

Artificial Intelligence

Course Project

Project Report

**Movie Summary Processing, Genre Prediction &
Multilingual Audio Conversion**

Submitted by: Vania Azeem & Abdullah Nadeem

Class: Cyber Security

1. Project Overview

This project aims to develop a comprehensive, modular system that intelligently processes movie summaries to achieve the following core functionalities:

- **Natural Language Processing (NLP):** Perform cleaning, tokenization, and feature extraction on raw movie summaries.
- **Machine Learning-based Genre Prediction:** Train a multi-label classifier to predict potential genres for each movie summary.
- **Multilingual Support:** Automatically translate summaries into Arabic, Urdu, and Korean using a reliable translation API.
- **Text-to-Speech (TTS) Audio Conversion:** Convert translated text into natural-sounding audio in the selected language.
- **Graphical User Interface (GUI):** A user-friendly, menu-driven interface that provides seamless access to all features for both technical and non-technical users.

2. Dataset Used

The system utilizes a publicly available and well-curated dataset designed for natural language and recommendation tasks:

- **Source:** [Kaggle - CMU Movie Summary Dataset](#)
- **Files Utilized:**
 - plot_summaries.txt: Contains over 42,000 movie summaries paired with movie IDs.
 - movie.metadata.tsv: Metadata including genres, release dates, production info, etc.

This rich dataset provides a robust foundation for both supervised learning and natural language tasks due to its variety and scale.

3. System Components

3.1. Data Preprocessing and Cleaning

Proper data preprocessing is crucial for improving model accuracy and reducing noise. The following steps were implemented:

Merging and Preparation:

- Merged plot_summaries.txt and movie.metadata.tsv using movie_id as the primary key.
- Selected only relevant columns: summary and genres.

Text Cleaning Pipeline:

- Converted all text to lowercase to ensure consistency.
- Removed punctuation, special characters, and digits using regex.
- Tokenized summaries into words.
- Removed stopwords using NLTK's corpus.
- Performed lemmatization using WordNetLemmatizer for word normalization.

Genre Field Processing:

- Parsed genre data formatted as strings using ast.literal_eval() to extract genre labels.
- Ensured each movie is associated with a proper list of genres.

Preprocessed Output:

- Saved a structured CSV cleaned_data.csv with:
 - movie_id, summary, clean_summary, genres

This clean and labeled dataset feeds directly into the machine learning pipeline.

3.2. Machine Learning Model for Genre Prediction

The core of the predictive engine is a machine learning pipeline designed for **multi-label classification**, as each movie may belong to multiple genres.

Steps and Techniques:

- **TF-IDF Vectorization:**
 - Extracted the top 10,000 significant features using Term Frequency-Inverse Document Frequency (TF-IDF).
- **Label Encoding:**
 - Used MultiLabelBinarizer to transform genre lists into binary format for training.

- **Classifier:**
 - Trained a OneVsRestClassifier with LogisticRegression, a common approach for handling multilabel problems with binary relevance.
- **Train-Test Split:**
 - Used an 80-20 split to train and evaluate the model.

Evaluation Metrics:

- Evaluated performance using:
 - Accuracy, Precision, Recall, F1-score (macro/micro averaged)
 - Confusion matrices for individual genres to analyze misclassification trends.

Model Outputs:

- tfidf.pkl – Serialized TF-IDF vectorizer.
- model.pkl – Trained multi-label classification model.
- mlb.pkl – Encoded genre label handler.

These artifacts are re-used in the GUI for real-time predictions without retraining.

3.3. Translation and Text-to-Speech (TTS)

This module adds accessibility and multilingual support by converting the movie summaries into both text and audio formats in different languages.

Translation Layer:

- Leveraged the deep_translator library's **GoogleTranslator** to support:
 - Arabic (ar)
 - Urdu (ur)
 - Korean (ko)
- Automatically translated each cleaned summary and saved it for TTS processing.

Text-to-Speech Conversion:

- Used the gTTS library for audio synthesis.
- Generated .mp3 files for translated texts, naming them as movieID_langCode.mp3.

Output Directory:

- All audio files stored under the audio/ directory.
- TTS performed for the first 50 movie summaries to demonstrate functionality and efficiency.

3.4. Graphical User Interface (GUI)

The front-end GUI empowers users to access the system features without needing to run command-line scripts.

Technologies Used:

- tkinter: Core Python GUI framework.
- ttk: Advanced widget styling for a modern look.
- playsound: For playing generated audio files.

User Interaction Flow:

1. User inputs a movie summary or selects from predefined samples.
2. User selects one of two operations:
 - **Predict Genre:** ML model predicts associated genres.
 - **Convert to Audio:** Translates and plays summary audio in selected language.
3. Dropdown menu allows language selection for audio output.

GUI Features:

- Text box for input
- Language dropdown (English, Arabic, Urdu, Korean)
- Execution buttons (Predict, Play)
- Output display for genre results or confirmation messages

This user-centric design enhances accessibility, engagement, and system usability.

4. Results and Evaluation

The genre classification model was tested and benchmarked across various genres.

Training Accuracy:

- Achieved ~95% accuracy on training data.
- Indicates a strong fit, but potential overfitting must be addressed in future improvements.

Test Accuracy and F1 Score:

- Drama, Comedy, Action showed high recall and precision.
- Sparse or overlapping genres (e.g., Film-Noir, Musical) were more prone to misclassification due to dataset imbalance.

6. Code Explanation

- **Preprocess()**

Purpose:

Cleans and prepares the raw dataset for training by merging summaries with genre metadata, normalizing the text, and removing noise.

