Detroit: Fighting Blight in Every Community Capstone Project

In [ ]:
```
by Janice Santiago
```

1 The purpose of this project is to discover methods and techniques to accurately determine the blight status of buildings using open source data now freely available to governments at all levels.

1.1 Import Libraries 1.2 Load Initial Helper Functions 1.2.2 Data has been cleaned and loaded into mongodb 1.2.3 Load Transaction Data from MongoDB 1.3 Summarize the Data 1.3.1 Look at the Data 1.3.2 Make the class field Numeric 1.3.3 Look at class distribution 1.3.4 Create an equal number of transactions for each group 1.3.5 Verify the Correct Counts and formating for the new group 1.3.6 Prepare the data to be modeled 1.3.7 Scale the data to be between 0 and 1 1.4 Evaluate Some Algorithms 1.4.1 Setup test and training datasets 1.4.2 Build Models 1.4.3 Test options and evaluation metric 1.4.5 evaluate each model in turn 1.4.6 Compare Algorithms¶ 1.4.7 Deep Learning Neural Network for Classification 1.5 Helper Files and Routines for creating the transaction data 1.6 Plotly Charting Data¶ Neighborhoods that receive the most complaints¶ The most common complaints¶ 1.7 Predicting the following month Demolition total based on the previous months¶ plot of pass monthly demolition and prediction for the next's month

1.7 Conclusion The conclusion is that a model's estimate accurary for the Neural Network produced very accurate statistics of 96%. This was a very basic model without any attemts to improve it. The prediction rate was 89%. A Naive Bayes model did produced a statistic for accuracy of 97% but it was an advanced implementation. As a result of this research I conclude that this is a viable approach to predicting current and future blight in buildings. This work can be coupled with inspection data to determine if using both dataset together could enable anticipation of future problems before they occur.

1.1 Import Libraries

In [1]:
```
import pandas as pd
import sqlite3
%pylab inline
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
from pandas import DataFrame, Series
from scipy.sparse import *
from scipy import *
import requests # pip install requests
```

```
Populating the interactive namespace from numpy and matplotlib
```

1.2 Load Initial Helper Functions

In [2]:

```
import json
import pymongo # pip install pymongo
#from pymongo import Connection
from pymongo import MongoClient
c = MongoClient()

def save_to_mongo(data, mongo_db, mongo_db_coll, **mongo_conn_kw):

    # Connects to the MongoDB server running on
    # localhost:27017 by default

    client = pymongo.MongoClient(**mongo_conn_kw)

    # Get a reference to a particular database

    db = client[mongo_db]

    # Reference a particular collection in the database

    coll = db[mongo_db_coll]

    # Perform a bulk insert and  return the IDs

    return coll.insert(data)

def insert_in_mongo(data, mongo_db, mongo_db_coll, **mongo_conn_kw):

    # Connects to the MongoDB server running on
    # localhost:27017 by default

    client = pymongo.MongoClient(**mongo_conn_kw)

    # Get a reference to a particular database

    db = client[mongo_db]

    # Reference a particular collection in the database

    coll = db[mongo_db_coll]

    # Perform a bulk insert and  return the IDs

    return coll.insert(data)
def load_from_mongo(mongo_db, mongo_db_coll, return_cursor=False,
                    criteria=None, projection=None, **mongo_conn_kw):

    # Optionally, use criteria and projection to limit the data that is
    # returned as documented in
    # http://docs.mongodb.org/manual/reference/method/db.collection.find/

    # Consider leveraging MongoDB's aggregations framework for more
    # sophisticated queries.

    client = pymongo.MongoClient(**mongo_conn_kw)
    db = client[mongo_db]
    coll = db[mongo_db_coll]
```

```
    if criteria is None:
        criteria = {}

    if projection is None:
        cursor = coll.find(criteria)
    else:
        cursor = coll.find(criteria, projection)

    # Returning a cursor is recommended for large amounts of data

    if return_cursor:
        return cursor
    else:
        return [ item for item in cursor ]

def save_json(filename, data):
    with io.open('{0}.json'.format(filename),
                 'w', encoding='utf-8') as f:
        f.write(unicode(json.dumps(data, ensure_ascii=False)))

def load_json(filename):
    with io.open('{0}.json'.format(filename),
                 encoding='utf-8') as f:
        return f.read()
```

1.2.2 Data has been cleaned and loaded into mongodb

mongoimport -d mydb -c 311Issues --type csv --file Improve_Detroit__Submitted_Issues.csv --headerline mongoimport -d mydb -c dlbaDemo --type csv --file DLBA_Demolitions2.csv --headerline mongoimport -d mydb -c detroitDemo --type csv --file Detroit_Demolitions.csv --headerline mongoimport -d mydb -c blightLocals --type csv --file Blight_Violation_Locations.csv --headerline mongoimport -d mydb -c blightVioInfo --type csv --file Blight_Violations.csv --headerline mongoimport -d mydb -c bldPermits --type csv --file Building_Permits.csv --headerline mongoimport -d mydb -c upcomingDemo --type csv --file Upcoming_Detroit_Demolitions.csv --headerline mongoimport -d mydb -c addressTotals --type csv --file addressTotals.csv --headerline mongoimport -d mydb -c bigBuildings --type csv --file bigBuildings.csv --headerline mongoimport -d mydb -c goodBlightDemoVioGeo --type csv --file goodBlightDemoVioGeo.csv --headerline mongoimport -d mydb -c allDemoTransBlight --type csv --file allDemoTransBlight.csv --headerline

1.2.3 Load Transaction Data from MongoDB

In [5]:
```
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)

mongo_a_search_results= load_from_mongo('mydb', 'allDemoTransBlight')
dftotals = pd.DataFrame(mongo_a_search_results)
del dftotals['_id']
#del dftotals['zipcode']
#del dftotals['Year']
#del dftotals['FullPayment']
```

1.3 Summarize the Data

1.3.1 Look at the Data

In [6]:
```
dftotals.head()
```

Out[6]:

	Blight	Compliance	FullPayment	NoPayment	NumViolations	PartialPayment	Violati
0	no	0	1	1	2	0	COVIN
1	no	0	0	1	1	0	MAND
2	no	0	0	2	2	0	MEND
3	no	0	0	1	1	0	TULLE
4	no	0	0	1	1	0	WARR

5 rows × 26 columns

In [9]:
```
dftotals.tail()
```

Out[9]:

	Blight	Compliance	FullPayment	NoPayment	NumViolations	PartialPayment	V
23923	no	1	1	0	1	0	C
23924	yes	1	0	2	2	0	M
23925	no	0	1	0	1	0	W
23926	no	1	0	1	1	0	G
23927	yes	2	0	2	2	0	R

5 rows × 26 columns

1.3.2 Make the class field Numeric

In [10]:
```
dftotals['Blight'] = dftotals['Blight'].str.replace('yes', '1')
dftotals['Blight'] = dftotals['Blight'].str.replace('no', '0')
```

1.3.3 Look at class distribution

In [11]:
```
dftotals.groupby('Blight').size()
```

Out[11]:
```
Blight
0    21545
1     2383
dtype: int64
```

In [12]:
```
dftotals.head()
```

Out[12]:

	Blight	Compliance	FullPayment	NoPayment	NumViolations	PartialPayment	Violati
0	0	0	1	1	2	0	COVIN
1	0	0	0	1	1	0	MAND
2	0	0	0	2	2	0	MEND
3	0	0	0	1	1	0	TULLE
4	0	0	0	1	1	0	WARR

5 rows × 26 columns

1.3.4 Create an equal number of transactions for each group

In [13]:
```
dfNo = dftotals.loc[dftotals['Blight'] == '0']
```

In [ ]:
```

```

In [ ]:
```

```

In [14]:
```
# split into input (X) and output (Y) variables
#datasetNo = dfNo[:1017]
datasetNo = dfNo[:2383]
```

In [15]:
```
datasetNo.tail()
```

Out[15]:

	Blight	Compliance	FullPayment	NoPayment	NumViolations	PartialPayment	Vio
2614	0	0	0	3	3	0	ME
2615	0	0	0	1	1	0	SE
2616	0	1	0	1	1	0	VIS
2617	0	1	0	2	2	0	DR
2618	0	0	1	0	1	0	FR

5 rows × 26 columns

In [16]:
```
dfYes = dftotals.loc[dftotals['Blight'] != '0']
#dfYes = dftotals[(dftotals['Blight']== 'yes')]
datasetYes = dfYes[:2383]
```

In [17]:
```
datasetYes.head()
```

Out[17]:

	Blight	Compliance	FullPayment	NoPayment	NumViolations	PartialPayment	Viola
22	1	1	0	5	5	0	BALD
38	1	0	0	2	2	0	NOT
40	1	4	2	2	4	0	14TH
60	1	0	0	1	1	0	ASBU
64	1	0	0	6	6	0	BEAC

5 rows × 26 columns

In [18]:
```
frames = [datasetYes, datasetNo]


result  = pd.concat(frames)
```

In [19]:
```
result.describe()
```

Out[19]:

	Compliance	FullPayment	NoPayment	NumViolations	PartialPayment	Violatio
count	4766.000000	4766.000000	4766.000000	4766.000000	4766.000000	4766.00
mean	0.800252	0.311792	2.253252	2.620436	0.032522	9851.67
std	1.247860	0.755626	2.867604	2.973700	0.229950	5333.31
min	0.000000	0.000000	0.000000	1.000000	0.000000	0.00000
25%	0.000000	0.000000	1.000000	1.000000	0.000000	5922.75
50%	0.000000	0.000000	1.000000	2.000000	0.000000	11300.0
75%	1.000000	0.000000	3.000000	3.000000	0.000000	12559.5
max	18.000000	12.000000	60.000000	60.000000	6.000000	119433.

8 rows × 23 columns

In [20]:
```
dftotals.head()
```

Out[20]:

	Blight	Compliance	FullPayment	NoPayment	NumViolations	PartialPayment	Violati
0	0	0	1	1	2	0	COVIN
1	0	0	0	1	1	0	MAND
2	0	0	0	2	2	0	MEND
3	0	0	0	1	1	0	TULLE
4	0	0	0	1	1	0	WARR

5 rows × 26 columns

cols = dftotals.columns.tolist() cols = cols[1:] + cols[0:1] dftotals = dftotals[cols]dftotals.head()

In [21]:
```
result.groupby('Blight').size()
```

Out[21]:
```
Blight
0    2383
1    2383
dtype: int64
```

In [22]:
```
result.head()
```

Out[22]:

	Blight	Compliance	FullPayment	NoPayment	NumViolations	PartialPayment	Viola
22	1	1	0	5	5	0	BALL
38	1	0	0	2	2	0	NOT
40	1	4	2	2	4	0	14TH
60	1	0	0	1	1	0	ASBU
64	1	0	0	6	6	0	BEAC

5 rows × 26 columns

dftotals = result.sort(['address'], ascending=[1]) dftotals = result

In [23]:
```
dftotals = result
dftotals.describe()
```

Out[23]:

	Compliance	FullPayment	NoPayment	NumViolations	PartialPayment	Violatio
count	4766.000000	4766.000000	4766.000000	4766.000000	4766.000000	4766.00
mean	0.800252	0.311792	2.253252	2.620436	0.032522	9851.67
std	1.247860	0.755626	2.867604	2.973700	0.229950	5333.31
min	0.000000	0.000000	0.000000	1.000000	0.000000	0.00000
25%	0.000000	0.000000	1.000000	1.000000	0.000000	5922.75
50%	0.000000	0.000000	1.000000	2.000000	0.000000	11300.0
75%	1.000000	0.000000	3.000000	3.000000	0.000000	12559.5
max	18.000000	12.000000	60.000000	60.000000	6.000000	119433.

8 rows × 23 columns

dfNewTotals = pd.DataFrame(dftotals['Blight']) dfNewTotals['address'] = dftotals['address']
dfNewTotals['NumViolations'] = dftotals['NumViolations'] #dftotals = pd.DataFrame(dfNewTotals)

In [24]:
```
dftotals.describe()
```

Out[24]:

	Compliance	FullPayment	NoPayment	NumViolations	PartialPayment	Violatio
count	4766.000000	4766.000000	4766.000000	4766.000000	4766.000000	4766.00
mean	0.800252	0.311792	2.253252	2.620436	0.032522	9851.67
std	1.247860	0.755626	2.867604	2.973700	0.229950	5333.31
min	0.000000	0.000000	0.000000	1.000000	0.000000	0.00000
25%	0.000000	0.000000	1.000000	1.000000	0.000000	5922.75
50%	0.000000	0.000000	1.000000	2.000000	0.000000	11300.0
75%	1.000000	0.000000	3.000000	3.000000	0.000000	12559.5
max	18.000000	12.000000	60.000000	60.000000	6.000000	119433.

8 rows × 23 columns

cols = dftotals.columns.tolist() cols = cols[1:] + cols[0:1] dftotals = dftotals[cols]

In [25]:
```
del dftotals['address']
```

#del dftotals['SalePrice']

In [26]:
```
#del dftotals['electrical']
```

In [27]:
```
del dftotals['ViolationStreetName']
```

1.3.5 Verify the Correct Counts and formating for the new group

In [28]:
```
dftotals.head()
```

Out[28]:

	Blight	Compliance	FullPayment	NoPayment	NumViolations	PartialPayment	Viola
22	1	1	0	5	5	0	1000
38	1	0	0	2	2	0	1000
40	1	4	2	2	4	0	1000
60	1	0	0	1	1	0	1001
64	1	0	0	6	6	0	1001

5 rows × 24 columns

In [29]:
```
dftotals.describe()
```

Out[29]:

	Compliance	FullPayment	NoPayment	NumViolations	PartialPayment	Violatio
count	4766.000000	4766.000000	4766.000000	4766.000000	4766.000000	4766.00
mean	0.800252	0.311792	2.253252	2.620436	0.032522	9851.67
std	1.247860	0.755626	2.867604	2.973700	0.229950	5333.31
min	0.000000	0.000000	0.000000	1.000000	0.000000	0.00000
25%	0.000000	0.000000	1.000000	1.000000	0.000000	5922.75
50%	0.000000	0.000000	1.000000	2.000000	0.000000	11300.0
75%	1.000000	0.000000	3.000000	3.000000	0.000000	12559.5
max	18.000000	12.000000	60.000000	60.000000	6.000000	119433.

8 rows × 23 columns

In [30]:
```
dftotals.head()
```

Out[30]:

	Blight	Compliance	FullPayment	NoPayment	NumViolations	PartialPayment	Viola
22	1	1	0	5	5	0	1000
38	1	0	0	2	2	0	1000
40	1	4	2	2	4	0	1000
60	1	0	0	1	1	0	1001
64	1	0	0	6	6	0	1001

5 rows × 24 columns

1.3.6 Prepare the data to be modeled

In [37]:
```
#dfnewTotals['blight'] = dftotals['Blight']
totalValues = dftotals.values
```

In [38]:
```
totalValues
```

Out[38]:
```
array([[1L, 0L, 0L, ..., u'01/01/38476 12:00:00 AM', 2000L,
        ObjectId('57dd516539d4cac6412df5b8')],
       [1L, 0L, 0L, ..., u'01/01/38498 12:00:00 AM', 2000L,
        ObjectId('57dd516539d4cac6412df5b9')],
       [1L, 1L, 0L, ..., u'11/13/2013 0:00', 2013L,
        ObjectId('57dd516539d4cac6412df5ba')],
       ...,
       [0L, 0L, 1L, ..., u'8/28/2007 0:00', 2007L,
        ObjectId('57dd516539d4cac6412e0a96')],
       [0L, 0L, 2L, ..., u'9/28/2010 0:00', 2010L,
        ObjectId('57dd516539d4cac6412e0a97')],
       [0L, 2L, 2L, ..., u'5/20/2008 0:00', 2008L,
        ObjectId('57dd516539d4cac6412e0a98')]], dtype=object)
```

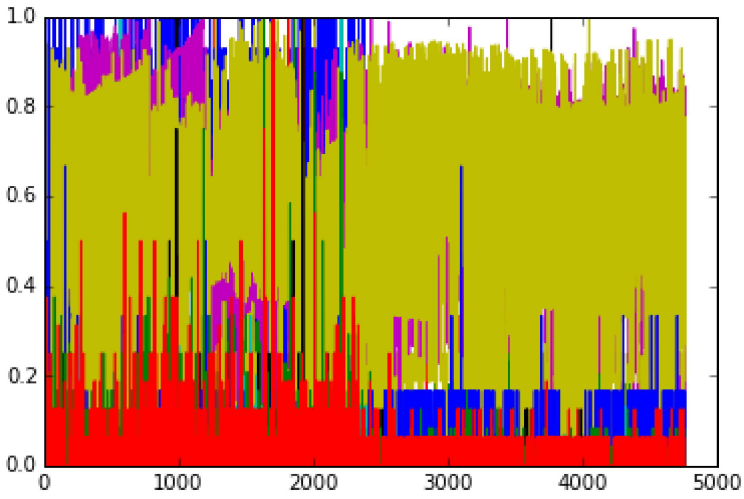
result.groupby('Blight').size()

In [39]:
```
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
```

1.3.7 Scale the data to be between 0 and 1

#normalize the dataset scaler = MinMaxScaler(feature_range=(0, 1)) dfnewTotals = scaler.fit_transform(totalValues)

In [35]:
```
import pandas
import matplotlib.pyplot as plt
#dataset = pandas.read_csv('international-airline-passengers.csv', usecols=
[1], engine='python', skipfooter=3)
plt.plot(dfnewTotals)
plt.show()
```



In [36]:
```
dfnewTotals
```

Out[36]:
```
array([[ 1.        ,  0.05555556,  0.        , ...,  0.        ,
         0.04166667,  0.0625    ],
       [ 1.        ,  0.        ,  0.        , ...,  0.        ,
         0.08333333,  0.        ],
       [ 1.        ,  0.22222222,  0.16666667, ...,  0.        ,
         0.        ,  0.        ],
       ...,
       [ 0.        ,  0.05555556,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        ,  0.05555556,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.08333333, ...,  0.        ,
         0.        ,  0.        ]])
```

