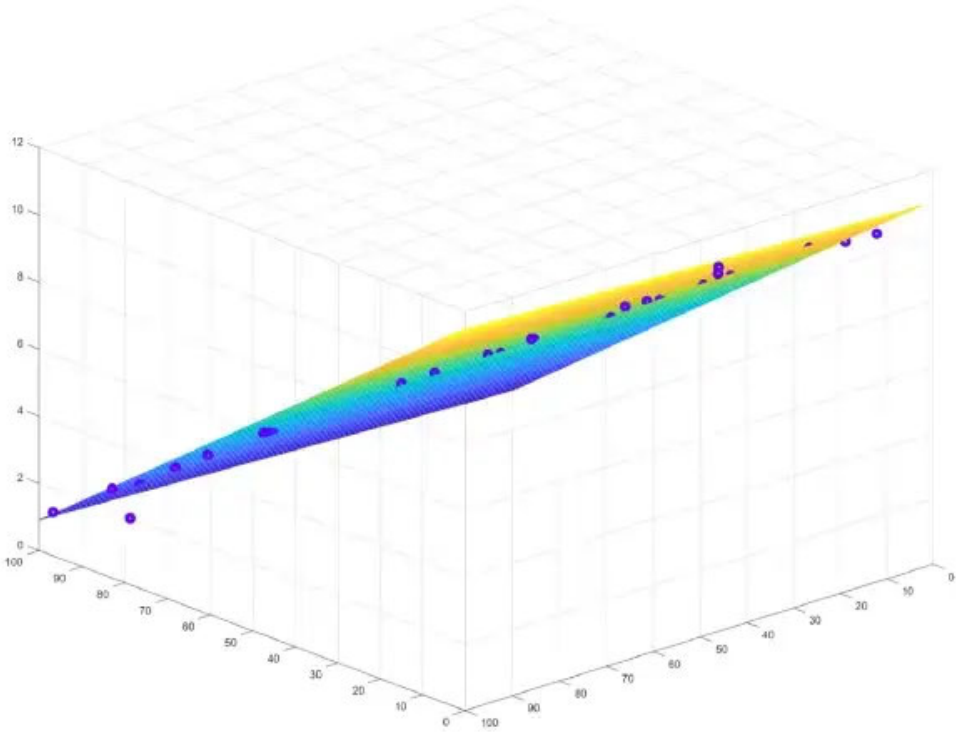
Towards AI Editorial Team Follow

Oct 7, 2020 · 18 min read · ✨ · Listen



Source: Image by the Author.

DATA SCIENCE, MACHINE LEARNING, EDITORIAL, PROGRAMMING, STATISTICS

Simple Linear Regression Tutorial for Machine Learning (ML)

Last updated, January 8, 2021

This tutorial's code is available on Github and its full implementation as well on Google Colab.

Table of contents:

📚 Check out our gradient descent tutorial. 📚

What is a Simple Linear Regression?

Simple linear regression is a statistical approach that allows us to study and summarize the relationship between two continuous quantitative variables. Simple linear regression is used in **machine learning** models, mathematics, statistical modeling, forecasting epidemics, and other quantitative fields

variable's value based on the value of the independent variable. A simple linear regression aims to find the best relationship between X (independent variable) and Y (dependent variable).

There are three types of relationships. The kind of relationship where we can predict the output variable using its function is called a **deterministic relationship**. In **random relationships**, there are no relationships between the variables. In our statistical world, it is not likely to have a deterministic relationship. In statistics, we generally have a relationship that is not so perfect, that is called a **statistical relationship**, which is a mixture of deterministic and random relationships [4].

Examples:

1. Deterministic Relationship:

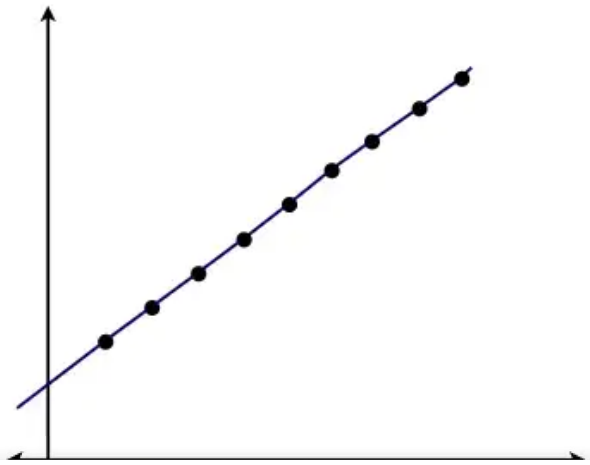
a. Diameter = 2*pi*radius

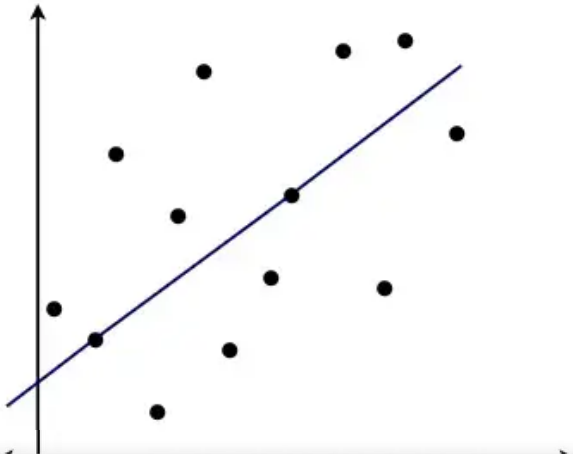
b. Fahrenheit = 1.8*celsius+32

2. Statistical Relationship:

a. Number of chocolates vs. cost

b. Income vs. expenditure





Understanding Simple Linear Regression:

The simplest type of regression model in machine learning is a **simple linear regression**. First of all, we need to know why we are going to study it. To understand it better, why don't we start with a story of some friends that lived in "Bikini Bottom" (referencing SpongeBob) [3].

SpongeBob, Patrick, Squidward, and Gary lived in the "Bikini Bottom!". One day Squidward went to SpongeBob, and they had this conversation. Let's check it out.

Squidward: "Hey, SpongeBob, I have heard that you are so smart!"

SpongeBob: "Yes, sir! There is no doubt in that."

Squidward: "Is that so?"

SpongeBob: "Umm…Yes!"

Squidward: "So here is the thing. I want to sell my house as I am going to shift to my new lavish house downtown. But I cannot figure out at which price I should sell my house! If I keep the price too high, then no one will buy it, and if I set the price low, I might face a large financial loss! So you have to help me find the best price for my house. But, please keep in mind that you have only one day!"

SpongeBob stressed as always, but optimistic about finding the solution. To discuss the problem, he went to his wise friend Patrick's house. Patrick is in his living room watching TV with a big bowl of popcorn in his hands, and after SpongeBob described the whole situation to Patrick:

Patrick: "That is a piece of cake. Follow me!"

(They decided to go to Squidward's neighborhood, where his two neighbors recently sold their houses. After making some discreet inquiries, they obtained the following details from Squidward's new neighbors. Now Patrick explained the whole plan to SpongeBob.)

Squidward's house's price. So let us get some data.

