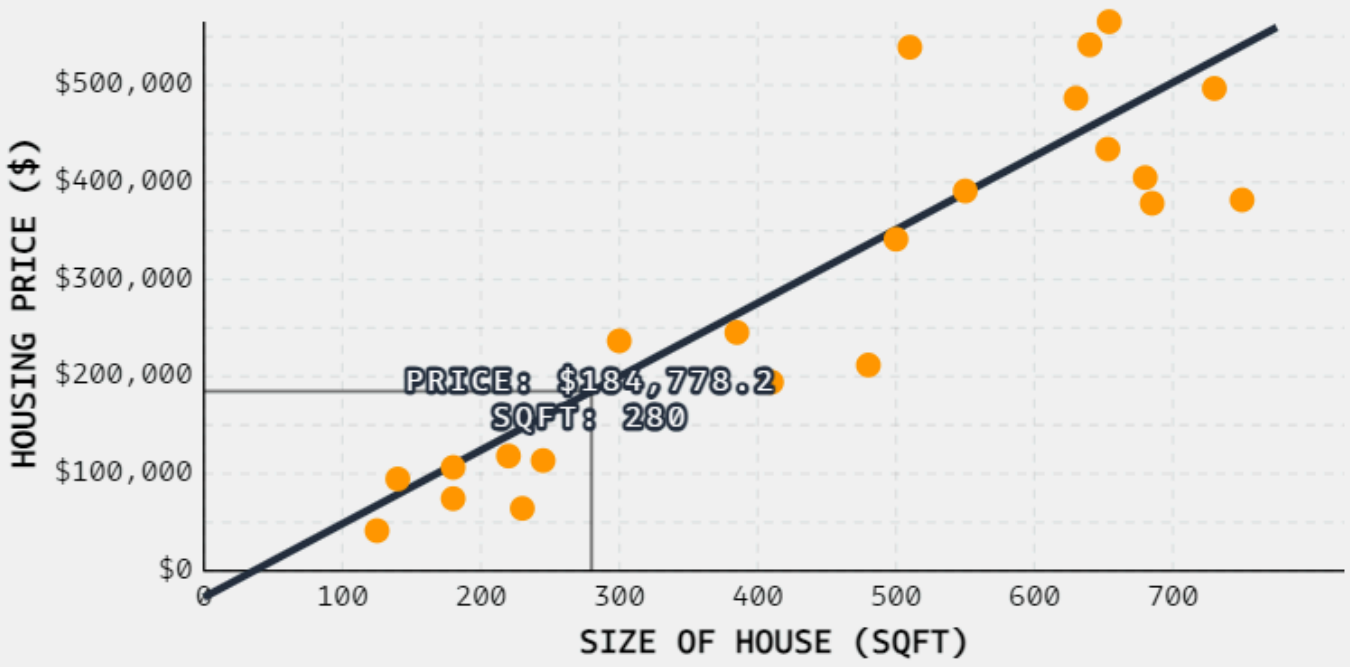
# LINEAR REGRESSION

A Visual Introduction To (Almost)
Everything You Should Know

Linear Regression is a simple and powerful model for predicting a numeric response from a set of one or more independent variables. This article will focus mostly on how the method is used in machine learning, so we won't cover common use cases like causal inference or experimental design. And although it may seem like linear regression is overlooked in modern machine learning's ever-increasing world of complex neural network architectures, the algorithm is still widely used across a large number of domains because it is effective, easy to interpret, and easy to extend. The key ideas in linear regression are recycled everywhere, so understanding the algorithm is a must-have for a strong foundation in machine learning.

## Can you predict the price of a house using linear regression?

Yes



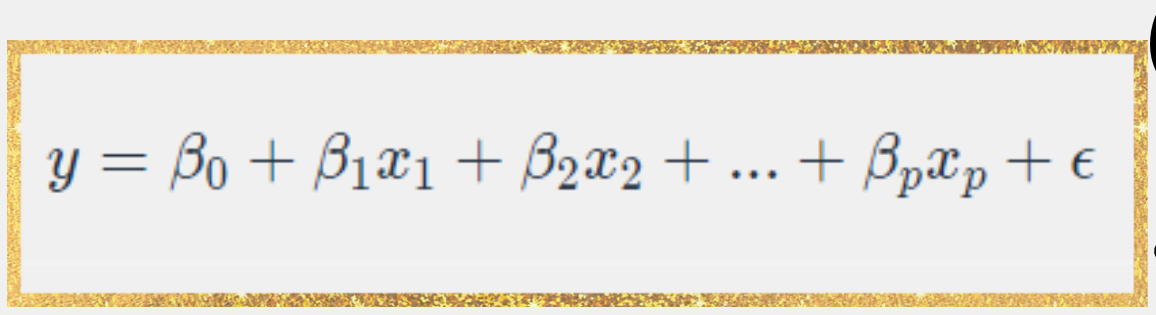

RAB DATA MAGIC

# DON'T AFRAID IT'S SIMPLE MATHS

**Let's Be More Specific**

Linear regression is a supervised algorithm[i] that learns to model a dependent variable, $y$, as a function of some independent variables (aka "features"), $x_i$, by finding a line (or surface) that best "fits" the data. In general, we assume $y$ to be some number and each $x_i$ can be basically anything. For example: predicting the price of a house using the number of rooms in that house ($y$: price, $x_1$: number of rooms) or predicting weight from height and age ($y$: weight, $x_1$: height, $x_2$: age).

In general, the equation for linear regression is




$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_p x_p + \epsilon$$

where:

$y$: the dependent variable; the thing we are trying to predict.[i]

$x_i$: the independent variables: the features our model uses to model y.[i]

$\beta_i$: the coefficients (aka "weights") of our regression model. These are the foundations of our model. They are what our model "learns" during optimization.[i]

$\epsilon$: the irreducible error in our model. A term that collects together all the unmodeled parts of our data.

Fitting a linear regression model is all about finding the set of cofficients that best model $y$ as a function of our features. We may never know the true parameters for our model, but we can estimate them (more on this later). Once we've estimated these coefficients, $\hat{\beta}_i$, we predict future values, $\hat{y}$, as:

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + ... + \hat{\beta}_p x_p$$

So predicting future values (often called inference), is as simple as plugging the values of our features $x_i$ into our equation!

# How It Works, Briefly

To make linear regression easier to digest, let's go through a quick, high-level introduction of how it works. We'll scroll through the core concepts of the algorithm at a high-level, and then delve into the details thereafter:

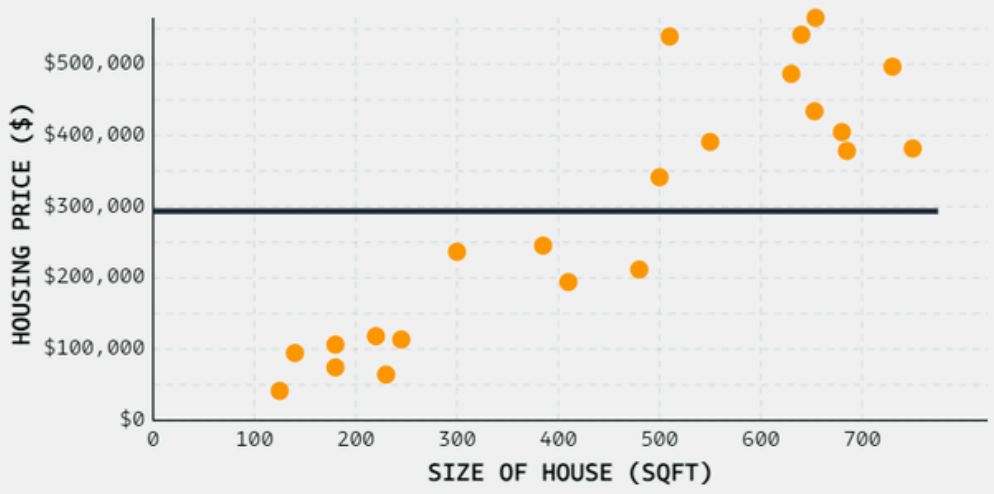
## Step-1

Let's fit a model to predict housing price ($) in San Diego, USA using the size of the house (in square-footage):

$$\text{house-price} = \hat{\beta}_1 * sqft + \hat{\beta}_0$$

We'll start with a very simple model, predicting the price of each house to be just the average house price in our dataset, ~$290,000, ignoring the different sizes of each house:

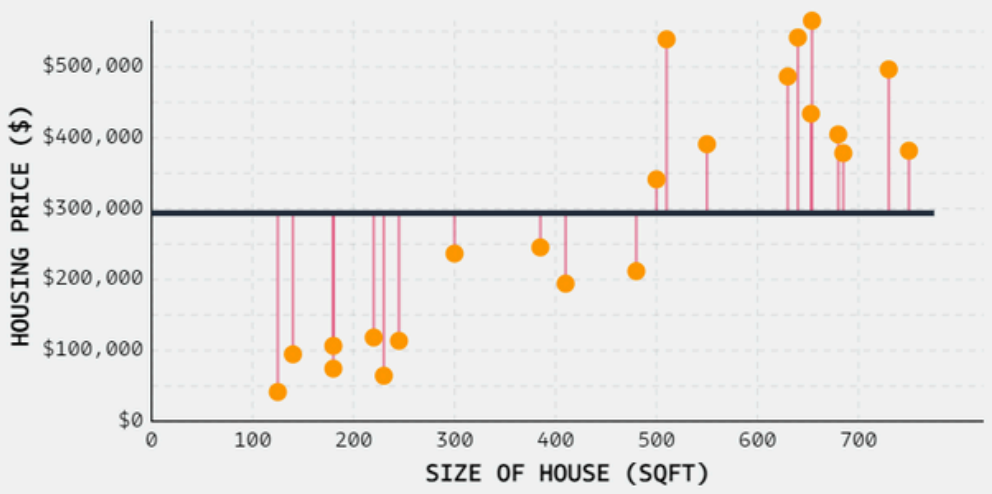
$$\text{house-price} = 0 * sqft + 290000$$




## Step-2

Of course we know this model is bad - the model doesn't fit the data well at all. But how can do quantify exactly *how* bad?

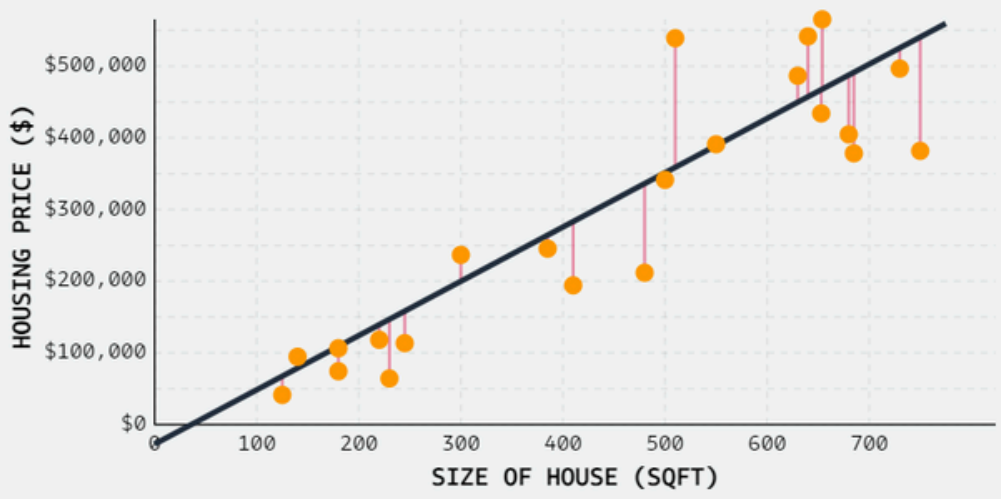
To evaluate our model's performance quantitatively, we plot the error of each observation directly. These errors, or **residuals**, measure the distance between each observation and the predicted value for that observation. We'll make use of these residuals later when we talk about evaluating regression models, but we can clearly see that our model has a lot of error.




## Step-3

The goal of linear regression is reducing this error such that we find a line/surface that 'best' fits our data. For our simple regression problem, that involves estimating the y-intercept and slope of our model, $\hat{\beta}_0$ and $\hat{\beta}_1$.

For our specific problem, the best fit line is shown. There's still error, sure, but the general pattern is captured well. As a result, we can be reasonably confident that if we plug in new values of square-footage, our predicted values of price would be reasonably accurate.




# Let’s Make our first Prediction Model

RAB DATA MAGIC

# PREDICT PRICE OF HOUSE

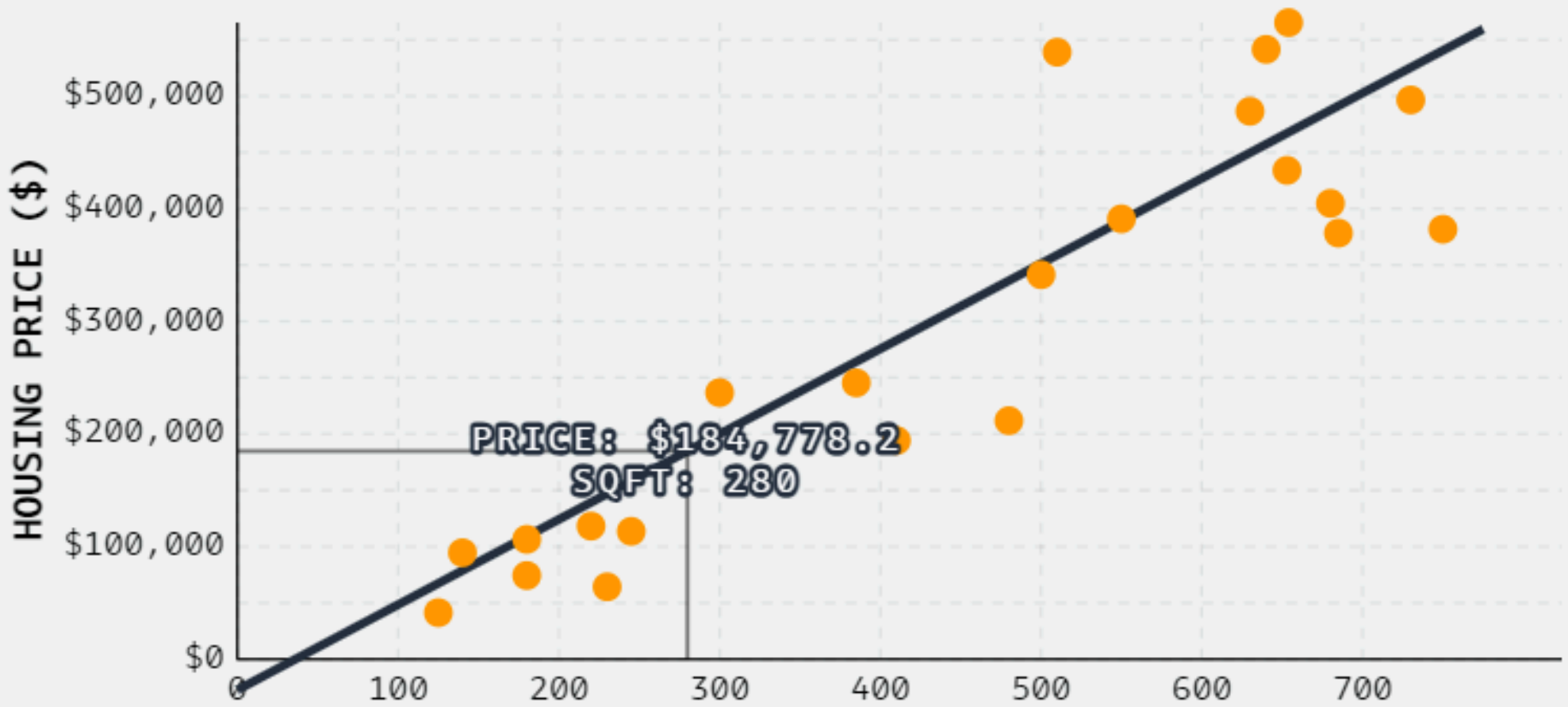
Once we've fit our model, predicting future values is super easy! We just plug in any $x_i$ values into our equation!

