🧠 Spam Email Detection using Naive Bayes

Step-by-Step Explanation with Purpose

🔹 Step 1: Import Required Libraries

What we did

We imported Python libraries such as:

- pandas, numpy
- matplotlib, seaborn
- sklearn modules for ML, feature extraction, and evaluation

Purpose

- **pandas** → Load and manipulate dataset
- **numpy** → Perform numerical operations
- **matplotlib / seaborn** → Visualization (confusion matrix, ROC curve)
- **sklearn** →
  - Build Naive Bayes models
  - Extract text features
  - Evaluate model performance

📌 **Why this step is important:**
Machine Learning requires multiple tools—data handling, modeling, and evaluation—all provided by these libraries.

🔹 Step 2: Load the Dataset

What we did

```
df = pd.read_csv("spam.csv")
```

We loaded the **SMS Spam Collection Dataset** into a DataFrame.

Purpose

- Convert raw CSV data into a **structured tabular format**
- Enable easy access to:
  - Email/SMS text
  - Spam or ham labels

📌 **Why this step is important:**
Machine learning models cannot work directly with raw files; data must be loaded into memory in a structured form.

🔹 **Step 3: Data Cleaning & Column Selection**

What we did

- Removed unnecessary columns
- Renamed columns as:
  - label
  - message

Purpose

- Eliminate noise and irrelevant information
- Improve readability and clarity
- Focus only on **features required for classification**

📌 **Why this step is important:**
Cleaner data improves model accuracy and reduces processing overhead.

🔹 **Step 4: Label Encoding**

What we did

Converted:

- spam → 1
- ham → 0

Using LabelEncoder.

Purpose

Machine learning algorithms **cannot understand text labels**, they require **numerical values**.

📌 **Why this step is important:**
Naive Bayes performs mathematical probability calculations, which require numeric inputs.

🔹 **Step 5: Train-Test Split**

What we did

Split dataset into:

- **Training data (80%)**

- **Testing data (20%)**

Purpose

- Train the model on known data
- Test the model on **unseen data**

📌 **Why this step is important:**
To check whether the model can **generalize** and not just memorize data.

🧠 This prevents **overfitting**.

🔹 **Step 6: Text Preprocessing**

What happens internally

- Tokenization (splitting text into words)
- Stop-word removal (removing common words like *is, the, and*)
- Lowercasing

Purpose

- Reduce noise
- Keep only meaningful words
- Improve classification accuracy

📌 **Why this step is important:**
Spam detection depends on important keywords like *free*, *win*, *offer*.

🔹 **Step 7: Feature Extraction**

Machine learning models **cannot understand raw text**, so we convert text into numbers.

🔸 **7.1 Bag of Words (BoW)**

What it does

- Creates a vocabulary of all unique words
- Counts word frequency in each message

Example:

"Free offer now" → [1,1,1,0,0]

Purpose

- Represent text numerically

- Capture word occurrence patterns

📌 **Why this step is important:**
Spam emails often repeat specific words frequently.

🔸 **7.2 TF-IDF (Term Frequency–Inverse Document Frequency)**

What it does

- Assigns **higher weight to important words**
- Reduces importance of common words

Purpose

- Improve feature quality
- Reduce bias toward frequently occurring words

📌 **Why TF-IDF is better than BoW:**
Words like *free*, *winner* get more importance than *the*, *is*.

🔹 **Step 8: Multinomial Naive Bayes**

What we did

mnb = MultinomialNB()

mnb.fit(X_train_tfidf, y_train)

Purpose

- Designed for **discrete count-based features**
- Ideal for **text classification**

📌 **Why Multinomial NB is best for spam detection:**
It works on word frequencies and probabilities.

🧠 **Formula Used:**

$$P(Class \mid Words) \propto P(Class) \times \prod P(Word \mid Class)$$

🔹 **Step 9: Gaussian Naive Bayes**

What we did

Used Gaussian NB on dense TF-IDF values.

Purpose

- Handle **continuous-valued features**

- Compare performance with Multinomial NB

📌 **Why this step is important:**
Project requires **comparison of Gaussian vs Multinomial Naive Bayes**.

🧠 Gaussian assumes data follows **normal distribution**, which is less suitable for text.

🔹 **Step 10: Prediction**

What we did

y_pred = model.predict(X_test)

Purpose

- Classify unseen emails as:
    - Spam
    - Not Spam

📌 **Why this step is important:**
This is the **core goal** of the project.

🔹 **Step 11: Model Evaluation**

We evaluated using:

🔸 **Accuracy**

✓ Measures overall correctness

$$Accuracy = \frac{CorrectPredictions}{TotalPredictions}$$

🔸 **Precision**

✓ How many predicted spam emails were actually spam

$$Precision = \frac{TP}{TP + FP}$$

🔸 **Recall**

✓ How many spam emails were correctly detected

$$Recall = \frac{TP}{TP + FN}$$

🔶 **F1-Score**

✔ Balance between precision and recall

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

📌 **Why multiple metrics:**
Accuracy alone can be misleading for imbalanced datasets.

🔷 **Step 12: Confusion Matrix**

What it shows

Actual / Predicted Spam Not Spam

Purpose

- Visualize correct vs incorrect predictions
- Identify:
    - False Positives
    - False Negatives

📌 **Why this step is important:**
Helps understand model errors clearly.

🔷 **Step 13: ROC Curve**

What it shows

- Trade-off between:
    - True Positive Rate
    - False Positive Rate

Purpose

- Measure model's discrimination capability
- Higher AUC = better model

📌 **Why this step is important:**
Used in **real-world spam filter evaluation**.

🔹 Step 14: Testing on Unseen Emails

What we did

Passed new messages not in training data.

Purpose

- Demonstrate **real-world application**
- Prove model works on new data

📌 **Why this step is important:**
Shows practical usability beyond training dataset.

🔹 Final Conclusion

✔ Multinomial Naive Bayes performs better
✔ TF-IDF improves accuracy
✔ Naive Bayes is efficient for spam detection
✔ Model is suitable for real-world email filtering