Week 1
Supervised Learning and Regression

Week 1.1
Models and Parameters

Supervised Learning

x: inputs y: outputs

Training data:
(x = 1, y = 1)
(x = 2, y = 4)
(x = 3, y = 9)

Test data:
(x = 4, y = ?)

estimate the relationship.

Supervised Learning

Training data:

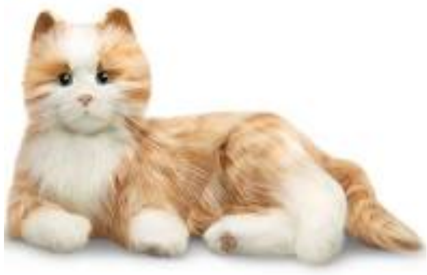
input and output could be anything.

(x = , y = 'cat')

(x = , y = 'dog')

(x = , y = 'cat')

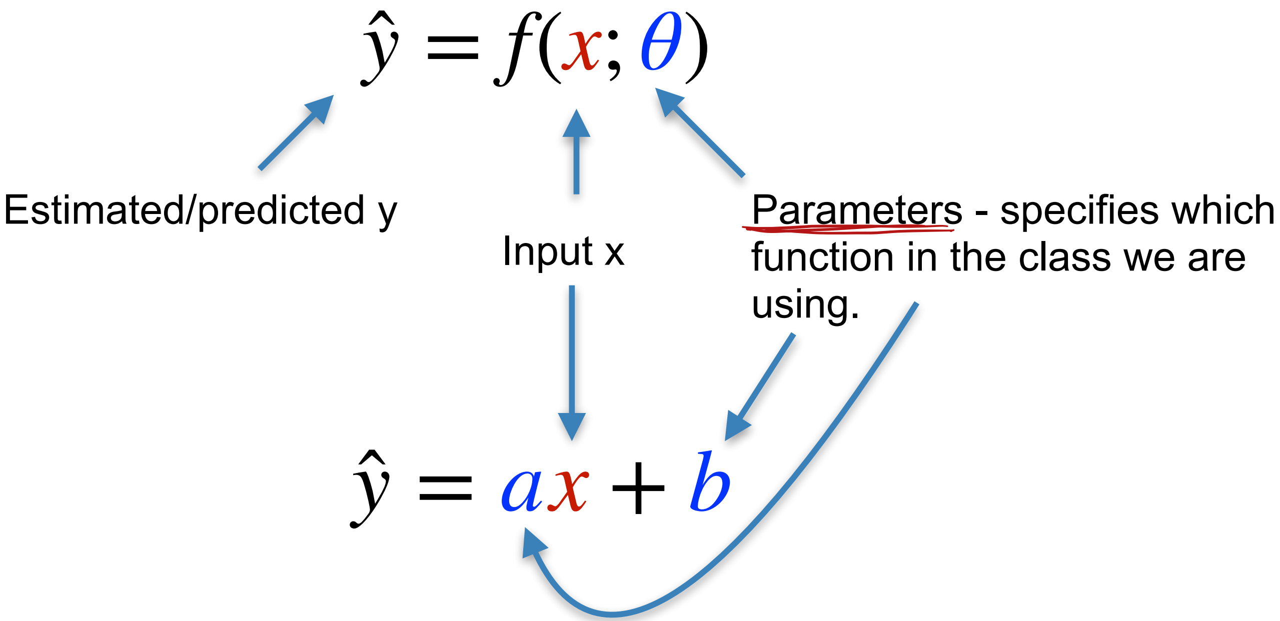
Test Data:

(x =  , y = ?)

Models

In ML, a Model is a function that relates some inputs (x) to some outputs (y)

Most models have parameters (θ), which allows them to represent whole classes of functions.



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

Estimated/predicted y

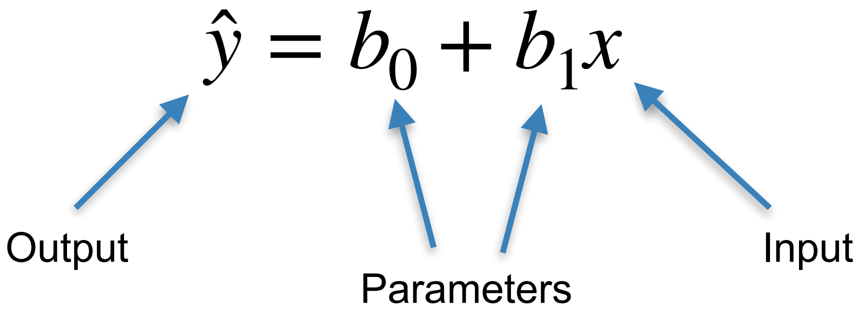
Input x

Parameters - specifies which function in the class we are using.

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

$$\hat{y} = ax^2 + bx + c \qquad \hat{y} = a\log(x) + b \qquad \dots$$

Linear regression

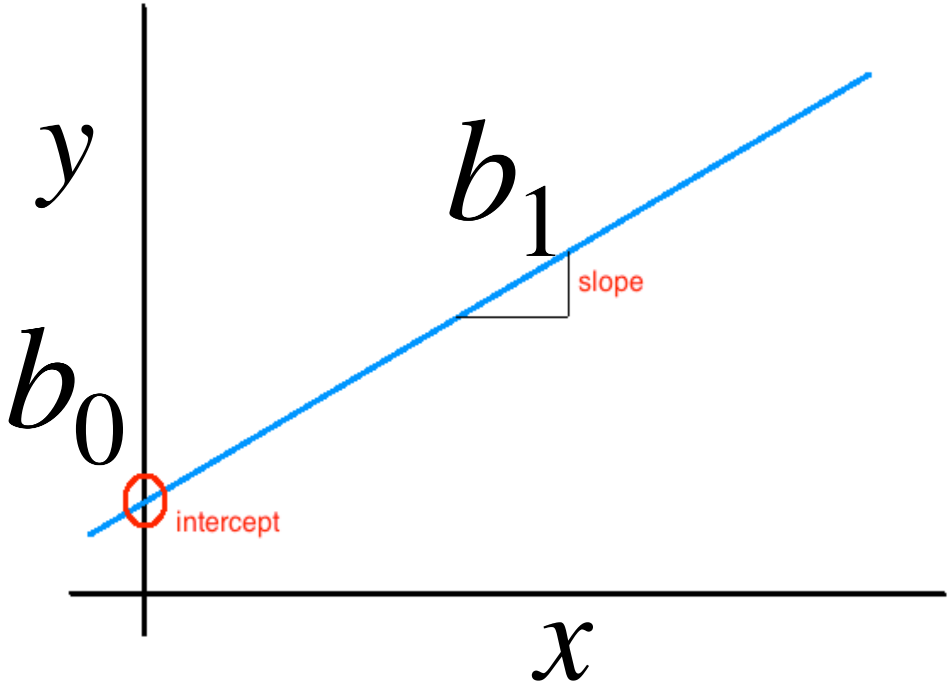


$$\hat{y} = b_0 + b_1 x$$

Output

Parameters

Input



Fitting a Model

1. Define the model: Choose the “class” of functions that relates the inputs (x) to the output (y)

2. Define your **training loss**

3. **Find the function** in your class/form that gives the **smallest training loss**

what's a good function/parameter?

