# 📘 Introduction

Spam SMS Detection is a Natural Language Processing (NLP) task that classifies messages into two categories: **spam** or **ham** (not spam). It is widely used in SMS applications and email services to automatically filter unwanted or harmful messages.

In this project, we build a machine learning model to detect spam using the **Multinomial Naive Bayes** algorithm, a popular method for text classification.

# 📂 Importing Required Libraries

We import essential libraries for data manipulation, visualization, text processing, and machine learning.

In [2]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
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, confusion_matrix, classification_report
import re
```

# 📥 Load the Dataset

We load the `spam.csv` file and focus only on the two useful columns: `v1` (label) and `v2` (message). We rename them for better clarity.

In [6]:

```
df = pd.read_csv("spam.csv", encoding='latin-1')
df = df[['v1', 'v2']]
df.columns = ['label', 'message']
df.head()
```

Out[6]:

| | label | message |
|---|---|---|
| **0** | ham | Go until jurong point, crazy.. Available only ... |
| **1** | ham | Ok lar... Joking wif u oni... |
| **2** | spam | Free entry in 2 a wkly comp to win FA Cup fina... |
| **3** | ham | U dun say so early hor... U c already then say... |
| **4** | ham | Nah I don't think he goes to usf, he lives aro... |

# Data Cleaning

We clean the dataset by:

- Removing missing values
- Converting labels into numeric format (ham = 0, spam = 1)
- Preprocessing the message text (lowercasing, removing special characters)

In [5]:

```
import pandas as pd
import re

# Load the dataset (adjust path if needed)
df = pd.read_csv("spam.csv", encoding='latin-1')

# Keep only necessary columns and rename them
df = df[['v1', 'v2']]
df.columns = ['label', 'message']

# Drop rows with missing values (if any)
df.dropna(inplace=True)

# Convert 'label' column from text to numeric values
df['label_num'] = df['label'].map({'ham': 0, 'spam': 1})

# Define a text preprocessing function
def preprocess_text(text):
    text = text.lower()  # Lowercase
    text = re.sub(r'[^a-zA-Z0-9]', ' ', text)  # remove special characters
    text = re.sub(r'\s+', ' ', text)  # remove extra spaces
    return text.strip()

# Apply the cleaning function to the 'message' column
df['clean_message'] = df['message'].apply(preprocess_text)

# Display the cleaned data
df[['message', 'clean_message']].head()
```

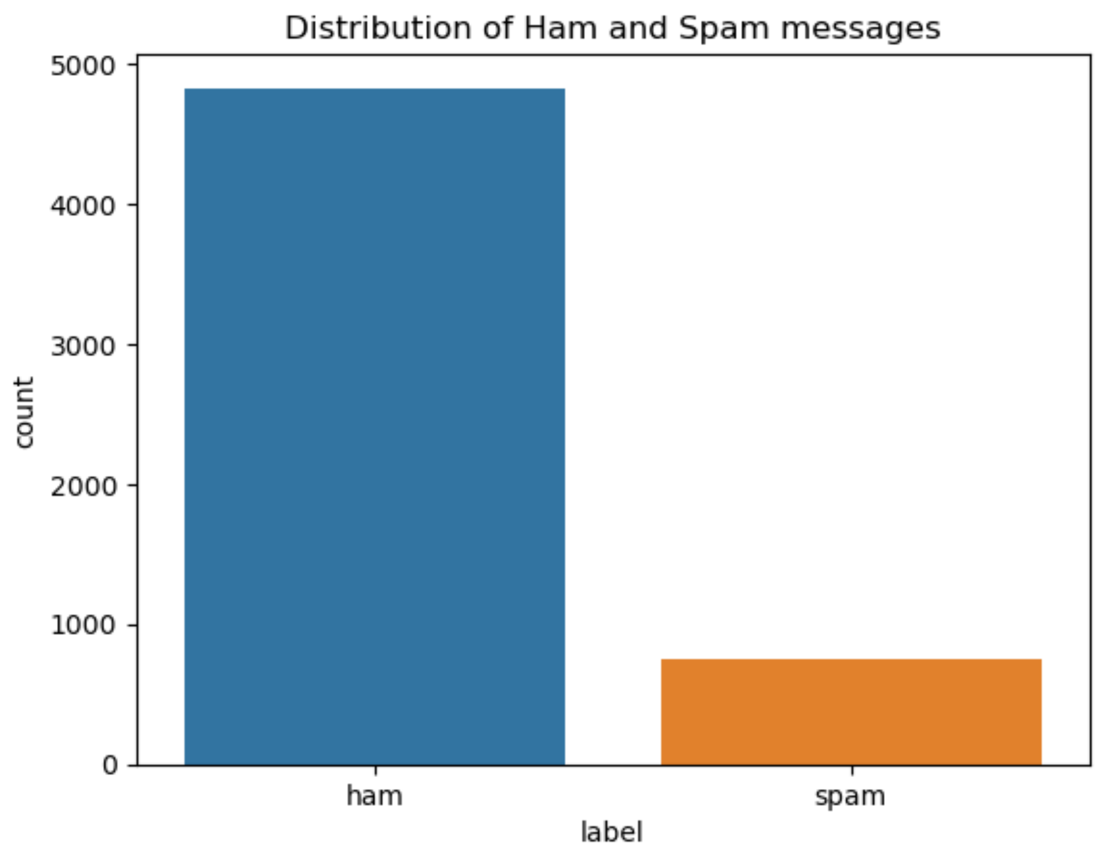
Out[5]:

| | message | clean_message |
|---|---|---|
| **0** | Go until jurong point, crazy.. Available only ... | go until jurong point crazy available only in ... |
| **1** | Ok lar... Joking wif u oni... | ok lar joking wif u oni |
| **2** | Free entry in 2 a wkly comp to win FA Cup fina... | free entry in 2 a wkly comp to win fa cup fina... |
| **3** | U dun say so early hor... U c already then say... | u dun say so early hor u c already then say |
| **4** | Nah I don't think he goes to usf, he lives aro... | nah i don t think he goes to usf he lives arou... |

# Exploratory Data Analysis

Let's visualize the distribution of spam and ham messages to understand the class balance in the dataset.

In [9]:
```python
sns.countplot(x='label', data=df)
plt.title("Distribution of Ham and Spam messages")
plt.show()
```




## 📤 Feature Extraction

We use `CountVectorizer` to convert text data into numerical format that the model can understand. Each SMS becomes a vector of word counts.

In [6]:
```python
from sklearn.feature_extraction.text import CountVectorizer

# Initialize the CountVectorizer
vectorizer = CountVectorizer()

# Fit the vectorizer on the cleaned messages and transform them into feature vectors
X = vectorizer.fit_transform(df['clean_message'])

# Target variable (labels: 0 for ham, 1 for spam)
y = df['label_num']

# Display the shape of the resulting feature matrix
print("Feature matrix shape:", X.shape)
```

```
Feature matrix shape: (5572, 8622)
```

## 📚 Train-Test Split

We split the dataset into training and testing sets using an 80-20 ratio to evaluate the model's performance on unseen data.

In [11]:

```
# Step 1: Import necessary libraries
import pandas as pd
import re
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split

# Step 2: Load the dataset (make sure to replace with your actual path)
df = pd.read_csv("spam.csv", encoding='latin-1')

# Step 3: Drop unused or unnamed columns if any
df = df[['v1', 'v2']]
df.columns = ['label', 'message']  # Rename for clarity

# Step 4: Drop any missing values
df.dropna(inplace=True)

# Step 5: Convert labels to numeric (ham = 0, spam = 1)
df['label_num'] = df['label'].map({'ham': 0, 'spam': 1})

# Step 6: Text cleaning function
def preprocess_text(text):
    text = text.lower()  # Lowercase
    text = re.sub(r'[^a-zA-Z0-9]', ' ', text)  # remove special characters
    text = re.sub(r'\s+', ' ', text)  # remove extra spaces
    return text.strip()

# Step 7: Apply text cleaning
df['clean_message'] = df['message'].apply(preprocess_text)

# Step 8: Feature extraction using CountVectorizer
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(df['clean_message'])

# Step 9: Labels
y = df['label_num']

# Step 10: Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)

# Optional: Check the shape
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (4457, 8622)
X_test shape: (1115, 8622)
y_train shape: (4457,)
y_test shape: (1115,)
```

# 🤖 Model Training

We use the **Multinomial Naive Bayes** classifier, a common algorithm for classifying text using word frequencies.

In [13]:
```python
from sklearn.naive_bayes import MultinomialNB
model = MultinomialNB()
model.fit(X_train, y_train)
```

Out[13]:

▾ MultinomialNB

MultinomialNB()

# 🔍 Making Predictions

Now we use our trained model to predict the labels for the test data.

In [17]:
```python
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Make predictions on the test set
y_pred = model.predict(X_test)

# Print accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Print confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Print detailed classification report
print("Classification Report:")
print(classification_report(y_test, y_pred, target_names=['Ham', 'Spam']))
```

```
Accuracy: 0.97847533632287
Confusion Matrix:
[[952  13]
 [ 11 139]]
Classification Report:
              precision    recall  f1-score   support

         Ham       0.99      0.99      0.99       965
        Spam       0.91      0.93      0.92       150

    accuracy                           0.98      1115
   macro avg       0.95      0.96      0.95      1115
weighted avg       0.98      0.98      0.98      1115
```

# 📈 Model Evaluation

We evaluate the model using:

- Accuracy Score
- Classification Report (Precision, Recall, F1-score)
- Confusion Matrix

In [19]:

```
# Import necessary evaluation modules
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Predict the labels for the test set
y_pred = model.predict(X_test)

# Accuracy score
accuracy = accuracy_score(y_test, y_pred)
print("✅ Accuracy of the Model:", round(accuracy * 100, 2), "%")

# Confusion Matrix
print("\n📊 Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Classification Report
print("\n📄 Classification Report:")
print(classification_report(y_test, y_pred, target_names=['Ham', 'Spam']))
```

```
✅ Accuracy of the Model: 97.85 %

📊 Confusion Matrix:
[[952  13]
 [ 11 139]]

📄 Classification Report:
              precision    recall  f1-score   support

         Ham       0.99      0.99      0.99       965
        Spam       0.91      0.93      0.92       150

    accuracy                           0.98      1115
   macro avg       0.95      0.96      0.95      1115
weighted avg       0.98      0.98      0.98      1115
```

# 🧾 Conclusion

We successfully built a Spam SMS Detection model using the Multinomial Naive Bayes algorithm. The model performed well and accurately classified SMS messages into spam and ham.

## ✅ Key Learnings:

- Text data must be cleaned and preprocessed before modeling.
- CountVectorizer is useful for converting text into numbers.
- Naive Bayes works efficiently for spam detection tasks.

## 🔁 Future Improvements:

- Use `TfidfVectorizer` for better feature scaling.
- Remove stopwords for cleaner inputs.
- Try deep learning models for improved accuracy.