MACHINE LEARNING PROJECT

General Objective

The objective of this project is to design **Machine Learning models**, then **expose them through a backend API**, in order to simulate a **real production integration**.

The frontend is **optional** and only serves to showcase the project.

Target Algorithms

1. Gradient Descent
2. Multiclass Logistic Regression
3. Decision Tree

Each algorithm must be:

- trained
- evaluated
- **consumed via a backend API**

Expected General Architecture

- **ML Backend (mandatory)**
  - Model loading
  - Prediction endpoint
  - JSON format (input / output)
- **Frontend (optional)**
  - Simple testing interface
  - Form or text field
  - REST API call

PART 1 : GRADIENT DESCENT

Topic: Dynamic Pricing Model Optimization

Problem Statement

Predict the **optimal price of a service** based on multiple factors, then expose this prediction through an API.

Required Work

- Implement regression with Gradient Descent (from scratch)
- Compare Batch / Stochastic / Mini-Batch
- Train the model
- Save the trained model
- Create a **backend API** that allows:
  - sending input parameters
  - receiving the predicted price as output

Expected API

- Endpoint: `/predict-price`
- Method: `POST`
- Input: numerical variables
- Output: predicted price + score

Expected Analysis

- Convergence
- Impact of the learning rate
- Prediction stability

PART 2 : MULTICLASS LOGISTIC REGRESSION

Topic: Intelligent Customer Ticket Classification

Problem Statement

Automatically classify customer tickets using an ML API.

Required Work

- Text preprocessing
- Vectorization
- Implement:
  - One-vs-Rest
  - Softmax Multiclass
- Train and save the model
- Create a **backend API** that allows:
  - sending a ticket text
  - receiving the predicted class + probabilities

Expected API

- Endpoint: `/predict-ticket`
- Method: `POST`
- Input: ticket text
- Output: predicted class + class probabilities

Expected Analysis

- Overall performance
- Error analysis

- Comparison of both approaches

PART 3 : DECISION TREE

Topic: Credit Risk Scoring

Problem Statement

Evaluate a client's **risk level** using a decision-based model exposed via an API.

Required Work

- Implement a Decision Tree Classifier
- Compare Gini vs Entropy
- Limit overfitting
- Train and save the model
- Create a **backend API** that allows:
  - sending a client profile
  - receiving the risk level

Expected API

- Endpoint: `/predict-risk`
- Method: `POST`
- Input: client features
- Output: risk class + simple explanation

Expected Analysis

- Rule interpretation
- Feature importance
- Business analysis

Frontend (OPTIONAL)

The frontend can be built using:

- jinja
- React
- Streamlit

Possible features:

- Input form
- Prediction display
- Simple result visualization

No penalty if the frontend is not implemented

Technical Constraints

- Language: **Python**
- Backend: **Flask**
- Communication: **REST API (JSON)**
- ML libraries:
  - numpy
  - pandas
  - scikit-learn
- Clean, structured, and well-commented code

Deliverables

- Backend code (functional API)
- Trained and saved models
- Training scripts
- Frontend (optional)

Grading System

Critère	Points
Modèles ML (qualité & logique)	35
API backend fonctionnelle	25
Analyse & interprétation	25
Qualité du code & structure	15
Total	**100**