import pandas as pd

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.naive\_bayes import MultinomialNB

import matplotlib.pyplot as plt

import seaborn as sns

import gradio as gr

data = pd.read\_csv('spam\_ham\_dataset.csv')

data

def data\_set():

spam\_df = pd.read\_csv('spam\_ham\_dataset.csv')

return spam\_df.drop('Unnamed: 0',axis=1)

data = data\_set()

data

data.shape

data.head()

data.tail()

data.info()

data.describe()

plt.hist(data['label'])

plt.show()

vectorizer=CountVectorizer()

spamham\_countVectorizer=vectorizer.fit\_transform(data['text'])

X=spamham\_countVectorizer

y= data['label\_num']

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

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

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.2)

NB\_classifier=MultinomialNB()

NB\_classifier.fit(X,y)

y\_predict\_test=NB\_classifier.predict(X\_test)

cm=confusion\_matrix(y\_test,y\_predict\_test)

sns.heatmap(cm,annot=True)

print(classification\_report(y\_test,y\_predict\_test)

def check(text):

df =data\_set()

X = df['text']

y = df['label\_num']

vectorizer=CountVectorizer()

X = vectorizer.fit\_transform(X)

model = MultinomialNB()

model.fit(X, y)

new\_email\_features = vectorizer.transform([text])

prediction = model.predict(new\_email\_features)

if prediction[0] == 1:

return "The email is spam!"

else:

return "The email is not spam."

app1 = gr.Interface(fn=data\_set,inputs=None, outputs=gr.Dataframe(),description="Reading csv")

app3 = gr.Interface(fn=check, inputs='text', outputs='text',description=" result")

demo = gr.TabbedInterface([app1, app3], ["Reading\_csv", "result"])

demo.launch()

1. Importing Libraries: The code begins by importing the necessary libraries for the tasks at hand. These include pandas for data manipulation and analysis, CountVectorizer from sklearn.feature\_extraction.text for text preprocessing, MultinomialNB from sklearn.naive\_bayes for the Naive Bayes classifier, matplotlib.pyplot for data visualization, seaborn for creating heatmaps, and gradio for building the graphical interface.
2. Loading the Data: The data\_set function is defined to read the data from a CSV file called 'spam\_ham\_dataset.csv'. The pd.read\_csv() function from the pandas library is used to load the data into a DataFrame, and the function returns the DataFrame after dropping the 'Unnamed: 0' column.
3. Data Exploration: The loaded DataFrame is stored in the variable data. Several operations are performed to explore the data:
   * data.shape returns the dimensions of the DataFrame, i.e., the number of rows and columns.
   * data.head() displays the first few rows of the DataFrame.
   * data.tail() displays the last few rows of the DataFrame.
   * data.info() provides information about the DataFrame, including the column names, data types, and non-null counts.
   * data.describe() generates descriptive statistics of the numerical columns in the DataFrame.
   * plt.hist(data['label']) creates a histogram of the 'label' column using matplotlib. This can help visualize the distribution of spam and ham (non-spam) emails in the dataset.
4. Data Preprocessing: In order to use text data as input for the Naive Bayes classifier, it needs to be transformed into a numerical representation. The following steps are performed:
   * CountVectorizer is initialized as vectorizer, which will convert the text data into a matrix of token counts.
   * spamham\_countVectorizer is created by fitting and transforming the 'text' column of the DataFrame using vectorizer.
   * The transformed data is assigned to X.
   * The target variable 'label\_num' is assigned to y.
5. Train-Test Split: The data is split into training and testing sets using the train\_test\_split function from sklearn.model\_selection. The training set comprises 80% of the data, while the testing set contains the remaining 20%. The split data is stored in the variables X\_train, X\_test, y\_train, and y\_test.
6. Model Training and Evaluation: The Naive Bayes classifier (MultinomialNB) is initialized as NB\_classifier, and it is trained on the training data using the fit method. The trained model is then used to make predictions on the test data (X\_test) with NB\_classifier.predict. The confusion matrix is computed using confusion\_matrix from sklearn.metrics, and a heatmap visualization of the confusion matrix is created using sns.heatmap. Finally, the classification report, which includes precision, recall, F1-score, and support, is printed using classification\_report.
7. Email Spam Detection Function: The check function is defined to classify whether a given email text is spam or not. It takes a text input, preprocesses it using the same CountVectorizer and MultinomialNB models used before, and returns the prediction as a string.
8. Building the Gradio Interface: Two Gradio interfaces are created: app1 and app3. app1 displays the loaded DataFrame when called, while app3 takes a text input and returns the spam detection result. The gr.TabbedInterface combines both interfaces into a single tabbed interface for easy switching between them. Finally, the demo.launch() method is called to start the Gradio interface.

In summary, this code loads a spam/ham email dataset, performs data exploration and