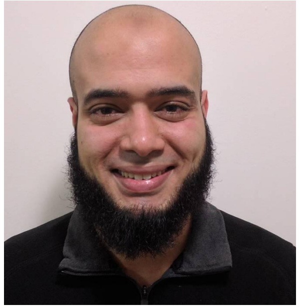
Machine Learning

Linear Regression Intuition

Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / MSc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



Recall

- **Supervised learning**
  - We are given both the input (X) and its output (Y)
  - We want to be able to map X to Y (**predict** [توقع] Y given the input X)
- **Regression**
  - The Y that we predict is a **real**-valued output (e.g. 0.7)
  - For example, we can predict the price of a property based on the features and/or location
  - Forget about the English meaning of regression. *There is some history for it*.
- **Classification**
  - The Y that we predict is a **discrete**-valued output (e.g. 3)
  - Example: is this image a cat, dot or cow?

House Price Prediction

- One of the most common ML tasks is predicting the price of a property
- For simplicity, we will assume we have only a single factor: the house size
- Goal: Learn and **Predict**
- 1) Learn
  - We collect data from '**N**' pairs of examples (input being the size, output being the price)
  - Learn the patterns/**associations** in these data
- 2) Predict
  - Given a new query of a ‘house size’, predict the ‘price’

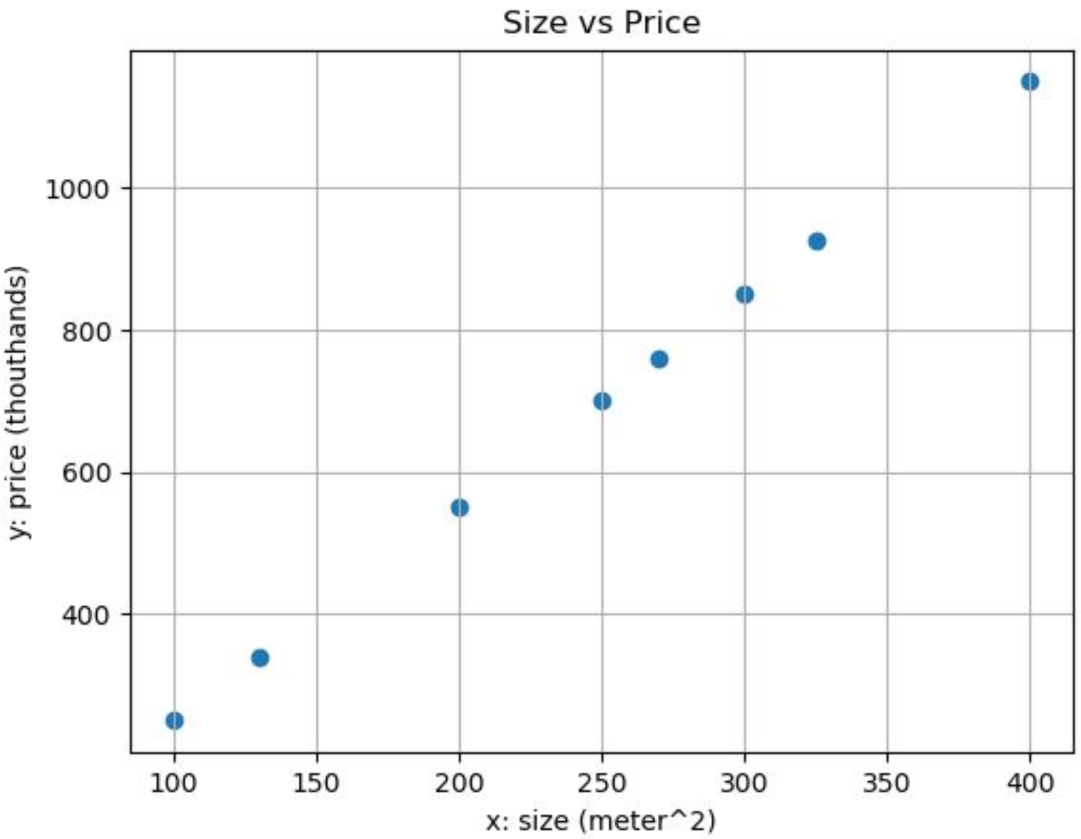
House Price Prediction: Dataset

ID	Size in meters^2	Price (**target**)
0	100	250,000
1	130	340,000
2	200	550,000
3	250	700,000
4	270	760,000
5	300	850,000
6	325	925,000
7	400	115,0000

