1. Connect to drive

**Code:**

from google.colab import drive

drive.mount('/content/drive')

1. Import datasets

**Code:**

import pandas as pd

df = pd.read\_csv('/content/drive/MyDrive/Language( Malay & English )\_dataset.csv')

print(df.head())

1. Checking details/properties of dataset

**Code:**

print(df.info())  # Check data types and missing values

print(df)

print(df.isnull().sum())  # Check for missing values

print(f"Dataset size: {len(df)} rows")

1. Data Cleaning

* **Tokenization**: Split text into words.
* **Remove stop words**: Common words that don't contribute much to meaning (like "the", "is", etc.).
* **Lowercasing**: Convert all text to lowercase.
* **Punctuation Removal**: Remove unnecessary punctuation.

**Code:**

# Handle missing values

df['query'] = df['query'].fillna('')

# Text cleaning (removing non-alphabetic characters, convert to lowercase)

df['query'] = df['query'].str.replace('[^a-zA-Z]', ' ', regex=True)

df['query'] = df['query'].str.lower()

from sklearn.feature\_extraction.text import TfidfVectorizer

# Create a TF-IDF Vectorizer

vectorizer = TfidfVectorizer(max\_features=5000)  # Limit the features to 5000 most important ones

# Fit and transform the text data

X = vectorizer.fit\_transform(df['query'])

1. Encoding the target labels

**Code:**

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

# Encode the target language labels

df['language\_encoded'] = le.fit\_transform(df['lan\_code'])

y = df['language\_encoded']

1. Builds automated pipeline

**Code:**

from sklearn.naive\_bayes import MultinomialNB

from sklearn.pipeline import Pipeline # Import Pipeline

from sklearn.feature\_extraction.text import TfidfVectorizer

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

def preprocess\_and\_train\_pipeline(df, test\_size=0.05, random\_state=42):

    # Sample the data for testing

    df\_sample = df.sample(n=10000, random\_state=random\_state)

    # Split the data into train and test

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(

        df\_sample['query'], df\_sample['language\_encoded'], test\_size=0.2, random\_state=42

    )

    # Build the pipeline with Naive Bayes

    pipeline = Pipeline([

        ('vectorizer', TfidfVectorizer(stop\_words='english', max\_features=200, min\_df=5, max\_df=0.8)),

        ('model', MultinomialNB())

    ])

    # Train the pipeline

    pipeline.fit(X\_train, y\_train)

    return pipeline, X\_test, y\_test  # Return both the trained pipeline and X\_test for evaluation

# Check the successful of training model

print("Model training complete")

1. Saving pipeline & label encoder ( Can avoid retraining the model )

**Code:**

# Train the pipeline and get X\_test and y\_test

trained\_pipeline, X\_test, y\_test = preprocess\_and\_train\_pipeline(df)

# Save the entire pipeline (model + vectorizer)

with open('language\_classifier\_pipeline.pkl', 'wb') as pipeline\_file:

    pickle.dump(trained\_pipeline, pipeline\_file)

# Optionally, also save the label encoder

with open('label\_encoder.pkl', 'wb') as le\_file:

pickle.dump(le, le\_file)

1. Evaluating the performance of the saved model

**Code:**

import pickle

from sklearn.metrics import classification\_report

# Load the saved pipeline

with open('language\_classifier\_pipeline.pkl', 'rb') as pipeline\_file:

    loaded\_pipeline = pickle.load(pipeline\_file)

# Load the label encoder

with open('label\_encoder.pkl', 'rb') as le\_file:

    le = pickle.load(le\_file)

# Make predictions

y\_pred = loaded\_pipeline.predict(X\_test)

# Convert encoded predictions back to original labels

y\_pred\_original = le.inverse\_transform(y\_pred)

# Convert the encoded true labels (y\_test) back to original labels for comparison

y\_test\_original = le.inverse\_transform(y\_test)

# Print the classification report

print(classification\_report(y\_test\_original, y\_pred\_original, target\_names=le.classes\_))