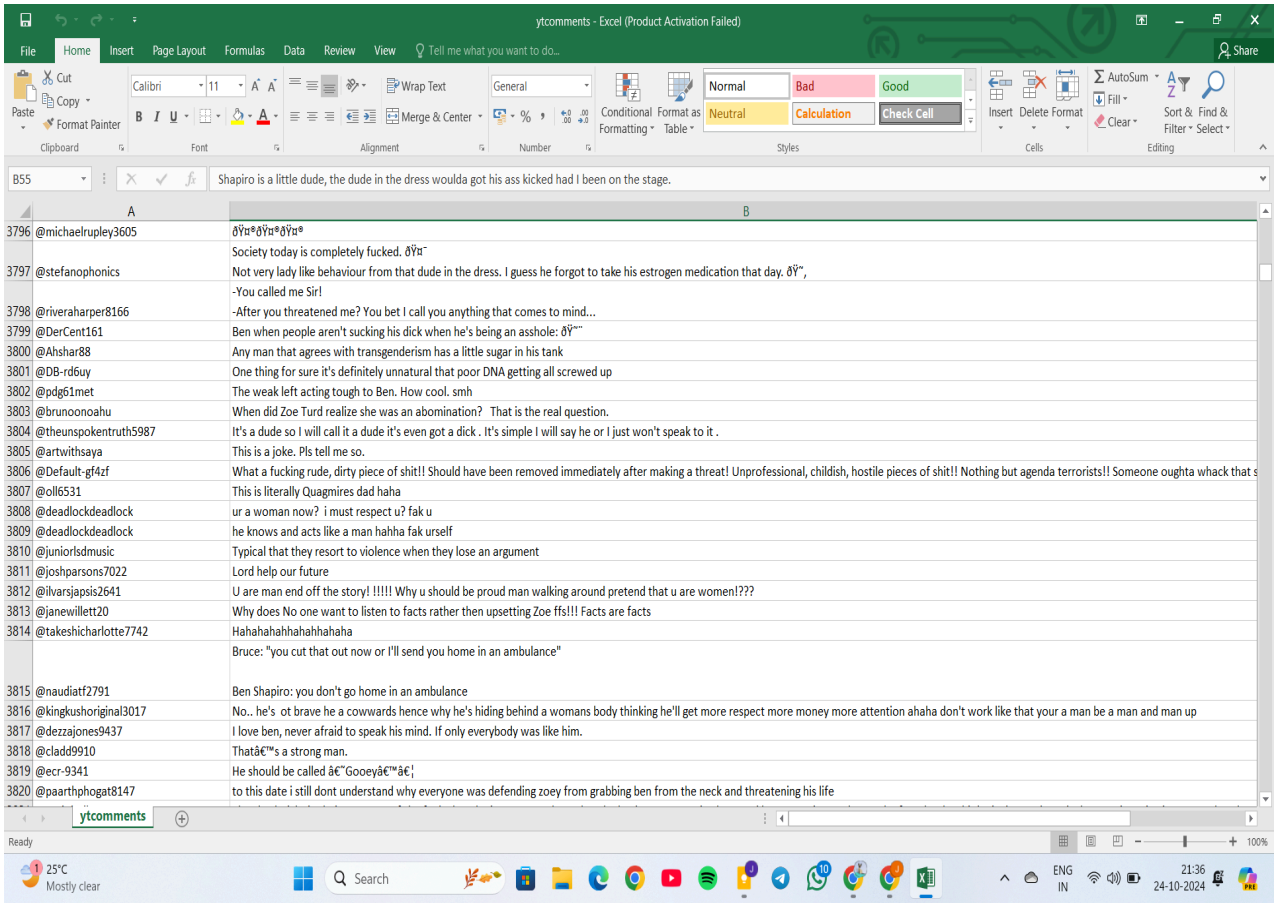
Cyberbullying Detection on Social Networks Using Hybrid RNN - LSTM Model

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

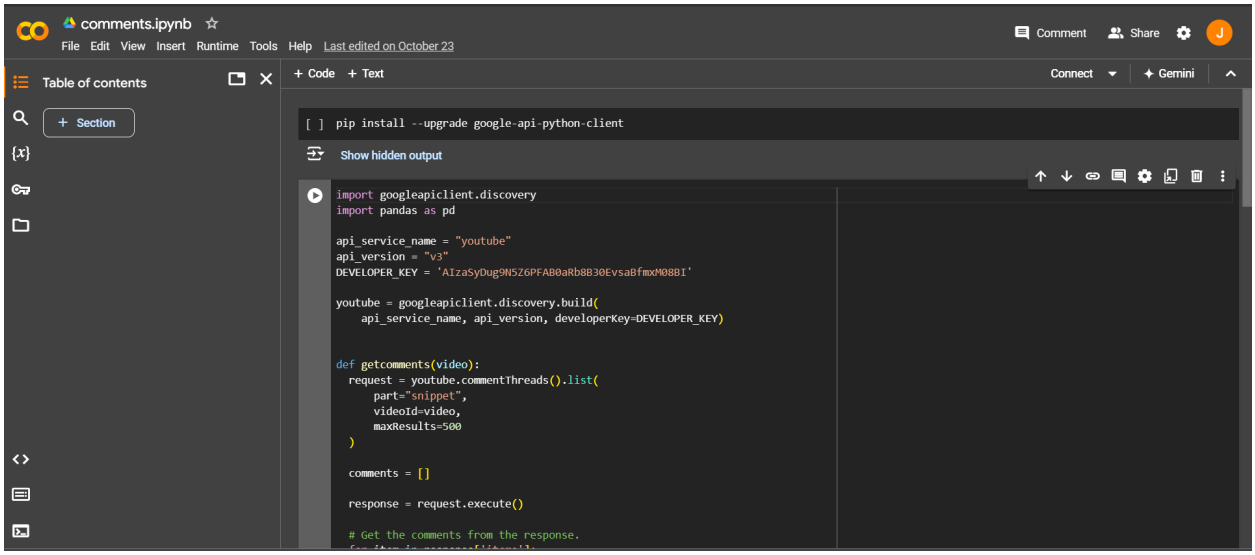


Fig: 1(b)

