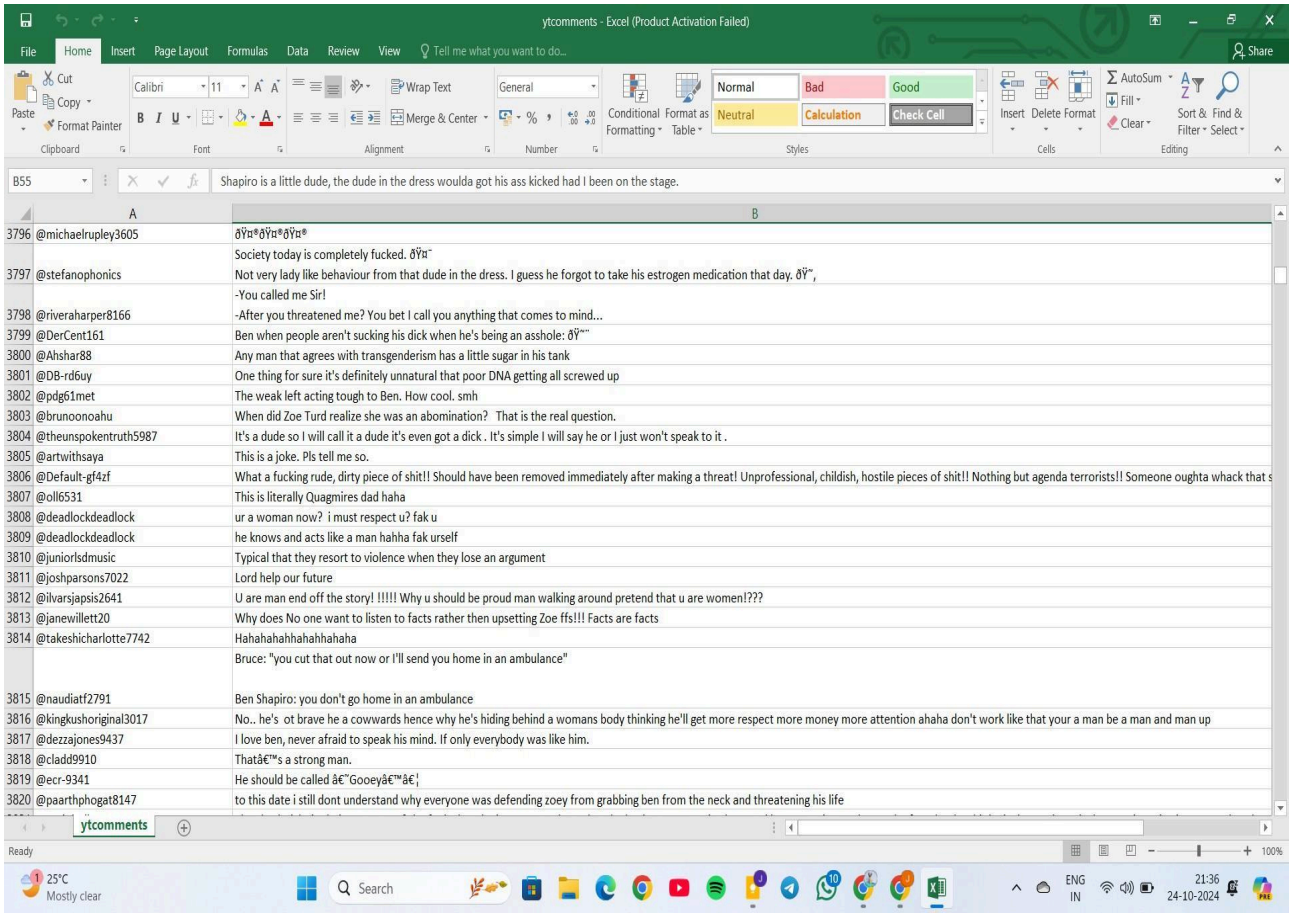
Cyberbullying Detection on Social Networks Using Hybrid RNN - LSTM Model

Milestone 1: Data Collection and Pre - Processing

DATA COLLECTION

1. Collecting comments from different social media platforms : We used YOUTUBE to collect our set of comments i.e. our data.
   Link : https://youtu.be/YgQy70_LPS4?si=aR3qko1M_niA8VAN



	A	B
3796	@michaelrupley3605	ðŸ¤®ðŸ¤®ðŸ¤®
3797	@stefanophonics	Society today is completely fucked. ðŸ¤¯ Not very lady like behaviour from that dude in the dress. I guess he forgot to take his estrogen medication that day. ðŸ˜‚,
3798	@riveraharper8166	-You called me Sir! -After you threatened me? You bet I call you anything that comes to mind...
3799	@DerCent161	Ben when people aren't sucking his dick when he's being an asshole: ðŸ˜‚
3800	@Ahshar88	Any man that agrees with transgenderism has a little sugar in his tank
3801	@DB-rd6uy	One thing for sure it's definitely unnatural that poor DNA getting all screwed up
3802	@pdg61met	The weak left acting tough to Ben. How cool. smh
3803	@brunoonoahu	When did Zoe Turd realize she was an abomination? That is the real question.
3804	@theunspokentruth5987	It's a dude so I will call it a dude it's even got a dick . It's simple I will say he or I just won't speak to it .
3805	@artwithsaya	This is a joke. Pls tell me so.
3806	@Default-gf4zf	What a fucking rude, dirty piece of shit!! Should have been removed immediately after making a threat! Unprofessional, childish, hostile pieces of shit!! Nothing but agenda terrorists!! Someone oughta whack that s
3807	@oll6531	This is literally Quagmires dad haha
3808	@deadlockdeadlock	ur a woman now? i must respect u? fak u
3809	@deadlockdeadlock	he knows and acts like a man hahha fak urself
3810	@juniorlsdmusic	Typical that they resort to violence when they lose an argument
3811	@joshparsons7022	Lord help our future
3812	@ilvarsjapsis2641	U are man end off the story! !!!!! Why u should be proud man walking around pretend that u are women!???
3813	@janewillett20	Why does No one want to listen to facts rather then upsetting Zoe ffs!!! Facts are facts
3814	@takeshicharlotte7742	Hahahahahhahahhahaha
3815	@naudiatf2791	Bruce: "you cut that out now or I'll send you home in an ambulance" Ben Shapiro: you don't go home in an ambulance
3816	@kingkushoriginal3017	No.. he's ot brave he a cowwards hence why he's hiding behind a womans body thinking he'll get more respect more money more attention ahaha don't work like that your a man be a man and man up
3817	@dezzajones9437	I love ben, never afraid to speak his mind. If only everybody was like him.
3818	@cladd9910	Thatâ€™s a strong man.
3819	@ecr-9341	He should be called â€˜Gooeyâ€™â€¦
3820	@paarthphogat8147	to this date i still dont understand why everyone was defending zoey from grabbing ben from the neck and threatening his life

Fig : 1(a)

Figure 1(a) Shows the collections of comments collected using Web Scraping

2. Code for Web Scraping Comments from the above post

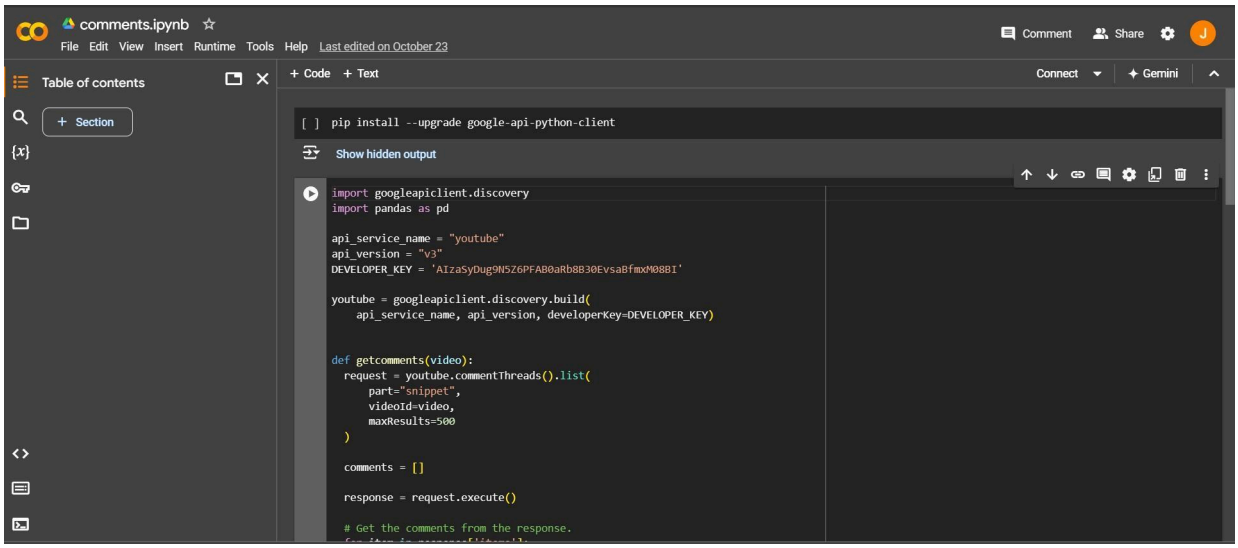


Fig: 1(b)

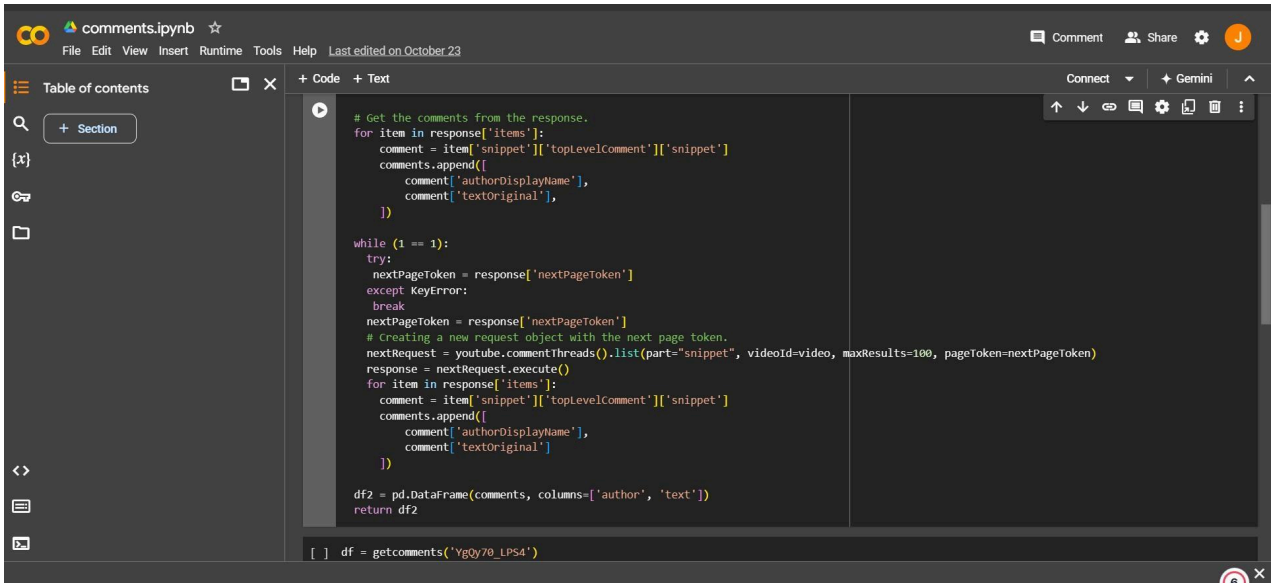


Fig: 1(c)

Figure 1(b) and 1(c) Shows the working code for web scraping of comments from the post.

EXPLANATION OF THE CODE

```
def getcomments(video):
 request = youtube.commentThreads().list(
    part="snippet",
videoId=video,
    maxResults=500
)
```

In this part of the code, we define the function that takes a video Id as an argument.

- request = youtube.commentThreads().list(...): Creates a request to retrieve the comment threads for the specified video.
- part="snippet": Specifies the data fields to return.
- videoId=video: The ID of the video from which to retrieve comments.
- maxResults=500: Sets the maximum number of results to return in one API call (the maximum is usually 100).

```
comments = []
response = request.execute()
```

- comments = []: Initializes an empty list to store the comments.
- response = request.execute(): Executes the request and stores the response.

```
# Get the comments from the response.
 for item in response['items']:
comment = item['snippet']['topLevelComment']['snippet']
    comments.append([
comment['authorDisplayName'],
        comment['textOriginal'],
])
```

The first for loop iterates through the items in the response:

- for item in response['items']:: Loops through the comments in the response.
- comment = item['snippet']['topLevelComment']['snippet']: Extracts the comment details.
- The comment author and text are appended to the comments list as a list of lists.

```
while (1 == 1):
try:
    nextPageToken =
   response['nextPageToken'] except
   KeyError:
break
nextPageToken = response['nextPageToken']
# Creating a new request object with the next page token.
                nextRequest        =        youtube.commentThreads().list(part="snippet",
videoId=video, maxResults=100, pageToken=nextPageToken)
response =
   nextRequest.execute() for item
   in response['items']:
comment = item['snippet']['topLevelComment']['snippet']
     comments.append([
comment['authorDisplayName'],
        comment['textOriginal']
])
```

A while loop is used to handle pagination, allowing retrieval of more comments if available:

- try: and except KeyError:: Checks for the presence of a nextPageToken. If it's not present, it breaks the loop.
- nextRequest = youtube.commentThreads().list(...): Creates a new request for the next page of comments using the nextPageToken.
- The new response is processed similarly to the first one, appending comments to the comments list.

```
df2 = pd.DataFrame(comments, columns=['author', 'text'])

 return df2
```

- df2 = pd.DataFrame(comments, columns=['author', 'text']): Converts the list of comments into a pandas DataFrame with columns for the author's name and comment text.
- return df2: Returns the DataFrame containing all retrieved comments.

```
df =

getcomments('YgQy70_LPS4')

df
```

The line df = getcomments() is calling the getcomments function

The result of the function call which is a pandas DataFrame containing the comments from the YouTube video is assigned to the variable df.

After this, df will hold the DataFrame containing the YouTube comments that the function retrieved.

```
df.head(10)

df.to_csv('ytcomments.csv',index=False)
```

These two lines of code are performing operations on a pandas DataFrame (df) that likely contains YouTube comments (as indicated by your previous context).

df.head(10):

- This line retrieves the first 10 rows of the DataFrame df.
- head(n) is a pandas method that returns the first n rows (in this case, 10) of the DataFrame.

df.to_csv('ytcomments.csv', index=False):

- This line saves the DataFrame df to a CSV (Comma-Separated Values) file named ytcomments.csv.
- The index=False parameter ensures that the DataFrame index (the default integer index) is not written to the CSV file, meaning only the data columns will be saved, not the index.

OUTPUT

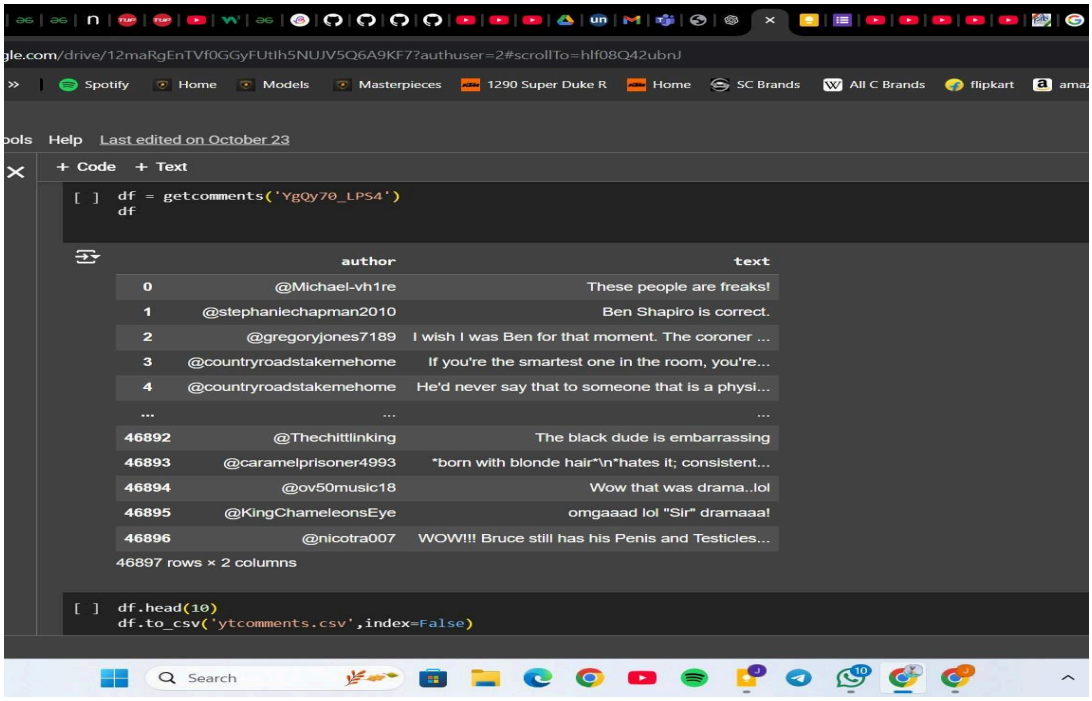


Fig:1(d) Shows the output of the previous code and the result we are required i.e.our raw dataset.

Errors encountered while web scraping:

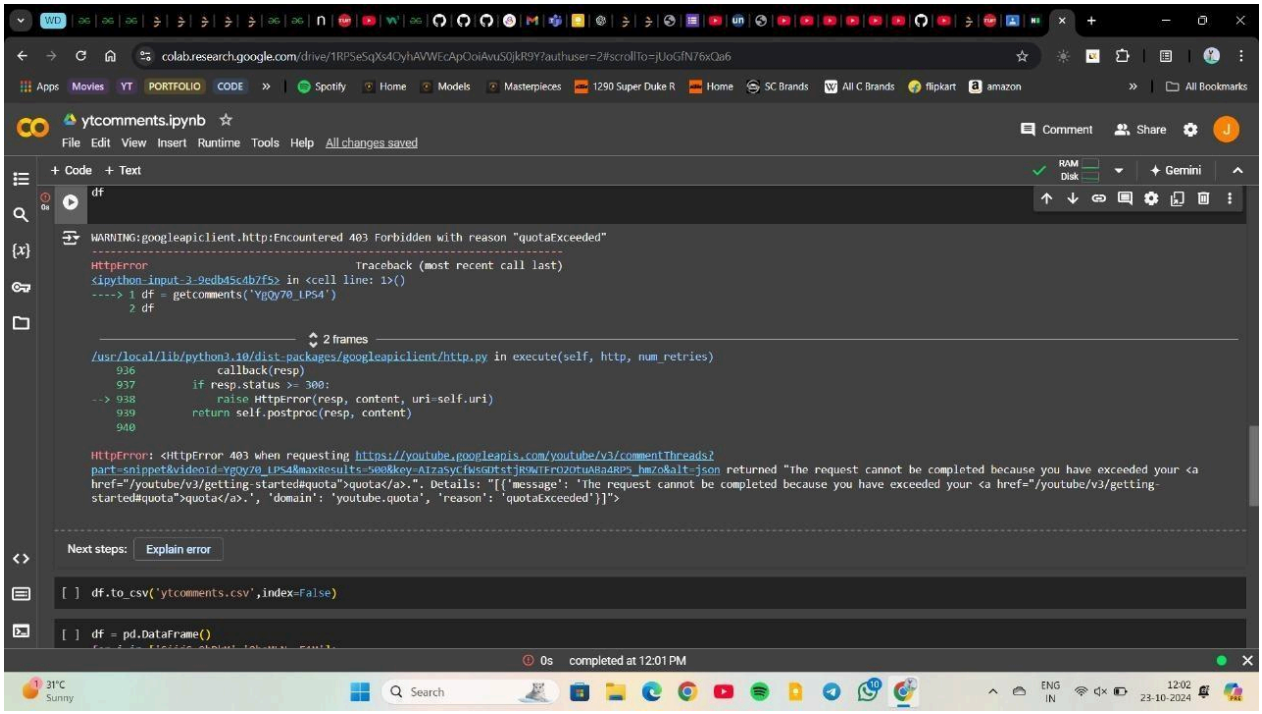


Fig: 1(e) Shows the errors we encountered i.e. ‘quotaExceeded’

Encountered 403 Forbidden with reason “quotaExceeded”.For this we searched on google and found out that for api calls there will be a limited quota which resets every day and if we create a new api key under same account then also we got error we tried to overcome this by logging in with another google account and got a new api key and the code successfully executed.

PRE - PROCESSING

Preprocessing is the process of transforming raw data into a clean, consistent, and usable format for analysis or machine learning. It involves a series of steps that prepare the data by addressing noise, errors, and inconsistencies to enhance the data's quality and relevance. Key tasks in preprocessing include:

1. **Data Cleaning:** Removing or correcting erroneous, duplicated, or incomplete data to reduce inaccuracies. This often involves removing irrelevant information, fixing typos, and filling or handling missing values.
2. **Data Transformation:** Converting data into a format that is more suitable for analysis, such as standardizing text, scaling numeric values, encoding categorical variables, and normalizing data distributions.
3. **Feature Extraction and Selection:** Identifying and retaining only the most relevant features for a specific analysis or model, which improves efficiency and can enhance model accuracy.

- Here are the steps of preprocessing we performed on the data set i.e. comments.
- Our data set has 1000 rows of comments which need to be pre-processed.

1. **File Upload and Data Loading**:

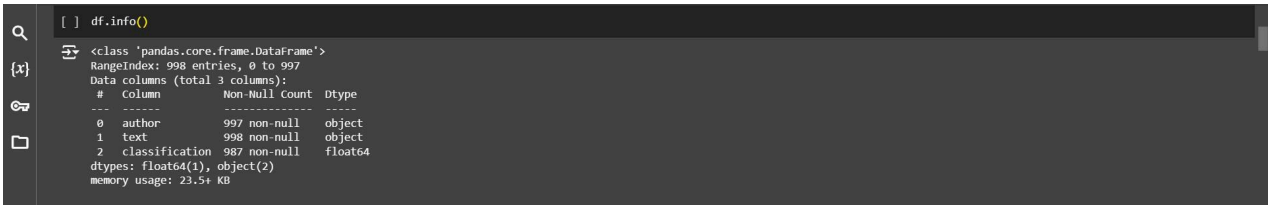


Fig: 1(f) Shows the code and output for file upload and loading the data.

Purpose: Load the data from a file and organize it into a DataFrame for easier manipulation.

Step in Preprocessing: Data ingestion is the first step in a preprocessing pipeline, as it brings in raw data.

Explanation: This code allows file upload in Google Colab and reads the file comments(1000).csv into a DataFrame df using Pandas. The file is assumed to have columns, where one of them is text.

Note: encoding='latin-1' is specified to handle non-standard characters i.e. encoding issues.

2. Creating a New DataFrame Copy df2:

Fig: 1(g) Shows the code and output for creating a new dataframe.

Explanation: This creates a new DataFrame df2 with two columns: comments (original text) and new_comments.

- **comments**: original text from df['text']
- **new_comments**: a copy of df['text'] to be cleaned and transformed.

df2 is used to keep a record of both the original and processed text.

Purpose: This step preserves the original comments for reference.

3. Text Lowercasing:

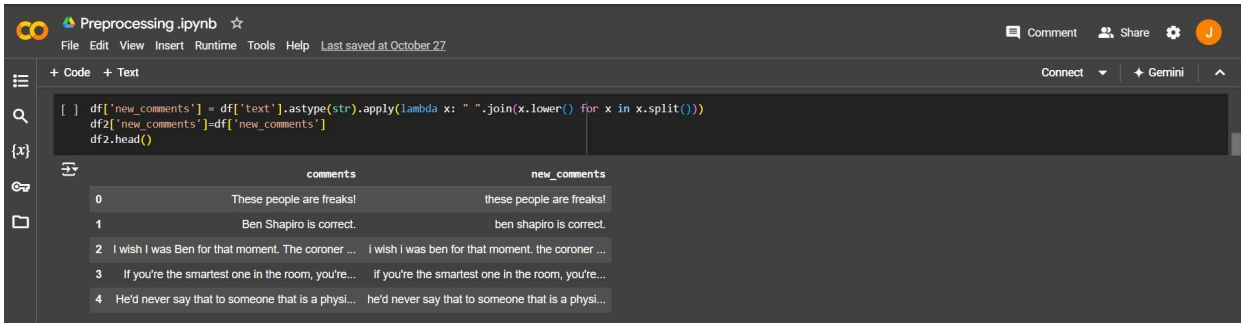


Fig: 1(h) Shows the code and output for lowering the case of the strings in the dataset.

- Converts text to lowercase, making it easier to work with consistently.
- **Tokenization** happens here indirectly: the text is split by whitespace (via split()) and joined back together, giving us lowercase tokens without changing word order.

df['new_comments']:

- This references the new_comments column of the DataFrame df. This column is expected to contain the cleaned text data that may include characters that cannot be directly encoded or decoded using certain character sets.

.apply(lambda x: ...):

- The apply() function is a Pandas method that allows you to apply a function along a specified axis (in this case, each element of the column). Here, it takes a lambda function as an argument.
- lambda x: defines an anonymous function that takes one input, x, which represents an individual entry (a comment) in the new_comments column.

df['text'].astype(str):

- This converts all values in the df['text'] column to strings. It's a precaution to ensure there are no non-string data types, as some operations in the pipeline (like lowercasing or splitting) assume the input is text.

.apply(lambda x: " ".join(x.lower() for x in x.split())):

- **x.split()**: Splits each string into a list of words based on whitespace.
- **x.lower() for x in x.split()**: Converts each word in the split list to lowercase, which helps standardize text by removing case sensitivity (e.g., "Hello" and "hello" become the same).

- **" ".join(...)**: Joins the list of lowercase words back into a single string with spaces in between.

