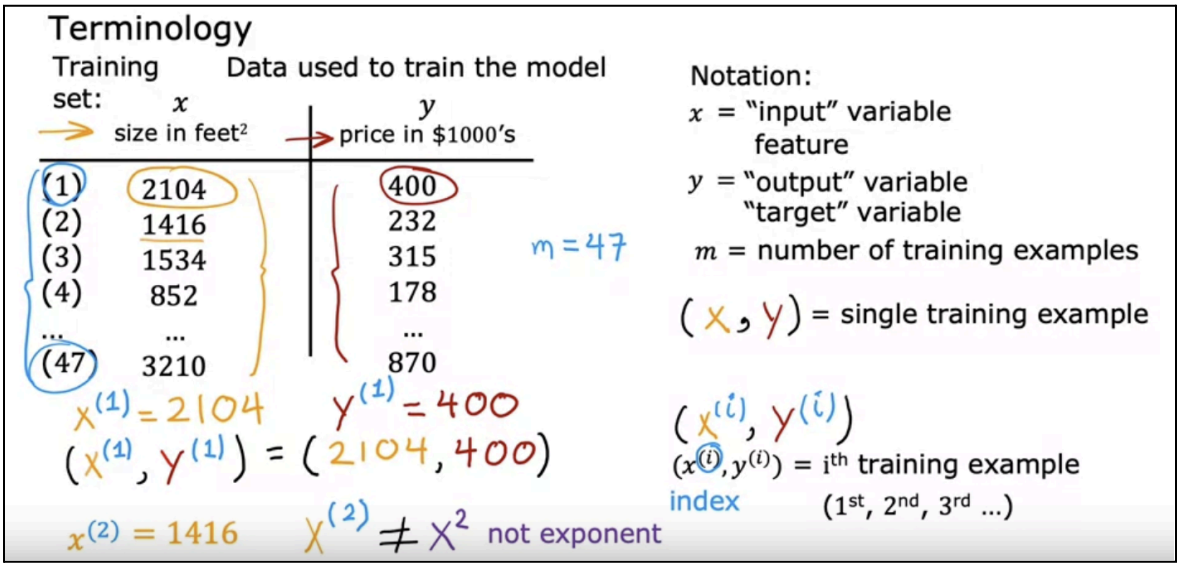
Resources List: **Coursera (Main)**, YouTube, Google Cloud, Medium, ChatGPT

Terminology:



Training set- Dataset used to train a model

Target variable- Output variable

1.Overview of ML

AI

An area of computer science, where the goal is to enable computers and machines to perform human-like tasks and simulate human behavior.

ML

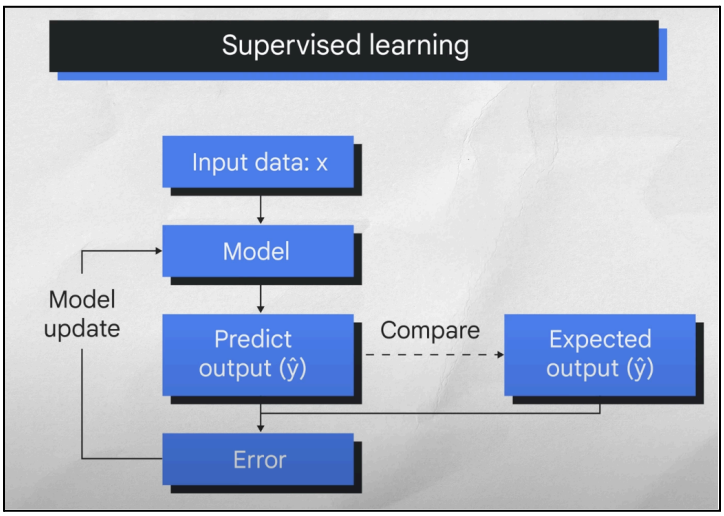
Machine learning is a subset of AI that tries to solve a specific problem and make predictions using data. ML gives a computer the ability to **learn** without **explicit** programming.

THREE SUBSETS OF ML Algorithms

Supervised Learning

- Supervised Learning uses **labeled** data (**input-output** pairs) to train models. The goal is for the model to learn a mapping from inputs to outputs so that it can make accurate predictions on new, unseen data. In supervised learning, each

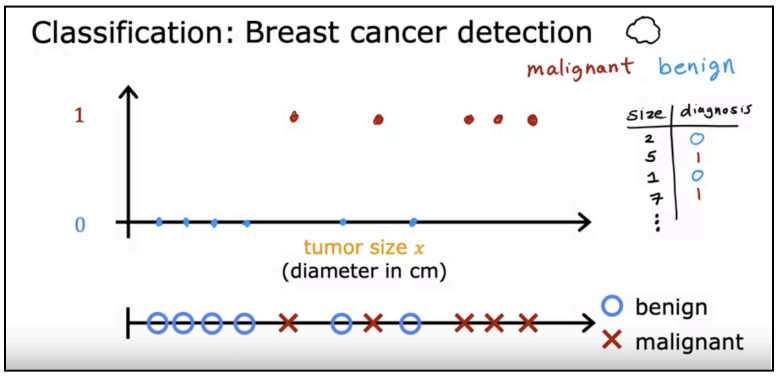
training example consists of an input and a corresponding output label, and the model learns by finding patterns and relationships between them. This approach enables the model to classify new inputs correctly in the future based on the patterns it has learned.



