Predict Boston House Prices Using Python & Linear Regression

In this article, I will write a Python program that predicts the price of houses in Boston using a machine learning algorithm called **Linear Regression**. Linear regression is a linear approach to modeling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables).

Linear Regression Pros:

1. Simple to implement.
2. Used to predict numeric values.

Linear Regression Cons:

1. Prone to overfitting.
2. Cannot be used when the relation between independent and dependent variables are nonlinear.

**Start Programming:**

First I will import the dependencies, that will make this program a little easier to write. I’m importing the machine learning library **sklearn**, **numpy**, and **pandas**.

**import** **pandas** **as** **pd**  
**import** **numpy** **as** **np**  
**from** **sklearn** **import** linear\_model  
**from** **sklearn.model\_selection** **import** train\_test\_split

Next, I will load the Boston Housing Data Set from **sklearn.datasets** and print it after storing it into the variable boston by using a function/method called load\_boston(), but first I will import the library sklearn.datasets**.**This method will return a dictionary-like object, the interesting attributes are: data the data to learn, target the regression targets, DESCR the full description of the dataset,filename the physical location of Boston CSV data set, and feature\_names the name of the columns or features. Note this data set is already clean meaning there is no corrupt, inaccurate, or missing data.

*#Load the Boston Housing Data Set from sklearn.datasets and print it*  
**from** **sklearn.datasets** **import** load\_boston  
boston = load\_boston()  
print(boston)

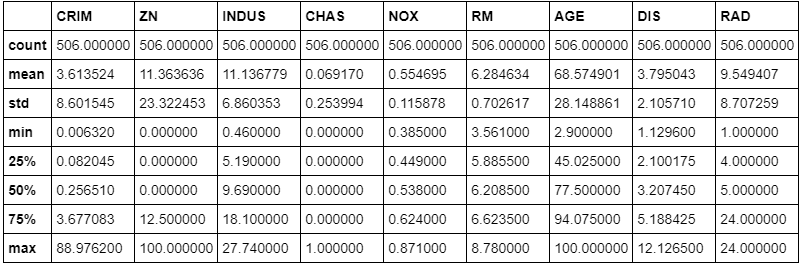
**Fig 1:** A partial image of the data returned from load\_boston() highlighting ‘data’

The above image **Fig 1,**looks a mess and is hard for me to read, so I will transform the data into something more manageable like a data frame using the pandas library. While I’m at it, I will split the data into independent variables (X’s) and dependent variable (Y)data sets. The data will be stored in df\_x for the dependent variables and df\_y for the dependent variable.

*#Transform the data set into a data frame*   
*#NOTE: boston.data = the data we want,*   
*# boston.feature\_names = the column names of the data*  
*# boston.target = Our target variable or the price of the houses*  
df\_x = pd.DataFrame(boston.data, columns = boston.feature\_names)  
df\_y = pd.DataFrame(boston.target)

I want to get some statistics from the df\_x data set like the count or the number of rows the data contains for each column, the minimum value for each column, the maximum value for each column, and the mean for each column.

*#Get some statistics from our data set, count, mean standard deviation etc.*  
df\_x.describe()



**Fig 2:** Statistics on a few columns from the df\_x data set

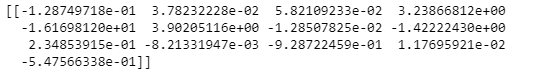
I will go ahead and initialize the Linear Regression model, split the data into 67% training and 33% testing data, and then train the model with the training data set that contains the independent variables.

*#Initialize the linear regression model*  
reg = linear\_model.LinearRegression()*#Split the data into 67% training and 33% testing data*  
*#NOTE: We have to split the dependent variables (x) and the target or independent variable (y)*  
x\_train, x\_test, y\_train, y\_test = train\_test\_split(df\_x, df\_y, test\_size=0.33, random\_state=42)*#Train our model with the training data*  
reg.fit(x\_train, y\_train)

Get the estimated coefficients for the linear regression model

*#Print the coefecients/weights for each feature/column of our model*  
print(reg.coef\_)



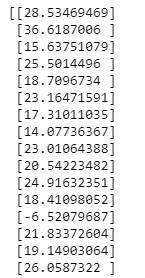


**Fig 3:** Coefficients of the linear regression model

Now that we are done training the linear regression model and looking at the coefficients that describe the linear function, let’s print the model's predictions (what it thinks the values will be for houses) on the test data.

*#print our price predictions on our test data*  
y\_pred = reg.predict(x\_test)  
print(y\_pred)





**Fig 4:** A small sample of the predicted values

I want to know what was the actual values for that test data set, so I will print those values to the screen, but first I will print at least one row from the model's prediction, just to make it a little easier to compare data.

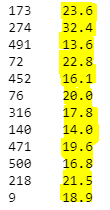
*#Print the the prediction for the third row of our test data actual price = 13.6*  
y\_pred[2]*#print the actual price of houses from the testing data set*  
y\_test[0]





**Fig 4.1:** The printed predicted value for row 3 of the testing data set





**Fig 4.2:** The printed actual values of the testing data set highlighted

By looking at the predicted values that the model came up with, and the actual values of the testing data set, it looks like the model is pretty good at making predictions. It’s not exact, but it is pretty close and much better than guessing. But I want to check the model performance using a more mathematical approach.

To check the model's performance/accuracy I will use a metric called **mean squared error (MSE)**. This measurement is simple to implement and easy to understand. The MSE is a measure of the quality of an estimator — it is always non-negative, and values closer to zero indicate a better fit. Usually you want to evaluate your model with other metrics as well to truly get an idea of how well your model performs. I’ll do this two different ways, one using **numpy**and the other using **sklearn.metrics**.

*# Two different ways to check model performance/accuracy using,*  
*# mean squared error which tells you how close a regression line is to a set of points.*  
  
*# 1. Mean squared error by numpy*  
print(np.mean((y\_pred-y\_test)\*\*2))  
  
*# 2. Mean squared error by sklearn*   
*# Resource: https://stackoverflow.com/questions/42453875/precision-score-and-accuracy-score-showing-value-error?rq=1*  
**from** **sklearn.metrics** **import** mean\_squared\_error  
print(mean\_squared\_error(y\_test, y\_pred))

The mean squared error function returns a value of about **20.72.** This is really good! If our model had predicted the exact value as the actual values then the mean squared error function would have returned a value of 0.