**Experiment No: 4**

**Aim:** Create a Linear Regression Model using Python/R to predict home prices using Boston Housing Dataset (https://www.kaggle.com/c/boston-housing). The Boston Housing dataset contains information about various houses in Boston through different parameters. There are 506 samples and 14 feature variables in this dataset. The objective is to predict the value of prices of the house using the given features.

**Theory:** Boston Housing with Linear Regression

With this data our objective is create a model using linear regression to predict the houses price

The data contains the following columns:

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

Ps: this is my first analysis, i'm learning how to interpret the plots.

Lets Start

First we need to prepare our enviroment importing some librarys

In [1]:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

In [2]:

*# Importing DataSet and take a look at Data*

BostonTrain = pd.read\_csv("../input/boston\_train.csv")

Here we can look at the BostonTrain data

In [3]:

BostonTrain.head()

Out[3]:

|  | ID | crim | zn | indus | chas | nox | rm | age | dis | rad | tax | ptratio | black | lstat | medv |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 | 24.0 |
| 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 | 21.6 |
| 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 | 33.4 |
| 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 | 36.2 |
| 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 | 22.9 |

In [4]:

BostonTrain.info()

BostonTrain.describe()

<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

Out[4]:

|  | ID | crim | zn | indus | chas | nox | rm | age | dis | rad | tax | ptratio | black | lstat | medv |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| count | 333.000000 | 333.000000 | 333.000000 | 333.000000 | 333.000000 | 333.000000 | 333.000000 | 333.000000 | 333.000000 | 333.000000 | 333.000000 | 333.000000 | 333.000000 | 333.000000 | 333.000000 |
| mean | 250.951952 | 3.360341 | 10.689189 | 11.293483 | 0.060060 | 0.557144 | 6.265619 | 68.226426 | 3.709934 | 9.633634 | 409.279279 | 18.448048 | 359.466096 | 12.515435 | 22.768769 |
| std | 147.859438 | 7.352272 | 22.674762 | 6.998123 | 0.237956 | 0.114955 | 0.703952 | 28.133344 | 1.981123 | 8.742174 | 170.841988 | 2.151821 | 86.584567 | 7.067781 | 9.173468 |
| min | 1.000000 | 0.006320 | 0.000000 | 0.740000 | 0.000000 | 0.385000 | 3.561000 | 6.000000 | 1.129600 | 1.000000 | 188.000000 | 12.600000 | 3.500000 | 1.730000 | 5.000000 |
| 25% | 123.000000 | 0.078960 | 0.000000 | 5.130000 | 0.000000 | 0.453000 | 5.884000 | 45.400000 | 2.122400 | 4.000000 | 279.000000 | 17.400000 | 376.730000 | 7.180000 | 17.400000 |
| 50% | 244.000000 | 0.261690 | 0.000000 | 9.900000 | 0.000000 | 0.538000 | 6.202000 | 76.700000 | 3.092300 | 5.000000 | 330.000000 | 19.000000 | 392.050000 | 10.970000 | 21.600000 |
| 75% | 377.000000 | 3.678220 | 12.500000 | 18.100000 | 0.000000 | 0.631000 | 6.595000 | 93.800000 | 5.116700 | 24.000000 | 666.000000 | 20.200000 | 396.240000 | 16.420000 | 25.000000 |
| max | 506.000000 | 73.534100 | 100.000000 | 27.740000 | 1.000000 | 0.871000 | 8.725000 | 100.000000 | 10.710300 | 24.000000 | 711.000000 | 21.200000 | 396.900000 | 37.970000 | 50.000000 |

Now, or goal is think about the columns, and discovery which columns is relevant to build our model, because if we consider to put columns with not relevant with our objective "medv" the model may be not efficient

In [5]:

*#ID columns does not relevant for our analysis.*

BostonTrain.drop('ID', axis = 1, inplace=True)

In [6]:

BostonTrain.plot.scatter('rm', 'medv')

Out[6]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fbe883a8080>

In this plot its clearly to see a linear pattern. Wheter more average number of rooms per dwelling, more expensive the median value is.

Now lets take a loot how the all variables relate to each other.

In [7]:

plt.subplots(figsize=(12,8))

sns.heatmap(BostonTrain.corr(), cmap = 'RdGy')

Out[7]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fbe883530b8>

At this heatmap plot, we can do our analysis better than the pairplot.

Lets focus ate the last line, where y = medv:

When shades of Red/Orange: the more red the color is on X axis, smaller the medv. Negative correlation  
When light colors: those variables at axis x and y, they dont have any relation. Zero correlation  
When shades of Gray/Black : the more black the color is on X axis, more higher the value med is. Positive correlation

Lets plot the paiplot, for all different correlations

Negative Correlation.

When x is high y is low and vice versa.

To the right less negative correlation.

In [8]:

sns.pairplot(BostonTrain, vars = ['lstat', 'ptratio', 'indus', 'tax', 'crim', 'nox', 'rad', 'age', 'medv'])

Out[8]:

<seaborn.axisgrid.PairGrid at 0x7fbe88285c50>

Zero Correlation. When x and y are completely independent

Positive Correlation. When x and y go together

to the right more independent.

In [9]:

sns.pairplot(BostonTrain, vars = ['rm', 'zn', 'black', 'dis', 'chas','medv'])

Out[9]:

<seaborn.axisgrid.PairGrid at 0x7fbdf20f9d30>

Trainning Linear Regression Model

Define X and Y

X: Varibles named as predictors, independent variables, features.  
Y: Variable named as response or dependent variable

In [10]:

X = BostonTrain[['crim', 'zn', 'indus', 'chas', 'nox', 'rm', 'age', 'dis', 'rad', 'tax',

'ptratio', 'black', 'lstat']]

y = BostonTrain['medv']

Import sklearn librarys:  
train\_test\_split, to split our data in two DF, one for build a model and other to validate.  
LinearRegression, to apply the linear regression.

In [11]:

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

In [12]:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.4)

In [13]:

lm = LinearRegression()

lm.fit(X\_train,y\_train)

Out[13]:

LinearRegression(copy\_X=True, fit\_intercept=True, n\_jobs=1, normalize=False)

In [14]:

predictions = lm.predict(X\_test)

In [15]:

plt.scatter(y\_test,predictions)

plt.xlabel('Y Test')

plt.ylabel('Predicted Y')

Out[15]:

Text(0,0.5,'Predicted Y')

In [16]:

from sklearn import metrics

print('MAE:', metrics.mean\_absolute\_error(y\_test, predictions))

print('MSE:', metrics.mean\_squared\_error(y\_test, predictions))

print('RMSE:', np.sqrt(metrics.mean\_squared\_error(y\_test, predictions)))

MAE: 3.53544941908

MSE: 20.8892997114

RMSE: 4.57048134351

Considering the RMSE: we can conclude that this model average error is RMSE at medv, which means RMSE \*1000 in money

In [17]:

sns.distplot((y\_test-predictions),bins=50);

As more normal distribution, better it is.

In [18]:

coefficients = pd.DataFrame(lm.coef\_,X.columns)

coefficients.columns = ['coefficients']

coefficients

Out[18]:

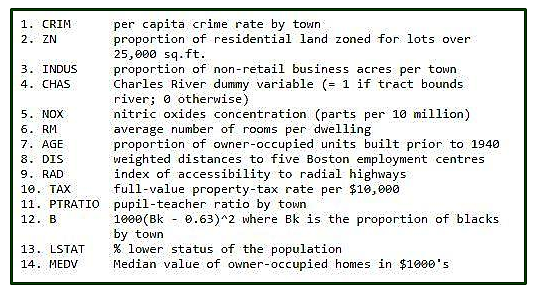
|  | coefficients |
| --- | --- |
| crim | -0.116916 |
| zn | 0.017422 |
| indus | -0.001589 |
| chas | 3.267698 |
| nox | -17.405512 |
| rm | 3.242758 |
| age | 0.006570 |
| dis | -1.414341 |
| rad | 0.404683 |
| Tax | -0.013598 |
| Ptratio | -0.724007 |
| Black | 0.007861 |
| Lstat | -0.711690 |

How to interpret those coefficients: they are in function of Medv, so for one unit that nox increase, the house value decrease 'nox'\*1000 (Negative correlation) money unit, for one unit that rm increase, the house value increase 'rm'\*1000 (Positive correlation) money unit.

\*1000 because the medv is in 1000 and this apply to the other variables/coefficients.

ML | Boston Housing Kaggle Challenge with Linear Regression

**Boston Housing Data:**This dataset was taken from the StatLib library and is maintained by Carnegie Mellon University. This dataset concerns the housing prices in the housing city of Boston. The dataset provided has 506 instances with 13 features.  
The Description of the dataset is taken from 



Let’s make the Linear Regression Model, predicting housing prices  
Inputting Libraries and dataset. 

* Python3

|  |
| --- |
| # Importing Libraries  import numpy as np  import pandas as pd  import matplotlib.pyplot as plt    # Importing Data  from sklearn.datasets import load\_boston  boston = load\_boston() |

The shape of input Boston data and getting feature\_names 

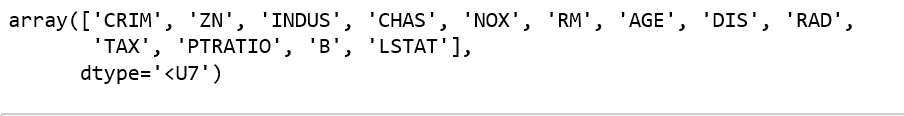
* Python3

|  |
| --- |
| boston.data.shape |

https://media.geeksforgeeks.org/wp-content/uploads/boston1.png

* Python3

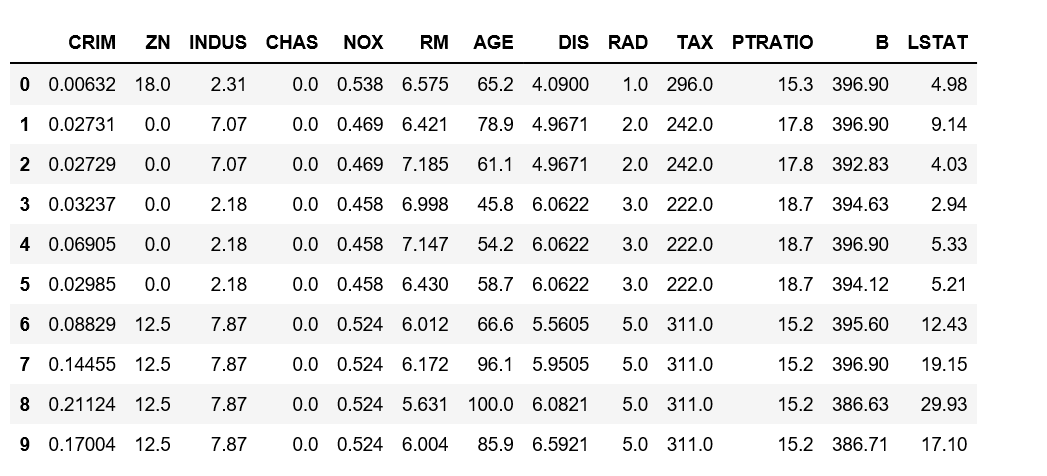
|  |
| --- |
| boston.feature\_names |



Converting data from nd-array to data frame and adding feature names to the data 

* Python3

|  |
| --- |
| data = pd.DataFrame(boston.data)  data.columns = boston.feature\_names    data.head(10) |



Adding ‘Price’ column to the dataset 

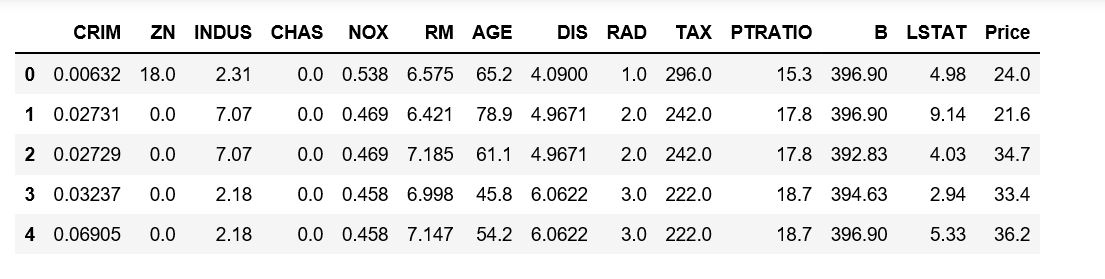
* Python3

|  |
| --- |
| # Adding 'Price' (target) column to the data  boston.target.shape |



* Python3

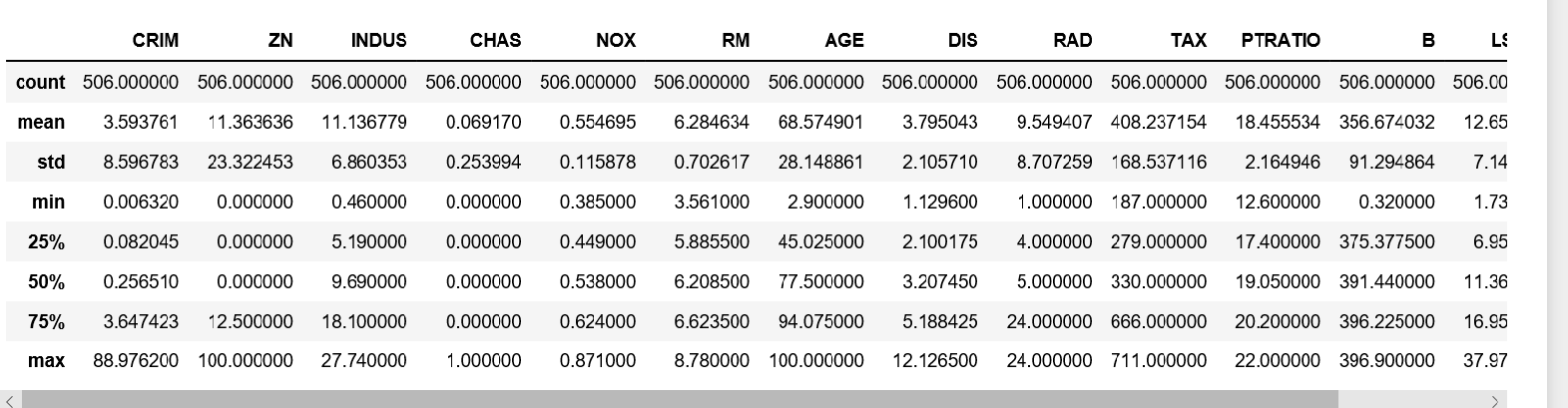
|  |
| --- |
| data['Price'] = boston.target  data.head() |



Description of Boston dataset 

* Python3

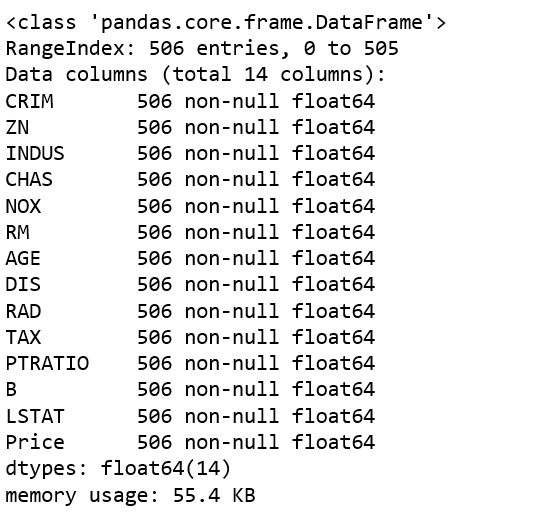
|  |
| --- |
| data.describe() |



Info of Boston Dataset 

* Python3

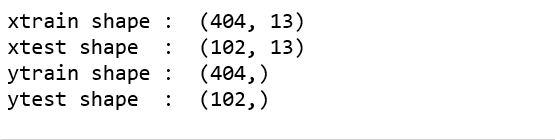
|  |
| --- |
| data.info() |



Getting input and output data and further splitting data to training and testing dataset. 

* Python3

|  |
| --- |
| # Input Data  x = boston.data    # Output Data  y = boston.target      # splitting data to training and testing dataset.    #from sklearn.cross\_validation import train\_test\_split  #the submodule cross\_validation is renamed and reprecated to model\_selection  from sklearn.model\_selection import train\_test\_split    xtrain, xtest, ytrain, ytest = train\_test\_split(x, y, test\_size =0.2,                                                      random\_state = 0)    print("xtrain shape : ", xtrain.shape)  print("xtest shape  : ", xtest.shape)  print("ytrain shape : ", ytrain.shape)  print("ytest shape  : ", ytest.shape) |



Applying Linear Regression Model to the dataset and predicting the prices. 

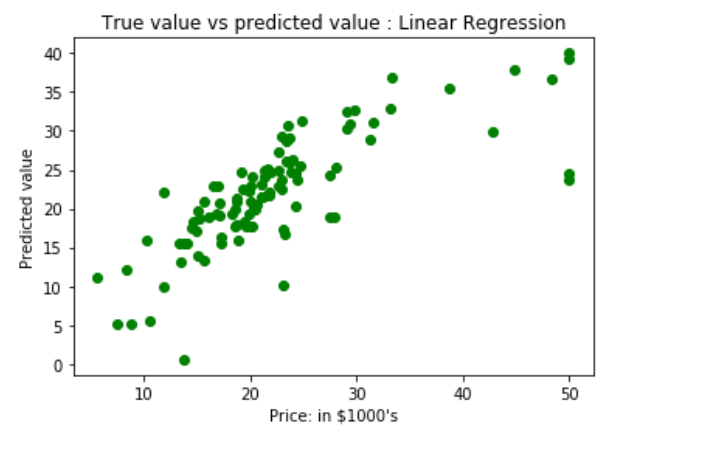
* Python3

|  |
| --- |
| # Fitting Multi Linear regression model to training model  from sklearn.linear\_model import LinearRegression  regressor = LinearRegression()  regressor.fit(xtrain, ytrain)    # predicting the test set results  y\_pred = regressor.predict(xtest) |

Plotting Scatter graph to show the prediction results – ‘ytrue’ value vs ‘y\_pred’ value 

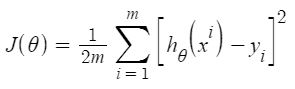
* Python3

|  |
| --- |
| # Plotting Scatter graph to show the prediction  # results - 'ytrue' value vs 'y\_pred' value  plt.scatter(ytest, y\_pred, c = 'green')  plt.xlabel("Price: in $1000's")  plt.ylabel("Predicted value")  plt.title("True value vs predicted value : Linear Regression")  plt.show() |



Results of Linear Regression i.e. Mean Squared Error. 

|  |
| --- |
| Python3  # Results of Linear Regression.  from sklearn.metrics import mean\_squared\_error  mse = mean\_squared\_error(ytest, y\_pred)  print("Mean Square Error : ", mse) |

https://media.geeksforgeeks.org/wp-content/uploads/boston11.png

As per the result, our model is only 66.55% accurate. So, the prepared model is not very good for predicting housing prices. One can improve the prediction results using many other possible machine learning algorithms and techniques.