Understanding Linear Regression with Python: Practical Guide 2

Author, Mayank Tripathi

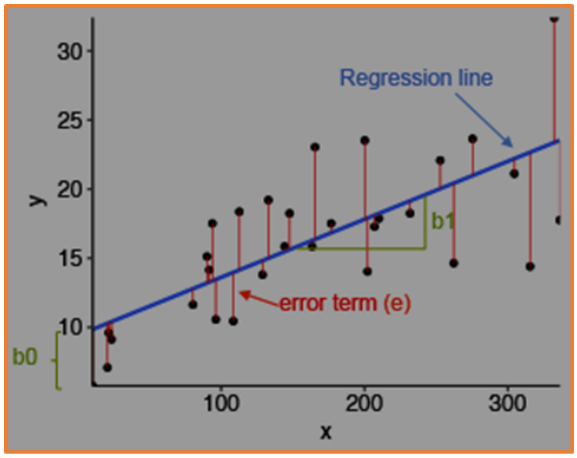
A Data Science Foundation White Paper

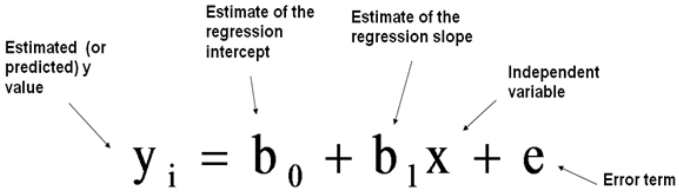
April 2020

---

www.datascience.foundation

Data Science Foundation
Data Science Foundation, Atlantic Business Centre, Atlantic Street, Altrincham, WA14 5NQ
Tel: 0161 926 3641 Email: admin@datascience.foundation Web: www.datascience.foundation
Registered in England and Wales 4th June 2015, Registered Number 9624670





This second practical guide will help you to brush-up your knowledge and go to the top of the class. Let us start with a simple statistical algorithm known as **Linear Regression** and begin to develop our skills by understanding the principles that underpin how it works.

Before we pe deep into the theories surrounding Linear Regression we need to start with a clear view on why it is needed in the first place.

So, what does the term Regression mean?

Understanding a Regression Problem

Example 1: Suppose I am planning a road trip with two of my best friends from Nashville, TN to Las Vegas. To make the trip enjoyable and stress-free and to ensure that we arrive in one piece, I would be well advised to create a schedule. The first thing I will create as part of this is a budget, how much money should I allocate for gas (petrol / diesel), for food and for accommodation along the way. My approach to this problem would be to find a way to estimate the amount of money needed, based on the distance we are travelling, and the number of stops required. We can all appreciate that the larger the distance travelled, the more expensive it would be.

Example 2: You are asked to examine the relationship between the age and the price of used cars sold in the previous year by a car dealership. We all are aware of the general rule that as a car age increases price goes down, this is an example of a negative relationship between car price (Y) and car age (X).

Example 3: Suppose we are planning to buy a new House, for this we will gather information such as sale prices from various property dealers along with information that would include number of bedrooms, number of bathrooms, location etc. Based on all this data, we can then, if we are presented with the same

data but not the sale price of another property, estimate the sale price within a reasonably accurate range. And if given the asking price of a house on the market, appreciate if it is a good deal or if it is overpriced.

Other examples include: Sleep hours Vs Test scores; Experience vs Salary; etc.

The point of these examples is that once we have established a relationship between 2 or more variables or have identified a statistically significant relationship, we can then proceed to forecast, predict, or estimate the value for new observation(s).

And the best thing about this? You are already doing it, day in day out. You just didn't realize that it was called linear regression.

Meaning of Regression & Linear Regression

Regression analysis attempts to predict one dependent variable or target (usually denoted by Y) and a series of other independent variables or features (usually denoted by X).

Linear regression is a statistical approach for modelling the relationship between a dependent variable with a set of explanatory variables. Linear regression is a common Statistical Data Analysis technique.

Problem-solving using linear regression has so many applications in business, social, biological, and many other areas.

Types of Linear Regression

In general, there are two types of linear regression

- Simple Linear Regression
- Multiple Linear Regression

Simple Linear Regression allows us to study the relationship between two variables.

In simple linear regression a single independent variable is used to predict the value of a dependent variable.

I.e. One independent variable (X) and One Dependent variable (Y).

This can be denoted by

$$y_i = b_0 + b_1 x + e$$

In school we became familiar with equations like the one shown below, then how it is different from the equation above?

$$y = m\,x + c$$

Well both are the same type of equation, we just changed the name of the variable and added something extra, the 'e' to minimize the chance of error.

Where **m** was slope, and c was Intercept. In the first equation **b0** is the intercept, and **b1** is the slope.

Slope direction

The slope of a line can be positive, negative, zero or undefined.

Positive slope: y increases as x increases, so the line slopes upwards to the right.

Negative slope: y decreases as x increases, so the line slopes downwards to the right. If you remember from the previous examples, we have seen an example of this, where the Age of the Car increases, the price decreases.

Zero slope: y does not change as x increases, so the line remains horizontal. The slope of any horizontal line is always zero.

Undefined slope: When the line is exactly vertical, it does not have a defined slope. The two x coordinates are the same, so the difference is zero.

Multiple Linear Regression allows us to study the relationship between three or more variables.

In Multiple Linear Regression two or more independent variables are used to predict the value of a dependent variable.

I.e. Two or more independent variables (X1, X2, X3, ...) and one Dependent variable (Y).

$$y_i = b_0 + b_1 x_1 + b_2 x_2 + b_3 x_3 \ldots + e$$

The difference between linear regression and multiple linear regression is the number of independent variables (X). In both cases there is only a single dependent variable (Y).

There is one more type of Linear Regression which is Polynomial Regression, this is a generalized case of linear regression, which we will come in another paper and, we will find that many of the challenges we encounter can be solved with Multiple Linear Regression.

Practical 1

Let's start by working our way through a basic example so that we understand how to use the equation before we convert it into Python code.

Our input data is as follows:

X	1	3	10	16	26	36
Y	42	50	75	100	150	200

Here our goal is to solve the equation **y = mx + c.**

There is a formula for finding the Intercept and Slope from the data. As the data values are few, we will be able to do it manually. Let us do it step by step.

- Sum of X = 1+3+10+16+26+36 = 92
- Sum of Y = 42+50+75+100+150+200 = 670
- Next is to get the Sum of X squared = (square of 1) + (square of 3) + (square of 10) + (square of 16) + (square of 26) + (square of 36) = 2338
- Similarly get the Sum of Y Squared = (square of 42) + (square of 50) + (square of 75) + (square of 100) + (square of 150) + (square of 200) = 82389
- Next is to multiply x and y = (1 * 42) + (3 * 50) + (10 * 75) + (16 * 100) + (26 * 150) + (26 * 150) = 13642

Cool now we do have the values, and we can find the value of slope (m) and intercept (c).

Below is the formula to find the Slope (m)

$$\frac{n(\sum xy) - (\sum x)(\sum y)}{n(\sum x^2) - (\sum x)^2}$$

The value for slope is 4.51

$$m = \frac{6\,(1364) - (92)\,(617)}{6\,(2338) - (92)\,\hat{}\,2} = \frac{25088}{5564} = 4.51$$

And formula to get the intercept (c) is

$$c = \bar{y} - b\,\bar{x}$$

$$\bar{y} = \text{Sum of y / Sample Count} = 617 / 6 = 102.83$$

$$\bar{x} = \text{Sum of x / Sample Count} = 92 / 6 = 15.3$$

$$c = 102.83 - 4.51 * 15.3 = 33.83$$

After this we have identified the Slope (m) as 4.51 and intercept (c) as 33.83.

Now that we have these values, if we need to predict any new value, we can easily get it. For example, if x is 12, we can predict the value of y as follows:

y = m x + c = 4.51 * 12 + 33.83 = 87.95

Wow, we did it. So, for X as 12, Y will be 87.95.

This seems a bit tedious, so we would have ignored this during our time at school. Which is why we are brushing-up on it now. Today we have Python to make life easy, but it is important to understand who the equation works

Getting Started with Python

First gather data or observation or sample.

```
[17] import numpy as np
     import matplotlib.pyplot as plt
```

Lets have some random values for our x and y.

```
[18] x = np.array([1,3,10,16,26,36])
     y = np.array([42,50,75,100,150,200])
```

Next import the required library for Linear Regression. And get the value of the Slope and intercept.

```
[21] from sklearn.linear_model import LinearRegression
```

```
[22] lr = LinearRegression()
     lr.fit(x,y)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
[23] m = lr.coef_
     c = lr.intercept_
     print ('Slope ', m)
     print ('Intercept ', c)
```

```
Slope  [4.50898634]
Intercept  33.695542774982016
```

Now calculate the value of y when the value of x is 12, which comes out as 87.80 which is near to what we calculated manually (87.95). The difference is mainly due to values after the decimal, which when done manually we only considered 2 places after decimal.

Predict the value

```
[32] y_pred = lr.predict([[12]])
     print("Predicted value when x is 12 ", y_pred)

     Predicted value when x is 12  [87.80337886]
```

So, in python with the 5 - 6 lines of code we are able to resolve the equation and start predicting values.

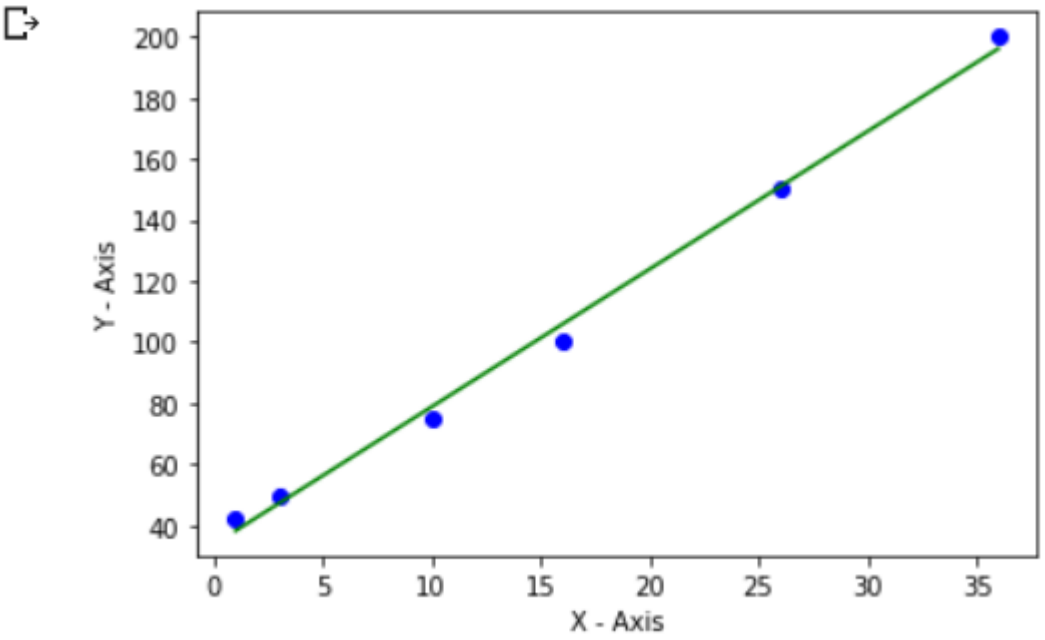
It helps to visualize. With the graph below we can see clearly that the blue dots are values and the green line is the one our model has predicted.

If you observe carefully the blue dot on X-Axis between 15-20 is not on the green line, so that gap or distance is measured as an error represented by e.

Graphical View

```
[35] plt.scatter(x, y, color = "b", marker = "o" )
     plt.plot(x, y_pred1, color = 'g')
     # putting labels
     plt.xlabel('X - Axis')
     plt.ylabel('Y - Axis')

     # function to show plot
     plt.show()
```



This doesn't end here, after we have created the model, I mean the Linear Regression, we have to test the accuracy of our model. Which can be achieved by the R-Squared method, which is denoted as

$$R^2 = \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

The R-Squared value is a statistical measure of how close the data are to the fitted regression line. It is also known as Coefficient of determination. This is used to calculate the distance or gap between the actual minus the mean or average & the distance or gap between the predicted minus the mean.

R-squared values range from 0 to 1, as the R-squared for the model reaches 1 is considered to be a good fit, and vice-versa if the R-squared value for the model is near to 0 is considered to be a bad fit. This fit excludes outliers.

However this is not always the case, it depends on the problem we are solving. In some fields it is entirely expected that the R-squared value will be low. For example any field that attempts to predict human behavior, such as Psychology, typically has R-squared values lower than around 50% which is less than 0.5, through which we can conclude that the actions and thoughts of humans are simply harder to predict.

I hope by now you have basic understanding of Linear Regression.

What we have seen above is an example of Simple Linear Regression, just imagine if we have multiple independent variables, working on this manually would be a difficult and tedious task but this can be easily handled by Python and many other programming languages.

Practical 2

Let's do some hands-on with an actual dataset. The best thing is we won't have any difficulty finding a dataset, sklearn have provided these for our use.

We already seen Simple Linear Regression, so this time we will work on Multiple Linear Regression.

You can follow along or directly check the source code, link is provided at the end of the paper.

We will use the sklearn Boston House-prices Dataset in our regression. There are 506 instances or records or observations and 13 attributes or Features and 01 Target.

As a general practice, first gather the dataset. Which we have already done.

Next step is to load the required libraries.

```
# Import required libarires.
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import datasets, metrics
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
```

Now load the Boston dataset into some variables. Here I am using boston as a variable to hold the entire dataset.

```
[2] # load the boston dataset
    boston = datasets.load_boston(return_X_y=False)
```

```
[3] print(boston.data.shape)
    print(boston.target.shape)
```

```
(506, 13)
(506,)
```

Note we have used the parameter return_X_y = False, thus it will return the whole dataset into a single variable, if we set the parameter return_X_y as True, then the result will be stored into two variables aka X and y.

The Boston dataset is a dictionary which holds keys and values for each key. We can view the keys from the Boston dataset with the **keys()** method.

boston.keys()

Though these are not related to Linear Regression, it is important to know that when dealing with a dataset from the sklearn library or any other source, if the dataset is in dictionary format.

As we can see from the above output, there are 506 rows of data and 13 columns. So must know what data the 13 columns contain. The feature_names method will provide the feature names of the columns.

This can be verified using the below code.

print(boston.feature_names)

The **DESCR** method will provide the dataset characteristics for the Boston dataset.

print (boston.DESCR)

Now it's time to convert the Dataset captured in the **boston** variable into the pandas DataFrame.

boston_df = pd.DataFrame(boston.data)

The resulting data frame has no column names in it. So let's add the column names by using the **feature_names** method and pass the column names into **boston_df.**

boston_df.columns = boston.feature_names
boston_df.head()

```
boston_df.columns = boston.feature_names
boston_df.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

From the results we can see that now we have Column Names, and there are 13 columns which are our Features. So, Feature 1 (X1) is CRIM, Feature 2 (X2) is ZN, and so on.

.As already mentioned, to get details on features, please check the **DESCR** method.

Now we have all the features, but we have not yet not talked about the Target. So, here we go,

boston.target

is holding the target column which is our Final Boston Housing Price based on the features.

Now we can split the data into X and y.

```
[11] # defining feature or independent variables (X) and  target or dependent variable (y)
     X = boston_df
     y = boston.target # Boston Housing Price
```

Next is to split the X and y into Train and Test Dataset so that we can train and test or validate the model. As usual will use 75 % for Train and 25 % for Test Dataset.

splitting X and y into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=10)

The next step is to create the instance or object for our Linear Regression model, which is similar to what we did in practice 1.

lr = LinearRegression()

lr.fit(X_train, y_train)

y_pred = lr.predict(X_test)

Also, we can get the slope and intercept value.

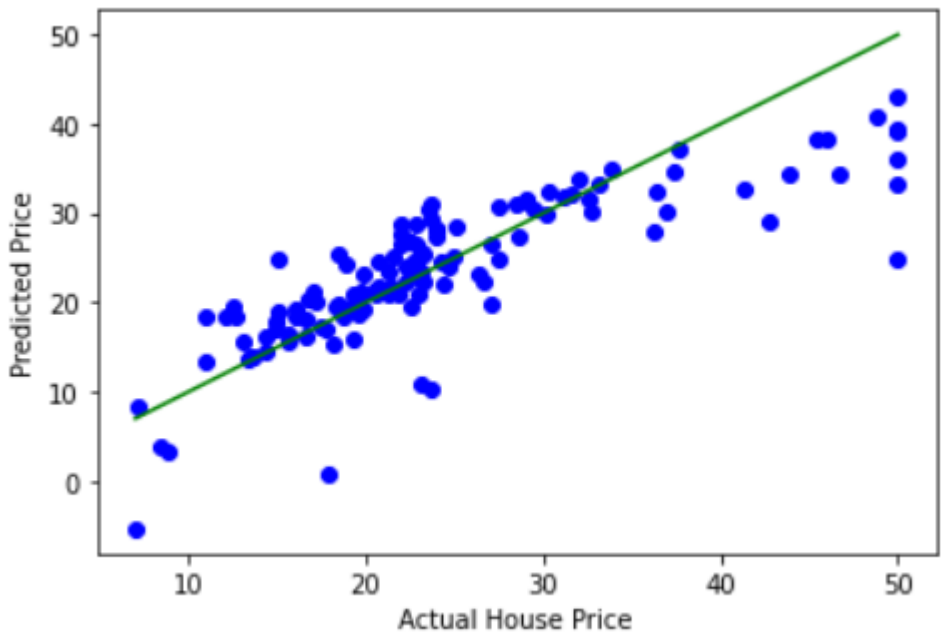
m = lr.coef_

print('Value for m (slope): \n', m)

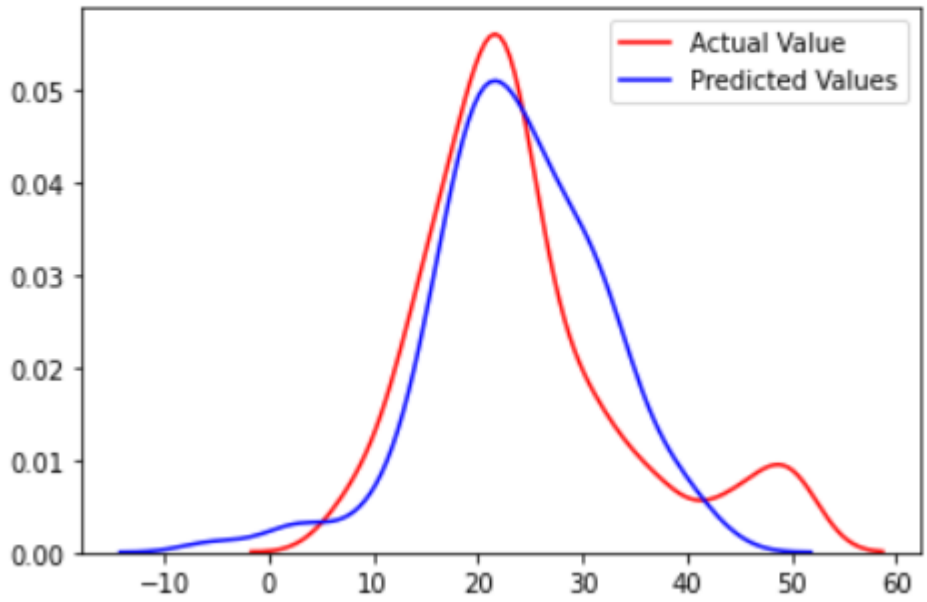
c = lr.intercept_

print ('Value of c (Intercept) ', c)

And finally, we will come up with the Predicted value which we can visualize using various graph methodology. Here I am using simple matplotlib pyplot and seaborn.




```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/
  import pandas.util.testing as tm
<matplotlib.axes._subplots.AxesSubplot at 0x7f8b600e7438>
```



Note: You cannot plot graphs for multiple regression in the same way we did in Simple Linear Regression. Multiple regression yields graphs with many dimensions. The dimensions of the graph increase as your features increase. In our case, X has 13 features.

I hope by now you have a better understanding of Linear Regression.

For Logistic Regression, please visit:

https://datascience.foundation/sciencewhitepaper/understanding-logistic-regression-with-python

Source Code

For Practice 1: https://colab.research.google.com/drive/1riQbz0VgGbgG3GhJo7cC8iEkXQcHu-1R

For Practice 2: https://colab.research.google.com/drive/1izDUnGYXhCqOQgjLGlJnsMTQbSJqZRIz

Boston Housing Dataset:

https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_boston.html#sklearn.datasets.load_boston

Scikit-learn site for other datasets:

https://scikit-learn.org/stable/modules/classes.html#module-sklearn.datasets

References:

- https://www.statisticallysignificantconsulting.com/RegressionAnalysis.htm
- https://www.statisticssolutions.com/what-is-multiple-linear-regression/
- https://en.wikipedia.org/wiki/Linear_regression
- https://www.investopedia.com/terms/r/r-squared.asp

About the Data Science Foundation

The Data Science Foundation is a professional body representing the interests of the Data Science Industry. Its membership consists of suppliers who offer a range of big data analytical and technical services and companies and individuals with an interest in the commercial advantages that can be gained from big data. The organisation aims to raise the profile of this developing industry, to educate people about the benefits of knowledge based decision making and to encourage firms to start using big data techniques.

Contact Data Science Foundation

Email:admin@datascience.foundation
Telephone: 0161 926 3641
Atlantic Business Centre
Atlantic Street
Altrincham
WA14 5NQ
web: www.datascience.foundation