
# CS 559: Machine Learning Fundamentals & Applications

## Lecture 1: Linear Regression


STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

# Outline

Linear Regression: Maximum Likelihood Estimator (MLE)

- Linear Model Assumptions
- MLE

Introduction to Machine Learning

Linear Regression, Machine Learning

- Sequential Learning
- Linear Regression with Scikit-learn (python)
- Linear Regression Assumptions
- NumPy Implementation

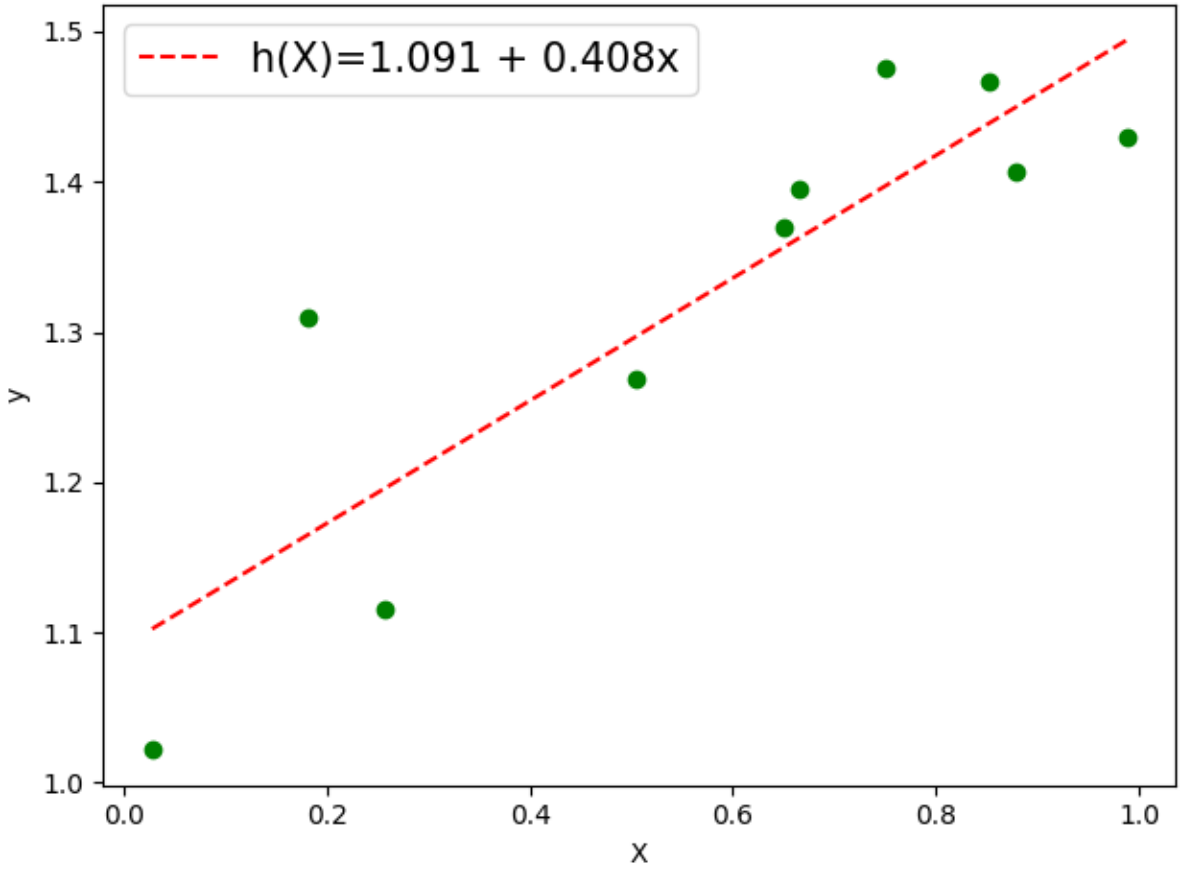
Conclusion


# Linear Regression: Maximum Likelihood Estimator (MLE)

# Linear Model

- Consider the data table below.
- We want to build a simple linear hypothesized model as $h(\mathbf{x}) = w_0 + w_1\mathbf{x}$.
- So, the model can predict values, $\hat{h}(x_i) = w_0 + w_1 x_i$ such that $y_i \approx \hat{h}(x_i) + \epsilon$, where $\epsilon(\ll 1)$ is noise.

| y | x |
|---|---|
| 1.022 | 0.028 |
| 1.310 | 0.182 |
| 1.115 | 0.257 |
| 1.269 | 0.505 |
| 1.369 | 0.650 |
| 1.395 | 0.665 |
| 1.476 | 0.750 |
| 1.466 | 0.852 |
| 1.406 | 0.879 |
| 1.429 | 0.988 |



# Linear Model

- Consider a model:

$$\mathbf{y} = w_0 + w_1\mathbf{x}$$

- **Linear Model**: A model that describes a linear relationship between the input variable (independent variable), $\mathbf{x}^{(1xN)}$, and the output variable (dependent variable), $\mathbf{y}^{(1xN)}$.
- **Linear Regression**: builds a linear model by estimating weight parameters ($w_0$ and $w_1$, often expressed in a vector format $\mathbf{w}$) and predicts $\mathbf{y}$.
  - Originally performed by Legendre (1805) and Gauss (1809) for the prediction of planetary movement.
  - **Simple linear regression**: having one feature (also called explanatory variable).
  - **Multiple linear regression**: having more than one explanatory variable.
  - **Multivariate linear regression**: when multiple correlated target variables are predicted.

$h = w_0 + w_1 x$

$h = w_0 + w_1 x_1 + w_2 x_2$

$h_1, h_2, \dots$

# Linear Model


**Solutions:**

- Statistics: relies on the **conditional probability distribution** of the target variable given the values of features and estimates **w**.
- Machine Learning: Learning the relationship between feature and target values and continuously updating the new learning.

**Goals:**

- To understand the relationship between features and the target.
  - e.g., positive/negative relationships, important features, etc.
- To optimize the model to be robust and useful by adding new features and reducing errors.

# Linear Model

- Ordinary least squares (OLS) find the parameters of the model:

$$w_1 = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{n\sum x_i^2 - (\sum x_i)^2} = \frac{10(8.016) - (5.756)(13.257)}{10(4.257) - (5.756)^2}$$

$$= 0.408$$

$$w_0 = \bar{y} - w_1 \bar{x} = 1.326 - 0.408(0.576) = 1.091$$

- The hypothesized model is $h(\mathbf{x}) = 1.091 + 0.408\mathbf{x}$.
- The solution can be expressed in a vector format as $\mathbf{w} = [1.091, 0.408]$.

| | x | $x^2$ | y | xy |
|---|---|---|---|---|
| 1 | 0.028 | 0.0008 | 1.022 | 0.029 |
| 2 | 0.182 | 0.0331 | 1.310 | 0.238 |
| 3 | 0.257 | 0.0660 | 1.115 | 0.287 |
| 4 | 0.505 | 0.2550 | 1.269 | 0.641 |
| 5 | 0.650 | 0.4225 | 1.369 | 0.890 |
| 6 | 0.665 | 0.4422 | 1.395 | 0.928 |
| 7 | 0.750 | 0.5625 | 1.476 | 1.107 |
| 8 | 0.852 | 0.7259 | 1.466 | 1.249 |
| 9 | 0.879 | 0.7726 | 1.406 | 1.236 |
| 10 | 0.988 | 0.9761 | 1.429 | 1.412 |
| sum | 5.756 | 4.257 | 13.257 | 8.016 |
| average | 0.576 | 0.426 | 1.326 | 0.802 |

# Linear Model

- The parameters can be found in the matrix formation as follows:

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

$w = [w_0, w_1]$

$h = w_0 + w_1 x_1 = w_0 1 + w_1 x$ (1-0)

$= wx$

  - Insert ones to $\mathbf{x}$, so $\mathbf{X} = \begin{bmatrix} 1 & 0.028 \\ \vdots & \vdots \\ 1 & 0.988 \end{bmatrix}$.
    - Adding ones is important so $w_0$ and $w_1$ can be computed at once.
  - $\mathbf{X}^{\mathrm{T}}$ is a transpose of $\mathbf{X}$: $\mathbf{X}$ has a dimension of $10 \times 2$, and $\mathbf{X}^{\mathrm{T}}$ has $2 \times 10$.
  - The product $\mathbf{X}^{\mathrm{T}}\mathbf{X}$ becomes a $2 \times 2$ matrix:

$$\begin{bmatrix} 1 & \cdots & 1 \\ 0.028 & \cdots & 0.988 \end{bmatrix} \begin{bmatrix} 1 & 0.028 \\ \vdots & \vdots \\ 1 & 0.988 \end{bmatrix} = \begin{bmatrix} 1(1) + \cdots + 1(1) & 1(0.028) + \cdots + 1(0.988) \\ 1(0.028) + \cdots + 1(0.988) & (0.028)(0.028) + \cdots + (0.988)(0.988) \end{bmatrix}$$

$$\mathbf{X}^{\mathrm{T}}\mathbf{X} = \begin{bmatrix} 10 & 5.756 \\ 5.756 & 4.257 \end{bmatrix}$$

# Linear Model

- The parameters can be found in the matrix formation as follows:

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

  - $\mathbf{X}^{-1}$ is an inverse matrix, and it is computable if and only if $\mathbf{X}$ is a square matrix (rows and columns have the same number).

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \frac{1}{ad - cb}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

Numpy. linalg.inv(X)

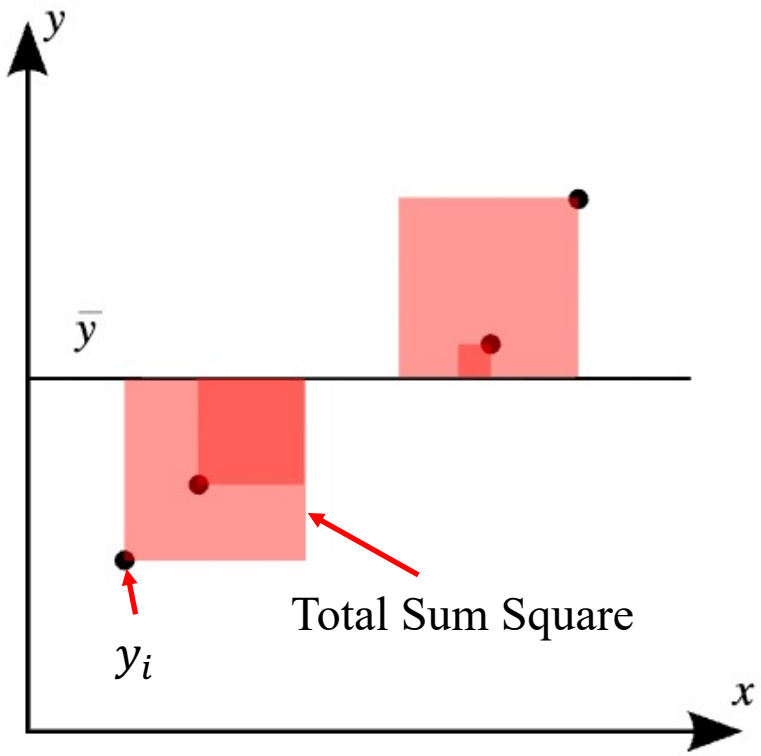
$$(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1} = \begin{bmatrix} 10 & 5.756 \\ 5.756 & 4.257 \end{bmatrix}^{-1} = \frac{1}{10(4.257) - (5.756)^2}\begin{bmatrix} 4.257 & -5.756 \\ -5.756 & 10 \end{bmatrix} = \begin{bmatrix} 0.451 & -0.610 \\ -0.610 & 1.060 \end{bmatrix}$$

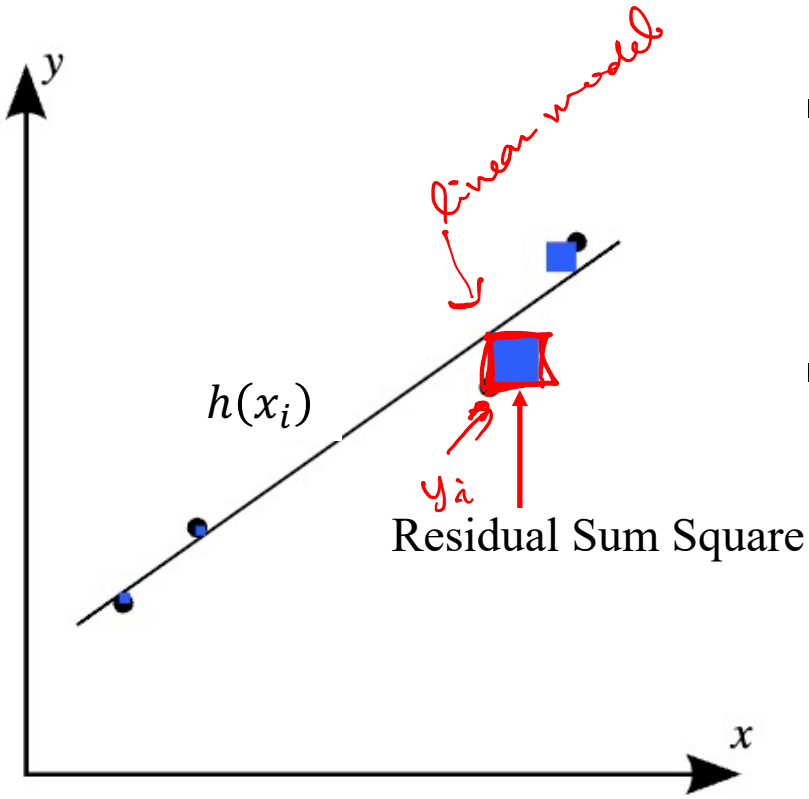
$$\mathbf{w} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y} = \begin{bmatrix} 0.451 & -0.610 \\ -0.610 & 1.060 \end{bmatrix}\begin{bmatrix} 1 & 0.028 \\ \vdots & \vdots \\ 1 & 0.988 \end{bmatrix}[1.022 \quad \cdots \quad 1.429] = [1.091 \quad 0.408]$$

$w_0$ $w_1$

# Linear Model

- Suppose the optimized model is built. How do we evaluate the fit?



