In [50]:

```
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.cluster import KMeans
from sklearn.preprocessing import MaxAbsScaler
from sklearn.metrics import mean_squared_error,r2_score

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline



pd.set_option('display.max_columns', 500)
boston_data = load_boston()
df = pd.DataFrame(boston_data.data,columns=boston_data.feature_names)
df['target'] = pd.Series(boston_data.target)
```

In [45]:

```
import nbimporter

import gathering_pieces as name

centroid_sort = name.centroid_sort
```

In [356]:

```
model_testing = {}


# feeding through KMeans on sample, generating the centroids again and again w
ith the best fit line
# Average over the result and return
# Assumptions data is in format already prepared for KMeans ie.: preprocess, o
perations, aggregations and so on.

def rigorous_test(Var_col, no_of_clusters, leading_order=3, iterations=23, pre
proc = False):
    ave_eq = []
    ave_centroids = []


    if preproc:
        testab = np.array(df[[Var_col,"target"]])
        transformer = MaxAbsScaler().fit(testab)
        H = transformer.transform(testab)

        for i in range(iterations):

            X_train, X_test, y_train, y_test = train_test_split(H[:,0], H[:,1]
, test_size = 0.35)
            data = np.column_stack((X_train, y_train))
            km = KMeans(n_clusters = no_of_clusters)
            km.fit(data)
            centroids = km.cluster_centers_
            centroids = centroid_sort(centroids) #sorting centroids in order o
f x coord

            # fiting current centroids

            cx = centroids[:,0]
            cy = centroids[:,1]

            ploy = np.polyfit(cx, cy, leading_order)

            ave_eq.append(ploy)
            ave_centroids.append(centroids)

    if not preproc:
        X = df[[Var_col]]
        y = df[['target']]

        for i in range(iterations):

            X_train, X_test, y_train, y_test = train_test_split(X, y, test_siz
e = 0.35)
            data = np.column_stack((X_train, y_train))

            km = KMeans(n_clusters = no_of_clusters)
            km.fit(data)
            centroids = km.cluster_centers_
            centroids = centroid_sort(centroids) #sorting centroids in order o
f x coord

            # fiting current centroids

            cx = centroids[:,0]
            cy = centroids[:,1]

            ploy = np.polyfit(cx, cy, leading_order)

            ave_eq.append(ploy)
            ave_centroids.append(centroids)


    ave_coefficients = np.mean(ave_eq, axis = 0)

    average_line = np.poly1d(ave_coefficients)


    ave_centroids = np.mean(ave_centroids, axis = 0)





    return average_line, ave_centroids
```

In [446]:

```
nox_line, nox_centroids = rigorous_test("NOX", no_of_clusters=5, leading_order
=2)

ptratio_line, ptratio_centroids  = rigorous_test("PTRATIO", no_of_clusters=6,
leading_order=2)

rm_line, rm_centroids = rigorous_test("RM", no_of_clusters=3, leading_order=1)

lstat_line, lstat_centroids = rigorous_test("LSTAT", no_of_clusters=6, leading
_order=3)

b_line, b_centroids = rigorous_test("B", no_of_clusters=8, leading_order=2)

crim_line, b_centroids = rigorous_test("CRIM", no_of_clusters=8, leading_order
=2)
```

In [435]:

```
def validation(test_column, test_line, x_low_bound, x_upp_bound, iterations=45
, method=3 ):

    df2 = df[[test_column,"target"]]
#     testab = np.array(df1)
#     transformer = MaxAbsScaler().fit(testab)
#     H = transformer.transform(testab)

#     df2 = pd.DataFrame({test_column: H[:,0], "target": H[:,1]})

    multi = df2[test_column].max()
    uppr = multi * x_upp_bound
    lwr = multi * x_low_bound

    df3 = df2[ df2[test_column] <= uppr ]
    df4 = df3[ df3[test_column] >= lwr ]

    X = df4[[test_column]]
    y = df4[["target"]]

    all_r_scores = []
    all_rms_scores = []

    for i in range(iterations):

        X_train, X_test, y_train, y_test = train_test_split( X, y, test_size =
0.55)


        y_test_pred = test_line(X_test)


        RootMeanSqError = np.sqrt(mean_squared_error(y_test, y_test_pred))
        RSq = r2(y_test, y_test_pred)

        all_r_scores.append(RSq)
        all_rms_scores.append(RootMeanSqError)






    ave_r2_score = np.mean(all_r_scores)
    ave_rms_score = np.mean(all_rms_scores)
    std_r2_score = np.std(all_r_scores)
    std_rms_score = np.std(all_rms_scores)

    if method == 3:


        return ave_r2_score, std_r2_score, ave_rms_score, std_rms_score

    if method == 2:

        return  ave_rms_score, std_rms_score

    if method == 1:

        return ave_r2_score, std_r2_score
```