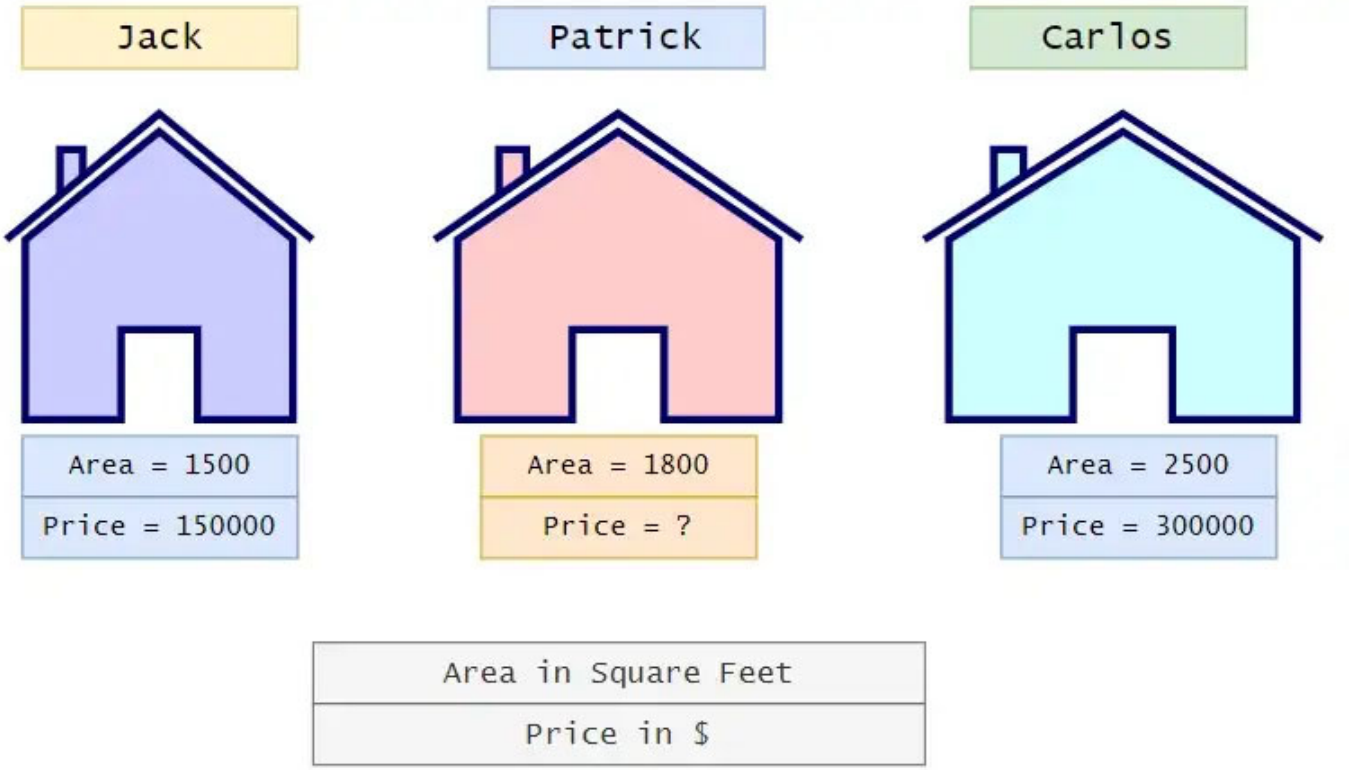
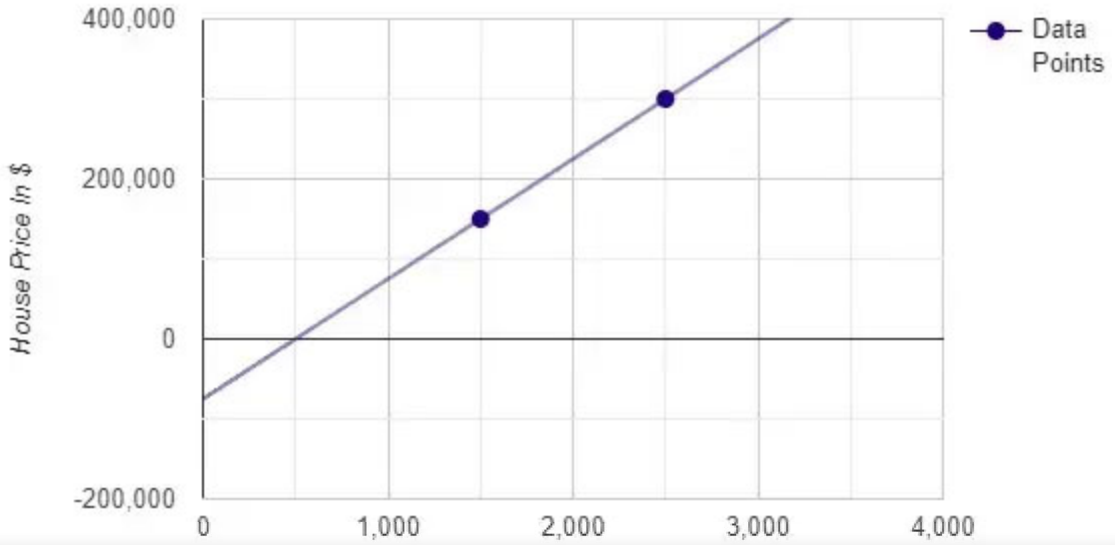


Figure 2: Area and house price of Squidward's neighborhood.

From the collected data, Patrick was able to plot the data on a scatter plot:



If we closely observe the graph above, we can notice that we can connect our two data points with a line, and as we know, each line has its equation. From figure 3, we can quickly get the house price if we have the house's area. It will be easier for us if we get the house price by using some formula. Please note that we can get the house price by plotting a horizontal and vertical line on the graph, but to generalize it, we use the line equation. First, we need to see some basics of geometry and dive into the equation of the line.

Basics of Coordinate Geometry:

1. We always look from left to right in the coordinate plane to name the points.
2. After looking from left-to-right, the first point we get must be named (x1,y1), and the second point will be (x2,y2).
3. Horizontal lines have a slope of 0.
4. Vertical lines have an "infinite" slope.
5. If the second point's Y-coordinate is greater than the Y-coordinate of the first point, then the line has a positive(+) slope. The line has a negative slope.
6. Points at the same vertical distance from X-axis have the same Y-coordinate.
7. Points at the same vertical distance from Y-axis have the same X-coordinate.

Now let's get back to our graph.

We all know that the equation of the line:


$$Where,$$

$$m = slope\ of\ the\ line$$

$$b = Y - intercept\ of\ the\ line$$

$$Y = Y\ coordinate\ of\ the\ point\ from\ which\ the\ line\ passes\ through$$

$$X = X\ coordinate\ of\ the\ point\ from\ which\ the\ line\ passes\ through$$

Figure 4: Equation of a straight line.

From the definition of the slope of a straight line:

$$m = \frac{Y_2 - Y_1}{X_2 - X_1}$$

$$Where,$$

$$m = slope\ of\ the\ line$$

$$(X_1, Y_1) = X\ and\ Y\ co\ldots he\ first\ data\ point$$

$$(X_2, Y_2) = X\ and\ Y\ coordinates\ of\ the\ second\ data\ point$$

Figure 5: Equation of slope of a straight line.

From the rules mentioned above, we can infer that in our graph:

(X1 , Y1) = (1500 , 150000)

(X2 , Y2) = (2500 , 300000)

Next, we can easily find the slope of the two points.

$$2500 - 1500$$

$$m = \frac{150000}{1000}$$

$$m = 150$$

Figure 6: Calculating the slope for our example.

Taking our example into consideration, in our equation, Y represents the house's price, and X represents the area of the house.

Now since we have all the other values, we can calculate the value of slope b.

$$b = Y - mX$$

$$(X, Y) = (1500, 150000)$$

$$b = 150000 - 150 * 1500$$

$$b = -75000$$

$$Y = mX + b$$

$$b = Y - mX$$

$$(X, Y) = (2500, 300000)$$

$$b = 300000 - 150 * 2500$$

$$b = -75000$$

Figure 7: Calculating the Y-intercept of the line.

Notice that we can use any of the points to calculate the slope value. The answer to the slope will always be the same for the same straight line.
Next, since we have all our parameters, we can write the equation of line as:

$$Y = mX + b$$

$$Y = 150X - 75000$$

Figure 8: Equation of the line.

To find the price of Squidward's house, we need to plug-in X=1800 in the above equation.


$$Y = 150X - 75000$$

$$Y = 150 * 1800 - 75000$$

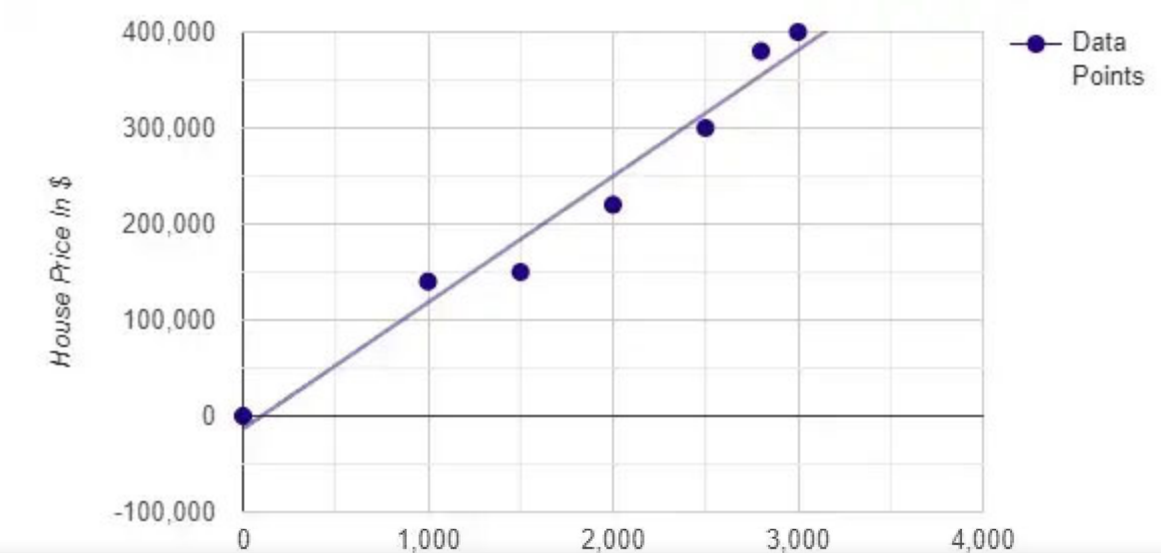
$$Y = 195000$$

Figure 9: Predicting Squidward's house price.

Now, we can say that Squidward should sell his house for $ 195,000.00. That was easy.

Please note that we only had two data points to quickly plot a single straight line through them and get our equation of a line. In this case, the critical thing to notice here is that our prediction will depend on the value of two data points. If we change the value of any of the two available data points, our prediction will likely also change. To cope with this problem, we have data sets in larger quantities. Real-world datasets may contain millions of data points.

Now let us get back to our example. When we have more than two data points in our dataset(the usual case), we cannot draw a single straight line that passes through all points, right? That is why we will use a line that best fits our data set. This line is called the best fit line or the regression line. By using this line's equation, we will make predictions about our dataset.



Please note that the central concept remains the same. We will find the equation of the line and plug-in X's value (independent variable) to find Y's value (dependent variable). We need to find the best fit line for our dataset.