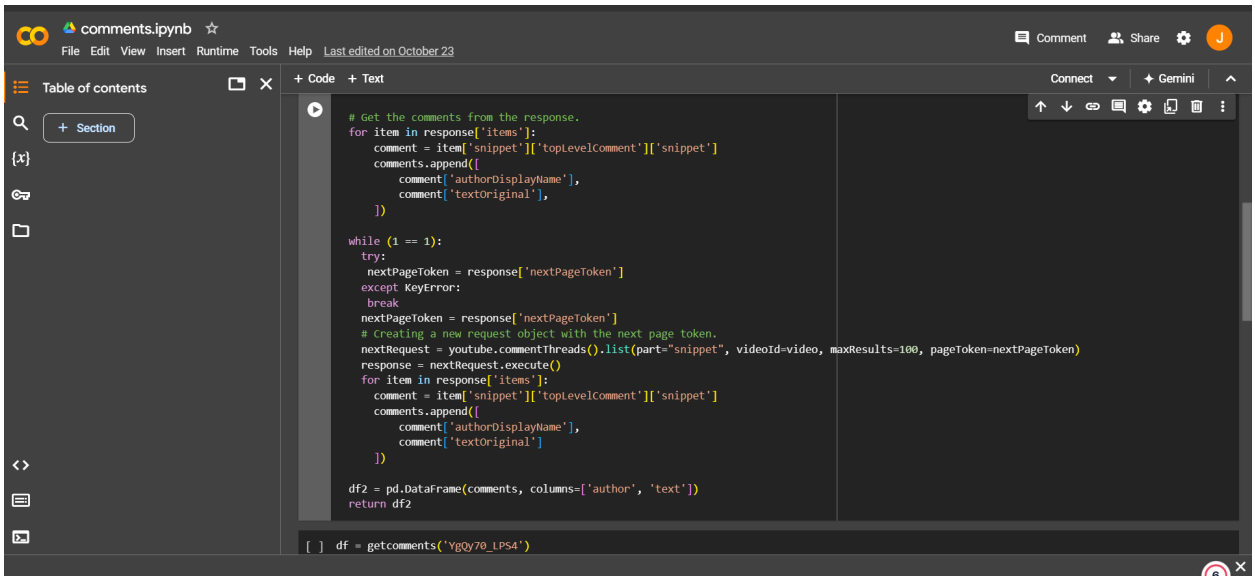


Fig: 1(c)

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

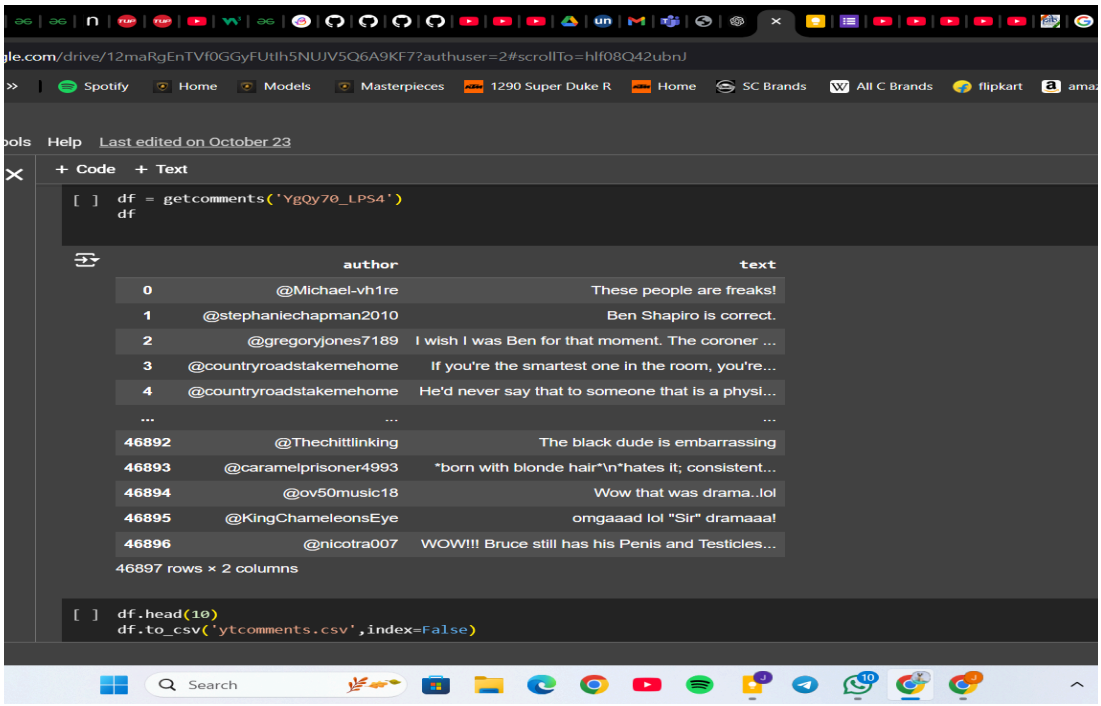


Fig:1(d) Shows the output of the previous code and the result we are required i.e.our raw dataset.

