CHAPTER 5
Linear Regression

One of the most practical techniques in data analysis is fitting a line through observed data points to show a relationship between two or more variables. A *regression* attempts to fit a function to observed data to make predictions on new data. A *linear regression* fits a straight line to observed data, attempting to demonstrate a linear relationship between variables and make predictions on new data yet to be observed.

It might make more sense to see a picture rather than read a description of linear regression. There is an example of a linear regression in Figure 5-1.

Linear regression is a workhorse of data science and statistics and not only applies concepts we learned in previous chapters but sets up new foundations for later topics like neural networks (Chapter 7) and logistic regression (Chapter 6). This relatively simple technique has been around for more than two hundred years and contemporarily is branded as a form of machine learning.

Machine learning practitioners often take a different approach to validation, starting with a train-test split of the data. Statisticians are more likely to use metrics like prediction intervals and correlation for statistical significance. We will cover both schools of thought so readers can bridge the ever-widening gap between the two disciplines, and thus find themselves best equipped to wear both hats.

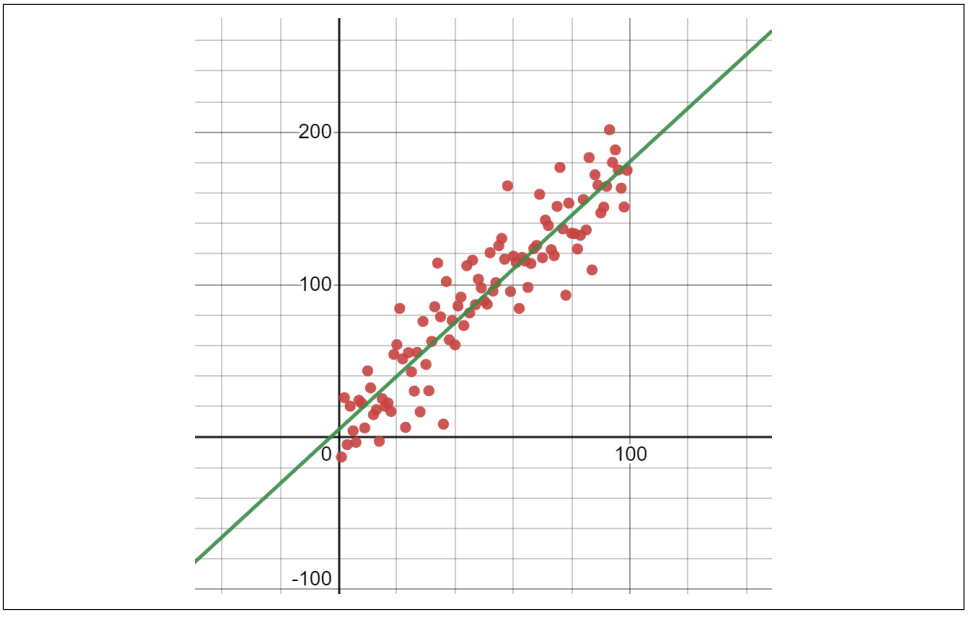

Figure 5-1. Example of a linear regression, which fits a line to observed data

Wait, Regression Is Machine Learning?

Machine learning has several techniques under its umbrella, but the one with the most use cases currently is supervised learning, and regressions play a big role here. This is why linear regression is branded as a form of machine learning. Confusingly, statisticians may refer to their regression models as *statistical learning*, whereas data science and machine learning professionals call their models *machine learning*.

While supervised learning is often regression, unsupervised machine learning is more about clustering and anomaly detection. Reinforcement learning often pairs supervised machine learning with simulation to rapidly generate synthetic data.

We will learn two more forms of supervised machine learning in Chapter 6 on logistic regression and Chapter 7 on neural networks.

A Basic Linear Regression

I want to study the relationship between the age of a dog and the number of veterinary visits it had. In a fabricated sample we have 10 random dogs. I am a fan of understanding complex techniques with simple datasets (real or otherwise), so we understand the strengths and limitations of the technique without complex data muddying the water. Let's plot this dataset as shown in Figure 5-2.