This part of the code essentially takes each comment, splits it into words, converts them to lowercase, and then joins them back together into a standardized, lowercase sentence.

df2['new_comments'] = df['new_comments']:

- Updates df2['new_comments'] to match df['new_comments']. Since df2 is a duplicate DataFrame containing the cleaned text data, this line ensures both DataFrames remain in sync.

df2.head():

- Displays the first five rows of df2 to verify the changes.
- **Standardization**: Lowercasing helps maintain consistency across text data, which is important for tasks like tokenization, word frequency counting, and many NLP models that treat "Word" and "word" as distinct.
- **Context in Pipeline**: This step should ideally be done earlier in the pipeline, before steps like expanding contractions, removing punctuation, and lemmatization, to ensure all text is handled uniformly throughout the cleaning process.

4. Remove Contractions:

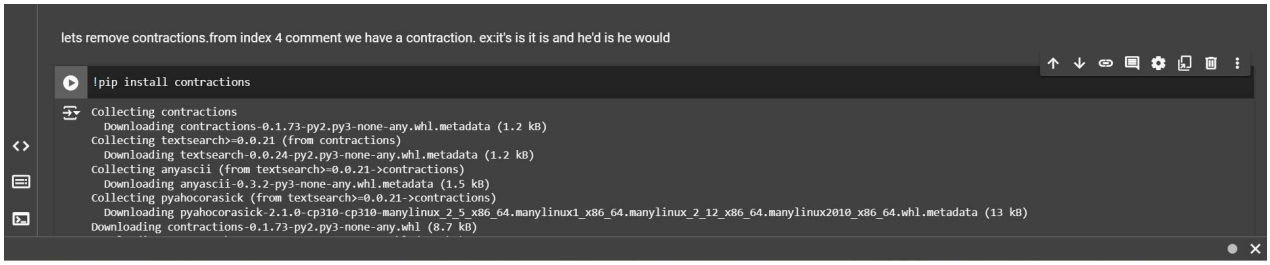
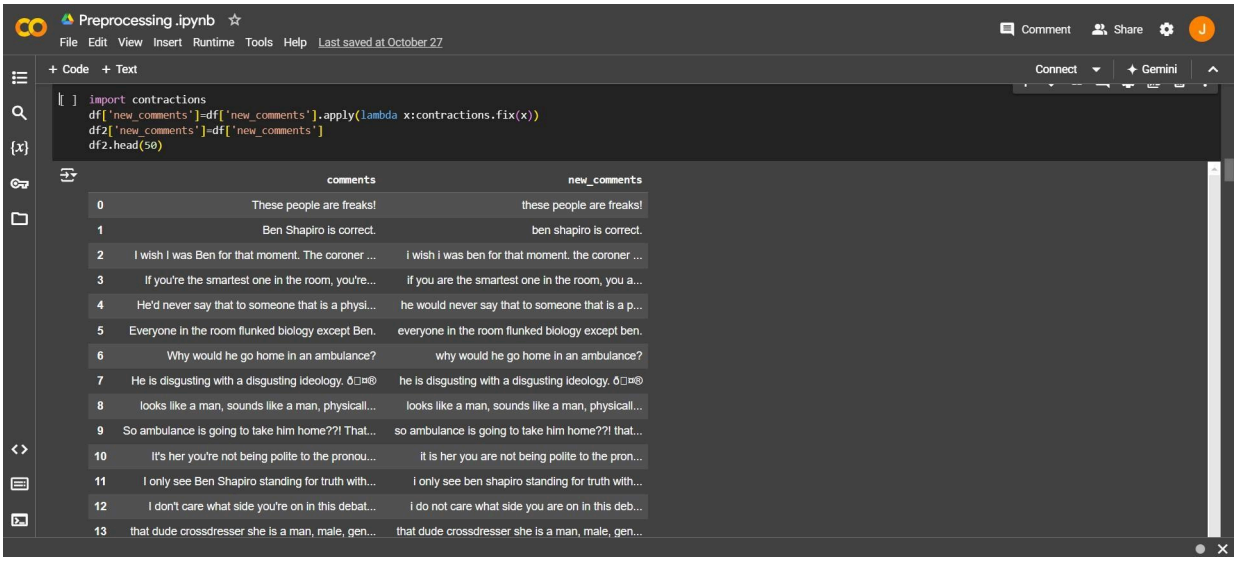


Fig: 1(i) Shows the installation contraction library.



	comments	new_comments
0	These people are freaks!	these people are freaks!
1	Ben Shapiro is correct.	ben shapiro is correct.
2	I wish I was Ben for that moment. The coroner ...	i wish i was ben for that moment. the coroner ...
3	If you're the smartest one in the room, you're...	if you are the smartest one in the room, you a...
4	He'd never say that to someone that is a physi...	he would never say that to someone that is a p...
5	Everyone in the room flunked biology except Ben.	everyone in the room flunked biology except ben.
6	Why would he go home in an ambulance?	why would he go home in an ambulance?
7	He is disgusting with a disgusting ideology. ð□¤®	he is disgusting with a disgusting ideology. ð□¤®
8	looks like a man, sounds like a man, physicall...	looks like a man, sounds like a man, physicall...
9	So ambulance is going to take him home??! That...	so ambulance is going to take him home??! that...
10	It's her you're not being polite to the pronou...	it is her you are not being polite to the pron...
11	I only see Ben Shapiro standing for truth with...	i only see ben shapiro standing for truth with...
12	I don't care what side you're on in this debat...	i do not care what side you are on in this deb...
13	that dude crossdresser she is a man, male, gen...	that dude crossdresser she is a man, male, gen...

Fig: 1(j) Shows the code and output for expanding the contraction and replacing words with an apostrophe.

Explanation: The contractions library is used to expand contractions (like changing "can't" to "cannot").

>>> Install the Contractions Library: Code [!pip install contractions]

- This command installs the **contractions** library, which is useful for expanding contractions in English text (e.g., "I'm" to "I am").
- This line is run in environments where libraries need to be installed before use, such as Google Colab or Jupyter Notebook.

>>> Import Contractions Library: Code [import contractions]

- Imports the contractions library to make its functions available in the code.
- The main function used from this library is contractions.fix(), which expands any contractions found in a given text.

>>> Expand Contractions in new_comments: Code [df['new_comments'] = df['new_comments'].apply(lambda x: contractions.fix(x))]

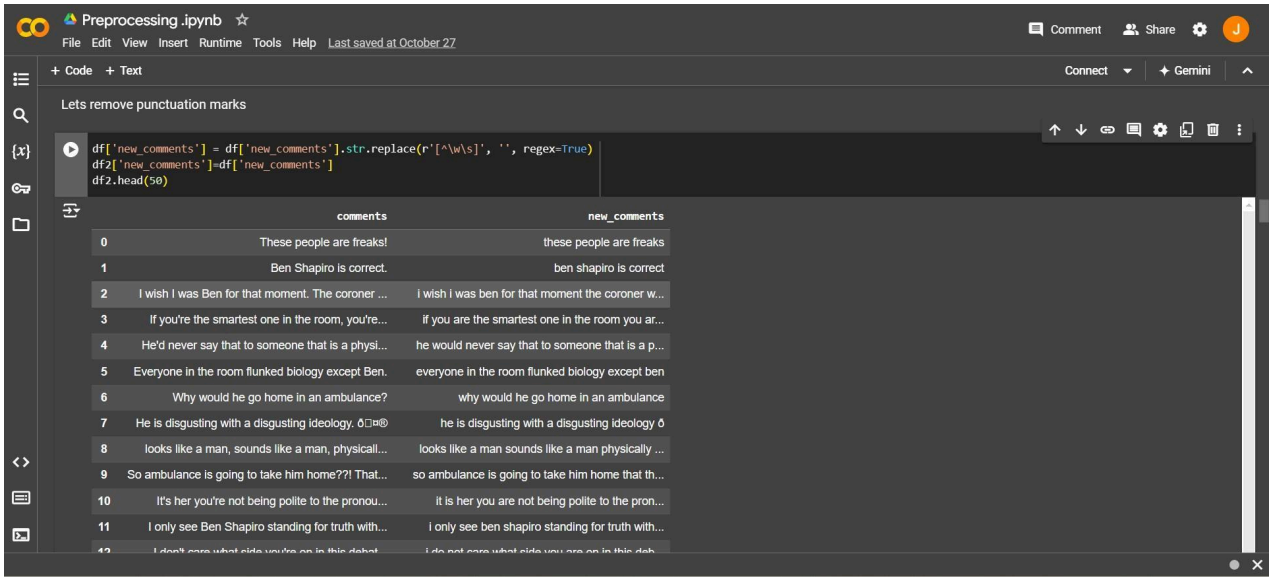
- This line expands contractions in the new_comments column of the DataFrame df.
- Breaking it down further:
  - **df['new_comments']**: Accesses the new_comments column in df, which is where the text data is stored.
  - **.apply(lambda x: contractions.fix(x))**: Uses the apply() function to apply a lambda function to each entry (row) in new_comments.

- **contractions.fix(x)**: The function fix() from the contractions library takes each text entry (x) and expands any contractions within it.
  - For example, if x is "I'm happy," it changes it to "I am happy."
- The expanded text replaces the original content in the new_comments column.

>>> Copy Expanded Contractions to df2: Code [df2['new_comments'] = df['new_comments']]

- This line copies the modified new_comments column from df to df2, updating df2 with the expanded contractions.
- df2 is a secondary DataFrame created to hold both the original and processed versions of the comments, allowing you to reference the changes made without altering the original text data.

5. **Remove Punctuation**:



	comments	new_comments
0	These people are freaks!	these people are freaks
1	Ben Shapiro is correct.	ben shapiro is correct
2	I wish I was Ben for that moment. The coroner ...	i wish i was ben for that moment the coroner w...
3	If you're the smartest one in the room, you're...	if you are the smartest one in the room you ar...
4	He'd never say that to someone that is a physi...	he would never say that to someone that is a p...
5	Everyone in the room flunked biology except Ben.	everyone in the room flunked biology except ben
6	Why would he go home in an ambulance?	why would he go home in an ambulance
7	He is disgusting with a disgusting ideology. ð□¤®	he is disgusting with a disgusting ideology ð
8	looks like a man, sounds like a man, physicall...	looks like a man sounds like a man physically ...
9	So ambulance is going to take him home??! That...	so ambulance is going to take him home that th...
10	It's her you're not being polite to the pronou...	it is her you are not being polite to the pron...
11	I only see Ben Shapiro standing for truth with...	i only see ben shapiro standing for truth with...

Fig: 1(k) Shows the code and output for removing punctuation marks from the dataset.

- Removes punctuation using regex ([^\w\s] matches anything that's not a word or whitespace character).
- This step also helps normalize the text by removing unnecessary symbols.

Remove Punctuation from new_comments: Code [df['new_comments'] = df['new_comments'].str.replace(r'[^\w\s]', '', regex=True)]

- This line modifies the new_comments column in df by removing any punctuation or special characters.

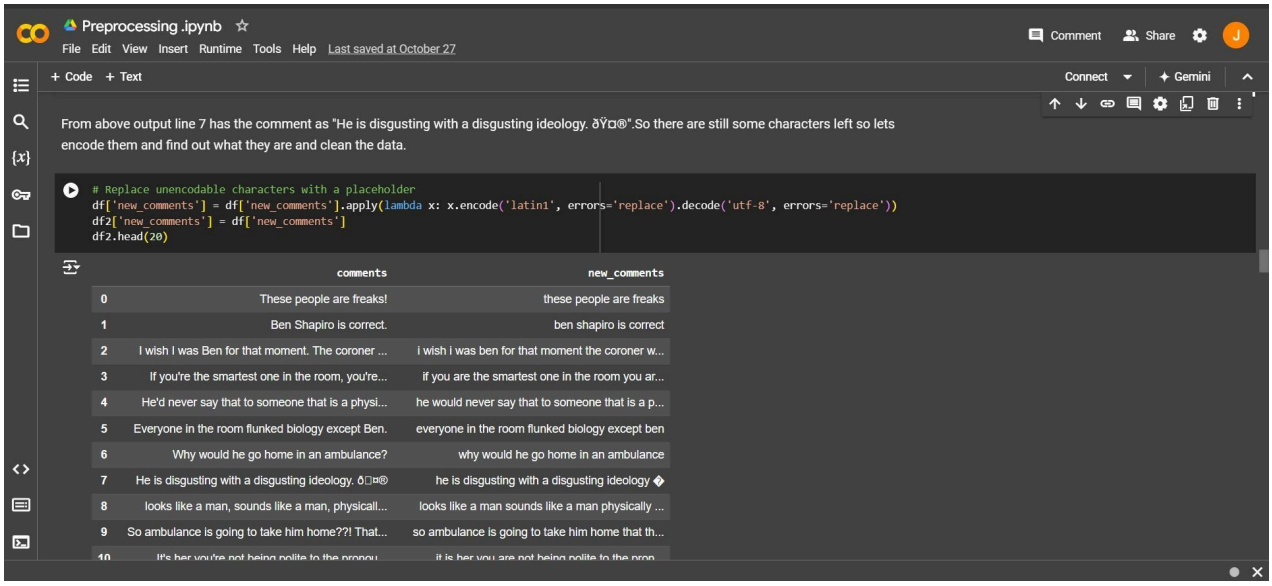
- Let's break it down:
    - **df['new_comments']**: Accesses the new_comments column in the df DataFrame.
    - **.str.replace(r'[^\w\s]', '', regex=True)**: Applies a regex replacement to each entry in the column.
        - **r'[^\w\s]'**: This is a **regular expression** pattern.
            - **[^\w\s]**: The ^ symbol inside the brackets negates the expression, meaning "match anything that is not a word character (\w) or whitespace (\s)."
                - **\w**: Matches any word character (equivalent to letters, digits, and underscores).
                - **\s**: Matches any whitespace character (spaces, tabs, line breaks).
            - Overall, r'[^\w\s]' matches any non-word, non-whitespace character (essentially, punctuation and special symbols).
        - **''**: The second argument, an empty string, specifies what to replace the matched characters with—in this case, nothing. This effectively removes any punctuation or special characters.
        - **regex=True**: Specifies that the replacement should be treated as a regex pattern.

Copy the Modified Column to df2: Code [df2['new_comments']=df['new_comments']]

- This line copies the modified new_comments column from df to df2, updating df2 to reflect the changes.
- df2 is a secondary DataFrame created to track both original and processed text for reference without altering the original text data.

6. Encoding Issues Handling

Encoding Fix: Re-encodes the comments to handle any unencodable characters, replacing them with a placeholder if necessary.



	comments	new_comments
0	These people are freaks!	these people are freaks
1	Ben Shapiro is correct.	ben shapiro is correct
2	I wish I was Ben for that moment. The coroner ...	i wish i was ben for that moment the coroner w...
3	If you're the smartest one in the room, you're...	if you are the smartest one in the room you ar...
4	He'd never say that to someone that is a physi...	he would never say that to someone that is a p...
5	Everyone in the room flunked biology except Ben.	everyone in the room flunked biology except ben
6	Why would he go home in an ambulance?	why would he go home in an ambulance
7	He is disgusting with a disgusting ideology. ð□¤®	he is disgusting with a disgusting ideology ◆
8	looks like a man, sounds like a man, physicall...	looks like a man sounds like a man physically ...
9	So ambulance is going to take him home??! That...	so ambulance is going to take him home that th...
10	It's her you're not being polite to the propou...	it is her you are not being polite to the prop...

Fig: 1(l) Shows the code and output for characters other than punctuation marks.

x.encode('latin1', errors='replace'):

- The encode() method converts the string x (the individual comment) from its current encoding (usually Unicode) to a byte representation using the specified encoding ('latin1' in this case).
- **'latin1' Encoding**: Also known as ISO-8859-1, this encoding can represent characters in Western European languages, which makes it useful for many types of text. However, it has limitations in representing characters outside of this range.
- **errors='replace'**: This parameter tells the encoder to replace any characters that cannot be encoded using 'latin1' with a placeholder character (typically ? or 8), rather than raising an error. This ensures that the function can process any text, regardless of character set issues.

.decode('utf-8', errors='replace'):

- After encoding the string to bytes, the decode() method converts the byte representation back into a string, interpreting it with the 'utf-8' encoding.
- **'utf-8' Encoding**: This is a variable-length encoding that can represent any character in the Unicode character set. It's widely used because it can handle a vast array of characters from various languages and symbols.

- **errors='replace'**: Similar to the encoding step, this parameter will replace any byte sequences that cannot be decoded into valid 'utf-8' characters with a placeholder character. This helps maintain the integrity of the text even if there are problematic byte sequences..

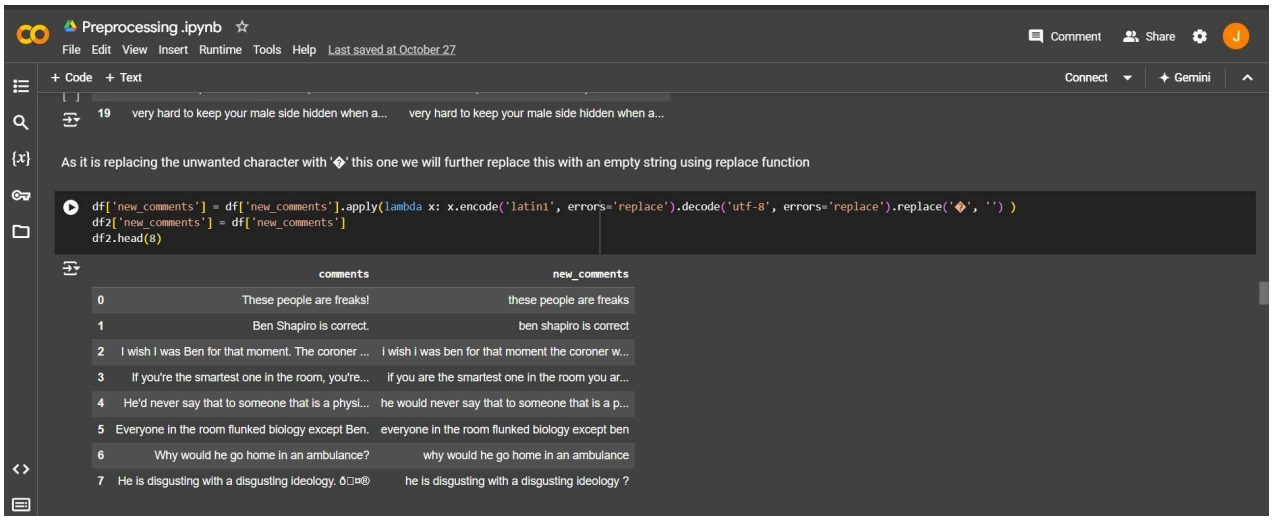


Fig: 1(m) Show the code and output for replacing unwanted characters with empty string.

x.encode('latin1', errors='replace'):

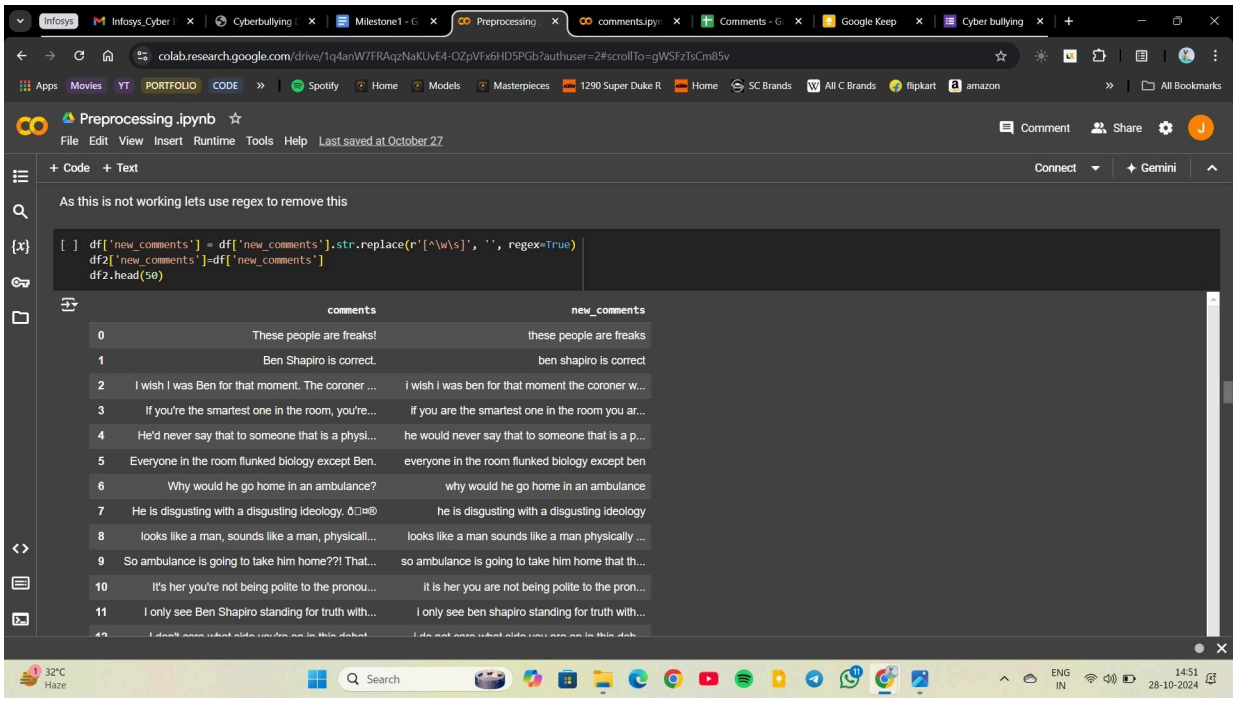
- The encode() method converts the string x from its current Unicode format into a byte representation using the 'latin1' encoding scheme.
- **'latin1' Encoding**: This encoding supports characters from Western European languages but is limited in handling characters outside this range.
- **errors='replace'**: This argument tells the encoder to replace any character that cannot be represented in 'latin1' with a placeholder character, typically ? or a similar symbol. This ensures that the encoding process does not fail and can handle a wider range of text,
  albeit at the cost of potential data loss.

.decode('utf-8', errors='replace'):

- After encoding the string to bytes, the decode() method is used to convert the byte string back into a regular string, interpreting it using 'utf-8' encoding.
- **'utf-8' Encoding**: This encoding can represent any character in the Unicode standard, making it suitable for a diverse set of characters from various languages.
- **errors='replace'**: Similar to the encoding step, this tells the decoder to replace any byte sequences that do not correspond to valid 'utf-8' characters with a placeholder. This step helps mitigate issues that may arise from invalid byte sequences.

.replace('8', ''):

- After decoding, this additional step uses the replace() method to remove any placeholder characters (specifically 8, which is often used to represent an unknown or unrecognized character).
- The presence of 8 indicates that there were characters in the original string that could not be encoded or decoded properly. By removing these placeholders, the code attempts to clean the text further and return a more readable result.



	comments	new_comments
0	These people are freaks!	these people are freaks
1	Ben Shapiro is correct.	ben shapiro is correct
2	I wish I was Ben for that moment. The coroner ...	i wish i was ben for that moment the coroner w...
3	If you're the smartest one in the room, you're...	if you are the smartest one in the room you ar...
4	He'd never say that to someone that is a physi...	he would never say that to someone that is a p...
5	Everyone in the room flunked biology except Ben.	everyone in the room flunked biology except ben
6	Why would he go home in an ambulance?	why would he go home in an ambulance
7	He is disgusting with a disgusting ideology. ð□¤®	he is disgusting with a disgusting ideology
8	looks like a man, sounds like a man, physicall...	looks like a man sounds like a man physically ...
9	So ambulance is going to take him home??! That...	so ambulance is going to take him home that th...
10	It's her you're not being polite to the pronou...	it is her you are not being polite to the pron...
11	I only see Ben Shapiro standing for truth with...	i only see ben shapiro standing for truth with...

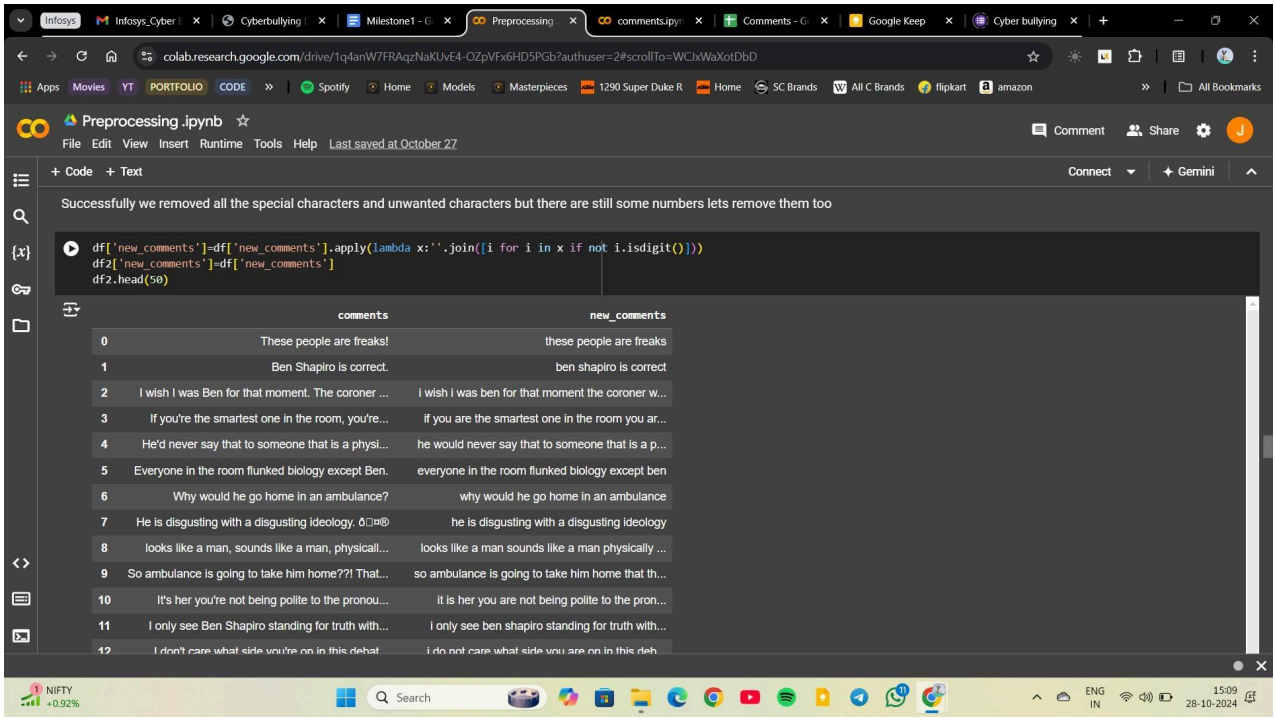
Fig: 1(n) Shows the code and output for removing the empty string.