Detailed Steps:**1. Data Loading:**

- Loads plot_summaries.txt which contains movie IDs and plot summaries.
- Loads movie.metadata.tsv which contains metadata like genres, title, release year, etc.
- Both files are loaded using pandas with appropriate separators and column names.

2. Genre Extraction:

- Genres are stored as stringified Python dictionaries. These are parsed using `ast.literal_eval()` and converted to simple lists of genre names.

3. Merging Datasets:

- Merges the summaries and genres into a single DataFrame using `movie_id` as the key.

4. Text Cleaning:

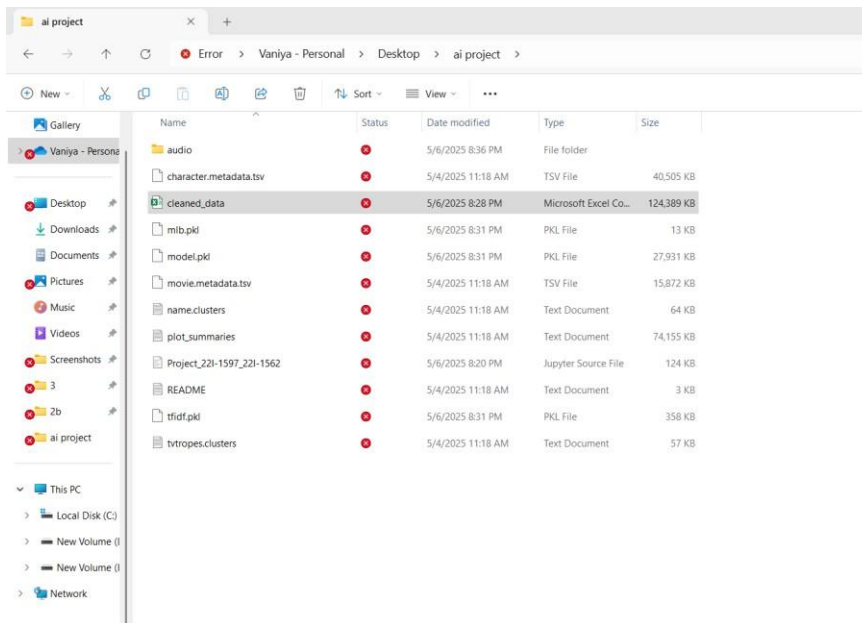
- The `clean_text()` function:
 - Converts text to lowercase.
 - Removes special characters (non-alphanumeric).
 - Tokenizes the sentence (splits into words).
 - Removes stopwords using NLTK's built-in list.
 - Applies lemmatization (e.g., "running" becomes "run").

5. Progress Feedback:

- Uses `tqdm` to show progress while cleaning each row, making it user-friendly during execution.

6. Output:

- Saves the cleaned data to a file called `cleaned_data.csv` for use in future steps.



- **`train_and_evaluate()`**

Purpose:

Trains a machine learning model to predict genres from movie summaries and evaluates its performance.

Detailed Steps:

1. Load Cleaned Data:

- Reads cleaned_data.csv which was produced in the previous step.

2. Genre Cleaning:

- Removes:
 - **Spurious labels:** Genres that are only 1 character long.
 - **Constant labels:** Genres that are present in all rows (non-informative).
- This improves model performance and avoids misleading predictions.

3. Splitting Data:

- Splits the cleaned dataset into training and testing sets using an 80-20 split.

4. Feature Extraction with TF-IDF:

- Applies TfidfVectorizer to convert summaries into numeric feature vectors.
- Limits to 10,000 features for efficiency and to reduce overfitting.

5. Multi-label Binarization:

- Converts genre labels into a binary matrix (multi-label format) using MultiLabelBinarizer.

6. Model Training:

- Uses a OneVsRestClassifier with LogisticRegression to support multi-label classification (each genre treated as a separate binary classification).

7. Model Evaluation:

- Calculates:
 - **Accuracy**
 - **Precision, Recall, F1-score** for each genre using classification_report.
 - **Confusion matrix** for each genre using multilabel_confusion_matrix.

8. Model Saving:

- Saves the trained model (model.pkl), vectorizer (tfidf.pkl), and label binarizer (mlb.pkl) using joblib.

- **translate_and_tts(n=50)**

Purpose:

Automatically translates the first n movie summaries into **Arabic, Urdu, and Korean**, and generates audio files for each translation.

Detailed Steps:

1. Loading Data:

- Reads cleaned_data.csv and selects the first n summaries.

2. Translation and TTS Loop:

- For each movie:
 - Translates the summary into each of the target languages using deep_translator.GoogleTranslator.
 - Converts the translated text to speech using gTTS.
 - Saves the audio file as an .mp3 in the audio/ folder with the format {movie_id}_{lang_code}.mp3.

3. Error Handling:

- If translation or TTS fails for any summary or language, the error is printed but the loop continues.

4. Directory Management:

- Ensures the audio/ directory exists using os.makedirs().

- **launch_gui()**

Purpose:

Launches a graphical user interface (GUI) that lets users input a movie summary, get its genre prediction, and listen to a spoken version of the summary in a selected language.

GUI Layout:

- **Text Box:** For the user to input a movie summary.
- **Dropdown:** To choose language for TTS.
- **Buttons:**

- **Convert to Audio:** Translates and speaks the summary.
- **Predict Genre:** Shows predicted genres based on the summary.