Calculating the Linear Best Fit

As we can see in figure 11, we cannot plot a single straight line that passes through all the points. So what we can do here is to minimize the error. It means that we find a bar and then find the prediction error. Since we have the actual value here, we can easily find the error in prediction. Our ultimate goal will be to find the line that has the minimal error. That line is called the **linear best fit.**

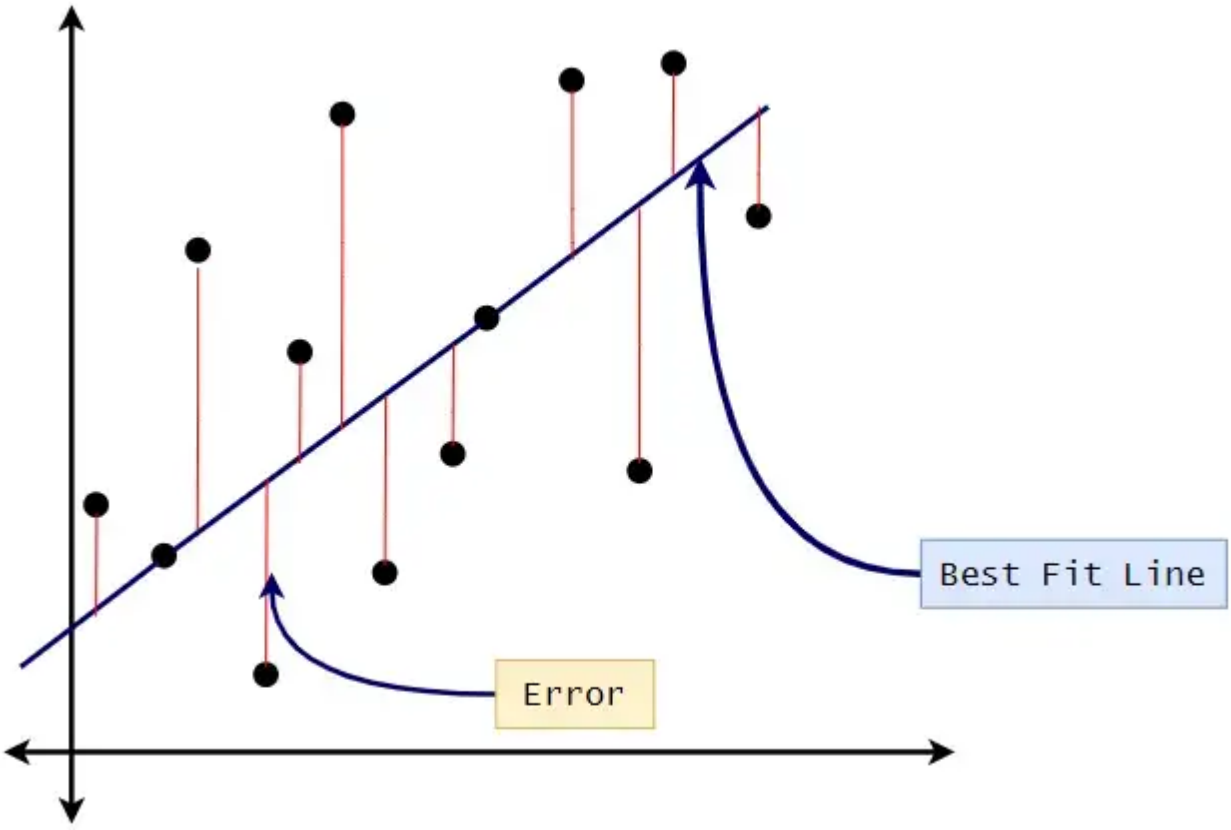


Figure 11: Calculating the linear best fit.

As discussed above, our goal is to find the **linear best fit** for our dataset, or in other words, we can say that our goal should be to reduce the error in prediction. Now the

$220,000. If we predict the house price based on the line equation, which is Y = 150X-75000, we will get the house price at $ 195,000. Now here we can see that there is a prediction error.

Therefore, we can use the Sum of Squared error calculation technique to find the error in prediction for each of the data points. We randomly choose the parameters of our line and then calculate the error. Afterward, we will adjust the parameter again and then calculate the error.

We will repeat this until we get the minimum possible error. This process is a part of the gradient descent algorithm, which we will cover in later tutorials. We think now it is clear that we will recalculate the line's parameters until we get the best fit line, or we get a minimum error in our prediction.

1. Positive error:

Actual selling price: $ 220,000

Predicted selling price: $ 195,000

Error in prediction: $ 220,000–$ 195,000 = $25,000

2. Negative error:

Actual selling price: $ 160,000

Predicted selling price: $ 195,000

Error in prediction: $160,000 — $195,000 = -$35,000

As we can see, it is also possible to get a negative error. To account for negative errors, we square the error.

$$Errors = \sum^{i=N} (Y_{i\,actual} - Y_{i\,predicted})$$

To account for the negative values, we will square the errors.

$$Errors = \sum_{i=1}^{i=N} \left(Y_{i\,actual} - Y_{i\,predicted}\right)^2$$

Figure 13: Formula for the sum of squared errors.

Next, we have to find the parameters of a line that has the least error. Once we have that, we can form an equation of a line and predict the dataset's data values. We will go through this part later in this tutorial.

Guidelines for regression line:

1. Use regression lines when there is a significant correlation to predict values.
2. Stay within the range of the data, and make sure not to extrapolate. For example, if the data is from 10 to 60, do not try to predict a value for 500.
3. Do not make predictions for a population that base on another population's regression line.

Use-cases for linear regression:

1. Height and weight.
2. Alcohol consumption and blood alcohol content.
3. Vital lung capacity and pack-years of smoking.
4. The driving speed and gas mileage.

Finding the Equation for the Linear Best Fit

Before we dive deeper into a simple linear regression formula's derivation, we will try to find the best fit line parameters without using any formulas. Consider the following table with data points X and Y. The next table Y' is the predicted value and Y-Y' gives us

3	1	3m+b	1-3m-b
5	6	5m+b	6-5m-b
7	8	7m+b	8-7m-b

Figure 14: Data points.

Next, we are going to use the sum of squares method to calculate the error. For such, we will have to find $(Y-Y')^2$. Please note that we have three terms in each row of (Y — Y'). First, we will dive into the formula to find the square with three terms.

$$(A+B+C)^2 = A^2 + B^2 + C^2 + 2AB + 2BC + 2AC$$

Figure 15: Expansion of $(a+b+c)^2$.

In our case, the value of $(Y - Y')^2$ for each row will be:

$$(1-3m-b)^2 = 1 + 9m^2 + b^2 - 6m + 6mb - 2b$$
$$(6-5m-b)^2 = 36 + 25m^2 + b^2 - 60m + 10mb - 12b$$
$$(8-7m-b)^2 = 64 + 49m^2 + b^2 - 112m + 14mb - 16b$$

Figure 16: Expanding our terms.

Next, notice that we need to add all the squared terms in our formula of the error sum of squares.

$$(1-3m-b)^2 + (6-5m-b)^2 + (8-7m-b)^2$$
$$= 101 + 83m^2 + 3b^2 - 178m - 30b + 30mb$$

$$The\ vertex\ of\ aX^2 + bX + c\ is\ \ldots\ X = \frac{-b}{2a}$$

Figure 18: The vertex of a second-degree polynomial.

Next, we need to rearrange our central equation to bring it in a second-degree polynomial form. As we know that if we have two linear equations, we can quickly solve them and get the required values. Hence, our ultimate goal will be to find two linear equations and solve them.

$$\longrightarrow 101 + 83m^2 + 3b^2 - 178m - 30b + 30mb$$

$$Rewriting\ the\ equation:$$

$$\longrightarrow 3b^2 + (30m - 30)b + (101 + 83m^2 - 178m)$$

$$\longrightarrow b = \frac{-(30m - 30)}{2 * 3}$$

$$\longrightarrow b = \frac{30 - 30m}{6}$$

$$\longrightarrow b = 5 - 5m$$

Figure 19: Finding linear equations.

Rewriting the equation

$$\longrightarrow 83m^2 + (30b - 178)m + (101 + 3b^2 - 30b)$$

$$\longrightarrow m = \frac{178 - 30b}{166}$$

$$\longrightarrow 166m = 178 - 30b$$

$$\longrightarrow 30b = 178 - 166m$$

Figure 20: Finding linear equations.

Now that we have two equations, we can solve them to find the slope and intercept values.

$$\longrightarrow b = 5 - 5m$$

$$\longrightarrow 30b = 178 - 166m$$

Multiplying equation (1) *by* 30:

$$\longrightarrow 30b = 150 - 150m$$

Final Equations:

$$\longrightarrow 30b = 178 - 166m$$

$$\longrightarrow 30b = 150 - 150m$$

Figure 32: Solving two linear equations.

*Subtracting eq.*2 *from eq.*1:

$$\longrightarrow 0 = 28 - 16m$$

$$\longrightarrow m = 1.75$$

Put the value of m *in any of the above equations*

$$\longrightarrow 30b = 178 - 166(1.75)$$

$$\longrightarrow b = -3.75$$

Figure 33: Solution of two linear equations.

$$Y = 1.75X - 3.75$$

Figure 34: Equation of the line.

We can also plot the data on a scatter plot with the line of best fit.

