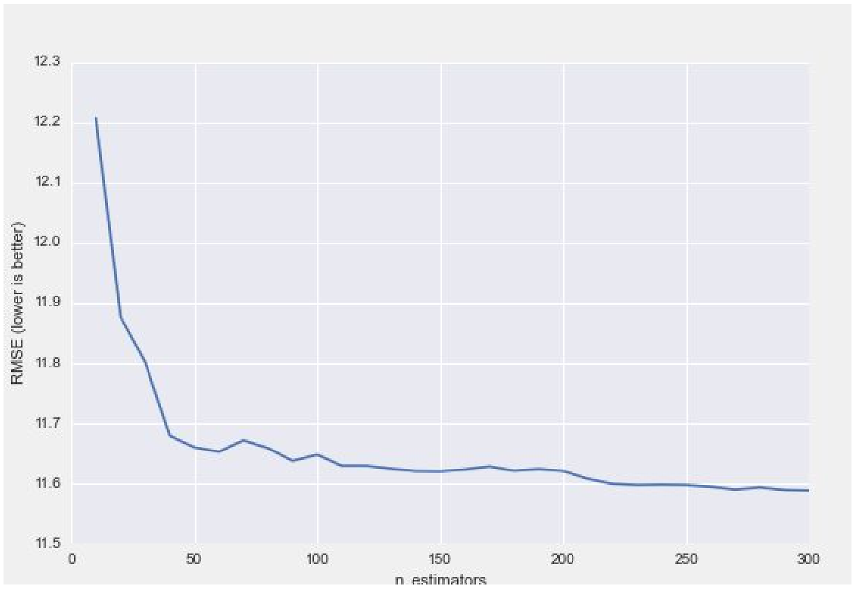
Example Code:

```
## Regression Trees
from sklearn.tree import DecisionTreeRegressor
treereg = DecisionTreeRegressor(random_state=1)
treereg

# list of values to try for max_depth
max_depth_range = range(1, 21)

# list to store the average RMSE for each value of max_depth
RMSE_scores = []
X = cnty_data_notnull[feature_cols]
y = cnty_data_notnull.MonthsReturn_pls1

# use 10-fold cross-validation with each value of max_depth
from sklearn.cross_validation import cross_val_score
for depth in max_depth_range:
    treereg = DecisionTreeRegressor(max_depth=depth, random_state=1)
    MSE_scores = cross_val_score(treereg, X, y, cv=10, scoring='mean_squared_error')
    RMSE_scores.append(np.mean(np.sqrt(-MSE_scores)))


# plot max_depth (x-axis) versus RMSE (y-axis)
plt.plot(max_depth_range, RMSE_scores)
plt.xlabel('max_depth')
plt.ylabel('RMSE (lower is better)')

# Calculate RMSE at best depth
treereg = DecisionTreeRegressor(max_depth=7, random_state=1)
scores = cross_val_score(treereg, X, y, cv=10, scoring='mean_squared_error')
np.mean(np.sqrt(-scores))
```

5.3.4 Random Forests

My best Regression tree model used all of my features and returned an RMSE of **11.6**, well below the null accuracy of 18.63, and the best model I found during this research. For each mode I used 5-fold cross-validation and computed every estimator between 10 and 310 moving up by 10. The ideal estimator was approximately 120. From there I used 5-fold cross validation for every number of features in the dataset (72!). The value of adding features leveled out around 32. I also calculated the most important features to give an idea of what was most helpful.

