SDG 15: Life on Land, a project focusing on preserving terrestrial ecosystems, managing forests, combating desertification, and halting and reversing land degradation is relevant. Here, I'll guide you through the process of fine-tuning a language model to analyze or classify reports related to SDG 15 goals, such as evaluating documents about deforestation, biodiversity loss, or land degradation.

**Project Explanation**

**Goal:** Fine-tune a pre-trained language model to enhance its ability to classify or summarize documents related to SDG 15: Life on Land. This could involve classifying text into categories related to environmental issues or summarizing long reports on land management and conservation.

**Steps for Implementation**

**1. Install Necessary Libraries**

Ensure you have the required libraries:

pip install transformers datasets pandas matplotlib scikit-learn

**2. Exploratory Data Analysis (EDA)**

**a. Load the Dataset:**

import pandas as pd

# Load the dataset

df = pd.read\_csv('land\_reports.csv')

# Display basic information about the dataset

print(df.info())

print(df.head())

**b. Visualize the Dataset:**

If it's a classification task:

import matplotlib.pyplot as plt

# Plot class distribution if it's a classification task

if 'label' in df.columns:

df['label'].value\_counts()

plot(kind='bar')

plt.title('Class Distribution')

plt.xlabel('Class')

plt.ylabel('Count')

plt.show()

**3. Dataset Preparation**

**a. Preprocess the Dataset:**

Convert your DataFrame into a format suitable for Hugging Face’s datasets library and tokenize it.

from transformers import AutoTokenizer

from datasets import Dataset

# Convert DataFrame to Hugging Face Dataset

dataset = Dataset.from\_pandas(df)

# Load tokenizer

tokenizer = AutoTokenizer.from\_pretrained('bert-base-uncased')

# Preprocessing function

def preprocess\_function(examples): return tokenizer(examples['text\_column'], padding='max\_length', truncation=True)

# Apply preprocessing encoded\_

dataset = dataset.map(preprocess\_function, batched=True)

**4. Model Selection**

For tasks like classification, BERT or RoBERTa are good choices; for summarization, T5 or BART are effective.

**a. Load the Pre-trained Model:**

For classification:

from transformers import AutoModelForSequenceClassification

# Load the pre-trained model

num\_labels = len(df['label'].unique())

# Number of classes

model = AutoModelForSequenceClassification.from\_pretrained('bert-base-uncased', num\_labels=num\_labels)

**5. Fine-Tuning the Model**

**a. Define Training Arguments:**

from transformers import TrainingArguments

training\_args = TrainingArguments( output\_dir='./results', # Output directory

evaluation\_strategy="epoch",

# Evaluation strategy

per\_device\_train\_batch\_size=8,

# Batch size for training

per\_device\_eval\_batch\_size=8,

# Batch size for evaluation

num\_train\_epochs=3,

# Number of training epochs

weight\_decay=0.01,

# Weight decay

logging\_dir='./logs',

# Directory for logs

logging\_steps=10,

# Log every 10 steps

)

**b. Fine-Tune the Model:**

from transformers import Trainer

trainer = Trainer(

model=model,

# The pre-trained model

args=training\_args,

# Training arguments

train\_dataset=encoded\_dataset['train'],

# Training dataset

eval\_dataset=encoded\_dataset['test'],

# Evaluation dataset

)

# Train the model trainer.train()

**6. Evaluation**

**a. Evaluate Performance:**

eval\_results = trainer.evaluate()

print("Evaluation Results:", eval\_results)