Rental Investment Analysis

Author: Wilson Lau
Date: Dec 6th 2016
Mentor: Ryan Rosario

Introduction

Real estate investment is a passive income opportunity for investors to accumulate return without active management. Significant long term appreciation is possible for real estate. However, appreciation often happens in a short period of time and the appreciation is heavily affected by local market conditions. It is difficult for out-of-the-market investors to detect local trends and acquire properties before prices go up. My capstone project is trying to help investors to detect real estate investment opportunities and be able to predict which zip codes will experience significant appreciation in the near future. This analysis is to help real estate investors to discover high potential zip codes to buy their investment rental properties. The analysis will produce a prediction model to predict whether a specific zip code will be a favour for investment or not. To keep the data problem at a manageable size, I decided to process only the zip codes in California.

Stakeholders

The target users are real estate investors interested in making investments outside of the market they live in. Although real estate generally appreciates steadily, significant gains happen within a short period of time. Real estate price is largely affected by changing conditions in the local market. Significant business development such as local government adding favourable policy or a large employer launching a new facility could drive up home and rental prices, but this kind of news often remains local until late stages of development, and it is hard for investors outside the region to discover the investment opportunity. This analysis attempts to detect the early signal of local real estate appreciation in different zip codes, so investors have opportunity to acquire properties before significant appreciation occurs.

Dataset

Two datasets are used in this analysis. The first dataset is the home price dataset from Quandl's Zillow[1] data. This dataset is a monthly time series showing the change of home price and other data related to home price such as median rent price, price to rent ratio, etc.
The second dataset is IRS tax income based on zip code. This IRS data includes variables such as Adjusted Gross Income, Schedule A deduction, etc.

[1] https://www.quandl.com/data/ZILL-Zillow-Real-Estate-Research

Home Price Dataset

The important columns for the home price dataset is the month, home price and the corresponding zip code. In addition, to fully evaluate the potential for rental property investment, few columns indicating rent price and the relationship between rent and home sales price will probably be important for the analysis as well. The home price dataset actually has 22 columns. For example, it has columns for different home size e.g. 1 bedroom, 2 bedroom, 3 bedroom, price per square foot, average listing price, etc. Since it seems that most of these columns are highly correlated to home price, they may not be very helpful for predicting home price.

IRS Dataset

The IRS dataset provides additional dimensions to the analysis. Job opportunities and income growth are probably the most significant factors to drive profitability for rental properties. The IRS dataset has columns such as AGI which indicates the income level for the corresponding zip codes, and columns like total number of tax return filed which could indicate the job growth in the region. The IRS dataset has columns showing the aggregated number of all tax filers in different zip codes. However, those numbers are by year, so there is no monthly breakdown available.

Approach

This project is divided into two stages:

Stage One

The first stage is to use anomaly detection to discover which zip codes have success sprike of upward price appreciation. The anomaly detection technique used is to compare new monthly home price with historical price as well as the price change of the surrounding areas. A zip code is considered to have a spike when it increases significantly over both its previous months and also the average of its entire region.

Stage Two

More data such as IRS income was collected for the zip codes that have sudden increase of home price. Those new dataset will be used to figure out which features contribute to the price appreciation and from that a predictive model was created to predict which zip codes would have extraordinary price appreciation in the near future.

Dataset Collection & Processing

Quandl's API is very powerful and very flexible. At the same time, it was a bit complicated to acquire the home price. One way to acquire home price was to pass zip code as input parameter, and Quandl would return the historical price data with all the data related to that zip code. So, to get the data, I used a zip code dataset that contains all the zip code of the US and then used the Quandl API to pull data one zip code at a time. The home price data was quite clean and there was not much cleaning required afterwards.

The home price dataset and IRS dataset needed to be merged to provide features for good prediction, as home price change is based on many factors. Job opportunity is probably one of the most significant factors to drive home and rental price up or down. The IRS dataset, with adjusted gross income and number of total returns filed, indicates both the level of income growth and job opening growth for a given zip code. Merging the home price and IRS datasets provides a more comprehensive picture of more significant factors affecting rental real estate opportunities.

Collecting Home Price Datasets

The home price dataset is published by Zillow[2] and made available by Quandl[3]. Quandl provides a number of APIs to acquire the data. I found that the best way for this project is to use the API that passes one target zip code and retrieves its historical time series price data.

```
==Output/Code - California zip code total=================================
Number of Zipcodes in California:  2792

Code
==============================================================
```

I found that there are about 2792 zip codes in California. Since the Quandl API has a restriction on the number of calls within a certain time period, I decided to manually call the API on a small number of zip codes at a time and store the datasets in separate pickle files.

Processing Home Price Dataset

Since the Quandl Zillow data is relatively clean, there is not much cleaning required. The main processing was to identify the "hot" zip codes and in which month and year they are "hot". These "hot" zip codes will be the main subject of my analysis and I will try to develop a prediction model to predict which zip codes may become hot before they actually happen.

[2] http://www.zillow.com/

[3] https://www.quandl.com/

A “hot” zip code is defined as a zip code and month such that the house price is much higher than other months and much higher than other zip codes in same month.

Labeling "Hot" and "Cold" Zip Codes

The way "hot" zip codes are labeled is through a systematic anomaly detection algorithm that calculates the sudden increase of home price within the same zip codes as well as across different zip codes. The current anomaly detection method used is to find the change of the three-month moving average of home price a 1.5 unit change in z-score over previous month as well as over the average price of all zip codes. Then, I label those zip codes as "hot" in a new column and label them as "cold" otherwise.

==Output/Code - Labeling “hot” and “code” zip codes =================
Code

==

Collecting IRS (Internal Revenue Service) Dataset

The IRS data I used is obtained from the IRS web site[4]. It is available as a series of zipped CSV files for each year from 1998 to 2011. Since different year's CSV files are different in columns, I created different scripts to parse the data for each year. The following is a sample of code I used in one year. The entire code for collecting IRS data is available in a separate Jupyter notebook[5].

==Output/Code - Parsing IRS Dataset ======================
Code

===

Merging Home Price Dataset with IRS Dataset

The home price data frame and the IRS data frame are merged together based on month. Each row in the combined data frame represents one month of data. The combined data frame is indexed by two levels - first by features and then by zip code. Since the IRS data doesn’t currently have monthly breakdown, the yearly based rows are transformed into monthly rows by adding eleven more rows to match the format of the home price dataset. The yearly based IRS income data for all different zip codes are assumed to remain the same for the entire year as the monthly rows are added.

[4] https://www.irs.gov/uac/soi-tax-stats-individual-income-tax-statistics-zip-code-data-soi

[5] File IRSdataCollection.ipynb on Github directory

Cleaning and Processing the Data Frame

The original IRS dataset was quite "messy". Some data cells include special characters such as "*" and "-". I applied various techniques to discover what special characters are used and remove them accordingly. Also, I used various forward filling methods to fill empty cells created from those special characters as well as those cells which come from the IRS data yearly-to-monthly conversion.

==Output/Code - Cleansing Combined Dataset ====================
Code

==

Data Exploration and Analysis

The following is the exploratory analysis on the full cleaned data frame combining home price data and IRS data. The main key for the analysis is to understand the important factors that determine whether a zip code turns hot or cold in any given month.

Data Pre-Processing for Plotting

I re-formatted the data frame to make it easier for exploratory data analysis. The main modification was to change is to change the column indicating whether a zip code is "hot" or "cold" from 1/0 into string hot/cold to make it easier for plots to show the name of the group.

==Output/Code - Re-Formating data frame =======================
Code

==

"Hot" zip codes with the most number of months

Zip codes can be hot or cold at different times. It is good to know how many times a zip code is considered as "hot" in a month throughout the entire time period of the data. The following bar chart shows the top 10 zip codes with the most number of times marked as "hot" in a month.

Results:
It appears that the top hot zip codes are spread throughout northern and southern California and between major metropolitan areas as well as more rural areas

The "hottest" zip codes are 94609 (Oakland, CA) and 93280 (Wasco, CA) with count of "hot" months of 16, followed by 95113 (San Jose, CA) with 15 and 96093 (Weaverville, CA) with 14.