.str.replace(r'[^\w\s]', '', regex=True):

- The str.replace() method is used to replace occurrences of a specified substring or pattern within string values in a Pandas Series (in this case, the new_comments column).
- **r'[^\w\s]'**: This is a regular expression (regex) pattern used to identify characters for replacement.
    - **[^\w\s]**:
        - ^: The caret symbol at the beginning of the square brackets indicates a negation, meaning it will match anything that is **not** included in the specified characters.

    - \w: This matches any word character, which includes letters (both uppercase and lowercase), digits (0-9), and underscores (_).
    - \s: This matches any whitespace character, including spaces, tabs, and newline characters.
  - Therefore, the entire pattern [^\w\s] matches any character that is **not** a word character or a whitespace character. This effectively targets punctuation and special characters.
- '',: The second argument specifies the replacement string, which is an empty string (''). This means that all characters matched by the regex pattern will be removed from the text.
- **regex=True**: This parameter indicates that the first argument should be treated as a regular expression. It allows for the use of regex patterns in the replacement operation.

8. **Removing Digits**: Filters out all digits from the comments.



	comments	new_comments
0	These people are freaks!	these people are freaks
1	Ben Shapiro is correct.	ben shapiro is correct
2	I wish I was Ben for that moment. The coroner ...	i wish i was ben for that moment the coroner w...
3	If you're the smartest one in the room, you're...	if you are the smartest one in the room you ar...
4	He'd never say that to someone that is a physi...	he would never say that to someone that is a p...
5	Everyone in the room flunked biology except Ben.	everyone in the room flunked biology except ben
6	Why would he go home in an ambulance?	why would he go home in an ambulance
7	He is disgusting with a disgusting ideology. ð□¤®	he is disgusting with a disgusting ideology
8	looks like a man, sounds like a man, physicall...	looks like a man sounds like a man physically ...
9	So ambulance is going to take him home??! That...	so ambulance is going to take him home that th...
10	It's her you're not being polite to the pronou...	it is her you are not being polite to the pron...
11	I only see Ben Shapiro standing for truth with...	i only see ben shapiro standing for truth with...

Fig: 1(o) Shows the code and output for removing the digits from the dataset.

[i for i in x if not i.isdigit()]:

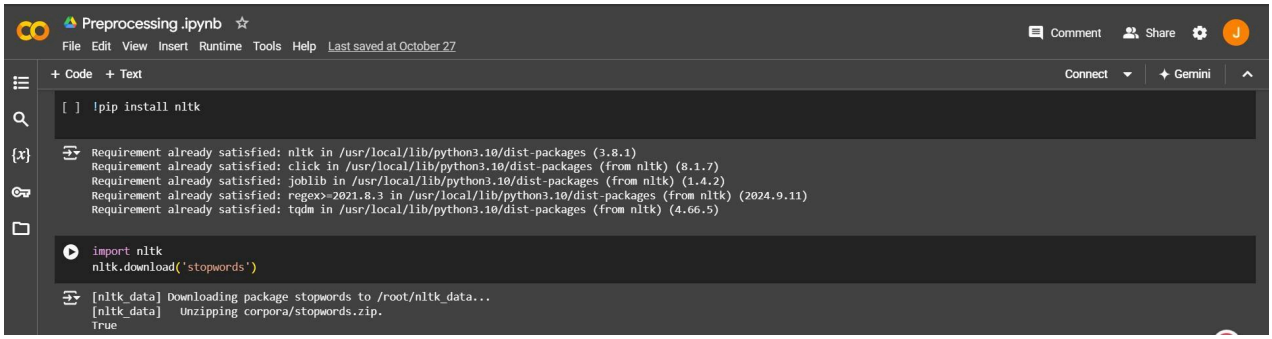
- This part is a list comprehension that iterates over each character i in the string x.
- **if not i.isdigit()**: The condition checks whether the character i is not a digit. The isdigit() method returns True if the character is a digit (0-9) and False otherwise.
- Therefore, this list comprehension creates a list of characters from the original string x, excluding any digits.

''.join([...]):

- The join() method takes the list of characters created by the list comprehension and concatenates them into a single string.
- The '' before join() specifies that there should be no characters between the joined elements (i.e., the characters are simply concatenated together without any separators).
- The result is a new string that contains all the original characters from x, except for any digits.

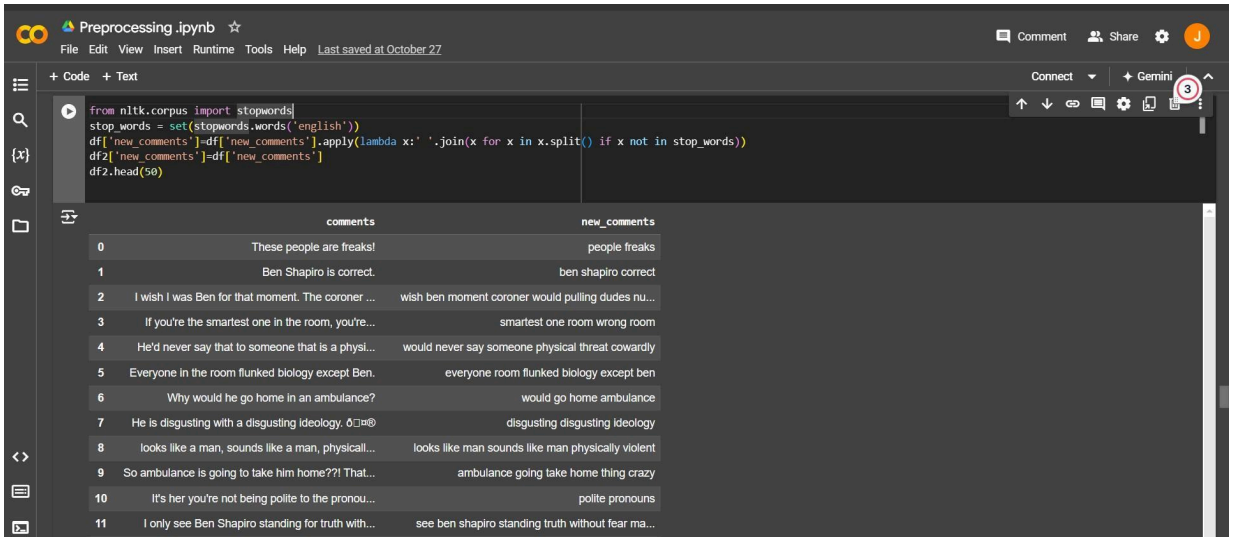
9. Downloading Stopword

NLTK Stopwords: Downloads a list of stopwords (common words that can be removed to improve the analysis).



10. Removing Stopwords

Stopwords Removal: Removes common stopwords from the new_comments column to focus on more meaningful words.



	comments	new_comments
0	These people are freaks!	people freaks
1	Ben Shapiro is correct.	ben shapiro correct
2	I wish I was Ben for that moment. The coroner ...	wish ben moment coroner would pulling dudes nu...
3	If you're the smartest one in the room, you're...	smartest one room wrong room
4	He'd never say that to someone that is a physi...	would never say someone physical threat cowardly
5	Everyone in the room flunked biology except Ben.	everyone room flunked biology except ben
6	Why would he go home in an ambulance?	would go home ambulance
7	He is disgusting with a disgusting ideology. ð□¤®	disgusting disgusting ideology
8	looks like a man, sounds like a man, physicall...	looks like man sounds like man physically violent
9	So ambulance is going to take him home??! That...	ambulance going take home thing crazy
10	It's her you're not being polite to the pronou...	polite pronouns
11	I only see Ben Shapiro standing for truth with...	see ben shapiro standing truth without fear ma...

from nltk.corpus import stopwords:

- This line imports the stopwords module from the Natural Language Toolkit (NLTK), which is a popular library in Python for natural language processing (NLP).
- Stopwords are commonly used words (such as "and," "the," "is," etc.) that are often filtered out in text processing because they carry little meaningful information for many NLP tasks.

stop_words = set(stopwords.words('english')):

- Here, the stopwords.words('english') function retrieves a list of English stopwords from the NLTK corpus.
- This list is then converted to a set using set(), which allows for faster membership testing (checking if a word is a stopword) compared to a list. The stop_words variable now contains all the English stopwords.

df['new_comments']:

- This part references the new_comments column of the DataFrame df. This column contains text data that has already undergone several preprocessing steps, but it may still include stopwords that need to be removed.

.apply(lambda x: ...):

- The apply() method is called on the new_comments column to apply a function to each entry x in the column. Each x represents an individual comment string.

' '.join(x for x in x.split() if x not in stop_words):

- This part is a combination of list comprehension and the join() method.
- **x.split()**: This method splits the string x into a list of words based on whitespace. For example, "This is a comment" becomes ['This', 'is', 'a', 'comment'].
- **(x for x in x.split() if x not in stop_words)**: This is a generator expression that iterates over each word in the split list. The if x not in stop_words condition filters out any word that is in the stop_words set, meaning only non-stopword words will be retained.
- **' '.join(...)**: The join() method then takes the remaining words and concatenates them into a single string, separated by spaces. This results in a new string that consists of the original words but excludes all stopwords.

12. Lemmatization

Lemmatization is a key step in text preprocessing for NLP tasks. It helps reduce words to their base forms, which can improve the performance of machine learning models by ensuring that different inflected forms of a word (like "running," "ran," and "runs") are treated as the same item. This process enhances the model's ability to understand and analyze the underlying meaning of the text.

Spacy: Imports the SpaCy library and loads the English language model.

import spacy:

- This line imports the SpaCy library, which is a powerful library for natural language processing (NLP) in Python. It provides tools for tasks like tokenization, part-of-speech tagging, named entity recognition, and lemmatization.

from langdetect import detect, LangDetectException:

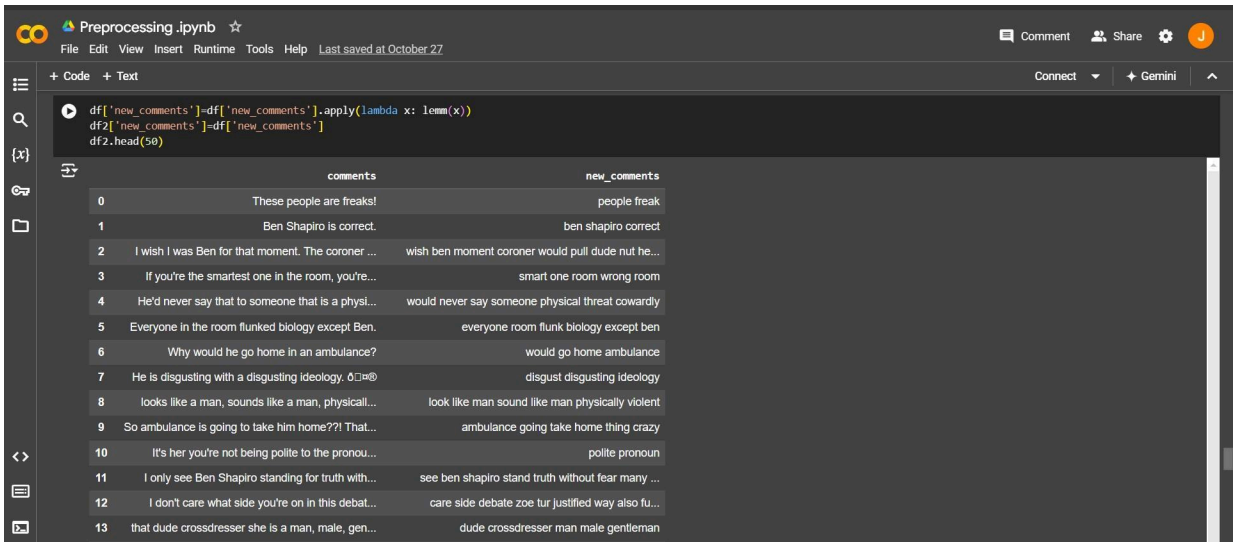
- This line imports functions from the langdetect library, which can detect the language of a given text. While it's imported here, it is not used in the current snippet.

nlp = spacy.load("en_core_web_sm"):

- This line loads a pre-trained English language model (en_core_web_sm) from SpaCy. The model contains information about the English language, such as vocabulary, grammar, and word vectors.
- The nlp object will be used to process text data, allowing for various NLP tasks.

Lemmatization Function: The lemm function lemmatizes each word in the comments. Lemmatization reduces words to their base or dictionary form (e.g., "running" becomes "run"). This process considers the context of words, which is more effective than stemming.

Apply Lemmatization: Applies the lemm function to the new_comments column.



	comments	new_comments
0	These people are freaks!	people freak
1	Ben Shapiro is correct.	ben shapiro correct
2	I wish I was Ben for that moment. The coroner ...	wish ben moment coroner would pull dude nut he...
3	If you're the smartest one in the room, you're...	smart one room wrong room
4	He'd never say that to someone that is a physi...	would never say someone physical threat cowardly
5	Everyone in the room flunked biology except Ben.	everyone room flunk biology except ben
6	Why would he go home in an ambulance?	would go home ambulance
7	He is disgusting with a disgusting ideology. ð□¤®	disgust disgusting ideology
8	looks like a man, sounds like a man, physicall...	look like man sound like man physically violent
9	So ambulance is going to take him home??! That...	ambulance going take home thing crazy
10	It's her you're not being polite to the pronou...	polite pronoun
11	I only see Ben Shapiro standing for truth with...	see ben shapiro stand truth without fear many ...
12	I don't care what side you're on in this debat...	care side debate zoe tur justified way also fu...
13	that dude crossdresser she is a man, male, gen...	dude crossdresser man male gentleman

def lemm(comment)::

- This line defines a function called lemm that takes a single argument comment, which is expected to be a string containing text that will be lemmatized.

c = nlp(comment):

- Inside the lemm function, the text comment is processed by the SpaCy pipeline using the nlp object.
- This converts the input string into a Doc object (c), which contains tokens with additional linguistic features like part-of-speech tags and lemmas.

return ' '.join(token.lemma_ for token in c):

- This line constructs a new string by iterating over each token in the Doc object c.
- **token.lemma_**: For each token, the lemma (the base or dictionary form of a word) is retrieved. For example, the lemma of "running" is "run".
- The join() method concatenates all the lemmas into a single string, with spaces in between. This results in a string that has the same meaning as the original comment but uses the base forms of the words instead.

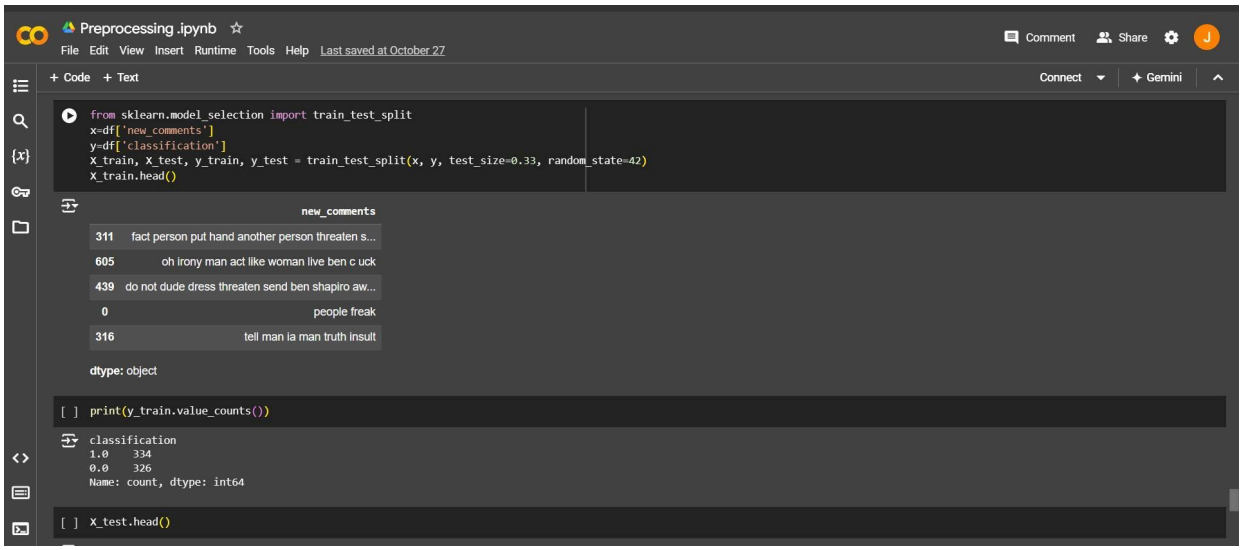
df['new_comments'] = df['new_comments'].apply(lambda x: lemm(x)):

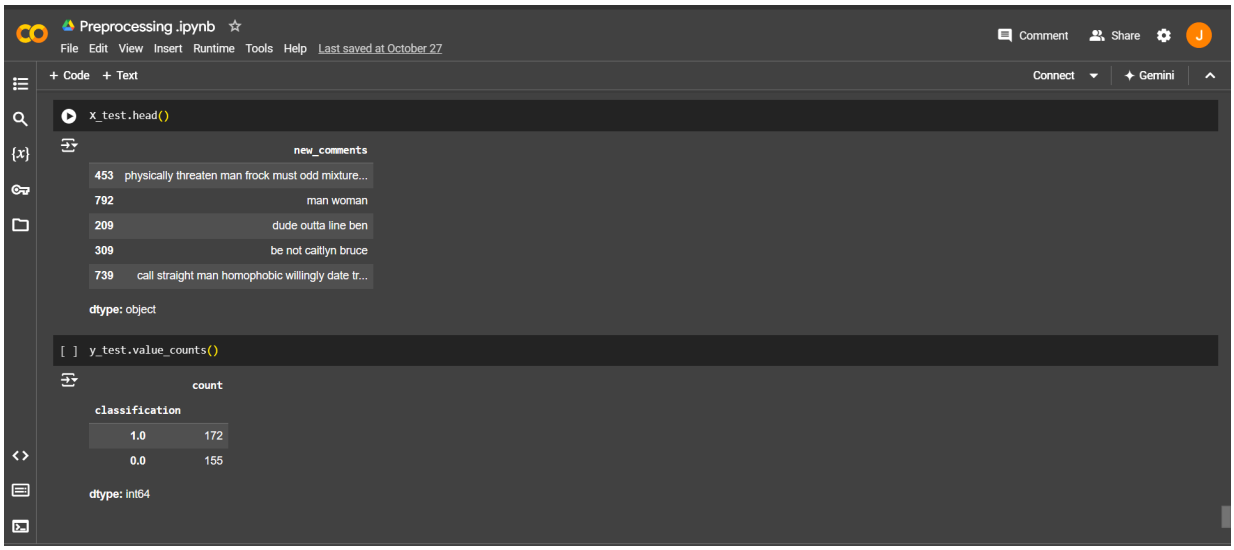
- This line applies the lemm function to each entry in the new_comments column of the DataFrame df.
- Each comment is processed to replace words with their lemmas, and the updated comments are assigned back to the new_comments column.

13. Splitting Data for Training and Testing

Train-Test Split: Imports the train_test_split function from Scikit-learn to divide the dataset into training and testing sets. Here, 33% of the data is allocated for testing.

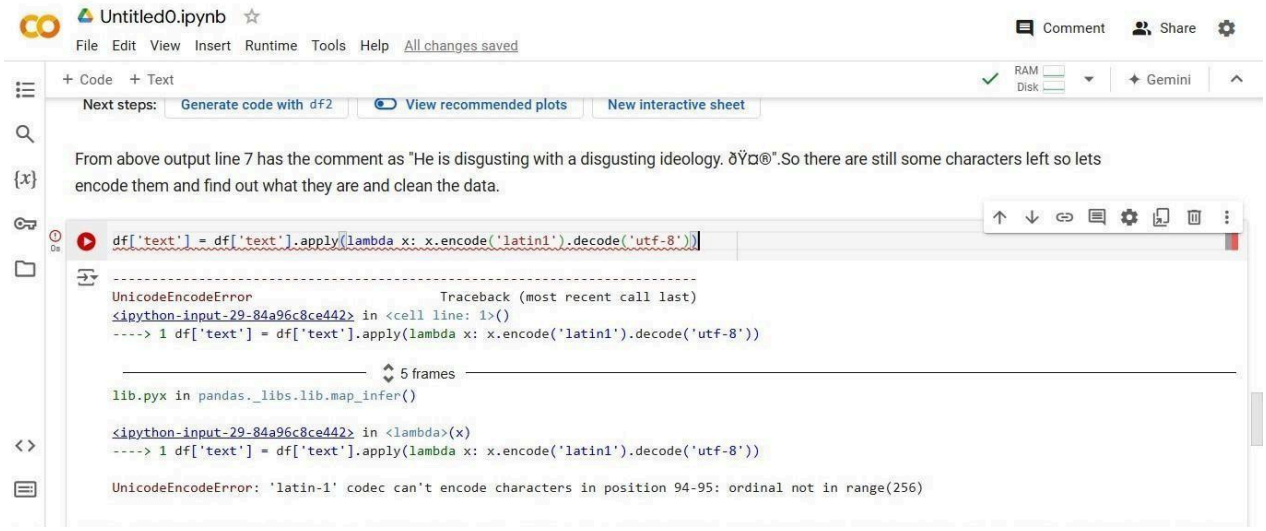
Variables: x holds the cleaned comments, while y contains the classification labels.





1. **`from sklearn.model_selection import train_test_split`**:
    - This line imports the `train_test_split` function from the `sklearn.model_selection` module, which is part of the Scikit-learn library. This function is used to split datasets into training and testing subsets for machine learning models.
2. **`x = df['new_comments']`**:
    - Here, the variable `x` is assigned the `new_comments` column from the DataFrame `df`. This column contains the preprocessed text data (comments) that will be used as features for training the model.
3. **`y = df['classification']`**:
    - The variable `y` is assigned the `classification` column from the DataFrame `df`. This column contains the target labels (classifications) that correspond to each comment. In a supervised learning task, this is the output the model will learn to predict based on the input features in `x`.
4. **`X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_state=42)`**:
    - This line calls the `train_test_split` function to split the data into training and testing sets.
    - **`x` and `y`**: These are the input features and target labels, respectively.
    - **`test_size=0.33`**: This parameter specifies the proportion of the dataset to include in the test split. In this case, 33% of the data will be reserved for testing, while the remaining 67% will be used for training.
    - **`random_state=42`**: This parameter sets the random seed for reproducibility. By specifying a random state, you ensure that the split will produce the same results each time you run the code. This is useful for debugging and for consistent results during experimentation.
    - The function returns four variables: `X_train`, `X_test`, `y_train`, and `y_test`, which represent the training and testing sets of features and labels, respectively.
5. **`X_train.head()`**:
    - This line calls the `head()` method on the `X_train` DataFrame to display the first few rows of the training data. This helps you quickly inspect the training set to verify that the split has been performed correctly.

Errors encountered during preprocessing

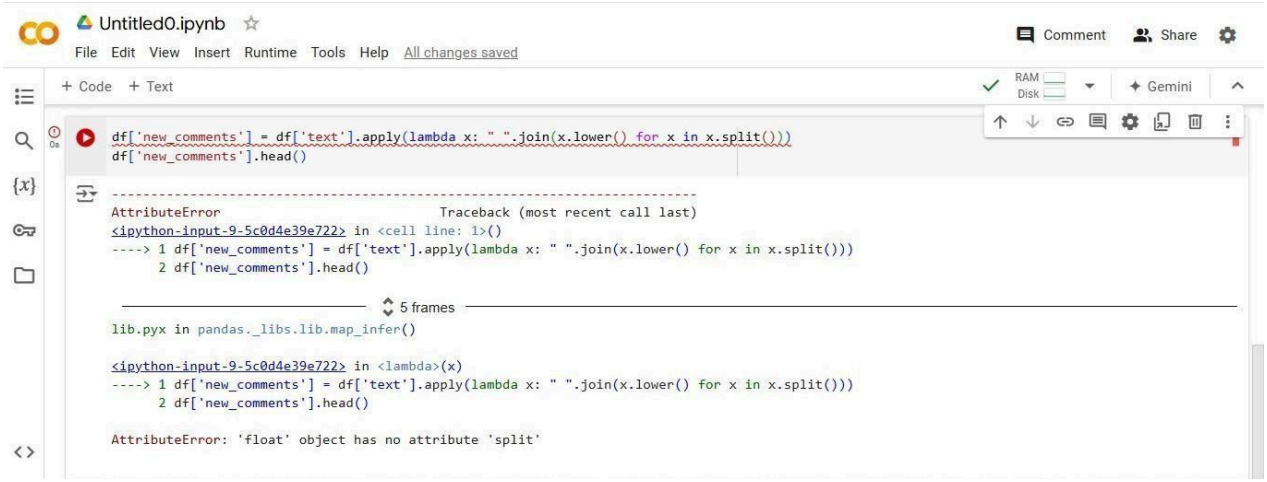


Reason for error:

The error "AttributeError: 'float' object has no attribute 'split' occurs because the lambda function within the apply method is trying to call the split method on a float value within the 'text' column of your DataFrame. The split method is designed for strings, not floats. This suggests that the 'text' column in your DataFrame contains a mix of data types, including some float values which are causing the error.