For our simple model, that means plugging in a value for $sqft$ into our model:

$sqft$ Value: 280

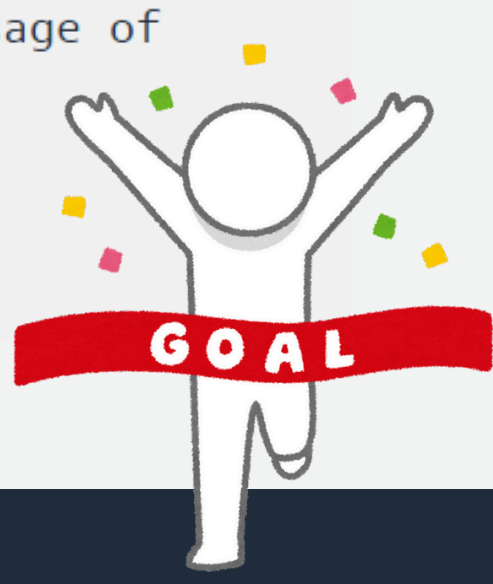
$$\hat{y} = 756.9 * 280 - 27153.8$$
$$\hat{y} = 184778$$

Thus, our model predicts a house that is 280 square-feet will cost $184,778.




Now that we have a high-level idea of how linear regression works, let's dive a bit deeper. The remainder of this article will cover how to evaluate regression models, how to find the "best" model, how to interpret different forms of regression models, and the assumptions underpinning correct usage of regression models in statistical settings.

Let's dive in!




**Let’s learn how to check the accuracy of our model**

# RAB DATA MAGIC

# Model Evaluation 

To train an accurate linear regression model, we need a way to quantify how good (or bad) our model performs. In machine learning, we call such performance-measuring functions *loss functions*. Several popular loss functions exist for regression problems.[i] To measure our model's performance, we'll use one of the most popular: mean-squared error (MSE).

### Mean-Squared Error (MSE)

MSE quantifies how close a predicted value is to the true value, so we'll use it to quantify how close a regression line is to a set of points. MSE works by squaring the distance between each data point and the regression line (the red residuals in the graphs above), summing the squared values, and then dividing by the number of data points:

$$MSE = \frac{1}{n}\Sigma_{i=1}^{n}(y_i - \hat{y}_i)^2$$

The name is quite literal: take the mean of the squared errors. The squaring of errors prevents negative and positive terms from canceling out in the sum,[i] and gives more weight to points further from the regression line, punishing outliers. In practice, we'll fit our regression model to a set training data, and evaluate it's performance using MSE on the test dataset.

### R-Squared

Regression models may also be evaluated with the so-called *goodness of fit* measures, which summarize how well a model fits a set of data. The most popular goodness of fit measure for linear regression is r-squared, a metric that represents the percentage of the variance in $y$ explained by our features $x$.[i] More specifically, r-squared measures the percentage of variance explained normalized against the baseline variance of our model (which is just the variance of the mean):

$$R^2 = 1 - \frac{\Sigma_{i=1}^{n}(y_i - \hat{y}_i)^2}{\Sigma_{i=1}^{n}(y_i - \bar{y})^2}$$

The highest possible value for r-squared is 1, representing a model that captures 100% of the variance. A negative r-squared means that our model is doing worse (capturing less variance) than a flat line through mean of our data would.

To build intuition for yourself, try changing the weight and bias terms below to see how the MSE and r-squared change across different model fits:

Shuffle Data

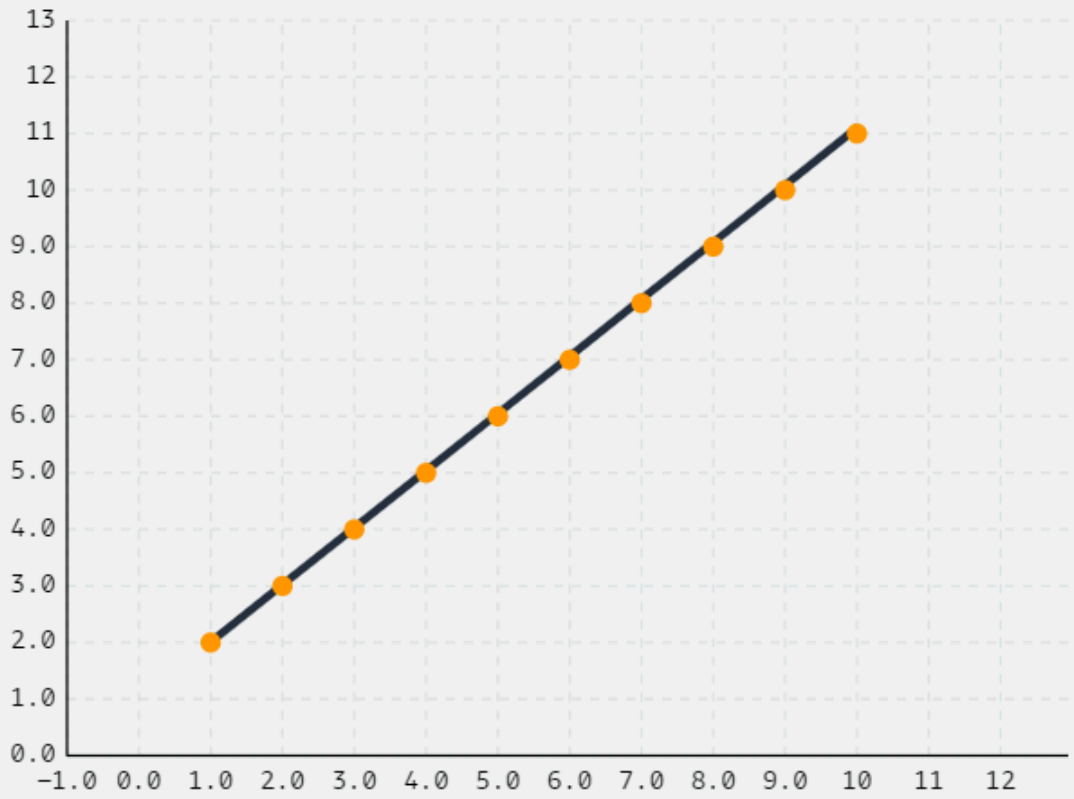
Bias ($\hat{\beta}_0$): 1.00

Weight ($\hat{\beta}_1$): 1.01

For our model: $\hat{y} = 1.01x + 1.00$

$$MSE = \frac{1}{n}\Sigma_{i=1}^{n}(y - (1.01x + 1.00))^2 = 0.00$$

$$R^2 = 1 - \frac{0.04}{82.50} = 1.00$$




Shuffle Data

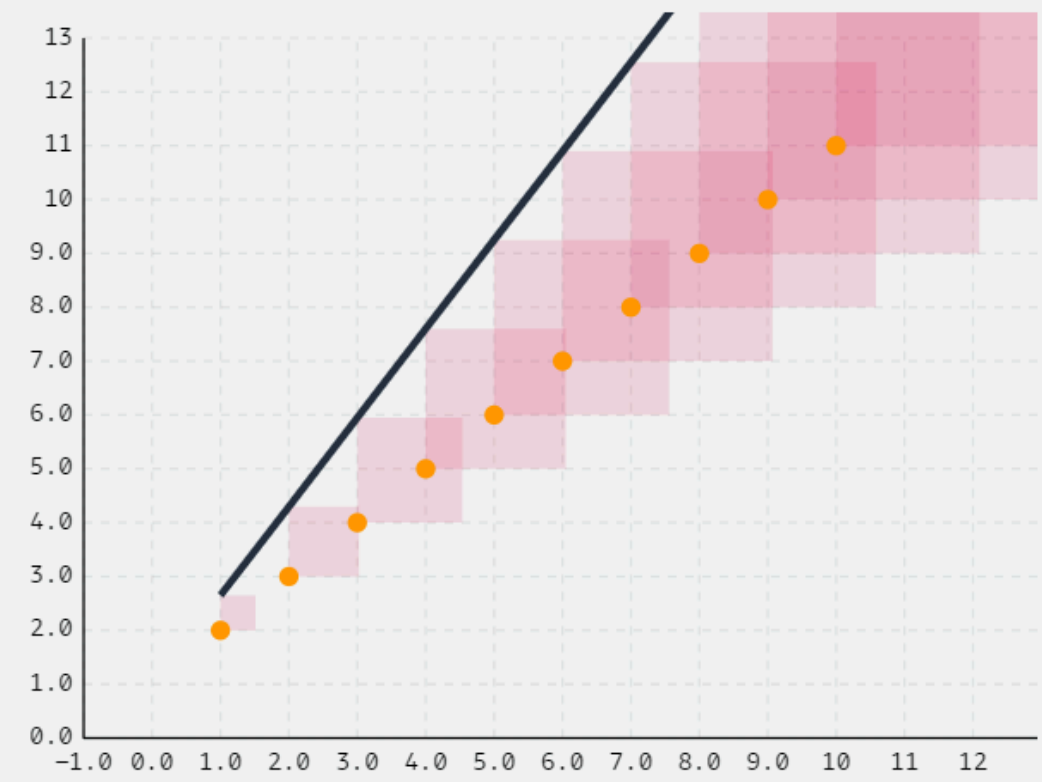
Bias ($\hat{\beta}_0$): 1.00

Weight ($\hat{\beta}_1$): 1.65

For our model: $\hat{y} = 1.65x + 1.00$

$$MSE = \frac{1}{n}\Sigma_{i=1}^{n}(y - (1.65x + 1.00))^2 = 16.27$$

$$R^2 = 1 - \frac{162.66}{82.50} = -0.97$$




Shuffle Data

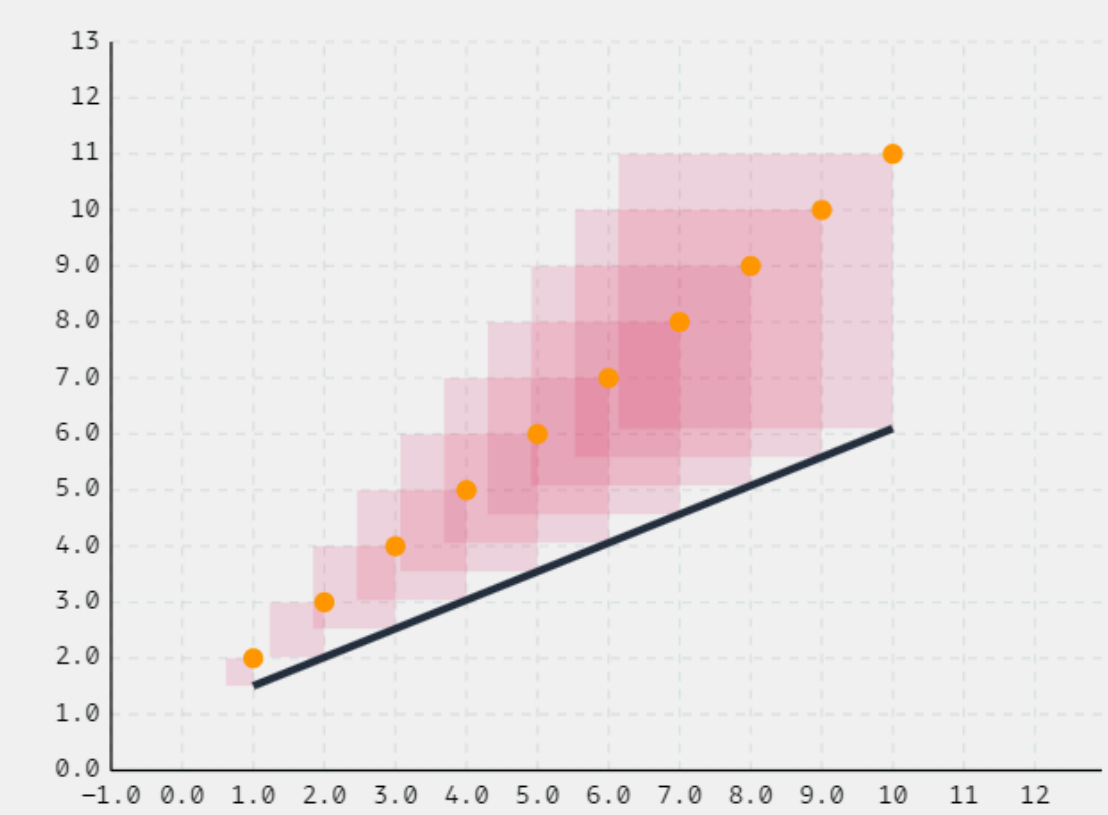
Bias ($\hat{\beta}_0$): 1.00

Weight ($\hat{\beta}_1$): 0.51

For our model: $\hat{y} = 0.51x + 1.00$

$$MSE = \frac{1}{n}\Sigma_{i=1}^{n}(y - (0.51x + 1.00))^2 = 9.24$$

$$R^2 = 1 - \frac{92.44}{82.50} = -0.12$$




You will often see R-Squared referenced in statistical contexts as a way to assess model fit.

### Selecting An Evaluation Metric

Many methods exist for evaluating regression models, each with different concerns around interpretability, theory, and usability. The evaluation metric should reflect whatever it is you actually care about when making predictions. For example, when we use MSE, we are implicitly saying that we think the cost of our prediction error should reflect the quadratic (squared) distance between what we predicted and what is correct. This may work well if we want to punish outliers or if our data is minimized by the mean, but this comes at the cost of interpretability: we output our error in squared units (though this may be fixed with **RMSE**). If instead we wanted our error to reflect the linear distance between what we predicted and what is correct, or we wanted our data minimized by the median, we could try something like Mean Abosulte Error (**MAE**). Whatever the case, you should be thinking of your evaluation metric as part of your modeling process, and select the best metric based on the specific concerns of your use-case.

## Learning The Coefficients

Let's recap what we've learned so far: Linear regression is all about finding a line (or surface) that fits our data well. And as we just saw, this involves selecting the coefficients for our model that minimize our evaluation metric. But how can we best estimate these coefficients? In practice, they're unknown, and selecting them by hand quickly becomes infeasible for regression models with many features. There must be a better way!

Luckily for us, several algorithms exist to do just this. We'll discuss two: an iterative solution and a closed-form solution.

### An Iterative Solution

### A Closed-Form Solution

## An Iterative Solution

Gradient descent is an iterative optimization algorithm that estimates some set of coefficients to yield the minimum of a convex function. Put simply: it will find suitable coefficients for our regression model that minimize prediction error (remember, lower MSE equals better model).

A full conversation on gradient descent is outside the course of this article (stay-tuned for our future article on the subject), but if you'd like to learn more, click the "Show Math" button below. Otherwise, read on!

Show Math


Gradient descent works as follows. We assume that we have some convex function representing the error of our machine learning algorithm (in our case, MSE). Gradient descent will iteratively update our model's coefficients in the direction of our error functions minimum [i] .

In our case, our model takes the form:

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1$$

and our error function takes the form:

$$MSE(\hat{\beta}_0, \hat{\beta}_1) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$= \frac{1}{n}\sum_{i=1}^{n}(y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_1))^2$$

Our goal is to find the coefficients, $\beta_0$ and $\beta_1$, to minimize the error function. To do this, we'll use the gradient, which represents the direction that the function is increasing, and the rate at which it is increasing. Since we want to find the minimum of this function, we can go in the opposite direction of where it's increasing. This is exactly what Gradient Descent does, it works by taking steps in the direction opposite of where our error function is increasing, proportional to the rate of change. To find the coefficients that minimize the function, we first calculate the derivatives of our error function is increasing. To find the coefficients that minimize first, calculate the derivatives of our loss function, MSE:

$$\frac{\delta}{\delta\beta_i} MSE = \begin{cases} -\frac{2}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i) \text{for i} = 0 \\ -\frac{2}{n}\sum_{i=1}^{n} x_i(y_i - \hat{y}_i) \text{for i} = 1 \end{cases}$$

Now that we have the gradients for our error function (with respect to each coefficient to be updated), we perform iterative updates:

$$\text{repeat until converge:} = \begin{cases} \beta_0 = \beta_0 - \alpha(-\frac{2}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)) \\ \beta_1 = \beta_1 - \alpha(-\frac{2}{n} x_i \sum_{i=1}^{n}(y_i - \hat{y}_i)) \end{cases}$$

Updating these values iteratively will yield coefficients of our model that minimize error.

Gradient descent will iteratively identify the coefficients our model needs to fit the data. Let's see an example directly. We'll fit data to our equation $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1$, so gradient descent will learn two coefficients, $\beta_0$ (the intercept) and $\beta_1$ (the weight). To do so, interact with the plot below. Try dragging the weights and values to create a 'poorly' fit (large error) solution and run gradient descent to see the error iteratively improve.

