Implement Simple Linear Regression Model

The linear regression model is a basic concept in machine learning and AI fields. Currently, many industries use this model to predict the future of their industry. Then they might get an idea about the preparation for the future. For example, let's consider about a company which is selling fish products. Basically, they can use previous data to predict the price variations of fish and customer requirements etc. Therefore, simply they can use this data to improve the business by preparing for the future. Then this model should be very accurate for the prediction. Following description includes a comparison between two implementations of the **Simple linear regression** model.

Implementation using statistical equations

This is a simplest implementation of the regression line. Basically, two statistical equations can be used to calculate the **gradient '*m*'** and **intercept (or bias) '*b*'** according to the given data. Then the regression model can be easily evaluated using the ***y = mx + b*** equation. Following equations can be used for finding a suitable value for the ***m*** and ***b***. But not that these values are just suitable values for the best fit line without any **optimization**.

gradient(slope) = ($mean(X_i)$ * $mean(y_i)$ – $mean(X_i*y_i)$) / ($mean(X_i)^2$ – $mean(X_i^2)$)

intercept = $mean(y_i)$ – m * $mean(X_i)$

following Table 1 illustrates calculated data using above equations for the regression line in Figure 1.

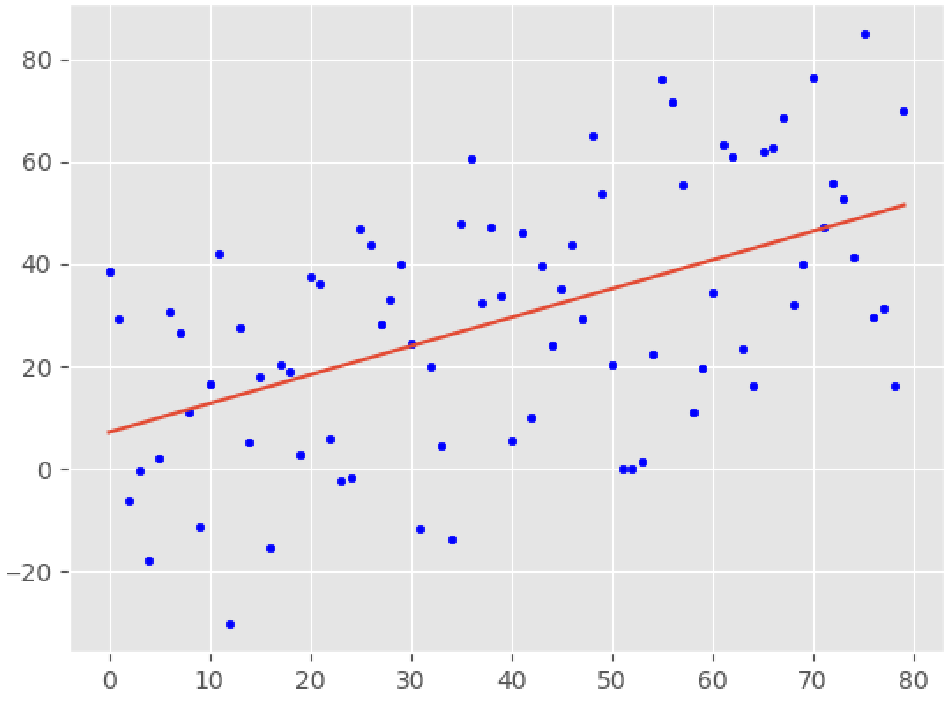


Figure 1

Table 1

Gradient/slope	Intercept	R- Squared (R^2) value
0.5608	7.1531	0.2809

Implementation using the Gradient Descent

As mentioned above the simple linear regression model follows the ***y = mx + b*** equation. The first implementation just gives a value for the ***m*** and ***b***. But it can be found most appropriate ***m*** and ***b*** values to build the model with better accuracy. Then, the gradient descent algorithm is one of the basic optimization algorithm that has used both machine learning and AI fields. A better understanding of this algorithm may be helpful to understand advanced optimization techniques and models.