Solution:

Before directly applying the split attribute we changed the text as string data type and resolved this error.



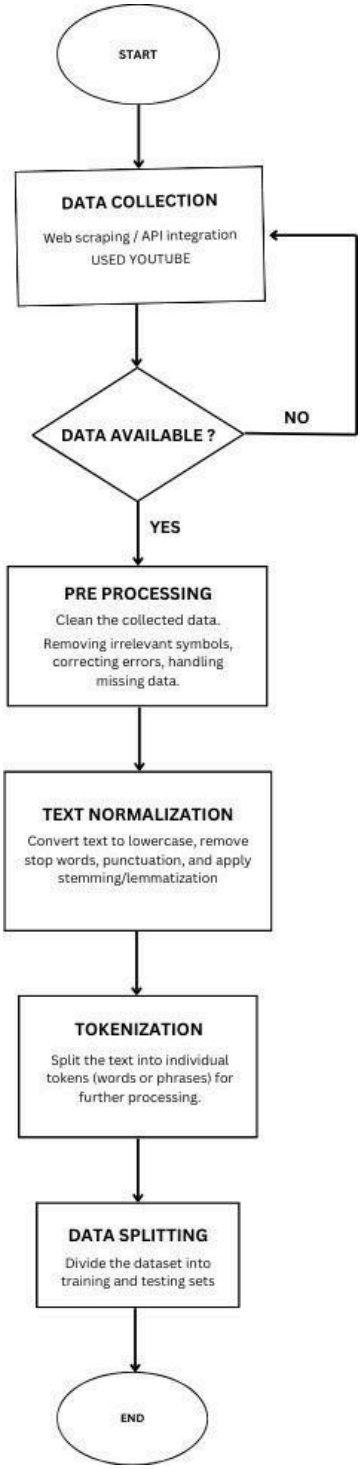
Reason for this error:

It is due to a character (œ) that cannot be encoded in latin-1 since it only supports characters with ordinals in the range 0-255.

Solution:

As we are just trying to remove unwanted characters and emojis we can just replace them with an empty string using a placeholder.

FLOWCHART FOR DATA COLLECTION AND PREPROCESSING

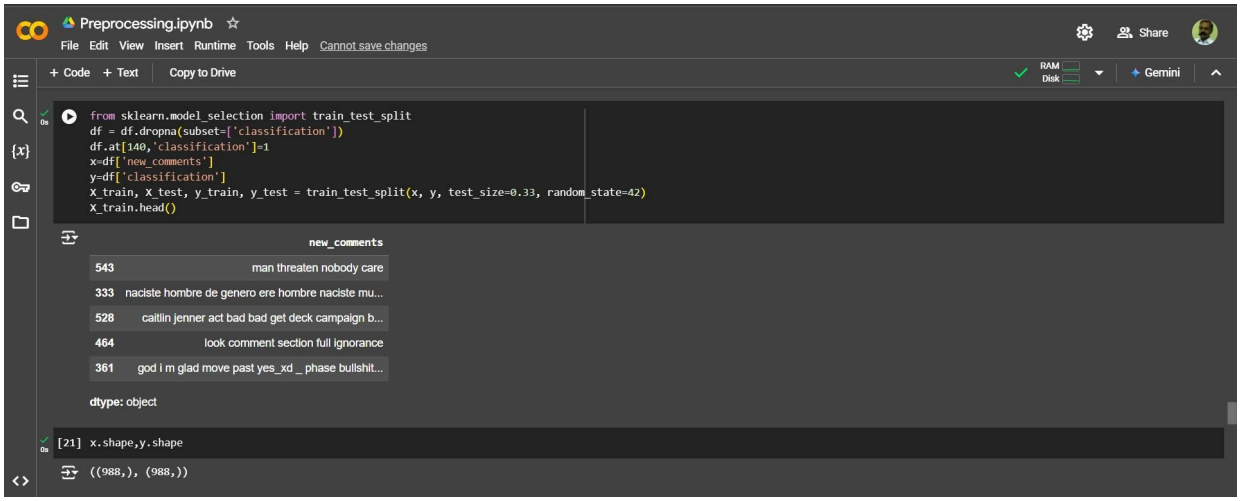


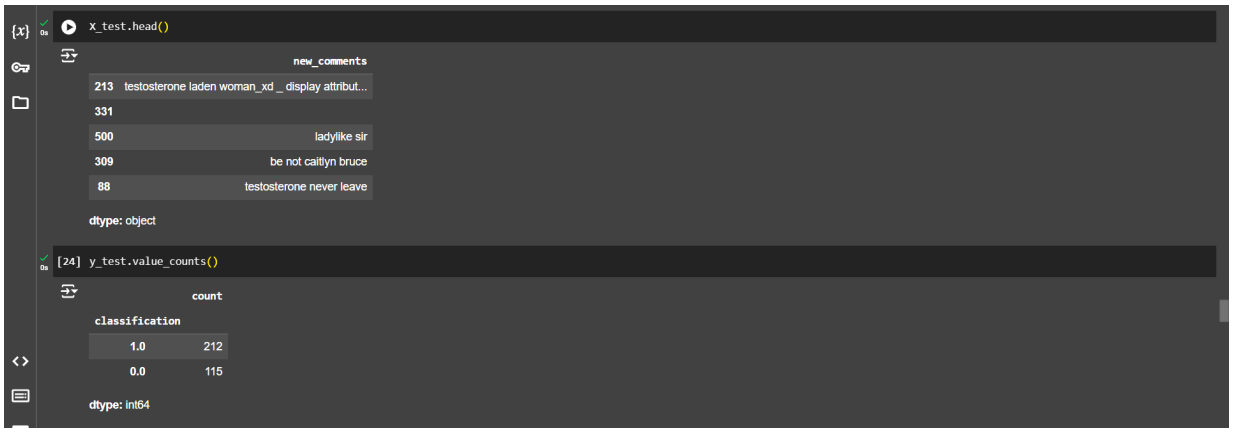
Milestone 2: Data Splitting & Model Training:

1. Splitting Data for Training and Testing

Train-Test Split: Imports the train_test_split function from Scikit-learn to divide the dataset into training and testing sets. Here, 33% of the data is allocated for testing.

Variables: x holds the cleaned comments, while y contains the classification labels.

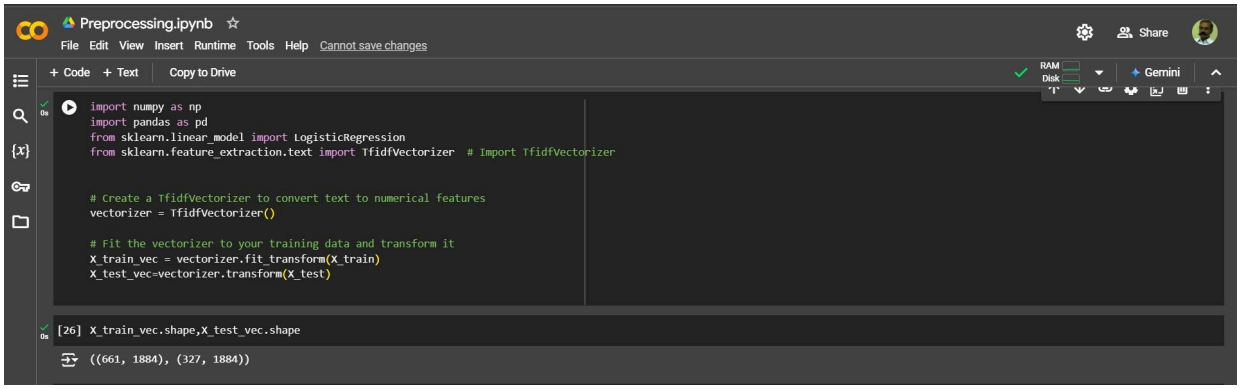




1. **from sklearn.model_selection import train_test_split**:
    - This line imports the train_test_split function from the sklearn.model_selection module, which is part of the Scikit-learn library. This function is used to split datasets into training and testing subsets for machine learning models.
2. **x = df['new_comments']**:
    - Here, the variable x is assigned the new_comments column from the DataFrame df. This column contains the preprocessed text data (comments) that will be used as features for training the model.
3. **y = df['classification']**:
    - The variable y is assigned the classification column from the DataFrame df. This column contains the target labels (classifications) that correspond to each comment. In a supervised learning task, this is the output the model will learn to predict based on the input features in x.
4. **X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_state=42)**:
    - This line calls the train_test_split function to split the data into training and testing sets.
    - **x and y**: These are the input features and target labels, respectively.
    - **test_size=0.33**: This parameter specifies the proportion of the dataset to include in the test split. In this case, 33% of the data will be reserved for testing, while the remaining 67% will be used for training.
    - **random_state=42**: This parameter sets the random seed for reproducibility. By specifying a random state, you ensure that the split will produce the same results each time you run the code. This is useful for debugging and for consistent results during experimentation.
    - The function returns four variables: X_train, X_test, y_train, and y_test, which represent the training and testing sets of features and labels, respectively.
5. **X_train.head()**:
    - This line calls the head() method on the X_train DataFrame to display the first few rows of the training data. This helps you quickly inspect the training set to verify that the split has been performed correctly.

6. **y_test.value_counts()**: Counts the occurrences of each unique value in y_test.

2. Model Training:



This project seems to be a text classification task where logistic regression is used to classify text data. Here's a detailed breakdown of each component in your code:

1. Import Libraries

- numpy and pandas are popular libraries in Python for data manipulation. numpy is often used for numerical operations, and pandas is mainly used for handling data in table format.
- LogisticRegression from sklearn.linear_model provides a logistic regression model, commonly used for binary or multi-class classification tasks.
- TfidfVectorizer from sklearn.feature_extraction.text is a tool to convert a collection of raw text into numerical data by applying TF-IDF (Term Frequency-Inverse Document Frequency) weighting, which gives more weight to informative words and less to common words.

2. Vectorizing Text with TfidfVectorizer:

- TfidfVectorizer transforms the text data into a matrix of TF-IDF features. TF-IDF helps give insight into how important a word is within a document, considering how frequently it appears in a document relative to other documents in the dataset.

  #### How TF-IDF Works :

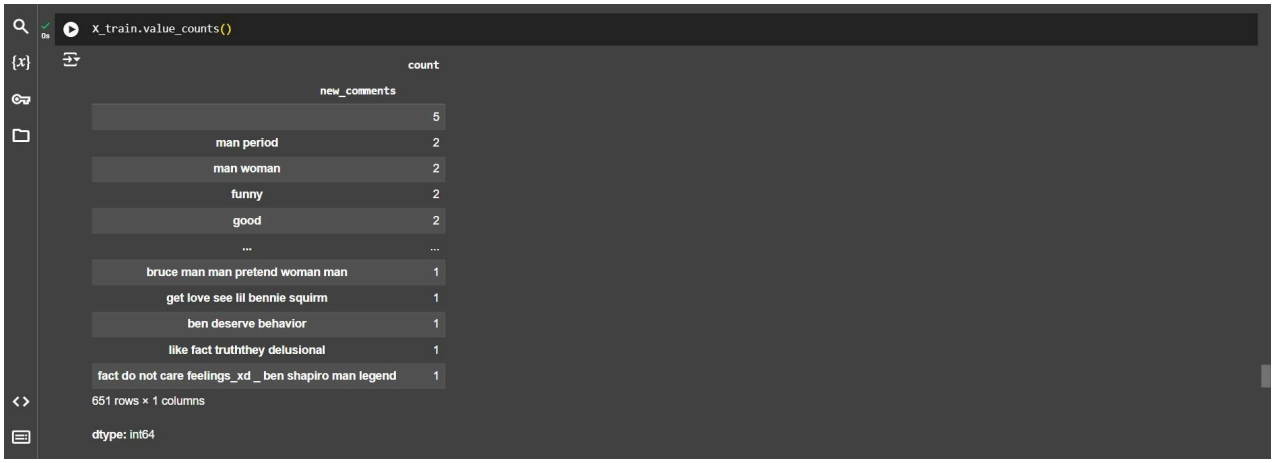
  - **Term Frequency (TF)** measures how often a term appears in a document.
  - **Inverse Document Frequency (IDF)** measures how common or rare a term is across all documents. Words that appear frequently across many documents (e.g., "the," "and") get lower weights.
  - Together, the TF-IDF score reflects how important a word is to a specific document in the context of the whole dataset.

3. Fitting and Transforming Training Data

- Here, fit_transform() is called on the training data (X_train).
    - fit learns the vocabulary from the training data and calculates the IDF for each term.
    - transform converts the training data into a sparse matrix of TF-IDF features.
- This results in X_train_vec, a matrix where each row represents a document, and each column represents a unique term in the vocabulary. The values in the matrix are the TF-IDF scores of each term in each document.

4. Transforming Test Data

- In this step, transform() is called on the test data (X_test) without fitting it again.
- This uses the vocabulary and IDF values learned from the training data to transform X_test into the same TF-IDF matrix format as X_train_vec. This ensures that the model receives input in a consistent format for both training and testing.



X_train.value_counts() would count the occurrences of each unique row in the X_train Series or DataFrame. This method is often used to understand the distribution of unique entries in a dataset. Here's a breakdown of how it works and where it might be used:

Explanation:

- **X_train**: This is typically your training data, which may contain text data or other features.
    - If X_train is a Series (like a single column of text data), X_train.value_counts() will count the number of times each unique value (e.g., a specific sentence or text snippet) appears in the training set.

- If X_train is a DataFrame, X_train.value_counts() will count the occurrences of each unique row in the entire DataFrame (this is more rare and only supported in pandas 1.1.0 and later).

Logistic Regression Model:

- lr is an instance of the LogisticRegression class from scikit-learn.
- LogisticRegression() initializes a logistic regression classifier.
- Logistic regression is a commonly used algorithm for classification tasks. Despite the name, it's mainly used to predict categorical outcomes.
- .fit() is the method used to train the model. Here, the logistic regression model learns from the training data.
- X_train_vec is the TF-IDF-transformed version of the training data. It's a matrix where each row represents a document, and each column represents a term in the vocabulary, with values representing the TF-IDF scores.
- y_train is the target labels (the correct classes for each document in X_train_vec), which the model uses to learn patterns in the data.
- y_pred stores the predictions made by the logistic regression model for each entry in the test set.
- X_test_vec is the test data that has been transformed using TfidfVectorizer, matching the format used to train the model.

CODE: pred_df = pd.DataFrame(data={'tested_comments': X_test, 'y_predicted': y_pred, 'y_actual': y_test})

This line creates a new DataFrame called pred_df, which helps you see how the model's predictions compare to the actual test labels.

tested_comments contains the original test comments from X_test (i.e., the text data before transformation).

y_predicted contains the model's predictions for each comment.

y_actual contains the actual labels for each comment, so you can see whether the prediction (y_predicted) matches the true label (y_actual).

CODE: pred_df.head()

This pred_df DataFrame provides a quick and interpretable way to:

- Check the accuracy of your model by visually comparing predictions.
- Analyze specific misclassifications by examining the text in tested_comments where y_predicted and y_actual differ.
- .head() displays the first few rows of pred_df to give you a quick look at how well the model performed on the test data.

Code Breakdown

- confusion_matrix and classification_report are imported from sklearn.metrics, which provides tools for evaluating classification models.
    - **confusion_matrix**: Provides a matrix that shows the counts of true positive, true negative, false positive, and false negative predictions.

    - **classification_report**: Generates a report that includes precision, recall, and F1 score for each class.
- y_actual_arr and y_pred_arr convert y_test and y_pred into numpy arrays. This step isn't strictly necessary (since confusion_matrix can handle lists or pandas Series directly), but it can sometimes ensure compatibility with certain metrics functions.
- cm stores the confusion matrix, which compares the actual labels (y_actual_arr) and predicted labels (y_pred_arr).
- This matrix helps in evaluating how well the model performed by showing the count of correct and incorrect predictions for each class.

Interpreting the Confusion Matrix

The confusion matrix cm will be a 2x2 matrix if this is a binary classification problem. It has the following format:

True Negatives False Negatives

False Positives True Positives

This code visualizes the confusion matrix cm using a heatmap, which makes it easier to interpret the model's performance in a visually intuitive way. Here's a breakdown of each part:

Code : import seaborn as sns

import matplotlib.pyplot as plt

- seaborn is a data visualization library built on top of Matplotlib that provides an easier and more visually appealing way to create plots.
- matplotlib.pyplot is the standard plotting library in Python.

Code : sns.heatmap(cm, annot=True, fmt='g', xticklabels=['cyberbullying', 'Not cyberbullying'], yticklabels=['cyberbullying', 'Not cyberbullying'])

- sns.heatmap(cm, ...) creates a heatmap based on the confusion matrix cm.
- annot=True displays the values in each cell of the heatmap.
- fmt='g' formats the annotations as integers instead of scientific notation.
- xticklabels and yticklabels provide labels for the x-axis (predicted classes) and y-axis (actual classes) to make it clear which class each axis represents.

Code : plt.ylabel('Actual', fontsize=13)

plt.title('Confusion Matrix', fontsize=17, pad=20)

- plt.ylabel('Actual', fontsize=13) labels the y-axis as "Actual" with font size 13.
- plt.title('Confusion Matrix', fontsize=17, pad=20) gives the plot a title with font size 17 and some padding above the title.

Code : plt.gca().xaxis.set_label_position('top')

plt.xlabel('Prediction', fontsize=13)

plt.gca().xaxis.tick_top()

- plt.gca() gets the current axis for customization.
- xaxis.set_label_position('top') and xaxis.tick_top() place the x-axis label and ticks at the top of the plot, which is often helpful for confusion matrices.

Code : plt.gca().figure.subplots_adjust(bottom=0.2)

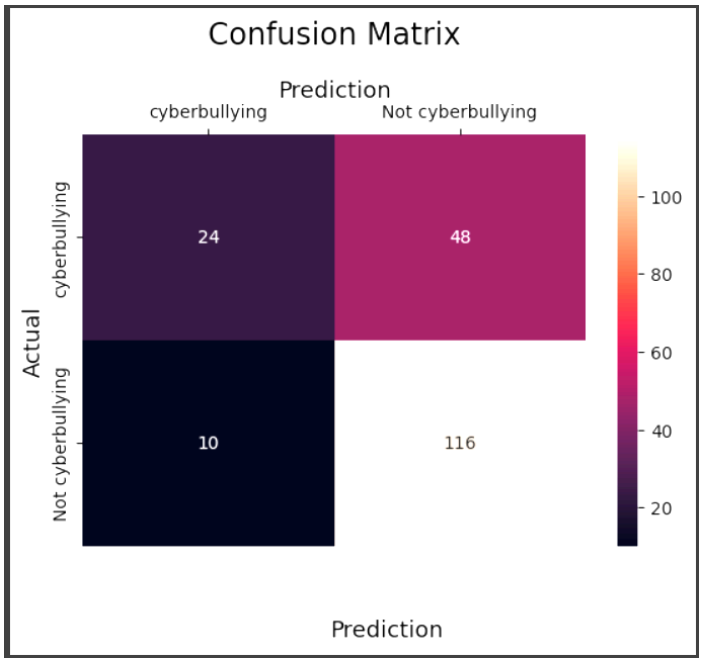
plt.gca().figure.text(0.5, 0.05, 'Prediction', ha='center', fontsize=13)

plt.show()

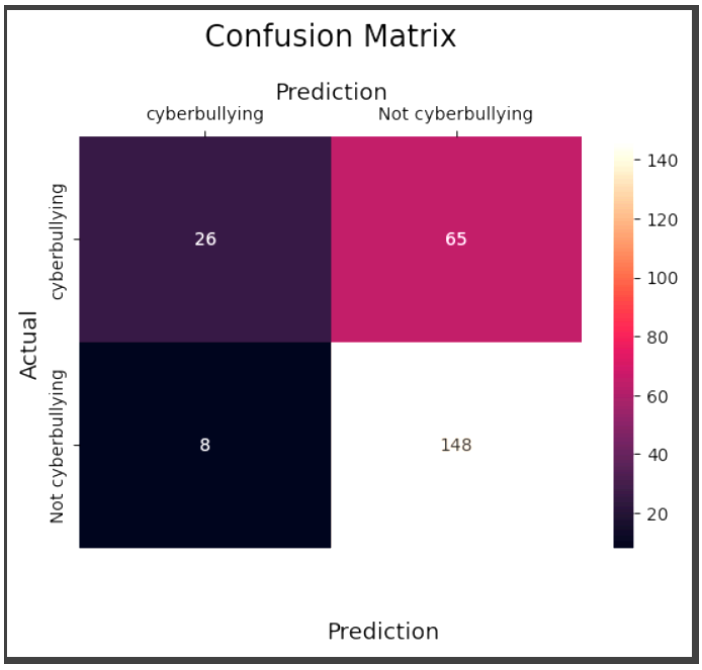
- subplots_adjust(bottom=0.2) provides a bit of space at the bottom of the plot.
- figure.text(...) adds a text label "Prediction" below the plot for additional clarity.
- plt.show() displays the heatmap.

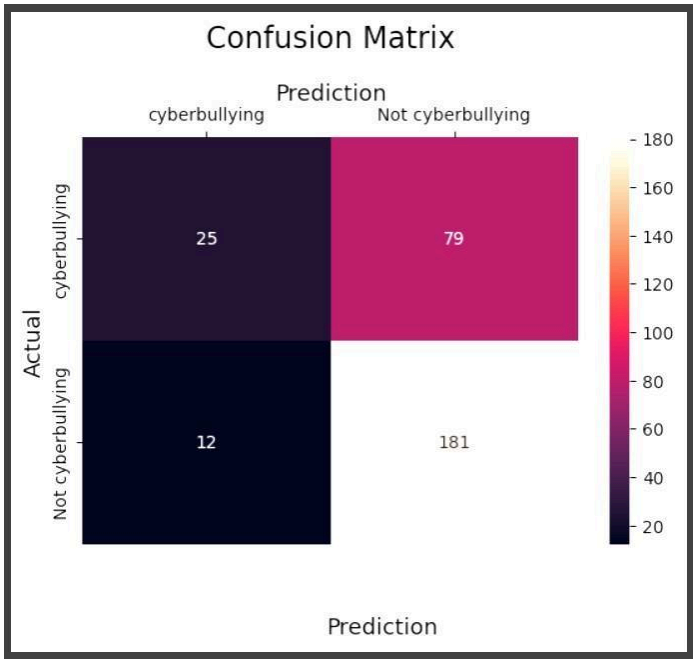
CONFUSION MATRIX:

TEST SIZE : 0.2

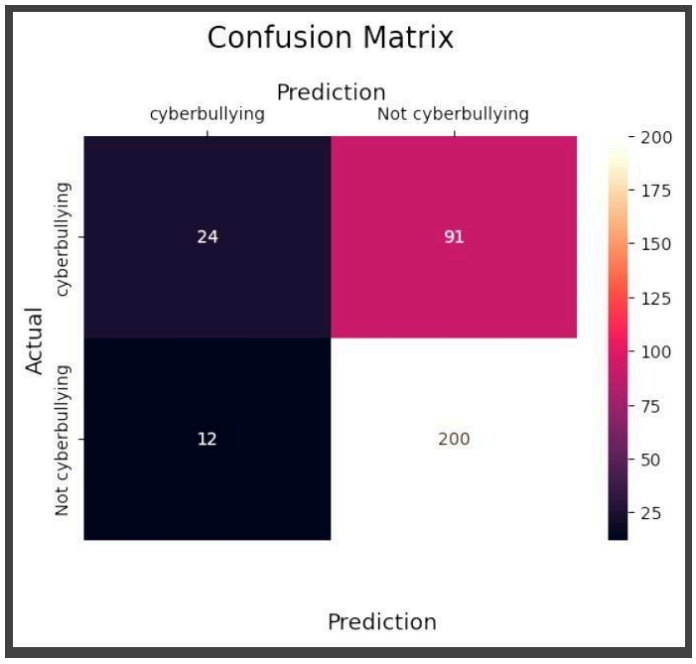


TEST SIZE : 0.25





TEST SIZE : 0.3



TEST SIZE : 0.33

The classification_report provides detailed metrics for evaluating the performance of a classification model on each class.

CODE: print(classification_report(y_actual_arr, y_pred_arr))

Output Metrics Explained

1. **Precision**: Measures the proportion of true positive predictions among all positive predictions (i.e., the accuracy of positive predictions).
    - Formula: Precision = True Positives/(TruePositives + False Positives)
    - A high precision score for a class indicates that when the model predicts that class, it is often correct.
2. **Recall**: Measures the proportion of true positives identified among all actual positives (i.e., the ability to find all positive instances).
    - Formula: Recall=True Positives/(True Positives + False Negatives)
    - A high recall score for a class indicates that the model is good at identifying most instances of that class.
3. **F1 Score**: The harmonic mean of precision and recall. It's a balanced measure that gives insight into the model's performance across both metrics.
    - Formula: F1 Score = 2× (Precision × Recall) / (Precision + Recall)
    - A high F1 score indicates a good balance between precision and recall.
4. **Support**: The number of actual instances in each class. It helps give context to precision and recall scores by showing how many true examples of each class were in the dataset.

The classification_report provides a comprehensive view of model performance, especially useful when evaluating a classifier in a multi-class or imbalanced data context.

The accuracy_score function calculates the overall accuracy of the model's predictions, which is the ratio of correct predictions to the total number of predictions.

accuracy_score(y_test, y_pred) computes the accuracy by comparing the true labels (y_test) to the predicted labels (y_pred).

Accuracy = Number of Correct Predictions / Total Number of Predictions

ACCURACY RATES:

TEST SIZE : 0.2 - 70.70 %

```
[33] print(classification_report(y_actul_arr, y_pred_arr))

              precision    recall  f1-score   support

         0.0       0.71      0.33      0.45        72
         1.0       0.71      0.92      0.80       126

    accuracy                           0.71       198
   macro avg       0.71      0.63      0.63       198
weighted avg       0.71      0.71      0.67       198

from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)

0.7070707070707071
```

TEST SIZE : 0.25 - 70.45 %

```
print(classification_report(y_actul_arr, y_pred_arr))

              precision    recall  f1-score   support

         0.0       0.76      0.29      0.42        91
         1.0       0.69      0.95      0.80       156

    accuracy                           0.70       247
   macro avg       0.73      0.62      0.61       247
weighted avg       0.72      0.70      0.66       247

from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)

0.7044534412955465
```

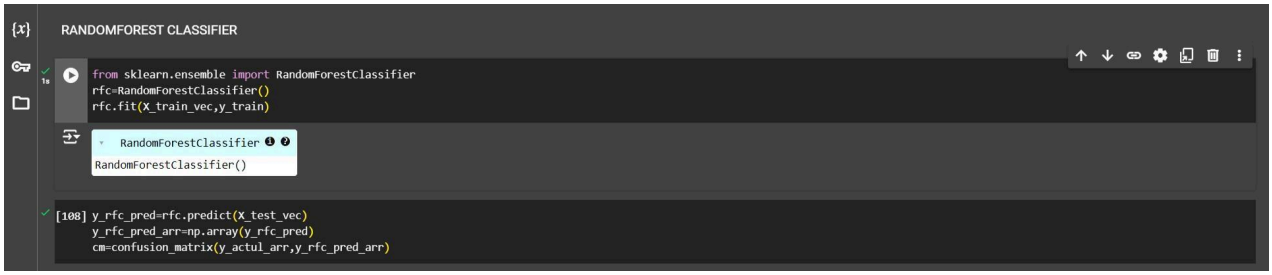
TEST SIZE : 0.3 - 69.36 %

```
[69] print(classification_report(y_actul_arr, y_pred_arr))

              precision    recall  f1-score   support

         0.0       0.68      0.24      0.35       104
         1.0       0.70      0.94      0.80       193

    accuracy                           0.69       297
   macro avg       0.69      0.59      0.58       297
weighted avg       0.69      0.69      0.64       297

from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)

0.6936026936026936
```

TEST SIZE: 0.33 - 68.50 %

```
[105] print(classification_report(y_actul_arr, y_pred_arr))

              precision    recall  f1-score   support

         0.0       0.67      0.21      0.32       115
         1.0       0.69      0.94      0.80       212

    accuracy                           0.69       327
   macro avg       0.68      0.58      0.56       327
weighted avg       0.68      0.69      0.63       327

from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)

0.6850152905198776
```

Random Forest :



Code Explanation :

- RandomForestClassifier() initializes a random forest model, a type of ensemble method that combines multiple decision trees to improve classification performance.
- The RandomForestClassifier can handle high-dimensional data and is robust to overfitting.

CODE: rfc.fit(X_train_vec, y_train)

- This trains the random forest classifier (rfc) on the TF-IDF-transformed training data (X_train_vec) and corresponding labels (y_train).
- Each decision tree in the forest will learn patterns in the TF-IDF feature space to distinguish between classes.

CODE: y_rfc_pred = rfc.predict(X_test_vec)

- This line generates predictions for the test data (X_test_vec) using the trained random forest model.
- The result, y_rfc_pred, contains the predicted class labels for each entry in the test set.

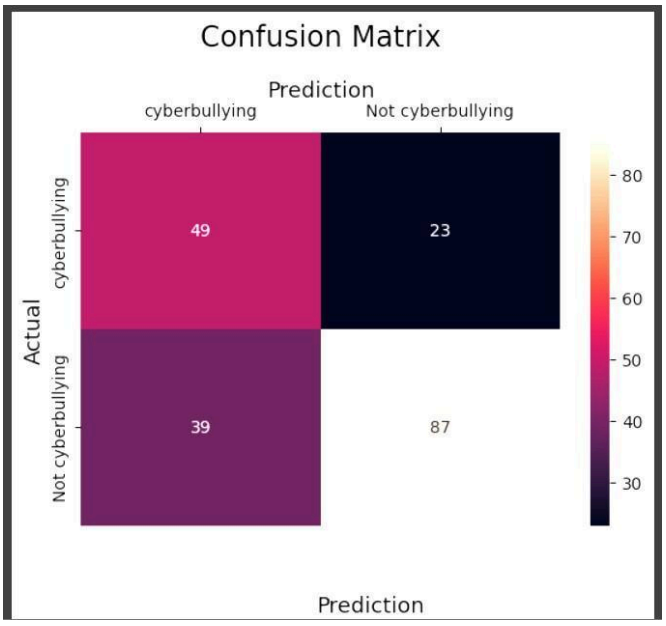
CODE: y_rfc_pred_arr = np.array(y_rfc_pred)

- Converts y_rfc_pred to a numpy array, y_rfc_pred_arr, which can make it easier to compare with y_actual_arr in certain metrics.

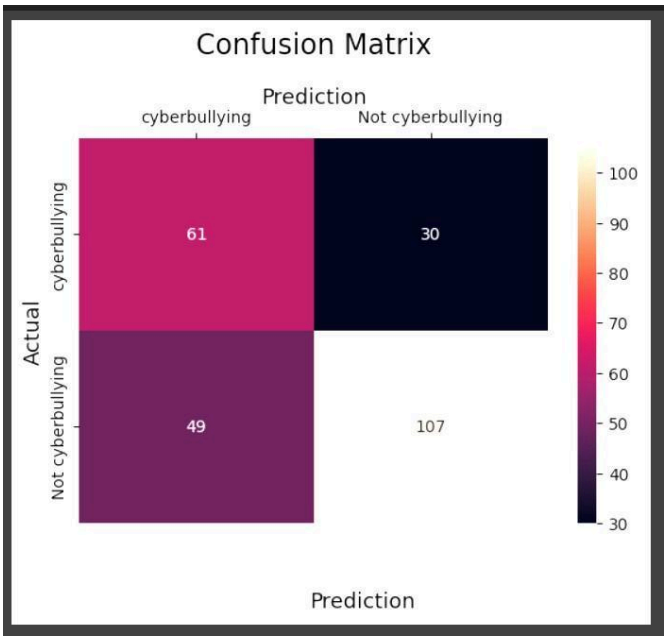
CODE: cm = confusion_matrix(y_actual_arr, y_rfc_pred_arr)

- cm stores the confusion matrix for the random forest predictions compared to the actual test labels.
- This matrix helps in analyzing the model's performance in terms of true positives, true negatives, false positives, and false negatives.

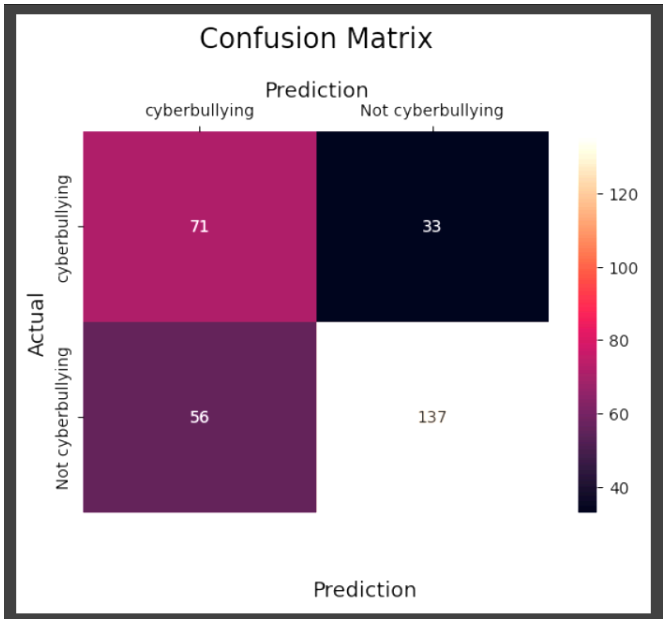
CONFUSION MATRIX :



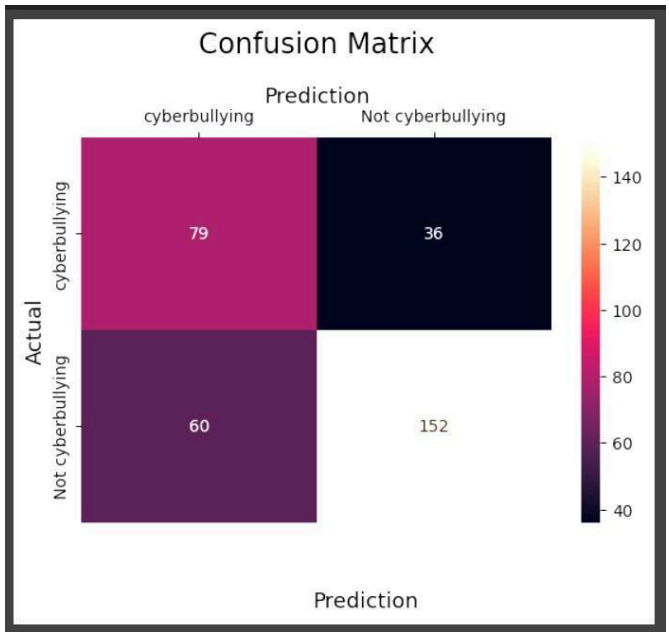
TEST SIZE : 0.2



TEST SIZE:0.25



TEST SIZE : 0.3



TEST SIZE : 0.33

ACCURACY RATES :

TEST SIZE : 0.2 - 68.68 %

TEST SIZE : 0.25 - 68.01%

TEST SIZE : 0.3 - 70.03 %

TEST SIZE : 0.33 - 71.25

NAIVE BAYES CLASSIFICATION

CODE:mnb = MultinomialNB()

- MultinomialNB: A Naive Bayes classifier that works well for text data. It is suited for classification tasks where features represent frequencies or counts (e.g., word counts in text).

CODE:mnb.fit(X_train_vec, y_train)

- `fit` method: Trains the Naive Bayes model using the training data (`X_train_vec`) and corresponding labels (`y_train`).

CODE:y_pred_mnb = mnb.predict(X_test_vec)

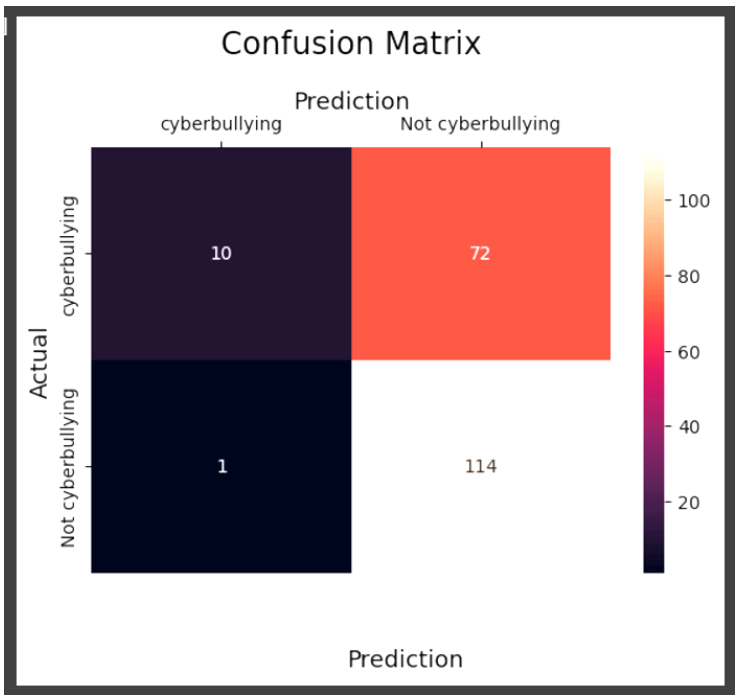
- `predict` method: Uses the trained model to predict labels for the test data (`X_test_vec`).

CODE:print("\nNaive Bayes Performance:")
print("Accuracy: ", accuracy_score(y_test, y_pred_mnb))
print(classification_report(y_test, y_pred_mnb))

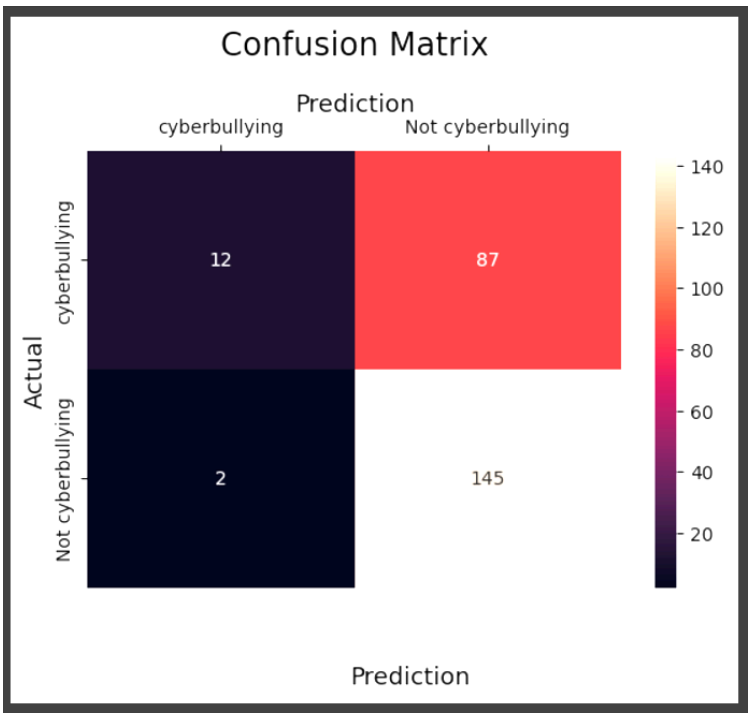
- `accuracy_score`: Calculates the ratio of correctly predicted labels to the total number of test samples.
- `classification_report`: Provides a detailed performance summary including precision, recall, F1-score, and support for each class.

CODE:plot_confusion_matrix(y_test,y_pred_mnb,['cyberbullying','Notcyberbullying'],'Naive Bayes')

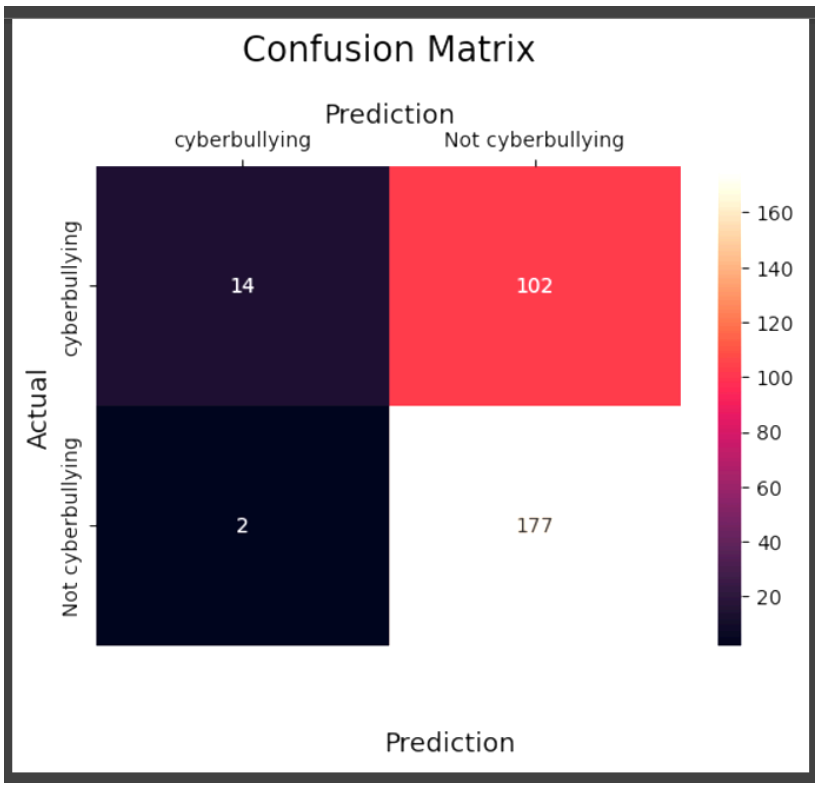
- Custom `plot_confusion_matrix` function: Visualizes a confusion matrix to understand how well the model performs in terms of true positives, false positives, true negatives, and false negatives for the two classes: 'cyberbullying' and 'Not cyberbullying'.



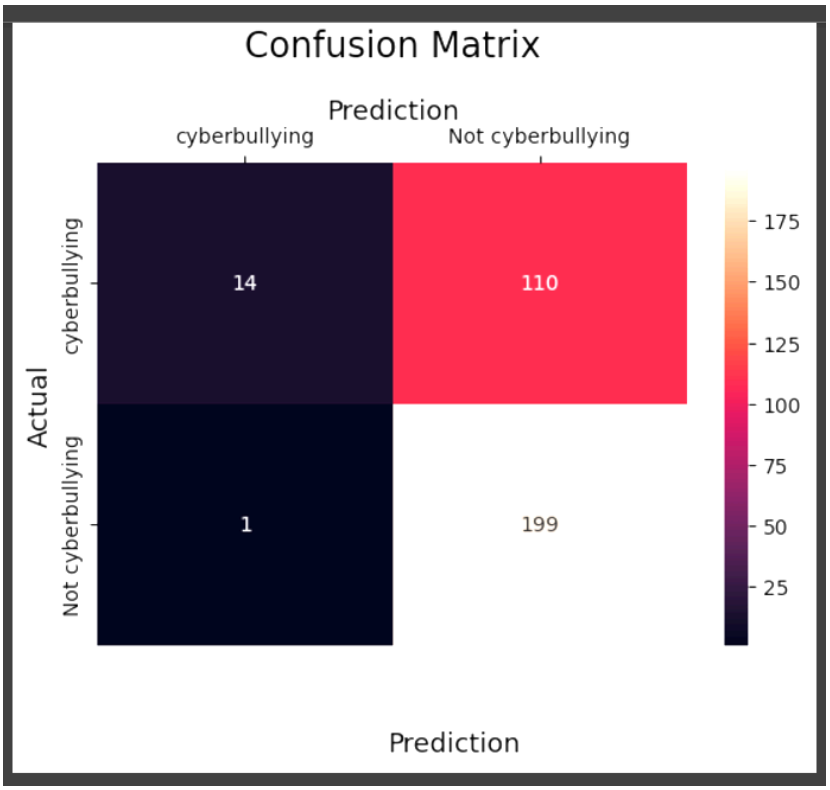
TEST SIZE : 0.2 - 62.94 %



TEST SIZE: 0.25 - 63.82 %

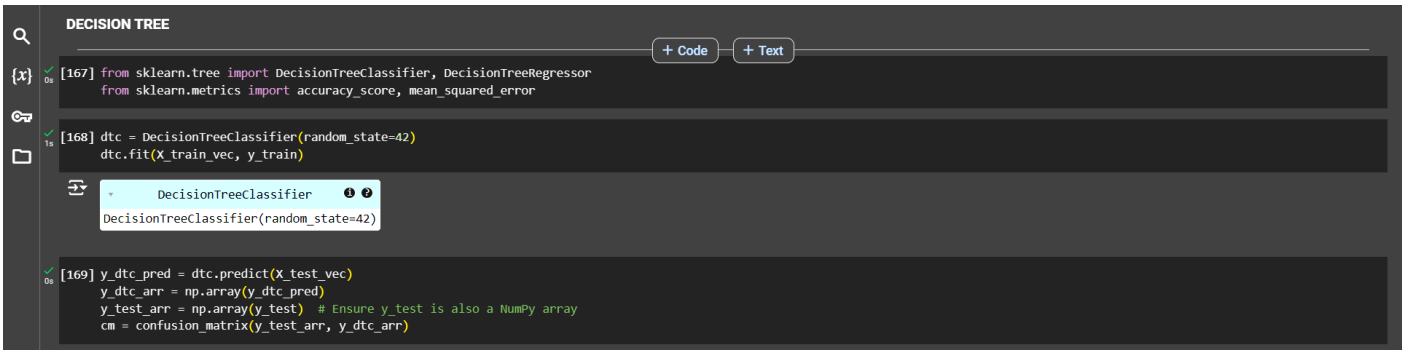


TEST SIZE: 0.3 - 64.74 %



TEST SIZE: 0.33 - 65.74 %

DECISION TREE CLASSIFIER



CODE:dtc = DecisionTreeClassifier(random_state=42)

- DecisionTreeClassifier: A classifier that splits data into subgroups based on feature thresholds, building a tree-like model for classification.
- `random_state`: Ensures reproducibility by fixing the random seed.

CODE:dtc.fit(X_train_vec, y_train)

- Trains the Decision Tree model with the training data and labels.

CODE:y_pred_dtc = dtc.predict(X_test_vec)

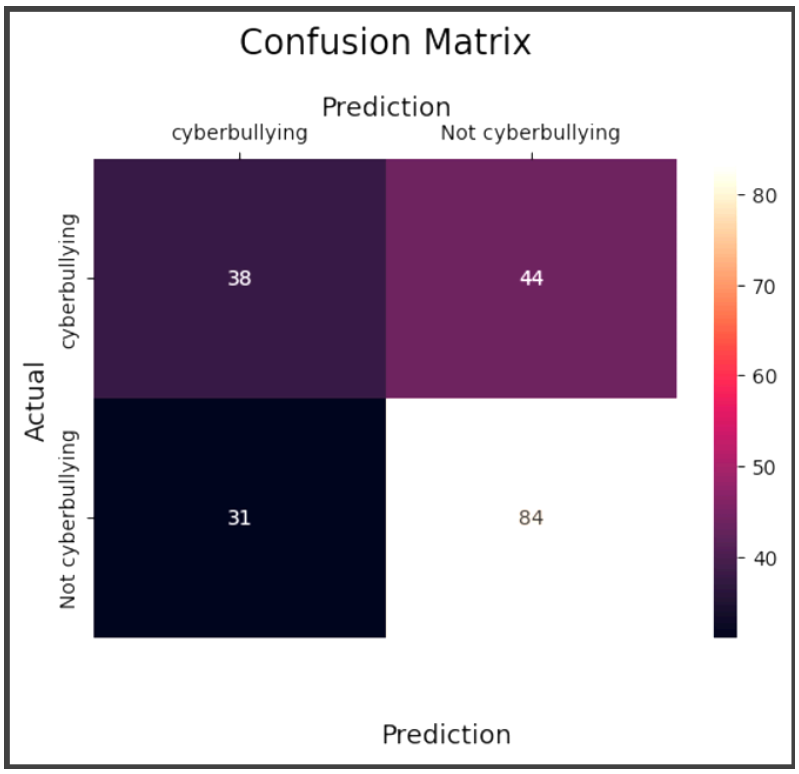
- Predicts the test labels using the trained Decision Tree model.

CODE:print("\nDecision Tree Performance:")
print("Accuracy: ", accuracy_score(y_test, y_pred_dtc))
print(classification_report(y_test, y_pred_dtc))

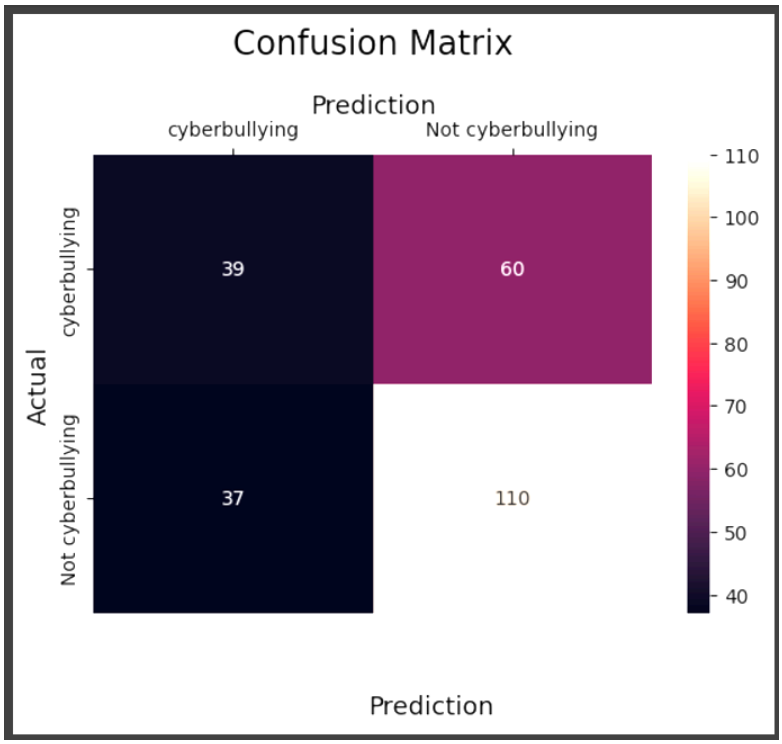
- Similar to the Naive Bayes section, evaluates and prints accuracy and classification report for the Decision Tree model.

CODE:plot_confusion_matrix(y_test, y_pred_dtc, ['cyberbullying', 'Not cyberbullying'], 'Decision Tree')

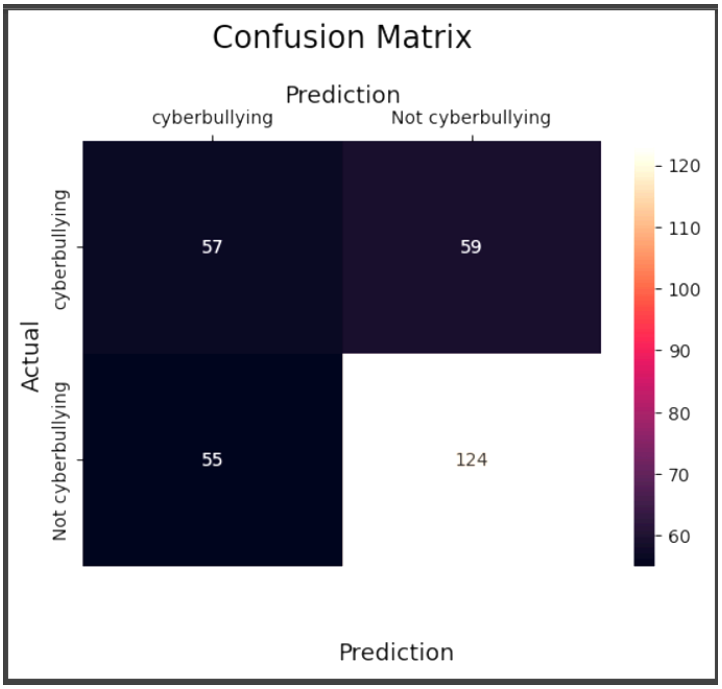
- Visualizes the confusion matrix for Decision Tree predictions.