- N = 8 (8 training samples)
- Size is X (input)
- Price is Y (output)
- The i-th examples is referred with: $x^{(i)}$, $y^{(i)}$

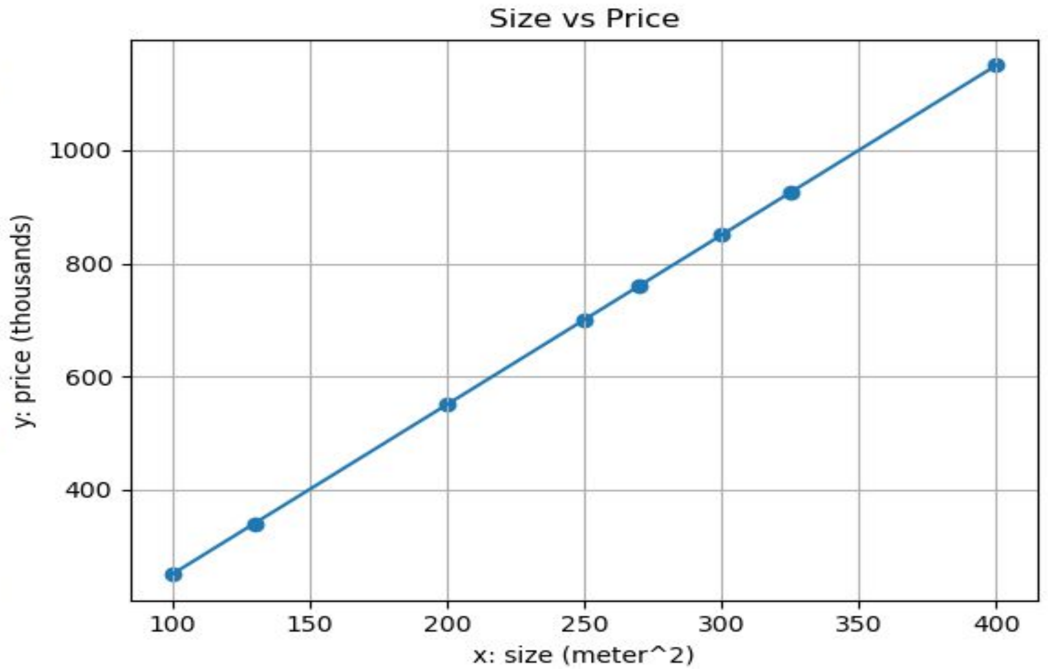
Visualization

- **Visualization** is a critical key for success
- Can you guess the price of a house of size 350 m^2?
- How can we model this data such that in future we can make a prediction **automatically?**