In [37]:
```
dtfsum = dfnewTotals
```

In [38]:
```
data = dtfsum.astype('float32')
```

In [39]:
```
data
```

Out[39]:
```
array([[ 1.        ,  0.05555556,  0.        , ...,  0.        ,
         0.04166667,  0.0625    ],
       [ 1.        ,  0.        ,  0.        , ...,  0.        ,
         0.08333334,  0.        ],
       [ 1.        ,  0.22222222,  0.16666667, ...,  0.        ,
         0.        ,  0.        ],
       ...,
       [ 0.        ,  0.05555556,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        ,  0.05555556,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.08333334, ...,  0.        ,
         0.        ,  0.        ]], dtype=float32)
```

In [40]:
```
# normalize the dataset
#scaler = MinMaxScaler(feature_range=(0, 1))
#data = scaler.fit_transform(data)
```

In [41]:
```
data.shape
```

Out[41]:
```
(4766L, 24L)
```

In [42]:
```
dataset = data.reshape((4766L, 24))
```

In [43]:
```
dataset
```

Out[43]:
```
array([[ 1.        ,  0.05555556,  0.        , ...,  0.        ,
         0.04166667,  0.0625    ],
       [ 1.        ,  0.        ,  0.        , ...,  0.        ,
         0.08333334,  0.        ],
       [ 1.        ,  0.22222222,  0.16666667, ...,  0.        ,
         0.        ,  0.        ],
       ...,
       [ 0.        ,  0.05555556,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        ,  0.05555556,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.08333334, ...,  0.        ,
         0.        ,  0.        ]], dtype=float32)
```

1.4 Evaluate Some Algorithms

1.4.1 Setup test and training datasets

In [44]:
```
# split into train and test sets
train_size = int(len(dataset) * 0.67)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]
print(len(train), len(test))
```

```
(3193, 1573)
```

In [45]:
```
# split into input (X) and output (Y) variables
trainX = train[:,1:24]
trainY = train[:,0]
testX = test[:,1:24]
testY = test[:,0]
```

In [46]:
```
X = dataset[:,1:24]
Y = dataset[:,0]
```

In [47]:
```
from sklearn import cross_validation
```

In [48]:
```
validation_size = 0.20
```

In [49]:
```
X_train, X_validation, Y_train, Y_validation = cross_validation.train_test_spl
it(X, Y, test_size=validation_size, random_state=seed)
```

In [50]:
```
X
```

Out[50]:
```
array([[ 0.05555556,  0.        ,  0.08333334, ...,  0.        ,
         0.04166667,  0.0625    ],
       [ 0.        ,  0.        ,  0.03333334, ...,  0.        ,
         0.08333334,  0.        ],
       [ 0.22222222,  0.16666667,  0.03333334, ...,  0.        ,
         0.        ,  0.        ],
       ...,
       [ 0.05555556,  0.        ,  0.01666667, ...,  0.        ,
         0.        ,  0.        ],
       [ 0.05555556,  0.        ,  0.03333334, ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        ,  0.08333334,  0.        , ...,  0.        ,
         0.        ,  0.        ]], dtype=float32)
```

In [51]:
```
Y
```

Out[51]:
```
array([ 1.,  1.,  1., ...,  0.,  0.,  0.], dtype=float32)
```

In [52]:
```
trainX
```

Out[52]:
```
array([[ 0.05555556,  0.        ,  0.08333334, ...,  0.        ,
         0.04166667,  0.0625    ],
       [ 0.        ,  0.        ,  0.03333334, ...,  0.        ,
         0.08333334,  0.        ],
       [ 0.22222222,  0.16666667,  0.03333334, ...,  0.        ,
         0.        ,  0.        ],
       ...,
       [ 0.05555556,  0.        ,  0.01666667, ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.01666667, ...,  0.        ,
         0.        ,  0.0625    ],
       [ 0.        ,  0.08333334,  0.        , ...,  0.        ,
         0.        ,  0.0625    ]], dtype=float32)
```

In [53]:
```
trainY
```

Out[53]:
```
array([ 1.,  1.,  1., ...,  0.,  0.,  0.], dtype=float32)
```

In [54]:
```
trainX.shape
```

Out[54]:
```
(3193L, 23L)
```

In [55]:
```
trainY.shape
```

Out[55]:
```
(3193L,)
```

In [56]:
```
print(dftotals.dtypes)
```

```
Blight                    object
Compliance                 int64
FullPayment                int64
NoPayment                  int64
NumViolations              int64
PartialPayment             int64
ViolationStreetNumber      int64
Year                       int64
certificate                int64
changeUse                  int64
closure                    int64
complywith                 int64
container                  int64
defectiveStructure         int64
electrical                 int64
emergencyDanger            int64
graffiti                   int64
illegalOccupation          int64
latitude                 float64
longitude                float64
notMaintained              int64
snowice                    int64
waste                      int64
weedsVehicles              int64
dtype: object
```

In [57]:
```
pd.set_option('precision', 2)
print(dftotals.describe())
# correlation
print(dftotals.corr(method='pearson'))
```

	Compliance	FullPayment	NoPayment	NumViolations	PartialPayment	\
count	4766.0	4766.0	4766.0	4766.0	4.8e+03	
mean	0.8	0.3	2.3	2.6	3.3e-02	
std	1.2	0.8	2.9	3.0	2.3e-01	
min	0.0	0.0	0.0	1.0	0.0e+00	
25%	0.0	0.0	1.0	1.0	0.0e+00	
50%	0.0	0.0	1.0	2.0	0.0e+00	
75%	1.0	0.0	3.0	3.0	0.0e+00	
max	18.0	12.0	60.0	60.0	6.0e+00	

	ViolationStreetNumber	Year	certificate	changeUse	closure	\
count	4766.0	4766.0	4766.0	4.8e+03	4.8e+03	
mean	9851.7	2008.8	1.2	8.4e-04	2.1e-04	
std	5333.3	2.4	1.9	3.5e-02	1.4e-02	
min	0.0	2000.0	0.0	0.0e+00	0.0e+00	
25%	5922.8	2009.0	0.0	0.0e+00	0.0e+00	
50%	11300.0	2009.0	0.0	0.0e+00	0.0e+00	
75%	12559.5	2009.0	2.0	0.0e+00	0.0e+00	
max	119433.0	2014.0	22.0	2.0e+00	1.0e+00	

	...	electrical	emergencyDanger	graffiti	\
count	...	4.8e+03	4766.0	4.8e+03	
mean	...	2.1e-04	0.1	4.8e-03	
std	...	1.4e-02	0.6	1.5e-01	
min	...	0.0e+00	0.0	0.0e+00	
25%	...	0.0e+00	0.0	0.0e+00	
50%	...	0.0e+00	0.0	0.0e+00	
75%	...	0.0e+00	0.0	0.0e+00	
max	...	1.0e+00	15.0	8.0e+00	

	illegalOccupation	latitude	longitude	notMaintained	snowice	waste	\
count	4.8e+03	4.8e+03	4.8e+03	4.8e+03	4.8e+03	4766.0	
mean	8.7e-02	4.2e+01	-8.3e+01	2.4e-02	6.2e-02	0.6	
std	5.1e-01	3.6e-02	9.7e-02	2.3e-01	2.8e-01	1.2	
min	0.0e+00	4.2e+01	-8.3e+01	0.0e+00	0.0e+00	0.0	
25%	0.0e+00	4.2e+01	-8.3e+01	0.0e+00	0.0e+00	0.0	
50%	0.0e+00	4.2e+01	-8.3e+01	0.0e+00	0.0e+00	0.0	
75%	0.0e+00	4.2e+01	-8.3e+01	0.0e+00	0.0e+00	1.0	
max	1.5e+01	4.2e+01	-8.3e+01	8.0e+00	6.0e+00	24.0	

	weedsVehicles
count	4766.0
mean	0.4
std	0.9
min	0.0
25%	0.0
50%	0.0
75%	1.0

```
max             16.0

[8 rows x 23 columns]
                       Compliance  FullPayment  NoPayment  NumViolations  \
Compliance               1.0e+00      1.4e-01    6.4e-01        6.6e-01
FullPayment              1.4e-01      1.0e+00   -7.4e-02        1.9e-01
NoPayment                6.4e-01     -7.4e-02    1.0e+00        9.5e-01
NumViolations            6.6e-01      1.9e-01    9.5e-01        1.0e+00
PartialPayment           3.7e-02      2.5e-02    4.7e-02        1.3e-01
ViolationStreetNumber   -3.3e-02     -2.8e-02    2.9e-02        2.3e-02
Year                     1.4e-02     -1.0e-02    1.7e-01        1.3e-01
certificate              9.4e-01      1.1e-01    6.6e-01        6.7e-01
changeUse                7.5e-02     -9.8e-03    7.4e-02        6.9e-02
closure                  4.9e-02     -6.0e-03    3.9e-02        3.6e-02
complywith               4.1e-03      7.1e-02   -7.9e-03        9.8e-03
container               -1.3e-01      8.0e-02   -3.3e-02       -3.0e-03
defectiveStructure       2.0e-01      2.3e-03    3.3e-01        3.2e-01
electrical               3.7e-02     -6.0e-03    6.4e-02        6.0e-02
emergencyDanger          2.6e-01      3.1e-02    4.2e-01        4.1e-01
graffiti                 1.1e-01      1.3e-02    1.1e-01        2.2e-01
illegalOccupation        2.2e-01     -4.0e-03    3.9e-01        3.8e-01
latitude                -5.2e-02     -3.5e-02    1.3e-02        6.1e-03
longitude                4.3e-03     -1.0e-02   -7.3e-03       -1.1e-02
notMaintained            1.7e-01      3.2e-02    2.8e-01        3.4e-01
snowice                 -6.4e-02      7.5e-02   -1.9e-02       -6.6e-04
waste                    9.1e-03      1.2e-01    5.5e-01        5.9e-01
weedsVehicles           -3.2e-02      1.2e-01    4.6e-01        5.0e-01

                       PartialPayment  ViolationStreetNumber     Year  \
Compliance                    3.7e-02               -3.3e-02  1.4e-02
FullPayment                   2.5e-02               -2.8e-02 -1.0e-02
NoPayment                     4.7e-02                2.9e-02  1.7e-01
NumViolations                 1.3e-01                2.3e-02  1.3e-01
PartialPayment                1.0e+00                4.3e-03  3.6e-03
ViolationStreetNumber         4.3e-03                1.0e+00  1.3e-02
Year                          3.6e-03                1.3e-02  1.0e+00
certificate                   4.4e-02               -2.5e-02  1.8e-02
changeUse                    -3.3e-03               -4.3e-03  2.2e-03
closure                      -2.0e-03                1.4e-02  7.4e-03
complywith                   -3.7e-03               -3.0e-02  5.1e-03
container                     1.7e-02                2.1e-02 -2.0e-02
defectiveStructure           -7.2e-03                1.1e-02  7.1e-03
electrical                   -2.0e-03                5.9e-03  1.9e-02
emergencyDanger              -1.5e-03                5.9e-03  4.3e-02
graffiti                     -4.7e-03               -2.1e-03 -7.1e-02
illegalOccupation             2.2e-02               -8.9e-03  4.4e-02
latitude                      4.2e-04                5.8e-01 -3.5e-02
longitude                    -2.5e-02               -1.3e-01 -5.2e-02
notMaintained                -6.8e-03                1.1e-02 -5.3e-04
snowice                       4.5e-03                4.1e-02  3.1e-02
waste                         1.5e-01                4.9e-02  1.4e-01
weedsVehicles                 9.8e-02                4.4e-02  1.8e-01

                       certificate  changeUse  closure      ...        \
Compliance                9.4e-01    7.5e-02  4.9e-02      ...
FullPayment               1.1e-01   -9.8e-03 -6.0e-03      ...
NoPayment                 6.6e-01    7.4e-02  3.9e-02      ...
```

```
NumViolations              6.7e-01    6.9e-02  3.6e-02      ...
PartialPayment             4.4e-02   -3.3e-03 -2.0e-03      ...
ViolationStreetNumber     -2.5e-02   -4.3e-03  1.4e-02      ...
Year                       1.8e-02    2.2e-03  7.4e-03      ...
certificate                1.0e+00    9.5e-02  2.9e-02      ...
changeUse                  9.5e-02    1.0e+00 -3.4e-04      ...
closure                    2.9e-02   -3.4e-04  1.0e+00      ...
complywith                -6.1e-03   -6.1e-04 -3.8e-04      ...
container                 -1.3e-01   -7.7e-03 -4.7e-03      ...
defectiveStructure         1.4e-01   -1.2e-03 -7.4e-04      ...
electrical                 3.7e-02   -3.4e-04 -2.1e-04      ...
emergencyDanger            2.5e-01    3.7e-02  9.9e-02      ...
graffiti                   1.2e-01   -7.8e-04 -4.8e-04      ...
illegalOccupation          2.3e-01   -4.0e-03  5.4e-02      ...
latitude                  -3.6e-02   -2.1e-03  7.6e-03      ...
longitude                 -1.6e-02    1.6e-02 -6.0e-03      ...
notMaintained              1.7e-01   -2.5e-03 -1.5e-03      ...
snowice                   -8.0e-02   -5.2e-03 -3.2e-03      ...
waste                     -1.8e-02   -1.2e-02 -7.1e-03      ...
weedsVehicles             -4.4e-02   -1.1e-02 -6.5e-03      ...

                        electrical  emergencyDanger  graffiti  \
Compliance                 3.7e-02          2.6e-01   1.1e-01
FullPayment               -6.0e-03          3.1e-02   1.3e-02
NoPayment                  6.4e-02          4.2e-01   1.1e-01
NumViolations              6.0e-02          4.1e-01   2.2e-01
PartialPayment            -2.0e-03         -1.5e-03  -4.7e-03
ViolationStreetNumber      5.9e-03          5.9e-03  -2.1e-03
Year                       1.9e-02          4.3e-02  -7.1e-02
certificate                3.7e-02          2.5e-01   1.2e-01
changeUse                 -3.4e-04          3.7e-02  -7.8e-04
closure                   -2.1e-04          9.9e-02  -4.8e-04
complywith                -3.8e-04          4.2e-04  -8.6e-04
container                 -4.7e-03         -4.8e-02   5.4e-02
defectiveStructure        -7.4e-04          3.4e-02   2.7e-01
electrical                 1.0e+00          9.9e-02  -4.8e-04
emergencyDanger            9.9e-02          1.0e+00   1.1e-02
graffiti                  -4.8e-04          1.1e-02   1.0e+00
illegalOccupation          1.1e-01          8.7e-01   1.7e-02
latitude                   1.0e-02          7.2e-02  -1.4e-02
longitude                  1.5e-02          1.0e-01  -1.0e-03
notMaintained              1.2e-01          8.8e-02   5.6e-01
snowice                   -3.2e-03         -1.7e-02  -7.3e-03
waste                      4.6e-03          1.1e-01   3.6e-02
weedsVehicles             -6.5e-03          3.1e-02   1.8e-02

                        illegalOccupation  latitude  longitude  notMaintained
 \
Compliance                      2.2e-01  -5.2e-02    4.3e-03        1.7e-01

FullPayment                    -4.0e-03  -3.5e-02   -1.0e-02        3.2e-02

NoPayment                       3.9e-01   1.3e-02   -7.3e-03        2.8e-01

NumViolations                   3.8e-01   6.1e-03   -1.1e-02        3.4e-01

PartialPayment                  2.2e-02   4.2e-04   -2.5e-02       -6.8e-03
```

ViolationStreetNumber	-8.9e-03	5.8e-01	-1.3e-01	1.1e-02
Year	4.4e-02	-3.5e-02	-5.2e-02	-5.3e-04
certificate	2.3e-01	-3.6e-02	-1.6e-02	1.7e-01
changeUse	-4.0e-03	-2.1e-03	1.6e-02	-2.5e-03
closure	5.4e-02	7.6e-03	-6.0e-03	-1.5e-03
complywith	8.3e-03	-2.4e-02	1.0e-02	-2.7e-03
container	-3.6e-02	-1.5e-02	-1.2e-01	1.4e-02
defectiveStructure	3.7e-02	-7.1e-03	3.2e-02	4.5e-01
electrical	1.1e-01	1.0e-02	1.5e-02	1.2e-01
emergencyDanger	8.7e-01	7.2e-02	1.0e-01	8.8e-02
graffiti	1.7e-02	-1.4e-02	-1.0e-03	5.6e-01
illegalOccupation	1.0e+00	5.7e-02	1.0e-01	9.7e-02
latitude	5.7e-02	1.0e+00	3.6e-01	-7.8e-03
longitude	1.0e-01	3.6e-01	1.0e+00	1.9e-02
notMaintained	9.7e-02	-7.8e-03	1.9e-02	1.0e+00
snowice	-2.5e-02	1.1e-04	6.2e-02	-2.0e-02
waste	1.1e-01	1.3e-02	-4.4e-02	7.7e-02
weedsVehicles	2.3e-02	5.6e-02	1.6e-02	8.3e-02

	snowice	waste	weedsVehicles
Compliance	-6.4e-02	9.1e-03	-3.2e-02
FullPayment	7.5e-02	1.2e-01	1.2e-01
NoPayment	-1.9e-02	5.5e-01	4.6e-01
NumViolations	-6.6e-04	5.9e-01	5.0e-01
PartialPayment	4.5e-03	1.5e-01	9.8e-02
ViolationStreetNumber	4.1e-02	4.9e-02	4.4e-02
Year	3.1e-02	1.4e-01	1.8e-01
certificate	-8.0e-02	-1.8e-02	-4.4e-02
changeUse	-5.2e-03	-1.2e-02	-1.1e-02
closure	-3.2e-03	-7.1e-03	-6.5e-03
complywith	-5.7e-03	-1.0e-02	-1.2e-02
container	-3.3e-02	-4.0e-03	-7.6e-02
defectiveStructure	-8.0e-03	9.7e-02	1.0e-01
electrical	-3.2e-03	4.6e-03	-6.5e-03
emergencyDanger	-1.7e-02	1.1e-01	3.1e-02
graffiti	-7.3e-03	3.6e-02	1.8e-02
illegalOccupation	-2.5e-02	1.1e-01	2.3e-02
latitude	1.1e-04	1.3e-02	5.6e-02

```
longitude              6.2e-02 -4.4e-02        1.6e-02
notMaintained         -2.0e-02  7.7e-02        8.3e-02
snowice                1.0e+00 -2.8e-02       -4.5e-02
waste                 -2.8e-02  1.0e+00        4.8e-01
weedsVehicles         -4.5e-02  4.8e-01        1.0e+00

[23 rows x 23 columns]
```