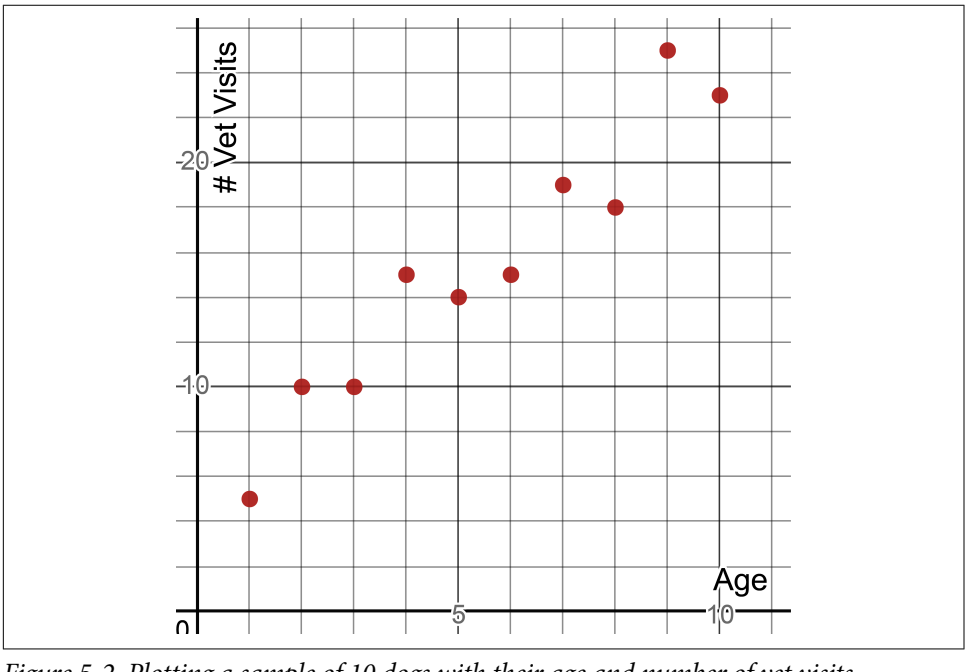

Figure 5-2. Plotting a sample of 10 dogs with their age and number of vet visits

We can clearly see there is a *linear correlation* here, meaning when one of these variables increases/decreases, the other increases/decreases in a roughly proportional amount. We could draw a line through these points to show a correlation like this in Figure 5-3.