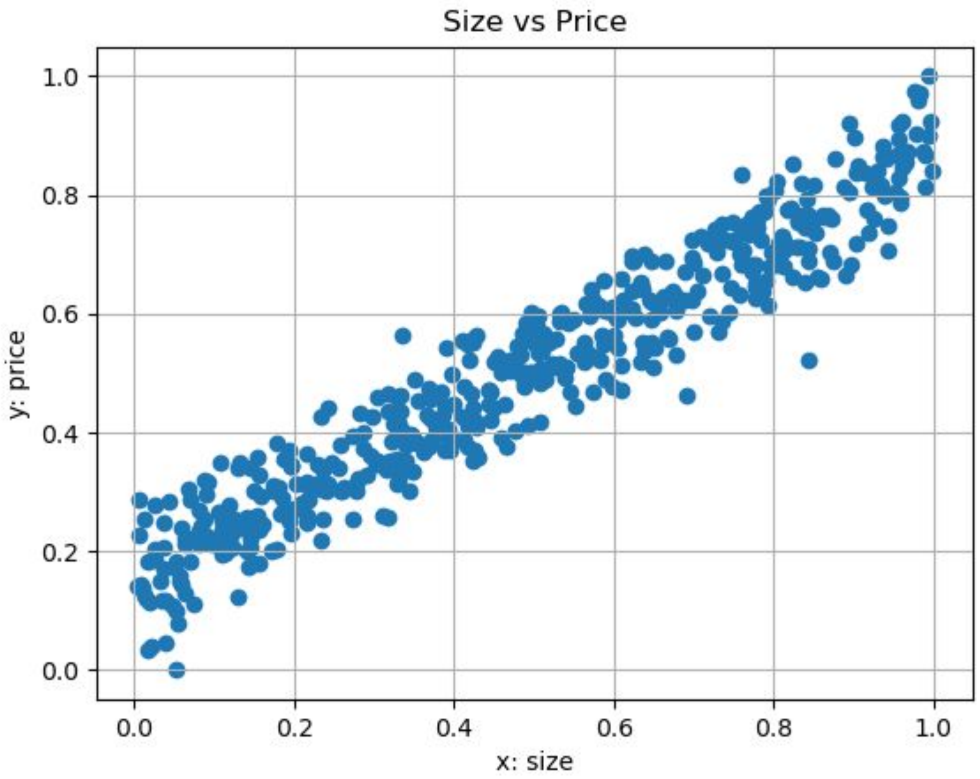
Modeling the data as a line

- The data seems came from a linear equation (mx+c)
- Use 2 points to compute these 2 **parameters (weights)**
- With some math:
  y = 3 * **x** - 50
- Given x = 350
  y = 1000 (thouthands)
- **We just learned from data**



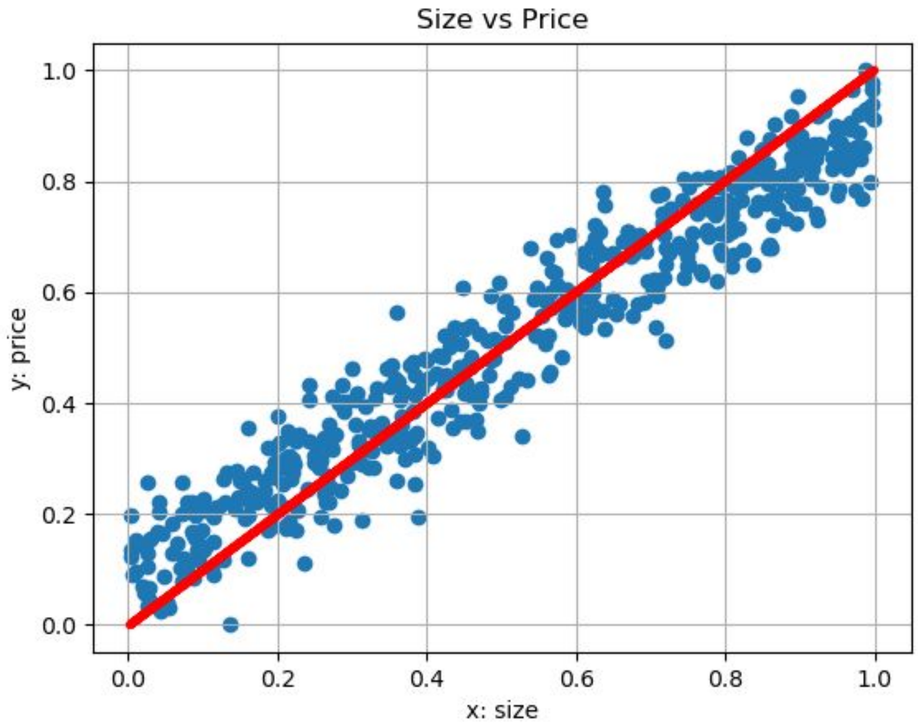
Real-life Data

- Sadly, real life data has variance (e.g. property house between 250k +/- 10k
- From numbers perspective, we can think, there is some **noise** added to our data
- Observe: the x and y range is now close to the [0, 1] range