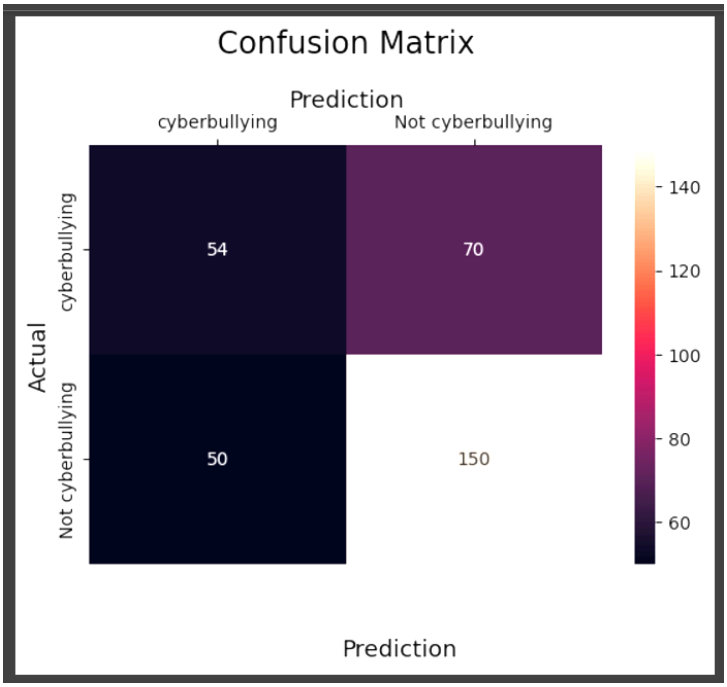
TEST SIZE:0.2 - 61.97%



TEST SIZE:0.25 - 60.56%



TEST SIZE: 0.3 - 61.35%



TEST SIZE: 0.33 - 62.96%

SUPPORT VECTOR MACHINE

CODE:svc = SVC(kernel='linear', C=1.0, random_state=42)

- SVC: Support Vector Classifier. It tries to find the optimal hyperplane that separates classes in the feature space.
- `kernel='linear'`: Specifies a linear kernel, which works well for linearly separable data.
- `C=1.0`: Regularization parameter; balances maximizing margin and minimizing classification error.
- `random_state`: Fixes randomness for reproducibility.

CODE:svc.fit(X_train_vec, y_train)

- Trains the SVM model using the training data and labels.

CODE:y_pred_svc = svc.predict(X_test_vec)

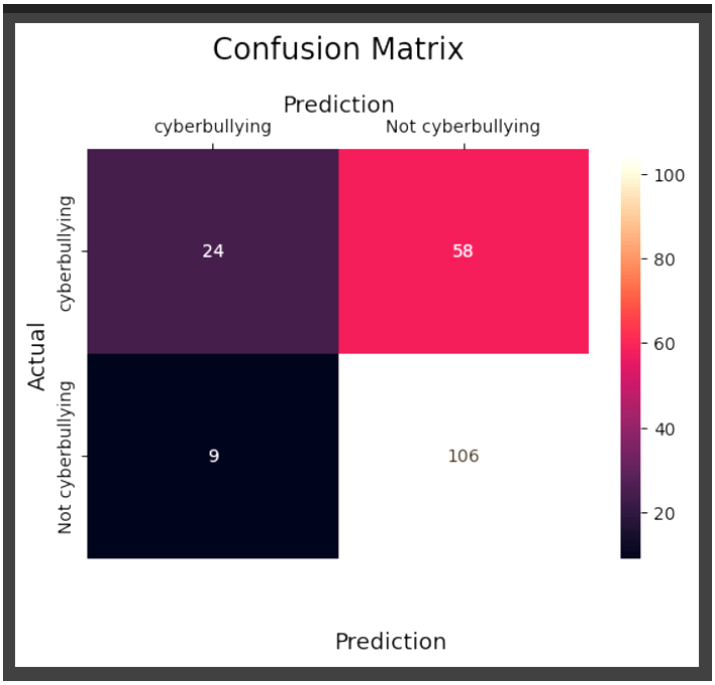
- Predicts labels for the test data using the trained SVM model.

CODE:print("\nSupport Vector Machine Performance:")
print("Accuracy: ", accuracy_score(y_test, y_pred_svc))
print(classification_report(y_test, y_pred_svc))

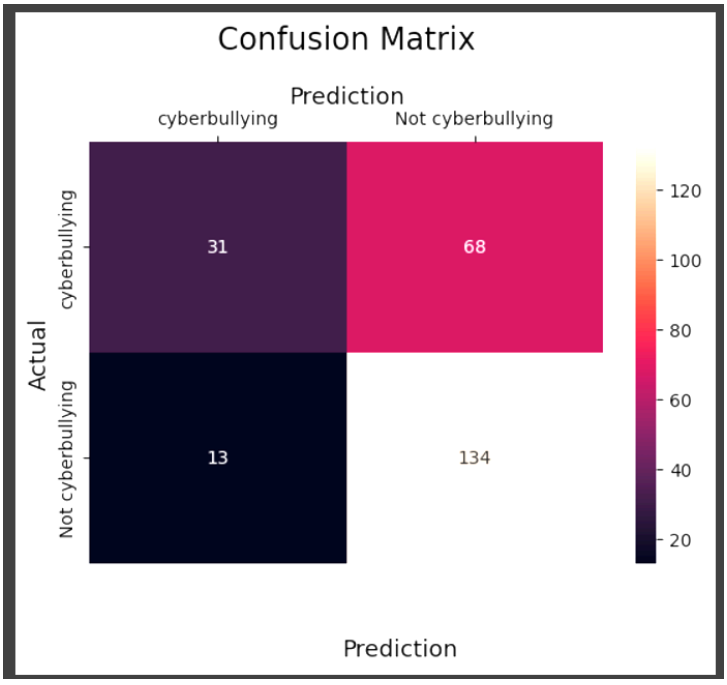
- Prints accuracy and a classification report for SVM predictions.

CODE:plot_confusion_matrix(y_test, y_pred_svc, ['cyberbullying', 'Not cyberbullying'], 'Support Vector Machine')

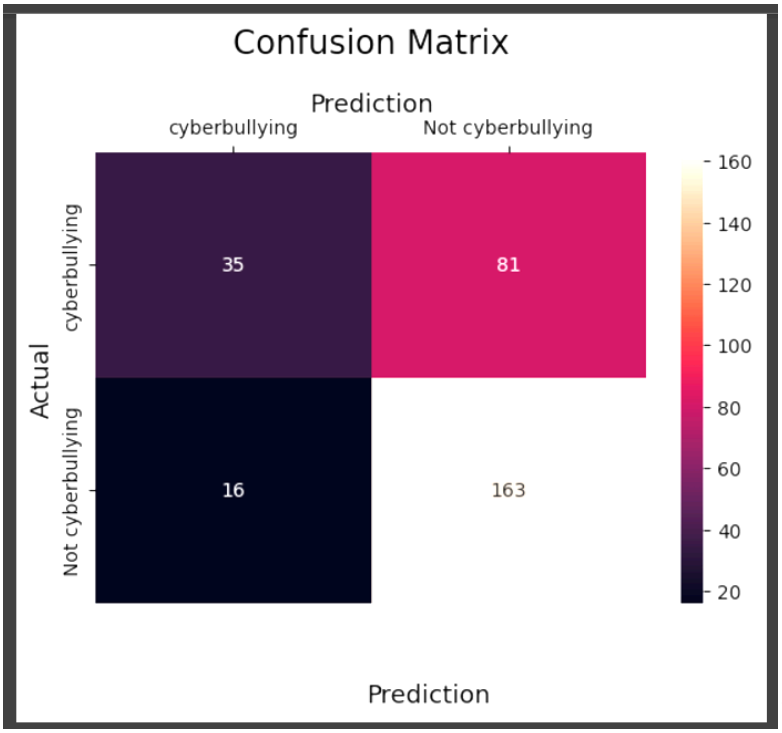
- Visualizes the confusion matrix for SVM.



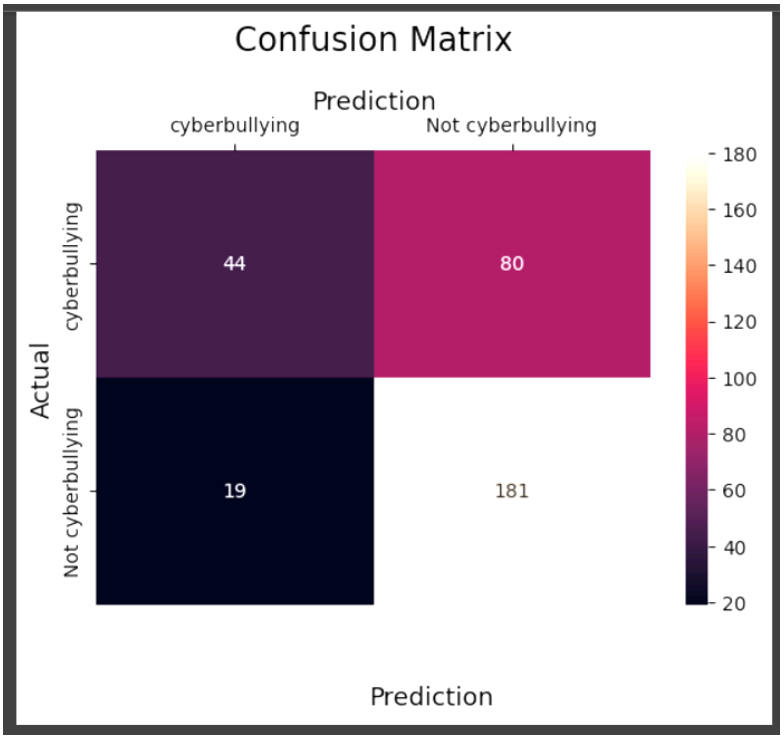
TEST SIZE: 0.2 - 61.92%



TEST SIZE: 0.25 - 60.56%

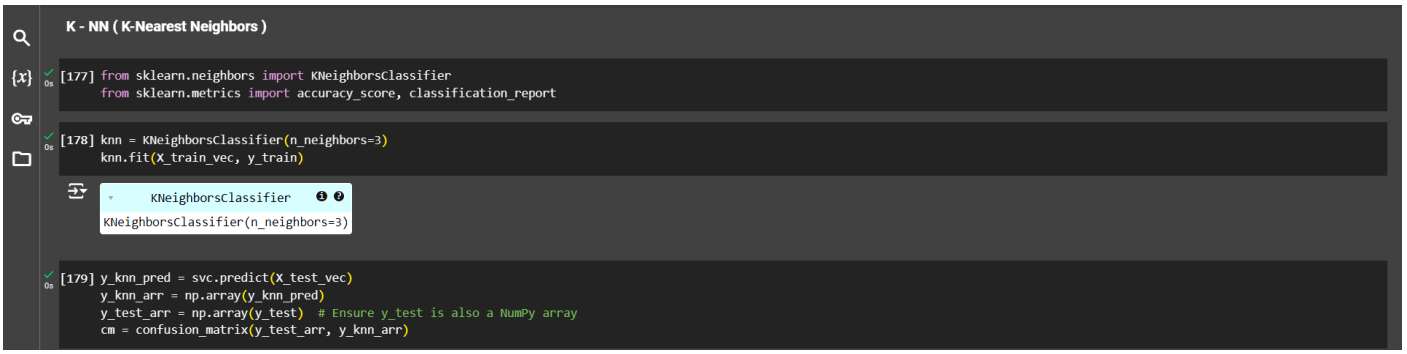


TEST SIZE: 0.3 - 61.35%



TEST SIZE: 0.33 - 62.96%

K-NEAREST NEIGHBOUR(KNN)



CODE:knn = KNeighborsClassifier(n_neighbors=3)

- KNeighborsClassifier: A lazy learning algorithm that classifies based on the majority vote of `k` nearest neighbors in the feature space.
- `n_neighbors=3`: Specifies `k=3`, meaning the algorithm considers the 3 closest neighbors to classify a data point.

CODE:knn.fit(X_train_vec, y_train)

- Trains the KNN model by storing the training data (KNN doesn't "train" in the conventional sense).

CODE:y_pred_knn = knn.predict(X_test_vec)

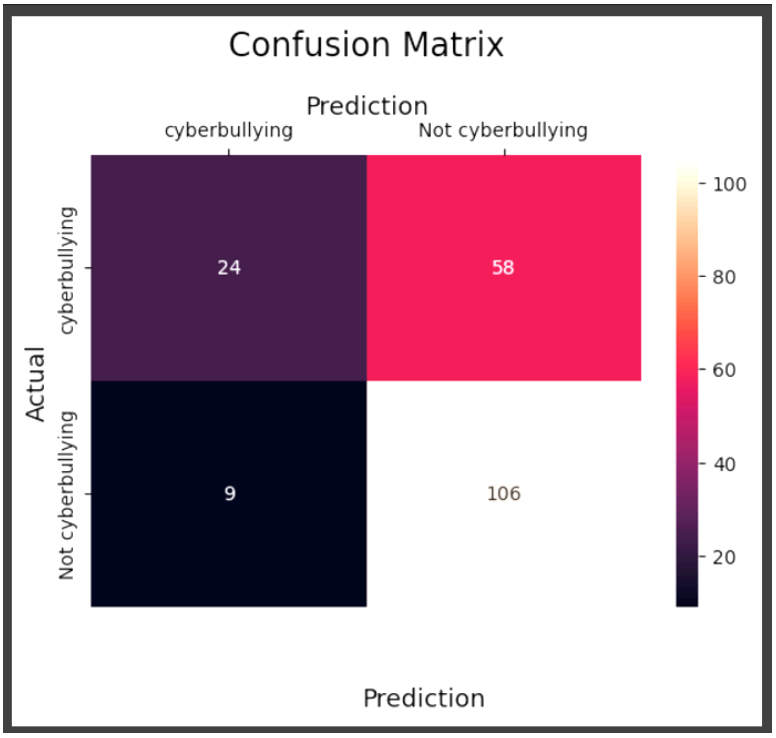
- Predicts labels for the test data by comparing distances to stored training samples.

CODE:print("\nK-Nearest Neighbors Performance:")
print("Accuracy: ", accuracy_score(y_test, y_pred_knn))
print(classification_report(y_test, y_pred_knn))

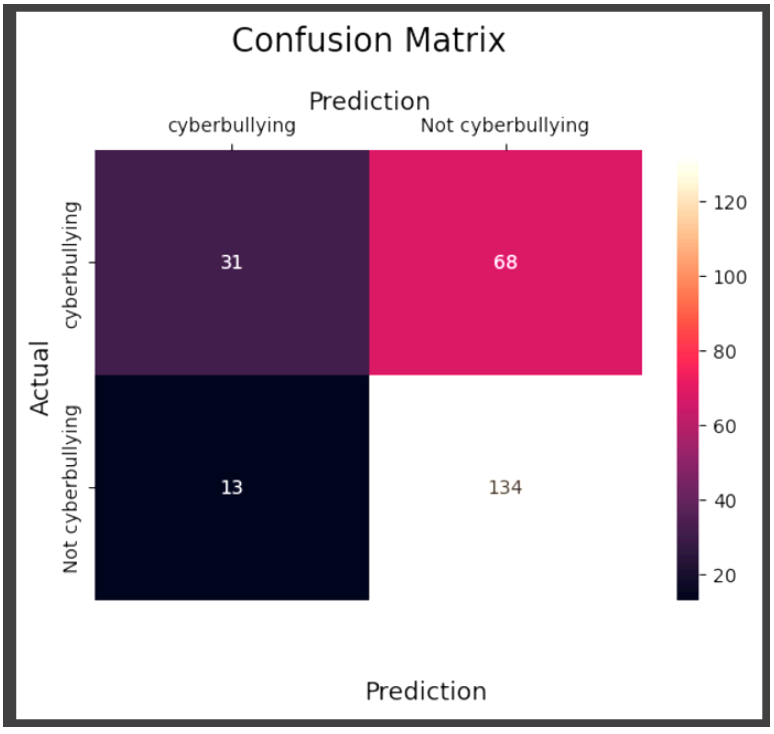
- Outputs the accuracy and classification report for KNN predictions.

CODE:plot_confusion_matrix(y_test, y_pred_knn, ['cyberbullying', 'Not cyberbullying'], 'K-Nearest Neighbors')

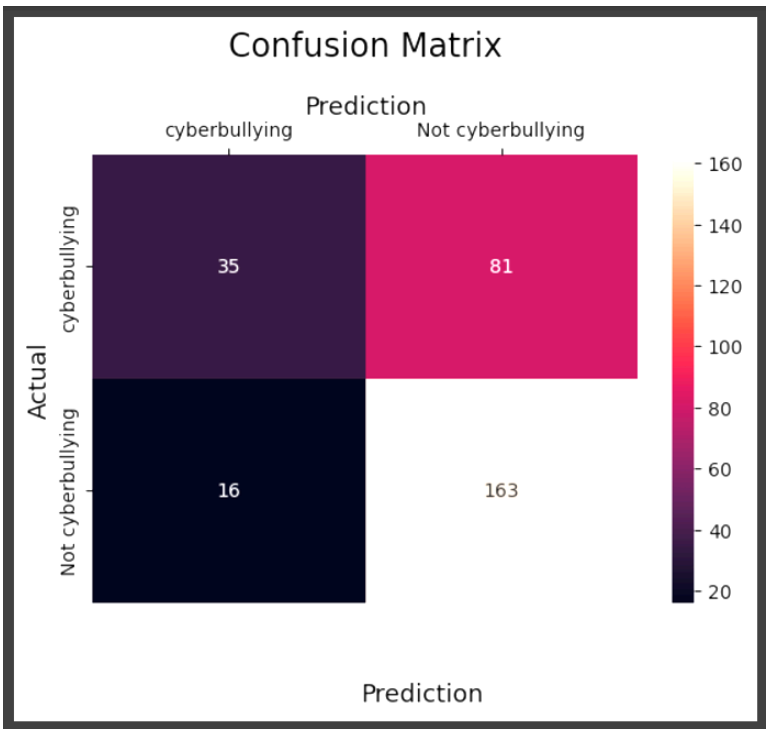
- Visualizes the confusion matrix for KNN predictions.



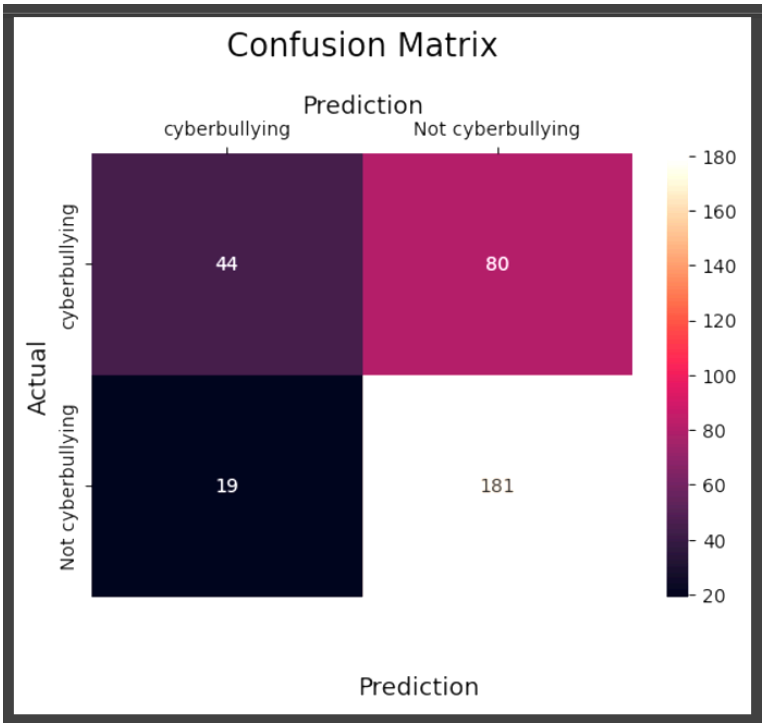
TEST SIZE: 0.2 - 65.98%



TEST SIZE:0.25 - 67.07%



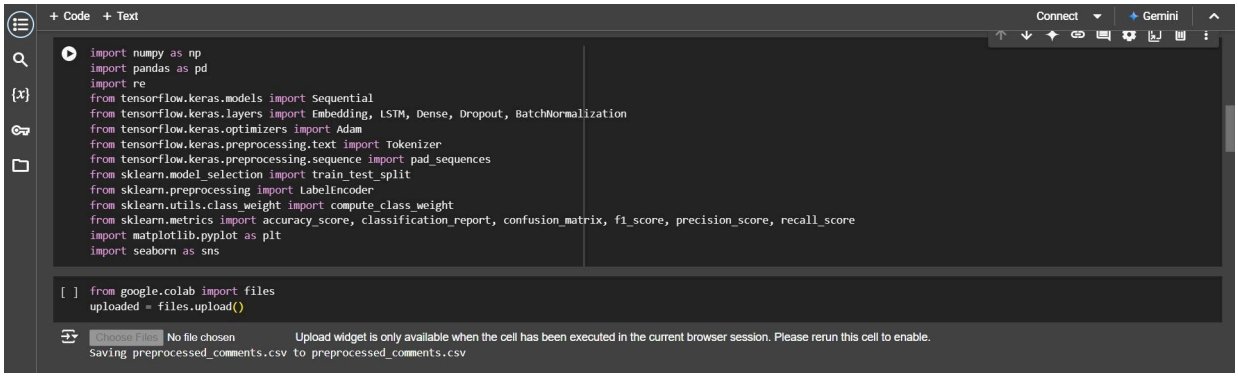
TEST SIZE: 0.3 - 67.11%



TEST SIZE:0.33 - 69.44%

MILESTONE 3 : Hybrid Deep Learning Model Architecture

LSTM (LONG SHORT TERM MEMORY NETWORKS)



---

1. **Install Required Libraries**

   CODE: pip install tensorflow numpy matplotlib scikit-learn

- Installs essential libraries like TensorFlow (for the LSTM model), NumPy, Matplotlib, and scikit-learn.

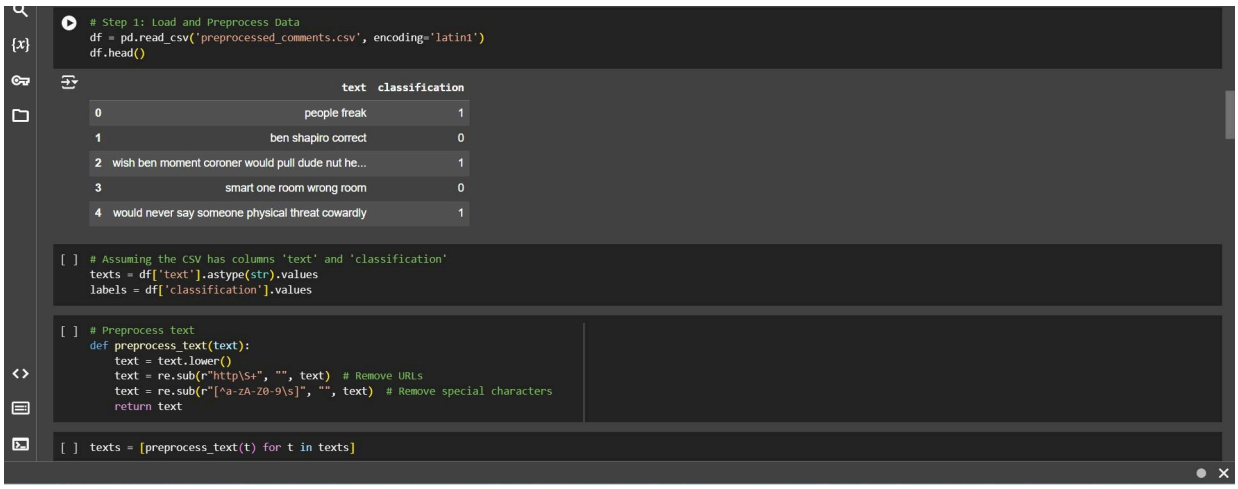
---

2. **Import Libraries**

   CODE:

```
import numpy as np
import pandas as
pd import re
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout,
BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.utils.class_weight import compute_class_weight
from sklearn.metrics import accuracy_score,
classification_report, confusion_matrix, f1_score, precision_score, recall_score
import matplotlib.pyplot as plt
import seaborn as sns
```

- Imports libraries for data manipulation (`numpy`, `pandas`), text preprocessing (`re`), model building (`tensorflow.keras`), evaluation metrics (`sklearn`), and visualization (`matplotlib`, `seaborn`).



```
# Step 1: Load and Preprocess Data
df = pd.read_csv('preprocessed_comments.csv', encoding='latin1')
df.head()
```

	text	classification
0	people freak	1
1	ben shapiro correct	0
2	wish ben moment coroner would pull dude nut he...	1
3	smart one room wrong room	0
4	would never say someone physical threat cowardly	1

```
# Assuming the CSV has columns 'text' and 'classification'
texts = df['text'].astype(str).values
labels = df['classification'].values
```

```
# Preprocess text
def preprocess_text(text):
    text = text.lower()
    text = re.sub(r"http\S+", "", text)  # Remove URLs
    text = re.sub(r"[^a-zA-Z0-9\s]", "", text)  # Remove special characters
    return text
```

```
texts = [preprocess_text(t) for t in texts]
```

3. **Upload and Read Data**

   CODE:

   ```
   from google.colab import files
   uploaded = files.upload()
   df = pd.read_csv('preprocessed_comments.csv', encoding='latin1')
   ```

- Uses Google Colab's file uploader to upload the `preprocessed_comments.csv` file.
- Reads the uploaded CSV into a Pandas DataFrame (`df`) for processing.

---

4. **Extract Text and Labels**

   CODE:

   ```
   texts =
   df['text'].astype(str).values labels
   = df['classification'].values
   ```

- Extracts the `text` column (comments) and `classification` column (labels) from the DataFrame.

---

5. **Preprocess Text**

CODE:

```
def preprocess_text(text):
    text = text.lower()
    text = re.sub(r"http\S+", "", text) # Remove URLs
    text = re.sub(r"[^a-zA-Z0-9\s]", "", text) # Remove special characters
    return text

texts = [preprocess_text(t) for t in texts]
```

- Defines a function to:
    - Convert text to lowercase.
    - Remove URLs using a regex pattern.
    - Remove non-alphanumeric characters.
- Applies this preprocessing function to all comments.

```
# Encode labels
label_encoder = LabelEncoder()
labels = label_encoder.fit_transform(labels)

# Tokenize and pad the text data
max_words = 10000  # Maximum vocabulary size
max_len = 100      # Maximum sequence length

tokenizer = Tokenizer(num_words=max_words, oov_token="<OOV>")
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
pre_padded_sequences = pad_sequences(sequences, maxlen=max_len, padding='pre', truncating='pre')  # Changed to 'pre'
```

```
# Split data into training and testing sets
X_train_pre, X_test_pre, y_train_pre, y_test_pre = train_test_split(
    pre_padded_sequences, labels, test_size=0.33, random_state=42
)

# Compute class weights
class_weights = compute_class_weight('balanced', classes=np.unique(y_train_pre), y=y_train_pre)
class_weights = dict(enumerate(class_weights))
```

6. **Encode Labels**

CODE:

```
label_encoder = LabelEncoder()
labels = label_encoder.fit_transform(labels)
```

- Encodes the text labels into numerical values using `LabelEncoder` for model compatibility.

---

7. **Tokenize and Pad Sequences**

CODE:

```
max_words = 10000
max_len = 100

tokenizer = Tokenizer(num_words=max_words,
oov_token="<OOV>") tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
pre_padded_sequences = pad_sequences(sequences, maxlen=max_len,
padding='pre', truncating='pre')
```

- Limits the vocabulary to 10,000 words (`max_words`) and sequences to 100 words (`max_len`).
- Uses a tokenizer to:
  - Replace out-of-vocabulary words with `<OOV>`.
  - Convert each comment into a sequence of integers.
- Pads sequences to ensure uniform length using pre-padding.

---

8. **Split Data**

CODE:

```
X_train_pre, X_test_pre, y_train_pre, y_test_pre = train_test_split(
   pre_padded_sequences, labels, test_size=0.33, random_state=42
)
```

- Splits the data into training (67%) and testing (33%) sets using `train_test_split`.

---

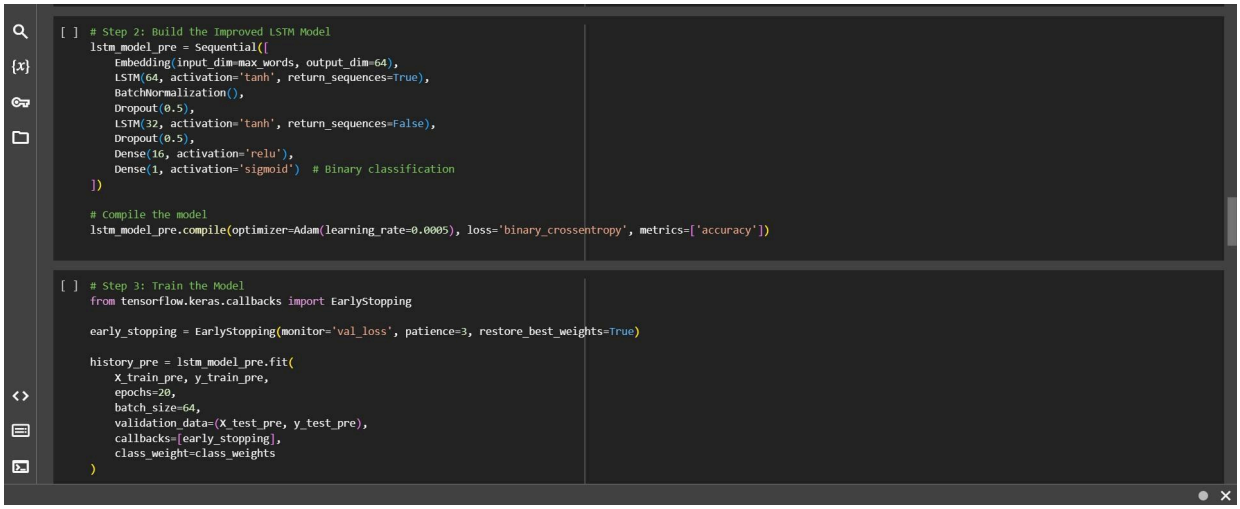
9. **Compute Class Weights**

CODE:

```
class_weights = compute_class_weight('balanced',
classes=np.unique(y_train_pre), y=y_train_pre)
class_weights = dict(enumerate(class_weights))
```

- Computes weights for classes to address class imbalance using `compute_class_weight`.

- Converts the result into a dictionary for model compatibility.



```
[ ]  # Step 2: Build the Improved LSTM Model
     lstm_model_pre = Sequential([
         Embedding(input_dim=max_words, output_dim=64),
         LSTM(64, activation='tanh', return_sequences=True),
         BatchNormalization(),
         Dropout(0.5),
         LSTM(32, activation='tanh', return_sequences=False),
         Dropout(0.5),
         Dense(16, activation='relu'),
         Dense(1, activation='sigmoid')  # Binary classification
     ])

     # Compile the model
     lstm_model_pre.compile(optimizer=Adam(learning_rate=0.0005), loss='binary_crossentropy', metrics=['accuracy'])
```

```
[ ]  # Step 3: Train the Model
     from tensorflow.keras.callbacks import EarlyStopping

     early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

     history_pre = lstm_model_pre.fit(
         X_train_pre, y_train_pre,
         epochs=20,
         batch_size=64,
         validation_data=(X_test_pre, y_test_pre),
         callbacks=[early_stopping],
         class_weight=class_weights
     )
```

10. Build LSTM Model

CODE:

```
lstm_model_pre = Sequential([
    Embedding(input_dim=max_words, output_dim=64),
    LSTM(64, activation='tanh', return_sequences=True),
    BatchNormalization(),
    Dropout(0.5),
    LSTM(32, activation='tanh', return_sequences=False),
    Dropout(0.5),
    Dense(16, activation='relu'),
    Dense(1, activation='sigmoid')
])
```

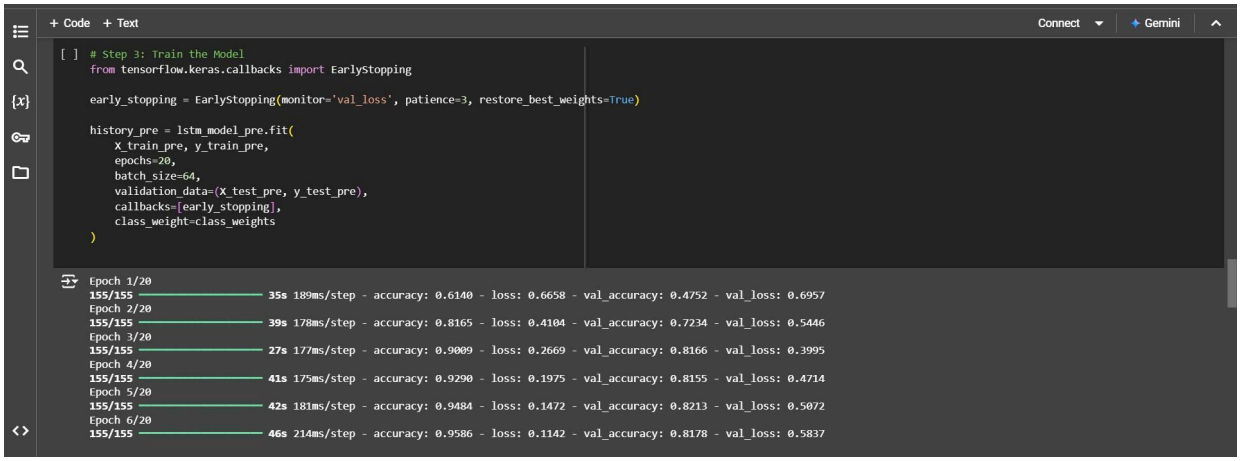
- Creates a sequential LSTM model with:
  - **Embedding layer**: Converts word indices into dense vectors.
  - **First LSTM layer**: Captures temporal dependencies; outputs sequences.
  - **BatchNormalization**: Stabilizes and accelerates training.
  - **Dropout**: Prevents overfitting.
  - **Second LSTM layer**: Outputs the final sequence representation.
  - **Dense layers**: Perform classification. The final layer uses `sigmoid` for binary output.

11. Compile the Model

CODE:

```
lstm_model_pre.compile(optimizer=Adam(learning_rate=0.0005), loss='binary_crossentropy', metrics=['accuracy'])
```

- Configures the model with:
  - `Adam` optimizer for adaptive learning.
  - `binary_crossentropy` as the loss function.
  - `accuracy` as the evaluation metric.

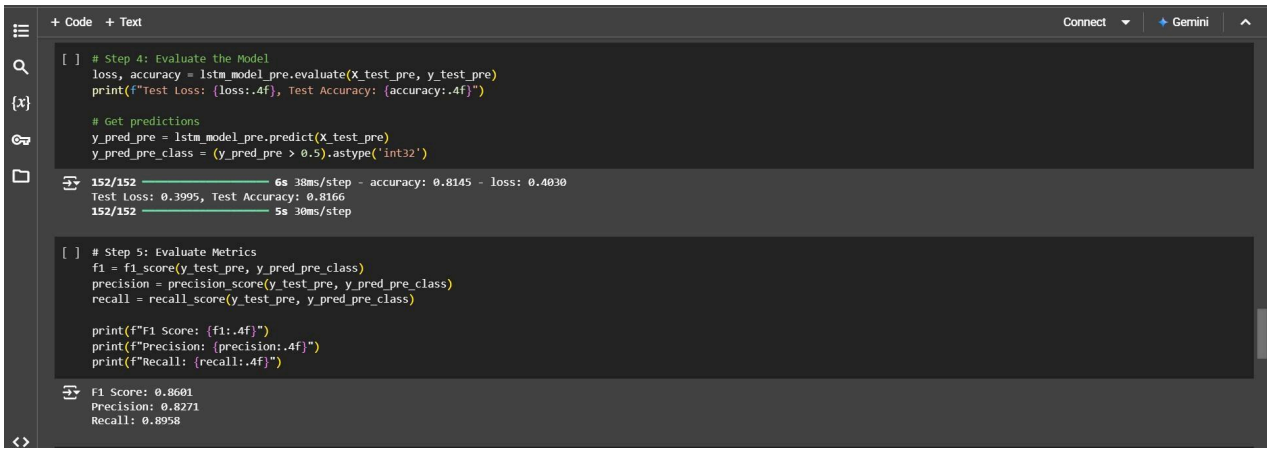


12. Train the Model

CODE:

```
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

history_pre = lstm_model_pre.fit(
    X_train_pre, y_train_pre,
    epochs=20,
    batch_size=64,
    validation_data=(X_test_pre, y_test_pre),
    callbacks=[early_stopping],
    class_weight=class_weights )
```

- Trains the model for up to 20 epochs with:
    - Batch size of 64.
    - Early stopping to prevent overfitting by monitoring validation loss.
    - `class_weights` to handle imbalance.



13. Evaluate the Model

CODE:

```
loss, accuracy = lstm_model_pre.evaluate(X_test_pre, y_test_pre)
```

- Evaluates the trained model on the test set and prints the loss and accuracy.

---

14. Get Predictions

CODE:

```
y_pred_pre = lstm_model_pre.predict(X_test_pre)
y_pred_pre_class = (y_pred_pre > 0.5).astype('int32')
```

- Gets the model's predictions and converts them to binary classifications using a 0.5 threshold.

---

15. Compute Metrics

CODE:

```
f1 = f1_score(y_test_pre, y_pred_pre_class)
precision = precision_score(y_test_pre, y_pred_pre_class)
recall = recall_score(y_test_pre, y_pred_pre_class)
```

- Computes the F1 score, precision, and recall for model evaluation.

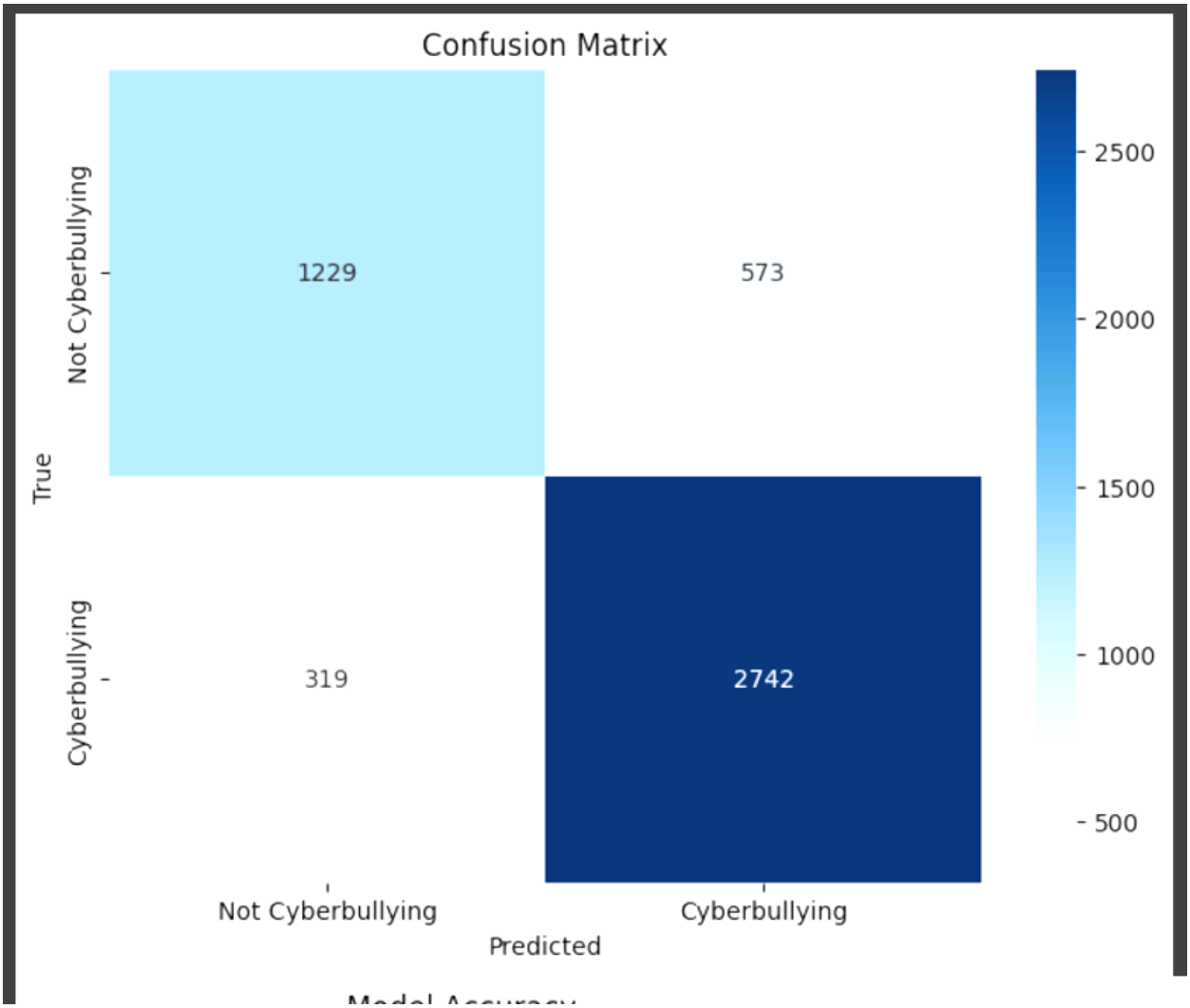
```
[ ] # Confusion Matrix
    conf_matrix_pre = confusion_matrix(y_test_pre, y_pred_pre_class)
    class_names = ['Not Cyberbullying', 'Cyberbullying']

    plt.figure(figsize=(8, 6))
    sns.heatmap(conf_matrix_pre, annot=True, fmt='d', cmap='Blues',
                xticklabels=class_names, yticklabels=class_names)
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.title('Confusion Matrix')
    plt.show()

    # Step 6: Visualize Training History
    plt.figure(figsize=(12, 5))

    # Accuracy Plot
    plt.subplot(1, 2, 1)
    plt.plot(history_pre.history['accuracy'], label='Training Accuracy')
    plt.plot(history_pre.history['val_accuracy'], label='Validation Accuracy')
    plt.title('Model Accuracy')
    plt.legend()

    # Loss Plot
    plt.subplot(1, 2, 2)
    plt.plot(history_pre.history['loss'], label='Training Loss')
    plt.plot(history_pre.history['val_loss'], label='Validation Loss')
    plt.title('Model Loss')
    plt.legend()

    plt.show()
```

16. Confusion Matrix

CODE:

```
conf_matrix_pre = confusion_matrix(y_test_pre, y_pred_pre_class)
class_names = ['Not Cyberbullying', 'Cyberbullying']

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_pre, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

- Plots a confusion matrix to visualize the model's predictions.

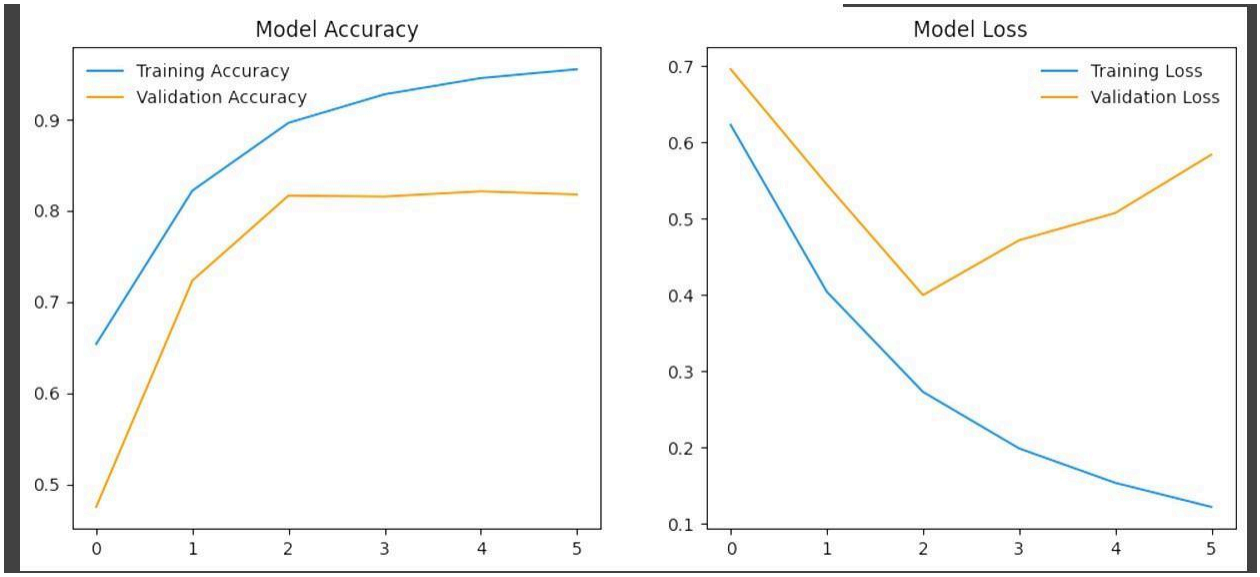


---

17. Visualize Training History

```
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history_pre.history['accuracy'], label='Training Accuracy')
plt.plot(history_pre.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.legend()
plt.subplot(1, 2,
2)
plt.plot(history_pre.history['loss'], label='Training Loss')
plt.plot(history_pre.history['val_loss'], label='Validation
Loss') plt.title('Model Loss')
plt.legend()
plt.show()
```

- Plots training and validation accuracy and loss over epochs.



---

18. **Save and Download the Model**

CODE:

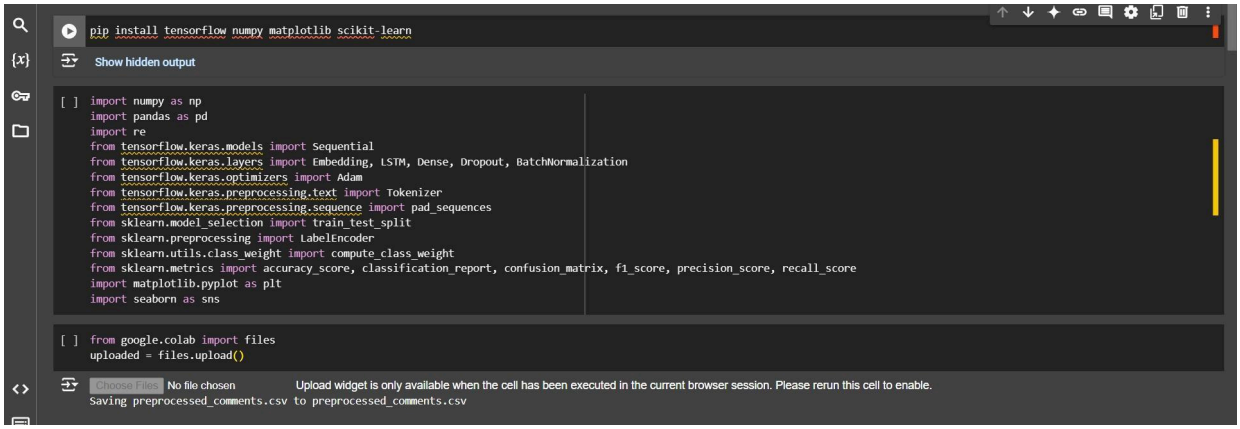
```
lstm_model_pre.save('lstm_model.h5')
files.download('lstm_model.h5')
```

- Saves the trained model to a `.h5` file and downloads it.

ACCURACY FOR LSTM:

Test Size	PRE	POST
0.2	0.83	0.83
0.25	0.81	0.82
0.3	0.81	0.76
0.33	0.80	0.71

RNN (Recurrent Neural Network)



---

1. **Setup and Libraries**

   CODE:pip install tensorflow numpy matplotlib scikit-learn

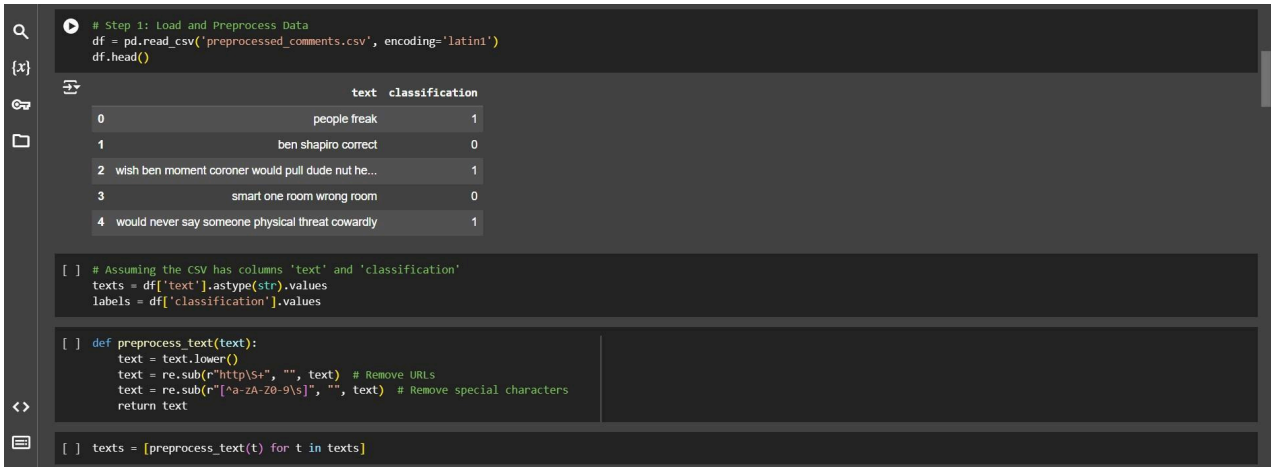
- Installs the required libraries: TensorFlow (for deep learning), NumPy (for numerical operations), Matplotlib (for visualization), and scikit-learn (for machine learning utilities).

```
import numpy as np
import pandas as
pd import re
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.utils.class_weight import compute_class_weight
from sklearn.metrics import accuracy_score,
classification_report, confusion_matrix, f1_score, precision_score, recall_score
import matplotlib.pyplot as plt
import seaborn as sns
```

- Imports libraries and modules required for preprocessing, deep learning, model evaluation, and visualization.

  CODE: from google.colab import files
  uploaded = files.upload()

- Allows file upload in Google Colab. Users upload a CSV file for processing.



```
# Step 1: Load and Preprocess Data
df = pd.read_csv('preprocessed_comments.csv', encoding='latin1')
df.head()
```

	text	classification
0	people freak	1
1	ben shapiro correct	0
2	wish ben moment coroner would pull dude nut he...	1
3	smart one room wrong room	0
4	would never say someone physical threat cowardly	1

```
# Assuming the CSV has columns 'text' and 'classification'
texts = df['text'].astype(str).values
labels = df['classification'].values
```

```
def preprocess_text(text):
    text = text.lower()
    text = re.sub(r"http\S+", "", text)  # Remove URLs
    text = re.sub(r"[^a-zA-Z0-9\s]", "", text)  # Remove special characters
    return text
```

```
texts = [preprocess_text(t) for t in texts]
```

2. **Step 1: Load and Preprocess Data**

   CODE: df = pd.read_csv('preprocessed_comments.csv', encoding='latin1')
   df.head()

- Loads the dataset into a Pandas DataFrame and displays the first few rows. Assumes the file has a column `text` (for input) and `classification` (for labels).

  CODE:
  texts =
  df['text'].astype(str).values labels
  = df['classification'].values

- Extracts the text and classification columns as NumPy arrays.

  CODE:

```
def preprocess_text(text):
    text = text.lower() # Converts to lowercase.
    text = re.sub(r"http\S+", "", text) # Removes URLs.
    text = re.sub(r"[^a-zA-Z0-9\s]", "", text) # Removes special characters.
    return text
texts = [preprocess_text(t) for t in texts]
```

- Defines a function to preprocess text data by lowercasing, removing URLs, and cleaning special characters. Applies the function to all texts.

```
[ ] # Encode labels
    label_encoder = LabelEncoder()
    labels = label_encoder.fit_transform(labels)

    # Tokenize and pad the text data
    max_words = 10000  # Maximum vocabulary size
    max_len = 100      # Maximum sequence length

    tokenizer = Tokenizer(num_words=max_words, oov_token="<OOV>")
    tokenizer.fit_on_texts(texts)
    sequences = tokenizer.texts_to_sequences(texts)
    post_padded_sequences = pad_sequences(sequences, maxlen=max_len, padding='pre', truncating='post')

[ ] # Split data into training and testing sets
    X_train_pre, X_test_pre, y_train_pre, y_test_pre = train_test_split(
        post_padded_sequences, labels, test_size=0.33, random_state=42
    )

[ ] # Compute class weights
    class_weights = compute_class_weight('balanced', classes=np.unique(y_train_pre), y=y_train_pre)
    class_weights = dict(enumerate(class_weights))
```

CODE:
label_encoder = LabelEncoder()
labels = label_encoder.fit_transform(labels)

- Encodes the class labels into integers for model compatibility (e.g., `positive`, `negative` → `1`, `0`).

CODE:
max_words = 10000 # Maximum vocabulary size
max_len = 100 # Maximum sequence length

- Specifies tokenizer vocabulary size and maximum sequence length.

CODE:
tokenizer = Tokenizer(num_words=max_words, oov_token="<OOV>") tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
post_padded_sequences = pad_sequences(sequences, maxlen=max_len, padding='pre', truncating='post')

- Initializes a tokenizer, converts texts to sequences of integers (word IDs), and pads/truncates sequences to a fixed length.

CODE:

```
X_train_pre, X_test_pre, y_train_pre, y_test_pre = train_test_split(
    post_padded_sequences, labels, test_size=0.33, random_state=42
)
```

- Splits data into training and testing sets (33% for testing).

CODE:

```
class_weights = compute_class_weight('balanced', classes=np.unique(y_train_pre), y=y_train_pre)
class_weights = dict(enumerate(class_weights))
```

- Computes class weights to handle class imbalance and converts them into a dictionary for use in training.

```
[ ] # Step 2: Build the Improved RNN Model
    rnn_pre = Sequential([
        Embedding(input_dim=max_words, output_dim=64),
        LSTM(64, activation='tanh', return_sequences=True),
        BatchNormalization(),
        Dropout(0.5),
        LSTM(32, activation='tanh', return_sequences=False),
        Dropout(0.5),
        Dense(16, activation='relu'),
        Dense(1, activation='sigmoid')
    ])

    # Compile the model
    rnn_pre.compile(optimizer=Adam(learning_rate=0.0005), loss='binary_crossentropy', metrics=['accuracy'])
```

3. **Step 2: Build the RNN Model**

CODE:

```
rnn_pre = Sequential([
    Embedding(input_dim=max_words, output_dim=64),
    LSTM(64, activation='tanh', return_sequences=True),
    BatchNormalization(),
    Dropout(0.5),
    LSTM(32, activation='tanh', return_sequences=False),
    Dropout(0.5),
    Dense(16, activation='relu'),
    Dense(1, activation='sigmoid')
])
```

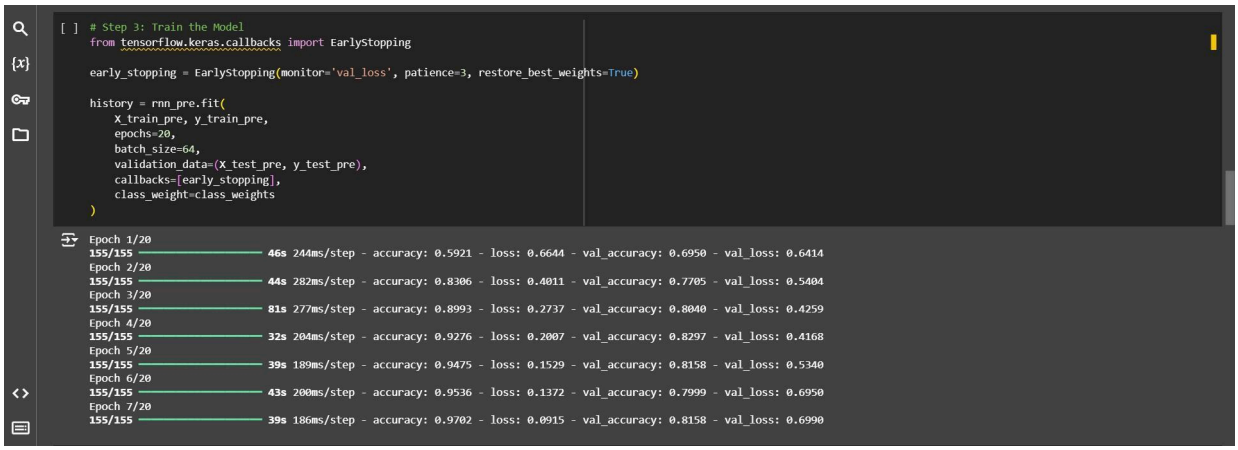
- Defines an RNN model:
    - **Embedding**: Converts word IDs into dense vectors of fixed size.
    - **LSTM**: Long Short-Term Memory layers to capture sequential dependencies.
    - **BatchNormalization**: Normalizes activations for faster convergence.

- **Dropout**: Reduces overfitting by randomly dropping connections.
- **Dense**: Fully connected layers, final layer outputs a single value with a sigmoid activation for binary classification.

CODE:

```
rnn_pre.compile(optimizer=Adam(learning_rate=0.0005),
loss='binary_crossentropy', metrics=['accuracy'])
```

- Compiles the model with:
  - **Adam** optimizer for gradient updates.
  - **Binary Cross-Entropy** loss for binary classification.
  - **Accuracy** as a metric for monitoring performance.



```
[ ] # Step 3: Train the Model
    from tensorflow.keras.callbacks import EarlyStopping

    early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

    history = rnn_pre.fit(
        X_train_pre, y_train_pre,
        epochs=20,
        batch_size=64,
        validation_data=(X_test_pre, y_test_pre),
        callbacks=[early_stopping],
        class_weight=class_weights
    )

Epoch 1/20
155/155 ━━━━━━━━━━━━━━━━━━━━ 46s 244ms/step - accuracy: 0.5921 - loss: 0.6644 - val_accuracy: 0.6950 - val_loss: 0.6414
Epoch 2/20
155/155 ━━━━━━━━━━━━━━━━━━━━ 44s 282ms/step - accuracy: 0.8306 - loss: 0.4011 - val_accuracy: 0.7705 - val_loss: 0.5404
Epoch 3/20
155/155 ━━━━━━━━━━━━━━━━━━━━ 81s 277ms/step - accuracy: 0.8993 - loss: 0.2737 - val_accuracy: 0.8040 - val_loss: 0.4259
Epoch 4/20
155/155 ━━━━━━━━━━━━━━━━━━━━ 32s 204ms/step - accuracy: 0.9276 - loss: 0.2007 - val_accuracy: 0.8297 - val_loss: 0.4168
Epoch 5/20
155/155 ━━━━━━━━━━━━━━━━━━━━ 39s 189ms/step - accuracy: 0.9475 - loss: 0.1529 - val_accuracy: 0.8158 - val_loss: 0.5340
Epoch 6/20
155/155 ━━━━━━━━━━━━━━━━━━━━ 43s 200ms/step - accuracy: 0.9536 - loss: 0.1372 - val_accuracy: 0.7999 - val_loss: 0.6950
Epoch 7/20
155/155 ━━━━━━━━━━━━━━━━━━━━ 39s 186ms/step - accuracy: 0.9702 - loss: 0.0915 - val_accuracy: 0.8158 - val_loss: 0.6990
```

4. **Step 3: Train the Model**

   CODE:

   ```
   from tensorflow.keras.callbacks import EarlyStopping
   early_stopping = EarlyStopping(monitor='val_loss', patience=3,
   restore_best_weights=True)
   ```

- Sets up early stopping to terminate training if validation loss does not improve for 3 epochs, restoring the best weights.

CODE:

```
history = rnn_pre.fit(
    X_train_pre, y_train_pre,
    epochs=20,
    batch_size=64,
    validation_data=(X_test_pre, y_test_pre),
    callbacks=[early_stopping],
    class_weight=class_weights
)
```

- Trains the model for 20 epochs with:
    - **Batch size** of 64.
    - Validation on the test set.
    - Early stopping and class weights applied.

5. **Step 4: Evaluate the Model**

    CODE:

    ```
    loss, accuracy = rnn_pre.evaluate(X_test_pre, y_test_pre)
    print(f"Test Loss: {loss:.4f}, Test Accuracy: {accuracy:.4f}")
    ```

- Evaluates the trained model on the test set and prints loss and accuracy.

CODE:

```
y_pred_pre = rnn_pre.predict(X_test_pre)
y_pred_pre_class = (y_pred_pre > 0.5).astype('int32')
```

- Makes predictions and converts probabilities to binary class labels using a threshold of 0.5.

---

6. **Step 5: Evaluate Metrics**

CODE:

```
f1 = f1_score(y_test_pre, y_pred_pre_class)
precision = precision_score(y_test_pre, y_pred_pre_class)
recall = recall_score(y_test_pre, y_pred_pre_class)
print(f"F1 Score: {f1:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
```

- Calculates and prints F1 Score, Precision, and Recall for performance evaluation.

CODE:

```
conf_matrix = confusion_matrix(y_test_pre, y_pred_pre_class)
class_names = ['Class 0', 'Class 1']
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

- Computes and visualizes the confusion matrix using Seaborn.

```
# Confusion Matrix
conf_matrix = confusion_matrix(y_test_pre, y_pred_pre_class)
class_names = ['Class 0', 'Class 1']

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()

# Step 6: Visualize Training History
plt.figure(figsize=(12, 5))

# Accuracy Plot
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.legend()

# Loss Plot
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.legend()

plt.show()
```

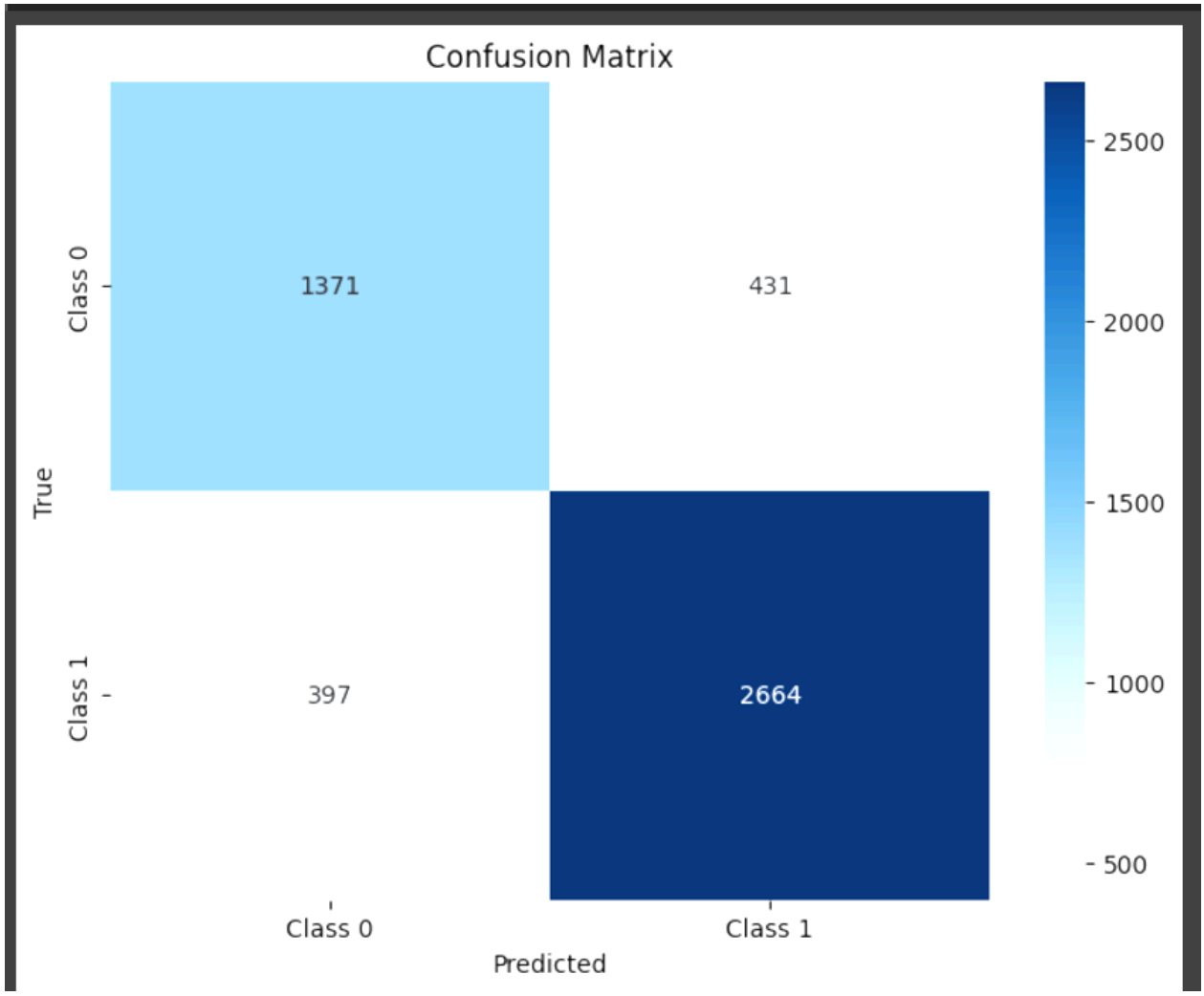
---

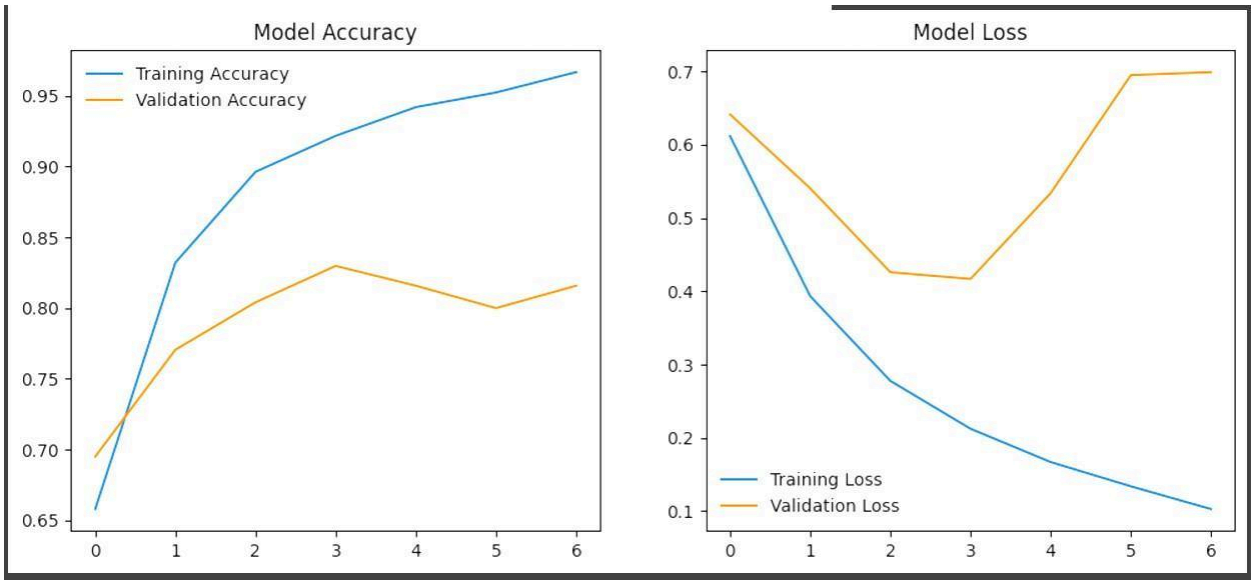
7. **Step 6: Visualize Training History**

   CODE:

```
plt.figure(figsize=(12, 5))

# Accuracy Plot
plt.subplot(1, 2,
1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.legend()

# Loss Plot
plt.subplot(1, 2,
2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.legend()

plt.show()
```

- Visualizes training and validation accuracy and loss over epochs.


Confusion Matrix
Class 0
Class 1
True
1371
431
397
2664
Class 0
Class 1
Predicted
2500
2000
1500
1000
500


Model Accuracy
Training Accuracy
Validation Accuracy
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0
1
2
3
4
5
6
Model Loss
0.7
0.6
0.5
0.4
0.3
0.2
0.1
Training Loss
Validation Loss
0
1
2
3
4
5
6

```
[ ]  rnn_pre.save('rnn_model.h5')
     files.download('rnn_model.h5')
     import pickle

     # Save the tokenizer to a file
     with open('tokenizer.pkl', 'wb') as f:
         pickle.dump(tokenizer, f)
     files.download('tokenizer.pkl')
```

8. **Save Model and Tokenizer**

   CODE:

   ```
   rnn_pre.save('rnn_model.h5')
   files.download('rnn_model.h5')
   ```

- Saves the trained model as `rnn_model.h5` and allows downloading.

  CODE:

  ```
  import pickle
  with open('tokenizer.pkl', 'wb') as f:
      pickle.dump(tokenizer, f)
  files.download('tokenizer.pkl')
  ```

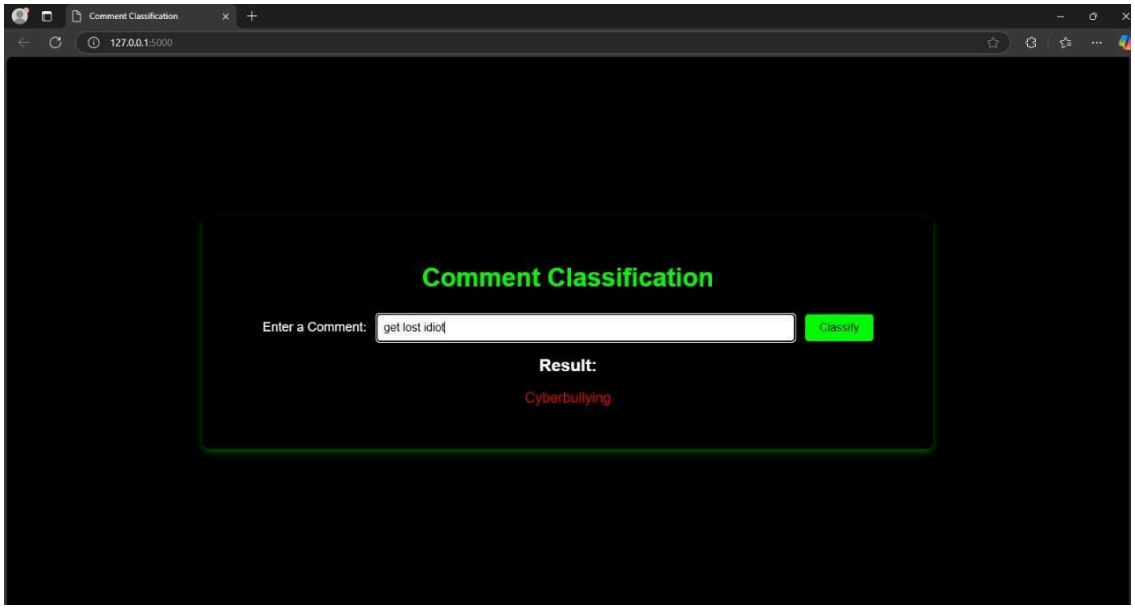
- Serializes and saves the tokenizer as `tokenizer.pkl`, then allows downloading.

ACCURACY FOR RNN:

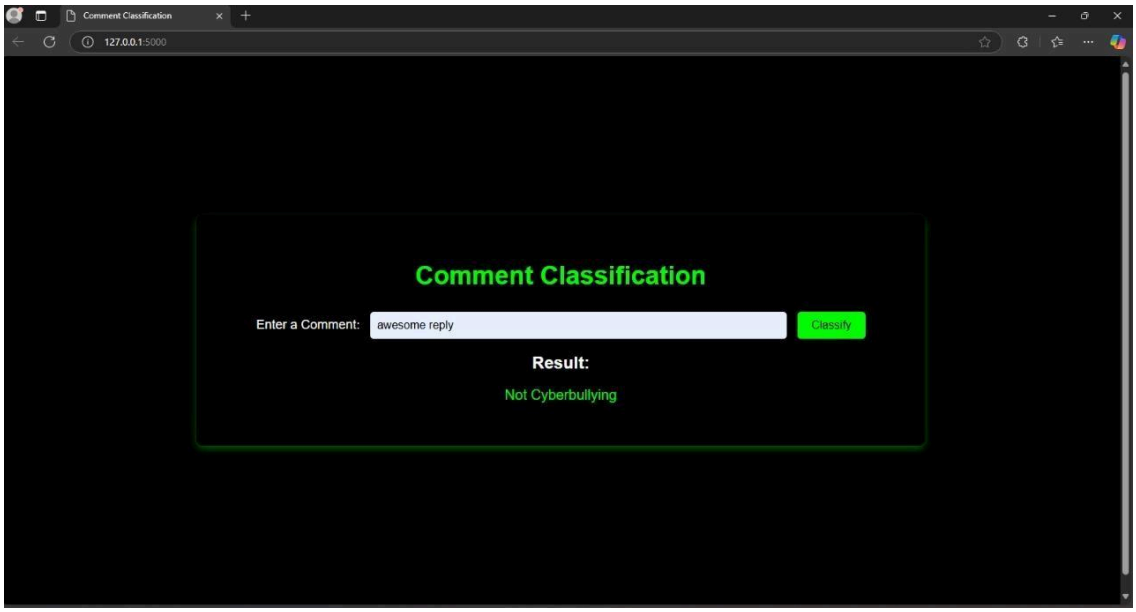
Test Size	PRE	POST
0.2	0.83	0.641
0.25	0.828	0.656
0.3	0.831	0.647
0.33	0.821	0.651

Front End GUI

With Negative Comment:“ GET LOST IDIOT ”

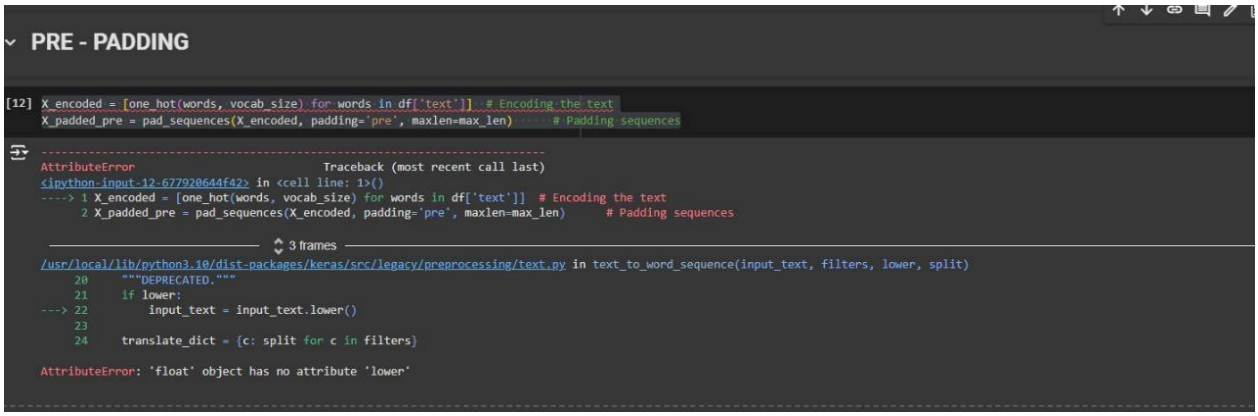


With Positive Comment:“AWESOME REPLY”



ERRORS ENCOUNTERED

ERROR 1:



REASON:

The error you encountered, `float object has no attribute`, often arises when you mistakenly treat a float (numeric value) as if it were an object with attributes (e.g., calling a method like `.shape` or `.fit` on a float)

****

ERROR 2:

ValueError Traceback (most recent call last) <ipython-input-14-31a443235a59> in <cell line: 2>() 1 # Train the model ----> 2 history = lstm.fit(X_train, y_train_categorical, epochs=10, batch_size=64, validation_data=(X_test, y_test_categorical)) 1 frames /usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/__init__.py in get_data_adapter(x, y, sample_weight, batch_size, steps_per_epoch, shuffle, class_weight) 118 #) 119 else: --> 120 raise ValueError(f"Unrecognized data type: x={x} (of type {type(x)})") 121 122

REASON:

The error occurs because the data provided to the `lstm.fit()` method is not in a format that Keras recognizes for training. The issue lies in the format of `X_train`, which appears to be a list of lists or an improperly prepared NumPy array. For training an LSTM model, Keras expects the input data (`X_train`) to be a 3D array of shape `(samples, timesteps, features)`, where:

- `samples`: Number of training examples.
- `timesteps`: Number of time steps in each sequence.
- `features`: Number of features per time step.