==Output/Code - hot zip codes with the most number of months =========

Zip Code	City Name	Counts of Hot Zip Codes
94609	Oakland	16
93280	Wasco	16
95113	San Jose	15
96093	Weaverville	14
93501	Mojave	13
95811	Sacramento	13
94301	Palo Alto	13
93426	Bradley	13
94022	Los Altos	13
95070	Saratoga	13

Code

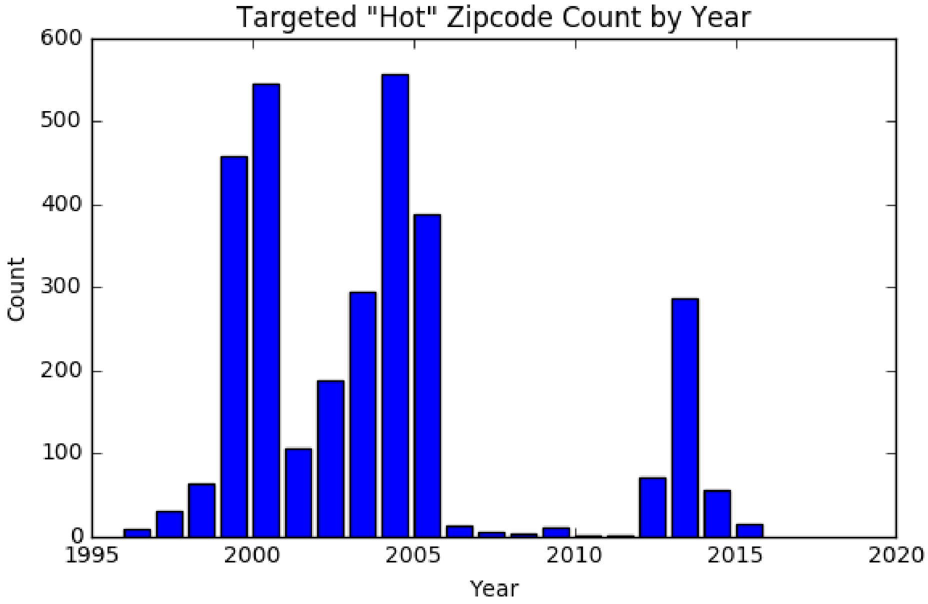
===

Counting The Number of Hot/Cold Zip codes by Year

The appearance of "hot" zip codes may depend on the general economy, so it may be interesting to know the number of "hot" zip codes in a given year. In the following bar chart, I count the number of identified "hot" zip code by year.

Results:
The highest number of "hot" zip codes occurred in 2004 with 556. Year 2000 has the second highest with 544 followed by Year 1999 with 458. One clear observation is that there are very little or no "hot" zip codes from 2006 to 2012, which happened to be the period that the real estate market collapsed. That economic downturn is known as “The Great Recession".

==Output/Code - Counting The Number of Hot/Cold zip codes ==============



Code

==

Compare Home Price grouped by Targeted Zip Code

This analysis tries to understand the differences in median home prices between hot zip codes and cold zip codes across different size of home. I used specifically the home price columns for 1, 2, 3, 4 and 5 or more bedroom as well as the overall price and calculate their average price grouped by zip code. Conventional wisdom may expect that "hot" zip codes have higher price because homes in "hot" zip codes may be more desirable. To verify, I created 6 boxplots to examine the median of home price for different home size.

Results
From the observation of all the boxplots, there is only one (the one bedroom) whose hot zip codes' median price is higher than its cold zip codes'. The rest of the boxplots including the overall home price boxplot have their hot zip code median home price less than cold zip code.

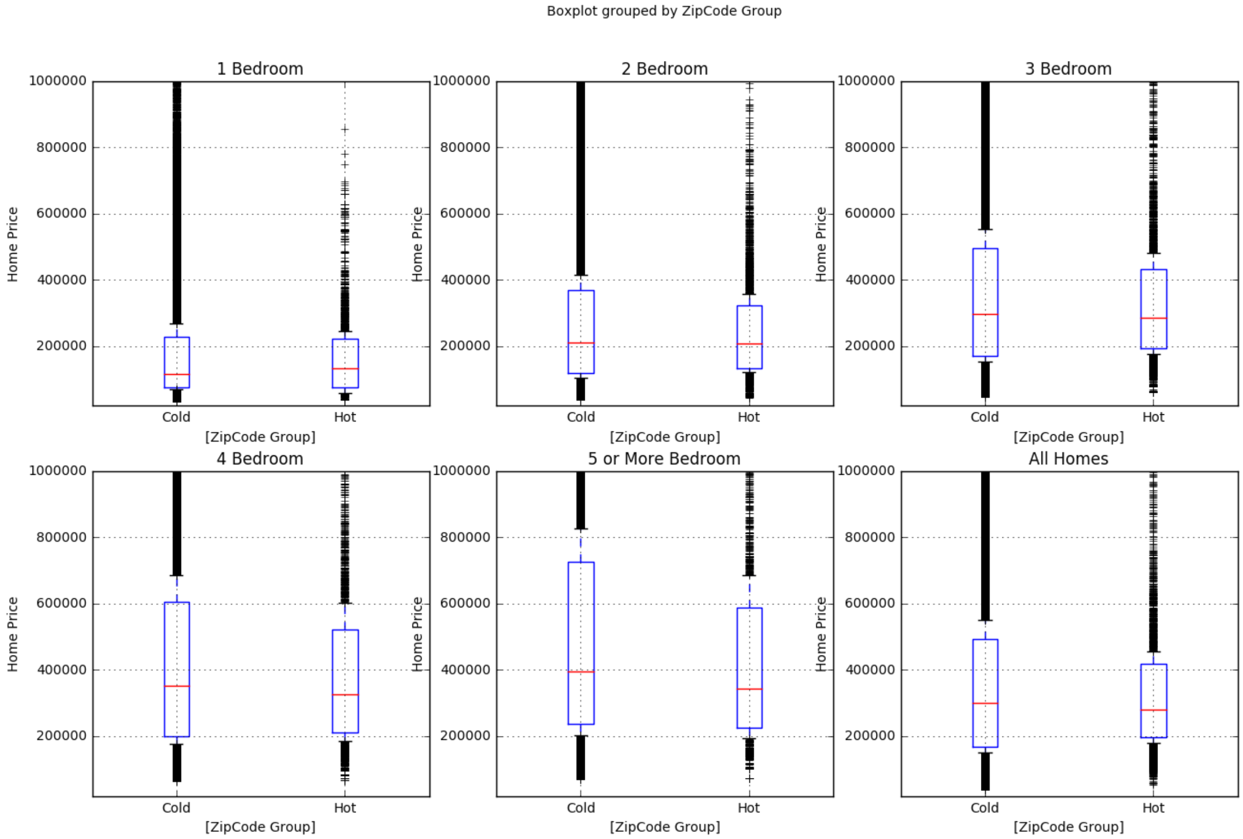
It is not totally clear why one bedroom property in hot zip codes has higher media price than cold zip codes. It is possible that one bedroom properties generally have higher demand compared to the other bedroom groups, especially in today's "Gig Economy". Young

population tends to change job much more often. One bedroom home allows singles and smaller families to move easily. They may drive up the rental price and median home price as they move to zip code areas with job opportunities.

On the other hand, the difference between hot and cold zip codes median price gets larger as the number of bedroom increases. The 5 bedroom properties' cold zip code median price is significantly higher than its hot zip code median price. Perhaps the larger (more bedrooms) the house is, the larger the room for price growth, especially for properties with relatively low median price.

This result suggests that for homes larger than one bedroom, hot zip codes generally have relatively lower home price, so investors should not look at high priced area for investment. The lower price base provide more appreciation opportunities for investors.

==Output/Code - Comparing home price grouped by targeted zip code =============



[Code](#)

==

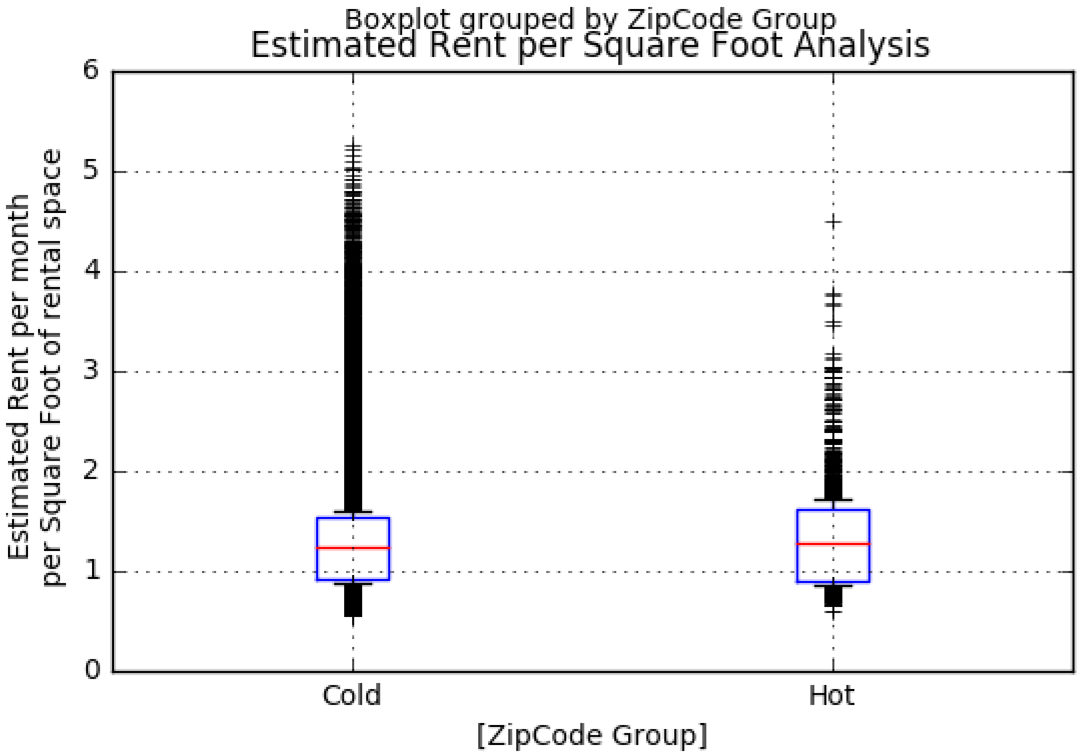
Comparing "Estimated Rent per Square Foot" between Hot and Cold Zip codes

Rent price is an important factor to determine whether rental property is profitable or not. It will be good to understand the relationship between rent per square foot and hot/cold zip codes. I calculated the mean rent per square foot for both the hot and cold zip code groups.

