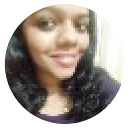


Surya Remanan Follow
GHCI Scholar | Electronics Engineer | Data Science Enthusiast | DIYHacker
Sep 4 · 5 min read

Linear Regression using Python

. . .

Meaning of Regression

Regression attempts to predict one dependent variable (usually denoted by Y) and a series of other changing variables (known as independent variables, usually denoted by X).

Linear Regression

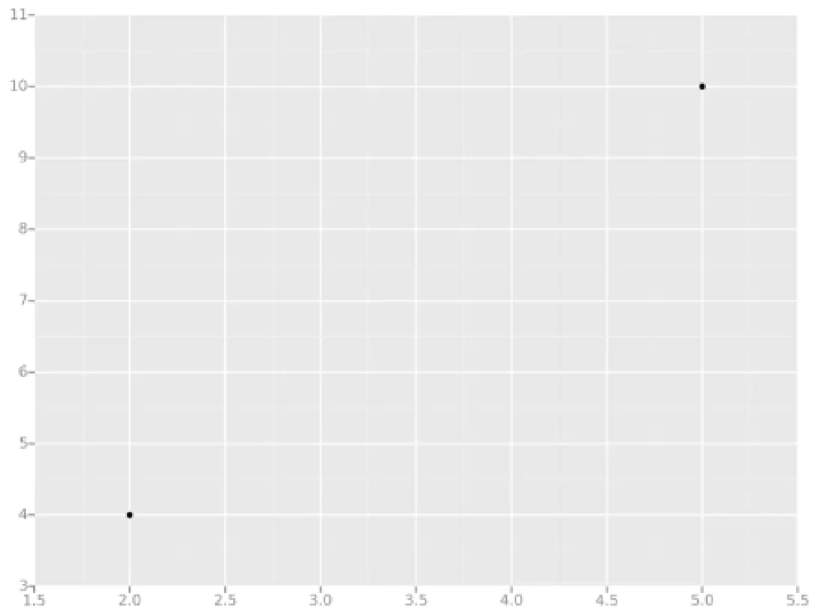
Linear Regression is a way of predicting a response Y on the basis of a single predictor variable X. It is assumed that there is approximately a linear relationship between X and Y. Mathematically, we can represent this relationship as:

Y ≈ ɒ + ß X + Ɛ

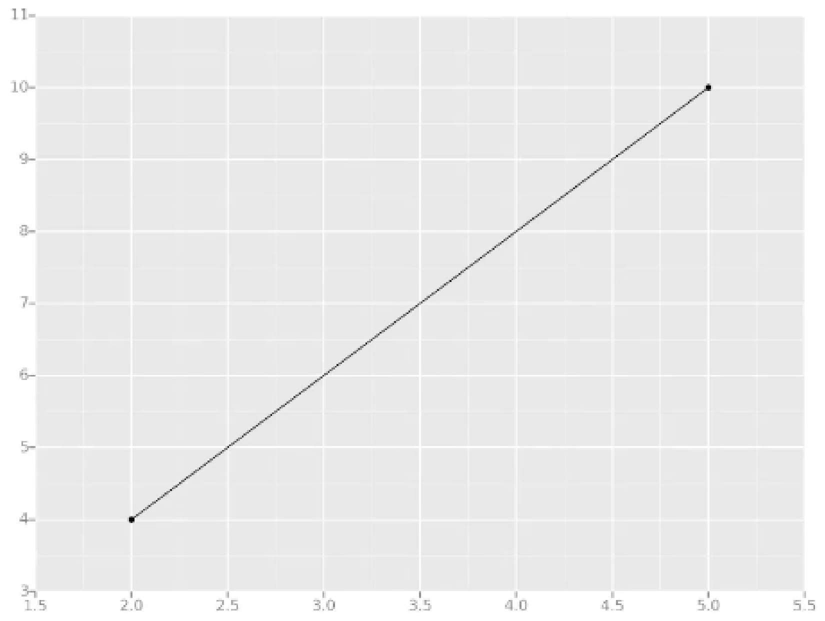
where ɒ and ß are two unknown constants that represent intercept and slope terms in the linear model and Ɛ is the error in the estimation.

Example

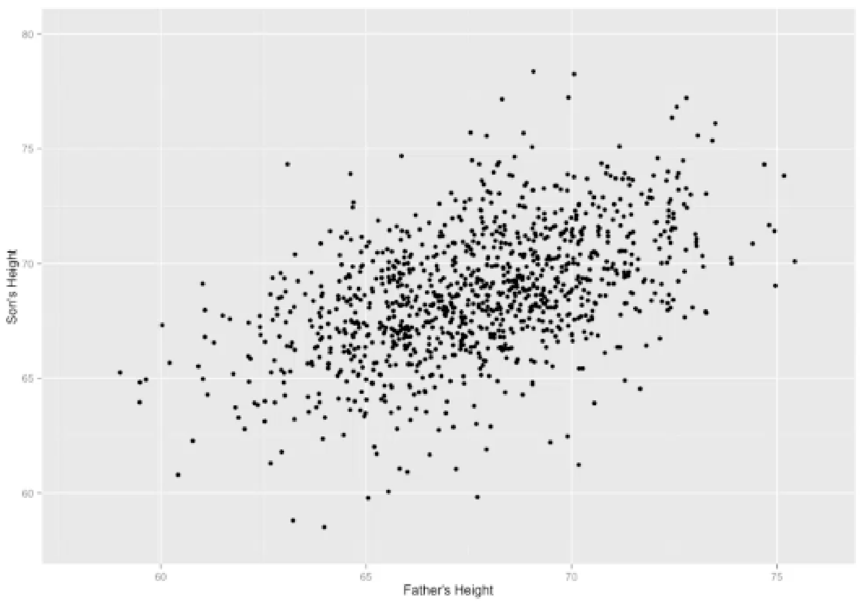
Let’s take the simplest possible example. Calculate the regression with only two data points.



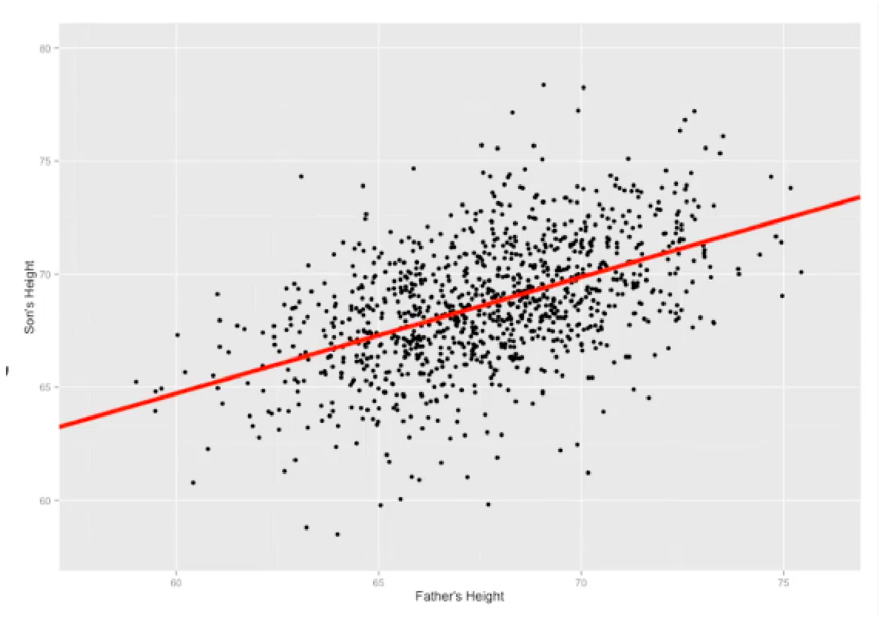
Here we have 2 data points represented by two black points. All we are trying to do when we calculate our regression line is draw a line that is as close to every point as possible.



Here, we have a perfectly fitted line because we only have two points.Now, we have to consider a case where there are more than 2 data points.



By applying linear regression we can take multiple X's and predict the corresponding Y values. This is depicted in the plot below:



Our goal with linear regression is to minimise the vertical distance between all the data points and our line.

So now I guess, you have got a basic idea what Linear Regression aims to achieve.

Python codes

First, let's import the libraries:

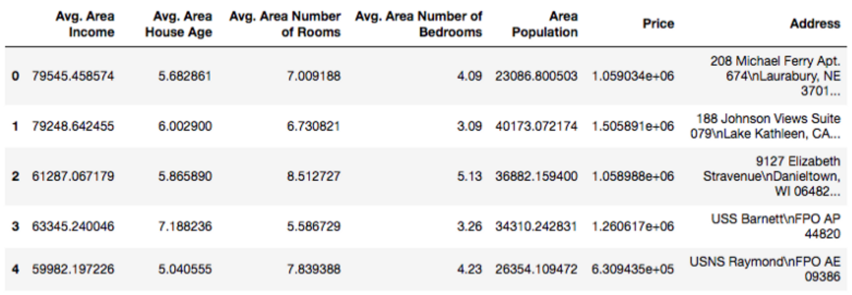
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt   #Data visualisation
libraries
```

```
import seaborn as sns
%matplotlib inline
```

The next step is importing and checking out the data.

```
USAhousing = pd.read_csv('USA_Housing.csv')
USAhousing.head()
USAhousing.info()
USAhousing.describe()
USAhousing.columns
```

Here, I have used USA_Housing.csv as the example dataset. It is always a good practice to explore the dataset. Try using your own file and run the above code to get all possible information about the dataset.



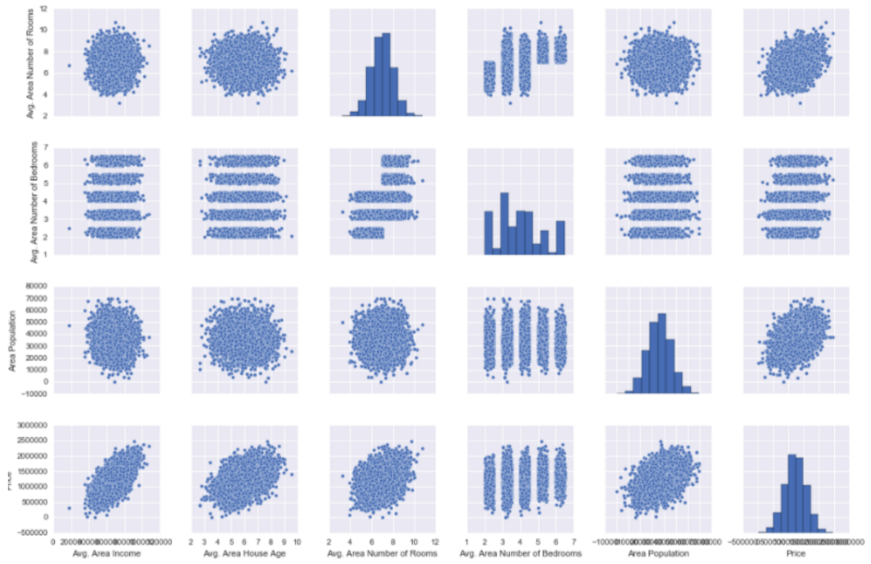
	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
0	79545.458574	5.682861	7.009188	4.09	23086.800503	1.059034e+06	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	79248.642455	6.002900	6.730821	3.09	40173.072174	1.505891e+06	188 Johnson Views Suite 079\nLake Kathleen, CA...
2	61287.067179	5.865890	8.512727	5.13	36882.159400	1.058988e+06	9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3	63345.240046	7.188236	5.586729	3.26	34310.242831	1.260617e+06	USS Barnett\nFPO AP 44820
4	59982.197226	5.040555	7.839388	4.23	26354.109472	6.309435e+05	USNS Raymond\nFPO AE 09386

Snapshot of the first five records of my dataset

Here, I'm considering Price as the dependent variable and the rest as independent variables. Which means I have to predict the Price given the independent variables.

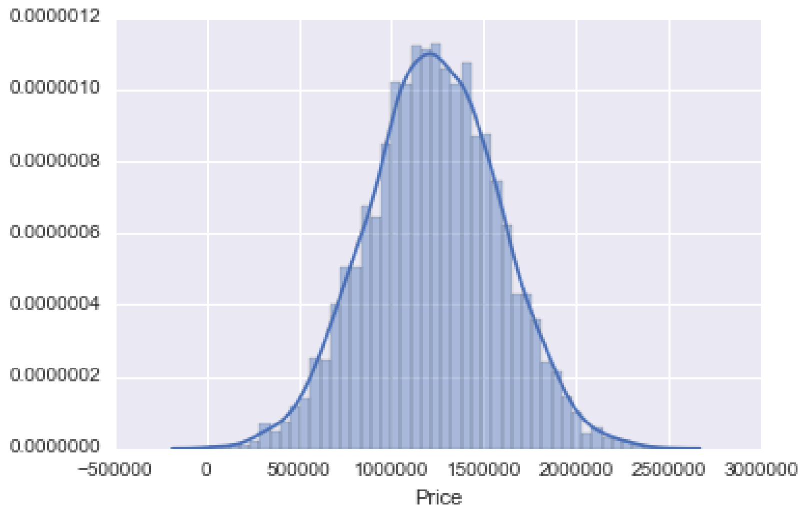
Now its time to play around with the data and create some visualizations.

```
sns.pairplot(USAhousing)
```



The pairs plot builds on two basic figures, the histogram and the scatter plot. The histogram on the diagonal allows us to see the distribution of a single variable while the scatter plots on the upper and lower triangles show the relationship (or lack thereof) between two variables.

```
sns.distplot(USAhousing['Price'])
```



A great way to get started exploring a single variable is with the histogram. A histogram divides the variable into bins, counts the data points in each bin, and shows the bins on the x-axis and the counts on the y-axis.

Correlation

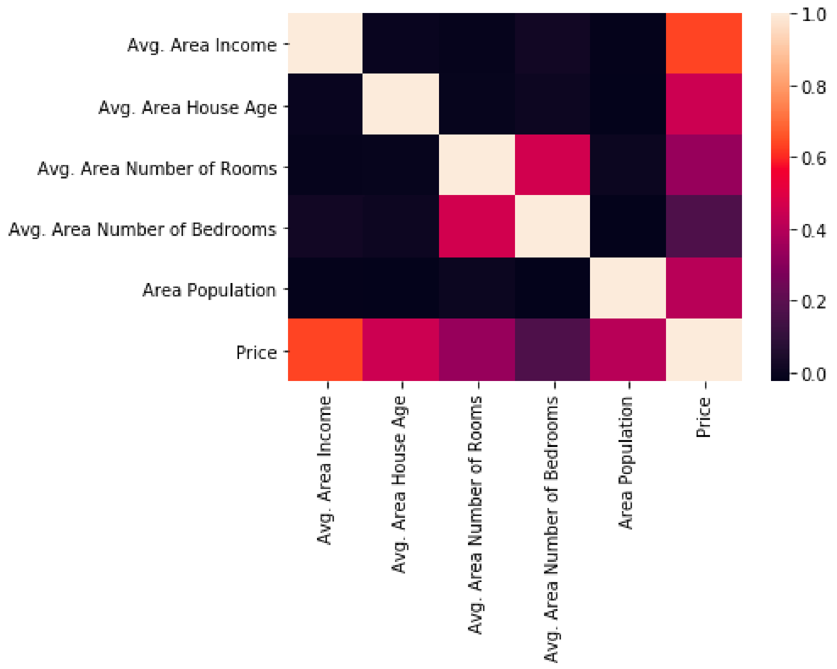
The correlation coefficient, or simply the correlation, is an index that ranges from -1 to 1. When the value is near zero, there is no linear relationship. As the correlation gets closer to plus or minus one, the relationship is stronger. A value of one (or negative one) indicates a perfect linear relationship between two variables.

Let’s find the correlation between the variables in the dataset.

```
USAhousing.corr()
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
Avg. Area Income	1.000000	-0.002007	-0.011032	0.019788	-0.016234	0.639734
Avg. Area House Age	-0.002007	1.000000	-0.009428	0.006149	-0.018743	0.452543
Avg. Area Number of Rooms	-0.011032	-0.009428	1.000000	0.462695	0.002040	0.335664
Avg. Area Number of Bedrooms	0.019788	0.006149	0.462695	1.000000	-0.022168	0.171071
Area Population	-0.016234	-0.018743	0.002040	-0.022168	1.000000	0.408556
Price	0.639734	0.452543	0.335664	0.171071	0.408556	1.000000

And now, let’s plot the correlation using a heatmap:



The black colour represents that there is no linear relationship between the two variables. A lighter shade shows that the relationship between the variables is more linear.

Coefficient of determination

Coefficient of determination R2 is the fraction (percentage) of variation in the response variable Y that is explainable by the predictor variable X. It ranges between 0 (no predictability) to 1 (or 100%) which indicates complete predictability.A high R2 indicates being able to predict response variable with less error.

Training a Linear Regression Model

Let’s now begin to train out regression model! We will need to first split up our data into an X array that contains the features to train on, and a y array with the target variable, in this case the Price column. We will toss out the Address column because it only has text info that the linear regression model can’t use.

```
X = USAhousing[['Avg. Area Income', 'Avg. Area House Age',
'Avg. Area Number of Rooms',
               'Avg. Area Number of Bedrooms', 'Area
Population']]
y = USAhousing['Price']
```

Train Test Split

Our goal is to create a model that generalises well to new data. Our test set serves as a proxy for new data.Trained data is the data on which we apply the linear regression algorithm. And finally we test that algorithm on the test data.The code for splitting is as follows:

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.4, random_state=101)
```

From the above code snippet we can infer that 40% of the data goes to the test data and the rest remains in the training set.

Creating and Training the Model

```
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(X_train,y_train)
```

The above code fits the linear regression model on the training data.

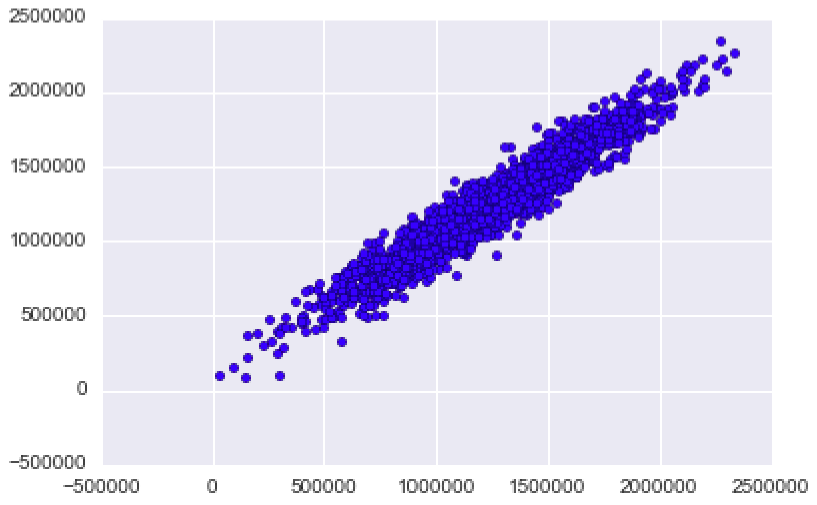
Predictions from our Model

Let’s grab predictions off the test set and see how well it did!

```
predictions = lm.predict(X_test)
```

Let’s visualise the prediction

```
plt.scatter(y_test,predictions)
```



A pretty good job has been done, a linear model has been obtained!