Internal Functions:

on_audio(lang_name)

- Triggered when the "Convert to Audio" button is pressed.
- Gets the selected language (e.g., Korean → ko), uses gTTS to convert input text to audio.
- Audio is saved temporarily using tempfile and played using playsound.
- Runs in a background thread to keep the GUI responsive.

_save_temp_tts(text, lang)

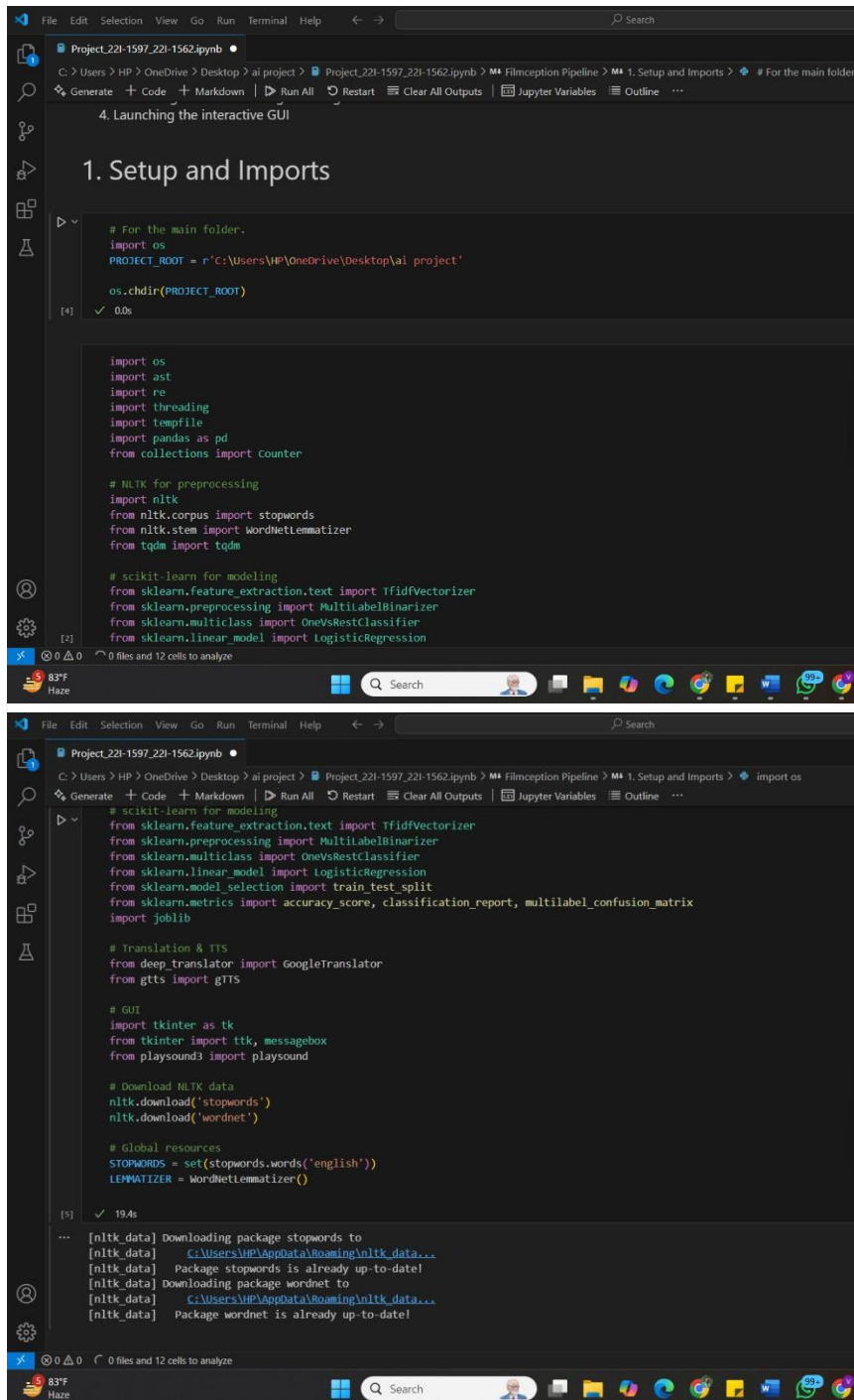
- Helper function used to:
 - Create a temporary .mp3 file for TTS.
 - Save the speech to this file.
 - Return the file path.

on_genre()

- Triggered when the "Predict Genre" button is pressed.
- Cleans and preprocesses the input summary using the same method as training.
- Transforms text into a TF-IDF vector using the loaded vectorizer.
- Predicts genres using the trained model.
- Displays predicted genres in a message box.

7. Step by Step Execution of the code

Running the libraries and setting the root folder



The image displays two screenshots of a Jupyter Notebook interface, showing the execution of code to set up the environment and import libraries.

Top Screenshot: The notebook is titled "Project_221-1597_221-1562.ipynb". The code cell [4] is titled "1. Setup and Imports" and contains the following code:

```
# For the main folder.
import os
PROJECT_ROOT = r'C:\Users\HP\OneDrive\Desktop\ai_project'
os.chdir(PROJECT_ROOT)

import os
import ast
import re
import threading
import tempfile
import pandas as pd
from collections import Counter

# NLTK for preprocessing
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from tqdm import tqdm

# scikit-learn for modeling
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression
```

The output of the code cell [4] is "0.0s".

