Title: AutoJudge - AI-Powered Difficulty Prediction System

Name: Sumit Solanki **Enrollment no:** 22323039

1. Problem Statement

Programming platforms often categorize problems subjectively. This creates inconsistency where one person's "Easy" is another's "Medium." The goal of this project is to automate this process using Natural Language Processing (NLP) and Machine Learning to predict difficulty based solely on the problem text and constraints.

2. Dataset Description

The dataset was constructed using a hybrid approach:

- **Source:** Real-world programming problems were scraped from **Project Euler** using a custom Python script (scraper.py).
- **Synthetic Data:** Additional problem statements were generated programmatically to balance the classes.
- **Structure:** Each entry contains the Problem Title, Description, Input Constraints, Output Description, and a ground-truth Difficulty Label.
- **Preprocessing:** The text data was cleaned (removing HTML tags, special characters) and vectorised.

3. Methodology & Feature Engineering

- **Text Representation:** We used **TF-IDF (Term Frequency-Inverse Document Frequency)** to convert raw text into numerical vectors. This highlights unique keywords relevant to difficulty (e.g., "optimize", "tree", "array").
- **Model Architecture:** A dual-model approach was implemented:
    1. **Classifier:** A **Random Forest Classifier** to predict categorical labels (Easy/Medium/Hard).
    2. **Regressor:** A **Random Forest Regressor** to predict a continuous difficulty score (0-100).
- **Why Random Forest?** It handles high-dimensional text data well and is robust against overfitting compared to simple decision trees.

4. Experimental Results

The models were trained on an 80/20 train-test split.

- **Classification Accuracy:** 78
- **Regression Error (MAE):** 7.2

- *Confusion Matrix Analysis:* The model performs best on "Easy" and "Hard" extremes, with minor overlap in the "Medium" category due to subjective labeling in the training data.

5. Web Interface

The user interface was built using **Streamlit** for its speed and interactivity.

- **Input:** Users paste the Problem Description, Input Constraints, and Output.
- **Processing:** The app loads the pre-trained .pkl models and the TF-IDF vectorizer.
- **Output:** It displays the predicted Class (colored Red/Green/Orange) and a dynamic progress bar for the Difficulty Score.

6. Conclusion

AutoJudge successfully demonstrates that NLP can be used to objectively quantify coding problem difficulty. Future work includes implementing Deep Learning (BERT) models to better capture context and semantic meaning.