**Machine Learning**

Machine Learning is the process of teaching computers to learn from data and make decisions or predictions **without being explicitly programmed**.

**Traditional Programming vs Machine Learning**

| **Aspect** | **Traditional Programming** | **Machine Learning** |
| --- | --- | --- |
| How it works | You write rules + give input → get output | You give input + output (examples) → computer learns the rules |
| Example task | Write rules to calculate tax | Train ML model to predict house price |
| Who creates the logic | Human programmer | Machine learns logic from data |

**Machine Learning way:**

* Feed the computer 1000 spam and 1000 non-spam emails.
* The computer **finds patterns** in the words, frequency, etc.
* Then it can **predict spam** on new emails.

[COLAB]

Use cases

**School Bus Pass Eligibility:** Students above class 5 get a free bus pass.

**Discount Eligibility Based on Purchase Count:** Customers who purchase more than 3 times get a discount.

MLModel

1. **Collect image data**  
   (Label: cancer / no cancer)
2. **Preprocess the images**  
   (Resize, normalize)
3. **Build a Model**
4. **Train the model on labeled images**
5. **Predict on new images**
6. **Evaluate model accuracy**

# Machine Learning Applications

## 1. Healthcare

**Disease Diagnosis:** Detecting cancer from X-rays

**Personalized Treatment:** Recommending dosage for diabetics

**Medical Imaging:** Brain tumor detection using CNN

**Clinical Decision Support System:** Using LLMs to assist doctors

## 2. Business & Finance

**Fraud Detection:** Credit card fraud alerts

**Credit Scoring:** Bank approves loans using ML scores

**Customer Segmentation:** Targeted marketing

**Chatbots:** ChatGPT, bank chatbots

## 3. Retail & E-commerce

**Recommendation Systems:** Amazon's 'You may also like'

1. **Collect or create a dataset**
   * Sample: Product names, descriptions, categories, ratings.
   * Example: Product\_Name, Description, Category, Price.
2. **Preprocess data**
   * Clean text (remove stop words, lowercasing).
   * Convert product descriptions into numerical format using TF-IDF.
3. **Compute similarity**
   * Use **cosine similarity** or **Euclidean distance** between items.
4. **Build recommendation logic**
   * For a given product, find top N similar products.
5. **Display results**
   * Print or display "You may also like..." suggestions.
6. **(Optional)** Add user ratings and apply collaborative filtering.

### Use case: 1. 🎬 ****Movie Recommendation System****

* Recommend similar movies to users based on genres, actors, or ratings.
* Use cosine similarity and content-based filtering.

**Inventory Forecasting:** Walmart predicting stock levels

**Customer Sentiment Analysis:** Analyzing reviews using NLP

## 4. Media & Entertainment

**Content Recommendation:** Netflix and Spotify

**Automatic Captioning:** YouTube auto-captions

**Deepfake Creation/Detection:** AI-generated faces or voices

## 5. Automotive & Transportation

**Autonomous Driving:** Tesla Autopilot

**Traffic Prediction:** Google Maps ETA

**Fleet Management:** Swiggy/Zomato/Logistics AI

## 6. Agriculture

**Crop Disease Detection:** Detect leaf blight using CNN

**Yield Prediction:** AI-based irrigation planning

**Precision Farming:** Drone-based crop monitoring

## 7. Education

**Personalized Learning Paths:** Duolingo, Khan Academy AI

**Automated Grading:** AI that evaluates student writing

**Virtual Teaching Assistants:** ChatGPT in LMS

## 8. Smart Home / IoT

**Voice Assistants:** Alexa, Google Assistant

**Energy Usage Optimization:** Smart thermostats

**Home Security:** AI surveillance systems

## 9. Scientific Research

**Drug Discovery:** AI-assisted pharma R&D

**Climate Modeling:** Predicting temperature rise

**Astronomy:** Finding exoplanets

## 10. Natural Language Processing (NLP)

**Text Classification:** Spam detection, sentiment analysis

**Machine Translation:** Google Translate

**Text Summarization:** Summarize patient history, news

**Chatbots:** ChatGPT, customer support

**Machine Learning Algorithms Types:**

### 1. ****Supervised Learning**** – Learning with labels (answers)

The algorithm learns from labeled data (i.e., you already know the correct answers).

#### 🧠 How it works:

* You give input data **and** the correct output.
* The model **learns the pattern** and makes predictions on new data.

#### 📌 Examples:

* Predict house prices
* Spam or not spam email detection
* Classify images of cats vs dogs

#### 🔧 Common Algorithms:

* **Linear Regression** – Predict numbers (e.g., price, age)
* **Logistic Regression** – Predict categories (yes/no)
* **Decision Trees**
* **Random Forest**
* **Support Vector Machines (SVM)**
* **K-Nearest Neighbors (KNN)**

### 2. ****Unsupervised Learning**** – Learning without labels

The algorithm explores data and finds patterns **without knowing the output**.

#### 🧠 How it works:

* You give only input data.
* The model groups or organizes the data on its own.

#### 📌 Examples:

* Customer segmentation
* Market basket analysis
* Grouping similar movies or songs

## 1. **Customer Segmentation**

Grouping customers based on similar characteristics or behavior.

Imagine you run a **clothing store**. You notice:

* Some customers buy only kids’ clothes.
* Some buy expensive formal wear.
* Some shop only during sales.

You can **group** them like:

* Segment A: Budget shoppers
* Segment B: Premium customers
* Segment C: Occasional buyers

This is **customer segmentation** – grouping customers based on:

* Age
* Gender
* Location
* Spending habits
* Purchase frequency, etc.

**2. Market Basket Analysis**

Imagine someone buys **bread** and **butter**.  
Most of the time, they also buy **jam**.

So, the rule the system learns is:

“If a customer buys bread & butter, there’s a high chance they’ll also buy jam.”

You can use this:

* To **recommend related products**
* For **store arrangement** (put items close together)
* For **combo offers**

### 📊 Used in:

* Supermarkets (Amazon, Big Bazaar)
* E-commerce sites (Flipkart, Amazon)

#### 🔧 Common Algorithms:

* **K-Means Clustering**
* **Hierarchical Clustering**
* **PCA (Principal Component Analysis)**
* **Apriori / Association Rules**

### 3. ****Reinforcement Learning**** – Learning by trial and error

The algorithm learns by interacting with an environment and **receiving rewards or penalties**.

#### 🧠 How it works:

* The model **takes actions**, gets **feedback**, and **learns the best strategy** over time.

### Real-Life Example: ****Self-driving Toy Car****

Imagine you have a toy car inside a square room. The car can move **left, right, forward, or backward**.

* If it bumps into a wall → ❌ Negative reward (–10)
* If it stays in the center → ⚠️ Neutral reward (0)
* If it reaches a charging station → ✅ Positive reward (+100)

Over time, the car **learns the best path** to reach the charging station while avoiding walls.

That’s **Reinforcement Learning** in action.

#### 📌 Examples:

* Self-driving cars
* Game playing (e.g., AlphaGo)
* Robot path finding

#### 🔧 Common Algorithms:

* **Q-Learning**
* **Deep Q-Networks (DQN)**
* **Policy Gradient Methods**

## Linear Regression

Linear Regression is a **supervised learning algorithm** used to **predict a numerical value** based on the relationship between variables.

Let’s say you want to **predict a student’s marks** based on how many **hours they studied**.

You collect this data:

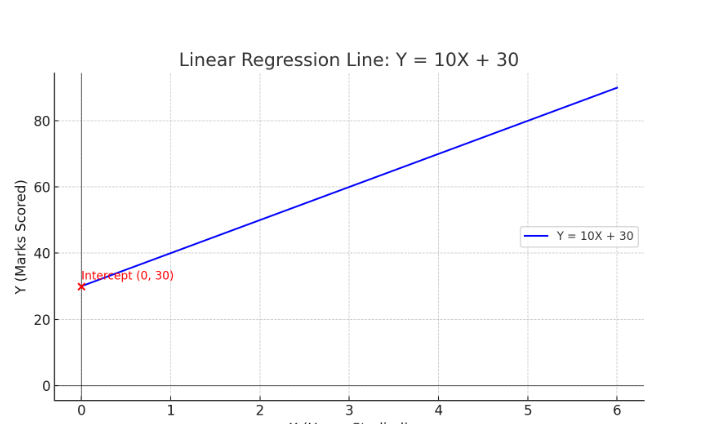
## **Mathematical Model of Linear Regression**

The formula for simple linear regression is:

Y = mX + c

### ****Table: Hours Studied vs Marks Scored****

| **Hours Studied (X)** | **Marks Scored (Y = 10X + 30)** |
| --- | --- |
| 0 | 30 |
| 1 | 40 |
| 2 | 50 |
| 3 | 60 |
| 4 | 70 |
| 5 | 80 |
| 6 | 90 |



Where:

* Y = predicted output (marks)
* X = input feature (hours studied)
* m = slope of the line (how much Y changes for each unit of X)
* c = intercept (Y value when X = 0)

Y=10\*0 +30 = 30

Y=10\*1+30=40

Y=10\*2+30=50

Y=10\*3+30=60

Y=10\*4+30=70

Y=10\*5+30=80

Y=10\*6+30=90

|  |
| --- |
| **#Linear regression**  # **Import pandas for tabular display**  import pandas as pd  **# Step 1: Define input values (X)**  X = list(range(0, 7))  # X values from 0 to 6  **# Step 2: Calculate Y values using the formula Y = 10X + 30**  Y = [10 \* x + 30 for x in X]  **# Step 3: Create a table using pandas DataFrame**  df = pd.DataFrame({      'Hours Studied (X)': X,      'Marks Scored (Y = 10X + 30)': Y  })  **# Step 4: Display the table**  print(df) |
| Hours Studied (X) Marks Scored (Y = 10X + 30)  0 0 30  1 1 40  2 2 50  3 3 60  4 4 70  5 5 80  6 6 90 |

**Use case: Student score prediction**

| **Hours\_Studied** | **Score** |
| --- | --- |
| 1 | 40 |
| 2 | 50 |
| 3 | 60 |
| 4 | 70 |
| 5 | 80 |

Use cases: **Salary Prediction,** Product Sales Forecast, Car Price Estimation, and Weight Prediction Based on Height

## Logistic Regression

Logistic Regression is used to **predict a binary outcome** (Yes/No, 0/1, Spam/Not Spam) based on one or more input features.

Unlike linear regression, which predicts **a number**, logistic regression predicts **a probability** between 0 and 1.

# Logistic Regression - Math behind Spam Detection

🎯 Objective  
We want to classify messages as:  
- Spam (1) – like "Win a free vacation now!"  
- Not Spam (0) – like "Project meeting at 10am"

✅ Step 1: Build the Vocabulary  
From the training messages, we extract all unique words (ignoring punctuation and converting to lowercase):

Training messages:  
1. "win", "a", "free", "vacation", "now"  
2. "can", "we", "meet", "at", "5pm", "today"  
3. "urgent", "your", "account", "is", "blocked"  
4. "happy", "birthday", "have", "a", "great", "day"  
5. "claim", "your", "prize", "money"  
6. "project", "meeting", "at", "10am"

Vocabulary Example (partial):  
| Index | Word |  
|--------|-----------|  
| 1 | win |  
| 2 | free |  
| 3 | vacation |  
| 4 | now |  
| 5 | urgent |  
| 6 | your |  
| 7 | account |  
| 8 | blocked |  
| 9 | claim |  
| 10 | prize |  
| ... | ... |

Each message becomes a binary vector:  
- 1 = word present  
- 0 = word absent

✅ Step 2: Vectorize the Test Message

Test Message: "Urgent: Your account is not blocked"

Words found in vocabulary:  
- urgent = 1  
- your = 1  
- account = 1  
- blocked = 1  
- not = 0 (ignored; not in training)

Vector representation (partial):  
[0, 0, 0, 0, 1, 1, 1, 1, 0, 0, ..., 0]

✅ Step 3: Logistic Regression Model Equation

Linear combination (z):  
z = w1\*x1 + w2\*x2 + ... + wn\*xn + b

Sigmoid function:  
ŷ = σ(z) = 1 / (1 + e^-z)

✅ Step 4: Use Trained Weights

| Word | Weight (w) |  
|-----------|-------------|  
| urgent | +1.5 |  
| your | +0.8 |  
| account | +1.2 |  
| blocked | +1.3 |  
| birthday | -1.5 |  
| meeting | -1.0 |  
| today | -0.9 |

Bias (b): -2.0

Calculate z:  
z = 1.5 + 0.8 + 1.2 + 1.3 - 2.0 = 2.8

Apply sigmoid:  
ŷ = 1 / (1 + e^-2.8) ≈ 0.942

✅ Step 5: Decision Rule  
- If ŷ > 0.5 → Spam (1)  
- Else → Not Spam (0)

Result: 0.942 → Spam

✅ Final Conclusion  
- The model classifies "Urgent: Your account is not blocked" as Spam.  
- Words like "urgent", "account", and "blocked" strongly influence the prediction.  
- The word "not" is ignored since it was never seen in the training data.

|  |
| --- |
| from sklearn.feature\_extraction.text import CountVectorizer  from sklearn.linear\_model import LogisticRegression  **# Step 1: Training Dataset**  messages = [      "Win a free vacation now!",      "Can we meet at 5pm today?",      "Urgent: Your account is blocked",      "Happy birthday, have a great day!",      "Claim your prize money",      "Project meeting at 10am"  ]  labels = [1, 0, 1, 0, 1, 0]  # 1 = Spam, 0 = Not Spam  **# Step 2: Convert text to numeric (features)**  vectorizer = CountVectorizer()  X = vectorizer.fit\_transform(messages)  **# Step 3: Train the Logistic Regression model**  model = LogisticRegression()  model.fit(X, labels)  # Step 4: Test on 5 new messages (mixed: spam & not spam)  test\_messages = [      "Urgent: Your account is not blocked",     # Spam-like      "Win a prize and claim it today",          # Spam-like      "Let's meet for lunch at 1pm",             # Not spam      "Your loan has been approved",             # Spam-like      "See you at the team meeting tomorrow"     # Not spam  ]  X\_test = vectorizer.transform(test\_messages)  predictions = model.predict(X\_test)  # Step 5: Output the results  print("Predictions for new messages:")  for i in range(len(test\_messages)):      label = "Spam" if predictions[i] == 1 else "Not Spam"      print(f'"{test\_messages[i]}" → {label}') |
| Predictions for new messages:  "Urgent: Your account is not blocked" → Spam  "Win a prize and claim it today" → Spam  "Let's meet for lunch at 1pm" → Not Spam  "Your loan has been approved" → Spam  "See you at the team meeting tomorrow" → Not Spam |

| **Message** | **win** | **free** | **vacation** | **now** | **...** |
| --- | --- | --- | --- | --- | --- |
| Win a free vacation now! | 1 | 1 | 1 | 1 | ... |
| Can we meet at 5pm today? | 0 | 0 | 0 | 0 | ... |
| Urgent: Your account is blocked | 0 | 0 | 0 | 0 | ... |
| Happy birthday, have a great day! | 0 | 0 | 0 | 0 | ... |
| Claim your prize money | 0 | 0 | 0 | 0 | ... |
| Project meeting at 10am | 0 | 0 | 0 | 0 | ... |

**Use case: Product categorization code**

|  |
| --- |
| **from sklearn.feature\_extraction.text import CountVectorizer**  **from sklearn.linear\_model import LogisticRegression**  **import pandas as pd**  **# Step 1: Product descriptions and their categories**  **products = [**  **"Red cotton t-shirt for men",**  **"Bluetooth wireless earphones with mic",**  **"Non-stick frying pan with lid",**  **"Leather wallet for women",**  **"Gaming laptop with 16GB RAM",**  **"Silk saree with embroidery",**  **"Electric kettle for boiling water",**  **"Office chair with lumbar support",**  **"Smartphone with 5G and dual camera",**  **"Running shoes for sports and gym"**  **]**  **categories = [**  **"Clothing",**  **"Electronics",**  **"Kitchen",**  **"Accessories",**  **"Electronics",**  **"Clothing",**  **"Kitchen",**  **"Furniture",**  **"Electronics",**  **"Clothing"**  **]**  **# Step 2: Convert text into numeric features using CountVectorizer**  **vectorizer = CountVectorizer()**  **X\_train = vectorizer.fit\_transform(products)**  **# Step 3: Train a Logistic Regression model**  **model = LogisticRegression(max\_iter=1000)**  **model.fit(X\_train, categories)**  **# Step 4: Test on new product descriptions**  **test\_products = [**  **"Wireless gaming mouse",**  **"Wooden dining table for 4 people",**  **"Stainless steel pressure cooker",**  **"Formal shirt with cufflinks",**  **"Yoga mat with carrying strap"**  **]**  **X\_test = vectorizer.transform(test\_products)**  **predicted\_categories = model.predict(X\_test)**  **# Step 5: Print results**  **print("Predicted Categories:")**  **for i in range(len(test\_products)):**  **print(f"{test\_products[i]} → {predicted\_categories[i]}")** |
|  |

## What Is Model Evaluation?

**Model evaluation** tells you **how well your trained model performs** on unseen (test) data. It helps answer:

* Is my model accurate?
* Does it work equally well for all classes?
* Should I trust its predictions?

| **Metric** | **Use Case** | **Function in sklearn** |
| --- | --- | --- |
| **Accuracy** | Overall correctness | accuracy\_score() |
| **Precision** | How many predicted positives are correct | precision\_score() |
| **Recall** | How many actual positives were found | recall\_score() |
| **F1 Score** | Balance between precision & recall | f1\_score() |
| **Confusion Matrix** | Shows TP, FP, TN, FN | confusion\_matrix() |
| **Classification Report** | All metrics class-wise | classification\_report() |

**Math Model**

**y\_true = [1, 0, 1, 1, 0]**

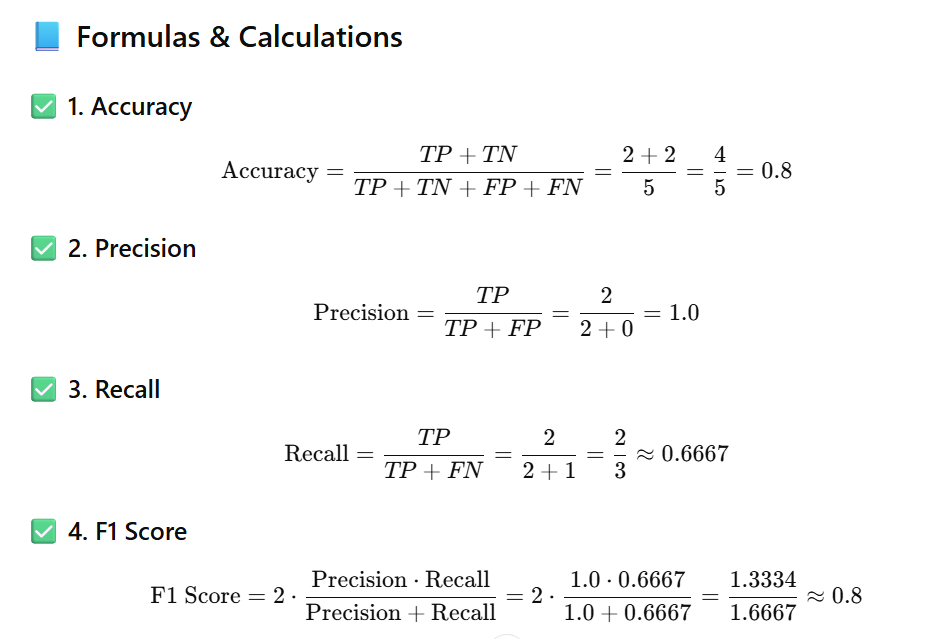
**y\_pred = [1, 0, 0, 1, 0]**

**Step-by-Step Confusion Matrix**

| **Index** | **y\_true** | **y\_pred** | **Category** |
| --- | --- | --- | --- |
| 0 | 1 | 1 | ✅ True Positive (TP) |
| 1 | 0 | 0 | ✅ True Negative (TN) |
| 2 | 1 | 0 | ❌ False Negative (FN) |
| 3 | 1 | 1 | ✅ True Positive (TP) |
| 4 | 0 | 0 | ✅ True Negative (TN) |

### Totals:

* **TP = 2**
* **TN = 2**
* **FP = 0**
* **FN = 1**

****

**Code**

|  |
| --- |
| from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix  # Given data  y\_true = [0, 0, 0, 1, 0]  y\_pred = [0, 1, 0, 1, 0]  # Calculate evaluation metrics  accuracy = accuracy\_score(y\_true, y\_pred)  precision = precision\_score(y\_true, y\_pred)  recall = recall\_score(y\_true, y\_pred)  f1 = f1\_score(y\_true, y\_pred)  conf\_matrix = confusion\_matrix(y\_true, y\_pred)  # Display results  print("Confusion Matrix:\n", conf\_matrix)  print("Accuracy:", accuracy)  print("Precision:", precision)  print("Recall:", recall)  print("F1 Score:", f1) |
| **Confusion Matrix:**  **[[2 0]**  **[1 2]]**  **Accuracy: 0.8**  **Precision: 1.0**  **Recall: 0.6666666666666666**  **F1 Score: 0.8** |

# Evaluation Metrics from Confusion Matrix

🔍 Use Case: Spam Detection

Actual = [1, 0, 1, 1, 0, 0, 1, 0, 1, 0]

Predicted = [1, 0, 1, 0, 0, 1, 1, 0, 1, 0]

Step-by-Step Comparison:

|  |  |  |  |
| --- | --- | --- | --- |
| Index | Actual | Predicted | Result |
| 0 | 1 | 1 | TP |
| 1 | 0 | 0 | TN |
| 2 | 1 | 1 | TP |
| 3 | 1 | 0 | FN |
| 4 | 0 | 0 | TN |
| 5 | 0 | 1 | FP |
| 6 | 1 | 1 | TP |
| 7 | 0 | 0 | TN |
| 8 | 1 | 1 | TP |
| 9 | 0 | 0 | TN |

📊 Confusion Matrix Counts:

TP = 4 (Indices 0, 2, 6, 8)

TN = 4 (Indices 1, 4, 7, 9)

FP = 1 (Index 5)

FN = 1 (Index 3)

📐 Calculations:

✅ Accuracy = (TP + TN) / Total = (4 + 4) / 10 = 0.8

✅ Precision = TP / (TP + FP) = 4 / (4 + 1) = 0.8

✅ Recall = TP / (TP + FN) = 4 / (4 + 1) = 0.8

✅ F1 Score = 2 \* (Precision \* Recall) / (Precision + Recall) = 0.8

🎯 Summary:

|  |  |
| --- | --- |
| TP | 4 |
| TN | 4 |
| FP | 1 |
| FN | 1 |
| Accuracy / Precision / Recall / F1 | 0.8 |

**DATA SETS**

* Loads the Iris dataset from a public CSV link using pandas.
* Splits the dataset into training and test sets.
* Trains a classifier (Logistic Regression).
* Calculates Accuracy, Precision, Recall, F1-score.
* Prints the evaluation metrics.

|  |
| --- |
| import pandas as pd  from sklearn.model\_selection import train\_test\_split  from sklearn.linear\_model import LogisticRegression  from sklearn.metrics import classification\_report  df = pd.read\_csv("https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv")  X, y = df.drop("species", axis=1), df["species"]  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  model = LogisticRegression(max\_iter=200).fit(X\_train, y\_train)  y\_pred = model.predict(X\_test)  print(classification\_report(y\_test, y\_pred)) |
| precision recall f1-score support  setosa 1.00 1.00 1.00 10  versicolor 1.00 1.00 1.00 9  virginica 1.00 1.00 1.00 11  accuracy 1.00 30  macro avg 1.00 1.00 1.00 30  weighted avg 1.00 1.00 1.00 30 |

**Pandas – Data frames**

|  |
| --- |
| **import pandas as pd # Step 1: Import pandas**  **# Step 2: Create a simple dictionary (like a table)**  **data = {**  **"Name": ["Alice", "Bob", "Charlie"],**  **"Age": [25, 30, 22],**  **"City": ["New York", "London", "Paris"]**  **}**  **# Step 3: Convert dictionary into a DataFrame**  **df = pd.DataFrame(data)**  **# Step 4: Display the DataFrame**  **print("📋 Original DataFrame:")**  **print(df)**  **# Step 5: Access a column**  **print("\n🔍 Access 'Name' column:")**  **print(df["Name"])**  **# Step 6: Add a new column**  **df["Score"] = [85, 90, 88]**  **print("\n➕ DataFrame after adding 'Score':")**  **print(df)**  **# Step 7: Filter rows (Age > 23)**  **print("\n🔎 Filter rows where Age > 23:")**  **print(df[df["Age"] > 23])**  **# Step 8: Show basic statistics**  **print("\n📊 Statistics of numerical columns:")**  **print(df.describe())** |
|  |

**Data Cleaning Operations**

|  |
| --- |
| import pandas as pd  # Step 1: Create messy data  data = {      "Name": ["Alice", "Bob", None, "David", "Eve"],      "Age": [25, 30, 22, None, 29],      "City": ["New York", "London", "Paris", "London", None],      "Score": [85, 90, None, 88, 95]  }  # Step 2: Create the DataFrame  df = pd.DataFrame(data)  print("📋 Original DataFrame:\n", df)  # Step 3: Check for missing values  print("\n🔍 Missing Values Check:\n", df.isnull())  # Step 4: Drop rows with missing values (optional preview)  df\_cleaned = df.dropna()  print("\n🧹 DataFrame after dropping rows with missing values:\n", df\_cleaned)  # Step 5: Fill missing values (safe syntax, no inplace=True)  df["Age"] = df["Age"].fillna(df["Age"].mean())  df["Score"] = df["Score"].fillna(df["Score"].mean())  df["Name"] = df["Name"].fillna("Unknown")  df["City"] = df["City"].fillna("Unknown")  print("\n🛠️ DataFrame after filling missing values:\n", df)  # Step 6: Remove duplicate rows  df = df.drop\_duplicates()  print("\n❎ DataFrame after removing duplicates:\n", df)  # Step 7: Convert data types  df["Age"] = df["Age"].astype(int)  print("\n🔄 DataFrame after converting 'Age' to integer:\n", df) |
| 📋 Original DataFrame:  Name Age City Score  0 Alice 25.0 New York 85.0  1 Bob 30.0 London 90.0  2 None 22.0 Paris NaN  3 David NaN London 88.0  4 Eve 29.0 None 95.0  🔍 Missing Values Check:  Name Age City Score  0 False False False False  1 False False False False  2 True False False True  3 False True False False  4 False False True False  🧹 DataFrame after dropping rows with missing values:  Name Age City Score  0 Alice 25.0 New York 85.0  1 Bob 30.0 London 90.0  🛠️ DataFrame after filling missing values:  Name Age City Score  0 Alice 25.0 New York 85.0  1 Bob 30.0 London 90.0  2 Unknown 22.0 Paris 89.5  3 David 26.5 London 88.0  4 Eve 29.0 Unknown 95.0  ❎ DataFrame after removing duplicates:  Name Age City Score  0 Alice 25.0 New York 85.0  1 Bob 30.0 London 90.0  2 Unknown 22.0 Paris 89.5  3 David 26.5 London 88.0  4 Eve 29.0 Unknown 95.0  🔄 DataFrame after converting 'Age' to integer:  Name Age City Score  0 Alice 25 New York 85.0  1 Bob 30 London 90.0  2 Unknown 22 Paris 89.5  3 David 26 London 88.0  4 Eve 29 Unknown 95.0 |

## What is NLP?

**Natural Language Processing (NLP)** is a field of Artificial Intelligence that enables computers to understand, interpret, generate, and respond to **human language** (text or speech).

It bridges the gap between **human communication** and **machine understanding**.

## NLP Applications

| **Domain** | **Application Example** |
| --- | --- |
| 📨 Text Classification | Spam detection, sentiment analysis |
| 💬 Chatbots | Customer support bots, virtual assistants |
| 🔍 Information Retrieval | Google Search, question answering |
| 🏥 Healthcare | Clinical note summarization, report analysis |
| 📝 Text Generation | Story or article writing, auto-summarization |
| 🌍 Translation | Google Translate, real-time translation |
| 📊 Text Mining | News analytics, trend detection |
| 📢 Speech & Audio | Speech-to-text (ASR), voice assistants |

**NLP Pipeline (Step-by-Step)**

### ****Text Preprocessing****

1. Tokenization, Lowercasing
2. Removing punctuation
3. Tokenization
4. Removing stopwords
5. Lemmatization

|  |
| --- |
| **#1. Step 1: Lowercasing**  text = "Claim your FREE prize!!! This is not a scam. Just click the link below..."  text = text.lower()  print("Lowercased Text:\n", text) |
| Lowercased Text:  claim your free prize!!! this is not a scam. just click the link below... |
| **#Step 2: Remove Punctuation**  import string  text\_no\_punct = text.translate(str.maketrans("", "", string.punctuation))  print("Without Punctuation:\n", text\_no\_punct) |
| Without Punctuation:  claim your free prize this is not a scam just click the link below |
| **pip install nltk** |
| Requirement already satisfied: nltk in /usr/local/lib/python3.11/dist-packages (3.9.1)  Requirement already satisfied: click in /usr/local/lib/python3.11/dist-packages (from nltk) (8.2.1)  Requirement already satisfied: joblib in /usr/local/lib/python3.11/dist-packages (from nltk) (1.5.1)  Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.11/dist-packages (from nltk) (2024.11.6)  Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from nltk) (4.67.1) |
| **import nltk**  **nltk.download('punkt')        # For tokenization**  **nltk.download('stopwords')    # For stopwords removal**  **nltk.download('wordnet')      # For lemmatization** |
| [nltk\_data] Downloading package punkt to /root/nltk\_data...  [nltk\_data] Package punkt is already up-to-date!  [nltk\_data] Downloading package stopwords to /root/nltk\_data...  [nltk\_data] Package stopwords is already up-to-date!  [nltk\_data] Downloading package wordnet to /root/nltk\_data...  True |

**Spacy**

|  |
| --- |
| !pip install spacy |
| !python -m spacy download en\_core\_web\_sm  ` |
| import spacy  # Load the English NLP model  nlp = spacy.load("en\_core\_web\_sm")  # Original text  text = "Claim your FREE prize!!! This is not a scam. Just click the link below..."  print(text) |
| Claim your FREE prize!!! This is not a scam. Just click the link below... |

|  |
| --- |
| # Lowercasing the entire text  lower\_text = text.lower()  print("1. Lowercased Text:\n", lower\_text) |
| 1. Lowercased Text:  claim your free prize!!! this is not a scam. just click the link below... |

|  |
| --- |
| # spaCy automatically handles punctuation during tokenization, but we can filter it  doc = nlp(lower\_text)  no\_punct\_tokens = [token.text for token in doc if not token.is\_punct]  print("2. Text without Punctuation:\n", no\_punct\_tokens) |
| 2. Text without Punctuation:  ['claim', 'your', 'free', 'prize', 'this', 'is', 'not', 'a', 'scam', 'just', 'click', 'the', 'link', 'below'] |

|  |
| --- |
| # Tokens from spaCy (includes punctuation and stopwords)  all\_tokens = [token.text for token in doc]  print("3. Tokenized Words:\n", all\_tokens) |
| . Tokenized Words:  ['claim', 'your', 'free', 'prize', '!', '!', '!', 'this', 'is', 'not', 'a', 'scam', '.', 'just', 'click', 'the', 'link', 'below', '...'] |

|  |
| --- |
| # Remove stopwords (e.g., 'your', 'is', 'just')  no\_stopwords = [token.text for token in doc if not token.is\_stop and not token.is\_punct]  print("4. Tokens without Stopwords:\n", no\_stopwords) |
| 4. Tokens without Stopwords:  ['claim', 'free', 'prize', 'scam', 'click', 'link'] |

|  |
| --- |
| # Lemmatize remaining tokens  lemmas = [token.lemma\_ for token in doc if not token.is\_stop and not token.is\_punct]  print("5. Lemmatized Tokens:\n", lemmas) |
| 5. Lemmatized Tokens:  ['claim', 'free', 'prize', 'scam', 'click', 'link'] |

|  |
| --- |
| final\_tokens = [token.lemma\_.lower() for token in doc if not token.is\_stop and not token.is\_punct]  print("✅ Final Preprocessed Output:\n", final\_tokens) |
| Final Preprocessed Output:  ['claim', 'free', 'prize', 'scam', 'click', 'link'] |

|  |
| --- |
| !pip install scikit-learn |
| Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (1.6.1)  Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (2.0.2)  Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.15.3)  Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.5.1)  Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.6.0) |

Project

|  |
| --- |
| from sklearn.feature\_extraction.text import CountVectorizer  from sklearn.linear\_model import LogisticRegression  # Sample labeled data  texts = [      "I love this product",       # Positive      "This is a great movie",     # Positive      "Fantastic experience",      # Positive      "Very happy with result",    # Positive      "Worst service ever",        # Negative      "I hate this app",           # Negative      "Very bad quality",          # Negative      "Not satisfied at all"       # Negative  ]  labels = [1, 1, 1, 1, 0, 0, 0, 0]  # 1 = Positive, 0 = Negative  # Vectorize and train  vectorizer = CountVectorizer()  X = vectorizer.fit\_transform(texts)  model = LogisticRegression()  model.fit(X, labels)  # Test on new data  test\_texts = ["Excellent support", "Worst product", "Happy with the service", "Not good at all"]  X\_test = vectorizer.transform(test\_texts)  preds = model.predict(X\_test)  # Output  for text, pred in zip(test\_texts, preds):      print(f'"{text}" → {"Positive 😊" if pred == 1 else "Negative 😡"}') |
| "Excellent support" → Negative 😡  "Worst product" → Negative 😡  "Happy with the service" → Positive 😊  "Not good at all" → Negative 😡 |

|  |
| --- |
| #Reule based chatbots  def chatbot():      print("🤖 Chatbot: Hi! I'm ChatGPT-bot. Type 'bye' to exit.")      while True:          user = input("You: ").lower()          if user in ['bye', 'exit', 'quit']:              print("🤖 Chatbot: Goodbye! 👋")              break          elif "hi" in user or "hello" in user:              print("🤖 Chatbot: Hello! How can I help you?")          elif "your name" in user:              print("🤖 Chatbot: I'm ChatGPT-bot!")          elif "how are you" in user:              print("🤖 Chatbot: I'm fine, thanks! What about you?")          elif "help" in user:              print("🤖 Chatbot: Sure! Ask me anything.")          else:              print("🤖 Chatbot: I'm not trained for that yet 😅")  chatbot() |
| Chatbot: Hi! I'm ChatGPT-bot. Type 'bye' to exit.  🤖 Chatbot: Hello! How can I help you?  🤖 Chatbot: Sure! Ask me anything.  🤖 Chatbot: I'm not trained for that yet 😅  🤖 Chatbot: I'm not trained for that yet 😅  🤖 Chatbot: I'm not trained for that yet 😅  🤖 Chatbot: I'm not trained for that yet 😅  🤖 Chatbot: I'm not trained for that yet 😅  🤖 Chatbot: Hello! How can I help you?  🤖 Chatbot: I'm not trained for that yet 😅 |

|  |
| --- |
| #Logistics chatbots  from sklearn.feature\_extraction.text import TfidfVectorizer  from sklearn.linear\_model import LogisticRegression  # Data  intents = {      "greeting": ["hi", "hello", "hey", "good morning"],      "goodbye": ["bye", "see you", "good night"],      "thanks": ["thanks", "thank you"],      "name": ["what's your name", "who are you"],      "feel": ["how are you", "how's it going"]  }  responses = {      "greeting": "Hello! 👋", "goodbye": "Goodbye! 👋", "thanks": "You're welcome! 😊",      "name": "I'm a simple chatbot 🤖", "feel": "I'm doing great! Thanks for asking."  }  # Training  X, y = sum([[q for q in v] for v in intents.values()], []), sum([[k]\*len(v) for k,v in intents.items()], [])  vec = TfidfVectorizer()  model = LogisticRegression().fit(vec.fit\_transform(X), y)  # Chat  print("🤖 Chatbot: Hi! Type 'bye' to end.")  while True:      msg = input("You: ")      if msg.lower() in ['bye', 'exit', 'quit']:          print("🤖 Chatbot: Bye! 👋"); break      pred = model.predict(vec.transform([msg]))[0]      print("🤖 Chatbot:", responses.get(pred, "Sorry, I don't understand 😅")) |
|  |