In [58]:
```
# Load libraries
import pandas
import numpy
import matplotlib.pyplot as plt
from pandas.tools.plotting import scatter_matrix
from sklearn.preprocessing import StandardScaler
from sklearn import cross_validation
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.pipeline import Pipeline
from sklearn.grid_search import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.metrics import mean_squared_error
```

1.4.2 Build Models

1.4.3 Test options and evaluation metric

num_folds = 10 num_instances = len(X_train) seed = 7 scoring = 'mean_squared_error'

1.4.4 Spot Check Algorithms

models = [] models.append(('LR', LinearRegression()))

models.append(('LASSO', Lasso()))

models.append(('EN', ElasticNet()))

models.append(('KNN', KNeighborsRegressor())) models.append(('CART', DecisionTreeRegressor())) models.append(('SVR', SVR()))

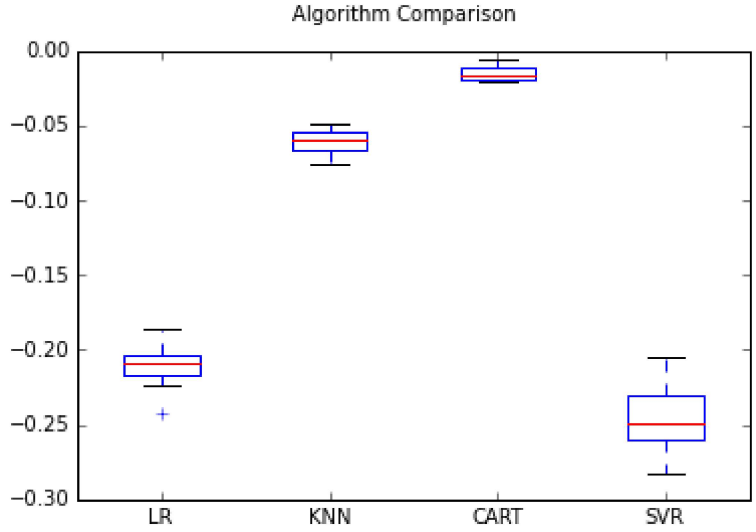
1.4.5 evaluate each model in turn

In [60]:
```
results = []
names = []
for name, model in models:
        kfold = cross_validation.KFold(n=num_instances, n_folds=num_folds, ran
dom_state=seed)
        cv_results = cross_validation.cross_val_score(model, X_train, Y_train,
 cv=kfold, scoring=scoring)
        results.append(cv_results)
        names.append(name)
        msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
        print(msg)
```

```
LR: -0.209986 (0.015409)
KNN: -0.060770 (0.008152)
CART: -0.015216 (0.004986)
SVR: -0.244961 (0.024291)
```

1.4.6 Compare Algorithms

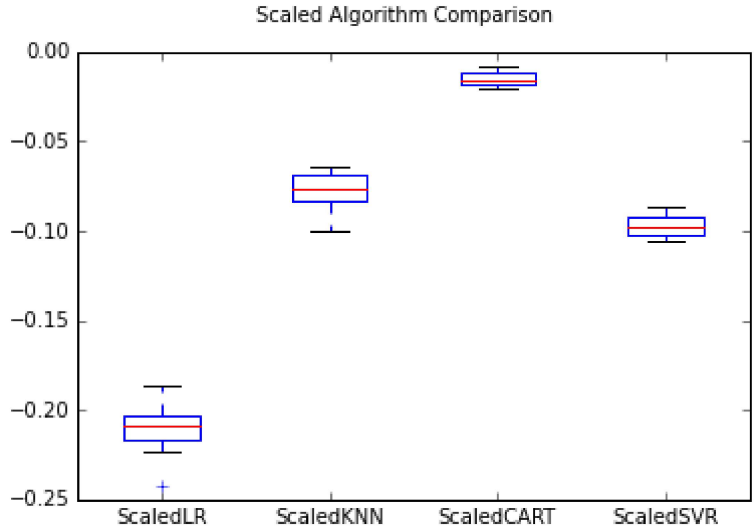
In [61]:
```
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```



In [62]:
```
# Standardize the dataset
pipelines = []
pipelines.append(('ScaledLR', Pipeline([('Scaler', StandardScaler()),('LR', Li
nearRegression())])))
#pipelines.append(('ScaledLASSO', Pipeline([('Scaler', StandardScaler()),('LAS
SO', Lasso())])))
#pipelines.append(('ScaledEN', Pipeline([('Scaler', StandardScaler()),('EN', E
lasticNet())])))
pipelines.append(('ScaledKNN', Pipeline([('Scaler', StandardScaler()),('KNN',
KNeighborsRegressor())])))
pipelines.append(('ScaledCART', Pipeline([('Scaler', StandardScaler()),
('CART', DecisionTreeRegressor())])))
pipelines.append(('ScaledSVR', Pipeline([('Scaler', StandardScaler()),('SVR',
SVR())])))
results = []
names = []
for name, model in pipelines:
        kfold = cross_validation.KFold(n=num_instances, n_folds=num_folds, ran
dom_state=seed)
        cv_results = cross_validation.cross_val_score(model, X_train, Y_train,
 cv=kfold, scoring=scoring)
        results.append(cv_results)
        names.append(name)
        msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
        print(msg)
```

```
ScaledLR: -0.209986 (0.015409)
ScaledKNN: -0.077684 (0.010380)
ScaledCART: -0.015019 (0.004585)
ScaledSVR: -0.097148 (0.006352)
```

In [63]:
```
# Compare Algorithms
fig = plt.figure()
fig.suptitle('Scaled Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```



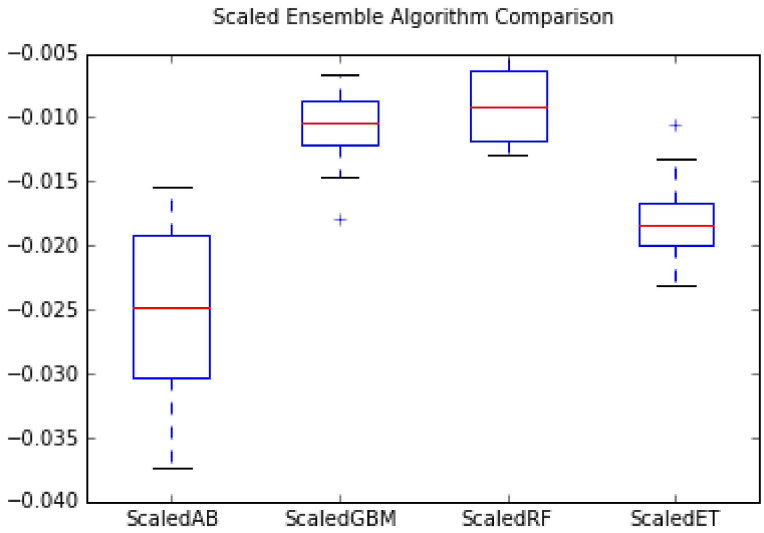
In [64]:
```
# KNN Algorithm tuning
scaler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
k_values = numpy.array([1,3,5,7,9,11,13,15,17,19,21])
param_grid = dict(n_neighbors=k_values)
model = KNeighborsRegressor()
kfold = cross_validation.KFold(n=num_instances, n_folds=num_folds, random_stat
e=seed)
grid = GridSearchCV(estimator=model, param_grid=param_grid, scoring=scoring, c
v=kfold)
grid_result = grid.fit(rescaledX, Y_train)

print("Best: %f using %s" % (grid_result.best_score_,
grid_result.best_params_))
for params, mean_score, scores in grid_result.grid_scores_:
    print("%f (%f) with: %r" % (scores.mean(), scores.std(), params))
```

```
Best: -0.074094 using {'n_neighbors': 3}
-0.091014 (0.017818) with: {'n_neighbors': 1}
-0.074087 (0.009995) with: {'n_neighbors': 3}
-0.077631 (0.010135) with: {'n_neighbors': 5}
-0.079836 (0.010411) with: {'n_neighbors': 7}
-0.083545 (0.010326) with: {'n_neighbors': 9}
-0.086690 (0.010544) with: {'n_neighbors': 11}
-0.089882 (0.010609) with: {'n_neighbors': 13}
-0.091268 (0.010435) with: {'n_neighbors': 15}
-0.093406 (0.010704) with: {'n_neighbors': 17}
-0.094649 (0.009777) with: {'n_neighbors': 19}
-0.096203 (0.009614) with: {'n_neighbors': 21}
```

In [65]:
```
# ensembles
ensembles = []
ensembles.append(('ScaledAB', Pipeline([('Scaler', StandardScaler()),('AB', Ad
aBoostRegressor())])))
ensembles.append(('ScaledGBM', Pipeline([('Scaler', StandardScaler()),('GBM',
GradientBoostingRegressor())])))
ensembles.append(('ScaledRF', Pipeline([('Scaler', StandardScaler()),('RF', Ra
ndomForestRegressor())])))
ensembles.append(('ScaledET', Pipeline([('Scaler', StandardScaler()),('ET', Ex
traTreesRegressor())])))
results = []
names = []
for name, model in ensembles:
        kfold = cross_validation.KFold(n=num_instances, n_folds=num_folds, ran
dom_state=seed)
        cv_results = cross_validation.cross_val_score(model, X_train, Y_train,
 cv=kfold, scoring=scoring)
        results.append(cv_results)
        names.append(name)
        msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
        print(msg)
```

```
ScaledAB: -0.025291 (0.007086)
ScaledGBM: -0.010903 (0.003256)
ScaledRF: -0.009013 (0.002919)
ScaledET: -0.017926 (0.003684)
```

In [66]:
```
# Compare Algorithms
fig = plt.figure()
fig.suptitle('Scaled Ensemble Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```



In [67]:
```
# Tune scaled GBM
scaler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
param_grid = dict(n_estimators=numpy.array([50,100,150,200,250,300,350,400]))
model = GradientBoostingRegressor(random_state=seed)
kfold = cross_validation.KFold(n=num_instances, n_folds=num_folds, random_stat
e=seed)
grid = GridSearchCV(estimator=model, param_grid=param_grid, scoring=scoring, c
v=kfold)
grid_result = grid.fit(rescaledX, Y_train)

print("Best: %f using %s" % (grid_result.best_score_,
grid_result.best_params_))
for params, mean_score, scores in grid_result.grid_scores_:
    print("%f (%f) with: %r" % (scores.mean(), scores.std(), params))
```

```
Best: -0.009543 using {'n_estimators': 200}
-0.014968 (0.003155) with: {'n_estimators': 50}
-0.010901 (0.003266) with: {'n_estimators': 100}
-0.009798 (0.003227) with: {'n_estimators': 150}
-0.009543 (0.003095) with: {'n_estimators': 200}
-0.009619 (0.003129) with: {'n_estimators': 250}
-0.009640 (0.003175) with: {'n_estimators': 300}
-0.009639 (0.003213) with: {'n_estimators': 350}
-0.009809 (0.003158) with: {'n_estimators': 400}
```

In [68]:
```
# Make predictions on validation dataset

# prepare the model
scaler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
model = GradientBoostingRegressor(random_state=seed, n_estimators=400)
model.fit(rescaledX, Y_train)

# transform the validation dataset
rescaledValidationX = scaler.transform(X_validation)
predictions = model.predict(rescaledValidationX)
print(mean_squared_error(Y_validation, predictions))
```

```
0.00784278107867
```

In [ ]:
```

```

In [69]:
```
# CART Classification
import pandas
from sklearn import cross_validation
from sklearn.tree import DecisionTreeClassifier
```

In [70]:
```
num_folds = 10
num_instances = len(X)
seed = 7
kfold = cross_validation.KFold(n=num_instances, n_folds=num_folds, random_stat
e=seed)
model = DecisionTreeClassifier()
results = cross_validation.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
0.75228934341
```

1.4.7 Deep Learning Neural Network for Classification

In [71]:
```
# 1. define the network
model = Sequential()
model.add(Dense(48, input_dim=23, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

In [72]:
```
# 2. compile the network
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accurac
y'])
```