Exploration of ideal estimators for best Random Forests Model



FeatureName	Feature Importance
LON	0.19848
LAT	0.137756
PctBlack	0.085512
poploss	0.052182
farm	0.051747
January	0.04425
Civilian_labor_force_2011	0.041937
Pct_less_than_hs_2009_2013	0.039097
Median_Household_Income_2013	0.02433
Pct_SomeCollege_2009_2013	0.023695

```
rfreg = RandomForestRegressor()
rfreg

feature_cols = ['serv', 'CoalPts', 'farm','OilRegion', 'Bakken', 'EagleFord', 'Marcellus',
'HolidaySeason', 'LAT', 'LON', 'poploss', 'PctBlack', 'Average2007',
'Pct_SomeCollege_2009_2013', 'StimPC_Limit', 'Median_Household_Income_2013']
X = cnty_data_notnull[feature_cols]
y = cnty_data_notnull.MonthsReturn_pls1

# list of values to try for n_estimators
#range --> Try every estimator from 10 to 310 by 10s
estimator_range = range(10, 310, 10)

# list to store the average RMSE for each value of n_estimators
RMSE_scores = []

# use 5-fold cross-validation with each value of n_estimators (WARNING: SLOW!)
for estimator in estimator_range:
    rfreg =  RandomForestRegressor(n_estimators=estimator, random_state=1)
    MSE_scores = cross_val_score(rfreg, X, y, cv=5, scoring='mean_squared_error')
    RMSE_scores.append(np.mean(np.sqrt(-MSE_scores)))
#Run for the best number of features
feature_range = range(1, len(feature_cols)+1)

# list to store the average RMSE for each value of max_features
RMSE_scores = []
```

```
# use 5-fold cross-validation with each value of max_features (WARNING: SLOW!)
for feature in feature_range:
    rfreg = RandomForestRegressor(n_estimators=120, max_features=feature,
random_state=1)
    MSE_scores = cross_val_score(rfreg, X, y, cv=5, scoring='mean_squared_error')
    RMSE_scores.append(np.mean(np.sqrt(-MSE_scores)))

# plot max_features (x-axis) versus RMSE (y-axis)
plt.plot(feature_range, RMSE_scores)
plt.xlabel('max_features')
plt.ylabel('RMSE (lower is better)')

# compute feature importances
pd.DataFrame({'feature':feature_cols,
'importance':rfreg.feature_importances_}).sort('importance')
```

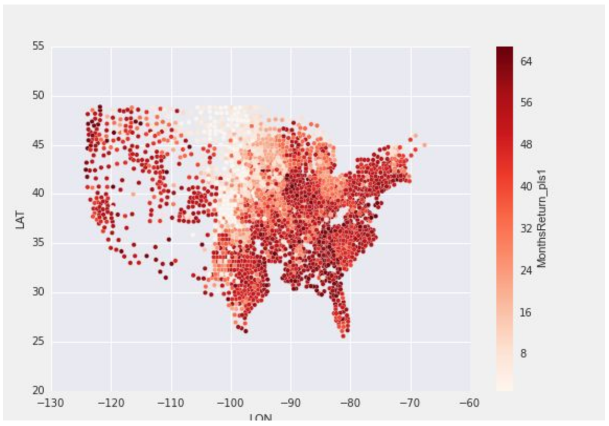
5.4 Regression without months that started study period within 1% of pre-recession unemployment

In an attempt to remove some anomalous data--counties whose recession rate were below 1% when the study period began--I repeated the steps in 5.3 and found that I wasn't able to predict better than with the whole dataset. There is more information in the code.

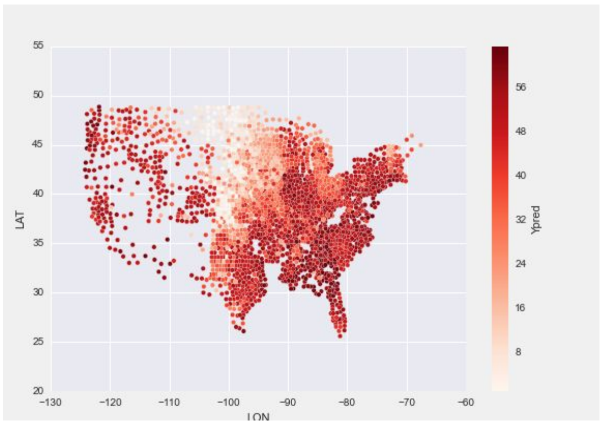
6. Findings and Conclusions

While this approach did not have much success in determining whether counties have returned to near their pre-recession unemployment rate or not, it was successful in helping to determine how quickly a county might return to that rate after the end of our recession.