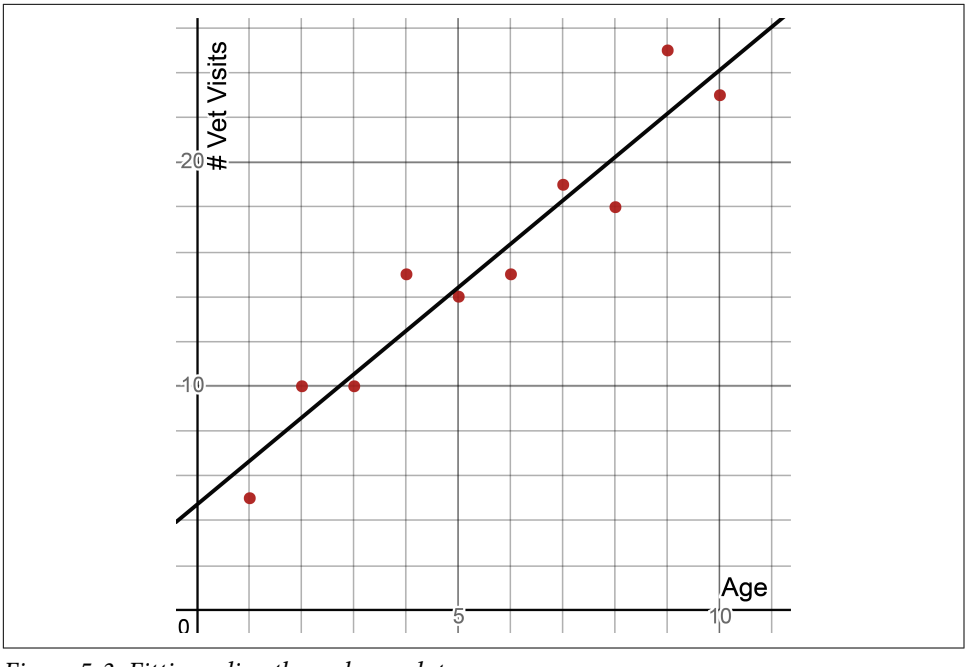

Figure 5-3. Fitting a line through our data

I will show how to calculate this fitted line later in this chapter. We will also explore how to calculate the quality of this fitted line. For now, let's focus on the benefits of performing a linear regression. It allows us to make predictions on data we have not seen before. I do not have a dog in my sample that is 8.5 years old, but I can look at this line and estimate the dog will have 21 veterinary visits in its life. I just look at the line where $x = 8.5$ and I see that $y = 21.218$ as shown in Figure 5-4. Another benefit is we can analyze variables for possible relationships and hypothesize that correlated variables are causal to one another.

Now what are the downsides of a linear regression? I cannot expect that every outcome is going to fall *exactly* on that line. After all, real-world data is noisy and never perfect and will not follow a straight line. It may not remotely follow a straight line at all! There is going to be error around that line, where the point will fall above or below the line. We will cover this mathematically when we talk about p-values, statistical significance, and prediction intervals, which describes how reliable our linear regression is. Another catch is we should not use the linear regression to make predictions outside the range of data we have, meaning we should not make predictions where $x < 0$ and $x > 10$ because we do not have data outside those values.

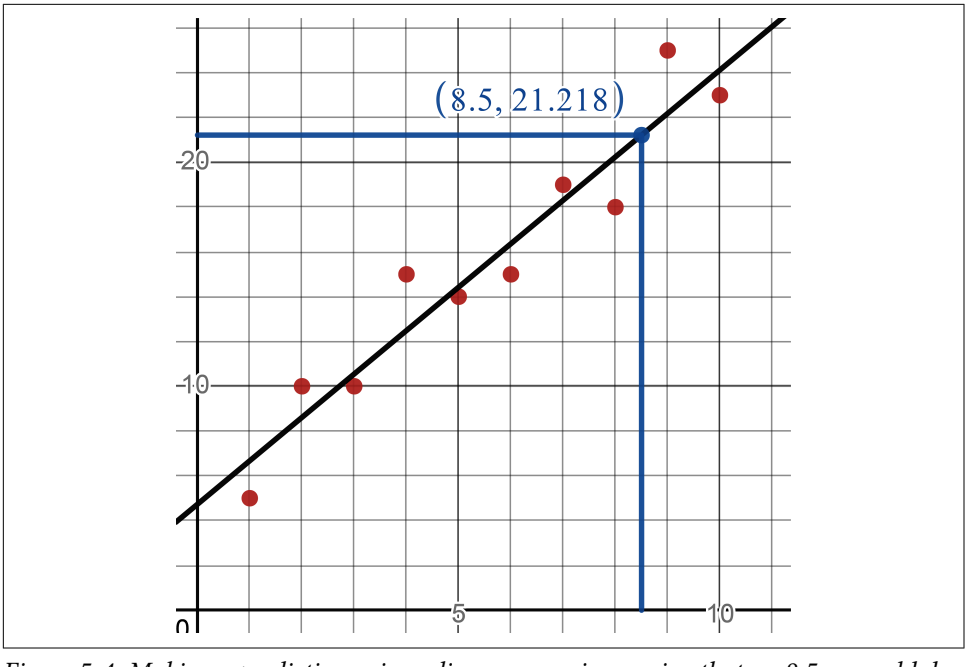

Figure 5-4. Making a prediction using a linear regression, seeing that an 8.5-year-old dog is predicted to have about 21.2 vet visits

Don't Forget Sampling Bias!