Errors encountered while web scraping:

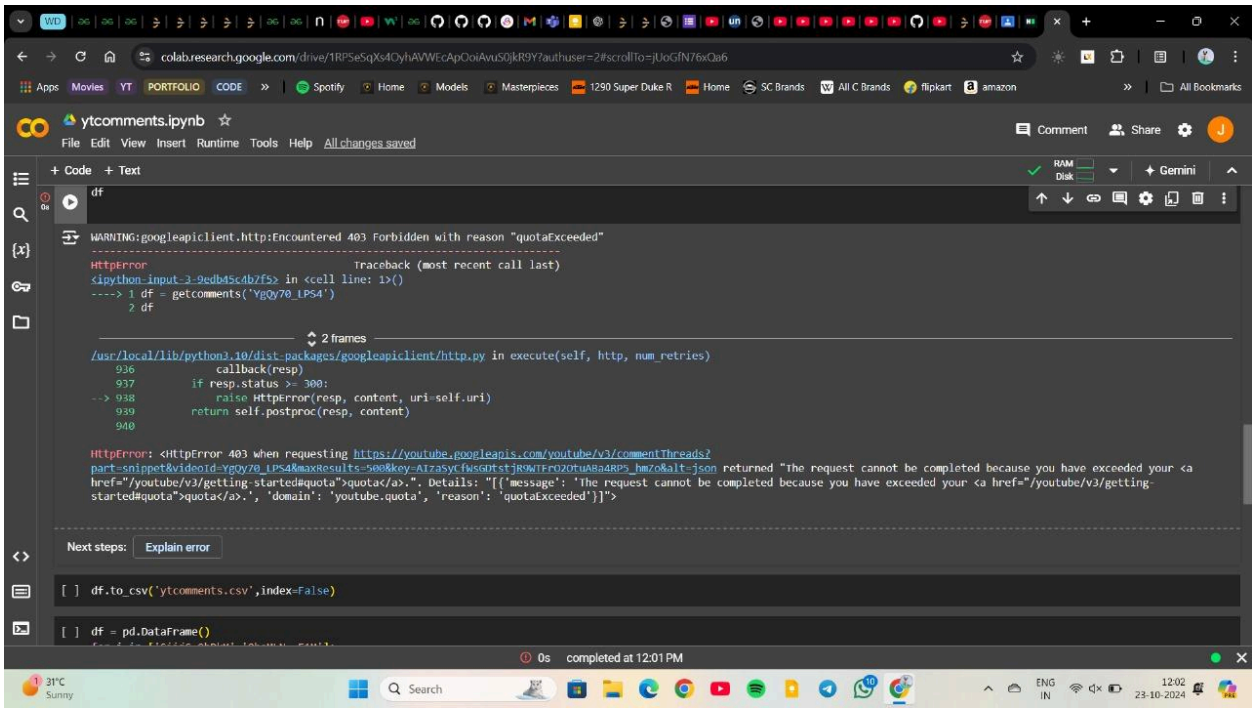


Fig: 1(e) Shows the errors we encountered i.e. ‘quotaExceeded’

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:

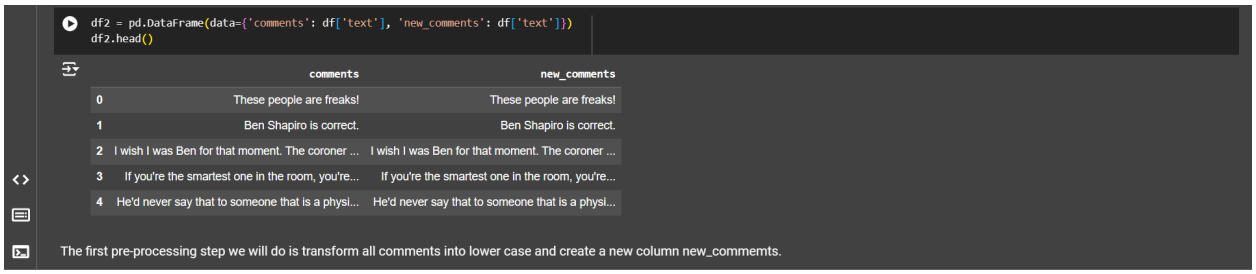


Fig: 1(g) Shows the code and output for creating a new dataframe.

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:

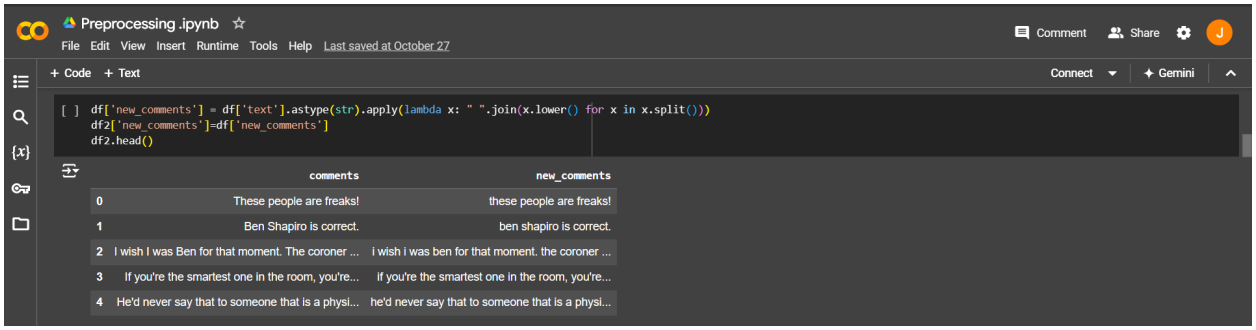


Fig: 1(h) Shows the code and output for lowering the case of the strings in the dataset.