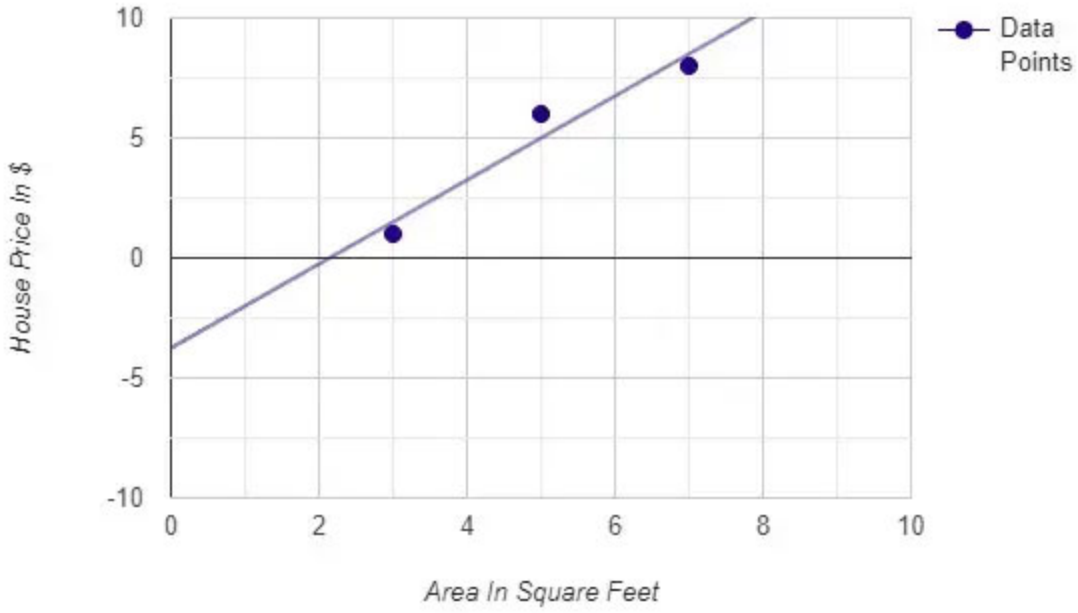


Figure 35: Area vs. house price on our scatter plot.

So this is how we can find the best fit line for a specific dataset. We can notice that for a larger dataset, this task can be cumbersome. As a solution to that, we will use a formula that will give us the required parameter values.

However, we will not dive into the formula. Instead, we will first see how the formula is derived, and then we will use it in a code example with Python to understand the math behind it.

In conclusion, a simple linear regression is a technique in which we find a line that best fits our dataset, and once we have that line, we can predict the value of the

$$(X_i, Y_i)$$

Figure 36: Our data points.

2. We define the linear best fit as:

$$\hat{Y}_i = a + BX_i$$

Figure 37: Linear best fit.

3. We can write the error function as following:

$$S = \sum_{i=1}^{i=n} (Y_i - \hat{Y}_i)^2$$

Figure 38: Error sum of squares.

4. We can substitute the value of equation 2 in equation 3:

$$S = \sum_{i=1}^{i=n} (Y_i - a - BX_i)^2$$

Figure 39: Simplifying the error sum of squares.

Next, our ultimate goal is to find the best fit line. To find the best fit line, the error function S should be minimum. To minimize our error function, S, we must find where the first derivative of S is equal to 0 concerning a and b.

$$\frac{\partial S}{\partial a} = \frac{\partial}{\partial a}\left[\sum_{i=1}^{i=n}(Y_i - a - BX_i)^2\right]$$

Figure 40: Partial derivative of S concerning a.

2. Simplifying the calculations:

$$Y_i - a - BX_i = u$$

$$S = \sum_{i=1}^{i=n} u^2$$

Figure 41: Simplifying the calculations.

3. Using chain rule of partial derivations:

$$\frac{\partial S}{\partial a} = \frac{\partial S}{\partial u} * \frac{\partial u}{\partial a}$$

Figure 42: Chain rule of partial derivations.

4. Finding partial derivatives:

$$\frac{\partial}{\partial u}\left[\sum_{i=1}^{} u^2\right] = 2 * \sum_{i=1}^{} u$$

$$\frac{\partial u}{\partial a} = \frac{\partial}{\partial u}(Y_i - a - BX_i) = 0 - 1 - 0 = -1$$

Figure 43: Finding partial derivatives.

5. Putting it together:

$$\frac{\partial S}{\partial a} = -2 * \sum_{i=1}^{i=n} u = -2 * \sum_{i=1}^{i=n} (Y_i - a - BX_i)$$

Figure 44: Merging them together.

6. To find the extreme values, we take the derivative=0:

$$-2 * \sum_{i=1}^{i=n} (Y_i - a - BX_i) = 0$$

Figure 45: Taking the derivative = 0.

7. Simplifying:

$$\sum_{i=1}^{i=n} (Y_i - a - BX_i) = 0$$

Figure 46: Simplifying the equation.

$$\sum_{i=1} Y_i - \sum_{i=1} a - \sum_{i=1} BX_i = 0$$

Figure 47: Simplifying the equation.

9. Finding the summation of a:

$$\sum_{i=1}^{i=n} a = (a + a + a \ldots + a)_{n\,times} = n * a$$

Figure 48: Finding the sum of a.

10. Substituting the values in the main equation:

$$\sum_{i=1}^{i=n} Y_i - na - B \sum_{i=1}^{i=n} X_i = 0$$

Figure 49: Putting it back in the main equation.

11. Simplifying the equation:

$$a = \frac{\sum_{i=1}^{i=n} Y_i - B \sum_{i=1}^{i=n} X_i}{n}$$

Figure 50: Simplifying the equation.

12. Further simplifying the equation:

$$n$$

$$\frac{\sum_{i=0}^{i=n} X_i}{n} = \bar{X}$$

Figure 51: Simplifying the equation.

13. Simplifying the equation for the value of a:

$$a = \bar{Y} - B\bar{X}$$

Figure 52: Value of a.

Finding B (Slope):

1. Finding the partial derivative of S concerning B:

$$\frac{\partial S}{\partial B} = \frac{\partial}{\partial B}\left[\sum_{i=1}^{i=n}(Y_i - a - BX_i)^2\right]$$

Figure 53: Finding the partial derivative of S concerning B.

Finding Partial Derivative of S concerning B

2. Simplifying the calculations:

$$Y_i - a - BX_i = u$$

$$S = \sum^{i=n} u^2$$


$$\frac{\partial S}{\partial B} = \frac{\partial S}{\partial u} * \frac{\partial u}{\partial B}$$

Figure 55: Chain rule of partial derivations.

4. Finding partial derivatives:

$$\frac{\partial S}{\partial u} = \frac{\partial}{\partial u}\left[\sum_{i=1}^{i=n} u^2\right] = 2 * \sum_{i=1}^{i=n} u$$

$$\frac{\partial u}{\partial B} = \frac{\partial}{\partial B}(Y_i - a - BX_i) = 0 - 0 - X_i = -X_i$$

Figure 56: Finding partial derivatives.

5. Putting it together:

$$\frac{\partial S}{\partial B} = -2\sum_{i=1}^{i=n} X_i * u = -2\sum_{i=1}^{i=n} X_i * (Y_i - a - BX_i)$$

Figure 57: Putting the calculated values together.

6. Distributing Xi:

$$\frac{\partial S}{\partial B} = -2\sum_{i=1}^{i=n} (X_i Y_i - X_i a - BX_i X_i)$$

Figure 58: Distributing the value of Xi.

$$-2\sum_{i=1}^{i=n}(X_iY_i - X_ia - BX_i^2) = 0$$

Figure 59: Taking the derivative = 0.

8. Simplifying:

$$\sum_{i=1}^{i=n}(X_iY_i - X_ia - BX_i^2) = 0$$

Figure 60: Simplifying the equation.

9. Substituting the value of a in our equation:

$$a = \bar{Y} - B\bar{X}$$

$$\sum_{i=1}^{i=n}(X_iY_i - X_i(\bar{Y} - B\bar{X}) - BX_i^2) = 0$$

Figure 61: Substituting the values in the equation.

10. Further simplifying:

$$\sum_{i=1}^{i=n}\left(X_iY_i - \bar{Y}X_i + BX_i\bar{X} - BX_i^2\right) = 0$$

Figure 62: Simplifying the equation.

11. Splitting up the sum:

$$\sum_{i=1} \qquad \sum_{i=1}$$

Figure 63: Splitting up terms.

12. Simplifying:

$$\sum_{i=1}^{i=n}(X_iY_i - \bar{Y}X_i) - B\sum_{i=1}^{i=n}(X_i^2 - X_i\bar{X}) = 0$$

Figure 64: Simplifying the equation.

13. Finding B from the above equation:

$$B = \frac{\sum_{i=1}^{i=n}(X_iY_i - \bar{Y}X_i)}{\sum_{i=1}^{i=n}(X_i^2 - X_i\bar{X})}$$

Figure 65: Value of B.

16. Further simplifying the equation:

$$\sum(X^2 - \frac{\sum X}{n} X)$$

$$B = \frac{n\sum XY - \sum Y \sum X}{n\sum X^2 - \sum X \sum X}$$

$$B = \frac{n\sum XY - \sum X \sum Y}{n\sum X^2 - (\sum X)^2}$$

Figure 66: Value of B.

Finding a (Intercept) in a generalized form:

1. Get the value of a:

$$a = \bar{Y} - B\bar{X}$$

$$a = \bar{Y} - \left(\frac{n\sum XY - \sum X \sum Y}{n\sum X^2 - (\sum X)^2}\right)\bar{X}$$

$$a = \frac{\sum Y}{n} - \left(\frac{n\sum XY - \sum X \sum Y}{n\sum X^2 - (\sum X)^2}\right)\frac{\sum X}{n}$$

Figure 67: Simplifying the equation.

2. Simplifying the formula:

$$n \qquad n\sum X^2 - (\sum X)^2$$

$$a = \frac{1}{n} * \frac{n\sum X^2 \sum Y - (\sum X)^2 \sum Y - n\sum XY \sum X + \sum X \sum Y \sum X}{n\sum X^2 - (\sum X)^2}$$

$$a = \frac{1}{n} * \frac{n\sum X^2 \sum Y - (\sum X)^2 \sum Y - n\sum XY \sum X + (\sum X)^2 \sum Y}{n\sum X^2 - (\sum X)^2}$$

$$a = \frac{1}{n} * \frac{n(\sum X^2 \sum Y - \sum XY \sum X)}{n\sum X^2 - (\sum X)^2}$$

$$a = \frac{\sum X^2 * \sum Y - \sum X * \sum XY}{n\sum X^2 - (\sum X)^2}$$

Figure 68: Simplified value of a.