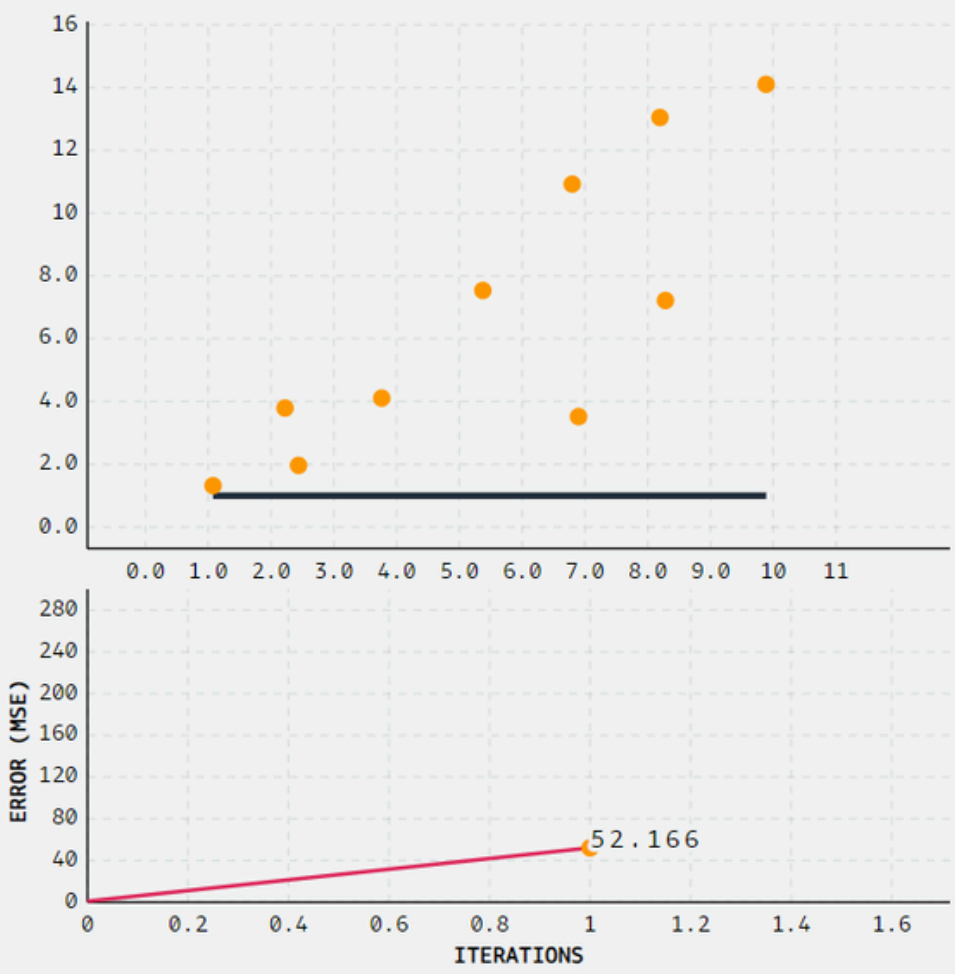
Click the buttons to run 1, 10, or 100 steps of gradient descent, and see the linear regression model update live. The error at each iteration of gradient descent (or manual coefficient update) is shown in the bottom chart. With each weight update, we recalculate the error, so you can see how gradient descent improves our model iteratively.

New Data | 1 Step | 25 Steps | 100 Steps

Bias ($\hat{\beta}_0$): 1.000

Weight ($\beta_1$): 0.000

Our model: $y = 0.000x + 1.000 + c$



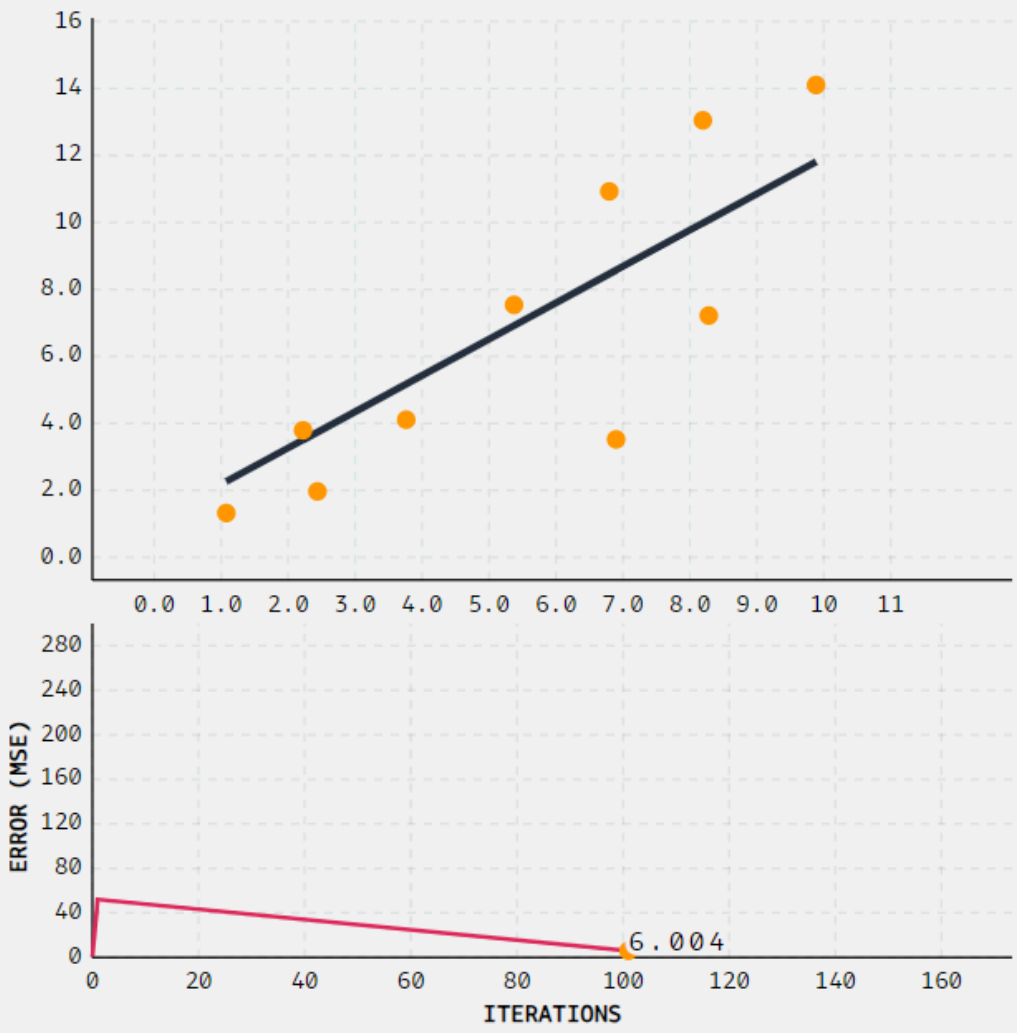

Click the buttons to run 1, 10, or 100 steps of gradient descent, and see the linear regression model update live. The error at each iteration of gradient descent (or manual coefficient update) is shown in the bottom chart. With each weight update, we recalculate the error, so you can see how gradient descent improves our model iteratively.

New Data | 1 Step | 25 Steps | 100 Steps

Bias ($\hat{\beta}_0$): 1.094

Weight ($\beta_1$): 1.084

Our model: $y = 1.084x + 1.094 + c$




Although gradient descent is the most popular optimization algorithm in machine learning, it's not perfect! It doesn't work for every loss function, and it may not always find the most optimal set of coefficients for your model. Still, it has many extensions to help solve these issues, and is widely used across machine learning.

### A Closed-Form Solution

We'd be remiss not to mention the Normal Equation, a widely taught method for obtaining estimates for our linear regression coefficients. The Normal Equation is a closed-form solution that allows us to estimate our coefficients directly by minimizing the *residual sum of squares* (RSS) of our data:

$$RSS = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

The RSS should look familiar - it was a key piece in both the MSE and r-squared formulas that represents our model's total squared error:

$$\hat{\beta} = (X^T X)^{-1} X^T Y$$

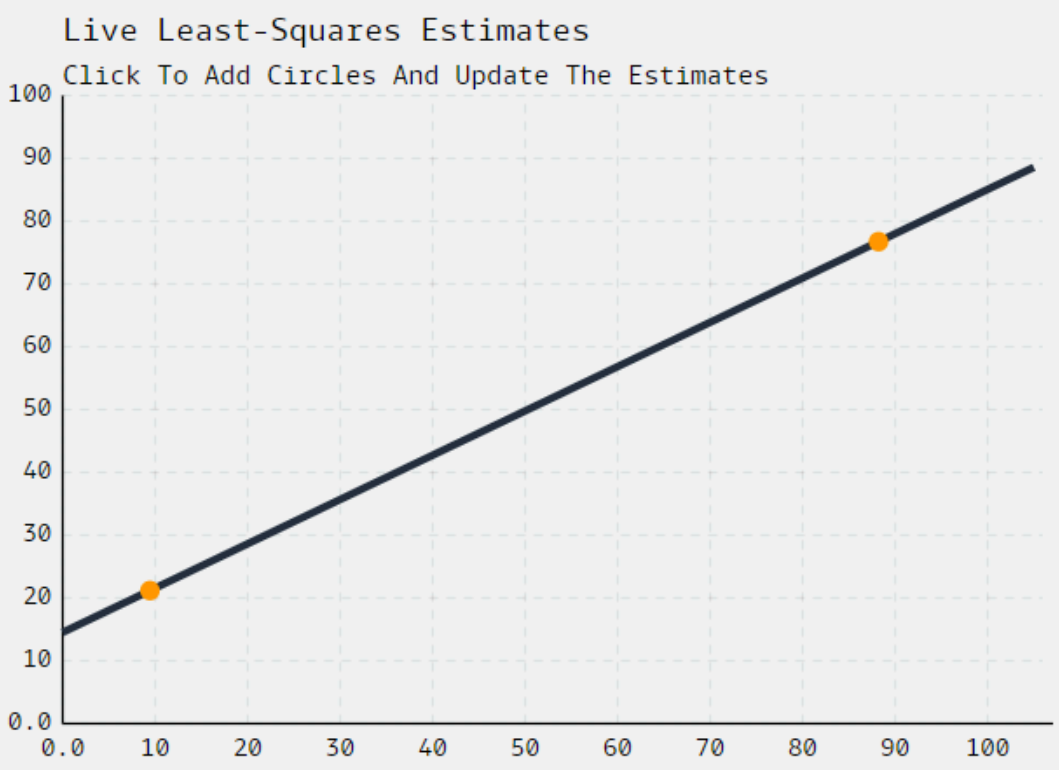
Add circles to the chart below to see how the Normal Equation calculates two featues, the bias and weight, for the corresponding regression model.

$$\hat{\beta} = (X^T X)^{-1} X^T Y$$

$$= \left(\begin{bmatrix} 1 & 9.4 \\ 1 & 88.2 \end{bmatrix}^T \begin{bmatrix} 1 & 9.4 \\ 1 & 88.2 \end{bmatrix}\right)^{-1} \begin{bmatrix} 1 & 9.4 \\ 1 & 88.2 \end{bmatrix}^T \begin{bmatrix} 21.2 \\ 76.7 \end{bmatrix}$$

$$= \left(\begin{bmatrix} 2 & 474 \\ 474 & 168116 \end{bmatrix}\right)^{-1} \begin{bmatrix} 1 & 9.4 \\ 1 & 88.2 \end{bmatrix}^T \begin{bmatrix} 21.2 \\ 76.7 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 14.53 \\ 1 & 0.71 \end{bmatrix}$$



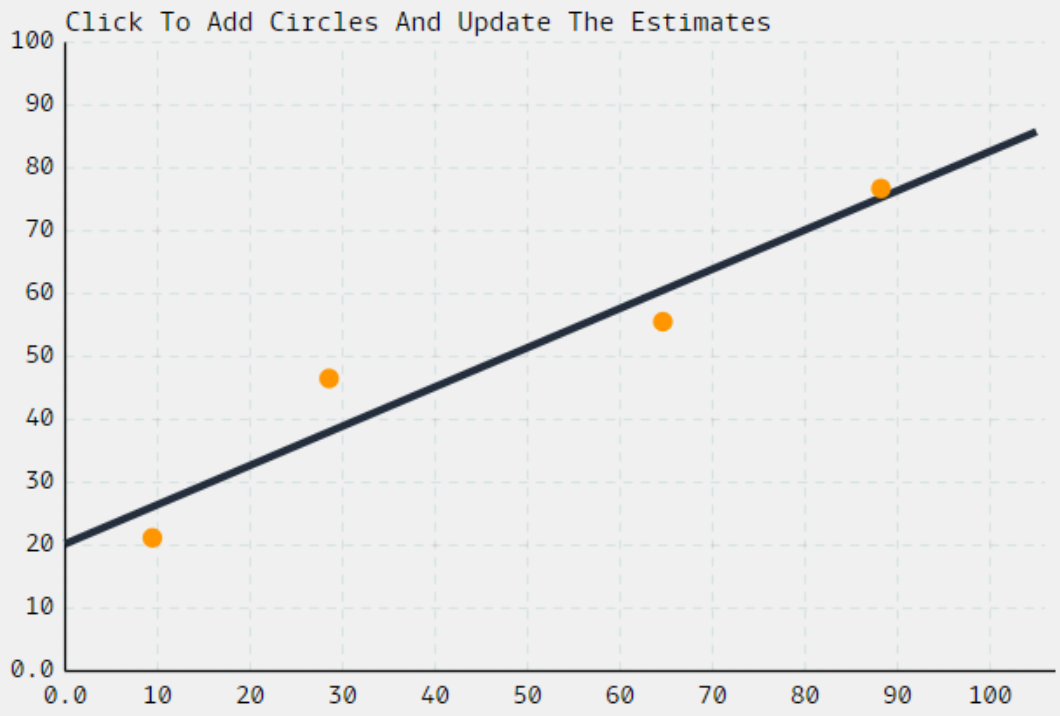

Reset

$$\hat{\beta} = (X^T X)^{-1} X^T Y$$

$$= \left(\begin{bmatrix} 1 & 9.4 \\ 1 & 88.2 \\ \vdots & \vdots \\ 1 & 64.6 \end{bmatrix}^T \begin{bmatrix} 1 & 9.4 \\ 1 & 88.2 \\ \vdots & \vdots \\ 1 & 64.6 \end{bmatrix}\right)^{-1} \begin{bmatrix} 1 & 9.4 \\ 1 & 88.2 \\ \vdots & \vdots \\ 1 & 64.6 \end{bmatrix}^T \begin{bmatrix} 21.2 \\ 76.7 \\ \vdots \\ 55.6 \end{bmatrix}$$

$$= \left(\begin{bmatrix} 4 & 929 \\ 929 & 283333 \end{bmatrix}\right)^{-1} \begin{bmatrix} 1 & 9.4 \\ 1 & 88.2 \\ \vdots & \vdots \\ 1 & 64.6 \end{bmatrix}^T \begin{bmatrix} 21.2 \\ 76.7 \\ \vdots \\ 55.6 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 20.22 \\ 1 & 0.62 \end{bmatrix}$$