- 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:

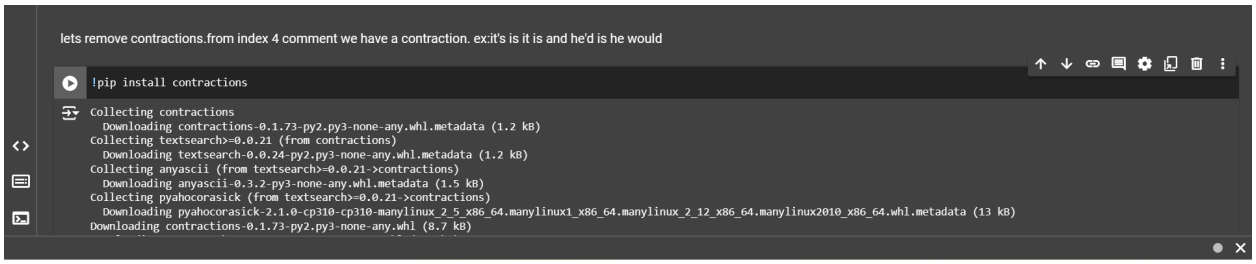
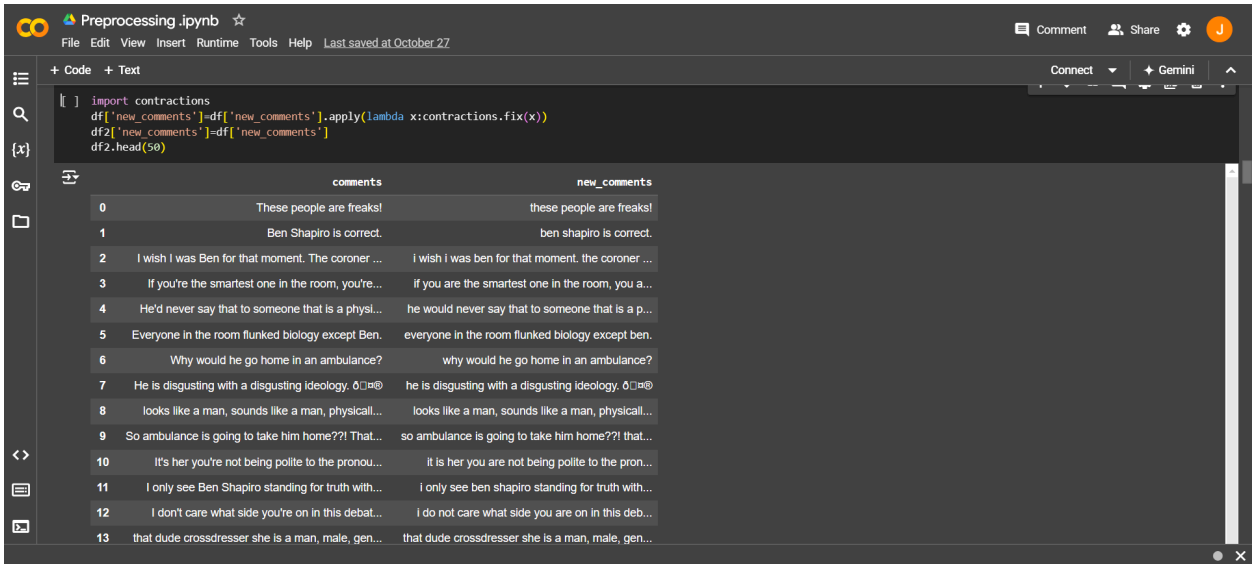


Fig: 1(i) Shows the installation contraction library.



	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. ô□¤®	he is disgusting with a disgusting ideology. ô□¤®
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))]

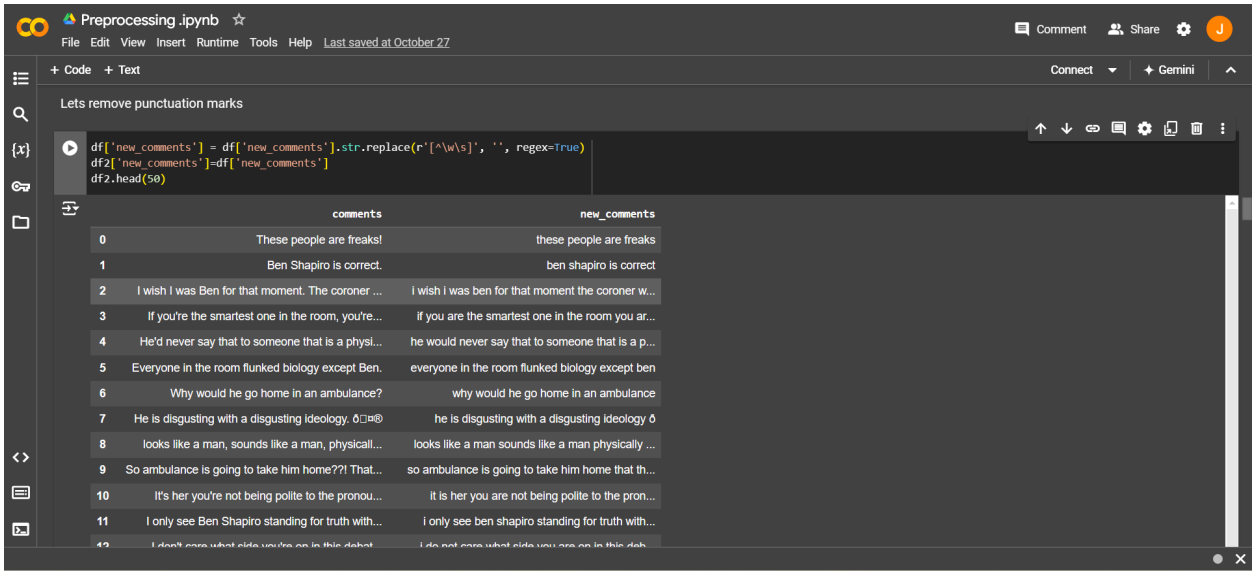
- This line expands contractions in the new_comments column of the DataFrame df.
- Breaking it down further:
  - **df['new_comments']**: Accesses the new_comments column in df, which is where the text data is stored.
  - **.apply(lambda x: contractions.fix(x))**: Uses the apply() function to apply a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

