Machine Learning with Java

With Fundamentals of Machine Learning

Thomas Nield

Thomas Nield – About the Speaker

Wrote three books on SQL, Java, and data science.

Instructor at University of Southern California.

Instructor at O'Reilly Media.

Co-founder of Yawman Arrow, creating handheld flight simulation hardware (http://yawmanflight.com).

Section I
Linear Regression

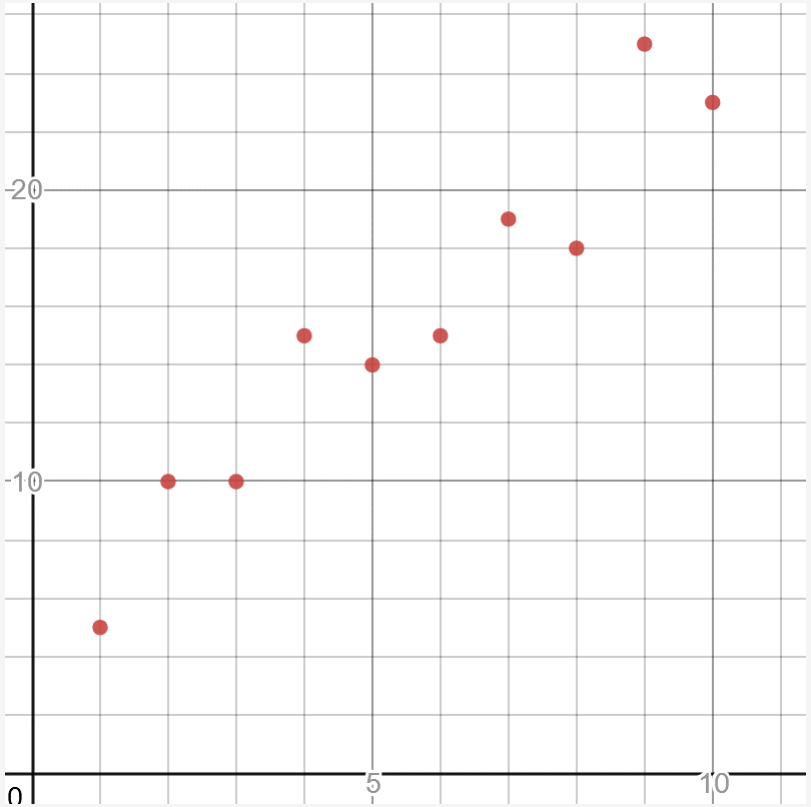
Simple Linear Regression

In practicality, machine learning is often about two tasks: regression and classification.

Let's start with regression, particularly the simplest one called **linear regression** which finds the best fit straight line through some points.

Here is a simple 2-dimensional plot of two variables, where x is independent and y is dependent.

- **Independent variables** are observed values that will serve as inputs into a function.
- **Dependent variables** are the outputted variables derived off the independent variables.



https://www.desmos.com/calculator/fmhotfn3qm

Simple Linear Regression

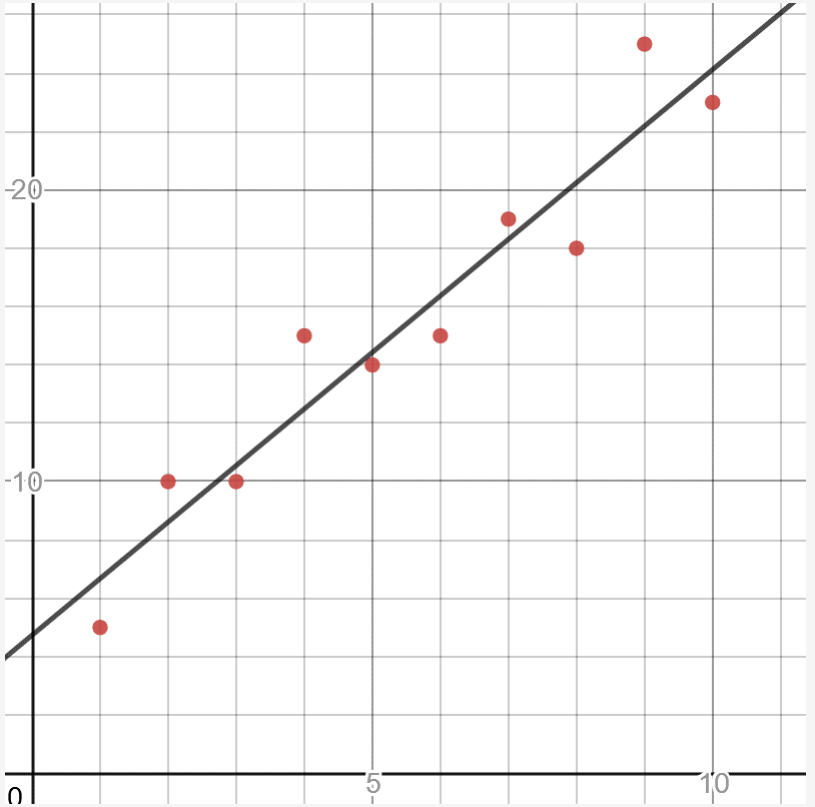
For a simple linear regression, we want to find a function $\boldsymbol{y} = \boldsymbol{mx} + \boldsymbol{b}$ that best fits to these points.

$$y = mx + b$$

We already know the $\boldsymbol{x}$ and $\boldsymbol{y}$ values from our existing data (the red points).

So the missing information is "what $\boldsymbol{m}$ and $\boldsymbol{b}$ values will create the best fit line"?

But before we solve for the $\boldsymbol{m}$ and $\boldsymbol{b}$ values, let's first ask "what defines a best fit anyway?"



https://www.desmos.com/calculator/fmhotfn3qm

Simple Linear Regression

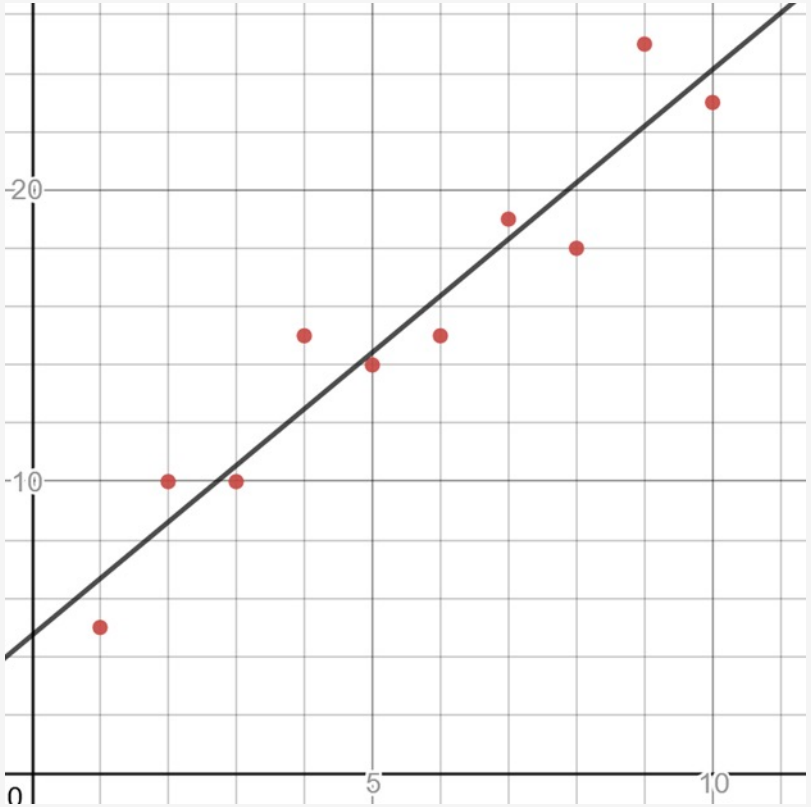
Pretend you drew any straight line through the points.

You will not be able to fit a perfect line, because the points do not exist on a straight line.

- Machine learning models are never perfect, as real-world data is never perfect.

- But you can fit a line to estimate a new y value for a given x value, even if there is an inevitable margin of error called loss.

Even if there is loss, it can be helpful in estimating predictions, such as how much y growth there will be at x time in the future.



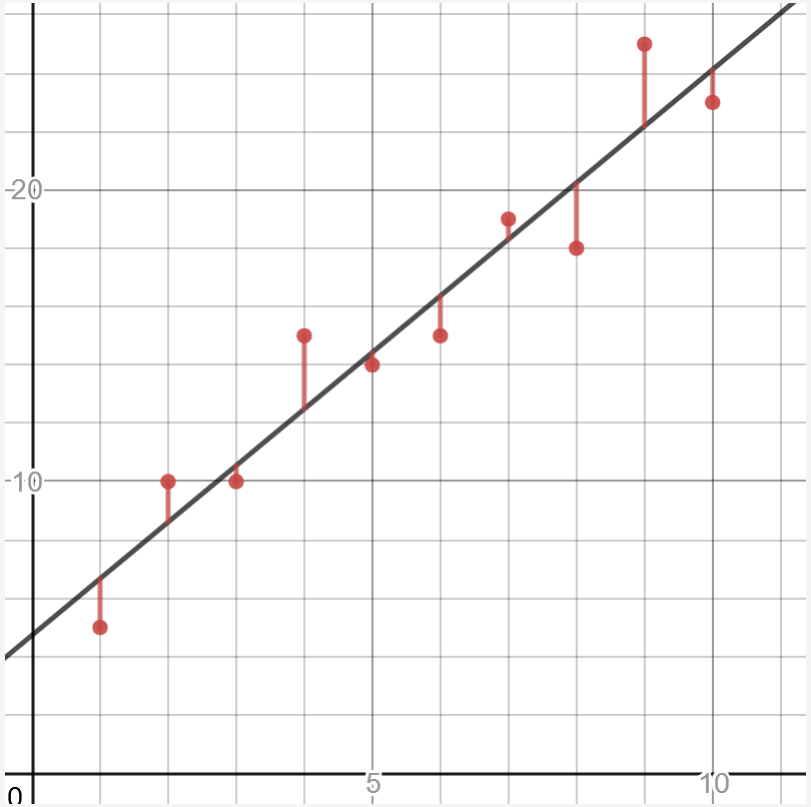
https://www.desmos.com/calculator/fmhotfn3qm

Simple Linear Regression

When you plot a given line, notice that there is a difference between the predicted y values and actual y values with our observed data.

These are called **residuals**, and in machine learning we want to minimize them.

So how do we minimize residuals in aggregate?



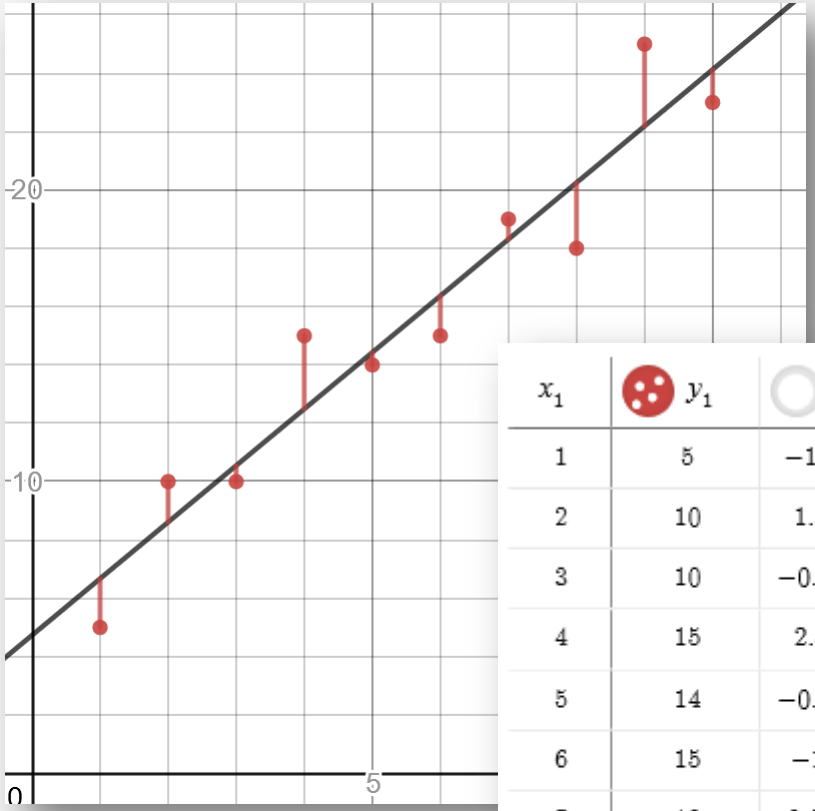
https://www.desmos.com/calculator/fmhotfn3qm

Sum of Least Squares

When we solve for $\boldsymbol{m}$ and $\boldsymbol{b}$ values, we need an objective to minimize total residuals using a **loss function**.

- We do not necessarily want to sum up the residuals, as the negatives will cancel out the positives.
- We can sum the absolute values, but that does not amplify larger residuals, and absolute values are mathematically difficult to work with.

So what is the best approach to total all the residuals to evaluate the quality of the fit?



x_1	y_1	e_1
1	5	−1.6727273
2	10	1.3878788
3	10	−0.55151515
4	15	2.5090909
5	14	−0.43030303
6	15	−1.369697
7	19	0.69090909
8	18	−2.2484848
9	25	2.8121212
10	23	−1.1272727

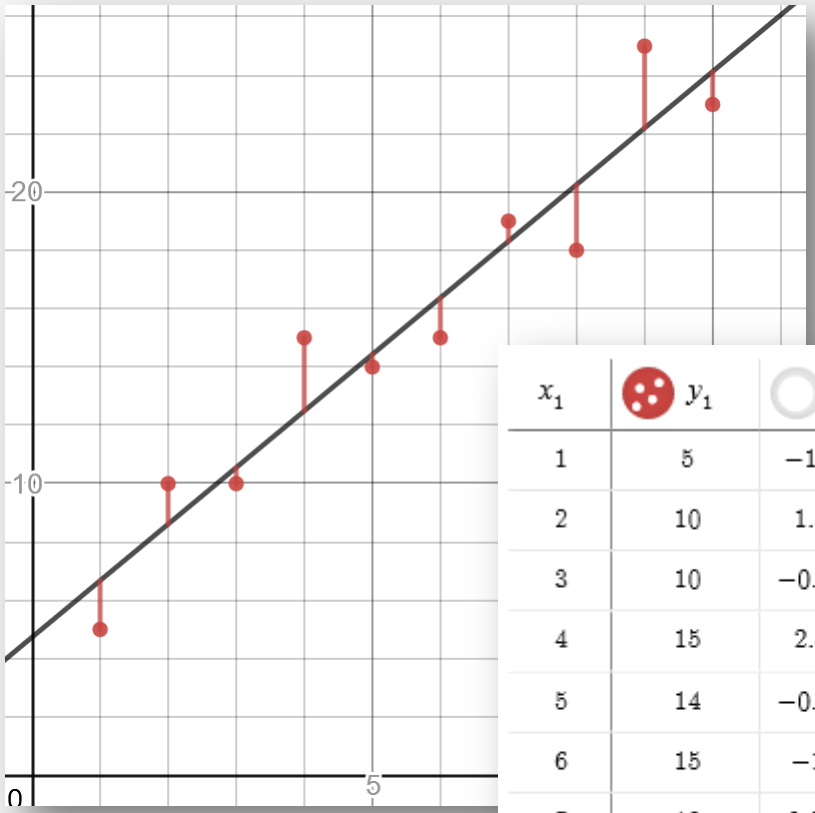
Sum of Least Squares

The best approach: we can square the residuals and sum them!

- Squaring will penalize large residuals by making them even larger!
- Squaring will also conveniently turn negatives into positives.
- Although we are not going to dive into calculus, squares are much easier to take the derivative of.

We can then find the $\boldsymbol{m}$ and $\boldsymbol{b}$ values that will find the sum of least squares.

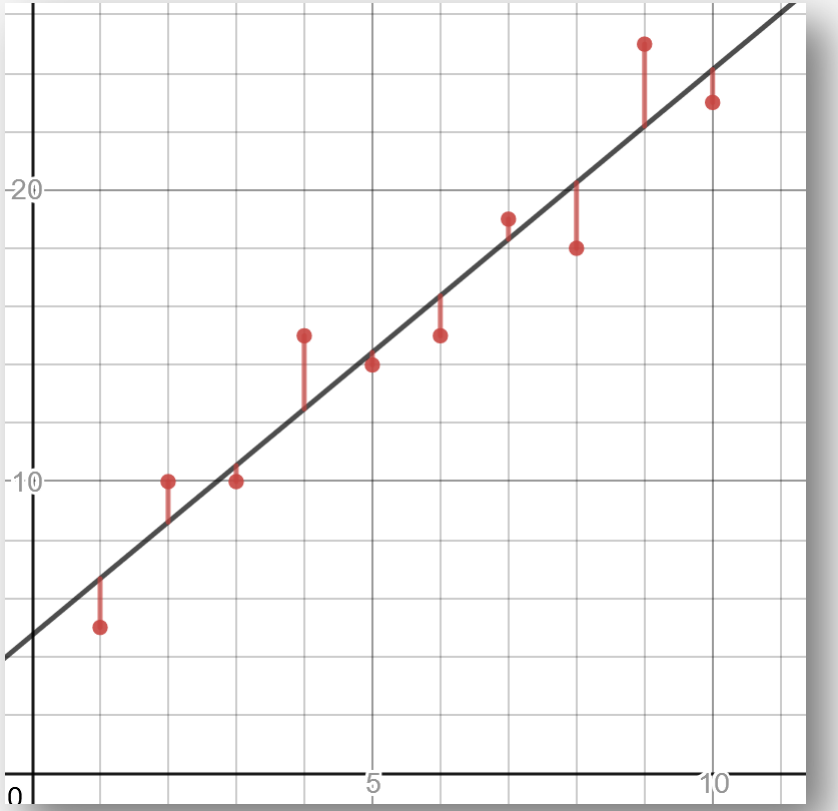
https://www.desmos.com/calculator/fvrnuhw0hy