Simple Linear Regression Formulas:

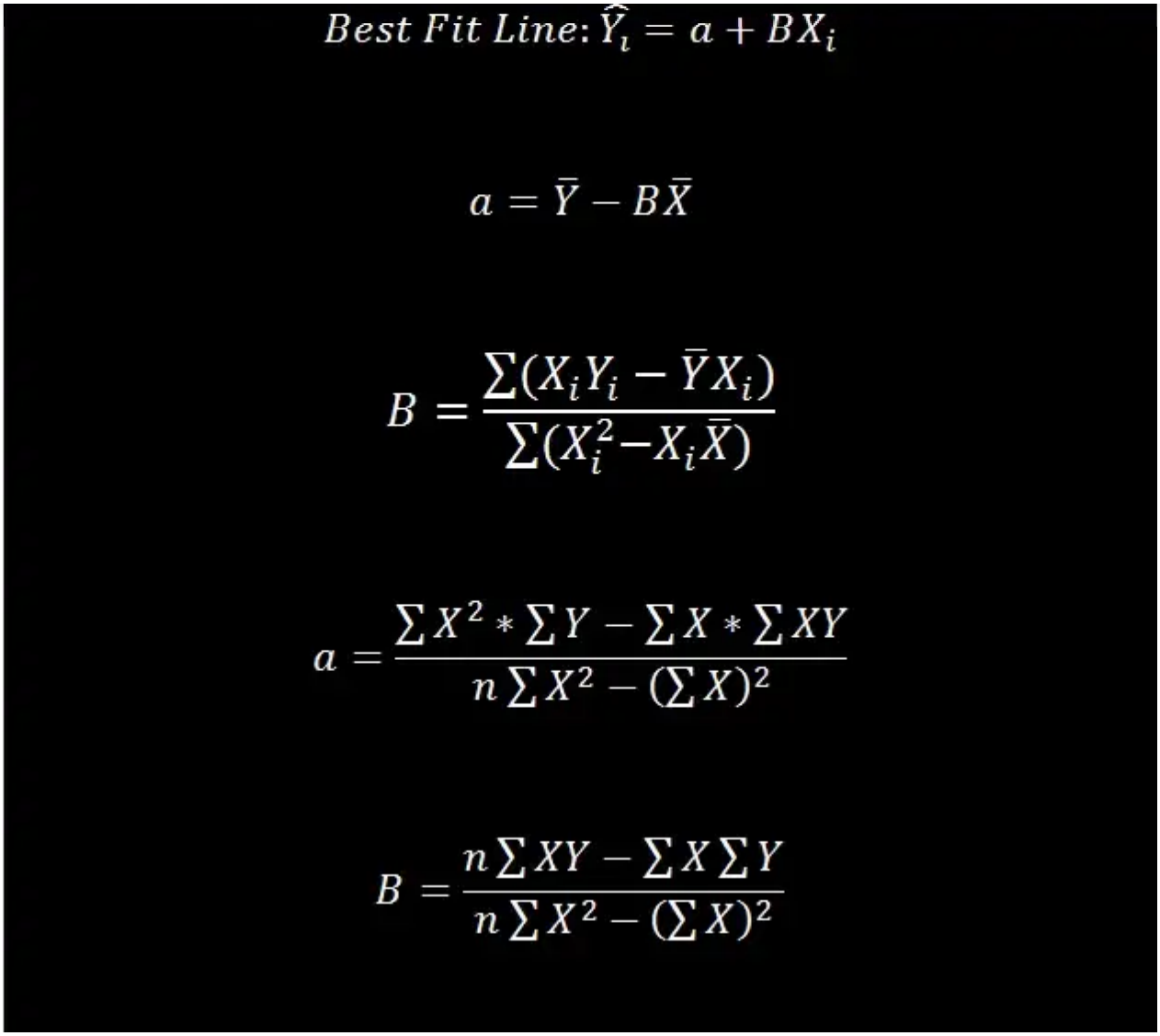


Figure 69: Summary of simple linear regression formulas.

Simple Linear Regression Python Implementation from Scratch:

In the following Python code for simple linear regression, we will not use a Python library to find the optimal parameters for the regression line; instead, we will use the formulas derived earlier to find the regression (best fit) line for our dataset.

1. Import the required libraries:

```
import pandas as pd
import matplotlib.pyplot as plt
```

Figure 70: Import the required libraries.

2. Read the CSV file:

```
#Read the csv file:

data = pd.read_csv("Fuel_Consumption.csv")
data.head()
```

	MODELYEAR	MAKE	MODEL	VEHICLECLASS	ENGINESIZE	CYLINDERS	TRANSMISSION	FUELTYPE	FUELCONSUMPTION_CITY
0	2014	ACURA	ILX	COMPACT	2.0	4	AS5	Z	9.9
1	2014	ACURA	ILX	COMPACT	2.4	4	M6	Z	11.2
2	2014	ACURA	ILX HYBRID	COMPACT	1.5	4	AV7	Z	6.0
3	2014	ACURA	MDX 4WD	SUV - SMALL	3.5	6	AS6	Z	12.7
4	2014	ACURA	RDX AWD	SUV - SMALL	3.5	6	AS6	Z	12.1

Figure 71: Reading the CSV file.

3. Get the list of columns in our dataset:

```
#Columns in our dataset:

data.columns
```

```
Index(['MODELYEAR', 'MAKE', 'MODEL', 'VEHICLECLASS', 'ENGINESIZE', 'CYLINDERS',
       'TRANSMISSION', 'FUELTYPE', 'FUELCONSUMPTION_CITY',
       'FUELCONSUMPTION_HWY', 'FUELCONSUMPTION_COMB',
       'FUELCONSUMPTION_COMB_MPG', 'CO2EMISSIONS'],
      dtype='object')
```

Figure 72: The columns of our dataset.

4. Checking for null values:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1067 entries, 0 to 1066
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   MODELYEAR                 1067 non-null   int64
 1   MAKE                      1067 non-null   object
 2   MODEL                     1067 non-null   object
 3   VEHICLECLASS              1067 non-null   object
 4   ENGINESIZE                1067 non-null   float64
 5   CYLINDERS                 1067 non-null   int64
 6   TRANSMISSION              1067 non-null   object
 7   FUELTYPE                  1067 non-null   object
 8   FUELCONSUMPTION_CITY      1067 non-null   float64
 9   FUELCONSUMPTION_HWY       1067 non-null   float64
 10  FUELCONSUMPTION_COMB      1067 non-null   float64
 11  FUELCONSUMPTION_COMB_MPG  1067 non-null   int64
 12  CO2EMISSIONS              1067 non-null   int64
dtypes: float64(4), int64(4), object(5)
memory usage: 108.5+ KB
```

Figure 73: Checking for null values.

5. Selecting columns to build our model:

```
#Select only those features from our dataset which we are going to use for predictions:

data = data[["ENGINESIZE","CO2EMISSIONS"]]
print(data.head())
```

```
   ENGINESIZE  CO2EMISSIONS
0         2.0           196
1         2.4           221
2         1.5           136
3         3.5           255
4         3.5           244
```

Figure 74: Columns of interest.

6. Plot the data on the scatterplot:

```
plt.scatter(data["ENGINESIZE"],data["CO2EMISSIONS"])
plt.title("ENGINESIZE VS CO2EMISSIONS")
plt.xlabel("Enginesize")
plt.ylabel("Emission")
plt.show()
```

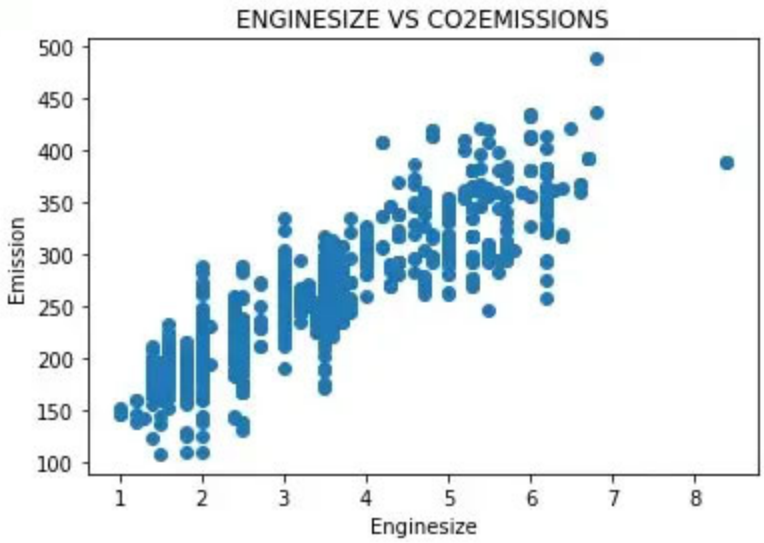


Figure 75: Plotting the data on the scatterplot.

7. Divide the data into training and testing dataset:

```
#Taking 80% of the data for training and 20% for testing:
num = int(len(data)*0.8)

#Training data:
train = data[:num]

#Testing data:
test = data[num:]

print ("Data: ",len(data))
print ("Train: ",len(train))
print ("Test: ",len(test))
```

```
Data:  1067
Train:  853
Test:  214
```

Figure 76: Dividing the data into a testing/training dataset.

8. Main function to calculate the coefficients of the linear best fit:

$$Slope = B = \frac{\sum(X_i Y_i - \bar{Y} X_i)}{\sum(X_i^2 - X_i \bar{X})}$$

$$Intercept = a = \bar{Y} - B\bar{X}$$

Figure 77: Slope and intercept equations.