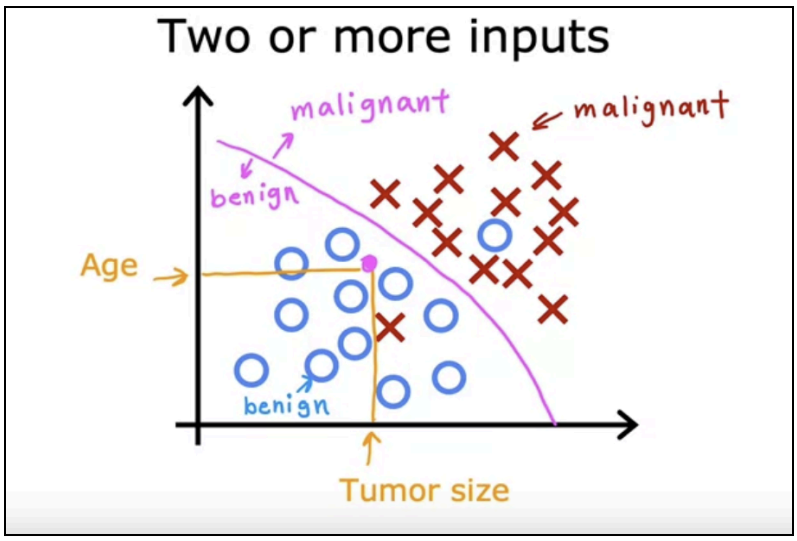
- **Loss (Cost) Function:** Most if not all Supervised learners generally use some form of a **loss function**. A loss function quantifies the difference between the expected output and the actual output (ground truth) of the model. It provides a **scalar value** representing this **difference**. The model tries to minimize this difference through the use of different **optimization techniques** (such as gradient descent) by **adjusting** the model's **parameters.**
  - **Loss Function Examples:** Mean Squared Error (Regression), Mean Absolute Error (Regression), Binary Cross-Entropy Loss (Classification), Categorical Cross-Entropy Loss (Classification), Hinge Loss (Classification)
- **Supervised Learner Examples:**
  - Regression: Predict a **number** from infinitely many possible outputs.
    - **Different types** of Regression Examples:
      - Linear Regression: Line of best fit.
      - Logistic Regression: Uses sigmoid function to produce probability values between 0 and 1.
  - Classification: Predicts categories (or classes; used interchangeably). It **does not** have to be a **number**. If it is a

number, then it has to be a **small/finite/limited** number of possible output categories such, for example, **0,1,2**.

- **Application** Example:



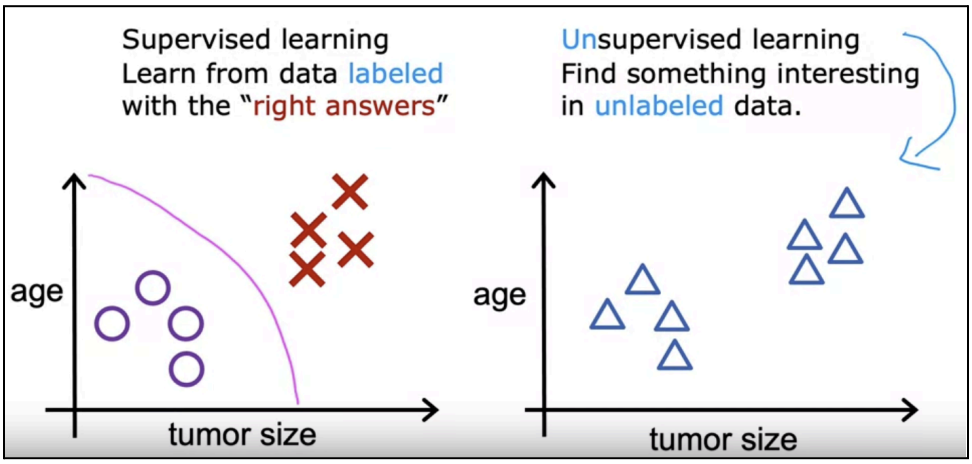
- Usually, many more dimensions/factors are considered when determining output. Here's an example of data with two inputs (Age, and Tumor size):



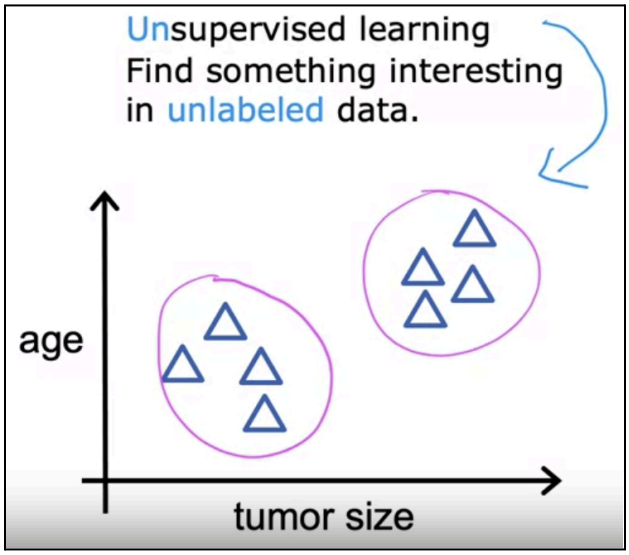
Unsupervised Learning

- Unsupervised Learning uses **unlabeled** data to learn about certain patterns in data. We don't have to supervise the algorithm to give a "correct answer". We ask the algorithm to find, all by itself, what's interesting or what patterns/structures there might be in the data. While for supervised learners, the data comes with an input label x and output label y, for unsupervised learners,

the data only comes with input label x, but not output label y.