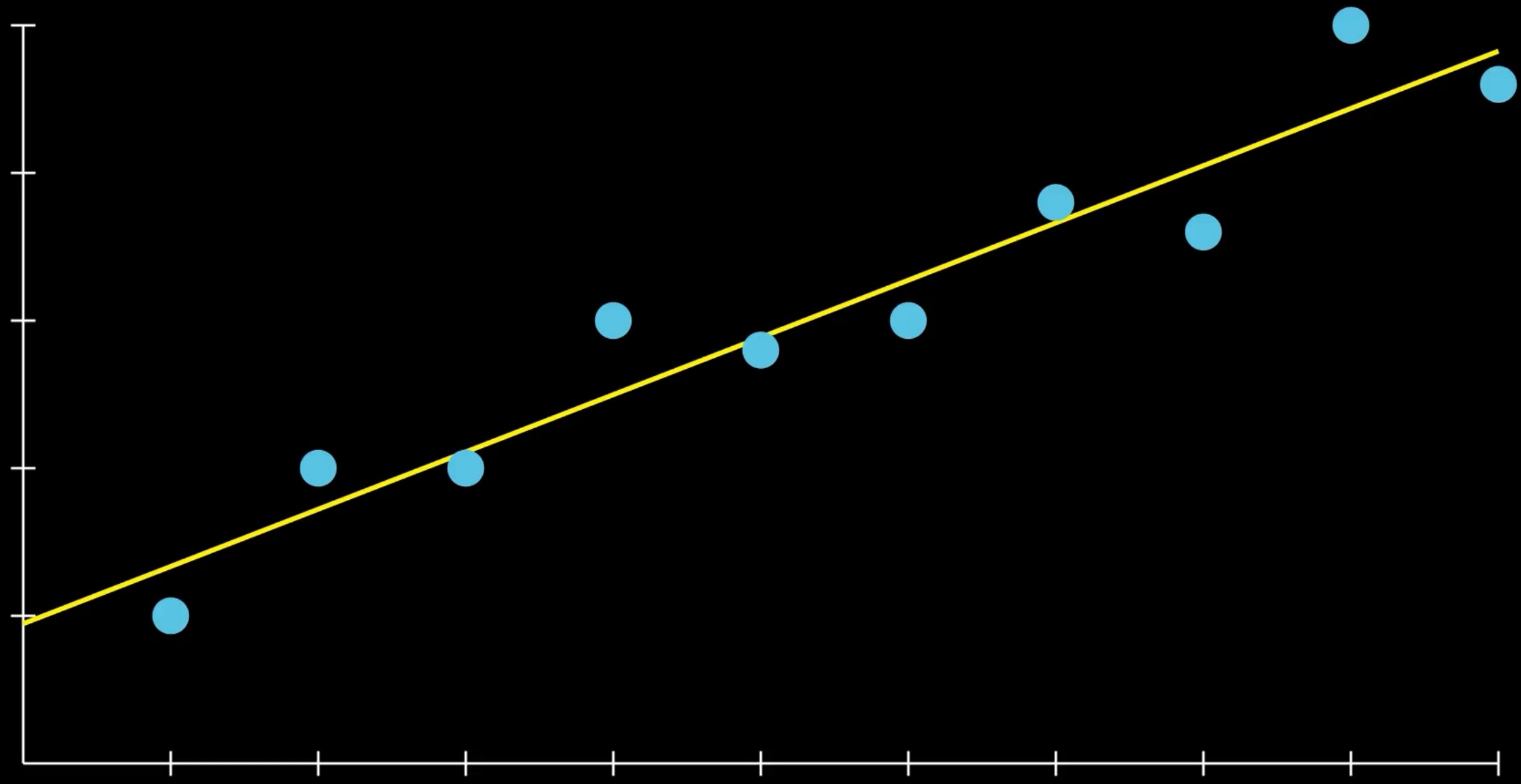


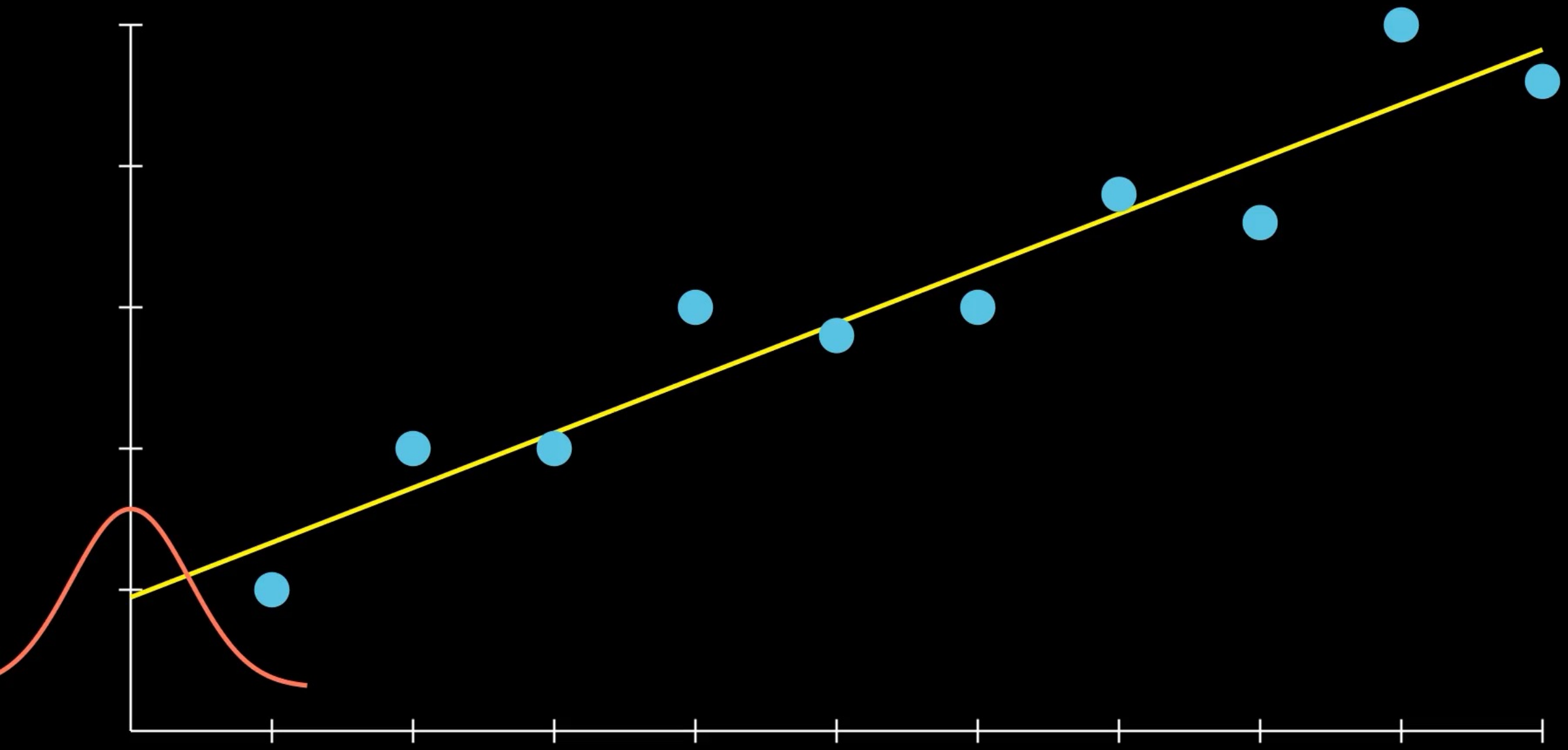
x_1	y_1	e_1
1	5	−1.6727273
2	10	1.3878788
3	10	−0.55151515
4	15	2.5090909
5	14	−0.43030303
6	15	−1.369697
7	19	0.69090909
8	18	−2.2484848
9	25	2.8121212
10	23	−1.1272727

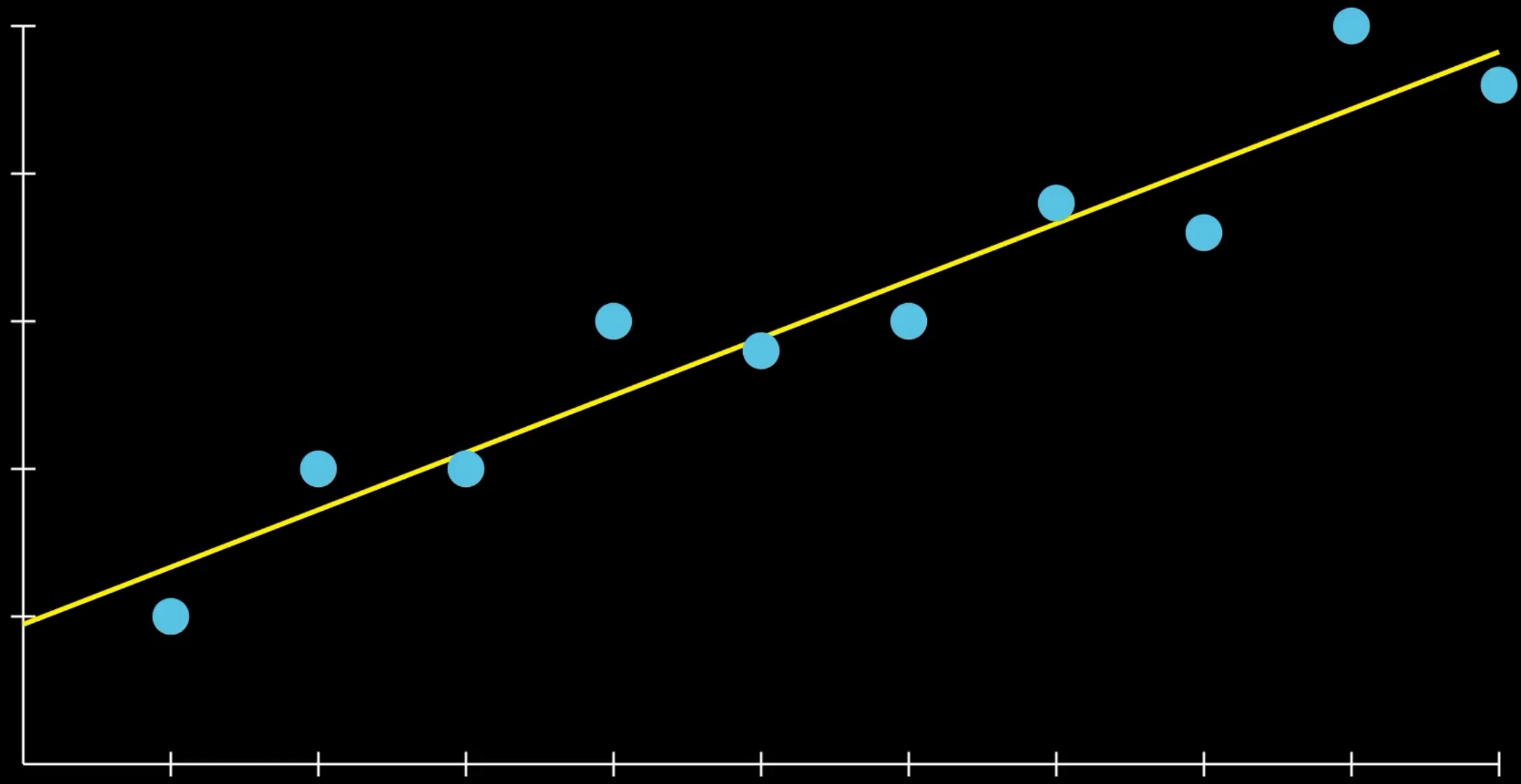
Calculating Sum of Squares

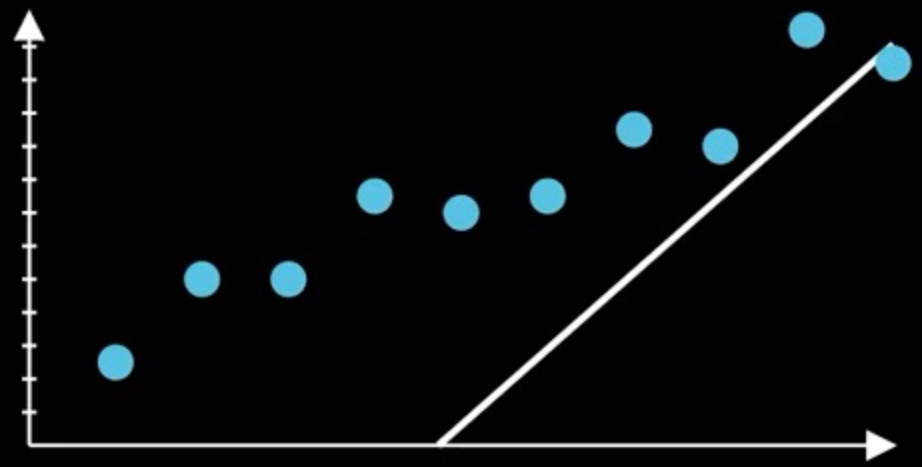
m	b			
1.93939	4.73333			
x	**y**	**y_predict (mx+b)**	**residual**	**residual squared**
1	5	6.67272	-1.67272	2.797992198
2	10	8.61211	1.38789	1.926238652
3	10	10.5515	-0.5515	0.30415225
4	15	12.49089	2.50911	6.295632992
5	14	14.43028	-0.43028	0.185140878
6	15	16.36967	-1.36967	1.875995909
7	19	18.30906	0.69094	0.477398084
8	18	20.24845	-2.24845	5.055527402
9	25	22.18784	2.81216	7.908243866
10	23	24.12723	-1.12723	1.270647473
			Sum of Squares	28.0969697





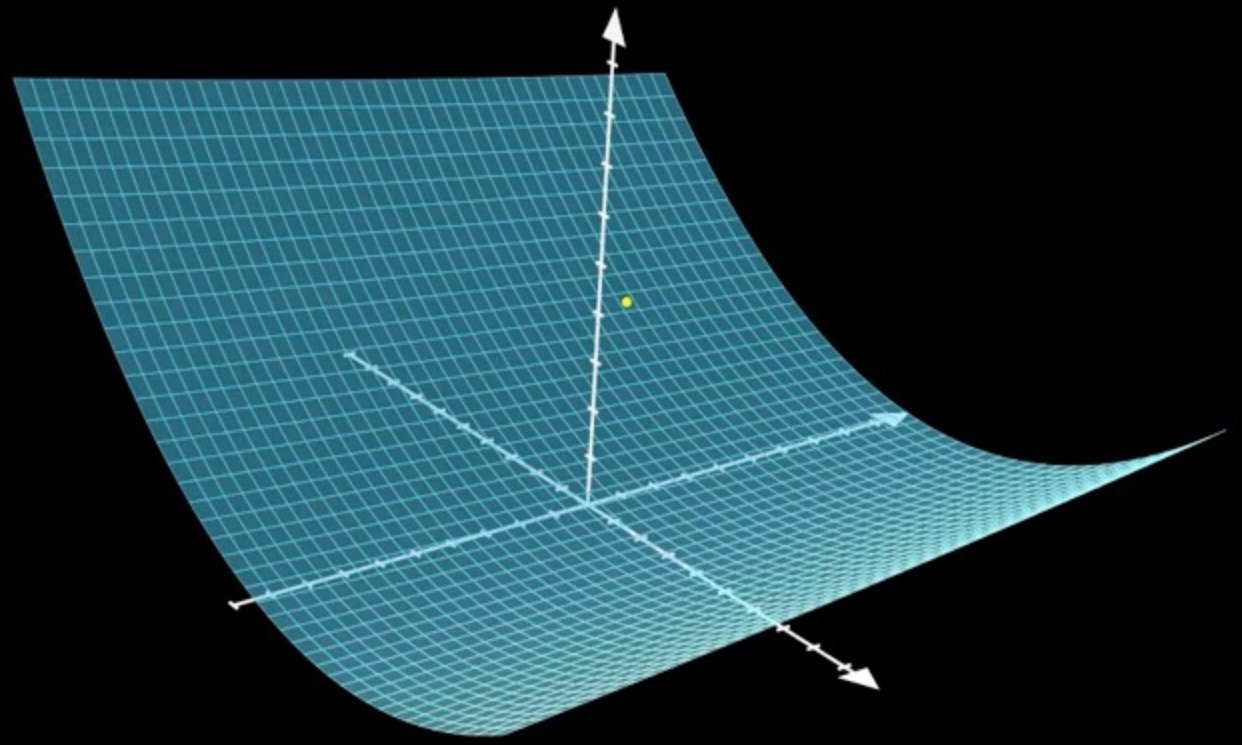






$$m = 8.285$$

$$b = 3.2416$$



Multiple Linear Regression

Solving for a function $\boldsymbol{y = mx + b}$ is elementary as it only has one independent variable $\boldsymbol{x}$.

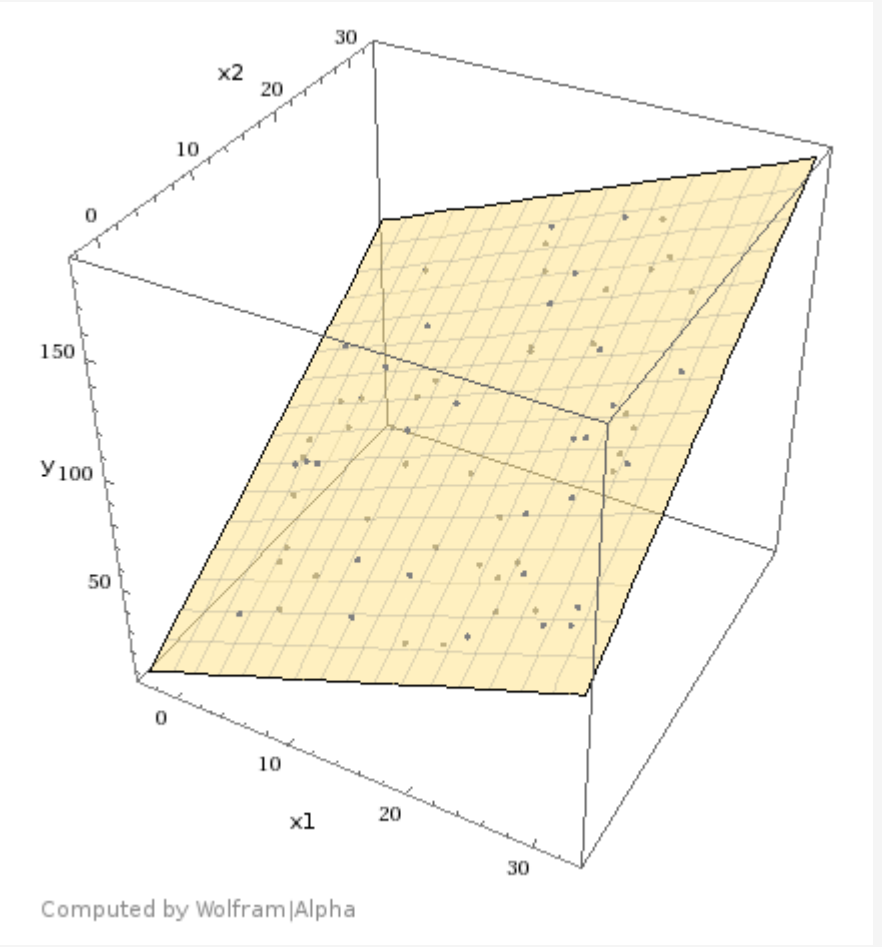
We can also solve for multiple independent variables, like x_1, x_2, x_3, and so on…

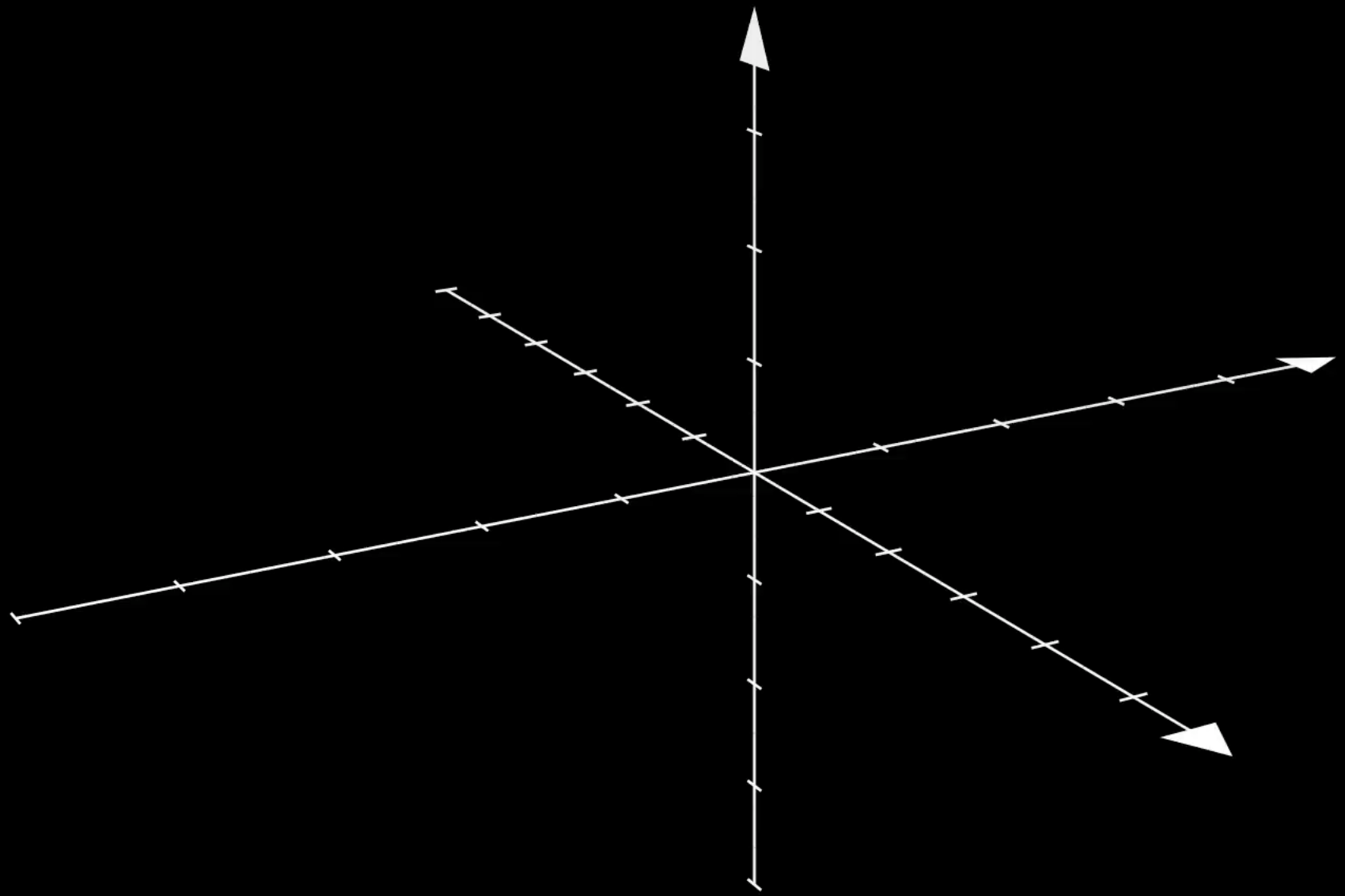
Let's say we have columns of independent variables x_1 and x_2, and a dependent variable y.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

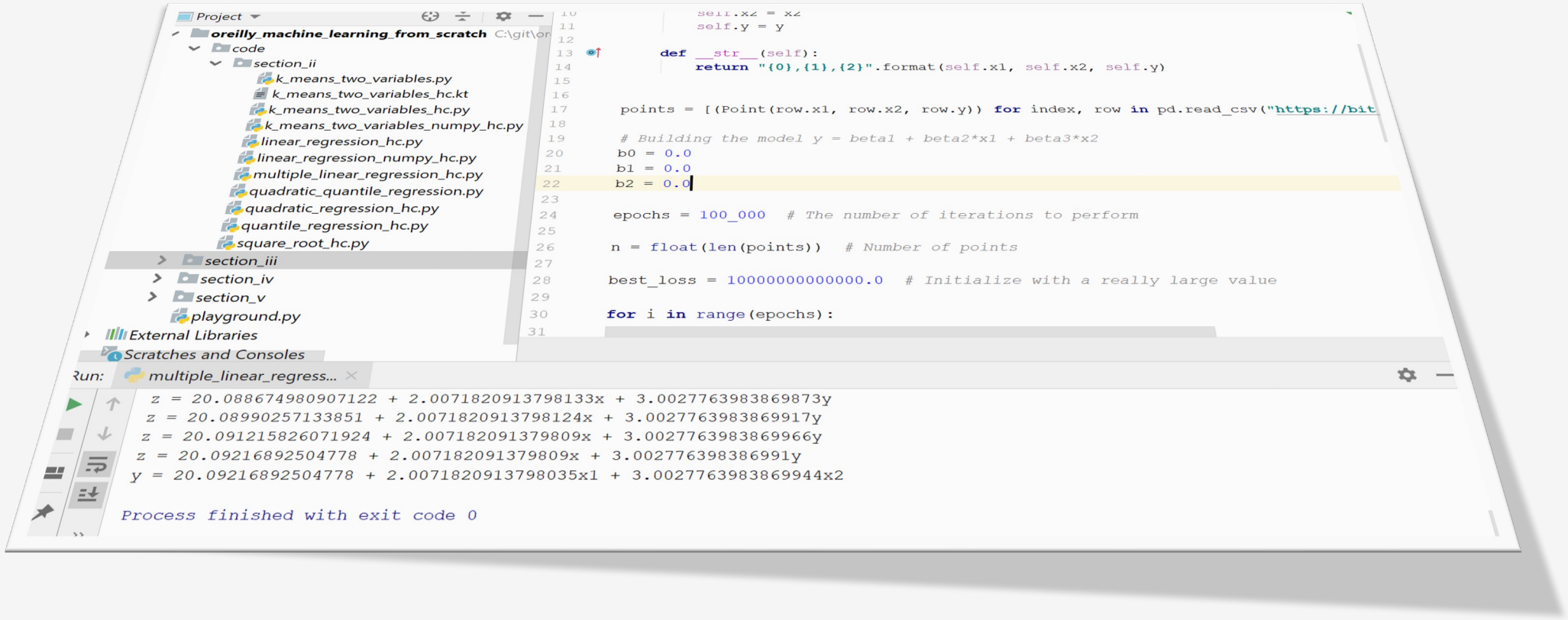
Instead of $\boldsymbol{m}$ and $\boldsymbol{b}$ constants, we now need to solve for each β parameter, where β_0 is the y-intercept and β_1 and β_2 are slopes for the respective x variables.