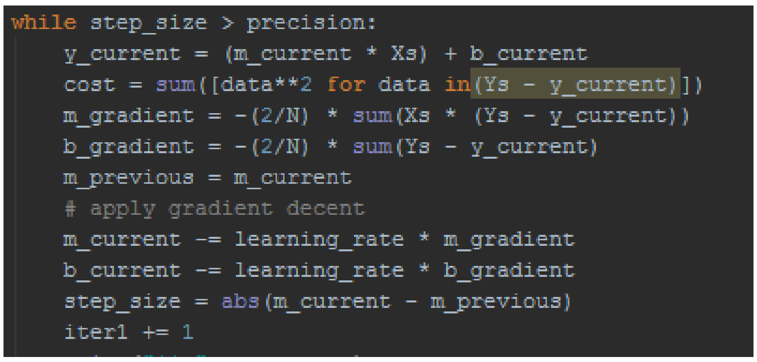
The cost function can determine the fitness of the regression line for the currently available data. The **Mean Square Error (MSE)** function with respect to data can be used as the cost function for the linear regression model. Since the MSE depends on the ***m*** and ***b*** of the regression line, it can be found a minimum value for MSE by choosing optimized values for those. Following equation can be used to calculate MSE for **n** size (number of points) dataset.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

Above equation can be used to calculate the total error due to gradient and intercept separately by using the partial derivatives. Then by applying the gradient descent algorithm with a suitable **learning rate** for the system, the best values for the ***m*** and ***b*** can be found. The derived equation using the partial derivatives are illustrated as follows. The implementation of the gradient descent using three equations shows in Figure 2. Note that here, it should be chosen a better value for the learning rate according to the precision. For a small learning rate, it will take a long time to finish iterations. And also for a higher learning rate, it will end up without taking to the local minimum or sometime it will take infinity for the ***m*** and ***b***.

$$\frac{\partial}{\partial \text{m}} = \frac{2}{N}\sum_{i=1}^{N} -x_i(y_i - (mx_i + b))$$

$$\frac{\partial}{\partial \text{b}} = \frac{2}{N}\sum_{i=1}^{N} -(y_i - (mx_i + b))$$



```
while step_size > precision:
    y_current = (m_current * Xs) + b_current
    cost = sum([data**2 for data in(Ys - y_current)])
    m_gradient = -(2/N) * sum(Xs * (Ys - y_current))
    b_gradient = -(2/N) * sum(Ys - y_current)
    m_previous = m_current
    # apply gradient decent
    m_current -= learning_rate * m_gradient
    b_current -= learning_rate * b_gradient
    step_size = abs(m_current - m_previous)
    iter1 += 1
```

Figure 2

Gradient descent algorithm provides better fitness values for the regression line than the first implementation. But the only drawback of this technique is, it takes a long time to return gradient and intercept value with given precision. Following Figure 3 shows the regression line for the same dataset that refers to the Figure 1. Table 2 provides the measurements for this regression line which is optimized using the gradient descent algorithm.

