

How to run :

Go to your project folder

```
cd "C:\Users\YourName\Desktop\spam_mail_detection"
```

Optional: Create virtual environment

```
python -m venv venv
```

```
.\venv\Scripts\activate
```

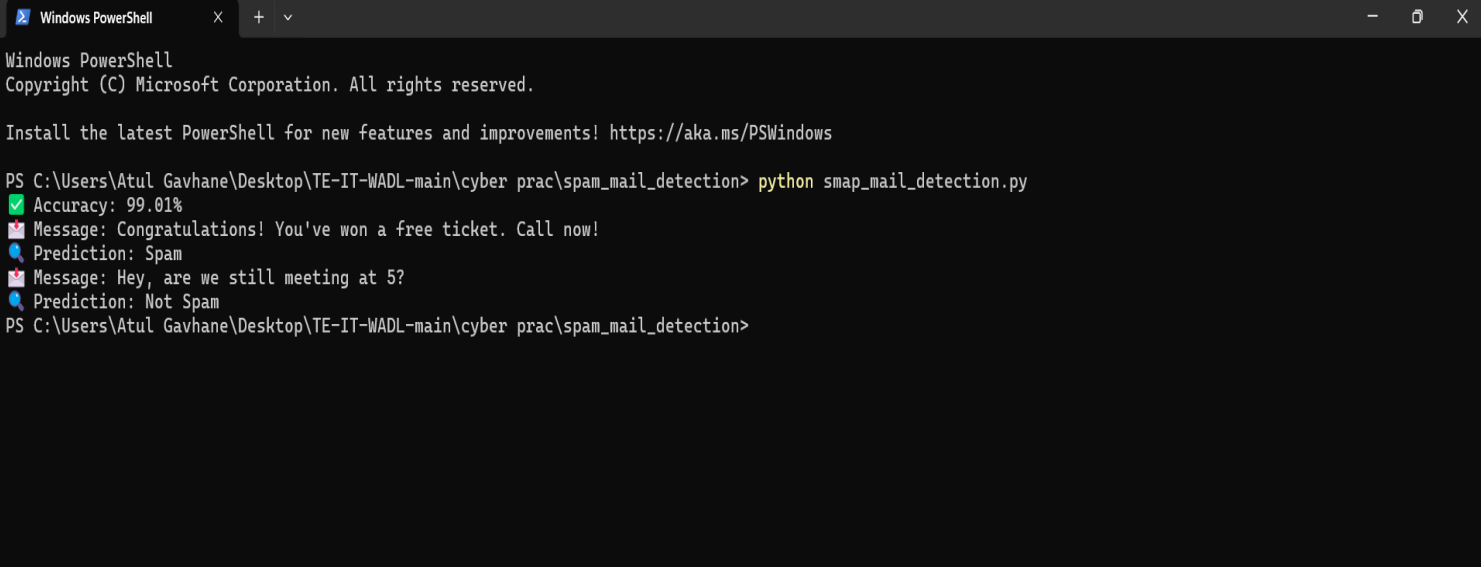
Install required packages

```
pip install pandas scikit-learn
```

Run the script

```
python spam_mail_detection.py
```

Output :



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Atul Gavhane\Desktop\TE-IT-WADL-main\cyber prac\spam_mail_detection> python smap_mail_detection.py
✅ Accuracy: 99.01%
📧 Message: Congratulations! You've won a free ticket. Call now!
🔍 Prediction: Spam
📧 Message: Hey, are we still meeting at 5?
🔍 Prediction: Not Spam
PS C:\Users\Atul Gavhane\Desktop\TE-IT-WADL-main\cyber prac\spam_mail_detection>
```

Code and examiner explanation :

Code Breakdown:

1. Importing 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
```

- **pandas**: Used for handling and manipulating datasets.
 - **train_test_split**: Splits the dataset into training and testing parts.
 - **CountVectorizer**: Converts the raw text data (messages) into numerical form that machine learning algorithms can understand.
 - **MultinomialNB**: Implements the **Naive Bayes classifier** for spam classification.
 - **accuracy_score**: Used to calculate the accuracy of the model.
-

2. Loading the Dataset

```
data = pd.read_csv('https://raw.githubusercontent.com/justmarkham/pycon-2016-tutorial/master/data/sms.tsv',
                  sep='\t', names=['label', 'message'])
```

- The dataset is loaded directly from an online source (a **tab-separated values** file).
 - It contains **two columns**:
 - **label**: Indicates whether the message is **spam** or **ham** (ham = non-spam).
 - **message**: The actual text content of the message.
-

3. Data Preprocessing

```
data['label_num'] = data.label.map({'ham': 0, 'spam': 1})
```

- This step **maps the labels** into **numerical values**:
 - **ham** → 0
 - **spam** → 1
 - It's easier for the machine learning model to work with numbers, so we convert the labels this way.
-

4. Splitting the Data

```
X_train, X_test, y_train, y_test = train_test_split(data['message'], data['label_num'], test_size=0.2, random_state=1)
```

- **train_test_split** randomly splits the data into:
 - **X_train**: 80% of the data (for training the model).
 - **X_test**: 20% of the data (for testing the model).
 - **y_train and y_test**: Corresponding labels (spam or ham).
 - **random_state=1** ensures reproducibility of results.
-

5. Text Vectorization (Converting text to numbers)

```
vectorizer = CountVectorizer()
X_train_vector = vectorizer.fit_transform(X_train)
X_test_vector = vectorizer.transform(X_test)
```

- **CountVectorizer** converts the text messages into a **bag-of-words representation**:

- Each unique word is treated as a feature.
 - For each message, it counts how many times each word appears.
 - **fit_transform**: This processes the training data.
 - **transform**: This applies the same transformation to the test data (to ensure consistency).
-

6. Training the Naive Bayes Model

```
model = MultinomialNB()
model.fit(X_train_vector, y_train)
```

- A **Naive Bayes classifier** is trained on the vectorized text data.
 - **MultinomialNB** is used for classification tasks where the features are counts of words (or occurrences).
 - **model.fit()** trains the model using the training data (**X_train_vector**).
-

7. Evaluating the Model

```
y_pred = model.predict(X_test_vector)
accuracy = accuracy_score(y_test, y_pred)
```

- The **model predicts** the labels for the test data (**X_test_vector**).
 - **accuracy_score** compares the predicted labels (**y_pred**) with the true labels (**y_test**) and calculates the accuracy.
-

8. Testing Custom Messages

```
def predict_message(msg):
    msg_vector = vectorizer.transform([msg])
    prediction = model.predict(msg_vector)[0]
    print("\n📧 Message:", msg)
    print("🔍 Prediction:", "Spam" if prediction == 1 else "Not Spam")
```

- **predict_message()** takes a custom message and transforms it into a vector using the same vectorizer.
 - The **Naive Bayes model** predicts if the message is **Spam** (1) or **Not Spam** (0).
-

9. Testing the Model with Sample Messages

```
predict_message("Congratulations! You've won a free ticket. Call now!")
predict_message("Hey, are we still meeting at 5?")
```

- These are example messages for testing the model.
-



How to Explain This to the Examiner:

1. Introduction to the Problem

- Explain that the problem you're solving is **Spam Mail Detection** using machine learning.
- Spam detection is important in filtering out unwanted or malicious emails from legitimate ones.

2. Dataset Overview

- Mention that you're using a publicly available SMS dataset with two labels: **spam** and **ham** (non-spam).
- Explain that the dataset contains two columns: one for the label (spam or ham) and the other for the message text.

3. Data Preprocessing

- Explain how the **labels** (spam and ham) are converted into numerical values (0 for ham, 1 for spam) because machine learning algorithms can only work with numbers.

4. Text Vectorization

- Discuss how text data is converted into **numerical form** using **CountVectorizer**.
 - This method counts the occurrences of each word in the message and converts the text into a matrix of numbers.

5. Training the Model

- Explain that you are using the **Naive Bayes classifier**, a probabilistic model, which works well for text classification problems like spam detection.
- Mention that the model was trained using the training data and the vectorized words.

6. Model Evaluation

- Talk about how you split the data into training and testing sets, and how you evaluate the model's performance by comparing the predicted labels with the actual labels in the test data.
- Mention the **accuracy score** (99.01% in your case) to show the effectiveness of the model.

7. Testing with Custom Messages

- Explain that after training, you can use the model to predict whether new (unseen) messages are spam or not.
- You can use custom messages as input and show how the model makes its prediction (e.g., "Spam" or "Not Spam").

8. Conclusion

- Conclude by stating that this model could be used to **filter out spam messages** in real-time applications, reducing the need for manual intervention.
-