**EMAIL SPAM DETECTION**

**INTRODUCTION**

Spam mail, or junk mail, is a type of email that is sent to a massive number of users at one time, frequently containing cryptic messages, scams, or most dangerously, phishing content.

While spam emails are sometimes sent manually by a human, most often, they are sent using a bot. Most popular email platforms, like Gmail and Microsoft Outlook, automatically filter spams by screening for recognizable phrases and patterns. A few common spam emails include fake advertisements, chain emails, and impersonation attempts.

Clicking on a spam email can be dangerous, exposing your computer and personal information to different types of malwares. Therefore, it’s important to implement additional safety measures to protect your device, especially when it handles sensitive information like user data.

**DATASET**

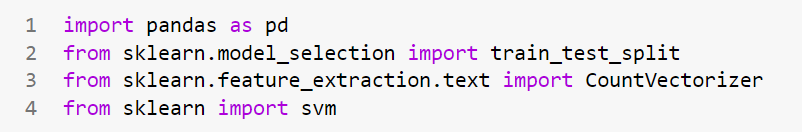
We have a spam.csv file, which we will turn into a data-frame and save to our folder spam. A data-frame is a structure that aligns data in a tabular fashion in rows and columns.

The dataset mimics the layout of a typical mailbox and includes over 5,000 examples that we’ll use to train our model.

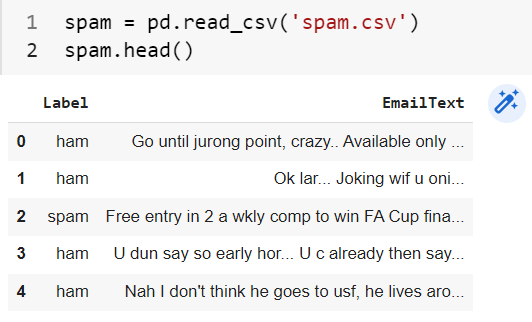
**METHODOLOGY**

* First, we’ll import the necessary dependencies. Pandas is a library used mostly by data scientists for data cleaning and analysis.

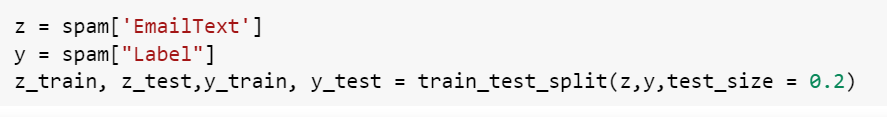
Scikit-learn, also called sklearn is a robust library for ML in Python. It provides a selection of efficient tools for ML and statistical modelling, including classification, regression, clustering and dimensionality reduction via a consistent interface.



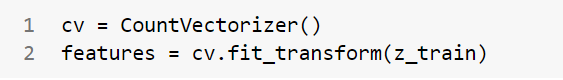
* Now we shall be importing the dataset ‘spam.csv’ and have a look at the first 5 rows of the data-frame.



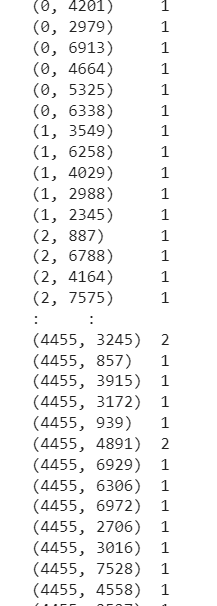
* We’ll use a train-test split to train our email spam detector to recognize and categorize spam emails. The procedure involves taking a dataset and dividing it into two separate datasets. The first dataset is used to fit the model and is referred to as the training dataset. For the second dataset, the test dataset, we provide the input element to the model. Finally we make predictions, comparing them against the actual output.
  + To split the data into our two datasets, we’ll use scikit-learn’s train\_test\_split() method.



* *Z=spam[‘EmailText’]* assigns the column *EmailText* from spam to *z*. It contains the data that we’ll run through the model.
  + *Y=spam[‘Label’]* assigns the column *Label* from spam to *y*, telling the model to correct the answer.
  + *Test\_size=0.2* sets the testing set to 20 percent of *z* and *y*.

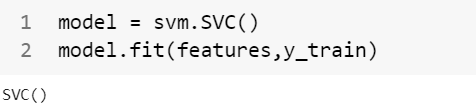


* **Building the model**
  + In *cv=CountVectorizer(), CountVectorizer()* randomly assigns a number to each word in a process called tokenizing. Then, it counts the number of occurrences of words and saves it to cv. At this point, we’ve only assigned a method to *cv*.
  + *Features=cv.fit\_transform(z\_train)* randomly assigns a number to each word. It counts the number of occurrences of each word, then saves it to *cv*. 0(zero) represents the index of the email. The number of sequences in the middle column represent a word recognized by our function, and the numbers on the right indicate the number of times that word was counted.



**ANALYSIS**

* SVM, support vector machine, is a linear model for classification and regression. The algorithm creates a line, or a hyperplane, which separates the data into classes. SVM can solve both linear and non-linear problems.

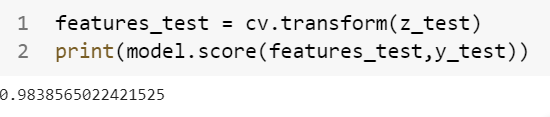


* *Model=svm.SVC()* assigns *svm.SVC()* to the model. In the *model.fit(features,y\_train)* function, *model.fit* trains the model with  *features* and *y\_train* label and adjusts its parameters until it reaches the highest possible accuracy.

**RESULT**

The *features\_test=cv.transform(z\_test)* function makes predictions from *z\_test* function makes predictions from

*Z\_test* that will go through count vectorization. It saves the results to the *features\_test* file.



In the *print(model.score(features\_test,y\_test))* function, *model.score()* scores the prediction of the *features\_test* against the actual labels in *y\_test.*