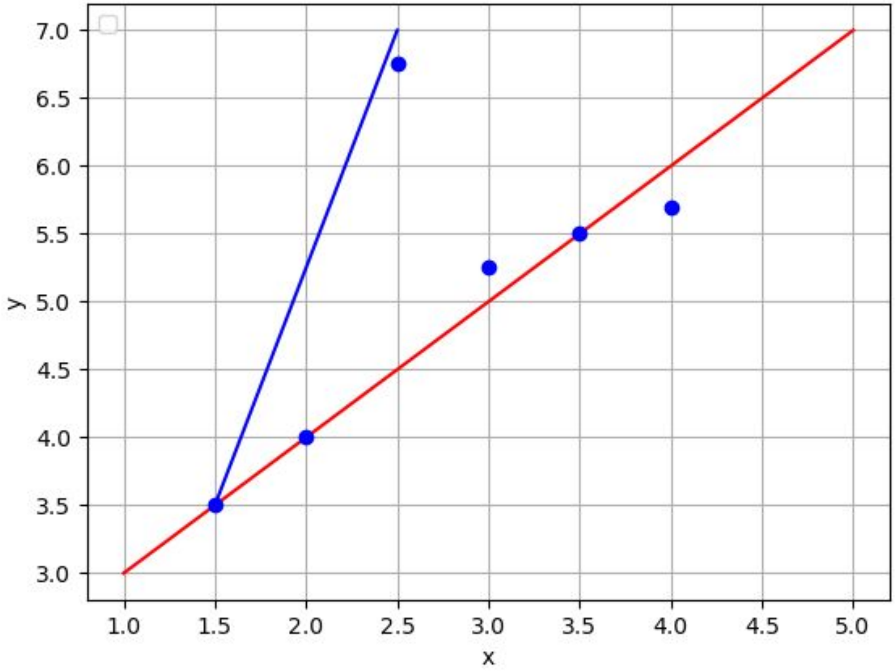
How to model such **noisy** data?

- Our intuition is, this data really came from some line
- How can we find one **good line** to fit the data as closely as possible?
  - Good is a vague word!
  - How to define the criteria?



Which line is a better fit?

- We have 6 data points (e.g. size vs price)
- 2 lines are proposed here
- Which one is a better fit?
- How did you decide so?