```
    Xi = input_feature
    Yi = output

    #Total number of data points:
    n = len(Xi)

    #X bar:
    Xi_mean = Xi.mean()

    #Y bar:
    Yi_mean = Yi.mean()

    #Sum of X:
    S_Xi = (Xi).sum()

    #Sum of Y:
    S_Yi = (Yi).sum()

    #Sum of (X*Y) multiplied by n:
    S_XiYi = ((Xi*Yi).sum())*n

    #Sum of X*Sum of Y:
    S_Xi_S_Yi = S_Xi*S_Yi

    #Sum of (X*X) multiplied by n:
    S_XiXi = ((Xi*Xi).sum())*n

    #Square of sum of X:
    S_Xi_Square = S_Xi*S_Xi

    #Slope:
    slope = (S_XiYi- S_Xi_S_Yi) / (S_XiXi-S_Xi_Square)

    #Intercept:
    intercept = Yi_mean - slope * Xi_mean

    return slope,intercept
```

Figure 78: The main function to calculate the slope and the intercept.

9. Check the working of the function with dummy data:

```
dummy_output = np.array((1,2,3,4,5))

dummy_slope,dummy_intercept = simple_linear_regression(dummy_input,dummy_output)

print ("Slope : " , dummy_slope)
print ("Intercept :",dummy_intercept)
```

```
Slope :  1.0
Intercept : 0.0
```

Figure 79: Checking the working function with sample data.

10. Plot the dummy data with the regression line:

```
#Graph for dummy data:

plt.scatter([1,2,3,4,5],[1,2,3,4,5])
plt.plot([1,2,3,4,5],[1,2,3,4,5],color="red")
plt.show()
```

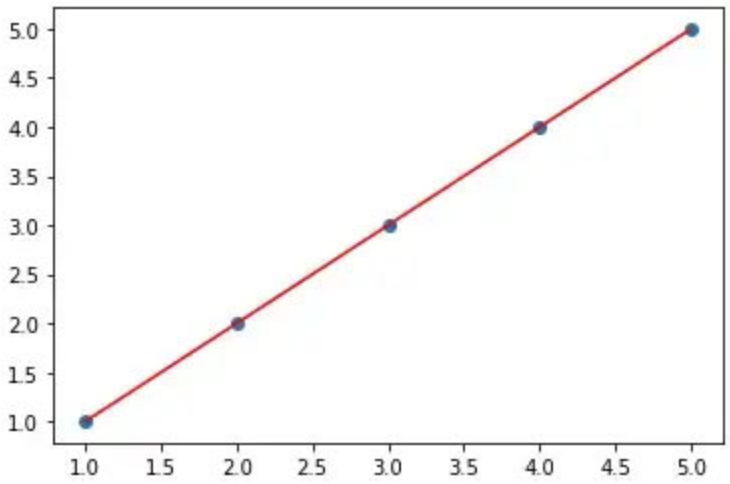


Figure 80: Plotting the data on our scatterplot.

11. Finding the coefficients for our actual dataset:

```
#Training the model with train data:
#Finding the coefficients of best fit line:

actual_slope,actual_intercept = simple_linear_regression(train["ENGINESIZE"],train["CO2EMISSIONS"])

print ("Slope: " ,actual_slope)
print ("Intercept: " ,actual_intercept)
```

12. Plot the regression line with actual data:

```
#Plot the regression line with training data:

plt.scatter(train["ENGINESIZE"],train["CO2EMISSIONS"])
plt.plot(train["ENGINESIZE"],actual_slope*train["ENGINESIZE"]+actual_intercept,color="green")
plt.xlabel("Engine_Size")
plt.ylabel("Emission")
plt.show()
```

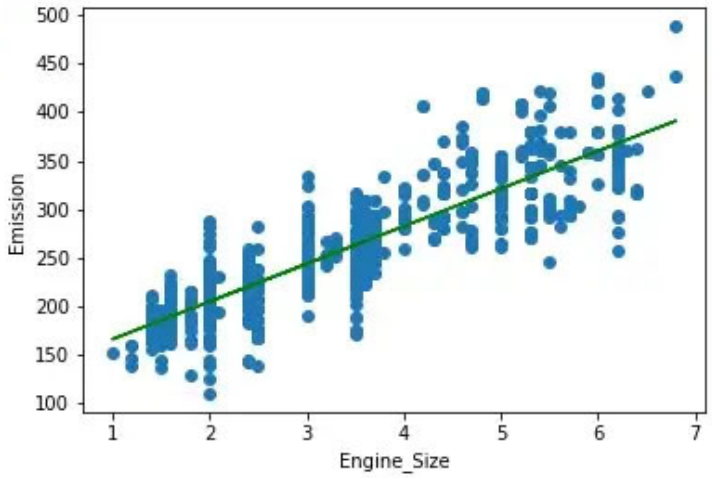


Figure 82: Plotting the data on our scatterplot.

13. Define the prediction function:

```
#Define the prediction function:

def get_regression_prediction(input_features,slope,intercept):
    predicted_value = actual_slope*input_features + actual_intercept
    return predicted_value
```

Figure 83: Defining our prediction function.

14. Predicting the values based on the prediction function:

```
estimated_emission = get_regression_prediction(my_engine_size,actual_slope,actual_intercept)

print ("Estimated Emission: ",estimated_emission)
```

```
Estimated Emission:  321.1455186891026
```

Figure 84: Predicting the values based on the prediction function.

15. Predicting values for the whole dataset:

```
#Predicting values for the whole dataset:

y_pred = get_regression_prediction(data["ENGINESIZE"],actual_slope,actual_intercept)
print(y_pred)
```

```
0       204.760147
1       220.278197
2       185.362585
3       262.952833
4       262.952833
           ...
1062    243.555271
1063    251.314296
1064    243.555271
1065    251.314296
1066    251.314296
Name: ENGINESIZE, Length: 1067, dtype: float64
```

Figure 85: Predicting the values of the whole dataset.

16. Plotting the test data with the regression line:

```
plt.plot(test["ENGINESIZE"],actual_slope*test["ENGINESIZE"]+actual_intercept,color="red")
plt.xlabel("Engine_Size")
plt.ylabel("Emission")
plt.show()
```

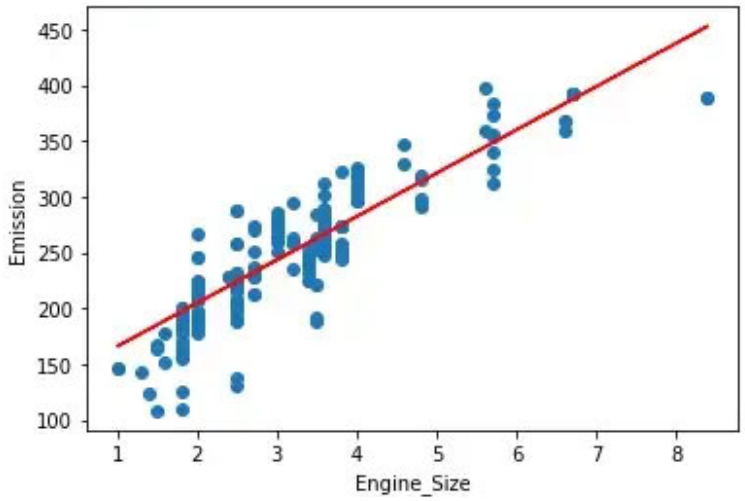


Figure 86: Plotting the test data on the scatter plot with the regression line.

17. Plot the training data with the regression line:

```
#Plot the regression line for the training data:

plt.scatter(train["ENGINESIZE"],train["CO2EMISSIONS"])
plt.plot(train["ENGINESIZE"],actual_slope*train["ENGINESIZE"]+actual_intercept,color="red")
plt.xlabel("Engine_Size")
plt.ylabel("Emission")
plt.show()
```

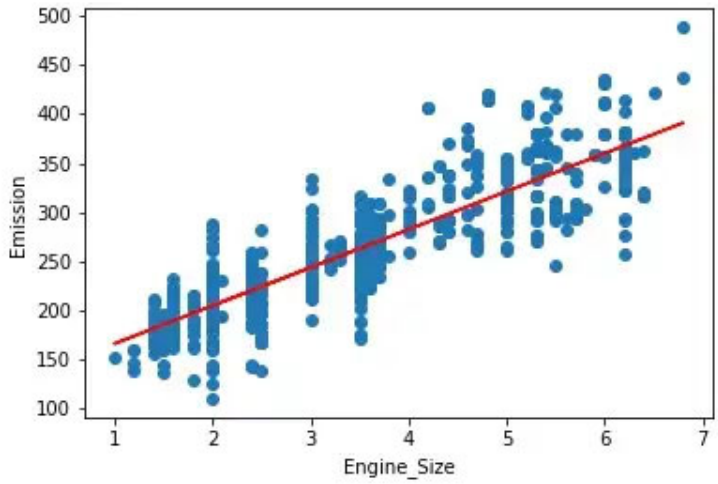


Figure 87: Plotting the training data with the regression line

