**Predicting Boston Housing Prices**

by

Aastha Sood 16bce1104

Shambhavi Parashar 16bce1099

Snigdha Gupta 15BCE1087

A project report submitted to

**Dr. PATTABIRAMAN V**

**SCHOOL OF COMPUTING SCIENCE AND ENGINEERING**

in partial fulfilment of the requirements for the course of

**CSE3020 –DATA VISUALISATION**

in

**B.Tech. (Computer Science Engineering)**

****

**VIT UNIVERSITY, CHENNAI**

**Vandalur – Kelambakkam Road**

**Chennai – 600127**

**NOVEMBER 2018**

**CERTIFICATE**

Certified that this project report entitled “**Predicting Boston Housing Prices”** is a bonafide work of Aastha Sood 16bce1104, Shambhavi Parashar 16bce1099 ,Snigdha Gupta 15BCE1087 who carried out the “J”-Project work under my supervision and guidance for CSE3020-Data Visualisation .

**Dr. Pattabiraman V**

**SCHOOL OF COMPUTING SCIENCE AND ENGINEERING**

VIT University, Chennai

Chennai – 600 127.

**ABSTRACT**

We have evaluated the performance and predictive power of 3 models that has

been trained and tested on data collected from homes in suburbs of Boston,

Massachusetts. A model trained on this data that is seen as a good fit could

then be used to make certain predictions about a home — in particular, its

monetary value. This model would prove to be invaluable for someone like a

real estate agent who could make use of such information on a daily basis.

To check the level of error of a model, we can Mean Squared Error. It is one of

the procedure to measures the average of the squares of error. Basically, it will

check the difference between actual value and the predicted value. For the full

theory, you can always search it online. To use it, we use the mean squared

error function of scikit-learn by running this snippet of code.

**ACKNOWLEDGEMENT**

We wish to express our sincere thanks and deep sense of gratitude to our project faculty, **Dr. Pattabiraman V,** for his consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

Finally, we would like to thank our deemed university, VIT Chennai, for providing us with the opportunity and facilities which ensured this project’s completion.

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **TOPIC** | **PAGE** |
| **1** | **INTRODUCTION** | **6-7** |
| **2** | **IOT BAISED PRICISION AGRICULTURE SYSTEM DESIGN** | **8-17** |
| **3** | SOFTWARE IMPLEMENTATION | **18-25** |
| **4**  **5** | CONCLUSION AND FUTURE WORK  REFERENCES | **26-27**  **28-19** |

1. **INTRODUCTION**

The Boston Housing Dataset consists of price of houses in various places in Boston. Alongside with price, the dataset also provide information such as Crime (CRIM), areas of non-retail business in the town (INDUS), the age of people who own the house (AGE), and there are many other attributes. Number of rooms in the house refers to the size of the house. More rooms has a house, the higher should be its price. Percentage of homeowners in the neighborhood considered "lower class" (working poor) refers to the number of working poor people among all people in the neighborhood. This feature might refer to how safe is the house. According to official statistics provided by the police, courts and the government, in countries like Britain and the USA the working class, the young and some minority ethnic groups are more likely to commit crimes than the middle class, the elderly, females and whites. So the higher percentage of working poor in the neighborhood, the lower should be the price. Ratio of students to teachers in primary and secondary schools is hard to predict without the knowledge of the topic. Richer schools (or district) has higher budgets for salary so lower ratio of students to teachers. Another guess is choosing the school by the amount of learners in a class, people would prefer the one with less learners and this school might be in higher demand. So higher ratio of students to teachers, the lower price of the houses.

1. **DATASET USED**

In this report we detail the machine learning (ML) models we implemented to accurately predict the housing prices in Boston suburbs. The dataset for this experiment is accessed from the UCI Machine Learning repository via <https://archive.ics.uci.edu/ml/datasets/Housing>.

The dataset for this project originates from the UCI Machine Learning

Repository. The Boston housing data was collected in 1978 and each of the 506

entries represent aggregated data about 14 features for homes from various

suburbs in Boston, Massachusetts. For the purposes of this project, the

following preprocessing steps have been made to the dataset:

• 16 data points have an 'MEDV' value of 50.0. These data points likely contain

missing or censored values and have been removed.

• 1 data point has an 'RM' value of 8.78. This data point can be considered an

outlier and has been removed.

