Dataset Link https://www.kaggle.com/datasets/gargmanas/sentimental-analysis-for-tweets

text preprocessing addresses issues like :

- Lowercase letters.
- Removing HTML tags.
- Removing URLs.
- Removing punctuation.
- Chat Words Treatment.
- Spelling Correction.
- Removing stop words
- Handling Emojies
- Tokenization
- Stemming
- Lemmatization

We Will discuss the Solutions to Handles the above mentioned Issues.

```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("gargmanas/sentimental-analysis-for-tweets")

print("Path to dataset files:", path)
```

```
Warning: Looks like you're using an outdated `kagglehub` version (installed: 0.3.12), please consider upgrading to the latest version (0
Downloading from https://www.kaggle.com/api/v1/datasets/download/gargmanas/sentimental-analysis-for-tweets?dataset_version_number=1...
100%|██████████| 476k/476k [00:00<00:00, 588kB/s]Extracting files...
Path to dataset files: /root/.cache/kagglehub/datasets/gargmanas/sentimental-analysis-for-tweets/versions/1
```

```
import os
os.listdir("/root/.cache/kagglehub/datasets/gargmanas/sentimental-analysis-for-tweets/versions/1")
```

```
['sentiment_tweets3.csv']
```

```
# Import Basis Libraries
import pandas as pd
df = pd.read_csv('/root/.cache/kagglehub/datasets/gargmanas/sentimental-analysis-for-tweets/versions/1/sentiment_tweets3.csv')
```

```
df
```

	Index	message to examine	label (depression result)
0	106	just had a real good moment. i missssssssss hi...	0
1	217	is reading manga http://plurk.com/p/mzp1e	0
2	220	@comeagainjen http://twitpic.com/2y2lx - http:...	0
3	288	@lapcat Need to send 'em to my accountant tomo...	0
4	540	ADD ME ON MYSPACE!!! myspace.com/LookThunder	0
...	...	...	...
10309	802309	No Depression by G Herbo is my mood from now o...	1
10310	802310	What do you do when depression succumbs the br...	1
10311	802311	Ketamine Nasal Spray Shows Promise Against Dep...	1
10312	802312	dont mistake a bad day with depression! everyo...	1
10313	802313	0	1

10314 rows × 3 columns

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
# rename column
df.rename(columns = {'label (depression result)':'Sentiment'}, inplace = True)
df.rename(columns = {'message to examine':'review'}, inplace = True)
# Drop Index Column
df.drop('Index',axis=1,inplace=True)
```

```
# Head
df.head()
```

	review	Sentiment
0	just had a real good moment. i missssssssss hi...	0
1	is reading manga http://plurk.com/p/mzp1e	0
2	@comeagainjen http://twitpic.com/2y2lx - http:...	0
3	@lapcat Need to send 'em to my accountant tomo...	0
4	ADD ME ON MYSPACE!!! myspace.com/LookThunder	0

Next steps: Generate code with df View recommended plots New interactive sheet

1. LoweCasing Text

Lowercasing text in NLP preprocessing involves converting all letters in a text to lowercase. This step is essential for standardizing text data because it treats words with different cases (e.g., "Word" and "word") as the same, reducing vocabulary size and improving model efficiency. It ensures consistency in word representations, making it easier for algorithms to recognize patterns and associations. For example, "The" and "the" are treated as identical after lowercasing. This normalization simplifies subsequent processing steps, such as tokenization and feature extraction, leading to more accurate and robust NLP models.

```
# Pick any random Review
df['review'][3]
```

```
'@lapcat Need to send 'em to my accountant tomorrow. Oddly, I wasn't even referring to my taxes. Those are supporting evidence, though.
```

if we have a Single Text or Sentence we can Lowercase it by using lower() func of Python.

```
# Lower Casing the review
df['review'][3].lower()
```

```
'@lapcat need to send 'em to my accountant tomorrow. oddly, i wasn't even referring to my taxes. those are supporting evidence, though.
```

We can also Lowercase the Whole Corpus by using lower() function of Python.

```
"""
In pandas, df['review'].str is the string accessor.
It allows you to apply vectorized string functions on an entire column of text (without writing a loop)
"""

df['review'] = df['review'].str.lower()
df.head()
```

	review	Sentiment
0	just had a real good moment. i missssssssss hi...	0
1	is reading manga http://plurk.com/p/mzp1e	0
2	@comeagainjen http://twitpic.com/2y2lx - http:...	0
3	@lapcat need to send 'em to my accountant tomo...	0
4	add me on myspace!!! myspace.com/lookthunder	0

Next steps: Generate code with df View recommended plots New interactive sheet

Now we see all the sentences in the corpus are in lowercase.

2. Remove HTML Tags

Removing HTML tags is an essential step in NLP text preprocessing to ensure that only meaningful textual content is analyzed. HTML tags contain formatting information and metadata irrelevant to linguistic analysis. Including these tags can introduce noise and distort the analysis results. Removing HTML tags helps to extract pure textual data, making it easier to focus on the actual content of the text. This step is particularly crucial when dealing with web data or documents containing HTML markup, as it ensures that the extracted text accurately represents the intended linguistic information for NLP tasks.

We can simply remove HTML tags by using the Regular Expressions.

🔷 What is a Regular Expression (Regex)?

A **regular expression** (often written as *regex* or *regexp*) is a sequence of characters that defines a **search pattern**. It's mainly used for **pattern matching** in text: finding, extracting, replacing, or validating strings.

Think of it as a **search rule language** for text.

🔷 Why use Regex in NLP / text processing?

- Clean text (remove special symbols, hashtags, URLs, etc.).
- Tokenize or split sentences/words.
- Validate formats (like emails, phone numbers, etc.).
- Find specific patterns (dates, hashtags, mentions).

🔷 Basic Regex Elements

Pattern	Meaning	Example Match
`.`	Any character (except newline)	`c.t` → matches "cat", "cut", "c8t"
`^`	Start of string	`^Hello` → matches if text starts with "Hello"
`$`	End of string	`end$` → matches if text ends with "end"
`*`	0 or more repetitions	`ab*` → "a", "ab", "abb", "abbb"
`+`	1 or more repetitions	`ab+` → "ab", "abb", but not "a"
`?`	0 or 1 occurrence	`colou?r` → "color" or "colour"
`[]`	Match any character inside	`[aeiou]` → matches vowels
`[^ ]`	Match anything **except** inside	`[^0-9]` → not a digit
`{m,n}`	Repeat between m and n times	`\d{2,4}` → 2 to 4 digits
`\d`	Digit (0–9)	`\d\d` → "23", "99"
`\w`	Word char (letters, digits, underscore)	`\w+` → "hello123"
`\s`	Whitespace (space, tab, newline)	`\s+` → spaces

🔷 Example in Python

```
import re

text = "My email is example123@gmail.com and my phone is 123-456-7890."

# Find an email
email = re.findall(r"[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}", text)
print(email)  # ['example123@gmail.com']

# Find all numbers
numbers = re.findall(r"\d+", text)
print(numbers)  # ['123', '456', '7890']
```

👉 In short: **Regex is a powerful text filter** that helps in searching and cleaning text in NLP projects.

```
# Import Regular Expression
import re

# Function to remove HTML Tags
```

```
def remove_html_tags(text):
    """This creates a regex pattern that matches any text enclosed in angle brackets (< >),
    which are typical for HTML tags. The .*? part ensures that the match is non-greedy,
    meaning it will match the smallest possible text between the angle brackets."""
    pattern = re.compile('<.*?>')
    return pattern.sub(r'', text)
```

```
# Suppose we have a text Which Contains HTML Tags
text = "<html><body><p> Movie 1</p><p> Actor - Aamir Khan</p><p> Click here to <a href='http://google.com'>download</a></p></body></html>"
text
```

```
'<html><body><p> Movie 1</p><p> Actor - Aamir Khan</p><p> Click here to <a href='http://google.com'>download</a></p></body></html>'
```

```
# Apply Function to Remove HTML Tags.
remove_html_tags(text)
```

```
' Movie 1 Actor - Aamir Khan Click here to download'
```

See How the Code perform well and clean the text from the HTML Tags , We can Also Apply this Function to Whole Corpus.