```
#Plot the regression line for complete data:

plt.scatter(data["ENGINESIZE"],data["CO2EMISSIONS"])
plt.plot(data["ENGINESIZE"],actual_slope*data["ENGINESIZE"]+actual_intercept,color="red")
plt.xlabel("Engine_Size")
plt.ylabel("Emission")
plt.show()
```

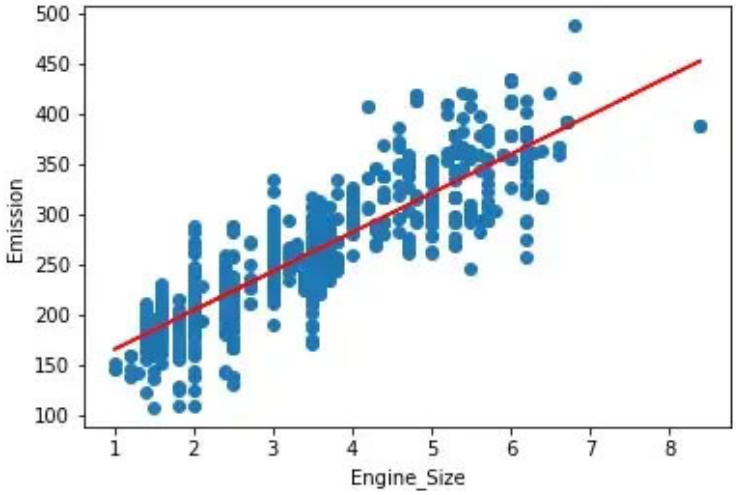


Figure 88: Plotting the data on the scatterplot with the regression line.

19. Create a data frame for actual and predicted values:

```
#Create a dataframe for Actual and Predicted values:

A_P_data = pd.DataFrame({"Actual":data["CO2EMISSIONS"] , "Predicted" :y_pred})
print(A_P_data.head())
```

```
   Actual   Predicted
0     196  204.760147
1     221  220.278197
2     136  185.362585
3     255  262.952833
4     244  262.952833
```

Figure 89: Actual vs. predicted values.

19. Plot the bar graph for actual and predicted values:

```
plt.show()
```

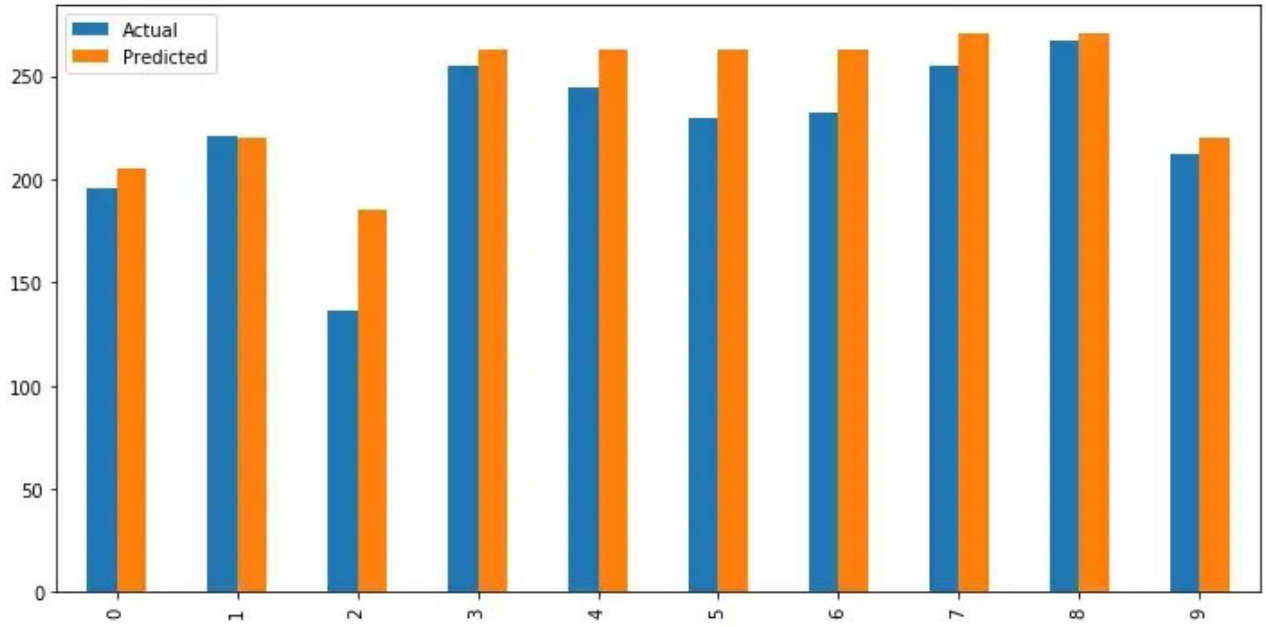


Figure 90: Bar graph for actual vs. predicted values.

20. Residual Sum of Square:

```
#Error calculation using Residual Sum of Squares:

def residual_sum_of_squares(input_feature,output,slope,intercept):
    prediction = slope*input_feature + intercept

    residual = (output - prediction)

    RSS = (residual*residual).sum()

    return(RSS)
```

Figure 91: Error calculation function.

21. Calculating error:

```
#Calculating error in prediction for our dataset:

print ("RSS : ",residual_sum_of_squares(test["ENGINESIZE"],test["CO2EMISSIONS"],actual_slope,actual_intercept))
```

```
RSS :  159741.10295932254
```

it is always good practice to know how these libraries perform such mathematical calculations.

Next, we will use the Scikit-learn library in Python to find the linear-best-fit regression line on the same data set. In the following code, we will see a straightforward way to calculate a simple linear regression using Scikit-learn.

Simple Linear Regression Using Scikit-learn:

1. Import the required libraries:

```
# Import required libraries :

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model
```

Figure 93: Importing the required libraries for our Scikit implementation.

2. Read the CSV file:

```
# Read csv file :

data = pd.read_csv("Fuel_Consumption.csv")
data.head()
```

	MODELYEAR	MAKE	MODEL	VEHICLECLASS	ENGINESIZE	CYLINDERS	TRANSMISSION	FUELTYPE	FUELCONSUMPTION_CITY
0	2014	ACURA	ILX	COMPACT	2.0	4	AS5	Z	9.9
1	2014	ACURA	ILX	COMPACT	2.4	4	M6	Z	11.2
2	2014	ACURA	ILX HYBRID	COMPACT	1.5	4	AV7	Z	6.0
3	2014	ACURA	MDX 4WD	SUV - SMALL	3.5	6	AS6	Z	12.7
4	2014	ACURA	RDX AWD	SUV - SMALL	3.5	6	AS6	Z	12.1

Figure 94: Reading the CSV file.

```
#Select only those features from our dataset which we are going to use for predictions:

data = data[["ENGINESIZE","CO2EMISSIONS"]]
print(data.head())
```

```
   ENGINESIZE  CO2EMISSIONS
0         2.0           196
1         2.4           221
2         1.5           136
3         3.5           255
4         3.5           244
```

Figure 95: Feature selection.

4. Plotting the data points on a scatter plot:

```
#Scatter Plot:
#ENGINESIZE VS CO2EMISSIONS :

plt.scatter(data["ENGINESIZE"],data["CO2EMISSIONS"])
plt.title("ENGINESIZE VS CO2EMISSIONS")
plt.xlabel("Enginesize")
plt.ylabel("Emission")
plt.show()
```

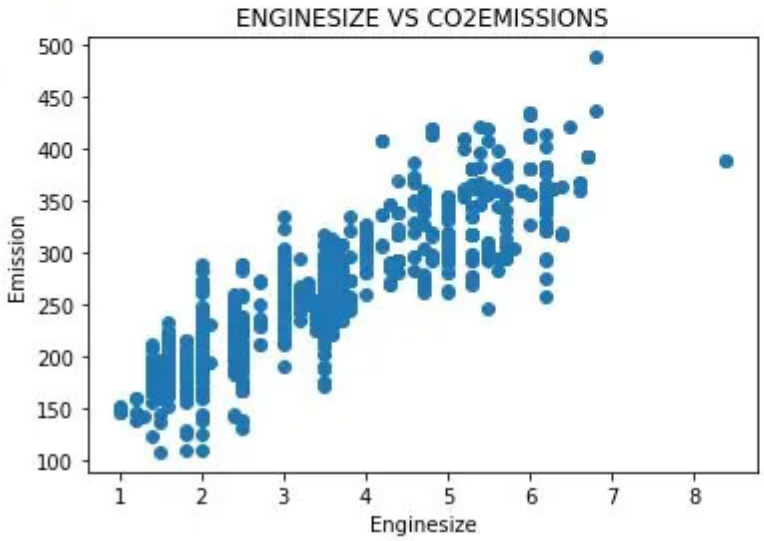


Figure 96: Plotting the data points on a scatter plot.

5. Dividing data into testing and training dataset:

```
#Training data:
train = data[:num]

#Testing data:
test = data[num:]

print ("Data: ",len(data))
print ("Train: ",len(train))
print ("Test: ",len(test))
```

```
Data:  1067
Train:  853
Test:  214
```

Figure 97: Dividing our data into test and training.

6. Training the model:

```
#Training the model:

regr = linear_model.LinearRegression()

train_x = np.array(train[["ENGINESIZE"]])
train_y = np.array(train[["CO2EMISSIONS"]])

regr.fit(train_x,train_y)

coefficients = regr.coef_
intercept = regr.intercept_

#Slope:
print ("Slope: ",coefficients[0])

#Intercept:
print("Intercept: ",intercept)
```

```
Slope:  [38.79512384]
Intercept:  [127.16989951]
```

Figure 98: Training our model.

7. Predicting values for a complete dataset:

```
predicted_data[0:5]
```

```
array([[204.76014718],
       [220.27819672],
       [185.36258526],
       [262.95283294],
       [262.95283294]])
```

Figure 99: Predicting the values.

8. Predicting values for training data:

```
predicted_train = regr.predict(train[["ENGINESIZE"]])
predicted_train[0:5]
```

```
array([[204.76014718],
       [220.27819672],
       [185.36258526],
       [262.95283294],
       [262.95283294]])
```

Figure 100: Predicting the values for our training dataset.

9. Predicting values for testing data:

```
predicted_test = regr.predict(test[["ENGINESIZE"]])
predicted_test[0:5]
```

```
array([[224.1577091 ],
       [262.95283294],
       [224.1577091 ],
       [224.1577091 ],
       [197.00112241]])
```

Figure 101: Predicting the values for our testing dataset.

10. Plotting regression line for complete data:

```
plt.plot(data["ENGINESIZE"],predicted_data,color="red")
plt.xlabel("Engine_Size")
plt.ylabel("Emission")
plt.show()
```

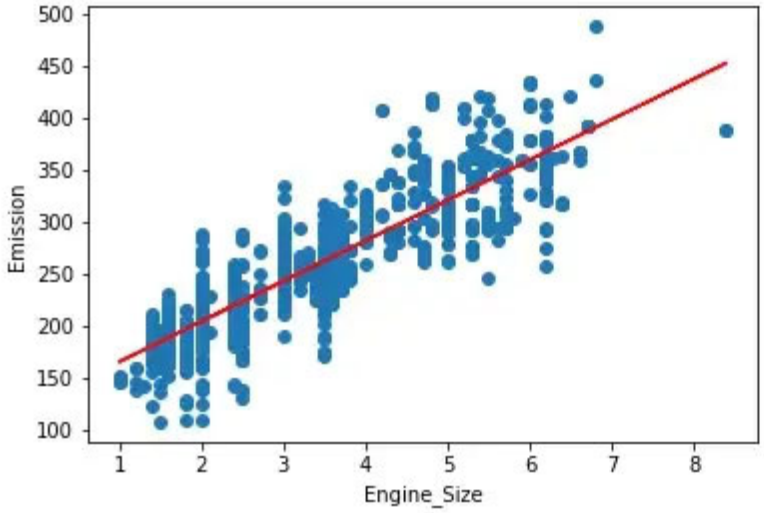
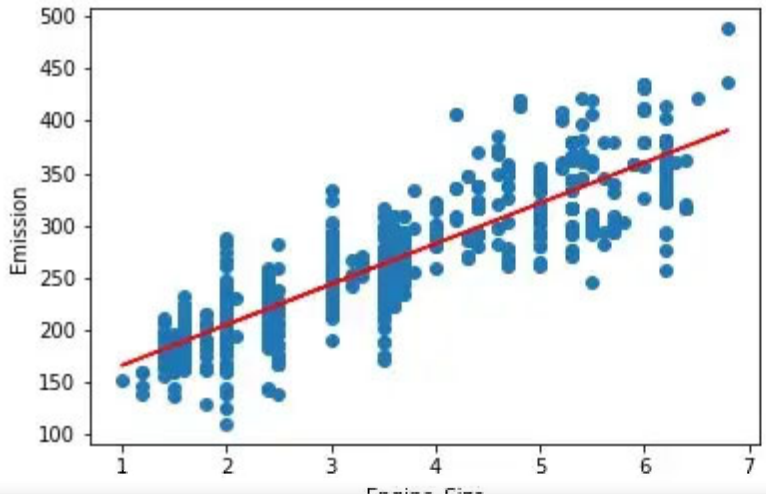


Figure 102: Plotting the data on the scatter plot with a regression line.

11. Plotting regression line with training data:

```
#Plot the regression line for training data:

plt.scatter(train["ENGINESIZE"],train["CO2EMISSIONS"])
plt.plot(train["ENGINESIZE"],predicted_train,color="red")
plt.xlabel("Engine_Size")
plt.ylabel("Emission")
plt.show()
```



```
#Plot the regression line for testing data:

plt.scatter(test["ENGINESIZE"],test["CO2EMISSIONS"])
plt.plot(test["ENGINESIZE"],predicted_test,color="red")
plt.xlabel("Engine_Size")
plt.ylabel("Emission")
plt.show()
```

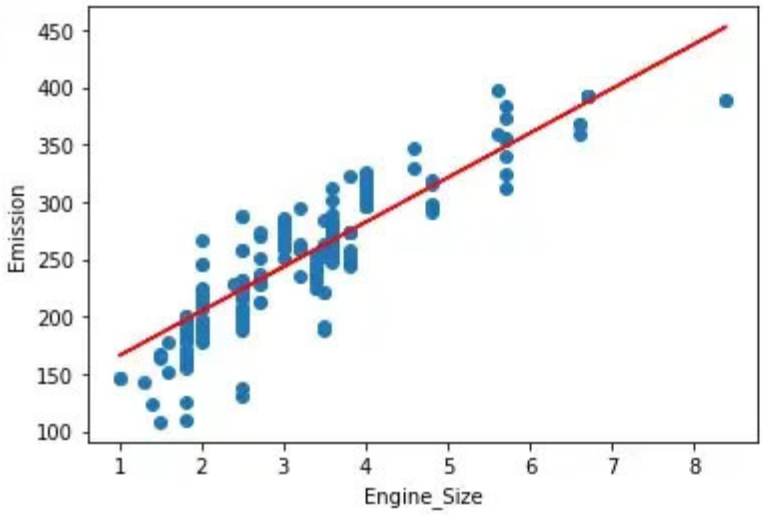


Figure 104: Plotting the testing data on our scatter plot with a regression line.

13. Create dataframe for actual and predicted data points:

```
#Create a dataframe for Actual and Predicted values:

A_P_data = pd.DataFrame({"Actual":data["CO2EMISSIONS"],"Predicted":predicted_data[:][0][0]})
print(A_P_data.head())
```

```
   Actual   Predicted
0     196  204.760147
1     221  204.760147
2     136  204.760147
3     255  204.760147
4     244  204.760147
```

Figure 105: Data frame for actual and predicted values.

14. Plotting the bar graph for actual and predicted values:

```
plt.show()
```

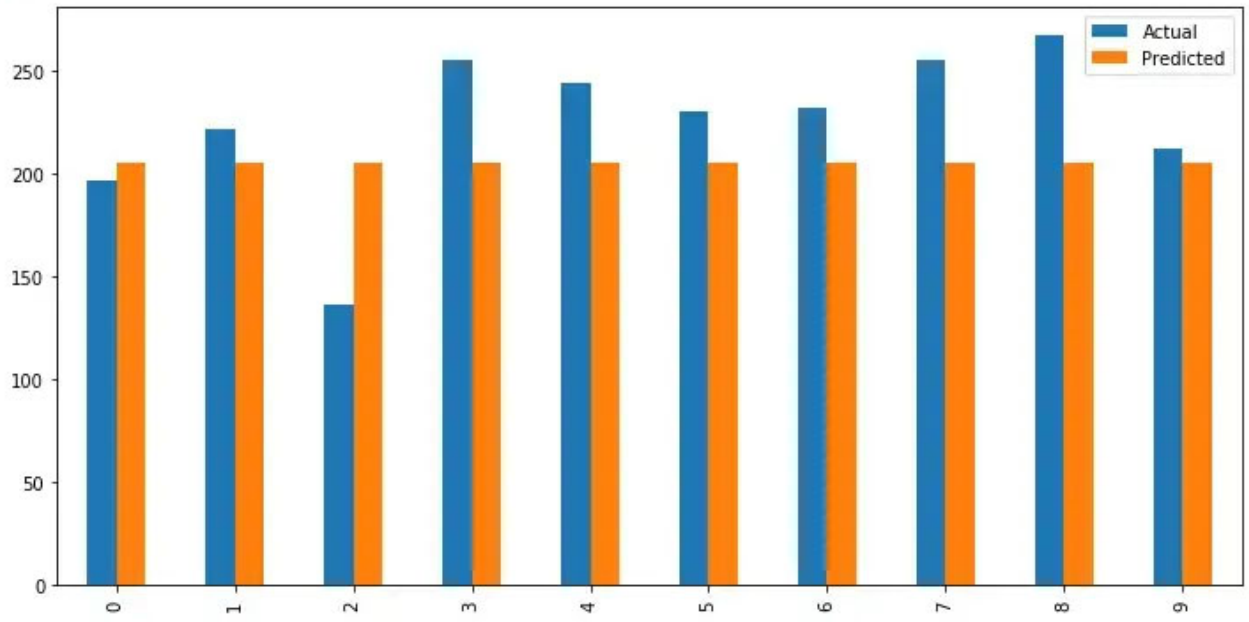


Figure 106: Plotting a bar graph for actual vs. predicted values.

15. Calculating error in prediction:

```
#Error calculations:

test_x = np.array(test[['ENGINESIZE']])
test_y = np.array(test[['CO2EMISSIONS']])

predicted_y= regr.predict(test_x)

res = (predicted_y - test_y)
RSS = (res*res).sum()

print("Residual Sum of Squares: ",RSS)
```

```
Residual Sum of Squares:  159741.10295932172
```

Figure 107: Calculating the error in prediction.

Here we can see that we got the same output even if we use the Scikit-learn library. Therefore we can be sure that all the calculations we performed and derivations we understood are precisely accurate.

Please note that there are other methods to calculate the prediction error, and we will

Thank you for reading!

Buy Pratik a Coffee!

DISCLAIMER: The views expressed in this article are those of the author(s) and do not represent the views of Carnegie Mellon University. These writings do not intend to be final products, yet rather a reflection of current thinking, along with being a catalyst for discussion and improvement.

Published via Towards AI

Resources:

Google colab implementation.

Github repository.

References:

[1] What is simple linear regression, Penn State, https://online.stat.psu.edu/stat462/node/91/

[2] scikit-learn, Getting Started, https://scikit-learn.org/

[3] SpongeBob SquarePants, Wikipedia