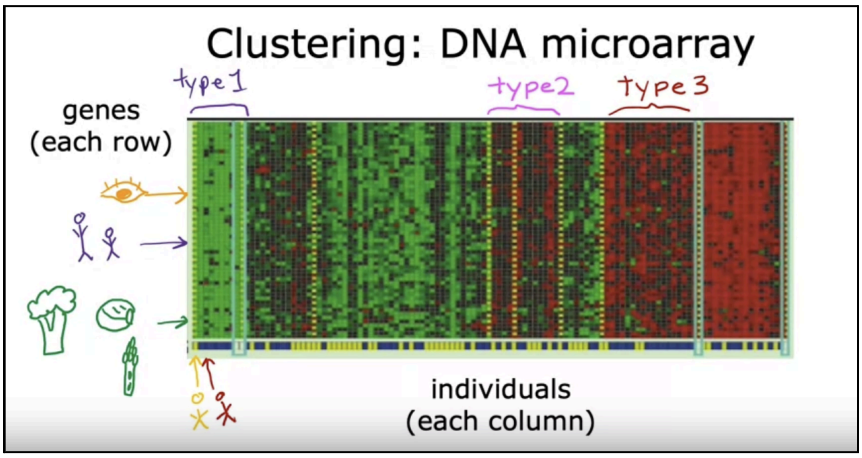


- **Unsupervised Learning Examples:**
  - Clustering: Places unlabeled data into different clusters.

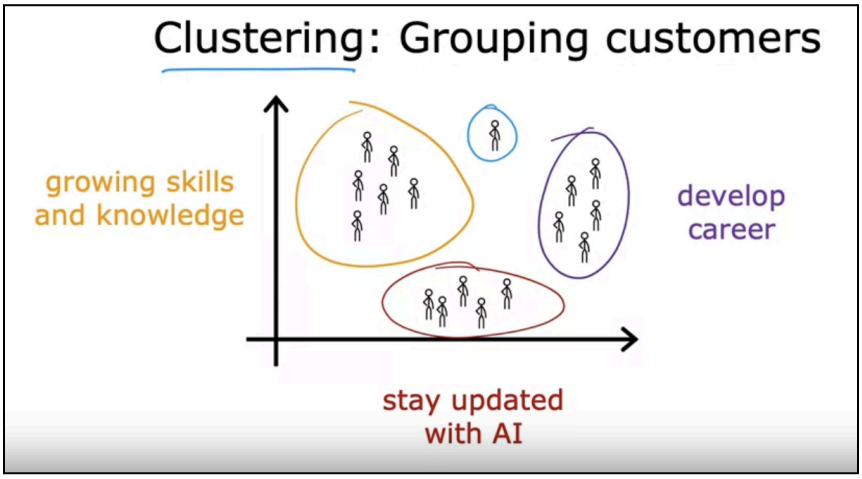


- Application Example **1**: Google News clusters similar news stories together.

- Application Example **2**: Grouping people together by genes.



- Application Example **3**: Grouping customers based on their reason for buying a product. (In this case, online classes)



  - Anomaly Detection: Find unusual data points.
  - Dimensionality Reduction: Compress data using fewer numbers. Allows you to take a very big dataset and magically compress it to a much smaller dataset using as little information as possible.

Reinforcement Learning

- Will touch more on this later.

TOOLS

Jupyter Notebook

Common web based machine learning code editor/interactive computing platform.

Python

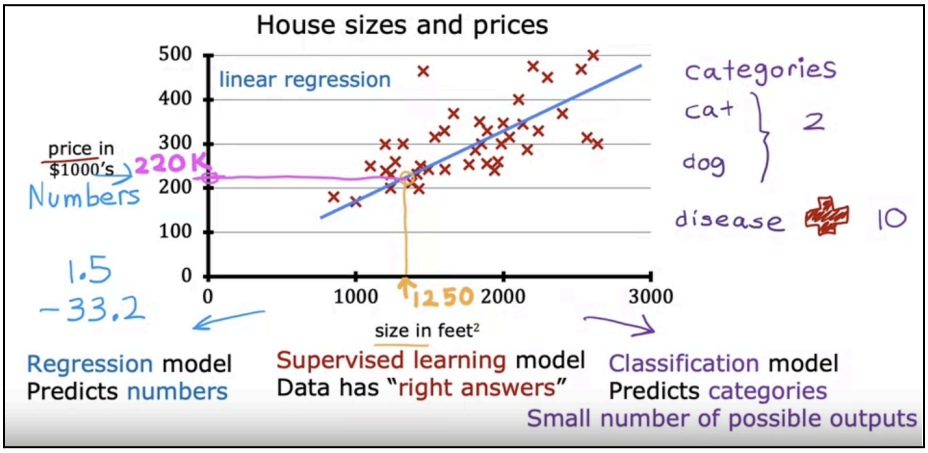
Programming Language.