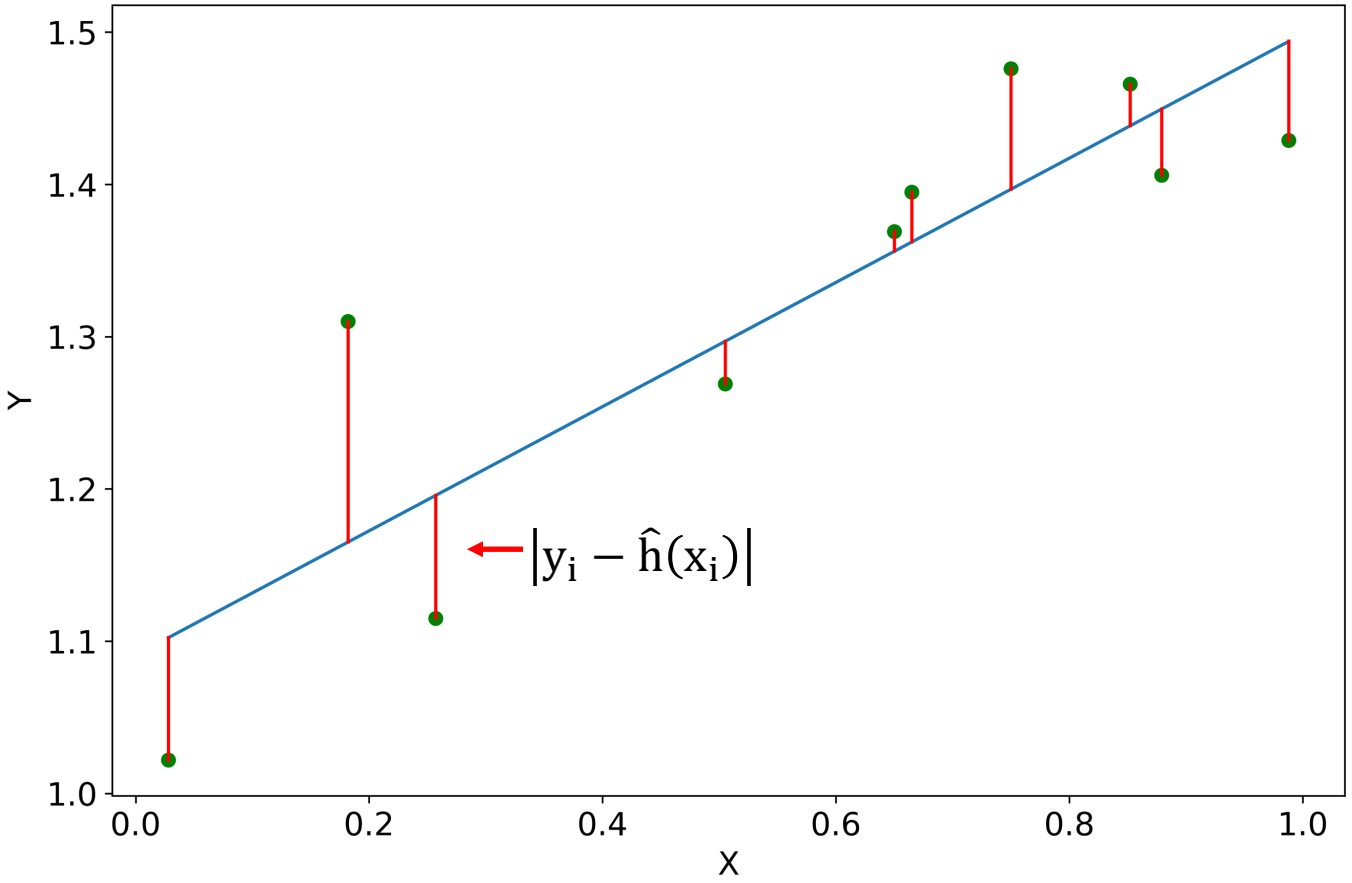

- Residual Sum Squares (RSS)

$$\text{RSS} = \sum_i (\hat{h}(x_i) - y_i)^2$$

- Total Sum of Squares (TSS)

$$\text{TSS} = \sum_i (y_i - \bar{y})^2$$

# Linear Model




- Calculate the coefficient of determination, $R^2$:

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

- The better the fit is, the closer $R^2$ to 1.

# Linear Model

Other evaluation metrics (often called error functions) used in regression are

- Mean Squared Error (MSE):

$$\mathrm{MSE} = \frac{\mathrm{RSS}}{\mathrm{N}} = \frac{1}{\mathrm{N}} \sum_{\mathrm{i}}^{\mathrm{N}} \left( \mathrm{y_i} - \hat{\mathrm{h}}(\mathrm{x_i}) \right)^2$$

- Root Mean Squared Error (RMSE):

$$\mathrm{RMSE} = \sqrt{\mathrm{MSE}} = \sqrt{\frac{1}{\mathrm{N}} \sum_{\mathrm{i}}^{\mathrm{N}} \left( \mathrm{y_i} - \hat{\mathrm{h}}(\mathrm{x_i}) \right)^2}$$

- Mean Absolute Error (MAE):

$$\mathrm{MAE} = \frac{1}{\mathrm{N}} \sum_{\mathrm{i}}^{\mathrm{N}} \left| \mathrm{y_i} - \hat{\mathrm{h}}(\mathrm{x_i}) \right|$$

# Linear Model

| X | y | h | y-h |
|---|---|---|---|
| 0.028 | 1.022 | 1.102 | -0.080 |
| 0.182 | 1.31 | 1.165 | 0.145 |
| 0.257 | 1.115 | 1.196 | -0.081 |
| 0.505 | 1.269 | 1.297 | -0.028 |
| 0.65 | 1.369 | 1.356 | 0.013 |
| 0.665 | 1.395 | 1.362 | 0.033 |
| 0.75 | 1.476 | 1.397 | 0.079 |
| 0.852 | 1.466 | 1.439 | 0.027 |
| 0.879 | 1.406 | 1.450 | -0.044 |
| 0.988 | 1.429 | 1.494 | -0.065 |

- $h(x_i) = 1.091 + 0.408x_i$
- $\hat{h}_1 = h(0.028) = 1.091 + 0.408(0.028) = 1.102$
- $y_1 - \hat{h}_1 = 1.022 - 1.102 = -0.080$
- $RSS = \sum_i (y_i - \hat{h}_i)^2 = ((1.022 - 1.102)^2 + \cdots) = 0.049$
- $TSS = \sum_i (y_i - \bar{y})^2 = ((1.022 - 1.326)^2 + \cdots) = 0.206$
- $R^2 = 1 - \frac{RSS}{TSS} = 1 - \frac{0.049}{0.206} = 0.762$
- $MSE = \frac{RSS}{N} = \frac{0.049}{10} = 0.005$
- $RMSE = \sqrt{MSE} = \sqrt{0.005} = 0.07$
- $MAE = \frac{1}{N}\sum|y_i - \hat{h}_i| = \frac{1}{10}(|1.022 - 1.102| + \cdots) = 0.008$

# Linear Model

- Consider a dataset $\mathbf{X}^{N\times D} \in \mathbb{R}$:
  - $\in$: means "in".
  - $\mathbb{R}$: All values are real numbers.
  - N: number of observations (rows)
  - D: number of features (columns).
- For one observation, $\mathbf{x}^{1\times D}$, is a vector and its linear model can be expressed as

$$h(\mathbf{x}) = w_0 + w_1x_1 + \cdots w_Dx_D = w_0 + \sum_{j=1}^{D} w_jx_j = \sum_{j=0}^{N} w_jx_j = \mathbf{w}\cdot\mathbf{x}_i \qquad (1\text{-}1)$$

  where $w_0$ is a bias parameter (or intercept) that fixes the offset in the data.
- An operator " $\cdot$ " is a dot product.
- The goal is to estimate parameters $\mathbf{w} = [w_0, w_1, \ldots, w_D]$.

# Linear Model

- Alternatively, Eq. (1-1) above can be expressed as

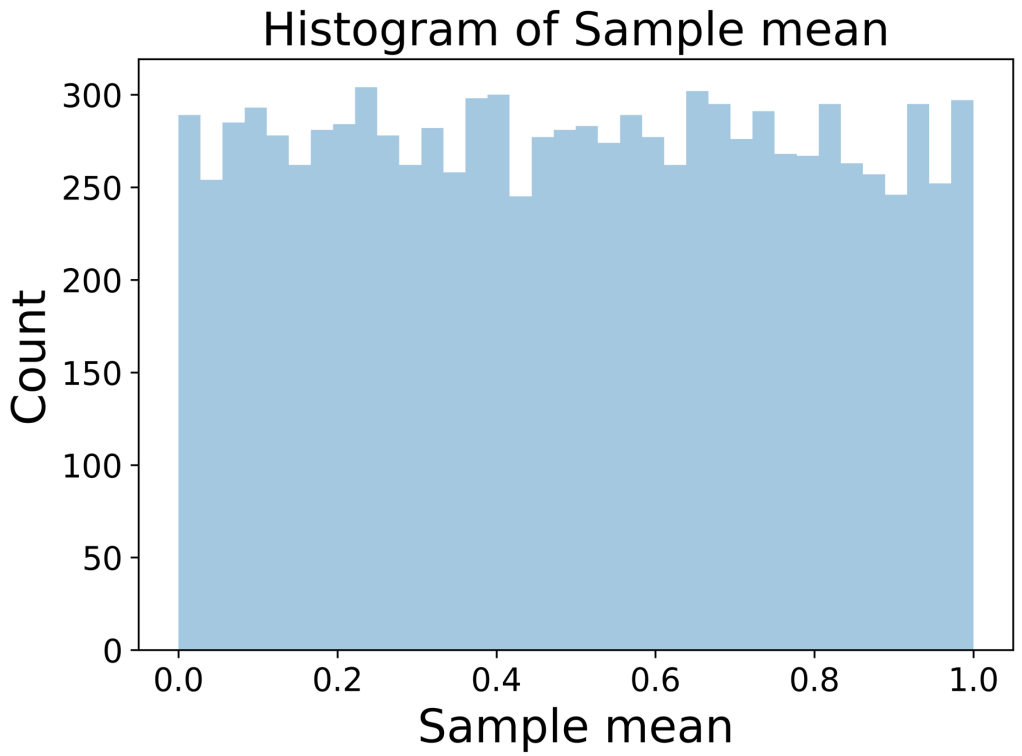
$$h(\mathbf{X}) = \sum_{j=0}^{D} w_j \cdot x_{ji} = \mathbf{w}\mathbf{X}^T = \mathbf{X}\mathbf{w}^T. \quad (1\text{-}2)$$

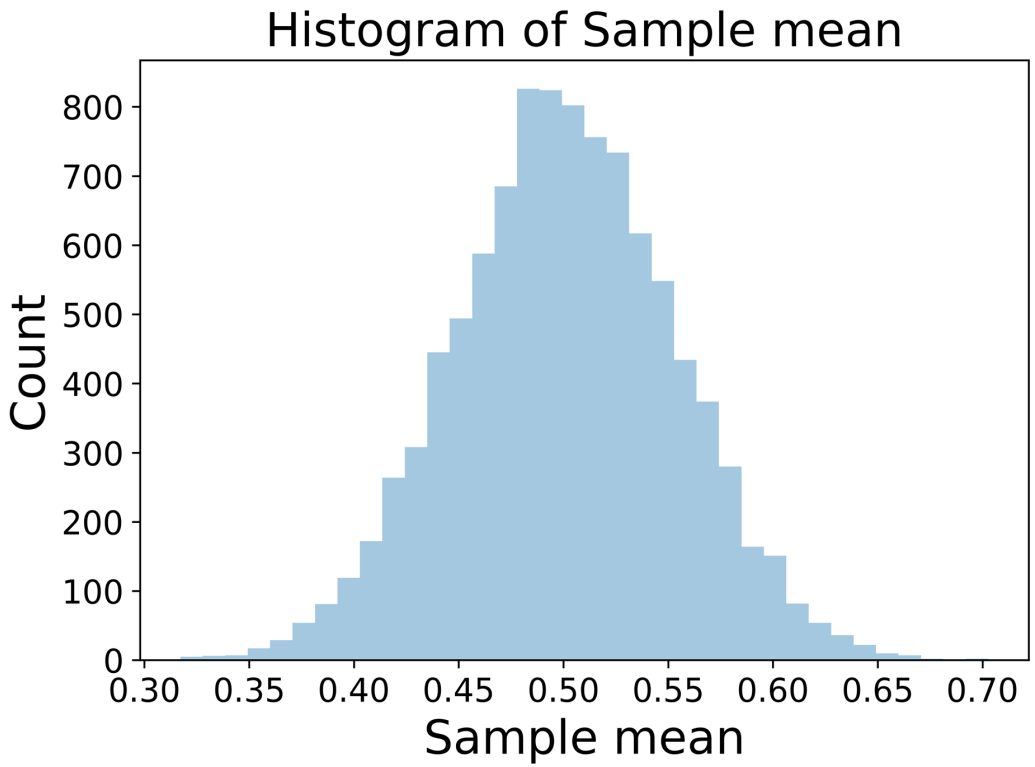
# Maximum Likelihood Estimator (MLE)

- The solution (Eq.1-0) is derived from the MLE setting.
- The likelihood is a measurement of the frequency of events that **already** occurred.
- In other words, the frequency of events can be understood when the data is given.
- There are some assumptions:
  - Data is a large data set.
  - A sample must be independently distributed.
  - A sample must be identically distributed.
- Such assumptions imply that the given data is a sample of a very large finite data set and each observation is from Gaussian (normal) distribution (central limit theorem).

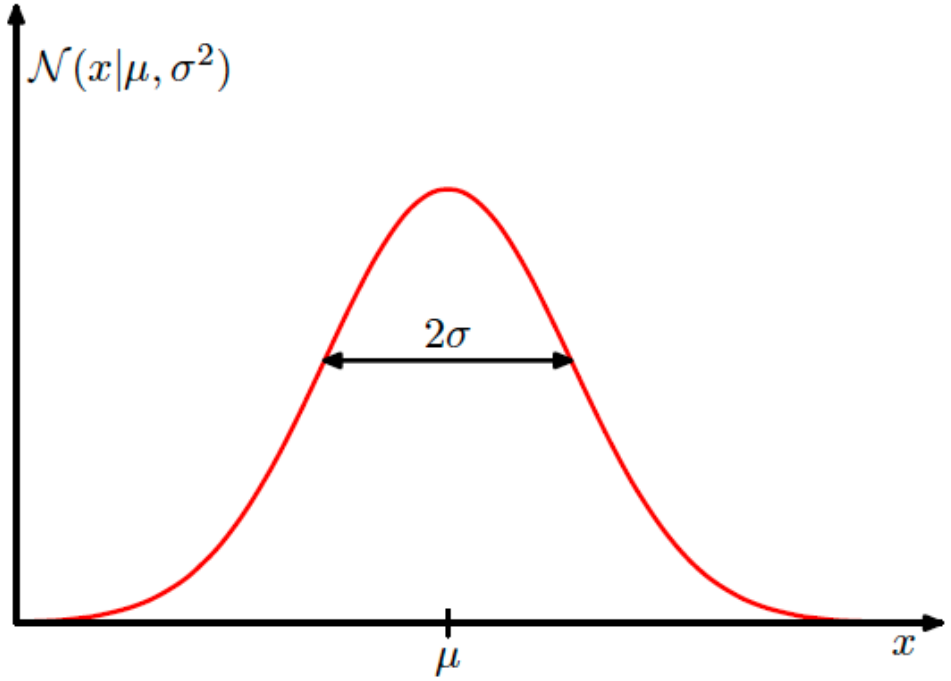
# Maximum Likelihood Estimator (MLE)

- The ***central limit theorem***: when independent random variables are added, their properly normalized sum tends toward a normal distribution even if the original variables are not normally distributed.



# Maximum Likelihood Estimator (MLE)



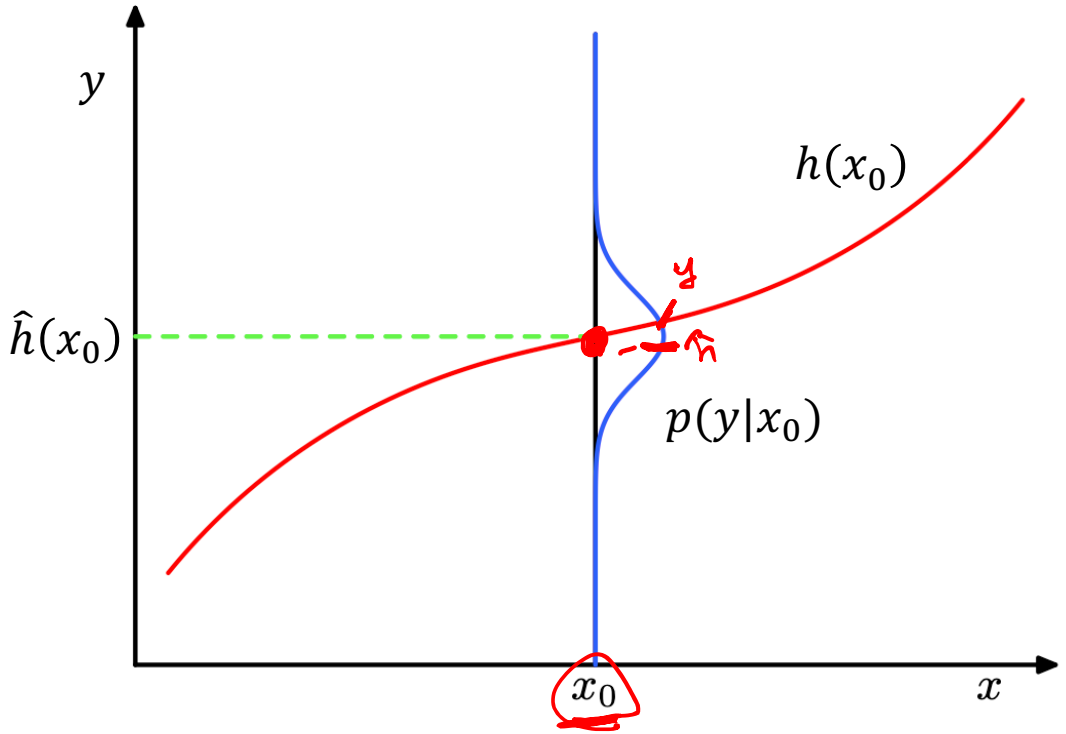

- Gaussian (normal) Distribution Properties
  - Symmetrical: both sides are identically mirrored at the center (mean)
  - Most mass is at the center around the mean.
  - Spread by the standard deviation
  - "tails" are the least likely outcomes and approach zero infinitely but never reach zero.

- The ***probability density function*** (PDF) creates the normal distribution as follows:

$$f(x) = \mathcal{N}(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right).$$

- $\mu$ : Mean, $\sigma$: standard deviation where $\sigma^2$ is the variance.
- Remember the Gaussian equation. This will be used many times throughout the semester.

# Maximum Likelihood Estimator (MLE)




- If y is given by a deterministic function $\mathrm{h}(\mathbf{x}, \mathbf{w})$ with additive Gaussian noise $\epsilon = \mathcal{N}(0, 1/\sigma^2)$, then, each $y_i$ can be expressed as

$$y_i = \hat{h}(\mathbf{x}_i, \mathbf{w}) + \epsilon = \sum_{j=0}^{D} w_j x_{ji} + \epsilon.$$

- Suppose y is distributed in Gaussian. Then, the conditional probability of y given $\mathbf{x}_i$, $\mathbf{w}$, and $\sigma_i^2$ is the ***likelihood***:

$$p(y_i|\mathbf{x}_i, \mathbf{w}, \sigma_i^2) = \mathcal{N}(y_i|\hat{h}_i, \sigma_i^2) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(-\frac{1}{2\sigma_i^2}(y_i - \hat{h}_i)^2\right). \quad (1\text{-}3)$$

# Maximum Likelihood Estimator (MLE)

- Consider a data set of inputs $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]$ with corresponding target vector $\mathbf{y} = [y_1, \ldots, y_N]$.
- Suppose all examples have the same variance, e.g., $\sigma_i^2 = \sigma^2$.
- Then, the likelihood of the data set becomes a product of individual likelihood (Eq. 1-3) as shown below

$$p(\mathbf{y}|\mathbf{X}, \mathbf{w}, \sigma^2) = \prod_{i=1}^{N} \mathcal{N}(y_i|\hat{h}_i, \sigma_i^2) = \prod_{i=1}^{N} \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) \exp\left(-\frac{1}{2\sigma^2}(y_i - \hat{h}_i)^2\right). \quad (1\text{-}4)$$

# Maximum Likelihood Estimator (MLE)

- Using Log properties, $\ln xy = \ln x + \ln y$, $\ln(x^a) = a \ln x$, and $\ln e^x = x$, Eq. (1-4) can be expressed as a log-likelihood:

$$\ln p(\mathbf{y}|\mathbf{X}, \mathbf{w}, \sigma^2) = \sum_{i=1}^{N} \ln\left[\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) \exp\left(-\frac{1}{2\sigma^2}(y_i - \hat{h}_i)^2\right)\right] \quad (1\text{-}5)$$

- The expansion of Eq. 1-5 becomes

$$\ln p = -\frac{N}{2}\ln\sigma^2 - \frac{N}{2}\ln(2\pi) - \frac{1}{2\sigma^2}\sum_{i=1}^{N}(y_i - \hat{h}_i)^2 = -\frac{N}{2}\ln\sigma^2 - \frac{N}{2}\ln(2\pi) - \frac{1}{\sigma^2}E_D(\mathbf{w}) \quad (1\text{-}6)$$

where $E_D(\mathbf{w})$ is an error function (loss function), $E_D(\mathbf{w}) = \frac{1}{2}\sum(y_i - \hat{h}_i)^2$.

- The solution of $\mathbf{w}$ maximizes the log-likelihood,

$$\mathbf{w} = \underset{w}{\operatorname{argmax}}[\ln p(\mathbf{y}|\mathbf{X}, \mathbf{w}, \sigma^2)] \quad (1\text{-}7)$$

which relies on the 3rd term.

# Maximum Likelihood Estimator (MLE)


- The optimization of Eq. 1-7 is the maximization of log-likelihood, which is equivalent to minimized $E_D(\mathbf{w})$.
- **Optimization**: Recall calculus, we find the given function's minima or maxima (either local or global) by setting function's derivative to 0.
- Since **w** is our interest, we take a gradient (or partial derivative) of $E_D(\mathbf{w})$ with respect to **w,** and set to 0.

$$\nabla \ln p(\mathbf{y}|\mathbf{X}, \mathbf{w}, \sigma^2) = \nabla E_D(\mathbf{w}) = 0$$

$$\nabla E_D(\mathbf{w}) = \frac{\partial}{\partial \mathbf{w}} \left( \frac{1}{2} \sum_{i=1}^{N} (y_i - \hat{h}_i)^2 \right) = \frac{2}{2} \sum_{i=1}^{N} (y_i - \hat{h}_i) \frac{\partial}{\partial \mathbf{w}} (y_i - \hat{h}_i)$$

$$= \sum_{i=1}^{N} (y_i - \hat{h}_i) \frac{\partial}{\partial \mathbf{w}} (y_i - \mathbf{w}^T \mathbf{x}_i) = \sum_{i=1}^{N} (y_i - \hat{h}_i)(-\mathbf{x}_i) = 0 \quad (1\text{-}8)$$

# Maximum Likelihood Estimator (MLE)

- Eq. 1-8 becomes

$$-\sum_{i=1}^{N} y_i \mathbf{x}_i + \sum_{i=1}^{N} \hat{h}_i \mathbf{x}_i = 0 \rightarrow \mathbf{X}^T\mathbf{y} = \mathbf{w}\mathbf{X}^T\mathbf{X}$$

- The solution of $\mathbf{w}$ is

$$\mathbf{w}_{\mathbf{MLE}} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}. \tag{1-9}$$

- The variance of the model, $\sigma^2$, can be found as the

$$\frac{\partial}{\partial\sigma^2}\left(-\frac{N}{2}\ln\sigma^2 - \frac{1}{2\sigma^2}\sum_{i=1}^{N}\left(y_i - \hat{h}_i\right)^2\right) = 0 \rightarrow -\frac{N}{2}\left(\frac{1}{\sigma^2}\right) + \frac{1}{2\sigma^4}\sum_{i=1}^{N}\left(y_i - \hat{h}_i\right)^2 = 0$$

$$\sigma^2_{MLE} = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \hat{h}_i\right)^2. \tag{1-10}$$

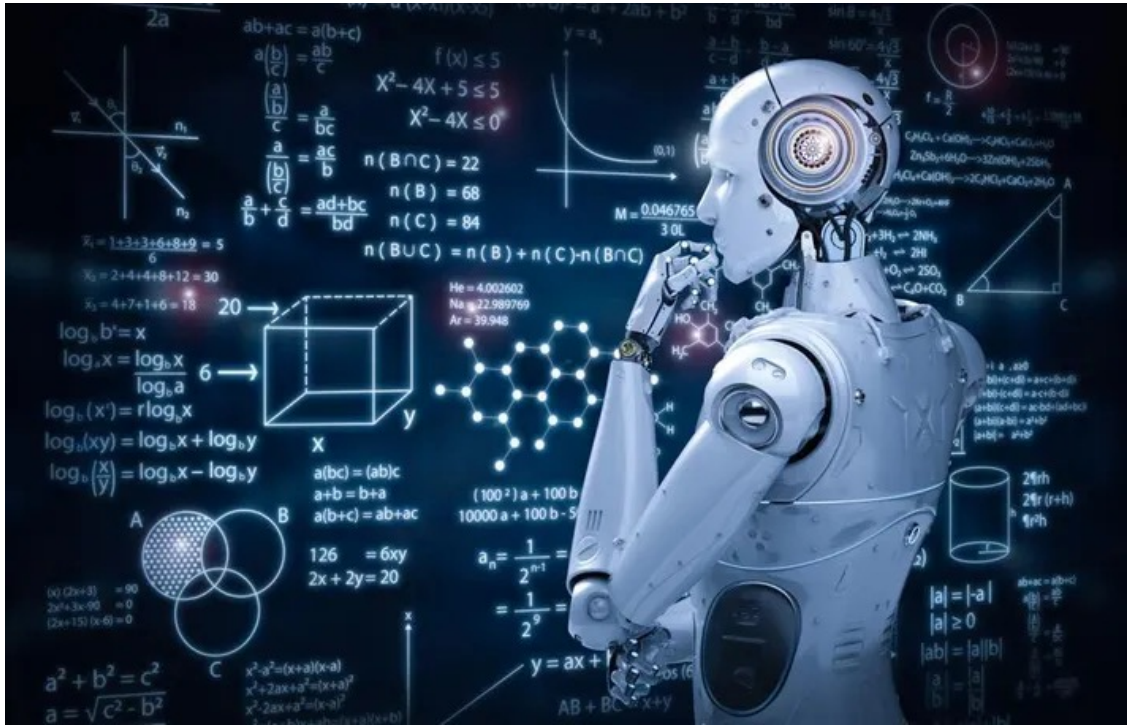

# Machine Learning

# Introduction to Machine Learning

- Introduction of Machine Learning
  - Machine Learning Definition
    - What is machine learning?
    - Where is it applied to?
    - How is machine learning different from traditional modeling processes?
  - Learning Types
    - What are the different learning tasks in ML?
    - How does the data look different in each learning?
    - What are the three components of ML algorithms?
- Workflows

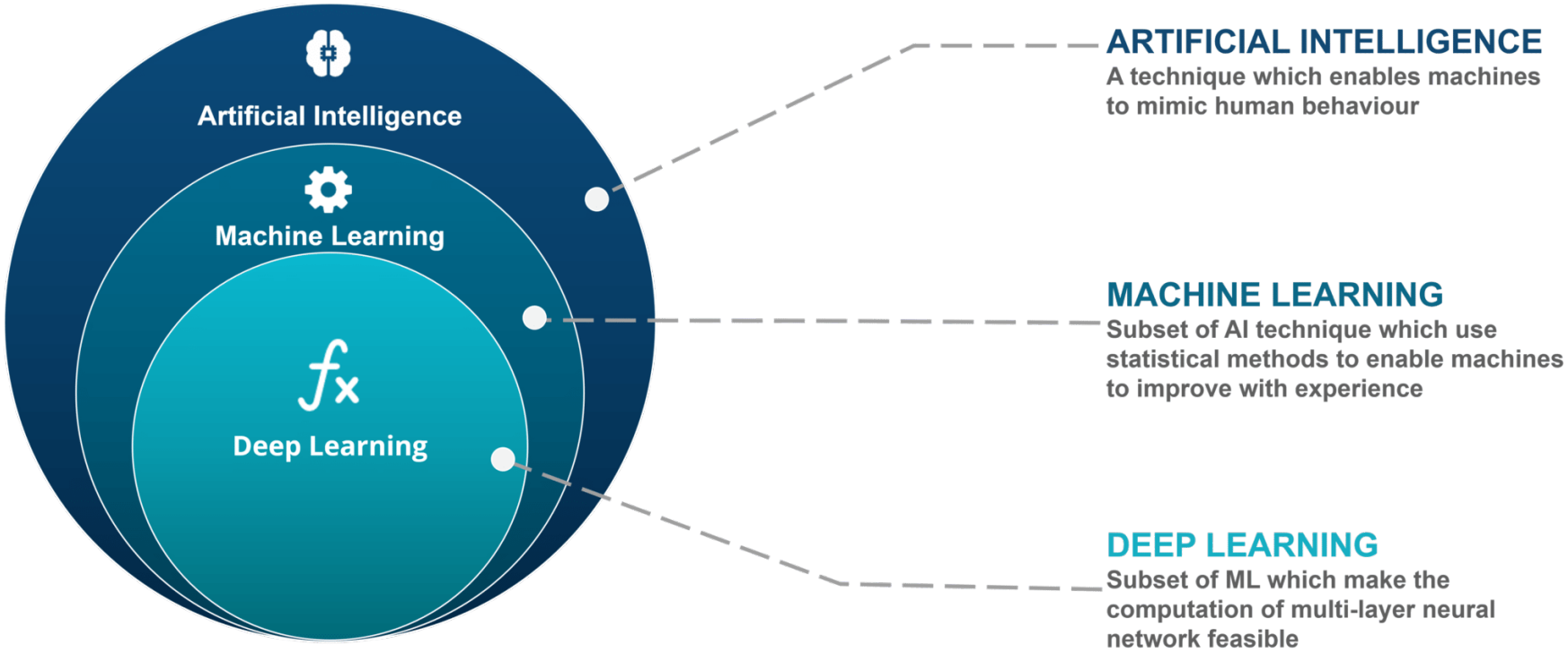
# Introduction to Machine Learning

- What is machine learning (ML)?
  - Arthur Samuel from IBM coined the term “Machine Learning” in 1959.
  - A computer algorithm to learn from experience to solve a problem and evaluate the performance.
    - Take advantage of data to capture characteristics of interest of unknown underlying patterns.
    - Use the learned pattern and predict the outcome.
  - Connect to the linear regression.
    - The linear relationship was explored by solving weight values in a sequential learning fashion.
    - Updating weights from the gradient of the sum squared loss function is the learning from experience.
    - The MSE or RMSE is the performance evaluation.
- Where is it applied to?
- How is ML different from the traditional modeling processes?



# Introduction to Machine Learning

- What is machine learning?
- Where is it applied to?
  - ML is a part of artificial intelligence (AI).
- How is ML different from the traditional modeling processes?

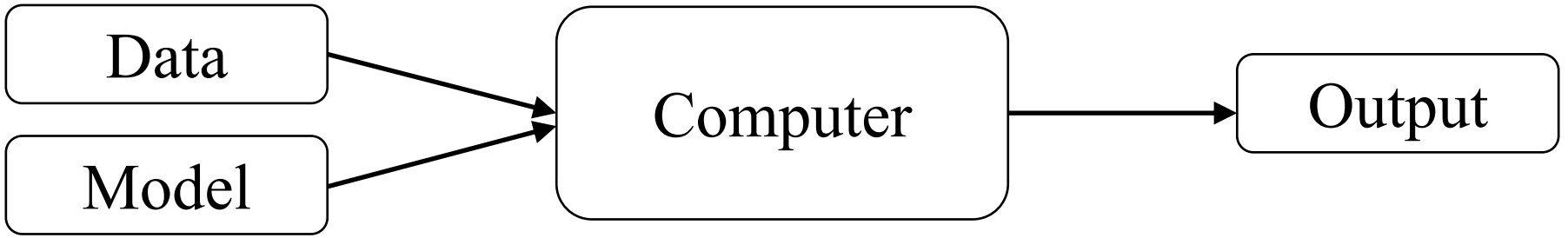


# Introduction to Machine Learning

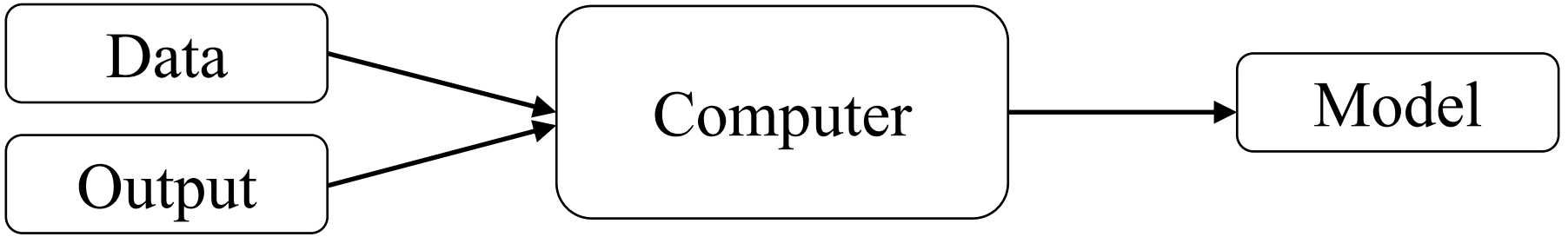

- What is machine learning?
- Where is it applied to?
  - ML makes machines improve the learning.
  - Deep Learning is a special technique extended from neural networks (one of the ML algorithms) and can capture more complex patterns in complex data (images, sounds, texts, sequential data).
  - AI allows machines to learn themselves.
  - Without ML, AI will not learn better.
    - The models must be well made statistically, robustly, and intelligently.
- How is ML different from the traditional modeling processes?

# Introduction to Machine Learning

- What is machine learning?
- Where is it applied to?
- How is ML different from the traditional modeling processes?
  - Traditional modeling process




  - ML modeling process



# Workflow of Machine Learning Projects

- Introduction of Machine Learning
  - Machine Learning Definition
    - What is machine learning?
    - Where is it applied to?
    - How is machine learning different from traditional modeling processes?
  - Learning Types
    - What are the different learning tasks in ML?
    - How does the data look different in each learning?
    - What are the three components of ML algorithms?

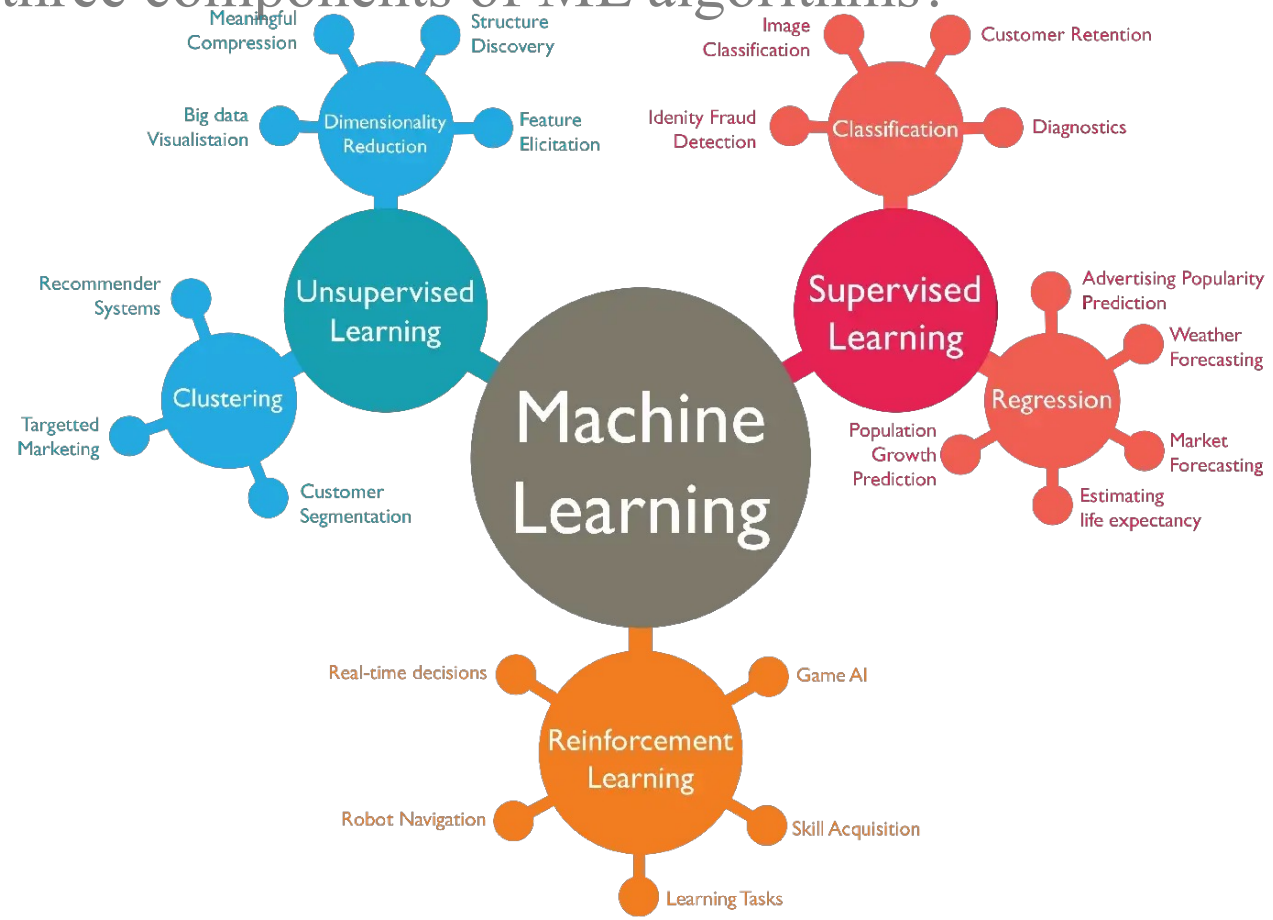
# Learning Types

- What are the different learning tasks in ML?
- How does the data look different in each learning?
- What are the three components of ML algorithms?

| Supervised Learning | Unsupervised Learning | Reinforcement Learning |
| --- | --- | --- |
| • Predict outcome<br>• Forecast future<br>• Direct Feedback<br>• Labeled Data | • Find hidden structures in data and cluster them into subgroups<br>• Reduce the data dimensions (features).<br>• No labels or targets<br>• No Feedback | • Learn a series of actions and forecast the future<br>• Requires decision Process |

# Learning Types

- What are the different learning tasks in ML?
- How does the data look different in each learning?
- What are the three components of ML algorithms?



# Learning Types

- What are the different learning tasks in ML?
- How does the data look different in each learning?
- What are the three components of ML algorithms?

# Learning Types

- What are the different learning tasks in ML?
- How does the data look different in each learning?
  - Supervised learning
- What are the three components of ML algor

Rows: observations or examples

Labels:
- headers, column names, or feature names

Columns:
- features, predictors, or attributes

Target

| | longitude | latitude | housing_median_age | total_rooms | total_bedrooms | population | households | median_income | median_house_value | ocean_proximity |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | -122.23 | 37.88 | 41 | 880 | 129.0 | 322 | 126 | 8.3252 | 452600 | NEAR BAY |
| 1 | -122.22 | 37.86 | 21 | 7099 | 1106.0 | 2401 | 1138 | 8.3014 | 358500 | NEAR BAY |
| 2 | -122.24 | 37.85 | 52 | 1467 | 190.0 | 496 | 177 | 7.2574 | 352100 | NEAR BAY |
| 3 | -122.25 | 37.85 | 52 | 1274 | 235.0 | 558 | 219 | 5.6431 | 341300 | NEAR BAY |
| 4 | -122.25 | 37.85 | 52 | 1627 | 280.0 | 565 | 259 | 3.8462 | 342200 | NEAR BAY |

Numerical – continuous data

Categorical – discrete data
- Integer (0 or 1)
- Text

- Typical questions:
  - Which features impact the house value?
  - Are nearby Bay houses more expensive than other locations?
  - And more…

# Learning Types

- What are the different learning tasks in ML?
- How does the data look different in each learning?
  - Unsupervised Learning
- What are the three components of ML algorithms?

| | A | B | C |
|---|---|---|---|
| 1 | -0.4527199 | -0.0182128 | 2 |
| 2 | -0.1697933 | 1.07374545 | 1 |
| 3 | 1.12872909 | -2.3950424 | 6 |
| 4 | 0.12250626 | -0.973069 | 1 |
| 5 | -0.4091998 | 0.12964911 | 2 |
| 6 | 0.47622927 | -0.0481475 | 2 |
| 7 | 0.26302451 | -0.309155 | 2 |
| 8 | -0.4783598 | 0.10375367 | 2 |
| 9 | -0.9788868 | -0.0926921 | 1 |

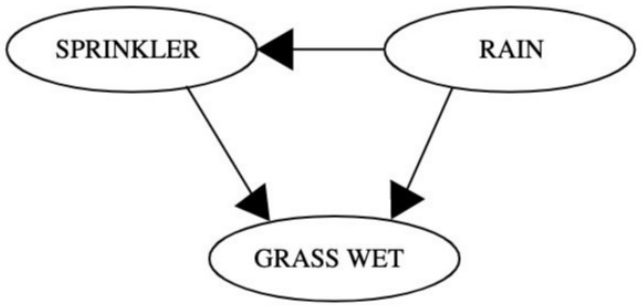
- Columns are not labeled.
- The target is not known.

# Learning Types

- What are the different learning tasks in ML?
- How does the data look different in each learning?
  - Reinforcement Learning

hms?

| RAIN | SPRINKLER T | SPRINKLER F |
|---|---|---|
| F | 0.4 | 0.6 |
| T | 0.01 | 0.99 |




| RAIN T | RAIN F |
|---|---|
| 0.2 | 0.8 |

| SPRINKLER | RAIN | GRASS WET T | GRASS WET F |
|---|---|---|---|
| F | F | 0.0 | 1.0 |
| F | T | 0.8 | 0.2 |
| T | F | 0.9 | 0.1 |
| T | T | 0.99 | 0.01 |

- What is the percentage of grass getting wet when it rains?
- Should we turn off the sprinkler if it rains?

# Learning Types

- What are the different learning tasks in ML?
- How does the data look different in each learning?
- What are the three components of ML algorithms?
  - Algorithm / Model
    - Each algorithm has its own way of learning.
    - Knowing the characteristics and conditions of each algorithm is essential.
  - Evaluation / Objective (loss) function
    - The objective or loss function sets the learning and measures the model's performance.
  - Optimization
    - A technique that an algorithm uses to increase the learning performance.
  - For example: Linear regression
    - Use the sum squared error function, $\frac{1}{2}\sum_i \left(y_i - \hat{h}_i\right)^2$, as a loss function in a gradient descent fashion to optimize the weight parameter.

# Introduction to Machine Learning


- Model: A computational algorithm that recognizes patterns of data using mathematics
- Machine Learning:  Use a model to produce the output with known inputs (data).
  - Features are part of the data that carries the information and are independent variables.
  - Target is a dependent variable that is to be predicted.
  - The process of modeling is called training.
  - Machine learning requires features and the target ahead of time so the machine can learn from inputs.

# Introduction to Machine Learning


- Data:
  - **X** will be used for the feature matrix in the dimension of N×D.
    - N rows represent examples (observations) and are independent of each other.
      - **x** denotes an example, and that is a 1×D vector.
    - D columns represent the number of features.
  - **y** will be used for the target and is an N×1 vector.
- Technology:
  - Python is one of the most popular languages used in machine learning.
  - Jupyter notebook is a work environment.
  - Jupyter notebook can be used via Visual Studio Code, Anaconda, or Google Colab.


# Sequential Learning

# Sequential Learning

- Is MLE preferable all the time?
  - The probability distribution of data needs to be known ahead of time.
  - A large sample size is required and distributed in Gaussian distribution.
  - Otherwise, the solution will be biased and inconsistent.

- Machine Learning Considerations:
  - How will the ML linear regression algorithm learn and solve weights?
  - How will the algorithm optimize weights?
  - What are the options in optimization?
  - How will the best model be selected?

# Sequential Learning

- ML uses a ***sequential learning*** technique to estimate solutions.
- Sequential learning: An iterative fashion to update **w** from the previously learned **w** values as follows

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_D$$

  where $\eta$ is a learning rate parameter ($0 < \eta < 1$) and $\nabla E_D$ is the gradient of the loss function.
- For a large $\tau$, **w** will converge – values will become stable and will not change anymore.
  - The loss function must be differentiable everywhere. The most common loss function is a squared loss function, $E_D = \frac{1}{2N}\sum_{i=1}^{N}(y_i - \hat{h}_i)^2$.
  - The learning rate controls the update.
    - The computational cost (time) is another major consideration.

# Sequential Learning

1. **Gradient Descent (GD):**
   - It involves using the **entire** dataset or training set to compute the gradient to find the optimal solution.
   - Searching solution movement toward the optimal solution, the *local* or *global* optimal solution, is always direct.
   - Must consider the memory and computational time trade-off.
2. **Stochastic Gradient Descent (SGD):**
   - The dataset is randomly shuffled to avoid pre-existing.
   - While all observations must pass in each iteration, the updates will be made one observation at a time.
   - Therefore, the computational cost will be relatively higher than GD, and convergence can be noisier.
3. **Mini-Batch Gradient Descent:** Bridge between GD and SGD.
   - Randomly shuffled small samples (batches) are used in SGD fashion.

# Sequential Learning

Consider a data set $\mathbf{X}^{NxD}$.

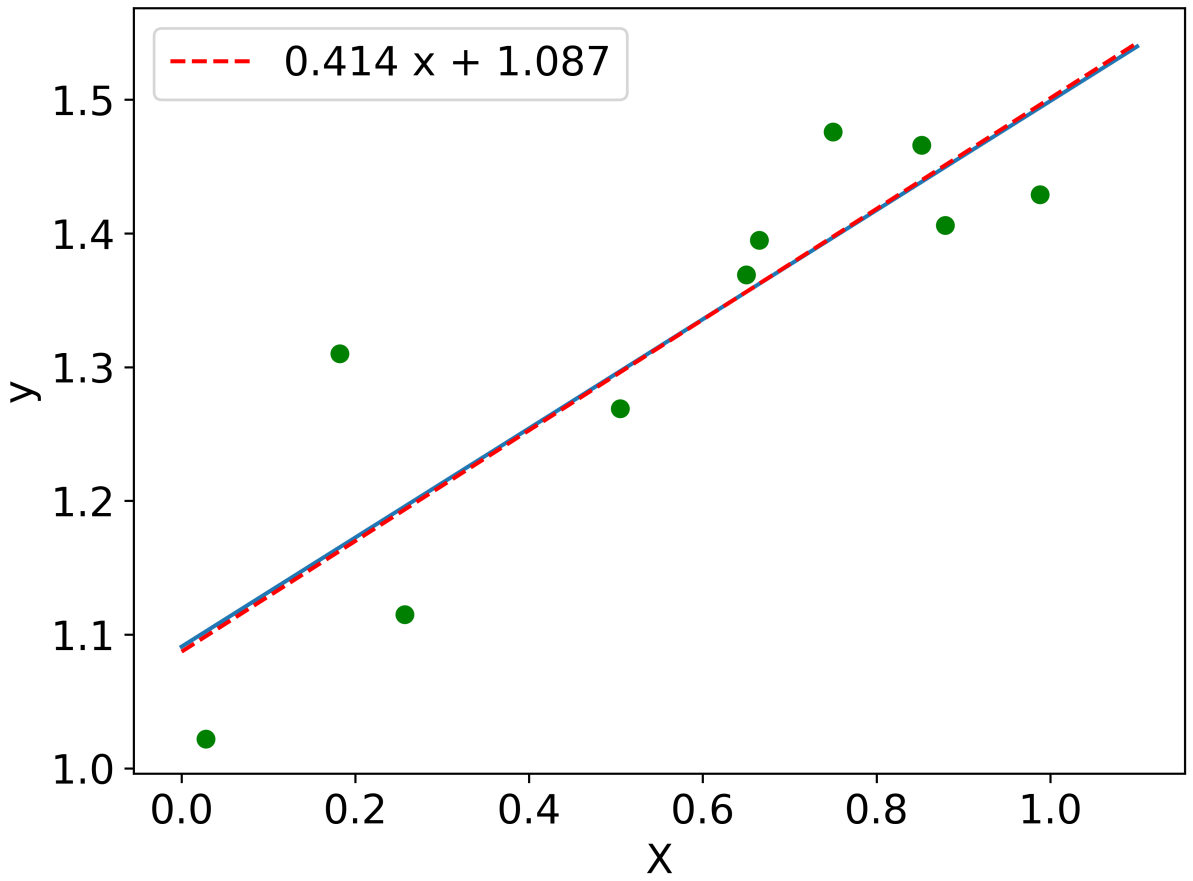
- GD: Iterates the weight computation until the error function converges

Initialize **w**
for $\tau$ until T{
    Calculate h(**X**)
    Calculate $\nabla E_j \; \forall \; w_j \in \mathbf{w}$
    $w_j^{\tau+1} = w_j^{\tau} - \eta \nabla E_j \; \forall \; j \in D$
}
Calculate h(**X**)
Calculate the error

- Denotation: $\forall$ represents for all.

- Early stop (option): Terminate early if $w_j^{(\tau+1)}$ and $w_j^{(\tau)}$ are the same before $\tau$ reaches T.
  - Or if an error with $E\left(w_j^{(\tau+1)}, \mathbf{X}\right)$ becomes larger than an error with $E\left(w_j^{(\tau)}, \mathbf{X}\right)$

# Sequential Learning




- Let $h(\mathbf{x}) = w_0 + w_1\mathbf{x}$ and $E_D = \frac{1}{N}\sum_i (y_i - h_i)^2$.
- $\nabla E_D = \frac{2}{N}\sum (y_i - h_i)(-x_i)$
  - $dw_1 = -\frac{2}{N}(\mathbf{y} - \mathbf{h})\mathbf{x}$
  - $dw_0 = -\frac{2}{N}\sum (y_i - h_i)$

```
w0, w1 = 0, 0
for i in range(m):
    h = w1 * X + w0
    dw1 = -(2*(X.T).dot(Y - h))/n
    dw0 = -2 * np.sum(Y - h)/n
    w1 = w1 - eta*dw1
    w0 = w0 - eta*dw0
```

- The dashed red line is the result from GD.
- The solid blue line is the result of MLE.

# Sequential Learning

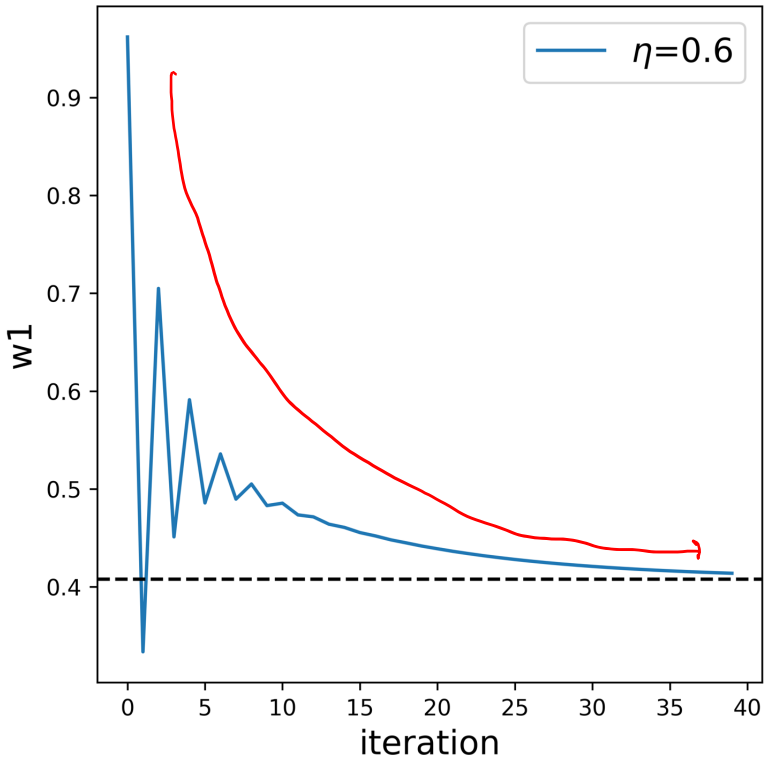
```
w0, w1 = 0, 0
for i in range(m):
  h = w1 * X + w0
  dw1 = -(2*(X.T).dot(Y - h))/n
  dw0 = -2 * np.sum(Y - h)/n
  w1 = w1 - eta*dw1
  w0 = w0 - eta*dw0
```

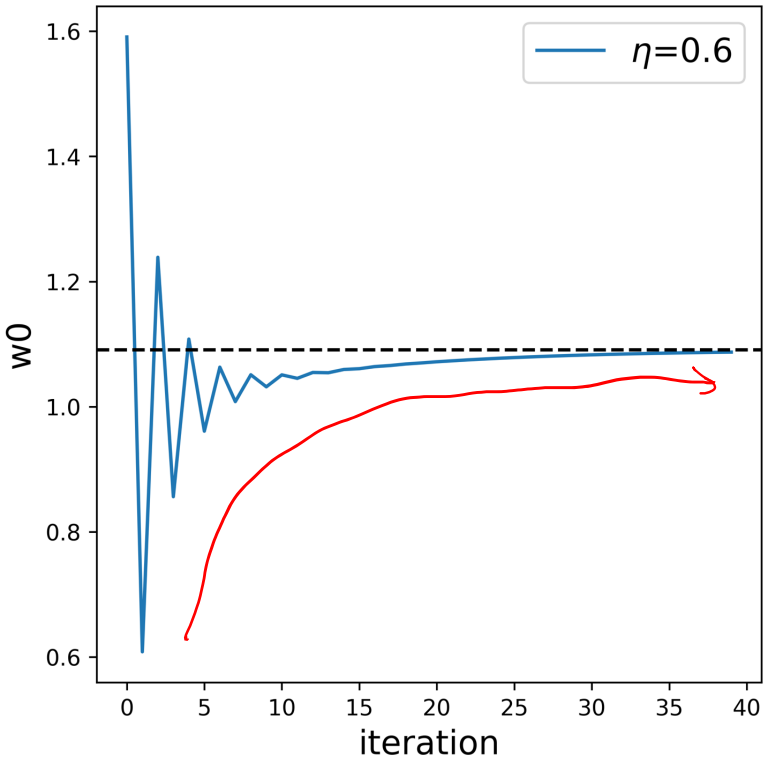
| X | y | $h^{(0)}$ | $h^{(1)}$ | $h^{(70)}$ |
|---|---|---|---|---|
| 0.028 | 1.022 | 1.618 | 0.618 | 1.102 |
| 0.182 | 1.31 | 1.766 | 0.669 | 1.165 |
| 0.257 | 1.115 | 1.838 | 0.694 | 1.196 |
| 0.505 | 1.269 | 2.077 | 0.777 | 1.297 |
| 0.65 | 1.369 | 2.216 | 0.825 | 1.356 |
| 0.665 | 1.395 | 2.230 | 0.830 | 1.362 |
| 0.75 | 1.476 | 2.312 | 0.858 | 1.397 |
| 0.852 | 1.466 | 2.410 | 0.892 | 1.439 |
| 0.879 | 1.406 | 2.436 | 0.902 | 1.450 |
| 0.988 | 1.429 | 2.541 | 0.938 | 1.494 |

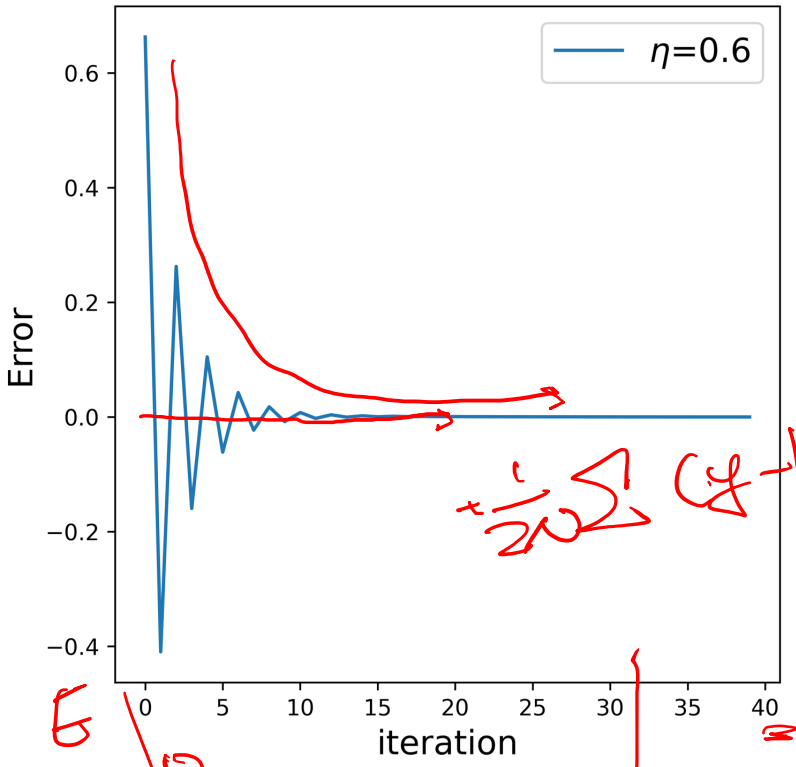
Iteration Examples:

- Let eta = 0.6.
- At the $0^{th}$ iteration:
  - $h^{(0)} = 0 * X + 0 = [0, \cdots, 0]$
  - $dw_1 = -\frac{2}{10}([0.028, \cdots, 0.988] \cdot [1.022 - 0, \cdots, 1.429 - 0]) = -1.603$
  - $dw_0 = -\frac{2}{10}[(1.022 - 0) + \cdots + (1.429 - 0)] = -2.651$
  - $w_1^{(1)} = 0 - 0.6(-1.603) = 0.962$
  - $w_0^{(1)} = 0 - 0.6(-2.651) = 1.591$
- At the $1^{st}$ iteration:
  - $h^{(1)} = 0.962 * X + 1.591 = [1.618, \cdots, 2.541]$
  - dw1=1.047, dw0 = 1.638
  - w1=0.334, w0 = 0.608
- At the $70^{th}$ iteration:
  - dw0=-4.050e-05, dw1=6.547e-05
  - w0=1.087, w1=0.408

# Sequential Learning




- After iterations, weights and an error converge.

# Sequential Learning

- SGD: Randomly shuffle X and update $w_j$ for each $\mathbf{x}_i$ through T iterations.

---

Initialize **w**
for $\tau$ until T{
    Shuffle **X**, **y**
    for each $\mathbf{x}_i$ {
        Calculate $h(\mathbf{x}_i)$
        Calculate $\nabla E_j \; \forall \; w_j \in \mathbf{w}$
        $w_j = w_j - \eta \nabla E_j \; \forall \; j \in D$
    }
}
Calculate $h(\boldsymbol{X})$
Calculate the error

---

T - many

observation

n - many times

GD $T(n) = \Theta(T)$

SGD $T(n) = \Theta(T \cdot N)$

# Sequential Learning

- Mini-Batch GD:
  - Randomly shuffle and partition into K many batches with m many equal observations.
  - Perform GD on each batch until the error function converges.

---

Initialize **w**
for $\tau$ until T {
    Shuffle **X**, **y**
    Partition into K many batches with an equal number of observations
    for each k{
        Calculate $h(\mathbf{X}_k)$
        Calculate $\nabla E_j \quad \forall w_j \in \mathbf{w}$
        $w_j = w_j - \eta \nabla E_j \quad \forall j \in D$
    }
}
Calculate $h(\mathbf{X})$
Calculate the error

---

$GD\ T(T) = \Theta(T)$

$SGD\ T(T, N) = \Theta(T \cdot N)$

Mini-Batch $T = \Theta(T \cdot k)$

$k << N$


# Linear Regression Using Python

# Linear Regression with Scikit-learn


All ML modeling using Scikit-Learn follows these steps:

1. Import the Scikit-Learn model and any necessary metrics package to evaluate the model.
2. Build the model.
3. Fit (train) the model to the train data.
4. Predict the training target and evaluate.
5. Validate the model (optional)
6. Predict the test target using the test set.

discuss in model selection

# Linear Regression with Scikit-learn

1. Import **linear_model** from **sklearn**.

```
from sklearn.linear_model import LinearRegression
```

- Some attributes and methods for Scikit-Learn Linear Regression
  - coef_: estimates coefficients for the linear problem.
  - intercept_: estimates independent terms in the linear model, $w_0$ in our example.
  - fit(X,y): fits linear model.
  - predict(X): Predict using the linear model.
  - score(X,y): returns $R^2$ of the prediction.
  - Attributes and methods may differ in a different version of Scikit-learn.

# Linear Regression with Scikit-learn

2. Build a model.

```
model = LinearRegression()
```

- Parameters can be initialized inside the parenthesis.
- Parameters are
  - **fit_intercept:** *bool, default=True*
    - Whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations (i.e. data is expected to be centered).
  - **copy_X:** *bool, default=True*
    - If True, X will be copied; else, it may be overwritten.
  - **n_jobs:** *int, default=None*
    - The number of jobs to use for the computation.
  - **Positive:** *bool, default=False*
    - When set to True, forces the coefficients to be positive. This option is only supported for dense arrays

# Linear Regression with Scikit-learn


3. Fit (train) the model with the train data set.

```
model.fit(X,y)
```

- The attributes can be accessed after the fitting as follow:

```
model.coef_
model.intercept_
model.score(X,y)
```

4. Predict the train target and evaluate the model.

```
y_hat = model.predict(X)
```

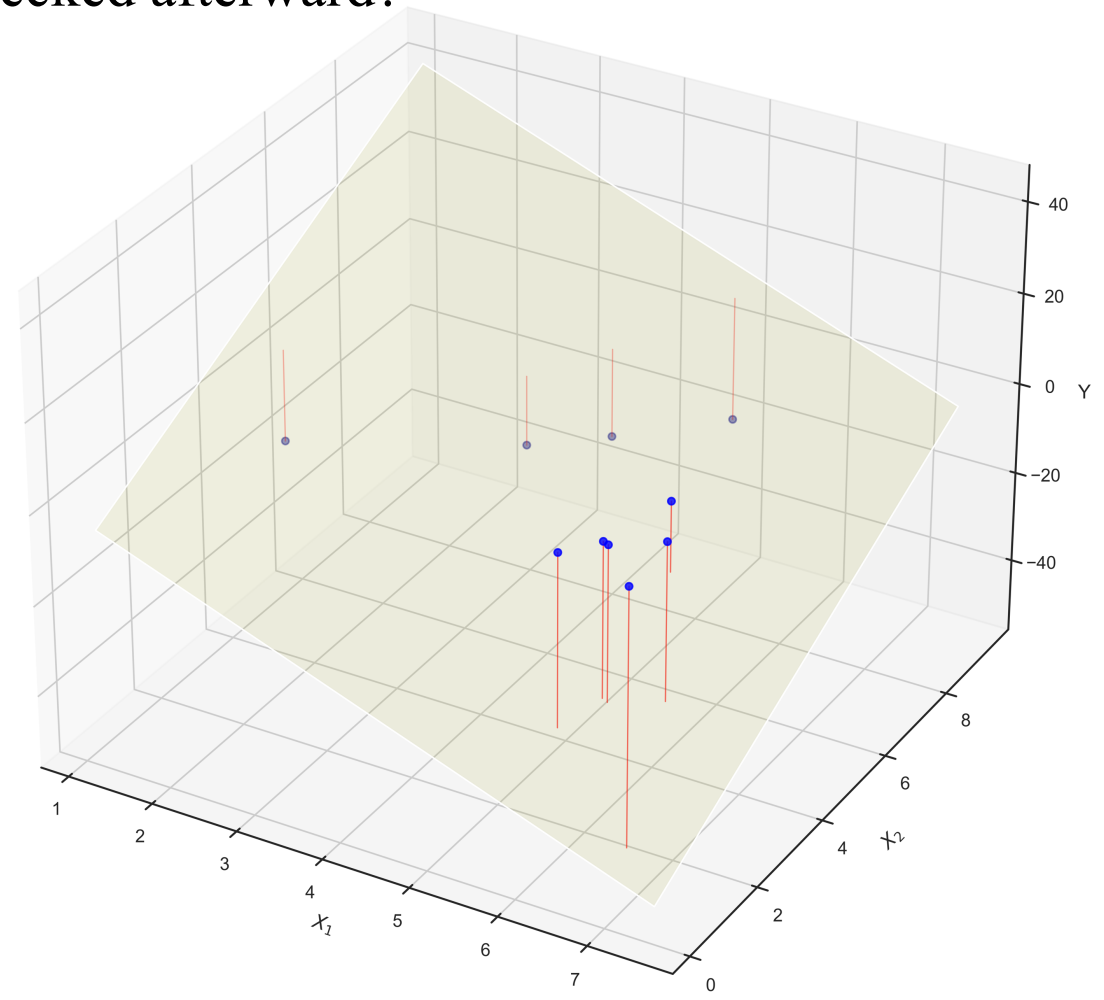
- Returns in an array format.
- Evaluation:

```
from sklearn.metrics import mean_absolute_error, mean_squared_error
mae = mean_absolute_error(y,y_hat)
mse = mean_squared_error(y,y_hat)
rmse = np.sqrt(mse)
```

# Linear Regression Assumptions

- How to build a multiple linear regression model using Scikit-learn?
- What linear regression conditions must be checked afterward?

| $x_1$ | $x_2$ | y |
|---|---|---|
| 6.11 | 2.36 | 5.35 |
| 3.76 | 5.98 | -9.74 |
| 6.98 | 0.68 | 12.89 |
| 5.96 | 4.61 | -1.06 |
| 5.06 | 8.94 | -15.95 |
| 4.32 | 7.12 | -12.12 |
| 5.84 | 1.61 | 7.23 |
| 6.57 | 2.87 | 5.20 |
| 1.63 | 4.17 | -8.71 |
| 6.05 | 2.37 | 5.68 |

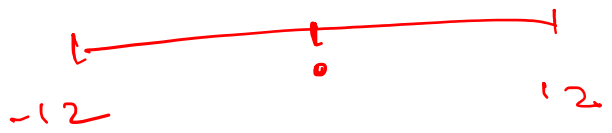


# Linear Regression Assumptions

- How to build a multiple linear regression model using Scikit-learn?
- What linear regression conditions must be checked afterward?

```
N = 10
## generate a random array with 2 columns
x1 = np.array([6.11, 3.76, 6.98, 5.96, 5.06, 4.32, 5.84, 6.57, 1.63, 6.05])
x2 = np.array([2.36, 5.98, 0.68, 4.61, 8.94, 7.12, 1.61, 2.87, 4.17, 2.37])
y = np.array([5.35, -9.74, 12.89, -1.06, -15.95, -12.12, 7.23, 5.20, -8.71, 5.68])
X = np.column_stack([x1, x2])
```

Data




```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error
model = LinearRegression()
model.fit(X,y)
```

import
build a model
fit.

```
LinearRegression()
```

```
y_hat = model.predict(X)
print(f'y_hat = {y_hat}')
```

prediction

```
y_hat = [  5.77089943  -9.80461396  12.54803658  -1.22896173 -15.95035666
 -12.05020083   7.44959054   5.19479673  -8.77760603   5.61841594]
```

```
mae = mean_absolute_error(y,y_hat)
mse = mean_squared_error(y,y_hat)
rmse = np.sqrt(mse)
print(f'MAE= {mae}, MSE= {mse},RMSE= {rmse}')
```

evaluation

```
MAE= 0.1420578266425024, MSE= 0.03883006415698549,RMSE= 0.19705345507497576
```

```
print("w1, w2: " + str(np.round(model.coef_,3)))
print("w0: " + str(np.round(model.intercept_,3)))
print("R^2 : %.5f" % model.score(X,y))
```

```
w1, w2: [ 2.046 -2.975]
w0: 0.292
R^2 : 0.99955
```

$\approx 1$

# Linear Regression Assumptions

- How to build a multiple linear regression model using Scikit-learn?
- What linear regression conditions must be checked afterward?
  - Linearity
    - Target vs. Features
    - Features vs. Features (Multicollinearity)
  - Residual ($y - \hat{h}$) conditions
    - Normality of residuals
    - Homoscedasticity vs. Heteroscedasticity
    - No autocorrelation of errors

repeated

$h = \sum w_i x_i$

$y = h + \varepsilon$

$h = w_0 + w_1 x_1 + w_2 x_2$

$\varepsilon \sim \sigma^2$

$\varepsilon, x, y \sim N$

then $\hat{h} \sim N$

$\varepsilon = y - \hat{h} \sim N$

# Linear Regression Assumptions

- Linearity between features and target
  - Check if features are linearly correlated to the target.
  - The correlation coefficient can be calculated as $\rho = \frac{\mathrm{cov}(\mathbf{x_i}, \mathbf{y})}{\sigma_{x_i}\sigma_y}$.
    - The correlation is strongly positive if $\rho \approx 1$ and strongly negative if $\rho \approx -1$.
    - The weak correlation is when $0 \leq \rho \leq 0.5$. $-0.5 \leq \rho \leq 0.5$

```
1 np.corrcoef(x1, y)
```

```
array([[1.        , 0.71886325],
       [0.71886325, 1.        ]])
```

```
1 np.corrcoef(x2, y)
```

```
array([[ 1.        , -0.95595136],
       [-0.95595136,  1.        ]])
```

```
1 cov = np.sum((x1-np.mean(x1))*(y-np.mean(y)))/N
2 cov/(np.std(x1)*np.std(y))
```

```
0.7188632519166426
```

```
1 cov = np.sum((x2-np.mean(x2))*(y-np.mean(y)))/N
2 cov/(np.std(x2)*np.std(y))
```

```
-0.955951359407545
```

```
1 import scipy.stats
2 r, p = scipy.stats.pearsonr(x1, y)
3 print(f'spearman r= {r}, kendall p= {p}')
```

```
spearman r= 0.7188632519166428, kendall p= 0.019146906325484662
```

```
1 r, p = scipy.stats.pearsonr(x2, y)
2 print(f'spearman r= {r}, kendall p= {p}')
```

```
spearman r= -0.955951359407545, kendall p= 1.5615821645140453e-05
```

- `numpy.corrcoef(x,y)`: calculates the the correlation coefficient.
- Alternatively, `scipy.stats.pearsonr(x,y)` can be used. It calculates the correlation coefficient and the p-value of the distribution between two variables.
  - Typically, we reject the null hypothesis with a significant level of 95% when $p<0.05$.

# Linear Regression Assumptions

- Multicollinearity: Make sure that features are **not correlated to each other**!
  - If $\rho(\mathbf{x}_1, \mathbf{x}_2) \approx 1$, the original model, $\mathbf{h} = w_0 + w_1\mathbf{x}_1 + w_2\mathbf{x}_2$, can be replaced with
    - $\mathbf{h} = a_0 + a_1\mathbf{x}_1$ or $\mathbf{h} = b_0 + b_1\mathbf{x}_2$

```
1 import pandas as pd
```

```
1 df = pd.DataFrame({'x1':x1, 'x2':x2, 'y':y})
```

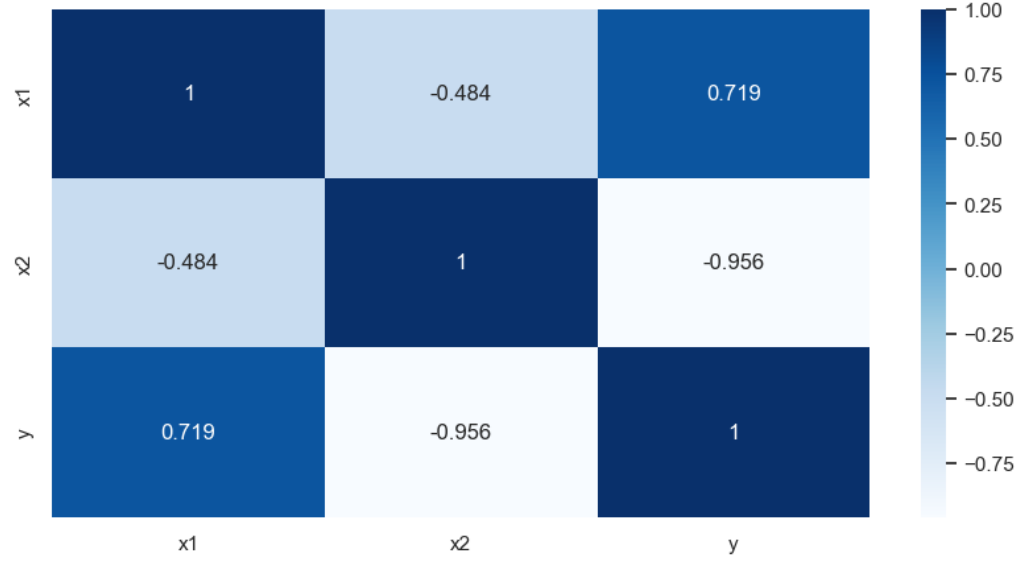
```
1 df.corr()
```

| | x1 | x2 | y |
|---|---|---|---|
| **x1** | 1.000000 | -0.483996 | 0.718863 |
| **x2** | -0.483996 | 1.000000 | -0.955951 |
| **y** | 0.718863 | -0.955951 | 1.000000 |

```
1 corr = df.corr()
2 corr.style.background_gradient(cmap='coolwarm')
```

| | x1 | x2 | y |
|---|---|---|---|
| **x1** | 1.000000 | -0.483996 | 0.718863 |
| **x2** | -0.483996 | 1.000000 | -0.955951 |
| **y** | 0.718863 | -0.955951 | 1.000000 |

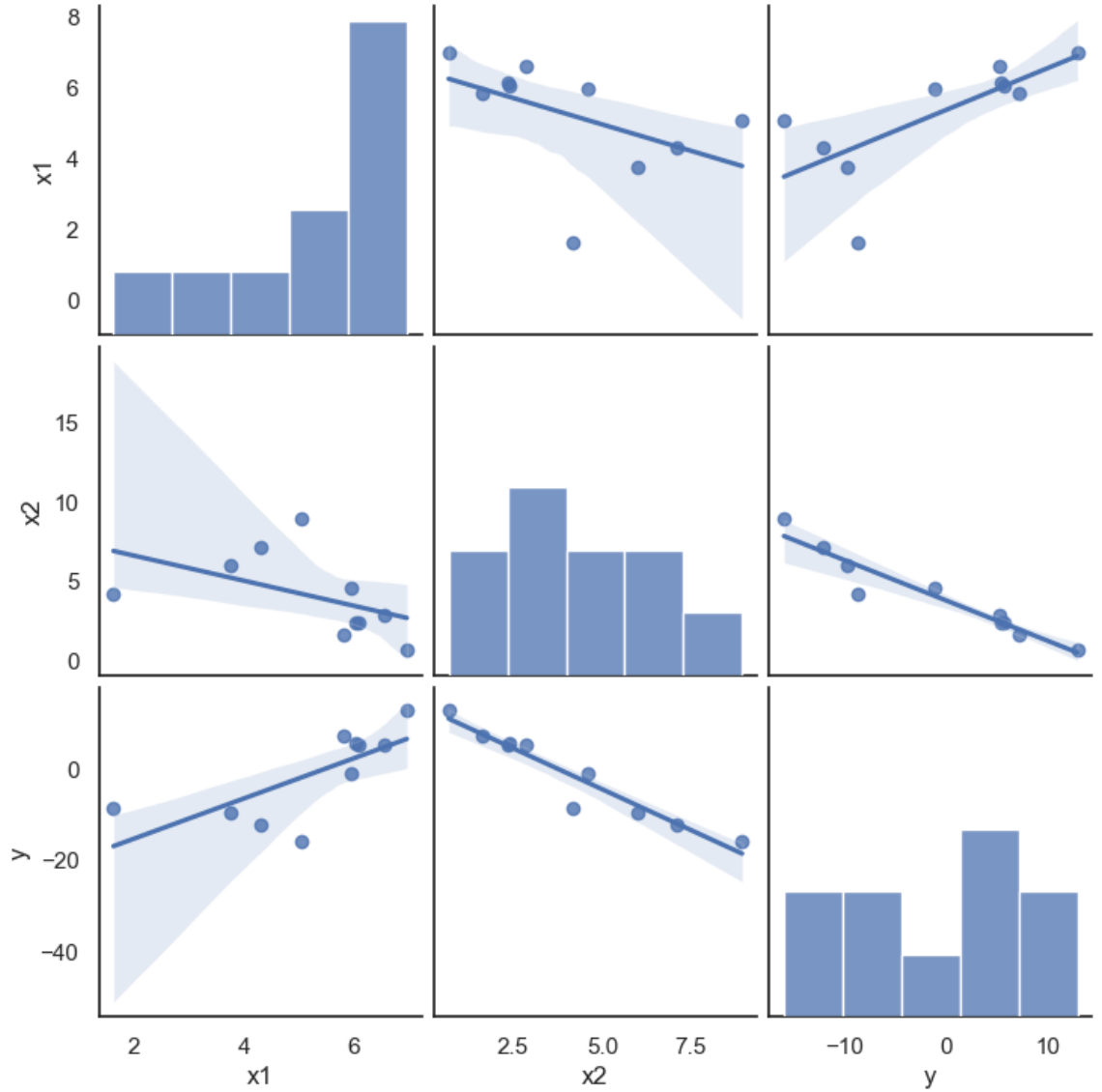
```
1 import matplotlib.pyplot as plt
2 import seaborn as sns
3
4
5 plt.figure(figsize=(10,5))
6 sns.set_theme(style="white")
7 corr = df.corr()
8 heatmap = sns.heatmap(corr, annot=True, cmap="Blues", fmt='.3g')
```



# Linear Regression Assumptions

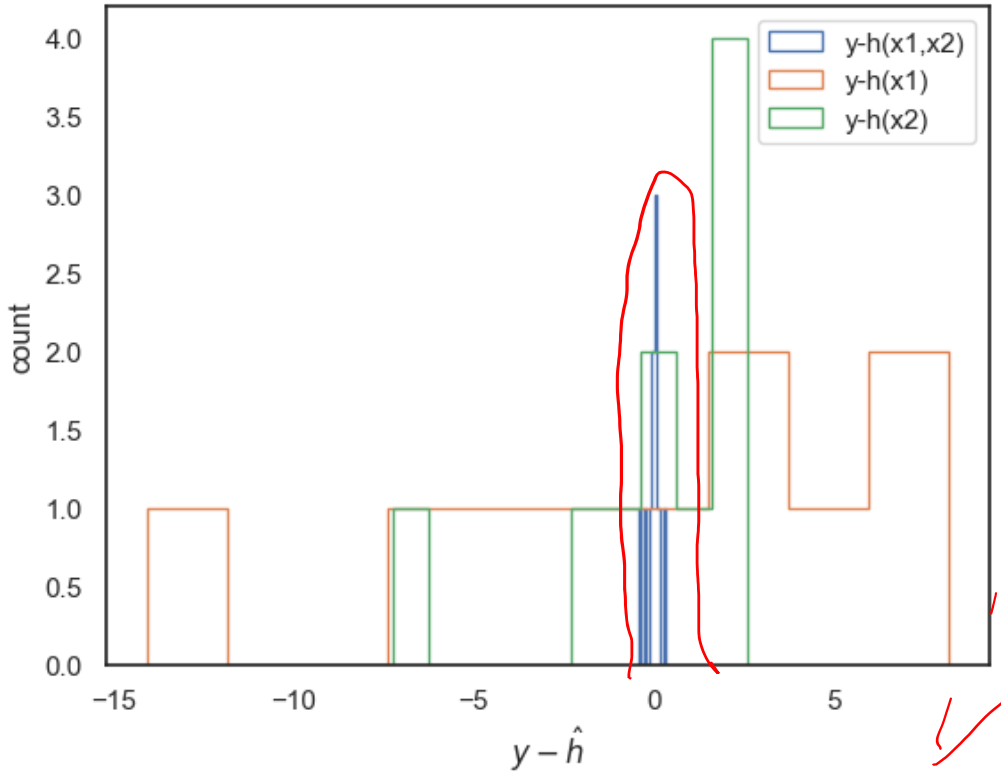
```
1 sns.pairplot(df, kind="reg")
```

```
<seaborn.axisgrid.PairGrid at 0x2c169fb50>
```



# Linear Regression Assumptions

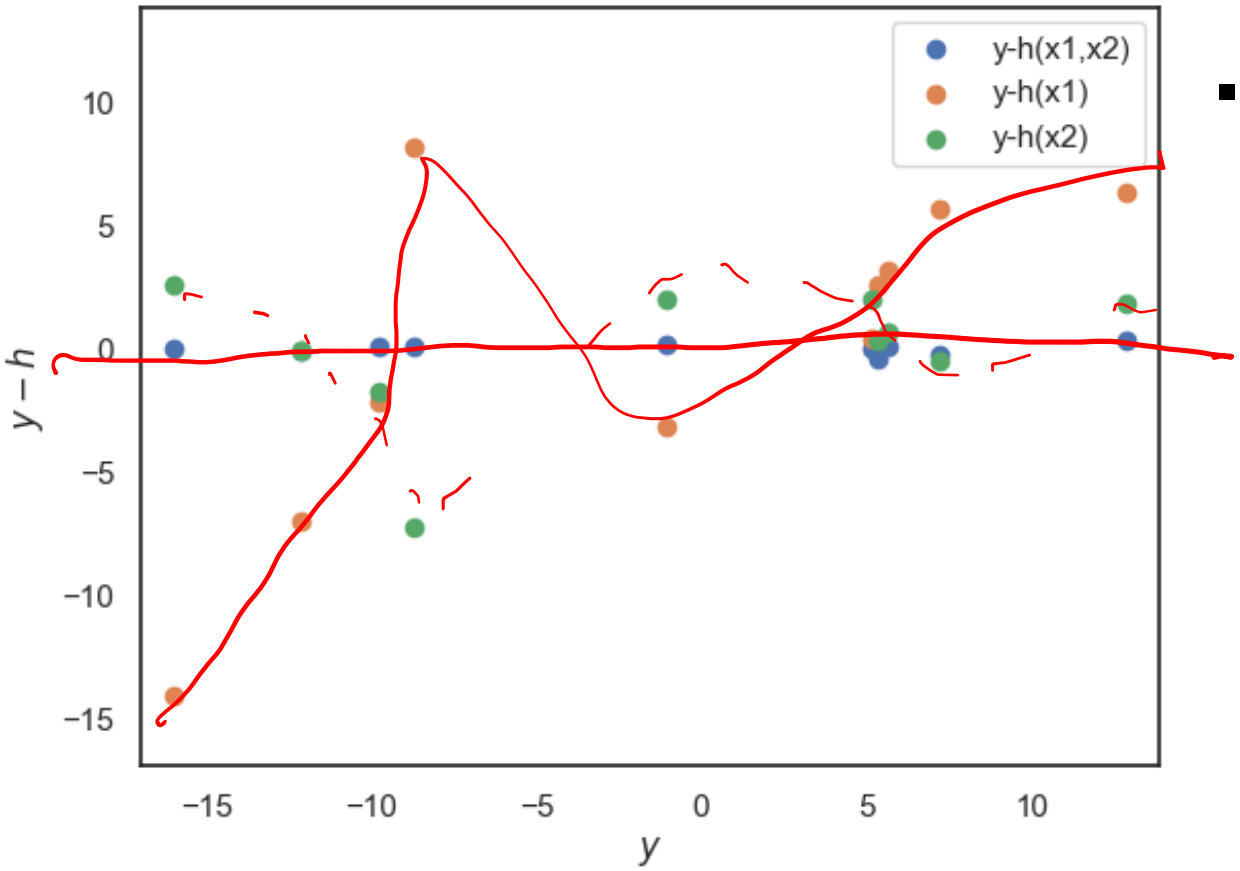
- Normality of Residuals: the mean of residuals is around 0.
  - They are distributed systematically around 0 with no skewness.
  - This implies that the model captures the main patterns of variations in the data and that errors are random and independent.




- Compare how the residuals are distributed between models
  - $\mathbf{h} = w_0 + w_1\mathbf{x}_1 + w_2\mathbf{x}_2$ (blue)
  - $\mathbf{h} = a_0 + a_1\mathbf{x}_1$ (orange)
  - $\mathbf{h} = b_0 + b_2\mathbf{x}_2$ (green)

# Linear Regression Assumption

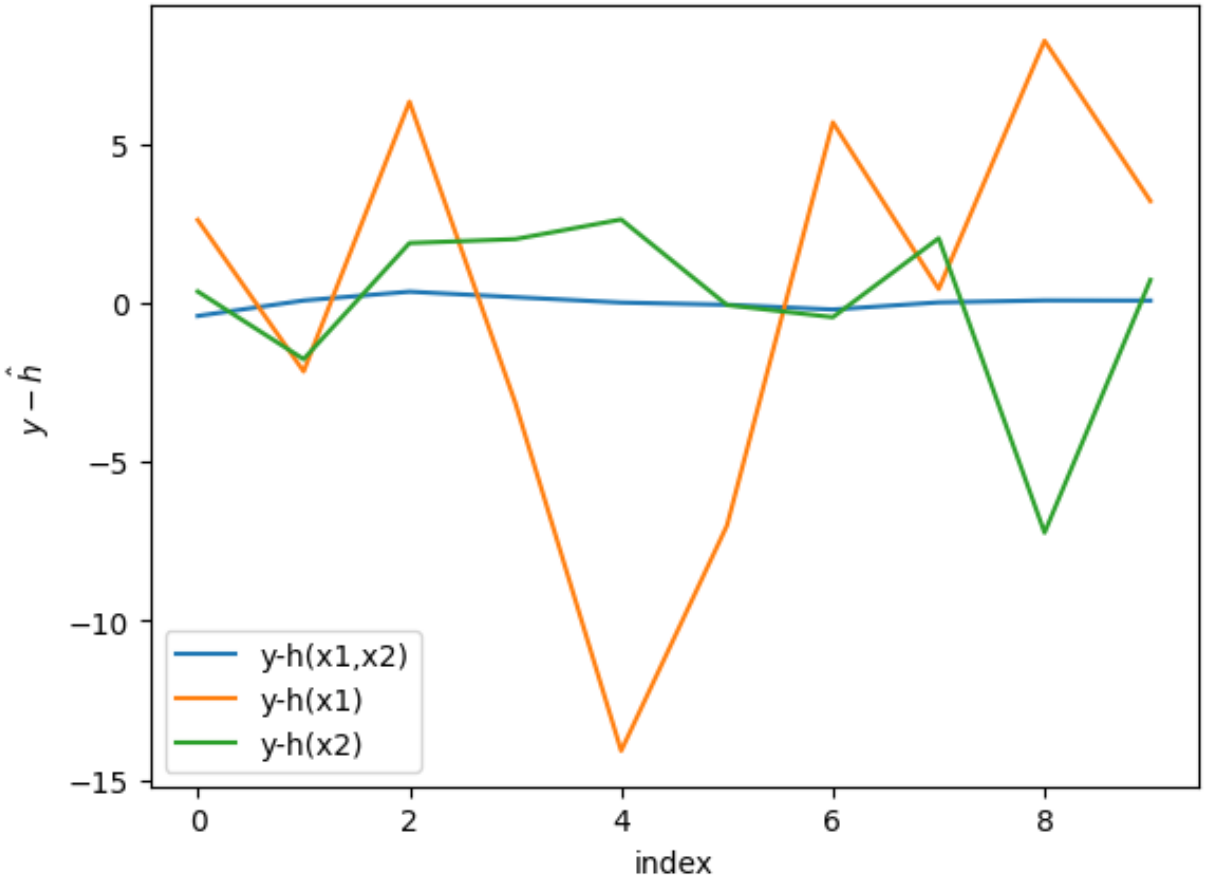
- Homoscedasticity (equal spread of residuals)
  - The model is sensitive to uneven variances (or residuals).
  - The residuals tend to be heteroscedastic if the data is not in Gaussian distribution.




- Compare how the residuals (y-axis) are spread between models
  - $\mathbf{h} = w_0 + w_1\mathbf{x}_1 + w_2\mathbf{x}_2$ (blue)
  - $\mathbf{h} = a_0 + a_1\mathbf{x}_1$ (orange)
  - $\mathbf{h} = b_0 + b_2\mathbf{x}_2$ (green)

# Linear Regression Assumption

- No autocorrelation of errors (no relationship between residuals)




- Compare how the residuals (y-axis) are changing between models
  - $\mathbf{h} = w_0 + w_1\mathbf{x}_1 + w_2\mathbf{x}_2$ (blue)
  - $\mathbf{h} = a_0 + a_1\mathbf{x}_1$ (orange)
  - $\mathbf{h} = b_0 + b_2\mathbf{x}_2$ (green)

$h(x_1, x_2) > h(x_2)$ or $h(x_1)$

# NumPy Implementation

- ML algorithms can be implemented if the function (loss function) and mathematical operations.
- Simple implementation can be done using the method seen in the gradient descent example.
- Alternatively, the implementation can be made as a class with a group of methods, similar to the Scikit-learn algorithm.
- Linear_regression(learning rate, iterations)
  - __init__(self, learning rate, iterations): Declare the parameters to be passed.
  - fit(self, X, y): pass features and target to estimate the weights.
  - update_weights(self): pass features, target, and parameters to update weights.
  - predict(self, X): predict the target using the updated weights.

# NumPy Implementation

```
class Linear_regression():
    def __init__(self, learning_rate, iterations):
        self.learning_rate = learning_rate
        self.iterations = iterations

    # Model Training Function
    def fit( self, X, Y):
        # Number of training example, Number of features
        self.m, self.n = X.shape

        # Weight initialization
        self.W = np.zeros( self.n)
        self.b = 0
        self.X = X
        self.Y = Y

        # Gradient Descent Learning
        for i in range(self.iterations):
            self.update_weights()
        return self
```

$h = w_0 + w_1 x_1 + \cdots + x_0 w_0$

$1 \times D$

# NumPy Implementation

```
class Linear_regression():
    def __init__(self, learning_rate, iterations):

    # Model Training Function
    def fit( self, X, Y):

    # Function to update weights in Gradient Descent
    def update_weights(self):
        Y_pred = self.predict( self.X)
        # Gradient calculations
        dW = - (2 * (self.X.T).dot( self.Y - Y_pred)) / self.m
        db = - 2 * np.sum( self.Y - Y_pred) / self.m
        # Weight updates
        self.W = self.W - self.learning_rate * dW
        self.b = self.b - self.learning_rate * db
        return self

    # Hypthetical Function
    def predict( self, X):
        return X.dot( self.W) + self.b
```

$E = \frac{1}{m} \sum (y - h)^2$

$= \sum_{n=1} X W + W_0$

# NumPy Implementation

```
model = Linear_regression(learning_rate = 0.001, iterations = 1000)
model.fit(X,y)
Y_pred = model.predict(X)
```

```
print(f'W= {model.W}')
print(f'b= {model.b}')
```

```
W= [ 2.07107127 -2.96252343]
b= 0.10300792032032006
```

```
mae = mean_absolute_error(y,Y_pred)
mse = mean_squared_error(y,Y_pred)
rmse = np.sqrt(mse)
print(f'MAE= {mae}, MSE= {mse},RMSE= {rmse}')
```

```
MAE= 0.1560926649733415, MSE= 0.04015886552855974,RMSE= 0.200396770254811
```

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error
model = LinearRegression()
model.fit(X,y)
```

```
▾ LinearRegression
LinearRegression()
```

```
y_hat = model.predict(X)
print(f'y_hat = {y_hat}')
```

```
y_hat = [  5.77089943  -9.80461396  12.54803658  -1.22896173 -15.95035666
 -12.05020083   7.44959054   5.19479673  -8.77760603   5.61841594]
```

```
mae = mean_absolute_error(y,y_hat)
mse = mean_squared_error(y,y_hat)
rmse = np.sqrt(mse)
print(f'MAE= {mae}, MSE= {mse},RMSE= {rmse}')
```

```
MAE= 0.1420578266425024, MSE= 0.03883006415698549,RMSE= 0.19705345507497576
```

```
print("w1, w2: " + str(np.round(model.coef_,3)))
print("w0: " + str(np.round(model.intercept_,3)))
print("R^2 : %.5f" % model.score(X,y))
```

```
w1, w2: [ 2.046 -2.975]
w0: 0.292
R^2 : 0.99955
```

# Conclusion

Overall, the implementation and interpretation of the linear regression model are straightforward. But,

- Simplicity: The model is too simple to capture the complexity of real data.
  - It is an easy start to modeling but will hardly be a final model.
  - It is essential to have different models.
- Assumptions are not realistic.
  - Many real-life data sets are much more complicated than linearity and normality.
  - It may require feature engineering, which often takes longer than the modeling.
  - Sensitivity: The model is sensitive to outliers as they are accounted for in the estimation, causing high variance and bias.
  - Remove outliers.