Results

The boxplot shows that the "Hot" zip codes have higher rent per square foot than "cold" zip codes. The estimated rent per square foot for hot and cold zip codes are 1.31 and 1.27 respectively. The difference between hot and cold zip codes are 0.04 which means that for a home with 1,000 square foot, the difference in rent price between hot and cold zip codes is $40 per month. This difference doesn't seem to make significant impact for investors' decision making.

==Output/Code - Estimated Rent per Square Foot ===================



Code

===

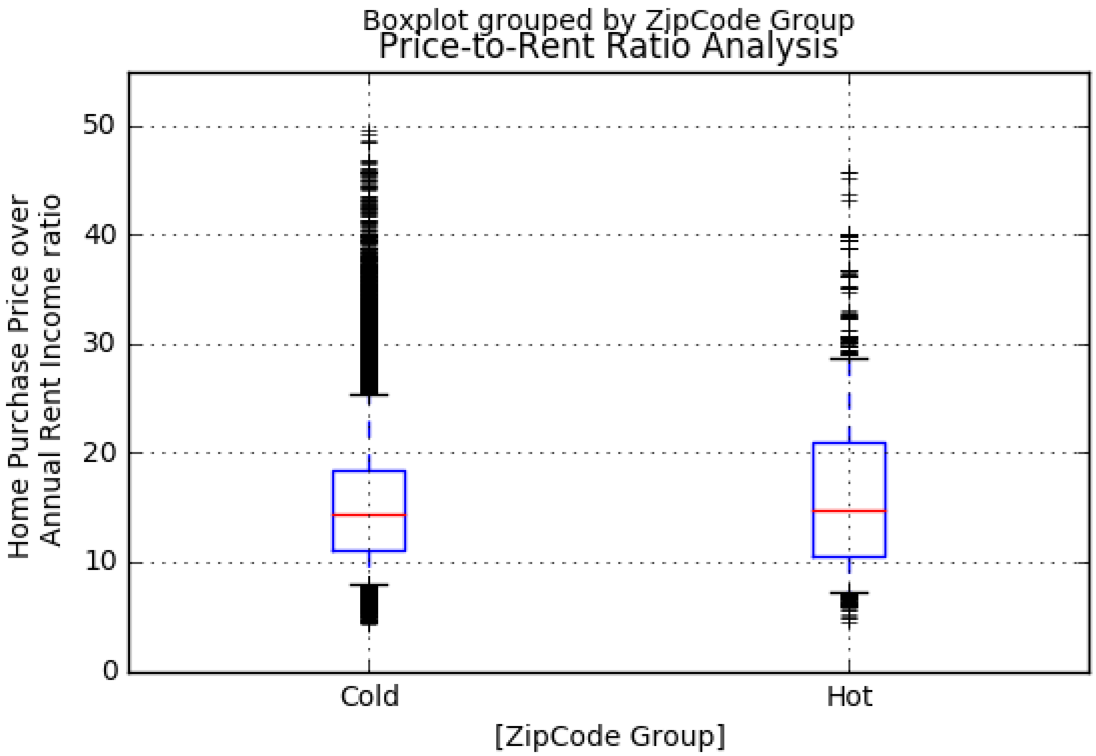
Comparing Price-to-Rent Ratio between hot and cold zip codes

It is also interesting to see the relationship between price, rent and hot/cold zip code group. Price-to-rent ratio is the median home sales price over median annual rent. Price-to-rent ratio indicates the cost for investors to acquire investment properties. High price-to-rent ratio means that investors will need to invest more to generate the same monthly rental income return. Low price-to-rent ratio means that investors will need to put up less initial investment, so their return-on-income will generally be higher.

Results:
As I calculate the average price-to-rent ratio between hot and cold zip code group, I found that "hot" zip codes have slightly higher median price-to-rent. The price-to-rent ratio difference between hot and cold zip codes is 1.129 which means that investors will need to pay 1.129 times more of their rentals' annual income to purchase homes in hot zip codes. This could be significant because that basically means that it may take investors a little more than one additional year on average to get their capital back if they invest in properties at hot zip codes. Investors may need to justify their investment by analyzing both the price appreciation as well as the annual rent income.

==Output/Code - Comparing Price-to-Rent Ratio ===================

Zip Code Group	Price-to-Rent Ratio
Cold	15.183392
Hot	16.311877



Code
==

Comparing Adjusted Gross Income between Hot and Cold Zip codes

Income level is probably another important factor driving zip codes from "cold" to "hot". Adjusted Gross Income is generally the total income amount of tax payers before tax deduction. When the median AGI in a region increases, that generally means that the economic activities in the region is growing, and people generally can afford more to pay the essentials like rent or other disposable items. I analyzed the relationship between average AGI between "hot" and "cold" zip code groups. I generate a table showing the mean, standard deviation, quantiles, etc between hot and cold zip code groups. Also, I made a boxplot to visually see the differences.

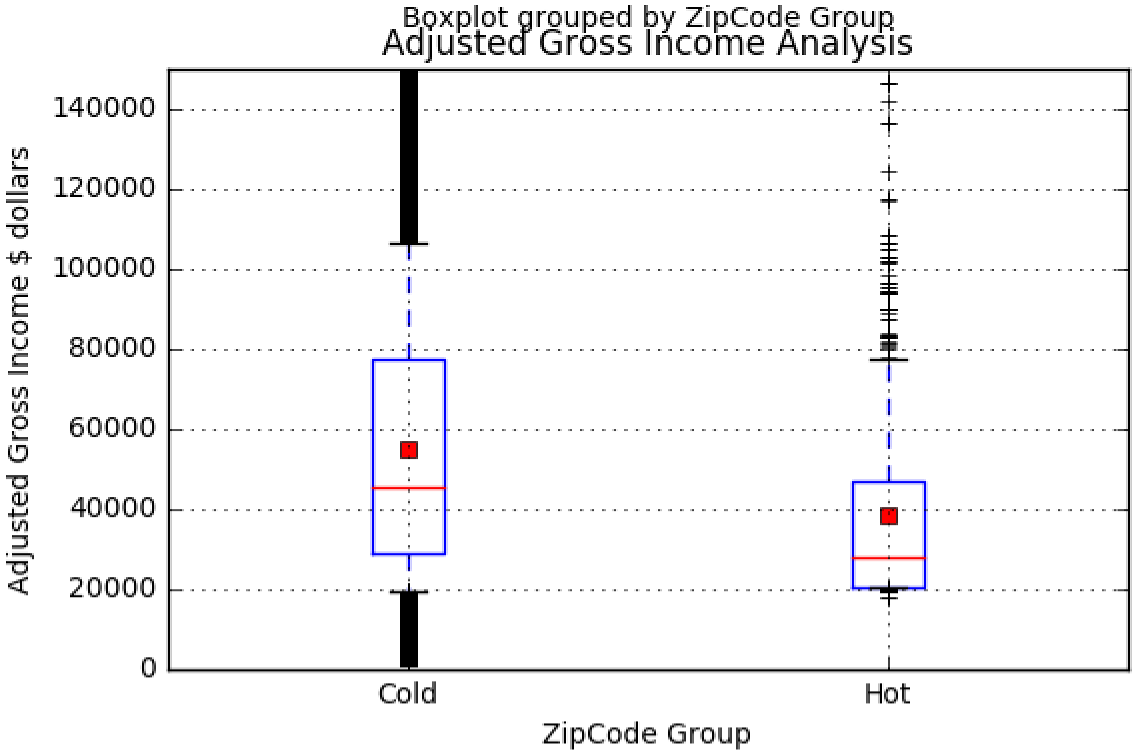
Results:
The analysis shows that the mean of the average AGI for hot zip codes is much smaller than cold zip codes. The mean of the average AGI for hot zip is about $38260 while the average AGI for cold zip code is $54952. Also, the standard deviation for average AGI for hot zip codes is smaller than that for cold zip codes.

This result seems to be a bit surprising. One explanation for this result is because lower income zip codes provide relatively low baseline for home price to appreciate and proportional wise their appreciation goes up in higher percentage.