```
# Apply Function to Remove HTML Tags in our Dataset Colum Review.
df['review'] = df['review'].apply(remove_html_tags)
```

```
df
```

	review	Sentiment
0	just had a real good moment. i missssssssss hi...	0
1	is reading manga http://plurk.com/p/mzp1e	0
2	@comeagainjen http://twitpic.com/2y2lx - http:...	0
3	@lapcat need to send 'em to my accountant tomo...	0
4	add me on myspace!!! myspace.com/lookthunder	0
...	...	...
10309	no depression by g herbo is my mood from now o...	1
10310	what do you do when depression succumbs the br...	1
10311	ketamine nasal spray shows promise against dep...	1
10312	dont mistake a bad day with depression! everyo...	1
10313	0	1

10314 rows × 2 columns

Next steps: Generate code with df View recommended plots New interactive sheet

3. Remove URLs

In NLP text preprocessing, removing URLs is essential to eliminate irrelevant information that doesn't contribute to linguistic analysis. URLs contain website addresses, hyperlinks, and other web-specific elements that can skew the analysis and confuse machine learning models. By removing URLs, the focus remains on the textual content relevant to the task at hand, enhancing the accuracy of NLP tasks such as sentiment analysis, text classification, and information extraction. This step streamlines the dataset, reduces noise, and ensures that the model's attention is directed towards meaningful linguistic patterns and structures within the text.

```
# Here We also Use Regular Expressions to Remove URLs from Text or Whole Corpus.
def remove_url(text):
    """https?://: This part matches URLs that start with http:// or https://.
    The s? makes the s optional, so it will match both http and https.
    \S+: This matches one or more non-whitespace characters, which typically form
    the rest of the URL after the http:// or https://.
    |: This is the OR operator in regular expressions. It allows matching either
    the pattern before it or the pattern after it.
    www\.\S+: This part matches URLs that start with www..
    The \. is used to escape the dot, ensuring it matches a literal dot. Again, \S+ matches the rest of the URL."""
```

```
    pattern = re.compile(r'https?://\S+|www\.\S+')
    return pattern.sub(r'', text)
```

```
<>:5: SyntaxWarning: invalid escape sequence '\S'
<>:5: SyntaxWarning: invalid escape sequence '\S'
/tmp/ipython-input-3177123758.py:5: SyntaxWarning: invalid escape sequence '\S'
  \S+: This matches one or more non-whitespace characters, which typically form
```

```
df['review'] = df['review'].apply(remove_url)
```

```
df
```

	review	Sentiment
0	just had a real good moment. i missssssssss hi...	0
1	is reading manga	0
2	@comeagainjen -	0
3	@lapcat need to send 'em to my accountant tomo...	0
4	add me on myspace!!! myspace.com/lookthunder	0
...	...	...
10309	no depression by g herbo is my mood from now o...	1
10310	what do you do when depression succumbs the br...	1
10311	ketamine nasal spray shows promise against dep...	1
10312	dont mistake a bad day with depression! everyo...	1
10313	0	1

10314 rows × 2 columns

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
# Suppose we have the FOllowings Text With URL.
text1 = 'Check out my notebook https://www.kaggle.com/campusx/notebook8223fc1abb'
text2 = 'Check out my notebook http://www.kaggle.com/campusx/notebook8223fc1abb'
text3 = 'Google search here www.google.com'
text4 = 'For notebook click https://www.kaggle.com/campusx/notebook8223fc1abb to search check www.google.com'
```

```
# Lets Remove The URL by Calling Function
print(remove_url(text1))
print(remove_url(text2))
print(remove_url(text3))
print(remove_url(text4))
```

```
Check out my notebook
Check out my notebook
Google search here
For notebook click  to search check
```

Here How the function beatuifully remove the URLs from the Text . We Can Simply Call this Function on Whole Corpus to Remove URLs.

4. Remove Punctuations

Removing punctuation marks is essential in NLP text preprocessing to enhance the accuracy and efficiency of analysis. Punctuation marks like commas, periods, and quotation marks carry little semantic meaning and can introduce noise into the dataset. By removing them, the text becomes cleaner and more uniform, making it easier for machine learning models to extract meaningful features and patterns. Additionally, removing punctuation aids in standardizing the text, ensuring consistency across documents and improving the overall performance of NLP tasks such as sentiment analysis, text classification, and named entity recognition.

```
# From String we Imorts Punctuation.
import string
string.punctuation
```

```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

```
# Storing Punctuation in a Variable
punc = string.punctuation
```

```
"""
str.maketrans(x, y, z)
x → string of characters to be replaced.

y → string of characters to replace them with. (must be the same length as x)

z → string of characters to delete
"""
```

```
'\nstr.maketrans(x, y, z)\nx → string of characters to be replaced.\n\ny → string of characters to replace them with. (must be the same
length as x)\n\nz → string of characters to delete\n'
```

```
# The code defines a function, remove_punc1, that takes a text input and removes all punctuation characters from it using
# the translate method with a translation table created by str.maketrans. This function effectively cleanses the text of punctuation symbols
def remove_punc(text):
    return text.translate(str.maketrans('', '', punc))
```

```
# Text With Punctuation.
text = "The quick brown fox jumps over the lazy dog. However, the dog doesn't seem impressed! Oh no, it just yawned. How disappointing! Mayb
text
```

```
'The quick brown fox jumps over the lazy dog. However, the dog doesn't seem impressed! Oh no, it just yawned. How disappointing! Maybe
a squirrel would elicit a reaction. Alas, the fox is out of luck.'
```

```
# Remove Punctuation.
remove_punc(text)
```

```
'The quick brown fox jumps over the lazy dog However the dog doesnt seem impressed Oh no it just yawned How disappointing Maybe a squir
rel would elicit a reaction Alas the fox is out of luck'
```

```
df['review'] = df['review'].apply(remove_punc)
df
```

	review	Sentiment
0	just had a real good moment i missssssssss him...	0
1	is reading manga	0
2	comeagainjen	0
3	lapcat need to send em to my accountant tomorr...	0
4	add me on myspace myspacecomlookthunder	0
...	...	...
10309	no depression by g herbo is my mood from now o...	1
10310	what do you do when depression succumbs the br...	1
10311	ketamine nasal spray shows promise against dep...	1
10312	dont mistake a bad day with depression everyon...	1
10313	0	1

10314 rows × 2 columns

Next steps: Generate code with df | View recommended plots | New interactive sheet

Hence the function removes the punctuations from the text and we can also use this function to remove the punctuations from the corpus.

```
# Exmaple on whole Dataset.
print(df['review'][9])

# Remove Punctuation
remove_punc(df['review'][9])
```

```
dananner night darlin  sweet dreams to you
'dananner night darlin  sweet dreams to you '
```

5. Handling ChatWords

Handling ChatWords, also known as internet slang or informal language used in online communication, is important in NLP text preprocessing to ensure accurate analysis and understanding of text data. By converting ChatWords into their standard English equivalents or formal language equivalents, NLP models can effectively interpret the meaning of the text. This preprocessing step helps in maintaining consistency, improving the quality of input data, and enhancing the performance of NLP tasks such as sentiment analysis, chatbots, and information retrieval systems. Ultimately, handling ChatWords ensures better comprehension and more reliable results in NLP applications.

```
# Here Come ChatWords Which i Get from a Github Repository
# Repository Link : https://github.com/rishabhverma17/sms_slang_translator/blob/master/slang.txt
chat_words = {
    "AFAIK": "As Far As I Know",
    "AFK": "Away From Keyboard",
    "ASAP": "As Soon As Possible",
    "ATK": "At The Keyboard",
    "ATM": "At The Moment",
    "A3": "Anytime, Anywhere, Anyplace",
    "BAK": "Back At Keyboard",
    "BBL": "Be Back Later",
    "BBS": "Be Back Soon",
    "BFN": "Bye For Now",
    "B4N": "Bye For Now",
    "BRB": "Be Right Back",
    "BRT": "Be Right There",
    "BTW": "By The Way",
    "B4": "Before",
    "B4N": "Bye For Now",
    "CU": "See You",
    "CUL8R": "See You Later",
    "CYA": "See You",
    "FAQ": "Frequently Asked Questions",
    "FC": "Fingers Crossed",
    "FWIW": "For What It's Worth",
    "FYI": "For Your Information",
    "GAL": "Get A Life",
    "GG": "Good Game",
    "GN": "Good Night",
    "GMTA": "Great Minds Think Alike",
    "GR8": "Great!",
    "G9": "Genius",
    "IC": "I See",
    "ICQ": "I Seek you (also a chat program)",
    "ILU": "ILU: I Love You",
    "IMHO": "In My Honest/Humble Opinion",
    "IMO": "In My Opinion",
    "IOW": "In Other Words",
    "IRL": "In Real Life",
    "KISS": "Keep It Simple, Stupid",
    "LDR": "Long Distance Relationship",
    "LMAO": "Laugh My A.. Off",
    "LOL": "Laughing Out Loud",
    "LTNS": "Long Time No See",
    "L8R": "Later",
    "MTE": "My Thoughts Exactly",
    "M8": "Mate",
    "NRN": "No Reply Necessary",
    "OIC": "Oh I See",
    "PITA": "Pain In The A..",
    "PRT": "Party",
    "PRW": "Parents Are Watching",
    "QPSA?": "Que Pasa?",
    "ROFL": "Rolling On The Floor Laughing",
    "ROFLOL": "Rolling On The Floor Laughing Out Loud",
    "ROTFLMAO": "Rolling On The Floor Laughing My A.. Off",
    "SK8": "Skate",
    "STATS": "Your sex and age",
    "ASL": "Age, Sex, Location",
    "THX": "Thank You",
    "TTFN": "Ta-Ta For Now!",
    "TTYL": "Talk To You Later",
    "U": "You",
    "U2": "You Too",
    "U4E": "Yours For Ever",
```