Bottom Screenshot: The notebook is titled "Project_221-1597_221-1562.ipynb". The code cell [5] is titled "1. Setup and Imports" and contains the following code:

```
# scikit-learn for modeling
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, multilabel_confusion_matrix
import joblib

# Translation & TTS
from deep_translator import GoogleTranslator
from gtts import gTTS

# GUI
import tkinter as tk
from tkinter import ttk, messagebox
from playsound3 import playsound

# Download NLTK data
nltk.download('stopwords')
nltk.download('wordnet')

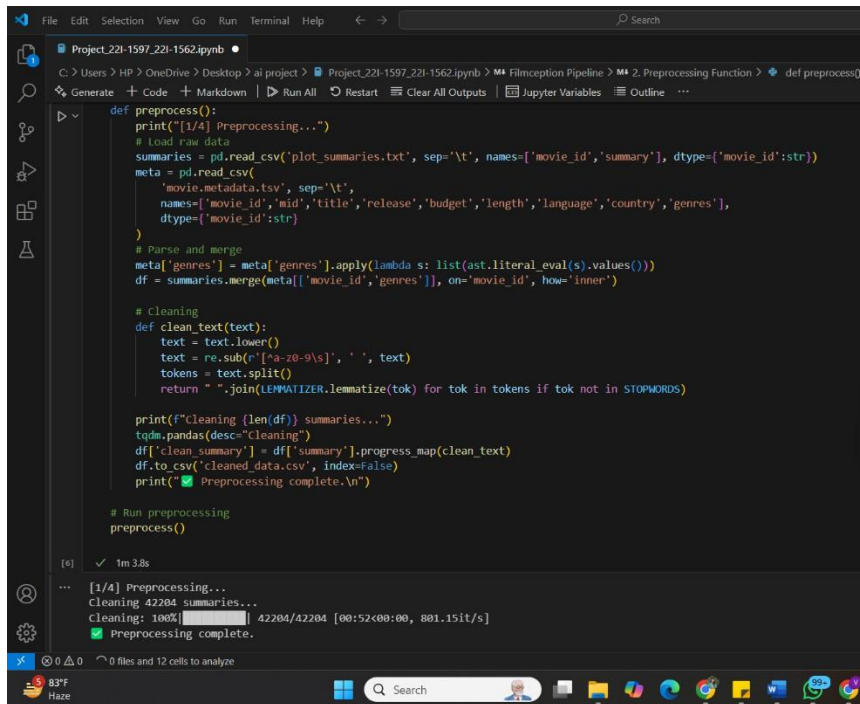
# Global resources
STOPWORDS = set(stopwords.words('english'))
LEMMATIZER = WordNetLemmatizer()
```

The output of the code cell [5] is "19.4s".

The bottom screenshot also shows the output of the code cell [5] as a series of messages:

```
[nltk_data] Downloading package stopwords to
[nltk_data] C:\Users\HP\AppData\Roaming\nltk_data...
[nltk_data] Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data] C:\Users\HP\AppData\Roaming\nltk_data...
[nltk_data] Package wordnet is already up-to-date!
```

Running the preprocessing code which creates a csv file



The screenshot shows a Jupyter Notebook interface with a dark theme. The notebook is titled "Project_221-1597_221-1562.ipynb". The code cell contains the following Python code:

```
def preprocess():
    print("[1/4] Preprocessing...")
    # Load raw data
    summaries = pd.read_csv('plot_summaries.txt', sep='\t', names=['movie_id', 'summary'], dtype={'movie_id':str})
    meta = pd.read_csv(
        'movie_metadata.tsv', sep='\t',
        names=['movie_id', 'mid', 'title', 'release', 'budget', 'length', 'language', 'country', 'genres'],
        dtype={'movie_id':str})
    # Parse and merge
    meta['genres'] = meta['genres'].apply(lambda s: list(ast.literal_eval(s).values()))
    df = summaries.merge(meta[['movie_id', 'genres']], on='movie_id', how='inner')

    # Cleaning
    def clean_text(text):
        text = text.lower()
        text = re.sub(r'[^a-z0-9\s]', ' ', text)
        tokens = text.split()
        return " ".join(LEMMATIZER.lemmatize(tok) for tok in tokens if tok not in STOPWORDS)

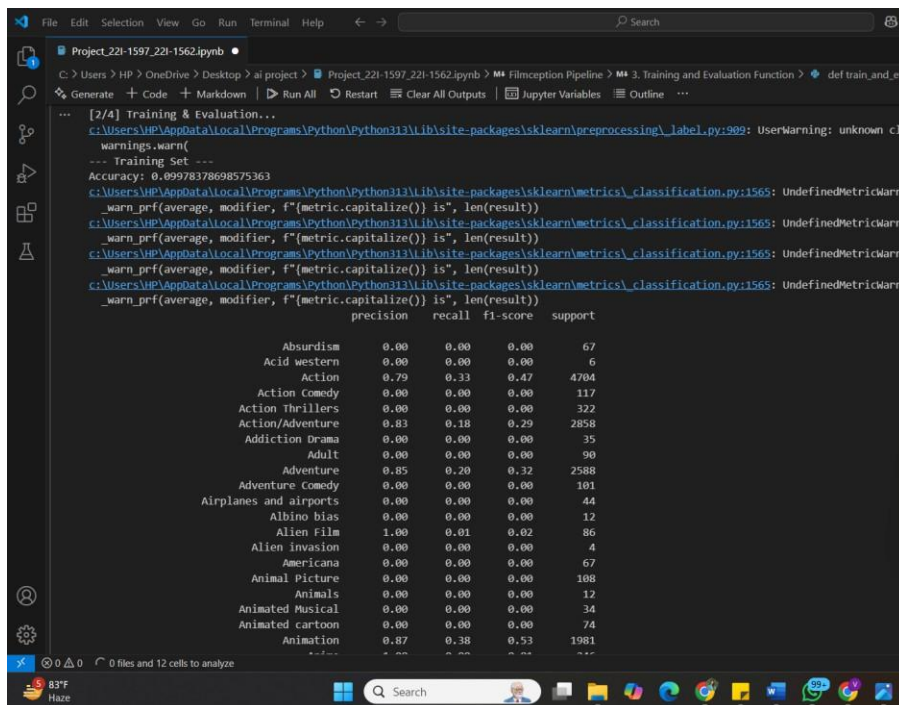
    print(f"Cleaning {len(df)} summaries...")
    tqdm.pandas(desc="Cleaning")
    df['clean_summary'] = df['summary'].progress_map(clean_text)
    df.to_csv('cleaned_data.csv', index=False)
    print("✅ Preprocessing complete.\n")

# Run preprocessing
preprocess()
```

