Contents

[MACHINE LEARNING 3](#_Toc215416220)

[TYPES OF MACHINE LEARNING 4](#_Toc215416221)

[SUPERVISED LEARNING 4](#_Toc215416222)

[UNSUPERVISED LEARNING 12](#_Toc215416223)

[TRAINING ML MODELS 13](#_Toc215416224)

[LIFECYCLE 13](#_Toc215416225)

[MACHINE LEARNING PROCESS 14](#_Toc215416226)

[DATA PREPROCESSING WORKFLOW 14](#_Toc215416227)

[DATA CLEANING 15](#_Toc215416228)

[DATA INTERGRATION 15](#_Toc215416229)

[DATA TRANSFORMATION 15](#_Toc215416230)

[DATA SPLITTING 16](#_Toc215416231)

[FEATURE ENGINEERING 16](#_Toc215416232)

[HANDLING IMBALANCED DATA (IF APPLICABLE) 16](#_Toc215416233)

[DATA AUGMENTATION (FOR IMAGES/TEXT) 16](#_Toc215416234)

[TYPES OF DATA IN ML 16](#_Toc215416235)

[FEATURE SCALING 18](#_Toc215416236)

[NORMALIZATION 19](#_Toc215416237)

[STANDARDIZATION 19](#_Toc215416238)

[EXAMPLE – NORMALIZATION 20](#_Toc215416239)

[DATA PRE-PROCESSING USING PYTHON 21](#_Toc215416240)

[DIFFREENT STEPS OF DATA PREPROCESSING 21](#_Toc215416241)

[STEP 1: IMPORTING THE LIBRARIES 21](#_Toc215416242)

[STEP 2: IMPORTING THE DATASET 22](#_Toc215416243)

[STEP 3: MANIPULATING MISSING DATA 22](#_Toc215416244)

[TYPES OF MISSING DATA 23](#_Toc215416245)

[IDENTIFICATON OF MISSING DATA 23](#_Toc215416246)

[STEP 4: ENCODING THE CATEGORIAL DATA 25](#_Toc215416247)

[COMMON ENCODING TECHNIQUES 25](#_Toc215416248)

[ONE HOT ENCODING 25](#_Toc215416249)

[STEP 5: SPLITING TRAINING VERSUS TEST DATA 30](#_Toc215416250)

[STEP 6: FEATURE SCALING 31](#_Toc215416251)

[NORMALIZATION OR STANDARDIZATION 31](#_Toc215416252)

[APPLYING FEATURE SCALING – STANDARDIZATION 33](#_Toc215416253)

[FINAL CODE 34](#_Toc215416254)

[REGRESSION 35](#_Toc215416255)

[SIMPLE LINEAR REGRESSION 35](#_Toc215416256)

[ORDINARY LEAST SQUARES 36](#_Toc215416257)

[EXAMPLE 37](#_Toc215416258)

[SETTING UP LINEAR REGRESSION 37](#_Toc215416259)

[CODE 39](#_Toc215416260)

[MULTIPLE LINEAR REGRESSION 40](#_Toc215416261)

[EXAMPLE 40](#_Toc215416262)

[EXAMPLE OF MUTIPLE LINEAR REGRESSION 40](#_Toc215416263)

[ASSUMPTION OF LINEAR REGRESSION 42](#_Toc215416264)

[HANDLING CATEGORICAL VARIABLES IN LINEAR REGRESSION MODELS 43](#_Toc215416265)

[POLYNOMIAL REGRESSION 44](#_Toc215416266)

[SUPPORT VECTOR REGRESSION 44](#_Toc215416267)

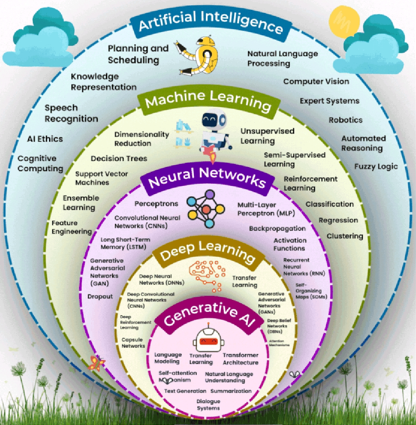
[DECISION TREE REGRESSION 44](#_Toc215416268)

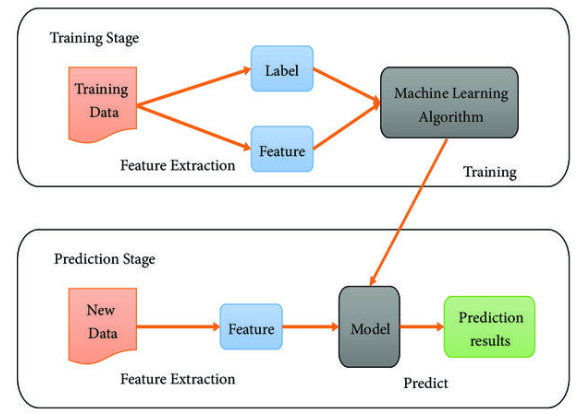
[RANDOM FOREST REGRESSION 44](#_Toc215416269)

[CLASSIFICATION 44](#_Toc215416270)

[LOGISTIC REGRESSION 44](#_Toc215416271)

# MACHINE LEARNING





* Machines learn by observing examples and identifying patterns, like how humans (or babies) learn through experience.
* Machine learning relies heavily on data — both its **quality** and **quantity** — to train models effectively.

Steps

Step 1: The data is used to train algorithms.

Step 2: Algorithms analyze data to find patterns, once it recognize the pattern from the data

Step 3: Then models capable can able to do prediction for the new & unseen data.

|  |
| --- |
| **Algorithms**: These are sets of rules, often based on statistical and mathematical techniques, that guide the learning process. |

## TYPES OF MACHINE LEARNING

A diagram of a machine learning

AI-generated content may be incorrect.

A comparison of words with black text

AI-generated content may be incorrect.

### SUPERVISED LEARNING

* The name supervised learning originates from the idea that training a machine while using this type of approach is like how humans are learning under the supervision of a teacher.

|  |
| --- |
| Analogy   * In regular school class, we have a group of students and a teacher. During a lcture about some specific topic, the teacher will provide several examples while teaching something. The students will use those examples to analyze and memorize them, something that will help them to extract the patterns from those examples. * At a later stage, based on the information provided, the students will be able to solve similar problems. Overall, the teacher decided what kind of examples to present and how many, he or she basically supervised the learning process. |

* **In supervised learning, we train machines by providing them with a set of examples, each provided example is a pair consisting of an input object and the desired output value for that object, This is called labeled data set.**

EXAMPLE: labelled data set

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **fur\_length** | **ear\_shape** | **tail\_length** | **label** | *Based on features like “fur\_length”, “ear\_shape” and “tail\_length” we label the data like”cat” & “dog”* |
| short | pointy | long | cat |
| long | floppy | medium | dog |
| medium | pointy | short | cat |
| long | floppy | long | Dog |

* Now based on this labelled data the model learns by identifying the pattern.
* After learning the pattern, it can be able to do predictions for new data inputs.

#### TRAINING PROCESS

|  |
| --- |
| The model learns a function (y=f(x))(using the training data) that maps the inputs to desired outputs that makes the predictions based on that function |

|  |  |
| --- | --- |
| A diagram of a function  AI-generated content may be incorrect. | * "X" is the input into the machine which can be a group of values called "features". * "Y" is the output of that machine, the target value. * Functions with the input x are basically some mathematical transformation function or mapping function discovered by the algorithm doing the training process |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **RELATION BETWEEN MAPPING FUNCTION AND ML ALGORITHM**  **WHAT IS A MAPPING FUNCTION?**   * Think of a mapping function as a recipe or rule that tells us how to turn input into output. * In machine learning, this function is what the algorithm is trying to learn from data.   **REAL-WORLD EXAMPLE: PREDICTING HOUSE PRICES:** Let’s say we want to predict the price of a house based on its size.   * Input (X): Size of the house (e.g., 1000 sq ft) * Output (Y): Price of the house (e.g., ₹50 lakhs)   The ML algorithm’s job is to learn a function like:  **Price = f(Size)**  This function could be:   * A straight line (if price increases steadily with size) * A curve (if price increases faster for bigger houses)   **HOW ML ALGORITHM USES THE MAPPING FUNCTION?**   1. **Step 1**: We give it data: Sizes and prices of many houses. 2. **Step 2**: It finds a pattern: Learns the best function (mapping) that connects size to price. 3. **Step 3**: We use it to predict: For a new house size, it uses the function to predict the price.   Example in Simple Terms (Training Data)   |  |  | | --- | --- | | Size (sq ft) | Price (₹ lakhs) | | 1000 | 50 | | 1500 | 75 | | 2000 | 100 |  * The ML algorithm might learn: **Price = 0.05 X Size** * So, for a 1200 sq ft house: **Price = 0.05 X 1200 = ₹60** * A mapping function is the rule that connects input to output. * An ML algorithm learns this rule from data. * Once learned, it can make predictions on new data.   **Here's a simple chart that shows how a machine learning algorithm learns a mapping function from house size to price:**  **A graph with a red line and blue dots  AI-generated content may be incorrect.**   * *Blue dots: Real data points (house size vs. price).* * *Red line: The learned function (Price = 0.05 × Size), which the ML algorithm uses to make predictions***.**   **This is a basic example of how ML finds a pattern (a mapping function) in data and uses it to predict outcomes.** |

Two typical tasks performed by Supervised Learning

1. CLASSIFICATION
2. REGRESSION

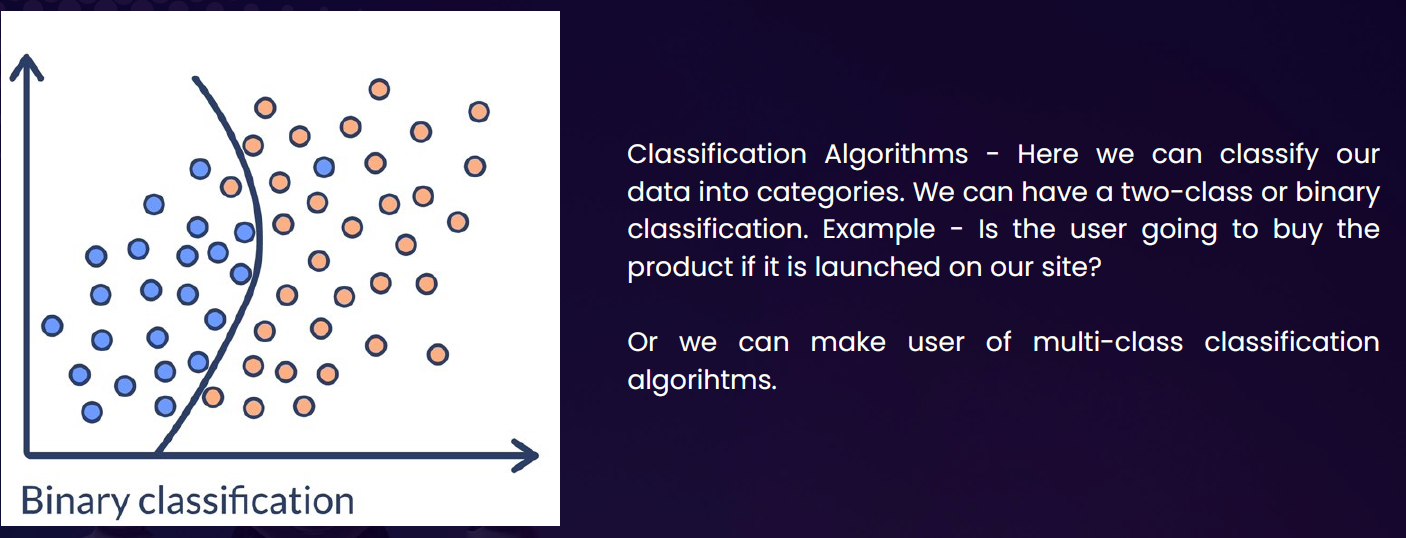
#### CLASSIFICATION (BINARY CLASSIFICATION)

A black and green rectangular sign with white text

AI-generated content may be incorrect.

* Classification involves assigning category labels to new observations based on past observations and their labels (as it a type of supervised learning)
* **Classification** is a type of supervised learning where the goal is to **predict a category or class**. Instead of predicting a number (like in regression), classification predicts a **label**.

##### BINARY CLASSIFICATION



Real-Life Example: Email Spam Detection

Imagine if we have a bunch of emails, and each one is labeled as either:

* **Spam**
* **Not Spam**

We train a machine learning model with this data. The model learns patterns like:

1. Emails with “win money” → likely spam
2. Emails from your contacts → likely not spam

Once trained, the model can classify **new emails** as spam or not spam.

A diagram of a classifier

AI-generated content may be incorrect.