In [73]:
```
# 3. fit the network
history = model.fit(trainX, trainY, nb_epoch=100, batch_size=10)
```

```
Epoch 1/100
3193/3193 [==============================] - 24s - loss: 0.5520 - acc: 0.7444

Epoch 2/100
3193/3193 [==============================] - 24s - loss: 0.4849 - acc: 0.7501

Epoch 3/100
3193/3193 [==============================] - 24s - loss: 0.4478 - acc: 0.7629

Epoch 4/100
3193/3193 [==============================] - 24s - loss: 0.4268 - acc: 0.7914

Epoch 5/100
3193/3193 [==============================] - 24s - loss: 0.4118 - acc: 0.8002

Epoch 6/100
3193/3193 [==============================] - 25s - loss: 0.4001 - acc: 0.8080

Epoch 7/100
3193/3193 [==============================] - 24s - loss: 0.3909 - acc: 0.8209

Epoch 8/100
3193/3193 [==============================] - 25s - loss: 0.3827 - acc: 0.8249

Epoch 9/100
3193/3193 [==============================] - 25s - loss: 0.3754 - acc: 0.8368

Epoch 10/100
3193/3193 [==============================] - 25s - loss: 0.3687 - acc: 0.8425

Epoch 11/100
3193/3193 [==============================] - 25s - loss: 0.3618 - acc: 0.8506

Epoch 12/100
3193/3193 [==============================] - 25s - loss: 0.3537 - acc: 0.8569

Epoch 13/100
3193/3193 [==============================] - 25s - loss: 0.3474 - acc: 0.8675

Epoch 14/100
3193/3193 [==============================] - 25s - loss: 0.3405 - acc: 0.8719

Epoch 15/100
3193/3193 [==============================] - 25s - loss: 0.3346 - acc: 0.8769

Epoch 16/100
3193/3193 [==============================] - 25s - loss: 0.3270 - acc: 0.8816

Epoch 17/100
3193/3193 [==============================] - 25s - loss: 0.3211 - acc: 0.8841

Epoch 18/100
3193/3193 [==============================] - 25s - loss: 0.3153 - acc: 0.8854

Epoch 19/100
3193/3193 [==============================] - 25s - loss: 0.3085 - acc: 0.8869
```

```
Epoch 20/100
3193/3193 [==============================] - 25s - loss: 0.3016 - acc: 0.8913

Epoch 21/100
3193/3193 [==============================] - 25s - loss: 0.2964 - acc: 0.8907

Epoch 22/100
3193/3193 [==============================] - 25s - loss: 0.2910 - acc: 0.8894

Epoch 23/100
3193/3193 [==============================] - 25s - loss: 0.2853 - acc: 0.8960

Epoch 24/100
3193/3193 [==============================] - 25s - loss: 0.2804 - acc: 0.8948

Epoch 25/100
3193/3193 [==============================] - 25s - loss: 0.2745 - acc: 0.8957

Epoch 26/100
3193/3193 [==============================] - 25s - loss: 0.2693 - acc: 0.9001

Epoch 27/100
3193/3193 [==============================] - 24s - loss: 0.2633 - acc: 0.9032

Epoch 28/100
3193/3193 [==============================] - 25s - loss: 0.2583 - acc: 0.9035

Epoch 29/100
3193/3193 [==============================] - 25s - loss: 0.2546 - acc: 0.9032

Epoch 30/100
3193/3193 [==============================] - 25s - loss: 0.2477 - acc: 0.9104

Epoch 31/100
3193/3193 [==============================] - 25s - loss: 0.2442 - acc: 0.9136

Epoch 32/100
3193/3193 [==============================] - 25s - loss: 0.2376 - acc: 0.9148

Epoch 33/100
3193/3193 [==============================] - 25s - loss: 0.2339 - acc: 0.9167

Epoch 34/100
3193/3193 [==============================] - 25s - loss: 0.2271 - acc: 0.9164

Epoch 35/100
3193/3193 [==============================] - 25s - loss: 0.2232 - acc: 0.9189

Epoch 36/100
3193/3193 [==============================] - 25s - loss: 0.2185 - acc: 0.9201

Epoch 37/100
3193/3193 [==============================] - 25s - loss: 0.2143 - acc: 0.9233

Epoch 38/100
3193/3193 [==============================] - 25s - loss: 0.2116 - acc: 0.9211
```

```
Epoch 39/100
3193/3193 [==============================] - 25s - loss: 0.2062 - acc: 0.9251

Epoch 40/100
3193/3193 [==============================] - 25s - loss: 0.2020 - acc: 0.9277

Epoch 41/100
3193/3193 [==============================] - 25s - loss: 0.1981 - acc: 0.9264

Epoch 42/100
3193/3193 [==============================] - 25s - loss: 0.1950 - acc: 0.9302

Epoch 43/100
3193/3193 [==============================] - 25s - loss: 0.1902 - acc: 0.9336

Epoch 44/100
3193/3193 [==============================] - 25s - loss: 0.1874 - acc: 0.9327

Epoch 45/100
3193/3193 [==============================] - 25s - loss: 0.1831 - acc: 0.9364

Epoch 46/100
3193/3193 [==============================] - 25s - loss: 0.1805 - acc: 0.9370

Epoch 47/100
3193/3193 [==============================] - 25s - loss: 0.1762 - acc: 0.9392

Epoch 48/100
3193/3193 [==============================] - 24s - loss: 0.1746 - acc: 0.9392

Epoch 49/100
3193/3193 [==============================] - 25s - loss: 0.1714 - acc: 0.9414

Epoch 50/100
3193/3193 [==============================] - 25s - loss: 0.1676 - acc: 0.9399

Epoch 51/100
3193/3193 [==============================] - 24s - loss: 0.1648 - acc: 0.9424

Epoch 52/100
3193/3193 [==============================] - 24s - loss: 0.1619 - acc: 0.9458

Epoch 53/100
3193/3193 [==============================] - 25s - loss: 0.1599 - acc: 0.9455

Epoch 54/100
3193/3193 [==============================] - 24s - loss: 0.1567 - acc: 0.9461

Epoch 55/100
3193/3193 [==============================] - 25s - loss: 0.1556 - acc: 0.9449

Epoch 56/100
3193/3193 [==============================] - 25s - loss: 0.1529 - acc: 0.9486

Epoch 57/100
3193/3193 [==============================] - 24s - loss: 0.1518 - acc: 0.9474
```

```
Epoch 58/100
3193/3193 [==============================] - 25s - loss: 0.1479 - acc: 0.9486

Epoch 59/100
3193/3193 [==============================] - 25s - loss: 0.1466 - acc: 0.9496

Epoch 60/100
3193/3193 [==============================] - 25s - loss: 0.1441 - acc: 0.9499

Epoch 61/100
3193/3193 [==============================] - 25s - loss: 0.1410 - acc: 0.9496

Epoch 62/100
3193/3193 [==============================] - 23s - loss: 0.1389 - acc: 0.9530

Epoch 63/100
3193/3193 [==============================] - 23s - loss: 0.1364 - acc: 0.9549

Epoch 64/100
3193/3193 [==============================] - 25s - loss: 0.1360 - acc: 0.9568

Epoch 65/100
3193/3193 [==============================] - 25s - loss: 0.1351 - acc: 0.9524

Epoch 66/100
3193/3193 [==============================] - 25s - loss: 0.1314 - acc: 0.9562

Epoch 67/100
3193/3193 [==============================] - 25s - loss: 0.1320 - acc: 0.9552

Epoch 68/100
3193/3193 [==============================] - 25s - loss: 0.1295 - acc: 0.9549

Epoch 69/100
3193/3193 [==============================] - 25s - loss: 0.1282 - acc: 0.9565

Epoch 70/100
3193/3193 [==============================] - 25s - loss: 0.1268 - acc: 0.9587

Epoch 71/100
3193/3193 [==============================] - 24s - loss: 0.1237 - acc: 0.9593

Epoch 72/100
3193/3193 [==============================] - 24s - loss: 0.1240 - acc: 0.9580

Epoch 73/100
3193/3193 [==============================] - 25s - loss: 0.1224 - acc: 0.9596

Epoch 74/100
3193/3193 [==============================] - 25s - loss: 0.1210 - acc: 0.9577

Epoch 75/100
3193/3193 [==============================] - 23s - loss: 0.1184 - acc: 0.9583

Epoch 76/100
3193/3193 [==============================] - 24s - loss: 0.1179 - acc: 0.9580
```

```
Epoch 77/100
3193/3193 [==============================] - 24s - loss: 0.1163 - acc: 0.9599

Epoch 78/100
3193/3193 [==============================] - 25s - loss: 0.1165 - acc: 0.9596

Epoch 79/100
3193/3193 [==============================] - 24s - loss: 0.1141 - acc: 0.9609

Epoch 80/100
3193/3193 [==============================] - 24s - loss: 0.1147 - acc: 0.9599

Epoch 81/100
3193/3193 [==============================] - 25s - loss: 0.1128 - acc: 0.9599

Epoch 82/100
3193/3193 [==============================] - 24s - loss: 0.1122 - acc: 0.9612

Epoch 83/100
3193/3193 [==============================] - 26s - loss: 0.1106 - acc: 0.9624

Epoch 84/100
3193/3193 [==============================] - 25s - loss: 0.1088 - acc: 0.9621

Epoch 85/100
3193/3193 [==============================] - 26s - loss: 0.1073 - acc: 0.9587

Epoch 86/100
3193/3193 [==============================] - 25s - loss: 0.1076 - acc: 0.9630

Epoch 87/100
3193/3193 [==============================] - 24s - loss: 0.1064 - acc: 0.9621

Epoch 88/100
3193/3193 [==============================] - 26s - loss: 0.1054 - acc: 0.9627

Epoch 89/100
3193/3193 [==============================] - 26s - loss: 0.1044 - acc: 0.9643

Epoch 90/100
3193/3193 [==============================] - 25s - loss: 0.1060 - acc: 0.9612

Epoch 91/100
3193/3193 [==============================] - 26s - loss: 0.1035 - acc: 0.9693

Epoch 92/100
3193/3193 [==============================] - 25s - loss: 0.1023 - acc: 0.9643

Epoch 93/100
3193/3193 [==============================] - 25s - loss: 0.1018 - acc: 0.9643

Epoch 94/100
3193/3193 [==============================] - 28s - loss: 0.1014 - acc: 0.9649

Epoch 95/100
3193/3193 [==============================] - 26s - loss: 0.1012 - acc: 0.9640
```

```
Epoch 96/100
3193/3193 [==============================] - 24s - loss: 0.0994 - acc: 0.9646

Epoch 97/100
3193/3193 [==============================] - 24s - loss: 0.0995 - acc: 0.9618

Epoch 98/100
3193/3193 [==============================] - 23s - loss: 0.0996 - acc: 0.9637

Epoch 99/100
3193/3193 [==============================] - 23s - loss: 0.0977 - acc: 0.9643

Epoch 100/100
3193/3193 [==============================] - 23s - loss: 0.0964 - acc: 0.9659
```

In [74]:
```
# 4. evaluate the network
loss, accuracy = model.evaluate(trainX, trainY)
print("\nLoss: %.2f, Accuracy: %.2f%%" % (loss, accuracy*100))
```

```
3193/3193 [==============================] - 2s

Loss: 0.09, Accuracy: 96.81%
```

In [75]:
```
# 5. make predictions
probabilities = model.predict(testX)
predictions = [float(round(x)) for x in probabilities]
accuracy = numpy.mean(predictions == testY)
print("Prediction Accuracy: %.2f%%" % (accuracy*100))
```

```
Prediction Accuracy: 89.45%
```

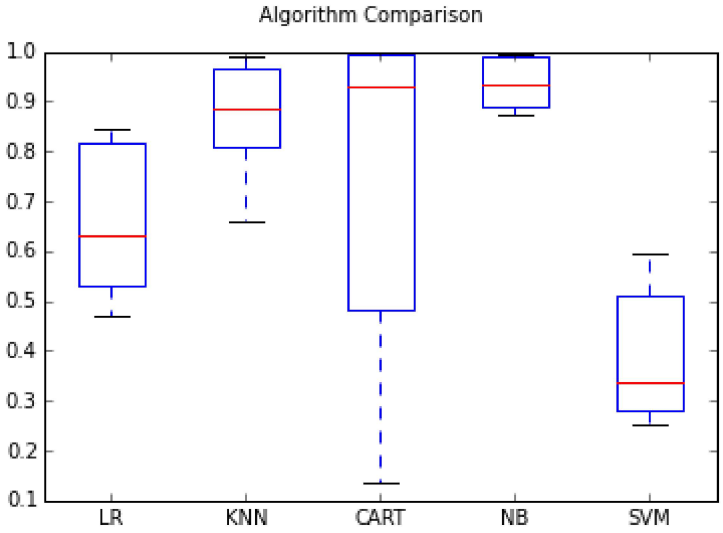
In [76]:
```
db = c.mydb
```

In [79]:
```
# Gaussian Naive Bayes Classification
import pandas
from sklearn import cross_validation
from sklearn.naive_bayes import GaussianNB

num_folds = 10
num_instances = len(X)
seed = 7
kfold = cross_validation.KFold(n=num_instances, n_folds=num_folds, random_stat
e=seed)
model = GaussianNB()
results = cross_validation.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
0.936680143756
```

In [84]:
```
# Compare Algorithms
import pandas
import matplotlib.pyplot as plt
from sklearn import cross_validation
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
#from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
num_folds = 10
num_instances = len(X)
seed = 7
# prepare models
models = []
models.append(('LR', LogisticRegression()))
#models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
# evaluate each model in turn
results = []
names = []
scoring = 'accuracy'
for name, model in models:
        kfold = cross_validation.KFold(n=num_instances, n_folds=num_folds, ran
dom_state=seed)
        cv_results = cross_validation.cross_val_score(model, X, Y, cv=kfold, s
coring=scoring)
        results.append(cv_results)
        names.append(name)
        msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
        print(msg)
# boxplot algorithm comparison
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```

```
LR: 0.659579 (0.146680)
KNN: 0.865388 (0.112075)
CART: 0.751870 (0.328947)
NB: 0.936680 (0.050174)
SVM: 0.393721 (0.128942)
```



In [98]:
```
# Grid Search for Algorithm Tuning
import pandas
import numpy
from sklearn import cross_validation
from sklearn.linear_model import Ridge
from sklearn.grid_search import GridSearchCV
alphas = numpy.array([1,0.1,0.01,0.001,0.0001,0])
param_grid = dict(alpha=alphas)
model = Ridge()
grid = GridSearchCV(estimator=model, param_grid=param_grid)
grid.fit(X, Y)
print(grid.best_score_)
print(grid.best_estimator_.alpha)
```

```
-0.00963724047744
1.0
```

In [99]:
```
# Randomized for Algorithm Tuning
import pandas
import numpy
from scipy.stats import uniform
from sklearn.linear_model import Ridge
from sklearn.grid_search import RandomizedSearchCV
param_grid = {'alpha': uniform()}
seed = 7
model = Ridge()
iterations = 100
rsearch = RandomizedSearchCV(estimator=model, param_distributions=param_grid,
n_iter=iterations, random_state=seed)
rsearch.fit(X, Y)
print(rsearch.best_score_)
print(rsearch.best_estimator_.alpha)
```

```
-0.00975660330876
0.976824910257
```

In [103]:
```
# Extra Trees Classification
import pandas
from sklearn import cross_validation
from sklearn.ensemble import ExtraTreesClassifier
num_folds = 10
num_instances = len(X)
seed = 7
num_trees = 100
max_features = 7
kfold = cross_validation.KFold(n=num_instances, n_folds=num_folds, random_stat
e=seed)
model = ExtraTreesClassifier(n_estimators=num_trees,
max_features=max_features)
results = cross_validation.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
0.935167714885
```

1.5 Helper Files and Routines for creating the transaction data

In [111]:
```
#-83.089457565151, 42.3818784429705
cursor = db.saleparcelyear.aggregate(
    [
     { "$geoNear": {
         "near": [ float(-83.089457565151), float(42.3818784429705) ],
         "maxDistance": 0.00001261617096,
         "distanceField": "distance",
         "includeLocs": "latlng",
         "uniqueDocs": True,
         "spherical":True,
         "query": {
             'Ward' : 4
         },
         "limit":5
     }},
     { "$sort": { "distance": 1 } }
])
for document in cursor:
    print(document)
    #pp_json(document)
```

{u'TotAcres': 0.153, u'PropAddr': u'831 EDISON', u'ParcelNo': 4002599.0, u'Sa
lePrice': u'$140000.00', u'Latitude': 42.3818784429705, u'IncomePerHouseho
d': 20708.0, u'PropNo': 831, u'PropDir': u'', u'SaleDate': {u'year': 2015.0,
 u'day': 24.0, u'month': 11.0}, u'Ward': 4, u'PropZip': 48202, u'Population':
 16603.0, u'Frontage': 50, u'ResYrBuilt': 1920, u'Longitude': -83.08945756515
1, u'Depth': 133.5, u'CIBYrBuilt': 0, u'distance': 0.0, u'_id': 4002599.0,
 u'geometry': {u'type': u'Point', u'coordinates': [-83.089457565151, 42.38187
84429705]}, u'latlng': {u'type': u'Point', u'coordinates': [-83.089457565151,
 42.3818784429705]}, u'TotSqFt': 6665, u'PropStr': u'EDISON', u'AverageHouseV
alue': 67700.0}
{u'TotAcres': 0.153, u'PropAddr': u'821 EDISON', u'ParcelNo': 4002598.0, u'Sa
lePrice': u'$49900.00', u'Latitude': 42.3819394423952, u'IncomePerHousehold':
 20708.0, u'PropNo': 821, u'PropDir': u'', u'SaleDate': {u'year': 2014.0, u'd
ay': 7.0, u'month': 1.0}, u'Ward': 4, u'PropZip': 48202, u'Population': 1660
3.0, u'Frontage': 50, u'ResYrBuilt': 0, u'Longitude': -83.0892920651423, u'De
pth': 133.5, u'CIBYrBuilt': 0, u'distance': 2.3845243366963763e-06, u'_id': 4
002598.0, u'geometry': {u'type': u'Point', u'coordinates': [-83.089292065142
3, 42.3819394423952]}, u'latlng': {u'type': u'Point', u'coordinates': [-83.08
92920651423, 42.3819394423952]}, u'TotSqFt': 6665, u'PropStr': u'EDISON', u'A
verageHouseValue': 67700.0}
{u'TotAcres': 0.153, u'PropAddr': u'839 EDISON', u'ParcelNo': 4002600.0, u'Sa
lePrice': u'$85524.00', u'Latitude': 42.38181694314, u'IncomePerHousehold': 2
0708.0, u'PropNo': 839, u'PropDir': u'', u'SaleDate': {u'year': 2009.0, u'da
y': 29.0, u'month': 8.0}, u'Ward': 4, u'PropZip': 48202, u'Population': 1660
3.0, u'Frontage': 50, u'ResYrBuilt': 1913, u'Longitude': -83.0896230648237,
 u'Depth': 133.5, u'CIBYrBuilt': 0, u'distance': 2.388434540811672e-06, u'_i
d': 4002600.0, u'geometry': {u'type': u'Point', u'coordinates': [-83.08962306
48237, 42.38181694314]}, u'latlng': {u'type': u'Point', u'coordinates': [-83.
0896230648237, 42.38181694314]}, u'TotSqFt': 6665, u'PropStr': u'EDISON', u'A
verageHouseValue': 67700.0}
{u'TotAcres': 0.123, u'PropAddr': u'849 EDISON', u'ParcelNo': 4002601.0, u'Sa
lePrice': u'$17000.00', u'Latitude': 42.3817614436879, u'IncomePerHousehold':
 20708.0, u'PropNo': 849, u'PropDir': u'', u'SaleDate': {u'year': 1992.0, u'd
ay': 1.0, u'month': 6.0}, u'Ward': 4, u'PropZip': 48202, u'Population': 1660
3.0, u'Frontage': 40, u'ResYrBuilt': 1910, u'Longitude': -83.0897720635786,
 u'Depth': 133.5, u'CIBYrBuilt': 0, u'distance': 4.539766030125526e-06, u'_i
d': 4002601.0, u'geometry': {u'type': u'Point', u'coordinates': [-83.08977206
35786, 42.3817614436879]}, u'latlng': {u'type': u'Point', u'coordinates': [-8
3.0897720635786, 42.3817614436879]}, u'TotSqFt': 5358, u'PropStr': u'EDISON',
 u'AverageHouseValue': 67700.0}
{u'TotAcres': 0.459, u'PropAddr': u'811 EDISON', u'ParcelNo': 4002597.0, u'Sa
lePrice': u'$18500.00', u'Latitude': 42.3820014419213, u'IncomePerHousehold':
 20708.0, u'PropNo': 811, u'PropDir': u'', u'SaleDate': {u'year': 2014.0, u'd
ay': 29.0, u'month': 9.0}, u'Ward': 4, u'PropZip': 48202, u'Population': 1660
3.0, u'Frontage': 50, u'ResYrBuilt': 1909, u'Longitude': -83.0891265659261,
 u'Depth': 133.5, u'CIBYrBuilt': 0, u'distance': 4.776856522277379e-06, u'_i
d': 4002597.0, u'geometry': {u'type': u'Point', u'coordinates': [-83.08912656
59261, 42.3820014419213]}, u'latlng': {u'type': u'Point', u'coordinates': [-8
3.0891265659261, 42.3820014419213]}, u'TotSqFt': 6665, u'PropStr': u'EDISON',
 u'AverageHouseValue': 67700.0}

In [125]:
```
cursor = db.saleparcelyear.aggregate(
    [
     { "$geoNear": {
         "near": [ float(-83.09176672199999), float(42.33574926800003) ],
#          "maxDistance": 0.00001261617096,
         "maxDistance": 10000,
         "distanceField": "distance",
         "includeLocs": "latlng",
         "uniqueDocs": True,
         "spherical":True,
         "query": {
             'Ward' : {'$gte': 0}
         }
         #,"limit":100
     }}
    ,{ '$lookup': {
            'from': "newTestBuildings",
            'localField': "PropAddr",
            'foreignField': "streetAddres",
            'as': "address"} }
        ,{ '$unwind': "$address"}
#         ,{ '$unwind': "$userAddressList"}
        ,{ "$group": {
            "_id": "$_id",
             "PropAddr": { "$first": "$PropAddr" },
            "SalePrice" : { "$first": "$SalePrice" },
             "distance": { "$first": "$distance" },
             "avgPrice": { "$avg": "$SalePrice"},
            "streetAddres": { "$first": "$address.streetAddres" },
            "source": { "$first": "$address.source" },
             "sumof":{ "$sum": "$SalePrice" }
        }}
        ,{ "$project" : {
             "_id": 1,
            "PropAddr": 1,
            "distance": 1,
            "buiding": 1,
            "SalePrice" : 1,
           #  "meanBuildings": { "$divide": [ "$sumof", "$buiding" ]}
             "avgPrice": { "$trunc": "$avgPrice"},
            "source" : 1,
             "streetAddres": 1
       }}
       ,{ "$limit": 25 }
       ,{ "$sort": { "distance": 1 } }
])
for document in cursor:
    print(document)
    #pp_json(document)
```

{u'distance': 0.0004649662696999953, u'PropAddr': u'6101 TRUMBULL', u'streetA
ddres': u'6101 TRUMBULL', u'avgPrice': None, u'source': u'vio', u'SalePrice':
 u'$3300.00', u'_id': 8005766.0}
{u'distance': 0.00047026314178269133, u'PropAddr': u'6131 TRUMBULL', u'street
Addres': u'6131 TRUMBULL', u'avgPrice': None, u'source': u'vio', u'SalePric
e': u'$600.00', u'_id': 8005762.0}
{u'distance': 0.0004743850520479402, u'PropAddr': u'6120 TRUMBULL', u'streetA
ddres': u'6120 TRUMBULL', u'avgPrice': None, u'source': u'vio', u'SalePrice':
 u'$32000.00', u'_id': 6006059.0}
{u'distance': 0.0004764263532487425, u'PropAddr': u'6175 TRUMBULL', u'streetA
ddres': u'6175 TRUMBULL', u'avgPrice': None, u'source': u'vio', u'SalePrice':
 u'$103000.00', u'_id': u'08005756-8'}
{u'distance': 0.0004765364609584159, u'PropAddr': u'6130 TRUMBULL', u'streetA
ddres': u'6130 TRUMBULL', u'avgPrice': None, u'source': u'demo', u'SalePric
e': u'$0.00', u'_id': 6006061.0}
{u'distance': 0.0004845135176249408, u'PropAddr': u'5841 FOURTH', u'streetAdd
res': u'5841 FOURTH', u'avgPrice': None, u'source': u'vio', u'SalePrice':
 u'$53000.00', u'_id': 4003903.0}
{u'distance': 0.0004846614318455159, u'PropAddr': u'5810 FOURTH', u'streetAdd
res': u'5810 FOURTH', u'avgPrice': None, u'source': u'vio', u'SalePrice':
 u'$6000.00', u'_id': 4003882.0}
{u'distance': 0.0004856185208454861 5, u'PropAddr': u'5816 FOURTH', u'streetAd
dres': u'5816 FOURTH', u'avgPrice': None, u'source': u'vio', u'SalePrice':
 u'$12000.00', u'_id': 4003883.0}
{u'distance': 0.0004863817760007965 3, u'PropAddr': u'5855 FOURTH', u'streetAd
dres': u'5855 FOURTH', u'avgPrice': None, u'source': u'vio', u'SalePrice':
 u'$80000.00', u'_id': 4003902.0}
{u'distance': 0.0004882944341065977, u'PropAddr': u'5865 FOURTH', u'streetAdd
res': u'5865 FOURTH', u'avgPrice': None, u'source': u'vio', u'SalePrice':
 u'$0.00', u'_id': 4003900.0}
{u'distance': 0.0004886436542938626, u'PropAddr': u'5834 FOURTH', u'streetAdd
res': u'5834 FOURTH', u'avgPrice': None, u'source': u'vio', u'SalePrice':
 u'$56000.00', u'_id': 4003886.0}
{u'distance': 0.0004898908863318879 3, u'PropAddr': u'5869 FOURTH', u'streetAddr
es': u'5869 FOURTH', u'avgPrice': None, u'source': u'vio', u'SalePrice': u'$3
3000.00', u'_id': 4003899.0}
{u'distance': 0.0004899136290653265, u'PropAddr': u'5875 FOURTH', u'streetAdd
res': u'5875 FOURTH', u'avgPrice': None, u'source': u'vio', u'SalePrice':
 u'$0.00', u'_id': 4003898.0}
{u'distance': 0.0004899261718436611, u'PropAddr': u'5842 FOURTH', u'streetAdd
res': u'5842 FOURTH', u'avgPrice': None, u'source': u'vio', u'SalePrice':
 u'$0.00', u'_id': 4003887.0}
{u'distance': 0.0004912206829564434, u'PropAddr': u'5850 FOURTH', u'streetAdd
res': u'5850 FOURTH', u'avgPrice': None, u'source': u'vio', u'SalePrice':
 u'$190000.00', u'_id': 4003888.0}
{u'distance': 0.0004927452160001033, u'PropAddr': u'6420 STERLING', u'streetA
ddres': u'6420 STERLING', u'avgPrice': None, u'source': u'vio', u'SalePrice':
 u'$45000.00', u'_id': 6006109.0}
{u'distance': 0.0004931369641254242, u'PropAddr': u'5895 FOURTH', u'streetAdd
res': u'5895 FOURTH', u'avgPrice': None, u'source': u'vio', u'SalePrice':
 u'$39000.00', u'_id': 4003894.0}
{u'distance': 0.0004932391455938232, u'PropAddr': u'5835 THIRD', u'streetAddr
es': u'5835 THIRD', u'avgPrice': None, u'source': u'vio', u'SalePrice': u'$13
500.00', u'_id': 4003448.0}
{u'distance': 0.0004943987580253796, u'PropAddr': u'5847 THIRD', u'streetAddr
es': u'5847 THIRD', u'avgPrice': None, u'source': u'vio', u'SalePrice': u'$55
000.00', u'_id': 4003447.0}