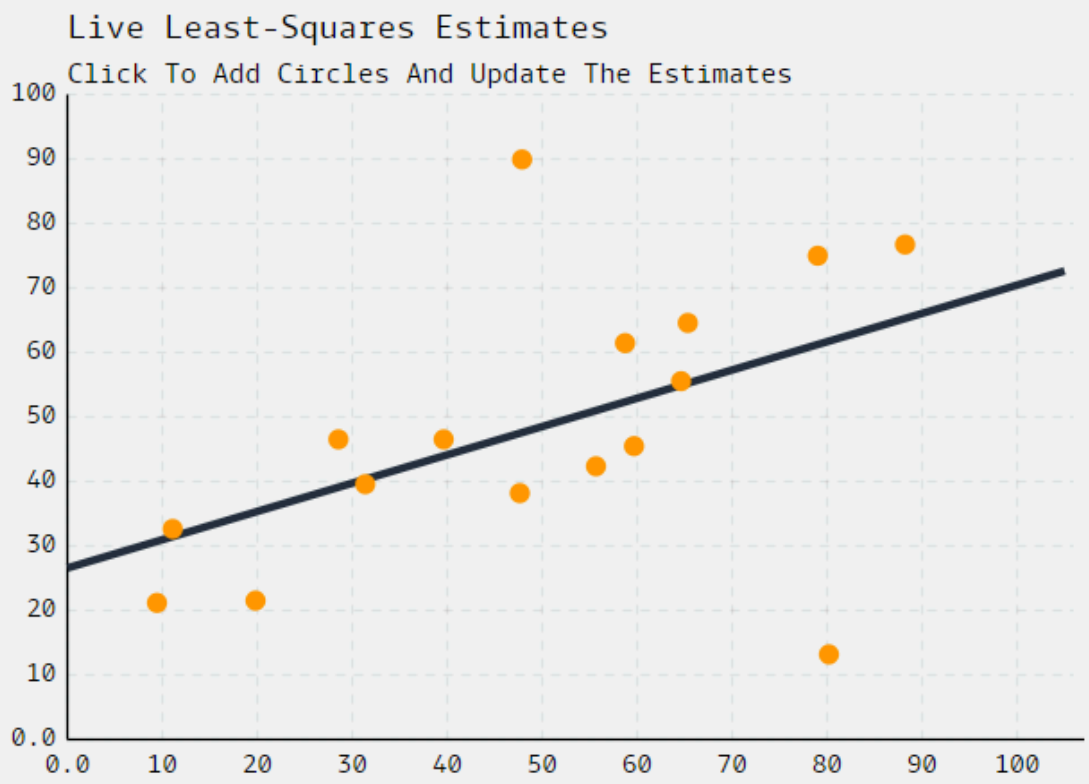

Reset

$$\hat{\beta} = (X^T X)^{-1} X^T Y$$

$$= \left( \begin{bmatrix} 1 & 9.4 \\ 1 & 88.2 \\ \vdots & \vdots \\ 1 & 80.2 \end{bmatrix}^T \begin{bmatrix} 1 & 9.4 \\ 1 & 88.2 \\ \vdots & \vdots \\ 1 & 80.2 \end{bmatrix} \right)^{-1} \begin{bmatrix} 1 & 9.4 \\ 1 & 88.2 \\ \vdots & \vdots \\ 1 & 80.2 \end{bmatrix}^T \begin{bmatrix} 21.2 \\ 76.7 \\ \vdots \\ 13.2 \end{bmatrix}$$

$$= (\begin{bmatrix} 16 & 3816 \\ 3816 & 1069204 \end{bmatrix})^{-1} \begin{bmatrix} 1 & 9.4 \\ 1 & 88.2 \\ \vdots & \vdots \\ 1 & 80.2 \end{bmatrix}^T \begin{bmatrix} 21.2 \\ 76.7 \\ \vdots \\ 13.2 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 26.59 \\ 1 & 0.44 \end{bmatrix}$$




Reset

Despite providing a convenient closed-form solution for finding our optimal coefficients, the Normal Equation estimates are often not used in practice, because of the computational complexity required to invert a matrix with too many features. While our two feature example above runs fast (we can run it in the browser!), most machine learning models are more complicated. For this reason, we often just use gradient descent.

**Are Our Coefficients Valid?**

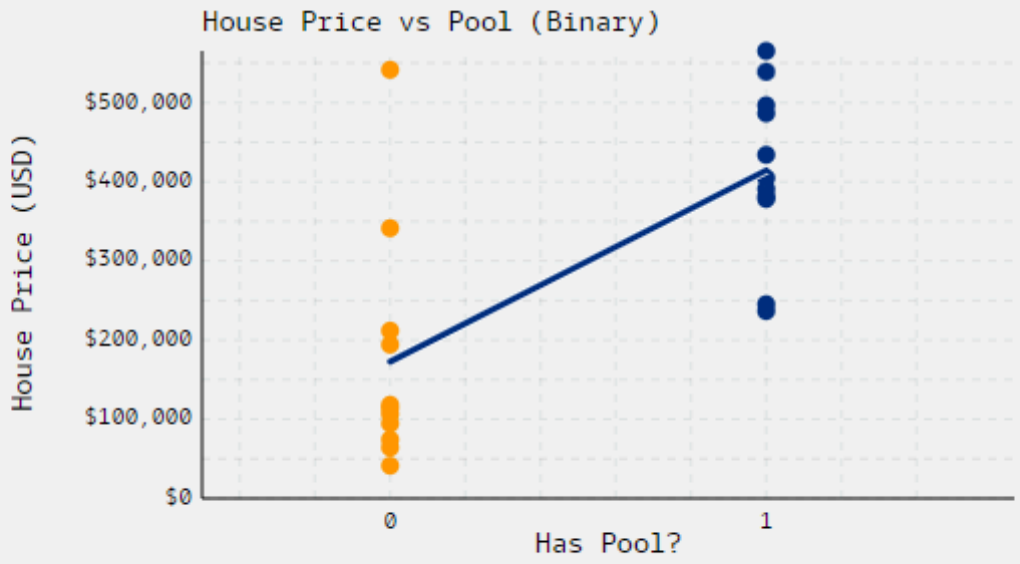
In research publications and statistical software, coefficients of regression models are often presented with associated p-values. These p-values come from traditional null hypothesis statistical tests: t-tests are used to measure whether a given cofficient is significantly different than zero (the null hypothesis that a particular coefficient $\beta_i$ equals zero), while F tests are used to measure whether *any* of the terms in a regression model are significantly different from zero. Different opinions exist on the utility of such tests (e.g. chapter 10.7 of [1] maintains they're not super important). We don't take a strong stance on this issue, but believe practitioners should always assess the standard error around any parameter estimates for themselves and present them in their research.

# Interpreting Regression Models

One of the most powerful aspects of regression models is their interpretability. However, different forms of regression models require different interpretations. To make this clear, we'll walk through several typical constructs of a regression model, and describe how to interpret each in turn. For all aforementioned models, we interpret the error term as irreducible noise not captured by our model.

A Binary Feature | A Continuous Feature | Multivariate Regression | Regression with Interaction

## A Regression Model With One Binary Feature




**Example:**

$$\text{house price} = 172893 + 241582 * pool$$

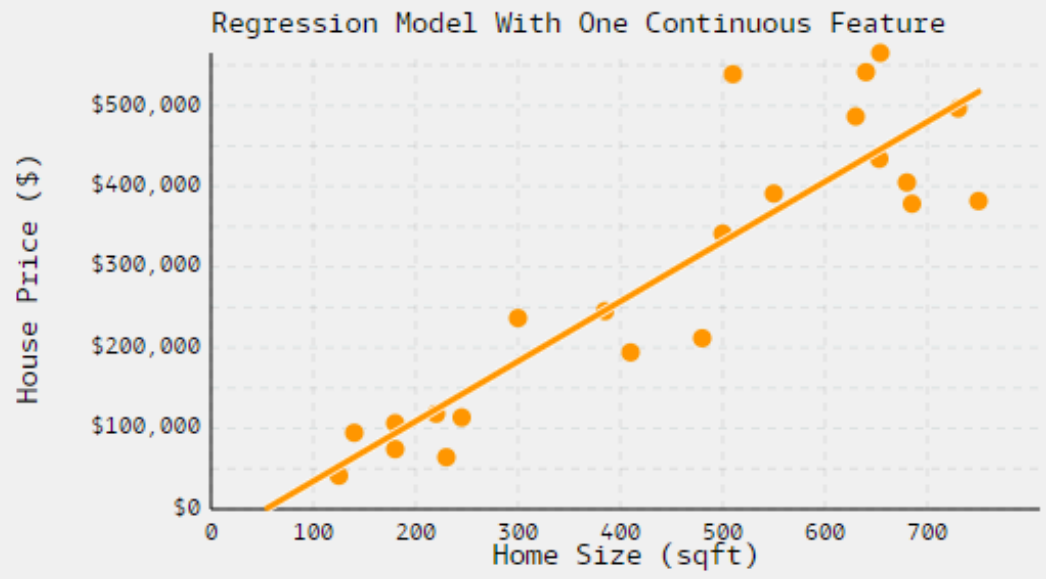
**Interpretation**: This model summarizes the difference in average housing prices between houses without swimming pools (●) and houses with swimming pools (●).

The intercept, $172,893, is the average predicted price for houses that do not have swimming pools (to see this, simply set $pool$ to 0 and solve the equation). To find the average price predicted price for houses with pools, we simply plug in $pool = 1$ to obtain $172,893 + $241,582 * 1 = $414,475.