Other Examples of Classification

|  |  |
| --- | --- |
| Problem | Classes (Labels) |
| Disease diagnosis | Sick / Healthy |
| Image recognition | Cat / Dog / Bird |
| Customer feedback sentiment | Positive / Negative / Neutral |
| Loan approval | Approved / Rejected |

##### MULTICLASS CLASSIFICATION



Multiclass classification is when a machine learning model predicts **one label out of three or more possible categories**.

|  |  |
| --- | --- |
| **Problem** | **Classes (Labels)** |
| Handwritten digit recognition | 0 to 9 |
| Animal image classification | Cat / Dog / Bird / Horse |
| Exam grading | A / B / C / D / F |

* SVM(Support Vector Machines) are one such classification ML Model

#### REGRESSION ALGORITHM

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| What is regression (General Concept): Regression helps us find a relationship between input and output and use that to predict future values.  Real-Life Example: Let’s say we want to predict the price of a house based on its size.   |  |  | | --- | --- | | Size (sq ft) | Price (₹ in lakhs) | | 1000 | 50 | | 1500 | 75 | | 2000 | 100 |   Now, if someone asks: "What would be the price of 1800 sq ft house?" Regression helps us predict that — maybe around ₹90 lakhs. | **What the Graph Shows:**   * **Blue dots**: Actual data points (house size vs. price) * **Green line**: The regression line shows the trend the model has learned * **Red dot**: The predicted price for **1800 sq ft** house — around **₹90 lakhs** |

A graph with arrows and dots

AI-generated content may be incorrect.

* **Regression** is a type of **supervised learning** where the goal is to **predict a continuous value** (a number) based on input data.

**Y= aX+b**

**Here Y= Dependent Variable & X= Independent Variable**

* In regression analysis we are continuously trying to find out the value of dependent variables. i.e. the value of y based on one or more predictors(X) which is the independent variable.

Real-World Example: Predicting House Prices

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Imagine a real estate agent and want to **predict the price of a house** based on certain features like:   * Size of the house (in square feet) * Number of bedrooms * Location * Age of the house | You collect data from past house sales. Each row in your dataset looks like this:   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Size (sqft)** | **Bedrooms** | **Age (years)** | **Location Score** | **Price (₹)** | | 1200 | 3 | 5 | 8 | 45,00,000 | | 1500 | 4 | 2 | 9 | 60,00,000 | | 1000 | 2 | 10 | 6 | 35,00,000 | |

* Now, we can train a **regression model** using this data. The model learns the relationship between the features and the price. Later, when we get a new house with known features but unknown price, the model can **predict the price**.

Real-Life Examples

|  |  |  |
| --- | --- | --- |
| **Problem** | **Input Features** | **Output (Continuous Value)** |
| Predict house price | Size, location, age | ₹50,00,000 |
| Predict temperature | Date, time, humidity | 32.5°C |
| Predict height of a child | Age, gender, parents' height | 145.2 cm |
| Predict fuel efficiency | Engine size, weight, speed | 18.7 km/l |

In each case, the **output is a number** that can vary smoothly — not just a fixed set of options.

**GRAPHICAL REPRESENTATION**

|  |  |
| --- | --- |
| A graph of a house size  AI-generated content may be incorrect. | * Each dot represents a house. * The x-axis is the size of the house (in square feet). * The y-axis is the price of the house (in INR). * The pattern shows that as house size increases, the price also tends to increase — this is the kind of relationship a regression model learns. * Red line: The regression line — this is the model's prediction of house price based on size. * The most common form of regression analysis is linear regression |

##### LINEAR REGRESSION

* **Linear regression** is a fundamental concept in statistics and machine learning used to model the relationship between a **dependent variable** (target) and one or more **independent variables** (predictors or features).
* **Key Idea:** Linear regression assumes that there is a **linear relationship** between the input (X) and output (Y), meaning:

|  |  |
| --- | --- |
| **Y = aX + b + error** | * **Y** is the dependent variable (what we want to predict), * **X** is the independent variable (input), * **a** is the slope (how much Y changes with X), * **b** is the intercept (value of Y when X = 0), * **error** is the difference between the predicted and actual value. |

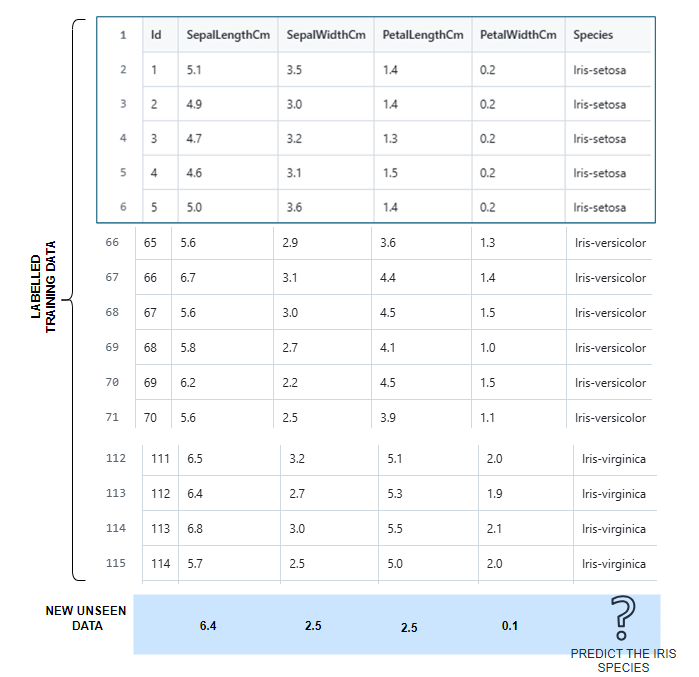
###### TYPES OF LINEAR REGRESSION

1. Simple Linear Regression: 1 independent variable: Y = aX + b
2. Multiple Linear Regression:
   1. more than 1 independent variable Y = a1X1 + a2X2 + ... + anXn + b
   2. Example: Estimating house prices based on size, location, etc

|  |
| --- |
| **Goal: Find the best values of a and b so that the predicted Y values are as close as possible to the actual Y values — usually by minimizing the mean squared error (MSE) between them.** |

###### EXAMPLE

* In this example we will use the data set iris.csv as a training data and then predict the species for new unseen data
* Data Set : [Docs/Machine Learning/Linear-Regression-Example/Iris.csv at master · avishekhsinhaRepo/Docs](https://github.com/avishekhsinhaRepo/Docs/blob/master/Machine%20Learning/Linear-Regression-Example/Iris.csv)



Note : Iris is a flower – based on the pertals and sepals dimension we are going to predict the iris species

|  |  |  |
| --- | --- | --- |
| Some common Pandas operations | | |
| **import pandas as pd**  **from sklearn.linear\_model import LogisticRegression**  **data = pd.read\_csv("./resources/Iris.csv")**  **print(data.head())**  **print(data.shape)** | * The code loads Iris dataset using pandas [read\_csv()](vscode-file://vscode-app/c:/Program%20Files/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) function. * [data.head()](vscode-file://vscode-app/c:/Program%20Files/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) displays the first 5 rows by default, giving us a quick preview of the data structure and values. * The [data.shape](vscode-file://vscode-app/c:/Program%20Files/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) shows the dimensions (rows, columns) of the dataset, which helps understand the dataset size. |
| data.head()    data.shape: **(150,6)** | | |
| **specific\_data= data[["Id","Species"]]**  **print(specific\_data.head())** | This creates a new DataFrame containing only the "Id" and "Species" columns. This is useful for:   * Focusing on specific variables of interest * Creating labels for machine learning (Species would be the target variable) * Reducing memory usage when working with large datasets |

MODEL TRAINING AND PREDICTION

|  |  |
| --- | --- |
| **data = pd.read\_csv("./resources/Iris.csv")**  **X= data.drop(columns=['Id','Species'])**  **Y=data['Species']**  **print (X.head)**  **print(Y.head)**  **ml\_model=LogisticRegression()**  **ml\_model.fit(X.values, Y)**  **ml\_predictions=ml\_model.predict([[6.4,2.5,0.5,0.1]])**  **print(ml\_predictions)**  OUTPUT: Prediction of Iris species | The code creates and trains a logistic regression model   * [X](vscode-file://vscode-app/c:/Program%20Files/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) represents the **features** (input variables) - all columns except 'Id' and 'Species'. * The [drop()](vscode-file://vscode-app/c:/Program%20Files/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) method removes these columns, leaving only the numerical measurements like petal length, sepal width, etc. * [Y](vscode-file://vscode-app/c:/Program%20Files/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) represents the **target variable** (what we want to predict) - the species of each iris flower. * [X.values](vscode-file://vscode-app/c:/Program%20Files/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) converts the pandas DataFrame to a NumPy array, which is often more efficient for scikit-learn algorithms. * The [fit()](vscode-file://vscode-app/c:/Program%20Files/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) method trains the model by learning the relationship between the features and species labels. * Finally, the code makes a prediction on new data: * This predicts the species for a flower with measurements: 6.4, 2.5, 0.5, 0.1. The input must be a 2D array (notice the double brackets [[]]) because scikit-learn expects multiple samples, even when predicting just one. |
| **X.head** | |
| **Y.head** | |

### UNSUPERVISED LEARNING

* **Unsupervised learning** is a type of machine learning where the model learns from data **without any labels**.
* It tries to find **patterns, structures, or groupings** in the data on its own.
* Imagine we have a bunch of photos of animals, but we don’t tell the model which ones are cats, dogs, or birds. The model looks at the photos and **groups similar ones together** — maybe all the cats in one group, dogs in another, and so on.
* **Hence – Unsupervised learning involves analyzing and clustering unlabeled datasets to discover hidden patterns or data groupings without human intervention.**

#### LEARNING TECHNIQUES IN UNSUPERVISED LEARNING

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| What is a Learning Technique in Machine Learning?  A **learning technique** is the **method or approach** used by a machine learning algorithm to **learn patterns** from data.  Think of it like this: : Just like humans use different ways to learn (reading, practicing, watching videos), machines also have different **techniques** to learn from data.  **🔸 In Unsupervised Learning, common learning techniques include:**   |  |  |  | | --- | --- | --- | | Technique | What It Does | Example Use Case | | Clustering | Groups similar data points | Customer segmentation | | Association Rules | Finds relationships between items | Market basket analysis | | Dimensionality Reduction | Simplifies data by reducing features | Visualizing high-dimensional data | | Anomaly Detection | Identifies unusual data points | Fraud detection | |

##### CLUSTERING

Groups similar data points together based on patterns.

* **Goal**: Find natural groupings in data.
* **Examples**:
  + Customer segmentation
  + Grouping news articles
* **Algorithms**:
  + K-Means
  + DBSCAN
  + Hierarchical Clustering

##### ASSOCIATION RULE LEARNING

Finds relationships between variables in large datasets.

* **Goal**: Discover rules like “If A, then B.”
* **Examples**:
  + Market basket analysis (e.g., people who buy bread also buy butter)
* **Algorithms**:
  + Apriori
  + FP-Growth

##### DIMENSIONALITY REDUCTION

Reduces the number of features while keeping important information.

* **Goal**: Simplify data, remove noise, visualize high-dimensional data.
* **Examples**:
  + Visualizing customer behavior
  + Preprocessing for other ML models
* **Algorithms**:
  + PCA (Principal Component Analysis)
  + t-SNE
  + UMAP

##### ANOMALY DETECTION

Identifies data points that don’t fit the pattern.

* **Goal**: Detect outliers or rare events.
* **Examples**:
  + Fraud detection
  + Network intrusion detection
* **Algorithms**:
  + Isolation Forest
  + One-Class SVM
  + Autoencoders

# TRAINING ML MODELS

A diagram of a training process

AI-generated content may be incorrect.

1. As part of the prerequisite of most machine learning projects, we need a training data set. A training data set is a large group of labeled examples. A machine-learning system is going to learn patterns inside the training data set and store that knowledge in something that is called a **model**.
2. This model is supposed to define as close as possible the relationship between features and the target label. In a common type of machine learning method called "supervised learning" the way to create this kind of model is based on analyzing a large group of labeled examples.
3. Once we have trained our model with those labeled examples, we can use that trained model to predict the label on unlabeled examples.

## LIFECYCLE

The lifecycle of a model in a machine learning system, we have two main phases

1. **TRAINING PHASE (LEARNING PHASE):** The main idea is to utilize or use some learning algorithms that will build the model using the training data set.
2. **INFERENCE PHASE (LEARNING PHASE):** In machine learning inference means applying the trained model in an actual machine learning system working in a production environment for making ongoing predictions.

# MACHINE LEARNING PROCESS

A diagram of data processing

AI-generated content may be incorrect.

The data preprocessing includes the following steps

* 1. Handling Missing values:
     1. Due to missing values model will struggle to learn the pattern
  2. Outlier Detection
     1. Outlier can distort the pattern leading to a inaccurate prediction
  3. Feature Scaling and Normalization
  4. Feature Engineering

## DATA PREPROCESSING WORKFLOW

A diagram of a software process

AI-generated content may be incorrect.

