# Assignment 1: Build a Spam Filter using Naive Bayes

Theory:  
Naive Bayes is a probabilistic classifier based on Bayes’ Theorem. It's effective for text classification tasks such as spam detection due to its simplicity, efficiency, and good performance on large datasets. In this assignment, TF-IDF is used to convert text into numerical features, and a Multinomial Naive Bayes model is used for classification.

## Code:

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 TfidfVectorizer  
from sklearn.naive\_bayes import MultinomialNB  
from sklearn.metrics import classification\_report, confusion\_matrix  
  
df = pd.read\_csv('spam.csv', encoding='latin-1')  
df = df[['v1', 'v2']]  
df.columns = ['label', 'message']  
df['label'] = df['label'].map({'ham': 0, 'spam': 1})  
  
tfidf = TfidfVectorizer(stop\_words='english')  
X = tfidf.fit\_transform(df['message'])  
y = df['label']  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
model = MultinomialNB()  
model.fit(X\_train, y\_train)  
  
y\_pred = model.predict(X\_test)  
print(classification\_report(y\_test, y\_pred))  
  
conf\_matrix = confusion\_matrix(y\_test, y\_pred)  
sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Ham', 'Spam'], yticklabels=['Ham', 'Spam'])  
plt.xlabel('Predicted')  
plt.ylabel('Actual')  
plt.title('Confusion Matrix')  
plt.show()

## Oral Questions and Answers:

* Q: What is Naive Bayes and why is it used for spam detection?

A: Naive Bayes is a classification algorithm based on Bayes’ Theorem. It's used for spam detection because it's simple, fast, and works well with high-dimensional text data.

* Q: Why do we use TF-IDF instead of raw text?

A: TF-IDF transforms text into numerical features by measuring word importance. It reduces the weight of common words and improves model accuracy.

* Q: Explain the purpose of train\_test\_split.

A: It divides the dataset into training and testing sets to evaluate how well the model generalizes to unseen data.

* Q: How does a confusion matrix help in model evaluation?

A: It shows actual vs predicted labels, allowing you to identify misclassifications like false positives and false negatives.

* Q: What does the classification\_report show?

A: It displays precision, recall, F1-score, and support to help evaluate classification performance.

# Assignment 2: DDoS Attack Classification

Theory:  
DDoS (Distributed Denial of Service) attacks attempt to make services unavailable by overwhelming them with traffic. Using machine learning, we can classify whether network traffic is normal or part of a DDoS attack. Random Forest is a suitable classifier here due to its robustness and ability to handle complex datasets.

## Code:

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.preprocessing import LabelEncoder, StandardScaler  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import classification\_report, confusion\_matrix  
  
df = pd.read\_csv('ddos\_dataset.csv')  
df.dropna(inplace=True)  
  
le = LabelEncoder()  
df['Label'] = le.fit\_transform(df['Label'])  
  
X = df.drop('Label', axis=1)  
y = df['Label']  
  
scaler = StandardScaler()  
X\_scaled = scaler.fit\_transform(X)  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)  
  
model = RandomForestClassifier(n\_estimators=100, random\_state=42)  
model.fit(X\_train, y\_train)  
  
y\_pred = model.predict(X\_test)  
print(classification\_report(y\_test, y\_pred))  
  
cm = confusion\_matrix(y\_test, y\_pred)  
sns.heatmap(cm, annot=True, fmt='d', cmap='Reds', xticklabels=le.classes\_, yticklabels=le.classes\_)  
plt.xlabel('Predicted')  
plt.ylabel('Actual')  
plt.title('Confusion Matrix - DDoS Detection')  
plt.show()

## Oral Questions and Answers:

* Q: What features are important in detecting DDoS attacks?

A: Packet rate, duration, byte size, and protocol type are common features used to detect anomalies in network traffic.

* Q: Why is feature scaling important in this task?

A: Scaling ensures that all features contribute equally to the model’s performance by normalizing their ranges.

* Q: Why did you choose Random Forest?

A: Random Forest is accurate, handles large datasets well, and is less prone to overfitting.

* Q: What does LabelEncoder do?

A: It converts categorical labels into numerical form so that they can be used by machine learning models.

* Q: How do you interpret the confusion matrix?

A: It shows the number of true positives, true negatives, false positives, and false negatives.

# Assignment 3: Splitting Dataset into Train and Test Sets

Theory:  
Splitting a dataset into training and testing subsets is crucial for evaluating the model’s performance on unseen data. This prevents overfitting and helps ensure that the model generalizes well. Typically, 70–80% of the data is used for training, and the rest for testing.

## Code:

import pandas as pd  
from sklearn.model\_selection import train\_test\_split  
  
df = pd.read\_csv('sample\_data.csv')  
  
X = df.drop('target', axis=1)  
y = df['target']  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=1)  
  
print('Training set size:', X\_train.shape)  
print('Test set size:', X\_test.shape)

## Oral Questions and Answers:

* Q: Why is it important to split data into training and test sets?

A: To evaluate the model’s generalization on unseen data and avoid overfitting.

* Q: What is the role of the random\_state parameter?

A: It ensures reproducibility by setting a seed for the random number generator used during splitting.

* Q: What is a good test size to use and why?

A: A test size between 20%-30% is common. It balances the amount of data used for training and for evaluating the model.

* Q: What is overfitting and how does splitting help prevent it?

A: Overfitting is when the model memorizes the training data but performs poorly on new data. Splitting helps detect this problem.

# Assignment 4: Feature Engineering on Raw Data

Theory:  
Feature engineering transforms raw data into meaningful features that improve model performance. It includes handling missing values, encoding categorical variables, scaling numerical values, and creating new derived features. It’s a critical step in the data preprocessing pipeline.

## Code:

import pandas as pd  
from sklearn.preprocessing import StandardScaler, OneHotEncoder  
  
df = pd.read\_csv('raw\_data.csv')  
  
# Example: Filling missing values  
df.fillna(method='ffill', inplace=True)  
  
# Example: Encoding categorical variables  
categorical\_cols = ['category']  
encoder = OneHotEncoder(sparse=False)  
encoded = encoder.fit\_transform(df[categorical\_cols])  
  
# Example: Scaling numerical features  
numerical\_cols = ['feature1', 'feature2']  
scaler = StandardScaler()  
scaled = scaler.fit\_transform(df[numerical\_cols])

## Oral Questions and Answers:

* Q: What is feature engineering and why is it important?

A: Feature engineering improves the predictive power of a model by transforming and creating input features from raw data.

* Q: What are different types of feature encoding?

A: Label Encoding (for ordinal data) and OneHot Encoding (for nominal data).

* Q: How do you handle missing values?

A: Common techniques include forward fill, backward fill, mean/mode imputation, or removing rows.

* Q: What is the purpose of feature scaling?

A: To normalize feature ranges, ensuring that no single feature dominates model training.

* Q: What’s the difference between OneHotEncoder and LabelEncoder?

A: LabelEncoder assigns a unique number to each category, while OneHotEncoder creates binary columns for each category.