2.Regression Model

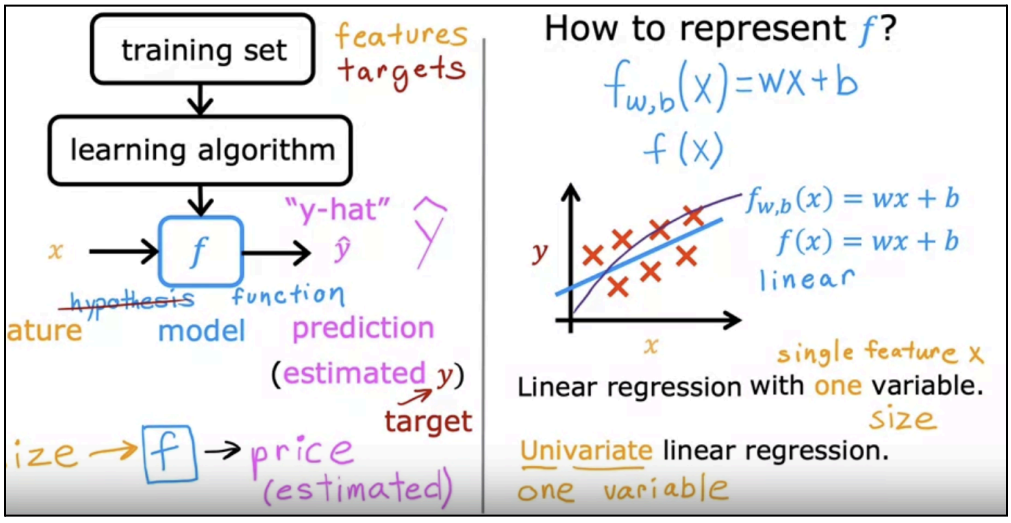
Linear Regression

- Many concepts from linear regression will be applicable to other machine learning models.

- Linear Regression Example: House price based on house size.

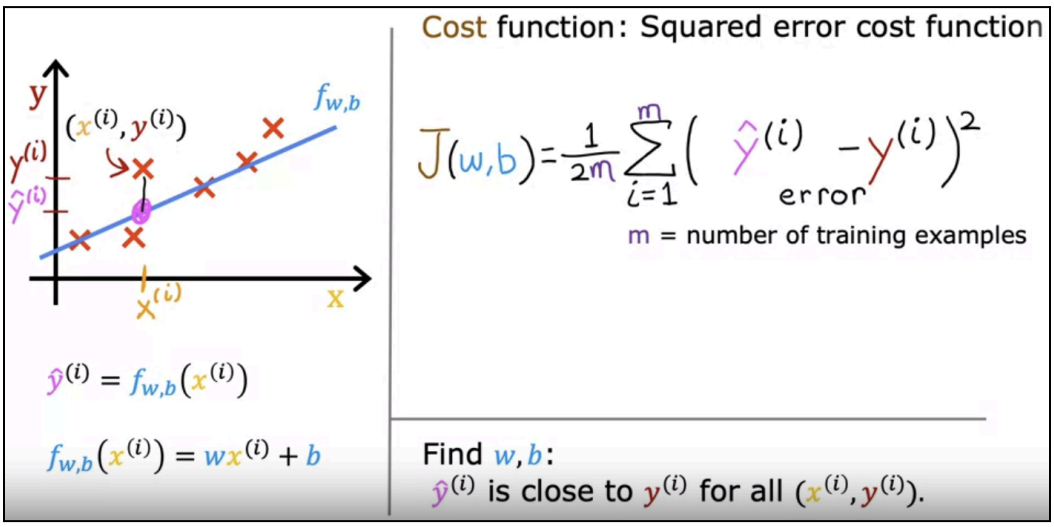


- Training Sets: Includes inputs (features), and output (targets) which the model will learn from.
- Training a Linear Regression Model (one variable) - General Overview:



  - Training is passed to the learning algorithm which then develops a function to best fit the input to output relationship.
  - Target (expected) output is denoted by y, and the actual output is denoted by y-hat.
  - For a linear regression model of one variable input (univariate), we use the common linear function y = mx+b formula we learned in elementary math, where w = weight and b = bias.
- **Loss/Cost Function:**

- Formula:

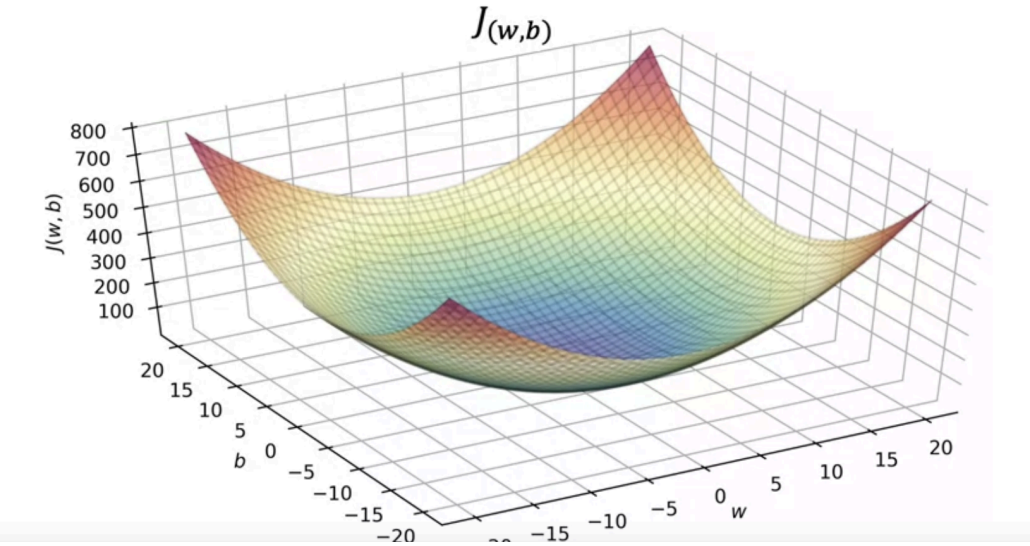


  - Squared error cost function: Most commonly used for almost all regression problems - not necessarily all linear regression problems. (y-hat minus y is just the **difference** between **expected value** and **actual value**.)

