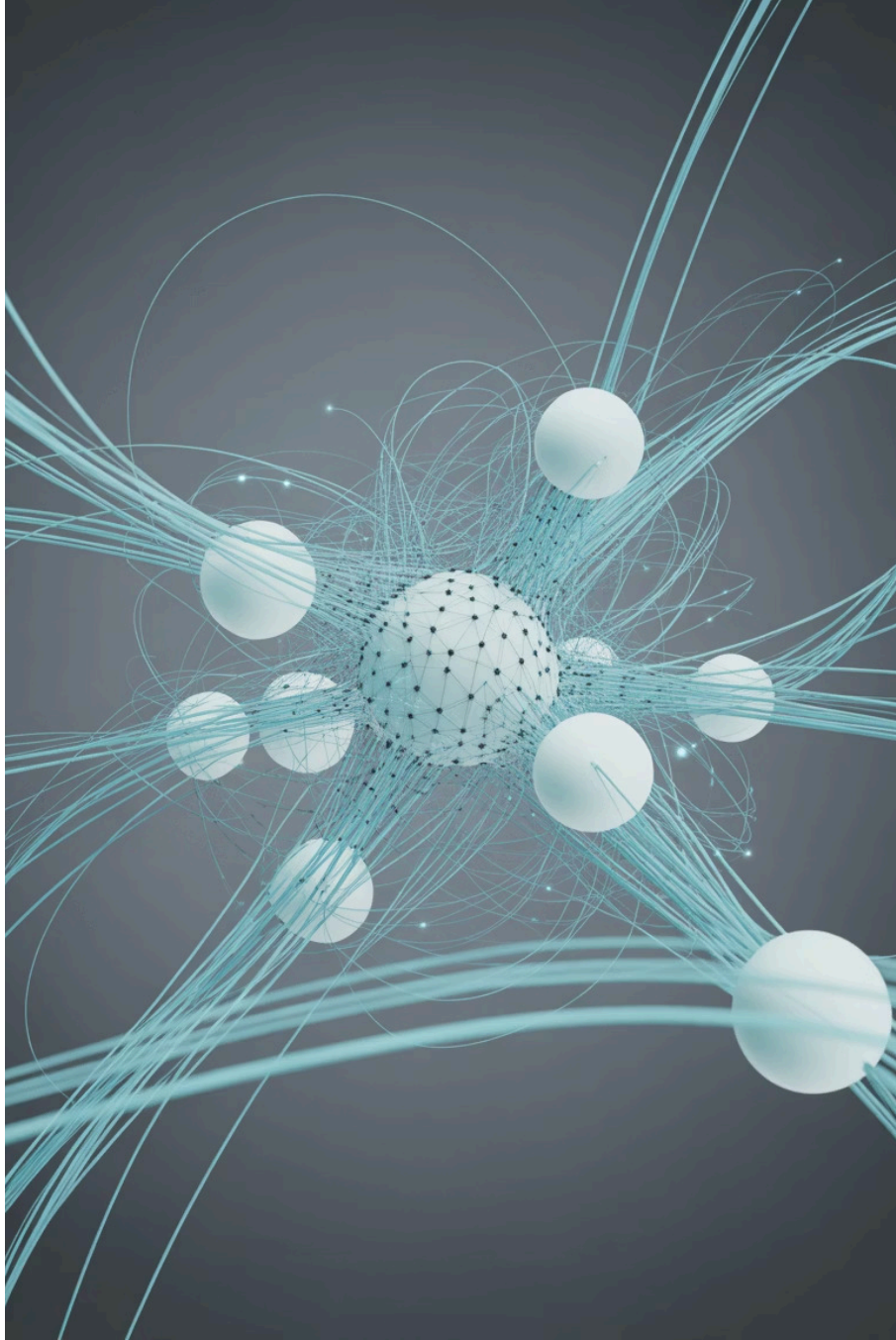
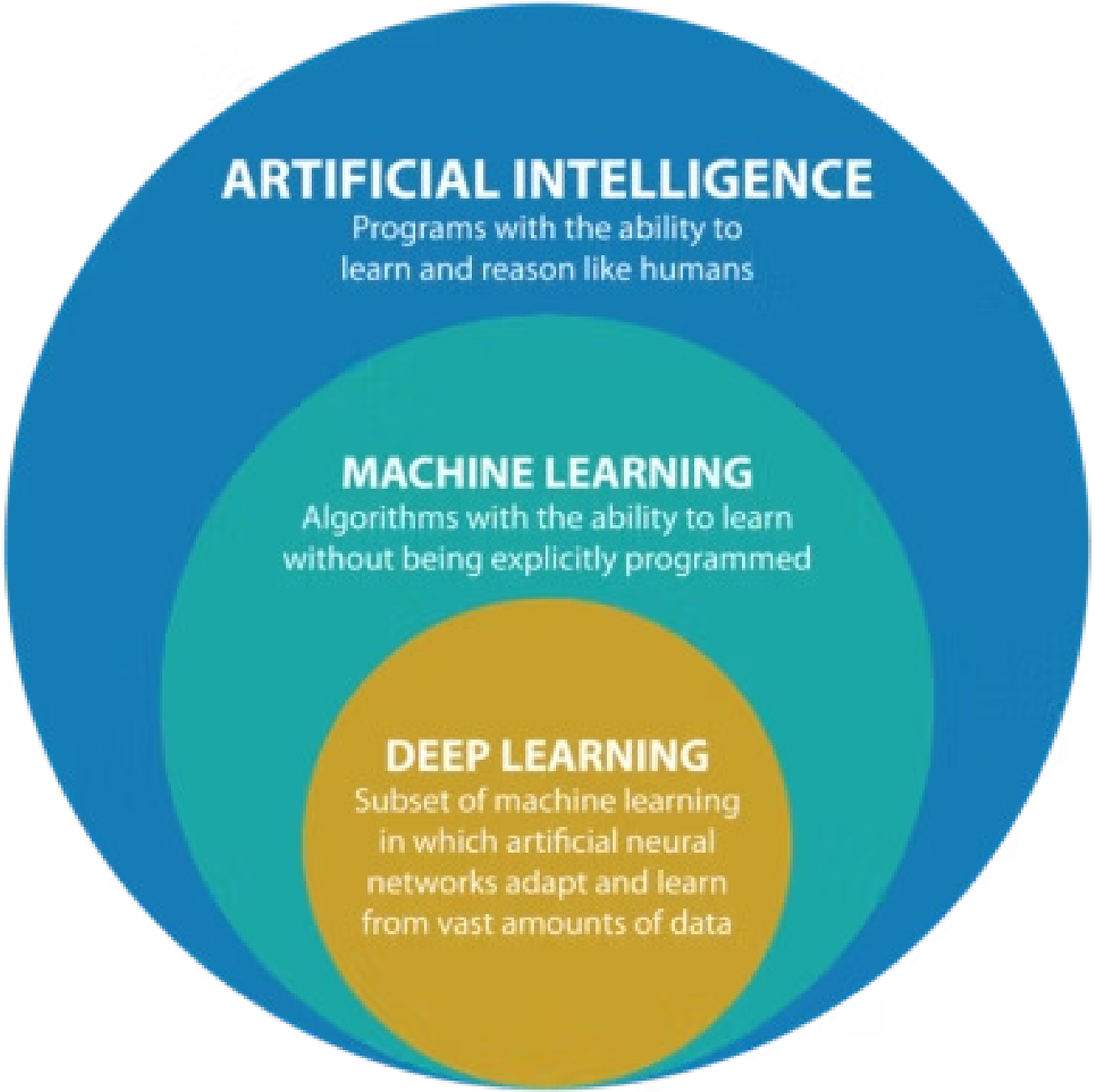
Machine Learning

Jonathan Zouari, Phd

email: Jonathan.Zouari@mail.huji.ac.il




ARTIFICIAL INTELLIGENCE
Programs with the ability to
learn and reason like humans
MACHINE LEARNING
Algorithms with the ability to learn
without being explicitly programmed
DEEP LEARNING
Subset of machine learning
in which artificial neural
networks adapt and learn
from vast amounts of data

What Is Machine Learning (ML)?

Core Concept

Machine Learning is the interdisciplinary field focused on enabling computers to **learn from data** without being explicitly programmed. It involves designing algorithms that can identify patterns and make predictions or decisions based on input data.

Arthur Samuel's Definition (1959)

A pioneer in AI, Arthur Samuel defined machine learning as a field of study that gives computers the ability to **learn without being explicitly programmed**.

Example: Spam Filter

Spam Filter Overview

A spam filter serves as an excellent example of an ML system. It is specifically designed to distinguish between spam and legitimate ("ham") emails by learning from a vast collection of labeled examples.

Training Set

The system's learning process relies on a **training set**, which is a comprehensive dataset of emails meticulously labeled as either spam or ham.

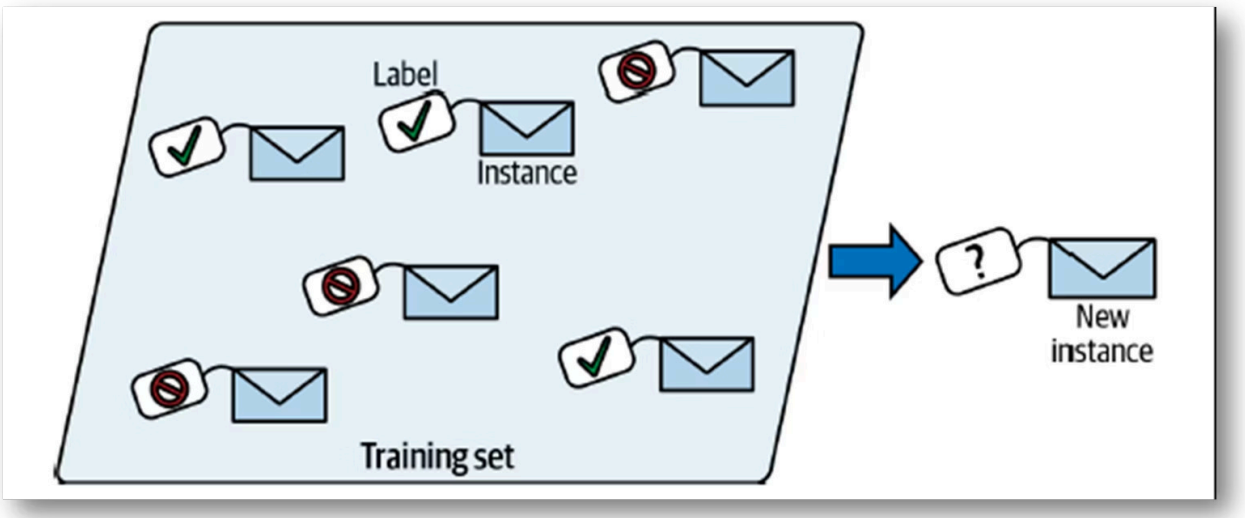
Training Instance (Sample)

Within this training set, each individual email represents a **training instance**, also known as a sample. These instances are the building blocks from which the ML system learns patterns.

The ML Model

Ultimately, the system constructs a **model**. This model is the core component of any ML system, learning intricate patterns from the data to accurately classify new, incoming emails as either spam or ham.

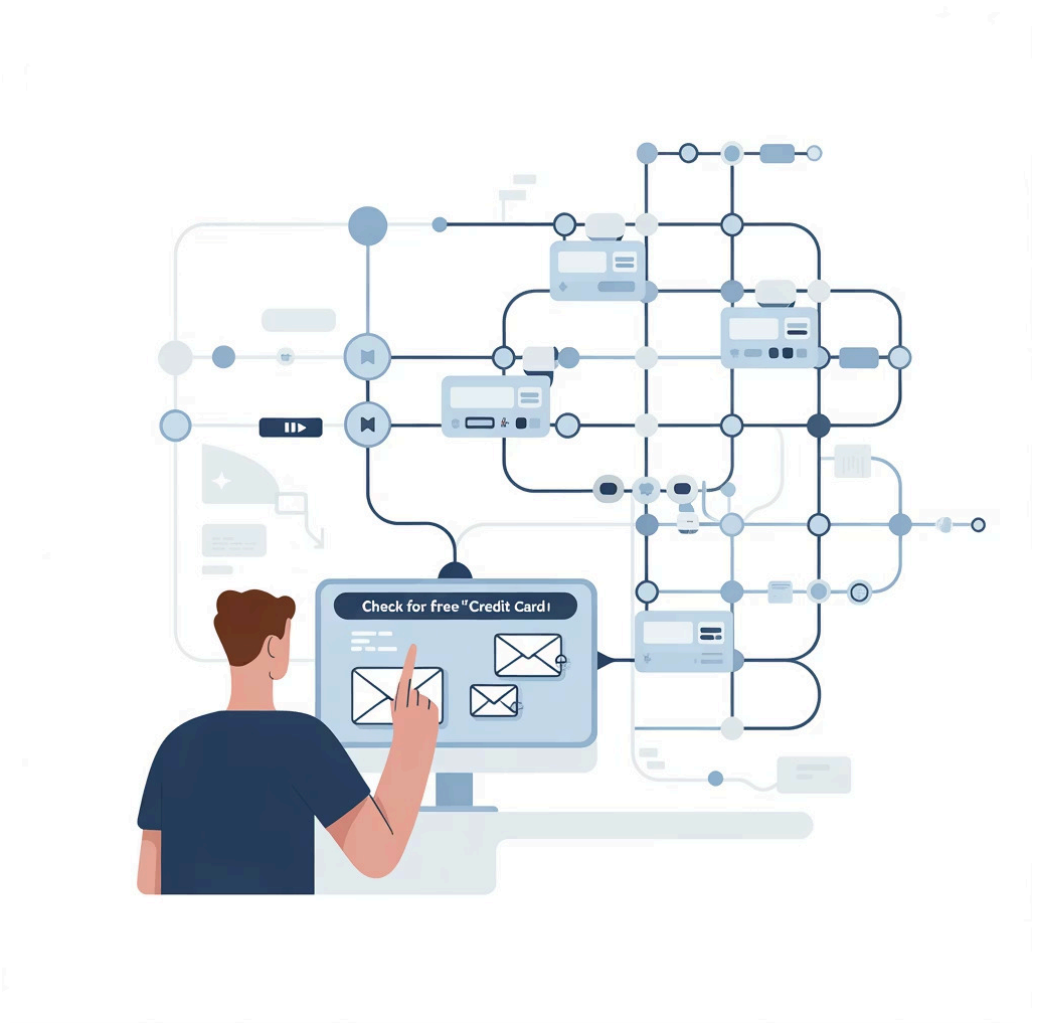
Example: Spam Filter



Why Use Machine Learning?

Understanding the fundamental differences between traditional programming and machine learning is key to appreciating the effectiveness of modern spam filtering solutions.

Traditional Programming: Rule-Based Filtering



Traditional spam filtering relies on manually defined, explicit rules based on observed patterns like keywords. This approach leads to complex rule sets that are difficult to maintain and struggle to adapt to evolving spam tactics.

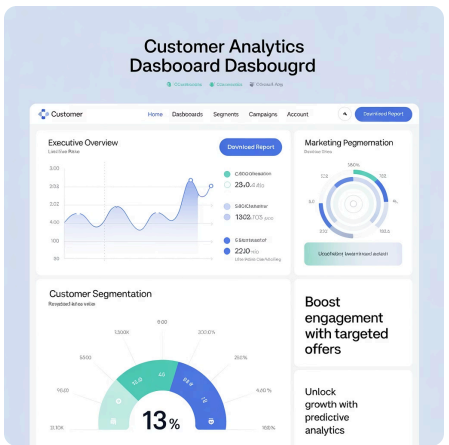
Machine Learning: Data-Driven Filtering

Machine learning uses a data-driven approach, learning to identify spam from vast datasets of labeled emails. The ML model automatically discovers patterns, adapting and improving its filtering as new spam emerges, resulting in a more robust and scalable solution.

ML Applications

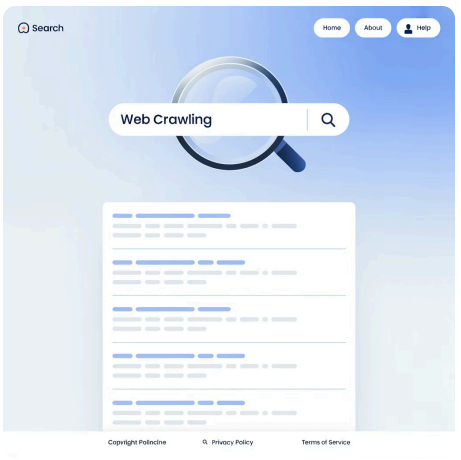
Exploring diverse applications of Machine Learning across industries

Marketing & Customer Insights



- Predict which customers are most likely to respond to promotions.
- Identify persuadable customers and adjust marketing strategies accordingly.
- Anticipate customer churn to take proactive retention actions.
- Provide personalized product recommendations.

Search & Web Intelligence



- Power search engines to deliver the most relevant results for a query (e.g., Google, Bing).
- Enhance content ranking, autocomplete suggestions, and related query recommendations.

Digital Advertising

- Match ads to users based on their interests and online behavior.
- Optimize ad placement to maximize engagement and conversions.

Finance & Trading

- Support algorithmic trading systems for real-time stock market decisions.
- Detect unusual transactions and forecast market trends.

ML Applications (2)

Exploring diverse applications of Machine Learning across industries

Communication & Filtering

- Classify emails and messages as either spam or legitimate.
- Enable smart inboxes and automated message routing.

Security & Fraud Detection

- Detect potential fraud in online financial transactions.
- Identify suspicious user behavior on digital platforms.
- Support cybersecurity systems in identifying threats.

Healthcare & Medical Research

- Predict disease risks based on patient data.
- Recommend personalized treatment plans.
- Enable early diagnosis and optimize medical resources.

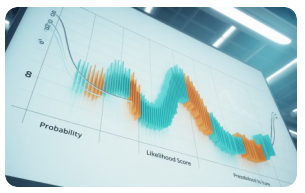
Emerging Applications

- Use biosurveillance to monitor health trends and detect outbreaks.
- Apply churn prediction in telecom to reduce customer loss.

ML Tasks: Understanding Machine Learning Objectives

Classification

Categorizes data into predefined groups. E.g., labeling emails as 'spam' or 'not spam.'



Class Probability Estimation (Scoring)

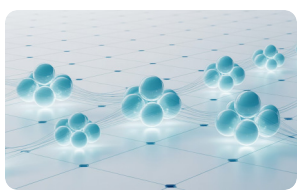
Estimates the probability a data point belongs to a class. E.g., likelihood of a customer purchasing (70% probability).

Regression (Value Estimation)

Predicts a specific numerical value. E.g., forecasting house prices or stock values.

Similarity Matching

Identifies similar individuals or items based on characteristics. E.g., recommending products to customers with similar tastes.

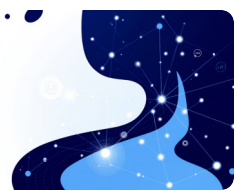


Clustering

Groups individuals into distinct segments based on inherent similarities. E.g., finding natural groupings in customer data.

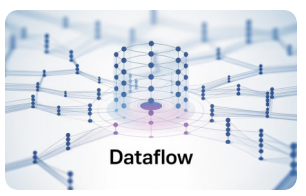
Co-occurrence Grouping

Finds items that frequently appear together. E.g., customers buying bread also buy milk ('market basket analysis').



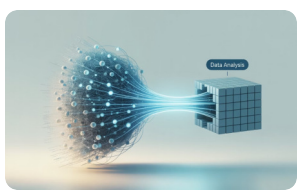
Profiling (Behavior Description)

Describes typical characteristics or behaviors of an individual/group. E.g., identifying normal network activity to detect anomalies.



Link Prediction

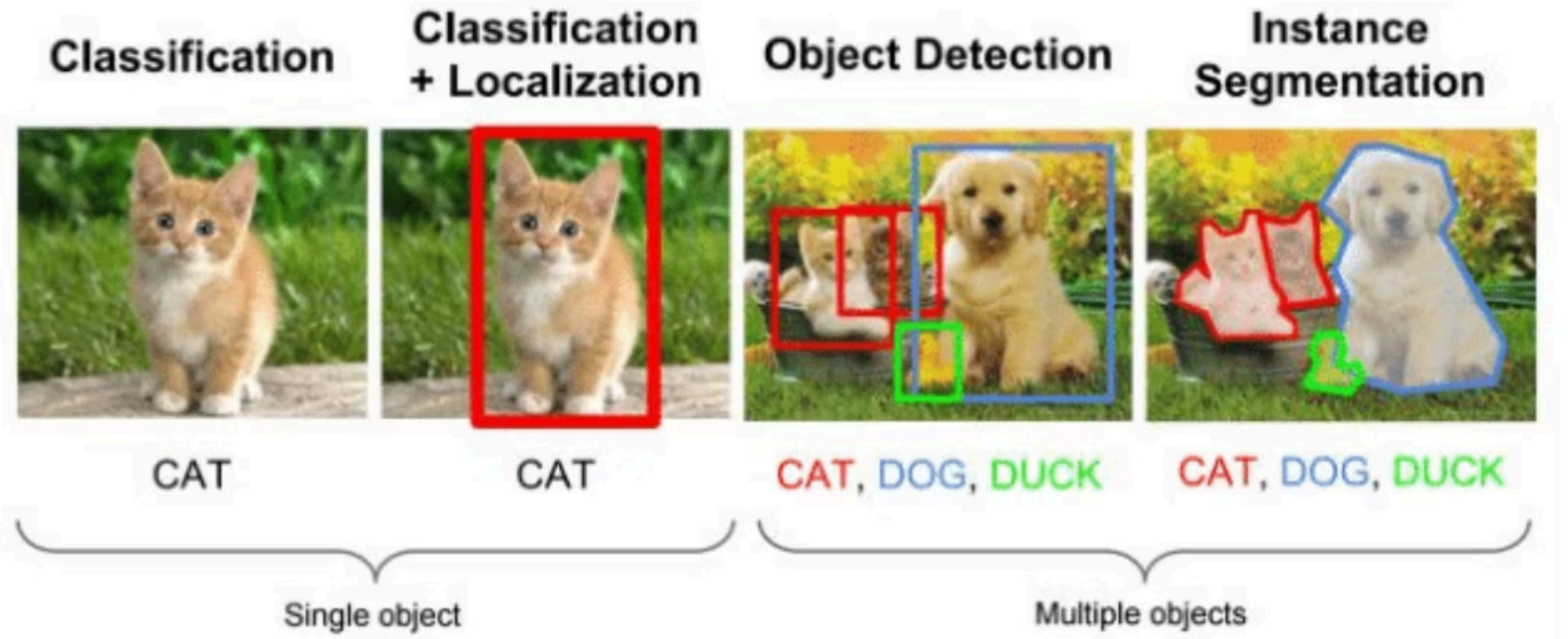
Predicts connections or relationships between data items. E.g., recommending friends on social media.



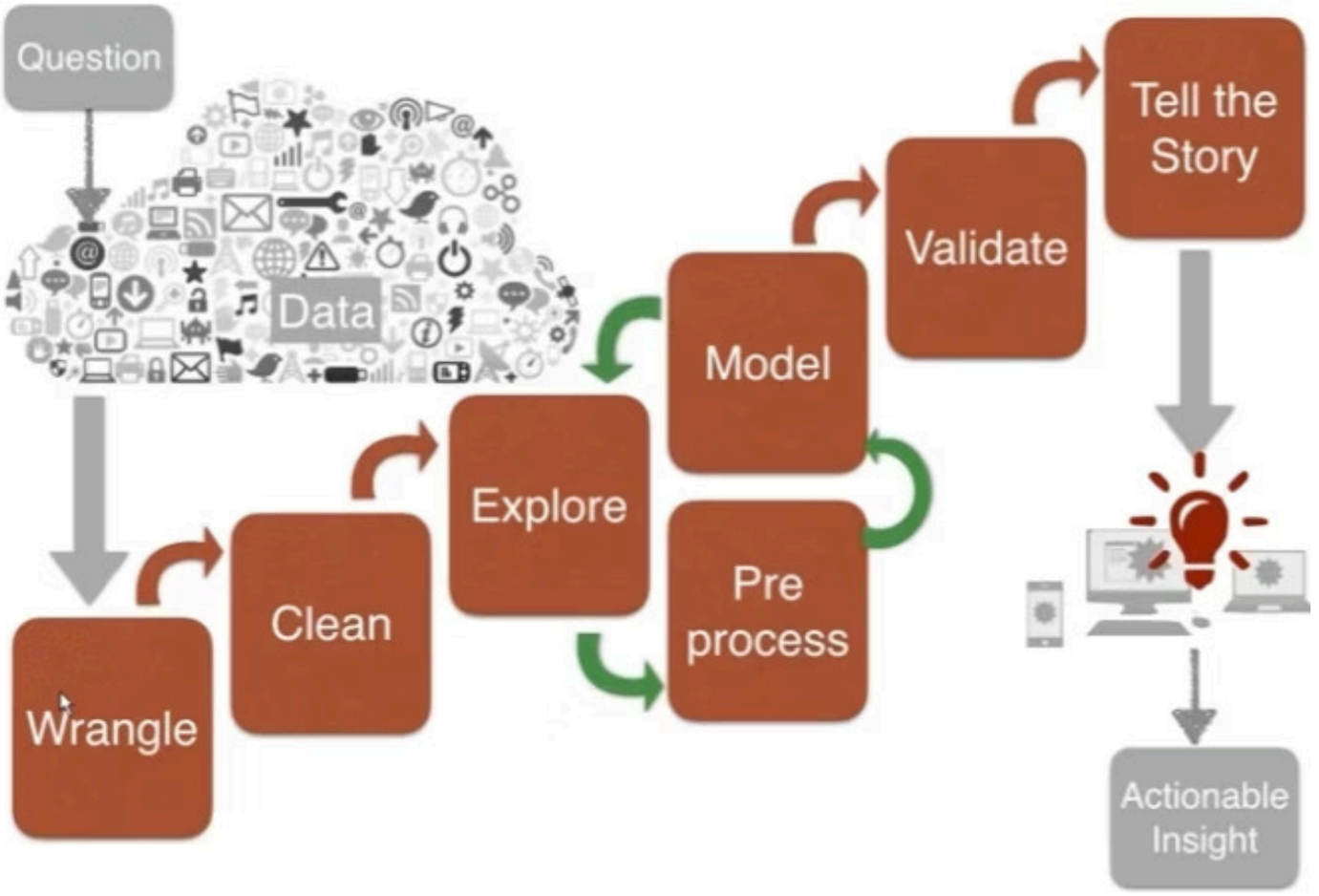
Data Reduction

Transforms large datasets into smaller, more manageable versions, preserving key information. E.g., simplifying data for analysis.

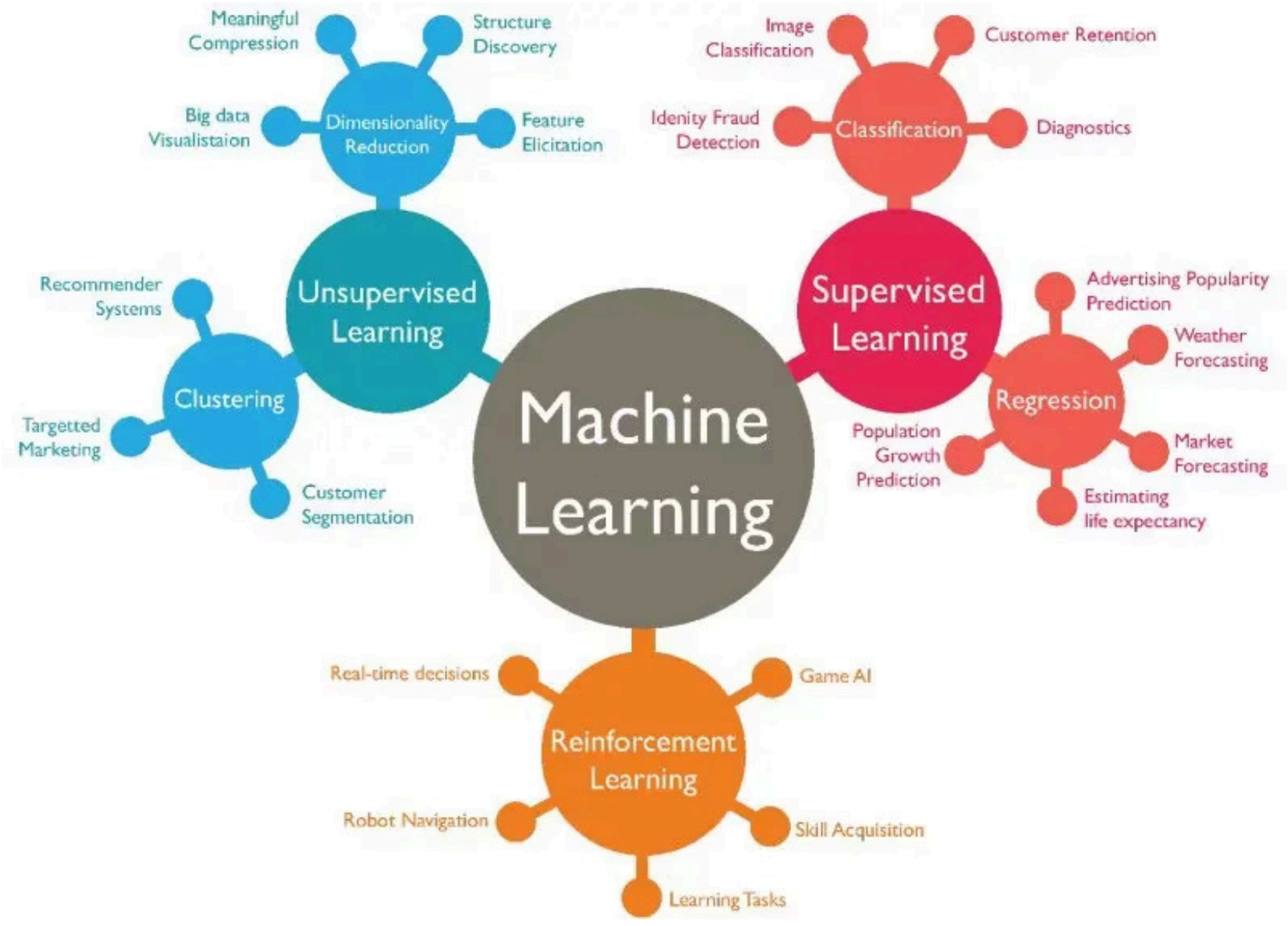
Computer Vision Tasks



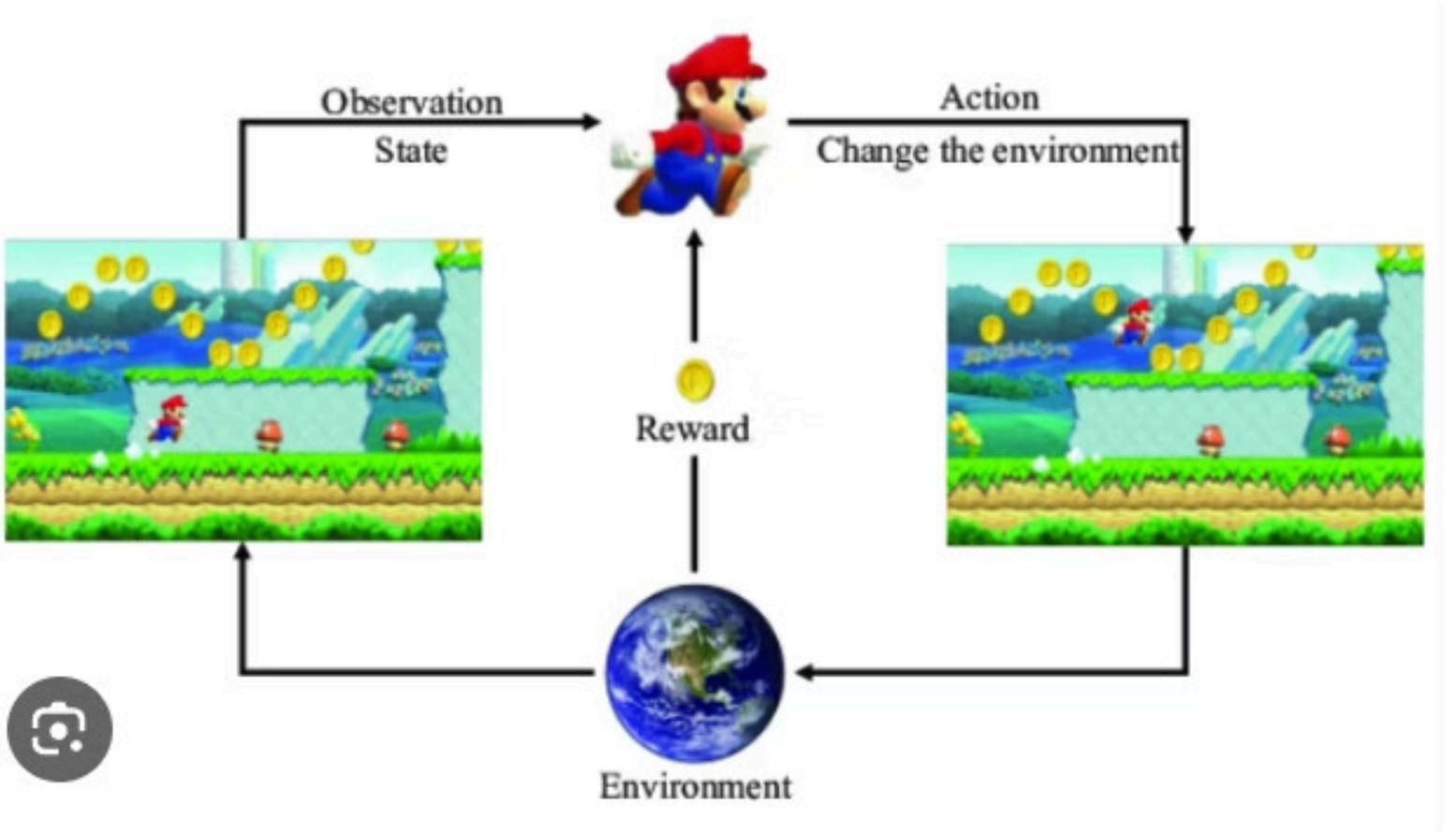
Data Science Pipeline



Types of Machine Learning and Their Applications



Reinforcement Learning

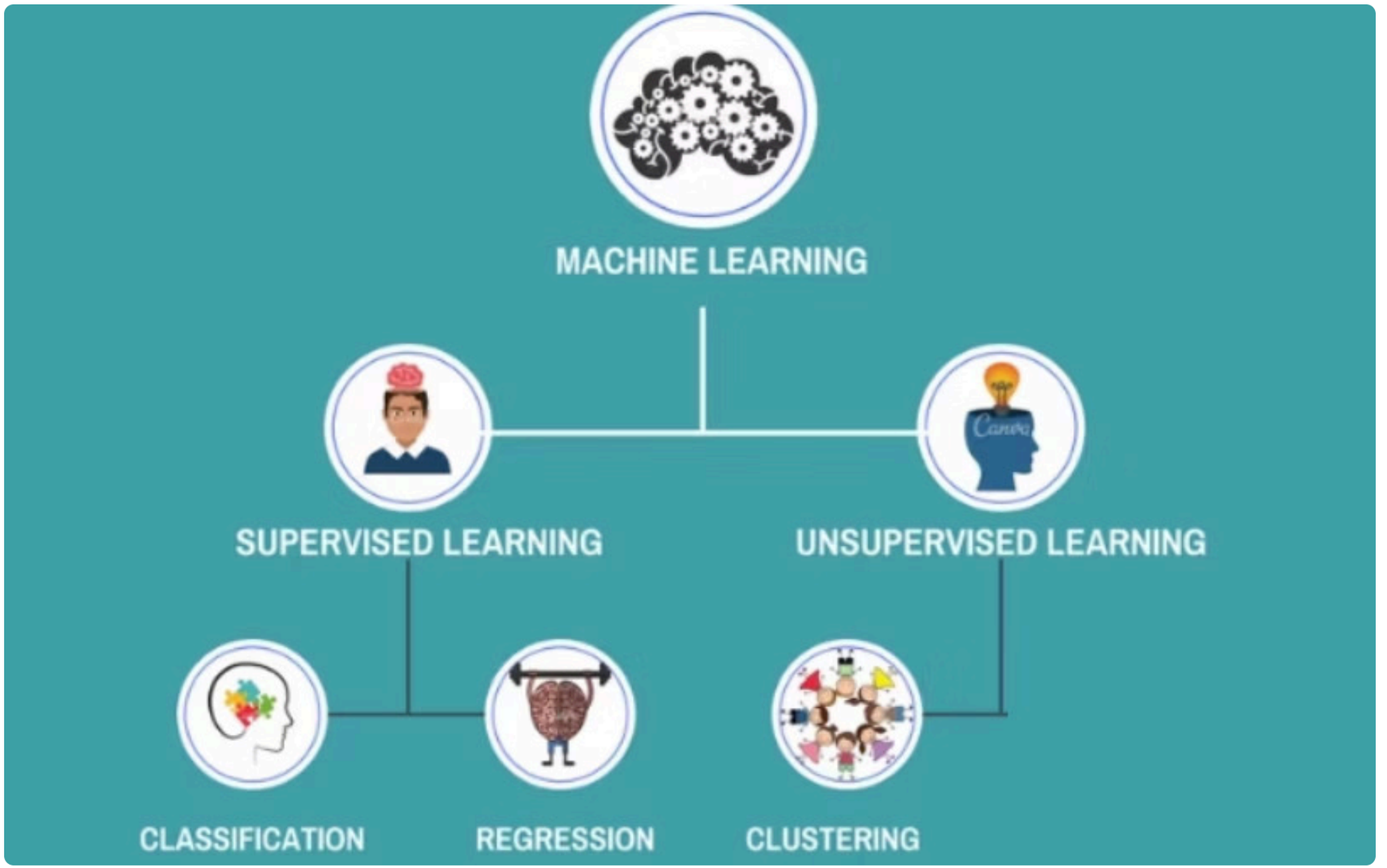


YouTube

Reinforcment Learning – Mario Training Video

I build a Double Q RL model using a convolutional network to train an AI to play Super Mario. this video shows the evolution of the training process as the AI learns to navigate...

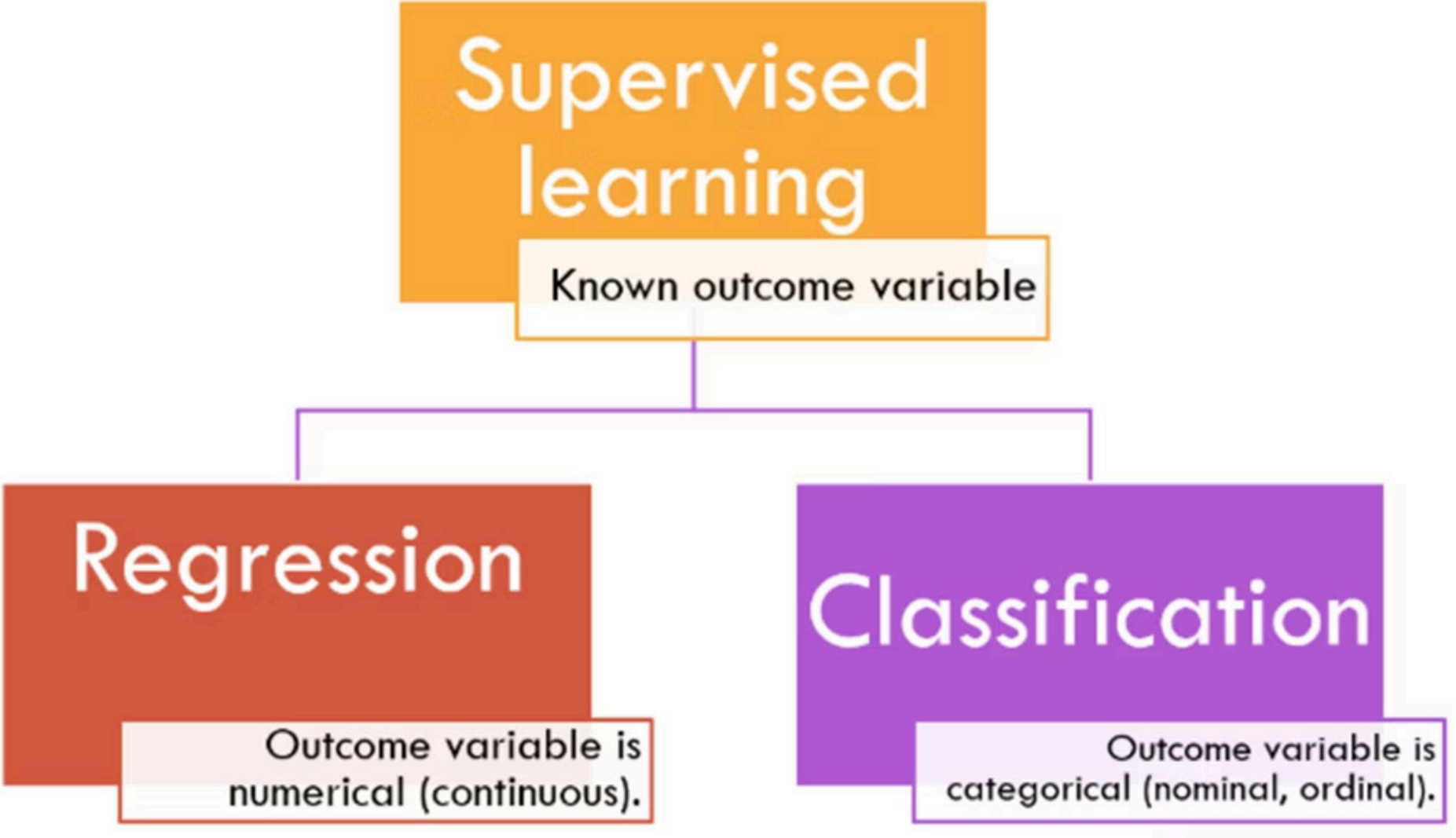
Supervised and Unsupervised Learning



Comparison of Machine Learning Types

	Supervised Learning	Unsupervised Learning
Goal	Learn a mapping from inputs to known outputs using labeled data	Discover hidden patterns or structure in unlabeled data
Input Data	Labeled data (input + correct output)	Unlabeled data
Output	Class label (classification) or numeric value (regression)	Clusters, groups, or low-dimensional structure
Examples	Email spam detection, housing price prediction	Customer segmentation, anomaly detection

Types of ML Models (Supervised Learning)



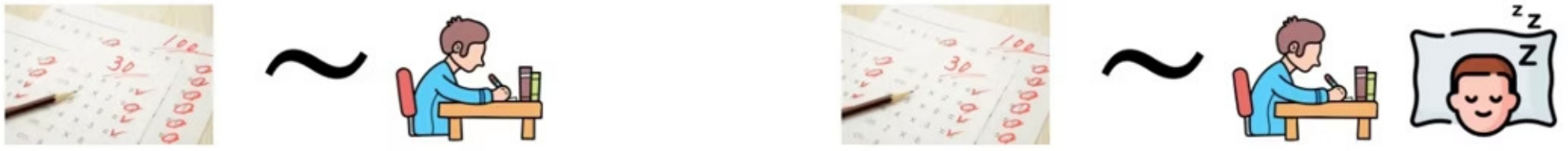
Comparison of Supervised Learning Models

	Regression Model	Classification Model
Goal	Predict continuous numerical values	Predict categories or labels
Output Type	Real-valued output (e.g., Price = $256.7)	Discrete classes (e.g., Spam / Not Spam)
Examples	House price prediction, Forecasting temperature	Email spam detection, Image recognition (Cat/Dog)
Loss Function	Mean Squared Error (MSE), MAE	Cross-Entropy, Log Loss
Evaluation	RMSE, MAE, R^2 Score	Accuracy, Precision, Recall, F1-Score

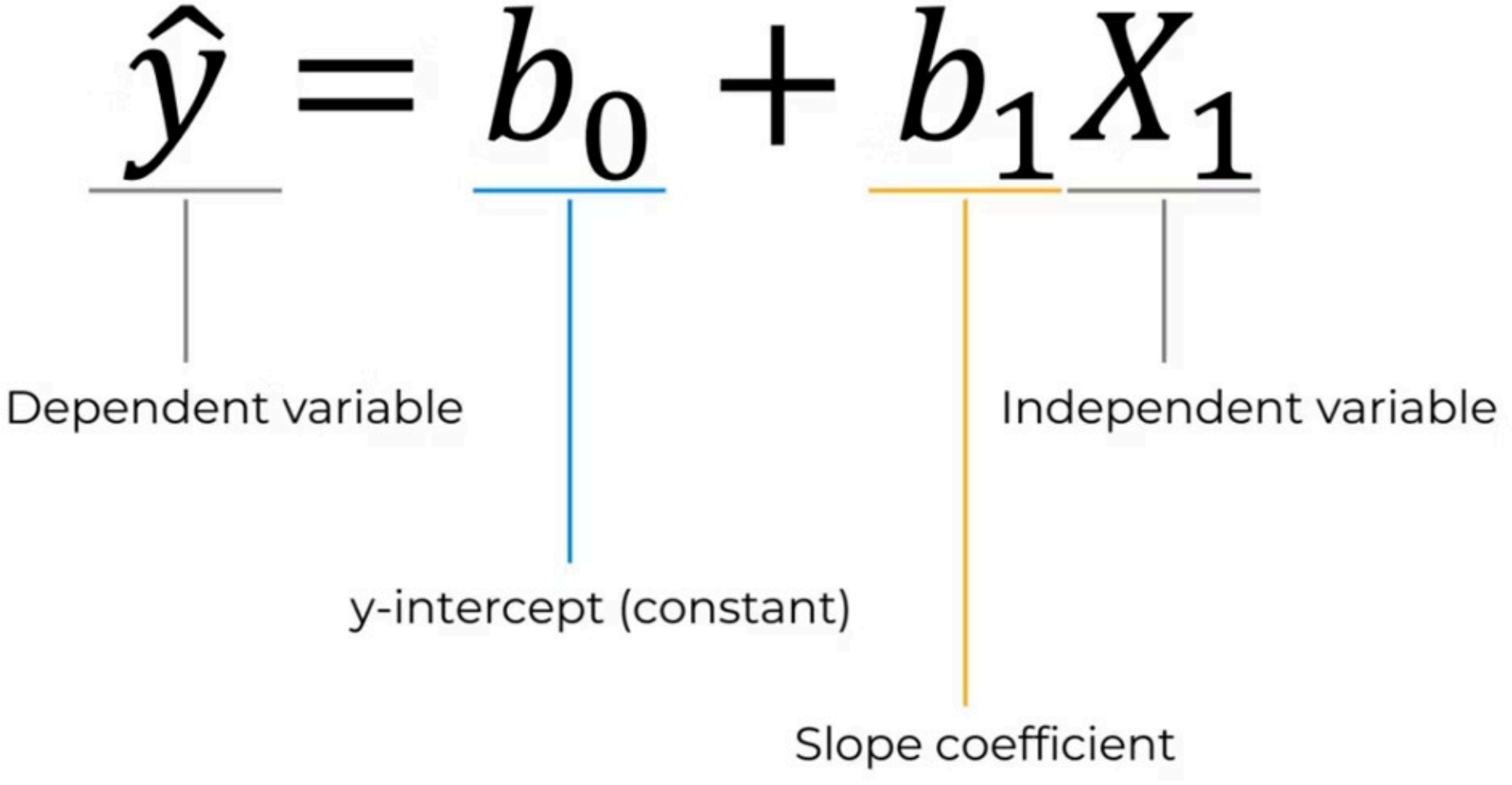
Regression Model - Linear Regression

Linear Regression is a fundamental statistical method used for predictive modeling. Its primary goal is to predict a continuous dependent variable (output) based on one or more independent variables (inputs).

$$y \approx X_1, X_2, ..., X_n$$



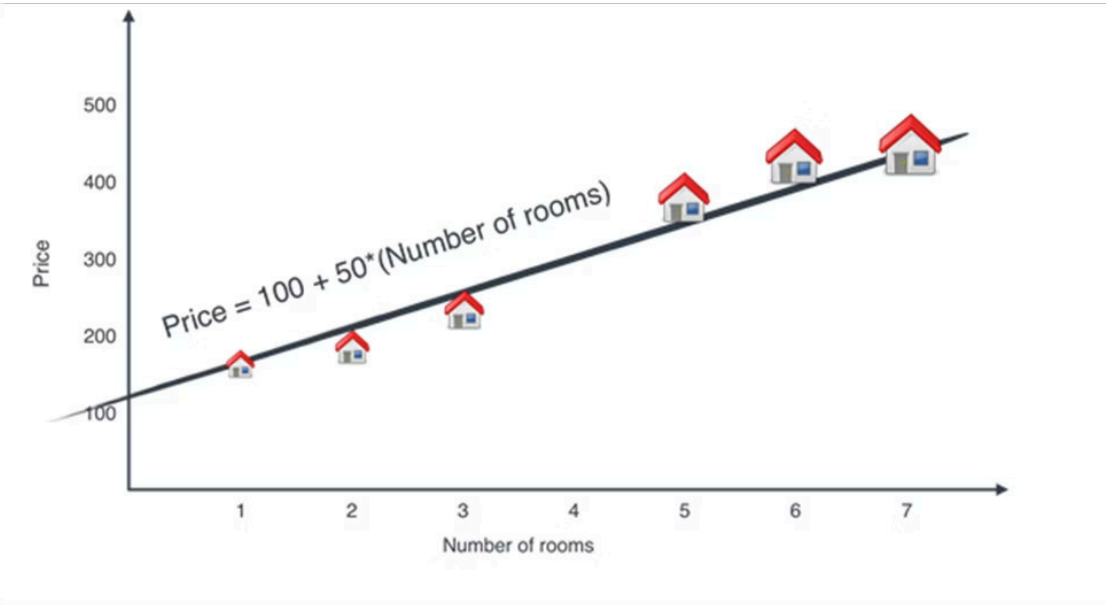
Simple Linear Regression



Model parameters are the numerical values that a model learns from the data, which determine how the model makes predictions.

Simple Linear Regression – EXAMPLE

Number of rooms	Price
1	155
2	197
3	244
4	?
5	356
6	407
7	448



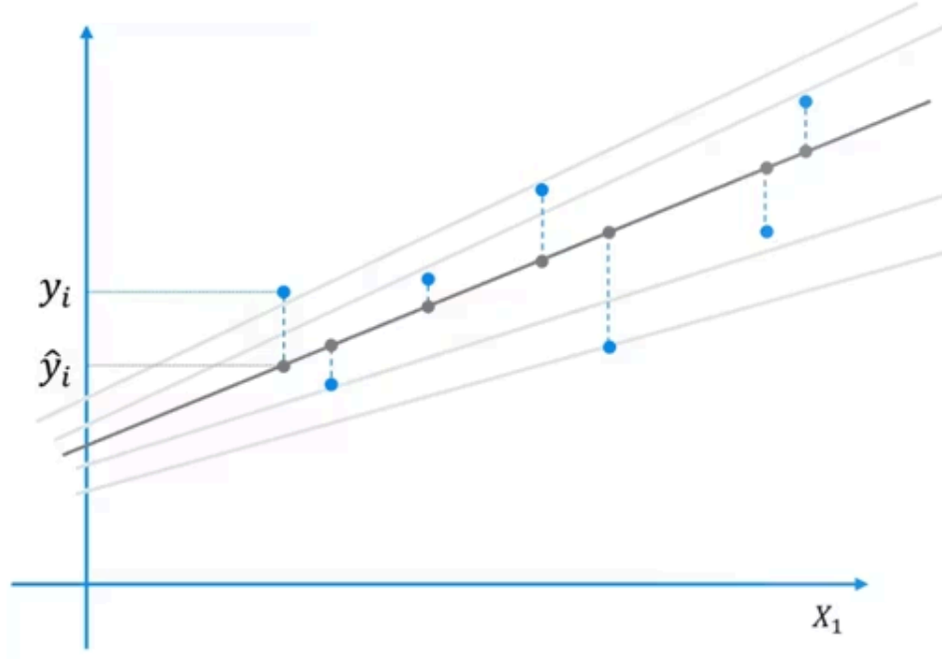
Simple Linear Regression - Residuals

$residual: \quad y_i - \hat{y}_i$

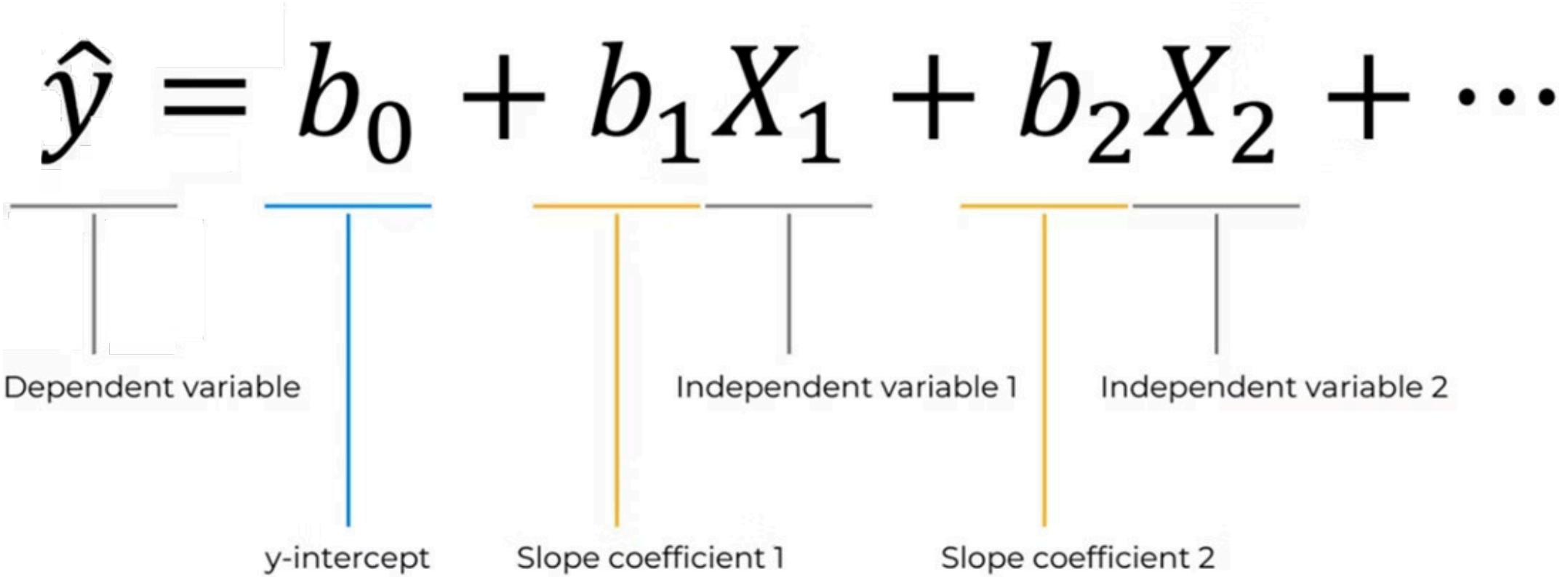
$$\hat{y} = b_0 + b_1 X_1$$

b_0, b_1 such that:

$SUM(y_i - \hat{y}_i)^2$ is minimized

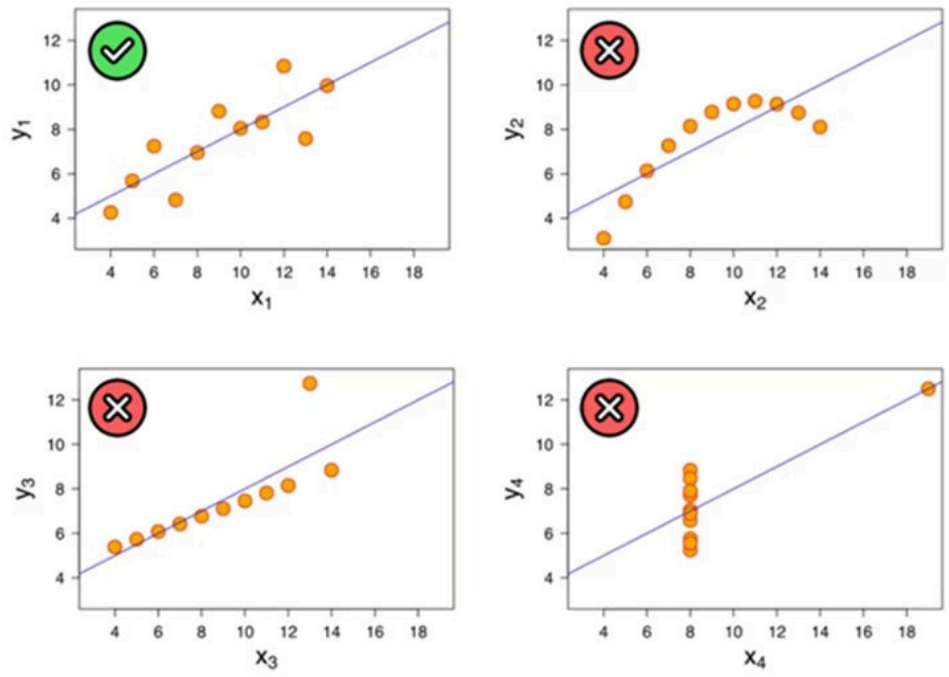


Multiple Linear Regression



Model parameters are the numerical values that a model learns from the data, which determine how the model makes predictions.

Anacombe's quartet (1973)



For all four datasets:

Property	Value	Accuracy
Mean of x	9	exact
Sample variance of x: s_x^2	11	exact
Mean of y	7.50	to 2 decimal places
Sample variance of y: s_y^2	4.125	±0.003
Correlation between x and y	0.816	to 3 decimal places
Linear regression line	$y = 3.00 + 0.500x$	to 2 and 3 decimal places, respectively
Coefficient of determination of the linear regression: R^2	0.67	to 2 decimal places

Create Linear Regression Model

Building a linear regression model follows a structured workflow to understand and forecast relationships between variables.

Data Collection

Gather relevant historical data (features and target) essential for model training.

Data Preprocessing

Clean, handle missing values, scale features, and prepare data for training.

Model Training

Fit the linear regression model to collected data, learning relationships and minimizing errors.

Parameter Estimation

Estimate coefficients to define the linear equation and quantify each feature's impact.

Making Predictions

Utilize the trained model to forecast outcomes for new, unseen data.

These steps ensure an effective linear regression model for insights and predictions.

Data Preprocessing

Step 1: Handling Missing Data

Step 2: Feature Scaling

Step 3: Categorical Data Encoding

Step 4: Outlier Detection & Removal

1.Handle Missing Values

Why?

Linear regression and many other machine learning models cannot directly process datasets containing missing values (NaNs). These gaps can lead to errors, biased models, and inaccurate predictions.

How?

Dropping Rows/Columns

If a row or column has too many missing values, or if the missingness is random and does not hold significant information, you can remove them entirely. This is a simple approach but can lead to loss of valuable data.

Imputation Methods

Replace missing values with calculated estimates. Common methods include filling with the mean, median, or mode of the respective column. More advanced techniques involve using predictive models to estimate missing data points.

1.Handle Missing Values - Code

```python
# Import the SimpleImputer class from sklearn to handle missing data
from sklearn.impute import SimpleImputer

# Create an imputer object that will replace missing values (NaN) with the mean of the column
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')

# Fit the imputer to the columns 1 and 2 of X to compute the mean of these columns
imputer.fit(X[:, 1:3])

# Transform the columns 1 and 2 of X by replacing missing values with the computed mean
X[:, 1:3] = imputer.transform(X[:, 1:3])
```

- `SimpleImputer` : A scikit-learn tool for filling in missing values.
- `missing_values=np.nan` : Specifies that missing values are represented as `NaN` .
- `strategy='mean'` : Missing values will be replaced with the **mean** of the non-missing values in that column.

1.Handle Missing Values - Code

```python
# Import the SimpleImputer class from sklearn to handle missing data
from sklearn.impute import SimpleImputer

# Create an imputer object that will replace missing values (NaN) with the mean of the column
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')

# Fit the imputer to the columns 1 and 2 of X to compute the mean of these columns
imputer.fit(X[:, 1:3])

# Transform the columns 1 and 2 of X by replacing missing values with the computed mean
X[:, 1:3] = imputer.transform(X[:, 1:3])
```

- `X[:, 1:3]` : Selects **columns 1 and 2** (Python indexing is zero-based, so column 0 is excluded).
- `.fit(...)` : Calculates the mean of each selected column (ignoring `NaN` values) and stores it inside the imputer.

1.Handle Missing Values - Code

- `.transform(...)`: Replaces all `NaN` values in these columns with the previously calculated mean.
- The transformed values are assigned **back** to `X[:, 1:3]`.

```
# Import the Si
from sklearn.impute import SimpleImputer

# Create an imputer object that will replace missing values (NaN) with the mean of the column
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')

# Fit the imputer to the columns 1 and 2 of X to compute the mean of these columns
imputer.fit(X[:, 1:3])

# Transform the columns 1 and 2 of X by replacing missing values with the computed mean
X[:, 1:3] = imputer.transform(X[:, 1:3])
```

2. Feature Scaling (Standardization)

Why Feature Scaling is Recommended:

While not strictly required by all models, feature scaling is highly recommended for:

- Ensuring faster convergence for many optimization algorithms (e.g., Gradient Descent).
- Improving coefficient interpretability in linear models, making it easier to understand the impact of each feature.
- Preventing features with larger numerical ranges from dominating the learning process, ensuring all features contribute proportionally.

How to Perform Feature Scaling:

Choose the appropriate scaling method based on your data distribution and model requirements:

- Use **StandardScaler** to transform data to have a mean of zero and a standard deviation of one (zero mean, unit variance). This is generally suitable for algorithms that assume a Gaussian distribution.
- Use **MinMaxScaler** if you need all features to be bounded within a specific range, typically between 0 and 1. This is robust to small standard deviations and preserves the shape of the original distribution.

If your model was trained on **normalized data**, you must also **normalize new data during prediction** using the **same scaler** fitted on the training set.

- **❌ You typically do *not* scale the output variable in linear regression.**

2. Feature Scaling (Standardization)

Normalization

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

[0 ; 1]

Standardization

$$X' = \frac{X - \mu}{\sigma}$$

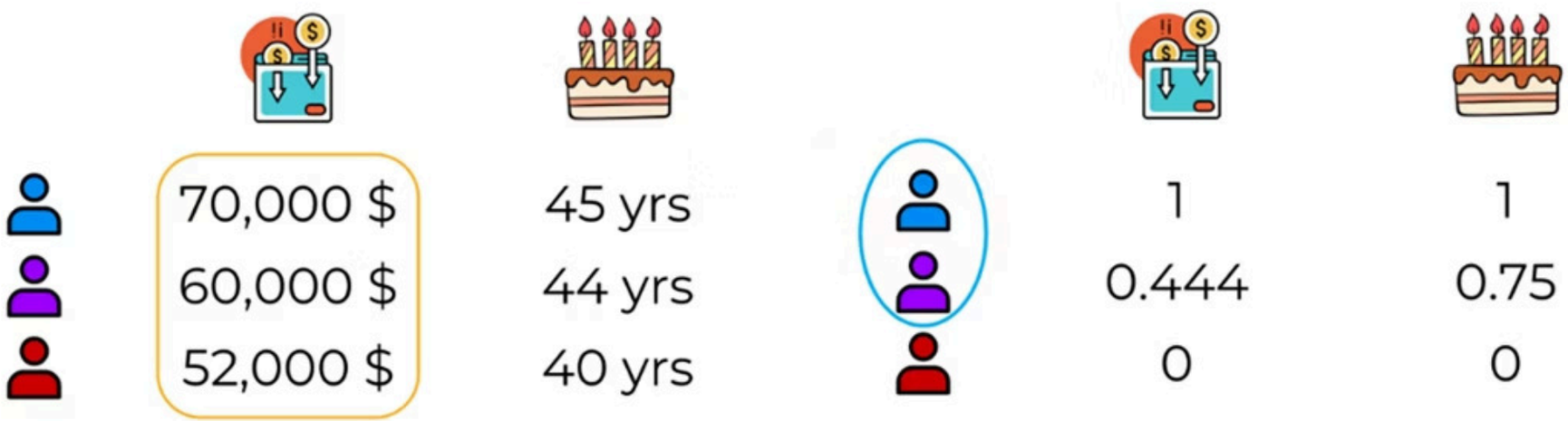
[-3 ; +3]

2. Feature Scaling (Standardization)

Normalization

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

[0 ; 1]



3. Categorical Data Encoding

Categorical data (qualitative information) needs numerical conversion for most machine learning algorithms. Encoding techniques transform this data while preserving its meaning.

- **One-Hot Encoding:** For **nominal features** (no intrinsic order, e.g., "Color"). Creates a new binary column for each category, assigning '1' if present and '0' otherwise. This prevents implying an artificial order.
  Example: "Color: Red" becomes 'Color_Red: 1', 'Color_Blue: 0', 'Color_Green: 0'.
- **Ordinal Encoding:** For **ordinal features** (clear order/rank, e.g., "Size"). Assigns a unique integer to each category based on its position in the sequence.
  Example: "Size: Small, Medium, Large" could be encoded as 0, 1, 2.

3. Categorical Data Encoding

OneHotEncoder

ColumnTransformer

- Allows you to apply different preprocessing steps to different columns of a dataset.

```python
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder

# Create a ColumnTransformer object that applies one-hot encoding to the first column (index 0) of X.
ct = ColumnTransformer(
    transformers=[('encoder', OneHotEncoder(), [0])],
    remainder='passthrough'
)

# Fit the ColumnTransformer to X and transform X, converting the result back into a NumPy array.
X = np.array(ct.fit_transform(X))
```

3. Categorical Data Encoding

OneHotEncoder

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder

# Create a ColumnTransformer object that applies one-hot encoding to the first column (index 0) of X.
ct = ColumnTransformer(
    transformers=[('encoder', OneHotEncoder(), [0])],
    remainder='passthrough'
)

# Fit the ColumnTransfor
X = np.array(ct.fit_tran
```

`transformers=[('encoder', OneHotEncoder(), [0])]`

- `'encoder'` → A **name** for this transformation step (you can choose any descriptive name).
- `OneHotEncoder()` → The transformer to apply; here it converts categorical values into multiple binary columns (one-hot encoding).
- `[0]` → The list of column indices to apply this transformer to — here, **column 0**.

3. Categorical Data Encoding

OneHotEncoder

```python
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder

# Create a ColumnTransformer object that applies one-hot encoding to the first column (index 0) of X.
ct = ColumnTransformer(
    transformers=[('encoder', OneHotEncoder(), [0])],
    remainder='passthrough'
)

# Fit the ColumnTransformer to X and transform X, converting the result back into a NumPy array
X = np.array(ct.fit_trans
```

`remainder='passthrough'`

- Tells the `ColumnTransformer` what to do with the other columns that are **not transformed**.
- `'passthrough'` means "leave them as they are" and include them in the output.

3. Categorical Data Encoding

Ordinal Encoder

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OrdinalEncoder

# Suppose the ordinal column is at index 0
ct = ColumnTransformer(
    transformers=[
        ('ordinal', OrdinalEncoder(categories=[['Low', 'Medium', 'High']]), [0])
    ],
    remainder='passthrough'
)

X_transformed = ct.fit_transform(X)
```

3. Categorical Data Encoding

Dummy Variables

Profit	R&D Spend	Admin	Marketing	State	New York	California
192,261.83	165,349.20	136,897.80	471,784.10	New York	1	0
191,792.06	162,597.70	151,377.59	443,898.53	California	0	1
191,050.39	153,441.51	101,145.55	407,934.54	California	0	1
182,901.99	144,372.41	118,671.85	383,199.62	New York	1	0
166,187.94	142,107.34	91,391.77	366,168.42	California	0	1

$$y = b_0 + b_1*x_1 + b_2*x_2 + b_3*x_3 + b_4*D_1$$

Dummy Variables

Profit	R&D Spend	Admin	Marketing	State	New York	California
192,261.83	165,349.20	136,897.80	471,784.10	New York	1	0
191,792.06	162,597.70			ia	0	1
191,050.39	153,441.51			ia	0	1
182,901.99	144,372.41			rk	1	0
166,187.94	142,107.34			ia	0	1

$D_2 = 1 - D_1$

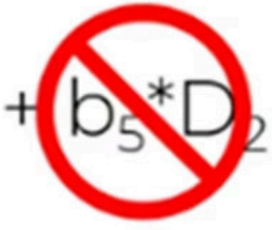
$$y = b_0 + b_1*x_1 + b_2*x_2 + b_3*x_3 + b_4*D_1 + b_5*D_2$$

3. Categorical Data Encoding

Dummy Variables

Profit	R&D Spend	Admin	Marketing	State	New York	California
192,261.83	165,349.20	136,897.80	471,784.10	New York	1	0
191,792.06	162,597.70	151,377.59	443,898.53	California	0	1
191,050.39	153,441.51	101,145.55	407,934.54	California	0	1
182,901.99	144,372.41	118,671.85	383,199.62	New York	1	0
166,187.94	142,107.34	91,391.77	366,168.42	California	0	1

$$y = b_0 + b_1*x_1 + b_2*x_2 + b_3*x_3 + b_4*D_1 + b_5*D_2$$



Always omit one dummy variable

4. Outlier Detection & Removal

Outliers are extreme data points that significantly deviate from other observations. Handling them is critical for robust statistical modeling.

Why Outliers Matter in Linear Regression

Linear regression models are highly sensitive to outliers. A single extreme value can distort the regression line, estimated coefficients, and error terms, leading to inaccurate models and unreliable predictions.

Methods for Detection & Removal

Key methods for outlier detection include:

- **Boxplots:** Visualizes data distribution, highlighting points beyond "whiskers" as potential outliers.
- **Z-Scores:** Measures standard deviations from the mean; Z-scores typically above 2 or 3 often indicate outliers.
- **IQR Method:** Uses the Interquartile Range (IQR). Outliers are values outside (Q1 - 1.5 * IQR) and (Q3 + 1.5 * IQR).

4. Outlier Detection & Removal

```python
import numpy as np

# Calculate the first quartile (25th percentile) of each column in X.
# This identifies the lower boundary for potential outliers.
Q1 = np.percentile(X, 25, axis=0)

# Calculate the third quartile (75th percentile) of each column in X.
# This identifies the upper boundary for potential outliers.
Q3 = np.percentile(X, 75, axis=0)

# Compute the interquartile range (IQR) by subtracting Q1 from Q3.
# The IQR measures statistical dispersion and helps in outlier detection.
IQR = Q3 - Q1

# Determine the lower bound for outlier detection by subtracting 1.5 times the IQR from Q1.
# Values below this threshold are considered outliers.
lower_bound = Q1 - 1.5 * IQR

# Determine the upper bound for outlier detection by adding 1.5 times the IQR to Q3.
# Values above this threshold are considered outliers.
upper_bound = Q3 + 1.5 * IQR

# Create a mask to filter out outliers from X.
# Include only rows where all column values are within the lower and upper bounds.
mask = ((X >= lower_bound) & (X <= upper_bound)).all(axis=1)
X = X[mask]

# Filter y to match the length of the filtered X_train.
# This ensures y_train corresponds to the samples in X_train.
y = y[mask]
```

`.all(axis=1)`

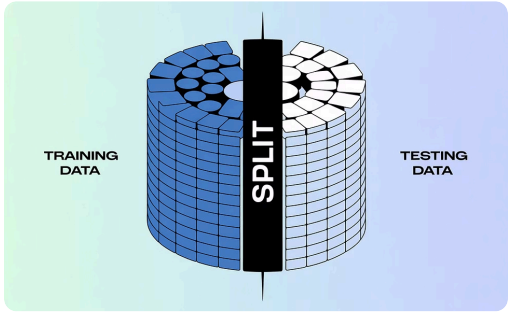
- Checks **across columns** for each row (`axis=1`) whether **all** values in that row meet the condition.
- Returns a boolean Series (mask) indicating which rows to keep.

Linear Regression

01 – data_preprocessing_tools + PreprocessData.csv

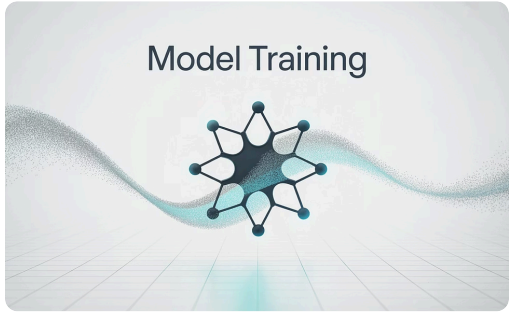
02 – linear_regression
(Part 1: data preprocessing)

Create Linear Regression Model - Steps



1. Creating the Training Set and the Test Set

Divide the dataset into two subsets: one for training the model and one for evaluating its performance.



2. Building and Training the Model

Construct the linear regression model and feed it the training data to learn the relationships between variables.

3. Inference

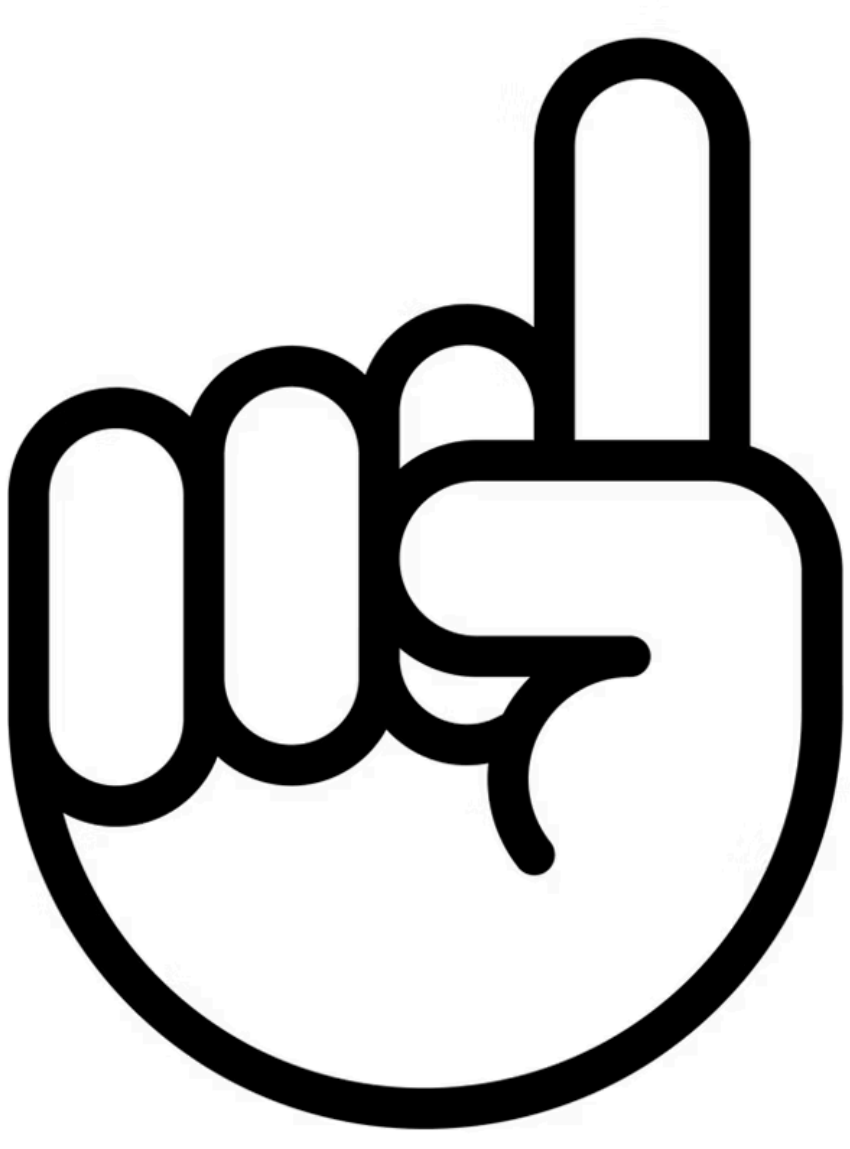
Use the trained model to make predictions on new, unobserved data points.

4. Evaluating the Model

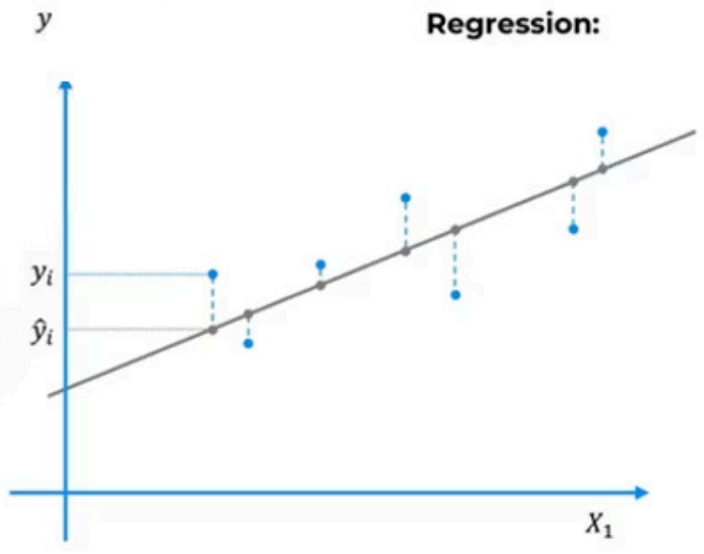
Assess the model's accuracy and effectiveness using various performance metrics on the test set.

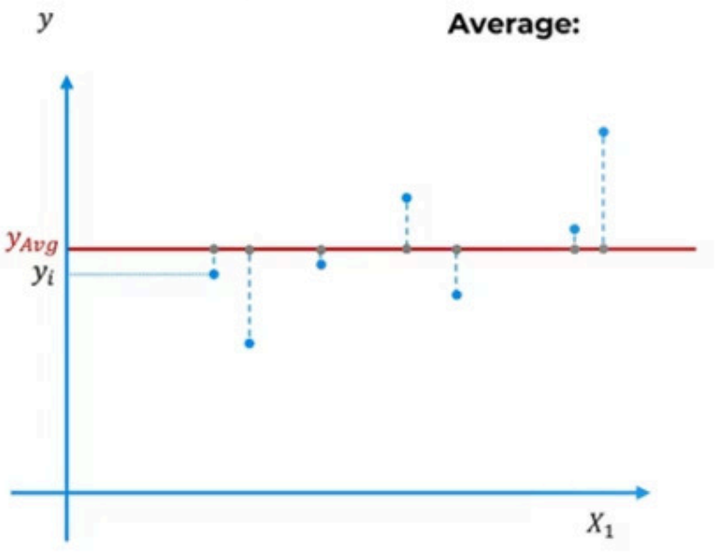
02 – linear_regression

Building and training the model



Model Evaluation - R Square





$$SS_{res} = SUM(y_i - \hat{y}_i)^2$$

$$SS_{tot} = SUM(y_i - y_{avg})^2$$

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

Rule of thumb (for our tutorials)*:
1.0 = Perfect fit (suspicious)
~0.9 = Very good
<0.7 = Not great
<0.4 = Terrible
<0 = Model makes no sense for this data

Model Evaluation - Adjusted R Square

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

R^2 – Goodness of fit
(greater is better)

Problem:

$\hat{y} = b_0 + b_1X_1 + b_2X_2 + b_3X_3$

SS_{tot} doesn't change
SS_{res} will decrease or stay the same

$$SS_{res} = SUM(y_i - \hat{y}_i)^2$$

Solution:

$$Adj\ R^2 = 1 - (1 - R^2) \times \frac{n-1}{n-k-1}$$

k – number of independent variables
n – sample size

$k \uparrow \rightarrow n-k \downarrow \rightarrow (1-R^2) \uparrow$
$\rightarrow$ Adj $R^2 \downarrow$

Model Evaluation - Root Mean Square Error

1 RMSE – Root Mean Squared Error

- Measures the square root of the average of squared errors.
- Penalizes large errors more than small ones.
- Sensitive to outliers.

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

Model Evaluation - Mean Absolute Error

2 MAE – Mean Absolute Error

- Measures the average of absolute differences between predicted and actual values.
- More robust to outliers than RMSE.

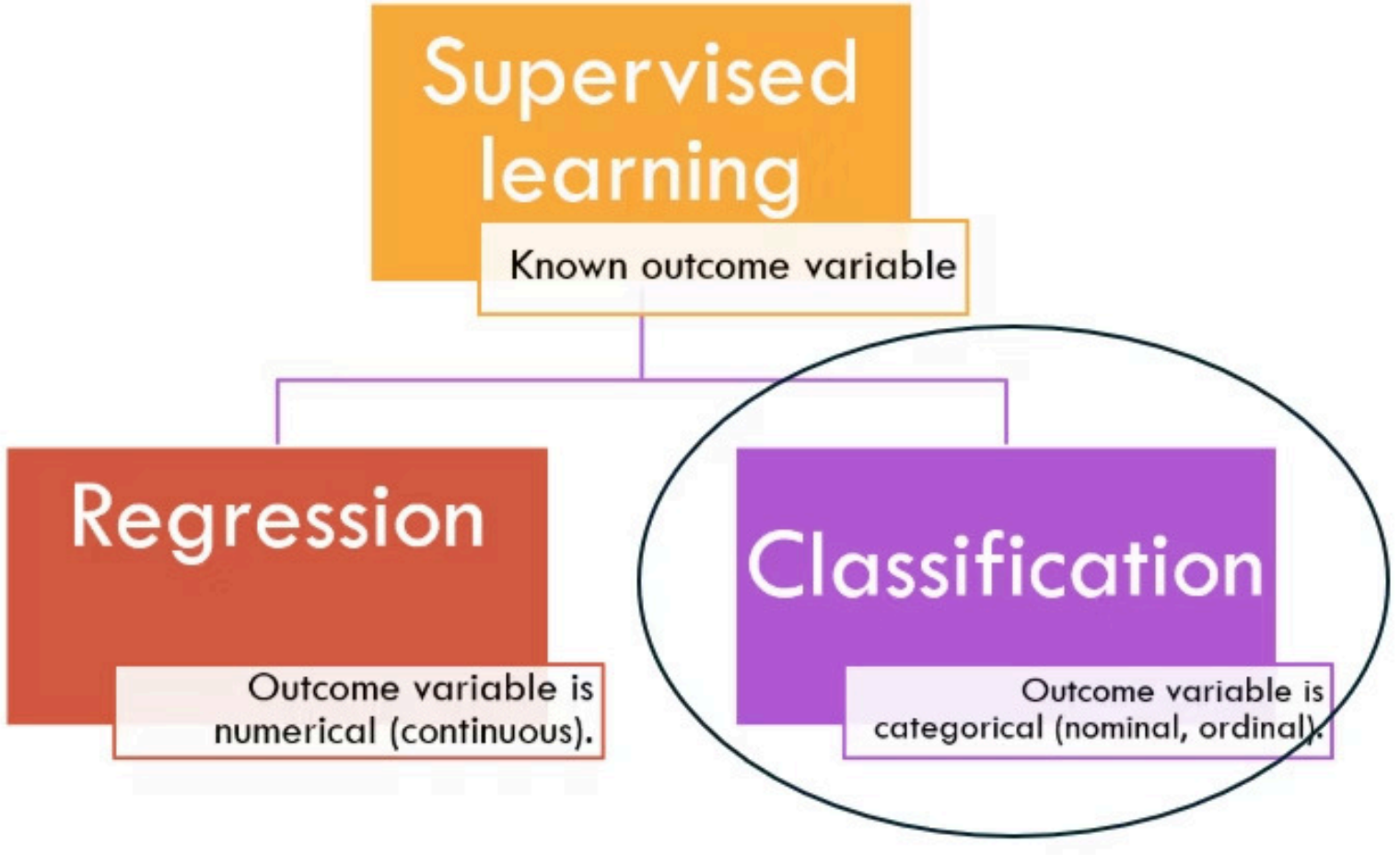
$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Remark

- **RMSE** and **MAE** measure prediction error.
- **R^2** measures how well the model fits the data.

Important: Always compute RMSE, MAE, and R^2 on the test data.

Types of ML Models (Supervised Learning)



Types of ML Models (Supervised Learning)

	Regression Model	Classification Model
Goal	Predict continuous numerical values	Predict categories or labels
Output Type	Real-valued output (e.g., Price = $256.7)	Discrete classes (e.g., Spam / Not Spam)
Examples	House price prediction, Forecasting temperature	Email spam detection, Image recognition (Cat/Dog)
Loss Function	Mean Squared Error (MSE), MAE	Cross-Entropy, Log Loss
Evaluation	RMSE, MAE, R^2 Score	Accuracy, Precision, Recall, F1-Score

Classification Model: Logistic Regression

What is Logistic Regression?

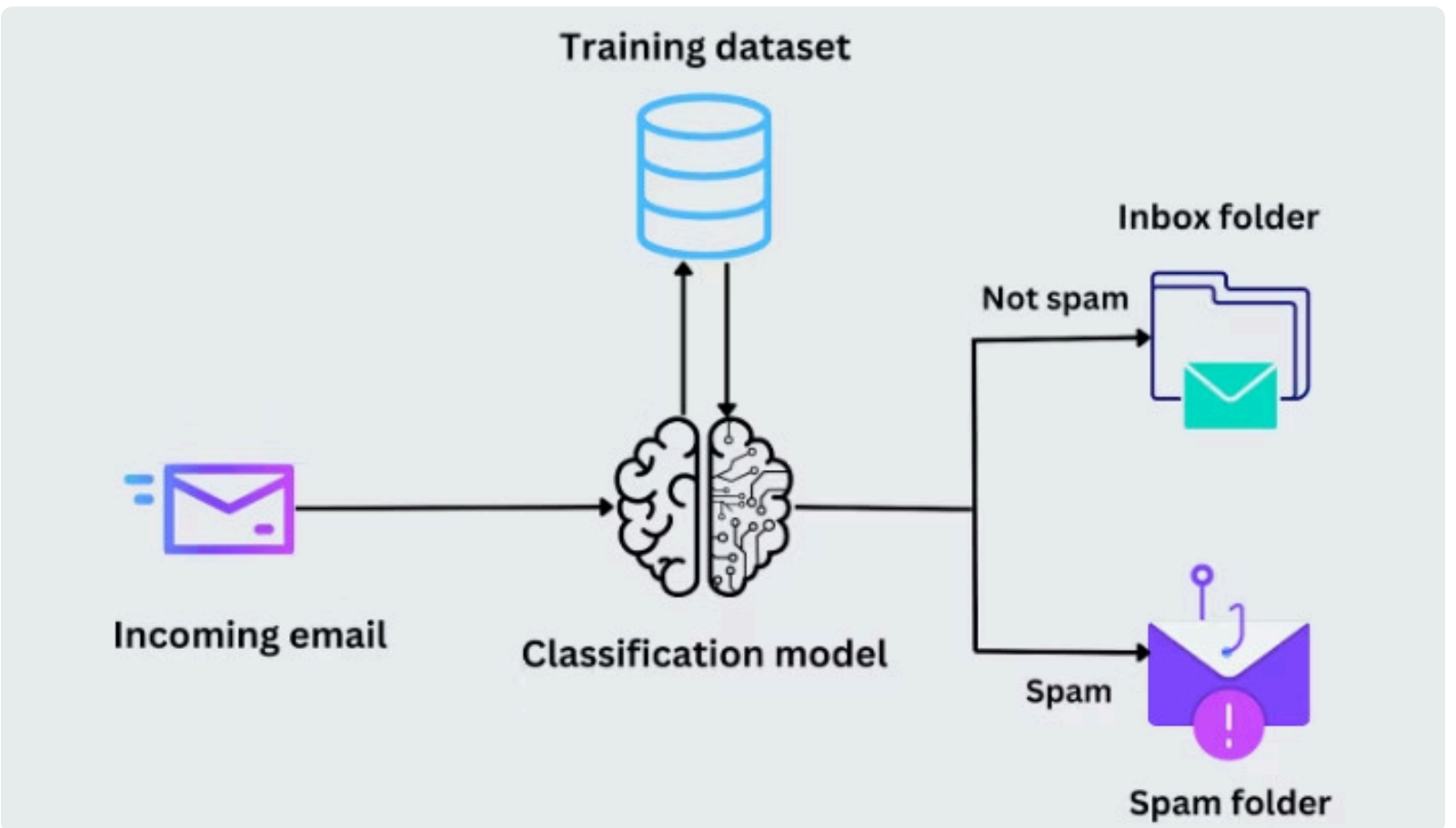
Despite its name, Logistic Regression is a powerful algorithm primarily used for binary classification problems. It models the probability of a certain class or event occurring, such as whether an email is spam or if a customer will churn. It's highly effective for categorical and binary outcome variables (e.g., 0 or 1, Yes or No).

How Logistic Regression Works

Logistic Regression applies a sigmoid function to a linear combination of input features. This function transforms the output into a probability value between 0 and 1, representing the likelihood an observation belongs to a specific class. A predefined threshold (e.g., 0.5) then classifies the observation into one of two classes based on this probability.

Common Applications

- **Medical Diagnosis:** Predicting disease likelihood.
- **Email Spam Detection:** Classifying emails as spam or not.
- **Customer Churn Prediction:** Identifying customers likely to discontinue service.



Odds

The odds of an event are the **ratio** of the probability that the event will occur to the probability that it will **not** occur:

$$\boxed{\text{Odds} = \frac{p}{1-p}}$$

📌 Example:

If the probability of success is $p = 0.75$, then:

$$\text{Odds} = \frac{0.75}{1-0.75} = \frac{0.75}{0.25} = 3$$

Meaning: The chance of success is **3 times greater** than failure.

Logistic Regression

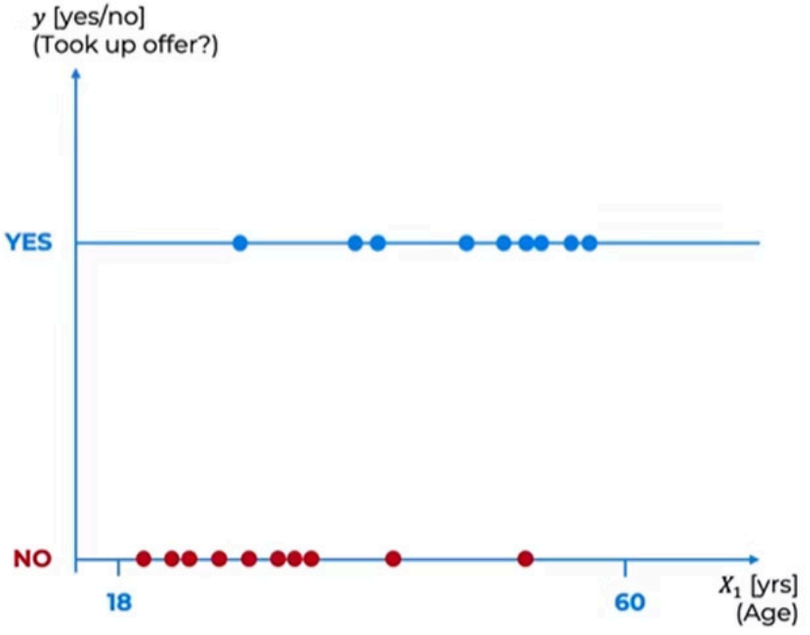
Logistic regression: predict a *categorical* dependent variable from a number of independent variables.

Will purchase health insurance: Yes / No

$\sim$

Age

$$\ln \frac{p}{1-p} = b_0 + b_1 X_1$$



Logistic Regression – Why use ln(p/1-p)?

1 **To make the model linear in parameters:**

- Probabilities p are bounded between 0 and 1.
- The log-odds $\ln\left(\frac{p}{1-p}\right)$ transforms this into a value between $-\infty$ and $+\infty$, so we can model it with a linear equation like $b_0 + b_1 \cdot \text{Age}$.

2 **To interpret coefficients:**

- Each b_i tells you how a **unit increase in** X_i affects the **log-odds**.
- You can exponentiate to get the **odds ratio**:
  e^{b_1} = how much the odds change for a 1-year increase in age, for example.

◆ **Real-world interpretation:**

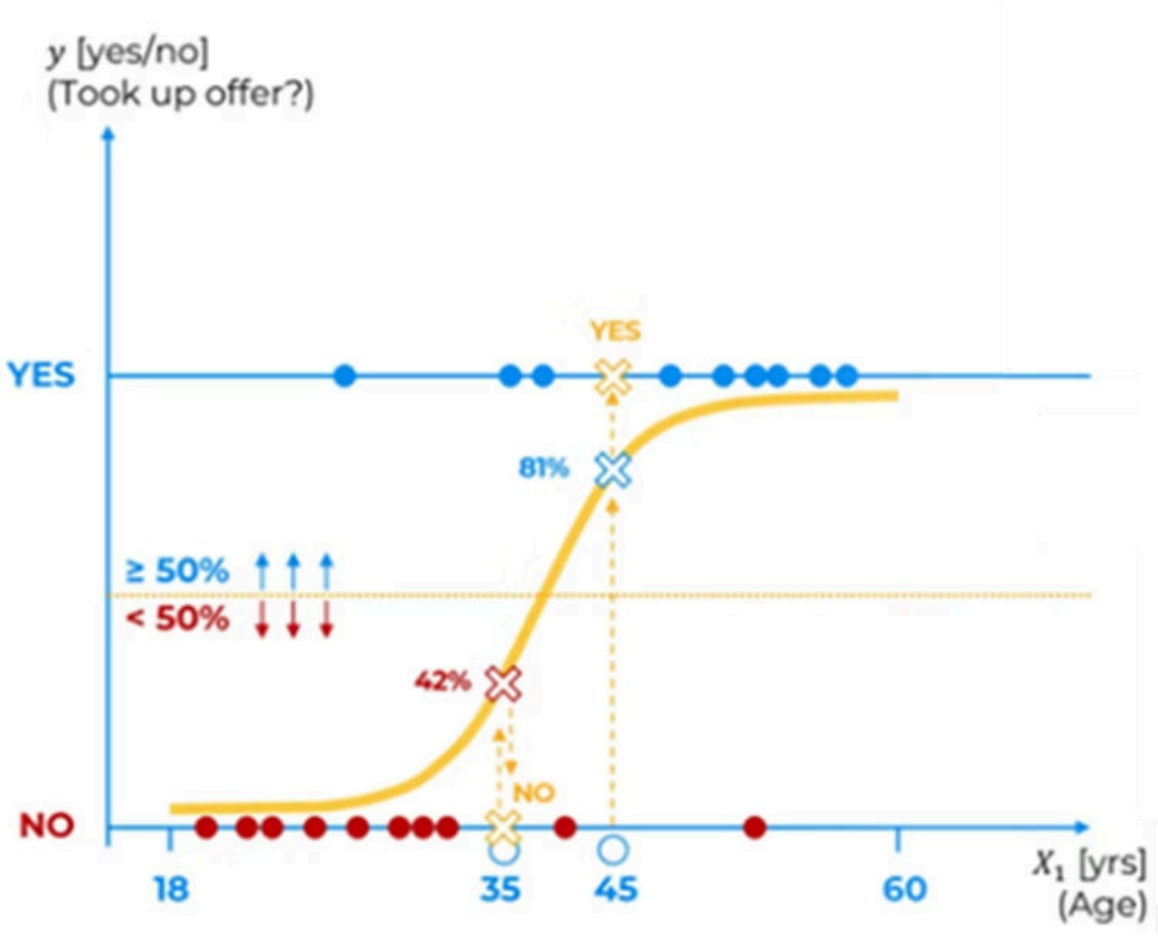
For every additional year of age, the odds of purchasing insurance change by a factor of e^{b_1}.

- If $b_1 = 0.07$, then:

$$e^{0.07} \approx 1.0725$$

⇒ Each year of age increases the **odds** of buying insurance by **about 7.25%**.

Logistic Regression



$$\ln \frac{p}{1-p} = b_0 + b_1 X_1$$

$$p = \frac{1}{1 + e^{-(b_0 + b_1 X_1)}}$$

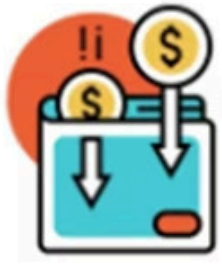
Represents the probability that an observation belongs to **Class 1** in a **logistic regression model**, given its input features.

Logistic Regression

Will purchase health insurance: Yes / No

$\sim$

Age



Income

Level of Education

Family or Single

$$\ln \frac{p}{1-p} = b_0 + b_1X_1 + b_2X_2 + b_3X_3 + b_4X_4$$

$$p = \frac{1}{1 + e^{-(b_0+b_1X_1+b_2X_2+b_3X_3+b_4X_4)}}$$

Maximum Likelihood – How do we do it?

- For each row in the data, the model predicts a probability (e.g., 0.92, 0.03, 0.78...)
- If someone **actually bought**, we want the predicted **probability to be high**
- If someone **did not buy**, we want the predicted **probability to be low**

👉 We then **multiply all those probabilities together**

→ The result is called the **Likelihood**

> The higher it is, the more the model reflects what actually happened in reality.

📌 A Simple Numerical Example

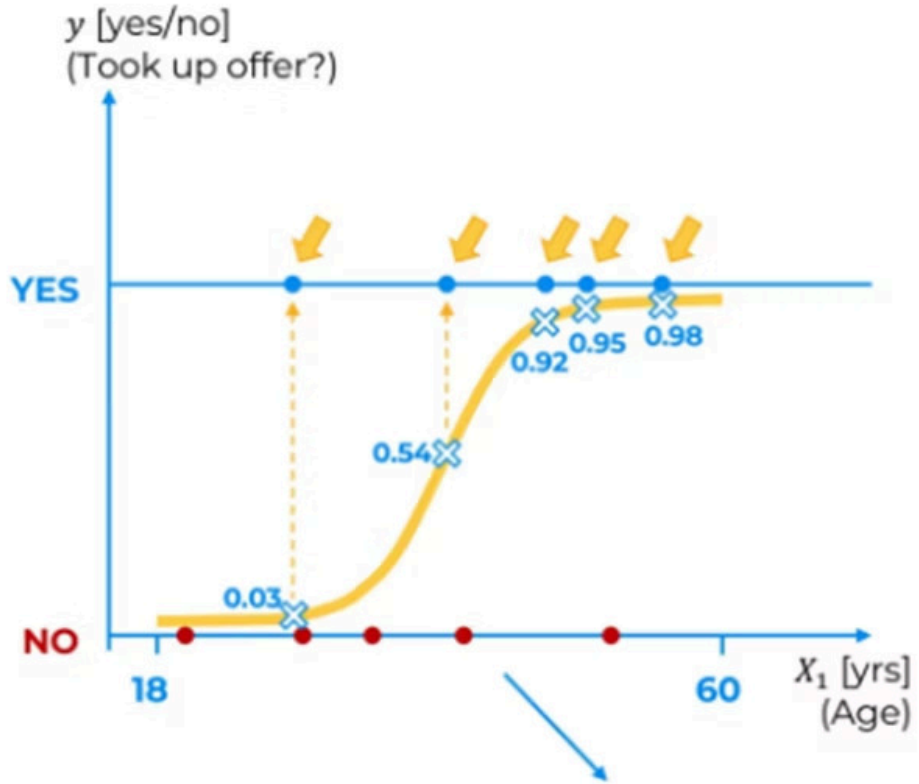
Let's assume you have 3 customers:

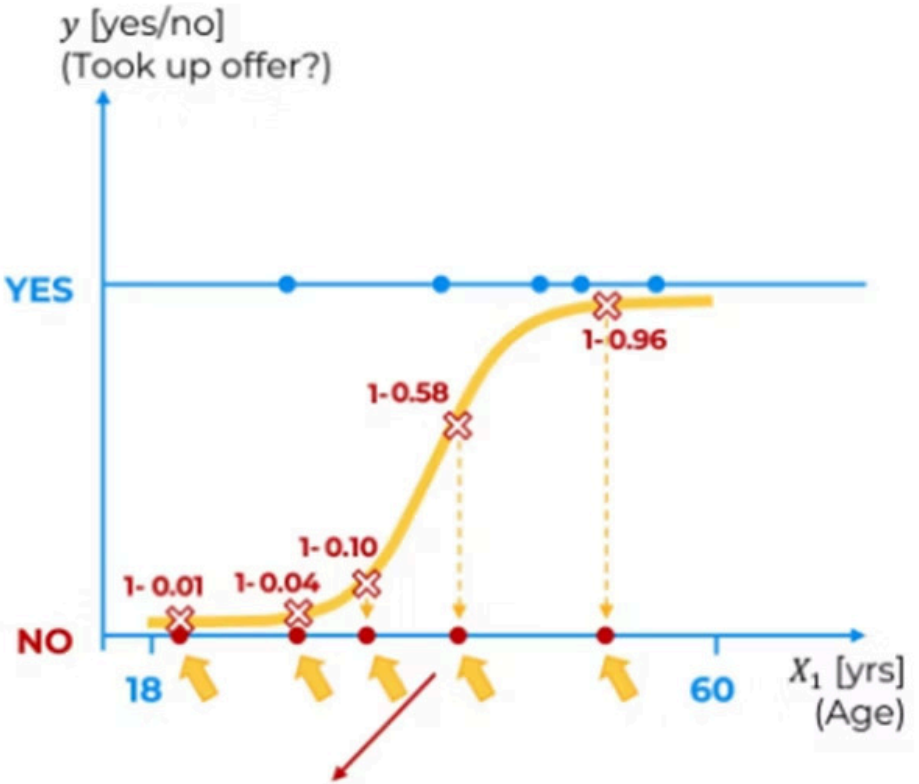
Customer	Bought? (1/0)	Model Prediction
A	1	0.9
B	0	0.1
C	1	0.8

So the total likelihood is:

$$L = 0.9 \cdot (1 - 0.1) \cdot 0.8 = 0.9 \cdot 0.9 \cdot 0.8 = 0.648$$

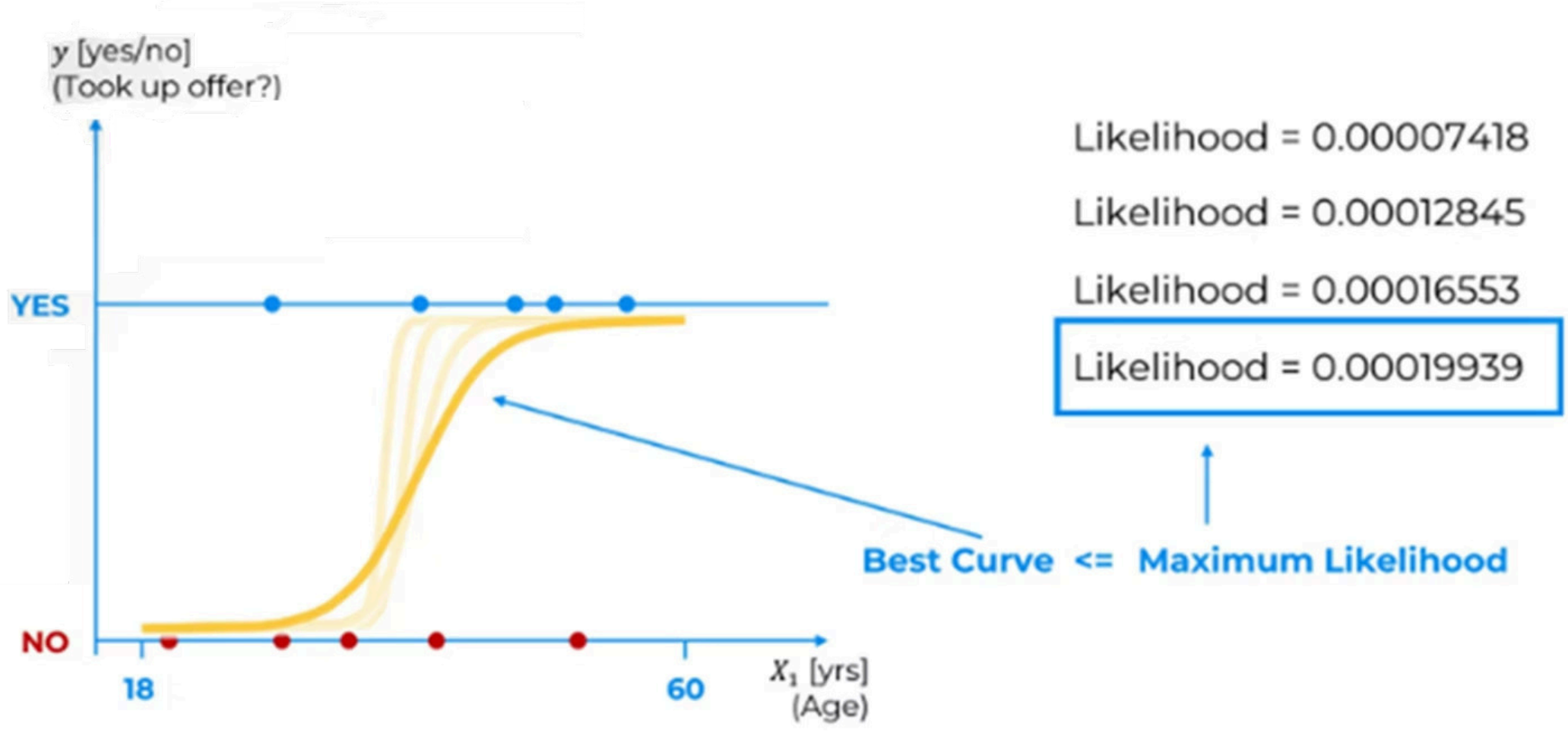
Maximum Likelihood





Likelihood = 0.03 x 0.54 x 0.92 x 0.95 x 0.98 x (1 – 0.01) x (1 – 0.04) x (1 – 0.10) x (1 – 0.58) x (1 – 0.96)

Maximum Likelihood



03 - Logistic Regression