### DATA CLEANING

This is the first and most critical step. Raw data often contains errors, inconsistencies, or missing values.

|  |  |
| --- | --- |
|  | Key Tasks:   * **Handling Missing Values**   + Missing data can distort analysis.   + We can either remove rows/columns with missing values or fill them using statistical methods (mean, median, mode) or predictive models. * **Removing Duplicates**: Duplicate entries can bias the model. Identifying and removing them ensures data integrity. * **Correcting Data Types**: Ensuring each column has the correct data type (e.g., integers, floats, dates) is essential for proper analysis. * **Fixing Inconsistencies**: This includes standardizing formats (e.g., date formats), correcting typos, and ensuring consistent labeling (e.g., “Male” vs “male”). |

### DATA INTERGRATION

|  |  |
| --- | --- |
| * The **data integration** step in machine learning involves combining data from multiple sources into a unified dataset that can be used for analysis and modeling. * This is especially important when working with distributed systems, multiple databases, or heterogeneous data formats |  |

### DATA TRANSFORMATION

This step prepares the data for machine learning algorithms, which often require numerical input and consistent scales.

Key Tasks:

* **Normalization**:
  + Rescales features to a fixed range (usually 0 to 1).
  + It is useful when features have different units or scales.
* **Standardization**
  + Centers data around the mean with unit variance.
  + Important for algorithms like SVM or logistic regression.
* **Encoding Categorical Variables**
  + Transforms categorical data (such as gender or country) into numbers.
  + Common methods include one-hot encoding and label encoding.
* **Log Transformation**:
  + Used to reduce skewness in data distributions, especially for features with exponential growth.

### DATA SPLITTING

Before training a model, the dataset is split into subsets to evaluate performance.

Key Tasks:

* **Training Set**: Used to train the model.
* **Validation Set** (optional): Used to tune hyperparameters and prevent overfitting.
* **Test Set**: Used to evaluate the final model’s performance on unseen data.

A typical split might be 70% training, 15% validation, and 15% testing.

### FEATURE ENGINEERING

This involves creating new features or modifying existing ones to improve model performance.

Key Tasks:

* **Creating New Features**: Deriving new variables from existing ones (e.g., age groups from age).
* **Feature Selection**: Choosing the most relevant features to reduce dimensionality and improve accuracy.
* **Dimensionality Reduction**: Techniques like PCA reduce the number of features while retaining most of the information.

### HANDLING IMBALANCED DATA (IF APPLICABLE)

When one class dominates the dataset (e.g., 95% negative, 5% positive), models may become biased.

Key Tasks:

* **Resampling**: Adjusts the class distribution by oversampling the minority class or undersampling the majority.
* **Synthetic Data Generation**: Techniques like SMOTE create synthetic examples of the minority class to balance the dataset.

### DATA AUGMENTATION (FOR IMAGES/TEXT)

Used to artificially expand the training dataset, especially in deep learning.

Key Tasks:

* **Image Augmentation**: Includes rotations, flips, zooms, and brightness adjustments to create varied training samples.
* **Text Augmentation**: Includes synonym replacement, sentence shuffling, or back translation to generate diverse textual data.

## TYPES OF DATA IN ML

Categorical Data

* Data that represents **categories or labels**, not numeric values.
* Examples:

Gender: Male, Female

City: New York, Chicago

Yes/No responses

|  |  |
| --- | --- |
|  | Subtypes   * Nominal   1. No inherent order (e.g., colors: Red, Blue, Green)   2. It cannot be quantified and can be put in any definite hierarchy. * Ordinal:   1. Has an order (e.g., ratings: Poor, Average, Good)   2. It’s a type that consist of categories with a natural rank order, however the difference between the rank may not be equal |

* How to handle categorical data
  1. Encoding using Label Encoding or One-Hot Encoding.

Numerical Data

* Data represented by numbers.
* Examples:

Age: 25, 30

Salary: 50000, 75000

|  |  |
| --- | --- |
|  | Types of Numerical Data  Discrete   * They are countable values which can be represented as a single value (e.g., number of children)   Continuous   * Continuous data refers to **numeric data that can take any value within a range** (including fractions and decimals). It’s measured rather than counted * Example - height, weight, salary |

Types of Continuous Data

|  |
| --- |
| * A meaningful true zero point means that zero represents the complete absence of the quantity being measured, making ratios and comparisons valid. * Examples   1. **True zero (Ratio data):**      1. Weight: 0 kg means no weight at all.      2. Height: 0 cm means no height.      3. Income: 0 means no income. → You can say 80 kg is twice as heavy as 40 kg.   2. **Not a true zero (Interval data):**      1. Temperature in Celsius: 0°C does not mean “no temperature”; it’s just a point on the scale. → You cannot say 20°C is twice as hot as 10°C. |

1. **Interval Data**
   * Ordered value with equal interval between values, but no true zero point
   * **Examples**:
     + Temperature in Celsius or Fahrenheit (0°C doesn’t mean “no temperature”).
     + Dates (difference between years is meaningful).
   * **Key point**: You can add/subtract values, but ratios don’t make sense (e.g., 20°C is not “twice as hot” as 10°C).
2. **Ratio Data**
   * **Definition**: Numeric data with a **true zero point**, so both differences and ratios are meaningful.
   * **Examples**:
     + Height, Weight, Age, Salary.
     + Distance traveled.
   * **Key point**: You can say “twice as much” because zero means absence of the quantity.

|  |  |  |
| --- | --- | --- |
| Type | True Zero? | Example |
| Interval | No | Temperature (°C), Dates |
| Ratio | Yes | Height, Weight, Income |

* How to handle:
  1. Scaling or Normalization (e.g., StandardScaler, MinMaxScaler).

Datetime

* Data representing dates and times.
* Examples:
  1. 2025-11-29
  2. 2025-11-29 13:54:32
* Why important:
  1. Used for time-series analysis, trend detection, seasonality.
* How to handle:
  1. Extract features like year, month, day, hour, or convert to timestamp.
  2. Use specialized libraries (e.g., pandas dt accessor).

## FEATURE SCALING

* Feature scaling **transforms the values of features to be on a similar scale**, typically to improve model performance and training stability.

Note: Feature scaling is always applied at column level

A table with numbers and lines

AI-generated content may be incorrect.

There are 2 main techniques of feature scaling

1. **NORMALIZATION**
2. **STANDARDIZATION**

### NORMALIZATION

|  |  |
| --- | --- |
| A math equation with black text  AI-generated content may be incorrect. | * The normalization value lies between the closed interval of [0;1] |

|  |  |  |
| --- | --- | --- |
| **X1 (Price)** | **X-XMIN** | **Normalized Value(X1)** |
| $179.43 | $0.00 | 0.00 |
| $641.87 | $179.43 | 0.39 |
| $556.30 | $376.87 | 0.814959779 |
| $578.47 | $116.03 | 0.250908226 |
| $591.12 | $411.69 | 0.890256033 |
|  |  |  |
| X1-MAX |  | $641.87 |
| X1-MIN |  | $179.43 |
| X1MAX - X1MIN |  | **$462.44** |

### STANDARDIZATION

|  |  |  |
| --- | --- | --- |
| A math equation with numbers and symbols  AI-generated content may be incorrect. | µ | Average |
|  | Standard Deviation |
| * The value lies in closes interval of [-3,3] * If data has some outliers – it will exist outside this range | |
| **Standard Deviation** calculation:  **Sample Data:** [10, 12, 23, 23, 16, 23, 21, 16]  Step-by-Step Calculation:   1. **Mean (Average)** Add all values and divide by the number of values:      1. **Variance** Find the squared difference from the mean for each value, then average those:      1. **Standard Deviation** Take the square root of the variance     **Final Result:**   * **Mean**: 18.0 * **Variance**: 24.0 * **Standard Deviation**: **≈ 4.90**   **Why It Matters**  Standard deviation tells you **how spread out the values are**:   * A **low** standard deviation means values are close to the mean. * A **high** standard deviation means values are more spread out. | | |

### EXAMPLE – NORMALIZATION

A screenshot of a computer

AI-generated content may be incorrect.

1. Let's imagine we have a data set where we have two columns, annual income of a person and their age of

a blue, purple and red person.

1. **We must identify whether the purple person is like a “red” person or “blue” person . This is the task of clustering data. For that we need to do normalization of data as the units of the data is not uniform**

A close-up of a number

AI-generated content may be incorrect.

1. After normalizing, our values will look like above. Hence with the normalized data – From salary column perspective. The purple person is almost right in the middle between the red and the blue people(0.44), whereas in the age column, the purple person is closest to the blue person.

|  |
| --- |
| Scikit-learn (also written as scikit-learn or sklearn) is a powerful and widely used open-source machine learning library for the Python. It provides simple and efficient tools for:  🔍 Key Features   * Classification: Identifying which category an object belongs to (e.g., spam detection). * Regression: Predicting a continuous-valued attribute (e.g., house prices). * Clustering: Grouping similar data points (e.g., customer segmentation). * Dimensionality Reduction: Reducing the number of features (e.g., PCA). * Model Selection: Comparing, validating, and choosing parameters and models. * Preprocessing: Feature extraction, normalization, and transformation.   🧰 Built On  Scikit-learn is built on top of:   * NumPy: For numerical operations. * SciPy: For scientific computing. * Matplotlib: For plotting (indirectly used). * joblib: For model persistence and parallel processing. |

# DATA PRE-PROCESSING USING PYTHON

|  |  |
| --- | --- |
|  | In this example we will perform data preprocessing step on this following data   * It’s a user profile data – of an ecommerce website with a flag which says whether user has made purchase or not! * As a data processing step – we create two entities –  1. The first is **the matrix of features**, which contains separately these three columns (country, age, salary.) 2. And second is the dependent variable vector, which is last column(“Purchased”), because that's the column we want to predict.   *Note : This is exactly what we must do in this first data pre-processing phase.* |

## DIFFREENT STEPS OF DATA PREPROCESSING

1. Importing libraries
2. Importing Dataset
3. Taking Care Of Missing Data
4. Encoding Categorical Data
   1. Encoding The Independent Variables
   2. Encoding The Dependent Variables
5. Splitting The Dataset Into Test & Training Dataset
6. Feature Scaling

## STEP 1: IMPORTING THE LIBRARIES

|  |  |  |
| --- | --- | --- |
| Step 1: Importing the libraries | import numpy as np import matplotlib.pyplot as plt import pandas as pd | **pip install numpy pandas matplotlib** |

## STEP 2: IMPORTING THE DATASET

|  |  |  |
| --- | --- | --- |
| Step 2: Importing the dataset | dataset = pd.read\_csv('Data.csv')   * The dataset variable is pandas dataframe | |
| **PANDAS iloc FUNCTION**   * The iloc function in **pandas** is used for **integer-location based indexing**. * It allows us to select rows and columns from a DataFrame using their **position (index numbers)** .   **Syntax :df.iloc[row\_index, column\_index]**   * row\_index: Integer or slice for row positions. * column\_index: Integer or slice for column positions.   Examples | | |
| Assume we have a DataFrame df:  **import pandas as pd**  **data = {**  **'Country': ['France', 'Spain', 'Germany'],**  **'Age': [44, 27, 30],**  **'Salary': [72000, 48000, 54000],**  **'Purchased': ['Yes', 'No', 'Yes']**  **}**  **df = pd.DataFrame(data)** | | **1. Select first row :** df.iloc[0]  **2. Select first column :** df.iloc[:, 0]  **3. Select rows 0 to 2 and columns 0 to 2 (features matrix)**  **X = df.iloc[:, :-1] # All rows, all columns except last**  **4. Select last column (dependent variable vector)**  **y = df.iloc[:, -1] # All rows, last column**  **Note: We u**se iloc when we want to select data by **position**, and use loc when you want to select by **label**. |
| **X = dataset.iloc[:, :-1].values y = dataset.iloc[:, -1].values**  OUTPUT  [['France' 44.0 72000.0]  ['Spain' 27.0 48000.0]  ['Germany' 30.0 54000.0]  ['Spain' 38.0 61000.0]  ['Germany' 40.0 nan]  ['France' 35.0 58000.0]  ['Spain' nan 52000.0]  ['France' 48.0 79000.0]  ['Germany' 50.0 83000.0]  ['France' 37.0 67000.0]]  ['No' 'Yes' 'No' 'No' 'Yes' 'Yes' 'No' 'Yes' 'No' 'Yes'] | After importing the dataset, the **first key step** is to **separate the data into two parts**:   1. **Part 1: Matrix of Features (X)**:    * Also called **independent variables**.    * These are the input columns used to make predictions.    * Typically include all columns **except the last one**.    * Example: Country, Age, Salary. 2. **Part 2: Dependent Variable Vector (y)**:    * Also called the **target** or **label**.    * This is the **output** you want to predict.    * Usually the **last column** in the dataset.    * Example: Purchased (Yes/No).   Key Principle   * In most ML datasets, the **features** are in the **first few columns**, and the **dependent variable** is in the **last column**. * This structure is common across many datasets and is essential for training supervised machine learning models. | |

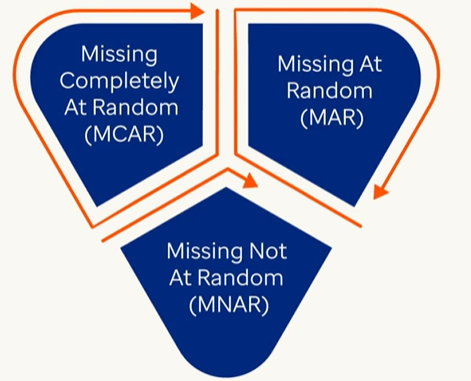
## STEP 3: MANIPULATING MISSING AND DUPLICATE DATA

### IDENTIFICATON OF DUPLICATE DATA

|  |
| --- |
| How To Identify Duplicate Values In Dataset   * Duplicate records are rows in a dataset that have identical values across all the columns. * They often occur due to data entry errors, merging datasets, or system issues, leading to redundancy. * **Their presence in the dataset may skew the performance of a model, and hence we need to remove such duplicate records as a first step in data preprocessing**.   **STEP 1: READ THE DATASET**  **import pandas as pd**  **data = pd.read\_csv("Hospital\_Provider\_Cost.csv")**  **print(data.shape) ## Return tuple of rows and column (37013, 15)**  **STEP 2: CHECK FOR DUPLICATE RECORDS**  **# check whether any duplicate records are in the dataset**  **duplicate\_rows = data.duplicated().sum()**  **print(f'There are {duplicate\_rows} duplicate rows in the data')**  **STEP 3: EXTRACT THE DUPLICATE RECORDS**  **# extract and display the duplicate records**  **duplicate\_rows = data[data.duplicated()]**  **STEP 4: REMOVE THE DUPLICATE RECORDS**  **# remove the duplicate records**  **data\_without\_duplicates = data.drop\_duplicates() ## There are 10 duplicate record**  **STEP 5: BEFORE AND AFTER RECORD LENGTH**  **# Printing the shape of the dataset before and after remove duplicate records**  **print(f'No. of rows before removing duplicates: {data.shape[0]}')** 🡪 **(37013, 15)**  **print(f'No. of rows after removing duplicates: {data\_without\_duplicates.shape[0]}') 🡪 (37003, 15)** |

### MISSING DATA

#### TYPES OF MISSING DATA



##### MCAR – Missing Completely at Random

* In **MCAR** data is missing for no reason related to the data itself. The missing data is
  1. Completely Unpredictable
  2. May be completely unrelated
  3. Without any pattern in missingness
* **Examples**:
  1. For instance, in a survey some participants skipped a question about their favorite food. Here the missing response does not depend on participant age, gender and eating habits
  2. A hospital survey form got damaged in the mail, so some patient ages are missing. It has nothing to do with the patient’s age or health.
* **Key point**: Missingness is purely random → easiest to handle.

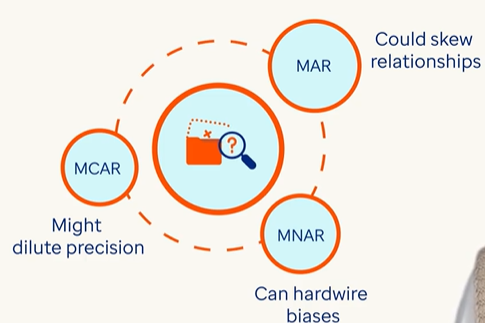
##### MAR – Missing at Random

* Missingness id related to known factor.
* **Example**:
  1. Income is missing more often for younger patients in the Hospital survey, if younger level the salary data blank in the survey intentionally.
  2. Because of the data, AI models might overlook the critical patterns. For example, if income impacts mental health outcome – the model will underestimate the financial stressor young individual faces
* You can use other columns to fill in the gaps/missing data using imputation techniques like grouped averages.

##### MNAR – Missing Not at Random

* Missingness of the data is tied to a reason.
* **Example**:
  1. Like heavy cigarette smoker deliberately not reporting the “number of cigarettes” they smoke in a day
  2. Patients with very high incomes don’t report their income. The reason it’s missing is because of the income itself.
* **Key point**: Hardest to fix because we can’t guess the missing value without bias.

IMPACT OF MISSING VALUES



##### Quick analogy

1. **MCAR**: Lost data by accident (like spilling coffee on a form).
2. **MAR**: Missing because of something else you know (young people skip income question).
3. **MNAR**: Missing because of the value itself (rich people hide income).

#### IDENTIFYING MISSING DATA

* Missing values are the values that are absent in one or more features.
* Missing values can be random or there may be some pattern in the absence of values.
* **.isna().sum()** computes the number of missing values in each column of the dataset.

|  |  |
| --- | --- |
| **import pandas as pd**  **data = pd.read\_csv("Hospital\_Provider\_Cost.csv")**  **data\_without\_duplicates = data.drop\_duplicates()** | |
| **# display the number of missing values**  **missing\_data\_count = data\_without\_duplicates.isna().sum()**  **print(missing\_data\_count)**  **# alternatively, one can use `isnull()` method to identify missing values**  **data\_without\_duplicates.isnull().sum()**  **Inference**   * The features Provider Type, CCN Facility Type, Type of Control and Inpatient Total Charges do not have any missing values. * However, all other features have some missing values in them. | **Number of Beds 473**  **Employees 699**  **Provider Type 0**  **CCN Facility Type 0**  **Rural Versus Urban 436**  **Type of Control 0**  **Total Salaries 444**  **Depreciation Cost 807**  **Overhead Non-Salary Costs 436**  **Total Fixed Assets 209**  **Total Discharges 543**  **Inventory Cost 545**  **Total Patient Bed Days 512**  **Prepaid Expenses 672**  **Inpatient Total Charges 0**  **dtype: int64** |
| **Percentage of missing values in each column**  **pct\_missing\_value = missing\_data\_count/ len(data\_without\_duplicates)\*100**  **print(pct\_missing\_value)** | **Number of Beds 1.278275**  **Employees 1.889036**  **Provider Type 0.000000**  **CCN Facility Type 0.000000**  **Rural Versus Urban 1.178283**  **Type of Control 0.000000**  **Total Salaries 1.199903**  **Depreciation Cost 2.180904**  **Overhead Non-Salary Costs 1.178283**  **Total Fixed Assets 0.564819**  **Total Discharges 1.467449**  **Inventory Cost 1.472854**  **Total Patient Bed Days 1.383672**  **Prepaid Expenses 1.816069**  **Inpatient Total Charges 0.000000**  **dtype: float64** |

#### HANDLING MISSING DATA

Techniques to Handle Missing Data

|  |  |  |
| --- | --- | --- |
| Drop Missing Data | **Use when:**   * Missing % is **very small (<5%)** * Dropping rows **won’t create bias** | **df.dropna()** |
| Impute with Simple Values | Best for numeric columns.  **Mean / Median / Mode**   * **Mean**: Good if distribution is normal * **Median**: Good if data has outliers * **Mode**: For categorical values | **from sklearn.impute import SimpleImputer**  **imputer = SimpleImputer(strategy='median')**  **df['age'] = imputer.fit\_transform(df[['age']])** |
| Constant / Placeholder | Useful for categorical or when missing means something.  Examples:   * "Unknown" * -1 * 0 (if meaningful) | **SimpleImputer(strategy='constant', fill\_value='Unknown')** |
| Model-Based Imputation | Use ML to predict missing values using other columns.  **Techniques:**   * **KNN Imputer** * **Random Forest / Extra Trees** * **Regression models** | **from sklearn.impute import KNNImputer**  **imputer = KNNImputer(n\_neighbors=3)**  **df\_imputed = imputer.fit\_transform(df)** |
| Multivariate Imputation (MICE) | More advanced: imputes missing values multiple times to reduce bias.   * Uses relationships between columns * Best for structured tabular datasets | **from sklearn.experimental import enable\_iterative\_imputer**  **from sklearn.impute import IterativeImputer**  **imputer = IterativeImputer()**  **df\_imputed = imputer.fit\_transform(df)** |
| Use Algorithms That Handle Missing Data Automatically | Some models **natively support missing values**:  **Examples:**   * **XGBoost** * **LightGBM** * **CatBoost**   They learn optimal splits even with blanks. | |
| Add “Missing Indicator” Columns | Useful when **missingness itself carries meaning**.  Example: People who didn’t report income may behave differently. | **from sklearn.impute import SimpleImputer**  **from sklearn.impute import MissingIndicator**  **Or in one line with scikit-learn:**  **SimpleImputer(add\_indicator=True)** |

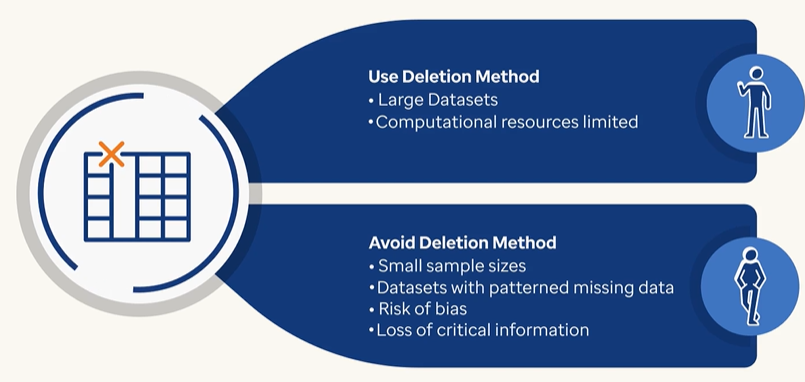
How to Decide Which Method?

|  |  |
| --- | --- |
| Situation | Best Method |
| <5% missing | Drop rows |
| Numeric with normal distribution | Mean |
| Numeric with outliers | Median |
| Categorical | Mode |
| Missingness has meaning | Constant + Missing Indicator |
| Complex relationships | MICE / KNN |
| Using tree-based boosting | Let model handle it |

##### HANDLING MISSING DATA USING DELETION

Deletion methods for handling missing values are the simplest approach, but they should be used carefully because they can lead to **loss of information**.

###### WHEN SHOULD WE OPT FOR DELETION ?



###### COMMON DELETION TECHNIQUES

LISTWISE DELETION (REMOVE ROWS)

* **Listwise deletion** removes any row that contains **at least one missing value** in any column.
* It’s simple and preserves the integrity of complete cases, but can shrink the dataset and may bias results unless missingness is **MCAR** (Missing Completely at Random).
* **Use when:**
* Missing data is **small** (e.g., <5–10% of rows).
* Missingness is **MCAR** (Missing Completely at Random).
* You have a **large dataset**, so losing a few rows won’t hurt.
* **Example:**Survey data where only 2% of responses have missing values → safe to drop those rows.
* Example (Hospital billing)

|  |  |
| --- | --- |
| * If we have **Hospital\_Provider\_Cost.csv** with columns:   1. Provider\_ID,   2. Hospital\_Name,   3. Average\_Covered\_Charges,   4. Average\_Total\_Payments,   5. State. * If only ~1–2% of rows have missing values randomly (e.g., system glitches), we can drop these rows without harming model performance. | **import pandas as pd**  **# Simulated hospital billing data**  **data = pd.DataFrame({**  **'Provider*ID': [101, 102, 103, 104, 105],***  ***'Hospital*Name': ['A', 'B', 'C', 'D', 'E'],**  **'Average*Covered*Charges': [5000, None, 7000, None, 6500],**  **'Average*Total*Payments': [4500, 4000, None, 4200, 4300],**  **'State': ['NY', 'CA', None, 'TX', None]**  **})**  **# Listwise deletion: drop any row with at least one NaN**  **listwise = data.dropna()**  **print("After Listwise Deletion:\n", listwise)**  **What happens:** Only rows with **no missing values** remain |

PAIRWISE DELETION (USE AVAILABLE DATA PER ANALYSIS)

* **Pairwise deletion means: Don’t throw away the whole row if it has some missing values. Instead, use whatever data is available for the specific calculation you are doing.**

**Use when:**

* We are doing **correlation or regression analysis** and want to keep as much data as possible.
* Missingness is **random** and we don’t want to lose entire rows.
* We need **exploratory analysis** (e.g., correlation matrix).
* **Example:**Healthcare dataset where some patients have missing blood pressure but complete cholesterol data → use pairwise deletion for correlation between cholesterol and age.

How It works?

* When you calculate something between two columns (like correlation), you only use rows where both columns have values.
* If a row is missing in one column, we skip it for that calculation only but keep it for other calculations.
* **Example**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Imagine this data:   |  |  |  | | --- | --- | --- | | Patient | Age | Blood Pressure | | A | 25 | 120 | | B | 30 | NaN | | C | NaN | 130 | | D | 40 | 140 | | * If we calculate correlation between Age and Blood Pressure, we use rows A and D (both have values). * Rows B and C are ignored only for this correlation, but they stay in the dataset for other analyses. |
| import pandas as pd  import numpy as np  data = pd.DataFrame({      'Age': [25, 30, np.nan, 40],      'BloodPressure': [120, np.nan, 130, 140]  })  # Pairwise deletion happens automatically in pandas .corr()  print(data.corr())  **# Uses only rows where both columns are non-missing** |  |

**Example 2 (Hospital billing)**

|  |  |
| --- | --- |
| We want to compute correlations:   * Between Average\_Covered\_Charges and Average\_Total\_Payments. * Between Average\_Covered\_Charges and State (not numeric, so skip or encode). With pairwise deletion, each correlation uses only rows where both variables are present. * **What happens:** Only rows where **both numeric variables are present** contribute to the correlation. Rows with NaN in either are **ignored for that pair**, but remain in the dataset otherwise. | **import pandas as pd**  **import numpy as np**  **data = pd.DataFrame({**  **'Provider*ID': [101, 102, 103, 104, 105],***  ***'Average*Covered*Charges': [5000, np.nan, 7000, np.nan, 6500],***  ***'Average*Total*Payments': [4500, 4000, np.nan, 4200, 4300],***  ***'State': ['NY', 'CA', np.nan, 'TX', np.nan]***  ***})***  ***# 2) Pairwise deletion for correlation:***  ***# Pandas .corr() uses pairwise complete observations by default.***  ***pairwise*corr = data[['Average*Covered*Charges', 'Average*Total*Payments']].corr()**  **print("Pairwise Correlation (pairwise complete rows):\n", pairwise*corr)***  ***# If you want to compute correlation manually with pairwise deletion:***  ***x = data['Average*Covered*Charges']***  ***y = data['Average*Total*Payments']***  ***mask = x.notna() & y.notna()***  ***manual*corr = x[mask].corr(y[mask])**  **print("Manual pairwise correlation:", manual\_corr)** |

ENTIRE VARIABLE DELETION (DROP COLUMNS/FEATURES)

* **Entire variable deletion** removes a column (feature) when it has **too many missing values** or is **low-importance** for the task. This is also called **column deletion**.
* It preserves row count but may remove potentially useful signal.
* Example (Hospital billing) : If State is missing in **70%** of rows and you don’t plan to use geospatial models or regional patterns, you may drop the State column to simplify preprocessing.

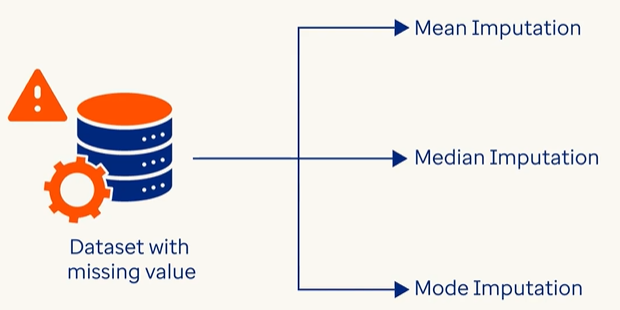
**Use when:**

* A column has **too many missing values** (e.g., >50–60%).
* The column is **not critical** for your analysis or model.
* Imputation would introduce too much bias.
* **Example:**Hospital dataset where Patient\_Email is missing in 80% of rows → drop the column because it’s not useful for cost prediction.

|  |  |  |
| --- | --- | --- |
| **import pandas as pd**  **import numpy as np**  **data = pd.DataFrame({**  **'Provider*ID': [101, 102, 103, 104, 105],***  ***'Hospital*Name': ['A', 'B', 'C', 'D', 'E'],**  **'Average*Covered*Charges': [5000, np.nan, 7000, np.nan, 6500],**  **'Average*Total*Payments': [4500, 4000, np.nan, 4200, 4300],**  **'State': ['NY', 'CA', np.nan, 'TX', np.nan]**  **})**  **# 3) Entire variable deletion: drop columns with >50% missing**  **# len(data) returns – number of rows in dataframe**  **threshold = len(data) \* 0.5  # at least 50% non-missing required**  **entire*variable*deleted = data.dropna(axis=1, thresh=threshold)**  **print("After Entire Variable Deletion (columns with too many NaNs removed):\n", entire*variable*deleted)**  **# Or explicitly drop a known problematic column:**  **data*no*state = data.drop(columns=['State'])**  **print("\nExplicitly dropped 'State' column:\n", data*no*state)**  **What happens:** Columns failing the non-missing threshold are removed. | |  |
| data = pd.DataFrame({      'ProviderID': [101, 102, 103, 104, 105],      'HospitalName': ['A', 'B', 'C', 'D', 'E'],      'AverageCoveredCharges': [5000, np.nan, 7000, np.nan, 6500],      'AverageTotalPayments': [4500, 4000, np.nan, 4200, 4300],      'State': ['NY', 'CA', np.nan, 'TX', np.nan]  }) | There are **5 rows** and **5 columns**.Missing values (np.nan) appear in:   * AverageCoveredCharges: 2 missing → 3 non-missing * AverageTotalPayments: 1 missing → 4 non-missing * State: 3 missing → 2 non-missing * ProviderID and HospitalName are fully complete (5 non-missing each). |  |
| **Goal: Entire Variable Deletion (dropping columns with too many NaNs)**   * We want to **drop columns** that fail a **non-missing threshold**—i.e., columns that have **more than 50% missing values** (equivalently, **less than 50% non-missing**).   Step 1: Set the threshold  **# len(data) returns number of rows in the dataframe**  **threshold = len(data) \* 0.5  # at least 50% non-missing required**   * **len(data) is 5 (rows).** * **threshold = 5 \* 0.5 = 2.5.**   ***Important:*** *thresh in* ***dropna*** *counts* ***non-missing entries****, not a percentage. So thresh=2.5 effectively means a column must have* ***at least 3 non-missing*** *values (because counts are integers; pandas compares against the numeric threshold).*  Step 2: Drop columns that don’t meet the threshold  **entirevariabledeleted = data.dropna(axis=1, thresh=threshold)**  **print("After Entire Variable Deletion (columns with too many NaNs removed):\n", entirevariabledeleted)**   * axis=1 → operate on **columns**.[axis=0 → drop rows; axis=1 → drop columns] * thresh=threshold → **keep columns** that have **≥ 2.5 non-missing values**, i.e., **≥ 3** non-missing.   Let’s check each column against the requirement (≥ 3 non-missing):   |  |  |  | | --- | --- | --- | | Column | Non-missing count | Keep? | | ProviderID | 5 | ✅ | | HospitalName | 5 | ✅ | | AverageCoveredCharges | 3 | ✅ | | AverageTotalPayments | 4 | ✅ | | State | 2 | ❌ (dropped) |   **Resulting DataFrame (entirevariabledeleted):**   |  | | --- | | ProviderID HospitalName AverageCoveredCharges AverageTotalPayments | | 0 101 A 5000.0 4500.0 | | 1 102 B NaN 4000.0 | | 2 103 C 7000.0 NaN | | 3 104 D NaN 4200.0 | | 4 105 E 6500.0 4300.0 |   *The State column is* ***removed*** *because it only had* ***2*** *non-missing values (below the threshold of 3).*  Alternative: Explicitly drop a known problematic column  **datanostate = data.drop(columns=['State'])**  **print("\nExplicitly dropped 'State' column:\n", datanostate)**   * This **directly removes** the State column regardless of missingness or thresholds.   **Resulting DataFrame (datanostate):**  ProviderID HospitalName AverageCoveredCharges AverageTotalPayments  0 101 A 5000.0 4500.0  1 102 B NaN 4000.0  2 103 C 7000.0 NaN  3 104 D NaN 4200.0  4 105 E 6500.0 4300.0  This result is the **same** as the threshold-based deletion in your case, because the only column that failed the threshold was State.  What’s happening conceptually?   * **Entire Variable Deletion** (column deletion) is used when a column has **too many missing values**, making it risky or meaningless to impute or use in modeling. * Using dropna(axis=1, thresh=…) is a neat way to **express a minimum completeness requirement** per column. | |

##### HANDLING MISSING DATA USING **IMPUTATION TECHNIQUES**

* The imputation techniques depends of type of data (categorical , numerical ,Data-time)
* The common imputation techniques are :



MEAN IMPUTATION

* Replace missing values with the **average** of all available values in that column.

When to Use

* Data is **numeric** and approximately **normally distributed** (bell-shaped).
* There are **no extreme outliers**, because outliers distort the mean.

Why**:** Mean represents the central tendency for symmetric distributions, so replacing missing values with the mean keeps the overall distribution balanced.

Example

|  |  |
| --- | --- |
| Column: Age = [25, 30, NaN, 35]   * Mean = (25 + 30 + 35) / 3 = **30** * Replace NaN → Age = [25, 30, 30, 35] | df['Age'].fillna(df['Age'].mean(), inplace=True) |

MEDIAN IMPUTATION

Replace missing values with the **middle value** when data is sorted.

When to Use

* Data is **numeric** but **skewed** (not normal).
* There are **outliers**, because median is robust to extreme values.

Why : Median better represents the center for skewed distributions, avoiding distortion from outliers.

|  |  |
| --- | --- |
| **Example**  Column: Income = [2000, 2500, NaN, 100000]   * Median = 2500 * Replace NaN → Income = [2000, 2500, 2500, 100000] | df['Income'].fillna(df['Income'].median(), inplace=True) |

MODE IMPUTATION

* Replace missing values with the **most frequent value** in the column.

When to Use

* Data is **categorical** (e.g., Gender, City).
* Or numeric data with repeated values (e.g., ratings).

Why :Mode preserves the most common category, which is often the safest guess for missing categorical data.

|  |  |
| --- | --- |
| **Example**   * Column: City = [NY, LA, NY, NaN] * Mode = NY * Replace NaN → City = [NY, LA, NY, NY] | df['City'].fillna(df['City'].mode()[0], inplace=True) |

A graph of a number of bars

AI-generated content may be incorrect.

|  |
| --- |
| **✅ Real-Time Example (Hospital Billing Data)**  Imagine a dataset:   * Average\_Covered\_Charges → numeric, normal → use **Mean** * Average\_Total\_Payments → numeric, skewed → use **Median** * State → categorical → use **Mode**   **Python Implementation:**  import pandas as pd  import numpy as np  data = pd.DataFrame({      'Average*Covered*Charges': [5000, np.nan, 7000, np.nan, 6500],      'Average*Total*Payments': [4500, 4000, np.nan, 4200, 4300],      'State': ['NY', 'CA', np.nan, 'TX', np.nan]  })  # Mean for normal numeric column  data['Average*Covered*Charges'].fillna(data['Average*Covered*Charges'].mean(), inplace=True)  # Median for skewed numeric column  data['Average*Total*Payments'].fillna(data['Average*Total*Payments'].median(), inplace=True)  # Mode for categorical column  data['State'].fillna(data['State'].mode()[0], inplace=True)  print(data) |
| How to Measure the skewness in dataset |

|  |  |  |
| --- | --- | --- |
| Step 3: Handling MEAN IMPUTATION TECHNIQUES FOR Missing Data   * For missing data, we can make use of Python library **scikit-learn**. It is an open-source machine learning library built on top of **NumPy**, **SciPy**, and **matplotlib**. * To Install **scikit-learn**: **pip install scikit-learn** * It provides simple and efficient tools for data mining and data analysis. * For example - For missing salary - We will replace the missing salary with average salary in the column   **from sklearn.impute import SimpleImputer**  **imputer= SimpleImputer(missing\_values=np.nan, strategy='mean') imputer.fit(X[:, 1:3]) # Assuming columns 1 and 2 have missing values X[:, 1:3] = imputer.transform(X[:, 1:3]) # Transform the data to fill missing values**   * SimpleImputer from scikit-learn to fill missing values (NaN) in specific columns of the feature matrix X with the mean of each column. * It fits the imputer on columns 1 and 2 (indexing starts at 0), replaces missing values with the computed mean | | |
| **imputer = SimpleImputer(missing\_values=np.nan, strategy='mean')** | * This tool is used to fill in missing values (np.nan) in your dataset. * 'mean' means it will replace missing values with the **mean of the column**. | |
| **imputer.fit(X[:, 1:3])** | * Fits the imputer to columns 1 and 2 of X (i.e., second and third columns). * **It** Calculates the **mean** of each column (ignoring NaN values) so it knows what to use for imputation. | |
| **X[:, 1:3] = imputer.transform(X[:, 1:3])** | * Applies the imputation — replaces NaN values in columns 1 and 2 with the calculated means. * The dataset X now has no missing values in those columns. | |
| A diagram of a strategy  AI-generated content may be incorrect. | | 1. A sample dataset with missing values. 2. The imputer calculating column means. 3. The transformation step where missing values are replaced. |

##### HANDLING MISSING DATA USING NON-**IMPUTATION TECHNIQUES**

## STEP 4: ENCODING THE CATEGORIAL DATA

* Encoding categorical data is **crucial in machine learning** because most ML algorithms require **numerical input** to perform mathematical computations.

### COMMON ENCODING TECHNIQUES

|  |  |  |
| --- | --- | --- |
| Technique | Description | Best For |
| Label Encoding | Assigns A Unique Number To Each Category | Ordinal Data (E.G., "Low", "High") |
| One-Hot Encoding | Creates Binary Columns For Each Category | Nominal Data (E.G., "Red", "Blue") |
| Ordinal Encoding | Encodes Categories With Meaningful Order | Ordered Categories |
| Target Encoding | Replaces Categories With The Mean Of The Target Variable For Each Category | High-Cardinality Categorical Data |

### ONE HOT ENCODING

* One-hot encoding is a way to convert text categories into numbers so that machine learning models can understand them.
* In one hot encoding the create binary vectors of each category

|  |  |  |
| --- | --- | --- |
| **Why Do We Need It?** | **Bad Way: Label Encoding** | **Good Way: One-Hot Encoding** |
| * Machine learning models work with **numbers**, not **words**. * If we have a column like:  | **Fruit** | | --- | | Apple | | Banana | | Orange | | Banana |  * We can't just give these words to a model. * We need to **convert them into numbers** — but in a smart way. | If we do this:   * Apple → 0 * Banana → 1 * Orange → 2   Then the model might think **Orange is greater than Banana**, which is **not true**. These are just different categories, not ranked values. | Instead, we create **new columns** — one for each category — and use 1 or 0 to show which one is present.   | **Fruit\_Apple** | **Fruit\_Banana** | **Fruit\_Orange** | | --- | --- | --- | | 1 | 0 | 0 | | 0 | 1 | 0 | | 0 | 0 | 1 | | 0 | 1 | 0 |   Now:   * Each fruit is represented by a **row of binary values**. * No false relationships between categories. * The model can treat each fruit **independently**. |

A diagram of fruit and fruit

AI-generated content may be incorrect.

Note: In the above example- We have categorial data – hence has bee transformed to 3 columns using One Hot encoding. The category column will be replaced by 3 columns, which are binary vector of each category. Similarly, if we have more categories – those many columns will be created to uniquely identify each category.

Why Is It Important?

* It helps models **understand categorical data** correctly.
* Prevents **wrong assumptions** about order or importance.
* Works well with algorithms like **linear regression**, **decision trees**, and **neural networks**.

CODE

Taking the example further will apply “One Hot Encoding” of “County” column and “label encoding” to the “Purchased” column.

One Hot Encoding OF Country Column

|  |
| --- |
| **from sklearn.compose import ColumnTransformer**  **from sklearn.preprocessing import OneHotEncoder**  ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(),[0])],remainder='passthrough') X = np.array(ct.fit\_transform(X)) # Apply one-hot encoding to the first column |

|  |
| --- |
| **ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [0])], remainder='passthrough')**   * **transformers=[...]**: It will be list of tuples.   + 'encoder': Name for the transformation.   + OneHotEncoder(): This is the actual transformation — it will convert the Country column into one-hot encoded format.   + [0]: This means apply it to **column 0**, which is Country. * **remainder='passthrough'**: This tells it to **keep all other columns** (Age, Salary, Purchased) as they are. |
| X = np.array(ct.fit\_transform(X))   * ct.fit\_transform(X):   + **fit**: Learns the unique values in the Country column (France, Spain, Germany).   + **transform**: Converts each country into a one-hot encoded row. * np.array(...): Converts the result into a NumPy array, which is easier to use in ML models. |

The original data:

|  |  |  |  |
| --- | --- | --- | --- |
| Country | Age | Salary | Purchased |
| France | 44 | 72000 | No |
| Spain | 27 | 48000 | Yes |
| Germany | 30 | 54000 | No |
| … | … | … | … |

Becomes:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Country\_France | Country\_Germany | Country\_Spain | Age | Salary | Purchased |
| 1 | 0 | 0 | 44 | 72000 | No |
| 0 | 0 | 1 | 27 | 48000 | Yes |
| 0 | 1 | 0 | 30 | 54000 | No |
| … | … | … | … | … | … |

LABELLED Encoding OF Purchased Column

* **Label Encoding** is a method used to convert **categorical text data** into **numerical values** — but in a way that assigns a **unique number to each category**.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| The purchased column contains **categorical values**: "Yes" and "No".   |  | | --- | | Purchased | | No | | Yes | | No | | No | | Yes | | Yes | | No | | Yes | | No | | Yes | | **What Label Encoding Does**  Label Encoding converts these **text categories** into **numbers**:   |  |  | | --- | --- | | Purchased | Encoded | | No | 0 | | Yes | 1 | | So purchased column becomes:   |  | | --- | | Purchased | | 0 | | 1 | | 0 | | 0 | | 1 | | 1 | | 0 | | 1 | | 0 | | 1 | |

Note: Label encoding is **perfect for binary categories** like "Yes" and "No" because:

* There are only **two values**.
* The model can easily understand 0 and 1 as **distinct classes**.
* **When to Avoid Label Encoding:** If we have a column like "Low", "Medium", "High" or "Red", "Green", "Blue", label encoding might **mislead the model** into thinking there's a ranking or distance between values — which may not be true.In such cases, **One-Hot Encoding** is safer.

|  |
| --- |
| from sklearn.preprocessing import LabelEncoder  le = LabelEncoder() y = le.fit\_transform(y) # Apply label encoding to the dependent variable |

|  |
| --- |
| **LabelEncoder**   * **Purpose**: Converts categorical labels (usually target variable or single categorical feature) into numeric codes. * **Use Case**: When we have a **single categorical column** (often the target variable in classification) and need to convert categories like ["cat", "dog", "mouse"] into integers [0, 1, 2]. * **Important Notes**:   + It does not handle multiple columns at once.   + It is not suitable for features that will be used in models expecting one-hot encoding or ordinal encoding unless the categories have a natural order. * **Example**:   **from sklearn.preprocessing import LabelEncoder**  **le = LabelEncoder()**  **y = le.fit\_transform(['cat', 'dog', 'mouse'])**  **ColumnTransformer**   * **Purpose**: Applies different transformations to **multiple columns** in a dataset, often combining numeric and categorical preprocessing. * **Use Case**: When we have a **DataFrame with mixed types** (numeric + categorical) and need to apply different transformations (e.g., scaling numeric columns, one-hot encoding categorical columns). * **Important Notes**:   + Works well with pipelines.   + We can apply multiple transformers in parallel to different subsets of columns. * **Example**:   **from sklearn.compose import ColumnTransformer**  **from sklearn.preprocessing import OneHotEncoder, StandardScaler**  **preprocessor = ColumnTransformer(transformers=[('num', StandardScaler(), ['age', 'income']), ('cat', OneHotEncoder(), ['gender', 'city'])**      )  When to use which?   * **Use LabelEncoder**:   + For encoding the **target variable** in classification tasks.   + For a single categorical feature when integer encoding is acceptable. * **Use ColumnTransformer**:   + For **feature preprocessing** when you have multiple columns and need different transformations.   + When building **pipelines** for ML models. |

EXAMPLE – CODING EXERCISE

Dataset

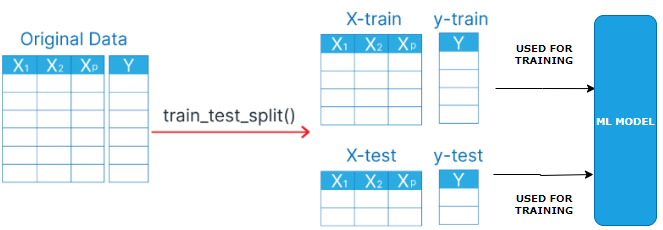
|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | Cabin | Embarked |
| 2 | 1 | 1 | Cumings, Mrs. John Bradley (Florence Briggs Thayer) | female | 38 | 1 | 0 | PC 17599 | 71.2833 | C85 | C |
| 4 | 1 | 1 | Futrelle, Mrs. Jacques Heath (Lily May Peel) | female | 35 | 1 | 0 | 113803 | 53.1 | C123 | S |
| 7 | 0 | 1 | McCarthy, Mr. Timothy J | male | 54 | 0 | 0 | 17463 | 51.8625 | E46 | S |
| 11 | 1 | 3 | Sandstrom, Miss. Marguerite Rut | female | 4 | 1 | 1 | PP 9549 | 16.7 | G6 | S |
| 12 | 1 | 1 | Bonnell, Miss. Elizabeth | female | 58 | 0 | 0 | 113783 | 26.55 | C103 | S |
| 22 | 1 | 2 | Beesley, Mr. Lawrence | male | 34 | 0 | 0 | 248698 | 13 | D56 | S |

Coding Exercise 3: Encoding Categorical Data for Machine Learning

1. Import required libraries - Pandas, Numpy, and required classes for this task - ColumnTransformer, OneHotEncoder, LabelEncoder.
2. Start by loading the Titanic dataset into a pandas data frame. This can be done using the pd.read\_csv function. The dataset's name is 'titanic.csv'.
3. Identify the categorical features in your dataset that need to be encoded. You can store these feature names in a list for easy access later.
4. To apply OneHotEncoding to these categorical features, create an instance of the ColumnTransformer class. Make sure to pass the OneHotEncoder() as an argument along with the list of categorical features.
5. Use the fit\_transform method on the instance of ColumnTransformer to apply the OneHotEncoding.
6. The output of the fit\_transform method should be converted into a NumPy array for further use.
7. The 'Survived' column in your dataset is the dependent variable. This is a binary categorical variable that should be encoded using LabelEncoder.
8. Print the updated matrix of features and the dependent variable vector

|  |
| --- |
| # Importing the necessary libraries import pandas as pd import numpy as np from sklearn.compose import ColumnTransformer from sklearn.preprocessing import OneHotEncoder, LabelEncoder  # Load the dataset dataset = pd.read\_csv("titanic.csv")  # Identify the categorical data categorical\_features = ['Sex', 'Embarked', 'Pclass']  # Implement an instance of the ColumnTransformer class  ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(),categorical\_features)],remainder='passthrough')  # Apply the fit\_transform method on the instance of ColumnTransformer ct\_fit = ct.fit\_transform(dataset) # Apply one-hot encoding to the first column  # Convert the output into a NumPy array X = np.array(ct.fit\_transform(dataset))  # Use LabelEncoder to encode binary categorical data le = LabelEncoder()  Y = le.fit\_transform(dataset["Survived"]) # Apply label encoding to the dependent variable  # Print the updated matrix of features and the dependent variable vector print(X) print(Y) |

## STEP 5: SPLITING TRAINING VERSUS TEST DATA



|  |
| --- |
| **from sklearn.model\_selection import train\_test\_split**  **x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=1)** |

After preprocessing (like one-hot encoding and label encoding), we have:

* X: All columns **except** Purchased (features)
* y: The Purchased column (target)

|  |  |  |
| --- | --- | --- |
| **train\_test\_split(...)** | **Parameters Explained** | **Output Variables** |
| This function **splits your dataset** into two parts:   * **Training set**: Used to train the machine learning model. * **Testing set**: Used to evaluate how well the model performs on unseen data. | * X: Input features (Country, Age, Salary) * y: Target labels (Purchased) * test\_size=0.2: 20% of the data will go into the test set, and 80% into the training set. * **random\_state=1**: Ensures the split is **reproducible** — same split every time we run it. If we omit random\_state, the split changes every run. | * x\_train: Features for training * x\_test: Features for testing * y\_train: Labels for training * y\_test: Labels for testing |

EXAMPLE

1. Import necessary Python libraries: pandas, train\_test\_split from sklearn.model\_selection, and StandardScaler from sklearn.preprocessing.
2. Load the Iris dataset using Pandas read.csv. Dataset name is iris.csv.
3. Use train\_test\_split to split the dataset into an 80-20 training-test set.
4. Apply random\_state with 42 value in train\_test\_split function for reproducible results.
5. Print X\_train, X\_test, Y\_train, and Y\_test to understand the dataset split.
6. Use StandardScaler to apply feature scaling on the training and test sets.
7. Print scaled training and test sets to verify feature scaling.