Gentrification could very well be the reason for zip codes to become hot. Older and poor neighborhood could experience sudden price spike after renovation. There could be a high correlation between Gentrification projects and hot zip codes. This analysis doesn’t have data related to gentrification. A follow-up analysis is recommended to further understand how gentrification contributes to zip codes to get “hot”.

==Output/Code - Comparing Average AGI between hot and cold zip codes =====

Average Adjusted Gross Income						
Zip code group	Count	Mean	Standard Deviation	25% Percentile	50% Percentile	75% Percentile
Cold	412370	$54953	$47393	$28902	$45372	$77217
Hot	3095	$38260	$23052	$20294	$27990	$46969



Code

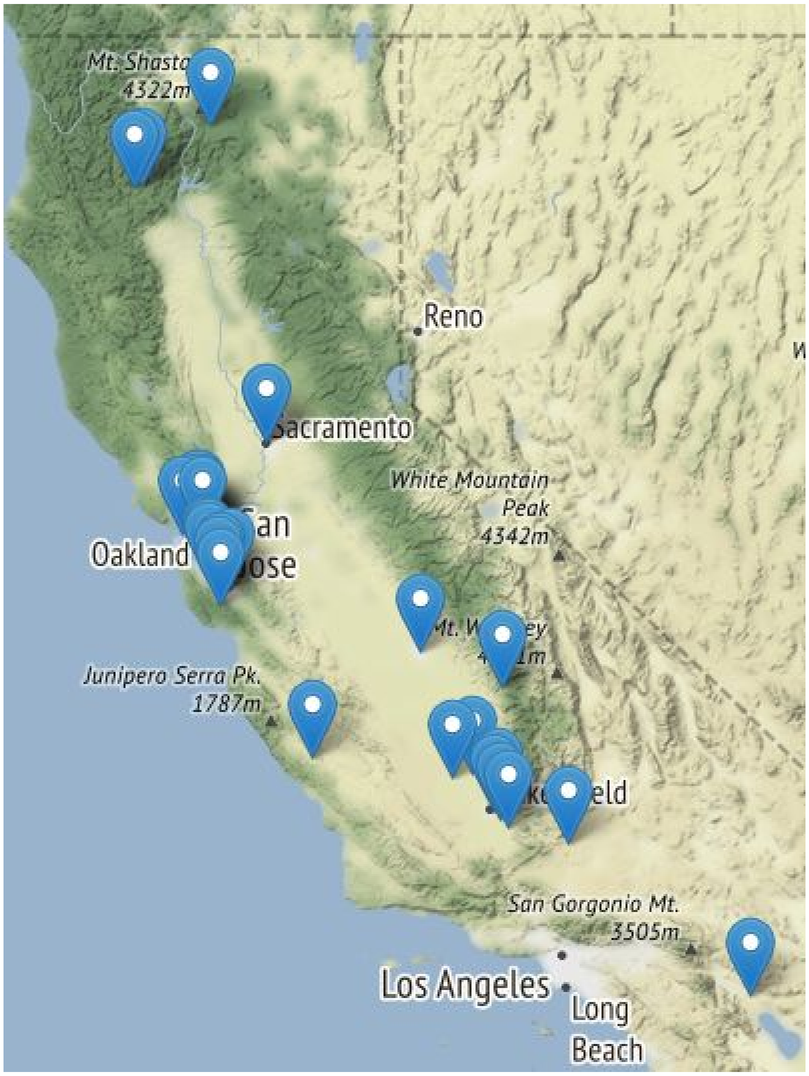
===

Geographic Location of the Zip Codes

Knowing where the hot zip codes are geographically will also be very helpful for investors. The following analysis plots the top hot zip codes with the highest number of months being marked as hot. The top 30 zip codes with highest frequency of being hot is plotted on the following map:

Results:
The hot zip codes seem to spread pretty evenly all over the entire California. There seems to have more hot zip codes in the San Francisco Bay Area in North California and around Bakersfield near South California.

==Output/Code - Plot the hot zip codes on a map ===================



Code
==

Building a Model to Predict Hot/Cold Zip Codes

The following steps focus on generating an appropriate datasets for model training and testing.

Generate Balanced Dataset

The original datasets are not balanced because there are very few “hot” zip codes identified in the dataset compared to the “cold” zip codes. There are a total of 3095 hot zip code-months compared to 412370 cold zip code-months. To produce a balanced training and testing dataset, I needed to take samples of the hot and cold zip code month and combine them. I take 10000 samples with replacement from each of the hot and cold zip code and combine them to generate a 20000 row dataset for modeling training and testing.

==Output/Code - Generating balanced dataset sample ===================

Code

==

Spliting Training and Testing Dataset

I decided to split the dataset into 80% training set and 20% testing set.

==Output/Code - Spliting Training and Testing Dataset ===================

Code

==

Training and Evaluating Model

I used two different models - Logistic Regression and Support Vector Classifier to train with my dataset to find the best performing prediction model.

Logistic Regression Training and Evaluation

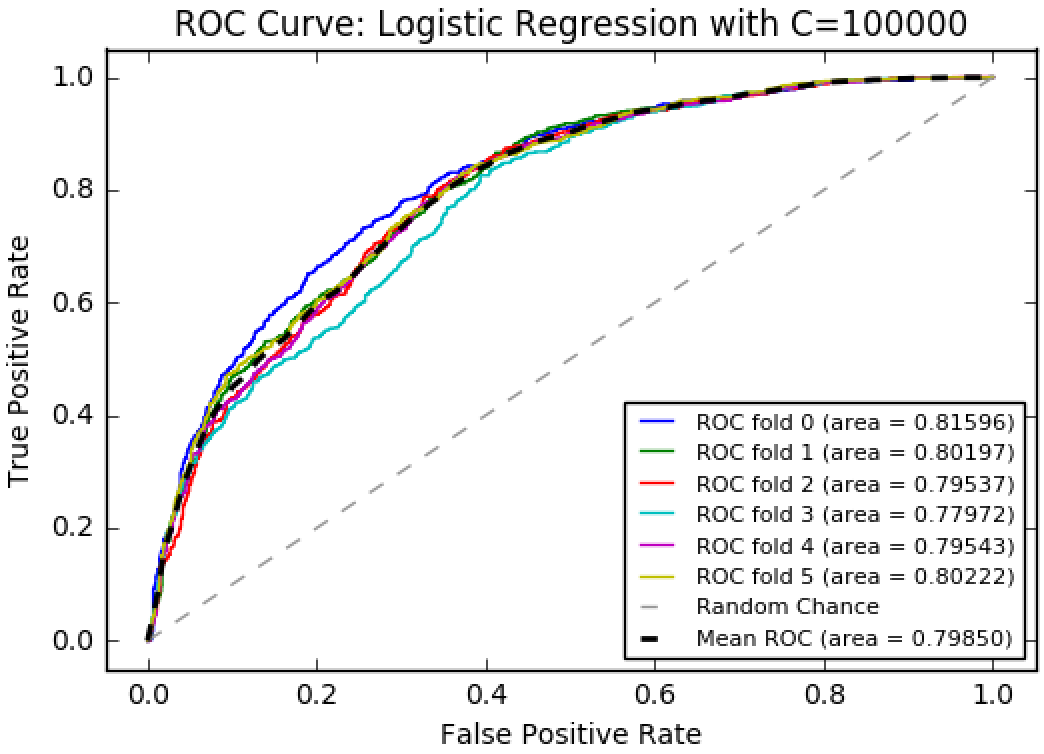
I separate the training dataset of 16000 rows of records into six folds using the StratifiedKFold function to achieve a fair cross validation . I also found the best regularization parameter C using C={10,1000,10000}. The score method I choose for my problem is “precision”. I want to evaluate the model using precision scoring because I want to have high confidence that the hot (positive) zip code that the model prediction will find true hot zip codes. I can't afford to invest properties in zip codes that turn out not to be hot. I won't have much capital to invest in my properties anyway, so I don't care too much about recall. There is no need for me to identify all the hot zip codes, but I just want to make sure that the ones that I decide to invest will be the hot ones.

