Foreclosure prediction using machine learning

Multiple Linear Regression/GradientBoostingRegressor/Ridge Regression

Mika Breyfogle Dec 2020

GitHub: find my source code in here.

Problem Statement

The goal of the analysis is to help us understanding the relation between unemployment rate/ income/ foreclosure and how zip code are used to determine it. I hope that creating a model will help predict foreclosure in near future. I also create recommended zip code for the home seeker in his early thirties who likes to go to coffee shops and like outdoor sports.

Data acquisition and cleaning

1. City of los angels foreclosure data by zip code form 2015 to 2020
2. City of los angles unemployment data by zip 2020
3. Total unemployment per month from 1990 to 2020.
4. Unemployment LA County 1990
5. Income and Liability data 2018 by Zip code
6. Average Condo price by zip. 2015 ~ 2020
7. Use Foreclosure to find coffee shops and sports venues

Updating missing data using data allocations. Creating new data frame for data analysis.

Training machine learning algorithms

Multiple Linear Regression, gradient boosting regression and Ridge Regression.

Step1 : Exploratory Data Analysis (EDA)

Import the data and check to see what kind of data we are dealing with:

1)Import required libraries.

2)Import Data to the data frame.

3)Marge datas into one Dataframe.

3)See the head of the data to know how the data looks like, also some data need to clean.

4)Use describe function to see the percentiles's

	zip code	unemployment_rate	yearmonth	foreclosure	income	condprice	lat	lon
0	90013	32.353884	2015-01-01	1.0	96694.701625	500461.0	34.04469	-118.241
1	90021	25.984959	2015-01-01	0.0	111552.909449	513140.0	34.02941	-118.240
2	90059	12.254380	2015-01-01	44.0	32530.141714	328985.0	33.92693	-118.248
3	90014	12.099669	2015-01-01	1.0	80076.552268	566374.0	34.04306	-118.251
4	90002	11.925620	2015-01-01	63.0	31013.428093	326200.0	33.94907	-118.247

Describe data to find %.