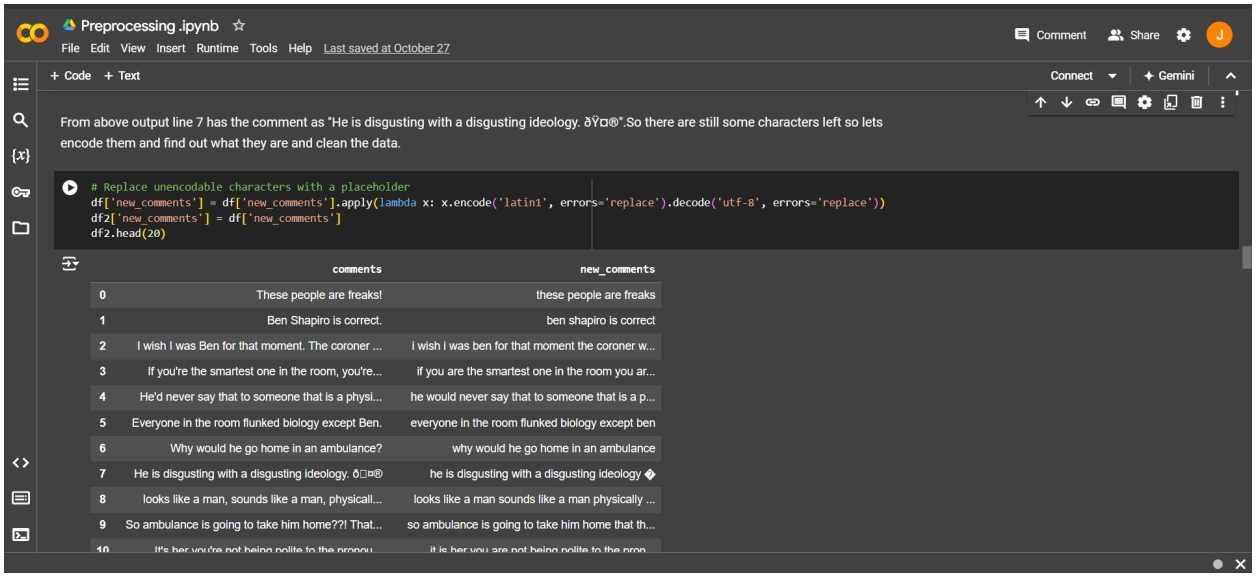


Fig: 1(l) Shows the code and output for characters other than punctuation marks.

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), 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..

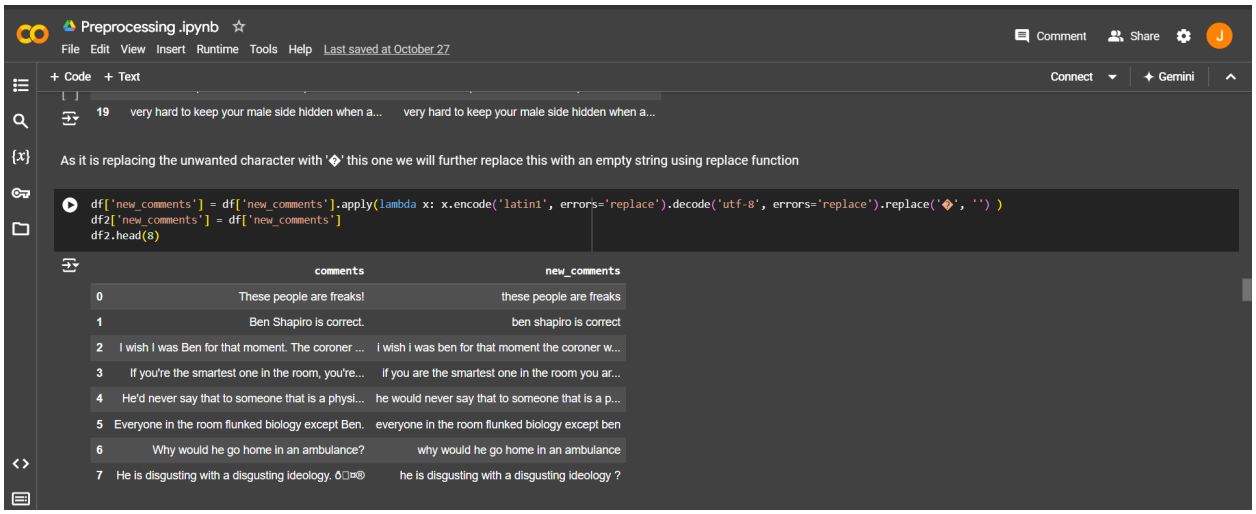


Fig: 1(m) Show the code and output for replacing unwanted characters with empty string.

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('', ''):

- After decoding, this additional step uses the replace() method to remove any placeholder characters (specifically , which is often used to represent an unknown or unrecognized character).
- The presence of 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.