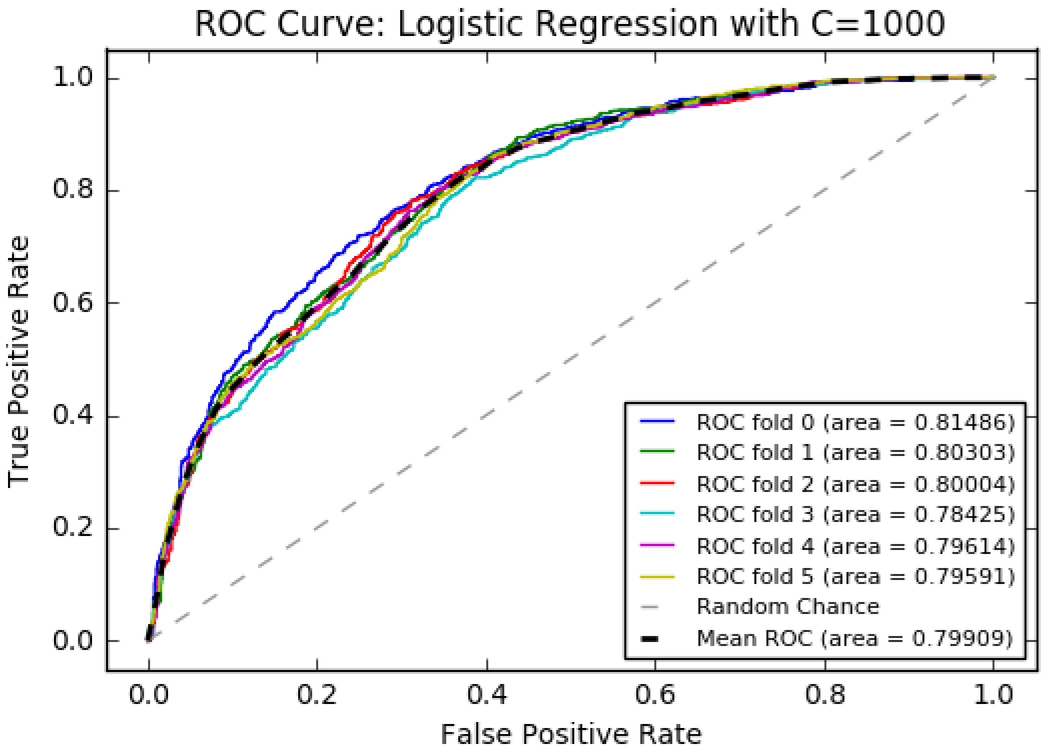
Cross Validation and ROC

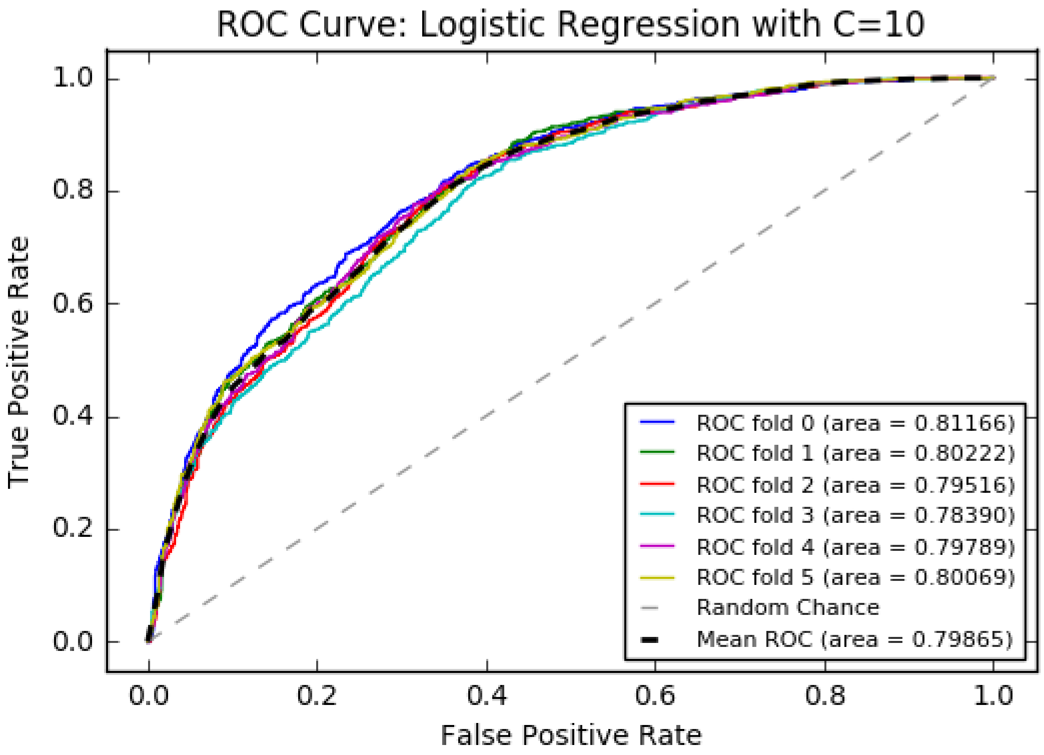
I also plot the ROC curve and evaluate the model with the same 6 folds cross validation for each of the C parameter I tried.

Results:
Comparing the score for all three options with C=10000, 1000 and 10, hyperparameter C being 1000 seems to generate the best performance with precision mean equal to 0.71063 and ROC mean of 0.79909.

==Output/Code - Training Logistic Regression ===================




ROC Curve: Logistic Regression with C=1000
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC fold 0 (area = 0.81486)
ROC fold 1 (area = 0.80303)
ROC fold 2 (area = 0.80004)
ROC fold 3 (area = 0.78425)
ROC fold 4 (area = 0.79614)
ROC fold 5 (area = 0.79591)
Random Chance
Mean ROC (area = 0.79909)


ROC Curve: Logistic Regression with C=10
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC fold 0 (area = 0.81166)
ROC fold 1 (area = 0.80222)
ROC fold 2 (area = 0.79516)
ROC fold 3 (area = 0.78390)
ROC fold 4 (area = 0.79789)
ROC fold 5 (area = 0.80069)
Random Chance
Mean ROC (area = 0.79865)

Code

==

Support Vector Classifier Training and Evaluation

Beside logistic regression, I also tried to use SVC estimator to see if I can improve the performance of the prediction model. I used PCA to reduce the feature dimension of my training data, and then I combined the PCA reduced features with the best selected original features to feed the model training. I used grid search to find hyperparameters. The parameters I tried were 1, 2 or 3 principal component, 1 or 2 original features, RBF or Sigmoid kernel and 0.1, 10 or 1 for C.

Results:
The default Radial Basis function (RBF) kernel seemed to work the best and took relatively little time to train while the score performance is also quite good. With the 16,000 training dataset, SVC with RBF kernel takes about two minutes for each hyperparameter combination training to complete. The Sigmoid kernel is also fast, but its performance is quite poor compared to RBF.

Using grid search, the best hyperparameter I found is C=1, one principal component, RBF kernel and two original best feature after the cross validation on the training dataset. The overall precision score I got after applying the model to the testing dataset is 0.96.

Since SVC generates the best performing model comparing to logistic regression, I will use the SVC model to find my next investment property.

The scores are computed on the full evaluation set				
Zipcode	**Precision**	**Recall**	**F1-score**	**Support**
Cold	**0.92**	**1.00**	**0.96**	**1981**
Hot	**1.00**	**0.91**	**0.95**	**2019**
avg / total	**0.96**	**0.96**	**0.96**	**4000**

==Output/Code - Training Support Vector Classifier ===================

Code

===

Overall Conclusion

This analysis tried to use the US housing price data and the IRS data to help investors discover high potential California zip codes for purchasing rental properties. Through the analysis, some unique characteristics of the high potential “hot” zip codes are identified; for example, “hot” zip codes are spread over Northern and Southern California. Also, “hot” zip codes have slightly higher estimated rent per square foot as well as price-to-rent ratio although their adjusted gross income is lower than “cold” zip codes. “Hot” zip codes generally have lower home price than “cold” zip codes for all numbers of bedroom home except those one bedroom homes.

To develop a prediction model for “hot” zip codes, I tried two modeling algorithm of logistic regression and support vector classifier(SVC). SVC turned out be a better algorithm. The best SVC model delivered a 96% precision score compared to logistic regression model’s 0.71. Rental investors could use my SVC based model to find the right zip codes for their next profitable rental investment.

Appendix

==Appendix - California zip code total================================

```
import pandas as pd
zipcode = pd.read_csv('./data/us_postal_codes.csv')
calzipcode = zipcode[zipcode.State == 'California']
print "Number of Zipcodes in California: ", calzipcode.shape[0]
```

back

==Appendix - Labeling “hot” and “code” zip codes ======================

```
### These codes are trying to identify the "hot" zipcodes.

import numpy as np
import pandas as pd
import pickle
from scipy import stats
import os
import glob
from os import path
import dateUtility
import datetime as dt
pd.set_option('display.max_columns', 500)
pd.set_option('display.max_row', 1000)
```

```
# function to trim fraction
def trim_fraction(text):
    if '.0' in text:
        return text[:text.rfind('.0')]
    return text

#load home price data from Datafiles
df = pd.DataFrame()
for filename in glob.glob(os.path.join('.\\DataPfiles\\', '9*.p')):
    df = pd.concat([df, pd.read_pickle(filename)], axis=0)

### Pivot individual columns into a new dataframe for analysis:
### 1) All Homes price

df = df.reset_index()
dfAllHomes = df.pivot(index='index', columns='ZipCode', values='All Homes')

### calculate the home price's moving average for every 3 months, with minimal of 3 months

dfAllHomesMovingAve = pd.rolling_mean(dfAllHomes,window=3,min_periods=3)

### shift entire set of Moving Average number downward for one month
### This is needed to line up the previous month moving average with the current ### month moving average in the same row
dfAllHomesMovingAve = dfAllHomesMovingAve.shift(periods=1,freq=None,axis=0)

### since I shift the all rows for one month earlier, so I can just subtract the ### two dataframe and calculate the price change.
dfAllHomesDiffFromMovAve = dfAllHomes - dfAllHomesMovingAve #price change
dfAllHomesDiffFromMovAvePercent = (dfAllHomes - dfAllHomesMovingAve)/dfAllHomes.shift(periods=1,freq=None,axis=0) #price percentage change

NumOfMonthForward = 6 #set the number of month to include for calculating the predicting average percentage increase
dfAllHomesDiffFromMovAvePercentMovAve = pd.rolling_mean(dfAllHomesDiffFromMovAvePercent,window=NumOfMonthForward,min_periods=NumOfMonthForward) # calculate the moving average percentage increase for 6 months
dfAllHomesDiffFromMovAvePercentMovAve2 = dfAllHomesDiffFromMovAvePercentMovAve.shift(periods=-(NumOfMonthForward-1),freq=None,axis=0) # shift dataframe upward for 5 record so the current month shows the moving average percentage increase for the next 6 months(including current month)
dfAllHomesStdDevInZipCode = dfAllHomesDiffFromMovAvePercentMovAve2.copy() #dataframe to hold z score within zipcode

##loop through each zipcode column
for x in dfAllHomesStdDevInZipCode.columns:
    if sum(dfAllHomesStdDevInZipCode[x].isnull()) > 0:  # if any month has null value,
drop the entire zipcode
        dfAllHomesStdDevInZipCode.drop(x,axis=1)
    dfAllHomesStdDevInZipCode[x] = (dfAllHomesStdDevInZipCode[x] -
dfAllHomesStdDevInZipCode[x].mean()) / dfAllHomesStdDevInZipCode[x].std()  #calculate
z score for All Homes price each month

##Calculate z score comparing price across zipcode
dfAllHomesStdDevAcrossZipCode = dfAllHomesDiffFromMovAvePercent.copy() #dataframe to
hold z score across different zipcode
dfAllHomesStdDevAcrossZipCode['MeanHomePrice'] =
dfAllHomesStdDevAcrossZipCode.mean(axis=1)
dfAllHomesStdDevAcrossZipCode['StdDevHomePrice'] =
dfAllHomesStdDevAcrossZipCode.std(axis=1)
dfAllHomesStdDevAcrossZipCode = (dfAllHomesStdDevAcrossZipCode -
dfAllHomesStdDevAcrossZipCode['MeanHomePrice']) /
dfAllHomesStdDevAcrossZipCode['StdDevHomePrice']

##drop the two added columns so that match the dataframe size of df..InZipCode and
df..AcrossZipCode
dfAllHomesStdDevAcrossZipCode =
dfAllHomesStdDevAcrossZipCode.drop('MeanHomePrice',axis=1)
dfAllHomesStdDevAcrossZipCode =
dfAllHomesStdDevAcrossZipCode.drop('StdDevHomePrice',axis=1)

##Abnormally Detection method
##Find out which zipcode and its time frame in the dataframe has z score large than
1.5 in both df..InZipCode and df..AcrossZipCode dataframe
targetZscore = 1.5
targetZipcodeBoolean = ((dfAllHomesStdDevInZipCode > targetZscore) &
(dfAllHomesStdDevAcrossZipCode > targetZscore))
selectedZipCodes = pd.DataFrame(dfAllHomesMovingAve[targetZipcodeBoolean].sum(axis =
0) > 0)  #find out which zipcodes are over target zscore
selectedZipCodes[selectedZipCodes[0] == True] #extract only the zipcode that are over
zscore

### pivot a new dataframe moving targeted zipcodes into index joining the month-end
time (multi-level) index
TargetZipCode = pd.DataFrame(targetZipcodeBoolean.stack())  # stack zipcode from
column names into column
TargetZipCode.columns = ['PredictZipCode'] # specify column name
TargetZipCode = pd.DataFrame(TargetZipCode.reset_index()) # move index into column
dfPredictZipCode = pd.merge(df,TargetZipCode,on=['index','ZipCode']) #merge targeted
zipcode column for prediction into main dataframe(which is loaded from 9*.p files)
dfPredictZipCode.rename(columns={'index':'Month'}, inplace=True)
dfPredictZipCode['Month'] = dfPredictZipCode['Month'].apply(lambda x:
dt.datetime.strftime(x, '%Y-%m-%d'))
dfPredictZipCode.set_index(['Month','ZipCode'],inplace=True)
```

```
### Convert dfPredictZipcode.PredictZipCode from boolean to float, so false = 0, true
= 1
dfPredictZipCode.ix[dfPredictZipCode.PredictZipCode == False,'PredictZipCode'] = 0;
dfPredictZipCode.ix[dfPredictZipCode.PredictZipCode == True,'PredictZipCode'] = 1;
dfPredictZipCode.PredictZipCode = dfPredictZipCode.PredictZipCode.astype(float)
# print sum(dfPredictZipCode.PredictZipCode)


### Clean dataframe by removing np.nan, nan(string), ' '(space) and 0 in the cells and
fill those cells with back filling value
# finding dataframe cell with 'nan',' ' and 0 and then turn them into np.nan
for x in dfPredictZipCode.columns:
    if x == 'PredictZipCode':
        break
    nalist = dfPredictZipCode[x].isin(['nan',' ',0])
    dfPredictZipCode.ix[nalist,x] = np.nan #turn them in np.nan so that fillna
function can fill values to them

# use fillna function to back filling the value from following rows back to rows above
dfPredictZipCode = dfPredictZipCode.reset_index()
dfPredictZipCode.set_index("Month")
dfPredictZipCode = dfPredictZipCode.fillna(method='bfill') #to fill most of the empty
cells
dfPredictZipCode = dfPredictZipCode.fillna(method='ffill') #to fill the last few cells
dfPredictZipCode = dfPredictZipCode.reset_index()
dfPredictZipCode.set_index(['Month','ZipCode'],inplace=True)
dfPredictZipCode = dfPredictZipCode.drop('index',axis=1)
```

back

==Appendix - Parsing IRS Dataset ========================

```
###########process IRS dataset
### prepare 2001 IRS data
irsdata = pd.read_csv('./DataPfiles/01zp05ca.csv',header=4, na_values=['**','--'])
irsCol =
['ZipCode','NumberOfReturns','ExemptionsRtn','DepedentExemptionsAmt','AGI','SalariesWa
gesRtn','SalariesWagesAmt', \
          'TaxableInterestRtn','TaxableInterestAmt', \

'TotalTaxRtn','TotalTaxAmt','ScheduleCTotalRtn','ScheduleCTotalAmt','ScheduleFTotalRtn
','ScheduleFTotalAmt',\
          'ScheduleARtn','ScheduleAAmt'
          ]
irsdata.columns= irsCol
irsdata['ScheduleCTotalAmt'] = irsdata['ScheduleCTotalAmt'].astype(str)  #convert
dtype from float64 to string object
irsdata['ScheduleCTotalAmt'] = irsdata['ScheduleCTotalAmt'].apply(trim_fraction)
#remove the decimals from the string object
irsdata['ScheduleFTotalAmt'] = irsdata['ScheduleFTotalAmt'].astype(str)  #convert
dtype from float64 to string object
```

```
irsdata['ScheduleFTotalAmt'] = irsdata['ScheduleFTotalAmt'].apply(trim_fraction)
#remove the decimals from the string object
irsdata['ScheduleAAmt'] = irsdata['ScheduleAAmt'].astype(str)  #convert dtype from
float64 to string object
irsdata['ScheduleAAmt'] = irsdata['ScheduleAAmt'].apply(trim_fraction) #remove the
decimals from the string object

#remove unuseful characters at dataframe cell
chars_to_remove = ['*', '**', '--',',']
for col in irsdata.columns:
    irsdata[col] = irsdata[col].str.translate(None, ''.join(chars_to_remove))

zipcode = pd.read_csv('./DataPfiles/us_postal_codes.csv')
calzipcode = zipcode[zipcode.State == 'California']

irsdata['ZipCode'] = irsdata['ZipCode'].str.strip()  #strip white space in the cell
irsdata2001['AverageAGI'] = irsdata2001['AGI'].astype(int).astype(float) * 1000 /
irsdata2001['NumberOfReturns'].astype(int).astype(float)


irsdata2001full =
pd.DataFrame(columns=irsdata2001.columns.to_series().append(pd.Series("Month")))

#create monthly rows for each column and fill with the yearly value to each month
because I don't have monthly data
for x in range(0,irsdata2001.shape[0]):
    listtemp = []
    irsdata2001t = pd.DataFrame(columns=irsdata2001.columns)
    irsdata2001t = irsdata2001t.append([irsdata2001.iloc[x]]*12,ignore_index=True)
    for i in range(12):
        thisMonth = ("0%i"%(i+1,))[-2:]
        d = dateUtility.mkDateTime("2001-%s-02"%thisMonth)
        listtemp.append(dateUtility.mkLastOfMonth(d).strftime("%Y-%m-%d"))
    irsdata2001t["Month"] = pd.Series(listtemp)
    irsdata2001full = irsdata2001full.append(irsdata2001t, ignore_index=True)

irsdata2001full.set_index(['Month','ZipCode'],inplace=True)  #set dataframe index to
match with zipcode price dataframe

### merge IRS data into aggreated main dataframe
irsdatafull = irsdatafull.append(irsdata2001full)
```