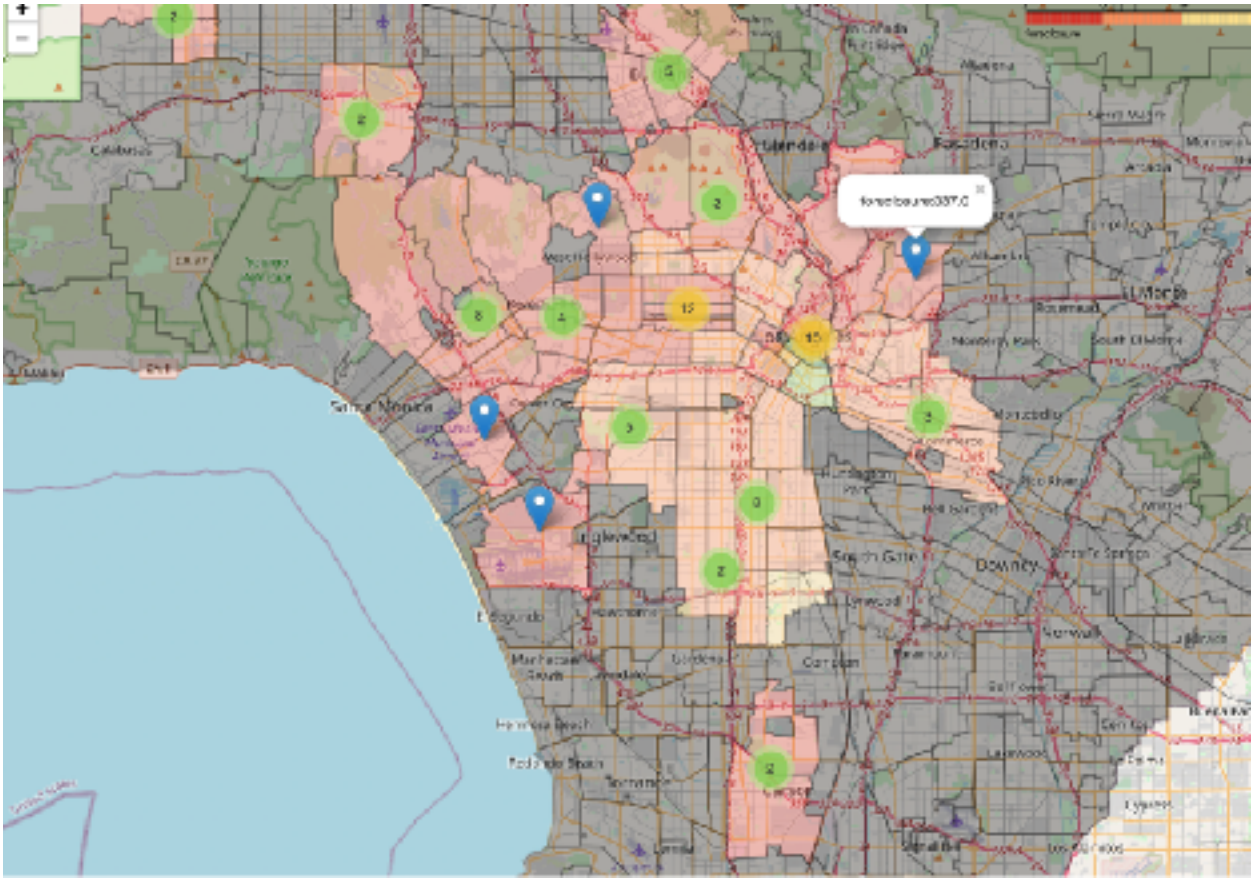
```
mydatas.describe()
```

	unemployment_rate	foreclosure	income	condprice	lat	lon
count	4380.000000	4380.000000	4380.000000	4.380000e+03	4380.000000	4380.000000
mean	4.564478	2.772831	115956.737334	6.048609e+05	34.055348	-118.319938
std	3.147048	7.961656	141966.969846	2.348765e+05	0.074496	0.093424
min	0.981818	0.000000	29493.563489	2.595070e+05	33.824414	-118.524402
25%	2.583684	0.000000	41543.924921	4.303970e+05	34.023995	-118.373334
50%	4.020967	1.000000	63404.085511	5.594650e+05	34.057308	-118.308035
75%	5.606466	2.000000	110858.131093	7.255750e+05	34.097449	-118.258872
max	32.353884	138.000000	721663.505345	1.809523e+06	34.220394	-118.150703

```
mydatas.head()
```

	zip code	unemployment_rate	yearmonth	foreclosure	income	condprice	lat	lon
0	90013	32.353884	2015-01-01	1.0	96694.701625	500461.0	34.044690	-118.241599
1	90021	25.984959	2015-01-01	0.0	111552.909449	513140.0	34.029411	-118.240375
2	90059	12.254380	2015-01-01	44.0	32530.141714	328985.0	33.926931	-118.248179
3	90014	12.099669	2015-01-01	1.0	80076.552268	566374.0	34.043061	-118.251756
4	90002	11.925620	2015-01-01	63.0	31013.428093	326200.0	33.949079	-118.247877

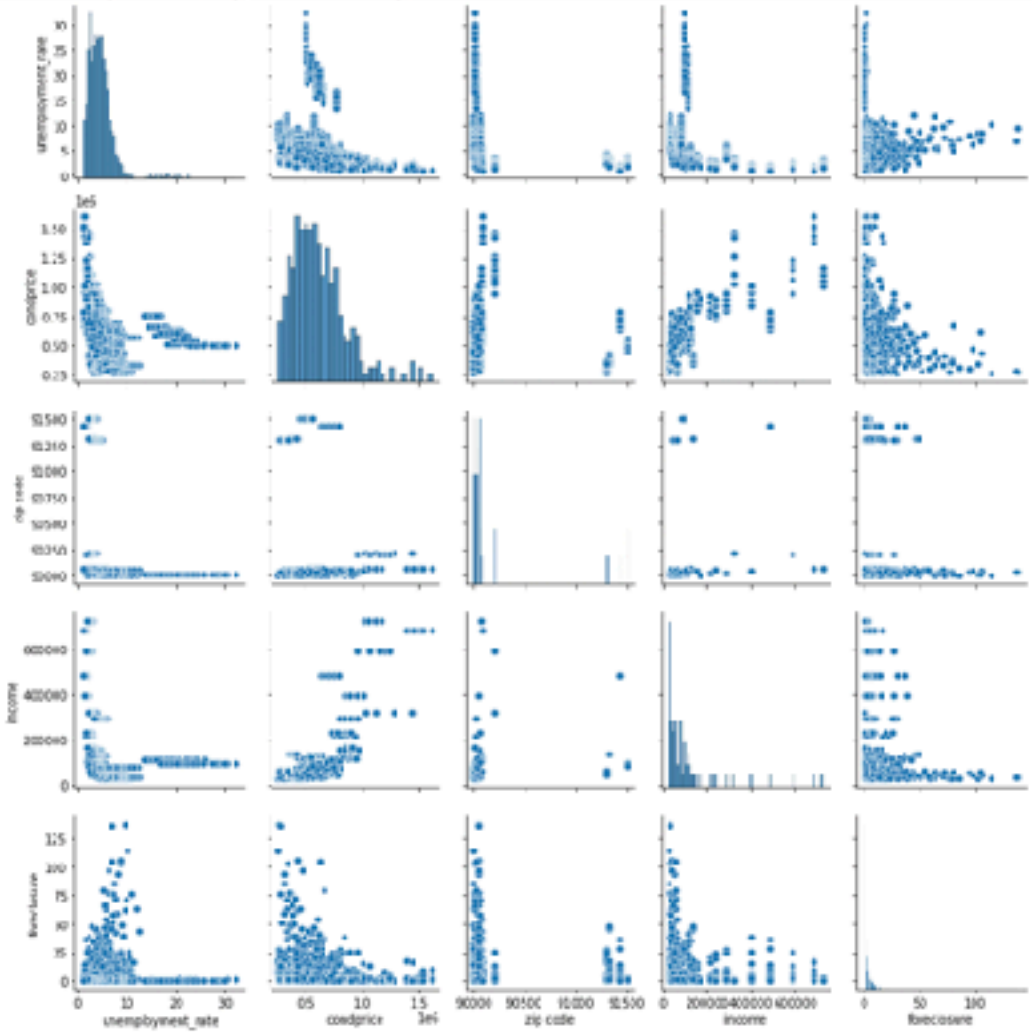
Now I created map to visualize the location of the foreclosure/unemployment rate based on latitude and longitude. I am going to see the common location and how the foreclosure are placed.