```
    "WB": "Welcome Back",
    "WTF": "What The F...",
    "WTG": "Way To Go!",
    "WUF": "Where Are You From?",
    "W8": "Wait...",
    "7K": "Sick:-D Laugher",
    "TFW": "That feeling when",
    "MFW": "My face when",
    "MRW": "My reaction when",
    "IFYP": "I feel your pain",
    "TNTL": "Trying not to laugh",
    "JK": "Just kidding",
    "IDC": "I don't care",
    "ILY": "I love you",
    "IMU": "I miss you",
    "ADIH": "Another day in hell",
    "ZZZ": "Sleeping, bored, tired",
    "WYWH": "Wish you were here",
    "TIME": "Tears in my eyes",
    "BAE": "Before anyone else",
    "FIMH": "Forever in my heart",
    "BSAAW": "Big smile and a wink",
    "BWL": "Bursting with laughter",
    "BFF": "Best friends forever",
    "CSL": "Can't stop laughing"
}
```

The code defines a function, chat_conversion, that replaces text with their corresponding chat acronyms from a predefined dictionary. It iterates through each word in the input text, checks if it exists in the dictionary, and replaces it if found. The modified text is then returned.

```
# Function
def chat_conversion(text):
    new_text = []
    for i in text.split():
        if i.upper() in chat_words:
            new_text.append(chat_words[i.upper()])
        else:
            new_text.append(i)
    return " ".join(new_text)
```

```
# Text
text = 'IMHO he is the best'
text1 = 'FYI Islamabad is the capital of Pakistan'
# Calling function
print(chat_conversion(text))
print(chat_conversion(text1))
```

```
In My Honest/Humble Opinion he is the best
For Your Information Islamabad is the capital of Pakistan
```

```
df['review'] = df['review'].apply(chat_conversion)
df
```

	review	Sentiment
0	just had a real good moment i missssssssss him...	0
1	is reading manga	0
2	comeagainjen	0
3	lapcat need to send em to my accountant tomorr...	0
4	add me on myspace myspacecomlookthunder	0
...	...	...
10309	no depression by g herbo is my mood from now o...	1
10310	what do you do when depression succumbs the br...	1
10311	ketamine nasal spray shows promise against dep...	1
10312	dont mistake a bad day with depression everyon...	1
10313	0	1

10314 rows × 2 columns

Next steps: Generate code with df View recommended plots New interactive sheet

Well this is how we Handle ChatWords in Our Data Simple u have to call the above Function.

6. Spelling Correction

Spelling correction is a crucial aspect of NLP text preprocessing to enhance data quality and improve model performance. It addresses errors in text caused by typographical mistakes, irregularities, or variations in spelling. Correcting spelling errors ensures consistency and accuracy in the dataset, reducing ambiguity and improving the reliability of NLP tasks like sentiment analysis, machine translation, and information retrieval. By standardizing spelling across the dataset, models can better understand and process text, leading to more precise and reliable results in natural language processing applications.

```
# Import this Library to Handle the Spelling Issue.
from textblob import TextBlob
```

```
from textblob import TextBlob

# Text with spelling mistakes
text = "I havv a speling error"
blob = TextBlob(text)

# Correct spelling
corrected_text = blob.correct()
print(corrected_text)
```

```
I have a spelling error
```

```
# Incorrect text
incorrect_text = 'ceertain conditionas duriing seveal ggenerations aree moodified in the saame maner.'
print(incorrect_text)
# Text 2
incorrect_text2 = 'The cat sat on the cuchion. while plyaiing'
# Calling function
textBlb = TextBlob(incorrect_text)
textBlb1 = TextBlob(incorrect_text2)
# Corrected Text
print(textBlb.correct().string)
print("======================================")
print(incorrect_text2)
print("======================================")
print(textBlb1.correct().string)
```

```
ceertain conditionas duriing seveal ggenerations aree moodified in the saame maner.
certain conditions during several generations are modified in the same manner.
======================================
The cat sat on the cuchion. while plyaiing
======================================
```

```
The cat sat on the cushion. while playing
```

```
def correct_text(text):
    textBlb = TextBlob(text)
    return textBlb.correct().string
```

```
correct_text(incorrect_text2)
```

```
'The cat sat on the cushion. while playing'
```

```
# df['review'] = df['review'].apply(correct_text)
df
```

	review	Sentiment
0	just had a real good moment i missssssssss him...	0
1	is reading manga	0
2	comeagainjen	0
3	lapcat need to send em to my accountant tomorr...	0
4	add me on myspace myspacecomlookthunder	0
...	...	...
10309	no depression by g herbo is my mood from now o...	1
10310	what do you do when depression succumbs the br...	1
10311	ketamine nasal spray shows promise against dep...	1
10312	dont mistake a bad day with depression everyon...	1
10313	0	1

10314 rows × 2 columns

Next steps: Generate code with df View recommended plots New interactive sheet

Well The Library is Doing Great Job and Handling the Spelling Mistakes , Well u can Use the same Process to Handle the Full corpus.

7. Handling StopWords

In NLP text preprocessing, removing stop words is crucial to enhance the quality and efficiency of analysis. Stop words are common words like "the," "is," and "and," which appear frequently in text but carry little semantic meaning. By eliminating stop words, we reduce noise in the data, decrease the dimensionality of the dataset, and improve the accuracy of NLP tasks such as sentiment analysis, topic modeling, and text classification. This process streamlines the analysis by focusing on the significant words that carry more meaningful information, leading to better model performance and interpretation of results.

```
# We use NLTK library to remove Stopwords.
from nltk.corpus import stopwords
import nltk
```

```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

```
# Here we can see all the stopwords in English.However we can chose different Languages also like spanish etc.
stopword = stopwords.words('english')
```

```
print(stopword[:20])
```

```
['a', 'about', 'above', 'after', 'again', 'against', 'ain', 'all', 'am', 'an', 'and', 'any', 'are', 'aren', "aren't", 'as', 'at', 'be',
```

The code defines a function, remove_stopwords, which removes stopwords from a given text. It iterates through each word in the text, checks if it is a stopword, and appends it to a new list if it is not. Then, it clears the original list, returns the modified text.

```
# Function
def remove_stopwords(text):
    new_text = []
    for word in text.split():
        if word in stopword:
            new_text.append('')
        else:
            new_text.append(word)
    x = new_text[:]
    new_text.clear()
    return " ".join(x)
```

```
# Text
text = 'probably my all-time favorite movie, a story of selflessness, sacrifice and dedication to a noble cause, but it\'s not preachy or bc
print(f'Text With Stop Words :{text}')
# Calling Function
remove_stopwords(text)
```

```
Text With Stop Words :probably my all-time favorite movie, a story of selflessness, sacrifice and dedication to a noble cause, but it's
'probably  all-time favorite movie,  story  selflessness, sacrifice  dedication   noble cause,    preachy  boring.   never gets old, de
spite   seen   15   times'
```

```
# We can Apply the same Function on Whole Corpus also
df['review'] = df['review'].apply(remove_stopwords)
```

```
df.head()
```

	review	Sentiment
0	real good moment missssssssss much	0
1	reading manga	0
2	comeagainjen	0
3	lapcat need send em accountant tomorrow odd...	0
4	add myspace myspacecomlookthunder	0

Next steps: Generate code with df | View recommended plots | New interactive sheet

Well This the function use to handle stopwords in Text.