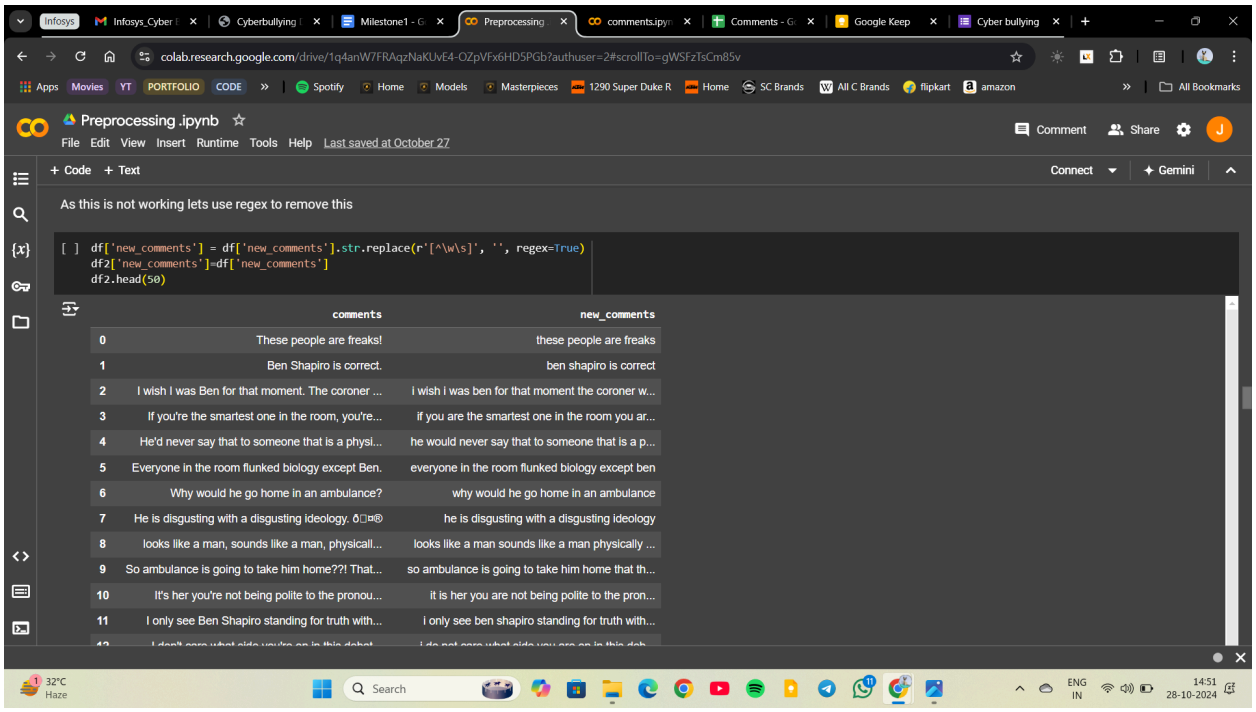


Fig: 1(n) Shows the code and output for removing the empty string.

.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.

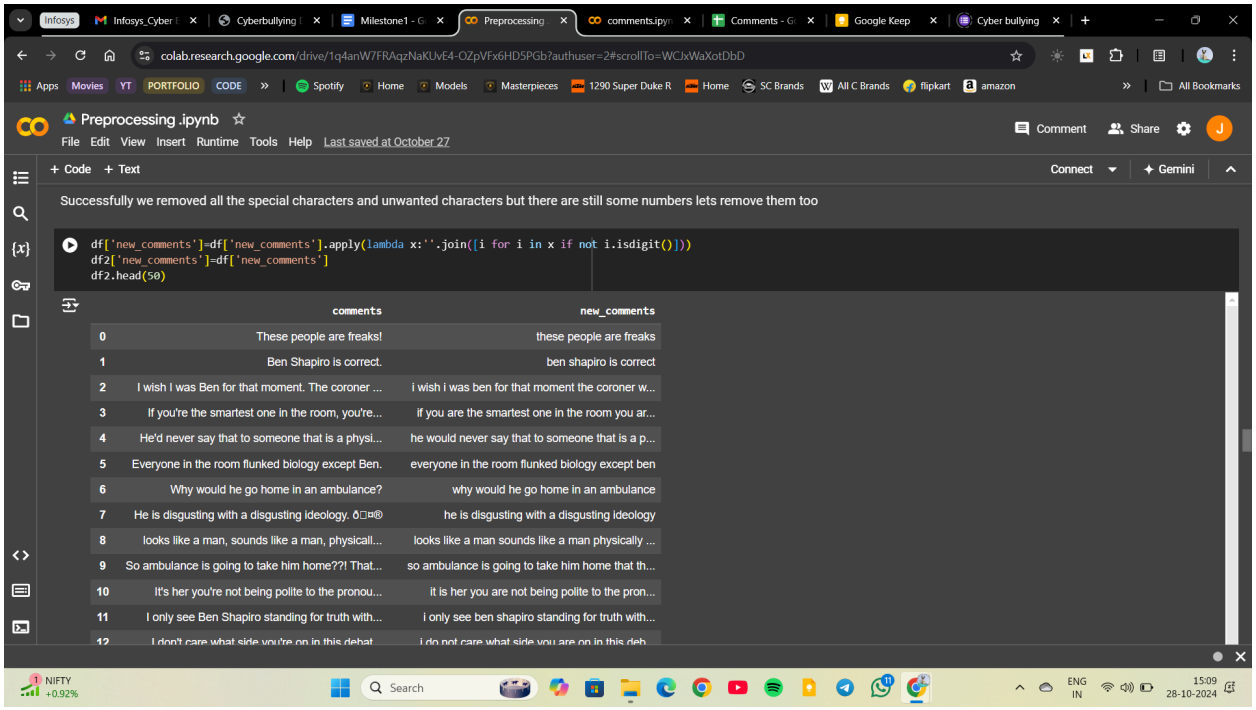


Fig: 1(o) Shows the code and output for removing the digits from the dataset.