Week 1.2
Loss functions

Fitting a Model

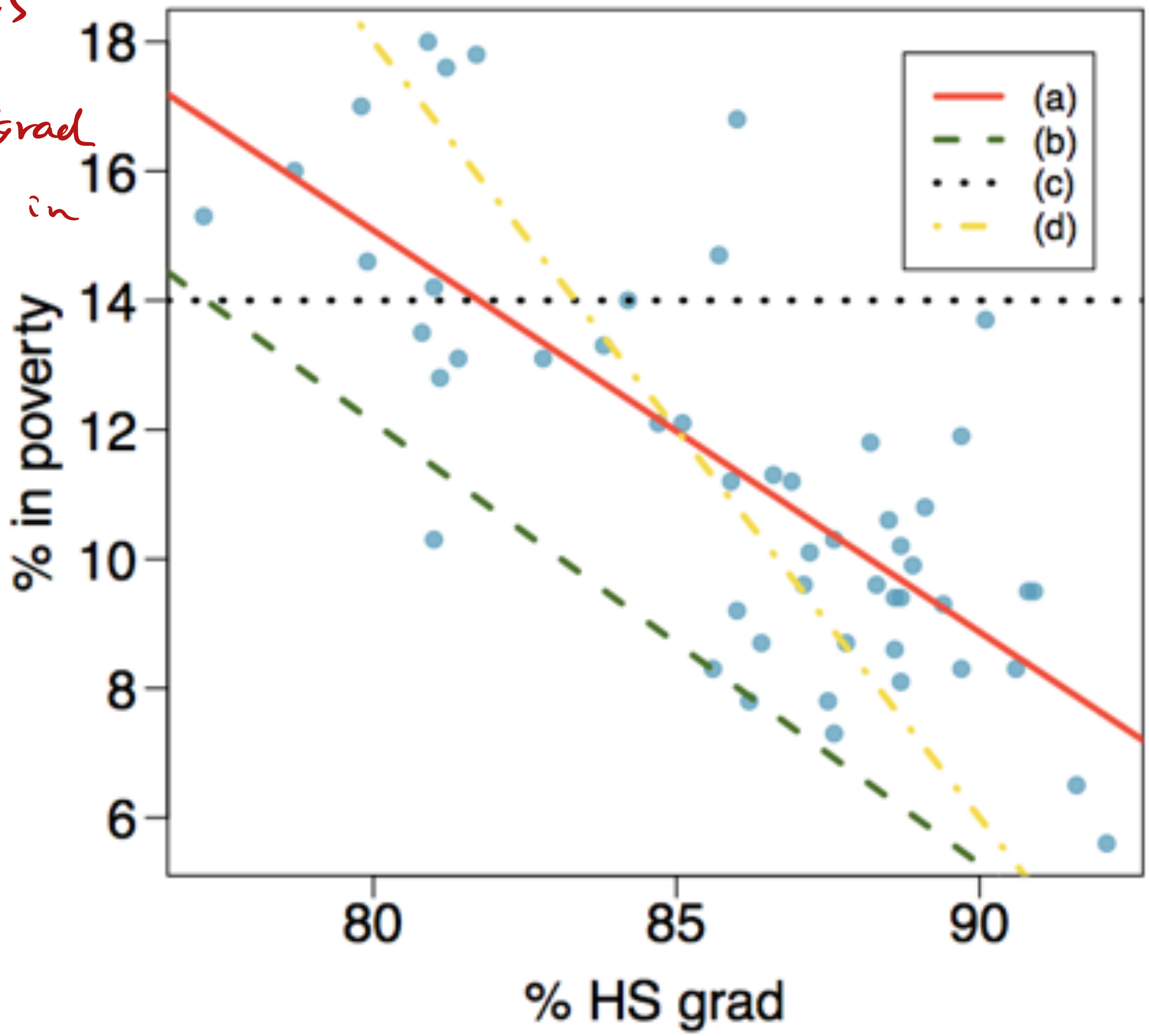
1. Define the model: Choose the “class” of functions that relates the inputs (x) to the output (y)

Applying the Model to Data

% HS Grad.

% in poverty -> $\hat{y} = b_0 + b_1 x$ ↙

* But there's no indicies that %HS Grad lead to % in poverty.



which one is the best ?

We need a standard

Training loss function

- A **training loss function** measures the deviation of the model fits from the observed data

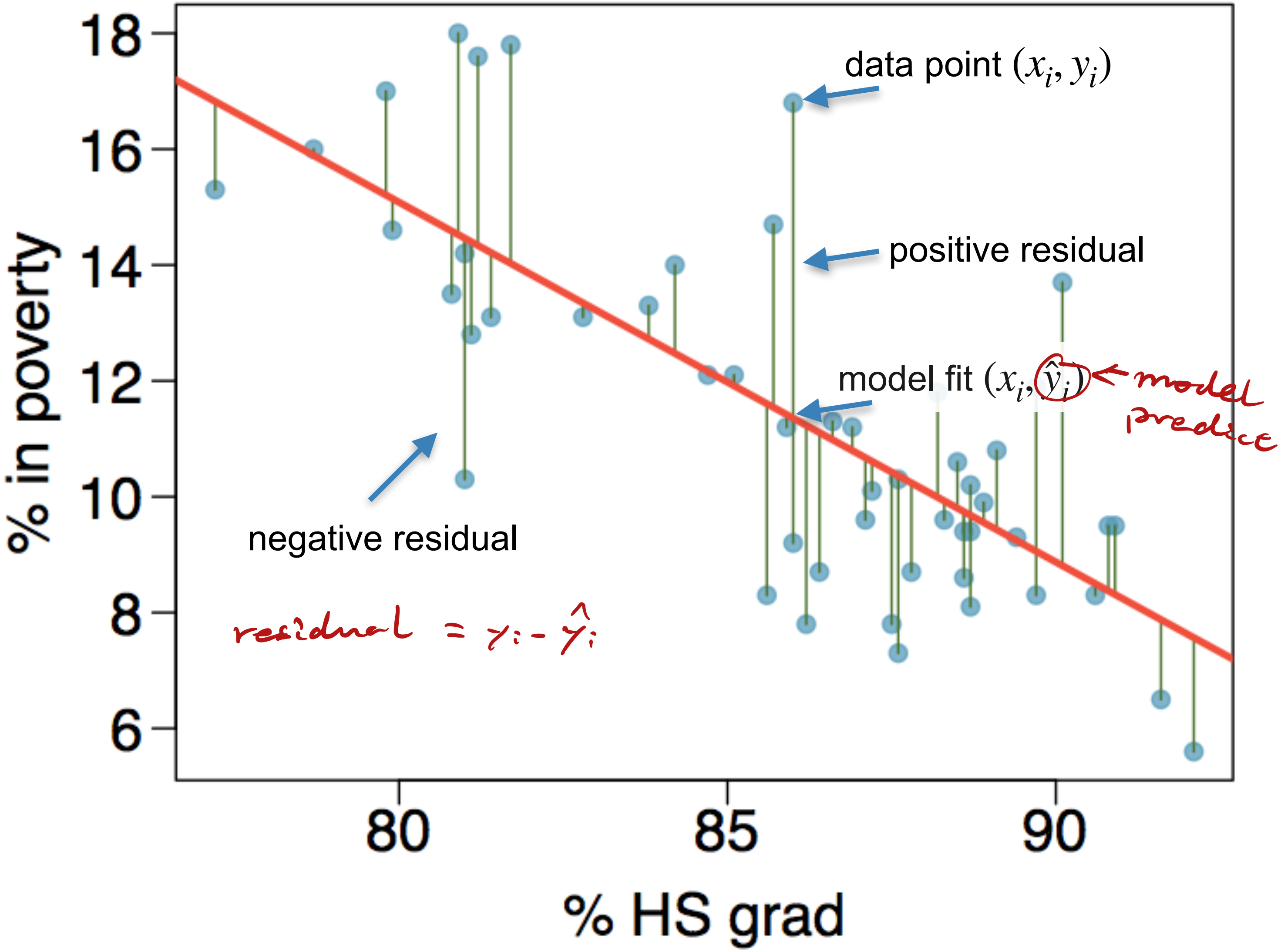
- A large loss indicates a poor fit to the training data

