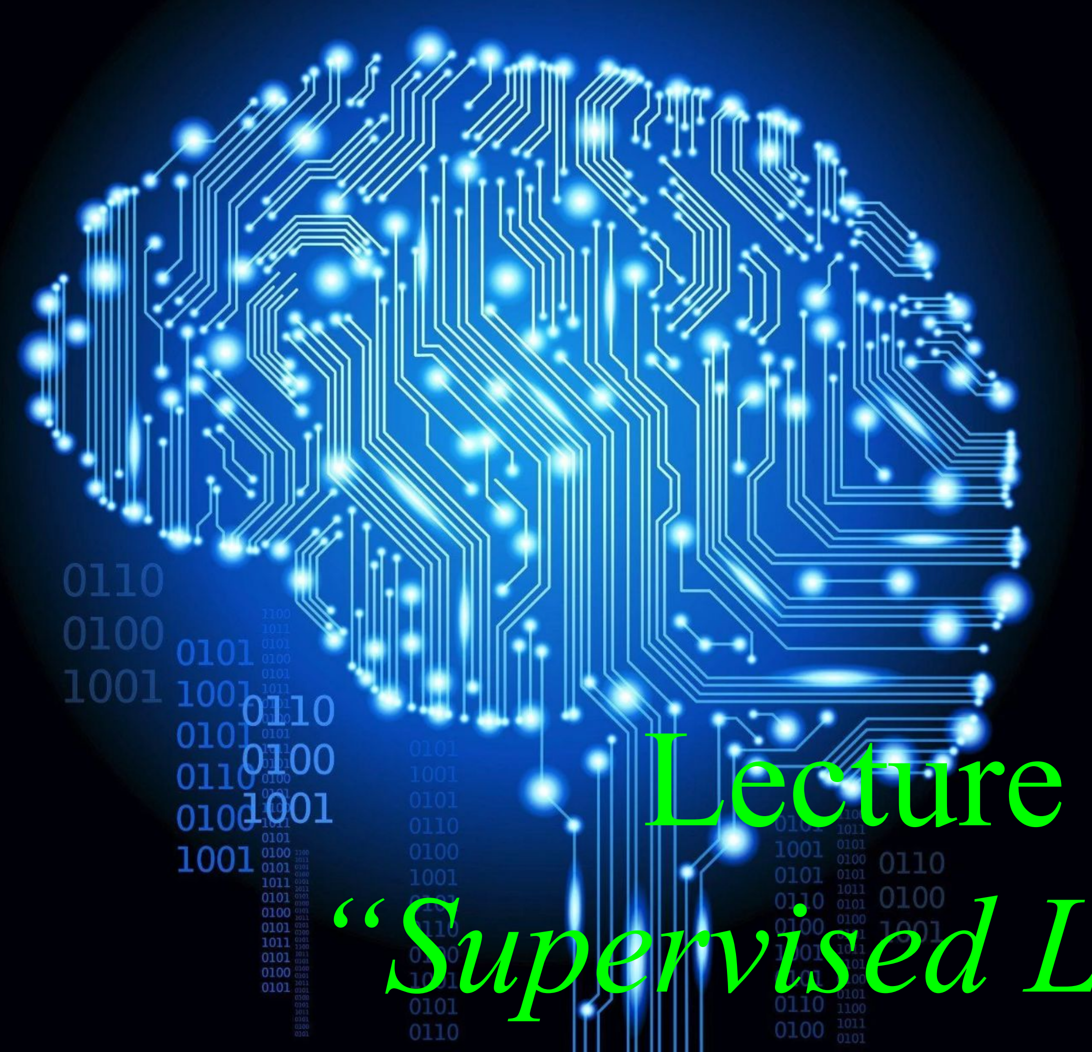


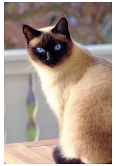
Lecture 01:
“*Supervised Learning*”

Supervised Learning

Training Input		Training Output
x = 6.1	→	y = 11.2
x = 9.2	→	y = 17.4

...

Test Input		Test Output
x = 8.5	→	y = ?

Training Input		Training Output
	→	“cat”
	→	“dog”

...

Test Input		Test Output
	→	?