CODE

|  |
| --- |
| **# Import necessary libraries**  **import pandas as pd**  **import sklearn.model\_selection as model\_selection**  **# Load the Iris dataset**  **dataset = pd.read\_csv("iris.csv")**  **# Separate features and target**  **X = dataset.iloc[:, :-1].values # Features (all columns except the last)**  **y = dataset.iloc[:, -1].values**  **# Split the dataset into an 80-20 training-test set**  **X\_train, X\_test,y\_train,y\_test = model\_selection.train\_test\_split(X, y, test\_size=0.2, random\_state=42)**  **print("Training set features:\n", X\_train)**  **print("Training set labels:\n", y\_train)**  **print("Test set features:\n", X\_test)**  **print("Test set labels:\n", y\_test)**  **# Apply StandardScaler feature scaling on the training and test sets**  **from sklearn.preprocessing import StandardScaler**  **scaler = StandardScaler()**  **X\_train = scaler.fit\_transform(X\_train) # Fit and transform the training set**  **X\_test = scaler.transform(X\_test) # Transform the test set using the same scaler**  **# Print the scaled training and test sets**  **print("Scaled Training set features:\n", X\_train)**  **print("Scaled Test set features:\n", X\_test)** |

## STEP 6: FEATURE SCALING

* **Feature Scaling** is a technique used to **normalize or standardize** the range of independent variables (features) in the dataset.
* In simple terms: It makes sure that all the features (like Age, Salary, etc.) are on a **similar scale**, so that no one feature dominates the others just because it has larger numbers.

**Why Is It Important?**

Let’s say your dataset has:

* Age: ranges from 20 to 60
* Salary: ranges from 20,000 to 100,000

Feature scaling is not required for all the ML models but like **linear regression**, **KNN**, or **SVM** might give more importance to Salary just because its values are bigger — even if Age is equally important.

Common feature Scaling Methods

|  |  |
| --- | --- |
| **Standardization(Recommended)** | **NORMALIZATION** |
|  |  |

Note

* Feature scaling is has to be done for training and test data separately
* Never to the feature scaling on whole dataset

### NORMALIZATION OR STANDARDIZATION

* Normalization is recommended when the features follow (or approximately follow) a **normal distribution**

How will I Know the Dataset is Normalized?

**Key Indicators of Normalization**

1. **Mean ≈ 0**
2. **Standard Deviation ≈ 1**
3. **Distribution shape remains the same** (if you used StandardScaler)

|  |  |
| --- | --- |
| **How to Verify in Python**  import pandas as pd  from sklearn.preprocessing import StandardScaler  **# Example data**  data = pd.DataFrame({'Height': [170, 180, 160, 175],                       'Weight': [65, 75, 55, 70]})  **# Apply normalization**  scaler = StandardScaler()  normalized = scaler.fittransform(data)  **# Convert back to DataFrame for easy checking**  normalizeddf = pd.DataFrame(normalized, columns=['Height', 'Weight'])  **# Check mean and std**  print("Means:\n", normalizeddf.mean())  print("Standard Deviations:\n", normalizeddf.std()) | **Expected Output:**  Means:  Height ~0  Weight ~0  Standard Deviations:  Height ~1  Weight ~1 |

* **Note :** When we plot **histograms** before and after normalization to see:
* Original data: different scales
* Normalized data: centered around 0, spread within ±3

A group of different colored graphs

AI-generated content may be incorrect.

**What you should notice:**

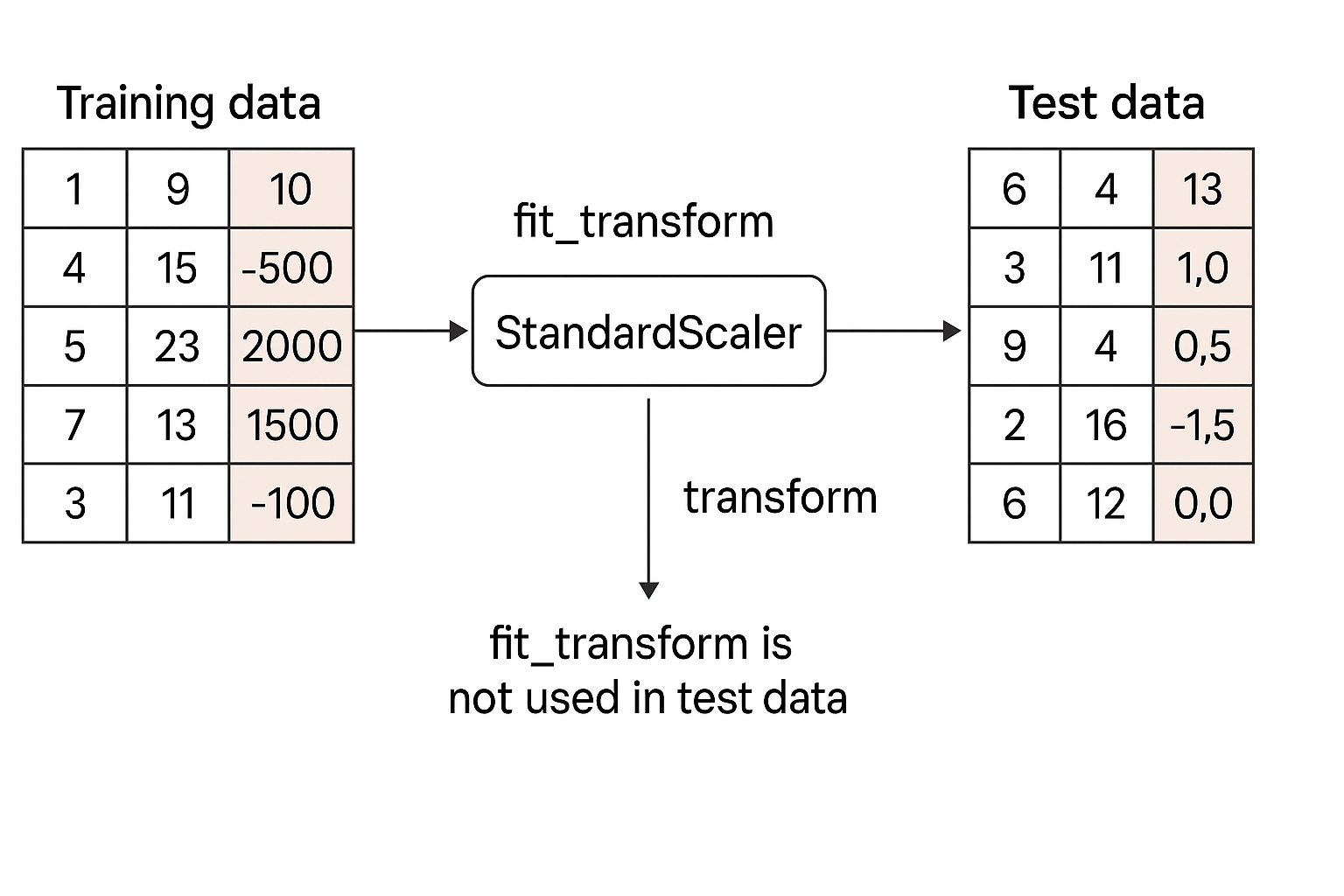
* The **shape** of each distribution is largely preserved (still bell-shaped).
* After normalization, both features are **centered around 0** and have **unit variance**.
* This puts features on a comparable scale, which is beneficial for algorithms sensitive to feature magnitude (e.g., KNN, SVM, PCA).

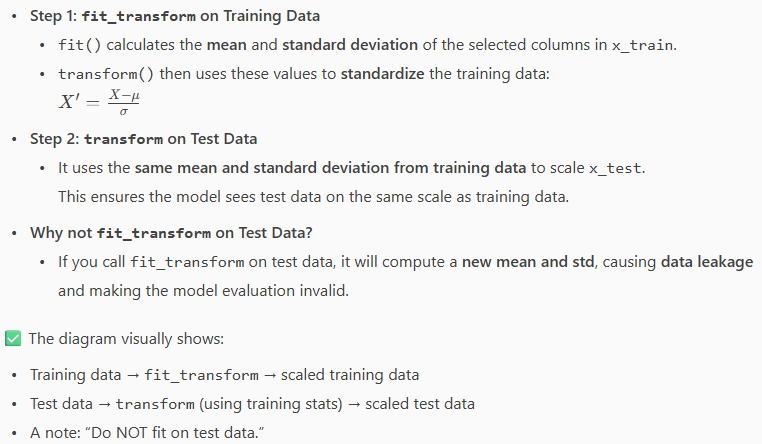
### APPLYING FEATURE SCALING – STANDARDIZATION

**Note: Feature scaling (standardization) is not required for columns produced through one-hot encoding*.***

|  |
| --- |
| sc = StandardScaler()  x\_train[:, 3:] = sc.fit\_transform(x\_train[:, 3:])  x\_test[:, 3:] = sc.transform(x\_test[:, 3:]) |

|  |  |
| --- | --- |
| **sc.fit\_transform(x\_train[:, 3:])** | **fit\_transform** does two things:   * **fit**: Calculates the **mean** and **standard deviation** of the training data. * **transform**: Uses those values to scale the training data. |
| **sc.transform(x\_test[:, 3:])** | * **transform** applies the **same scaling** (using the mean and std from training data) to the test data.   **Why not fit\_transform on test data?**  Because:   * We **already fitted** the scaler on the training data. * We want to apply **the same transformation** to the test data. * Using fit\_transform on test data would calculate a **different mean and std**, which breaks consistency. |





## FINAL CODE

|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  from sklearn.impute import SimpleImputer  from sklearn.compose import ColumnTransformer  from sklearn.preprocessing import OneHotEncoder, LabelEncoder, StandardScaler  from sklearn.model\_selection import train\_test\_split  dataset = pd.read\_csv("Data.csv")  X = dataset.iloc[:, :-1].values  Y = dataset.iloc[:, -1].values  imputer = SimpleImputer(missing\_values=np.nan, strategy="mean")  imputer.fit(X[:, 1:3])  X[:, 1:3] = imputer.transform(X[:, 1:3])  ct = ColumnTransformer(transformers=[("encoder", OneHotEncoder(), [0])], remainder="passthrough")  X = np.array(ct.fit\_transform(X))  le = LabelEncoder()  Y = le.fit\_transform(Y)  x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=1)  sc = StandardScaler()  x\_train[:, 3:] = sc.fit\_transform(x\_train[:, 3:])  x\_test[:, 3:] = sc.transform(x\_test[:, 3:]) |

# REGRESSION

* A **regression model** is a type of statistical or machine learning model used to understand the relationship between a **dependent variable** (what you're trying to predict) and one or more **independent variables** (the inputs or predictors).
* **In simple terms:** A regression model helps answer questions like:
  + "How does the price of a house depend on its size, location, and number of bedrooms?"
  + "How does advertising spending affect sales?"

Types Of Regression Models

* Linear Regression
* Assumes a straight-line relationship between variables.
* Example: y = a + bx
* Multiple Linear Regression:
* Like linear regression, but with multiple input variables.
* Example: y = a + b*1x*1 + b*2x*2 + … + b*nx*n
* Polynomial Regression:
* Models curved relationships by including powers of the input variables.
* Example: y = a + bx + cx2
* Logistic Regression:
* Used when the output is categorical (e.g., yes/no, 0/1).
* Despite the name, it's used for classification, not regression.
* Ridge, Lasso, And Elastic Net Regression:
* Variants of linear regression that include regularization to prevent overfitting.

What It’s Used For

* Predicting future values (e.g., stock prices, sales).
* Understanding relationships between variables.
* Making data-driven decisions in business, science, and engineering.

## SIMPLE LINEAR REGRESSION

* Linear Regression is a **supervised learning algorithm** used to model the relationship between a **dependent variable (target)** and one or more **independent variables (predictors)** by fitting a straight line

A diagram of equations

AI-generated content may be incorrect.

Example: Predict potato yield (tons) based on amount of fertilizer used (kg)

A diagram of a formula

AI-generated content may be incorrect.

From Visualization perspective, if we plot the scatter plot

* X-axis: Fertilizer (kg)
* Y-axis: Potato yield (tons)
* Each point = one harvest record

The Regression line summarizes the relationship between fertilizer and yield helps predict future yields based on fertilizer usage

### ORDINARY LEAST SQUARES

It’s a method used in **linear regression** to draw the “best fit” line through the data points.

* Imagine we have a scatter plot of points (x, y) and we want a straight line that predicts y from x.
* OLS chooses the line so that the **sum of the squared differences** between the actual y-values and the predicted y-values is as small as possible.

**Why squared differences?** Squaring makes all errors positive and penalizes large errors more.

Ordinary Least Square= “Find the line that makes the total squared error as small as possible.”

A diagram of a graph

AI-generated content may be incorrect.

* Residual**:** The difference between actual and predicted values: Residual = yi − ŷi.
* **OLS Goal:** Minimize the **sum of squared residuals** (SSE): SSE = ∑(yi − ŷi)².
* **Why square residuals?**
  + Ensure errors are positive
  + Penalizes larger mistakes more

Best Regression Line: The one with the smallest SSE, fitting closest to all data points.