Let's plot couple of feature to get a better feel of the data.

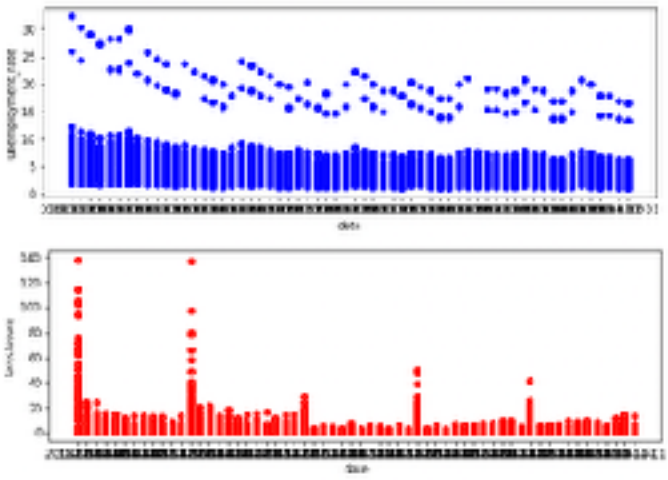
```
# calculate and show pairplot
z=mydatas[['unemployment_rate','condprice','zip code','income','foreclosure']]
sns.pairplot(z, size=2.5)
plt.tight_layout()
```

```
/opt/anaconda3/lib/python3.8/site-packages/seaborn/axisgrid.py:1912: UserWarning: The `size` parameter has been re
  warnings.warn(msg, UserWarning)
```



```
#simple regression zip vs price
msk=np.random.rand(len(mydatas))< 0.8
train=mydatas[msk]
test=mydatas[~msk]
fig = plt.figure(figsize=(10,7))
fig.add_subplot(2,1,1)
plt.scatter(train['yearmonth'],train.unemployment_rate,color='blue')
plt.xlabel("date")
plt.ylabel("unemployment_rate")
plt.show
#simple regression zip vs price
fig = plt.figure(figsize=(10,7))
fig.add_subplot(2,1,1)
plt.scatter(train['yearmonth'],train.foreclosure,color='red')
plt.xlabel("date")
plt.ylabel("foreclosure")
plt.show
```

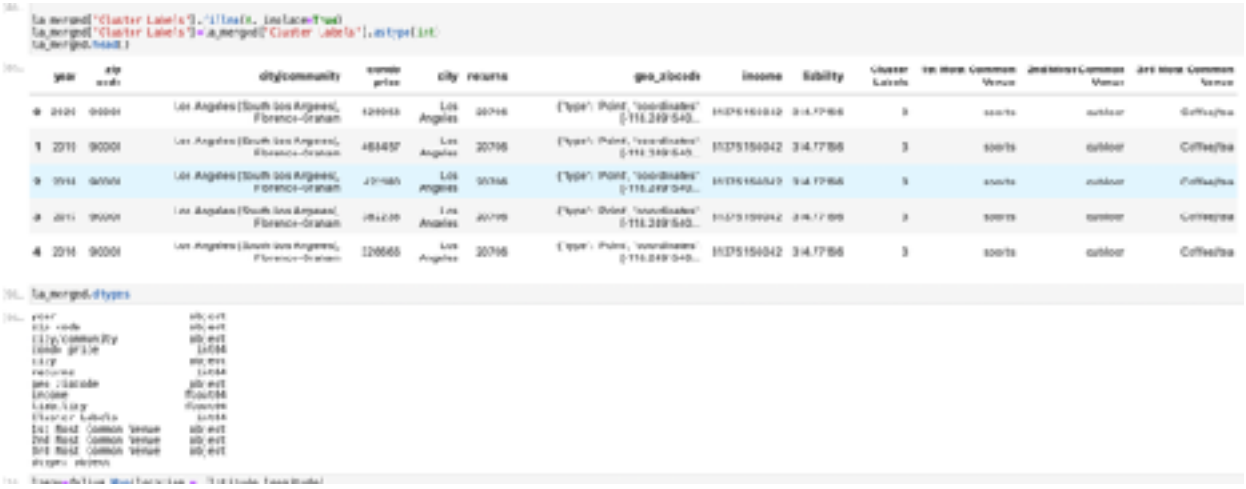
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



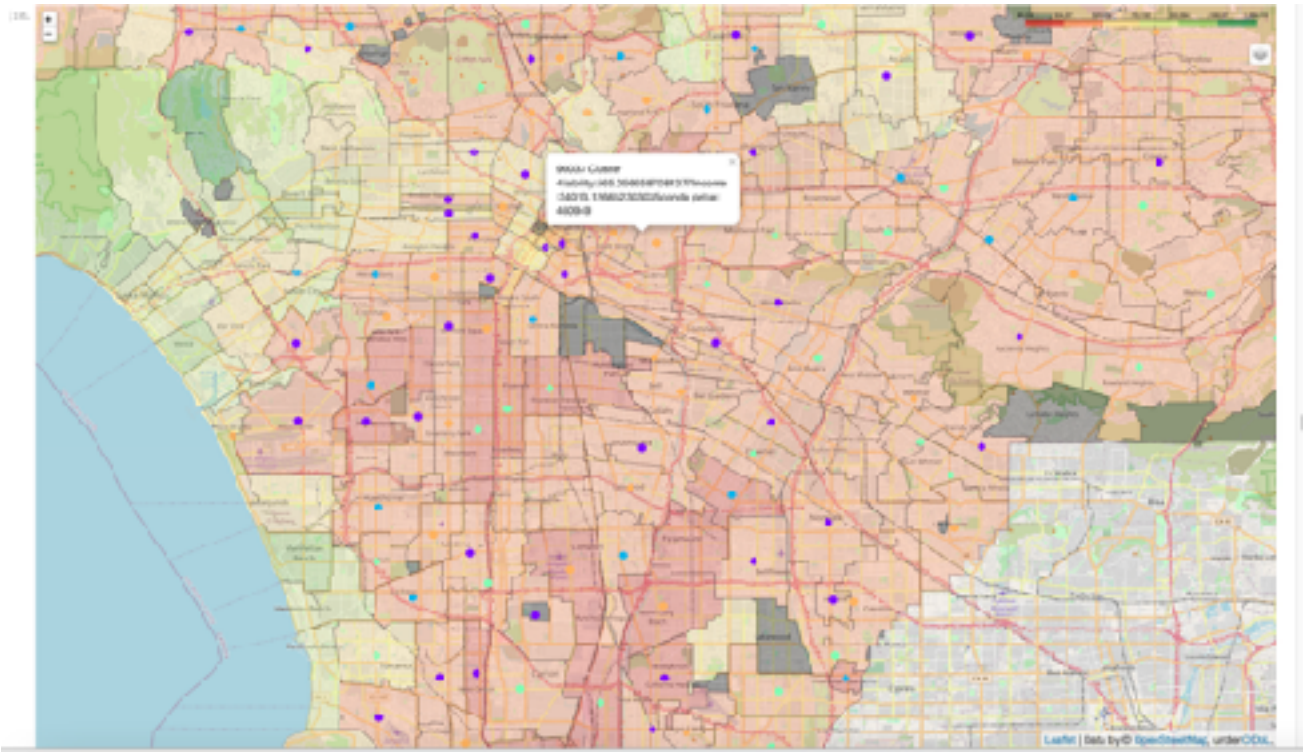
It looks like foreclosure and unemployment has some relationship.

Now I will find Coffee and sports venues using foursquare per each zip code

Here is a sample of code and Dataframe which I created.



Now I created map to visualize the location of the venues based on latitude and longitude. I am going to see the common location and how the venues are placed. I cluster are labeled.



Now that I am familiar with all these representation and I can tell my own story .

Now I create a model to which would predict the foreclosure based upon the other factors such as income , zip code. I use train data and test data , train data to train the machine and test data to see if it has learnt the data well or not.

Step 2: Dataset Preparation (Splitting and Scaling)

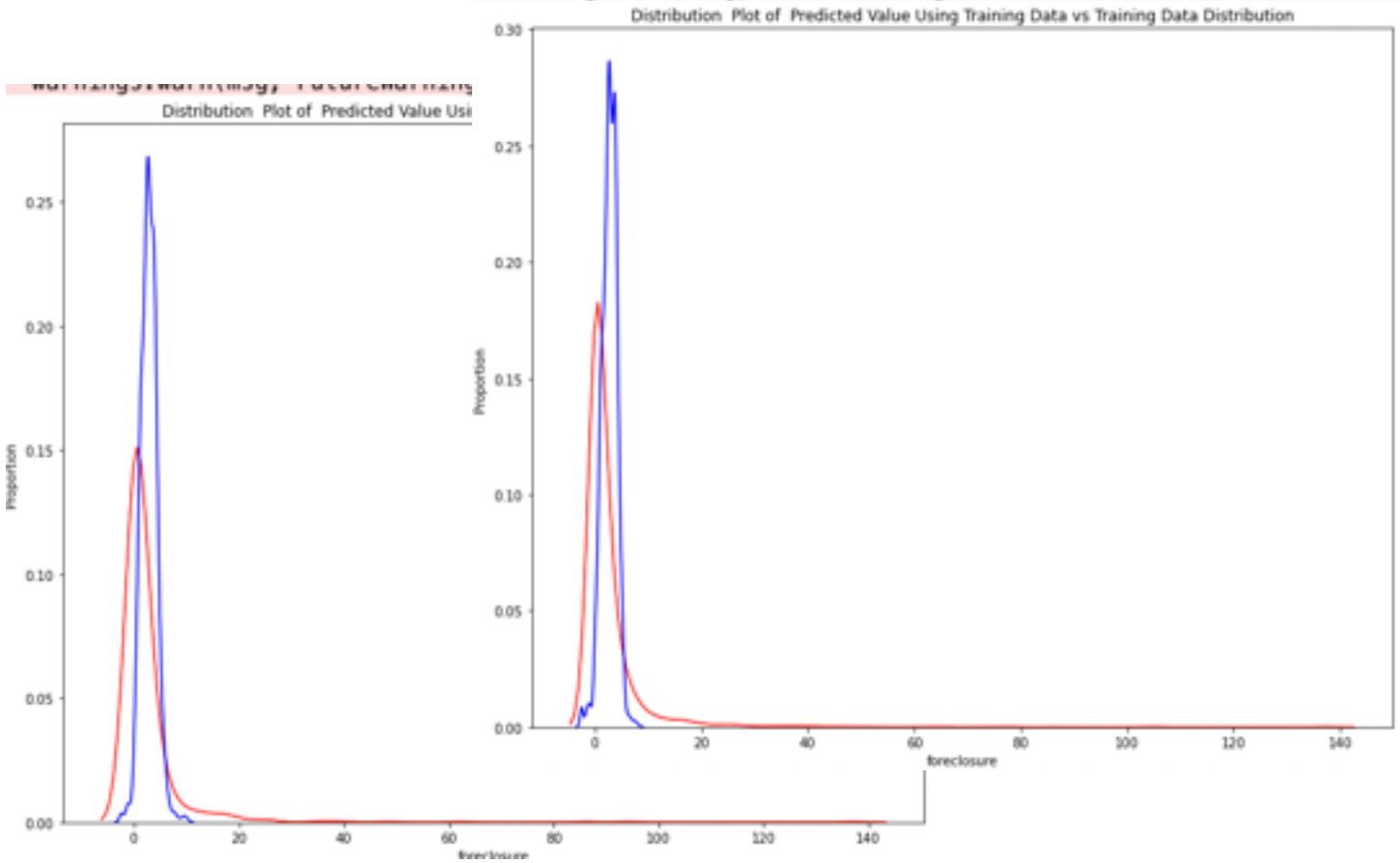
Data is divided into the Train set and Test set.

Step 3: Model creation

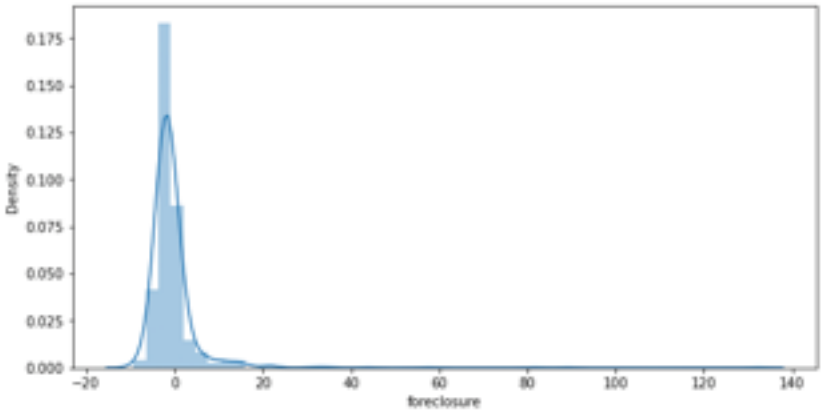
Here is a code for Multiple Liner model coding

```
Type to enter text#Create mulitiple regression with train & test
mydataz=mydatas
lr1 = LinearRegression()
label=mydataz['foreclosure']
train1 = mydataz.drop(['foreclosure','lat','lon','yearmonth','zip code'],axis=1)
#Creating train / test set
x_train, x_test, y_train, y_test = train_test_split(train1, label, test_size=0.20, random_state=4)

lr1.fit(x_train, y_train)
print("number of test samples :", x_test.shape[0])
print("number of training samples:",x_train.shape[0])
yhat_train = lr1.predict(x_train)
yhat_test = lr1.predict(x_test)

Title = 'Distribution  Plot of  Predicted Value Using Training Data vs Training Data Distribution'
DistributionPlot(y_train, yhat_train, "Actual Values (Train)", "Predicted Values (Train)", Title)
Title='Distribution  Plot of  Predicted Value Using Test Data vs Data Distribution of Test Data'
DistributionPlot(y_test,yhat_test,"Actual Values (Test)","Predicted Values (Test)",Title)
    # visualizing residuals
fig = plt.figure(figsize=(10,5))
residuals = (y_test- yhat_test)
sns.distplot(residuals)
print('The R-square Linear is: ', lr1.score(train1, mydataz['foreclosure']))
# evaluate the performance of the algorithm (MAE - MSE - RMSE)
print('MAE:', metrics.mean_absolute_error(y_test, yhat_test))
print('MSE:', metrics.mean_squared_error(y_test, yhat_test))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, yhat_test)))
print('VarScore:',metrics.explained_variance_score(y_test,yhat_test))
```

```
warnings.warn(msg, FutureWarning)
```




```
The R-square Linear is:  0.0342734584356702
MAE: 3.2251486345606897
MSE: 60.809271268367986
RMSE: 7.7980299607252075
VarScore: 0.03576978059669422
```



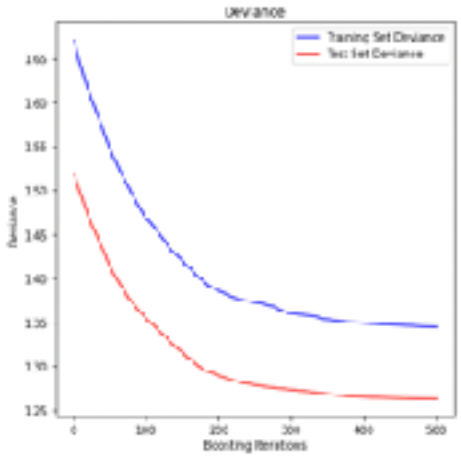
As you can see both training and test is equally bad. R-square is .03. Error is high too.

Let train another model. GradientBoostingRegresson

```
params1 = {'n_estimators': 500,
        'max_depth': 4,
         'min_samples_split': 4,
         'learning_rate': 0.01,
         'loss': 'quantile',
        'alpha': .70}
label=mydataz['foreclosure']
train1 = mydataz.drop(['foreclosure','lat','lon','yearmonth','zip code'],axis=1)
x_train, x_test, y_train, y_test = train_test_split(train1, label, test_size=0.20, random_state=5)
clf = ensemble.GradientBoostingRegressor(**params1)
clf.fit(x_train,y_train)
clf.score(x_test,y_test)
yhat_test2=clf.predict(x_test)
mse = mean_squared_error(y_test, clf.predict(x_test))
print('The GRADIENT BOOST R-square is: ', clf.score(x_train,y_train))
print('MAE:', metrics.mean_absolute_error(y_test, yhat_test2))
print('MSE:', metrics.mean_squared_error(y_test, yhat_test2))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, yhat_test2)))
print('VarScore:',metrics.explained_variance_score(y_test,yhat_test2))

test_score = np.zeros((params['n_estimators'],), dtype=np.float64)
for i, y_pred in enumerate(clf.staged_predict(x_test)):
    test_score[i] = clf.loss_(y_test, y_pred)

fig = plt.figure(figsize=(6, 6))
plt.subplot(1, 1, 1)
plt.title('Deviance')
plt.plot(np.arange(params['n_estimators']) + 1, clf.train_score_, 'b-',
         label='Training Set Deviance')
plt.plot(np.arange(params['n_estimators']) + 1, test_score, 'r-',
         label='Test Set Deviance')
plt.legend(loc='upper right')
plt.xlabel('Boosting Iterations')
plt.ylabel('Deviance')
fig.tight_layout()
plt.show()
```

Here is a prediction looks like

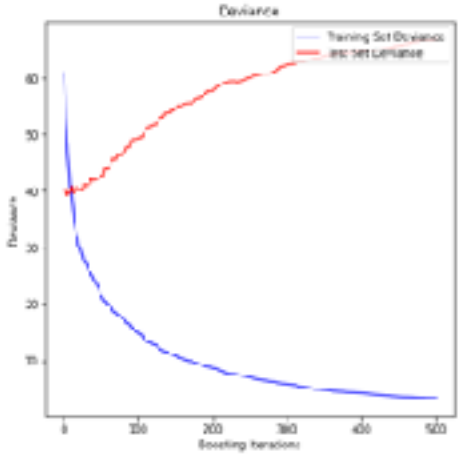
```
The GRADIENT BOOST R-square is:  0.16220846717771853
MAE: 2.2989417112552912
MSE: 54.004390823332931
RMSE: 7.348767993570957
VarScore: 0.14711472221550078
```



Still R-square is too low

I will tray another parameters and see what happen

```
The GRADIENT BOOST R-square is:  0.9512916654565363
MAE: 3.3228306510822516
MSE: 66.730055627325691
RMSE: 8.168846652210439
VarScore: -0.62327857983106724
```



R-square is near 1.00. this is good model, however MSE is still high

Now I am going to create Redge Model.

First I will find best alpha using GridSearch

Here is a code

```
#Ridge regression
pr=PolynomialFeatures(degree=2)
x_train_pr=pr.fit_transform(x_train)
x_test_pr=pr.fit_transform(x_test)
```

```
from sklearn.linear_model import Ridge
```

```
#para2= [{'alpha': [0.001,0.1,1, 10, 100, 1000, 10000, 100000, 100000],'normalize':[True,False]}]
para2= [{'alpha': [0.001,0.1,1, 10, 100, 1000, 6000,10000, 50000,100000, 100000]}]
RR=Ridge()

Grid1 = GridSearchCV(RR, para2,cv=4)
```

```
Grid1.fit(train1, label)
```

```
GridSearchCV(cv=4, estimator=Ridge(),
             param_grid=[{'alpha': [0.001, 0.1, 1, 10, 100, 1000, 6000, 10000,
                                    50000, 100000, 100000]}])
```

```
BestRR=Grid1.best_estimator_
BestRR
```

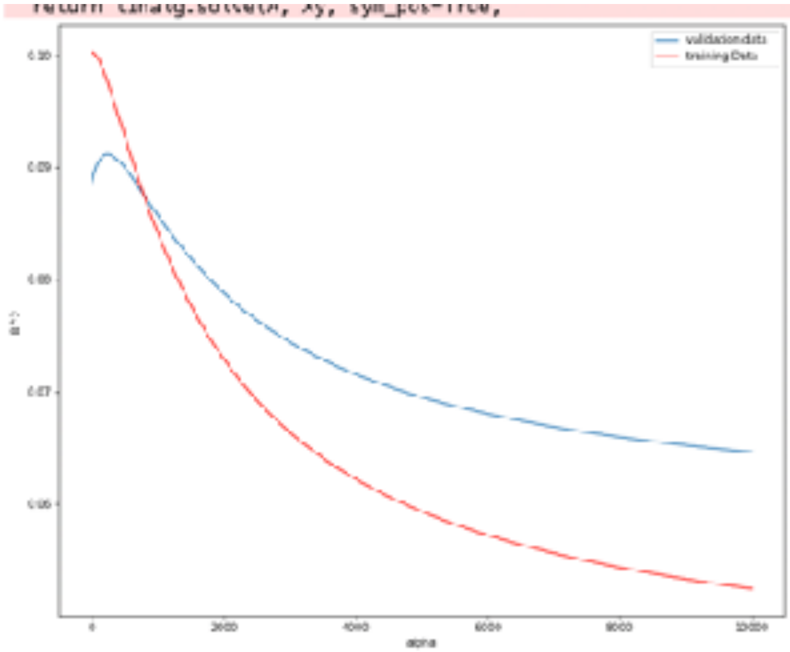
```
Ridge(alpha=10000)
```

Best Alpha is 10000, let create model

```
RigeModel=Ridge(alpha=10000)
RigeModel.fit(x_train_pr, y_train)
yhat2 = RigeModel.predict(x_test_pr)
Rsqu_test = []
Rsqu_train = []
dummy1 = []
Alpha = 10 * np.array(range(0,1000))
for alpha in Alpha:
    RigeModel = Ridge(alpha=alpha)
    RigeModel.fit(x_train_pr, y_train)
    Rsqu_test.append(RigeModel.score(x_test_pr, y_test))
    Rsqu_train.append(RigeModel.score(x_train_pr, y_train))
width = 12
height = 10
plt.figure(figsize=(width, height))

plt.plot(Alpha,Rsqu_test, label='validation data  ')
plt.plot(Alpha,Rsqu_train, 'r', label='training Data ')
plt.xlabel('alpha')
plt.ylabel('R^2')
plt.legend()
print ('R-score train:',RidgeModel.score(x_train_pr,y_train))
print ('R-score test:',RidgeModel.score(x_test_pr,y_test))
print('The GRADIENT BOOST R-square is: ', RidgeModel.score(x_train_pr,y_train))
print('MAE:', metrics.mean_absolute_error(y_test, yhat2))
print('MSE:', metrics.mean_squared_error(y_test, yhat2))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, yhat2)))
print('VarScore:',metrics.explained_variance_score(y_test,yhat2))
```

Here is a prediction look like.

```
    return linalg.solve(A, Xy, sym_pos=True,
```




```
R-score train: 0.10027274079088324
R-score test: 0.08889618440575006
The GRADIENT BOOST R-square is:  0.10027274079088344
MAE: 3.109488590071713
MSE: 58.87722462607468
RMSE: 7.673149589710517
VarScore: 0.06556781197369133
/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_ridge.py:147: LinAlgWarning: Ill-condit
```

R-score is still low.

I will evaluate above models using actual number.

```
#check result

house1 =train1.loc[(mydatas['zip code']== 30023 ) & ( mydatas['unemployment_rate'] > 0.0) & ( mydatas['unemployment_rate'] < 4.5)]
house2=pr.fit_transform(house1)
print('Liner :',lr1.predict(house1).mean())
print('Gradiant :',clf.predict(house1).mean())
print('Ridge :',RigeModel.predict(house2).mean())
house3 =mydatas.loc[(mydatas["zip code"]== 30023 ) & ( mydatas['unemployment_rate'] > 0.0) & ( mydatas['unemployment_rate'] < 4.5)]
#print (house2)
print(house3["foreclosure"].mean())
```

```
Liner : 2.9615370964027007
Gradiant : 0.4115123522079552
Ridge : 2.897012408597 12
0.5000000000000000
```

```
#check
house1 =train1.loc[(mydatas['zip code']== 30065 ) & ( mydatas['unemployment_rate'] > 0.0) & ( mydatas['unemployment_rate'] < 4.5)]
house2=pr.fit_transform(house1)
print('Liner :',lr1.predict(house1).mean())
print('Gradiant :',clf.predict(house1).mean())
print('Ridge :',RigeModel.predict(house2).mean())
house3 =mydatas.loc[(mydatas["zip code"]== 30008 ) & ( mydatas['unemployment_rate'] > 0.0) & ( mydatas['unemployment_rate'] < 4.5)]
print(house3["foreclosure"].mean())
```

```
Liner : 3.7013017100034445
Gradiant : 2.6915917867110205
Ridge : 2.5137408849062927
2.0555555555555554
```

```
house1 =train1.loc[(mydatas['zip code']== 30008 ) & ( mydatas['unemployment_rate'] > 0.0) & ( mydatas['unemployment_rate'] > 4.5)]
house2=pr.fit_transform(house1)
print('Liner :',lr1.predict(house1).mean())
print('Gradiant :',clf.predict(house1).mean())
print('Ridge :',RigeModel.predict(house2).mean())
house3 =mydatas.loc[(mydatas["zip code"]== 30008) & ( mydatas['unemployment_rate'] > 0.0) & ( mydatas['unemployment_rate'] > 4.5)]
print(house3["foreclosure"].mean())
```

```
Liner : 3.5589196645802585
Gradiant : 6.5144672559780082
Ridge : 3.8627920411 1156
7.714285714285714
```

Conclusion:

Out of three model GradientBoostingRegressor create best result.

Most of bank foreclosed house on beginning of year. Maybe because of holiday back log.

When unemployment rate is real high (example 15% or more), foreclosure will not impact unemployment_rate. Probably, most of unemployed person do not own the house.