8. Handling Emojies

Handling emojis in NLP text preprocessing is essential for several reasons. Emojis convey valuable information about sentiment, emotion, and context in text data, especially in informal communication channels like social media. However, they pose challenges for NLP algorithms due to their non-textual nature. Preprocessing involves converting emojis into meaningful representations, such as replacing them with textual descriptions or mapping them to specific sentiment categories. By handling emojis effectively, NLP models can accurately interpret and analyze text data, leading to improved performance in sentiment analysis, emotion detection, and other NLP tasks.

8.1 Simply Remove Emojis

The code defines a function, remove_emoji, which uses a regular expression to match and remove all emojis from a given text string. It targets various Unicode ranges corresponding to different categories of emojis and replaces them with an empty string, effectively removing them from the text.

```
import re

def remove_emoji(text):
    # Define a regular expression pattern to match emojis
    emoji_pattern = re.compile("["
                           u"\U0001F600-\U0001F64F"  # Match emoticons (e.g., 😀 😁 😂)
                           u"\U0001F300-\U0001F5FF"  # Match symbols & pictographs (e.g., 🌍 💥 💡)
                           u"\U0001F680-\U0001F6FF"  # Match transport & map symbols (e.g., 🚀 🚘 🗺)
                           u"\U0001F1E0-\U0001F1FF"  # Match flags (iOS) (e.g., US GB CA)
```

```
                           u"\U00002702-\U000027B0"  # Match miscellaneous symbols (e.g., ✂ ✈ ➖)
                           u"\U000024C2-\U0001F251"  # Match additional symbols (e.g., Ⓜ 🉐 🉑)
                           "]+", flags=re.UNICODE)    # Enable the Unicode flag to correctly interpret the above ranges

    # Substitute all matched emojis in the text with an empty string (i.e., remove them)
    return emoji_pattern.sub(r'', text)
```

```
# Texts
text = "Loved the movie. It was 😘"
text1 = 'Python is 🔥'
print(text ,'\n', text1)

# Remove Emojies using Fucntion
print(remove_emoji(text))
remove_emoji(text1)
```

```
Loved the movie. It was 😘
 Python is 🔥
Loved the movie. It was
'Python is '
```

Well the fucntion is removing the emojies easily.

```
df['review'] = df['review'].apply(remove_emoji)
```

8.2 Simply Convert Emojis into text

```
!pip install emoji
```

```
Collecting emoji
  Downloading emoji-2.14.1-py3-none-any.whl.metadata (5.7 kB)
Downloading emoji-2.14.1-py3-none-any.whl (590 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 590.6/590.6 kB 9.8 MB/s eta 0:00:00
Installing collected packages: emoji
Successfully installed emoji-2.14.1
```

```
# We will USe the Emoji Libray to handle this task
# Pip Install emoji
import emoji
```

```
# Calling the Emoji tool Demojize.
print(emoji.demojize(text))
print(emoji.demojize(text1))
```

```
Loved the movie. It was :face_blowing_a_kiss:
Python is :fire:
```

Well this is the output , and the tool is working best.

9. Tokenization

Tokenization is a crucial step in NLP text preprocessing where text is segmented into smaller units, typically words or subwords, known as tokens. This process is essential for several reasons. Firstly, it breaks down the text into manageable units for analysis and processing. Secondly, it standardizes the representation of words, enabling consistency in language modeling tasks. Additionally, tokenization forms the basis for feature extraction and modeling in NLP, facilitating tasks such as sentiment analysis, named entity recognition, and machine translation. Overall, tokenization plays a fundamental role in preparing text data for further analysis and modeling in NLP applications.

We Generally do 2 Type of tokenization 1. Word tokenization 2. Sentence Tokenization

9.1 NLTK

NLTK is a Library used to tokenize text into sentences and words.

```
# Import Libraray
from nltk.tokenize import word_tokenize, sent_tokenize
```

```
nltk.download("punkt")       # main tokenizer model
nltk.download("punkt_tab")   # newer variant sometimes required
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
True
```

```
# Text
sentence = 'I am going to visit delhi!'
# Calling tool
word_tokenize(sentence)
```

```
['I', 'am', 'going', 'to', 'visit', 'delhi', '!']
```

```
# Whole text Containing 2 or more Sentences
text = """Lorem Ipsum is simply dummy text of the printing and typesetting industry?
Lorem Ipsum has been the industry's standard dummy text ever since the 1500s,
when an unknown printer took a galley of type and scrambled it to make a type specimen book."""

# Sentence Based Tokenization
sent_tokenize(text)
```

```
['Lorem Ipsum is simply dummy text of the printing and typesetting industry?',
 "Lorem Ipsum has been the industry's standard dummy text ever since the 1500s,\nwhen an unknown printer took a galley of type and
scrambled it to make a type specimen book."]
```

```
# Some Sentences
sent5 = 'I have a Ph.D in A.I'
sent6 = "We're here to help! mail us at nks@gmail.com"
sent7 = 'A 5km ride cost $10.50'

# Word Tokenize the Sentences
print(word_tokenize(sent5))
print(word_tokenize(sent6))
print(word_tokenize(sent7))
```

```
['I', 'have', 'a', 'Ph.D', 'in', 'A.I']
['We', "'re", 'here', 'to', 'help', '!', 'mail', 'us', 'at', 'nks', '@', 'gmail.com']
['A', '5km', 'ride', 'cost', '$', '10.50']
```

NLTK is Performing Well Altough it has some of issue , Like in above text u see it cannot handle the mail. But U can Use it Acording to the data problem

9.1 Spacy

Spacy is a Library used to tokenize text into sentences and words.

```
# Installation
# conda install -c conda-forge spacy
# conda install -c conda-forge spacy-model-en_core_web_sm
```

```
# This code imports the Spacy library and loads the English language model 'en_core_web_sm' for natural language processing.
# Pip install spacy library.
import spacy
nlp = spacy.load('en_core_web_sm')## Load the small English model
```

```
# Tokenize the Sentences in Words
doc1 = nlp(sent5)
doc2 = nlp(sent6)
doc3 = nlp(sent7)
```

```
# Print Token Genrated
for token in doc2:
    print(token.text)
```

```
We
're
here
to
help
!
mail
us
at
nks@gmail.com
```

this tool Handle the mail also , so the choice of best tokenizer tool depend on your problem, u can try both and select the best oen.

10. Stemming

Stemming is a text preprocessing technique in NLP used to reduce words to their root or base form, known as a stem, by removing suffixes. It helps in simplifying the vocabulary and reducing word variations, thereby improving the efficiency of downstream NLP tasks like information retrieval and sentiment analysis. By converting words to their common root, stemming increases the overlap between related words, enhancing the generalization ability of models.

```
# Import PorterStemmer from NLTK Library
from nltk.stem.porter import PorterStemmer
```

```
# Intilize Stemmer
stemmer = PorterStemmer()

# This Function Will Stem Words
def stem_words(text):
    return " ".join([stemmer.stem(word) for word in text.split()])
```

```
# A single Sentence
st = "walk walks walking walked"
# Calling Function
stem_words(st)
```

```
'walk walk walk walk'
```

```
text = """probably my alltime favorite movie a story of selflessness sacrifice and dedication to a noble cause but its not preachy
or boring it just never gets old despite my having seen it some 15 or more times in the last 25 years paul lukas performance brings
 tears to my eyes and bette davis in one of her very few truly sympathetic roles is a delight the kids are as grandma says more like
 dressedup midgets than children but that only makes them more fun to watch and the mothers slow awakening to whats happening in the
 world and under her own roof is believable and startling if i had a dozen thumbs theyd all be up for this movie"""
print(text)

# Calling Function
stem_words(text)
```

```
probably my alltime favorite movie a story of selflessness sacrifice and dedication to a noble cause but its not preachy
or boring it just never gets old despite my having seen it some 15 or more times in the last 25 years paul lukas performance brings
 tears to my eyes and bette davis in one of her very few truly sympathetic roles is a delight the kids are as grandma says more like
 dressedup midgets than children but that only makes them more fun to watch and the mothers slow awakening to whats happening in the
 world and under her own roof is believable and startling if i had a dozen thumbs theyd all be up for this movie
'probabl my alltim favorit movi a stori of selfless sacrific and dedic to a nobl caus but it not preachi or bore it just never get old
despit my have seen it some 15 or more time in the last 25 year paul luka perform bring tear to my eye and bett davi in one of her veri
few truli sympathet role is a delight the kid are as grandma say more like dressedup midget than children but that onli make them more
fun to watch and the mother slow awaken to what happen in the world and under her own roof is believ and startl if i had a dozen thumb
```

Thats How the Stemming will work

However, stemming may sometimes result in the production of non-existent or incorrect words, known as stemming errors, which need to be carefully managed to avoid impacting the accuracy of NLP applications.