3 steps:

1 Choose a form for the function which relates inputs (x) to output (y)

2 Define training loss

3 Find function in a form which gives the smallest training loss

1

Choose a form for the function which relates inputs (x) to output (y)

$$\hat{y} = b_0 + b_1 x$$

$$\hat{y} = b_0 + b_1 x + b_2 x^2$$

$$\hat{y} = f(x; \theta)$$

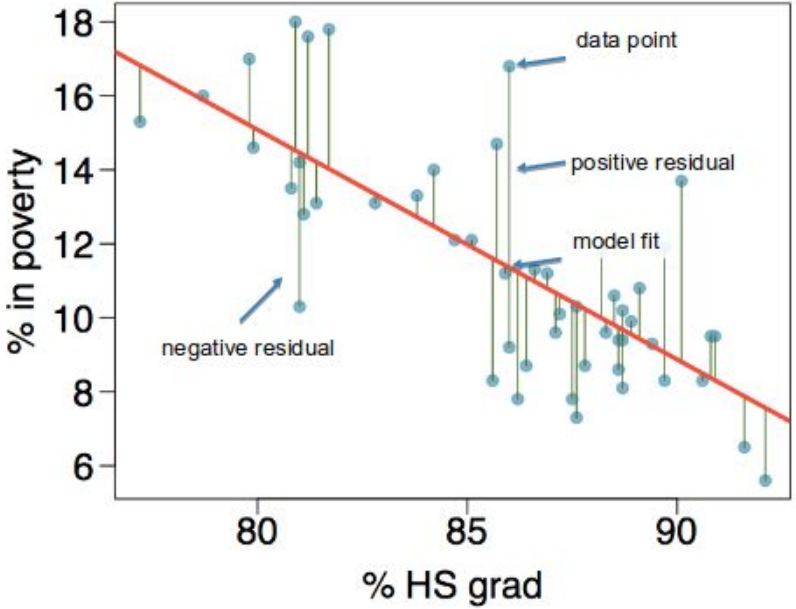
predicted y

input x

parameters: these specify which function in the class we are using.

2 Define training loss

- A **loss function** measures the deviation of a model's fit from observed data
- We minimize our loss function by tweaking the parameters, given training data



- Residuals are the errors from the model fit
- Data = Fit + Residual

We want a line with small residuals…

Option 1: Minimize the sum of magnitudes (absolute values) of residuals: **The L_1-norm** (also called LAD or Least Absolute Deviation)

$$L(\theta) = \sum_{i=1}^{n} \left| y_i - \widehat{y_i} \right| = \sum_{i=1}^{n} \left| r_i \right| = \|\mathbf{r}\|_1$$

Option 2: Minimize the sum of squared residuals: **The squared L_2-norm** (also called OLS or Ordinary Least Squares)

$$L(\theta) = \sum_{i=1}^{n} \left(y_i - \widehat{y_i} \right)^2 = \sum_{i=1}^{n} r_i^2 = \|\mathbf{r}\|_2^2$$

The most commonly used is **least squares**

- Solutions can be easily computed
- Big errors count relatively more than small errors