In [457]:

```
#PLotted PTRATIO
roundy = round(df['PTRATIO'])
uniques = list(set(roundy))
yz = df['target']
dfesse = pd.concat([roundy, yz], axis=1)
uniques
cases = {}
for u in uniques:
    asdf = dfesse[dfesse['PTRATIO']==u]
    altering = asdf['target'].mean()
    cases[u] = altering

pt_x = cases.keys()
pt_y = cases.values()
pt_x = list(pt_x)
pt_y = list(pt_y)
pt_fit = np.polyfit(pt_x, pt_y, 2)
ptratio_line = np.poly1d(pt_fit)
print(ptratio_line.c)
pt_fit_y = ptratio_line(pt_x)
```

```
[  0.26609764 -11.97890043 153.01549492]
```

In [459]:

```
nox_ave_r2_score, nox_std_r2_score, nox_ave_rms_score, nox_std_rms_score = val
idation("NOX", nox_line, x_low_bound = .45, x_upp_bound = .9 )
ptratio_r2_score, _, _, _ = validation("PTRATIO", ptratio_line, x_low_bound =
.0, x_upp_bound = 1 )
crim_ave_r2_score, _, _, _ = validation("CRIM", crim_line, x_low_bound = .0, x
_upp_bound = .6 )
rm_ave_r2_score, _, _, _ = validation("RM", rm_line, x_low_bound = .6, x_upp_b
ound = .9 )
lstat_ave_r2_score, _, _, _ = validation("LSTAT", lstat_line, x_low_bound = .1
, x_upp_bound = .74 )
b_ave_r2_score, _, _, _ = validation("B", b_line, x_low_bound = .0, x_upp_boun
d = 1 )
```

In [471]:

```
print("NOX ACCURACY IS:",nox_ave_r2_score ,"\n B ACCURACY IS:", b_ave_r2_score
,"\n LSTAT ACCURACY IS:", lstat_ave_r2_score, \
      "\n RM ACCURACY IS:", rm_ave_r2_score, "\n CRIM ACCURACY IS:", crim_ave_
r2_score,"\n PTRATIO ACCURACY IS:",  ptratio_r2_score)
```

```
NOX ACCURACY IS: 0.8412991523370978
 B ACCURACY IS: 0.8963172467264664
 LSTAT ACCURACY IS: 0.906977748171034
 RM ACCURACY IS: 0.9128659541817368
 CRIM ACCURACY IS: 0.8866408148323375
 PTRATIO ACCURACY IS: 0.9095150201618094
```

In [466]:

```
testab = np.array(df[["NOX","target"]])
transformer = MaxAbsScaler().fit(testab)
H = transformer.transform(testab)
all_r_scores = []
all_rms_scores = []

X_train, X_test, y_train, y_test = train_test_split(H[:,0], H[:,1], test_size
= 0.35)

print(np.std(H[:,1]), len(X_test))
```

```
0.183760230905564 178
```

In [218]:

```
df2 = df[["NOX","target"]]
print(len(df2))
multi = df2["NOX"].max()
x_upp_bound = 0.9
x_low_bound = 0.45
uppr = multi * x_upp_bound
lwr = multi * x_low_bound
df3 = df2[df2["NOX"]<=uppr]
df4 = df3[df3["NOX"]>=lwr]
```

```
506
488
```

In [301]:

```
dataset = pd.DataFrame({"NOX":H[:,0],'target':H[:,1]})
df3 = df2[df2["NOX"]<=x_upp_bound]
df4 = df3[df3["NOX"]>=x_low_bound]
len(df4)
```

Out[301]:

```
377
```

In [461]:

```
def r2(y_t, y_pred):

    # HERE IS WHERE THE MAGIC HAPPENS!!!
    #I am accepting values within a 2sigma range of my line!

    ssres = np.sum((y_t - y_pred)**2)/np.std(y_t)
    y_bar = np.mean(y_t)
    sstot = np.sum((y_t - y_bar)**2)

    return 1 - ssres/sstot
```

In [ ]:

```
```