1. Business Understanding

This initial phase focuses on understanding the project objectives and requirements from a business perspective, and then converting this knowledge into a data mining problem definition, and a preliminary plan designed to achieve the objectives.

In this situation let's pretend we are a real estate agency in Boston MA and we are interested in purchasing some houses. We would like to know which houses are under value to help us narrow down the list and put in an accurate bid on a house.

Objective: Identify what makes a property valuable? What is a fair price for a house?

Dataset : Boston

Goal : Predict medv column in Test Dataset!

1. **crim :** per capita crime rate by town.
2. **zn :** proportion of residential land zoned for lots over 25,000 sq.ft.
3. **indus :** proportion of non-retail business acres per town.
4. **chas :** Charles River dummy variable (= 1 if tract bounds river; 0 otherwise).
5. **nox :** nitrogen oxides concentration (parts per 10 million).
6. **rm :** average number of rooms per dwelling.
7. **age :** proportion of owner-occupied units built prior to 1940.
8. **dis :** weighted mean of distances to five Boston employment centres.
9. **rad :** index of accessibility to radial highways.
10. **tax :** full-value property-tax rate per $10,000.
11. **ptratio :** pupil-teacher ratio by town.
12. **black :** 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town.
13. **lstat :** lower status of the population (percent).
14. **medv :** median value of owner-occupied homes in $1000s.

Load Library

In [1]:

```python
#import libraries for data handling
import os
import pandas as pd
import numpy as np

#import for visualization
import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline

#import for Linear regression
from sklearn.linear_model import LinearRegression
```

Load Data into Pandas Dataframe

In [2]:
```
#Get Working Directory
cwd = os.getcwd()
cwd

# load the dataset
file = cwd+'/train.xlsx'
basetable1 = pd.read_excel(file)
```

In [3]:
```
# peek preview into the data
basetable1.head(6)
```

Out[3]:

	ID	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat
0	1	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98
1	2	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14
2	4	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94
3	5	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33
4	7	0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311	15.2	395.60	12.43
5	11	0.22489	12.5	7.87	0	0.524	6.377	94.3	6.3467	5	311	15.2	392.52	20.45

Here we see first 5 rows. Data is loaded Successfully!

2. Data Understanding (EDA)

Print a concise summary of a DataFrame.

This method prints information about a DataFrame including the index dtype and column dtypes, non-null values and memory usage.

In [4]:
```
# Information on the Dataframe
print("\n\n", basetable1.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 333 entries, 0 to 332
Data columns (total 15 columns):
ID         333 non-null int64
crim       333 non-null float64
zn         333 non-null float64
indus      333 non-null float64
chas       333 non-null int64
nox        333 non-null float64
rm         333 non-null float64
age        333 non-null float64
dis        333 non-null float64
rad        333 non-null int64
tax        333 non-null int64
ptratio    333 non-null float64
black      333 non-null float64
lstat      333 non-null float64
medv       333 non-null float64
dtypes: float64(11), int64(4)
memory usage: 39.1 KB


 None
```

In [5]:
```
#how big is the data?
print(basetable1.size)
```

```
4995
```

Understand Correlation between Dependednt (medv) and other Features

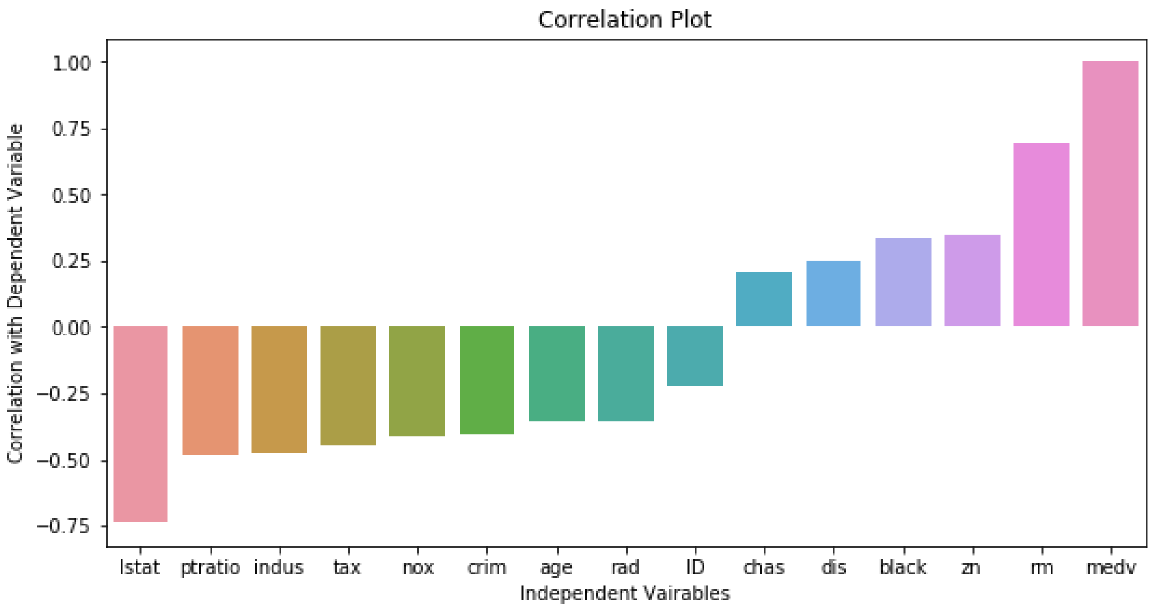
In [6]:
```
corr = basetable1.corr().tail(1)
corr.sort_values(by='medv',axis=1)
```

Out[6]:

	lstat	ptratio	indus	tax	nox	crim	age	rad
medv	-0.7386	-0.481376	-0.473932	-0.448078	-0.413054	-0.407454	-0.358888	-0.352251	-0.2

In [7]:

```
fig, ax = plt.subplots(figsize=(10,5))

plt.title("Correlation Plot")
plt.xlabel("Independent Vairables")
plt.ylabel("Correlation with Dependent Variable")

ax = sns.barplot(data=corr.sort_values(by='medv',axis=1))
```

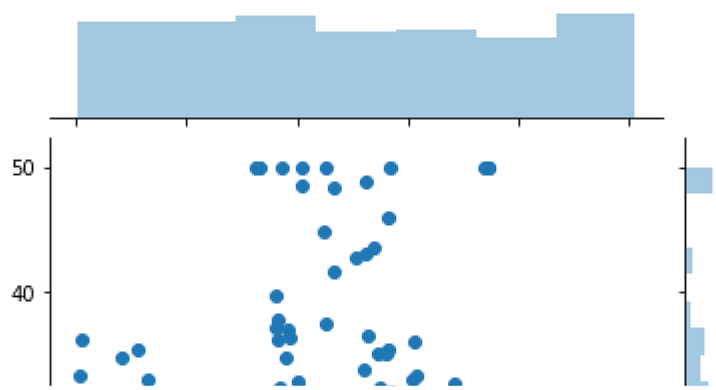


Correlation Based Inferences

1. ID Column though has Negative correlation but actually does not have any value to the Data. Eliminating at at later stage
2. rm : +vely MOST Impacting: average number of rooms per dwelling has High Correlation with medv (Dependent Variable)
3. lstat : -vely MOST Impacting: lower status of the population (percent) has High Correlation with medv (Dependent Variable)

In [8]:
```python
for index, columns in enumerate(basetable1.columns):
    svm = sns.jointplot(basetable1[basetable1.columns[index]],basetable1.medv
```

```
C:\Users\kvibhaas\AppData\Local\Continuum\anaconda3\lib\site-packages\sci
py\stats\stats.py:1713: FutureWarning: Using a non-tuple sequence for mul
tidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `a
rr[seq]`. In the future this will be interpreted as an array index, `arr
[np.array(seq)]`, which will result either in an error or a different res
ult.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

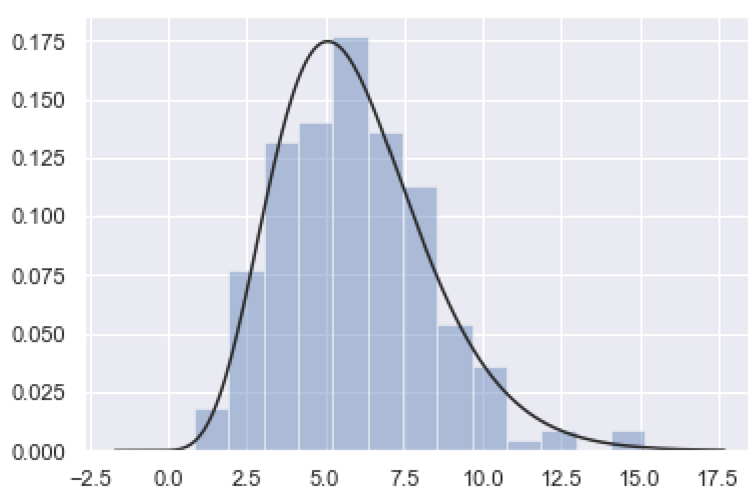


In [9]:
```python
# Set the palette to the "pastel" default palette:
sns.set_palette("pastel")
#Seaborn has six variations of its default color palette:: deep, muted, paste
```

In [10]:
```python
from scipy import stats
#import stats for distribution graph
```

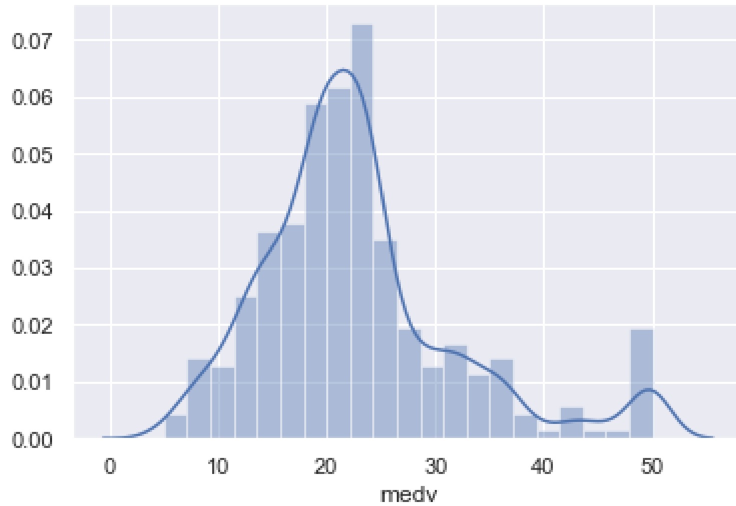
In [11]:
```python
sns.set(color_codes=True)
# set colour true
```

In [12]:
```python
x = np.random.gamma(6, size=200)
sns.distplot(x, kde=False, fit=stats.gamma);
```



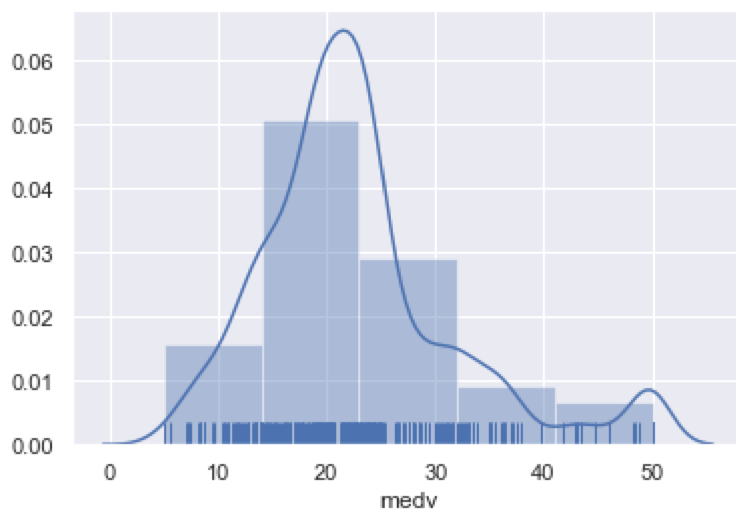
In [13]:
```
sns.distplot(basetable1.medv)
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x2bb7a2f10b8>



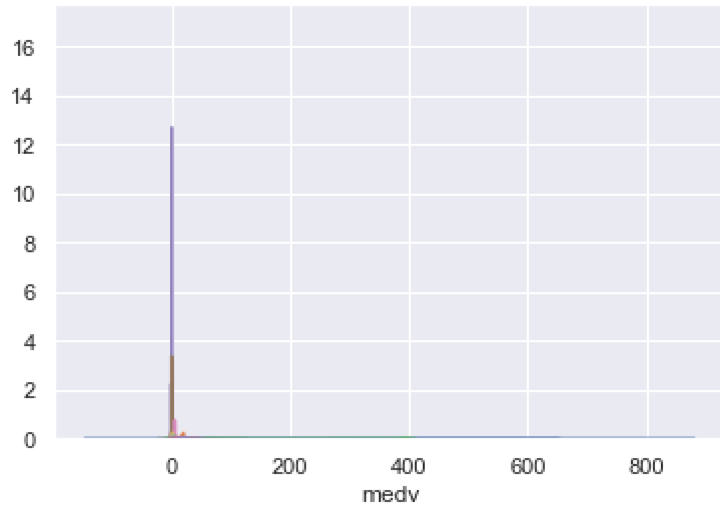
In [14]:
```
sns.distplot(basetable1.medv, bins = 5, hist = True, rug = True)
```

Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x2bb7a3780b8>



In [15]:

```
for index, columns in enumerate(basetable1.columns):
    sns.distplot(basetable1[basetable1.columns[index]])
    #svm = sns.jointplot(basetable1[basetable1.columns[index]],basetable1.mec
```



In [16]:

```
basetable1.columns
```

Out[16]:

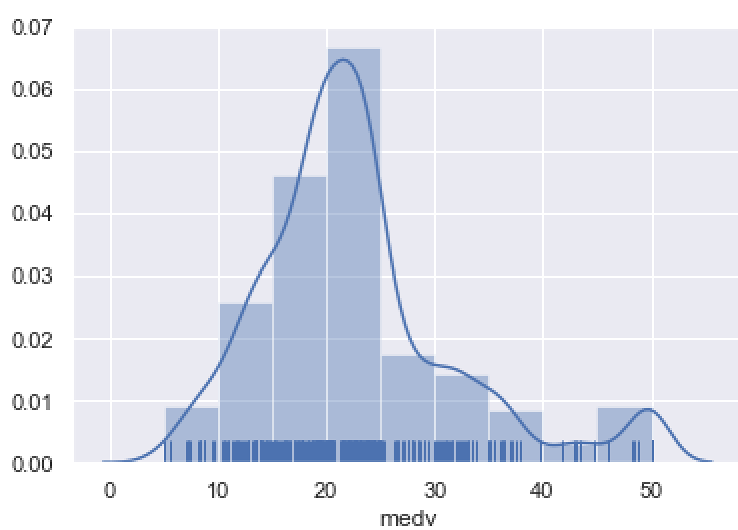
```
Index(['ID', 'crim', 'zn', 'indus', 'chas', 'nox', 'rm', 'age', 'dis', 'ra
d',
       'tax', 'ptratio', 'black', 'lstat', 'medv'],
      dtype='object')
```

In [17]:
```
# Python code to demonstrate the working of
# log(a,Base)

import math
# Printing the log base 5 of 14
print ("Logarithm base 10 of 14 is : ", end="")
print (math.log10(14))

#length of train dataset
len(basetable1)

#log of length
print (math.log10(len(basetable1)))

# Struge formula
print(1 + 3.322*(math.log10(len(basetable1))))

# round off the bin size
bin_size = round((1 + 3.322*(math.log10(len(basetable1)))))
```

```
Logarithm base 10 of 14 is : 1.146128035678238
2.5224442335063197
9.379559743707995
```

In [18]:
```
sns.distplot(basetable1.medv, bins = bin_size, hist = True, rug = True)
# number of bins are calculated as per Sturge’s rule K = 1 + 3. 322 logN
```

Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x2bb78bc2668>



Linear Regression Assumptions

1. Linear relationship between target and features
2. No outliers
3. No high-leverage points
4. Homoscedasticity of error terms
5. Uncorrelated error terms
6. Independent features

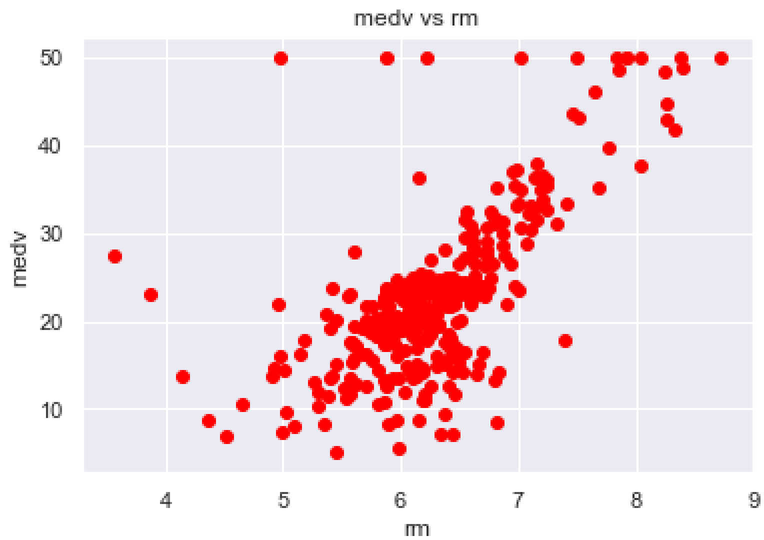
#1 Linear Relationship Between Target & Features

In [19]:

```
x = basetable1.rm
y = basetable1.medv
```

In [20]:

```
plt.scatter(x,y,color = 'red')
#plot.plot(xTrain, linearRegressor.predict(xTrain), color = 'blue')
plt.title('medv vs rm')
plt.xlabel('rm')
plt.ylabel('medv')
plt.show()
```



In [21]:

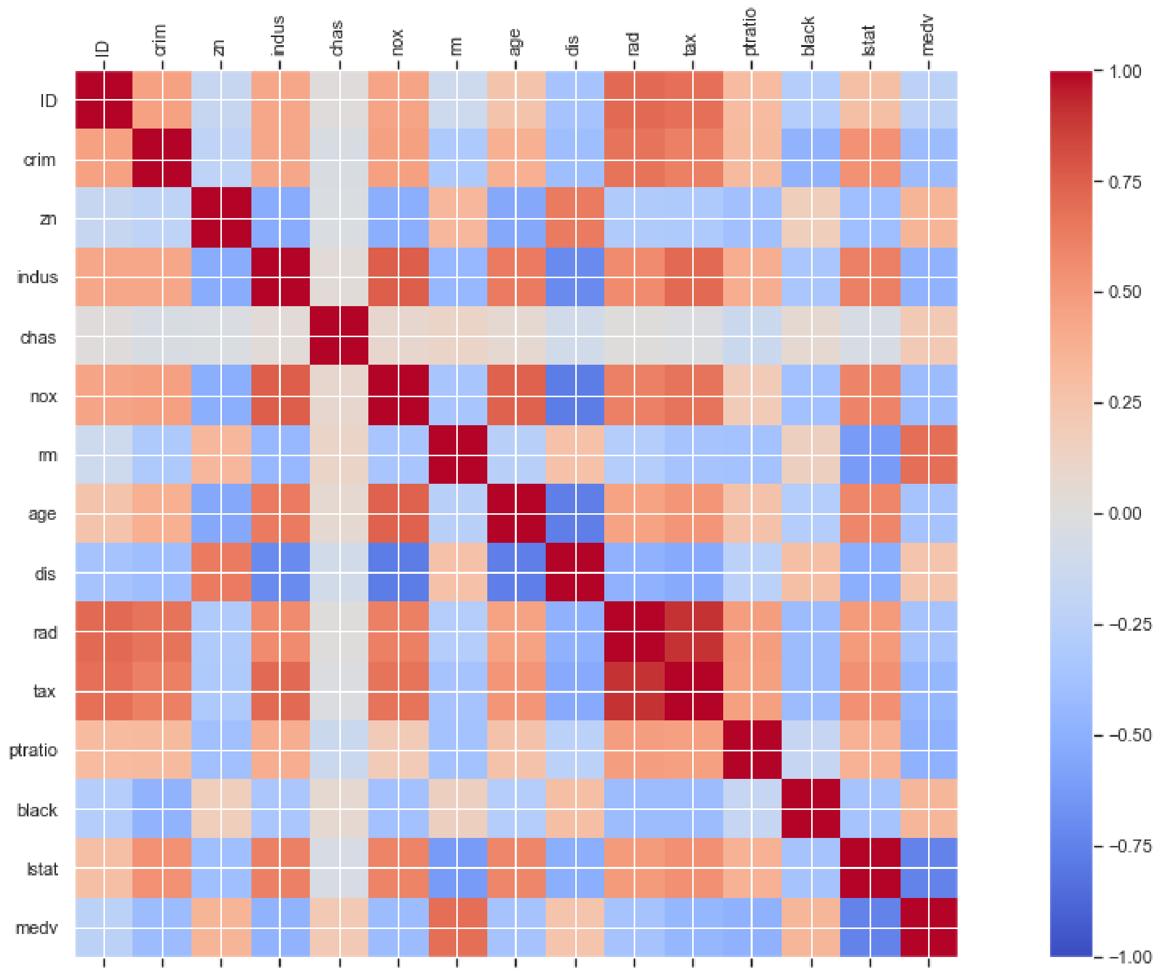
```
corr = basetable1.corr()
print(corr)
```

```
                ID      crim        zn     indus      chas       nox       
rm  \
ID        1.000000  0.456312 -0.155639  0.421978  0.007958  0.440185 -0.1127
90
crim      0.456312  1.000000 -0.210913  0.422228 -0.041195  0.463001 -0.3101
80
zn       -0.155639 -0.210913  1.000000 -0.518679 -0.024442 -0.501990  0.3281
97
indus     0.421978  0.422228 -0.518679  1.000000  0.037496  0.750087 -0.4403
65
chas      0.007958 -0.041195 -0.024442  0.037496  1.000000  0.080275  0.1122
51
nox       0.440185  0.463001 -0.501990  0.750087  0.080275  1.000000 -0.3385
15
rm       -0.112790 -0.310180  0.328197 -0.440365  0.112251 -0.338515  1.0000
00
age       0.257300  0.379034 -0.544513  0.638378  0.068286  0.736000 -0.2485
73
dis      -0.356461 -0.397067  0.637142 -0.702327 -0.081834 -0.769364  0.2691
91
rad       0.707526  0.666636 -0.303663  0.569779  0.007714  0.612180 -0.2727
83
tax       0.686246  0.617081 -0.311180  0.708313 -0.021826  0.670722 -0.3569
87
ptratio   0.309838  0.313409 -0.380449  0.391087 -0.125067  0.192513 -0.3669
27
black    -0.271619 -0.475796  0.168130 -0.335049  0.062029 -0.369416  0.1552
02
lstat     0.281953  0.532077 -0.388112  0.614155 -0.050055  0.598874 -0.6157
47
medv     -0.221694 -0.407454  0.344842 -0.473932  0.204390 -0.413054  0.6895
98

               age       dis       rad       tax   ptratio     black      lst
at  \
ID        0.257300 -0.356461  0.707526  0.686246  0.309838 -0.271619  0.2819
53
crim      0.379034 -0.397067  0.666636  0.617081  0.313409 -0.475796  0.5320
77
zn       -0.544513  0.637142 -0.303663 -0.311180 -0.380449  0.168130 -0.3881
12
indus     0.638378 -0.702327  0.569779  0.708313  0.391087 -0.335049  0.6141
55
chas      0.068286 -0.081834  0.007714 -0.021826 -0.125067  0.062029 -0.0500
55
nox       0.736000 -0.769364  0.612180  0.670722  0.192513 -0.369416  0.5988
74
rm       -0.248573  0.269191 -0.272783 -0.356987 -0.366927  0.155202 -0.6157
47
age       1.000000 -0.764208  0.447380  0.511893  0.259293 -0.268054  0.5888
34
dis      -0.764208  1.000000 -0.477610 -0.529539 -0.231101  0.284374 -0.5059
39
rad       0.447380 -0.477610  1.000000  0.903562  0.470849 -0.406405  0.4845
```

```
68
tax      0.511893 -0.529539  0.903562  1.000000  0.467437 -0.406477  0.5444
85
ptratio  0.259293 -0.231101  0.470849  0.467437  1.000000 -0.164614  0.3748
02
black   -0.268054  0.284374 -0.406405 -0.406477 -0.164614  1.000000 -0.3566
93
lstat    0.588834 -0.505939  0.484568  0.544485  0.374802 -0.356693  1.0000
00
medv    -0.358888  0.249422 -0.352251 -0.448078 -0.481376  0.336660 -0.7386
00

             medv
ID      -0.221694
crim    -0.407454
zn       0.344842
indus   -0.473932
chas     0.204390
nox     -0.413054
rm       0.689598
age     -0.358888
dis      0.249422
rad     -0.352251
tax     -0.448078
ptratio -0.481376
black    0.336660
lstat   -0.738600
medv     1.000000
```

In [23]:

```
fig = plt.figure(figsize=(20,10))
ax = fig.add_subplot(111)
cax = ax.matshow(corr,cmap='coolwarm', vmin=-1, vmax=1)
fig.colorbar(cax)
ticks = np.arange(0,len(basetable1.columns),1)
ax.set_xticks(ticks)
plt.xticks(rotation=90)
ax.set_yticks(ticks)
ax.set_xticklabels(basetable1.columns)
ax.set_yticklabels(basetable1.columns)
plt.show()
```



3. Data Preparation

In [24]:

```python
basetable1.head()
```

Out[24]:

	ID	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat
0	1	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98
1	2	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14
2	4	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94
3	5	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33
4	7	0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311	15.2	395.60	12.43

In [25]:

```python
# Drop ID
basetable2 = basetable1.drop('ID',axis=1)

print("ID Column Dropped from Dataframe")

basetable2.head()
```

```
ID Column Dropped from Dataframe
```

Out[25]:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	m
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	2
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	2
2	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	3
3	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	3
4	0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311	15.2	395.60	12.43	2

Split Data into Test & Train

Benefit to splitting a dataset into some ratio of training and testing subsets for a learning algorithm

- **Motivation:** we need a way to choose between machine learning models and our goal is to estimate likely performance of a model on out-of-sample data.
- **Initial idea:** we can train and test on the same data. However this will cause overfitting. As the number of features in a dataset increases the problem will increase
- **Alternative idea:** we can use train/test split. We can split the dataset into two pieces so that the model can be trained and tested on different data. Then, testing accuracy is a better estimate than training accuracy of out-of-sample performance.

In [26]:
```
predictor = basetable2.drop('medv', axis = 1)
print(" Dependent variable : 'medv' Column removed from features")
predictor.head()
```

```
 Dependent variable : 'medv' Column removed from features
```

Out[26]:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14
2	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94
3	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33
4	0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311	15.2	395.60	12.43

In [30]:
```
target = basetable2['medv']
print(" Target variable : 'medv' Column retained from features")
target.head()
```

```
 Target variable : 'medv' Column retained from features
```

Out[30]:
```
0    24.0
1    21.6
2    33.4
3    36.2
4    22.9
Name: medv, dtype: float64
```

train_test_split

In [31]:
```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(predictor, target, test_s

# Success
print("Training and testing split by 70/30 was successful")
```

```
Training and testing split by 70/30 was successful
```

In [32]:
```
print("Training Predictor dimension :",x_train.shape)
print("Training Target dimension :",y_train.shape)
print("Test Predictor dimension :",x_test.shape)
print("Test Target dimension :",y_test.shape)
```

```
Training Predictor dimension : (233, 13)
Training Target dimension : (233,)
Test Predictor dimension : (100, 13)
Test Target dimension : (100,)
```

4. Model

lm1 : Raw data only removing ID

In [33]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

#Model Training
lm1 = LinearRegression(fit_intercept=True,normalize=False)
print("Parameters of Linear Regressor function : ",lm1.get_params)

#Model Training
lm1.fit(x_train,y_train)
# model is trained on raw data
```

```
Parameters of Linear Regressor function :  <bound method BaseEstimator.get_
params of LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)>
```

Out[33]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

Prediction of Y based on test sample

In [34]:
```
y_pred = lm1.predict(x_test)
print("Total number of predicted values = ",y_pred.shape)
```

```
Total number of predicted values =  (100,)
```

In [35]:
```
# The coefficients
print(lm1.coef_)
```

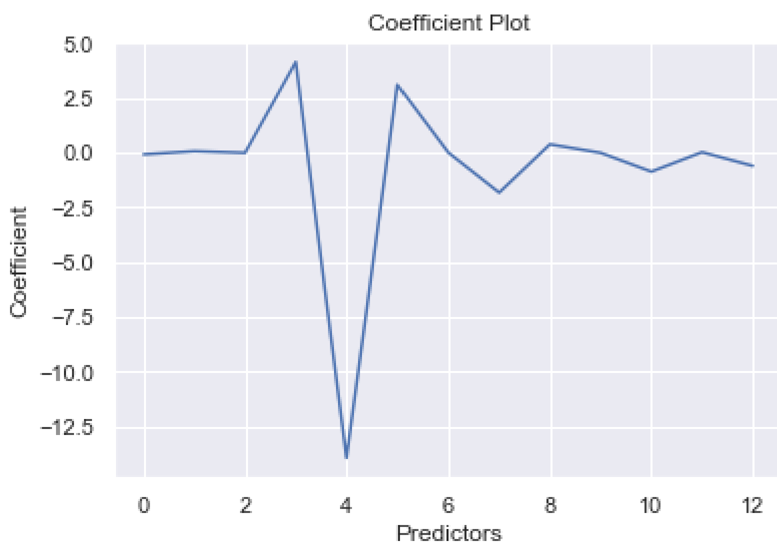
```
[-8.58952519e-02  5.98720246e-02 -1.28016377e-02  4.13703504e+00
 -1.39288018e+01  3.09043992e+00 -6.73962111e-03 -1.83154926e+00
  3.75417497e-01 -1.38415373e-02 -8.73159687e-01  1.46218453e-02
 -6.10464808e-01]
```

In [ ]:

In [48]:

```
# The coefficients
plt.plot( lm1.coef_)

plt.title("Coefficient Plot")
plt.xlabel("Predictors")
plt.ylabel("Coefficient")
#crim   zn  indus   chas    nox rm  age dis rad tax ptratio black   lstat
```

Out[48]: Text(0, 0.5, 'Coefficient')



5. Model Evaluation

1. root-mean-square error (RMSE) for the Model

2. R-Sqauared for the Model

In [50]:

```
from math import sqrt

#Calculate root-mean-square error (RMSE):
print("R-Squared for the above model : ",r2_score(y_test,y_pred)*100,"%")

#Calculate R-squared for the Model:
print("\nroot-mean-square error (RMSE) for the model is : ",sqrt(mean_squarec
```

```
R-Squared for the above model :  78.08137322900414 %

root-mean-square error (RMSE) for the model is :  4.362458566264191
```

1. Using Statmodels.api to train the model

2. Print Summary for the Model

2.[illegible] Print Summary for the Model

In [51]:

```
import statsmodels.api as sm

model = sm.OLS(y_train,x_train).fit()
model.summary()
```

Out[51]:

OLS Regression Results

Dep. Variable:	medv	**R-squared:**	0.954
Model:	OLS	**Adj. R-squared:**	0.951
Method:	Least Squares	**F-statistic:**	348.0
Date:	Tue, 17 Dec 2019	**Prob (F-statistic):**	1.05e-138
Time:	07:45:33	**Log-Likelihood:**	-714.65
No. Observations:	233	**AIC:**	1455.
Df Residuals:	220	**BIC:**	1500.
Df Model:	13		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
crim	-0.0408	0.068	-0.597	0.551	-0.175	0.094
zn	0.0612	0.023	2.710	0.007	0.017	0.106
indus	-0.0192	0.095	-0.203	0.840	-0.206	0.167
chas	4.5415	1.631	2.784	0.006	1.326	7.757
nox	1.7283	5.583	0.310	0.757	-9.276	12.732
rm	5.2955	0.504	10.509	0.000	4.302	6.289
age	-0.0180	0.022	-0.821	0.412	-0.061	0.025
dis	-1.2825	0.343	-3.744	0.000	-1.958	-0.607
rad	0.2140	0.106	2.015	0.045	0.005	0.423
tax	-0.0096	0.006	-1.592	0.113	-0.022	0.002
ptratio	-0.2779	0.179	-1.556	0.121	-0.630	0.074
black	0.0214	0.004	5.036	0.000	0.013	0.030
lstat	-0.5250	0.079	-6.679	0.000	-0.680	-0.370

Omnibus:	100.684	**Durbin-Watson:**	1.930
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	455.674
Skew:	1.707	**Prob(JB):**	1.13e-99
Kurtosis:	8.939	**Cond. No.**	9.01e+03

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 9.01e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

coef std err t P>|t| [0.025 0.975] crim -0.0408 0.068 -0.597 0.551 -0.175 0.094 indus -0.0192 0.095 -0.203 0.840 -0.206 0.167 nox 1.7283 5.583 0.310 0.757 -9.276 12.732 age -0.0180 0.022 -0.821 0.412 -0.061 0.025 tax -0.0096 0.006 -1.592 0.113 -0.022 0.002

zn 0.0612 0.023 2.710 0.007 0.017 0.106

chas 4.5415 1.631 2.784 0.006 1.326 7.757

rm 5.2955 0.504 10.509 0.000 4.302 6.289

dis -1.2825 0.343 -3.744 0.000 -1.958 -0.607 rad 0.2140 0.106 2.015 0.045 0.005 0.423

ptratio -0.2779 0.179 -1.556 0.121 -0.630 0.074 black 0.0214 0.004 5.036 0.000 0.013 0.030 lstat -0.5250 0.079 -6.679 0.000 -0.680 -0.370

rad : index of accessibility to radial highways. tax : full-value property-tax rate per $10,000.

High multi colinearity

In [53]:
```
# Create New Feature Tax_Rad
basetable2['tax_rad'] = basetable2.tax * basetable1.rad
basetable2.head()
```

Out[53]:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	m
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	2
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	2
2	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	3
3	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	3
4	0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311	15.2	395.60	12.43	2

In [54]:

```python
#drop Tax and Rad as they are highly multicolinear
basetable3 = basetable2.drop(['tax', 'rad'],axis=1)

print("tax & rad Column Dropped from Dataframe")

basetable3.head()
```

```
tax & rad Column Dropped from Dataframe
```

Out[54]:

	crim	zn	indus	chas	nox	rm	age	dis	ptratio	black	lstat	medv	Tax_R
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	15.3	396.90	4.98	24.0	2
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	17.8	396.90	9.14	21.6	4
2	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	18.7	394.63	2.94	33.4	6
3	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	18.7	396.90	5.33	36.2	6
4	0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	15.2	395.60	12.43	22.9	15

In [55]:

```python
# Drop indus, nox, crim, age Based on High p Value

basetable3 = basetable2.drop(['indus', 'nox', 'crim', 'age'],axis=1)

print("indus, nox, crim, age Column Dropped from Dataframe")

basetable3.head()
```

```
indus, nox, crim, age Column Dropped from Dataframe
```

Out[55]:

	zn	chas	rm	dis	rad	tax	ptratio	black	lstat	medv	Tax_Rad
0	18.0	0	6.575	4.0900	1	296	15.3	396.90	4.98	24.0	296
1	0.0	0	6.421	4.9671	2	242	17.8	396.90	9.14	21.6	484
2	0.0	0	6.998	6.0622	3	222	18.7	394.63	2.94	33.4	666
3	0.0	0	7.147	6.0622	3	222	18.7	396.90	5.33	36.2	666
4	12.5	0	6.012	5.5605	5	311	15.2	395.60	12.43	22.9	1555

In [58]:

```python
basetable3 = basetable3.rename(columns={'Tax_Rad': 'tax_rad'})
```

In [59]:

```python
basetable3.head()
```

Out[59]:

	zn	chas	rm	dis	rad	tax	ptratio	black	lstat	medv	tax_rad
0	18.0	0	6.575	4.0900	1	296	15.3	396.90	4.98	24.0	296
1	0.0	0	6.421	4.9671	2	242	17.8	396.90	9.14	21.6	484
2	0.0	0	6.998	6.0622	3	222	18.7	394.63	2.94	33.4	666
3	0.0	0	7.147	6.0622	3	222	18.7	396.90	5.33	36.2	666
4	12.5	0	6.012	5.5605	5	311	15.2	395.60	12.43	22.9	1555

In [ ]:

Model Attempt 2

In [60]:

```
#get Predictor Dataframe
predictor = basetable3.drop('medv', axis = 1)
print(" Dependent variable : 'medv' Column removed from features")
predictor.head()

#get Target Dataframe
target = basetable3['medv']
target.head()

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(predictor, target, test_s

# Success
print("Training and testing split by 70/30 was successful")

print("Training Predictor dimension :",x_train.shape)
print("Training Target dimension :",y_train.shape)
print("Test Predictor dimension :",x_test.shape)
print("Test Target dimension :",y_test.shape)
```

```
 Dependent variable : 'medv' Column removed from features
Training and testing split by 70/30 was successful
Training Predictor dimension : (233, 10)
Training Target dimension : (233,)
Test Predictor dimension : (100, 10)
Test Target dimension : (100,)
```

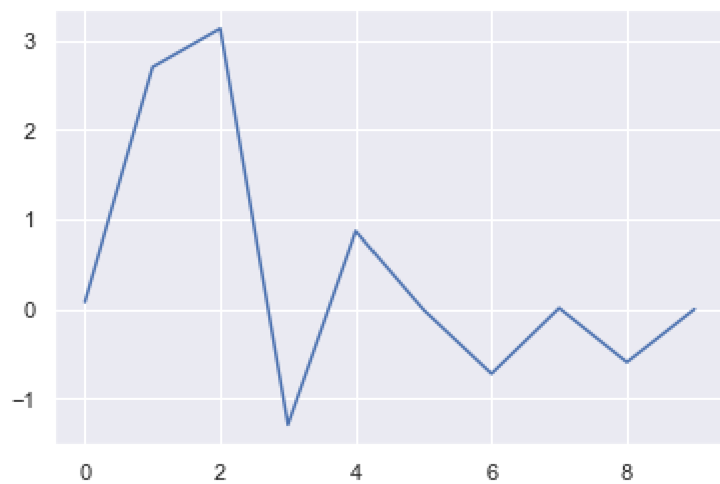
In [61]:

```
#import library
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

#Model Training
lm2 = LinearRegression(fit_intercept=True,normalize=False)
print("Parameters of Linear Regressor function : ",lm2.get_params)

#Model Training
lm2.fit(x_train,y_train)

#Predict
y_pred = lm2.predict(x_test)
print("Total number of predicted values = ",y_pred.shape)

# The coefficients
plt.plot(lm2.coef_)
```

```
Parameters of Linear Regressor function :  <bound method BaseEstimator.get_
params of LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)>
Total number of predicted values =  (100,)
```

Out[61]:

```
[<matplotlib.lines.Line2D at 0x2bb7742d358>]
```



0zn 1chas 2rm 3dis 4rad 5tax 6ptratio 7black 8lstat 9tax_rad chas : Charles River dummy variable (= 1 if tract bounds river; 0 otherwise). rm : average number of rooms per dwelling.

Room and Charles river has highest coefficient

Model Evaluation

In [62]:

```python
#Model Evaluation

from math import sqrt

#Calculate root-mean-square error (RMSE):
print("R-Squared for the above model : ",r2_score(y_test,y_pred)*100,"%")

#Calculate R-squared for the Model:
print("\nroot-mean-square error (RMSE) for the model is : ",sqrt(mean_squared
```

```
R-Squared for the above model :  59.77538914536744 %

root-mean-square error (RMSE) for the model is :  5.894945770741745
```

In [63]:

```
model = sm.OLS(y_train,x_train).fit()
model.summary()
```

Out[63]:

OLS Regression Results

Dep. Variable:	medv	**R-squared:**	0.964
Model:	OLS	**Adj. R-squared:**	0.962
Method:	Least Squares	**F-statistic:**	596.7
Date:	Tue, 17 Dec 2019	**Prob (F-statistic):**	7.03e-155
Time:	08:26:03	**Log-Likelihood:**	-683.65
No. Observations:	233	**AIC:**	1387.
Df Residuals:	223	**BIC:**	1422.
Df Model:	10		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
zn	0.0776	0.019	4.148	0.000	0.041	0.114
chas	3.0757	1.495	2.058	0.041	0.130	6.021
rm	4.8211	0.379	12.718	0.000	4.074	5.568
dis	-1.1184	0.238	-4.695	0.000	-1.588	-0.649
rad	1.1862	0.380	3.119	0.002	0.437	1.935
tax	-0.0029	0.005	-0.600	0.549	-0.013	0.007
ptratio	-0.3249	0.154	-2.111	0.036	-0.628	-0.022
black	0.0158	0.004	4.214	0.000	0.008	0.023
lstat	-0.4835	0.060	-8.089	0.000	-0.601	-0.366
tax_rad	-0.0016	0.001	-2.829	0.005	-0.003	-0.000

Omnibus:	77.703	**Durbin-Watson:**	2.097
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	303.603
Skew:	1.317	**Prob(JB):**	1.18e-66
Kurtosis:	7.933	**Cond. No.**	4.24e+04

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.24e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

In [65]:

```
basetable4 = basetable3.drop(['tax', 'rad'],axis=1)
basetable4.head()
```

Out[65]:

	zn	chas	rm	dis	ptratio	black	lstat	medv	tax_rad
0	18.0	0	6.575	4.0900	15.3	396.90	4.98	24.0	296
1	0.0	0	6.421	4.9671	17.8	396.90	9.14	21.6	484
2	0.0	0	6.998	6.0622	18.7	394.63	2.94	33.4	666
3	0.0	0	7.147	6.0622	18.7	396.90	5.33	36.2	666
4	12.5	0	6.012	5.5605	15.2	395.60	12.43	22.9	1555

Model Attempt 3

In [66]:

```
#get Predictor Dataframe
predictor = basetable4.drop('medv', axis = 1)
print(" Dependent variable : 'medv' Column removed from features")
predictor.head()

#get Target Dataframe
target = basetable4['medv']
target.head()

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(predictor, target, test_s

# Success
print("Training and testing split by 70/30 was successful")

print("Training Predictor dimension :",x_train.shape)
print("Training Target dimension :",y_train.shape)
print("Test Predictor dimension :",x_test.shape)
print("Test Target dimension :",y_test.shape)
```

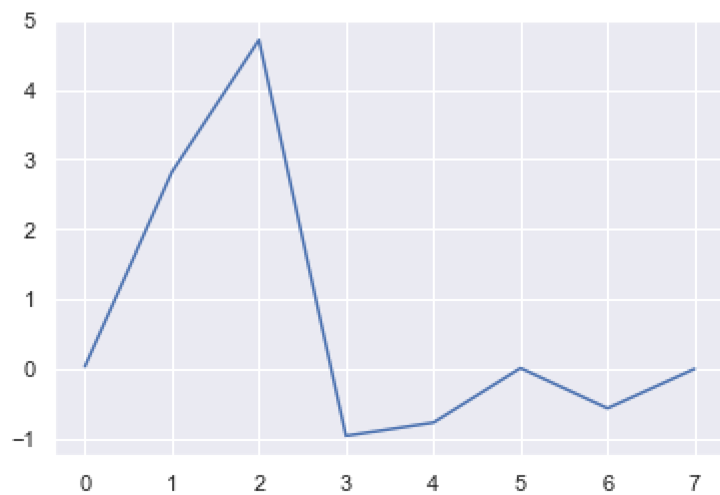
```
 Dependent variable : 'medv' Column removed from features
Training and testing split by 70/30 was successful
Training Predictor dimension : (233, 8)
Training Target dimension : (233,)
Test Predictor dimension : (100, 8)
Test Target dimension : (100,)
```

In [67]:

```
#import library
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

#Model Training
lm3 = LinearRegression(fit_intercept=True,normalize=False)
print("Parameters of Linear Regressor function : ",lm3.get_params)

#Model Training
lm3.fit(x_train,y_train)

#Predict
y_pred = lm3.predict(x_test)
print("Total number of predicted values = ",y_pred.shape)

# The coefficients
plt.plot(lm3.coef_)
```

```
Parameters of Linear Regressor function :  <bound method BaseEstimator.get_
params of LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)>
Total number of predicted values =  (100,)
```

Out[67]: [<matplotlib.lines.Line2D at 0x2bb78d7a080>]



0zn 1chas 2rm 3dis 4ptratio 5black 6lstat 7tax_rad

Model Evaluation

In [68]:

```python
#Model Evaluation

from math import sqrt

#Calculate root-mean-square error (RMSE):
print("R-Squared for the above model : ",r2_score(y_test,y_pred)*100,"%")

#Calculate R-squared for the Model:
print("\nroot-mean-square error (RMSE) for the model is : ",sqrt(mean_squarec
```

```
R-Squared for the above model :  70.86083121002729 %

root-mean-square error (RMSE) for the model is :  5.205159161462403
```

R squared is still low 70% and RMSE is high 5.2

In [69]:
```
model = sm.OLS(y_train,x_train).fit()
model.summary()
```

Out[69]:

OLS Regression Results

Dep. Variable:	medv	**R-squared:**	0.959
Model:	OLS	**Adj. R-squared:**	0.958
Method:	Least Squares	**F-statistic:**	660.7
Date:	Tue, 17 Dec 2019	**Prob (F-statistic):**	1.20e-151
Time:	08:33:15	**Log-Likelihood:**	-701.85
No. Observations:	233	**AIC:**	1420.
Df Residuals:	225	**BIC:**	1447.
Df Model:	8		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
zn	0.0352	0.020	1.764	0.079	-0.004	0.075
chas	2.9610	1.290	2.296	0.023	0.420	5.502
rm	5.8555	0.379	15.468	0.000	5.110	6.601
dis	-0.8757	0.257	-3.401	0.001	-1.383	-0.368
ptratio	-0.5564	0.156	-3.574	0.000	-0.863	-0.250
black	0.0138	0.004	3.207	0.002	0.005	0.022
lstat	-0.4786	0.059	-8.121	0.000	-0.595	-0.362
tax_rad	-3.185e-05	6.79e-05	-0.469	0.640	-0.000	0.000

Omnibus:	94.266	**Durbin-Watson:**	1.895
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	446.643
Skew:	1.559	**Prob(JB):**	1.03e-97
Kurtosis:	9.024	**Cond. No.**	3.18e+04

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.18e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

tax_rad is not helping us. this is in significant. lets remove this and create model again. also Chas is zn has p value

Model Attempt 4

In [71]:
```
basetable5 = basetable4.drop(['tax_rad', 'zn'],axis=1)
basetable5.head()
```

Out[71]:

	chas	rm	dis	ptratio	black	lstat	medv
0	0	6.575	4.0900	15.3	396.90	4.98	24.0
1	0	6.421	4.9671	17.8	396.90	9.14	21.6
2	0	6.998	6.0622	18.7	394.63	2.94	33.4
3	0	7.147	6.0622	18.7	396.90	5.33	36.2
4	0	6.012	5.5605	15.2	395.60	12.43	22.9

In [72]:
```
#get Predictor Dataframe
predictor = basetable5.drop('medv', axis = 1)
print(" Dependent variable : 'medv' Column removed from features")
predictor.head()

#get Target Dataframe
target = basetable5['medv']
target.head()

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(predictor, target, test_s

# Success
print("Training and testing split by 70/30 was successful")

print("Training Predictor dimension :",x_train.shape)
print("Training Target dimension :",y_train.shape)
print("Test Predictor dimension :",x_test.shape)
print("Test Target dimension :",y_test.shape)
```

```
 Dependent variable : 'medv' Column removed from features
Training and testing split by 70/30 was successful
Training Predictor dimension : (233, 6)
Training Target dimension : (233,)
Test Predictor dimension : (100, 6)
Test Target dimension : (100,)
```

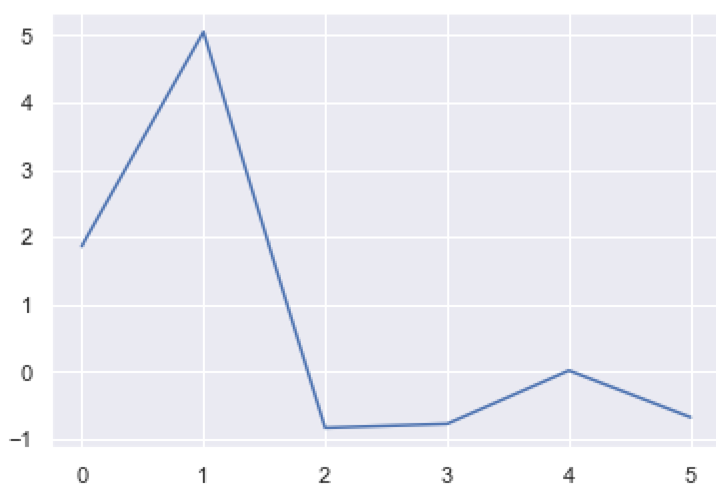
In [73]:

```python
#import library
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

#Model Training
lm4 = LinearRegression(fit_intercept=True,normalize=False)
print("Parameters of Linear Regressor function : ",lm4.get_params)

#Model Training
lm4.fit(x_train,y_train)

#Predict
y_pred = lm4.predict(x_test)
print("Total number of predicted values = ",y_pred.shape)

# The coefficients
plt.plot(lm4.coef_)
```

```
Parameters of Linear Regressor function :  <bound method BaseEstimator.get_
params of LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)>
Total number of predicted values =  (100,)
```

Out[73]:

```
[<matplotlib.lines.Line2D at 0x2bb7795ca58>]
```



0chas 1rm 2dis 3ptratio 4black 5lstat

Model Evaluation

In [102]:
```
#Model Evaluation

from math import sqrt

#Calculate root-mean-square error (RMSE):
print("R-Squared for the above model : ",r2_score(y_test,y_pred)*100,"%")

#Calculate R-squared for the Model:
print("\nroot-mean-square error (RMSE) for the model is : ",sqrt(mean_squared
```

```
R-Squared for the above model :  -84.6901264244536 %

root-mean-square error (RMSE) for the model is :  11.520263468475049
```

In [75]:
```
model = sm.OLS(y_train,x_train).fit()
model.summary()
```

Out[75]:

OLS Regression Results

Dep. Variable:	medv	**R-squared:**	0.962
Model:	OLS	**Adj. R-squared:**	0.961
Method:	Least Squares	**F-statistic:**	966.3
Date:	Tue, 17 Dec 2019	**Prob (F-statistic):**	1.49e-158
Time:	08:47:02	**Log-Likelihood:**	-703.44
No. Observations:	233	**AIC:**	1419.
Df Residuals:	227	**BIC:**	1440.
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
chas	1.9041	1.303	1.461	0.145	-0.663	4.471

Model Attempt 5

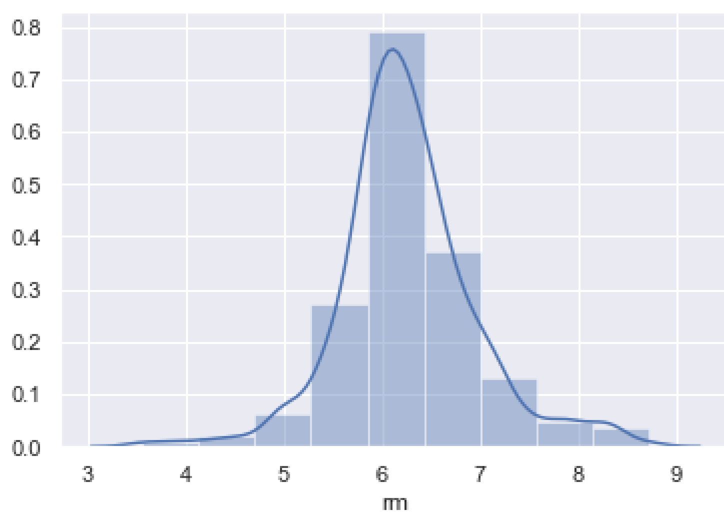
In [77]:
```python
basetable6 = basetable5.drop(['chas'],axis=1)
basetable6.head()
```

Out[77]:

	rm	dis	ptratio	black	lstat	medv
0	6.575	4.0900	15.3	396.90	4.98	24.0
1	6.421	4.9671	17.8	396.90	9.14	21.6
2	6.998	6.0622	18.7	394.63	2.94	33.4
3	7.147	6.0622	18.7	396.90	5.33	36.2
4	6.012	5.5605	15.2	395.60	12.43	22.9

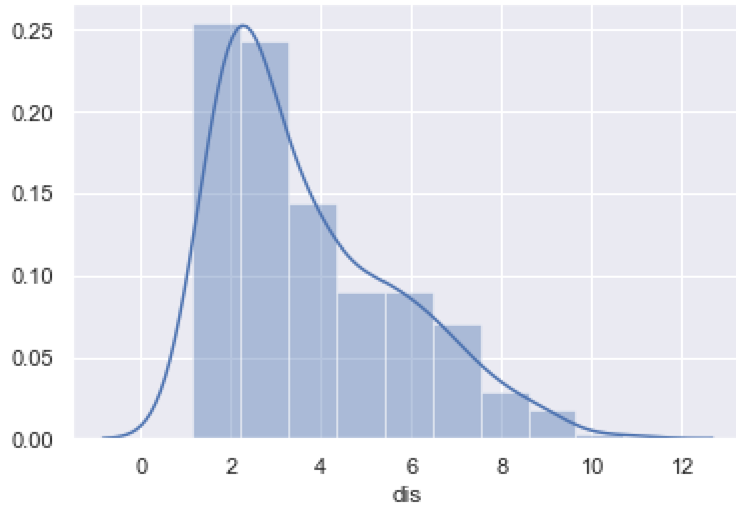
In [79]:
```python
sns.distplot(basetable5.rm, bins = bin_size, hist = True)
# Visualize the outliers in normal distribution
```

Out[79]: <matplotlib.axes._subplots.AxesSubplot at 0x2bb777ca780>



In [80]:
```
sns.distplot(basetable5.dis, bins = bin_size, hist = True)
# Visualize the outliers in normal distribution
```

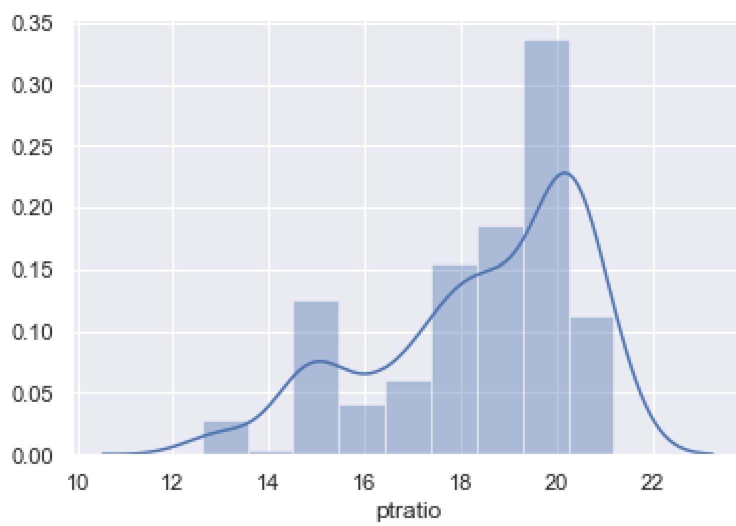
Out[80]: <matplotlib.axes._subplots.AxesSubplot at 0x2bb77223cc0>



In [81]:
```
#dis is negatively skewed
```

In [82]:
```
sns.distplot(basetable5.ptratio, bins = bin_size, hist = True)
# Visualize the outliers in normal distribution
```

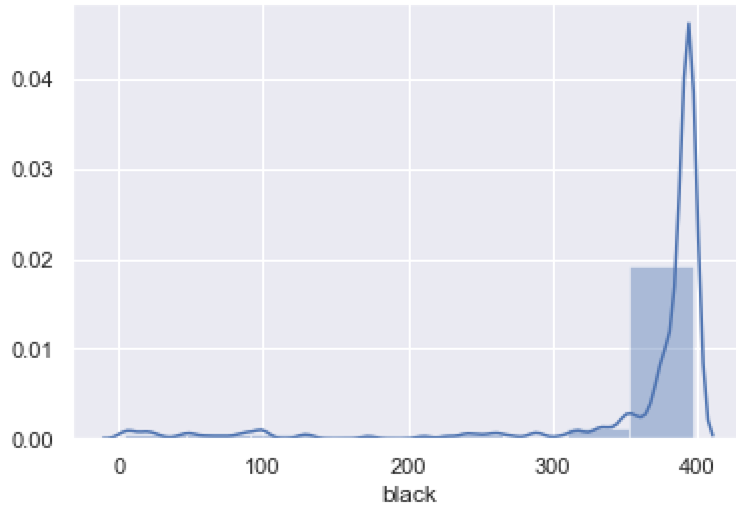
Out[82]: <matplotlib.axes._subplots.AxesSubplot at 0x2bb77052240>



In [83]:
```
#pt ratio is potitively skewed
```

In [84]:
```
sns.distplot(basetable5.black, bins = bin_size, hist = True)
# Visualize the outliers in normal distribution
```

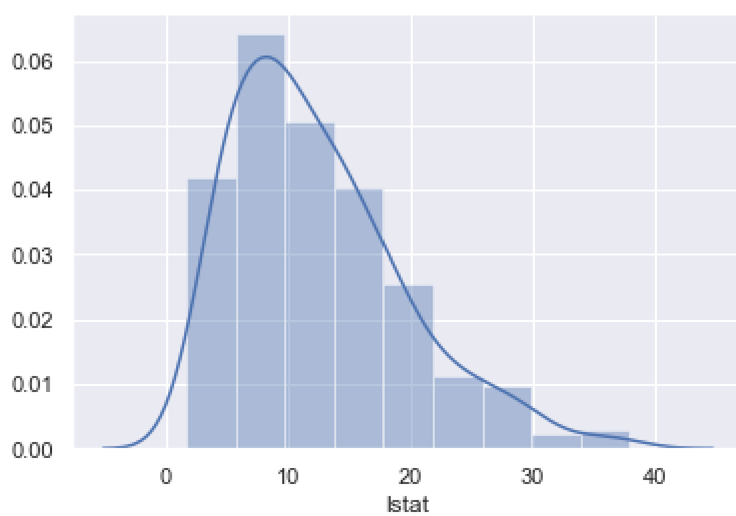
Out[84]: <matplotlib.axes._subplots.AxesSubplot at 0x2bb77462588>



In [85]:
```
#black is highly positively skewed
```

In [86]:
```

sns.distplot(basetable5.lstat, bins = bin_size, hist = True)
# Visualize the outliers in normal distribution
```

Out[86]: <matplotlib.axes._subplots.AxesSubplot at 0x2bb775a8f98>



In [87]:

```
#lstat is normal but negatively skewed
```

In [88]:

```

#remove ouliers
from scipy import stats
basetable6 = basetable5[(np.abs(stats.zscore(basetable5)) < 3).all(axis=1)]
print ("outliers removed beyond 3SD.")

basetable6.head()
```

```
outliers removed beyond 3SD.
```

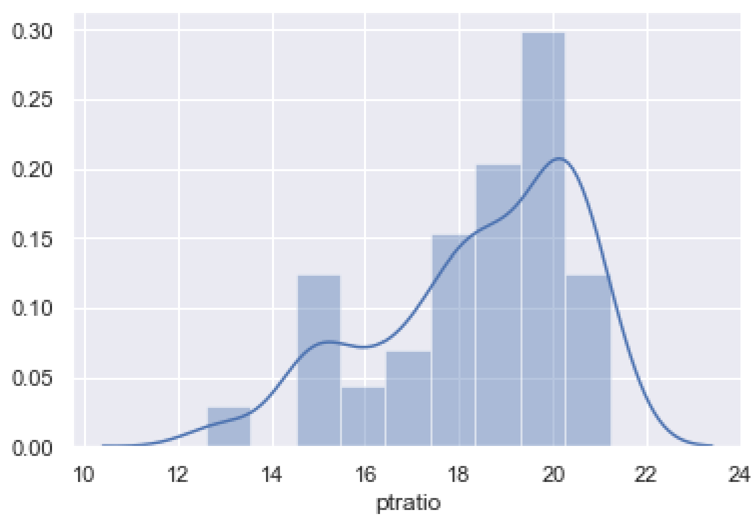
Out[88]:

	chas	rm	dis	ptratio	black	lstat	medv
0	0	6.575	4.0900	15.3	396.90	4.98	24.0
1	0	6.421	4.9671	17.8	396.90	9.14	21.6
2	0	6.998	6.0622	18.7	394.63	2.94	33.4
3	0	7.147	6.0622	18.7	396.90	5.33	36.2
4	0	6.012	5.5605	15.2	395.60	12.43	22.9

In [89]:

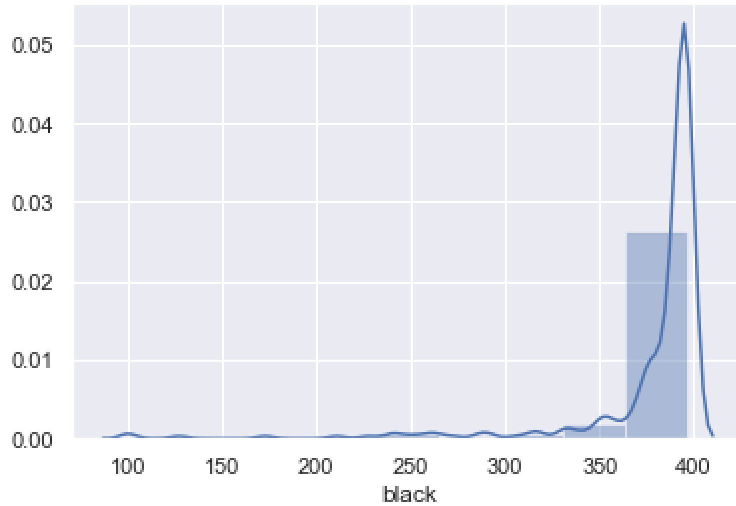
```
sns.distplot(basetable6.ptratio, bins = bin_size, hist = True)
# Visualize the outliers in normal distribution
```

Out[89]: <matplotlib.axes._subplots.AxesSubplot at 0x2bb77408048>



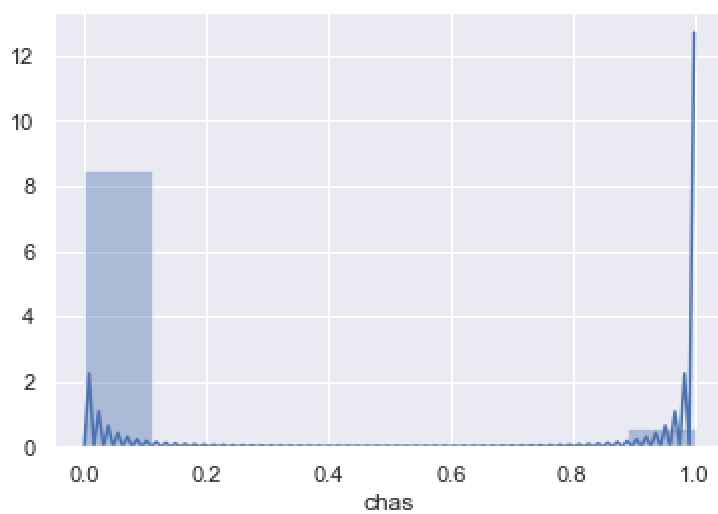
In [90]:
```python
sns.distplot(basetable6.black, bins = bin_size, hist = True)
# Visualize the outliers in normal distribution
```

Out[90]: <matplotlib.axes._subplots.AxesSubplot at 0x2bb7a849780>



In [96]:
```python
sns.distplot(basetable5.chas, bins = bin_size, hist = True)
# Visualize the outliers in normal distribution
```

Out[96]: <matplotlib.axes._subplots.AxesSubplot at 0x2bb7aaa4898>



In [97]:

```python
basetable6.head()
```

Out[97]:

	chas	rm	dis	ptratio	black	lstat	medv
0	0	6.575	4.0900	15.3	396.90	4.98	24.0
1	0	6.421	4.9671	17.8	396.90	9.14	21.6
2	0	6.998	6.0622	18.7	394.63	2.94	33.4
3	0	7.147	6.0622	18.7	396.90	5.33	36.2
4	0	6.012	5.5605	15.2	395.60	12.43	22.9

In [98]:

```python
basetable6 = basetable6.drop(['chas'],axis=1)
basetable6.head()
```

Out[98]:

	rm	dis	ptratio	black	lstat	medv
0	6.575	4.0900	15.3	396.90	4.98	24.0
1	6.421	4.9671	17.8	396.90	9.14	21.6
2	6.998	6.0622	18.7	394.63	2.94	33.4
3	7.147	6.0622	18.7	396.90	5.33	36.2
4	6.012	5.5605	15.2	395.60	12.43	22.9

In [99]:

```python
#get Predictor Dataframe
predictor = basetable6.drop('medv', axis = 1)
print(" Dependent variable : 'medv' Column removed from features")
predictor.head()

#get Target Dataframe
target = basetable6['medv']
target.head()

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(predictor, target, test_s

# Success
print("Training and testing split by 70/30 was successful")

print("Training Predictor dimension :",x_train.shape)
print("Training Target dimension :",y_train.shape)
print("Test Predictor dimension :",x_test.shape)
print("Test Target dimension :",y_test.shape)
```

```
 Dependent variable : 'medv' Column removed from features
Training and testing split by 70/30 was successful
Training Predictor dimension : (200, 5)
Training Target dimension : (200,)
Test Predictor dimension : (87, 5)
Test Target dimension : (87,)
```

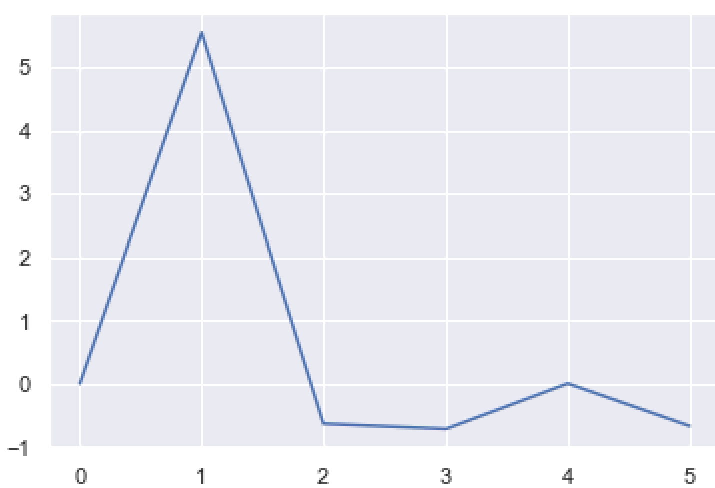
In [94]:

```
#import library
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

#Model Training
lm5 = LinearRegression(fit_intercept=True,normalize=False)
print("Parameters of Linear Regressor function : ",lm5.get_params)

#Model Training
lm5.fit(x_train,y_train)

#Predict
y_pred = lm5.predict(x_test)
print("Total number of predicted values = ",y_pred.shape)

# The coefficients
plt.plot(lm5.coef_)
```

```
Parameters of Linear Regressor function :  <bound method BaseEstimator.get_
params of LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)>
Total number of predicted values =  (87,)
```

Out[94]:

```
[<matplotlib.lines.Line2D at 0x2bb7a9c6438>]
```



In [100]:

```
### MOdel Evalutation
```

In [101]:

```
#Model Evaluation

from math import sqrt

#Calculate root-mean-square error (RMSE):
print("R-Squared for the above model : ",r2_score(y_test,y_pred)*100,"%")

#Calculate R-squared for the Model:
print("\nroot-mean-square error (RMSE) for the model is : ",sqrt(mean_squarec
```

```
R-Squared for the above model :  -84.6901264244536 %

root-mean-square error (RMSE) for the model is :  11.520263468475049
```

Now, this is bad. Ha ha ha :)

In [104]:

```
basetable6.describe()
```

Out[104]:

	rm	dis	ptratio	black	lstat	medv
count	287.000000	287.000000	287.000000	287.000000	287.000000	287.000000
mean	6.273425	3.880460	18.418118	377.752753	11.941289	22.845645
std	0.626856	1.981831	2.132160	43.153169	6.422150	8.329401
min	4.368000	1.169100	12.600000	100.190000	1.730000	5.000000
25%	5.883500	2.262050	17.150000	379.540000	6.925000	18.100000
50%	6.208000	3.331700	18.700000	392.780000	10.420000	21.800000
75%	6.591500	5.344000	20.200000	396.900000	15.470000	25.000000
max	8.337000	9.222900	21.200000	396.900000	31.990000	50.000000

In [ ]:

```

```