**Machine Learning : 'Regression' - Day 1**

* Published on Published onFebruary 22, 2018

[[](https://www.linkedin.com/in/panchalshivam/)](https://www.linkedin.com/in/panchalshivam/)

**[Shivam Panchal](https://www.linkedin.com/in/panchalshivam/)**

 21

*  16

[Write an article](https://www.linkedin.com/post/new)

Hi everyone!

This is the first post of my new series of articles which will be specifically focusing on Machine LearningAlgorithms and its practical implementation in Python using real world data sets.

I assume that you have basic knowledge of python programming and have used the basic libraries such as Numpy, Scipy, Pandas, Matplotlib and Sklearn.

If you haven't, I am attaching the links here for you!

**Numpy**: http://bit.ly/2k4igLd

**Scipy**: http://bit.ly/1oyJqYk

**Pandas**: http://bit.ly/2qs1lAJ

**Matplotlib**: http://bit.ly/2EMuVNG

**Sklearn**: http://bit.ly/2j049C4

I wanted to provide a quick introduction to building models in Python, and what better way to start than one of the very basic models, linear regression?

This will be the first post about machine learning and I plan to write about more complex models in the future. Stay tuned! But for right now, let’s focus on linear regression.

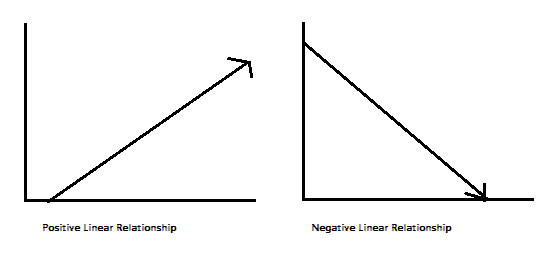


I want to focus on the concept of linear regression and mainly on the implementation of it in Python.

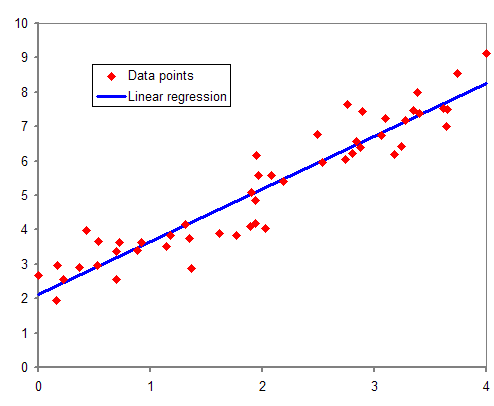
Here we go!!

**Chapter 1 : Regression**

[Linear regression](https://en.wikipedia.org/wiki/Linear_regression) is a statistical model that examines the linear relationship between two (Simple Linear Regression ) or more (Multiple Linear Regression) variables — a dependent variable and independent variable(s). Linear relationship basically means that when one (or more) independent variables increases (or decreases), the dependent variable increases (or decreases) too.



As you can see, a linear relationship can be positive (independent variable goes up, dependent variable goes up) or negative (independent variable goes up, dependent variable goes down) or other. Like I said, I will focus on the implementation of regression models in Python, so I don’t want to delve too much into the math under the regression hood, but I will write a little bit about it.



**A Little Bit About the Math**

A relationship between variables Y and X is represented by this equation:

Y`i = mX + b

In this equation, *Y* is the dependent variable — or the variable we are trying to predict or estimate; X is the independent variable — the variable we are using to make predictions; m is the slope of the regression line — it represent the effect *X* has on *Y*.

This is Simple Linear Regression (SLR). In a SLR model, we build a model based on data — the slope and Y-intercept derive from the data; furthermore, we don’t need the relationship between *X* and *Y*to be exactly linear. SLR models also include the errors in the data (also known as residuals). I won’t go too much into it now, maybe in a later post, but residuals are basically the differences between the true value of Y and the predicted/estimated value of Y. It is important to note that in a linear regression, we are trying to predict a continuous variable. In a regression model, we are trying to minimize these errors by finding the “line of best fit” — the regression line from the errors would be minimal. We are trying **to minimize** **the distance of the red dots from the blue line** — as close to zero as possible. It is related to (or equivalent to) minimizing the [mean squared error (MSE)](https://en.wikipedia.org/wiki/Mean_squared_error) or the sum of [squares of error (SSE)](https://en.wikipedia.org/wiki/Residual_sum_of_squares), also called the “residual sum of squares.” (RSS) but this might be beyond the scope of this blog post :-)

In most cases, we will have more than one independent variable — we’ll have multiple variables; it can be as little as two independent variables and up to hundreds (or theoretically even thousands) of variables. in those cases we will use a Multiple Linear Regression model (MLR). The regression equation is pretty much the same as the simple regression equation, just with more variables:

Y’i = b0 + b1X1i + b2X2i

This concludes the math portion of this post :) Ready to get to implementing it in Python?

**Project 1: Predicting Boston Housing Prices**

Here we perform a simple regression analysis on the Boston housing data, exploring simple types of linear regression model.

I will use [Boston Housing data set](https://archive.ics.uci.edu/ml/datasets/Housing), the data set  contains information about the housing values in suburbs of Boston. This dataset was originally taken from the StatLib library which is maintained at Carnegie Mellon University and is now available on the[UCI Machine Learning Repository](http://archive.ics.uci.edu/ml/). UCI machine learning repository contains many interesting data sets, I encourage you to go through it.

We will be using sklearn to import the boston data as it contains a bunch of useful datasets to practise along and we will also import Linear Regression Model from sklearn. Althoug, you can also code your Linear Regression model as a function or class in python and its easy.

from sklearn.datasets import load\_boston

data = load\_boston()

Print a histogram of the quantity to predict: price

import matplotlib.pyplot as plt

%matplotlib inline

plt.style.use('bmh')

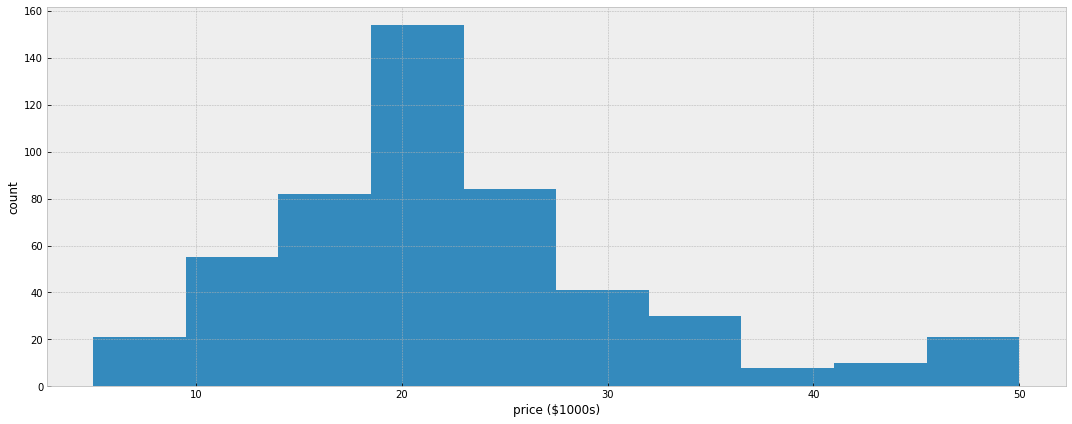
plt.figure(figsize=(15, 6))

plt.hist(data.target)

plt.xlabel('price ($1000s)')

plt.ylabel('count')

plt.tight\_layout()



Print the join histogram for each feature

for index, feature\_name in enumerate(data.feature\_names):

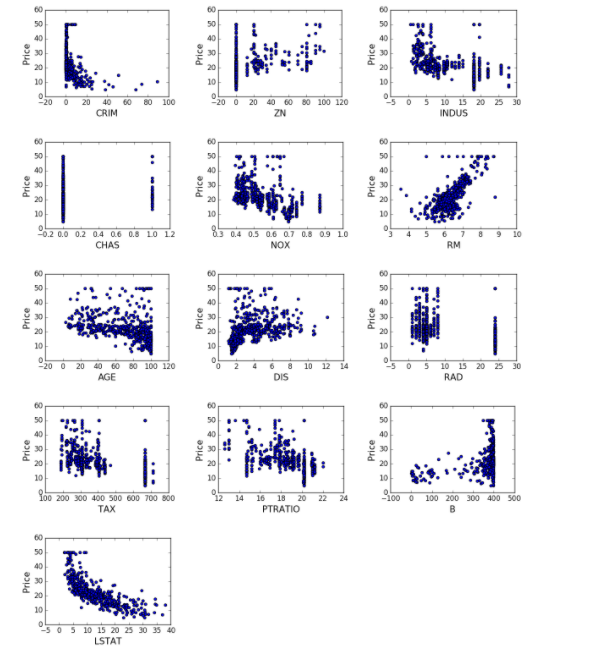
plt.figure(figsize=(4, 3))

plt.scatter(data.data[:, index], data.target)

plt.ylabel('Price', size=15)

plt.xlabel(feature\_name, size=15)

plt.tight\_layout()



Prediction

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

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

from sklearn.linear\_model import LinearRegression

clf = LinearRegression()

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

predicted = clf.predict(X\_test)

expected = y\_test

plt.figure(figsize=(15, 6))

plt.scatter(expected, predicted)

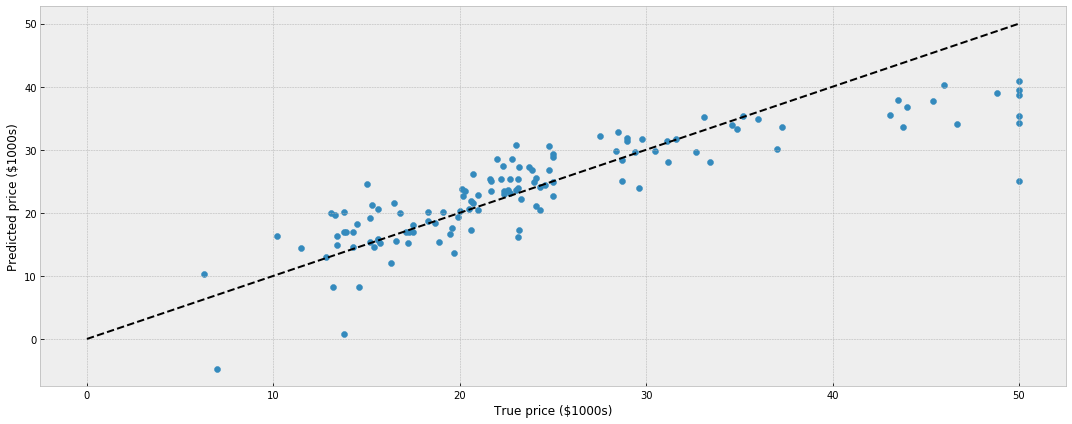
plt.plot([0, 50], [0, 50], '--k')

plt.axis('tight')

plt.xlabel('True price ($1000s)')

plt.ylabel('Predicted price ($1000s)')

plt.tight\_layout()



Print the error rate

import numpy as np

print("RMS: %r " % np.sqrt(np.mean((predicted - expected) \*\* 2)))

RMS: 5.3588890300591467

So, this was the simplest implmentation of Linear Regression to predict the house prices and validate your model, in the next post, I will come up with a strong implmentation of predictive model for the same problem.

Stay Tuned!! Surprrises Coming!!

Feel free to criticise.