### EXAMPLE

|  |  |
| --- | --- |
| * In the below example- We will predict the salary based on years of experience. | **Salary\_Data.csv**  YearsExperience,Salary  1.1,39343.00  1.3,46205.00  1.5,37731.00  ….. |

### SETTING UP LINEAR REGRESSION

|  |  |
| --- | --- |
| STEP 1: Importing the libraries | import numpy as np  import matplotlib.pyplot as plt  import pandas as pd |
| STEP 2: Importing the dataset | dataset = pd.read\_csv('Salary\_Data.csv')  X = dataset.iloc[:, :-1].values  y = dataset.iloc[:, -1].values |
| STEP 3: Splitting the dataset into the Training set and Test set | from sklearn.model\_selection import train\_test\_split  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 1/3, random\_state = 0) |

#### STEP 4: TRAINING THE SIMPLE LINEAR REGRESSION MODEL ON THE TRAINING SET

The linear regression model is trained with training data

|  |  |
| --- | --- |
| Step 4: Training the Simple Linear Regression model on the Training set | from sklearn.linear\_model import LinearRegression  regressor = LinearRegression()  **regressor.fit(X\_train, y\_train)** |

#### STEP 5: PREDICTING THE TEST SET RESULTS

* Predication of salary(y\_pred) for the test data(X\_test)

|  |  |
| --- | --- |
| STEP 5: Predicting the Test set results | **y\_pred = regressor.predict(X\_test)** |

#### STEP 6: VISUALISING THE TRAINING SET RESULTS

|  |  |
| --- | --- |
| * In the below visualization we plotted the scatter plot(in red) of the actual data(training data) * The blue line(regression line) is created using actual training data(years of experience) and predicted salary by model * If the actual salary plots are close to the regression line means the prediction of the salary by model is more accurate | plt.scatter(X\_train, y\_train, color = 'red')  plt.plot(X\_train, regressor.predict(X\_train), color = 'blue')  plt.title('Salary vs Experience (Training set)')  plt.xlabel('Years of Experience')  plt.ylabel('Salary')  plt.show() |
|  |

#### STEP 7: VISUALISING THE TEST SET RESULTS

|  |  |
| --- | --- |
| * Same regression model is used to predict the salary of the test data * If the model is accurate – the predicted salary (regression line) will be close to the actual test data | plt.scatter(X\_test, y\_test, color = 'red')  plt.plot(X\_train, regressor.predict(X\_train), color = 'blue')  plt.title('Salary vs Experience (Test set)')  plt.xlabel('Years of Experience')  plt.ylabel('Salary')  plt.show() |
|  | A graph with red dots and a blue line  AI-generated content may be incorrect. |

### CODE

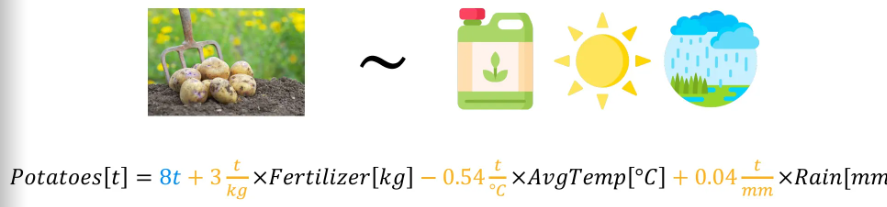
|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  from sklearn.model\_selection import train\_test\_split  from sklearn.linear\_model import LinearRegression  # Import the dataset  dataset = pd.read\_csv("Salary\_Data.csv")  # Separate features and target  X = dataset.iloc[:, :-1].values  Y = dataset.iloc[:, -1].values  # Split the dataset into an 80-20 training-test set  X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=0)  print(X\_test)  ## Training the Simple Linear Regression model on the Training set  regressor = LinearRegression()  regressor.fit(X\_train, Y\_train)  plt.scatter(X\_train, Y\_train, color="red")  plt.plot(X\_train, regressor.predict(X\_train), color="blue")  plt.title("Salary vs Experience (Training set)")  plt.xlabel("Years of Experience")  plt.ylabel("Salary")  plt.show()  y\_pred = regressor.predict(X\_test)  plt.scatter(X\_test, Y\_test, color="red")  plt.plot(X\_test, y\_pred, color="blue")  plt.title("Salary vs Experience (Test set)")  plt.xlabel("Years of Experience")  plt.ylabel("Salary")  plt.show() |

## MULTIPLE LINEAR REGRESSION

A diagram of mathematical equations

AI-generated content may be incorrect.

### EXAMPLE



**Potato Yield Prediction Example**

* **Goal**: Predict potato yield based on multiple factors.
* **Independent variables**:
  1. **Fertilizer used (kg of nitrogen)**
     + Coefficient: **+3 tons/kg**
     + Interpretation: More fertilizer → higher yield.
  2. **Average temperature (°C)**
     + Coefficient: **−0.54 tons/°C**
     + Interpretation: Higher temperature → lower yield.
  3. **Rainfall (mm)**
     + Coefficient: **+0.04 tons/mm**
     + Interpretation: More rainfall → slightly higher yield.
* **Intercept (β₀)**:
  1. **8 tons** (baseline yield without any inputs)

### EXAMPLE OF MUTIPLE LINEAR REGRESSION

DATASET – COMPANY DATA

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **R&D Spend** | **Administration** | **Marketing Spend** | **State** | **Profit** |
| **165349.2** | **136897.8** | **471784.1** | **New York** | **192261.83** |
| **162597.7** | **151377.59** | **443898.53** | **California** | **191792.06** |
| **153441.51** | **101145.55** | **407934.54** | **Florida** | **191050.39** |
| **144372.41** | **118671.85** | **383199.62** | **New York** | **182901.99** |
| **142107.34** | **91391.77** | **366168.42** | **Florida** | **166187.94** |
| **131876.9** | **99814.71** | **362861.36** | **New York** | **156991.12** |
| **….** | **…** | **…** | **…** | **…** |

**Rows:** 50 (each row = one company);**Columns:** 5

1. **R&D Spend**: Amount spent on research and development (numeric)
2. **Administration**: Amount spent on administration (numeric)
3. **Marketing Spend**: Amount spent on marketing (numeric)
4. **State**: Location of the company (categorical: New York, California, Florida)
5. **Profit**: Profit earned by the company (numeric, target variable)

Multiple Linear Regression Application

**Objective:**Build a model to predict **Profit** (dependent variable) using the other columns (independent variables).

**Independent Variables:**

* **R&D Spend:** Does investing more in R&D lead to higher profits?
* **Administration:** How does spending on administration affect profit?
* **Marketing Spend:** Is there a link between marketing budget and profit?
* **State:** Does the company’s location impact profit, after accounting for spending?

**Dependent Variable:**

* **Profit:** The outcome the venture capitalist fund cares about.

Business Challenge Context

* **Scenario:** A venture capitalist fund wants to know which types of companies (based on spending and location) are likely to be most profitable.

Why Multiple Regression for such dataset?

* It allows us to **quantify** the effect of each variable on profit, **holding others constant**.
* For example, we can answer the questions lime:
  + Does higher R&D spend increase profit, regardless of state or marketing spend?
  + Is it better to invest in companies in New York or California, all else equal?
  + Should they prioritize companies with high marketing spend or high R&D spend?
* **Outcome:**The model will help the fund set **investment guidelines** (e.g., prefer companies with high R&D spend in New York).

**🧑‍💻 How the Model Works**

**Equation Example:**  
$\text{Profit} = \beta*0 + \beta*1 \times \text{R\&D Spend} + \beta*2 \times \text{Administration} + \beta*3 \times \text{Marketing Spend} + \beta*4 \times \text{State}*{NY} + \beta*5 \times \text{State}*{CA}$

**State** is encoded as dummy variables (since it’s categorical).

**Interpretation:**

Each coefficient ($\beta$) tells you how much profit changes with a unit increase in that variable, keeping others constant.

**✅ Summary**

**You’ll use multiple linear regression** to analyze how spending and location affect profit.

**The model helps the VC fund** decide which companies to invest in, based on predicted profit.

### ASSUMPTION OF LINEAR REGRESSION

* Linear regression only works well if certain conditions (assumptions) are met. Those assumptions are

#### LINEARITY

|  |  |
| --- | --- |
| A diagram of a relationship between y and y  AI-generated content may be incorrect. | * The relationship between the variables (independent and dependent variables) should be straight (linear). ***If the data forms a curve, linear regression isn’t suitable*** |

#### HOMOSCEDASTICITY (EQUAL VARIANCE)

|  |  |
| --- | --- |
| A diagram of a diagram of a number of dots  AI-generated content may be incorrect. | * The spread of data points should be roughly the same across all values. ***If the spread gets wider or narrower (like a cone), results can be unreliable****.* |

#### NORMALITY OF ERRORS

|  |  |
| --- | --- |
| A diagram of a normality  AI-generated content may be incorrect. | * The differences between actual and predicted values (errors) should follow a normal (bell-shaped) distribution. ***If errors are skewed or have strange patterns, predictions may be off.*** |

What is Normality of Errors?

* In linear regression, the errors (also called residuals) are the differences between the actual values and the values predicted by your model.
* Normality of errors means these residuals should follow a normal distribution (the classic bell-shaped curve).

Suppose you’re predicting house prices: You build a linear regression model using features like size, location, and number of bedrooms. For each house, you calculate the error:

**Error = Actual Price - Predicted Price**

Check for Normality:

* Plot a histogram of all these errors.
* If the histogram looks like a bell curve (most errors are close to zero, fewer are very large or very small), your errors are normally distributed.
* If the histogram is skewed or has strange peaks, your errors are not normal.

#### INDEPENDENCE OF OBSERVATIONS (NO AUTOCORRELATION)

|  |  |
| --- | --- |
| A close-up of a diagram  AI-generated content may be incorrect. | * Each data point should be independent. **If there’s a pattern (like in time series data), previous values affect future ones, which breaks this rule.** |

#### NO MULTICOLLINEARITY

|  |  |
| --- | --- |
| A close-up of a sign  AI-generated content may be incorrect. | * The independent variables shouldn’t be too closely related to each other. ***If they are, it’s hard to tell which variable is influencing the result.*** |

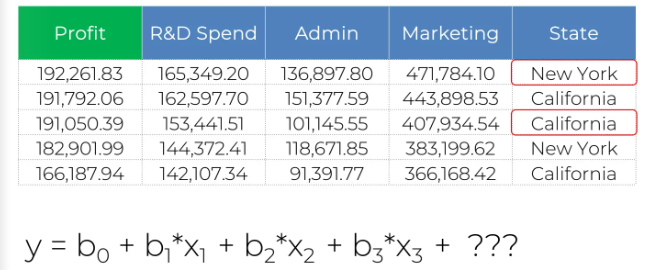
OUTLIER CHECK (EXTRA)

|  |  |
| --- | --- |
| A diagram of a line with dots  AI-generated content may be incorrect. | * Outliers (extreme values) can distort your regression line. ***You need to decide whether to remove them or keep them, based on your understanding of the data.*** |

|  |
| --- |
| * **Don’t blindly use linear regression!** * **Always check these assumptions to make sure your model is valid.** * **If the assumptions aren’t met, consider using a different model or fixing the data.** |

### HANDLING CATEGORICAL VARIABLES IN LINEAR REGRESSION MODELS

##### DUMMY VARIABLE



## POLYNOMIAL REGRESSION

## SUPPORT VECTOR REGRESSION

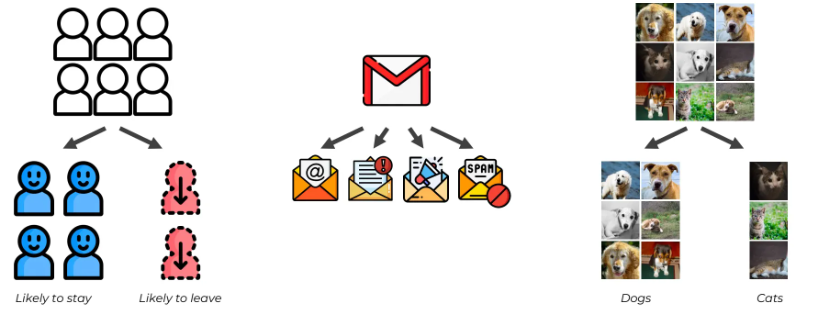
## DECISION TREE REGRESSION

## RANDOM FOREST REGRESSION

# CLASSIFICATION

A machine learning technique to identify the category of new observations based on training data

Examples



## LOGISTIC REGRESSION

* **Logistic Regression** is a statistical method used for **binary classification** problems — where the output is one of two possible categories (e.g., yes/no, spam/not spam, pass/fail).
* Instead of predicting a continuous value like linear regression, logistic regression predicts the **probability** that a given input belongs to a particular class.
* It uses the **logistic (sigmoid) function** to squeeze the output of a linear equation into a **range between 0 and 1**

|  |  |
| --- | --- |
| A black text with black letters  AI-generated content may be incorrect. | A white background with black text  AI-generated content may be incorrect. |

Example