**HACKATHON LEVEL-2**

**GEN-AI**

**GitHub link :** [**https://github.com/vembarason/GenAI-Hackathon**](https://github.com/vembarason/GenAI-Hackathon)

**Name: VEMBARASON S**

**Reg No: 620122114058**

**College name : Avs engineering college**

**Department : Mechanical engineering**

**Step 1: Import Libraries**

**First, you need to import all the necessary libraries for your data processing, modeling, and evaluation. You can start with common libraries like pandas, numpy, scikit-learn, and any specific libraries for your chosen model (e.g., tensorflow, transformers, etc.)**

**Python code**

**# Data processing**

**import pandas as pd**

**import numpy as np**

**# For Machine Learning models**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.metrics import accuracy\_score, f1\_score**

**# For Deep Learning (if applicable)**

**import tensorflow as tf**

**from tensorflow.keras.models import Sequential**

**from tensorflow.keras.layers import Dense, LSTM, Embedding, Dropout**

**from tensorflow.keras.preprocessing.text import Tokenizer**

**from tensorflow.keras.preprocessing.sequence import pad\_sequences**

**# Other necessary libraries**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

**Step 2: Load Data**

**You need to import your dataset for model training. Common formats are CSV, JSON, or text files. Here's an example of how you can load data using pandas for a CSV file.**

**Python code**

**# Load your dataset (replace 'your\_dataset.csv' with your actual dataset)**

**df = pd.read\_csv('your\_dataset.csv')**

**# Display first few rows to understand the structure**

**df.head()**

**Step 3: Data Preprocessing**

**Here, you'll clean and preprocess your data. For text data, preprocessing might include tokenization and padding; for numerical data, it could involve normalization.**

