SMS Spam Collection Dataset and Preprocessing

In this step, we load the dataset for the spam detection project.The dataset is stored in a CSV file located at '/content/spam.csv'. We use the pandas library to read the CSV file and do some preprocessing to the dataset like text Cleaning, Stemming and etc.

To perform natural language processing tasks, we'll first install the Natural Language Toolkit (NLTK) library.

# Analysis:

We import the pandas library using import pandas as pd.

We use pd.read\_csv() to read the CSV file containing the dataset. The encoding='latin-1' argument is used to handle special characters.

We select only the relevant columns ('v1' for labels, 'v2' for email content) using data[['v1', 'v2']].

Finally, we display the resulting DataFrame to inspect the loaded data.

# Code:

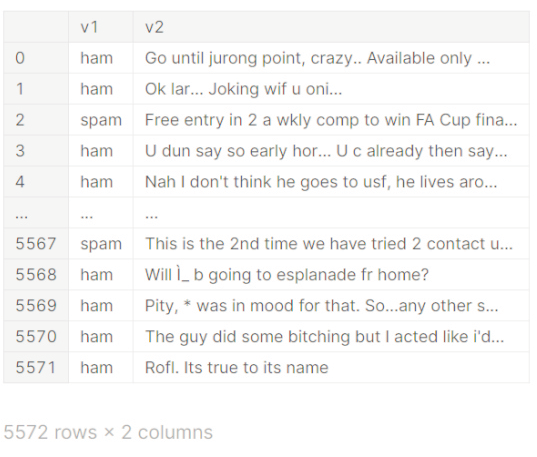
import pandas as pd

# Load the dataset

data = pd.read\_csv('/kaggle/input/sms-spam-collection-dataset/spam.csv', encoding='latin-1')

data = data[['v1', 'v2']] # Selecting only the relevant columns data #printing

**Output**



# Data Preprocessing

In this step, we perform data preprocessing tasks, which include converting labels to binary values and removing duplicates from the dataset.

#### Explanation:

We use data['v1'].apply(lambda x: 1 if x == 'spam' else 0) to convert the labels. 'ham' is mapped to 0, and 'spam' is mapped to 1 in the 'v1' column.

We then remove duplicate rows from the dataset using data = data.drop\_duplicates(). The resulting DataFrame is displayed to show the cleaned dataset.

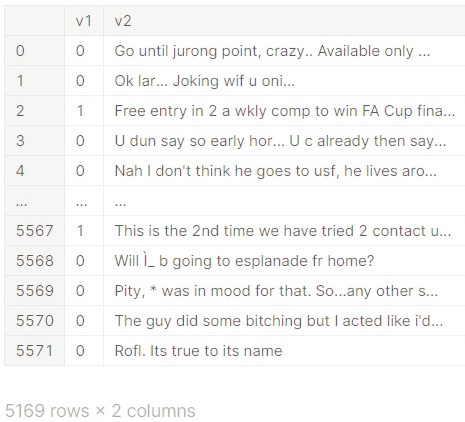
# Code:

# Convert 'ham' to 0 and 'spam' to 1 directly in the 'v1' column data['v1'] = data['v1'].apply(lambda x: 1 if x == 'spam' else 0)

# removing duplicates

data = data.drop\_duplicates() data

**Output**



# Text Cleaning:

Text cleaning involves removing any unnecessary characters, symbols, or noise from the text data. This might include punctuation, special characters, and numbers.

**Explanation:**

We import the regular expression (re) module using import re.

The function clean\_text() takes a string text as input and uses a regular expression to remove all characters except alphabetic characters .

The cleaned text is then returned.

We apply this function to the 'v2' column of the DataFrame using data['v2'].apply(lambda x: clean\_text(x)). This cleans the text in each email.

## Code:

### import re

def clean\_text(text):

cleaned\_text = re.sub(r'[^a-zA-Z]', ' ', text) return cleaned\_text

data['v2'] = data['v2'].apply(lambda x: clean\_text(x))

# Lowercasing:

Converting all text to lowercase ensures that the model doesn't treat "Hello" and "hello" as different words.

#### Explanation:

We use the str.lower() method to convert all text in the 'v2' column to lowercase. This helps standardize the text data and ensure that the model is not case-sensitive.

data['v2'] = data['v2'].str.lower()

# Tokenization:

Tokenization involves splitting the text into individual words or tokens. The NLTK library can be used for this.

#### Explanation:

In this code cell, we use nltk.download('punkt') to download the necessary resources for tokenization from the Natural Language Toolkit (NLTK). This resource includes pre-trained models for tokenizing text into words or sentences. This step is essential for further text processing.

# Code:

### import nltk

nltk.download('punkt')

[nltk\_data] Downloading package punkt to /usr/share/nltk\_data... [nltk\_data] Package punkt is already up-to-date!

**Output**

True

## Stemming:

Stemming reduces words to their base forms. This can help in reducing the dimensionality of the feature space.

**Explanation:**

We import the PorterStemmer class from the NLTK library. We initialize an instance of the PorterStemmer as stemmer.

We define a function stem\_words(words) that takes a list of words and applies stemming to each word using the stemmer.stem() method.

We apply this function to the 'v2' column of the DataFrame, effectively reducing words to their base forms through stemming. This step can help improve the model's performance by reducing the feature space.

## Code:

from nltk.stem import PorterStemmer stemmer = PorterStemmer()

def stem\_words(words):

return [stemmer.stem(word) for word in words]

data['v2'] = data['v2'].apply(stem\_words) data

# Output