```
{u'distance': 0.0004955516022968973, u'PropAddr': u'5853 THIRD', u'streetAddr
es': u'5853 THIRD', u'avgPrice': None, u'source': u'vio', u'SalePrice': u'$20
0000.00', u'_id': 4003446.0}
{u'distance': 0.0004960672645115203, u'PropAddr': u'6331 TRUMBULL', u'streetA
ddres': u'6331 TRUMBULL', u'avgPrice': None, u'source': u'demo', u'SalePric
e': u'$107500.00', u'_id': 6006108.0}
{u'distance': 0.0004961870595683783, u'PropAddr': u'6430 STERLING', u'streetA
ddres': u'6430 STERLING', u'avgPrice': None, u'source': u'demo', u'SalePric
e': u'$22500.00', u'_id': 6006110.0}
{u'distance': 0.0004969573946380019, u'PropAddr': u'6435 STERLING', u'streetAd
dres': u'6435 STERLING', u'avgPrice': None, u'source': u'demo', u'SalePrice':
 u'$50000.00', u'_id': 8005977.0}
{u'distance': 0.0004975534599411897, u'PropAddr': u'6337 TRUMBULL', u'streetA
ddres': u'6337 TRUMBULL', u'avgPrice': None, u'source': u'demo', u'SalePric
e': u'$100000.00', u'_id': 6006107.0}
```

In [ ]:
```
#db.parcelOwnership.ensureIndex( { 'Location' : "2d" } )
db.newGeoPt1.create_index([
    ('geometry', '2dsphere')])
```

In [126]:
```
# get geoNearnewGeoPt1
db = c.mydb
result = db.command(
    'geoNear', 'newGeoPt1',
    near={
        'type': 'Point',
        'coordinates': [
            -83.121307421999, 42.4159074370000]},
    spherical=True,
    num=2)
print result
```

```
{u'stats': {u'avgDistance': 7.978685308268732e-08, u'maxDistance': 7.97868530
8268732e-08, u'objectsLoaded': 13, u'nscanned': 24, u'time': 87}, u'ok': 1.0,
 u'waitedMS': 0L, u'results': [{u'obj': {u'TicketNumber': u'15011175DAH', u'g
eometry': {u'type': u'Point', u'coordinates': [-83.12130742199997, 42.4159074
37000044]}, u'field1': 307543, u'longitude': -83.12130742199997, u'ViolationC
ode': u'9-1-104', u'latitude': 42.415907437000044, u'_id': ObjectId('57897d9a
65d68b87b6a8f672')}, u'dis': 7.978685308268732e-08}, {u'obj': {u'TicketNumbe
r': u'15011487DAH', u'geometry': {u'type': u'Point', u'coordinates': [-83.121
30742199997, 42.415907437000044]}, u'field1': 307781, u'longitude': -83.12130
742199997, u'ViolationCode': u'9-1-104', u'latitude': 42.415907437000044, u'_
id': ObjectId('57897d9a65d68b87b6a8f760')}, u'dis': 7.978685308268732e-08}]}
```

In [ ]:
```
cursor = db.parcelOwnership.aggregate([
#    { '$match': {'SaleDate.year' : 2015, 'PropZip': 48202} }
    { '$match': {'PropZip': 48202} }
    ,{ '$unwind': "$SaleDate"}
    ,{ '$lookup': {
             'from': "zipHoods",
             'localField': "PropZip",
             'foreignField': "zipcode",
             'as': "zipName"} }
    ,{ '$unwind': "$zipName"}
    ,{ '$project': {"Year": "$SaleDate.year",
                    "TotSqFt":1, "TotAcres":1,  "CIBYrBuilt" : 1, "ResYrBuilt"
 : 1,
                    "Frontage" : 1, "Depth" : 1,
                    "AverageHouseValue": "$zipName.AverageHouseValue",
                    "IncomePerHousehold": "$zipName.IncomePerHousehold",
                    "Population": "$zipName.Population",
                    #"zipName.AverageHouseValue":1, "zipName.Population":1, "z
ipName.IncomePerHousehold":1,
                    "SalePrice":1,
                    "_id":0} }
    #,{ "$limit": 500 }
    ,{ "$out": "saletempyear" }
])
for document in cursor:
    print(document)
    #pp_json(document)
```

In [127]:

```
cursor = db.ownertempyear.aggregate([
    { '$match': {'PropZip': 48202} }
    ,{ '$unwind': "$type"}
    ,{ '$project': {
                    "ParcelNo": "$ParcelNo",
                    "Ward" :    "$Ward",
                    "PropAddr" : "$PropAddr",
                    "PropNo" :   "$PropNo",
                    "PropStr" :  "$PropStr",
                    "PropDir" :  "$PropDir",
                    "PropZip" :  "$PropZip",
                    "TotSqFt":1, "TotAcres":1,  "CIBYrBuilt" : 1, "ResYrBuilt"
 : 1,
                    "Frontage" : 1, "Depth" : 1,
                    "AverageHouseValue": 1,
                    "IncomePerHousehold": 1,
                    "Population": 1,
                    "SalePrice":1,
                    "SaleDate":1,
                     "Latitude": 1,
                     "Longitude": 1,
                     "type": { "$literal": ["Latitude","Longitude"] },
                    "_id":0} },
    { "$unwind": "$type" },
    { "$group": {
        "_id": "$ParcelNo",
        "ParcelNo":  { "$first":  "$ParcelNo"},
        "Ward":      { "$first":  "$Ward"},
        "PropAddr" :  { "$first":  "$PropAddr"},
        "PropNo" :   { "$first":  "$PropNo"},
        "PropStr" :  { "$first":  "$PropStr"},
        "PropDir" :  { "$first":  "$PropDir"},
        "PropZip" :  { "$first":  "$PropZip"},
        "TotSqFt":   { "$first":  "$TotSqFt"},
        "TotAcres":  { "$first":  "$TotAcres"},
        "CIBYrBuilt" : { "$first":  "$CIBYrBuilt"},
        "ResYrBuilt" : { "$first":  "$ResYrBuilt"},
        "Frontage" : { "$first":  "$Frontage"},
        "Depth" : { "$first":  "$Depth"},
        "AverageHouseValue": { "$first":  "$AverageHouseValue"},
        "IncomePerHousehold": { "$first":  "$IncomePerHousehold"},
        "Population": { "$first":  "$Population"},
        "SalePrice":{ "$first":  "$SalePrice"},
        "SaleDate":{ "$first":  "$SaleDate"},
        "Latitude": { "$first":   "$Latitude" },
        "Longitude": { "$first": "$Longitude" },
        "coordinates": {
            "$push": {
                "$cond": [
                    { "$eq": [ "$type", "Latitude" ] },
                    "$Longitude",
                    "$Latitude"
                ]
            }
        }
    }},
    { "$project": {
```

```
        "ParcelNo": "$ParcelNo",
        "Ward" :    "$Ward",
        "PropAddr" : "$PropAddr",
        "PropNo" :   "$PropNo",
        "PropStr" :  "$PropStr",
        "PropDir" :  "$PropDir",
        "PropZip" :  "$PropZip",
        "TotSqFt":1, "TotAcres":1,  "CIBYrBuilt" : 1, "ResYrBuilt" : 1,
        "Frontage" : 1, "Depth" : 1,
        "AverageHouseValue": 1,
        "IncomePerHousehold": 1,
        "Population": 1,
        "SalePrice":1,
        "SaleDate":1,
         "Latitude": 1,
         "Longitude": 1,
        "geometry": {
            "type": { "$literal": "Point" },
            "coordinates": "$coordinates"
                    }
    }}

    #,{ "$limit": 500 }
    ,{ "$out": "saleparcelyear" }
])
for document in cursor:
    print(document)
    #pp_json(document)
```

In [ ]:
```
db.saleparcelyear.create_index([
    ('geometry', '2dsphere')])
```

In [128]:
```
#db = c.test
db = c.mydb
result = db.command(
    'geoNear', 'saleparcelyear',
    near={
        'type': 'Point',
        'coordinates': [
            -83.089457565151,
            42.3818784429705]},
    spherical='True',
    num=2)
print result
```

{u'stats': {u'avgDistance': 7.604367335941578, u'maxDistance': 15.208734671883157, u'objectsLoaded': 3, u'nscanned': 39, u'time': 0}, u'ok': 1.0, u'waitedMS': 0L, u'results': [{u'obj': {u'TotAcres': 0.153, u'PropAddr': u'831 EDISON', u'ParcelNo': 4002599.0, u'SalePrice': u'$140000.00', u'Latitude': 42.3818784429705, u'IncomePerHousehold': 20708.0, u'PropNo': 831, u'PropDir': u'', u'SaleDate': {u'year': 2015.0, u'day': 24.0, u'month': 11.0}, u'Ward': 4, u'PropZip': 48202, u'Population': 16603.0, u'Frontage': 50, u'ResYrBuilt': 1920, u'Longitude': -83.089457565151, u'Depth': 133.5, u'CIBYrBuilt': 0, u'_id': 4002599.0, u'geometry': {u'type': u'Point', u'coordinates': [-83.089457565151, 42.3818784429705]}, u'TotSqFt': 6665, u'PropStr': u'EDISON', u'AverageHouseValue': 67700.0}, u'dis': 0.0}, {u'obj': {u'TotAcres': 0.153, u'PropAddr': u'821 EDISON', u'ParcelNo': 4002598.0, u'SalePrice': u'$49900.00', u'Latitude': 42.3819394423952, u'IncomePerHousehold': 20708.0, u'PropNo': 821, u'PropDir': u'', u'SaleDate': {u'year': 2014.0, u'day': 7.0, u'month': 1.0}, u'Ward': 4, u'PropZip': 48202, u'Population': 16603.0, u'Frontage': 50, u'ResYrBuilt': 0, u'Longitude': -83.0892920651423, u'Depth': 133.5, u'CIBYrBuilt': 0, u'_id': 4002598.0, u'geometry': {u'type': u'Point', u'coordinates': [-83.0892920651423, 42.3819394423952]}, u'TotSqFt': 6665, u'PropStr': u'EDISON', u'AverageHouseValue': 67700.0}, u'dis': 15.208734671883157}]}

In [ ]:

In [72]:
```
cursor = db.salesHistory.aggregate([
#    { '$match': {'SALEDATE.year' : 2015, 'PROPZIP': 48202} }
    { '$match': {'PROPNO': 1000, "PROPSTR":'BALDWIN'} }
    ,{ '$unwind': "$SALEDATE"}
    ,{ '$lookup': {
            'from': "zipHoods",
            'localField': "PROPZIP",
            'foreignField': "zipcode",
            'as': "zipName"} }
    ,{ '$unwind': "$zipName"}
    ,{ '$project': {"Year": "$SALEDATE.year",
                    "PROPZIP":1, "PROPNO":1, "PROPSTR":1,
                    "AverageHouseValue": "$zipName.AverageHouseValue",
                    "IncomePerHousehold": "$zipName.IncomePerHousehold",
                    "Population": "$zipName.Population",
                    #"zipName.AverageHouseValue":1, "zipName.Population":1, "z
ipName.IncomePerHousehold":1,
                    "SALEPRICE":1,
                    "_id":0} }
    #,{ "$limit": 500 }
    #,{ "$out": "saletempyear" }
])
for document in cursor:
    print(document)
    #pp_json(document)
```

```
{u'AverageHouseValue': 34200.0, u'PROPNO': 1000, u'PROPSTR': u'BALDWIN', u'SA
LEPRICE': u'$43000.00', u'Year': 2005.0, u'IncomePerHousehold': 21600.0, u'Po
pulation': 22759.0, u'PROPZIP': 48214}
```

In [122]:

```
cursor = db.addressTotals.aggregate(
    [
     { '$match': {'address': '1000 BALDWIN'} }
        #,{ '$unwind': "$type"}
        ,{ "$project" : {
             "_id": 1,
            "address": 1,
            "Blight": 1,
            "Compliance": 1,
            "FullPayment": 1,
            "NoPayment": 1,
            "NumViolations": 1,
            "PartialPayment": 1,
            "address": 1,
            "certificate": 1,
            "changeUse": 1,
            "closure": 1,
            "container": 1,
            "defectiveStructure": 1,
            "electrical": 1,
            "emergencyDanger": 1,
            "graffiti": 1,
            "illegalOccupation": 1,
            "notMaintained": 1,
            "snowice": 1,
            "waste": 1,
            "weedsVehicles": 1,
             "type": { "$literal": ["Latitude","Longitude"] },
                    "_id":0} }
        ,{ '$lookup': {
                'from': "parcelOwnership",
                'localField': "address",
                'foreignField': "PropAddr",
                'as': "parcel_address"} }
            ,{ '$unwind': "$parcel_address"}
        ,{ '$lookup': {
                'from': "zipHoods",
                'localField': "parcel_address.PropZip",
                'foreignField': "zipcode",
                'as': "zipName"} }
        ,{ '$unwind': "$zipName"}
        ,{ "$unwind": "$type" }
        ,{ "$group": {
            "_id": "$address",
             "address": { "$first": "$address" },
            "ParcelNo": { "$first": "$parcel_address.ParcelNo" },
            "zip": { "$first": "$parcel_address.PropZip" },
            "AverageHouseValue": { "$first": "$zipName.AverageHouseValue" },
            "IncomePerHousehold": { "$first": "$zipName.IncomePerHousehold" },
            "Population": { "$first": "$zipName.Population" },
            "Ward" : { "$first": "$parcel_address.Ward" },
            "Longitude" : { "$first": "$parcel_address.Longitude" },
            "TaxStatus"  : { "$first": "$parcel_address.TaxStatus" },
            "Taxpayer1" : { "$first": "$parcel_address.Taxpayer1" },
             "sumof":{ "$sum": "$NumViolations" },
            "Blight":   { "$first": "$Blight" },
            "Compliance":   { "$first": "$Compliance" },
```