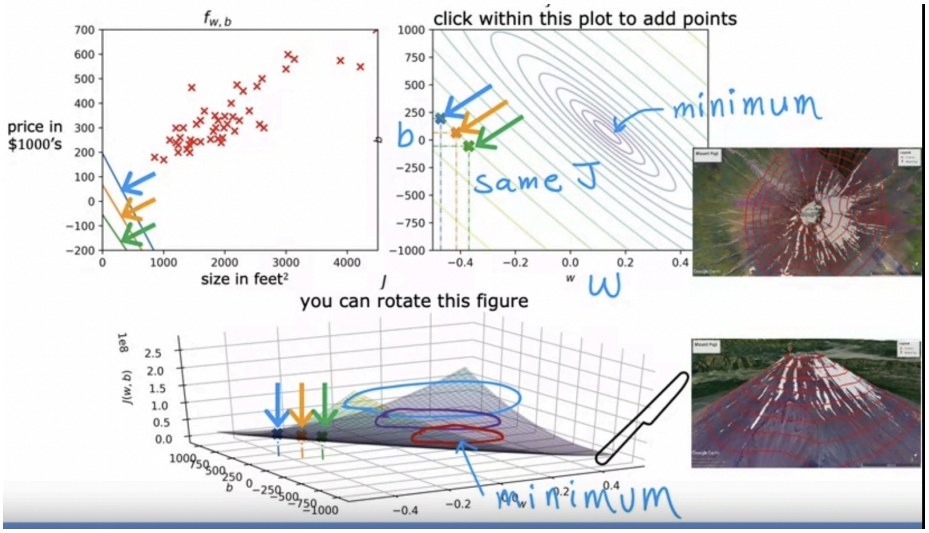
$$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}\left(f_{w,b}(x^{(i)}) - y^{(i)}\right)^2$$

- **Cost Function Intuition:** When the value of function J is at the minimum, the expected value is close to the actual value. This makes sense as function J represents the error. We can see this by getting rid of the bias value and graphing the error function (which, in this case, was a parabola) along with the original regression model function. In this case, the error function was at a minimum (vertex) when the original regression model function had a weight (w or slope) of one which best fit the data set or train set.

- **Visualizing the Cost Function:**



  - Now considering the bias, the Cost Function becomes a function of **two parameters**, making it **three** dimensional.

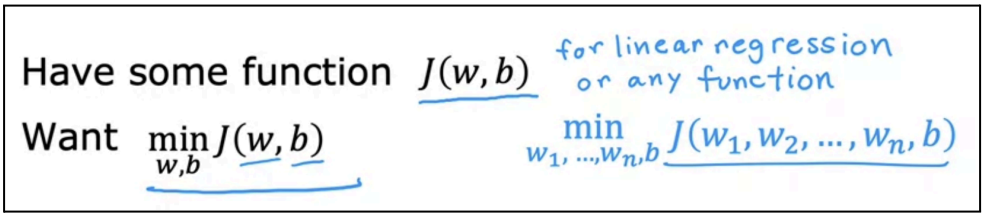


    - Now, we can use a contour plot to analyze a three dimensional graph where each contour represents the same J value for different values of w and b. (Same amount of error can occur with different biases and weights).

3.Train the Model with Gradient Descent

Gradient Descent

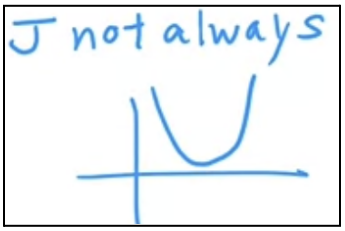
Gradient Descent is an algorithm you can use to minimize any function. In our case, for now, we would of course want to minimize the cost function J(w, b). (Note: It can be applied to minimize cost functions with many weight parameters such as the picture shown below)



- So, how do we apply it?

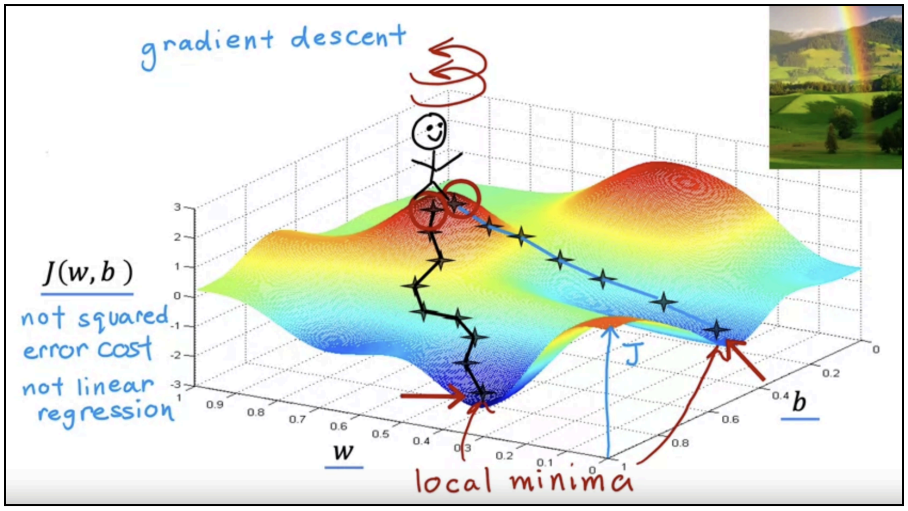
1. We choose and start with some w and b. A common choice is to set them both to 0.
2. Keep changing w and b to reduce the cost, J(w,b).
3. Until... we settle at or near a minimum.

Note: It is important to note that **J** is **not always** going to be parabolic, meaning that it can have multiple minimums.



Example:

Let's take this error function for example:



First, note that the graph of J (error function) is **no longer a bowl/hammock shape**. This is **because only** the **squared error cost** function produces such a graphical graph.

The gradient descent **algorithm** works by starting at some point on the graph (let's say where the stickman is standing) and **looking around** to decide in which direction their next step should be in (steepest descent) for it to reach the minimum the most **efficiently**. It takes a little step in the direction of steepest descent, then stops to look around again. It repeats this process again until it reaches a local minimum (which is where the error is relatively low or maybe even 0).

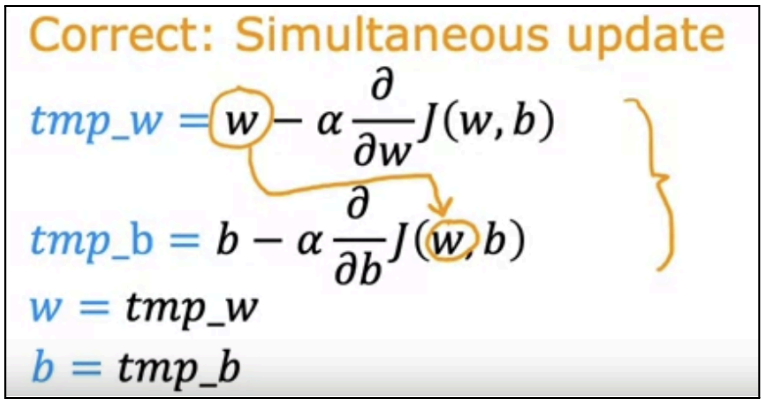
Gradient Descent Algorithm

$$w := w - \alpha \frac{d}{dw} J(w,b)$$

Above is the algorithmic formula for gradient descent with respect to weight, w.
Weight = Weight - alpha*(the gradient of the error function with respect to the weight)
Let's break this down:
Alpha is the learning rate - usually a small positive number between 0 and 1. Alpha determines and controls how big of a step you take downhill. If Alpha is very large, then that corresponds to a very aggressive gradient descent procedure where you're trying to take huge steps downhill. If Alpha is very small, then you'd be taking small baby steps downhill. There are ways to choose a good alpha -> more on this later.
d/dw(J(w,b)) is the derivative (gradient) which tells us which direction we want to take the step in.

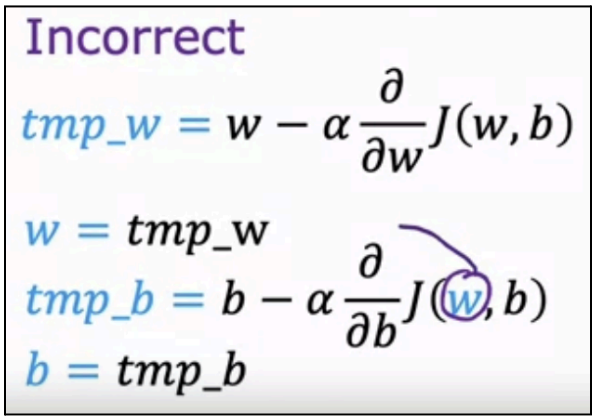
$$b = b - \alpha \frac{d}{db} J(w,b)$$

Above is the algorithmic formula for gradient descent with respect to bias, b.
Same logic applies except that the derivative is now with respect to b.
So, the idea of Gradient Descent:
We are basically **repeatedly updating** the weight (**w**) and bias (**b**) until **convergence** is reached (local minimum is reached and w,b no longer change much with each additional step that I take).
Important detail:
For Gradient Descent, we want to **simultaneously** update w and b.
Here is the **correct** way to **simultaneously** update:



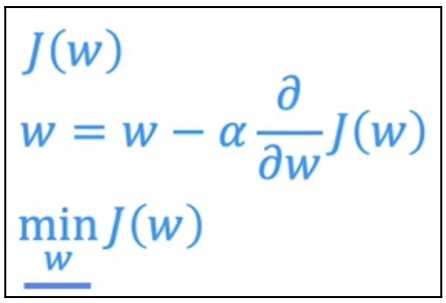
Notice how the w value we use to calculate tmp_b (which later gets assigned to b), uses the non-updated value of w.

Here is the incorrect way which is **NOT simultaneous**:

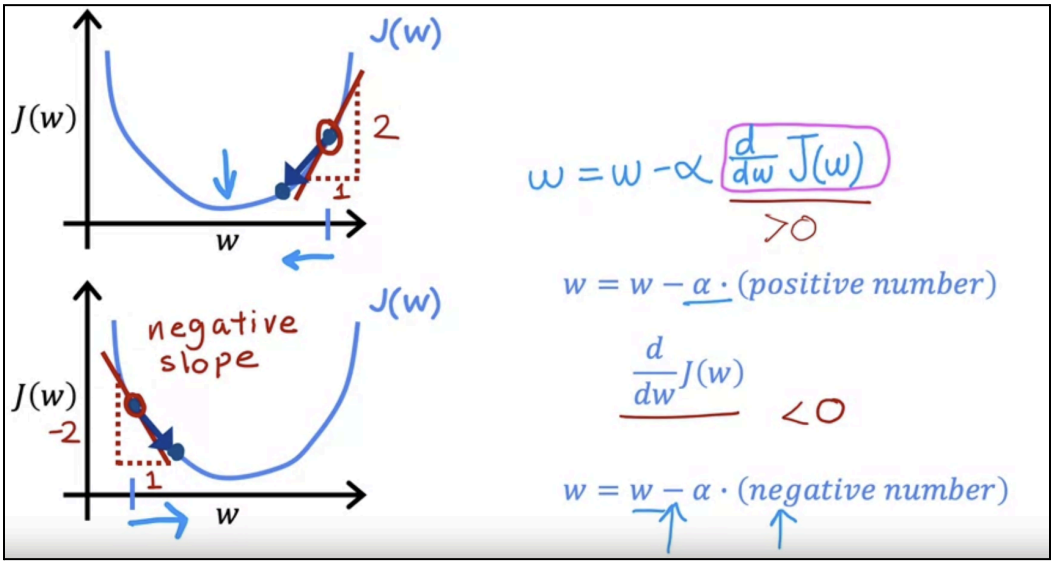


Here, we can see that the updated value of w is what's being used to update the value of b. This is NOT simultaneous. (Might still work - but stick to simultaneous updates!)

- **Gradient Descent Intuition:**
    - Let's simplify the original equation to get a better grasp of gradient descent:



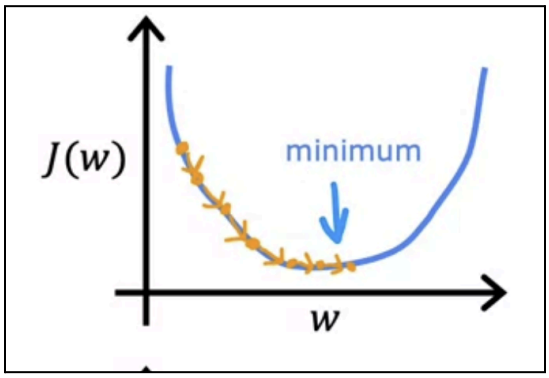
- Looking at the graphs now:



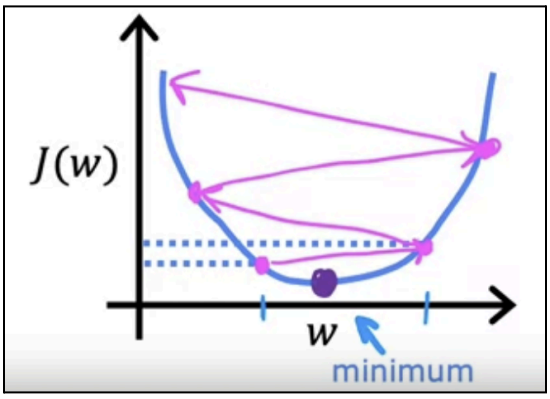
  - The upper graph shows gradient descent being applied to w when the starting point is set to the side of the graph with the positive slope. The lower graph shows gradient descent being applied to w when the starting point is set to the side of the graph with the negative slope.
    - From this example, we can see that if we start from where the slope is positive (starting from the right side of the graph), the derivative term becomes positive, making alpha*derivative positive. This causes w to decrease. Looking at the new point, after gradient descent has been applied, it has moved closer to where the error function evaluates to 0.
    - If we start from where the slope is negative (starting from the left side of the graph), the derivative term becomes negative, making alpha*derivative negative. This causes the updated w to increase. Looking at the new point, after gradient descent has been applied, it has moved closer to where the error function evaluates to 0.
    - Assuming we move enough times, we will eventually get really close to when the error function evaluates to 0. Then what? Well, as we get closer to when that happens, the

derivative also goes to 0. This means that the value of w will converge.

- **Learning Rate:** It is important to choose the learning rate appropriately. Let's look at two cases to better understand how we can choose the learning rate:
    - When the learning rate is very small - The process of gradient descent becomes **slow** as changes in w are extremely miniscule.

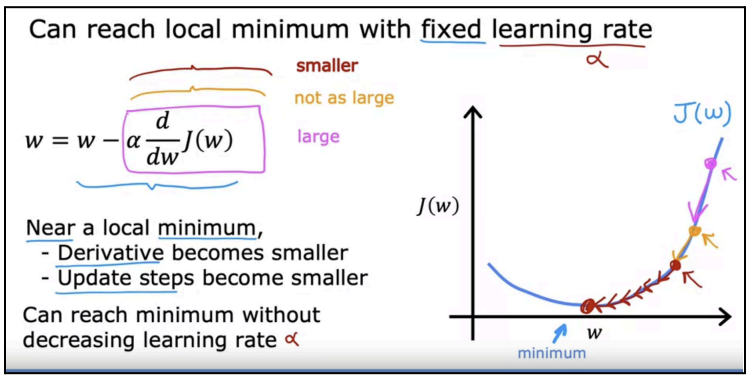


    - When the learning rate is very large - We tend to overshoot our next value for w. Let's look at the graph of when this happens:

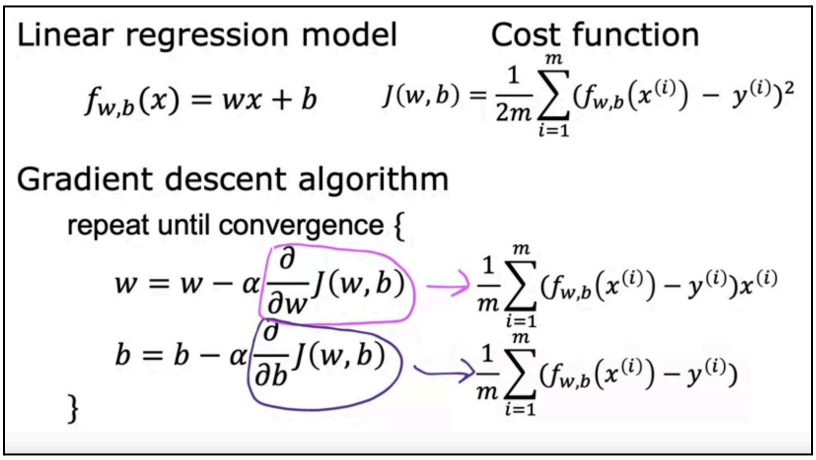


        - As you can see, we start from the point closest to w on the left. Because the **learning rate** is **very large**, after multiplying the derivative by the learning rate, we end up way over to the right of our desired value (J(w)=0). We apply gradient descent again, but this time, we end up taking an even bigger step to the left because the **magnitude** of the **slope** also **increased**. This causes us to overshoot much more this time. This cycle continues. (Fails to converge -> may even diverge).
    - As stated before, the term: learning rate * the derivative of the loss function decreases (because the derivative decreases) as it gets closer to

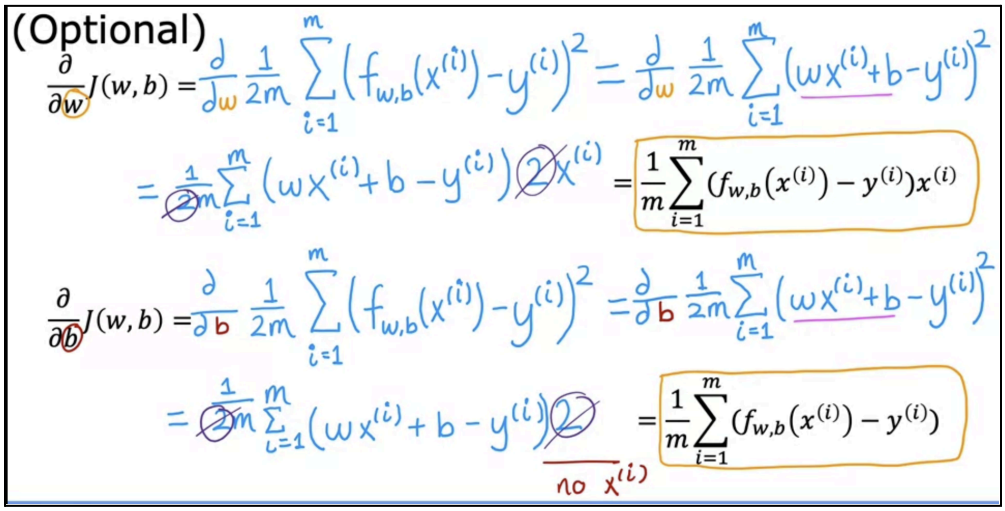
the local minimum, so we can reach it even with a fixed learning rate.



- Gradient Descent for Linear Regression:

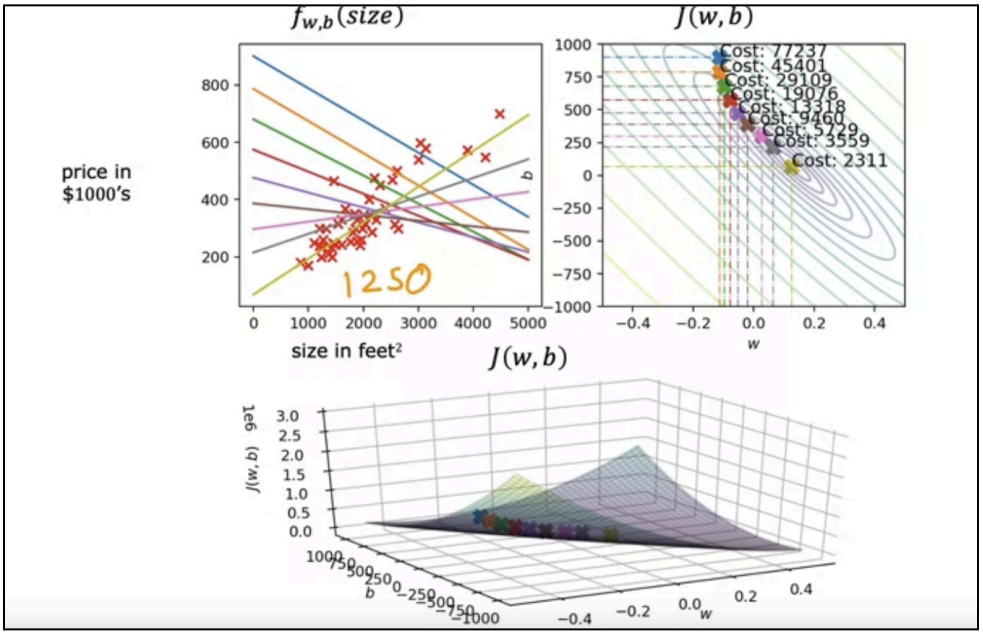


  - Above, we are given the formula for gradient descent on a linear regression model. The partial derivatives are calculated by using simple calculus on the loss function. **Here is how they are derived:**



- For applying gradient descent on linear regression, we use the **squared error function.** This function has a global minimum (only one minimum), meaning that it will always converge to this point.

- Running Gradient Descent:
  - As we can see below, running gradient descent on the error function also changes the graph of our original linear regression function so that it more accurately represents/predicts our actual data.



  - This was calculated using the Batch gradient descent technique where the error function is calculated using all of the training examples at once: