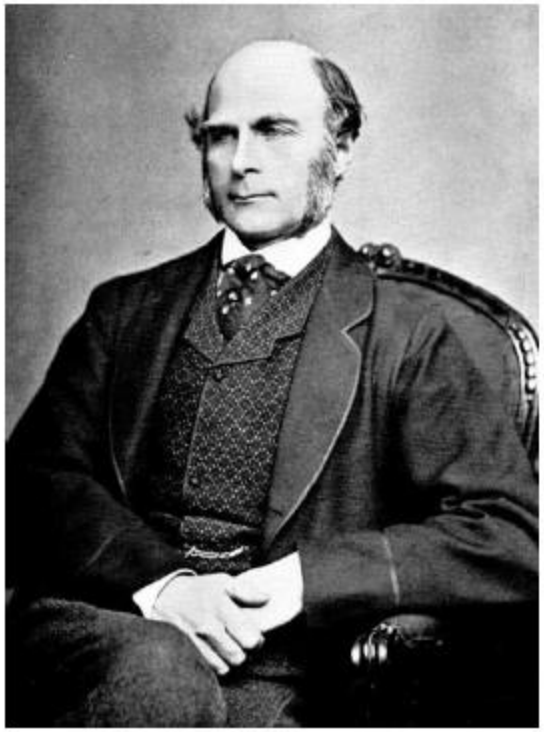
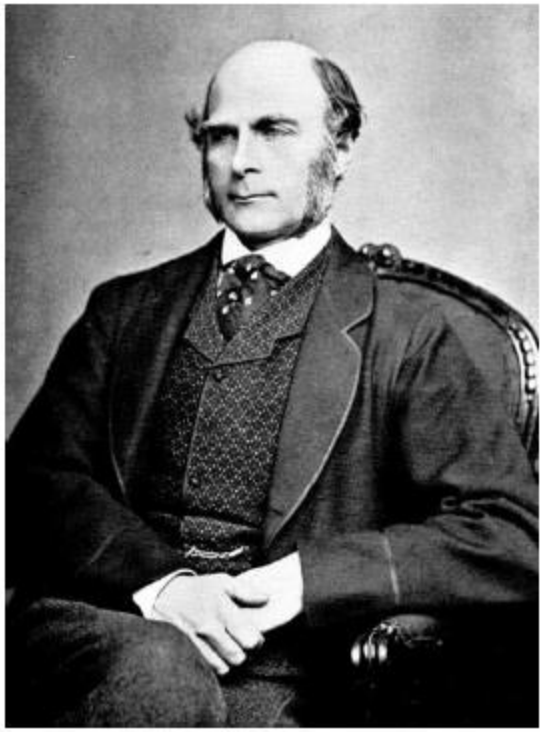
Introduction to Linear Regression

This all started in the 1800s with a guy named Francis Galton. Galton was studying the relationship between parents and their children. In particular, he investigated the relationship between the heights of fathers and their sons.



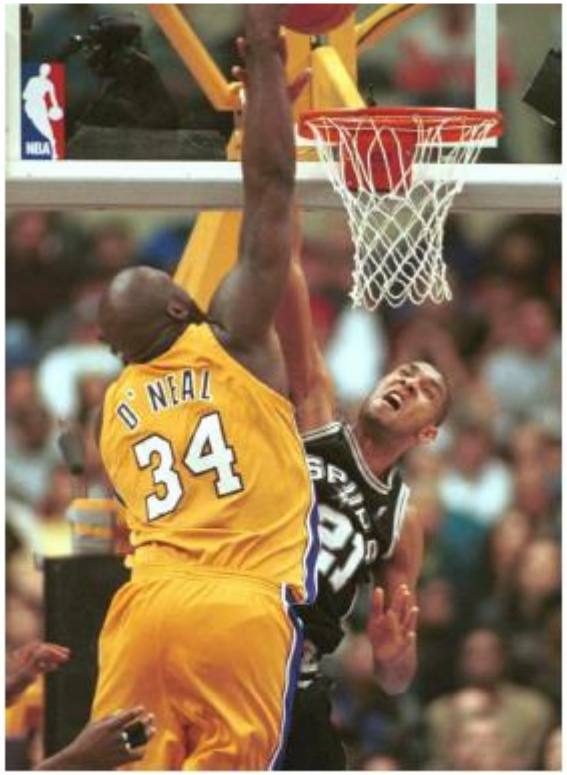
What he discovered was that a man's son tended to be roughly as tall as his father.

However Galton's breakthrough was that the son's height **tended to be closer to the overall average** height of all people.



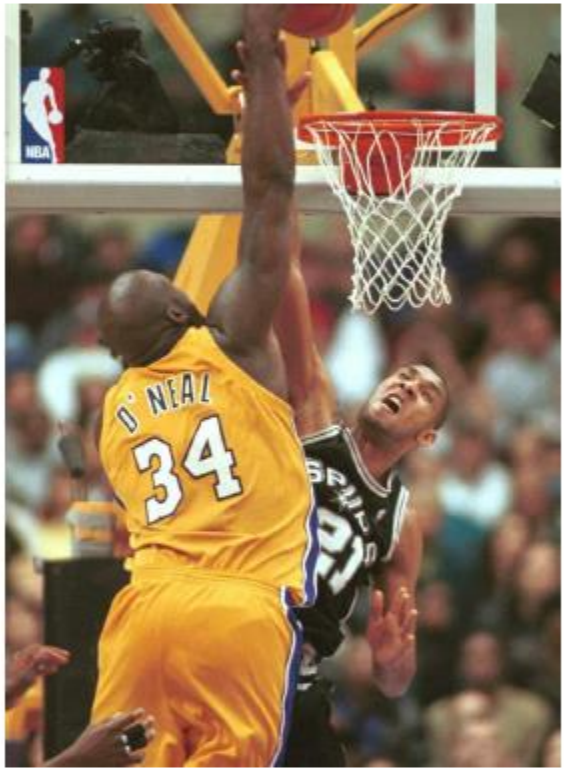
Let's take Shaquille O'Neal as an example. Shaq is really tall:7ft 1in (2.2 meters).

If Shaq has a son, chances are he'll be pretty tall too. However, Shaq is such an anomaly that there is also a very good chance that his son will be **not be as tall as Shaq**.