- Different loss functions penalize different deviations differently

- We find the **parameters** that **minimize** our loss function given the **training data**

Residuals

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



data point (x_i, y_i)

positive residual

model fit $(x_i, \hat{y}_i)$ ← model predict.

negative residual

residual $= y_i - \hat{y}_i$

A criterion for the best line

- We want a line that has small residuals
  1. Option 1: Minimize the sum of magnitudes (absolute values) of residuals: The L_1-norm

     $$L(\theta) = \sum_{i=1}^{n} |y_i - \hat{y}_i| = \sum_{i=1}^{n} |r_i| = \|\mathbf{r}\|_1$$

     LAD: Least Absolute Deviation (Median regression).

  2. Option 2: Minimize the sum of squared residuals: The squared L_2-norm

     $$L(\theta) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} r_i^2 = \underline{\|\mathbf{r}\|_2^2}$$

     it could be written in vector form.

     OLS: Ordinary Least Squares

- The most commonly used is least squares
  1. Motivated by normal distribution of errors
  2. Solutions can be easily computed
  3. Big errors count relatively more than small errors

Week 1.3
Optimization
Finding the best model fit

Optimization

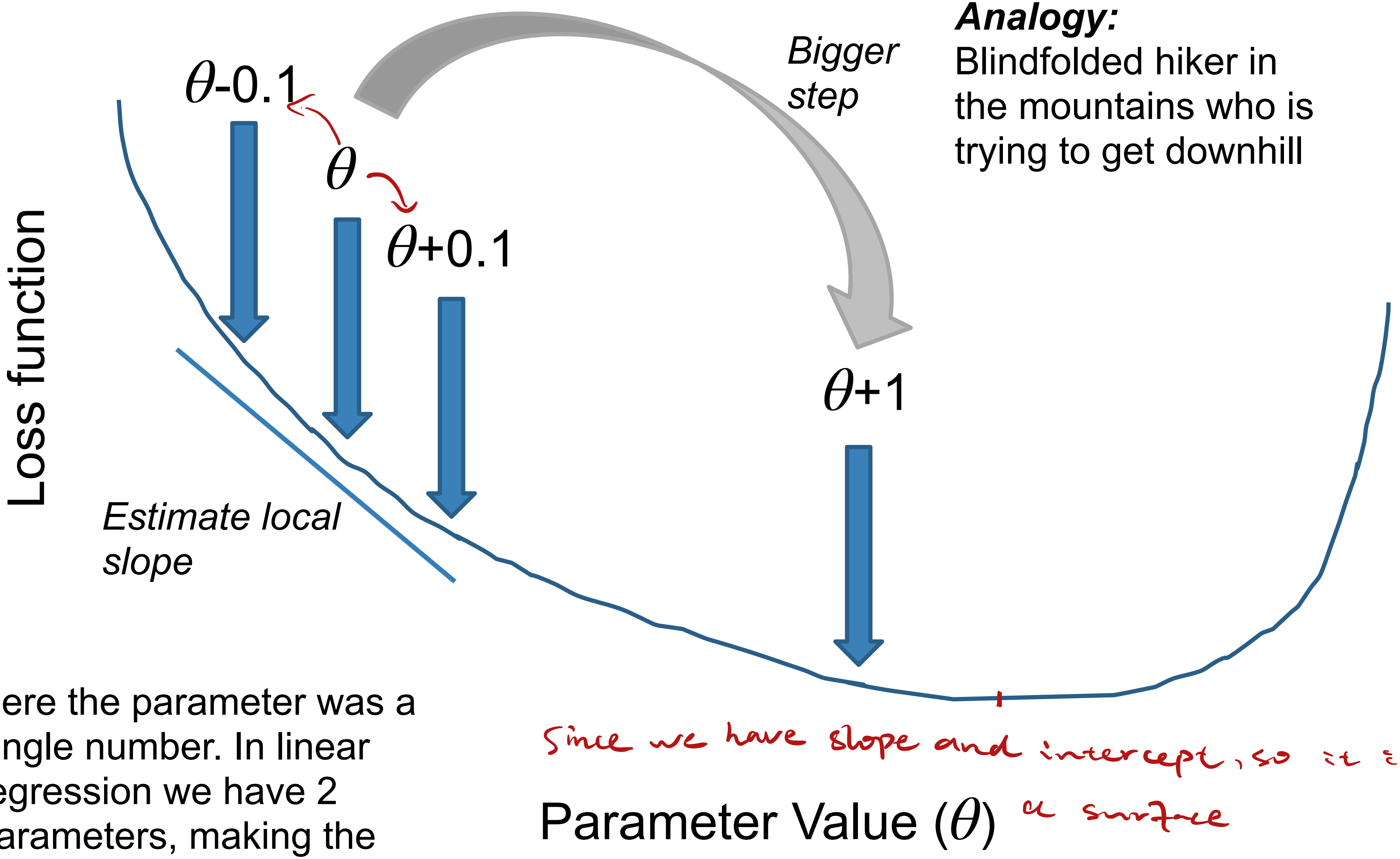
- **Finding the best *function* is the same problem as finding the best *parameters.***

- Parameter estimation is the process of minimizing the **training loss** by trying different values of the parameters

- The setting of parameters that gives you the smallest loss is the best estimate of the parameters, and the best function in your class

Optimization

Analogy:
Blindfolded hiker in the mountains who is trying to get downhill



Since we have slope and intercept, so it is a surface

Here the parameter was a single number. In linear regression we have 2 parameters, making the loss-function a surface

Using the derivative of the loss

By providing the derivative of the loss function in respect to the parameters, optimization can be sped up.

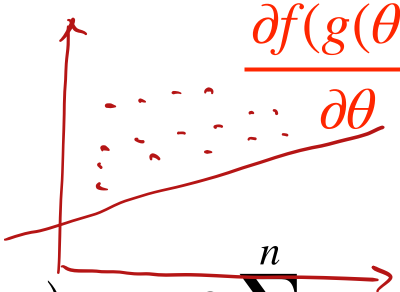
Fit: $\hat{y}_i = b_0 + b_1 x_i$

Residual: $r_i = y_i - \hat{y}_i$

$$\frac{\partial \sum f_i(\theta)}{\partial \theta} = \sum \frac{\partial f_i(\theta)}{\partial \theta}$$

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

$$\frac{\partial f(g(\theta))}{\partial \theta} = \frac{\partial f(g)}{\partial g} \frac{\partial g(\theta)}{\partial \theta}$$



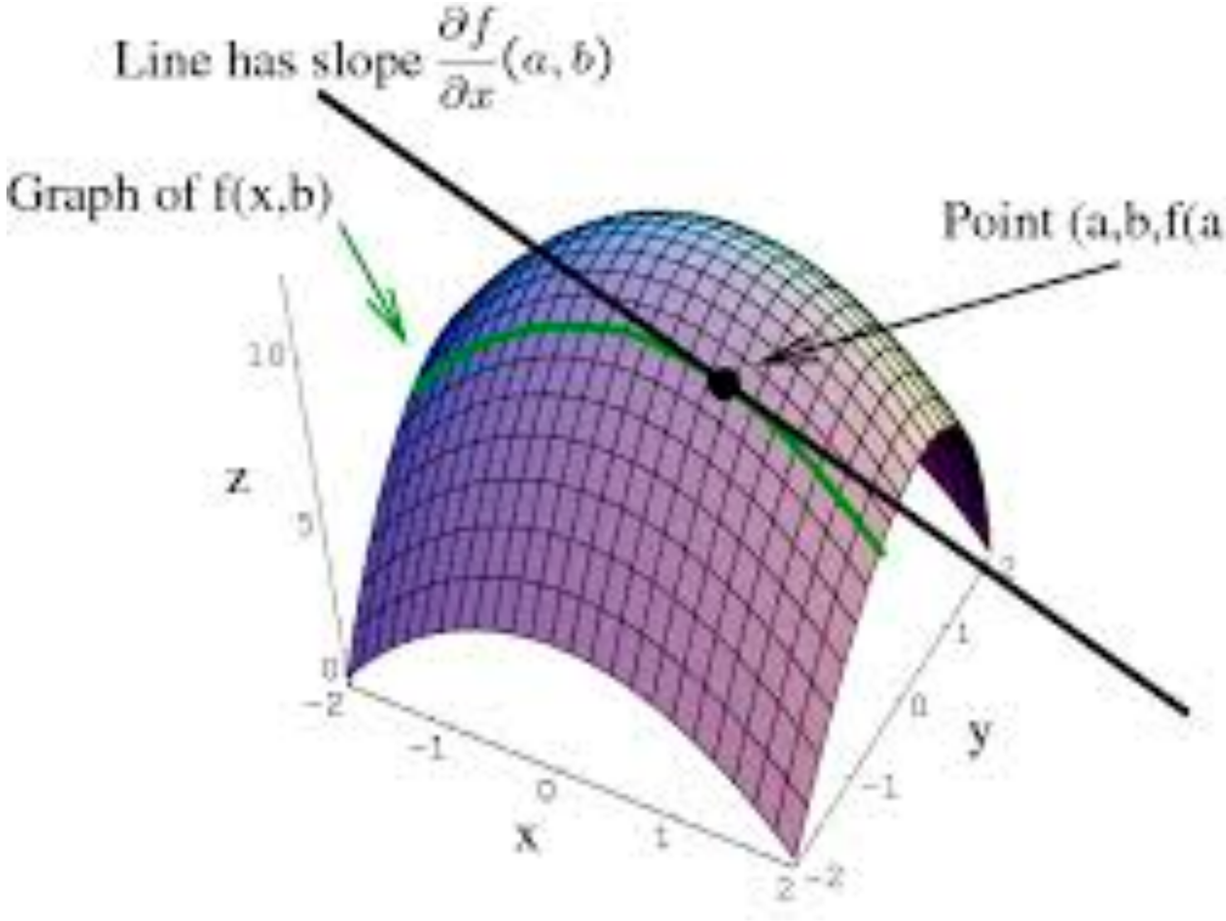
Derivative b$_0$: $\frac{\partial L}{\partial b_0} = -2 \sum_{i=1}^{n} \left(y_i - b_0 - b_1 x_i\right) = \underline{-2 \sum_{i=1}^{n} r_i} \Rightarrow$

Derivative b$_1$: $\frac{\partial L}{\partial b_1} = -2 \sum_{i=1}^{n} \left(y_i - b_0 - b_1 x_i\right) x_i = \underline{-2 \sum_{i=1}^{n} r_i x_i}$

Assuming that all residual are positive, that is, the intercept is too low: For larger value of intercept (b0), the residual would go down

Using the derivative of the loss

The vector of partial derivatives is called a *gradient* or *Jacobian*.



plop it to a vector.

$$\nabla_\theta L = \begin{bmatrix} \frac{\partial L}{\partial \theta_0} \\ \frac{\partial L}{\partial \theta_1} \end{bmatrix}$$

partial deritives of θ_0, θ_1

Derivatives of the loss

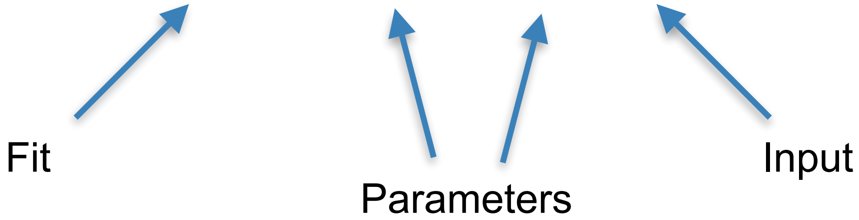
- Remember: Derivative = slope
- Also remember: want to make loss *small*

- If $\frac{\partial L}{\partial \theta}$ is positive, should I increase or decrease θ? ↓

- If $\frac{\partial L}{\partial \theta}$ is negative, should I increase or decrease θ? ↑

Linear regression in matrix notation

$$\hat{y}_i = b_0 + b_1 x_i$$



Fit

Parameters

Input

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$$

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{b}$$

For each point in the dataset, we get a fit/estimate/prediction from the model. Next, we'll compare those to the actual data.

Using the derivative (vector notation)

By providing the derivative of the loss function in respect to the parameters, optimization can be sped up.

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$$

Prediction: $\hat{\mathbf{y}} = \mathbf{X}\mathbf{b}$

Residual: $\mathbf{r} = \mathbf{y} - \hat{\mathbf{y}}$

Loss: $$L = \left(\mathbf{y} - \mathbf{X}\mathbf{b}\right)^{\mathsf{T}} \left(\mathbf{y} - \mathbf{X}\mathbf{b}\right)$$

$$= \mathbf{y}^{\mathsf{T}}\mathbf{y} - 2\mathbf{y}^{\mathsf{T}}\mathbf{X}\mathbf{b} + \mathbf{b}^{\mathsf{T}}\mathbf{X}^{\mathsf{T}}\mathbf{X}\mathbf{b}$$

Gradient: $\nabla_b J = -2\mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{X}\mathbf{b} = -2\mathbf{X}^T\mathbf{r}$

Week 1.4
Implementing OLS regression through minimization of L2 loss

Step 1: Write the model function

Write a function that returns the

vector

First input is parameter list or np.array

(param)

matrix

np.array (2d) of explanatory variables

(input)

```
def linearModelPredict(b,X):
    # Get Model prediction
    predY =
    # return Model prediction
    return predY
```

Step 2: Write a loss function

We are modifying the loss function to also return the gradient

First input is parameter list or np.array

Explanatory and response variable

```
def linearModelLossRSS(b,X,y):
    # Get Model prediction
    predY = linearModelPredict(b,X)
    # Get the vector of residuals
    res =
    # Get the residuals sums of squares
    rss =
    # Get the gradient
    gradient =
    # return rss and gradient
    return (rss,gradient)
```

← the actual data

partial derivative of the loss with given param; it should has same dimension as the input param. (b).

Step 3: Call the optimizer

```
import scipy.optimize as so
# Set some starting values
bstart=[0,0]
# Call the optimization function
RESULT=so.minimize(linearModelLossRSS,bstart,args=(X, y),jac=True)
```

; if your functions are correct, starting value does not affect the result, but the processing tim

Loss function

Starting value

Additional arguments

Use gradient

return is not only loss, but also gradien

this would speed up the optimization.

Remember our definition:

```
def linearModelLossRSS(b,X,y):
```

Step 4: Check the results

this jacobian here should be close to zero.

minimal loss value found.

jacobian

jacobian at the place of the best fitting parameter.

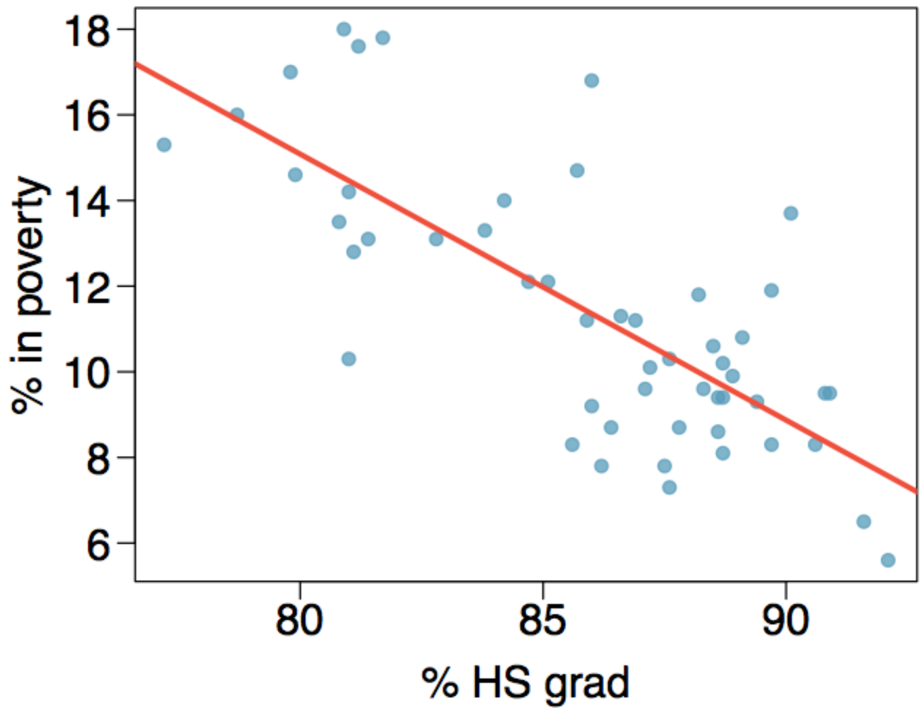
```
RESULT
fun: 14.25                # Final loss function value
hess_inv: array([[0.33,-0.07],[-0.07,0.01]])
jac: array([1.19-06, -1.90-06])
message:'Optimization terminated successfully.'
nfev: 24                  # Number of evaluations
nit: 3                    # Number of iterations
njev: 6
status: 0
success: True
x: array([65,-0.6]) # Parameter estimates
```

- Check if successful
- Get parameter estimates
- Maybe check the loss

Step 5: Visualize the results

```
b=RESULT.x      # Get the parameters
x_grid = np.linspace(y.min(), y.max(),10) # get grid
Xn = np.c_[np.ones(x_grid.size), x_grid] # Make Design
yp=linearModelPredict(b,X) # get prediction
ax.plot(x_grid, yp, color = 'red')
```

$$\widehat{\%\ in\ poverty} = 64.68 - 0.62\ \%\ HS\ grad$$



Week 1.5
Evaluating model fit - R^2

Evaluating the fit with R²

- The quality of the fit of a linear regression model is most commonly evaluated using *R*² the coefficient of determination.
- *R*² is calculated from the ratio of residual sum of squares – total sum of squares.

$$R^2 = 1 - \frac{\text{RSS}}{\text{TSS}} = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2}$$

Residual sum of squares (RSS)

Total sum of squares (TSS)

datapoint - mean of data.

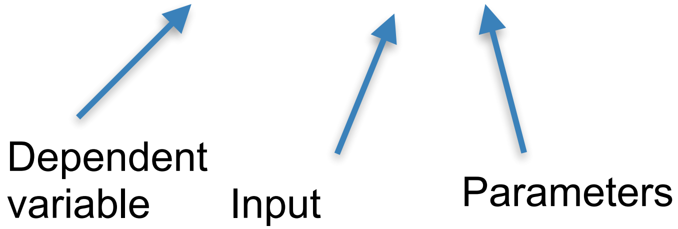
if it is not perfectly fit: 1) data is very noisy.
2) model is incompleted.

- It tells us what percent of variability in the response variable is explained by the model. (0=no fit, 1=perfect fit)
- The remainder of the variability is explained by variables not included in the model or by inherent randomness in the data.
- Because OLS is miming the RSS, it will always have the highest R² value possible for that class of models.

Summary

- **Models** can be written in general as:

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



Dependent variable

Input

Parameters

- The **training loss function** tells how bad a fit is:

Example: squared error

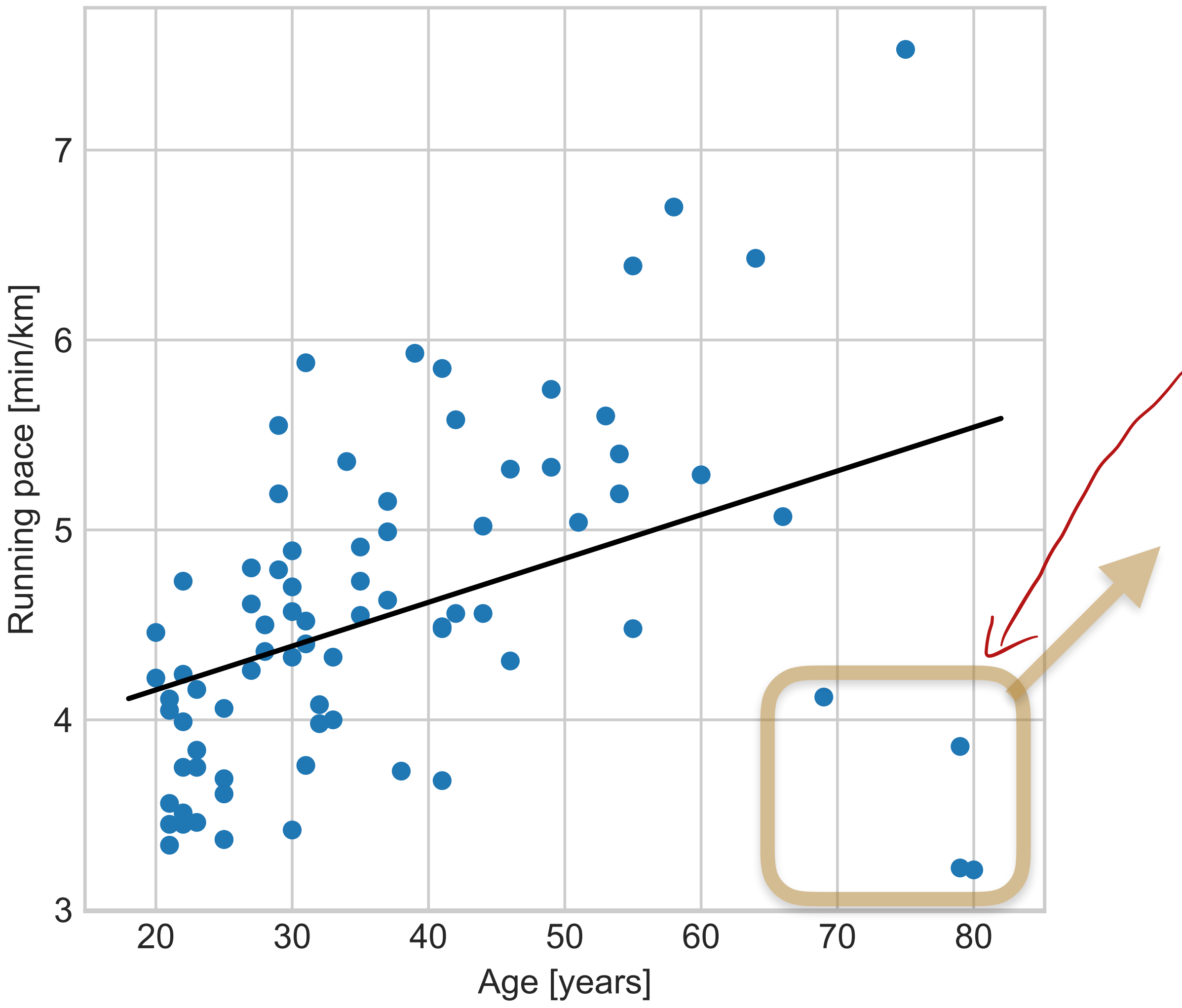
$$L = \sum_{i=1}^{n} \left(y_i - \hat{y}_i\right)^2$$

- **Model fitting** involves selecting the parameters that minimizes the loss function: **Parameter estimation**

- In linear regression, the **proportion of the explained variance** in the response variable is expressed by the **coefficient of determination** (**R**^{2})

Week 1.6
L1-loss and median regression
Robust techniques

The Impact of the Loss Function



Example of best running speeds for a 3km Strava segment.

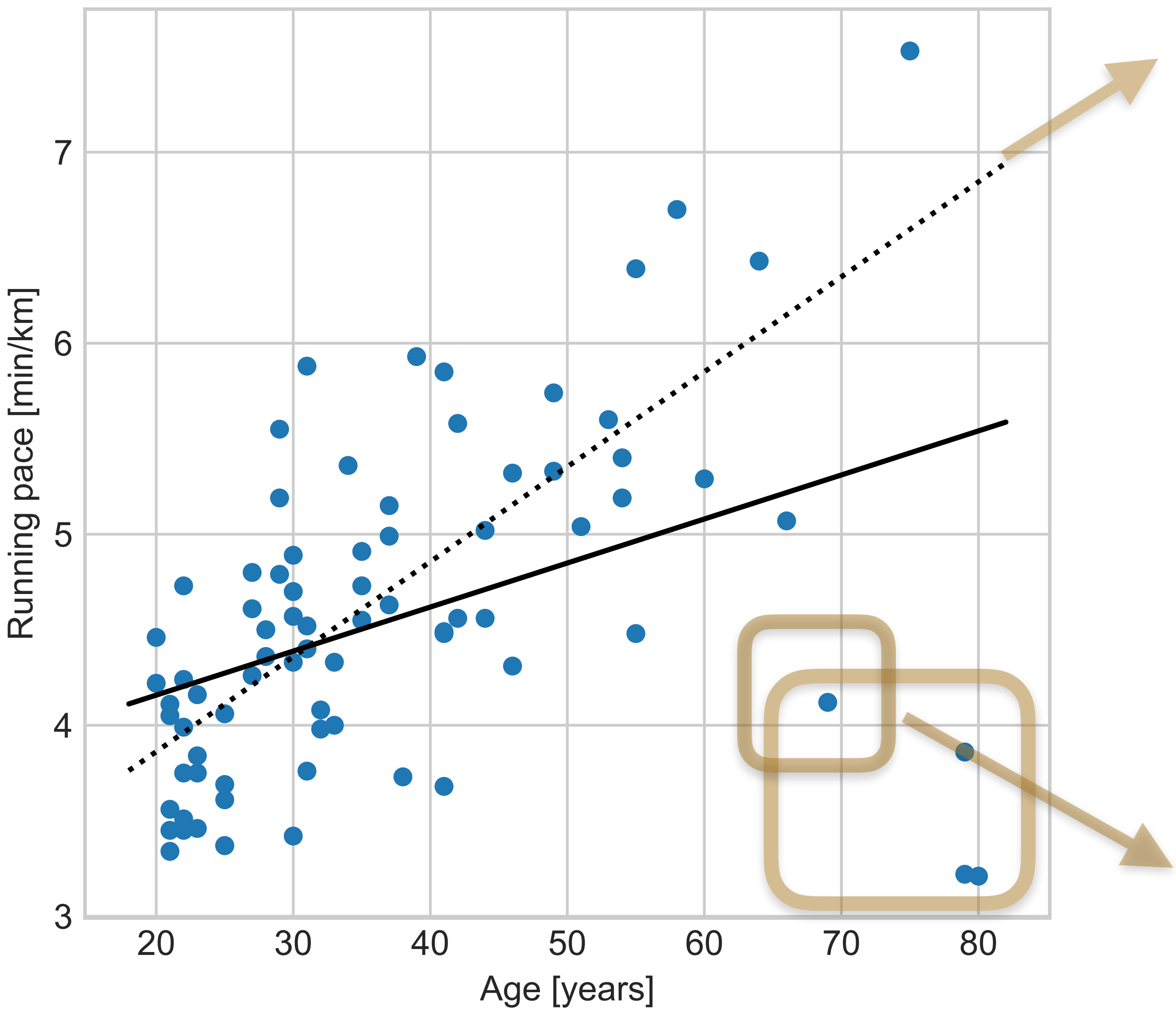
What do you notice?

Are these old guys for real?

Maybe these data points have other explanations:

- *old guys on bikes*
- *young runners lying about their age*
- *really good doping*
- ….

The Impact of the Loss Function



Excluding these “outliers” changes the predicted running speed for 80-year olds

Answers that change a lot with exclusion of a few data points are called **non-robust**.

It is not the most desired try.

So are we sure that this one is cheater as well?

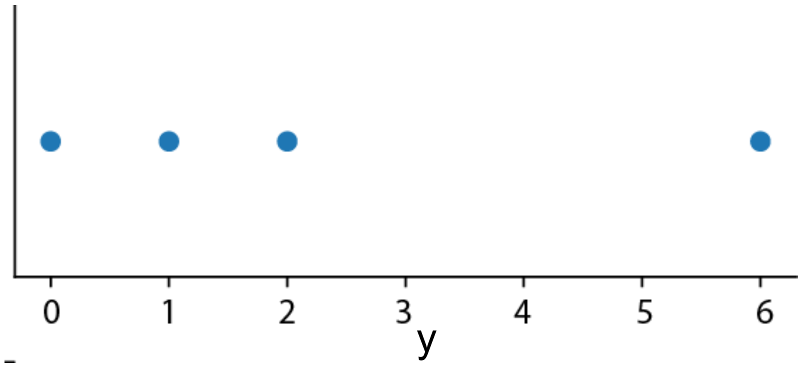
Robust statistics

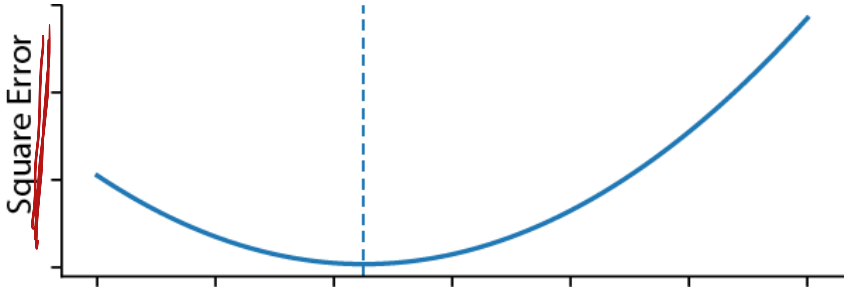
Robust statistics seek to provide methods that emulate popular statistical methods, but which are not unduly affected by outliers or other small departures from model assumptions.

- **Mean** is a measure of central tendency that is sensitive to outliers but it would be greatly affected by so extreme value.
- **Median** is a measure of central tendency that is robust against outliers

Robust regression

To develop a robust regression technique, we can think how to change the loss function.





b0 is our parameter

Say we have these 4 data points

If we find the point that minimizes the sum of squared error

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

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

minimize L2 loss.

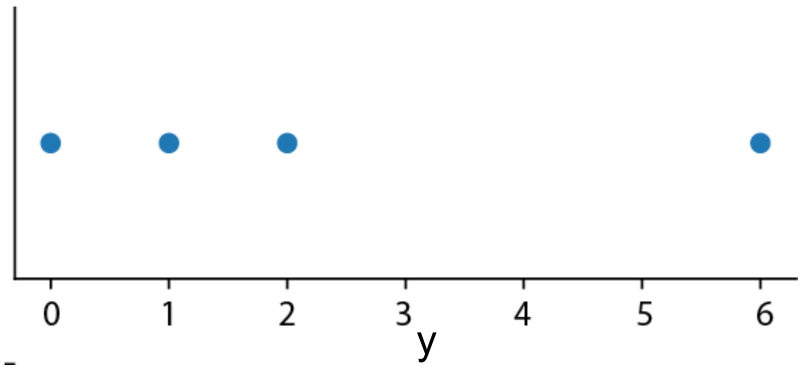
$$\boxed{\frac{\partial L}{\partial b_0} = 0} \Longrightarrow b_0 = \frac{\sum_{i=1}^{n} y_i}{n}$$

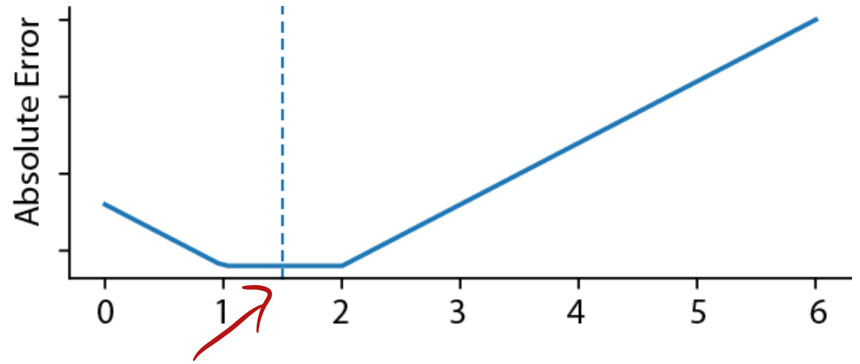
The minimum is reached at the **mean**

Robust regression

To develop a robust regression technique, we can think how to change the cost function.

Say we have these 4 data points





the minimal here is not only one value but a range

If we find the point that minimize the sum of **absolute** errors

L_1

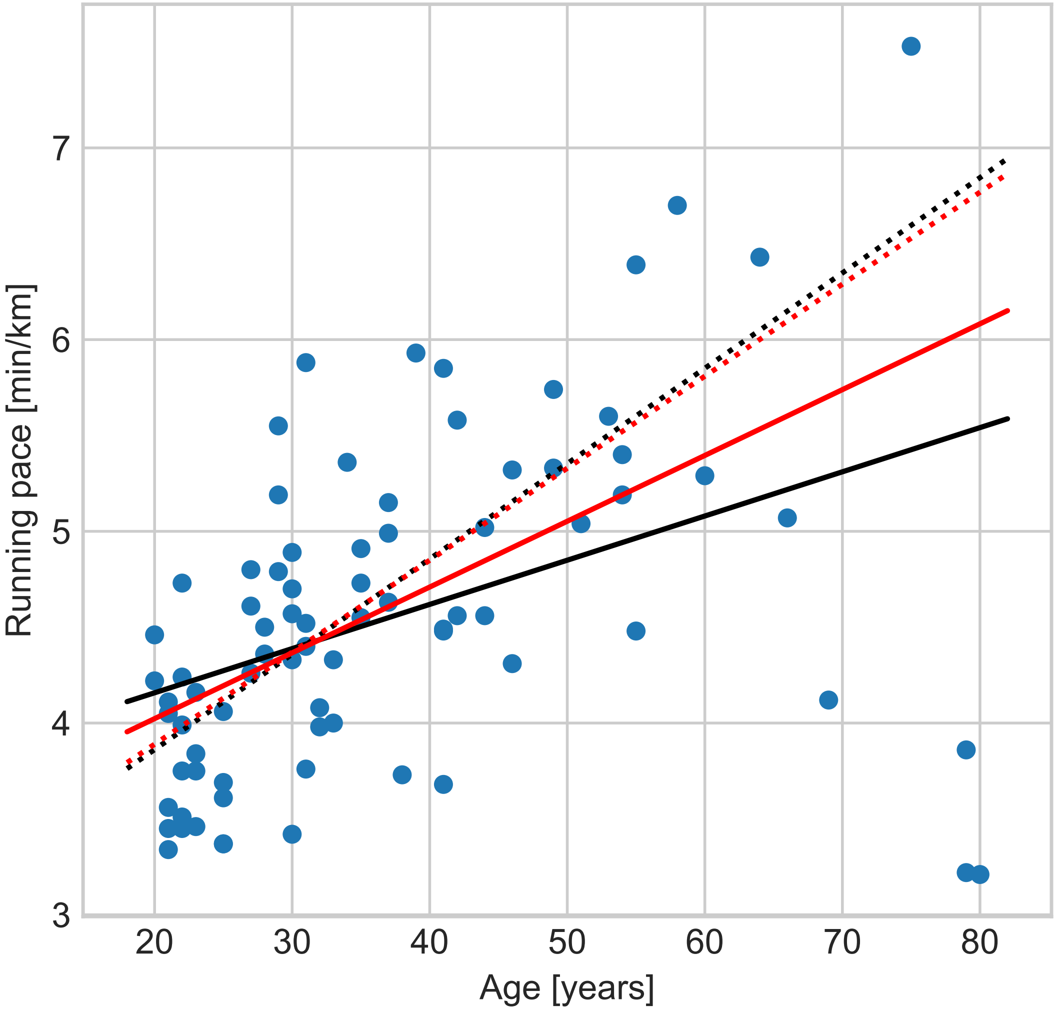
$$L = \sum_{i=1}^{n} |y_i - b_0|$$

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

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

The minimum is reached at the **median**

Outliers and robustness



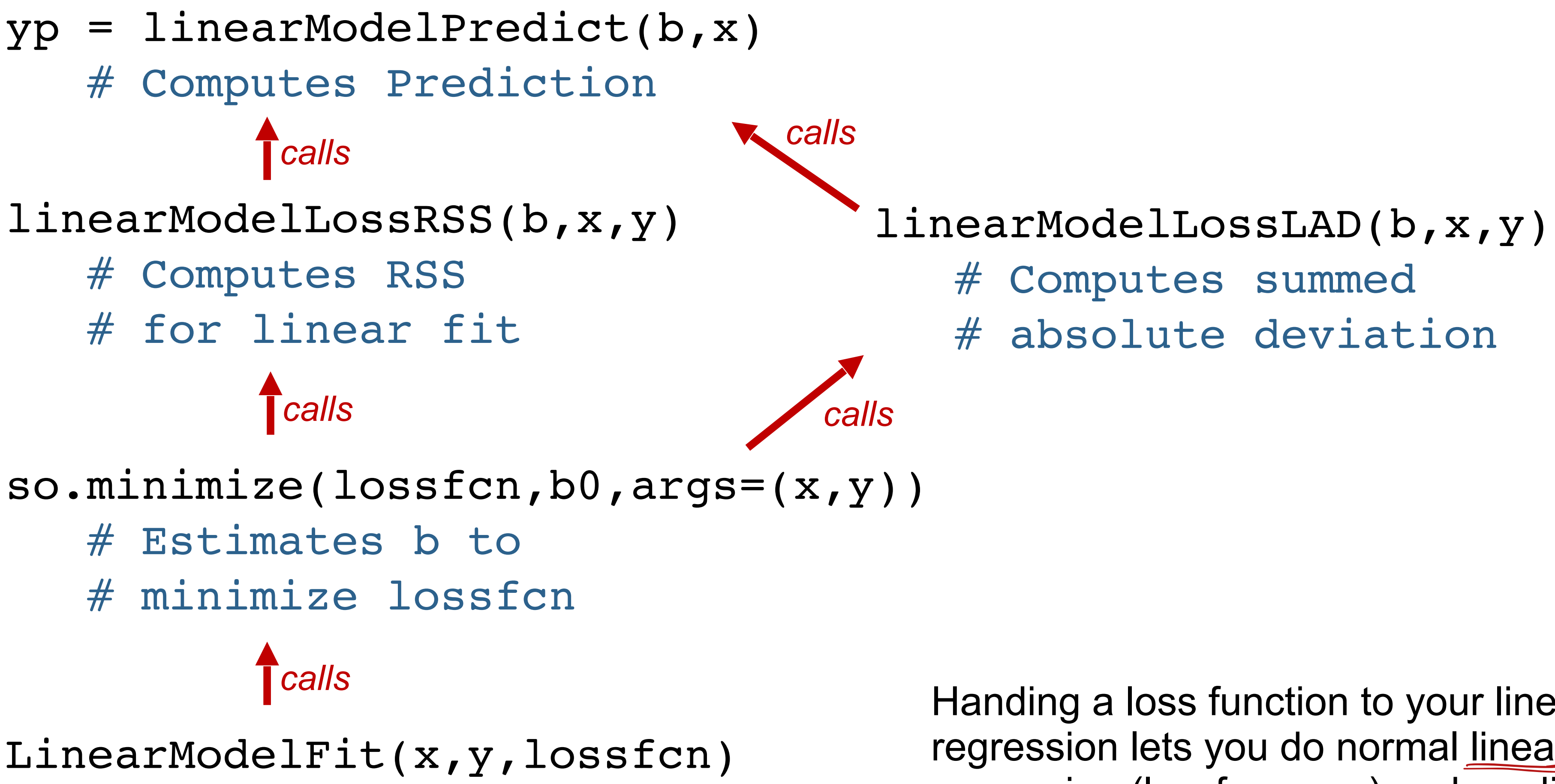
Median regression leads to results in a regression line that is closer to the one that excludes the outlier.

	all data	w/o outliers
RSS	━━━━	········
LAD	━━━━	········

Median regression is a robust regression technique

Week 1.7
Implementing Median regression through minimization of L1 loss

Implementing median regression



```
yp = linearModelPredict(b,x)
   # Computes Prediction
```

```
linearModelLossRSS(b,x,y)
   # Computes RSS
   # for linear fit
```

```
linearModelLossLAD(b,x,y)
   # Computes summed
   # absolute deviation
```

```
so.minimize(lossfcn,b0,args=(x,y))
   # Estimates b to
   # minimize lossfcn
```

```
LinearModelFit(x,y,lossfcn)
```

Handing a loss function to your linear regression lets you do normal linear regression (lossfcn = rss) and median regression (lossfcn = lad) with the same code

Using the derivative

Again, by providing the derivative of the training loss with respect to the parameters, we can make the fit faster.

Prediction: $\hat{y}_i = b_0 + b_1 x_i$

Residual: $r_i = y_i - \hat{y}_i$

Loss: $$L = \sum_{i=1}^{n} |y_i - (b_0 + b_1 x_i)|$$

sign

Derivative b_0: $$\frac{\partial L}{\partial b_0} = -\sum_{i=1}^{n} \text{sgn}(y_i - (b_0 + b_1 x_i)) = -\sum_{i=1}^{n} \text{sgn}(r_i)$$

Derivative b_1: $$\frac{\partial L}{\partial b_0} = -\sum_{i=1}^{n} \text{sgn}(y_i - (b_0 + b_1 x_i)) \cdot x_i = -\sum_{i=1}^{n} \text{sgn}(r_i) \cdot x_i$$

Parameter estimation

- “Parameter estimation” is the process of minimizing the **loss function** by trying different values of the parameters
- The Gradient can speed up optimization a lot! (But you can get by without it.)
- For nearly every problem there is a more specialized solution that is faster, for example Sklearn-methods usually have a fit (estimation) and predict (prediction) function built in.
- But using general minimization algorithms, such as `scipy.optimize.minimize`, is an incredibly useful and universal tool.

Summary

- We can fit mathematical models to capture the relationship between x and y

line / smooth curve / etc ...

1. Select a function class/form
2. Select a loss function <= OLS/LAD .
3. Estimation/ fitting: Find the function that minimizes the loss

- “supervised learning” is a cornerstone of statistics, machine learning, and data science.