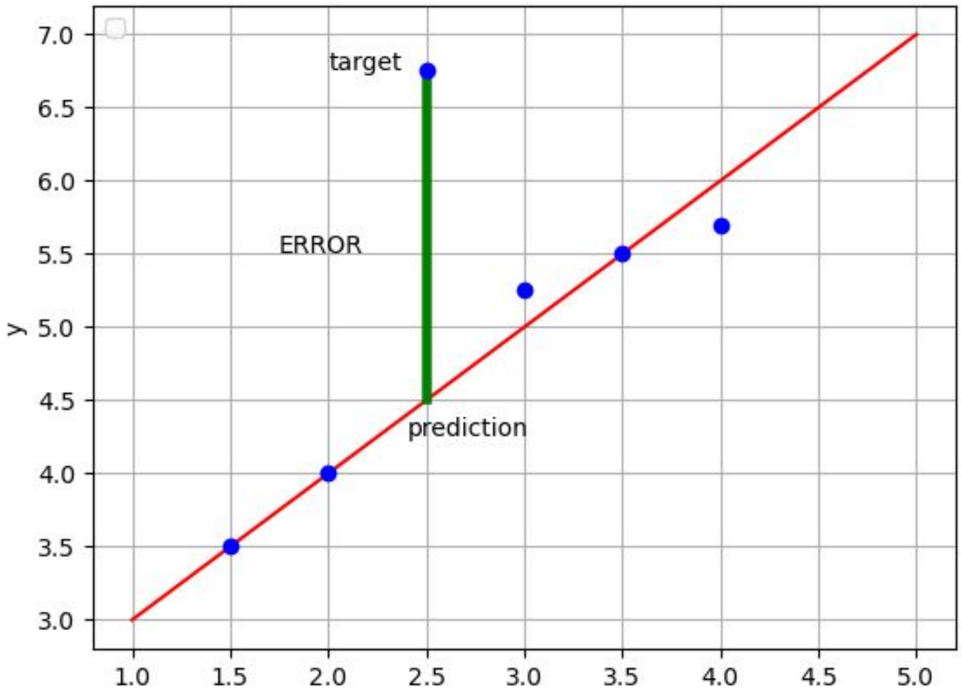


Criteria: The closest!

- We need the line that is **closer** to most of the points!
- How can we measure how close a line is to a set of data points?
- We need to use some **distance metric** between the ground truth and the prediction
  - Assume our dataset has point (size=200, price = 350,000)
  - Using size=200 in a line gives us price = 350,427
  - We need to compute the **distance** between 350,000 and 350,427
  - There is an **error** from the difference between these 2 values
    - Target (ground truth) vs prediction (of our model, the line)

Distance metric between 2 values

- Linear regression typically uses the squared error **cost** function
  - A cost function returns a numerical value based on the **error**
- The squared error function computes: (**target** - prediction)^2
  - **Error**: 6.75 - 4.5 = 2.25
    - Aka residual
  - **Squared error** =2.25 x 2.25



Mean Squared Error (MSE)

- Now, we know how to compute the error of a single point
- What about a dataset of M points?
- Simply sum the cost and average it
- Why average? To give the average **squared** error per example
  - Yi is the ground truth
  - We can calculate **square root** to get the average error (+ve)

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y_i}\right)^2.$$

- Without the 1/n, it is called the sum of square error (**SSE**)

Can we use other metrics?

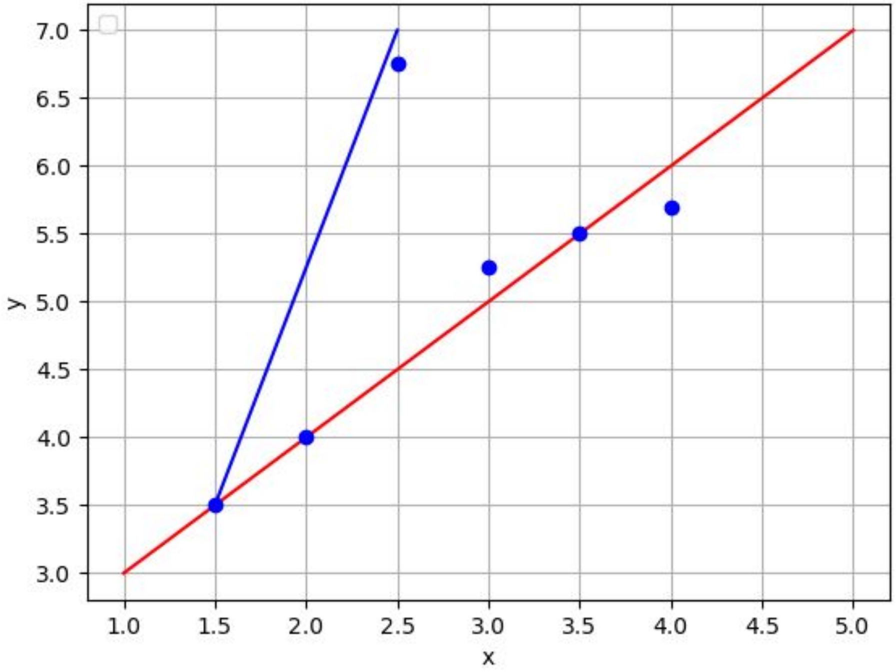
- Can we use other metrics such as the **absolute** **error** |T-P|? Yes
- There are many reasons for preferring squared error (easy to optimize, derivative everywhere, has a closed form, works in practice)
- However, one important reason is related to gaussian **noise**
  - You can find more mathematical details in famous books, such as Bishop's book
  - The web also has a lot of details
  - Gaussian also plays a vital role in math and statistics