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

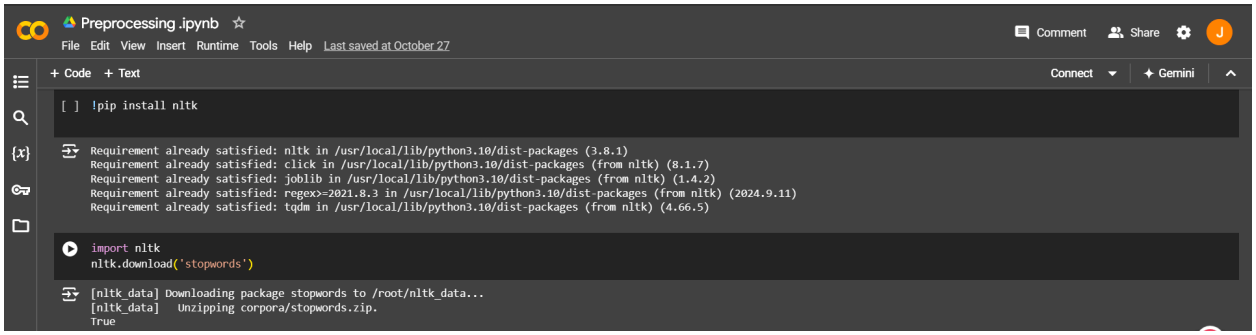
- This part is a list comprehension that iterates over each character i in the string x.
- **if not i.isdigit()**: The condition checks whether the character i is not a digit. The isdigit() method returns True if the character is a digit (0-9) and False otherwise.
- Therefore, this list comprehension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

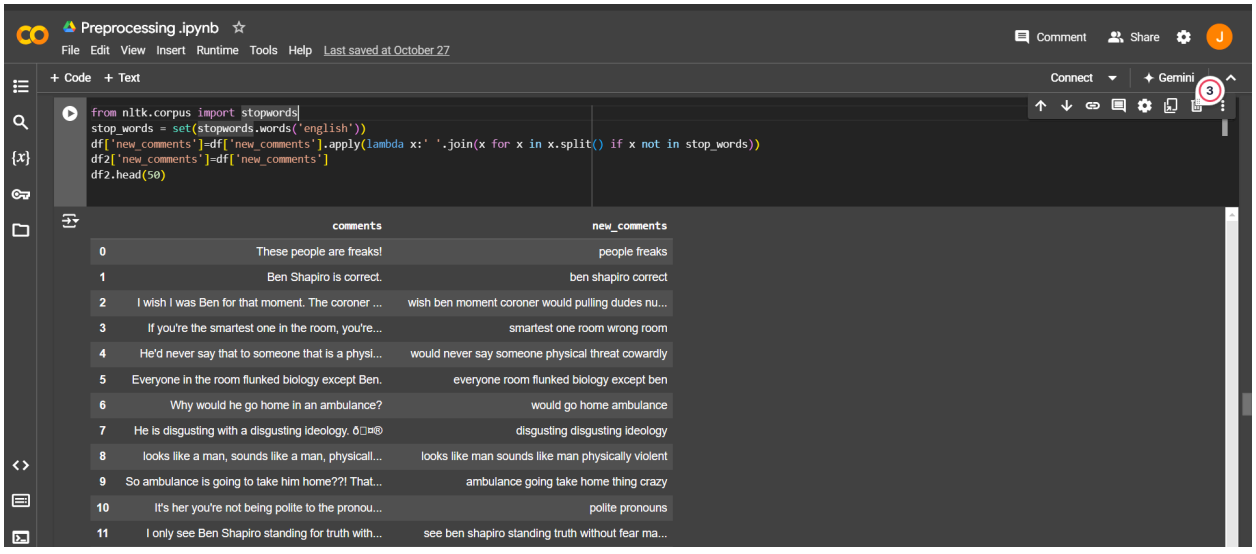
9. **Downloading Stopword**

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:

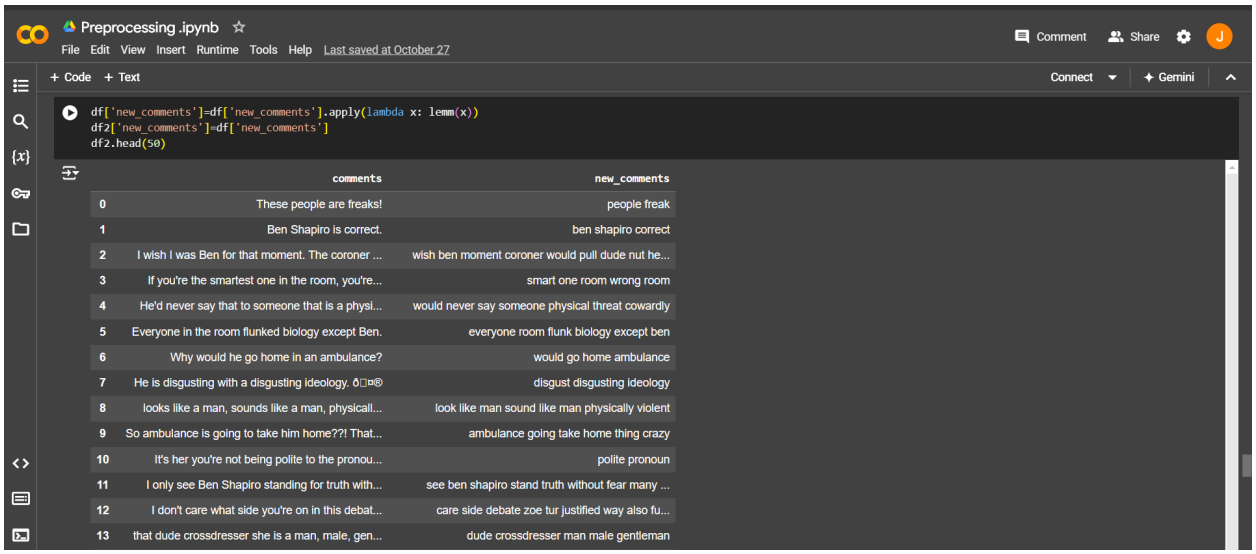
- This line imports functions from the langdetect library, which can detect the language of a given text. While it's imported here, it is not used in the current snippet.

