1. Connect to drive

   Code:

   ```
   from google.colab import drive
   drive.mount('/content/drive')
   ```

2. Import datasets

   Code:

   ```
   import pandas as pd
   df = pd.read_csv('/content/drive/MyDrive/Language( Malay & English )_dataset.csv')
   print(df.head())
   ```

3. 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")
   ```

4. 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'])
```

5. 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']
```

6. 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")
```

7. 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)
```

8. 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_))
```