We should question this data and how it was sampled to detect bias. Was this at a single veterinary clinic? Multiple random clinics? Is there self-selection bias by using veterinary data, only polling dogs that visit the vet? If the dogs were sampled in the same geography, can that sway the data? Perhaps dogs in hot desert climates go to vets more for heat exhaustion and snake bites, and this would inflate our veterinary visits in our sample.

As discussed in Chapter 3, it has become fashionable to make data an oracle for truth. However data is simply a sample from a population, and we need to practice discernment on how well represented our sample is. Be just as interested (if not more) in where the data comes from and not just what the data says.

Basic Linear Regression with SciPy

We have a lot to learn regarding linear regression in this chapter, but let's start out with some code to execute what we know so far.

There are plenty of platforms to perform a linear regression, from Excel to Python and R. But we will stick with Python in this book, starting with scikit-learn to do the

work for us. I will show how to build a linear regression "from scratch" later in this chapter so we grasp important concepts like gradient descent and least squares.

Example 5-1 is how we use scikit-learn to perform a basic, unvalidated linear regression on the sample of 10 dogs. We pull in this data using Pandas, convert it into NumPy arrays, perform linear regression using scikit-learn, and use Plotly to display it in a chart.

Example 5-1. Using scikit-learn to do a linear regression

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

# Import points
df = pd.read_csv('https://bit.ly/3goOAnt', delimiter=",")

# Extract input variables (all rows, all columns but last column)
X = df.values[:, :-1]

# Extract output column (all rows, last column)
Y = df.values[:, -1]

# Fit a line to the points
fit = LinearRegression().fit(X, Y)

# m = 1.7867224, b = -16.51923513
m = fit.coef_.flatten()
b = fit.intercept_.flatten()
print("m = {0}".format(m))
print("b = {0}".format(b))