back

==Appendix - Cleansing Combined Dataset =======================

```
### Clean irsdatafull dataframe by removing nan(string), ' '(space) and 0 in the cells
and fill those cells with back filling value

# finding dataframe cell with 'nan',' ' and 0 and then turn them into np.nan
for x in dfPredictZipCodeFinal.columns:
    if (dfPredictZipCodeFinal[x].dtypes == 'object'): # check if the column data type
is "O" object,
        dfPredictZipCodeFinal[x] = dfPredictZipCodeFinal[x].astype(str).str.strip() #
then strip any space in the cell
    nalist = dfPredictZipCodeFinal[x].isin(['nan',' ',0,'','.','None'])
    dfPredictZipCodeFinal.ix[nalist,x] = np.nan

#use fillna funcion to back filling the value from following rows back to rows above
dfPredictZipCodeFinal['PredictZipCode'].fillna(value=False,inplace=True) # Change the
NaN values created during the dataframe merge because of stack/unstack
dfPredictZipCodeFinal = dfPredictZipCodeFinal.reset_index() # need to reset index
because the current first level index is grouped by zipcode
dfPredictZipCodeFinal = dfPredictZipCodeFinal.set_index('level_0') #reset index to
month; rows for the same zipcode from different month are listed consecutively
dfPredictZipCodeFinal = dfPredictZipCodeFinal.fillna(method='bfill') #apply back
filling with fillna
dfPredictZipCodeFinal = dfPredictZipCodeFinal.fillna(method='ffill') #apply forward
filling to correct cells the last cells that didn't get fill with back filling
dfPredictZipCodeFinal = dfPredictZipCodeFinal.reset_index()
dfPredictZipCodeFinal.rename(columns={'level_0':'Month'}, inplace=True)
```

back

==Appendix - Re-Formating data frame =======================

```
dfPredictZipCodeFinalGraph = dfPredictZipCodeFinal.copy()  #copy into a new dataframe
for diagram plotting
dfPredictZipCodeFinalGraph.set_index(['Month', 'ZipCode'],inplace=True) #month and
zipcode won't be used for group; so make them index
dfPredictZipCodeFinalGraph.rename(columns = {'PredictZipCode':'ZipCode
Group'},inplace=True) #change name for diagram display
dfPredictZipCodeFinalGraph['ZipCode Group'].ix[dfPredictZipCodeFinalGraph['ZipCode
Group'] == 1.0,:] = 'Hot' #change name for diagram display
dfPredictZipCodeFinalGraph['ZipCode Group'].ix[dfPredictZipCodeFinalGraph['ZipCode
Group'] == 0.0,:] = 'Cold'#change name for diagram display
```

back

==Appendix - hot zipcodes with the most number of months

```
#for the zipcode ever consided to be targetted, find out for how many years they are
selected as target
TargetedZipCodeCount =
pd.DataFrame(dfPredictZipCodeFinal[dfPredictZipCodeFinal['PredictZipCode'] ==
True].groupby(['ZipCode'])['PredictZipCode'].count())
TargetedZipCodeCount.rename(columns = {'PredictZipCode':'PredictZipCodeCount'},
inplace = True)
TargetedZipCodeCount10 = TargetedZipCodeCount.sort('PredictZipCodeCount',
ascending=False)[:10] #top 10 zipcodes
print TargetedZipCodeCount10
```

```
#plot a bar chart
barchart = plt.bar(TargetedZipCodeCount10.index.astype(str),
TargetedZipCodeCount10['PredictZipCodeCount'].astype(int) ) #plot bar chart

plt.xlabel('ZipCode')
plt.ylabel('Count')
plt.title('Frequency of hot zipcode by number of months')
plt.legend()

plt.tight_layout()
plt.show()
```

back

==Appendix - Counting The Number of Hot/Cold zip codes ==============

```
## the number of targeted zipcode in different year.
dfPredictZipCodeFinal['Year'] = dfPredictZipCodeFinal.Month.str[:4]  # add a new
column to store which year(first 4 digits of date) based on Month
TargetZipCodeCount =
pd.DataFrame(dfPredictZipCodeFinal.groupby(['Year','PredictZipCode']).count()['ZipCode
'])
TargetZipCodeCount = TargetZipCodeCount.unstack(1)
TargetZipCodeCount = TargetZipCodeCount.rename(columns = {'ZipCode':'ZipCodeCount'})
TargetZipCodeCount.fillna(0,inplace=True)
print TargetZipCodeCount

TargetedZipCodeCount =
pd.DataFrame(dfPredictZipCodeFinal[dfPredictZipCodeFinal['PredictZipCode'] ==
True].groupby(['ZipCode'])['PredictZipCode'].count())
TargetedZipCodeCount.rename(columns = {'PredictZipCode':'PredictZipCodeCount'},
inplace = True)
TargetedZipCodeCount.sort('PredictZipCodeCount', ascending=False)

# plot the bar chart for hot zipcode count by year
zipcodeCountBarChart =
plt.bar(TargetZipCodeCount['ZipCodeCount'][0.0].reset_index()['Year'].astype(int),
TargetZipCodeCount['ZipCodeCount'][1.0].reset_index()[1.0])


plt.xlabel('Year')
plt.ylabel('Count')
plt.title('Targeted "Hot" Zipcode Count by Year')
plt.legend()

plt.tight_layout()
plt.show()
```

back

==Appendix - Comparing home price grouped by targeted zip code ===============

```
#pull only the columns related to selling price plus the label column PredictZipCode
dfPredictZipCodeFinalGraphHomePrice = dfPredictZipCodeFinalGraph[['1 Bedroom','2
Bedroom','3 Bedroom','4 Bedroom','5 or More Bedroom','All Homes','ZipCode Group']]
fig, axes = plt.subplots(1,6,figsize=(15,10)) #create fig with subplot of one row and
six columns
for ax in fig.axes: #loop through the axes
    ax.set_ylim(20000,800000) #set y axis range
    ax.set_xlabel('ZipCode Group')

dfPredictZipCodeFinalGraphHomePrice.boxplot(by='ZipCode Group',ax=axes, whis=[20, 80])
#plot boxplot grouped by zipcode group
```

back

==Appendix - Estimated Rent per Square Foot ====================

```
### Comparing "Estimated Rent per Square Foot" between hot and cold zipcodes

dfPredictZipCodeFinalGraph[['Estimated Rent per Square Foot','ZipCode
Group']].boxplot(by='ZipCode Group', whis=[20, 80]) #plot boxplot grouped by zipcode
group
dfPredictZipCodeFinalGraph[['Estimated Rent per Square Foot','ZipCode
Group']].groupby('ZipCode Group').mean()
```

back

==Appendix - Comparing Price-to-Rent Ratio ====================

```
### Comparing Price-to-Rent Ratio between hot and cold zipcodes

dfPredictZipCodeFinalGraph[['Price-to-Rent Ratio','ZipCode
Group']].boxplot(by='ZipCode Group', whis=[5, 95]) #plot boxplot grouped by zipcode
group
dfPredictZipCodeFinalGraph[['Price-to-Rent Ratio','ZipCode Group']].groupby('ZipCode
Group').mean()
```

Back

==Appendix - Comparing Average AGI between hot and cold zipcodes =====

```
## Comparing AGI between hot and cold zipcodes
```

```
fig, axes = plt.subplots(1,1,figsize=(15,10)) #create fig with subplot of one row and
six columns
for ax in fig.axes: #loop through the axes
    ax.set_ylim(10000,200000) #set y axis range
    ax.set_xlabel('ZipCode Group')

dfAGI = dfPredictZipCodeFinalGraph[['AverageAGI','ZipCode Group']]
dfAGI['AverageAGI'] = dfAGI['AverageAGI'].convert_objects(convert_numeric=True)
#convert AGI column from object to float because groupby would have thrown error
dfAGI.boxplot(by='ZipCode Group', ax=axes, whis=[5, 95], showmeans=True) #plot boxplot
grouped by zipcode group

dfAGI.dropna().groupby('ZipCode Group').describe().unstack()
```