```
        "FullPayment":   { "$first": "FullPayment" },
        "NoPayment":   { "$first": "$NoPayment" },
        "NumViolations":   { "$first": "$NumViolations" },
        "PartialPayment":   { "$first": "$PartialPayment" },
        "address":   { "$first": "$address" },
        "certificate": { "$first": "$certificate" },
        "changeUse":   { "$first": "$changeUse" },
        "closure":   { "$first": "$closure" },
        "container":   { "$first": "$container" },
        "defectiveStructure": { "$first": "$defectiveStructure" },
        "electrical":   { "$first": "$electrical" },
        "emergencyDanger":   { "$first": "$emergencyDanger" },
        "graffiti":   { "$first": "$graffiti" },
        "illegalOccupation": { "$first": "$illegalOccupation" },
        "notMaintained":   { "$first": "$notMaintained" },
        "snowice":   { "$first": "$snowice" },
        "waste":   { "$first": "$waste" },
        "weedsVehicles":   { "$first": "$weedsVehicles" },
        "Location" : { "$first": "$parcel_address.Location" },
        "Latitude" : { "$first": "$parcel_address.Latitude" },
        "coordinates": {
            "$push": {
                "$cond": [
                    { "$eq": [ "$type", "Latitude" ] },
                    "$parcel_address.Longitude",
                    "$parcel_address.Latitude"
                ]
            }
        }
    }}
    ,{ "$project" : {
         "_id": 1,
        "address": 1,
        "zip" : 1,
        "AverageHouseValue": 1,
        "IncomePerHousehold": 1,
        "Population": 1,
        "TaxStatus"  : 1,
        "Taxpayer1" : 1,
        "Ward" : 1,
        "Location" : 1,
        "ParcelNo": 1,
        "Blight": 1,
        "Compliance": 1,
        "FullPayment": 1,
        "NoPayment": 1,
        "NumViolations": 1,
        "PartialPayment": 1,
        "address": 1,
        "certificate": 1,
        "changeUse": 1,
        "closure": 1,
        "container": 1,
        "defectiveStructure": 1,
        "electrical": 1,
        "emergencyDanger": 1,
        "graffiti": 1,
```

```
            "illegalOccupation": 1,
            "notMaintained": 1,
            "snowice": 1,
            "waste": 1,
            "weedsVehicles": 1,
            "Longitude" : 1,
            "Latitude" : 1,
            "geometry": {
                "type": { "$literal": "Point" },
                "coordinates": "$coordinates"
                        }
       }}
       ,{ "$limit": 25 }
       #,{ "$sort": { "distance": 1 } }
])
for document in cursor:
    print(document)
    #pp_json(document)
```

```
{u'changeUse': 0, u'TaxStatus': u'COUNTY OWNED', u'defectiveStructure': 0,
 u'Taxpayer1': u'WAYNE COUNTY TREASURER', u'illegalOccupation': 0, u'FullPaym
ent': u'FullPayment', u'Latitude': 42.3522659383036, u'ParcelNo': 17011063.0,
 u'graffiti': 0, u'Location': u'(42.3522659383036, -82.9992682509306)', u'Bli
ght': u'yes', u'weedsVehicles': 1, u'IncomePerHousehold': 21600.0, u'snowic
e': 0, u'NoPayment': 4, u'container': 0, u'certificate': 2, u'NumViolations':
 4, u'electrical': 0, u'Ward': 17, u'waste': 0, u'Population': 22759.0, u'Par
tialPayment': 0, u'Compliance': 1, u'Longitude': -82.9992682509306, u'addres
s': u'1000 BALDWIN', u'closure': 0, u'AverageHouseValue': 34200.0, u'zip': 48
214, u'notMaintained': 1, u'geometry': {u'type': u'Point', u'coordinates': [-
82.9992682509306, 42.3522659383036]}, u'emergencyDanger': 0, u'_id': u'1000 B
ALDWIN'}
```

In [25]:

```
def find_building(testAddress):
    cursor = db.addressTotals.aggregate(
        [
         { '$match': {'address': testAddress} }
            #,{ '$unwind': "$type"}
            ,{ "$project" : {
                 "_id": 1,
                "address": 1,
                "Blight": 1,
                "Compliance": 1,
                "FullPayment": 1,
                "NoPayment": 1,
                "NumViolations": 1,
                "PartialPayment": 1,
                "address": 1,
                "certificate": 1,
                "changeUse": 1,
                "closure": 1,
                "container": 1,
                "defectiveStructure": 1,
                "electrical": 1,
                "emergencyDanger": 1,
                "graffiti": 1,
                "illegalOccupation": 1,
                "notMaintained": 1,
                "snowice": 1,
                "waste": 1,
                "weedsVehicles": 1,
                 "taxstatus": { "$literal": ["Taxpayer1","TaxStatus"] },
                 "type": { "$literal": ["Latitude","Longitude"] },
                        "_id":0} }
            ,{ '$lookup': {
                    'from': "parcelOwnership",
                    'localField': "address",
                    'foreignField': "PropAddr",
                    'as': "parcel_address"} }
                ,{ '$unwind': "$parcel_address"}
            ,{ '$lookup': {
                    'from': "zipHoods",
                    'localField': "parcel_address.PropZip",
                    'foreignField': "zipcode",
                    'as': "zipName"} }
            ,{ '$unwind': "$zipName"}
            ,{ "$unwind": "$type" }
            ,{ "$unwind": "$taxstatus" }
            ,{ "$group": {
                "_id": "$address",
                 "address": { "$first": "$address" },
                "ParcelNo": { "$first": "$parcel_address.ParcelNo" },
                "zip": { "$first": "$parcel_address.PropZip" },
                "AverageHouseValue": { "$first": "$zipName.AverageHouseValue"
},
                "IncomePerHousehold": { "$first": "$zipName.IncomePerHouseholl
d" },
                "Population": { "$first": "$zipName.Population" },
                "Ward" : { "$first": "$parcel_address.Ward" },
                "SalePrice"  : { "$first": "$parcel_address.SalePrice" },
```

```
                "Longitude" : { "$first": "$parcel_address.Longitude" },
                 "sumof":{ "$sum": "$NumViolations" },
                "Blight":   { "$first": "$Blight" },
                "Compliance":   { "$first": "$Compliance" },
                "FullPayment":   { "$first": "$FullPayment" },
                "NoPayment":   { "$first": "$NoPayment" },
                "NumViolations":   { "$first": "$NumViolations" },
                "PartialPayment":   { "$first": "$PartialPayment" },
                "address":   { "$first": "$address" },
                "certificate": { "$first": "$certificate" },
                "changeUse":   { "$first": "$changeUse" },
                "closure":   { "$first": "$closure" },
                "container":   { "$first": "$container" },
                "defectiveStructure": { "$first": "$defectiveStructure" },
                "electrical":   { "$first": "$electrical" },
                "emergencyDanger":   { "$first": "$emergencyDanger" },
                "graffiti":   { "$first": "$graffiti" },
                "illegalOccupation": { "$first": "$illegalOccupation" },
                "notMaintained":   { "$first": "$notMaintained" },
                "snowice":   { "$first": "$snowice" },
                "waste":   { "$first": "$waste" },
                "weedsVehicles":   { "$first": "$weedsVehicles" },
                "Location" : { "$first": "$parcel_address.Location" },
                "Latitude" : { "$first": "$parcel_address.Latitude" },
                "coordinates": {
                    "$push": {
                        "$cond": [
                            { "$eq": [ "$type", "Latitude" ] },
                            "$parcel_address.Longitude",
                            "$parcel_address.Latitude"
                        ]
                    }
                },
                "TaxStatus"  : { "$first": "$parcel_address.TaxStatus" },
                "Taxpayer1" : { "$first": "$parcel_address.Taxpayer1" },
                "status": {
                    "$addToSet": {
                        "$cond": [
                            { "$eq": [ "$taxstatus", "TaxStatus" ] },
                            "$parcel_address.Taxpayer1",
                            "$parcel_address.TaxStatus"
                        ]
                    }
                },

            }}
            ,{ "$project" : {
                 "_id": 0,
                "address": 1,
                "zip" : 1,
                "AverageHouseValue": 1,
                "IncomePerHousehold": 1,
                "Population": 1,
                #"TaxStatus"  : 1,
                #"Taxpayer1" : 1,
                "Ward" : 1,
                 "SalePrice"  : { "$substr": [ "$SalePrice", 1, -1 ] },
```

```
                #"Location" : 1,
                #"ParcelNo": 1,
                "Blight": 1,
                "Compliance": 1,
                "FullPayment": 1,
                "NoPayment": 1,
                "NumViolations": 1,
                "PartialPayment": 1,
                "address": 1,
                "certificate": 1,
                "changeUse": 1,
                "closure": 1,
                "container": 1,
                "defectiveStructure": 1,
                "electrical": 1,
                "emergencyDanger": 1,
                "graffiti": 1,
                "illegalOccupation": 1,
                "notMaintained": 1,
                "snowice": 1,
                "waste": 1,
                "weedsVehicles": 1,
                #"ownerTaxStatus" :
                #{
                 #"$cond" : { "if": { "$eq": [ "$parcel_address.TaxStatus",
 "COUNTY OWNED" ] }, "then": 0, "else": 20 }
                #},
                #"status": "$status"
                "Longitude" : 1,
                "Latitude" : 1
                #"geometry": {
                    #"type": { "$literal": "Point" },
                    #"coordinates": "$coordinates"
                            #}
            }}
            #,{ "$limit": 25 }
            #,{ "$sort": { "distance": 1 } }
    ])
    return cursor;
```

In [26]:
```
cursor = find_building('1000 BALDWIN')
for document in cursor:
    print(document)
    #pp_json(document)
```

```
{u'changeUse': 0, u'weedsVehicles': 1, u'defectiveStructure': 0, u'illegalOcc
upation': 0, u'FullPayment': 0, u'Latitude': 42.3522659383036, u'graffiti':
 0, u'SalePrice': u'43000.00', u'Blight': u'yes', u'IncomePerHousehold': 2160
0.0, u'snowice': 0, u'NoPayment': 4, u'container': 0, u'zip': 48214, u'NumVio
lations': 4, u'electrical': 0, u'Ward': 17, u'waste': 0, u'Population': 2275
9.0, u'PartialPayment': 0, u'Compliance': 1, u'Longitude': -82.9992682509306,
 u'address': u'1000 BALDWIN', u'closure': 0, u'certificate': 2, u'notMaintain
ed': 1, u'emergencyDanger': 0, u'AverageHouseValue': 34200.0}
```

In [112]:

```
def all_buildings():
    cursor = db.addressTotals.aggregate(
        [
         { '$match': {'address':  {'$gt': '0'}} }
            #,{ '$unwind': "$type"}
            ,{ "$project" : {
                 "_id": 1,
                "address": 1,
                "Blight": 1,
                "Compliance": 1,
                "FullPayment": 1,
                "NoPayment": 1,
                "NumViolations": 1,
                "PartialPayment": 1,
                "address": 1,
                "certificate": 1,
                "changeUse": 1,
                "closure": 1,
                "container": 1,
                "defectiveStructure": 1,
                "electrical": 1,
                "emergencyDanger": 1,
                "graffiti": 1,
                "illegalOccupation": 1,
                "notMaintained": 1,
                "snowice": 1,
                "waste": 1,
                "weedsVehicles": 1,
                 "taxstatus": { "$literal": ["Taxpayer1","TaxStatus"] },
                 "type": { "$literal": ["Latitude","Longitude"] },
                       "_id":0} }
            ,{ '$lookup': {
                    'from': "parcelOwnership",
                    'localField': "address",
                    'foreignField': "PropAddr",
                    'as': "parcel_address"} }
                ,{ '$unwind': "$parcel_address"}
            ,{ '$lookup': {
                    'from': "zipHoods",
                    'localField': "parcel_address.PropZip",
                    'foreignField': "zipcode",
                    'as': "zipName"} }
            ,{ '$unwind': "$zipName"}
            ,{ "$unwind": "$type" }
            ,{ "$unwind": "$taxstatus" }
            ,{ "$group": {
                "_id": "$address",
                 "address": { "$first": "$address" },
                "ParcelNo": { "$first": "$parcel_address.ParcelNo" },
                "zip": { "$first": "$parcel_address.PropZip" },
                "AverageHouseValue": { "$first": "$zipName.AverageHouseValue"
},
                "IncomePerHousehold": { "$first": "$zipName.IncomePerHouseholdd" },
                "Population": { "$first": "$zipName.Population" },
                "Ward" : { "$first": "$parcel_address.Ward" },
                "SalePrice"  : { "$first": "$parcel_address.SalePrice" },
```

```
                "Longitude" : { "$first": "$parcel_address.Longitude" },
                 "sumof":{ "$sum": "$NumViolations" },
                "Blight":   { "$first": "$Blight" },
                "Compliance":   { "$first": "$Compliance" },
                "FullPayment":   { "$first": "$FullPayment" },
                "NoPayment":   { "$first": "$NoPayment" },
                "NumViolations":   { "$first": "$NumViolations" },
                "PartialPayment":   { "$first": "$PartialPayment" },
                "address":   { "$first": "$address" },
                "certificate": { "$first": "$certificate" },
                "changeUse":   { "$first": "$changeUse" },
                "closure":   { "$first": "$closure" },
                "container":   { "$first": "$container" },
                "defectiveStructure": { "$first": "$defectiveStructure" },
                "electrical":   { "$first": "$electrical" },
                "emergencyDanger":   { "$first": "$emergencyDanger" },
                "graffiti":   { "$first": "$graffiti" },
                "illegalOccupation": { "$first": "$illegalOccupation" },
                "notMaintained":   { "$first": "$notMaintained" },
                "snowice":   { "$first": "$snowice" },
                "waste":   { "$first": "$waste" },
                "weedsVehicles":   { "$first": "$weedsVehicles" },
                "Location" : { "$first": "$parcel_address.Location" },
                "Latitude" : { "$first": "$parcel_address.Latitude" },
                "coordinates": {
                    "$push": {
                        "$cond": [
                            { "$eq": [ "$type", "Latitude" ] },
                            "$parcel_address.Longitude",
                            "$parcel_address.Latitude"
                        ]
                    }
                },
                "TaxStatus"  : { "$first": "$parcel_address.TaxStatus" },
                "Taxpayer1" : { "$first": "$parcel_address.Taxpayer1" },
                "status": {
                    "$addToSet": {
                        "$cond": [
                            { "$eq": [ "$taxstatus", "TaxStatus" ] },
                            "$parcel_address.Taxpayer1",
                            "$parcel_address.TaxStatus"
                        ]
                    }
                },

            }}
            ,{ "$project" : {
                 "_id": 0,
                "address": 1,
                "zip" : 1,
                "AverageHouseValue": 1,
                "IncomePerHousehold": 1,
                "Population": 1,
                #"TaxStatus"  : 1,
                #"Taxpayer1" : 1,
                "Ward" : 1,
                 "SalePrice"  : { "$substr": [ "$SalePrice", 1, -1 ] },
```

```
                #"Location" : 1,
                #"ParcelNo": 1,
                "Blight": 1,
                "Compliance": 1,
                "FullPayment": 1,
                "NoPayment": 1,
                "NumViolations": 1,
                "PartialPayment": 1,
                "address": 1,
                "certificate": 1,
                "changeUse": 1,
                "closure": 1,
                "container": 1,
                "defectiveStructure": 1,
                "electrical": 1,
                "emergencyDanger": 1,
                "graffiti": 1,
                "illegalOccupation": 1,
                "notMaintained": 1,
                "snowice": 1,
                "waste": 1,
                "weedsVehicles": 1,
                #"ownerTaxStatus" :
                #{
                 #"$cond" : { "if": { "$eq": [ "$parcel_address.TaxStatus",
 "COUNTY OWNED" ] }, "then": 0, "else": 20 }
                #},
                #"status": "$status"
                "Longitude" : 1,
                "Latitude" : 1
                #"geometry": {
                    #"type": { "$literal": "Point" },
                    #"coordinates": "$coordinates"
                            #}
            }}
            ,{ "$limit": 25 }
            #,{ "$sort": { "distance": 1 } }
    ])
    return cursor;
```

In [ ]:

```
cursor = all_buildings()
for document in cursor:
    print(document)
    #pp_json(document)
```

In [454]:
```
def find_nearBuilding(testLat,testLng):
    db = c.mydb
    print(testLat)
    cursor = db.newGeoPt1.aggregate(
        [
         { "$geoNear": {
             "near": {
                        "type": "Point",
                        "geometry": [-82.9992682509306, 42.3522659383036]
             },
             "maxDistance": 0.00001261617096,
             "distanceField": "distance",
             "includeLocs": "latlng",
             "uniqueDocs": True,
             "spherical":True,
             "query": {
                 'TicketNumber' : {'$gt': '0'}
             },
             "limit":100
         }},
         { "$sort": { "distance": 1000 } }
    ])
    for document in cursor:
        print(document)
        #pp_json(document)
```

In [9]:

```
cursor = db.addressTotals.aggregate(
    [
     { '$match': {'address': '1000 BALDWIN'} }
        #,{ '$unwind': "$type"}
        ,{ "$project" : {
             "_id": 1,
            "address": 1,
            "Blight": 1,
            "Compliance": 1,
            "FullPayment": 1,
            "NoPayment": 1,
            "NumViolations": 1,
            "PartialPayment": 1,
            "address": 1,
            "certificate": 1,
            "changeUse": 1,
            "closure": 1,
            "container": 1,
            "defectiveStructure": 1,
            "electrical": 1,
            "emergencyDanger": 1,
            "graffiti": 1,
            "illegalOccupation": 1,
            "notMaintained": 1,
            "snowice": 1,
            "waste": 1,
            "weedsVehicles": 1,
            "longitude": 1,
            "latitude": 1} }
        ,{ '$lookup': {
                'from': "parcelOwnership",
                'localField': "address",
                'foreignField': "PropAddr",
                'as': "parcel_address"} }
            ,{ '$unwind': "$parcel_address"}
        ,{ '$lookup': {
                'from': "zipHoods",
                'localField': "parcel_address.PropZip",
                'foreignField': "zipcode",
                'as': "zipName"} }
        ,{ '$unwind': "$zipName"}
        ,{ "$unwind": "$type" }
        ,{ "$group": {
            "_id": "$address",
             "address": { "$first": "$address" },
            "ParcelNo": { "$first": "$parcel_address.ParcelNo" },
            "zip": { "$first": "$parcel_address.PropZip" },
            "AverageHouseValue": { "$first": "$zipName.AverageHouseValue" },
            "IncomePerHousehold": { "$first": "$zipName.IncomePerHousehold" },
            "Population": { "$first": "$zipName.Population" },
            "Ward" : { "$first": "$parcel_address.Ward" },
            "Longitude" : { "$first": "$parcel_address.Longitude" },
            "TaxStatus"  : { "$first": "$parcel_address.TaxStatus" },
            "Taxpayer1" : { "$first": "$parcel_address.Taxpayer1" },
             "sumof":{ "$sum": "$NumViolations" },
            "Blight":   { "$first": "$Blight" },
            "Compliance":   { "$first": "$Compliance" },
```

```
        "FullPayment":   { "$first": "FullPayment" },
        "NoPayment":   { "$first": "$NoPayment" },
        "NumViolations":   { "$first": "$NumViolations" },
        "PartialPayment":   { "$first": "$PartialPayment" },
        "address":   { "$first": "$address" },
        "certificate": { "$first": "$certificate" },
        "changeUse":   { "$first": "$changeUse" },
        "closure":   { "$first": "$closure" },
        "container":   { "$first": "$container" },
        "defectiveStructure": { "$first": "$defectiveStructure" },
        "electrical":   { "$first": "$electrical" },
        "emergencyDanger":   { "$first": "$emergencyDanger" },
        "graffiti":   { "$first": "$graffiti" },
        "illegalOccupation": { "$first": "$illegalOccupation" },
        "notMaintained":   { "$first": "$notMaintained" },
        "snowice":   { "$first": "$snowice" },
        "waste":   { "$first": "$waste" },
        "weedsVehicles":   { "$first": "$weedsVehicles" },
        "Location" : { "$first": "$parcel_address.Location" },
        "Latitude" : { "$first": "$parcel_address.Latitude" },
        "coordinates": {
            "$push": {
                "$cond": [
                    { "$eq": [ "$type", "Latitude" ] },
                    "$parcel_address.Longitude",
                    "$parcel_address.Latitude"
                ]
            }
        }
    }}
    ,{ "$project" : {
         "_id": 1,
        "address": 1,
        "zip" : 1,
        "AverageHouseValue": 1,
        "IncomePerHousehold": 1,
        "Population": 1,
        "TaxStatus"  : 1,
        "Taxpayer1" : 1,
        "Ward" : 1,
        "Location" : 1,
        "ParcelNo": 1,
        "Blight": 1,
        "Compliance": 1,
        "FullPayment": 1,
        "NoPayment": 1,
        "NumViolations": 1,
        "PartialPayment": 1,
        "address": 1,
        "certificate": 1,
        "changeUse": 1,
        "closure": 1,
        "container": 1,
        "defectiveStructure": 1,
        "electrical": 1,
        "emergencyDanger": 1,
        "graffiti": 1,
```

```
            "illegalOccupation": 1,
            "notMaintained": 1,
            "snowice": 1,
            "waste": 1,
            "weedsVehicles": 1,
            "Longitude" : 1,
            "Latitude" : 1,
            "geometry": {
                "type": { "$literal": "Point" },
                "coordinates": "$coordinates"
                        }
       }}
       ,{ "$limit": 25 }
       #,{ "$sort": { "distance": 1 } }
])
for document in cursor:
    print(document)
    #pp_json(document)
```

In [413]:
```
db = c.mydb
cursor=db.blightVioInfo.aggregate([
     { '$match': {"TicketNumber" : "13077645DAH"} }
        #{'$sort': {'c': -1}},
        #,{'$group': {'_id': { 'StreetNumber': '$ViolationStreetNumber',
        #                     'StreetName': '$ViolationStreetName'}, 'count': {
 '$sum': 1 }}}
       ,{'$project':
          {
            'sourceID': '$TicketID',
            #'address': { '$concat': [ "$ViolationStreetNumber", " ", "$Violat
ionStreetName" ] } ,
            'streetNumber': '$ViolationStreetNumber',
            'streetName': '$ViolationStreetName',
            "lng" : "$Longitude" ,
            "lat" : "$Latitude" ,
            "source" : { "$literal": "vio" } ,
            'count': { '$sum': 1 } }}
    ,{ "$limit": 500 }
#    ,{ "$out": "blighttempstreet" }
    ]
)
for document in cursor:
    print(document)
```

```
{u'count': 1, u'sourceID': 304496, u'source': u'vio', u'streetNumber': 22580,
 u'lat': u'', u'lng': u'', u'_id': ObjectId('57f8348847fbffca288bf0a5'), u'st
reetName': u'ARGUS'}
```

In [102]:
```
cursor=db.newTestBuildings.aggregate([
        {'$match': {'source': {'$eq': 'crime'}}}
        #{'$sort': {'c': -1}},
        ,{ "$project" : {
             "_id": 1,
            "streetAddres": 1,
            "lat": 1,
            "lng": 1,
            "source": 1,
            "type": { "$literal": ["Latitude","Longitude"] },
                    "_id":0} }
        ,{ "$unwind": "$type" }
        ,{'$group': {'_id':  '$streetAddres',
                     'streetAddress': { "$first":'$streetAddres'},
                     'source': { "$first":'$source'},
                    'count': { '$sum': 1 },
                    'Latitude': { "$first":'$lat'},
                    'Longitude': { "$first":'$lng'},
                    "coordinates": {
                        "$push": {
                            "$cond": [
                                { "$eq": [ "$type", "Latitude", ] },
                                "$lng",
                                "$lat"
                            ]
                        }
                    }

        }}
        ,{'$project': {
                  "_id": 0,
                 'streetAddress': 1 ,
                 'latitude': 1,
                 'longitude': 1,
                 'source': 1,
                 "coordinates": "$coordinates",
                 'count':  '$count'  }}
       # ,{ "$limit": 500 }
    ,{ "$out": "crimetempstreet" }
    ]
)
for document in cursor:
    print(document)
```

In [103]:
```
mongo_a_search_results= load_from_mongo('mydb', 'crimetempstreet')
dfCrime = pd.DataFrame(mongo_a_search_results)
dfCrime.head()
```

Out[103]:

	_id	coordinates	count	source	streetAddress
0	57fe8292fbb996bb0960fca5	[-83.0795, 42.3481, -83.079, 42.3483, -83.0789...	10	crime	04500 AVERY
1	57fe8292fbb996bb0960fca6	[-83.2105, 42.442]	2	crime	20200 ASHBURY PARK
2	57fe8292fbb996bb0960fca7	[-83.0421, 42.334]	2	crime	00 E. LAFAYETTE
3	57fe8292fbb996bb0960fca8	[-83.0641, 42.3237]	2	crime	01600 W FORT ST APT 204
4	57fe8292fbb996bb0960fca9	[-83.1856, 42.4075]	2	crime	00 PILGRIM AND FREELAND

In [104]:
```
dfCrime.count()
```

Out[104]:
```
_id              57323
coordinates      57323
count            57323
source           57323
streetAddress    57323
dtype: int64
```

In [106]:
```
dfCrime.to_csv("crimeGeoCoord.csv")
```

In [ ]:
```
cursor=db.recsBlightGeo.aggregate([
        {'$match': {'ViolationStreetNumber': {'$gt': '0'}}}
        #{'$sort': {'c': -1}},
        #,{'$group': {'_id': { 'StreetNumber': '$ViolationStreetNumber',
        #                      'StreetName': '$ViolationStreetName'}, 'count': {
 '$sum': 1 }}}
        ,{'$project': { 'streetAddress': { '$concat': [ "$ViolationStreetNumbe
r", " ", "$ViolationStreetName" ] } ,
          'StreetNumber': '$ViolationStreetNumber',
          'StreetName': '$ViolationStreetName', 'count': { '$sum': 1 } }}
    ,{ "$limit": 5 }
#    ,{ "$out": "blighttempstreet" }
    ]
)
for document in cursor:
    print(document)
```

db.triainNewBlightGeo.drop() for index, row in dfNewTotals.iterrows(): print row['address'], index cursor = find_building(row['address']) for document in cursor: print(document) db.triainNewBlightGeo.insert(document)

In [424]:
```
testAddress = '10041 ARTESIAN'
cursor = db.addressTotals.aggregate(
    [
     { '$match': {'address': testAddress} }
    ,{ '$lookup': {
              'from': "newTestBuildings",
              'localField': "address",
              'foreignField': "streetAddres",
              'as': "address"} }
         ,{ '$unwind': "$address"}
#         ,{ '$unwind': "$userAddressList"}
         ,{ "$group": {
              "_id": "$address",
               "address": { "$first": "$address" },
              "streetAddres": { "$first": "$address.streetAddres" },
              "source": { "$first": "$address.source" },
               "sumof":{ "$sum": "$NumViolations" }
         }}
         ,{ "$project" : {
               "_id": 1,
              "address": 1,
              #"source" : 1,
               "streetAddres": 1
        }}
        ,{ "$limit": 25 }
        #,{ "$sort": { "distance": 1 } }
])
for document in cursor:
    print(document)
    #pp_json(document)
```

```
{u'streetAddres': u'10041 ARTESIAN', u'_id': {u'streetAddres': u'10041 ARTESI
AN', u'source': u'vio', u'streetNo': u'10041', u'location': u'42.369288895000
07, -83.22645803799998', u'address': u'10041 ARTESIAN Detroit, MI', u'lat':
 0, u'lng': 0, u'_id': ObjectId('57e0123039d4cac6412e0ccf'), u'streetName':
 u'ARTESIAN'}, u'address': {u'streetAddres': u'10041 ARTESIAN', u'source':
 u'vio', u'streetNo': u'10041', u'location': u'42.36928889500007, -83.2264580
3799998', u'address': u'10041 ARTESIAN Detroit, MI', u'lat': 0, u'lng': 0,
 u'_id': ObjectId('57e0123039d4cac6412e0ccf'), u'streetName': u'ARTESIAN'}}
```

1.6 Plotly Charting Data

In [3]:
```
import pandas as pd
from sqlalchemy import create_engine # database connection
import datetime as dt
from IPython.display import display

import plotly.plotly as py # interactive graphing
from plotly.graph_objs import Bar, Scatter, Marker, Layout
```

In [4]:
```
disk_engine = create_engine('sqlite:///detroit-blight-demo-crime-311.db') # In
itializes database with filename 311_8M.db in current directory
```

In [5]:
```
start = dt.datetime.now()
chunksize = 20000
j = 0
index_start = 1

for df in pd.read_csv('detroit-blight-violations.csv', chunksize=chunksize, it
erator=True, encoding='utf-8'):

    #df = df.rename(columns={c: c.replace(' ', '') for c in df.columns}) # Rem
ove spaces from columns

    #df['"index"'] = pd.to_datetime(df['"index"']) # Convert to datetimes

    df.index += index_start

    # Remove the un-interesting columns
    columns = ['Agency', 'CreatedDate', 'ClosedDate', 'ComplaintType', 'Descri
ptor',
               'CreatedDate', 'ClosedDate', 'TimeToCompletion',
               'City']

    #for c in df.columns:
        #if c not in columns:
            #df = df.drop(c, axis=1)


    j+=1
    print '{} seconds: completed {} rows'.format((dt.datetime.now() - start).s
econds, j*chunksize)

    df.to_sql('data', disk_engine, if_exists='append')
    index_start = df.index[-1] + 1
```

```
0 seconds: completed 20000 rows
7 seconds: completed 40000 rows
11 seconds: completed 60000 rows
15 seconds: completed 80000 rows
18 seconds: completed 100000 rows
22 seconds: completed 120000 rows
25 seconds: completed 140000 rows
28 seconds: completed 160000 rows
32 seconds: completed 180000 rows
36 seconds: completed 200000 rows
39 seconds: completed 220000 rows
43 seconds: completed 240000 rows
46 seconds: completed 260000 rows
49 seconds: completed 280000 rows
52 seconds: completed 300000 rows
56 seconds: completed 320000 rows
```

In [6]:
```
start = dt.datetime.now()
chunksize = 20000
j = 0
index_start = 1

for df in pd.read_csv('detroit-crime.csv', chunksize=chunksize, iterator=True,
 encoding='utf-8'):

    #df = df.rename(columns={c: c.replace(' ', '') for c in df.columns}) # Rem
ove spaces from columns

    #df['"index"'] = pd.to_datetime(df['"index"']) # Convert to datetimes

    df.index += index_start

    # Remove the un-interesting columns
    columns = ['Agency', 'CreatedDate', 'ClosedDate', 'ComplaintType', 'Descri
ptor',
               'CreatedDate', 'ClosedDate', 'TimeToCompletion',
               'City']

    #for c in df.columns:
        #if c not in columns:
            #df = df.drop(c, axis=1)


    j+=1
    print '{} seconds: completed {} rows'.format((dt.datetime.now() - start).s
econds, j*chunksize)

    df.to_sql('datacrime', disk_engine, if_exists='append')
    index_start = df.index[-1] + 1
```

```
0 seconds: completed 20000 rows
1 seconds: completed 40000 rows
4 seconds: completed 60000 rows
7 seconds: completed 80000 rows
9 seconds: completed 100000 rows
11 seconds: completed 120000 rows
```

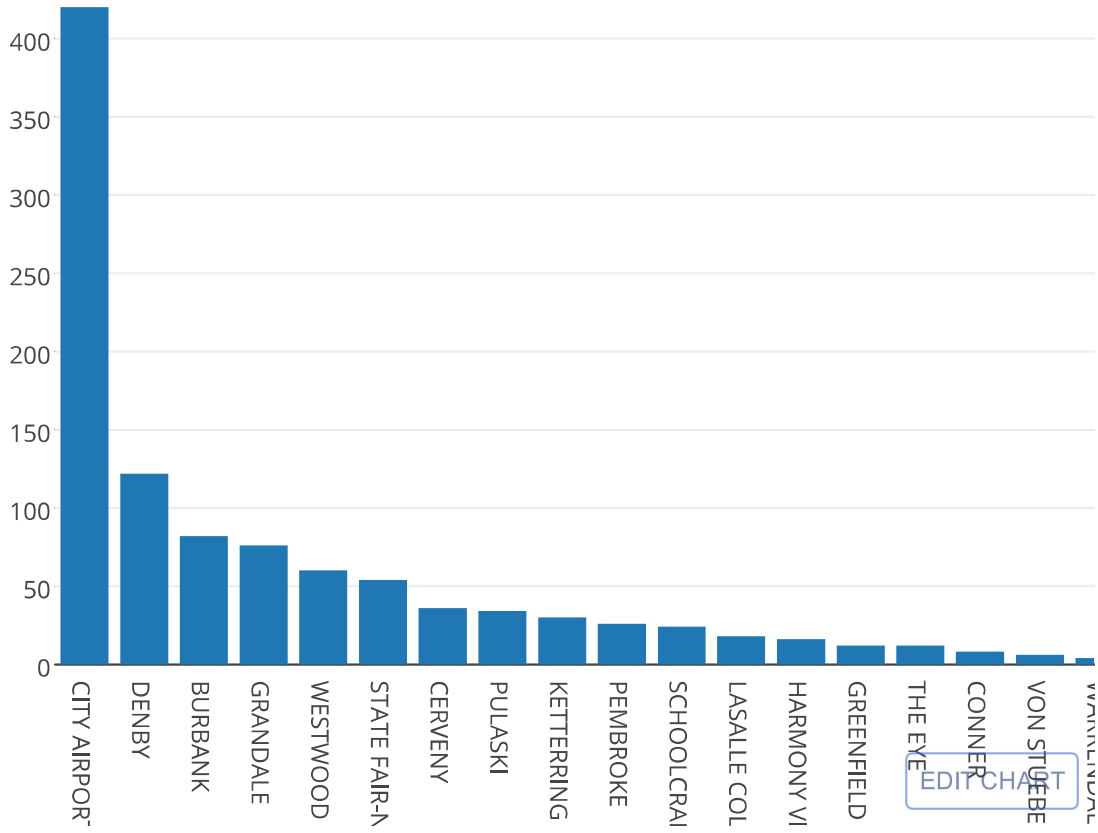
In [7]:
```
start = dt.datetime.now()
chunksize = 20000
j = 0
index_start = 1

for df in pd.read_csv('detroit-blight-demo-crime.csv', chunksize=chunksize, it
erator=True, encoding='utf-8'):

    #df = df.rename(columns={c: c.replace(' ', '') for c in df.columns}) # Rem
ove spaces from columns

    #df['"index"'] = pd.to_datetime(df['"index"']) # Convert to datetimes

    df.index += index_start

    # Remove the un-interesting columns
    columns = ['Agency', 'CreatedDate', 'ClosedDate', 'ComplaintType', 'Descri
ptor',
               'CreatedDate', 'ClosedDate', 'TimeToCompletion',
               'City']

    #for c in df.columns:
        #if c not in columns:
            #df = df.drop(c, axis=1)

    df['city']=df['NEIGHBORHOOD']
    j+=1
    print '{} seconds: completed {} rows'.format((dt.datetime.now() - start).s
econds, j*chunksize)

    df.to_sql('datablightdemoscrime', disk_engine, if_exists='append')
    index_start = df.index[-1] + 1
```

```
0 seconds: completed 20000 rows
```

Neighborhoods that receive the most complaints

In [8]:
```
import plotly.tools as tls
tls.set_credentials_file(username='vrjanice', api_key='xxxxxxx')

df = pd.read_sql_query('SELECT NEIGHBORHOOD, COUNT(*) as `num_complaints`'
                       'FROM datablightdemoscrime '
                       'GROUP BY NEIGHBORHOOD '
                       'ORDER BY -num_complaints', disk_engine)

py.iplot([Bar(x=df.NEIGHBORHOOD, y=df.num_complaints)], filename='Blight/most
 common complaints by agency')
```

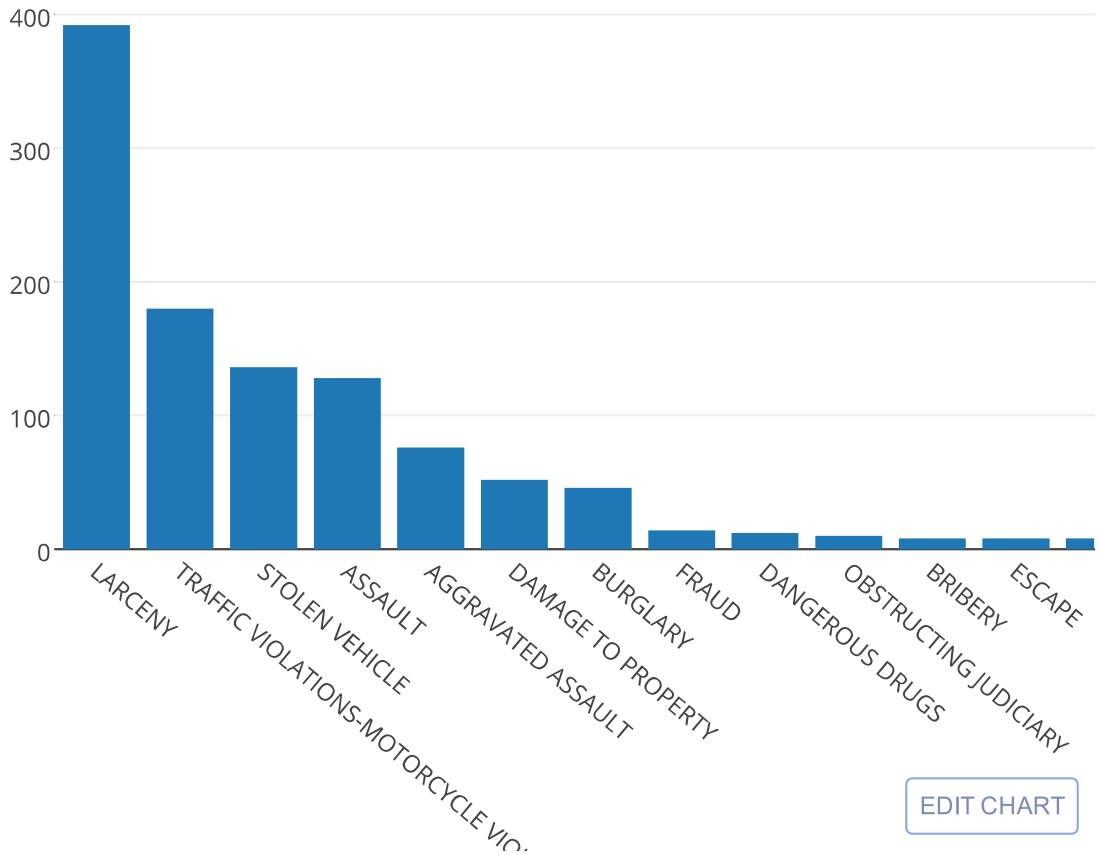
Out[8]:



The most common complaints

In [9]:
```
df = pd.read_sql_query('SELECT CATEGORY, COUNT(*) as `num_complaints`, NEIGHBO
RHOOD '
                       'FROM datablightdemoscrime '
                       'GROUP BY `CATEGORY` '
                       'ORDER BY -num_complaints', disk_engine)


most_common_complaints = df # used later
py.iplot({
    'data': [Bar(x=df['CATEGORY'], y=df.num_complaints)],
    'layout': {
        'margin': {'b': 150}, # Make the bottom margin a bit bigger to handle
 the long text
        'xaxis': {'tickangle': 40}} # Angle the labels a bit
    }, filename='311/most common complaints by complaint type')
```

Out[9]:



In [10]:
```
df.head()
```

Out[10]:

	CATEGORY	num_complaints	NEIGHBORHOOD
0	LARCENY	392	CONNER
1	TRAFFIC VIOLATIONS-MOTORCYCLE VIOLATIONS	180	DENBY
2	STOLEN VEHICLE	136	HARMONY VILLAGE
3	ASSAULT	128	DENBY
4	AGGRAVATED ASSAULT	76	GRANDALE

1.7 Predicting the following month Demolition total based on the previous months

In [12]:
```
mongo_a_search_results= load_from_mongo('mydb', 'mthBlightDemoTotals')
dftotals = pd.DataFrame(mongo_a_search_results)
dftotals.head()
```

Out[12]:

	BSEDept	CityNotResponsible	DPW	DefaultResponsible	DeterminationNotRespons
0	1	0	0	0	0
1	1	0	0	0	0
2	1	1	0	0	0
3	0	0	1	0	0
4	0	1	1	0	0

5 rows × 23 columns

In [13]:
```
dftotals.describe()
```

Out[13]:

	BSEDept	CityNotResponsible	DPW	DefaultResponsible	Determinat
count	5345.000000	5345.000000	5345.000000	5345.000000	5345.00000
mean	0.846586	0.208980	0.694481	1.008606	0.032180
std	1.236620	0.639512	0.917650	1.128662	0.231528
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	1.000000	0.000000
75%	2.000000	0.000000	1.000000	2.000000	0.000000
max	30.000000	8.000000	12.000000	30.000000	6.000000

In [14]:
```
dftotals.dtypes
```

Out[14]:
```
BSEDept                        int64
CityNotResponsible             int64
DPW                            int64
DefaultResponsible             int64
DeterminationNotResponsible    int64
DismissalNotResponsible        int64
FullPayment                    int64
HealthDept                     int64
InvalidYear                    int64
JudgmentAmt                  float64
Month                          int64
NoPayment                      int64
NumViolations                  int64
PARCEL_CLUSTER_SECTOR         object
PENDINGJUDGMENT                int64
PartialPayment                 int64
Police                         int64
Responsible                    int64
ResponsibleAdmission           int64
SITE_ADDRESS                  object
TicketIssuedDT                object
Year                           int64
_id                           object
dtype: object
```

In [15]:
```
dfmthTotals = pd.DataFrame(dftotals['Year'])

dfmthTotals['Month'] = dftotals['Month']
dfmthTotals['NumViolations'] = dftotals['NumViolations']
```

In [16]:
```
dfmthTotals.sort(['Year', 'Month'], ascending=[True, True])
```

Out[16]:

	Year	Month	NumViolations
0	2000	1	1
1	2000	1	1
15	2000	1	1
34	2000	1	1
36	2000	1	1
63	2000	1	1
68	2000	1	2
81	2000	1	1
82	2000	1	2
87	2000	1	2
92	2000	1	1
105	2000	1	1
107	2000	1	3
109	2000	1	1
111	2000	1	2
112	2000	1	1
115	2000	1	1
130	2000	1	1
136	2000	1	2
137	2000	1	1
140	2000	1	1
142	2000	1	2
144	2000	1	1
177	2000	1	1
182	2000	1	3
183	2000	1	3
221	2000	1	1
234	2000	1	6
235	2000	1	3
236	2000	1	30
...	...	...	...
5161	2014	7	2

	Year	Month	NumViolations
5216	2014	7	2
5254	2014	7	1
5320	2014	7	2
5325	2014	7	2
94	2014	8	2
146	2014	8	1
147	2014	8	1
216	2014	8	1
217	2014	8	2
1601	2014	8	1
2258	2014	8	2
3810	2014	8	1
4240	2014	8	2
4957	2014	8	2
5013	2014	8	2
5187	2014	8	2
797	2014	9	2
1511	2014	9	1
2378	2014	9	1
2379	2014	9	1
3393	2014	9	1
4891	2014	9	1
5163	2014	9	2
4051	2014	10	3
4202	2014	10	2
4506	2014	10	1
5011	2014	10	2
3832	2014	12	1
5317	2014	12	2

5345 rows × 3 columns

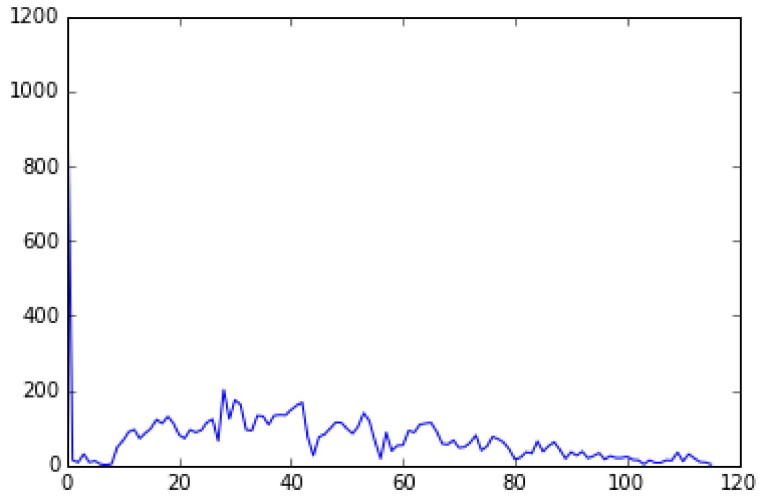
In [17]:
```
grouped = dfmthTotals.groupby(['Year', 'Month'])
```

In [18]:
```
dtfsum = dfmthTotals.groupby(['Year', 'Month'])['NumViolations'].sum()
```

In [19]:
```
dtfsum.values
```

Out[19]:
```
array([1115,   14,    8,   30,    8,   12,    3,    1,    4,   49,   66,
         89,   96,   72,   86,   99,  122,  112,  130,  113,   82,   72,
         95,   88,   94,  115,  123,   65,  202,  124,  175,  162,   95,
         93,  133,  131,  109,  133,  135,  134,  148,  160,  168,   75,
         26,   76,   83,   99,  115,  114,   98,   85,  104,  140,  118,
         64,   18,   88,   39,   53,   54,   93,   88,  109,  112,  114,
         91,   58,   56,   67,   47,   49,   61,   80,   40,   51,   77,
         70,   61,   42,   16,   21,   36,   32,   64,   37,   52,   62,
         42,   18,   36,   26,   37,   19,   25,   33,   16,   25,   20,
         20,   23,   15,   14,    3,   14,    7,    7,   14,   13,   35,
         11,   30,   19,    9,    8,    3], dtype=int64)
```

In [20]:
```
import pandas
import matplotlib.pyplot as plt
#dataset = pandas.read_csv('international-airline-passengers.csv', usecols=
[1], engine='python', skipfooter=3)
plt.plot(dtfsum)
plt.show()
```



In [22]:
```
import numpy
import matplotlib.pyplot as plt
import pandas
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
```

In [23]:
```
data = dtfsum.values
```

In [24]:
```
data = dtfsum.astype('float32')
```

In [25]:
```
data.reshape((116, 1))
```

Out[25]:

```
array([[  1.11500000e+03],
       [  1.40000000e+01],
       [  8.00000000e+00],
       [  3.00000000e+01],
       [  8.00000000e+00],
       [  1.20000000e+01],
       [  3.00000000e+00],
       [  1.00000000e+00],
       [  4.00000000e+00],
       [  4.90000000e+01],
       [  6.60000000e+01],
       [  8.90000000e+01],
       [  9.60000000e+01],
       [  7.20000000e+01],
       [  8.60000000e+01],
       [  9.90000000e+01],
       [  1.22000000e+02],
       [  1.12000000e+02],
       [  1.30000000e+02],
       [  1.13000000e+02],
       [  8.20000000e+01],
       [  7.20000000e+01],
       [  9.50000000e+01],
       [  8.80000000e+01],
       [  9.40000000e+01],
       [  1.15000000e+02],
       [  1.23000000e+02],
       [  6.50000000e+01],
       [  2.02000000e+02],
       [  1.24000000e+02],
       [  1.75000000e+02],
       [  1.62000000e+02],
       [  9.50000000e+01],
       [  9.30000000e+01],
       [  1.33000000e+02],
       [  1.31000000e+02],
       [  1.09000000e+02],
       [  1.33000000e+02],
       [  1.35000000e+02],
       [  1.34000000e+02],
       [  1.48000000e+02],
       [  1.60000000e+02],
       [  1.68000000e+02],
       [  7.50000000e+01],
       [  2.60000000e+01],
       [  7.60000000e+01],
       [  8.30000000e+01],
       [  9.90000000e+01],
       [  1.15000000e+02],
       [  1.14000000e+02],
       [  9.80000000e+01],
       [  8.50000000e+01],
       [  1.04000000e+02],
       [  1.40000000e+02],
       [  1.18000000e+02],
       [  6.40000000e+01],
       [  1.80000000e+01],
```

```
[  8.80000000e+01],
[  3.90000000e+01],
[  5.30000000e+01],
[  5.40000000e+01],
[  9.30000000e+01],
[  8.80000000e+01],
[  1.09000000e+02],
[  1.12000000e+02],
[  1.14000000e+02],
[  9.10000000e+01],
[  5.80000000e+01],
[  5.60000000e+01],
[  6.70000000e+01],
[  4.70000000e+01],
[  4.90000000e+01],
[  6.10000000e+01],
[  8.00000000e+01],
[  4.00000000e+01],
[  5.10000000e+01],
[  7.70000000e+01],
[  7.00000000e+01],
[  6.10000000e+01],
[  4.20000000e+01],
[  1.60000000e+01],
[  2.10000000e+01],
[  3.60000000e+01],
[  3.20000000e+01],
[  6.40000000e+01],
[  3.70000000e+01],
[  5.20000000e+01],
[  6.20000000e+01],
[  4.20000000e+01],
[  1.80000000e+01],
[  3.60000000e+01],
[  2.60000000e+01],
[  3.70000000e+01],
[  1.90000000e+01],
[  2.50000000e+01],
[  3.30000000e+01],
[  1.60000000e+01],
[  2.50000000e+01],
[  2.00000000e+01],
[  2.00000000e+01],
[  2.30000000e+01],
[  1.50000000e+01],
[  1.40000000e+01],
[  3.00000000e+00],
[  1.40000000e+01],
[  7.00000000e+00],
[  7.00000000e+00],
[  1.40000000e+01],
[  1.30000000e+01],
[  3.50000000e+01],
[  1.10000000e+01],
[  3.00000000e+01],
[  1.90000000e+01],
[  9.00000000e+00],
```

```
[  8.00000000e+00],
[  3.00000000e+00]], dtype=float32)
```

In [26]:
```
# normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
data = scaler.fit_transform(data)
```

In [27]:
```
dataset = data.reshape((116, 1))
```

In [28]:
```
# split into train and test sets
train_size = int(len(dataset) * 0.67)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]
print(len(train), len(test))
```

```
(77, 39)
```

In [29]:
```
# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1):
        dataX, dataY = [], []
        for i in range(len(dataset)-look_back-1):
                a = dataset[i:(i+look_back), 0]
                dataX.append(a)
                dataY.append(dataset[i + look_back, 0])
        return numpy.array(dataX), numpy.array(dataY)
```

In [30]:
```
# reshape into X=t and Y=t+1
look_back = 1
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
```

In [31]:
```
# reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
```

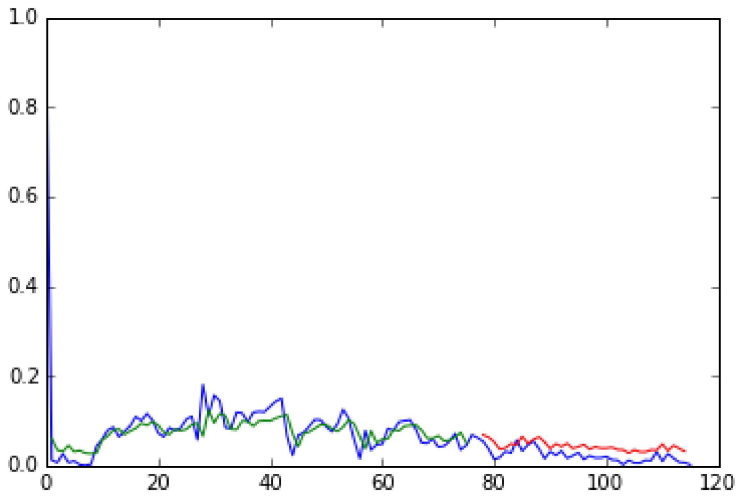
In [32]:
```
# create and fit the LSTM network
model = Sequential()
model.add(LSTM(4, input_dim=look_back))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, nb_epoch=100, batch_size=1, verbose=0)
```

Out[32]:
```
<keras.callbacks.History at 0x297d8358>
```

In [33]:
```
# Estimate model performance
trainScore = model.evaluate(trainX, trainY, verbose=0)
trainScore = math.sqrt(trainScore)
trainScore = scaler.inverse_transform(numpy.array([[trainScore]]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = model.evaluate(testX, testY, verbose=0)
testScore = math.sqrt(testScore)
testScore = scaler.inverse_transform(numpy.array([[testScore]]))
print('Test Score: %.2f RMSE' % (testScore))
```

```
Train Score: 32.15 RMSE
Test Score: 27.27 RMSE
```

In [34]:
```
# generate predictions for training
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)

# shift train predictions for plotting
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict

# shift test predictions for plotting
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-1, :] = testPre
dict

# plot baseline and predictions
plt.plot(dataset)
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```



In [ ]:

In [ ]: