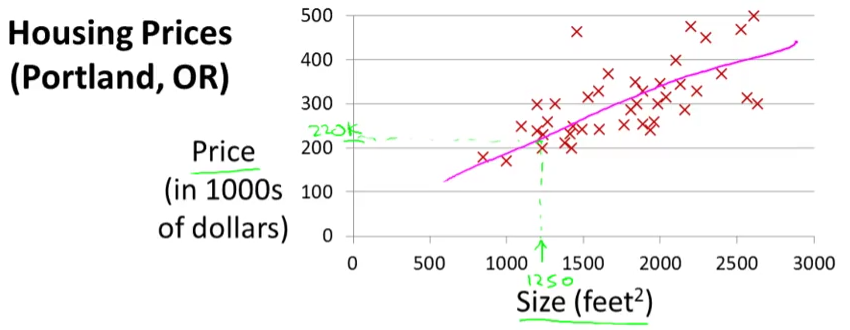
***LINEAR REGRESSION + MODEL + COST FUNCTION***

**I. MODEL REPRESENTATION**

* Ex: Predicting housing prices w/ a data set of housing prices from Portland, OR w/ different sizes sold for a range of different prices.
* Someone is trying to sell a house of 1250 ft2 + wants to predict how much they might be able to sell for.
* 1 thing to do is fit a model, like a straight best-fit line



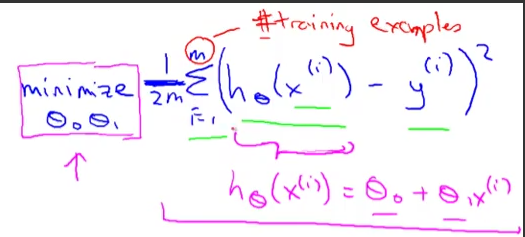
* Based on this, maybe he can sell the house for around $220k
* This was a supervised learning algorithm, b/c we're given the "right answer" for each example.
* Told the actual size + price of each of house in the data + used **regression** = predicting *a real-valued output* (price).
* **classification** = predict discrete-valued outputs (if tumors are malignant/benign (0-1 valued discrete output)
* In supervised learning, we have a **training set**
* Ex: Different housing prices w/ the job to learn from data how to predict prices of houses
* **m** = sample size/obs **x** = input variables/**features**  **y** = output/**target** variable to predict
* **(x, y)** = a single training example/observation/row in a table
* **(x(i), y(i))** = the ith training example
* We feed a training set to a ML algorithm whose job it to then output a function, which by convention, is usually denoted **h** for hypothesis



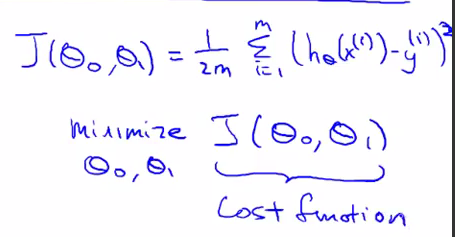
* The job of the hypothesis function, h, is to take in the value of x + try to output the estimated value of y for the corresponding x 🡪 *h is a function that maps x's to y's.*
* When designing a ML algorithm, the next thing to decide is how to *represent* this hypothesis h.
* This means we’re going to predict that y is a *linear function* of x = predict that y is some straight line function of x.
* This model = **linear regression** **w/ 1 variable** (x) = predicting all prices (y) as functions of 1 variable (x) = **univariate linear regression**.
* ***NOTE:*** Sometimes we'll want to fit a more complicated, perhaps *non-linear* function

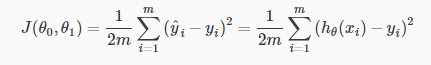
**II. COST FUNCTION**

* **The cost function** lets us figure out how to fit the best possible straight line to data.
* In linear regression, we have a training set + a hypothesis used to make predictions via a linear function
* **Parameters** of a model = intercept (θ0) and **regression coefficient** (θ1 = slope)
* W/ different choices of the parameters θ0 + θ1, we get different hypothesis functions for results
* In linear regression, we have a training set + we want to come up w/ values for the parameters so that the straight line we get corresponds to a straight line that somehow fits the data well
* The idea is to choose parameters so that h(x) (the y value predicted on input x) is *at least* *close* to the *actual* values y for the examples in the training set
* In our training set, we've given a number of x’s (house size) + we know the actual price (y) it was sold for
* We try to choose values for the parameters so that, at least in the training set, given our x’s, we make reasonably accurate predictions for y
* Linear regression is, overall, a **minimization problem 🡪** minimize the squared difference between the output of the hypothesis, h(x), and the actual price of a house, y (**sum of squared errors**)
* Want to sum the square difference between (h(x(i)) and y(i)) over a training set
* Then we halve m (sample size) + minimize the overall result

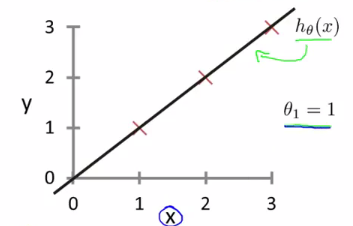


* “current values of θ0 + θ1 minimize the SSE”
* So we want to find the values of θ0 + θ1 so that expression on the right is minimized, giving **the overall objective function** for linear regression (in pink)
* find values of θ0 + θ1 so that (1/2m)\*SSE is minimized
* Define **a cost function (square error function or SSE cost function)**

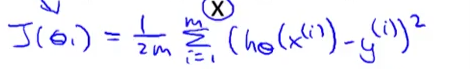




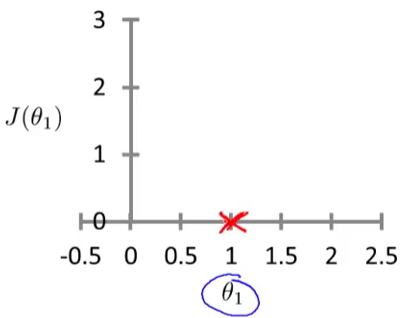
* The *squared* error cost function is a reasonable choice + works well for most regression problems.
* There are other cost functions that work pretty well, but the squared error cost function is probably the most commonly used one for regression problems.
* To summarize:
* **Hypothesis = h(x) = θ0 + θ1\*x**
* **Parameters = θ0 and θ1**
* **Cost Function = J(θ0,θ1) = 1/2m\*sum(from i to m) of (h(x(i)) – y(i))^2**
* **Goal = minimize J(θ0,θ1) for values of (θ0,θ1)**
* So the 2 key functions we want to understand are the **hypothesis** **function** + a **cost** **function**.
* *Hypotheses (LR) function hθ(x) is a function of x for a fixed θ1*
* *Cost function J(θ1) is a function of the parameter θ1*
* Simplified:
* hθ(x) = θ1\*x where (θ0 = 0)
* J(θ1) = (1/2m)\*sum(from i = 1 to m) (hθ(x(i)) – y(i))^2
* Minimize J(θ1) for value of θ1
* The hypothesis, for a fixed value of θ1, is a function of x (size of a house).
* In contrast, the cost function, **J**, is a function of the parameter, θ1, which controls the *slope* of the straight line. See simple cost function (θ0 = 0) below:



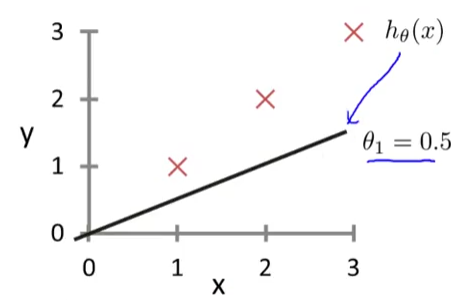
* W/ this hθ(x), we have 3 + θ1 = 1
* Now to figure out what is J(θ1) when θ1 = 1 by computing the cost function for θ1 = 1.



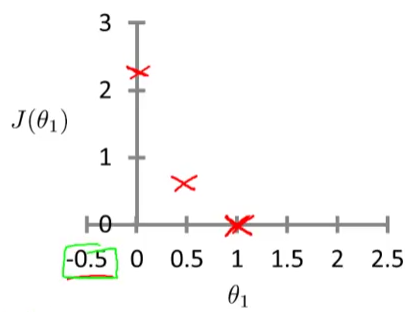
* So it turns out to be the sum of (1-1)^2 + (2-2)^2 + (3-3)^2 = 0 b/c if θ1 = 1, hθ(x(i)) = y(i) exactly
* So, we know now that J(θ1) = 0



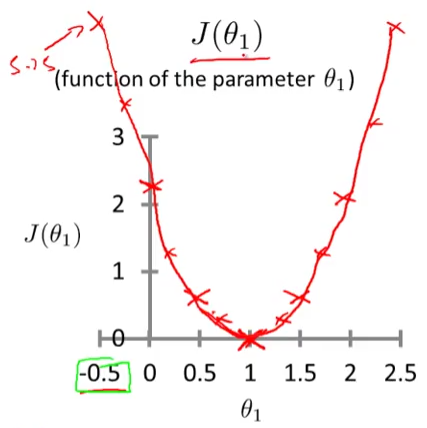
* θ1 can take on a range of different values 🡪 negative values, 0, positive values.
* So if θ1 = 0.5 🡪 slope = 0.5



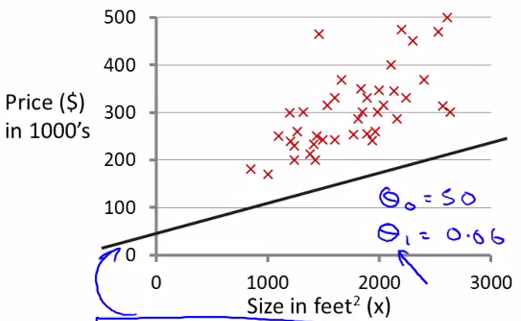
* J(θ1) = **J(0.5)** = (1/2m) \* (0.5 – 1)^2 + (1 – 2)^2 + (1.5 – 3)^2 = (1/(2\*3))\*3.5) = 3.5/6 = 0.58
* J(θ1) = **J(0)** = 1/2m \* (0 – 1)^2 + (0 – 2)^2 + (0 – 3)^2 = (1/(2\*3))\*14) = 14/6 = 2.3



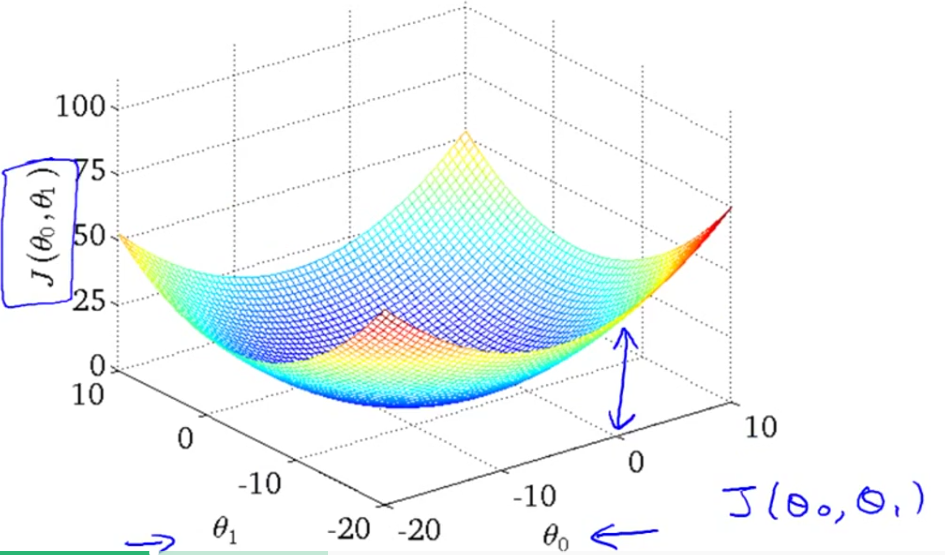
* It turns out you can have negative values of θ1 as well, then hθ(x) = a hypothesis w/ a *negative* slope
* You can actually keep on computing these errors, and θ1 = -0.5 turns out to have really high error, ~5.25
* By computing this range of values, we can slowly create plot out J(θ1)



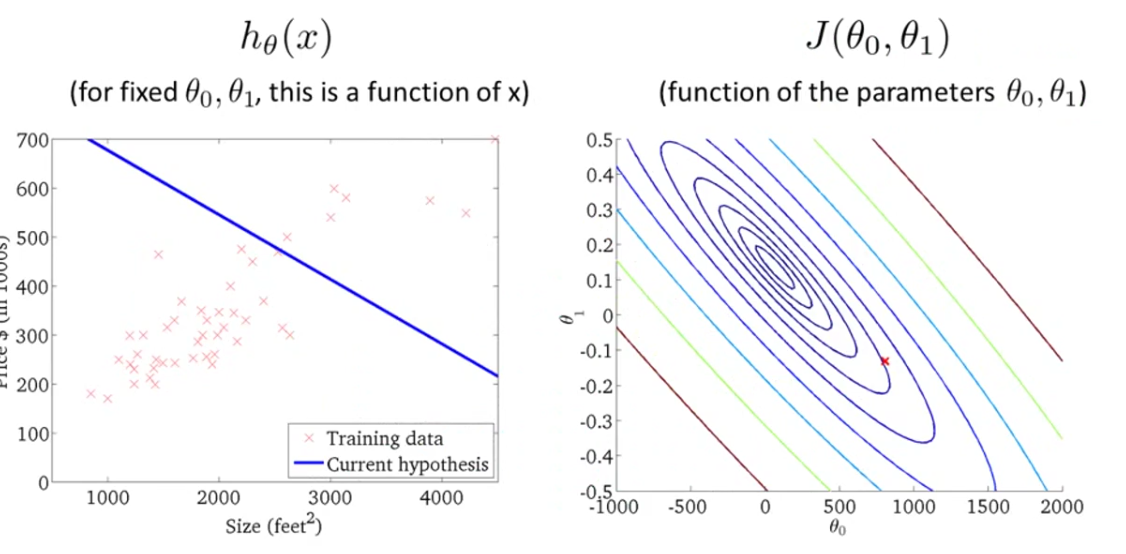
* To recap, each value of θ1 corresponds to a different hθ(x) = to a different straight line fit
* For each value of θ1, we wind up w/ a different value of J(θ1) + used it to trace out a plot of J(θ1)
* Remember, the *optimization objective* for our ML algorithm is to *choose the value of θ1 that minimizes J(θ1)*
* Looking at this curve, the value that minimizes J(θ1) is 1, which is indeed the best possible straight line fit through our data, just for this particular training set
* Set θ0 = 50 and θ1 = 0.06 to end up w/ hθ(x) = 50 + 0.06x



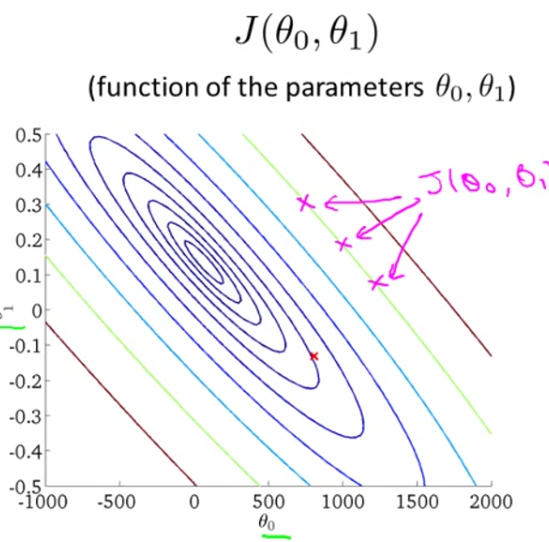
* Given these value of the parameters, we want to plot the corresponding cost, but now w/ TWO parameters, the plot gets a little more complicated
* It still has a similar bow shape, and, in fact, depending on the training set, you might get a cost function that looks like a 3D surface plot:
* As you vary the 2 parameters, you get different values of the cost function J(θ0,θ1) = the height of the surface above a particular point of {θ0, θ1} (the x + y-axis values)
* i.e. the height of the surface of the points indicates the value of J for values of θ0 and θ1.



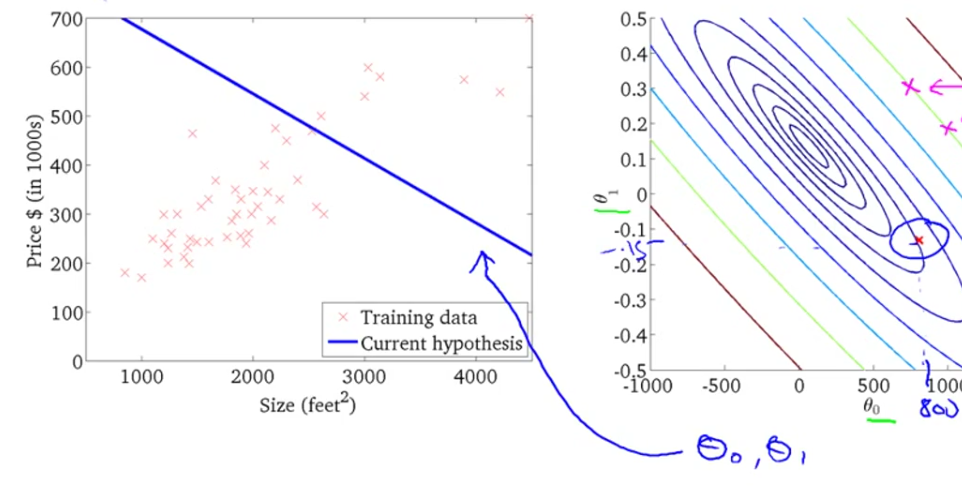
* **Contour Figures/Plots** w/ axis of θ0 and θ1



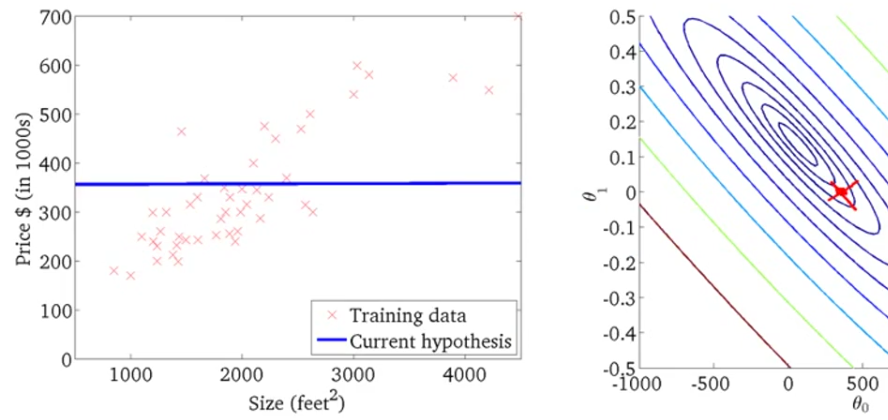
* Each of ellipse shows a set of points that takes on the SAME value for J(θ0,θ1).
* So, take 3 random points on the magenta ellipse 🡪 they ALL have the same value for J(θ0,θ1)



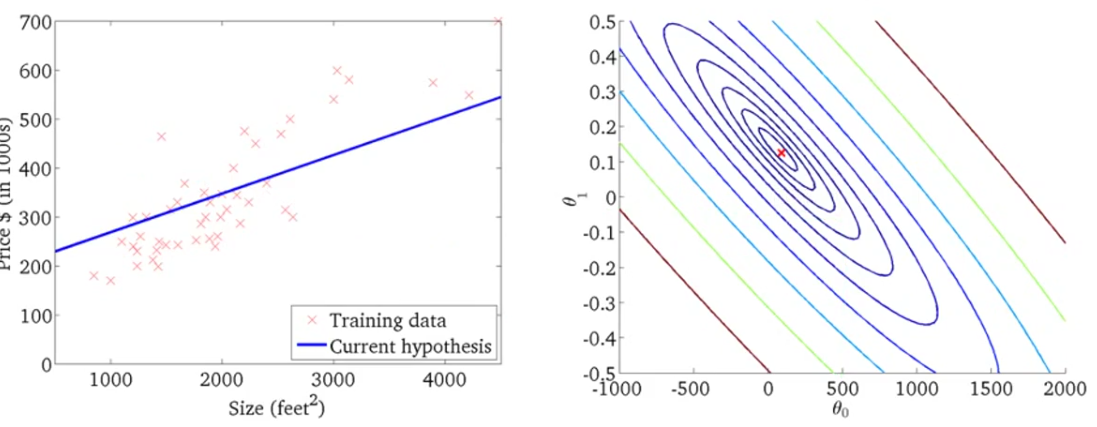
* A contour figures is a more convenient way to visualize J(θ0,θ1)
* Try to imagine this as a bow-shaped function coming out of the screen, where the minimum value is the center of the smallest of the concentric ellipses
* Each ellipse is the same height from that red x (above the minimum)
* Ex: θ0 = 800, θ1 = -0.15
* The point (red x) on the right corresponds to the hθ(x) value on the left
* hθ(x) intersects y-axis at ~800 w/ a slope of ~-0.



* This line is really not such a good fit to the data, + you find that it's cost (red x) is a value that's pretty far from the minimum (middle of smallest ellipse in the middle)
* So this is a pretty high cost b/c its NOT a good fit to the data.



* This is a different hypothesis that's still not a great fit for the data but may be slightly better w/ θ0 = 360 + θ1 = zero 🡪 h(x) = 360 + 0\*x
* It’s cost the height of J(θ0,θ1) for those points {θ0,θ1} (from the minimum value)



* This is actually not *quite* at the minimum, but is pretty close, so this is not such a bad fit to the data
* The SEE for this hθ(x) error is pretty close to the minimum
* What we really want is an efficient algorithm/piece of software for *automatically* finding the values of θ0 + θ1 that minimizes the cost function