# show in chart
plt.plot(X, Y, 'o') # scatterplot
plt.plot(X, m*X+b) # line
plt.show()
```

First we import the data from this CSV on GitHub. We separate the two columns into *X* and *Y* datasets using Pandas. We then `fit()` the `LinearRegression` model to the input *X* data and the output *Y* data. We can then get the *m* and *b* coefficients that describe our fitted linear function.

In the plot, sure enough you will get a fitted line running through these points shown in Figure 5-5.

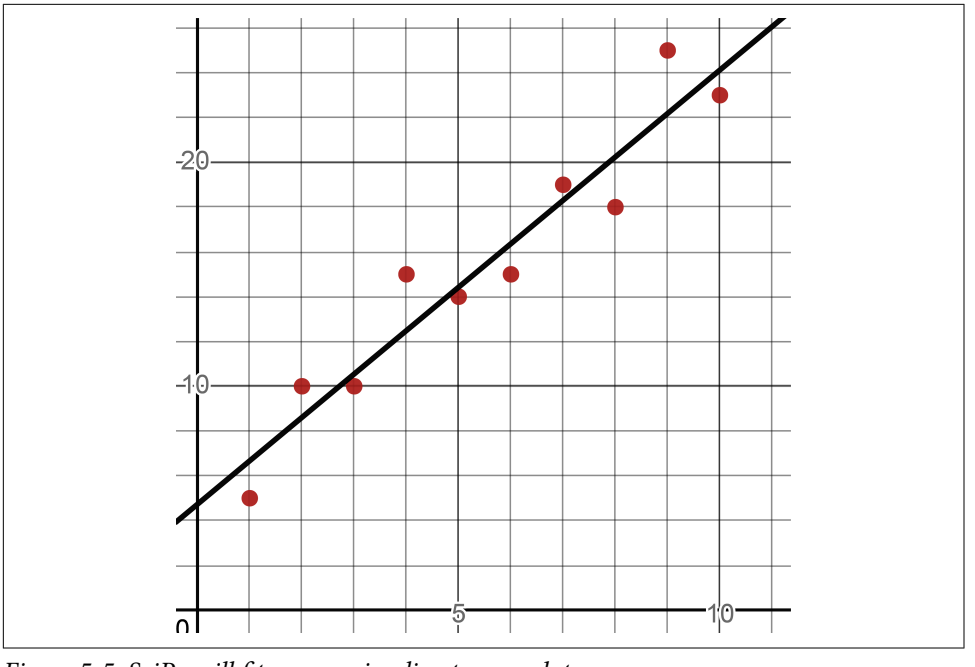

Figure 5-5. SciPy will fit a regression line to your data

What decides the best fit line to these points? Let's discuss that next.

Residuals and Squared Errors

How do statistics tools like scikit-learn come up with a line that fits to these points? It comes down to two questions that are fundamental to machine learning training:

- What defines a "best fit"?
- How do we get to that "best fit"?

The first question has a pretty established answer: we minimize the squares, or more specifically the sum of the squared residuals. Let's break that down. Draw any line through the points. The *residual* is the numeric difference between the line and the points, as shown in Figure 5-6.

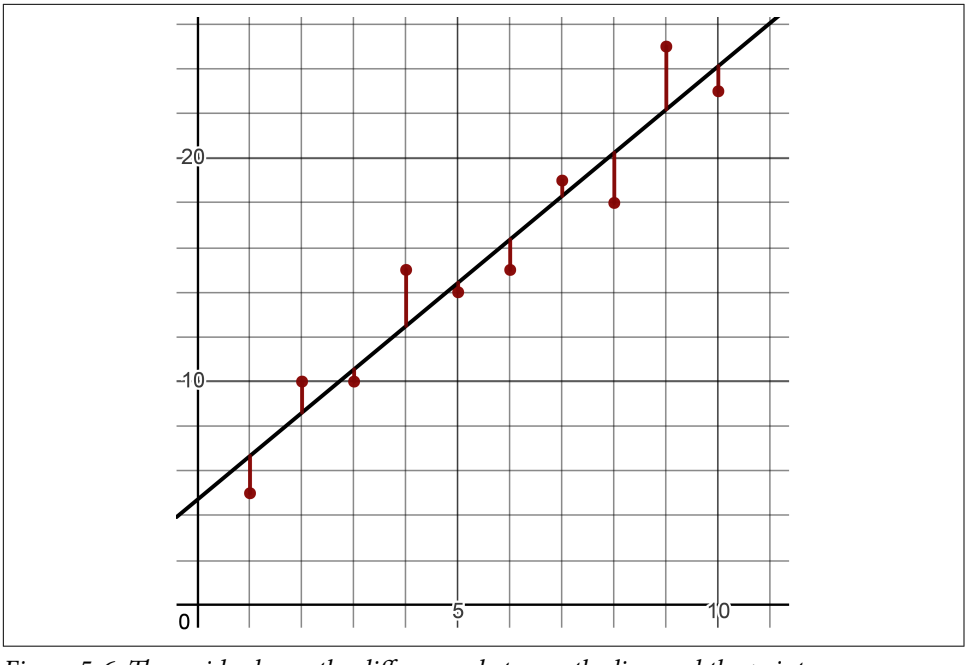

Figure 5-6. The residuals are the differences between the line and the points

Points above the line will have a positive residual, and points below the line will have a negative residual. In other words, it is the subtracted difference between the predicted y-values (derived from the line) and the actual y-values (which came from the data). Another name for residuals are *errors*, because they reflect how wrong our line is in predicting the data.

Let's calculate these differences between these 10 points and the line $y = 1.93939x + 4.73333$ in Example 5-2 and the residuals for each point in Example 5-3.

Example 5-2. Calculating the residuals for a given line and data

```
import pandas as pd

# Import points
points = pd.read_csv('https://bit.ly/3goOAnt', delimiter=",").itertuples()

# Test with a given line
m = 1.93939
b = 4.73333

# Calculate the residuals
for p in points:
    y_actual = p.y
    y_predict = m*p.x + b
```

```
residual = y_actual - y_predict
print(residual)
```

Example 5-3. The residuals for each point

```
-1.67272
1.3878900000000005
-0.5515000000000008
2.5091099999999997
-0.4302799999999998
-1.3696699999999993f
0.6909400000000012
-2.2484499999999983
2.812160000000002
-1.1272299999999973
```

If we are fitting a straight line through our 10 data points, we likely want to minimize these residuals in total so there is the least gap possible between the line and points. But how do we measure the "total"? The best approach is to take the *sum of squares*, which simply squares each residual, or multiplies each residual by itself, and sums them. We take each actual y-value and subtract from it the predicted y-value taken from the line, then square and sum all those differences.

Why Not Absolute Values?

You might wonder why we have to square the residuals before summing them. Why not just add them up without squaring? That will not work because the negatives will cancel out the positives. What if we add the absolute values, where we turn all negative values into positive values? That sounds promising, but absolute values are mathematically inconvenient. More specifically, absolute values do not work well with calculus derivatives that we are going to use later for gradient descent. This is why we choose the squared residuals as our way of totaling the loss.

A visual way to think of it is shown in Figure 5-7, where we overlay a square on each residual and each side is the length of the residual. We sum the area of all these squares, and later we will learn how to find the minimum sum we can achieve by identifying the best m and b.

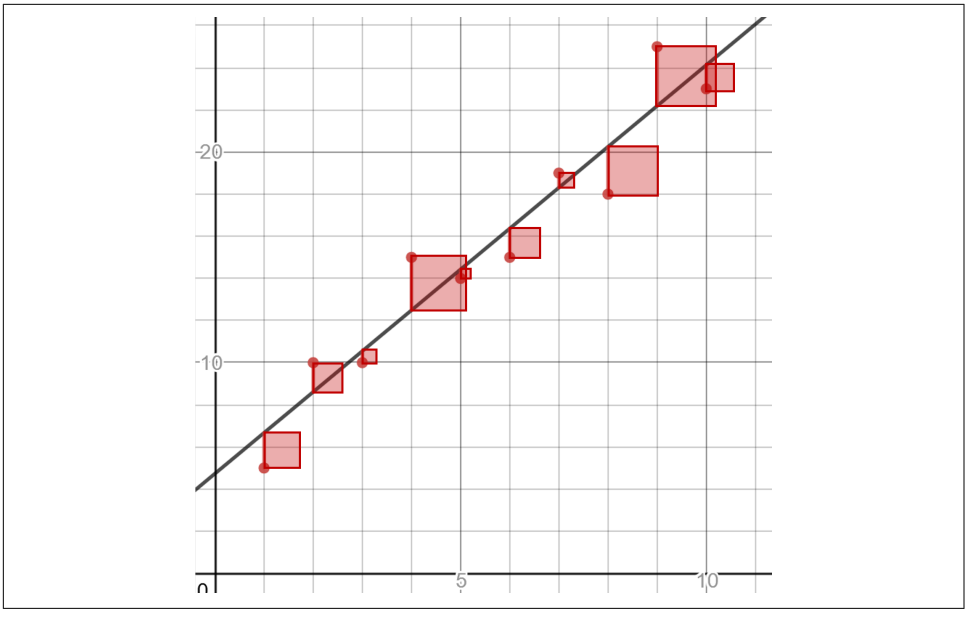

Figure 5-7. Visualizing the sum of squares, which would be the sum of all areas where each square has a side length equal to the residual

Let's modify our code in Example 5-4 to find the sum of squares.

Example 5-4. Calculating the sum of squares for a given line and data

```
import pandas as pd

# Import points
points = pd.read_csv("https://bit.ly/2KF29Bd").itertuples()

# Test with a given line
m = 1.93939
b = 4.73333

sum_of_squares = 0.0

# calculate sum of squares
for p in points:
    y_actual = p.y
    y_predict = m*p.x + b
    residual_squared = (y_predict - y_actual)**2
    sum_of_squares += residual_squared

print("sum of squares = {}".format(sum_of_squares))
# sum of squares = 28.096969704500005
```