11. Lemmatization

Lemmatization is performed in NLP text preprocessing to reduce words to their base or dictionary form (lemma), enhancing consistency and simplifying analysis. Unlike stemming, which truncates words to their root form without considering meaning, lemmatization ensures that words are transformed to their canonical form, considering their part of speech. This process aids in reducing redundancy, improving

text normalization, and enhancing the accuracy of downstream NLP tasks such as sentiment analysis, topic modeling, and information retrieval. Overall, lemmatization contributes to refining text data, facilitating more effective linguistic analysis and machine learning model performance.

- ***The code imports the WordNetLemmatizer from NLTK library and initializes it.***
- ***It defines a sentence and a set of punctuation characters. The sentence is tokenized into words.***
- ***Then, it iterates through each word in the sentence, removing punctuation if present.***
- ***Next, it lemmatizes each word using the WordNetLemmatizer with a specific part-of-speech tag ('v' for verb).***
- ***Finally, it prints each word along with its corresponding lemma after lemmatization, aligning them in a formatted table.***
- ***This process helps to normalize the words in the sentence by reducing them to their base or dictionary form.***

```
import nltk
```

```
nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
True
```

```
nltk.download('wordnet', "nltk_data/")

nltk.download('omw-1.4', "nltk_data/")

nltk.data.path.append('nltk_data/')
```

```
[nltk_data] Downloading package wordnet to nltk_data/...
[nltk_data] Downloading package omw-1.4 to nltk_data/...
```

```
!unzip /usr/share/nltk_data/corpora/wordnet.zip -d /usr/share/nltk_data/corpora/
```

```
unzip:  cannot find or open /usr/share/nltk_data/corpora/wordnet.zip, /usr/share/nltk_data/corpora/wordnet.zip.zip or /usr/share/nltk_da
```

```
# import these modules
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

print("rocks :", lemmatizer.lemmatize("rocks"))
print("corpora :", lemmatizer.lemmatize("corpora"))

# a denotes adjective in "pos"
print("better :", lemmatizer.lemmatize("better", pos="a"))
```

```
rocks : rock
corpora : corpus
better : good
```

```
# Import WordNet Lemmatizer from NLTK
from nltk.stem import WordNetLemmatizer
import nltk

# Initialize the WordNet Lemmatizer
wordnet_lemmatizer = WordNetLemmatizer()

# Example sentence
sentence = "He was running and eating at same time. He has bad habit of swimming after playing long hours in the Sun."

# Define punctuation characters that we want to remove
punctuations = "?:!.,;"

# Step 1: Tokenize the sentence into individual words
# Example: ["He", "was", "running", "and", "eating", ...]
sentence_words = nltk.word_tokenize(sentence)

# Step 2: Remove punctuation tokens from the list
for word in sentence_words:
    if word in punctuations:
        sentence_words.remove(word)

# Step 3: Print each original word and its lemmatized form
print("{0:20}{1:20}".format("Word", "Lemma"))  # header row
```

```
for word in sentence_words:
    # 'pos="v"' tells the lemmatizer to treat the word as a verb
    print("{0:20}{1:20}".format(word, wordnet_lemmatizer.lemmatize(word, pos='v')))
```

```
Word                Lemma
He                  He
was                 be
running             run
and                 and
eating              eat
at                  at
same                same
time                time
He                  He
has                 have
bad                 bad
habit               habit
of                  of
swimming            swim
after               after
playing             play
long                long
hours               hours
in                  in
the                 the
Sun                 Sun
```

```
df['review'] = df['review'].apply(lambda x: wordnet_lemmatizer.lemmatize(x))
df
```

	review	Sentiment
0	real good moment missssssssss much	0
1	reading manga	0
2	comeagainjen	0
3	lapcat need send em accountant tomorrow odd...	0
4	add myspace myspacecomlookthunder	0
...	...	...
10309	depression g herbo mood im done stressi...	1
10310	depression succumbs brain makes feel l...	1
10311	ketamine nasal spray shows promise depression...	1
10312	dont mistake bad day depression everyone em	1
10313	0	1

10314 rows × 2 columns

Next steps: Generate code with df | View recommended plots | New interactive sheet

Well That's how the Lemmatizer Works.One Best Thing of Lemmatization is That, lemmatization ensures that words are transformed to their canonical form, considering their part of speech.However this Process is Slow

Label Encoding

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
le.fit(df['Sentiment'])

df['Sentiment_encoded'] = le.transform(df['Sentiment'])
df.head()
```

	review	Sentiment	Sentiment_encoded
0	real good moment missssssssss much	0	0
1	reading manga	0	0
2	comeagainjen	0	0
3	lapcat need send em accountant tomorrow odd...	0	0
4	add myspace myspacecomlookthunder	0	0

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
# how to define X and y (from the SMS data) for use with COUNTVECTORIZER
x = df['review']
y = df['Sentiment_encoded']

print(len(x), len(y))
```

```
10314 10314
```

```
# Split into train and test sets
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=42)
print(len(x_train), len(y_train))
print(len(x_test), len(y_test))
```

```
7735 7735
2579 2579
```

Count Vectorization

image.png

```
from sklearn.feature_extraction.text import CountVectorizer

# instantiate the vectorizer
vect = CountVectorizer()
vect.fit(x_train)
```

```
▾ CountVectorizer
CountVectorizer()
```

```
vect.get_feature_names_out()
```

```
array(['00', '033654', '040', ..., 'žã', 'žå', 'ˆç'], dtype=object)
```

```
# Use the trained to create a document-term matrix from train and test sets
x_train_dtm = vect.transform(x_train)
x_test_dtm = vect.transform(x_test)
```

```
x_test_dtm.toarray().shape
```

```
(2579, 18123)
```

```
x_test[0]
```

```
'   real good moment missssssssss   much'
```

```
print(x_test_dtm[0].toarray().shape)
```

```
(1, 18123)
```

Start coding or generate with AI.

🔷 Common Parameters in `CountVectorizer`

1. `stop_words`

Removes very common, less meaningful words.

```
from sklearn.feature_extraction.text import CountVectorizer

docs = ["I love data science", "Data is the new oil", "I love coding"]

# Remove common English stop words like "I", "the", "is"
vect = CountVectorizer(stop_words='english')
X = vect.fit_transform(docs)

print(vect.get_feature_names_out())
```

Output:

```
['coding' 'data' 'love' 'new' 'oil' 'science']
```

👉 Words like `"I"`, `"the"`, `"is"` are dropped.

---

2. `ngram_range`

Includes multi-word features (n-grams).

```
vect = CountVectorizer(ngram_range=(1,2))  # unigrams + bigrams
X = vect.fit_transform(docs)

print(vect.get_feature_names_out()[:10])  # first 10 features
```

Output (partial):

```
['coding' 'data' 'data is' 'data science' 'is' 'is the'
 'love' 'love coding' 'love data' 'new']
```

👉 Notice both single words and pairs (`"data science"`, `"love data"`) appear.

---

3. `min_df` and `max_df`

Filter out words that appear **too rarely** or **too often**.

```
docs = ["apple banana apple",
        "banana fruit banana",
        "apple fruit orange"]

# Remove words appearing in less than 2 documents
vect = CountVectorizer(min_df=2)
X = vect.fit_transform(docs)

print(vect.get_feature_names_out())
```

Output:

```
['apple' 'banana' 'fruit']
```

👉 `"orange"` was dropped because it appears only once.

---

4. `max_features`

Keep only the top N most frequent words.

```
vect = CountVectorizer(max_features=3)
X = vect.fit_transform(docs)

print(vect.get_feature_names_out())
```

Output:

```
['apple' 'banana' 'fruit']
```

👉 Only the 3 most common words are kept.