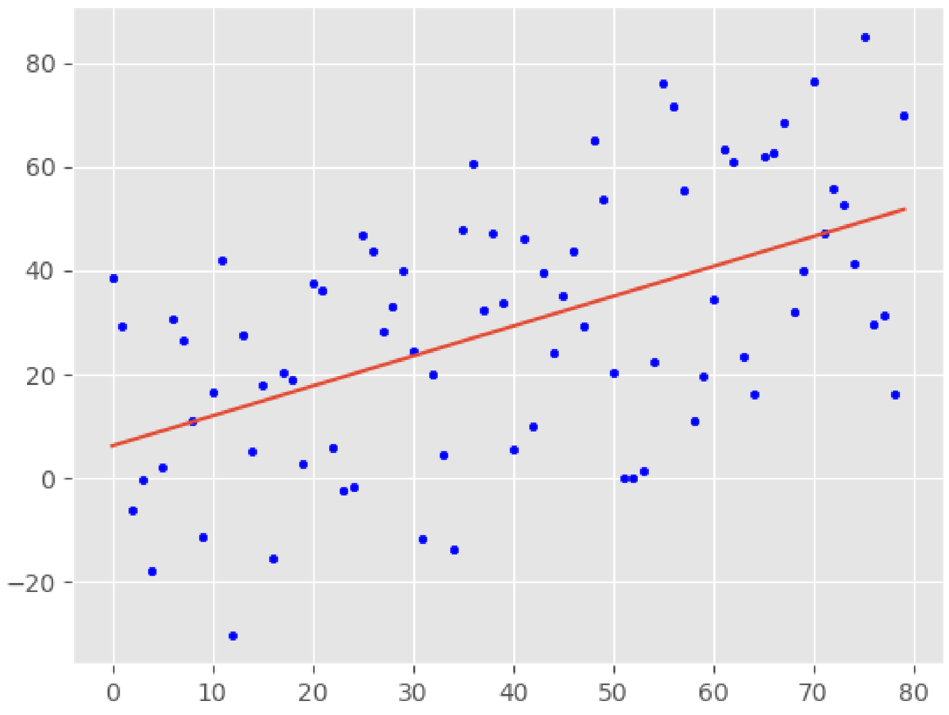


Figure 3

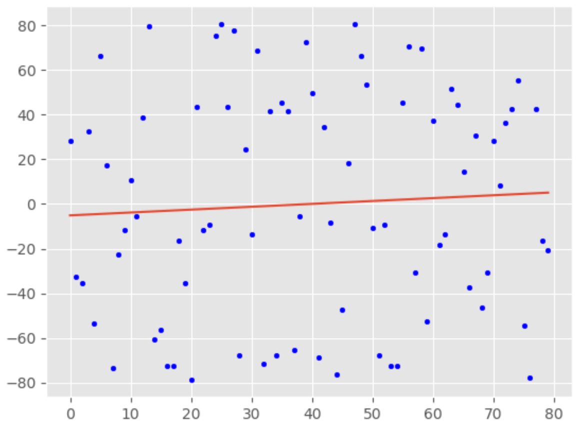
Table 2

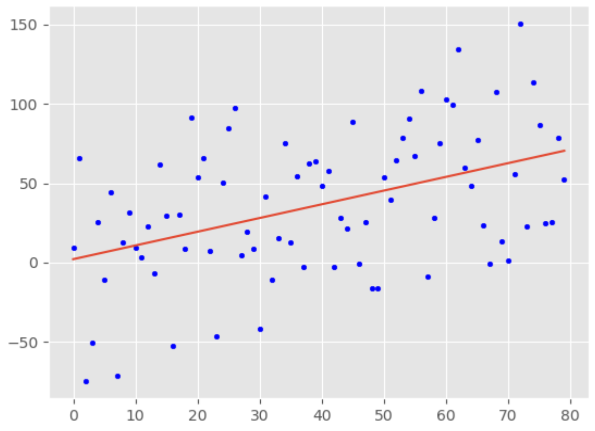
Gradient/slope	Intercept	R- Squared (R^2) value	MSE	Iterations
0.5763	6.2339	0.281193150	449.668942853	80652
0.5745	6.3275	0.281197515	449.666212239	125866
0.5743	6.3369	0.281197558	449.666184933	171080
0.5743	6.3378	0.281197559	449.666184660	216294
0.5743	6.3379	0.281197559	449.666184657	306721

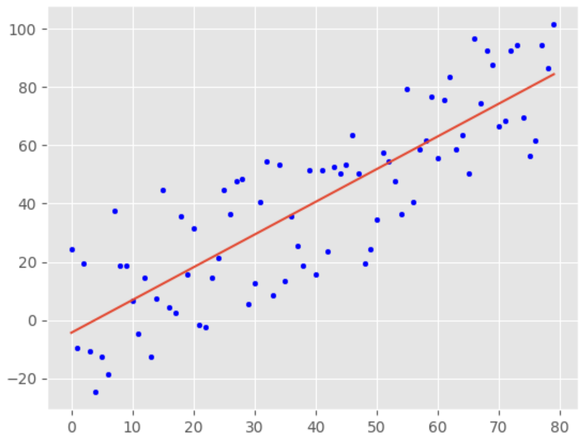
According to the Table 2, gradient and intercepts values (up to four decimal places) which give best regression line are 0.5743 and 6.3379. Therefore, the equation which describes the best regression line is, ***y = 0.5743x + 6.3379***. Following Table 3 illustrate the variation of R^2 and MSE according to the correlation of regression lines in Figure 4.

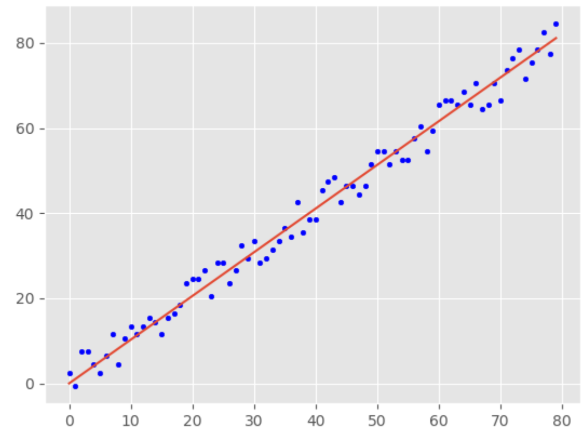
Table 3

Figure	Gradient/slope	Intercept	R- Squared (R^2) value	MSE	Iterations
4.a	0.1288	-5.0867	0.0036	2491.2819	76774
4.b	0.8647	2.1302	0.1932	1656.9710	60069
4.c	1.1233	-4.3552	0.7102	275.5518	73876
4.d	1.0264	0.0205	0.9846	8.4856	25


4.a


4.b


4.c


4.d