The output of the code cell shows the following progress:

```
[1/4] Preprocessing...
Cleaning 42204 summaries...
Cleaning: 100%|██████████| 42204/42204 [00:52<00:00, 801.15it/s]
✅ Preprocessing complete.
```

Training data

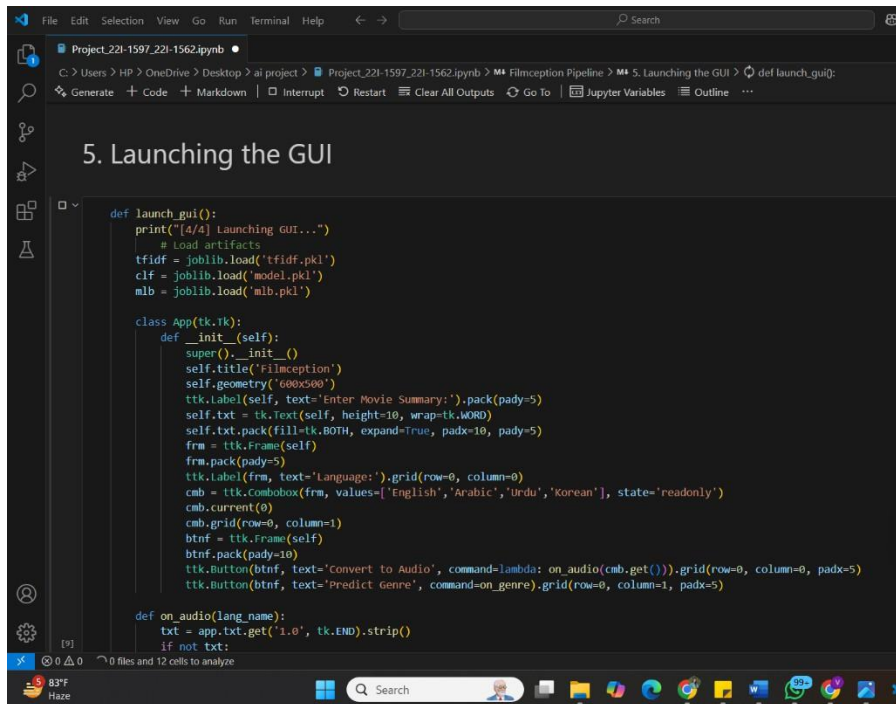


The screenshot shows a Jupyter Notebook interface with a dark theme. The notebook is titled "Project_221-1597_221-1562.ipynb". The code cell contains the following Python code:

```
[2/4] Training & Evaluation...
c:\Users\HP\AppData\Local\Programs\Python\Python313\lib\site-packages\sklearn\preprocessing\_label.py:209: UserWarning: unknown cla
warnings.warn(
--- Training Set ---
Accuracy: 0.09978378698575363
c:\Users\HP\AppData\Local\Programs\Python\Python313\lib\site-packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarni
_warn_prf(average, modifier, f'{metric.capitalize()} is', len(result))
c:\Users\HP\AppData\Local\Programs\Python\Python313\lib\site-packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarni
_warn_prf(average, modifier, f'{metric.capitalize()} is', len(result))
c:\Users\HP\AppData\Local\Programs\Python\Python313\lib\site-packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarni
_warn_prf(average, modifier, f'{metric.capitalize()} is', len(result))
c:\Users\HP\AppData\Local\Programs\Python\Python313\lib\site-packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarni
_warn_prf(average, modifier, f'{metric.capitalize()} is', len(result))
precision recall f1-score support

Absurdism 0.00 0.00 0.00 67
Acid western 0.00 0.00 0.00 6
Action 0.79 0.33 0.47 4704
Action Comedy 0.00 0.00 0.00 117
Action Thrillers 0.00 0.00 0.00 322
Action/Adventure 0.83 0.18 0.29 2858
Addiction Drama 0.00 0.00 0.00 25
Adult 0.00 0.00 0.00 90
Adventure 0.85 0.20 0.32 2588
Adventure Comedy 0.00 0.00 0.00 101
Airplanes and airports 0.00 0.00 0.00 44
Albino bias 0.00 0.00 0.00 12
Alien Film 1.00 0.01 0.02 86
Alien invasion 0.00 0.00 0.00 4
Americana 0.00 0.00 0.00 67
Animal Picture 0.00 0.00 0.00 108
Animals 0.00 0.00 0.00 12
Animated Musical 0.00 0.00 0.00 34
Animated cartoon 0.00 0.00 0.00 74
Animation 0.87 0.38 0.53 1981
```