We can use the same hill-climbing technique as before!





Hands-On: Multiple Linear Regression



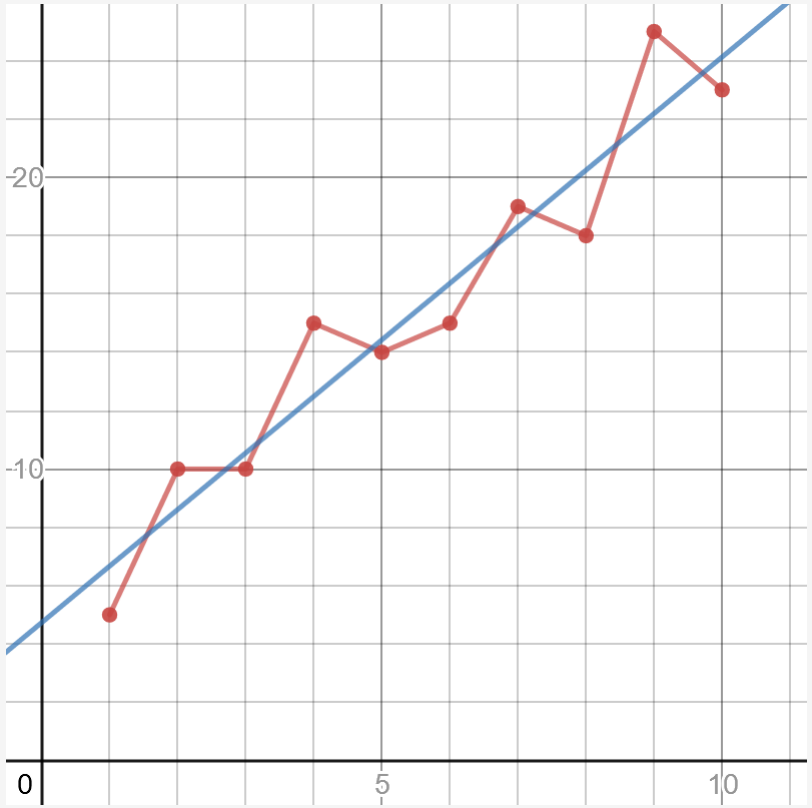
A Note About Overfitting

We are not going to focus too much on validating and analyzing machine learning models, but we should at least mention overfitting.

Overfitting means that our ML model works well with the data it was trained on but fails to predict correctly with new data.

- This can be due to many factors, but a common cause is the sampled data does not represent the larger population and more data is needed.
- The red line has high **variance**, meaning its predictions are sensitive to outliers and therefore can vary greatly.
- The blue line has high **bias**, meaning the model is less sensitive to outliers because it prioritizes a method (maintaining a straight line) rather than bend and respond to variance.

The red line to the right is likely overfitted (high variance, low bias), but the blue linear regression line (low variance, high bias) is less likely to be overfit.



https://www.desmos.com/calculator/wmwfolbvdk

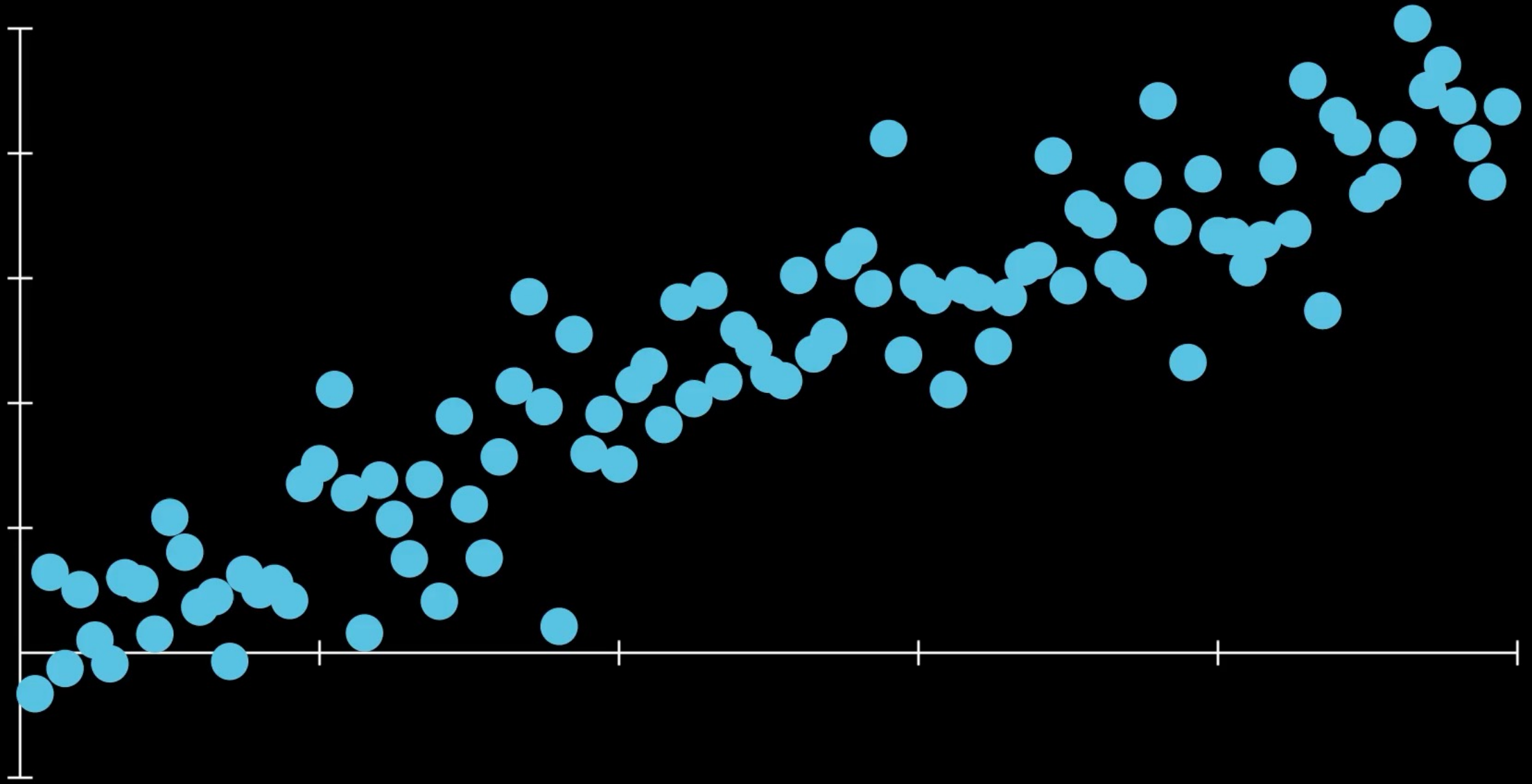
A Note About Overfitting

Linear regression is a highly biased method and is resilient to overfitting.

There are other remedies to mitigate overfitting, the most basic being separating **training data** and **test data**.

- The model is fit to the training data, and then is tested with the test data.
- If the test data performs poorly compared to the training data, there is a possibility of overfit (or just no correlation altogether).

You can also try to train with more data as well as utilize cross-validation, regularization, bagging, boosting, and other techniques.



A Note About Normalization

Normalization is another topic we do not have time to explore deeply in this 4-hour class but should at least get mentioned.

Normalization is scaling and converting the values of each variable, so they are relatively close together.

- Imagine you had a variable ***age*** whose values typically range in 0-99.
- But you had another variable ***income*** that typically ranges from 30,000 to 1,000,000.
- Because these ranges are so drastically different, fitting a model is not going to be productive until you transform them somehow.

There are a variety of techniques you can employ from linear scaling to fitting to a standard normal distribution.

Section III
Logistic Regression

Classifying Things

So far we just did regressions via linear regression.

However, linear regressions are awkward to use for classification.

- Lines extend in a straight direction for positive infinity and negative infinity, well outside a range of acceptable values.
- Lines do not do a good job representing a probability and staying within the limits of 0.0 and 1.0.
- When doing classification, probability is a critical tool.

Hot dog: 0.92

Classifying Things

Classification tasks are pretty common in machine learning:

- How do I classify images of *dogs* versus *cats*?
- Will a shipment be most likely be *late*, *early*, or *on-time*?
- Is this email *spam* or *not spam*?
- Will this movie get *1 star*, *2 stars*... or *5 stars*?

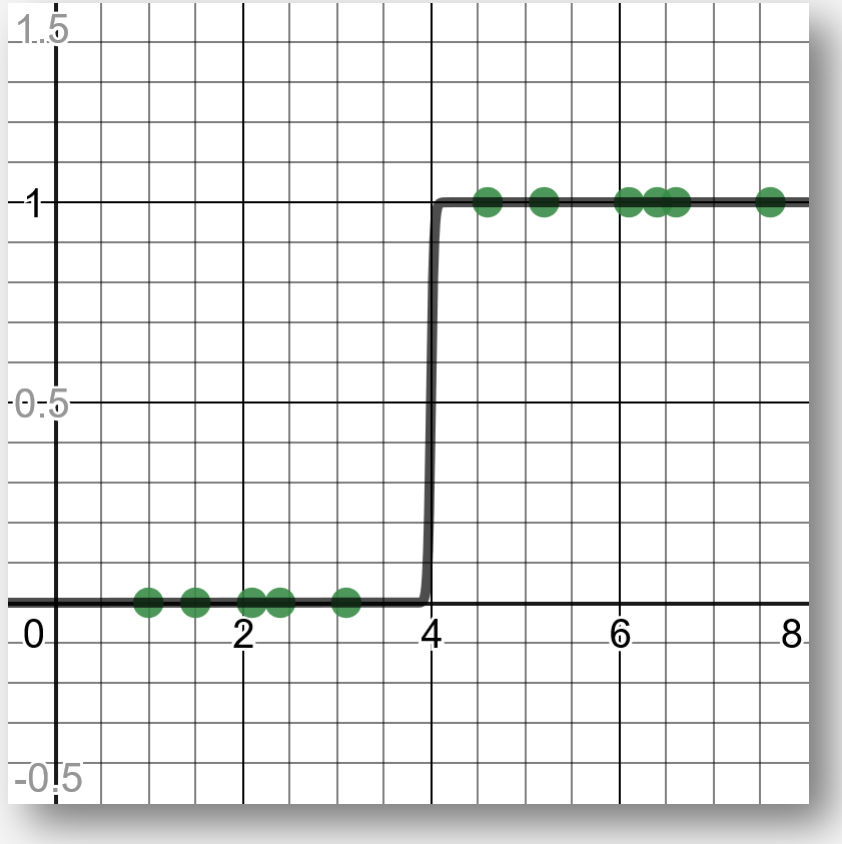
There are several machine learning algorithms that work well for classification, and we are going to learn first about Logistic Regression.

Hot dog: 0.92

Logistic Regression Intuition

Imagine you have 11 patients exposed to a chemical for x hours, and you plot whether they exhibited symptoms (1) or not (0).

Plotting our patient data (right), we can easily eyeball a clear cutoff at 4 hours where patients transition from ***not showing symptoms (0)*** to ***showing symptoms (1)***.



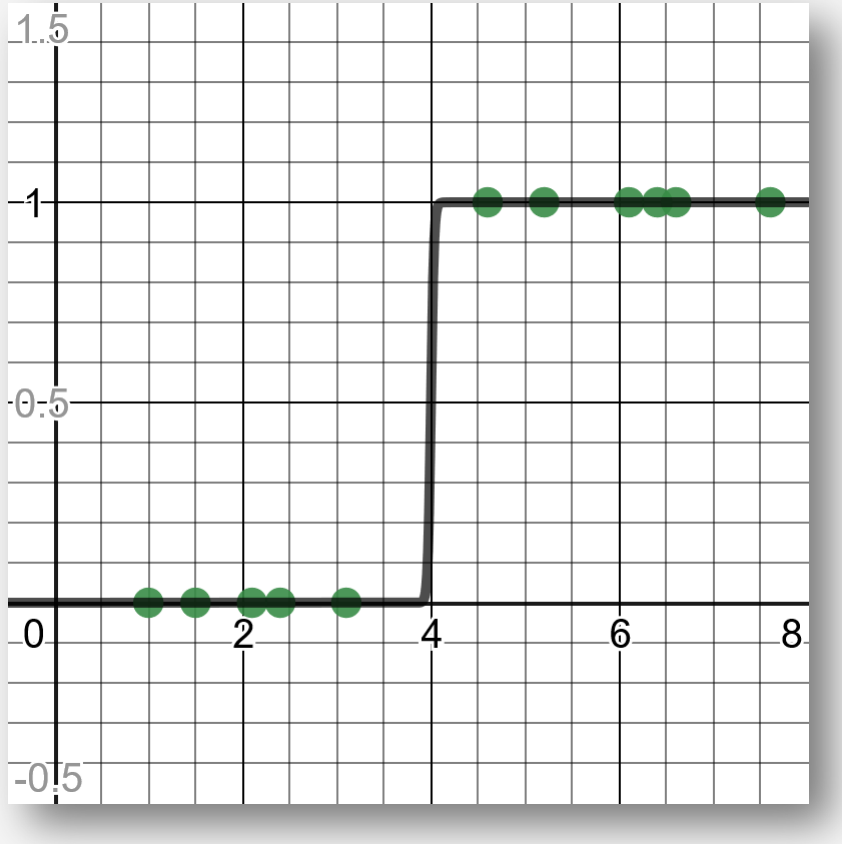
https://www.desmos.com/calculator/prs2p0sofc

Logistic Regression Intuition

This indicates any patient exposed for less than 4 hours will have a 0% chance of showing symptoms, but greater than 4 will have a 100% chance of showing symptoms.

Because there is a distinct separation at 4 hours, a logistic regression is going to “jump” from 0% to 100% at that boundary.

Of course, real life rarely works out this way…



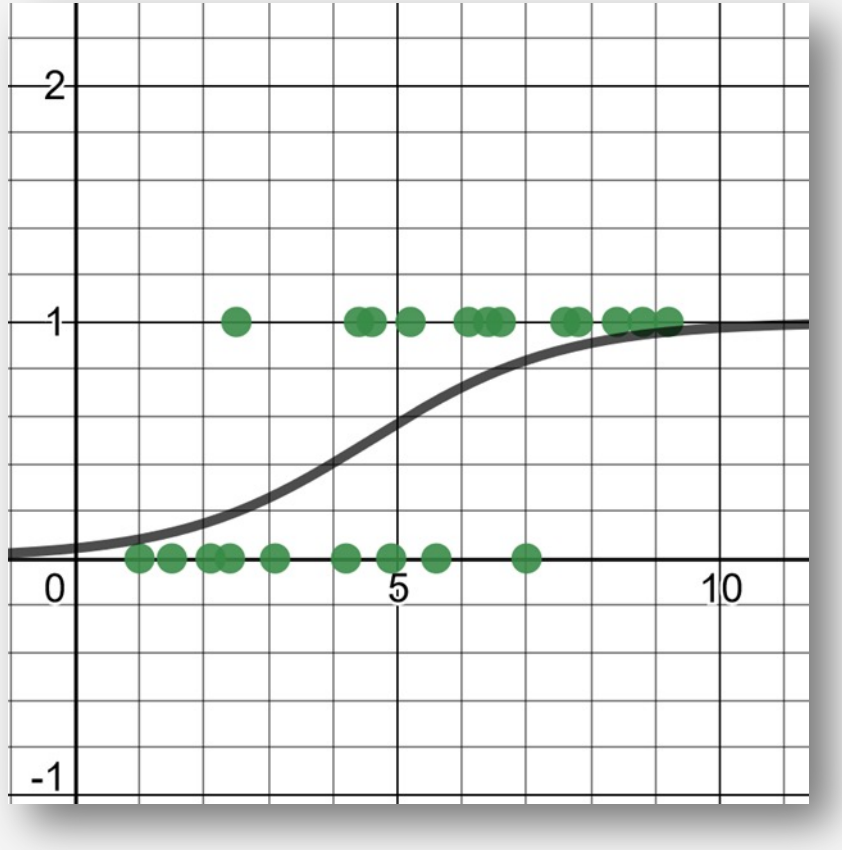
https://www.desmos.com/calculator/prs2p0sofc

Logistic Regression Intuition

Now let's say you gathered more data and got a realistic picture, where the middle of the range has a mix of patients showing symptoms and not showing symptoms.

The way to interpret this is the probability of patients showing symptoms gradually increases with each hour of exposure.

Because of this overlap of points in the middle, there is no distinct cutoff when patients show symptoms, but rather a gradual transition from 0% probability to 100% probability ("0" and "1").

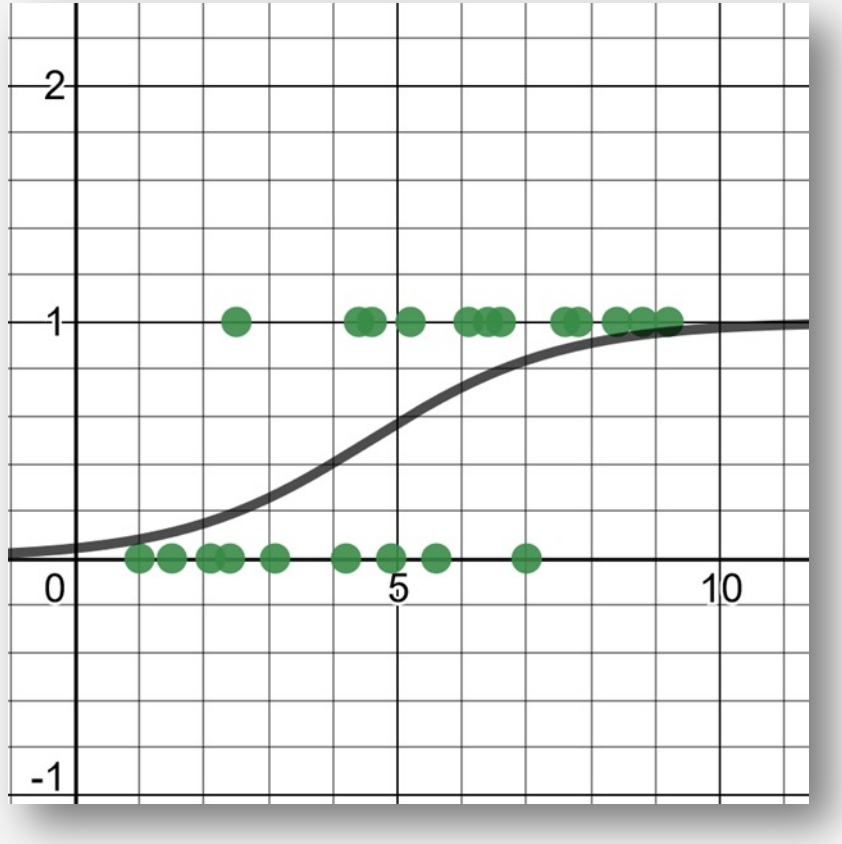


https://www.desmos.com/calculator/bsqtqfians

Logistic Regression Intuition

More technically, a logistic regression results in a curve indicating a probability of belonging to the **true** (1) category, which in this case means ***a patient showed symptoms***.

As the hours of chemical exposure increases, the number of patients showing symptoms increases, and thus the probability of showing symptoms increases.



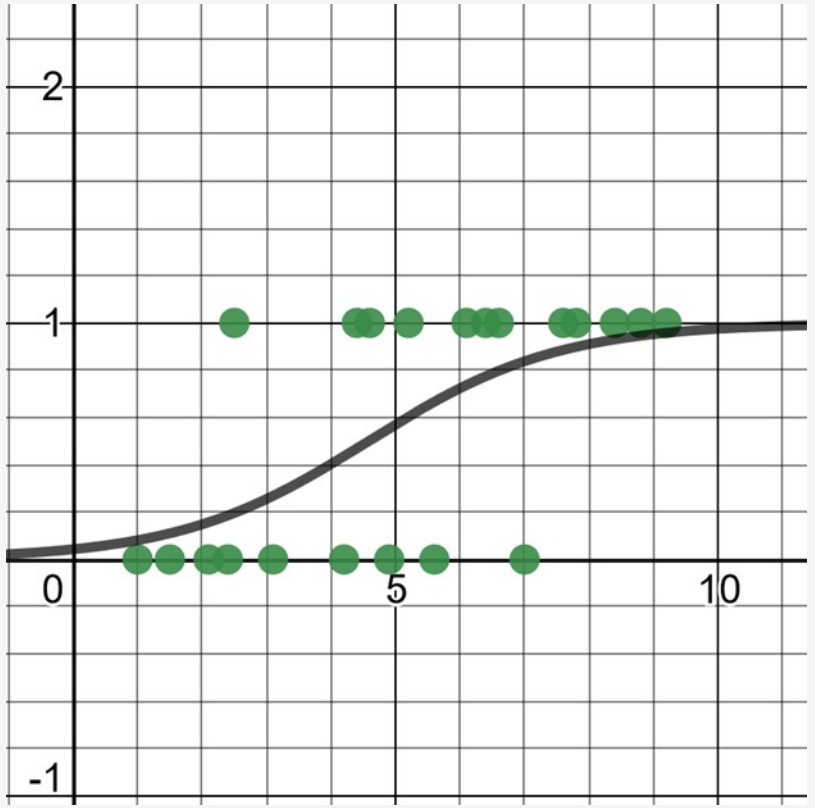
https://www.desmos.com/calculator/bsqtqfians

Logistic Regression

Logistic Regression is a classification tool that predicts a **true** or **false** value for one or more variables.

Training data must have outcomes of 0 (false) or 1 (true), but the regression will output a probability value between 0 and 1.

- An S-shaped curve (a **logistic function**) is fit to the points and then used to predict probability.
- If a predicted value (the y-axis) is less than .5 it is typically categorized as false (0), and if the predicted value is greater than/equal to .5 it is typically categorized as true (1).



https://www.desmos.com/calculator/bsqtqfians

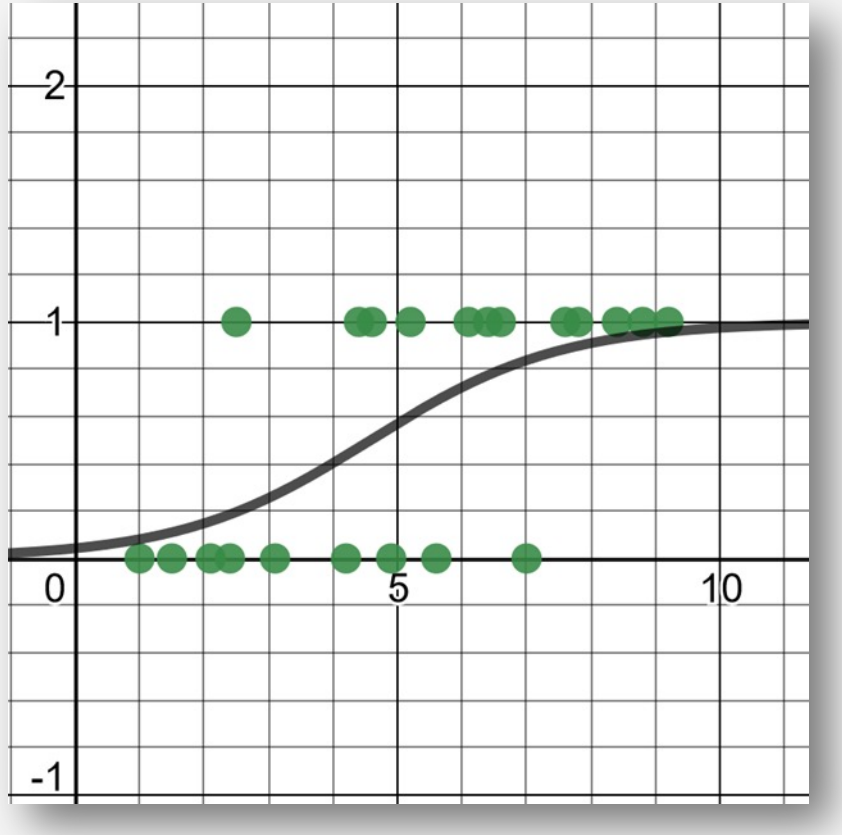
Just Show Me the Math!

For a single independent variable x to predict a dependent probability variable y, you need to fit the data to the logistic function:

$$y = \frac{1.0}{1.0 + e^{-(\beta_0 + \beta_1 x)}}$$

You can express this in Python as:

```
def predict_probability(x):
    p = 1.0 / (1.0 + math.exp(-(b0 + b1 * x)))
    return p
```



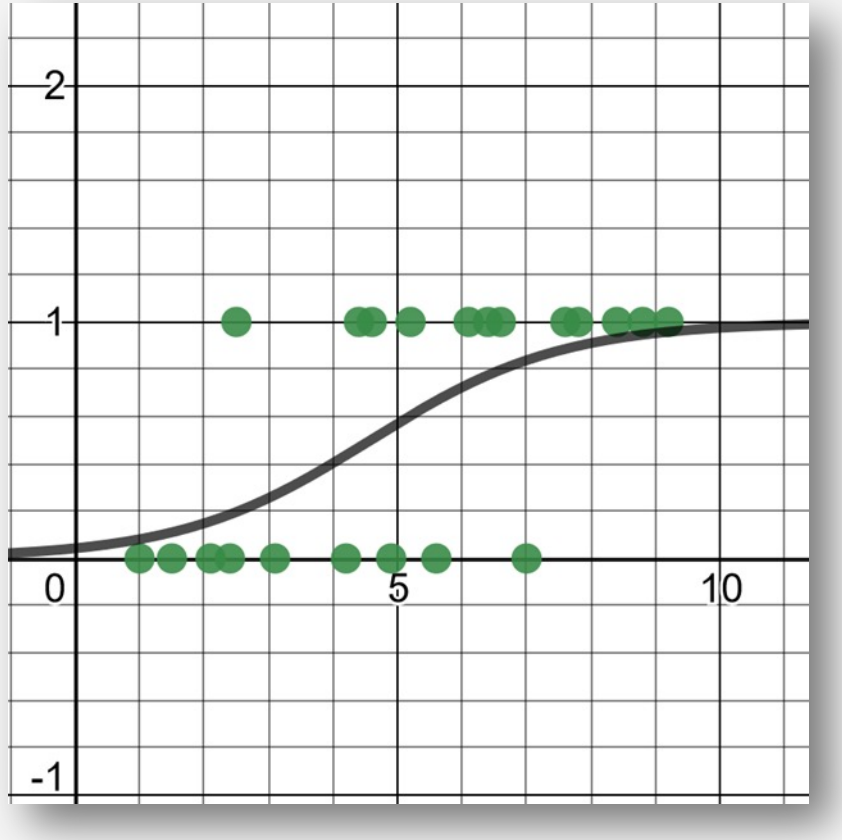
https://www.desmos.com/calculator/blfcwrlnuw

Just Show Me the Math!

You may see also see this logistic function expressed as:

$$y = \frac{e^{\beta_0+\beta_1 x}}{1.0 + e^{\beta_0+\beta_1 x}}$$

However it is the same and just algebraically expressed differently.



https://www.desmos.com/calculator/blfcwrlnuw

Just Show Me the Math!

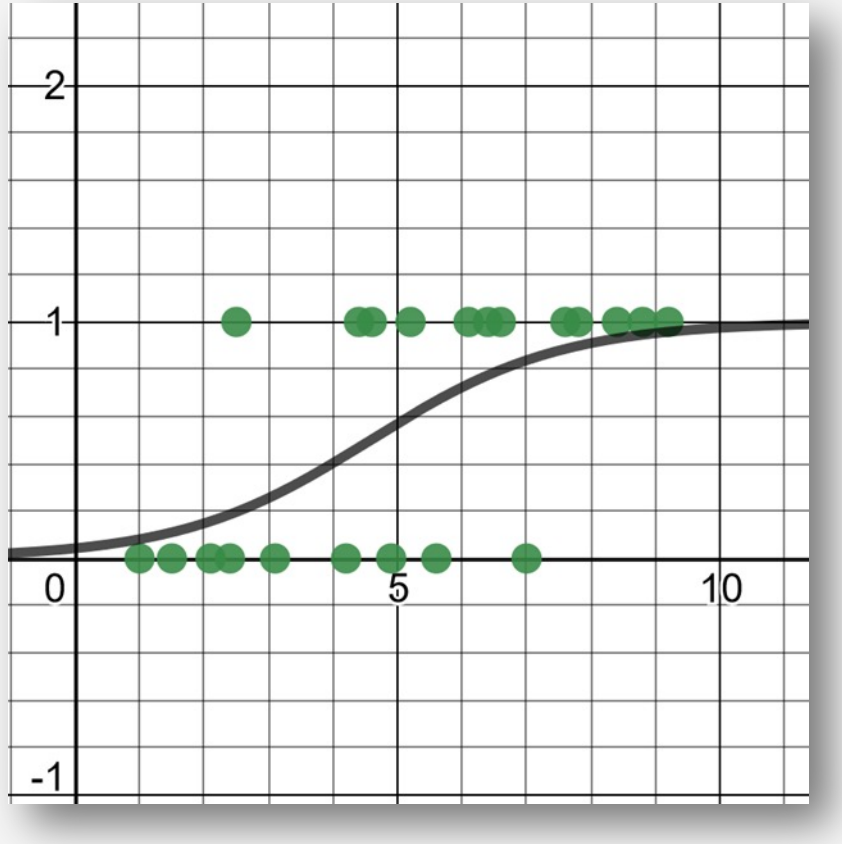
$$y = \frac{1.0}{1.0 + e^{-(\beta_0 + \beta_1 x)}}$$

```
def predict_probability(x):
    p = 1.0 / (1.0 + math.exp(-(b0 + b1 * x)))
    return p
```

Notice the expression $\beta_0 + \beta_1 x$ is linear, and this known as the **log odds function** which is translated logarithmically into a probability.

In the interest of time, we will avoid going into proofs and mathematical details about how this function works.

Just know it produces this S-shaped curve we need to output a probability between 0 and 1.



https://www.desmos.com/calculator/blfcwrlnuw

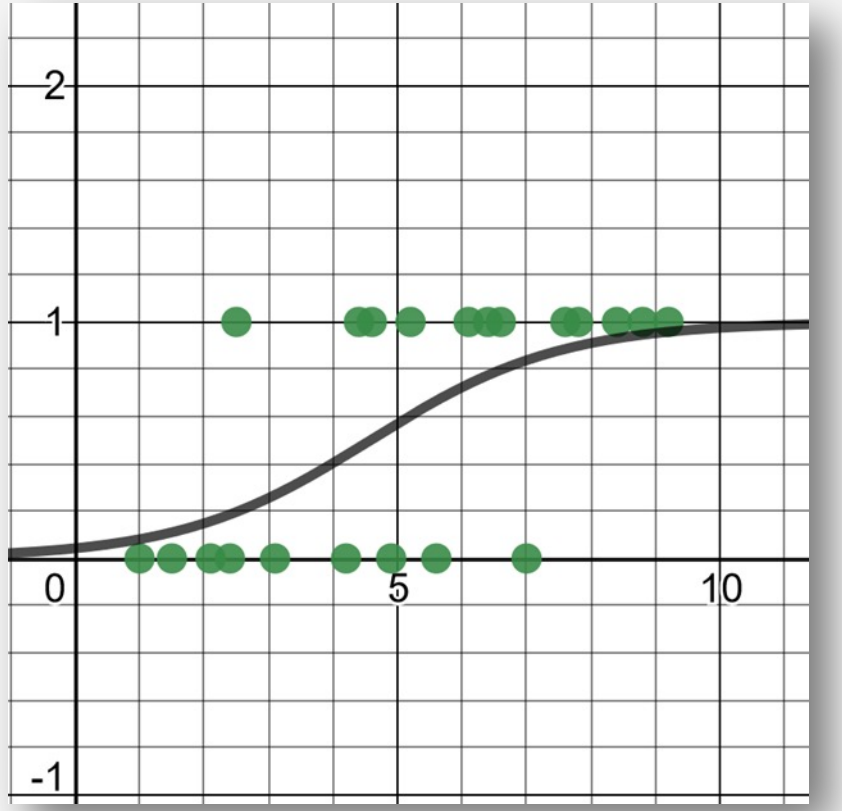
Just Show Me the Math!

$$y = \frac{1.0}{1.0 + e^{-(\beta_0 + \beta_1 x)}}$$

```
def predict_probability(x):
    p = 1.0 / (1.0 + math.exp(-(b0 + b1 * x)))
    return p
```

We need to solve for β_0 and β_1, but we cannot use least squares like in linear regression.

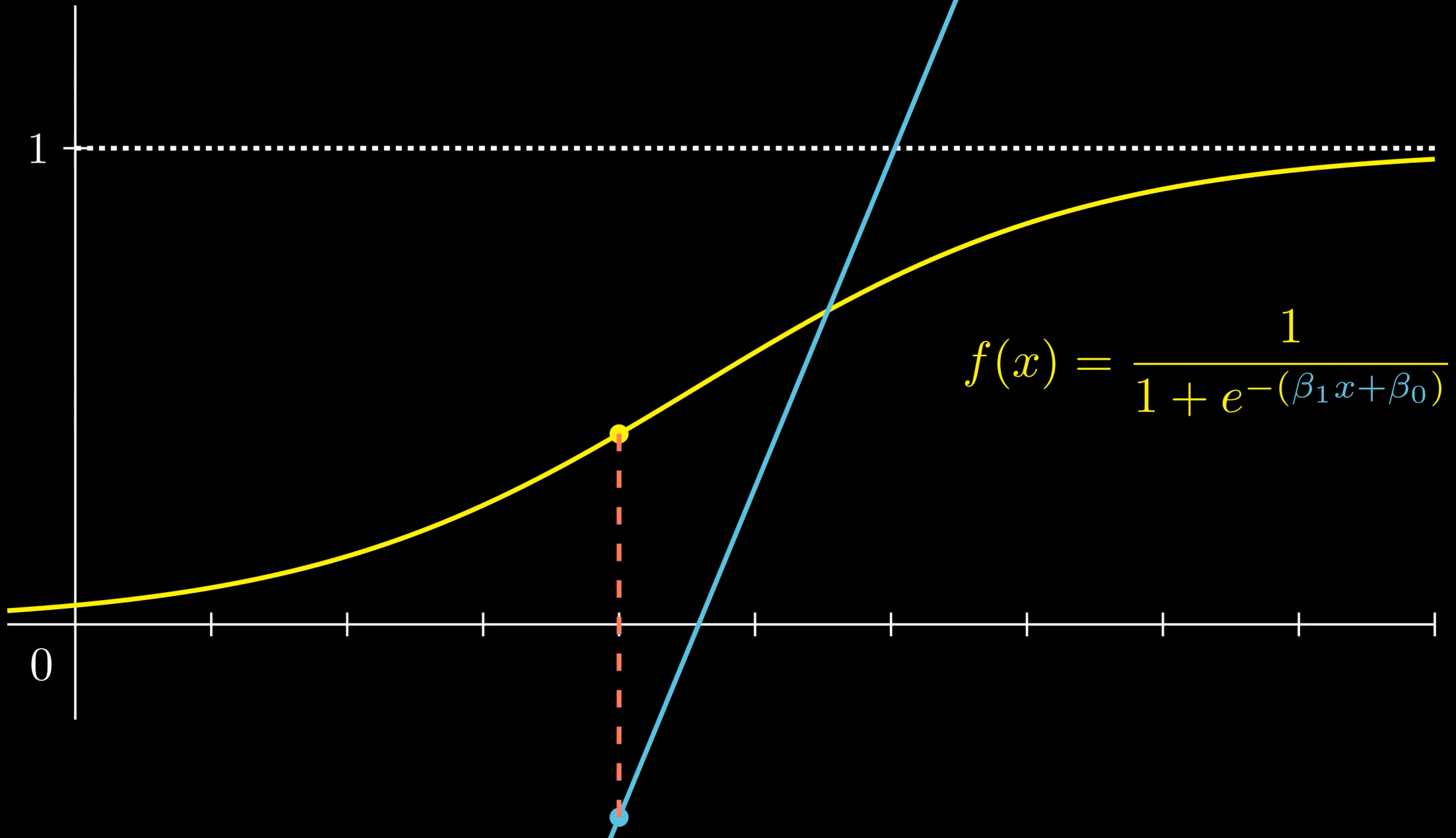
- We are trying to maximize probability of the curve predicting correctly, not finding the best fit.
- Using hill climbing, we need to find β_0 and β_1 that produces the maximum likelihood.

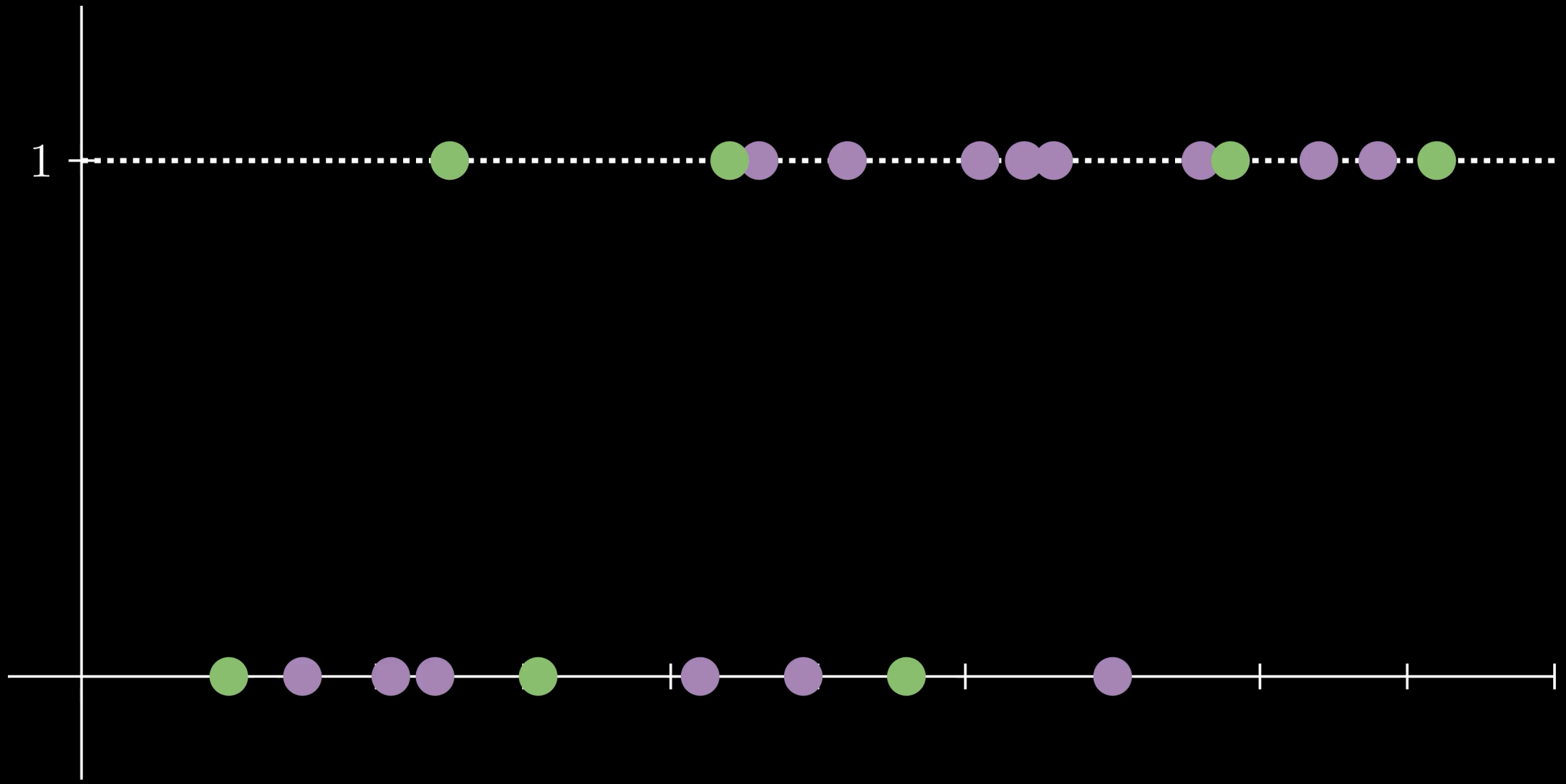


https://www.desmos.com/calculator/blfcwrlnuw

Logistic Regression

in 3 Minutes


1
0
$f(x) = \frac{1}{1 + e^{-(\beta_1 x + \beta_0)}}$


1

Maximum Likelihood

Maximum likelihood is a technique to estimate parameters that have the highest probability of outputting the observed data.

In our case, we need to find values for β_0 and β_1 that will yield the highest likelihood of outputting the correct true/false values.

Remember that our logistic function outputs a probability y for a given value x.

$$y = \frac{1.0}{1.0 + e^{-(\beta_0 + \beta_1 x)}}$$

Maximum Likelihood

Let's say during our hill-climbing, we test parameters $\beta_0 = -.3.17$ and $\beta_1 = 0.69$

$$y = \frac{1.0}{1.0 + e^{-(-3.17+0.69x)}}$$

To calculate the total likelihood for these parameters…

1. Get the true (1) data, calculate the probability $\boldsymbol{y}$ for each $\boldsymbol{x}$ value, and multiply them together.

2. Get the false (0) data, calculate the probability $(\boldsymbol{1.0 - y})$ for each $\boldsymbol{x}$ value, and multiply them together.

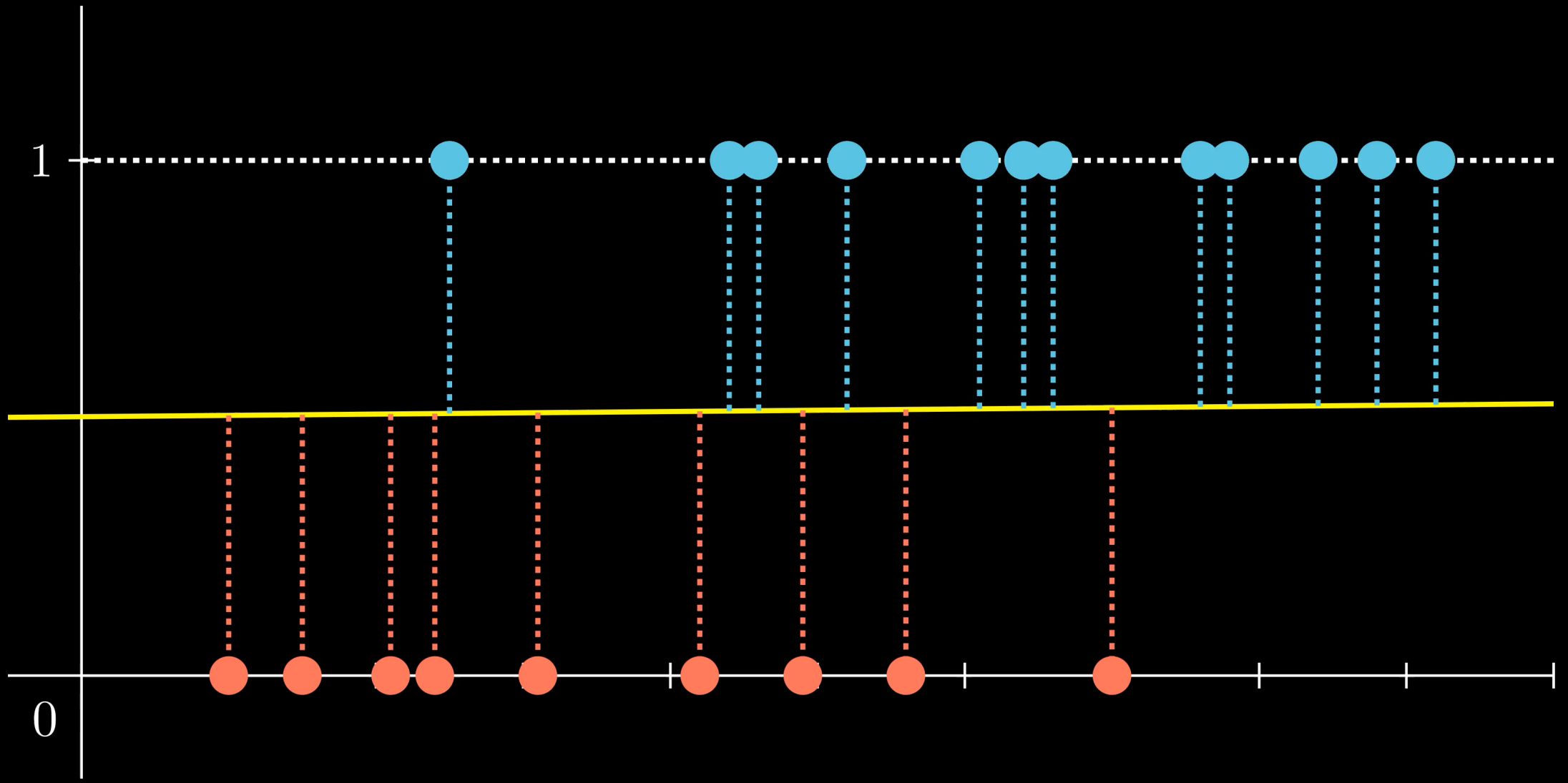
3. Multiply the two products above together, and that is your total likelihood.

Maximum Likelihood

We now know how to calculate the likelihood for a given set of parameters.

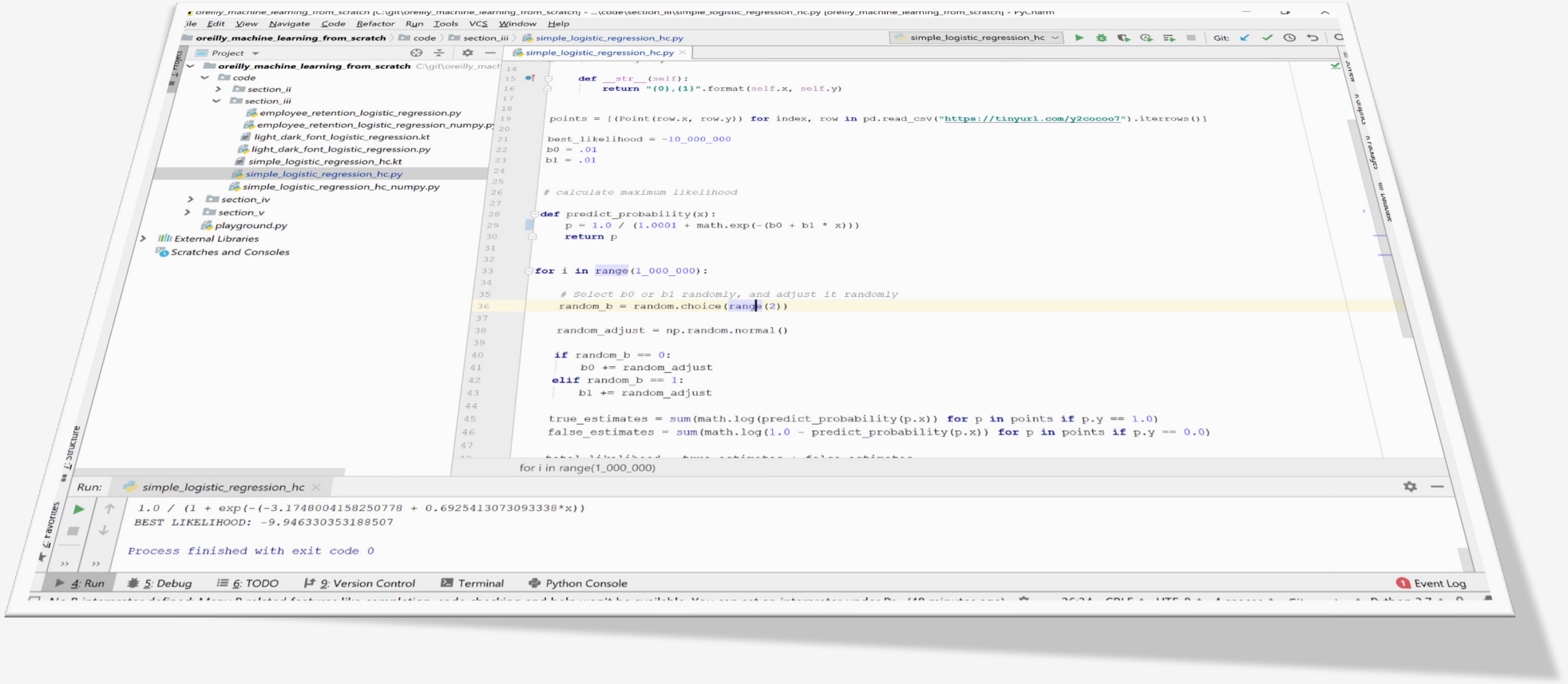
To calculate the maximum likelihood…

1. Randomly adjust the β_0 and β_1 values (using hill-climbing, gradient descent, or other optimization)
2. Calculate the likelihood (as shown in the previous slide)
3. If the likelihood improves, keep the changes to β_0 and β_1 , otherwise revert.
4. Do this for as many iterations as necessary, until the likelihood stops improving.



$L(x_i) =$

Hands-On: Logistic Regression



Multivariable Logistic Regression

We can easily extend logistic regression to handle multiple independent variables, simply by adding more β_x variables for each additional variable.

We then solve for those β_x variables the same way as before.

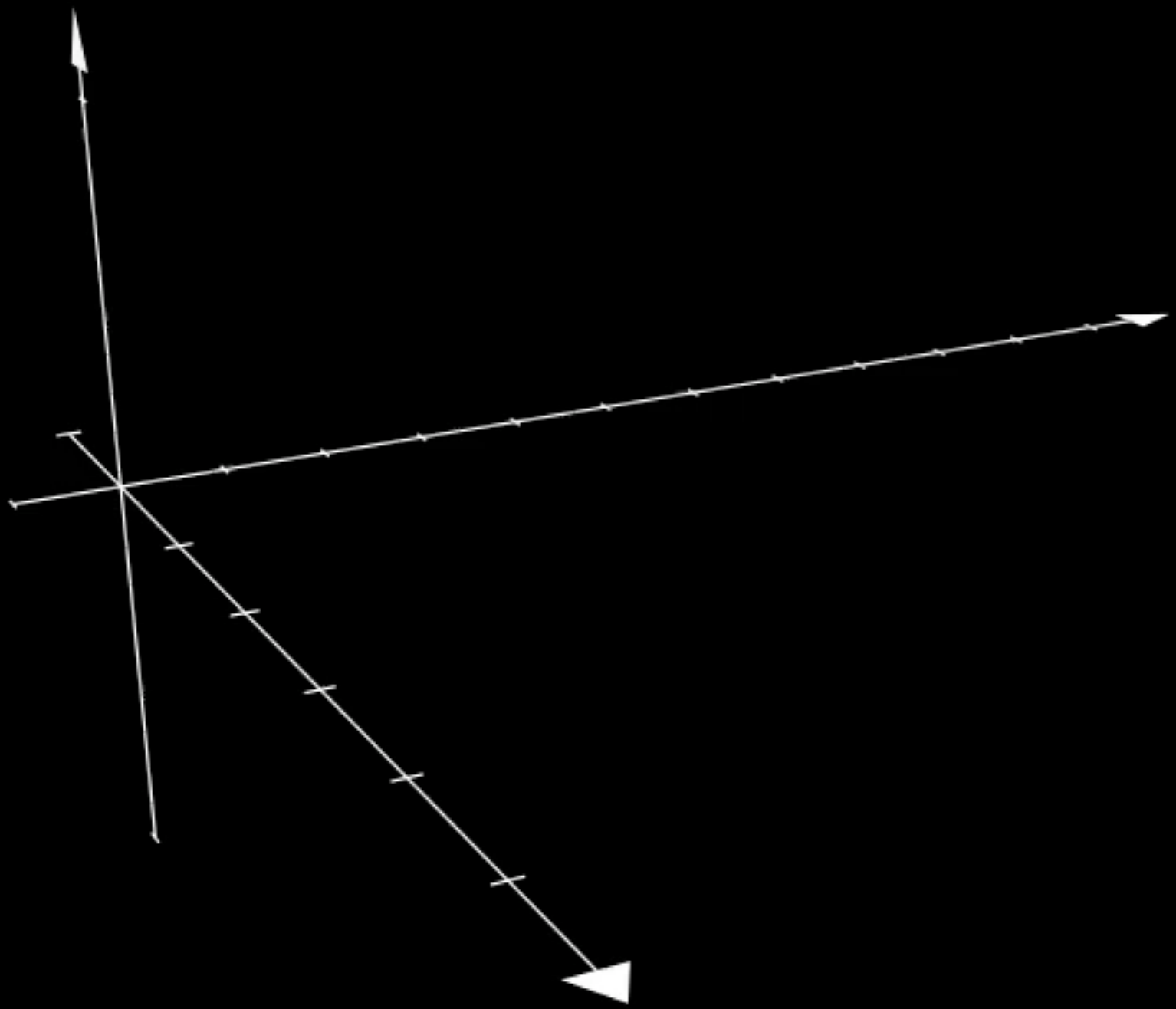
$$y = \frac{1.0}{1.0 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \ldots + \beta_n x_n)}}$$

Multivariable Logistic Regression

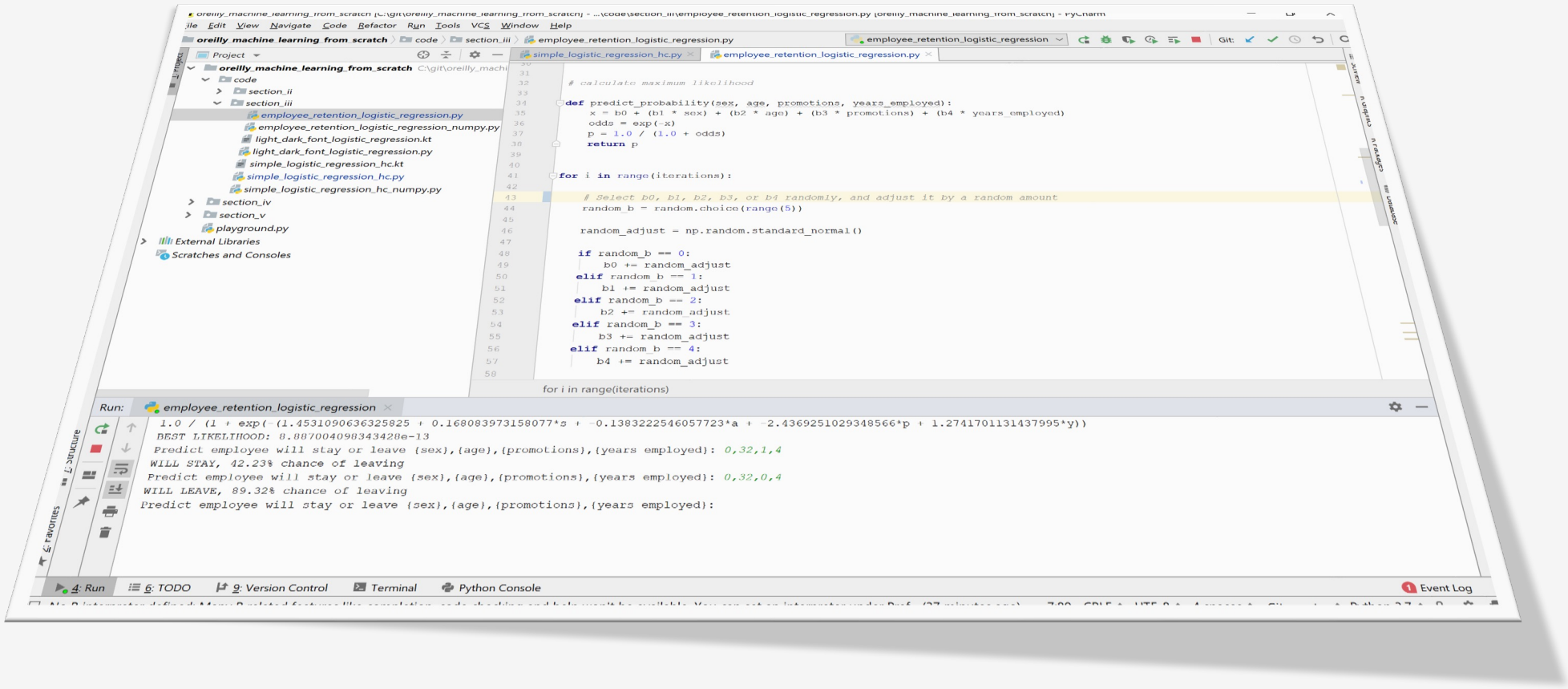
We have some historical employee data (https://tinyurl.com/y6r7qjrp) and want to use it to predict whether an employee will quit or not.

SEX, AGE, $PROMOTIONS$, and $YEARS\_EMPLOYED$ are the predictor variables, and DID_QUIT is the outcome variable where 1 = true and 0 = false.

SEX	AGE	PROMOTIONS	YEARS_EMPLOYED	DID_QUIT
1	43	4	10	0
1	38	3	8	0
1	44	4	11	1
0	41	2	6	0
1	45	2	6	1
0	36	3	9	0
1	33	1	3	0
1	44	3	10	0
…				



Hands-On: Multivariable Logistic Regression



Using Logistic Regression for Classification

Logistic regression may seem limited in that it only supports two categories: true (1) or false (0).

But you can make it support any number of categories!

To use logistic regression for more than two categories:

1. Build a separate logistic regression for each category, where a given item belongs to that category (1) or doesn't belong to that category (0).

2. To predict which category an item belongs to, pass it through each category's logistic regression, and choose the one with the highest probability.

Exercise: "There's No Correlation"

It is one day from a manned space launch.

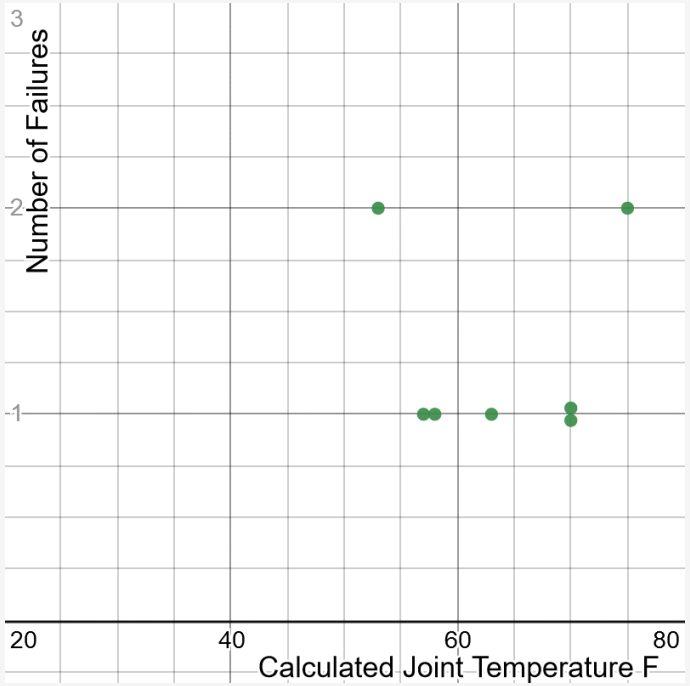
However, your engineering team has been expressing concern about the O-Rings that seal rocket gases from releasing, and whether they perform in colder temperatures.

You share this concern with other parties and are provided data of all 7 O-Ring failures from 24 launches and the temperature (shown to the right).

The consensus from other parties is there is no correlation between number of failures and temperature.

Is this assessment correct?

temperature	o_ring_failures
53	2
57	1
58	1
63	1
70	1
70	1
75	2



Data Source: http://www.math.usu.edu/~symanzik/teaching/2009_stat6560/RDataAndScripts/sharif_abbass_project1_challenger.csv

Exercise: "There's No Correlation"

It is one day from a manned space launch.

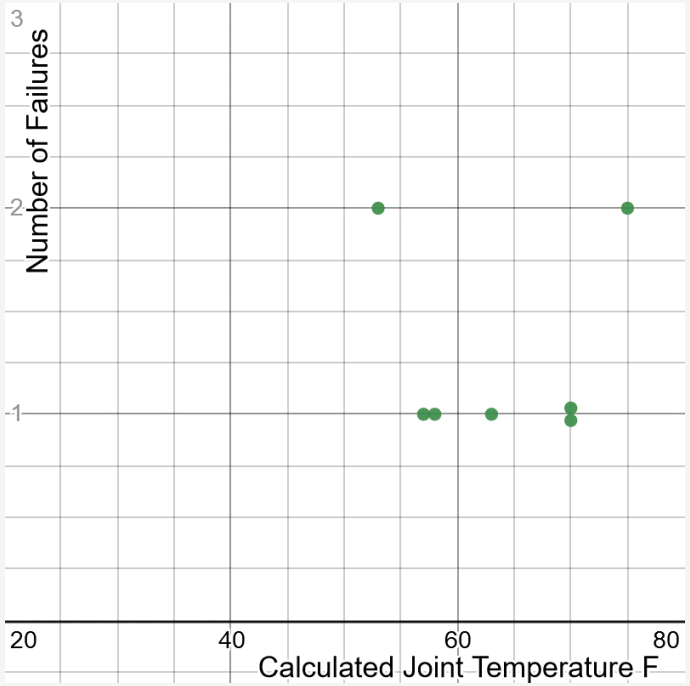
However, your engineering team has been expressing concern about the O-Rings that seal rocket gases from releasing, and whether they perform in colder temperatures.

You share this concern with other parties and are provided data of all **7 O-Ring failures** from **24 launches** and the temperature (shown to the right).

The consensus from other parties is there is no correlation between number of failures and temperature.

Is this assessment correct? **Is anything missing**?

temperature	o_ring_failures
53	2
57	1
58	1
63	1
70	1
70	1
75	2



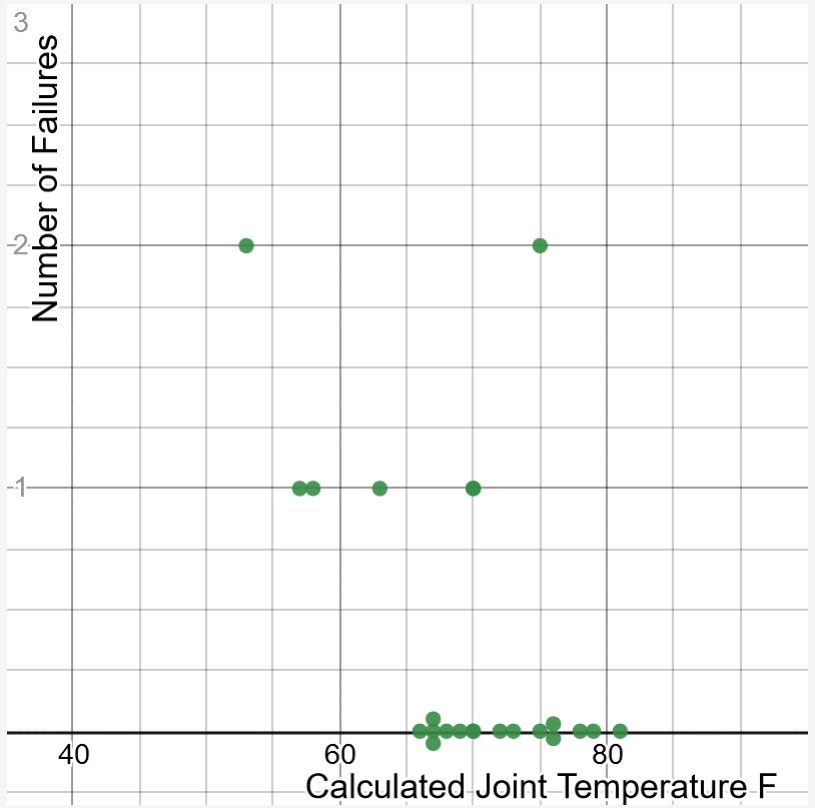
Data Source: http://www.math.usu.edu/~symanzik/teaching/2009_stat6560/RDataAndScripts/sharif_abbass_project1_challenger.csv

Exercise: “There’s No Correlation”

Notice that data from successful launches were not included, which may tell an entirely different story.

What are your thoughts now? Is there a correlation? Is there a model that can be used to predict risk?

temperature	o_ring_failures
53	2
57	1
58	1
63	1
66	0
67	0
67	0
67	0
68	0
69	0
70	1
70	0
70	1
70	0
72	0
73	0
75	0
75	2
76	0
76	0
78	0
79	0
81	0



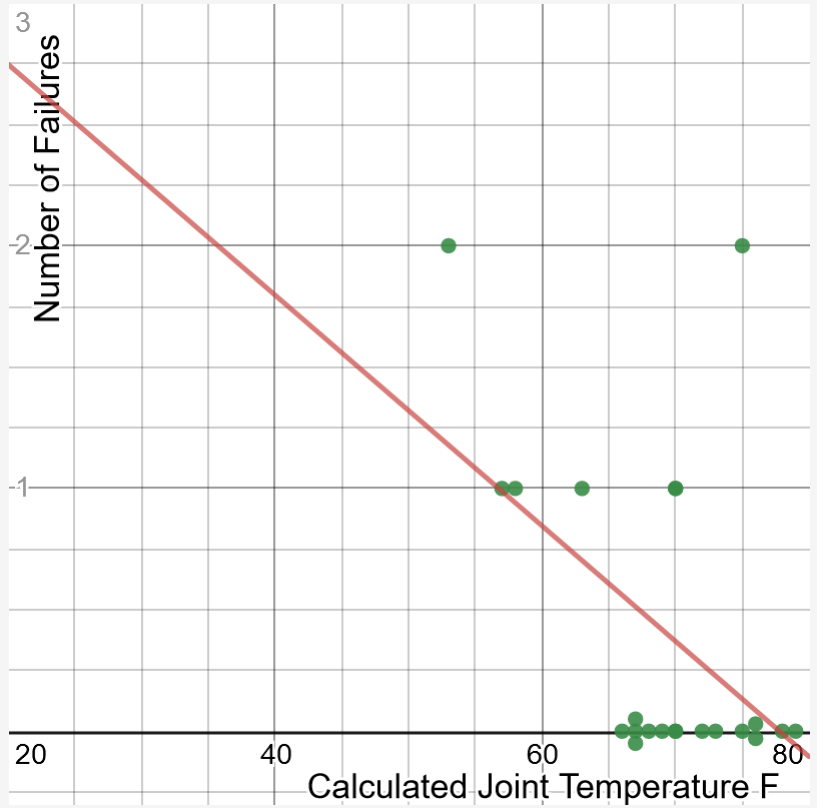
Exercise: "There's No Correlation"

You can try to apply a linear regression here, and while it does show a trend it is a little awkward especially since our data is sparse.

Should we transform our data somehow?

Are there any other models we can try?

temperature	o_ring_failures
53	2
57	1
58	1
63	1
66	0
67	0
67	0
67	0
68	0
69	0
70	1
70	0
70	1
70	0
72	0
73	0
75	0
75	2
76	0
76	0
78	0
79	0
81	0



Linear Regression Source Code

```
import pandas as pd
from sklearn.linear_model import LinearRegression

# Learn more: https://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

# Import points

df = pd.read_csv('https://bit.ly/2DgjTk5', 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]

# Plain ordinary least squares
fit = LinearRegression().fit(X, Y)

# Print "m" and "b" coefficients
print("m = {0}".format(fit.coef_.flatten()))
print("b = {0}".format(fit.intercept_.flatten()))
```

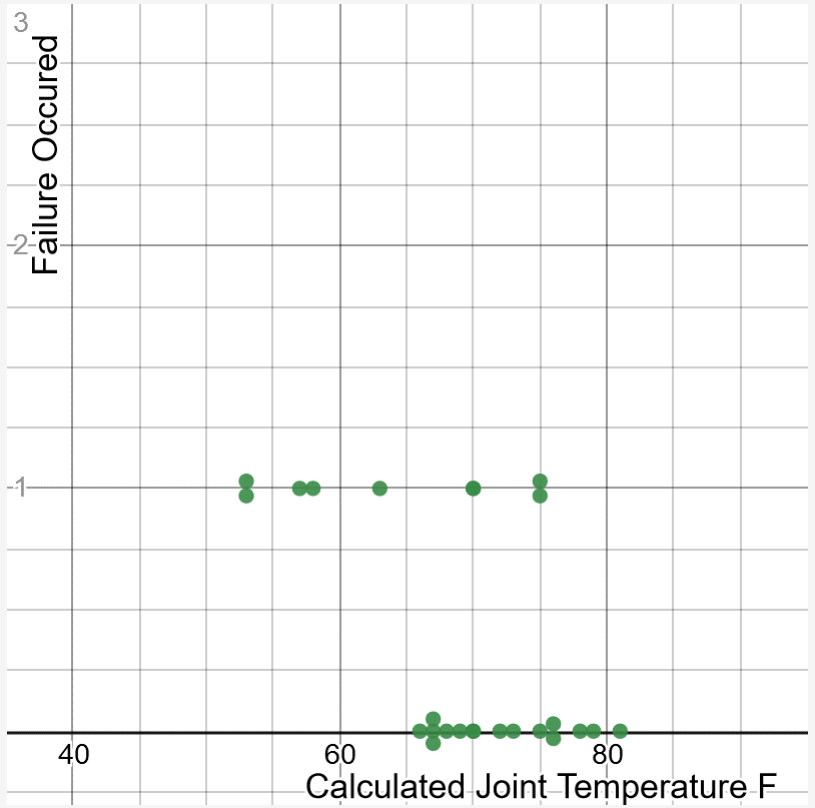
Exercise: “There’s No Correlation”

What if we converted the data to be binary, showing whether a failure occurred or not occurred, by separating each instance into its own record?

This reduces the domain of output variables to “0” and “1” creating a binary model.

Is a story now becoming clear? What model can we use to predict probability of failure at a given temperature?

temperature	o_ring_failures
53	1
53	1
57	1
58	1
63	1
66	0
67	0
67	0
67	0
68	0
69	0
70	1
70	0
70	1
70	0
72	0
73	0
75	0
75	1
75	1
76	0
76	0
78	0
79	0
81	0



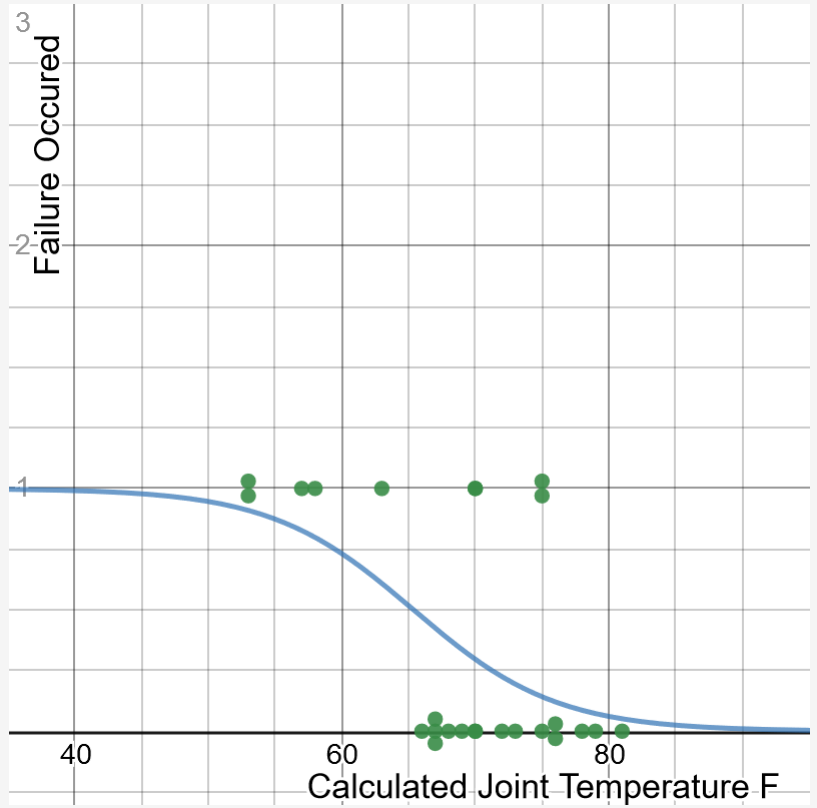
Exercise: "There's No Correlation"

Logistic regression might be the best way to model this risk.

Even though we lack freezing temperature data, the logistic regression points to a high probability of risk for O-ring failure.

If our launch is going to happen in freezing temperatures, this does not bode well.

temperature	o_ring_failures
53	1
53	1
57	1
58	1
63	1
66	0
67	0
67	0
67	0
68	0
69	0
70	1
70	0
70	1
70	0
72	0
73	0
75	0
75	1
75	1
76	0
76	0
78	0
79	0
81	0



Exercise: “There’s No Correlation”

This is exactly what happened to the space shuttle Challenger on January 28, 1986, and if you already have not figured out already, we are doing the analysis.

Through a series of unfortunate events, only partial data was accessible and omitted non-failure data, which showed a correlation with temperature and O-ring failure.

The analysis we just did should have happened before the accident, but unfortunately it occurred afterwards.

We will learn later about how biases, class imbalance, and broken data can derail machine learning and statistical models.

The space shuttle Challenger just moments before disaster on January 28, 1986 (above) and Richard Feynman famously demonstrating O-ring failure with a glass of ice water (left).

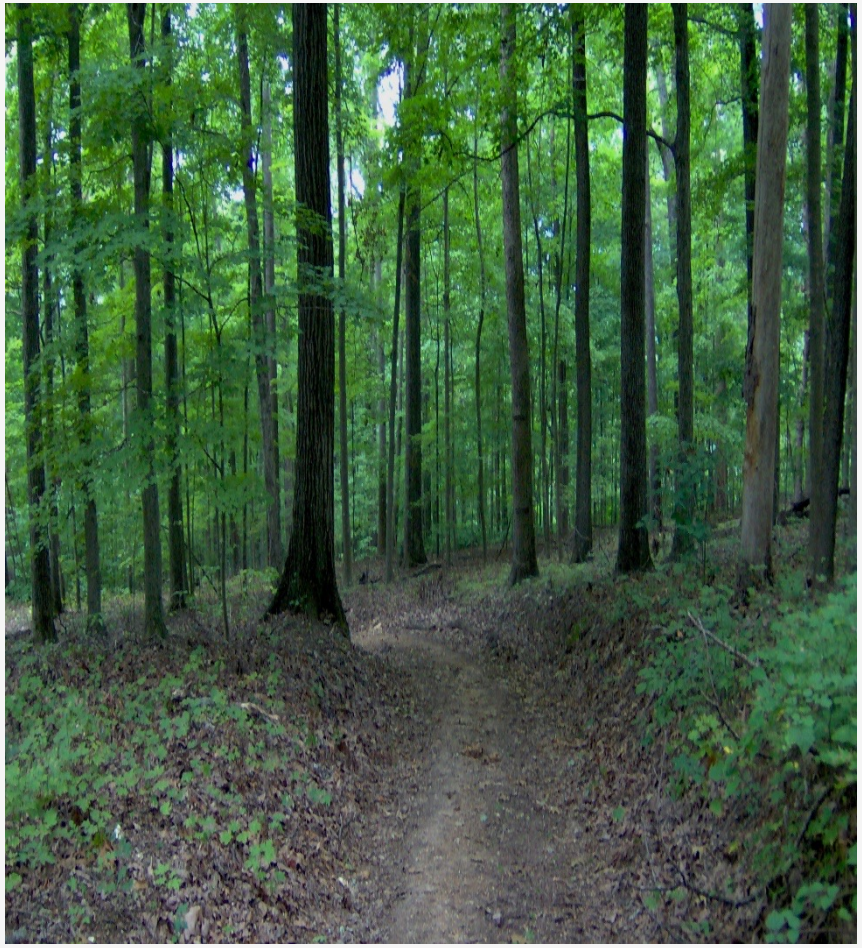
Section V
Decision Trees and Random Forests

What Are Decision Trees?

Decision trees are a powerful machine learning tool and work well for a lot of machine learning problems.

As a matter of fact, decisions trees work so well they are notorious for overfitting.

- This can be remedied with random forests which generates hundreds of decision trees with randomly sampled data.
- Decision trees can also be improved with gradient boosting and other techniques.
- Other flavors of decision trees exist, like regression trees.



Decision Tree Intuition

We have some weather data and labeled each record as being “1” (good weather) or “0” (bad weather).

We could solve this using a logistic regression but let us try using decision trees instead.

RAIN	LIGHTNING	CLOUDY	TEMPERATURE	GOOD_WEATHER_IND
0	1	1	74	0
0	0	0	69	1
1	0	1	58	0
0	0	0	71	1
0	0	0	73	1
0	1	1	80	0
0	1	1	74	0
0	0	0	73	1
…				

Decision Tree Intuition

Start with the variable that is most likely to separate good weather and bad weather as best as possible.

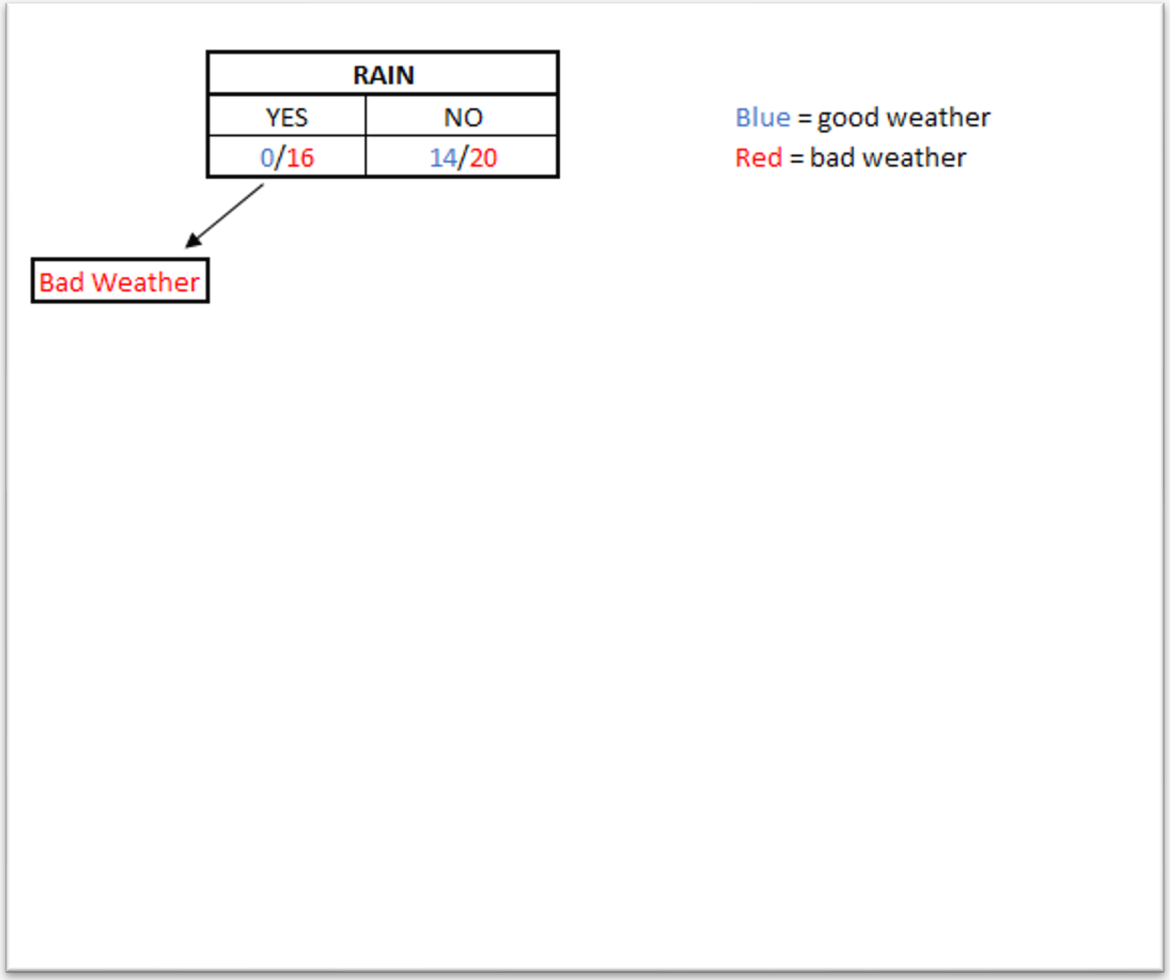
For reasons we will discuss later, you determine ***RAIN*** is good at separating good weather and bad weather, so you bucket records like this:

RAIN				
YES	NO		Blue = good weather	
0/16	14/20		Red = bad weather	

Decision Tree Intuition

Since all records where ***RAIN*** was present always yielded bad weather, we will simply predict any new record with ***RAIN*** as bad weather.

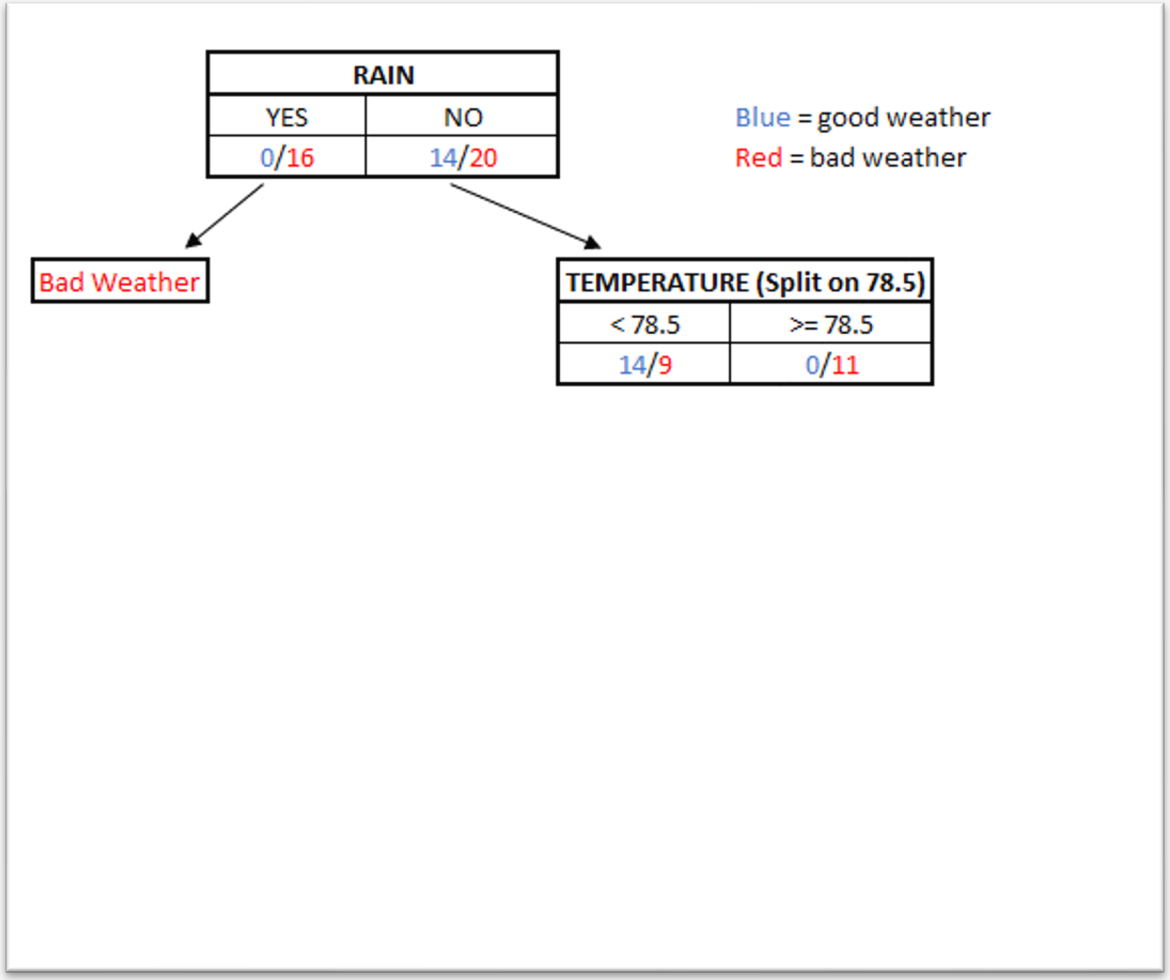
However, records that did not have ***RAIN*** are mixed, where some are good weather, and others are bad weather. These will need to be split again.



Decision Tree Intuition

We take records that did not have ***RAIN*** and now split them on ***TEMPERATURE***.

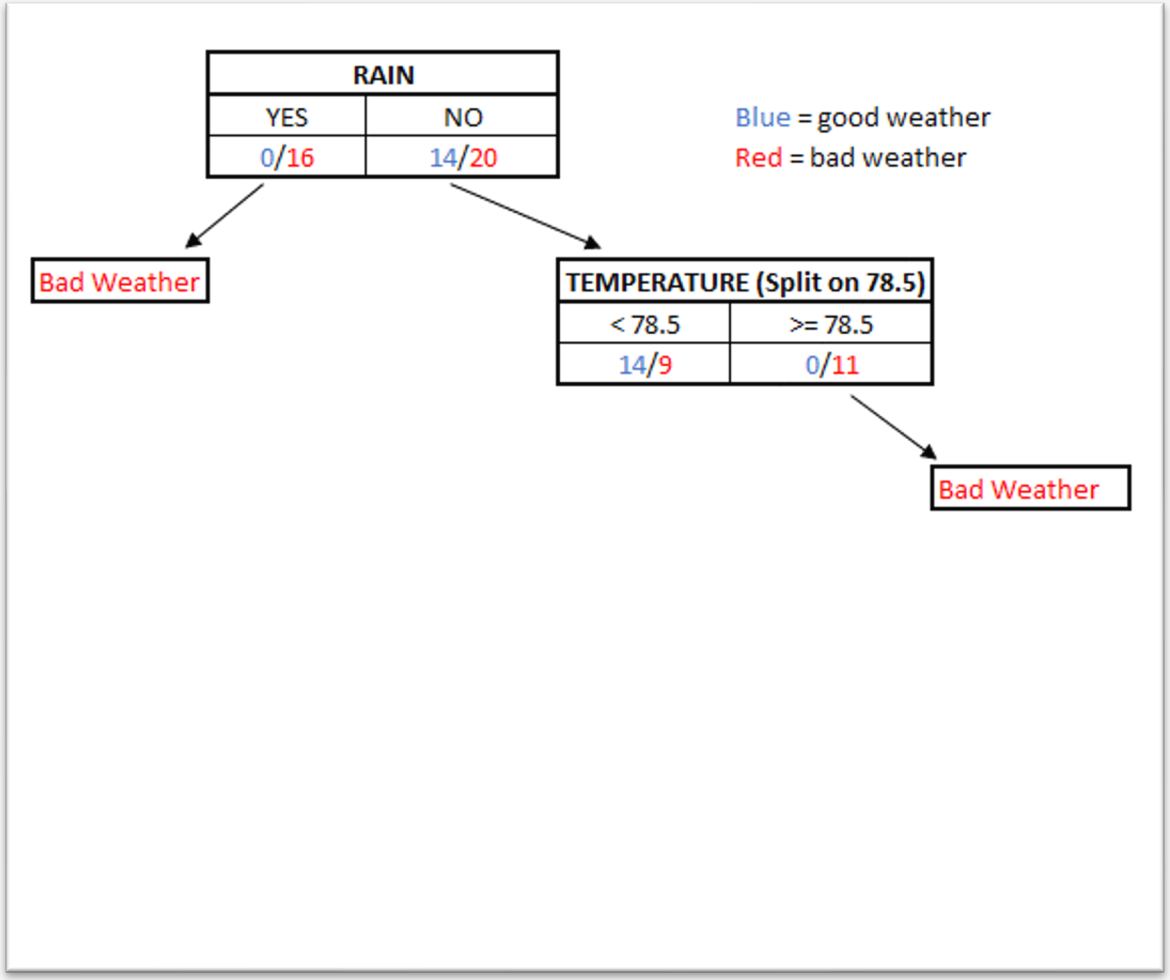
Since ***TEMPERATURE*** is continuous and not a simple *yes/no* binary, we split on a value 78.5 that optimizes the split (more on this later).



Decision Tree Intuition

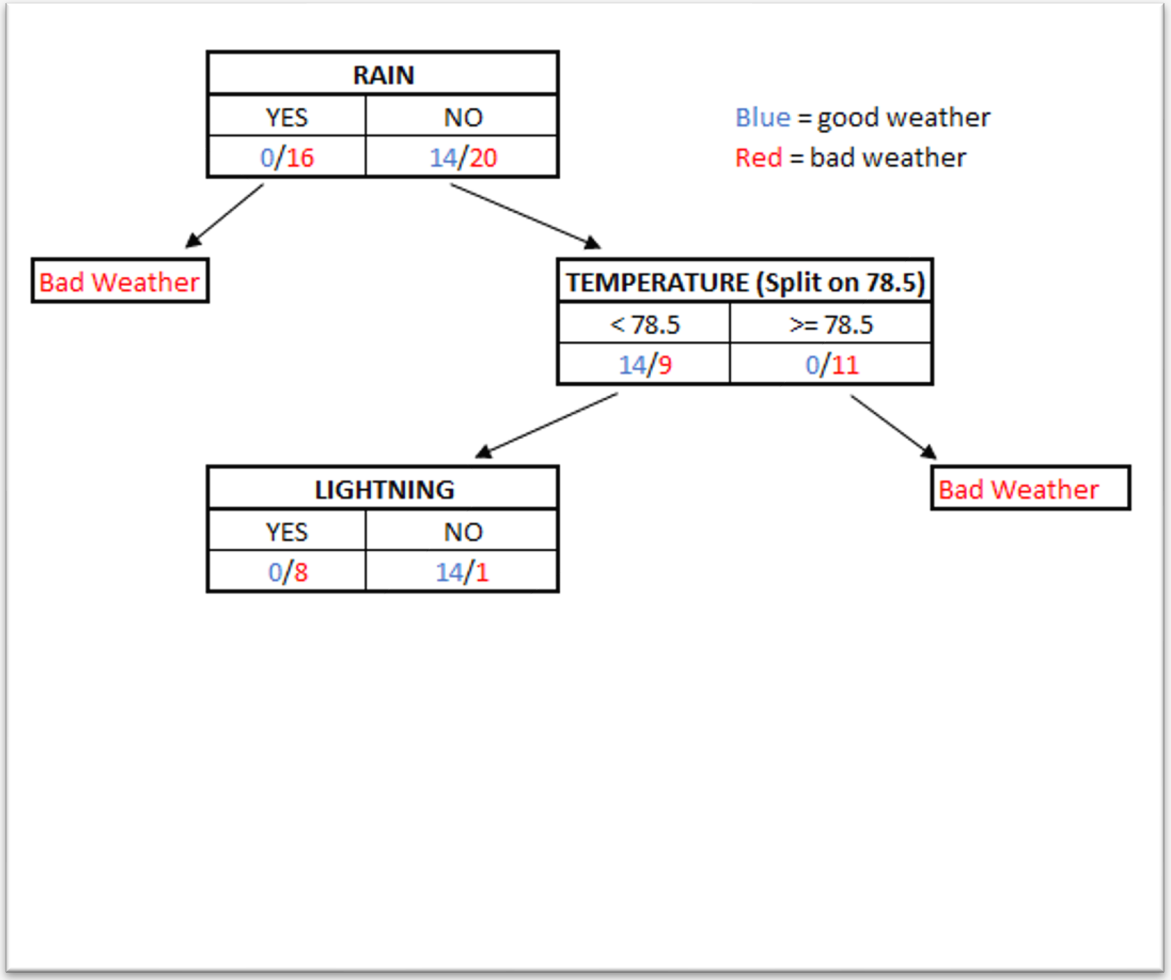
Notice how all records with ***TEMPERATURE*** greater than/equal to 78.5 are bad weather, so we will predict any records with a ***TEMPERATURE*** at least 78.5 as bad weather.

But records with a ***TEMPERATURE*** less than 78.5 have a mix of good weather and bad weather. So we need to split those again.



Decision Tree Intuition

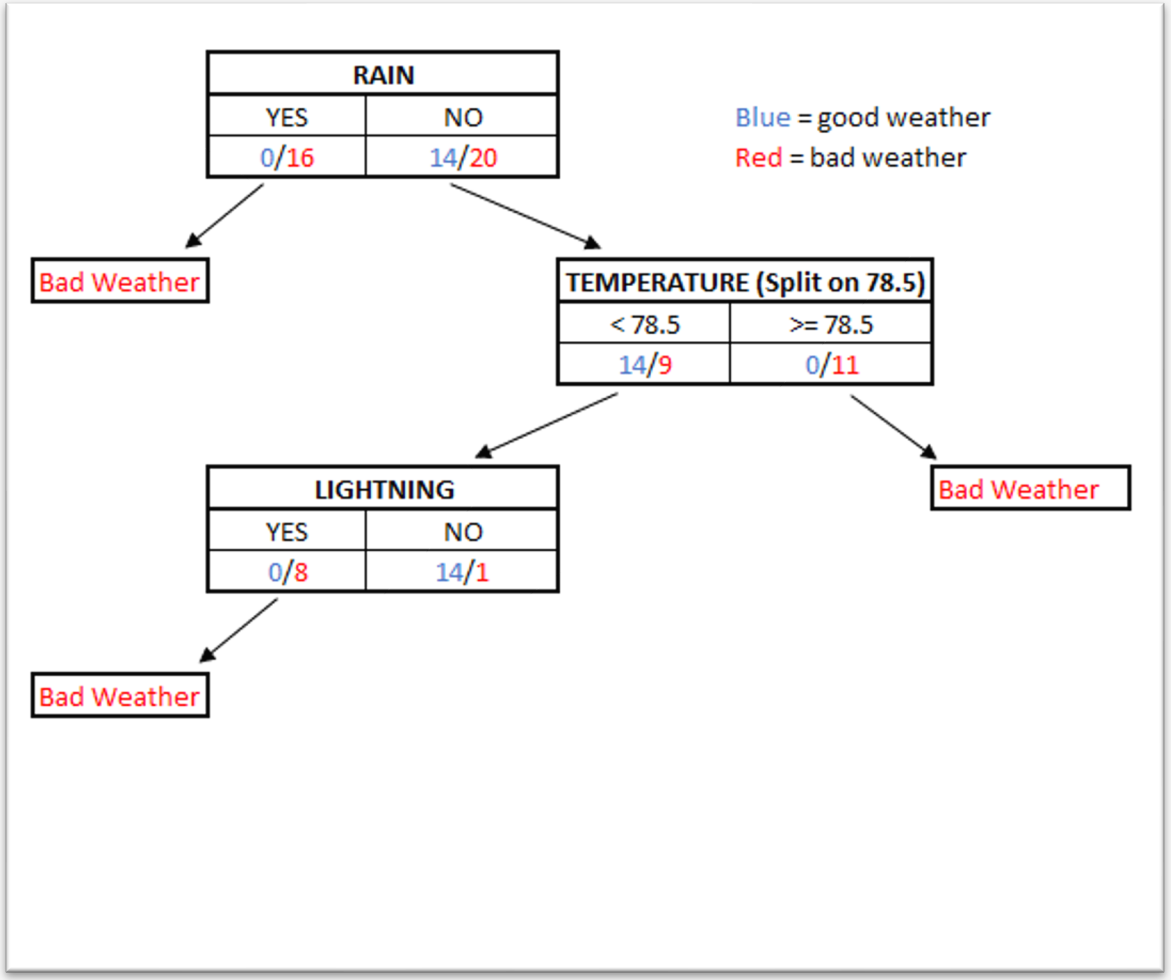
We take those records with ***TEMPERATURE*** less than 78.5 and now split them on whether ***LIGHTNING*** was present.



Decision Tree Intuition

We take those records with ***TEMPERATURE*** less than 78.5 and now split them on whether ***LIGHTNING*** was present.

Notice how all records with ***LIGHTNING*** are labelled as bad weather, so we can predict any new record with ***LIGHTNING*** as bad weather.

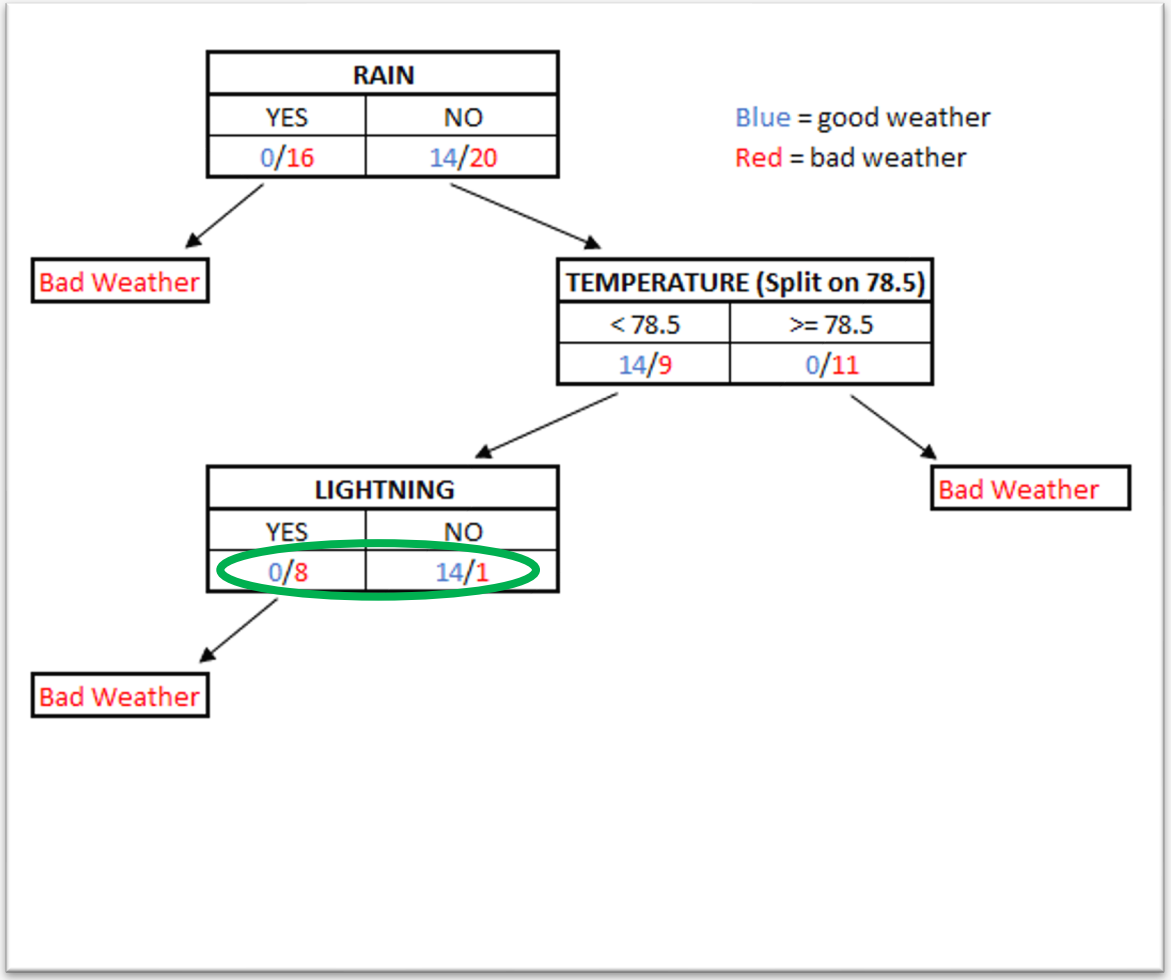


Decision Tree Intuition

We take those records with ***TEMPERATURE*** less than 78.5 and now split them on whether ***LIGHTNING*** was present.

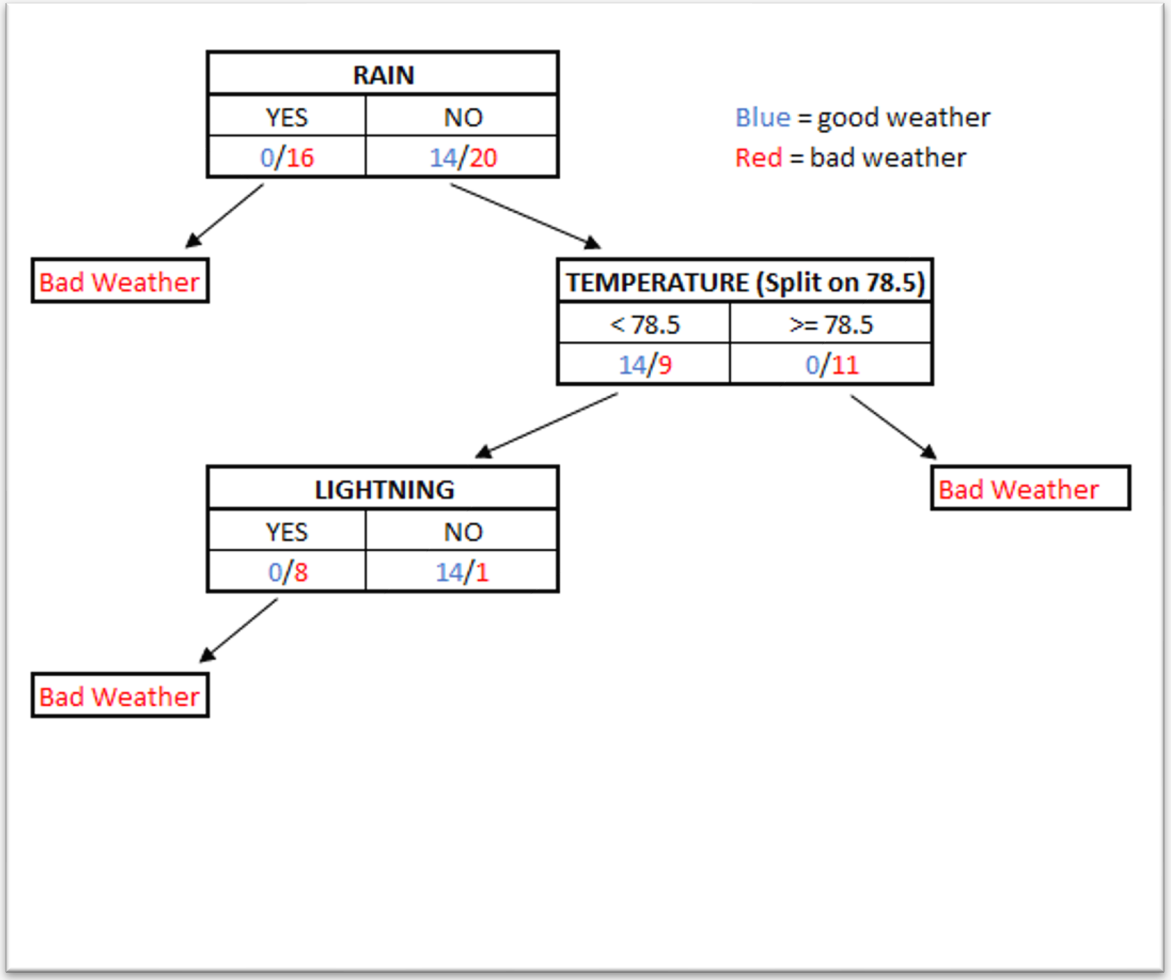
Notice how all records with ***LIGHTNING*** are labelled as bad weather, so we can predict any new record with ***LIGHTNING*** as bad weather.

Notice how close we are to a perfect clean split now!



Decision Tree Intuition

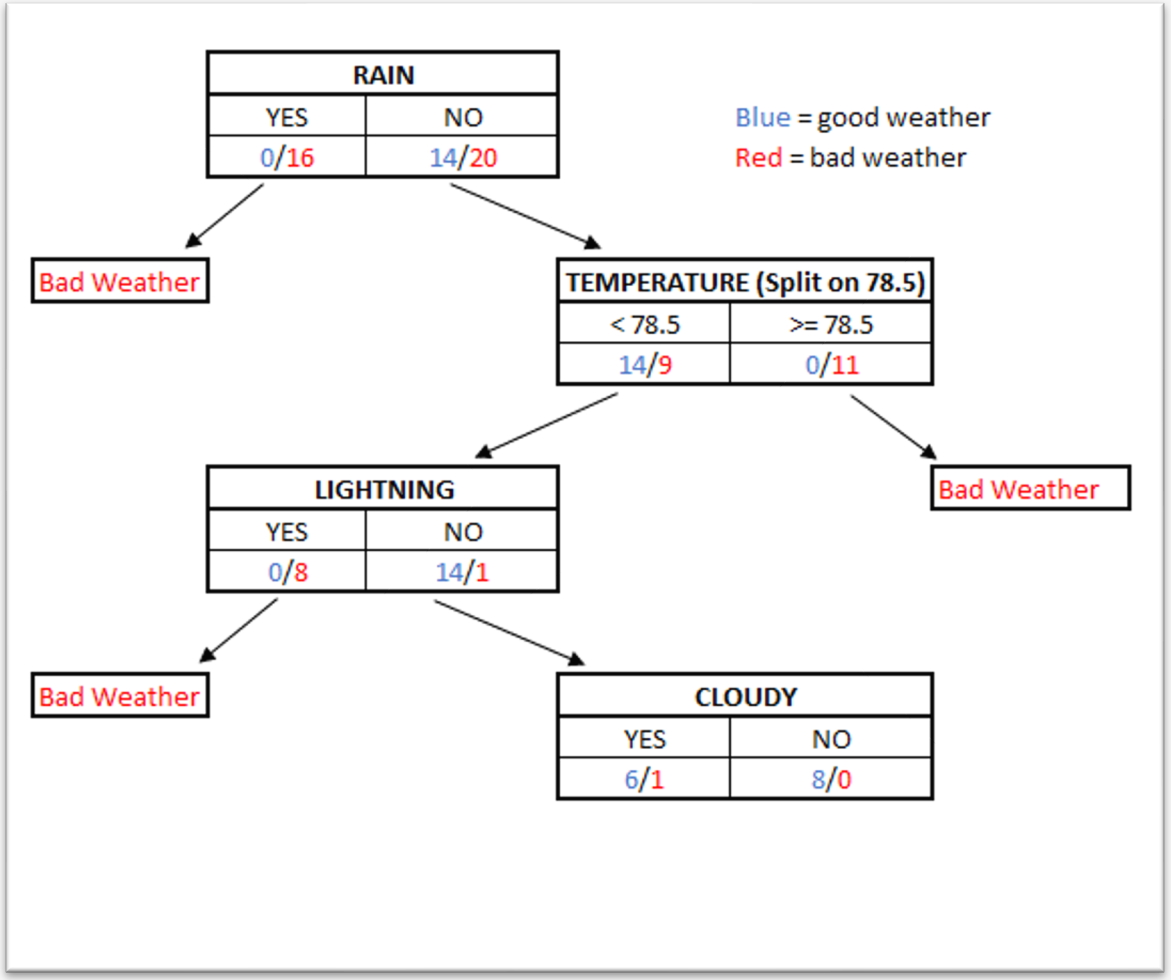
LIGHTNING almost gave us a perfect split, where lightning being present would predict bad weather and no lightning *almost* always meant good weather.



Decision Tree Intuition

LIGHTNING almost gave us a perfect split, where lightning being present would predict bad weather and no lightning *almost* always meant good weather.

We could probably stop here, but let's split these remaining records with no lightning. Finally we split on whether it was ***CLOUDY*** or not.

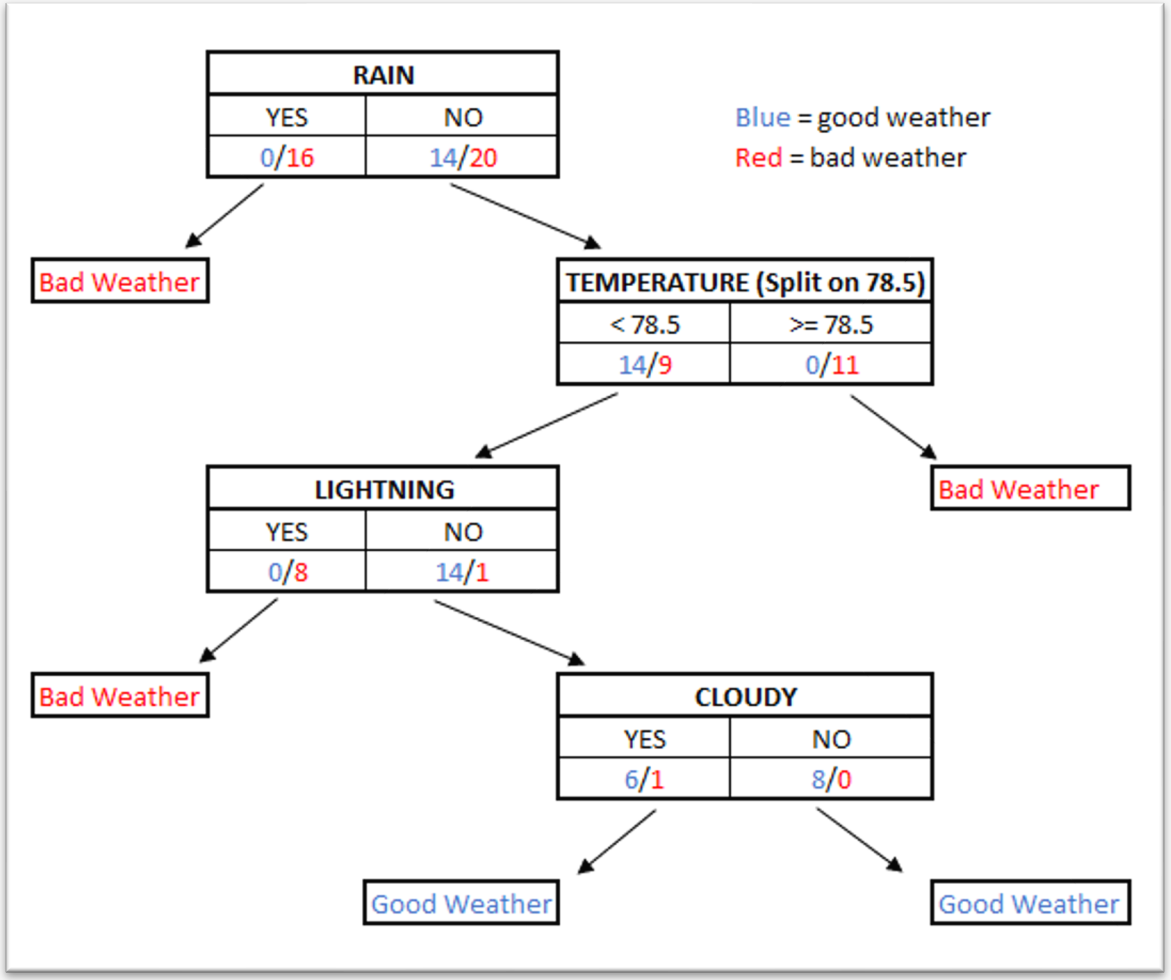


Decision Tree Intuition

At this point, we establish we cannot separate productively anymore and ***CLOUDY*** does not have any impact to our prediction.

It will always be good weather, given we already established there was no ***LIGHTNING***, the ***TEMPERATURE*** is less than 78.5, and there is no ***RAIN***.

Congratulations! We just built our first decision tree. But you might have a few questions…

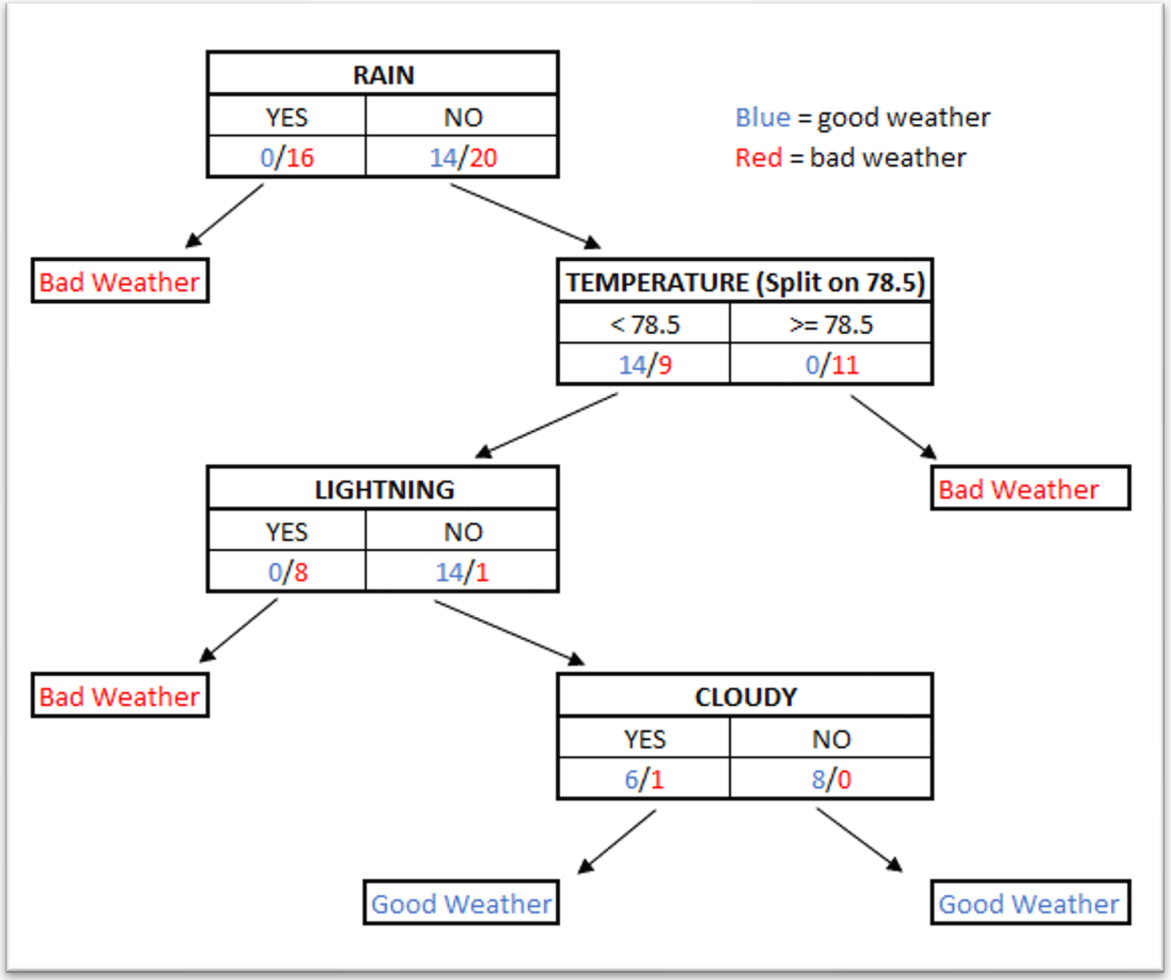


Decision Trees – Getting to the "Decision" Part

Hopefully by now, you have a strong intuition on what decision trees are trying to accomplish.

However, you may have some lingering questions on ***HOW*** to build a decision tree.

- At a given step, how do I determine which property is the best to split on?
- Where do I split continuous variables like ***TEMPERATURE***?
- When do I stop splitting and end my decision tree?



Gini Impurity

To build a decision tree, we must discuss the concept of **impurity** which describes how mixed something is.

EXAMPLE: If we have 6 dogs and 3 cats in a kennel, we do not purely have dogs or cats.

Gini Impurity is a common way to measure impurity using this function (for events *A* and *B*):

$$1 - (Probability\ of\ A)^2 - (Probability\ of\ B)^2$$

So the Gini impurity of dogs versus cats in the kennel:

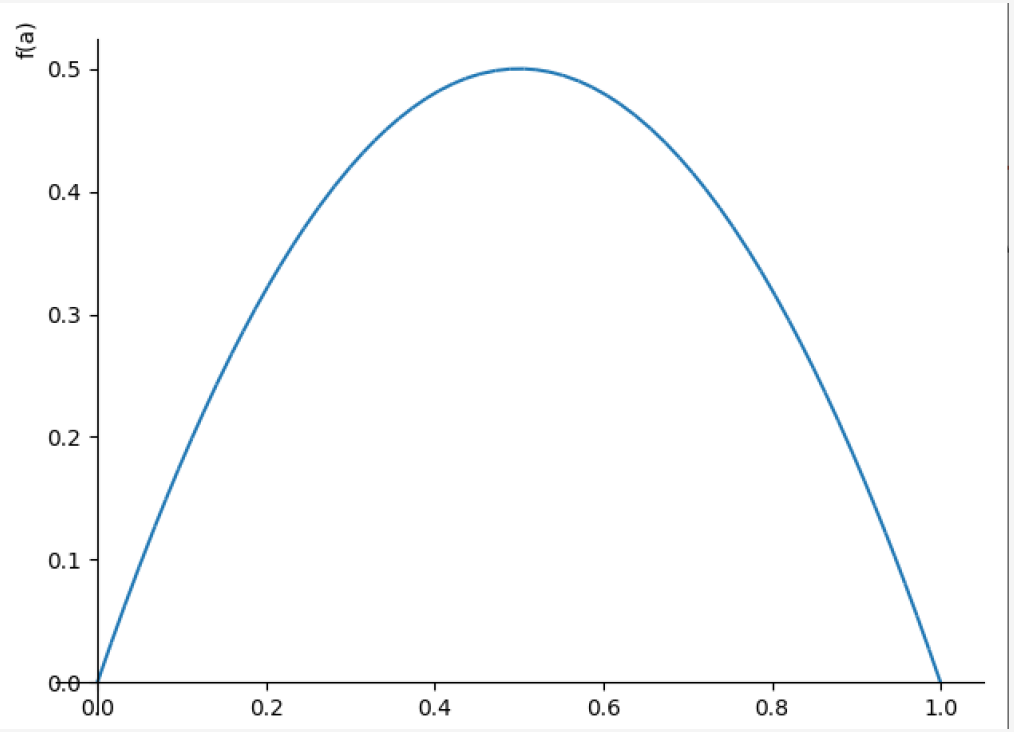
$$1 - \left(\frac{6}{6+3}\right)^2 - \left(\frac{3}{6+3}\right)^2 = .44444$$

Gini Impurity – Plotting for the Curious

The Gini impurity can never exceed 0.5, and this makes sense because having something 100% mixed (1.0) doesn't make sense.

To the right we use SymPy to plot the gini impurity function.

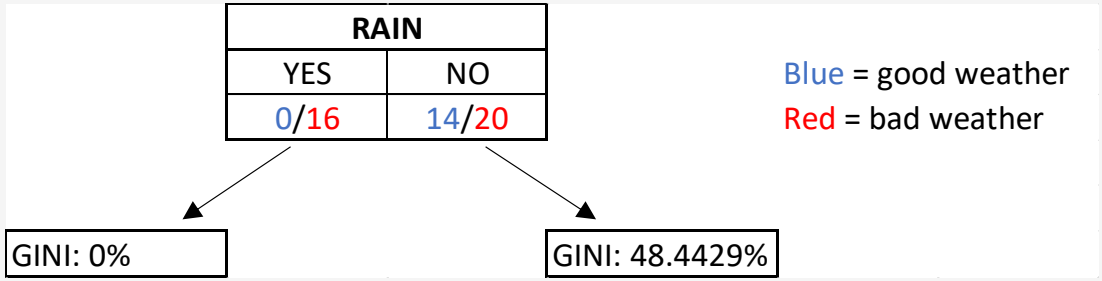
```
from sympy import *

a = symbols('a')
b = 1.0 - a
gini_impurity_f = 1 - a**2 - b**2
plot(gini_impurity_f, (a, 0, 1))
```



Gini Impurity

Going back to the weather example, here's how we look at Gini impurity for ***RAIN***.

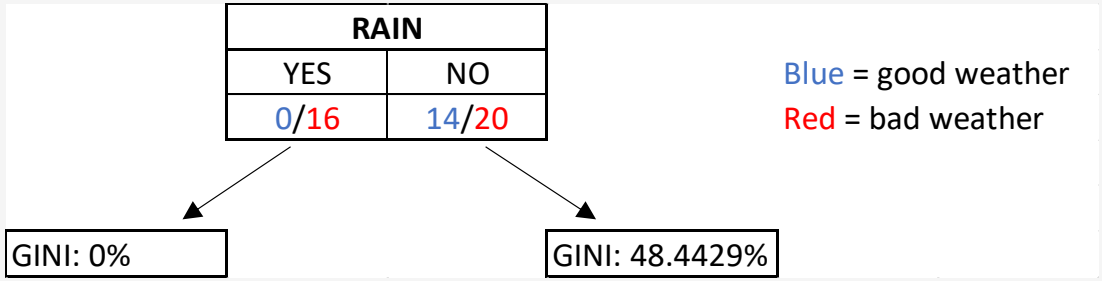
Notice how the impurity for "good/bad weather" on the YES side is 0% meaning it is pure.



$$1 - \left(\frac{0}{0+16}\right)^2 - \left(\frac{16}{0+16}\right)^2 = 0$$

Gini Impurity

Going back to the weather example, here's how we look at Gini impurity for ***RAIN***.



Notice how the impurity for "good/bad weather" on the YES side is 0% meaning it is pure.

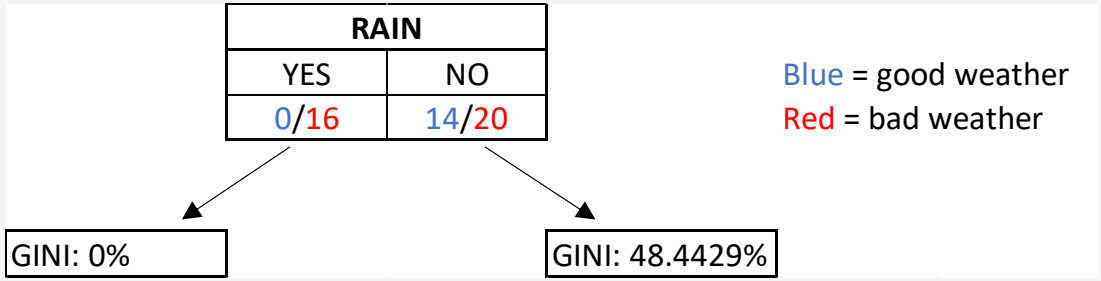
$$1 - \left(\frac{0}{0+16}\right)^2 - \left(\frac{16}{0+16}\right)^2 = 0$$

On the NO side we got an impurity of 48.4429%

$$1 - \left(\frac{14}{14+20}\right)^2 - \left(\frac{20}{14+20}\right)^2 = .484429$$

Weighing Gini Impurities

To calculate the entire impurity of using the ***RAIN*** property for a split, we weight these two impurities together.



$$0\ \frac{0+16}{0+16+14+20} + 0.484429\frac{14+20}{0+16+14+20} = .32941172$$

This is known as the **weighted average Gini impurity**, and it can be used as a measure of quality for splitting with that property. *Lowest impurity is best!*

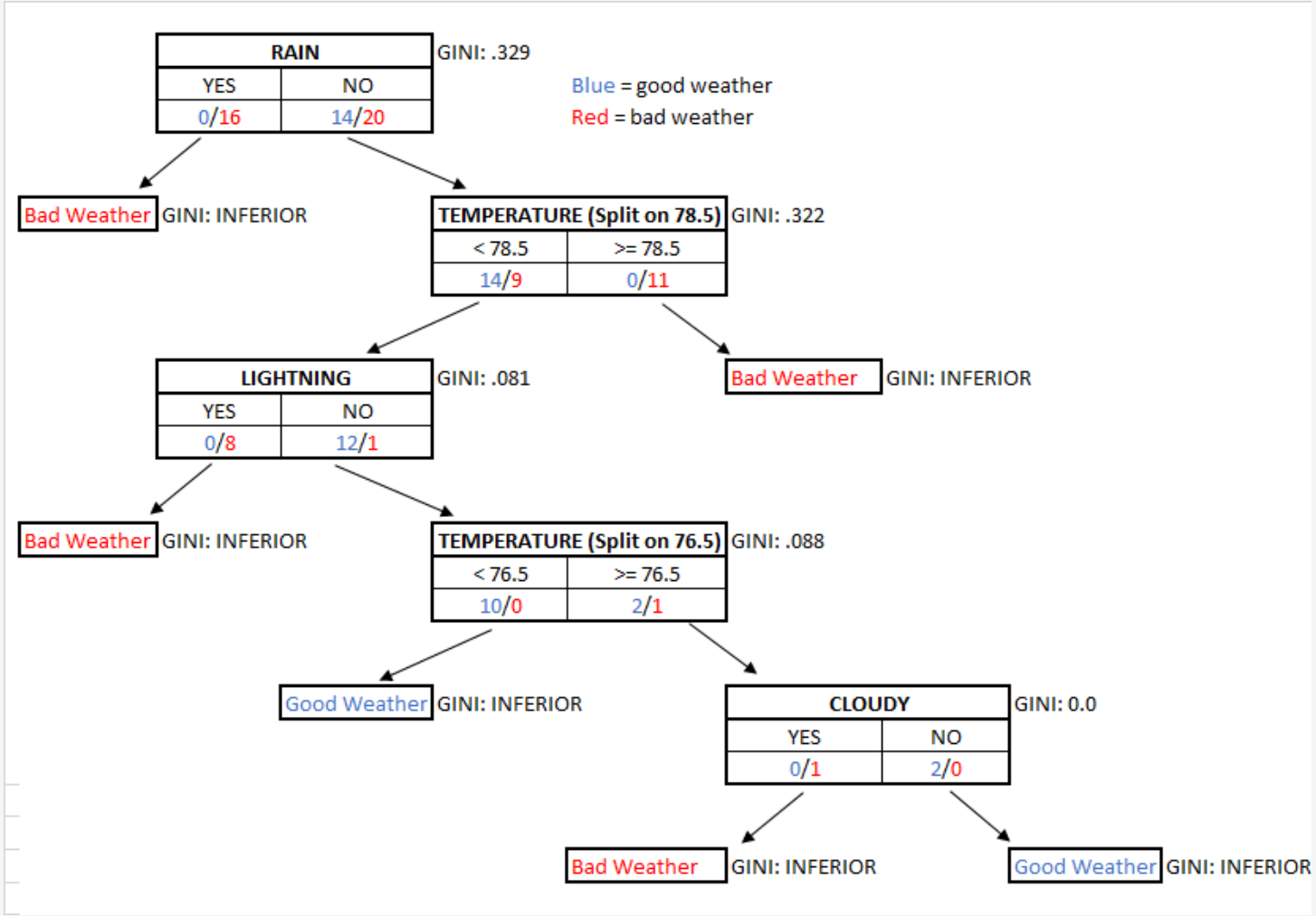
At every step we choose the property that provides the least weighted average Gini impurity.

Building a Decision Tree with GINI

At each decision, we choose the next property with the best GINI impurity and split on it

EXAMPLE: After ***RAIN,*** the property ***TEMPERATURE*** has the next best GINI for those 34 items on the "NO" side.

When the next property's weighted GINI is inferior to the previous GINI, we stop the branch there.



How to Split Continuous Variables?

Once you got the GINI and weighted GINI down, implementing a decision tree becomes relatively simple.

The goal always is to do a split that reduces the GINI impurity until it cannot be reduced anymore.

The only remaining question is how to split on continuous variables, which is not as straightforward as binary (true/false) or discrete (dog/cat/bird) variables.

Observe the following temperatures on the right. Any guesses on what is the best way to find the optimal split?

TEMERATURE
74
76
77
80
81
83
84

TEMPERATURE (Split on 78.5)	
< 78.5	>= 78.5
14/9	0/11

How to Split Continuous Variables?

Continuous variables extend the GINI concept further.

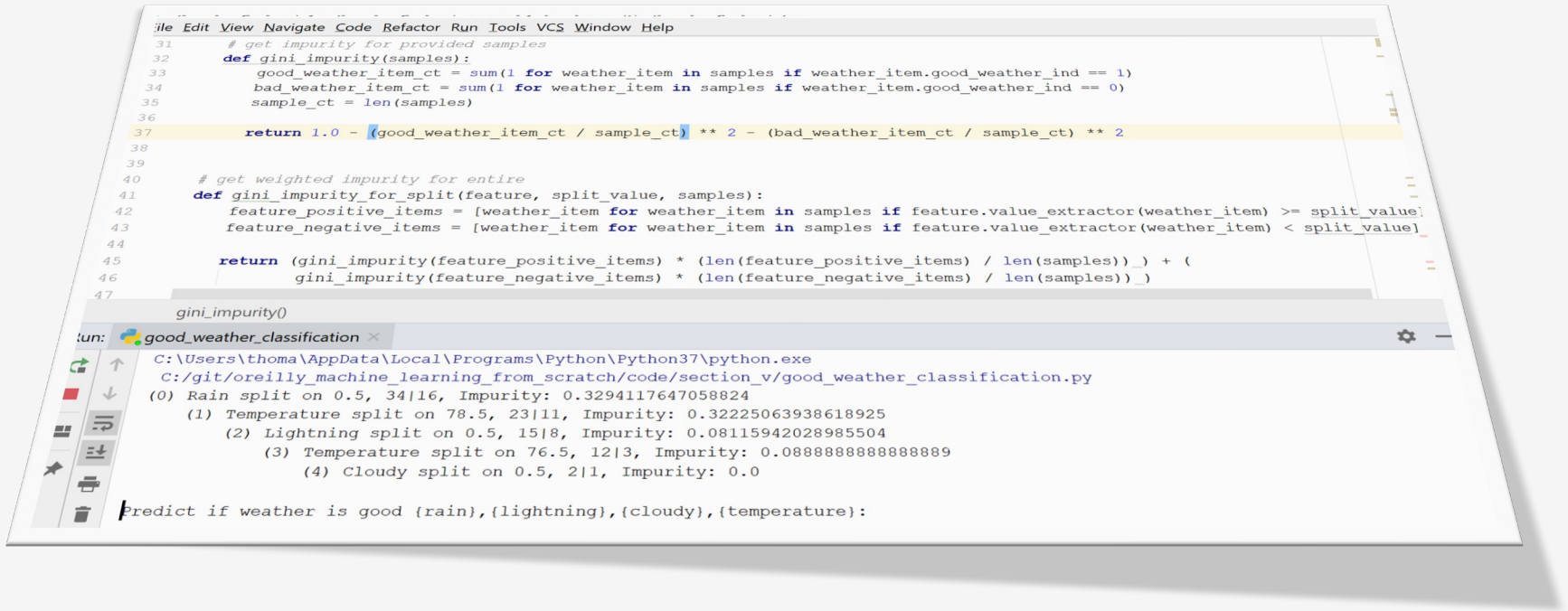
When you evaluate a continuous variable for its GINI, first sort the data for that variable then produce the rolling 2-value averages, each of which is a candidate for the split value.

Then choose the 2-value average that produces the best GINI impurity that splits on that value, which for the Temperature node we saw previously was 78.5.

TEMERATURE	ROLLING 2-VALUE AVG
74	
76	75
77	76.5
80	**78.5**
81	80.5
83	82
84	83.5

TEMPERATURE (Split on 78.5)	
< 78.5	>= 78.5
14/9	0/11

Hands On: Decision Trees



Decision Trees and Overfitting

Decision trees work well, so much they are notorious for overfitting.

Overfitting again means the model fits to the training data too exactly, and therefore becomes unreliable for predicting new data.

One way to adapt decision trees to not overfit is to utilize random forests, which generates hundreds of decision trees with randomly sampled data and features.

This forces the model to use different subsets of data and properties, and therefore not overfit to the data set.

Random Forests

Random Forests are a machine learning technique that generates hundreds of decision trees, where each one builds off partial random data and properties, rather than all the data.

- Typically each decision tree will train with only 2/3 of the randomly sampled data, which is known as **bootstrapping**.
- You should also only consider a subset of variables when evaluating each node, forcing your decision tree to utilize other variables.
- You can use the other 1/3 of the data (known as the **"out-of-bag" data**) as the test data to evaluate prediction performance.

With these hundreds of decision trees built, you then have each tree "vote" on a prediction. The prediction with the highest votes wins.

Fine-Tuning Random Forests

Because each decision tree in a random forest uses only 2/3 of the data for training, you can use the other 1/3 as test data.

You can choose the number of variables that yields the best accuracy on that test data.

For this exercise, I'm just going to randomly sample 2-3 properties for each node and call it a day.

Hands-On: Random Forests