The difference between these two subpopulation means is equal to the coefficient on $pool$. It tells us that houses with pools cost $241,582 more on average than houses that do not have pools.[i]

A Binary Feature | A Continuous Feature | Multivariate Regression | Regression with Interaction

## A Regression Model With One Continuous Feature




**Example:**

$$\text{house price} = -39591 + 742 * sqft$$

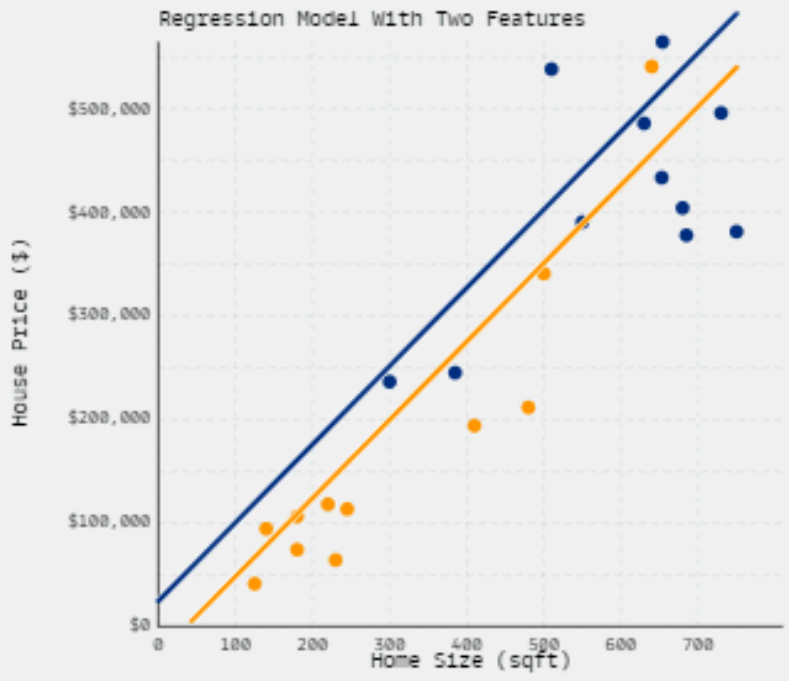
**Interpretation**: This model summarizes the average house prices across differently sized houses (●) as measured in square feet.

The coefficient, $742, represents the average difference in housing price for one-unit difference in the square-footage of the house. In other words, we expect each additional square-foot, on average, to raise the price of a house by $742.

The intercept, -$39,591, represents the predicted housing price for houses with $sqft = 0$, that is, it represents the average price of a zero square-foot house. Because this value doesn't make much intuitive sense, it's common for models to be transformed and standardized before carrying out a regression model.[i]

A Binary Feature | A Continuous Feature | Multivariate Regression | Regression with Interaction

## A Multivariate Regression Model




**Example:**
$\text{house price} = -27154 + 757 * sqft + 51867 * pool$

**Interpretation**: Typically, a regression model will contain more than one feature. We call this a *multivariate regression model*. In our example, we model home prices as a function of both the size of the house ($sqft$) and whether or not it has a pool ($pool$, where ● = no pool and ● = has a pool).

The intercept, -$27,154, represents the predicted average housing price for houses with all $x_i = 0$. For our model, it represents the cost of houses with no pools and a square-footage of zero.[i]

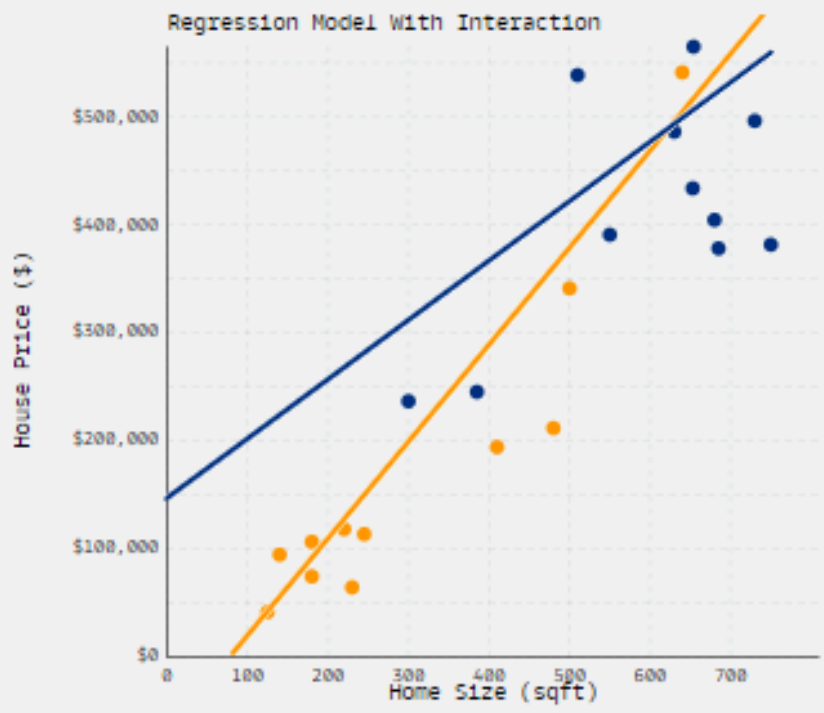
The coefficient of $pool$, $51,867, represents the average expected price difference in houses of the same size (in $sqft$) if they do or do not have a pool. In other words, we expect, on average, houses of the same size to cost $51,867 more if they have a pool than if they do not.

The coefficient of $sqft$, $757, represents the average expected price difference in housing price for houses that have the same value of $pool$ but differ in size by one square-foot.

Note that, regardless of whether or not a house has a pool (—) or not (—), we assume the *same* slope for $sqft$. To visualize this, we show two lines above. This isn't always a valid assumption to make, and when we think those subpopulations should have different slopes, we can use interaction terms.

A Binary Feature | A Continuous Feature | Multivariate Regression | Regression with Interaction

## A Regression Model With Interaction Terms




**Example:**
$\text{house price} = -70296 + 899 * sqft + 217111 * pool - 347 * (sqft : pool)$

**Interpretation**: If we believe that the slope for $sqft$ should differ between houses that do have pools and houses that do not, we can add an interaction term to our model, $(sqft : pool)$.

The coefficient of the interaction term $(sqft : pool)$, -$347, represents the difference in the slope for $sqft$, comparing houses that do and do not have pools. Visually, this represents the difference between the slopes of the two lines, — and —, above.

The intercept, -$70,296, represents the predicted housing price for houses with no pools and a square-footage of zero.[i]

The coefficient of $pool$, $217,111, represents the average expected difference in houses of the same size (0 $sqft$) that differed in whether or not they had a pool. (It's not super useful since we don't have houses with 0 square-feet).

The coefficient of $sqft$, $899, represents the average expected difference in housing price for houses that do not have a pool ($pool = 0$) but differ in size by one square-foot.

Of course, this is not an exhaustive list of regression models, many other forms exist!

# Regression Model Assumptions

When teaching regression models, it's common to mention the various assumptions underpinning linear regression. For completion, we'll list some of those assumptions here. However, in the context of machine learning we care most about if the predictions made from our model generalize well to unseen data. We'll use our model if it generalizes well even if it violates statistical assumptions. Still, no treatment of regression is complete without mentioning the assumptions.

- **Validity:** Does the data we're modeling matches to the problem we're actually trying to solve?
- **Representativeness:** Is the sample data used to train the regression model representative of the population to which it will be applied?
- **Additivity and Linearity:** The deterministic component of a regression model is a linear function of the separate predictors: $y = B_0 + B_1x_1 + ... + B_px_p$.
- **Independence of Errors:** The errors from our model are independent.
- **Homoscedasticity:** The errors from our model have equal variance.
- **Normality of Errors:** The errors from our model are normally distributed.

**When Assumptions Fail?**

What should we do if the assumptions for our regression model aren't met? Don't fret, it's not the end of the world! First, double-check that the assumptions even matter in the first place: if the predictions made from our model generalize well to unseen data, and our task is to create a model that generalizes well, then we're probably fine. If not, figure out which assumption is being violated, and how to address it! This will change depending on the assumption being violated, but in general, one can attempt to extend the model, accompany new data, transform the existing data, or some combination thereof. If a model transformation is unfit, perhaps the application (or research question) can be changed or restricted to better align with the data. In practice, some combination of the above will usually suffice.

# Dive Deep

The study of linear regression is a very deep topic: there's a ton of different things to talk about and we'd be foolish to try to cover them all in one single article. Some of those topics left unmentioned are: regularization methods, selection techniques, common regression transformations, bayesian formulations of regression, and additional evaluation techniques. For those interested in learning more, we recommend diving deep into the aforementioned topics, or reading the resources below. Hopefully this article serves as a nice starting point for learning about linear regression.

## References + Open Source

This article is a product of the following resources + the awesome people who made (and contributed to) them:


**[1] Regression And Other Stories**
(Gelman, Hill and Vehtari 2020).
**[2] Elements of Statistical Learning**
(John Ross Quinlan, 1986).
**[3] Mathematical Statistics and Data Analysis**
(John A. Rice, 2010).
**D3.js**
(Mike Bostock & Philippe Rivière)
**KaTeX**
(Emily Eisenberg & Sophie Alpert)
**Svelte**
(Rich Harris)