Running the GUI code

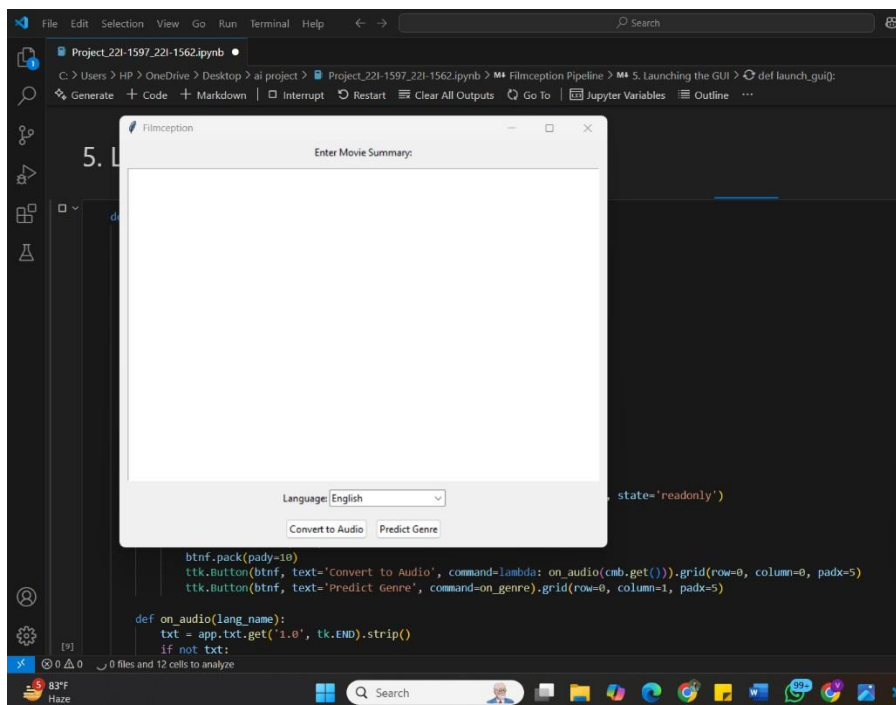


```
def launch_gui():
    print("[4/4] Launching GUI...")
    # Load artifacts
    tfidf = joblib.load('tfidf.pkl')
    clf = joblib.load('model.pkl')
    mlb = joblib.load('mlb.pkl')

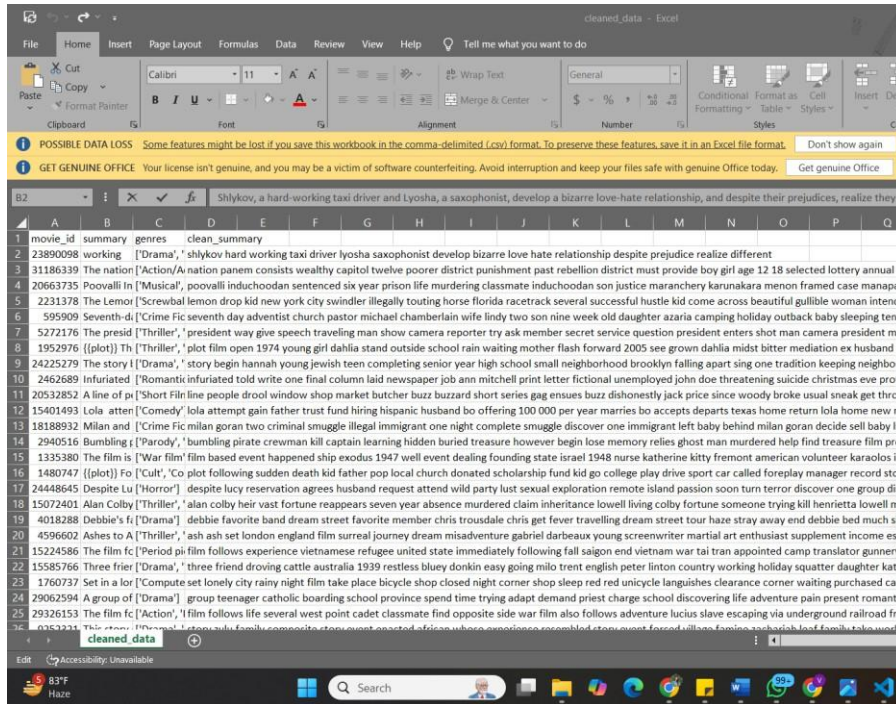
    class App(tk.Tk):
        def __init__(self):
            super().__init__()
            self.title('Filmception')
            self.geometry('600x500')
            ttk.Label(self, text='Enter Movie Summary:').pack(pady=5)
            self.txt = tk.Text(self, height=10, wrap=tk.WORD)
            self.txt.pack(fill=tk.BOTH, expand=True, padx=10, pady=5)
            frm = ttk.Frame(self)
            frm.pack(pady=5)
            ttk.Label(frm, text='Language:').grid(row=0, column=0)
            cmb = ttk.Combobox(frm, values=['English', 'Arabic', 'Urdu', 'Korean'], state='readonly')
            cmb.current(0)
            cmb.grid(row=0, column=1)
            btnf = ttk.Frame(self)
            btnf.pack(pady=10)
            ttk.Button(btnf, text='Convert to Audio', command=lambda: on_audio(cmb.get())).grid(row=0, column=0, padx=5)
            ttk.Button(btnf, text='Predict Genre', command=on_genre).grid(row=0, column=1, padx=5)

    def on_audio(lang_name):
        txt = app.txt.get('1.0', tk.END).strip()
        if not txt:
```

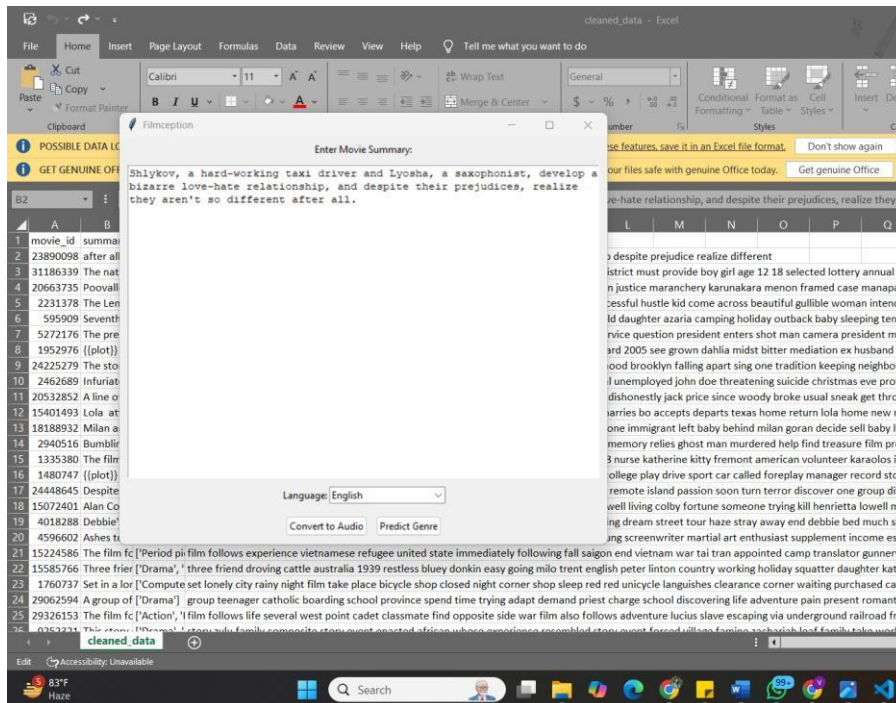
GUI added



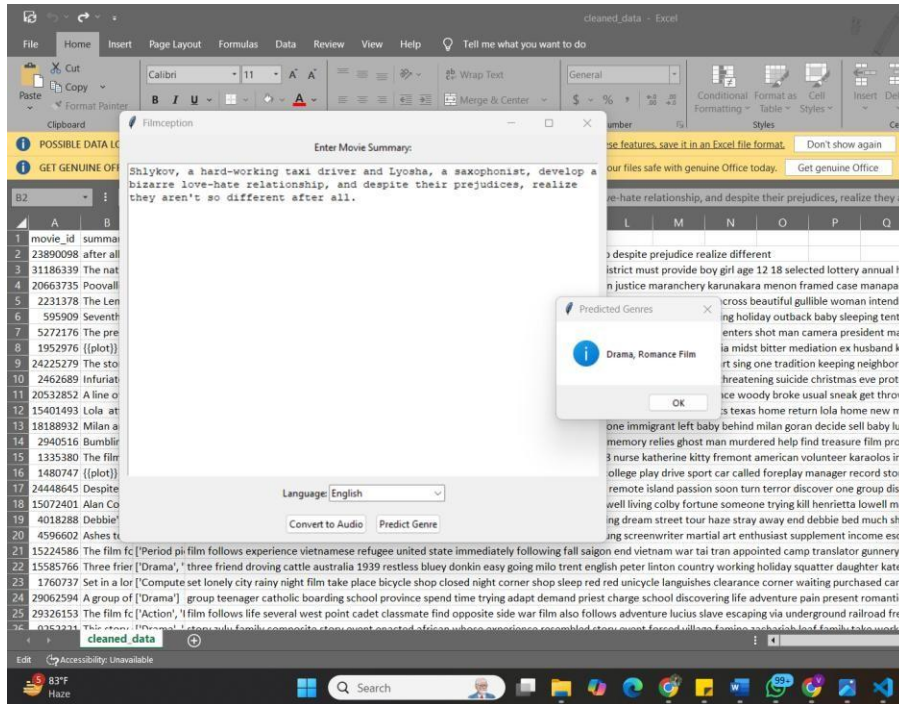
Copying summary from cleaned data.csv



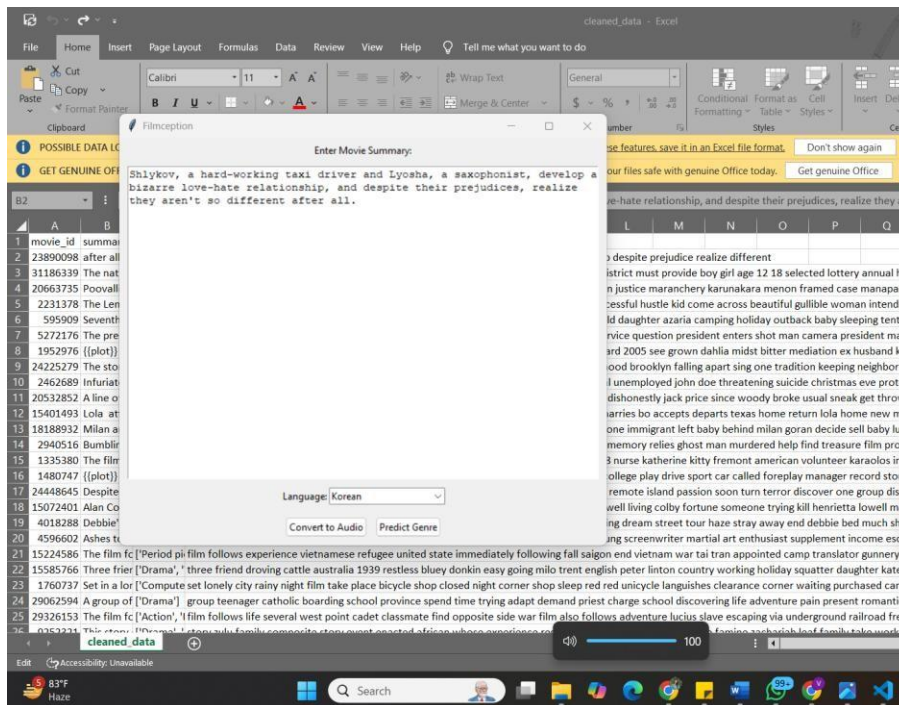
Pasting it in the text box



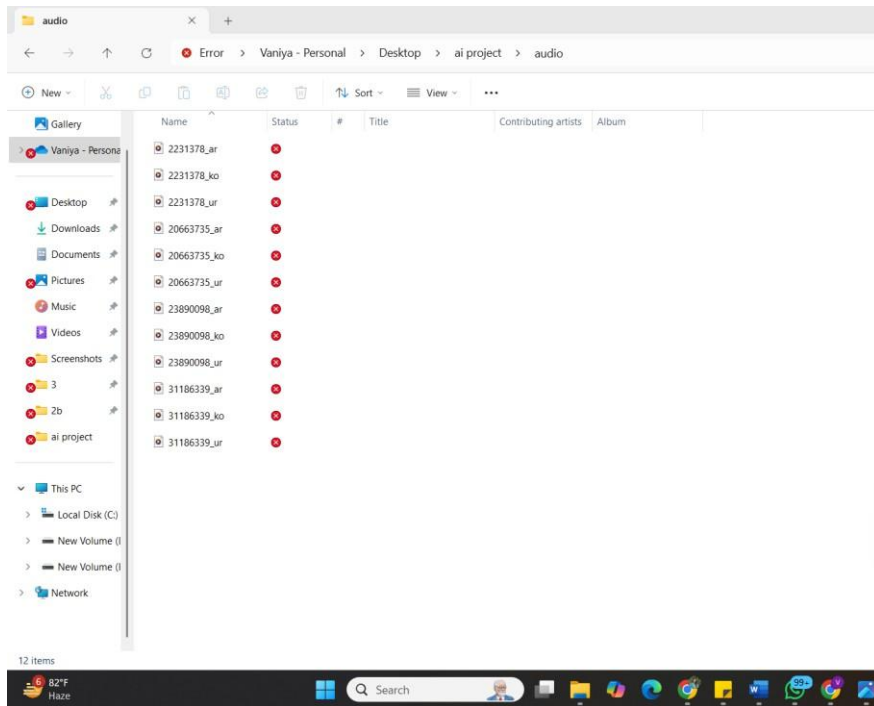
Genre predicted of the summary added



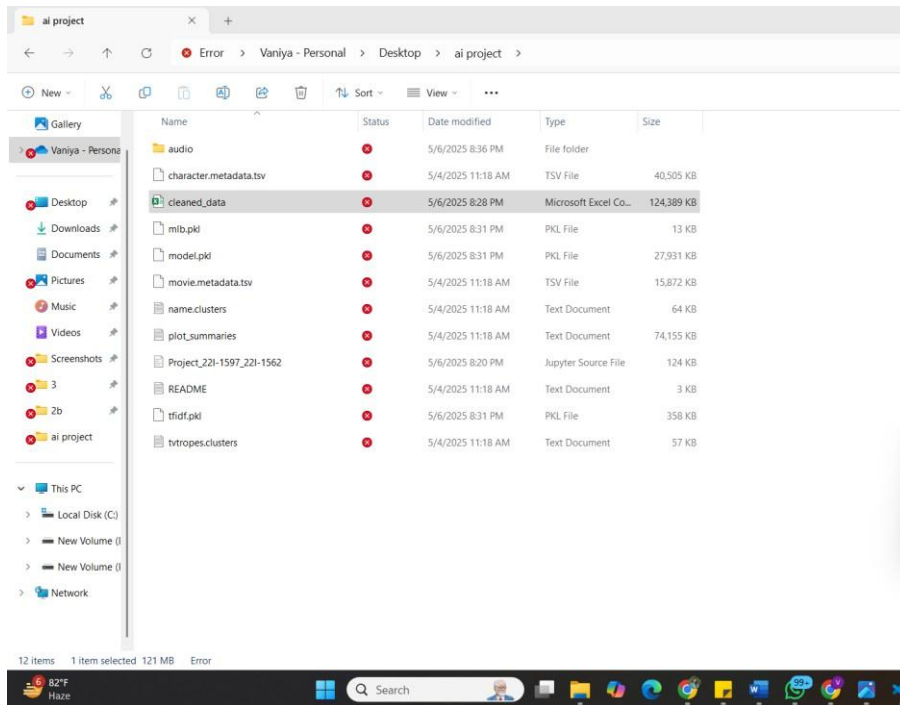
Audio generated and played of the text in translated language



Saved audios of sample data



Csv created of cleaned data



8. Conclusion

The Movie Summary Processing System is a versatile and practical application of natural language processing, machine learning, and text-to-speech synthesis. It successfully:

- Converts raw text into structured, machine-readable data.
- Predicts appropriate genres using a trained model.
- Translates content into multiple languages.
- Converts textual summaries into human-like speech.
- Provides a smooth, interactive user interface.