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

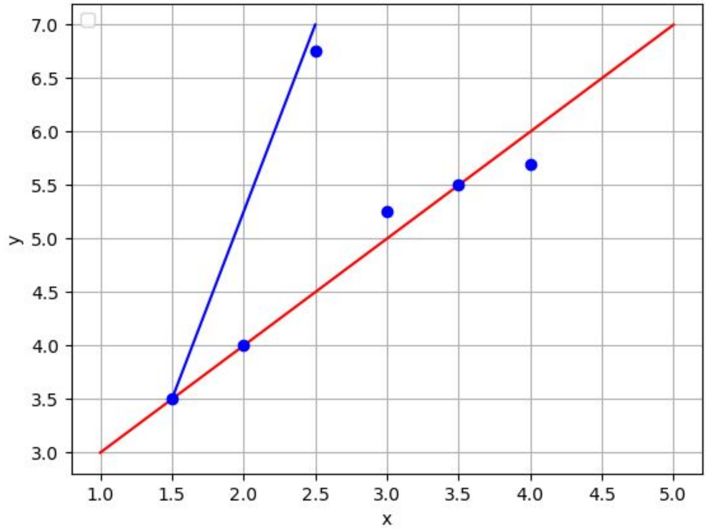
- One major drawback: sensitivity to outliers
- We may return to this concern later on
  - While rare, interviewers can sometimes stress regarding the deep differences
  - Observe: the normal distribution formula is a function in Euclidean distance

Back to the 2 lines

- Now we know a formula that can assign the total error/cost of a line
    - MSE: target vs prediction
- Then we can simply choose the line that has the smallest error
    - E.g. 5.0 (**red**) vs 27.9 (**blue**)
- The key question now is: Given the dataset, how can we find the **parameters** (m, c) that **minimize** the **cost** function?



```python
# let's pretend answer is
M = [2, 3.5, 4, 10, 5]
C = [1, 6, 3]
# ground truth (X, Y)
X = [1, 2, 3, 4, 5, 6]
Y = [6, 5, 4, 3, 2, 1]

def compute_cost(m, c):
    cost = 0
    for (x, y_gt) in zip(X, Y):
        y_pd = m * x + c
        err = y_gt - y_pd
        squared_err = err ** 2
        cost += squared_err
    return cost / len(Y) / 2

if __name__ == '__main__':
    best_cost = float("inf")
    for m in M:
        for c in C:
            this_cost = compute_cost(m, c)

            if best_cost > this_cost:
                best_cost = this_cost
```

