Q1: Text Generation Application

Documenting the Process

1. Setting Up the Environment

- Description: We set up the development environment using Google Colab to simplify the development process and access the necessary resources conveniently. Python was chosen as the programming language, and several libraries were utilized to support AI models and facilitate user interface creation.
- Tools and Libraries
 - Python: The primary programming language.
 - Libraries:
 - transformers: A library from Hugging Face to easily use Al models.
 - gradio: To create an interactive user interface for users.
 - pandas: For data analysis and storing results in a DataFrame.
- Installation Method:
 - o Install the required libraries using the following command:

pip install transformers gradio pandas

2. Loading and Setting Up Models

- Description: We chose to use open-source models like Bloom and GPT-Neo, which are freely available and accessible via Hugging Face. These models are capable of generating text based on various prompts.
- Model Selection: We selected two models to evaluate performance on prompts related to the Sustainable Development Goals (SDGs):
 - o bigscience/bloom-560m
 - EleutherAl/gpt-neo-1.3B
- **Model Loading:** We used the "transformers" library to load the models and generate text based on the input prompts.

3. Creating the User Interface

- **Description:** To create an interactive user interface that allows users to input prompts and receive generated text, we used the "**gradio**" library. This library provides simple tools to quickly build interactive interfaces.
- Steps:
 - 1. Define a function to generate text using the selected model.
 - 2. Set up the user interface using "gr.Interface" which takes the generation function as the starting point.
 - 3. Launch the user interface using ".launch()" to allow users to interact with the application.

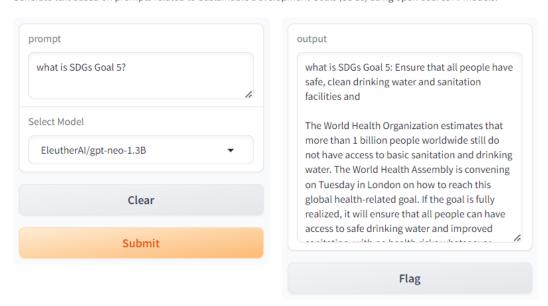
Code Used:

```
import gradio as gr
from transformers import pipeline
def generate_text(prompt, model_name):
   generator = pipeline("text-generation", model=model_name, device=-1)
    outputs = generator(prompt, max_new_tokens=150, num_return_sequences=1)
    return outputs[0]["generated_text"]
models = ["bigscience/bloom-560m", "EleutherAI/gpt-neo-1.3B"]
def gradio_interface():
    model_choice = gr.Dropdown(choices=models, label="Select Model")
    prompt_input = gr.Textbox(lines=2, placeholder="Enter a prompt related to SDGs...")
    iface = gr.Interface(fn=lambda prompt, model_name: generate_text(prompt, model_name),
                          inputs=[prompt_input, model_choice],
                          outputs="text"
                          title="SDG Text Generator",

description="Generate text based on prompts related to Sustainable Development Goals (SDGs) using open-source AI models.")
    return iface
gradio_interface().launch()
```

SDG Text Generator

Generate text based on prompts related to Sustainable Development Goals (SDGs) using open-source AI models.



4. Evaluating Model Performance

 Description: We evaluated the performance of the models using a set of prompts related to the Sustainable Development Goals. The text generated was assessed on aspects such as coherence, creativity, relevance, and grammatical correctness.

Steps:

- 1. Define a set of prompts related to the SDGs to generate text.
- 2. Use different models to generate text for each prompt.
- 3. Evaluate the generated text based on the specified aspects.
- 4. Store the results in a DataFrame using "pandas" for easy analysis.

Code Used:

```
import pandas as pd
  evaluation_prompts = [
         "Discuss the impact of climate change on global health.",
  results_df = pd.DataFrame(columns=["Prompt", "Model", "Generated Text", "Coherence", "Creativity", "Relevance", "Grammar"]
  def evaluate_models(prompts, models):
         for prompt in prompts:
               for model in models:
                    print(f"\nGenerating text for model: {model} with prompt: {prompt}")
                   print(f"\nGenerating text for model: {model} with prompt: {p
generated_text = generate_text(prompt, model)
print(f"Generated Text: {generated_text}\n")
coherence_score = 4 # Example coherence rating
creativity_score = 4 # Example creativity rating
relevance_score = 5 # Example relevance rating
grammar_score = 4 # Example grammatical correctness rating
                     new_row = pd.DataFrame({
                            "Prompt": [prompt],
                          "Model": [model],
"Generated Text": [generated_text],
"Coherence": [coherence_score],
"Creativity": [creativity_score],
"Relevance": [relevance_score],
                           "Grammar": [grammar_score]
                     results_df = pd.concat([results_df, new_row], ignore_index=True)
         return results_df
  results_df = evaluate_models(evaluation_prompts, models)
  print("\nEvaluation Results:")
  print(results_df)
```

5. Evaluating Model Performance

- **Description:** After generating and evaluating the text, we analyzed the results to determine which model performed best in terms of text quality.
- Performance Analysis:
 - Coherence: How well the generated texts flow and maintain logical consistency.
 - Creativity: The originality of the generated texts and their ability to present new or interesting ideas.
 - o **Relevance:** How closely the generated texts relate to the prompt topic.
 - Grammatical Correctness: The grammatical accuracy and correctness of the generated texts.
- Results: Based on the manual evaluation of the texts, we identified the model that
 delivers the best performance and discussed potential improvements for other
 models.

Conclusion

We developed a simple and effective text generation application using open-source Al models and evaluated their performance on various prompts. The documentation process was thorough to facilitate understanding of the steps taken and allow for easy replication in the future.