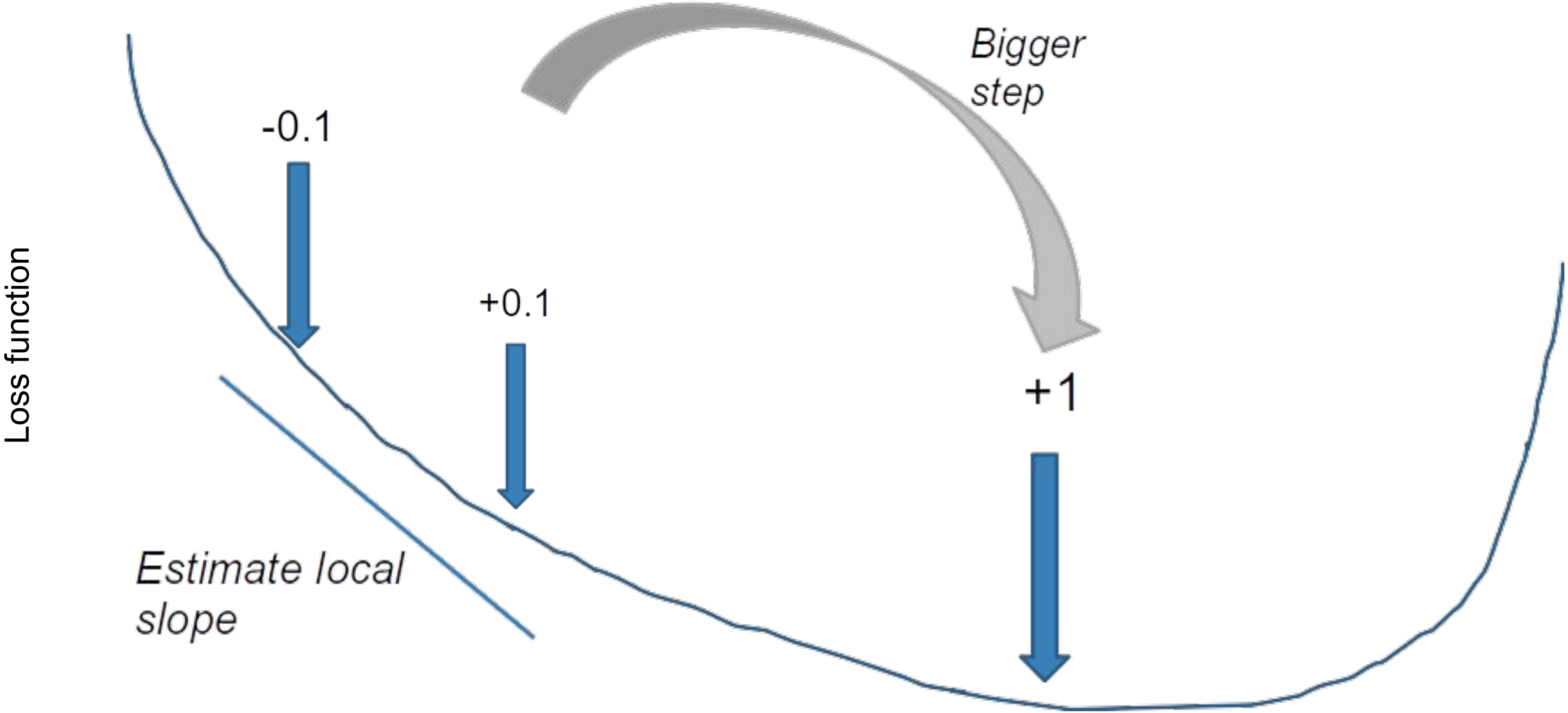
3 Find function in a form which gives the smallest training loss

- This is equivalent to finding the ***best parameters***.
- We want to minimize the training loss by trying different parameter values.

$$y = f(x;\theta) \longrightarrow \hat{y} = b_0 + b_1 x$$

$$b_0 = ?,\ b_1 = ? \longleftarrow L(b_0, b_1) = ?$$

Minimizing the cost



Derivative of Loss

Why? To speed up optimization (instead of guess and iterate)

$$L = \sum_{i=1}^{n} \left(y_i - b_0 - b_1 x_i \right)^2$$

$$\frac{\partial L}{\partial b_0} = -2 \sum_{i=1}^{n} \left(y_i - b_0 - b_1 x_1 \right)$$

$$= -2 \sum_{i=1}^{n} r_i$$

$$\frac{\partial L}{\partial b_1} = -2 \sum_{i=1}^{n} \left(y_i - b_0 - b_1 x_i \right) x_i$$

$$= -2 \sum_{i=1}^{n} r_i x_i$$

Derivative of Loss (L_1)

$$L = \sum_{i=1}^{n} \left| y_i - b_0 - b_1 x_i \right|$$

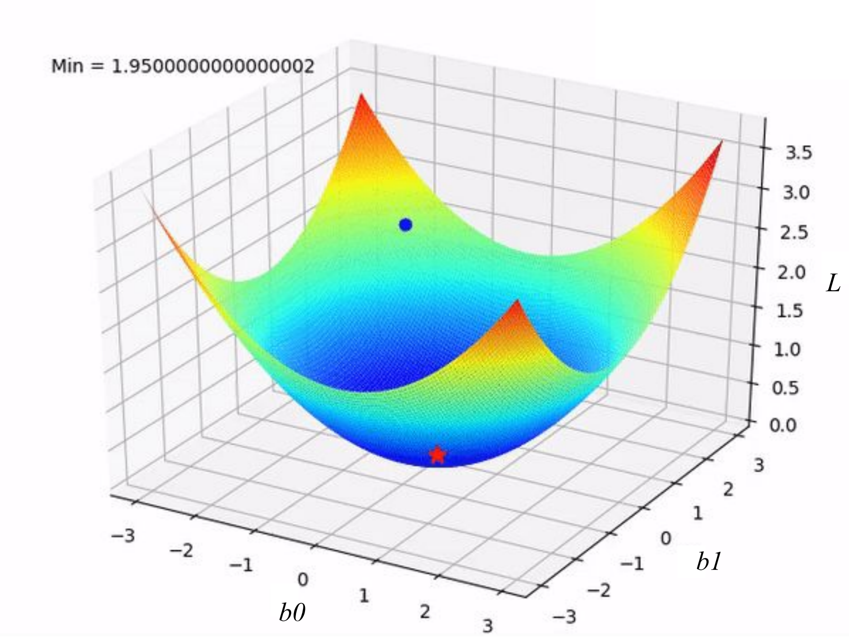
$$\frac{\partial L}{\partial b_0} = - \sum_{i=1}^{n} \operatorname{sgn}\left(y_i - b_0 - b_1 x_1 \right)$$

$$= - \sum_{i=1}^{n} \operatorname{sgn}\left(r_i \right)$$

$$\frac{\partial L}{\partial b_1} = - \sum_{i=1}^{n} \operatorname{sgn}\left(y_i - b_0 - b_1 x_i \right) \cdot x_i$$

$$= - \sum_{i=1}^{n} \operatorname{sgn}\left(r_i \right) \cdot x_i$$

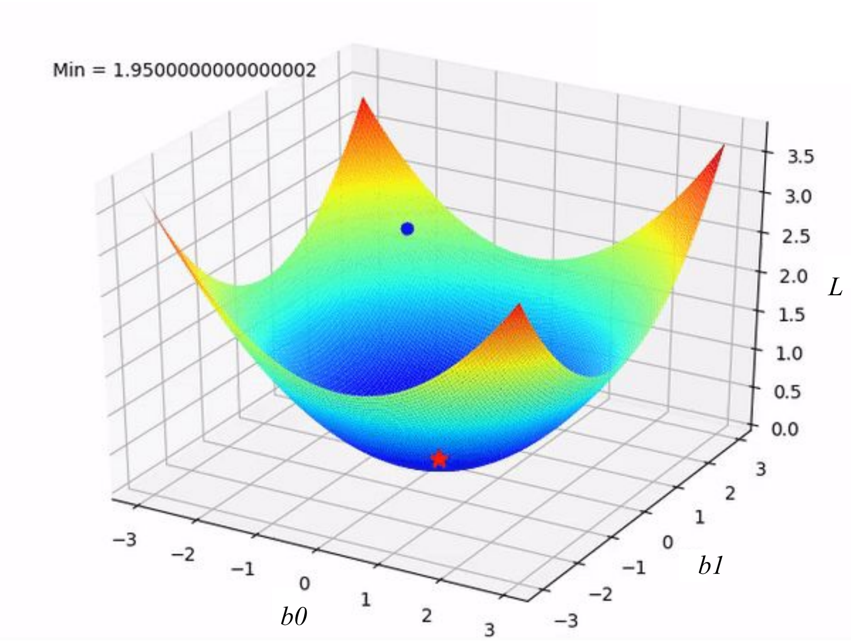
Derivative of loss (vectors)



$$\nabla_b L = \begin{bmatrix} \dfrac{\partial L}{\partial b_0} \\ \dfrac{\partial L}{\partial b_1} \end{bmatrix} = \begin{bmatrix} -2\sum_{i=1}^{n} r_i \\ -2\sum_{i=1}^{n} r_i x_i \end{bmatrix}$$

This is called the **Gradient** or **Jacobian**

Derivative of loss (vectors)



In vector notation...

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{b} \rightarrow \begin{bmatrix} \widehat{y1} \\ \widehat{y2} \\ \dots \\ \widehat{y}_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \dots & \dots \\ 1 & x_n \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$$

(Residual) $\mathbf{r} = \mathbf{y} - \hat{\mathbf{y}}$

(Loss) $L = (\mathbf{y} - \mathbf{X}\mathbf{b})^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\mathbf{b})$

(Gradient) $\nabla_b J = -2\mathbf{X}^{\mathrm{T}}\mathbf{r}$

How good is the fit?

For linear regression we often use $\mathbf{R^2}$: the **coefficient of determination**

$$R^2 = 1 - \frac{RSS}{TSS}$$

Residual Sum of Squares

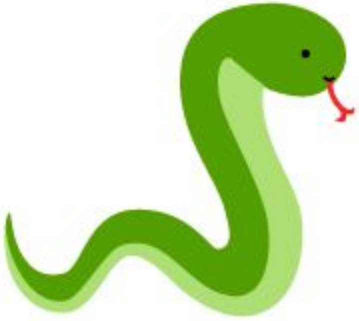
Total Sum of Squares

$$= 1 - \frac{\sum_{i=1}^{n} (y - \hat{y})^2}{\sum_{i=1}^{n} (y - \bar{y})^2}$$

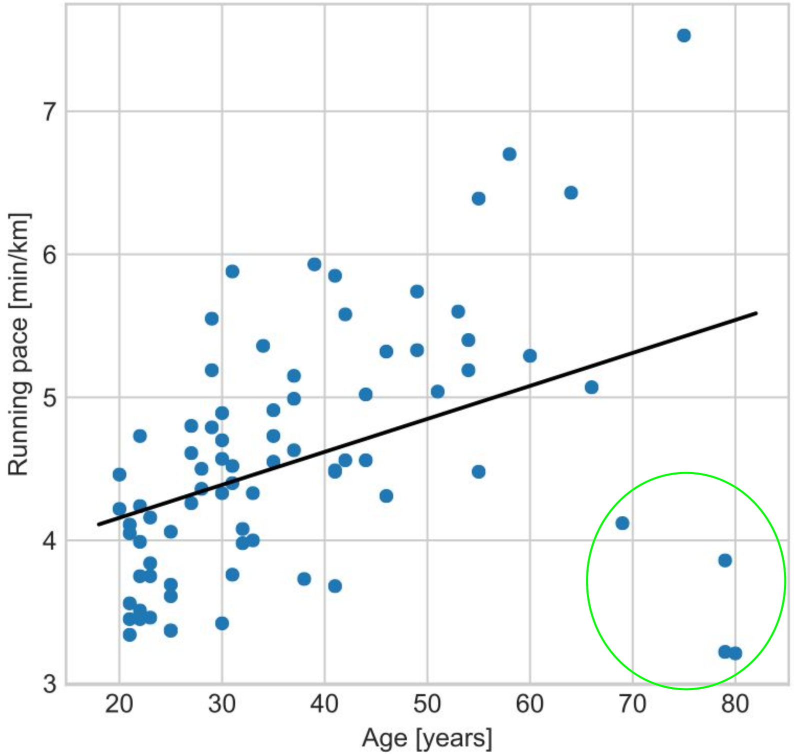
Mean value of the dataset

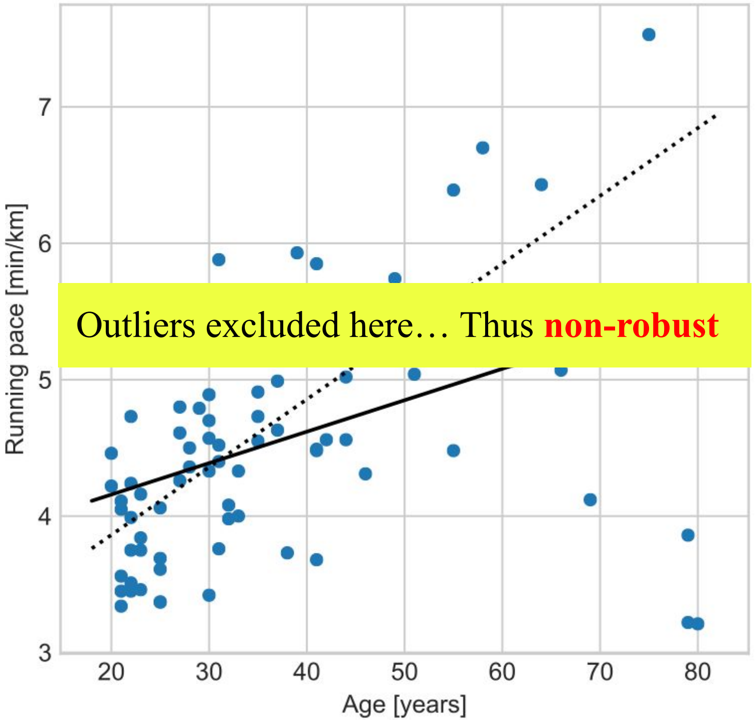
$R^2 = 0 \rightarrow$ no fit; $R^2 = 1 \rightarrow$ perfect fit. Note: OLS will always have the highest possible R^2 value.

Let’s try it in Python...



A Different Loss Function

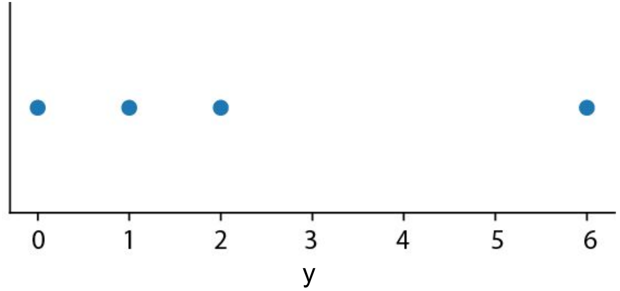


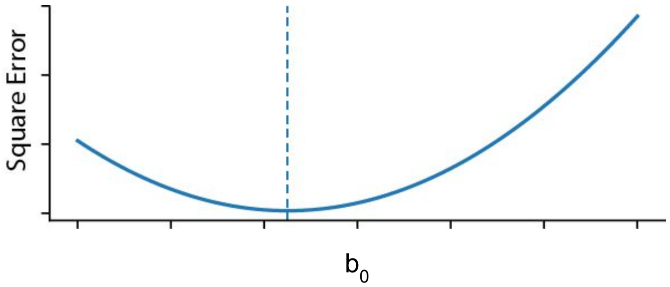


Robust regression

<u>Robust statistics</u>: provides methods which are not affected by (less sensitive to) outliers

- **Mean** (a measure of central tendency) is **sensitive** to outliers, **median** is **robust** to outliers





Where is the sum of squared error minimal?

Recall:

$$L = \sum_{i=1}^{n} \left(y_i - b_0 \right)^2$$

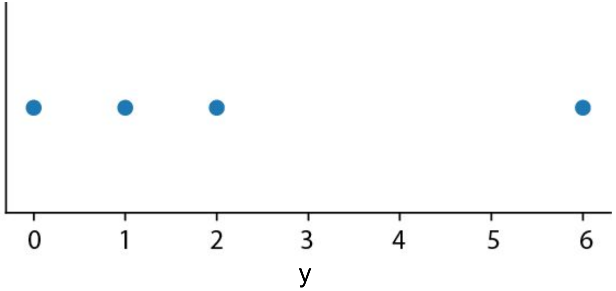
$$\frac{\partial L}{\partial b_0} = -2 \sum_{i=1}^{n} \left(y_i - b_0 \right)$$

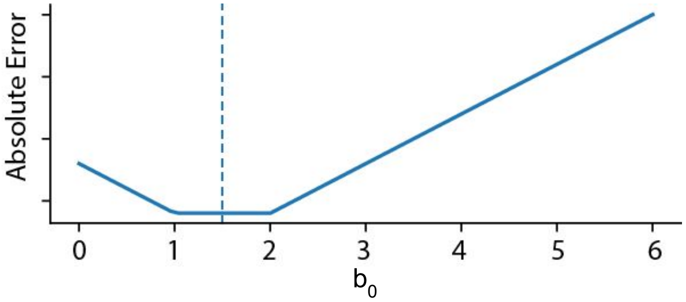
$$\frac{\partial L}{\partial b_0} = 0 \rightarrow \boxed{b_0 = \frac{\sum_{i=1}^{n} y_i}{n}}$$

Answer: **at the mean.**

Robust regression

Question: Can we change the loss function to improve its robustness to outliers? Let's consider minimizing the sum of absolute errors.





Where is the sum of absolute errors minimal?

$$L = \sum_{i=1}^{n} \begin{cases} y_i - b_0 & \text{if } y_i > b_0 \\ -(y_i - b_0) & \text{if } y_i \leq b_0 \end{cases}$$