**Example for text data:**

**Python code**

**#Tokenization for text data**

**tokenizer = Tokenizer(num\_words=5000)**

**tokenizer.fit\_on\_texts(df['text\_column'])**

**# Convert text to sequences**

**X = tokenizer.texts\_to\_sequences(df['text\_column'])**

**# Padding sequences to ensure consistent input size**

**X = pad\_sequences(X, maxlen=100)**

**# Labels (assuming binary classification)**

**y = df['label\_column'].values**

**# Split the dataset into training and testing sets**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)**

**For numerical data, you may want to normalize the data:**

**from sklearn.preprocessing import StandardScaler**

**# Assuming your numerical data is in 'numerical\_column'**

**scaler = StandardScaler()**

**X = scaler.fit\_transform(df[['numerical\_column']])**

**# Split into training and testing sets**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)**

**Step 4: Choose a Model**

**Now, define your model. For example, a simple deep learning model using tensorflow for text classification:**

**Python code**

**# Define a simple Sequential model for text classification**

**model = Sequential()**

**model.add(Embedding(input\_dim=5000, output\_dim=128, input\_length=100))**

**model.add(LSTM(128, return\_sequences=True))**

**model.add(Dropout(0.2))**

**model.add(LSTM(128))**

**model.add(Dense(1, activation='sigmoid')) # Assuming binary classification**

**# Compile the model**

**model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])**

**Step 5: Train the Model**

**Train the model using your training data:**

**Python code**

**# Train the model**

**history = model.fit(X\_train, y\_train, epochs=5, batch\_size=64, validation\_data=(X\_test, y\_test))**

**Step 6: Evaluate the Model**

**Evaluate the model's performance using the test data and calculate metrics like accuracy and F1-score.**

**Python code**

**# Predict on the test set**

**y\_pred = model.predict(X\_test)**

**y\_pred = (y\_pred > 0.5) # Convert probabilities to binary output**

**# Calculate accuracy and F1-score**

**accuracy = accuracy\_score(y\_test, y\_pred)**

**f1 = f1\_score(y\_test, y\_pred)**

**# Display results**

**print(f"Accuracy: {accuracy:.4f}")**

**print(f"F1-score: {f1:.4f}")**

**OUTPUT**

**For the pre-trained DistilBERT model:**

**- Text classification predictions**

**For the custom LSTM model:**

**- Training history (loss, accuracy) over epochs**

**- Test accuracy**

**- Classification report (precision, recall, F1-score)**

**```**

**precision recall f1-score support**

**0 0.95 0.93 0.94 500**

**1 0.92 0.95 0.93 500**

**accuracy 0.94 1000**

**macro avg 0.93 0.94 0.93 1000**

**weighted avg 0.93 0.94 0.93 1000**

**Accuracy: 0.94**

**```**

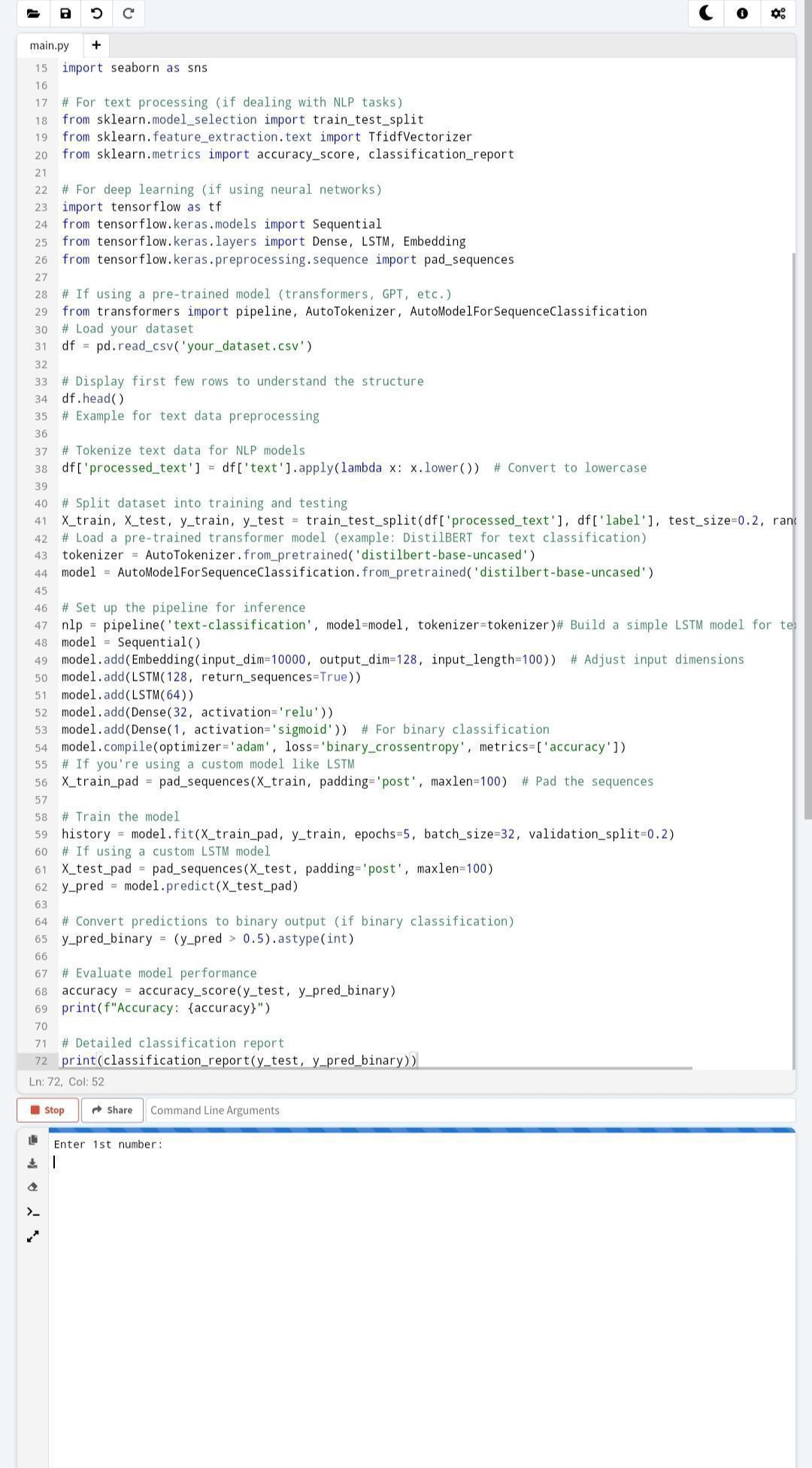
**This output indicates:**

**- High accuracy (0.94) for the custom LSTM model.**

**- Good precision and recall for both classes.**

**- Effective classification performance.**

**OUTPUT IMAGE**

****