Back

==Appendix - Plot the hot zipcodes on a map ===================

```
## plot the hot zipcodes on a map
import folium

#load zipcode list from file
zipcode = pd.read_csv('./DataPfiles/us_postal_codes.csv')
calzipcode = zipcode[zipcode.State == 'California']

#create map object and set the initial center position to Bakersfield
map_cal = folium.Map(location=[35.3733, -119.0187],
                   zoom_start=6,
                   tiles= "Stamen Terrain")

#find hot zipcodes sorted by the number of period considered as hot
TargetedZipCodeCount =
pd.DataFrame(dfPredictZipCodeFinal[dfPredictZipCodeFinal['PredictZipCode'] ==
True].groupby(['ZipCode'])['PredictZipCode'].count())
TargetedZipCodeCount.rename(columns = {'PredictZipCode':'PredictZipCodeCount'},
inplace = True)
TargetedZipCodeCount.sort('PredictZipCodeCount', ascending=False)

numOfZipCodeToMap = 30 #set the number of zipcode to be mapped
#find the top zipcodes to map
hotzipindex = calzipcode['Postal
Code'].isin(TargetedZipCodeCount.sort('PredictZipCodeCount',ascending=False).ix[:numOf
ZipCodeToMap,].index.astype(float))
dfHotzip = calzipcode.ix[hotzipindex,['Postal Code','Latitude','Longitude']] #extract
the matching zipcodes and its lot & long

# mark selected zipcodes on the map
```

```
for index, row in dfHotzip.iterrows():
    folium.Marker([row['Latitude'].astype(str), row['Longitude'].astype(str)],
popup=row['Postal Code'].astype(int).astype(str)).add_to(map_cal)

# print the map
map_cal
```

back

==Appendix - Generating balanced dataset sample ===================

```
dfPredictZipCodeFinalHot = dfPredictZipCodeFinal[dfPredictZipCodeFinal.PredictZipCode
== 1]
dfPredictZipCodeFinalCold = dfPredictZipCodeFinal[dfPredictZipCodeFinal.PredictZipCode
== 0]
print "Number of row of hot zipcode: ", dfPredictZipCodeFinalHot.shape[0]
print "Number of row of hot zipcode: ", dfPredictZipCodeFinalCold.shape[0]

# sampling with replacement to generate a dataset of 20000 rows with 10000 hot and
10000 cold zip codes
dfPredictZipCodeFinalHot = dfPredictZipCodeFinalHot.sample(10000,replace=True,
random_state=42)
dfPredictZipCodeFinalCold = dfPredictZipCodeFinalCold.sample(10000,replace=True,
random_state=42)
dfPredictZipCodeFinalBalanced =
dfPredictZipCodeFinalHot.append(dfPredictZipCodeFinalCold)
print "size of balanced dataset ready for training and testing: ",
dfPredictZipCodeFinalBalanced.shape[0]
```

back

==Appendix - Spliting Training and Testing Dataset ====================

```
###  Prepare training and testing dataset
from sklearn.cross_validation import train_test_split

# separate dataset into training and testing set
X = dfPredictZipCodeFinalBalanced
X.set_index(['Month'],inplace=True) #putting back index of Monthly
y = X['PredictZipCode']  #set dependent variable PredictZipCode from dataframe
X.drop('PredictZipCode',1,inplace=True) #remove PredictZipCode from dataframe

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

print "training dataset size: ", X_train.shape
print "training predictionlabel size: ", y_train.shape
print "testing dataset size: ", X_test.shape
print "testing predictionlabel size: ", y_test.shape
```

back

==Appendix - Training Logistic Regression ===================

```
 # function to print ROC curve; passing parameter C, the trained model and cross
validation object
def printROC(C, model, cvobj):
    mean_tpr = 0.0
    mean_fpr = np.linspace(0, 1, 100)
    all_tpr = []

    for i, (train, test) in enumerate(cvobj):
        probas_ = model.fit(X_train.iloc[train],
y_train.iloc[train]).predict_proba(X_train.iloc[test])
        # Compute ROC curve and area the curve
        fpr, tpr, thresholds = roc_curve(y_train[test], probas_[:, 1])
        mean_tpr += interp(mean_fpr, fpr, tpr)
        mean_tpr[0] = 0.0
        roc_auc = auc(fpr, tpr)
        plt.plot(fpr, tpr, lw=1, label='ROC fold %d (area = %0.5f)' % (i, roc_auc))

    plt.plot([0, 1], [0, 1], '--', color=(0.6, 0.6, 0.6), label='Luck')

    mean_tpr /= len(cv)
    mean_tpr[-1] = 1.0
    mean_auc = auc(mean_fpr, mean_tpr)
    plt.plot(mean_fpr, mean_tpr, 'k--',
             label='Mean ROC (area = %0.5f)' % mean_auc, lw=2)

    plt.xlim([-0.05, 1.05])
    plt.ylim([-0.05, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(str('ROC Curve: Logistic Regression with C=' + str(C)))
    plt.legend(loc="lower right")
    return plt

# Prepare cross validation training data
KFoldcv = StratifiedKFold(y_train, n_folds=6, random_state=42) #divide training set
into 6 folds

##########################################
## train model using X_train with C=100000
C=100000
clfc5 = linear_model.LogisticRegression(C=C)
print "Logistic Regression CV start: ",datetime.now()
scores = cross_validation.cross_val_score(clfc5, X_train, y_train, cv=KFoldcv,
scoring='precision')
print "Logistic Regression CV end: ",datetime.now()
print str("Logistic Regression with C=" + str(C))
```

```
print("Precision mean with standard deviation: %0.5f (+/- %0.5f)" % (scores.mean(),
scores.std() * 2))
# Run ROC analysis and plot ROC curves
plt = printROC(C,clfc5, KFoldcv)
plt.show()

##########################################
## train model using X_train with C=1000
C=1000
clfc3 = linear_model.LogisticRegression(C=C)
print "Logistic Regression CV start: ",datetime.now()
scores = cross_validation.cross_val_score(clfc3, X_train, y_train, cv=KFoldcv,
scoring='precision')
print "Logistic Regression CV end: ",datetime.now()
print str("Logistic Regression with C=" + str(C))
print("Precision mean with standard deviation: %0.5f (+/- %0.5f)" % (scores.mean(),
scores.std() * 2))
# Run ROC analysis and plot ROC curves
plt = printROC(C,clfc3, KFoldcv)
plt.show()

##########################################
## train model using X_train with C=10
C=10
clfc1 = linear_model.LogisticRegression(C=C)
print "Logistic Regression CV start: ",datetime.now()
scores = cross_validation.cross_val_score(clfc1, X_train, y_train, cv=KFoldcv,
scoring='precision')
print "Logistic Regression CV end: ",datetime.now()
print str("Logistic Regression with C=" + str(C))
print("Precision mean with standard deviation: %0.5f (+/- %0.5f)" % (scores.mean(),
scores.std() * 2))
# Run ROC analysis and plot ROC curves
plt = printROC(C,clfc1, KFoldcv)
plt.show()
```

Back

==Appendix - Training Support Vector Classifier ===================

```
# Apply PCA to reduce feature and train with SVM
from sklearn import cross_validation
import matplotlib.pyplot as plt
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.grid_search import GridSearchCV
from sklearn.svm import SVC
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from datetime import datetime,date,time
from sklearn.metrics import confusion_matrix
from sklearn.cross_validation import StratifiedKFold
from sklearn.metrics import classification_report
%matplotlib inline

# Prepare cross validation training data
KFoldcv = StratifiedKFold(y_train, n_folds=6, random_state=42) #divide training set
into 6 folds

pca = PCA(n_components=2)

# Maybe some original features where good
selection = SelectKBest(k=1)

# Build estimator from PCA and Univariate selection:
combined_features = FeatureUnion([("pca", pca), ("univ_select", selection)])

# Use combined features to transform dataset:
# print "SVC training start: ",datetime.now()
X_features = combined_features.fit_transform(X_train, y_train)#.transform(X_train)

svm = SVC()

# Do grid search over k, n_components and C:
pipeline = Pipeline([("features", combined_features), ("svm", svm)])
param_grid = dict(features__pca__n_components=[1, 2, 3],
                  features__univ_select__k=[1, 2],
                  svm__kernel=['rbf','sigmoid'],
                  svm__C=[0.1, 1, 10])
scores = ['precision']
for score in scores:
    print("# Tuning hyper-parameters for %s" % score)
    print()

    clf = GridSearchCV(pipeline, param_grid=param_grid, cv=KFoldcv, scoring='%s' %
score)
    clf.fit(X_features, y_train)

    print("Best parameters set found on development set:")
    print()
    print(clf.best_params_)
    print()
    print("Grid scores on development set:")
    print()

    for params, mean_score, scores in clf.grid_scores_:
        print("%0.3f (+/-%0.03f) for %r" % (mean_score, scores.std() / 2, params))
    print()
    print()

    print("Detailed classification report:")
    print()
    print("The model is trained on the full development set.")
    print("The scores are computed on the full evaluation set.")
    print()
    X_features_test = combined_features.transform(X_test) #.transform(X_test)
    y_true, y_pred = y_test, clf.predict( X_features_test)
    print(classification_report(y_true, y_pred))
    print()
```

back