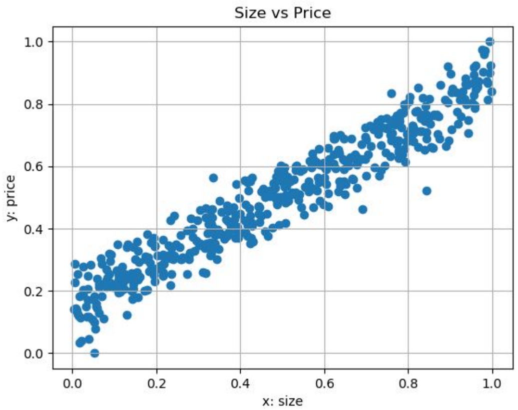


$$\underset{m,c}{\text{minimize}}\, cost(m, c)$$

$$cost(m, c) = \frac{1}{2N} \sum_{n=1}^{N} ((mx^{(n)} + c) - t^{(n)})^2$$

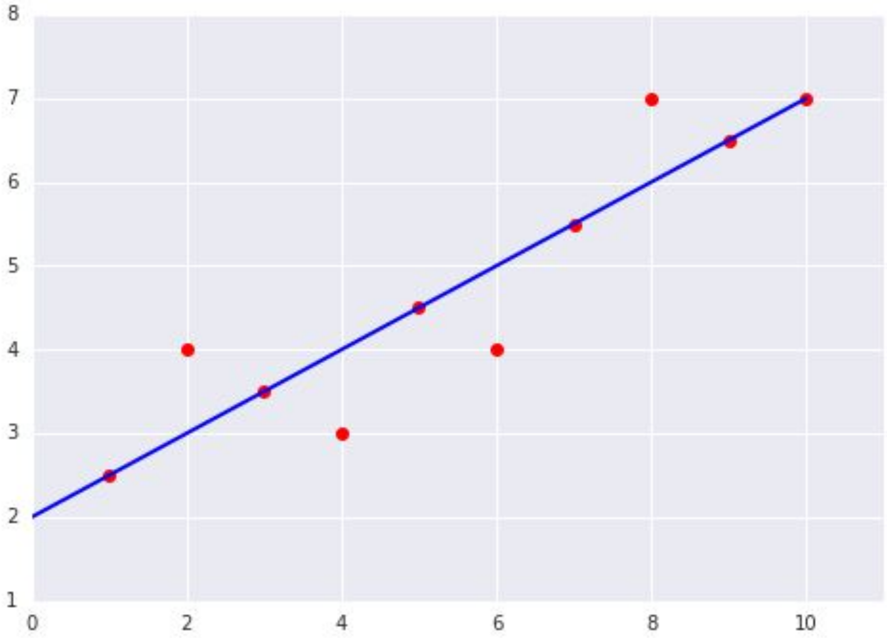
Code snippet

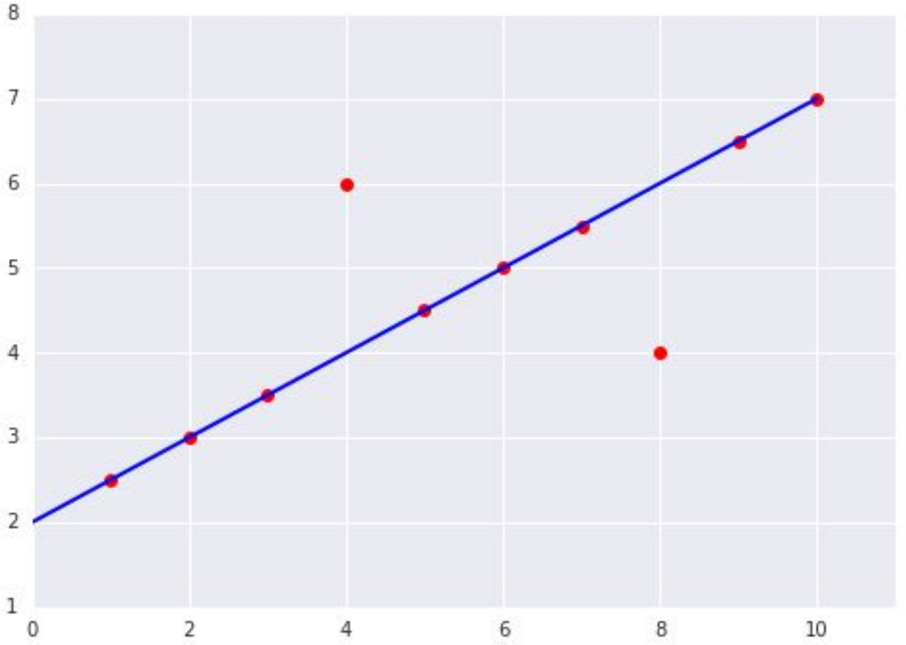
```
def scale(y):    # convert y to [0, 1] range. LATER
    mn, mx = np.min(y), np.max(y)
    y = (y - mn) / (mx - mn)
    return y

x = np.random.rand(500)           # 500 random value to visualize

mu, sigma = 0, 0.3      # Mean and standard deviation
noise = np.random.normal(mu, sigma, 500)     # also see np.random.randn


y = 3*x - 50 + noise
y = scale(y)
```



Question (src)

- We have a line and 2 datasets. Which data has the higher MSE?