🔷 Example: Combining Parameters

```
vect = CountVectorizer(
    stop_words='english',
    ngram_range=(1,2),
    min_df=2,
    max_df=0.9,
    max_features=100
)
```

This will:

- Remove English stopwords.
- Include unigrams + bigrams.
- Keep terms appearing in **at least 2 documents** but **less than 90%** of documents.
- Restrict vocabulary to the **100 most frequent terms**.

```
Start coding or generate with AI.
```

`CountVectorizer` has a few parameters you should know.

stop_words: Since `CountVectorizer` just counts the occurrences of each word in its vocabulary, extremely common words like `the`, `and`, etc.
will become very important features while they add little meaning to the text. Your model can often be improved if you don't take those words into account. Stop words are just a list of words you don't want to use as features. You can set the parameter `stop_words='english'` to use a built-in list. Alternatively you can set stop_words equal to some custom list. This parameter defaults to None.

ngram_range: An n-gram is just a string of n words in a row. E.g. the sentence 'I am Groot' contains the 2-grams 'I am' and 'am Groot'. The sentence is itself a 3-gram. Set the parameter ngram_range=(a,b) where a is the minimum and b is the maximum size of ngrams you want to include in your features. The default ngram_range is (1,1). In a recent project where I modeled job postings online, I found that including 2-grams as features boosted my model's predictive power significantly. This makes intuitive sense; many job titles such as 'data scientist', 'data engineer', and 'data analyst' are 2 words long.

min_df, max_df: These are the minimum and maximum document frequencies words/n-grams must have to be used as features. If either of these parameters are set to integers, they will be used as bounds on the number of documents each feature must be in to be considered as a feature. If either is set to a float, that number will be interpreted as a frequency rather than a numerical limit. min_df defaults to 1 (int) and max_df defaults to 1.0 (float).

max_features: This parameter is pretty self-explanatory. The CountVectorizer will choose the words/features that occur most frequently to be in its' vocabulary and drop everything else.

You would set these parameters when initializing your CountVectorizer object as shown below.

```
"""
min_df=0.1 → keep words appearing in < 10% documents.

max_df=0.7 → ignore words appearing in >80% of documents.

max_features:
  Limit vocabulary size to the top-100 most frequent words.
"""


vect_tunned = CountVectorizer(stop_words='english', min_df=0.1, max_features=100)
```

```
vect_tunned
```

```
CountVectorizer
CountVectorizer(max_features=100, min_df=0.1, stop_words='english')
```

```
from sklearn.feature_extraction.text import TfidfTransformer

tfidf_transformer = TfidfTransformer()

tfidf_transformer.fit(x_train_dtm)
x_train_tfidf = tfidf_transformer.transform(x_train_dtm)

x_train_tfidf
```

```
<Compressed Sparse Row sparse matrix of dtype 'float64'
	with 66329 stored elements and shape (7735, 18123)>
```

```
x_test_tfidf = tfidf_transformer.transform(x_test_dtm)
```

```
x_train_tfidf.toarray().shape
```

```
(7735, 18123)
```

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
```

```
# Train the model
model = MultinomialNB()
model.fit(x_train_tfidf, y_train)

# Predict and evaluate
predictions = model.predict(x_test_tfidf)
print(f'Accuracy: {accuracy_score(y_test, predictions)}')
```

```
Accuracy: 0.8933695230709577
```

Using Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier
```

```
# Initialize and train the Decision Tree Classifier
tree_model = DecisionTreeClassifier(random_state=42)
tree_model.fit(x_train_tfidf, y_train)

# Predict and evaluate
predictions = tree_model.predict(x_test_tfidf)
print(f'Accuracy: {accuracy_score(y_test, predictions)}')
```

```
Accuracy: 0.9274912756882513
```

Using Dense Model

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
```

```
# Build the Dense Neural Network model
dense_model = Sequential([
    Dense(64, input_shape=(x_train_tfidf.toarray().shape[1],), activation='relu'),
    Dropout(0.5),
    Dense(32, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')  # Use 'softmax' if you have more than 2 classes
])

# Compile the model
```

```
dense_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
dense_model.fit(x_train_tfidf.toarray(), y_train, epochs=10, batch_size=32, validation_split=0.2)
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input_shape`/`input_dim` argumen
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
194/194 ━━━━━━━━━━━━━━━━━━━━ 7s 20ms/step - accuracy: 0.7667 - loss: 0.5690 - val_accuracy: 0.9108 - val_loss: 0.2628
Epoch 2/10
194/194 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.9501 - loss: 0.1772 - val_accuracy: 0.9716 - val_loss: 0.0877
Epoch 3/10
194/194 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9929 - loss: 0.0304 - val_accuracy: 0.9754 - val_loss: 0.0732
Epoch 4/10
194/194 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9984 - loss: 0.0110 - val_accuracy: 0.9780 - val_loss: 0.0687
Epoch 5/10
194/194 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9995 - loss: 0.0066 - val_accuracy: 0.9754 - val_loss: 0.0778
Epoch 6/10
194/194 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9989 - loss: 0.0069 - val_accuracy: 0.9741 - val_loss: 0.0833
Epoch 7/10
194/194 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9992 - loss: 0.0059 - val_accuracy: 0.9754 - val_loss: 0.0784
Epoch 8/10
194/194 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9997 - loss: 0.0031 - val_accuracy: 0.9748 - val_loss: 0.0765
Epoch 9/10
194/194 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9993 - loss: 0.0040 - val_accuracy: 0.9722 - val_loss: 0.0781
Epoch 10/10
194/194 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9999 - loss: 0.0022 - val_accuracy: 0.9722 - val_loss: 0.0799
<keras.src.callbacks.history.History at 0x7bd7a4ab0e00>
```

```
dense_model.evaluate(x_test_tfidf.toarray(), y_test)
```

```
81/81 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.9717 - loss: 0.1229
[0.10449174046516418, 0.9732454419136047]
```

```
# Evaluate the model
y_pred = (dense_model.predict(x_test_tfidf.toarray()) > 0.5).astype("int32")  # Convert probabilities to binary output
print(f'Accuracy: {accuracy_score(y_test, y_pred)}')
```

```
81/81 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
Accuracy: 0.9732454439705313
```

Using Conv 1D

```
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropout
# Build the Conv1D Neural Network model
cnn_model = Sequential([
    Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(x_train_tfidf.toarray().shape[1], 1)),
    MaxPooling1D(pool_size=2),
    Dropout(0.5),
    Flatten(),
    Dense(32, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')  # Use 'softmax' if you have more than 2 classes
])

# Compile the model
cnn_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
cnn_model.fit(x_train_tfidf.toarray(), y_train, epochs=10, batch_size=32, validation_split=0.2)
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
194/194 ━━━━━━━━━━━━━━━━━━━━ 23s 86ms/step - accuracy: 0.7832 - loss: 0.5162 - val_accuracy: 0.9619 - val_loss: 0.1578
Epoch 2/10
194/194 ━━━━━━━━━━━━━━━━━━━━ 12s 61ms/step - accuracy: 0.9624 - loss: 0.1259 - val_accuracy: 0.9761 - val_loss: 0.0666
Epoch 3/10
194/194 ━━━━━━━━━━━━━━━━━━━━ 20s 61ms/step - accuracy: 0.9917 - loss: 0.0375 - val_accuracy: 0.9813 - val_loss: 0.0563
Epoch 4/10
194/194 ━━━━━━━━━━━━━━━━━━━━ 21s 61ms/step - accuracy: 0.9933 - loss: 0.0206 - val_accuracy: 0.9787 - val_loss: 0.0624
Epoch 5/10
194/194 ━━━━━━━━━━━━━━━━━━━━ 12s 60ms/step - accuracy: 0.9985 - loss: 0.0101 - val_accuracy: 0.9800 - val_loss: 0.0589
Epoch 6/10
194/194 ━━━━━━━━━━━━━━━━━━━━ 21s 61ms/step - accuracy: 0.9986 - loss: 0.0089 - val_accuracy: 0.9819 - val_loss: 0.0570
Epoch 7/10
```

```
194/194 ━━━━━━━━━━━━━━━━━━━━ 20s 61ms/step - accuracy: 0.9993 - loss: 0.0040 - val_accuracy: 0.9806 - val_loss: 0.0642
Epoch 8/10
194/194 ━━━━━━━━━━━━━━━━━━━━ 12s 61ms/step - accuracy: 0.9988 - loss: 0.0039 - val_accuracy: 0.9741 - val_loss: 0.0724
Epoch 9/10
194/194 ━━━━━━━━━━━━━━━━━━━━ 21s 61ms/step - accuracy: 0.9991 - loss: 0.0048 - val_accuracy: 0.9825 - val_loss: 0.0617
Epoch 10/10
194/194 ━━━━━━━━━━━━━━━━━━━━ 12s 61ms/step - accuracy: 0.9981 - loss: 0.0044 - val_accuracy: 0.9825 - val_loss: 0.0654
<keras.src.callbacks.history.History at 0x7bd7a0132cc0>
```

```
# Evaluate the model
y_pred = (cnn_model.predict(x_test_tfidf.toarray()) > 0.5).astype("int32")  # Convert probabilities to binary output
print(f'Accuracy: {accuracy_score(y_test, y_pred)}')
```

```
81/81 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step
Accuracy: 0.9763474214811942
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

🍼 Explaining RNN from Scratch (Like to a Baby)

👉 Imagine you're reading a **storybook**:

- Page 1: "Once upon a time..."
- Page 2: "There was a little cat..."
- Page 3: "The cat loved milk..."