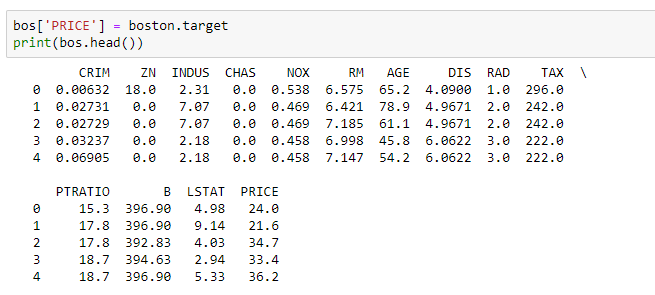
• The features 'RM', 'LSTAT', 'PTRATIO', and 'MEDV' are essential. The

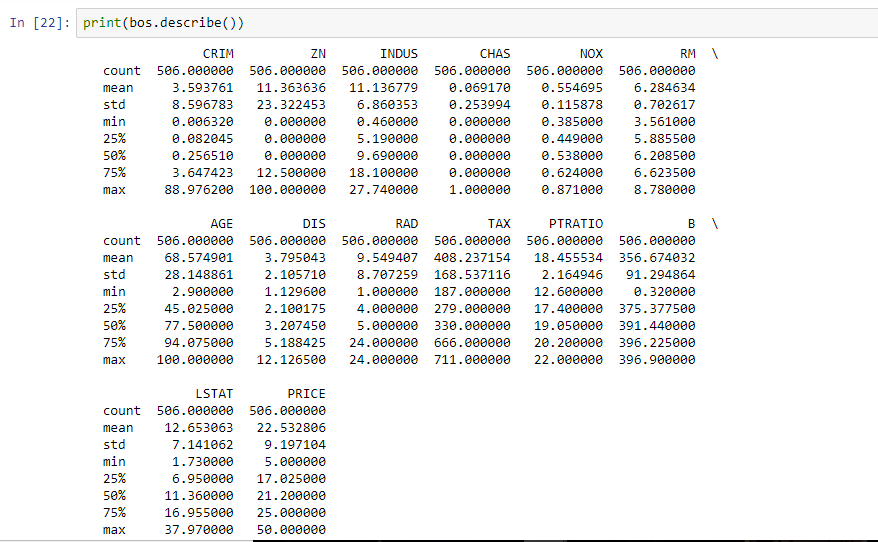
remaining non-relevant features have been excluded.

• The feature 'MEDV' has been multiplicatively scaled to account for 35 years

of market inflation.

This picture shows the top 5 data of the dataset.





**3. IMPLEMENTATION CODE**

1. **Linear Regression**

%**matplotlib** inline

**import** **numpy** **as** **np**

**import** **pandas** **as** **pd**

**import** **scipy.stats** **as** **stats**

**import** **matplotlib.pyplot** **as** **plt**

**import** **sklearn**

**import** **statsmodels.api** **as** **sm**

**import** **sklearn.cross\_validation**

**from** **sklearn.linear\_model** **import** LinearRegression

**import** **seaborn** **as** **sns**

sns.set\_style("whitegrid")

sns.set\_context("poster")

*# special matplotlib argument for improved plots*

**from** **matplotlib** **import** rcParams

In [ ]:

**from** **sklearn.datasets** **import** load\_boston

boston = load\_boston()

In [ ]:

print(boston.keys())

In [ ]:

print(boston.data.shape)

In [ ]:

print(boston.feature\_names)

In [17]:

print(boston.DESCR)

Boston House Prices dataset

===========================

Notes

------

Data Set Characteristics:

:Number of Instances: 506

:Number of Attributes: 13 numeric/categorical predictive

:Median Value (attribute 14) is usually the target

:Attribute Information (in order):

- 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 nitric 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 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

- B 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town

- LSTAT % lower status of the population

- MEDV Median value of owner-occupied homes in $1000's

:Missing Attribute Values: None

:Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.

http://archive.ics.uci.edu/ml/datasets/Housing

This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic

prices and the demand for clean air', J. Environ. Economics & Management,

vol.5, 81-102, 1978. Used in Belsley, Kuh & Welsch, 'Regression diagnostics

...', Wiley, 1980. N.B. Various transformations are used in the table on

pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that address regression

problems.

\*\*References\*\*

- Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources of Collinearity', Wiley, 1980. 244-261.

- Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the Tenth International Conference of Machine Learning, 236-243, University of Massachusetts, Amherst. Morgan Kaufmann.

- many more! (see http://archive.ics.uci.edu/ml/datasets/Housing)

In [18]:

bos = pd.DataFrame(boston.data)

print(bos.head())

0 1 2 3 4 5 6 7 8 9 10 \

0 0.00632 18.0 2.31 0.0 0.538 6.575 65.2 4.0900 1.0 296.0 15.3

1 0.02731 0.0 7.07 0.0 0.469 6.421 78.9 4.9671 2.0 242.0 17.8

2 0.02729 0.0 7.07 0.0 0.469 7.185 61.1 4.9671 2.0 242.0 17.8

3 0.03237 0.0 2.18 0.0 0.458 6.998 45.8 6.0622 3.0 222.0 18.7

4 0.06905 0.0 2.18 0.0 0.458 7.147 54.2 6.0622 3.0 222.0 18.7

11 12

0 396.90 4.98

1 396.90 9.14

2 392.83 4.03

3 394.63 2.94

4 396.90 5.33

In [19]:

bos.columns = boston.feature\_names

print(bos.head())

CRIM ZN INDUS CHAS NOX RM AGE DIS RAD TAX \

0 0.00632 18.0 2.31 0.0 0.538 6.575 65.2 4.0900 1.0 296.0

1 0.02731 0.0 7.07 0.0 0.469 6.421 78.9 4.9671 2.0 242.0

2 0.02729 0.0 7.07 0.0 0.469 7.185 61.1 4.9671 2.0 242.0

3 0.03237 0.0 2.18 0.0 0.458 6.998 45.8 6.0622 3.0 222.0

4 0.06905 0.0 2.18 0.0 0.458 7.147 54.2 6.0622 3.0 222.0

PTRATIO B LSTAT

0 15.3 396.90 4.98

1 17.8 396.90 9.14

2 17.8 392.83 4.03

3 18.7 394.63 2.94

4 18.7 396.90 5.33

In [20]:

print(boston.target.shape)

(506,)

In [21]:

bos['PRICE'] = boston.target

print(bos.head())

CRIM ZN INDUS CHAS NOX RM AGE DIS RAD TAX \

0 0.00632 18.0 2.31 0.0 0.538 6.575 65.2 4.0900 1.0 296.0

1 0.02731 0.0 7.07 0.0 0.469 6.421 78.9 4.9671 2.0 242.0

2 0.02729 0.0 7.07 0.0 0.469 7.185 61.1 4.9671 2.0 242.0

3 0.03237 0.0 2.18 0.0 0.458 6.998 45.8 6.0622 3.0 222.0

4 0.06905 0.0 2.18 0.0 0.458 7.147 54.2 6.0622 3.0 222.0

PTRATIO B LSTAT PRICE

0 15.3 396.90 4.98 24.0

1 17.8 396.90 9.14 21.6

2 17.8 392.83 4.03 34.7

3 18.7 394.63 2.94 33.4

4 18.7 396.90 5.33 36.2

In [22]:

print(bos.describe())

CRIM ZN INDUS CHAS NOX RM \

count 506.000000 506.000000 506.000000 506.000000 506.000000 506.000000

mean 3.593761 11.363636 11.136779 0.069170 0.554695 6.284634

std 8.596783 23.322453 6.860353 0.253994 0.115878 0.702617

min 0.006320 0.000000 0.460000 0.000000 0.385000 3.561000

25% 0.082045 0.000000 5.190000 0.000000 0.449000 5.885500

50% 0.256510 0.000000 9.690000 0.000000 0.538000 6.208500

75% 3.647423 12.500000 18.100000 0.000000 0.624000 6.623500

max 88.976200 100.000000 27.740000 1.000000 0.871000 8.780000

AGE DIS RAD TAX PTRATIO B \

count 506.000000 506.000000 506.000000 506.000000 506.000000 506.000000

mean 68.574901 3.795043 9.549407 408.237154 18.455534 356.674032

std 28.148861 2.105710 8.707259 168.537116 2.164946 91.294864

min 2.900000 1.129600 1.000000 187.000000 12.600000 0.320000

25% 45.025000 2.100175 4.000000 279.000000 17.400000 375.377500

50% 77.500000 3.207450 5.000000 330.000000 19.050000 391.440000

75% 94.075000 5.188425 24.000000 666.000000 20.200000 396.225000

max 100.000000 12.126500 24.000000 711.000000 22.000000 396.900000

LSTAT PRICE

count 506.000000 506.000000

mean 12.653063 22.532806

std 7.141062 9.197104

min 1.730000 5.000000

25% 6.950000 17.025000

50% 11.360000 21.200000

75% 16.955000 25.000000

max 37.970000 50.000000

In [23]:

X = bos.drop('PRICE', axis = 1)

Y = bos['PRICE']

In [24]:

X\_train, X\_test, Y\_train, Y\_test = sklearn.cross\_validation.train\_test\_split(X, Y, test\_size = 0.33, random\_state = 5)

print(X\_train.shape)

print(X\_test.shape)

print(Y\_train.shape)

print(Y\_test.shape)

(339, 13)

(167, 13)

(339,)

(167,)

In [45]:

**from** **sklearn.linear\_model** **import** LinearRegression

lm = LinearRegression()

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

Y\_pred = lm.predict(X\_test)

plt.scatter(Y\_test, Y\_pred)

plt.xlabel("Prices: $Y\_i$")

plt.ylabel("Predicted prices: $\hat**{Y}**\_i$")

plt.title("Prices vs Predicted prices: $Y\_i$ vs $\hat**{Y}**\_i$")

Out[45]:

Text(0.5,1,'Prices vs Predicted prices: $Y\_i$ vs $\\hat{Y}\_i$')

In [46]:

mse = sklearn.metrics.mean\_squared\_error(Y\_test, Y\_pred)

print(mse)

28.541367275618242

In [47]:

print(score)

0.74

**b) Multilayer Perceptron**

**from** **sklearn.neural\_network** **import** MLPRegressor

**from** **sklearn.datasets** **import** load\_boston

**from** **sklearn.preprocessing** **import** StandardScaler

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

**from** **sklearn.model\_selection** **import** GridSearchCV

**from** **sklearn.metrics** **import** mean\_squared\_error

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

In [2]:

boston = load\_boston()

In [3]:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(boston.data, boston.target)

In [4]:

scaler = StandardScaler()

scaler.fit(X\_train)

X\_train = scaler.transform(X\_train)

X\_test = scaler.transform(X\_test)

In [5]:

tuned\_parameters = [{'hidden\_layer\_sizes': [1,2,3,4,5,6,7,8,9,10,20,30,40],

'activation': ['relu'],

'solver':['lbfgs'], 'alpha':[0.0001],

'batch\_size':['auto'], 'learning\_rate':['constant'],

'learning\_rate\_init':[0.001], 'max\_iter':[500]}]

rgr = GridSearchCV(MLPRegressor(), tuned\_parameters, cv=5)

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

Out[5]:

GridSearchCV(cv=5, error\_score='raise',

estimator=MLPRegressor(activation='relu', alpha=0.0001, batch\_size='auto', beta\_1=0.9,

beta\_2=0.999, early\_stopping=False, epsilon=1e-08,

hidden\_layer\_sizes=(100,), learning\_rate='constant',

learning\_rate\_init=0.001, max\_iter=200, momentum=0.9,

nesterovs\_momentum=True, power\_t=0.5, random\_state=None,

shuffle=True, solver='adam', tol=0.0001, validation\_fraction=0.1,

verbose=False, warm\_start=False),

fit\_params=None, iid=True, n\_jobs=1,

param\_grid=[{'hidden\_layer\_sizes': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40], 'activation': ['relu'], 'solver': ['lbfgs'], 'alpha': [0.0001], 'batch\_size': ['auto'], 'learning\_rate': ['constant'], 'learning\_rate\_init': [0.001], 'max\_iter': [500]}],

pre\_dispatch='2\*n\_jobs', refit=True, return\_train\_score='warn',

scoring=None, verbose=0)

In [6]:

*#mlp = MLPRegressor(hidden\_layer\_sizes=(20,), activation='logistic', solver='lbfgs', alpha=0.0001, batch\_size='auto', learning\_rate='constant', learning\_rate\_init=0.001, max\_iter=500)*

*#mlp.fit(X\_train,y\_train)*

train\_mse = mean\_squared\_error(y\_train, rgr.predict(X\_train))

test\_mse = mean\_squared\_error(y\_test, rgr.predict(X\_test))

In [7]:

print(rgr.best\_params\_)

print(rgr.best\_score\_)

print("Train MSE:", np.round(train\_mse,2))

print("Test MSE:", np.round(test\_mse,2))

{'activation': 'relu', 'alpha': 0.0001, 'batch\_size': 'auto', 'hidden\_layer\_sizes': 7, 'learning\_rate': 'constant', 'learning\_rate\_init': 0.001, 'max\_iter': 500, 'solver': 'lbfgs'}

0.8284261954960827

Train MSE: 4.89

Test MSE: 19.97

**c) Decision Tree**

**import** **numpy** **as** **np**

**import** **pandas** **as** **pd**

**from** **sklearn.cross\_validation** **import** ShuffleSplit

C:\Users\AASTHA SOOD\Anaconda3\lib\site-packages\sklearn\cross\_validation.py:41: DeprecationWarning: This module was deprecated in version 0.18 in favor of the model\_selection module into which all the refactored classes and functions are moved. Also note that the interface of the new CV iterators are different from that of this module. This module will be removed in 0.20.

"This module will be removed in 0.20.", DeprecationWarning)

In [2]:

%**matplotlib** inline

In [10]:

data = pd.read\_csv('boston.csv')

prices = data['MV']

features = data.drop('MV', axis = 1)

data.head(5)

Out[10]:

|  | **CRIM** | **ZN** | **INDUS** | **CHAS** | **NOX** | **RM** | **AGE** | **DIS** | **RAD** | **TAX** | **PT** | **B** | **LSTAT** | **MV** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0.00632 | 18.0 | 2.31 | 0 | 0.538 | 6.575 | 65.199997 | 4.0900 | 1 | 296 | 15.300000 | 396.899994 | 4.98 | 24.000000 |
| **1** | 0.02731 | 0.0 | 7.07 | 0 | 0.469 | 6.421 | 78.900002 | 4.9671 | 2 | 242 | 17.799999 | 396.899994 | 9.14 | 21.600000 |
| **2** | 0.02729 | 0.0 | 7.07 | 0 | 0.469 | 7.185 | 61.099998 | 4.9671 | 2 | 242 | 17.799999 | 392.829987 | 4.03 | 34.700001 |
| **3** | 0.03237 | 0.0 | 2.18 | 0 | 0.458 | 6.998 | 45.799999 | 6.0622 | 3 | 222 | 18.700001 | 394.630005 | 2.94 | 33.400002 |
| **4** | 0.06905 | 0.0 | 2.18 | 0 | 0.458 | 7.147 | 54.200001 | 6.0622 | 3 | 222 | 18.700001 | 396.899994 | 5.33 | 36.200001 |

In [7]:

*# Minimum price of the data*

minimum\_price = np.min(prices)

*# Maximum price of the data*

maximum\_price = np.max(prices)

*# Mean price of the data*

mean\_price = np.mean(prices)

*# Median price of the data*

median\_price = np.median(prices)

*# Standard deviation of prices of the data*

std\_price = np.std(prices)

In [8]:

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

clr = ['blue', 'green', 'red']

In [11]:

fig, axs = plt.subplots(ncols=3,figsize=(15,3))

plt.figure(1)

**for** i, var **in** enumerate(['RM', 'LSTAT', 'PT']):

plt.subplot(131 + i)

sns.distplot(data[var], color = clr[i])

plt.axvline(data[var].mean(), color=clr[i], linestyle='solid', linewidth=2)

plt.axvline(data[var].median(), color=clr[i], linestyle='dashed', linewidth=2)

C:\Users\AASTHA SOOD\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[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 result.

return np.add.reduce(sorted[indexer] \* weights, axis=axis) / sumval

In [12]:

fig, axs = plt.subplots(ncols=3,figsize=(15,3))

plt.figure(1)

**for** i, var **in** enumerate(['RM', 'LSTAT', 'PT']):

plt.subplot(131 + i)

**if** i==0:

sns.distplot(data[var], color = clr[i])

plt.axvline(data[var].mean(), color=clr[i], linestyle='solid', linewidth=2)

plt.axvline(data[var].median(), color=clr[i], linestyle='dashed', linewidth=2)

**else**:

sns.distplot(np.log(data[var]), color = clr[i])

plt.axvline(np.log(data[var]).mean(), color=clr[i], linestyle='solid', linewidth=2)

plt.axvline(np.log(data[var]).median(), color=clr[i], linestyle='dashed', linewidth=2)

C:\Users\AASTHA SOOD\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[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 result.

return np.add.reduce(sorted[indexer] \* weights, axis=axis) / sumval

In [15]:

fig, axs = plt.subplots(ncols=3,figsize=(15,3))

**for** i, var **in** enumerate(['RM', 'LSTAT', 'PTR']):

lm = sns.regplot(data[var], prices, ax = axs[i], color=clr[i])

lm.set(ylim=(0, **None**))

C:\Users\AASTHA SOOD\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[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 result.

return np.add.reduce(sorted[indexer] \* weights, axis=axis) / sumval

**---------------------------------------------------------------------------**

**KeyError** Traceback (most recent call last)

**~\Anaconda3\lib\site-packages\pandas\core\indexes\base.py** in get\_loc**(self, key, method, tolerance)**

3077 **try:**

**-> 3078 return** self**.**\_engine**.**get\_loc**(**key**)**

3079 **except** KeyError**:**

**pandas\\_libs\index.pyx** in pandas.\_libs.index.IndexEngine.get\_loc**()**

**pandas\\_libs\index.pyx** in pandas.\_libs.index.IndexEngine.get\_loc**()**

**pandas\\_libs\hashtable\_class\_helper.pxi** in pandas.\_libs.hashtable.PyObjectHashTable.get\_item**()**

**pandas\\_libs\hashtable\_class\_helper.pxi** in pandas.\_libs.hashtable.PyObjectHashTable.get\_item**()**

**KeyError**: 'PTR'

During handling of the above exception, another exception occurred:

**KeyError** Traceback (most recent call last)

**<ipython-input-15-783974289b60>** in <module>**()**

2

3 **for** i**,** var **in** enumerate**(['RM',** **'LSTAT',** **'PTR']):**

**----> 4** lm **=** sns**.**regplot**(**data**[**var**],** prices**,** ax **=** axs**[**i**],** color**=**clr**[**i**])**

5 lm**.**set**(**ylim**=(0,** **None))**

**~\Anaconda3\lib\site-packages\pandas\core\frame.py** in \_\_getitem\_\_**(self, key)**

2686 **return** self**.**\_getitem\_multilevel**(**key**)**

2687 **else:**

**-> 2688 return** self**.**\_getitem\_column**(**key**)**

2689

2690 **def** \_getitem\_column**(**self**,** key**):**

**~\Anaconda3\lib\site-packages\pandas\core\frame.py** in \_getitem\_column**(self, key)**

2693 **# get column**

2694 **if** self**.**columns**.**is\_unique**:**

**-> 2695 return** self**.**\_get\_item\_cache**(**key**)**

2696

2697 **# duplicate columns & possible reduce dimensionality**

**~\Anaconda3\lib\site-packages\pandas\core\generic.py** in \_get\_item\_cache**(self, item)**

2487 res **=** cache**.**get**(**item**)**

2488 **if** res **is** **None:**

**-> 2489** values **=** self**.**\_data**.**get**(**item**)**

2490 res **=** self**.**\_box\_item\_values**(**item**,** values**)**

2491 cache**[**item**]** **=** res

**~\Anaconda3\lib\site-packages\pandas\core\internals.py** in get**(self, item, fastpath)**

4113

4114 **if** **not** isna**(**item**):**

**-> 4115** loc **=** self**.**items**.**get\_loc**(**item**)**

4116 **else:**

4117 indexer **=** np**.**arange**(**len**(**self**.**items**))[**isna**(**self**.**items**)]**

**~\Anaconda3\lib\site-packages\pandas\core\indexes\base.py** in get\_loc**(self, key, method, tolerance)**

3078 **return** self**.**\_engine**.**get\_loc**(**key**)**

3079 **except** KeyError**:**

**-> 3080 return** self**.**\_engine**.**get\_loc**(**self**.**\_maybe\_cast\_indexer**(**key**))**

3081

3082 indexer **=** self**.**get\_indexer**([**key**],** method**=**method**,** tolerance**=**tolerance**)**

**pandas\\_libs\index.pyx** in pandas.\_libs.index.IndexEngine.get\_loc**()**

**pandas\\_libs\index.pyx** in pandas.\_libs.index.IndexEngine.get\_loc**()**

**pandas\\_libs\hashtable\_class\_helper.pxi** in pandas.\_libs.hashtable.PyObjectHashTable.get\_item**()**

**pandas\\_libs\hashtable\_class\_helper.pxi** in pandas.\_libs.hashtable.PyObjectHashTable.get\_item**()**

**KeyError**: 'PTR'

In [16]:

fig, axs = plt.subplots(ncols=3,figsize=(15,3))

**for** i, var **in** enumerate(['RM', 'LSTAT', 'PTRATIO']):

lm = sns.regplot(np.log(data[var]), prices, ax = axs[i], color=clr[i])

lm.set(ylim=(0, **None**))

C:\Users\AASTHA SOOD\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[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 result.

return np.add.reduce(sorted[indexer] \* weights, axis=axis) / sumval

**---------------------------------------------------------------------------**

**KeyError** Traceback (most recent call last)

**~\Anaconda3\lib\site-packages\pandas\core\indexes\base.py** in get\_loc**(self, key, method, tolerance)**

3077 **try:**

**-> 3078 return** self**.**\_engine**.**get\_loc**(**key**)**

3079 **except** KeyError**:**

**pandas\\_libs\index.pyx** in pandas.\_libs.index.IndexEngine.get\_loc**()**

**pandas\\_libs\index.pyx** in pandas.\_libs.index.IndexEngine.get\_loc**()**

**pandas\\_libs\hashtable\_class\_helper.pxi** in pandas.\_libs.hashtable.PyObjectHashTable.get\_item**()**

**pandas\\_libs\hashtable\_class\_helper.pxi** in pandas.\_libs.hashtable.PyObjectHashTable.get\_item**()**

**KeyError**: 'PTRATIO'

During handling of the above exception, another exception occurred:

**KeyError** Traceback (most recent call last)

**<ipython-input-16-01fb46566ab7>** in <module>**()**

2

3 **for** i**,** var **in** enumerate**(['RM',** **'LSTAT',** **'PTRATIO']):**

**----> 4** lm **=** sns**.**regplot**(**np**.**log**(**data**[**var**]),** prices**,** ax **=** axs**[**i**],** color**=**clr**[**i**])**

5 lm**.**set**(**ylim**=(0,** **None))**

**~\Anaconda3\lib\site-packages\pandas\core\frame.py** in \_\_getitem\_\_**(self, key)**

2686 **return** self**.**\_getitem\_multilevel**(**key**)**

2687 **else:**

**-> 2688 return** self**.**\_getitem\_column**(**key**)**

2689

2690 **def** \_getitem\_column**(**self**,** key**):**

**~\Anaconda3\lib\site-packages\pandas\core\frame.py** in \_getitem\_column**(self, key)**

2693 **# get column**

2694 **if** self**.**columns**.**is\_unique**:**

**-> 2695 return** self**.**\_get\_item\_cache**(**key**)**

2696

2697 **# duplicate columns & possible reduce dimensionality**

**~\Anaconda3\lib\site-packages\pandas\core\generic.py** in \_get\_item\_cache**(self, item)**

2487 res **=** cache**.**get**(**item**)**

2488 **if** res **is** **None:**

**-> 2489** values **=** self**.**\_data**.**get**(**item**)**

2490 res **=** self**.**\_box\_item\_values**(**item**,** values**)**

2491 cache**[**item**]** **=** res

**~\Anaconda3\lib\site-packages\pandas\core\internals.py** in get**(self, item, fastpath)**

4113

4114 **if** **not** isna**(**item**):**

**-> 4115** loc **=** self**.**items**.**get\_loc**(**item**)**

4116 **else:**

4117 indexer **=** np**.**arange**(**len**(**self**.**items**))[**isna**(**self**.**items**)]**

**~\Anaconda3\lib\site-packages\pandas\core\indexes\base.py** in get\_loc**(self, key, method, tolerance)**

3078 **return** self**.**\_engine**.**get\_loc**(**key**)**

3079 **except** KeyError**:**

**-> 3080 return** self**.**\_engine**.**get\_loc**(**self**.**\_maybe\_cast\_indexer**(**key**))**

3081

3082 indexer **=** self**.**get\_indexer**([**key**],** method**=**method**,** tolerance**=**tolerance**)**

**pandas\\_libs\index.pyx** in pandas.\_libs.index.IndexEngine.get\_loc**()**

**pandas\\_libs\index.pyx** in pandas.\_libs.index.IndexEngine.get\_loc**()**

**pandas\\_libs\hashtable\_class\_helper.pxi** in pandas.\_libs.hashtable.PyObjectHashTable.get\_item**()**

**pandas\\_libs\hashtable\_class\_helper.pxi** in pandas.\_libs.hashtable.PyObjectHashTable.get\_item**()**

**KeyError**: 'PTRATIO'

In [17]:

sns.heatmap(data.corr(), square=**True**,annot=**True**)

Out[17]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x140009157f0>

In [21]:

**from** **sklearn.metrics** **import** r2\_score

**def** performance\_metric(y\_true, y\_predict):

*""" Calculates and returns the performance score between*

*true and predicted values based on the metric chosen. """*

score = r2\_score(y\_true, y\_predict)

**return** score

In [23]:

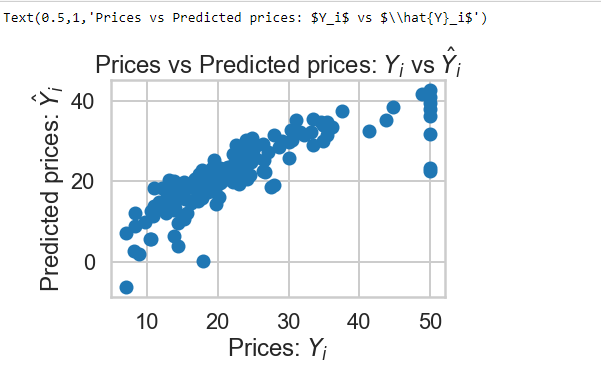
score = performance\_metric([3, -0.5, 2, 7, 4.2], [2.5, 0.0, 2.1, 7.8, 5.3])

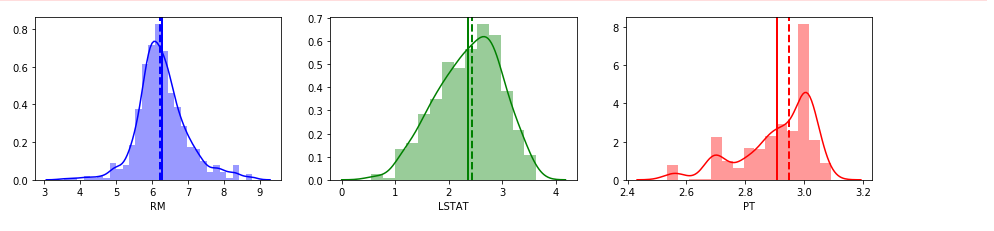
print("Model has a coefficient of determination, R^2, of **{:.3f}**.".format(score))

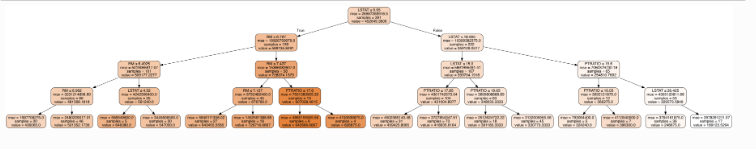
0.9228556485355649

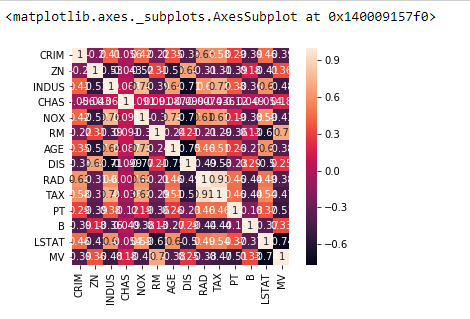
Model has a coefficient of determination, R^2, of 0.923.

**4. VISUAL OUTPUT**









**4.1 CONCLUSION**

Linear Regression gives us an accuracy of 74%. Therefore it isn’t a really great

linear model. But, as a start, it is a good way to go.

Multi Layer Perceptron gives us MSE as 19.97. This is a better model than Linear Regression.

Accuracy is highest, 92.3 for Decision Tree model.

We hence conclude that Decision Tree provides us with the best Prediction

amongst the three algorithms which we analyzed.