The dataset on the right.

The eight examples on the line incur a total loss of 0. However, although only two points lay off the line, both of those points are *twice* as far off the line as the outlier points in the left figure. Squared loss amplifies those differences, so an offset of two incurs a loss four times as great as an offset of one.

$$MSE = \frac{0^2 + 0^2 + 0^2 + 2^2 + 0^2 + 0^2 + 0^2 + 2^2 + 0^2 + 0^2}{10} = 0.8$$

Correct answer.

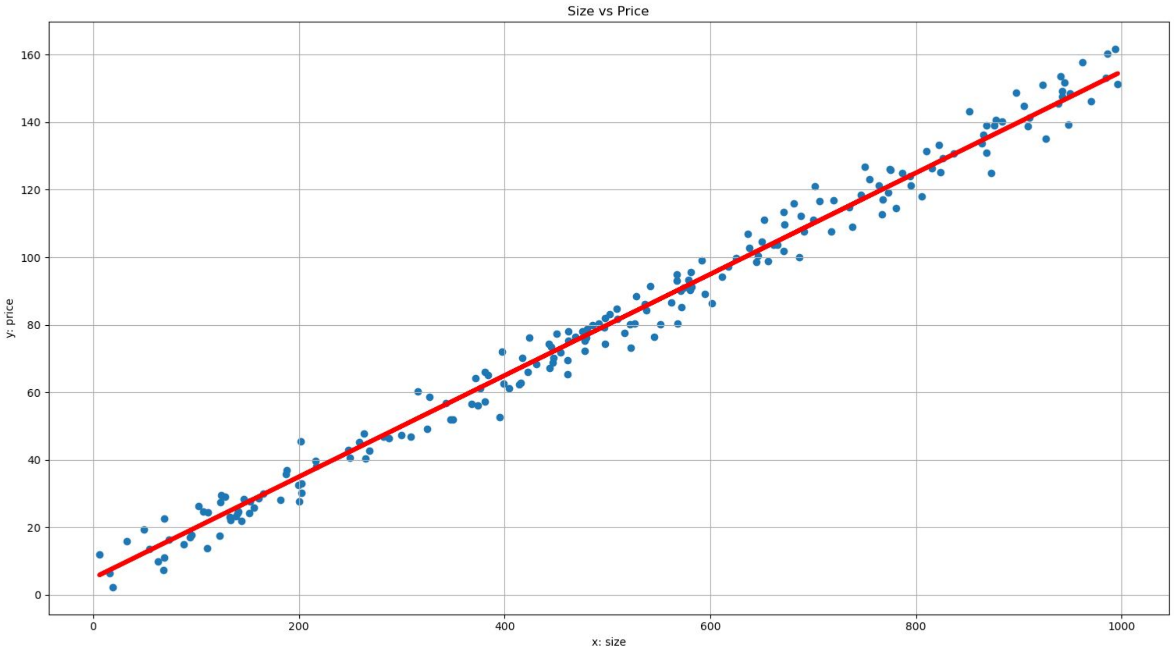
The dataset on the left.

The six examples on the line incur a total loss of 0. The four examples not on the line are not very far off the line, so even squaring their offset still yields a low value:

$$MSE = \frac{0^2 + 1^2 + 0^2 + 1^2 + 0^2 + 1^2 + 0^2 + 1^2 + 0^2 + 0^2}{10} = 0.4$$

Question!

- Assume we learned the below line of data (size vs price in thouthands)
- Now a new query is: what is the price for a house of size 600?



Question!

- Assume all our data have the same x (e.g. multiple prices of the same house specs)
  - E.g. (5, 3), (5, 7), (5, 50)
- What is an optimal line representing the data?

- Any line passing with the point x = x, y = average(ys)
  - x = 5, y = 20

Relevant Materials

- Prof Andrew Ng: video1, video2
- Hesham Asem (Arabic): video1, video

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."