Task 3:Email Spam prediction.

# Import necessary libraries

import pandas as pd

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

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.naive\_bayes import MultinomialNB

from sklearn.metrics import accuracy\_score, classification\_report

def load\_data(file\_path):

"""Load data from CSV file."""

try:

data = pd.read\_csv(file\_path, encoding='latin-1')

return data

except FileNotFoundError:

print(f"Error: File '{file\_path}' not found.")

return None

except Exception as e:

print(f"Error loading data: {e}")

return None

def preprocess\_data(X, y):

"""Preprocess text data and convert labels to numerical."""

try:

vectorizer = CountVectorizer()

X = vectorizer.fit\_transform(X)

return X, y

except Exception as e:

print(f"Error in preprocessing data: {e}")

return None, None

def train\_model(X\_train, y\_train):

"""Train a Naive Bayes classifier."""

try:

model = MultinomialNB()

model.fit(X\_train, y\_train)

return model

except Exception as e:

print(f"Error in training model: {e}")

return None

def evaluate\_model(model, X\_test, y\_test):

"""Evaluate the trained model."""

try:

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

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

report = classification\_report(y\_test, y\_pred)

return accuracy, report

except Exception as e:

print(f"Error in evaluating model: {e}")

return None, None

def main():

# Adjust file path according to your actual file location

file\_path = r"D:\User\Downloads\spam.csv"

# Step 1: Load the dataset

data = load\_data(file\_path)

if data is None:

return

# Step 2: Preprocess the data

X = data['v2']

y = data['v1'].map({'ham': 0, 'spam': 1})

X, y = preprocess\_data(X, y)

if X is None or y is None:

return

# Step 3: Split data 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: Train the model

model = train\_model(X\_train, y\_train)

if model is None:

return

# Step 5: Evaluate the model

accuracy, report = evaluate\_model(model, X\_test, y\_test)

if accuracy is not None and report is not None:

print(f"Accuracy: {accuracy}")

print(f"Classification Report:\n{report}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Email Spam Detector Documentation**

**Introduction**

The Email Spam Detector is a Python-based machine learning application designed to classify emails as either "spam" or "ham" (legitimate). This documentation provides a detailed overview of the implementation, design choices, and considerations made during development.

**Dataset**

The dataset used for training and testing the Email Spam Detector is the SMS Spam Collection. This dataset consists of SMS messages labeled as either "ham" or "spam". Each message is accompanied by its label and raw text content. The dataset was chosen due to its availability and suitability for training a binary classification model.

**Implementation Overview**

The implementation of the Email Spam Detector follows a structured approach involving several key steps:

1. **Data Loading and Preprocessing:**
   * **Data Loading:** The dataset is loaded from a CSV file (spam.csv). The Pandas library is utilized for efficient data handling and manipulation.
   * **Text Preprocessing:** Text preprocessing involves cleaning the raw text data by removing unnecessary characters, converting text to lowercase, and tokenizing the text into individual words or tokens. This step also includes removing stopwords (commonly occurring words like "and", "the", etc.) and transforming the text into a numerical format suitable for machine learning algorithms.
2. **Feature Extraction:**
   * **Count Vectorization:** The CountVectorizer from scikit-learn is used to convert text data into a matrix of token counts. Each row represents a document (email in this case), and each column represents a unique word in the dataset. This transformation enables the machine learning model to understand and process textual data.
3. **Model Selection and Training:**
   * **Naive Bayes Classifier:** A Multinomial Naive Bayes classifier is chosen for its simplicity and effectiveness in text classification tasks. Naive Bayes classifiers are well-suited for this application due to their ability to handle large feature spaces efficiently and their robust performance with relatively small amounts of training data.
4. **Model Evaluation:**
   * **Training and Testing Split:** The dataset is split into training and testing sets using a 80-20 split ratio. The training set is used to train the model, while the testing set evaluates its performance on unseen data.
   * **Evaluation Metrics:** Performance metrics such as accuracy, precision, recall, and F1-score are calculated to assess the model's effectiveness in distinguishing between spam and ham emails. These metrics provide insights into the model's strengths and weaknesses.
5. **Error Handling and Robustness:**
   * **Error Handling:** Throughout the implementation, robust error handling practices are employed to catch and handle potential exceptions. This ensures the application gracefully handles errors such as missing files, data preprocessing failures, or model training issues.
6. **Deployment Considerations:**
   * **Scalability:** The implemented solution is designed to handle moderate-sized datasets efficiently. For larger datasets or real-time classification requirements, considerations for scalability and performance optimizations would be necessary.
   * **Security:** In a production environment, additional measures would be taken to secure sensitive data and ensure the model's predictions are integrated securely into email processing pipelines.

**Challenges Faced**

During the implementation of the Email Spam Detector, several challenges were encountered and addressed:

* **Data Cleaning:** Ensuring the text data is properly cleaned and standardized required careful handling of special characters, URLs, and email addresses present in the dataset.
* **Model Selection:** Choosing an appropriate machine learning model that balances accuracy and computational efficiency was crucial. Naive Bayes was selected for its simplicity and initial good performance, but other models could be explored for further optimization.
* **Performance Tuning:** Optimizing the model's hyperparameters and evaluating different feature extraction techniques (e.g., TF-IDF) were considered to improve classification accuracy and robustness.

**Conclusion**

The Email Spam Detector provides a foundational framework for classifying emails based on their content. By leveraging machine learning techniques and a carefully curated dataset, the detector demonstrates effective spam detection capabilities. Future enhancements could include exploring advanced models, integrating with real-time email systems, and deploying the solution in a scalable and secure manner. This documentation serves as a guide to understanding the implementation details and considerations involved in building an email spam detection system.