# Spam or Not Spam

A Java-based email spam classification tool that reads labeled email data, extracts text-based features, trains a simple classifier, and reports performance metrics.

## Overview

**EmailClassifierApplication** processes a CSV dataset of emails labeled as spam or not spam. It systematically extracts a set of predefined features from each email's content, uses those features to train a basic classification model, and evaluates the model's accuracy on the training data.

## How It Works

1. **Data Loading**
   Reads `spam_or_not_spam.csv`, where each row consists of:
   - The raw email text (quoted if it contains commas)
   - A binary label (`1` for spam, `0` for not spam)
2. **Feature Extraction**
   The `FeatureExtractor` computes the following for every email:
   - **WordCount**: total number of words
   - **SpamWordCount**: occurrences of common spam trigger words
   - **MisspelledCount**: words not found in a basic dictionary
   - **SpecialCharCount**: count of non-alphanumeric characters
   - **UrlCount**: number of detected URLs
   - **AllCapsCount**: count of words in ALL CAPS
3. These values populate a `Map<String, Integer>` within each `Email` object.
4. **Training the Classifier**
   The `Classifier` class analyzes feature distributions conditioned on the spam label. During `train()`, it computes summary statistics (min, max, average) for each feature separately for spam and non-spam emails.
5. **Classification & Evaluation**
   - **Classification**: For each email, `classify()` computes a score based on its feature values and compares it to a learned threshold.
   - **Evaluation**: The application tallies total emails, actual vs. predicted spam counts, and computes overall accuracy.
6. **Summary Statistics**
   The nested `SummaryStats` class captures the minimum, maximum, and average

values for each feature across the dataset, aiding in understanding feature behavior and threshold selection.

# Components

- **EmailClassifierApplication.java**: Orchestrates data loading, model training, classification, and reporting.
- **Email.java**: Represents an email instance with its text, true label, and extracted features.
- **FeatureExtractor.java**: Contains static methods that parse the email text and compute each feature.
- **Classifier.java**: Implements the training logic (feature statistics) and the classification rule based on those statistics.
- **SummaryStats.java**: Utility for aggregating feature statistics (min, max, average) during training.

# Example Output:

**Total emails: 1200**

**Actual spam emails: 450**

**Predicted spam emails: 430**

**Model accuracy: 0.92**