Now, to **understand Page 3 properly**, you need to **remember what was on Page 1 and 2**.

If you only look at **one page at a time** and forget the past, you won't understand the story.

That's exactly what happens in **text, speech, or time-series data**. The meaning depends on **previous words/signals**.

🧠 Dense Layer (Baby analogy)

A **dense layer** sees only the input right now. 👉 It's like a baby looking at a single picture flashcard:

- You show the word "cat" 🐱
- Baby says "cat"
- You show "milk" 🥛
- Baby says "milk" But the baby **doesn't connect cat + milk = cats like milk**.

🎨 CNN Layer (Baby analogy)

A **CNN** looks at local patterns, like recognizing shapes in an image. 👉 Example:

- CNN sees whiskers + ears + tail → says "cat".
- Great for **images** where local patterns matter.

But CNN is like a baby that can recognize "cat" in many pictures, but doesn't remember **the sequence of pictures** in a story.

⏳ RNN Layer (Baby analogy)

An **RNN** adds **memory**. 👉 It's like a baby listening to a bedtime story:

- Baby hears "Once upon a time..."
- Then "there was a cat..." (baby remembers "once upon a time")
- Then "the cat loved milk..." (baby uses memory of cat).

So RNNs process things **step by step** while keeping track of **what happened before**.

🛠️ RNN Working (Technical but Simple)

At each time step:

1. Input = current word/signal (e.g., "cat")

2. Hidden state = memory from the past
3. Output = prediction/understanding

Equation:

$$h_t = f(W \cdot x_t + U \cdot h_{t-1} + b)$$

- $x_t \rightarrow$ current input
- $h_{t-1} \rightarrow$ memory from past step
- $h_t \rightarrow$ new memory
- $W, U, b \rightarrow$ weights

So it's like updating memory **every time** something new comes in.

🌟 Why RNN is Needed (Limitations of CNN & Dense)

✅ Dense (Fully Connected) Layers Limitation

- Looks at input independently.
- Cannot handle sequences.
- Example: Predicting the next word in "I am going to the ___" → Dense sees only last word "the", ignores past context.

✅ CNN Limitation

- CNN is good for local patterns ("cat face") but not **long sequences**.
- If you try to use CNN for a story or sentence, it only sees small chunks (n-grams) but not the **whole timeline meaning**.

❌ Both CNN and Dense **forget the past**. ✅ RNN remembers history, so it works for:

- Text (sentences, translation, chatbot)
- Audio (speech recognition)
- Time series (stock, ECG/PPG, weather prediction).

⚠️ But RNN also has problems

- **Vanishing Gradient**: Memory fades after many steps (it forgets long stories).
- **Slow Training**: Processes one step at a time.
- **Better Alternatives**: LSTM, GRU, Transformer.

📌 Summary as Baby Analogy

- Dense = Baby seeing flashcards, but forgets previous card.
- CNN = Baby can recognize a cat in pictures, but forgets story order.
- RNN = Baby listens to a bedtime story step by step and remembers what came before.

Double-click (or enter) to edit

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

Using LSTM

```
from tensorflow.keras.layers import LSTM
```

```
# Build the LSTM Neural Network model
lstm_model = Sequential([
    LSTM(32, input_shape=(x_train_tfidf.toarray().shape[1], 1), return_sequences=False),
    Dropout(0.2),
    Dense(16, activation='relu'),
    Dropout(0.1),
    Dense(1, activation='sigmoid')  # Use 'softmax' if you have more than 2 classes
])

# Compile the model
lstm_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
# Train the model
lstm_model.fit(x_train_tfidf.toarray(), y_train, epochs=10, batch_size=32, validation_split=0.2)
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/rnn/rnn.py:199: UserWarning: Do not pass an `input_shape`/`input_dim` argument
  super().__init__(**kwargs)
Epoch 1/10
194/194 ━━━━━━━━━━━━━━━━━━━━ 93s 454ms/step - accuracy: 0.7699 - loss: 0.5934 - val_accuracy: 0.7796 - val_loss: 0.5397
Epoch 2/10
194/194 ━━━━━━━━━━━━━━━━━━━━ 91s 472ms/step - accuracy: 0.7719 - loss: 0.5448 - val_accuracy: 0.7796 - val_loss: 0.5385
Epoch 3/10
194/194 ━━━━━━━━━━━━━━━━━━━━ 139s 456ms/step - accuracy: 0.7699 - loss: 0.5453 - val_accuracy: 0.7796 - val_loss: 0.5312
Epoch 4/10
194/194 ━━━━━━━━━━━━━━━━━━━━ 141s 452ms/step - accuracy: 0.7756 - loss: 0.5392 - val_accuracy: 0.7796 - val_loss: 0.5276
Epoch 5/10
194/194 ━━━━━━━━━━━━━━━━━━━━ 89s 462ms/step - accuracy: 0.7630 - loss: 0.5518 - val_accuracy: 0.7796 - val_loss: 0.5295
Epoch 6/10
194/194 ━━━━━━━━━━━━━━━━━━━━ 89s 458ms/step - accuracy: 0.7630 - loss: 0.5562 - val_accuracy: 0.7796 - val_loss: 0.5275
Epoch 7/10
194/194 ━━━━━━━━━━━━━━━━━━━━ 142s 458ms/step - accuracy: 0.7730 - loss: 0.5427 - val_accuracy: 0.7796 - val_loss: 0.5279
Epoch 8/10
194/194 ━━━━━━━━━━━━━━━━━━━━ 142s 459ms/step - accuracy: 0.7750 - loss: 0.5352 - val_accuracy: 0.7796 - val_loss: 0.5284
Epoch 9/10
194/194 ━━━━━━━━━━━━━━━━━━━━ 142s 459ms/step - accuracy: 0.7698 - loss: 0.5429 - val_accuracy: 0.7796 - val_loss: 0.5284
Epoch 10/10
194/194 ━━━━━━━━━━━━━━━━━━━━ 142s 461ms/step - accuracy: 0.7770 - loss: 0.5353 - val_accuracy: 0.7796 - val_loss: 0.5302
<keras.src.callbacks.history.History at 0x7bd7a01b1880>
```

```
# Evaluate the model
y_pred = (lstm_model.predict(x_test_tfidf.toarray()) > 0.5).astype("int32")  # Convert probabilities to binary output
print(f'Accuracy: {accuracy_score(y_test, y_pred)}')
```

```
81/81 ━━━━━━━━━━━━━━━━━━━━ 15s 185ms/step
Accuracy: 0.7797595967429236
```

Embedding layer from tensorflow

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout
```

```
# Example set of small text documents
documents = [
    "This is a sample document.",
    "This document is another example.",
    "TF-IDF is a text representation method."
]

# Example labels (like classification targets)
labels = np.array([0, 1, 0])
# Here: 0 and 1 could represent different categories

# Initialize the tokenizer
# oov_token="<OOV>" ensures that any word not seen during training
# will be replaced by the token "<OOV>" instead of being skipped
tokenizer = Tokenizer(oov_token="<OOV>")

# Build the word index (vocabulary) from the documents
# This assigns a unique integer to each unique word
tokenizer.fit_on_texts(documents)

# Get the mapping of words to their unique integer indices
word_index = tokenizer.word_index

# Print the generated word index (vocabulary)
print("Word Index:", word_index)
```

```
Word Index: {'<OOV>': 1, 'is': 2, 'this': 3, 'a': 4, 'document': 5, 'sample': 6, 'another': 7, 'example': 8, 'tf': 9, 'idf': 10, 'text':
```

```
len(word_index.values())
```

```
13
```

```
# Convert texts to sequences of integers
sequences = tokenizer.texts_to_sequences(documents)

# Pad sequences to ensure equal length
padded_sequences = pad_sequences(sequences, padding='post')
```

```
padded_sequences
```

```
array([[ 3,  2,  4,  6,  5,  0,  0],
       [ 3,  5,  2,  7,  8,  0,  0],
       [ 9, 10,  2,  4, 11, 12, 13]], dtype=int32)
```

```
from sklearn.model_selection import train_test_split
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(padded_sequences, labels, test_size=0.3, random_state=42)
```

```
vocab_size = len(word_index) + 1  # +1 for OOV token
vocab_size
```

```
14
```

```
# Import necessary libraries
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout

# Build the model
model = Sequential([

    # 1. Embedding Layer
    # input_dim = size of the vocabulary (number of unique tokens + 1 for OOV)
    # output_dim = size of the embedding vector (how many dimensions to represent each word)
    # Example: if input_dim=5000, each word will be represented by a 16-dimensional vector
    Embedding(input_dim=14, output_dim=16),

    # 2. LSTM Layer (Long Short-Term Memory)
    # 64 units = number of LSTM "memory cells"
    # return_sequences=False → only return the last hidden state (suitable for classification)
    # if return_sequences=True, it would return hidden states for each time step (useful for seq2seq tasks)
    LSTM(64, return_sequences=False),

    # 3. Dropout Layer
    # Dropout randomly "turns off" 50% of neurons during training
    # This prevents overfitting (when model memorizes instead of generalizing)
    Dropout(0.5),

    # 4. Dense Layer (fully connected layer)
    # 32 neurons, activation='relu' (introduces non-linearity and learns features)
    Dense(32, activation='relu'),

    # 5. Another Dropout Layer
    # Again, 50% of neurons are turned off randomly to improve generalization
    Dropout(0.5),

    # 6. Output Layer
    # 1 neuron, activation='sigmoid' (since this is binary classification: 0 or 1)
    # If you have more than 2 classes → use Dense(num_classes, activation='softmax')
    Dense(1, activation='sigmoid')
])

# Compile the model
# optimizer='adam' → popular adaptive optimizer
# loss='binary_crossentropy' → used for binary classification
# metrics=['accuracy'] → track accuracy during training
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Print model summary (to see all layers and number of parameters)
model.summary()
```

```
Model: "sequential_3"
```

Layer (type)	Output Shape	Param #
embedding_3 (Embedding)	?	0 (unbuilt)
lstm_3 (LSTM)	?	0 (unbuilt)
dropout_6 (Dropout)	?	0
dense_6 (Dense)	?	0 (unbuilt)
dropout_7 (Dropout)	?	0
dense_7 (Dense)	?	0 (unbuilt)

```
Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)
```

```
# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32)
```

```
Epoch 1/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step - accuracy: 0.5000 - loss: 0.6925
Epoch 2/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step - accuracy: 0.5000 - loss: 0.6913
Epoch 3/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - accuracy: 0.5000 - loss: 0.6950
Epoch 4/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - accuracy: 0.5000 - loss: 0.6908
Epoch 5/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - accuracy: 1.0000 - loss: 0.6894
Epoch 6/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - accuracy: 0.5000 - loss: 0.6884
Epoch 7/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step - accuracy: 0.5000 - loss: 0.6946
Epoch 8/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step - accuracy: 0.5000 - loss: 0.6919
Epoch 9/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step - accuracy: 1.0000 - loss: 0.6886
Epoch 10/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step - accuracy: 0.5000 - loss: 0.6860
<keras.src.callbacks.history.History at 0x7e053a8003b0>
```

```
model.summary()
```

```
Model: "sequential_3"
```

Layer (type)	Output Shape	Param #
embedding_3 (Embedding)	(None, 7, 16)	224
lstm_3 (LSTM)	(None, 64)	20,736
dropout_6 (Dropout)	(None, 64)	0
dense_6 (Dense)	(None, 32)	2,080
dropout_7 (Dropout)	(None, 32)	0
dense_7 (Dense)	(None, 1)	33

```
Total params: 69,221 (270.40 KB)
Trainable params: 23,073 (90.13 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 46,148 (180.27 KB)
```

```
from sklearn.metrics import accuracy_score
```

```
# Evaluate the model
y_pred = (model.predict(X_test) > 0.5).astype("int32")  # Convert probabilities to binary output
print(f'Accuracy: {accuracy_score(y_test, y_pred)}')
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
Accuracy: 0.0
```

Lets apply on our dataset

```
df
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipython-input-1396537375.py in <cell line: 0>()
----> 1 df

NameError: name 'df' is not defined
```

Next steps: Explain error

```
# Tokenize the text data
tokenizer = Tokenizer(num_words=10000, oov_token="<OOV>")
tokenizer.fit_on_texts(df['review'].values)
word_index = tokenizer.word_index
```

```
word_index
```

```
 'pass': 944,
 'forever': 945,
 'society': 946,
 'keeping': 947,
 'fat': 948,
 'hangover': 949,
 'fake': 950,
 'cooking': 951,
 'relax': 952,
 'happens': 953,
 'final': 954,
 'tickets': 955,
 'close': 956,
 'feelings': 957,
 'pool': 958,
 'healing': 959,
 'child': 960,
 'treat': 961,
 'sadness': 962,
 'medical': 963,
 'cuts': 964,
 'pics': 965,
 'means': 966,
 'figure': 967,
 'pm': 968,
 '25': 969,
 'folks': 970,
 'excellent': 971,
 'relaxing': 972,
 'model': 973,
 'mad': 974,
 'wife': 975,
 'todays': 976,
 'usually': 977,
 'fixed': 978,
 'process': 979,
 'playlist': 980,
 'alot': 981,
 'spending': 982,
 'changed': 983,
 'sir': 984,
 'cover': 985,
 'supposed': 986,
 'opinion': 987,
 'rather': 988,
 'blessed': 989,
 'terrible': 990,
 'played': 991,
 'graduation': 992,
 'note': 993,
 'service': 994,
 'lately': 995,
 'twilight': 996,
 'round': 997,
 'indeed': 998,
 'gives': 999,
 'secret': 1000,
 ...}
```

```
list(word_index.values())[22001] , list(word_index.keys())[20000]
```

```
(22002, 'consequence')
```

```
# Convert texts to sequences of integers
sequences = tokenizer.texts_to_sequences(df['review'].values)
sequences[:10]
```

```
[[114, 4, 424, 6287, 32],
 [389, 6288],
 [3086],
 [6289, 49, 460, 798, 6290, 69, 6291, 378, 75, 1851, 6292, 1486, 1126, 109],
 [318, 580, 6293],
 [1216, 4, 219, 101, 109],
 [6294, 6295, 89, 156, 179, 6296, 64, 243, 209, 1346],
 [1347, 6297, 799, 23, 65],
 [581, 3087, 1127, 6298],
 [6299, 33, 6300, 237, 759]]
```

```
# Pad sequences to ensure equal length
padded_sequences = pad_sequences(sequences, padding='post')
padded_sequences[:1]
```

```
array([[ 114,    4,  424, 6287,   32,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0]],
      dtype=int32)
```

```
padded_sequences.shape
```

```
(10314, 75)
```

```
# Convert labels to numpy array
labels = np.array(df['Sentiment'].values)
labels
```

```
array([0, 0, 0, ..., 1, 1, 1])
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(padded_sequences, labels, test_size=0.3, random_state=42)
```

```
from tensorflow.keras.layers import Flatten
```

```


# Build the model
model = Sequential([
    Embedding(input_dim=10000, output_dim=16),  # Embedding layer
    LSTM(64, return_sequences=True),  # LSTM layer
    Dropout(0.2),
    LSTM(32, return_sequences=True),
    Flatten(),
    Dense(32, activation='relu'),
    Dropout(0.1),
    Dense(1, activation='sigmoid')  # Use 'softmax' if you have more than 2 classes
])
```

```
model.summary()
```

Model: "sequential_4"

Layer (type)	Output Shape	Param #
embedding_5 (Embedding)	?	0 (unbuilt)
lstm_6 (LSTM)	?	0 (unbuilt)
dropout_9 (Dropout)	?	0
dense_9 (Dense)	?	0 (unbuilt)

Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)

```
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)
```

```
Epoch 1/10
181/181 ━━━━━━━━━━━━━━━━━━━━ 6s 17ms/step - accuracy: 0.8291 - loss: 0.4135 - val_accuracy: 0.9924 - val_loss: 0.0369
Epoch 2/10
181/181 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - accuracy: 0.9933 - loss: 0.0331 - val_accuracy: 0.9931 - val_loss: 0.0359
Epoch 3/10
181/181 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - accuracy: 0.9959 - loss: 0.0118 - val_accuracy: 0.9917 - val_loss: 0.0344
Epoch 4/10
181/181 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - accuracy: 0.9969 - loss: 0.0103 - val_accuracy: 0.9875 - val_loss: 0.0476
Epoch 5/10
181/181 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - accuracy: 0.9931 - loss: 0.0305 - val_accuracy: 0.9799 - val_loss: 0.0568
Epoch 6/10
181/181 ━━━━━━━━━━━━━━━━━━━━ 5s 12ms/step - accuracy: 0.9981 - loss: 0.0116 - val_accuracy: 0.9799 - val_loss: 0.0606
Epoch 7/10
```