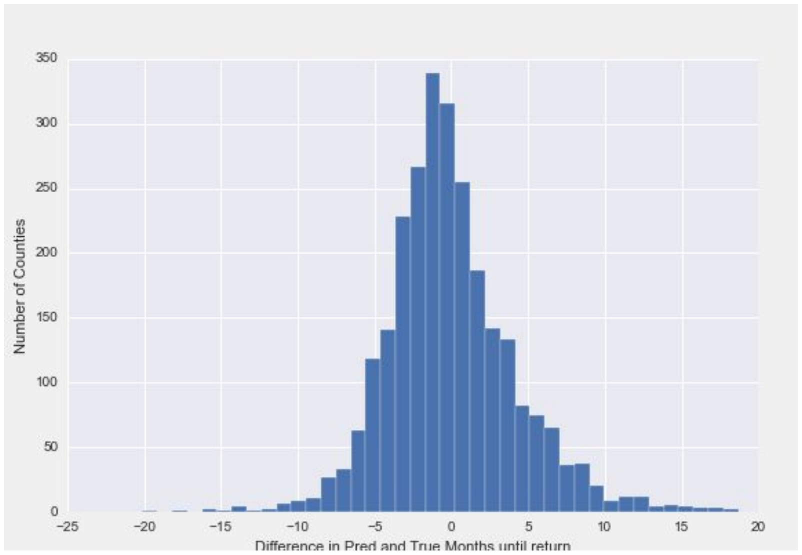
Months until Return



Predicted Months until Return

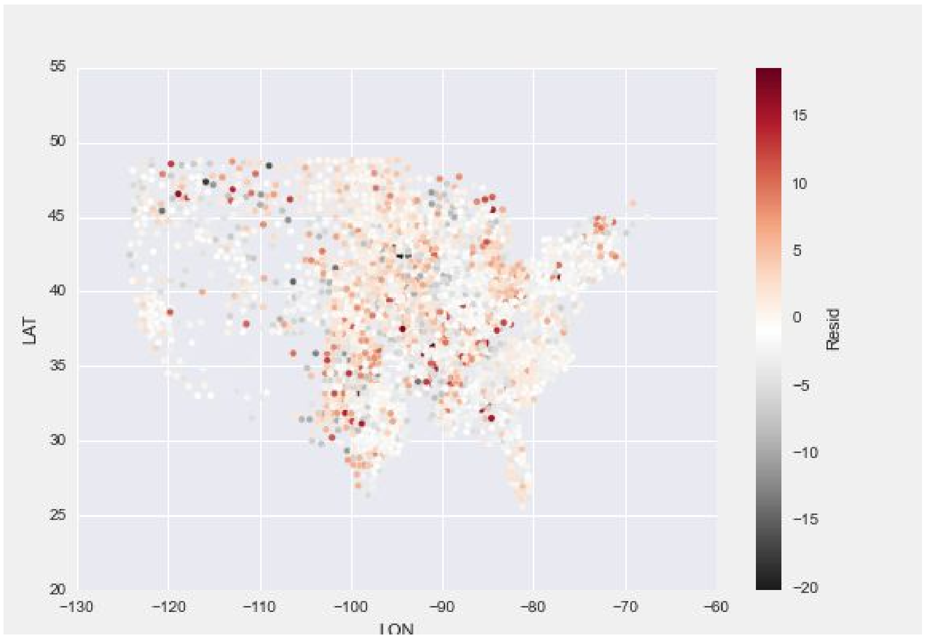


Histogram of Predicted Months Until Return Minus True Months Until Return



Of the 2,663 counties that had returned to within 1% of their pre-recession unemployment rate, I was able to predict the month when a county actually returned within 1 month 23% of the time, within 3 months of its actual date 59% of the time and when it returned within 6 months 87% of the time.

Map of Residuals



Counties in red were predicted to take longer to return to their pre-recession unemployment rate, while counties in black returned more quickly than the model predicted. These colors are lightly clustered in the middle of the country -- indicating that there may be spatial processes that were not accounted for in the model, however there are few clear clusters (other than maybe Vermont), that seem easily modelable.