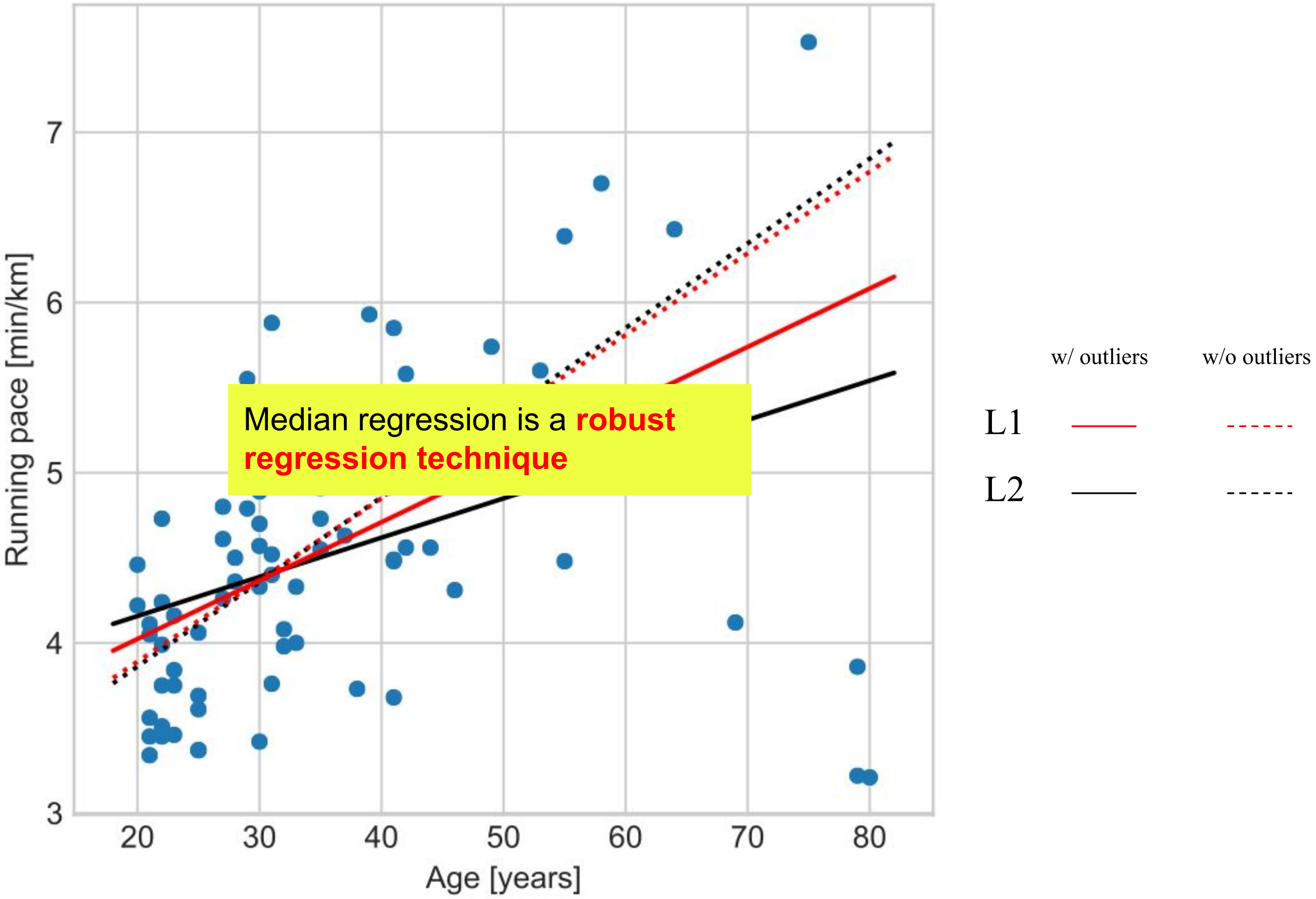
$$\frac{\partial L}{\partial b_0} = \sum_{i=1}^{n} \begin{cases} -1 & \text{if } y_i > b_0 \\ 1 & \text{if } y_i \leq b_0 \end{cases}$$

Recall:

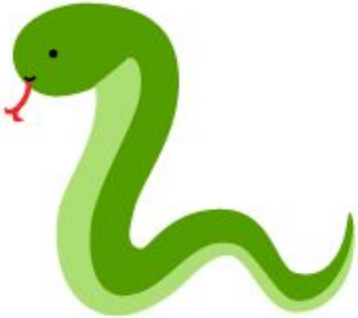
$$L = \sum_{i=1}^{n} \left| (y_i - b_0) \right|$$

Answer: **at the median**.

Median regression is a **robust regression technique**



Let’s try it in Python...



Summary

- **Models** can be written as $\hat{y} = f(x;\theta)$
- A **(training) loss function** measures how good or bad a fit is

  i.e. $L(\theta) = \sum_{i=1}^{n} \left(y_i - \hat{y}_i \right)^2$
- **Recall 3 steps for supervised learning**:
  1) Choose a function form
  2) Choose a loss function
  3) Find the function form (i.e. parameters) which minimizes said loss