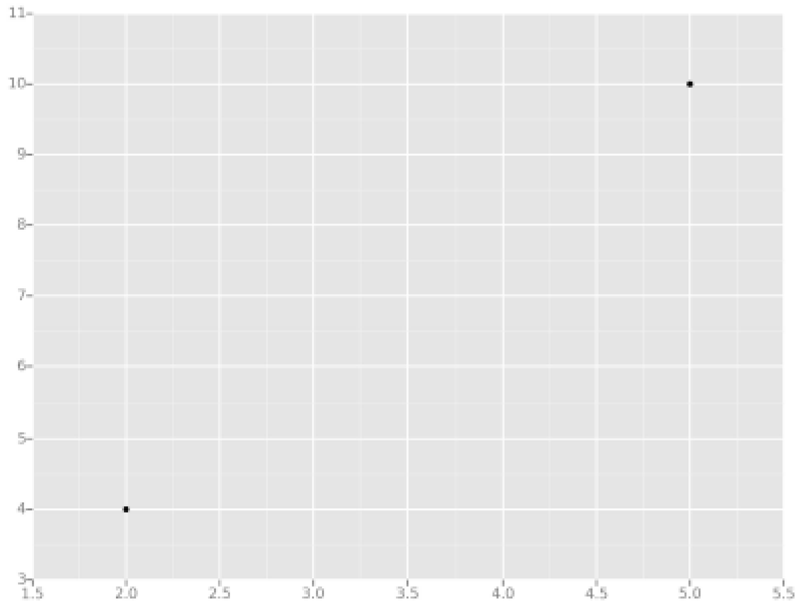


Turns out this is the case: Shaq's son is pretty tall (6 ft 7 in), but not nearly as tall as his dad.

Galton called this phenomenon **regression**, as in "A father's son's height tends to regress (or drift towards) the mean (average) height."

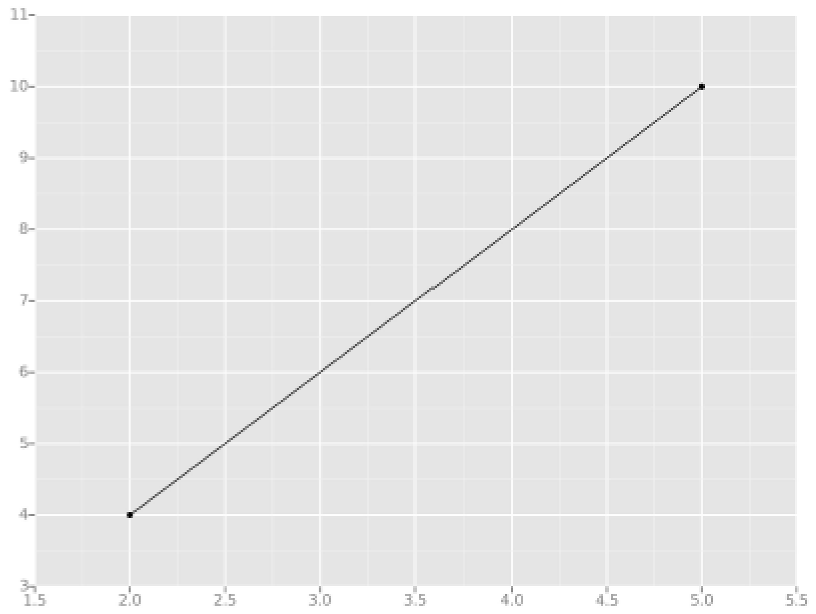


Let's take the simplest possible example: calculating a regression with only 2 data points.



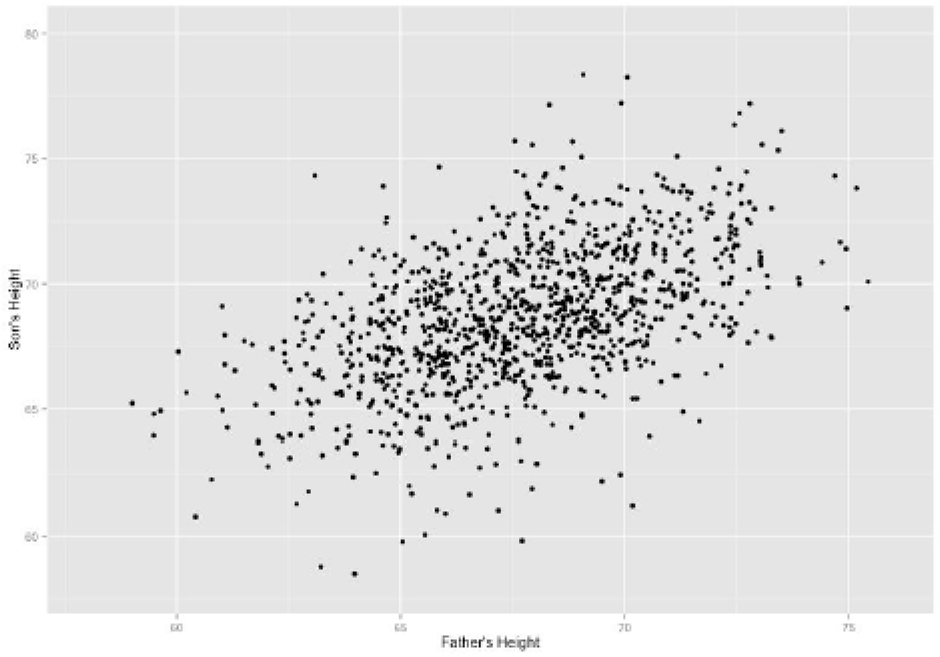
All we're trying to do when we calculate our regression line is draw a line that's as close to every dot as possible.

For classic linear regression, or "Least Squares Method", you only measure the closeness in the "up and down" direction



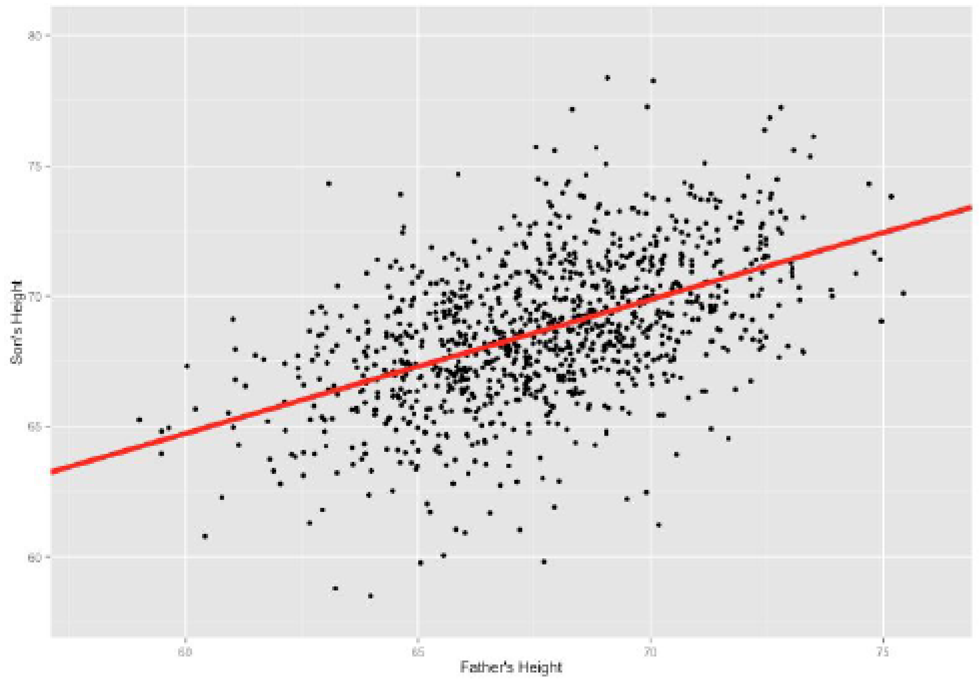
Now wouldn't it be great if we could apply this same concept to a graph with more than just two data points?

By doing this, we could take multiple men and their son's heights and do things like tell a man how tall we expect his son to be...before he even has a son!

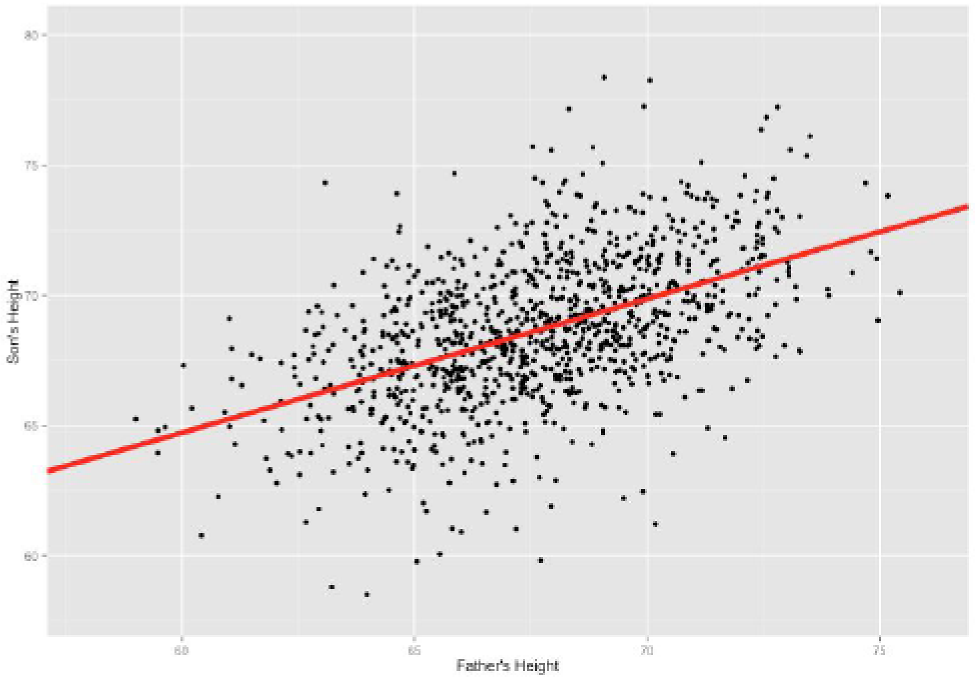


Our goal with linear regression is to **minimize the vertical distance** between all the data points and our line.

So in determining the **best line**, we are attempting to minimize the distance between **all** the points and their distance to our line.

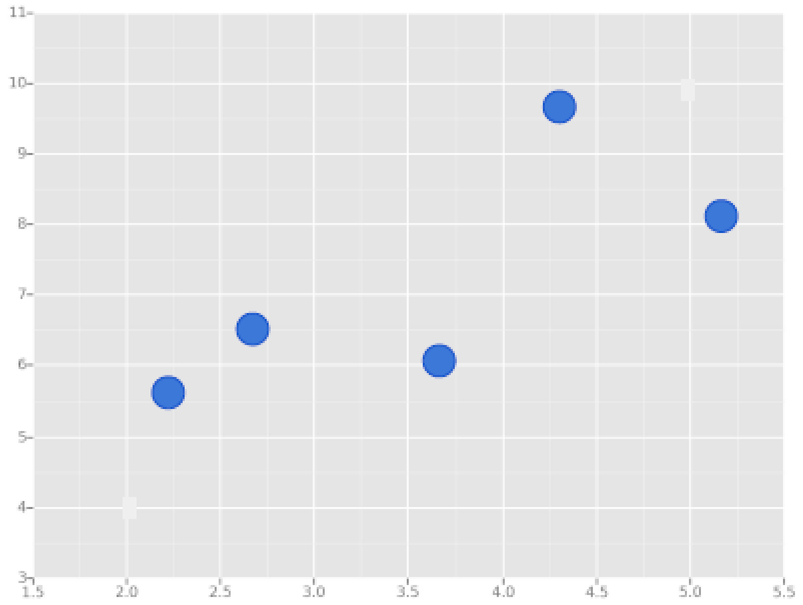


There are lots of different ways to minimize this, (sum of squared errors, sum of absolute errors, etc), but all these methods have a general goal of minimizing this distance.



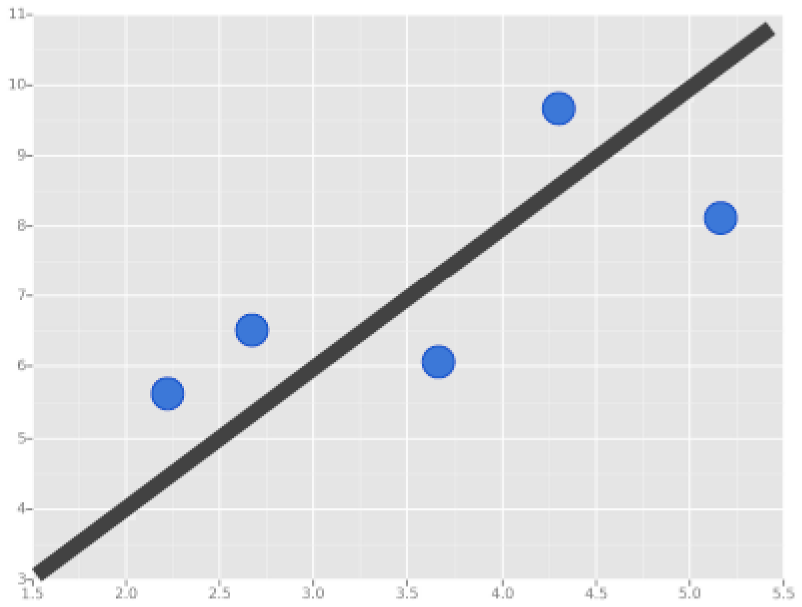
For example, one of the most popular methods is the least squares method.

Here we have blue data points along an x and y axis.



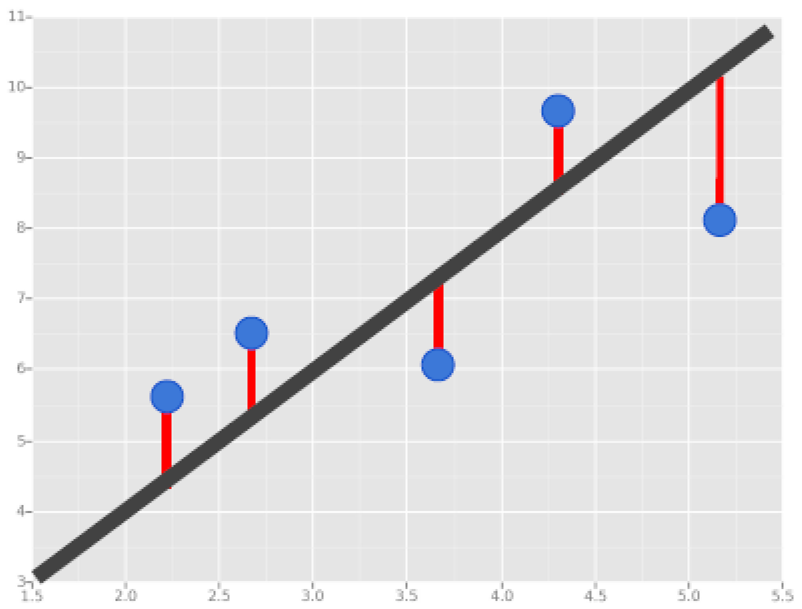
Now we want to fit a linear regression line.

The question is, how do we decide which line is the best fitting one?

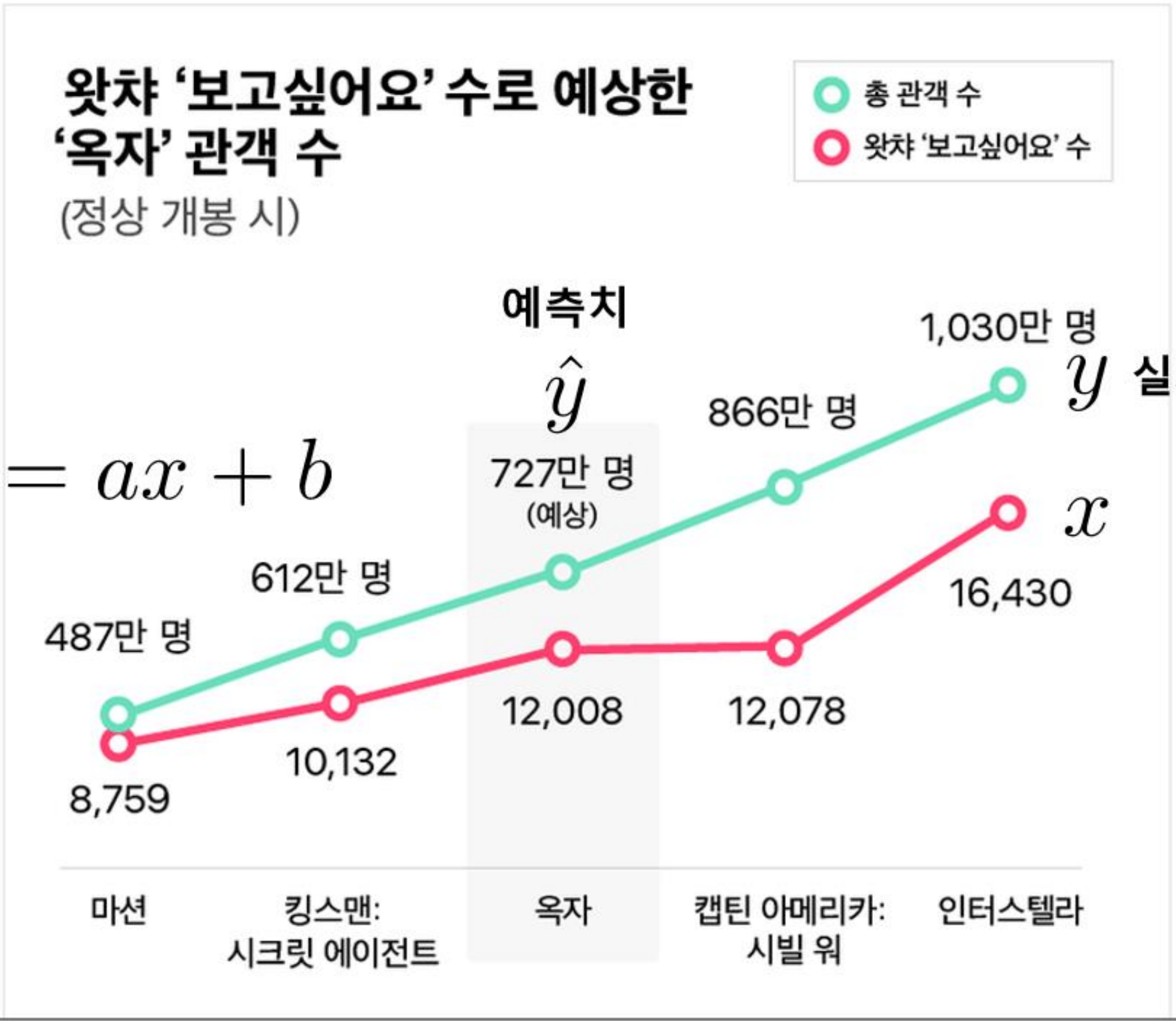


We'll use the Least Squares Method, which is fitted by minimizing the ***sum of squares of the residuals.***

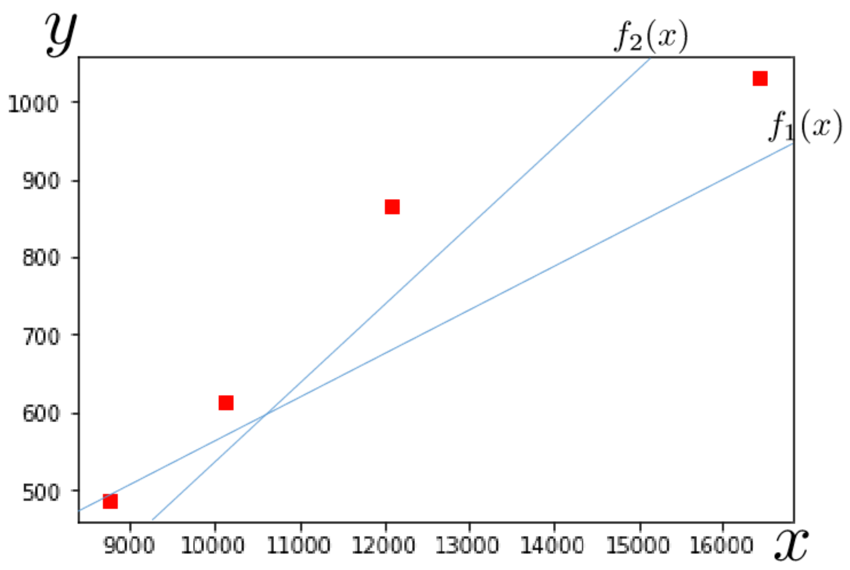
The residuals for an observation is the difference between the observation (the y-value) and the fitted line.



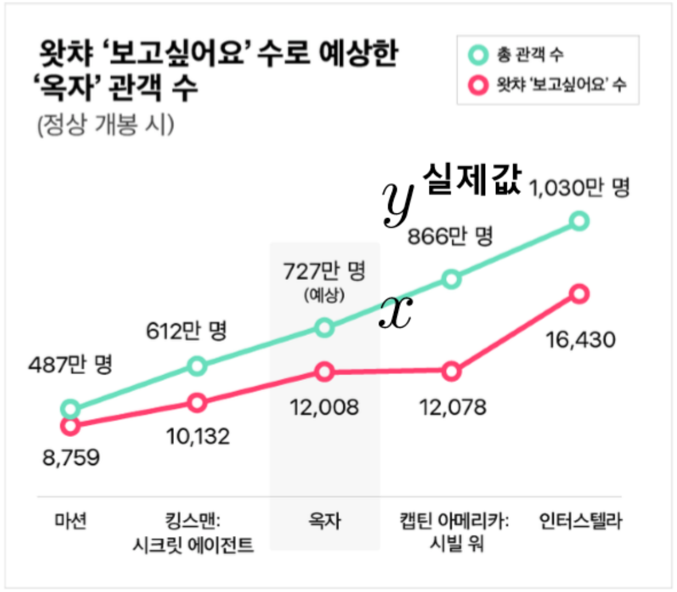
$$\hat{y} = ax + b$$


왓챠 '보고싶어요' 수로 예상한 '옥자' 관객 수
(정상 개봉 시)
○ 총 관객 수
○ 왓챠 '보고싶어요' 수
예측치
$\hat{y}$
487만 명
612만 명
727만 명 (예상)
866만 명
1,030만 명
y 실제값
x
8,759
10,132
12,008
12,078
16,430
마션
킹스맨: 시크릿 에이전트
옥자
캡틴 아메리카: 시빌 워
인터스텔라

$$f(x) = \hat{y} = ax + b$$

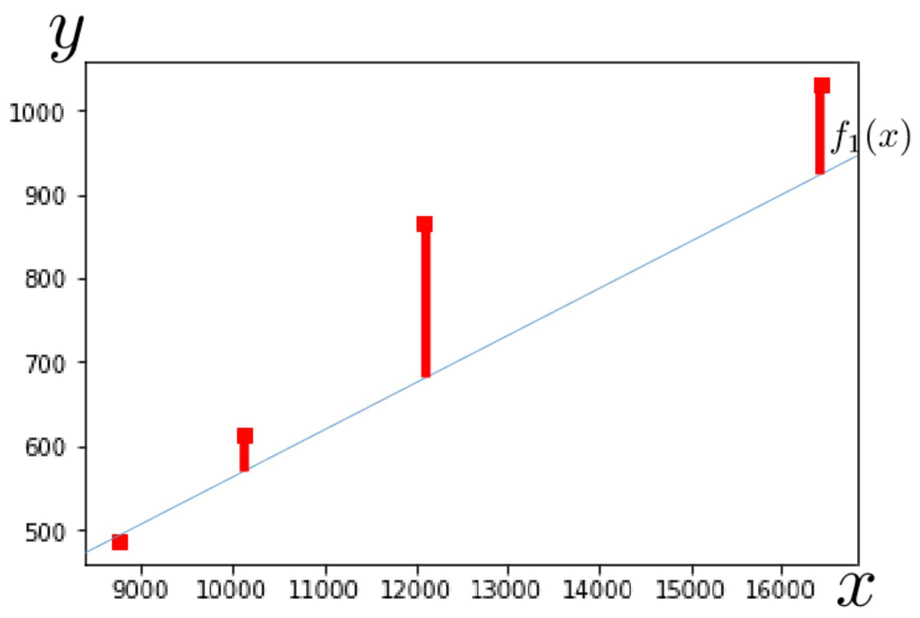


y
$f_2(x)$
$f_1(x)$
1000
900
800
700
600
500
9000
10000
11000
12000
13000
14000
15000
16000
x

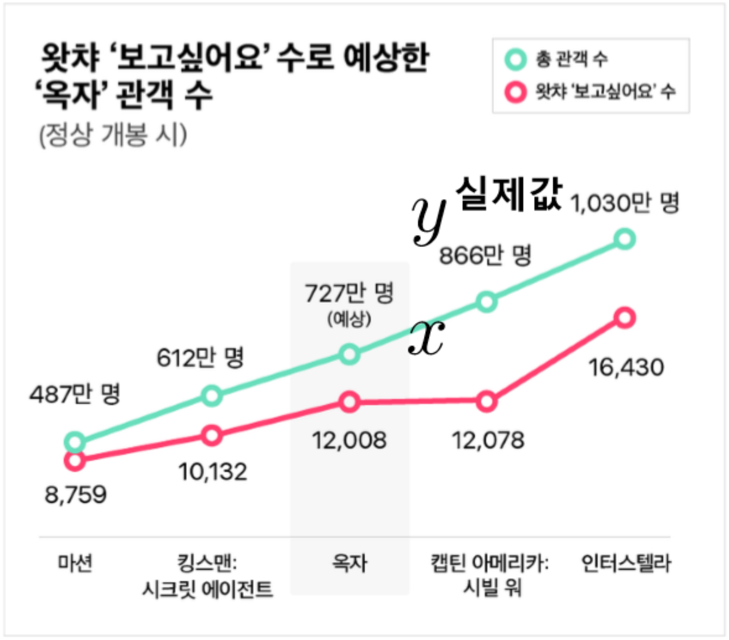


왓챠 '보고싶어요' 수로 예상한 '옥자' 관객 수
(정상 개봉 시)
○ 총 관객 수
○ 왓챠 '보고싶어요' 수
y 실제값
1,030만 명
866만 명
727만 명 (예상)
612만 명
487만 명
x
16,430
12,008
12,078
10,132
8,759
마션
킹스맨: 시크릿 에이전트
옥자
캡틴 아메리카: 시빌 워
인터스텔라

$$f(x) = \hat{y} = ax + b$$



y
1000
900
800
700
600
500
9000
10000
11000
12000
13000
14000
15000
16000
x
$f_1(x)$



왓챠 '보고싶어요' 수로 예상한 '옥자' 관객 수
(정상 개봉 시)
총 관객 수
왓챠 '보고싶어요' 수
y 실제값
1,030만 명
866만 명
727만 명 (예상)
612만 명
487만 명
x
16,430
12,008
12,078
10,132
8,759
마션
킹스맨: 시크릿 에이전트
옥자
캡틴 아메리카: 시빌 워
인터스텔라

오차의 합

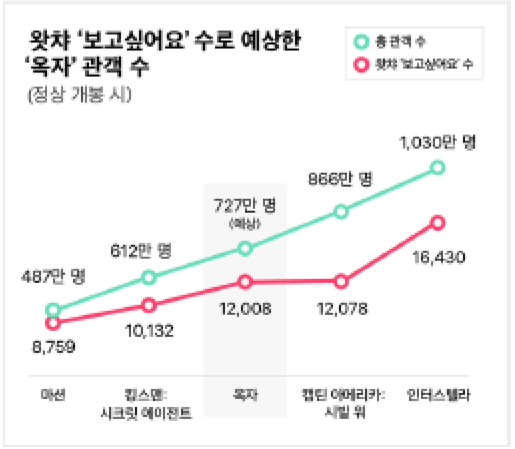
$$(\hat{y}^{(1)} - y^{(1)}) + (\hat{y}^{(2)} - y^{(2)}) + (\hat{y}^{(3)} - y^{(3)}) + (\hat{y}^{(4)} - y^{(4)})$$

오차는 양수 또는 음수 가능 ➔ 상쇄될 수 있음

$$(\hat{y}^{(1)} - y^{(1)})^2 + (\hat{y}^{(2)} - y^{(2)})^2 + (\hat{y}^{(3)} - y^{(3)})^2 + (\hat{y}^{(4)} - y^{(4)})^2$$

제곱의 합으로 변환

$$\sum_{i=1}^{n}(\hat{y}^{(i)} - y^{(i)})^2 \quad \hat{y} = \begin{bmatrix} w_1 \times 8759 + w_0 \\ w_1 \times 10132 + w_0 \\ w_1 \times 12078 + w_0 \\ w_1 \times 16430 + w_0 \end{bmatrix} \quad y = \begin{bmatrix} 487 \\ 612 \\ 866 \\ 1030 \end{bmatrix}$$



$$(\hat{y} - y)^2 = \begin{bmatrix} (w_1 \times 8759 + w_0 - 487)^2 \\ (w_1 \times 10132 + w_0 - 612)^2 \\ (w_1 \times 12078 + w_0 - 866)^2 \\ (w_1 \times 16430 + w_0 - 1030)^2 \end{bmatrix}$$

Squared Error

$$\sum_{i=1}^{n}(w_1 x^{(i)} + w_0 \times 1 - y^{(i)})^2$$

최소 또는 최대의 문제 → 미분으로 해결하기

찾고자 하는 값? w_1, w_0

앞으로 우리는

$$f(x) = h_\theta(x)$$

예측 함수를 가설 함수라고 부를 예정

실제값과 가설함수의 차이

$$J(w_0, w_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Cost function이라고 부를 예정

Cost function에서 구하는 것

$$\arg\min_{\theta} \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

cost function의 최소화를 위한 weight 값

$$y^{(1)} = w_0 + w_1 x^{(1)} + \epsilon^{(1)}$$
$$y^{(2)} = w_0 + w_1 x^{(2)} + \epsilon^{(2)}$$
$$y^{(3)} = w_0 + w_1 x^{(3)} + \epsilon^{(3)}$$
$$y^{(4)} = w_0 + w_1 x^{(4)} + \epsilon^{(4)}$$
$$y^{(5)} = w_0 + w_1 x^{(5)} + \epsilon^{(5)}$$

$$\mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ y^{(3)} \\ y^{(4)} \\ y^{(5)} \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} 1 & x^{(1)} \\ 1 & x^{(2)} \\ 1 & x^{(3)} \\ 1 & x^{(4)} \\ 1 & x^{(5)} \end{bmatrix}$$

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} \quad \mathbf{y} = \mathbf{X}\mathbf{w}$$

$$J = \frac{1}{2}\sum_{i=1}^{m}(w_1 x^{(i)} + w_0 - y^{(i)})^2$$

$$\frac{\partial J}{\partial w_0} = \sum(w_1 x^{(i)} + w_0 - y^{(i)}) = 0$$

$$\frac{\partial J}{\partial w_1} = \sum(w_1 x^{(i)} + w_0 - y^{(i)})x^{(i)} = 0$$

위 식을 만족하는 $\hat{w}_j$ 값을 구하기

$$\hat{w}_0 m + \hat{w}_1 \sum x^{(i)} = \sum y^{(i)}$$
$$\hat{w}_0 \sum x^{(i)} + \hat{w}_1 \sum (x^{(i)})^2 = \sum y^{(i)} x^{(i)}$$

$$\mathbf{X}^T\mathbf{X} = \begin{bmatrix} m & \sum x^{(i)} \\ \sum x^{(i)} & \sum (x^{(i)})^2 \end{bmatrix}$$

$$(\mathbf{X}^T\mathbf{X})\hat{\mathbf{w}} = \mathbf{X}^T\mathbf{y}$$
$$\Downarrow$$
$$\hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

$$\mathbf{X} = \begin{bmatrix} 1 & x^{(1)} \\ 1 & x^{(2)} \\ 1 & x^{(3)} \\ 1 & x^{(4)} \\ 1 & x^{(5)} \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ y^{(3)} \\ y^{(4)} \\ y^{(5)} \end{bmatrix}$$

$$\hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

$$\begin{aligned}\hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} &= \begin{bmatrix}\hat{w}_0 \\ \hat{w}_1\end{bmatrix} \\ &= \frac{1}{\sum(x^{(i)} - \bar{x})^2}\begin{bmatrix}\sum(x^{(i)})^2/m & -\bar{x} \\ -\bar{x} & m\end{bmatrix}\begin{bmatrix}\sum y^{(i)} \\ \sum x^{(i)}y^{(i)}\end{bmatrix}\end{aligned}$$

$$\begin{aligned}\hat{w}_1 &= \frac{\sum x^{(i)}y^{(i)} - m\bar{x}\bar{y}}{\sum(x^{(i)} - \bar{x})^2} \\ \hat{w}_0 &= \bar{y} - \hat{w}_1\bar{x}\end{aligned}$$

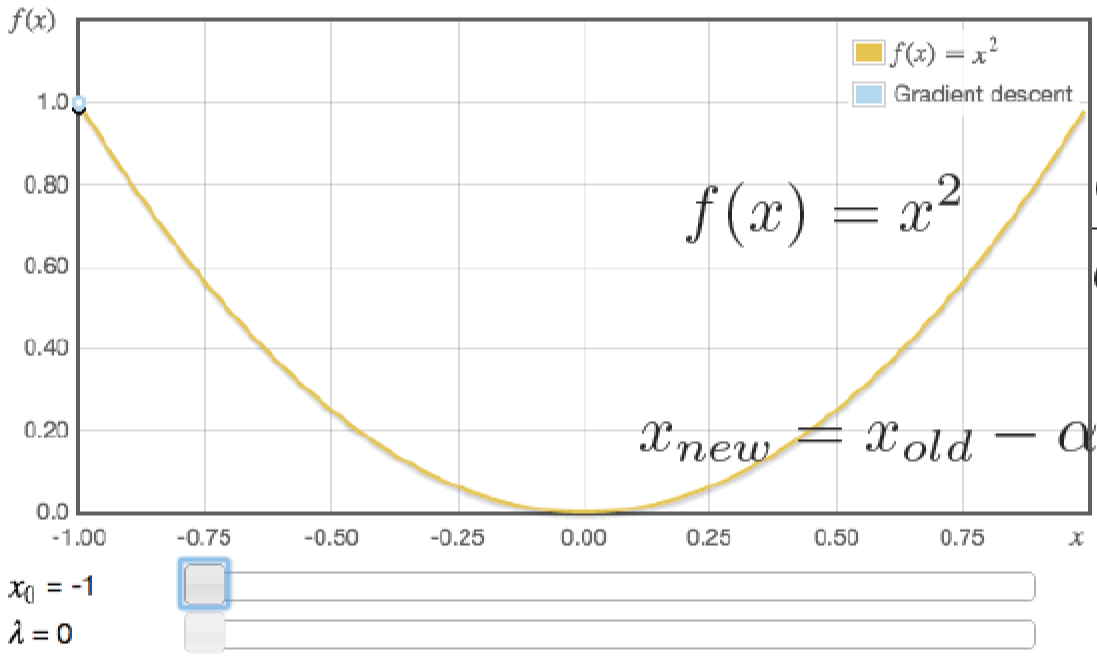
결론

$$\hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Normal equation

- $X^T X$ 의 역행렬이 존재할 때 사용
- Iteration 등 사용자 지정 parameter가 없음
- Feature가 많으면 계산 속도가 느려짐

컴퓨터에 x^2 의 최적값 찾기



$$f(x) = x^2 \qquad \frac{dy}{dx} = 2x$$

$$x_{new} = x_{old} - \alpha \times (2x_{old})$$

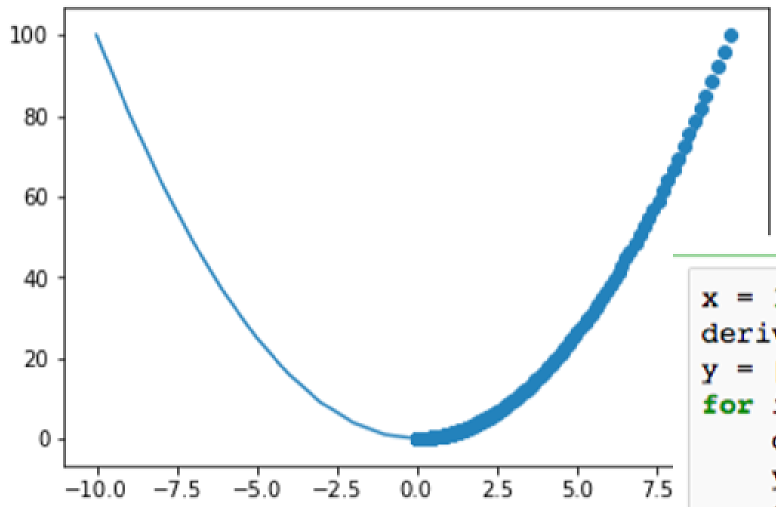
컴퓨터에 x^2 의 최적값 찾기

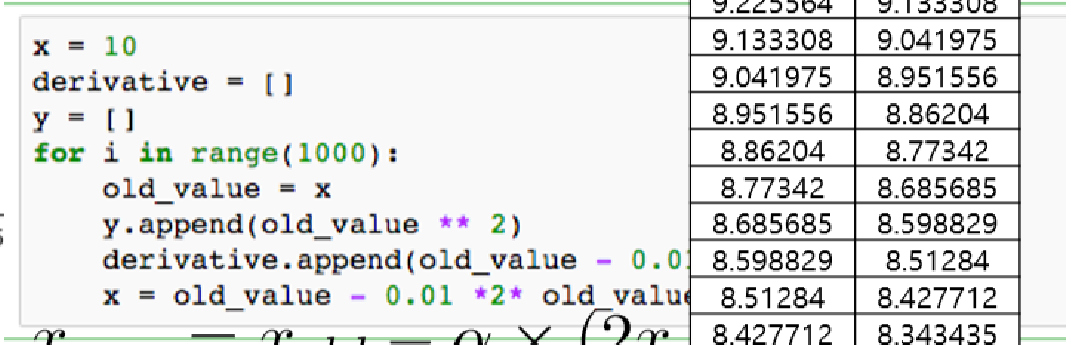
$$f(x) = x^2 \qquad \frac{dy}{dx} = 2x$$

$$x_{new} = x_{old} - \alpha \times (2x_{old})$$

$$\begin{bmatrix} 1 \\ 1 - 0.1 * 2 * 1 = 0.8 \\ 0.8 - 0.1 * 2 * 0.8 = 0.64 \\ 0.64 - 0.1 * 2 * 0.64 = 0.512 \\ \vdots \end{bmatrix}$$

컴퓨터에 x^2 의 최적값 찾기





```
x = 10
derivative = []
y = []
for i in range(1000):
    old_value = x
    y.append(old_value ** 2)
    derivative.append(old_value - 0.0
    x = old_value - 0.01 *2* old_value
```

$$x_{new} = x_{old} - \alpha \times (2x_{old})$$

gradient	v
	10
20	9.8
19.6	9.604
9.604	9.50796
9.50796	9.41288
9.41288	9.318752
9.318752	9.225564
9.225564	9.133308
9.133308	9.041975
9.041975	8.951556
8.951556	8.86204
8.86204	8.77342
8.77342	8.685685
8.685685	8.598829
8.598829	8.51284
8.51284	8.427712
8.427712	8.343435

Linear regression with GD

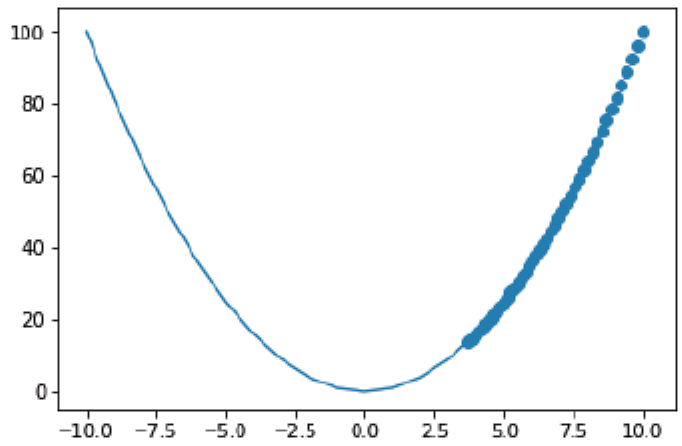
$$x_{new} = x_{old} - \alpha \times (2x_{old})$$

$$\frac{\partial J}{\partial w_0} = \frac{1}{m}\sum_{i=1}^{m}(w_1 x^{(i)} + w_0 - y^{(i)})$$

$$\frac{\partial J}{\partial w_1} = \frac{1}{m}\sum_{i=1}^{m}(w_1 x^{(i)} + w_0 - y^{(i)})x^{(i)}$$

```
loop until convergence{
    do $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$
}
```

너무 작을 경우



$$x_{new} = x_{old} - \alpha \times (2x_{old})$$

끝까지 못감
시간이 오래 걸림

식은 많아지지만 여전히 Cost 함수의 최적화

$$J(w_0, w_1) = \frac{1}{2m} \sum_{i=1}^{m} (w_1 x^{(i)} + w_0 - y^{(i)})^2$$

$$\frac{\partial J}{\partial w_0} = \frac{1}{m} \sum_{i=1}^{m} (w_1 x^{(i)} + w_0 - y^{(i)})$$

$$\frac{\partial J}{\partial w_1} = \frac{1}{m} \sum_{i=1}^{m} (w_1 x^{(i)} + w_0 - y^{(i)}) x^{(i)}$$

Regression metrics

- Mean Absolute Error

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| = \frac{1}{n}\sum_{i=1}^{n}|e_i|.$$

잔차의 절대값의 Sum

```python
from sklearn.metrics import median_absolute_error
y_true = [3, -0.5, 2, 7]
y_pred = [2.5, 0.0, 2, 8]
median_absolute_error(y_true, y_pred)
```

Regression metrics

- Root Mean Squared Error (RMSE)

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

잔차 제곱의 sum의 루트

```
from sklearn.metrics import mean_squared_error
y_true = [3, -0.5, 2, 7]
y_pred = [2.5, 0.0, 2, 8]
mean_squared_error(y_true, y_pred)
```

Regression metrics

- R squared

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \mu)^2}.$$

0과 1사이 숫자로 크면 클 수록 높은 적합도를 지님

```
from sklearn.metrics import r2_score
y_true = [3, -0.5, 2, 7]
y_pred = [2.5, 0.0, 2, 8]
r2_score(y_true, y_pred)
```

Holdout Method (Sampling)

- 데이터를 Training과 Test와 나눠서 모델을 생성하고 테스트하는 기법

- 가장 일반적인 모델 생성을 위한 데이터 램덤 샘플링 기법

- Training과 Test를 나누는 비율은 데이터의 크기에 따라 다름

- 일반적으로 Training Data 2/3 , Test Data 1/3를 활용함

모델을 만드는 데이터와 평가받는 데이터를 나누자

Training & Test data set

- Training한 데이터로 다시 Test를 할 경우,
  Training 데이터에 과도하게 fitting 된 모델을 사용될 수 있음

- 새로운 데이터가 출현했을 때, 기존 모델과의 차이 존재

- 모델은 새로운 데이터가 처리가능하도록 generalize되야함

- 이를 위해 Training Set과 Test Set을 분리함