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. ó□¤®	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.

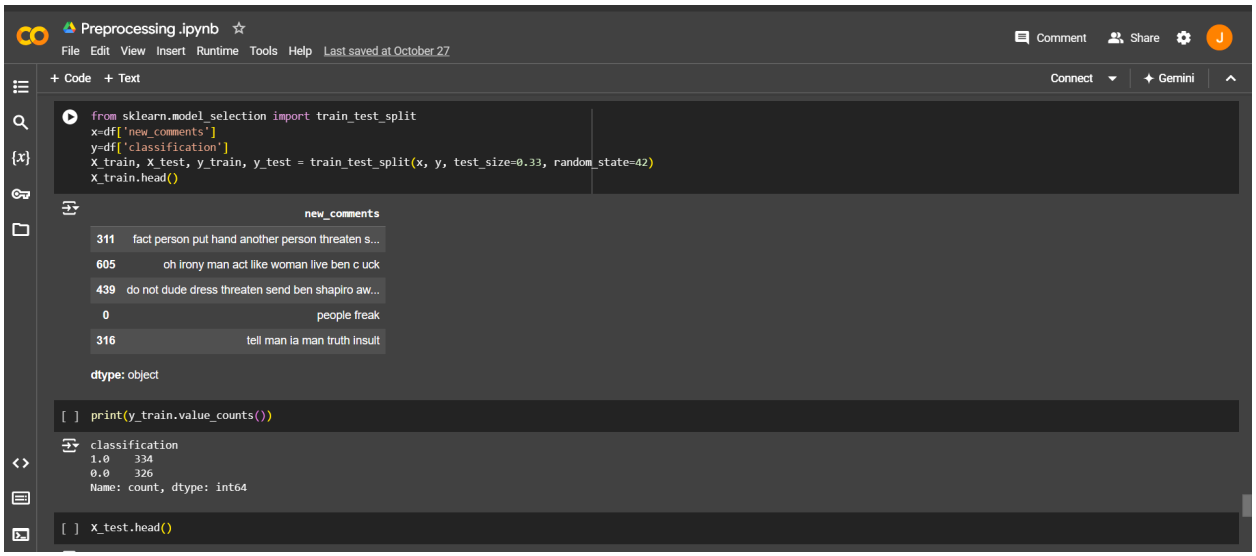
df['new_comments'] = df['new_comments'].apply(lambda x: lemm(x)):

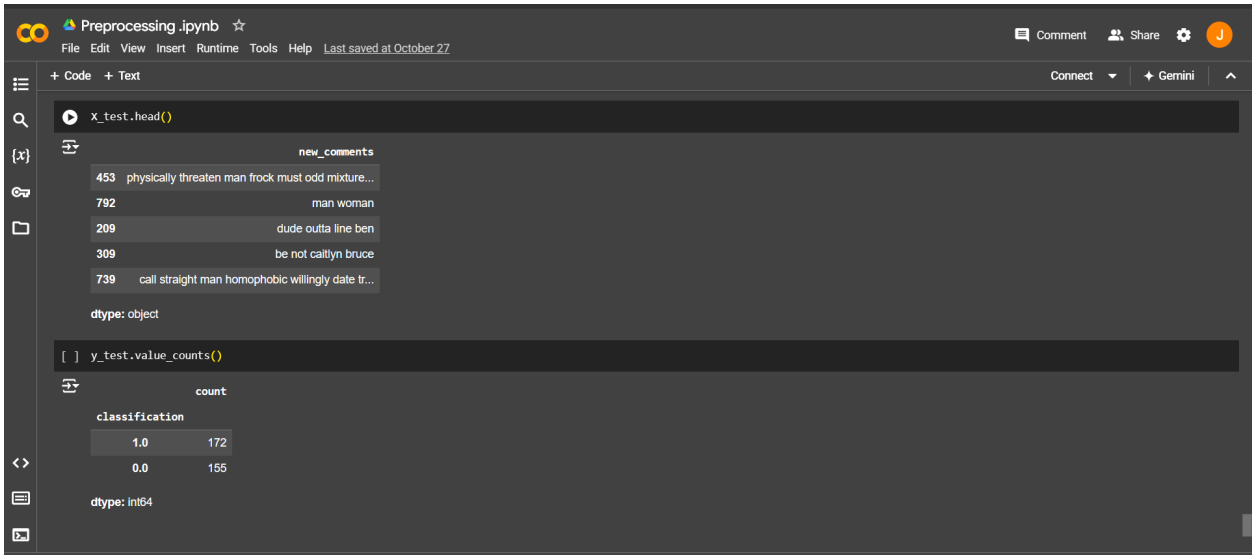
- This line applies the lemm function to each entry in the new_comments column of the DataFrame df.
- Each comment is processed to replace words with their lemmas, and the updated comments are assigned back to the new_comments column.

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.

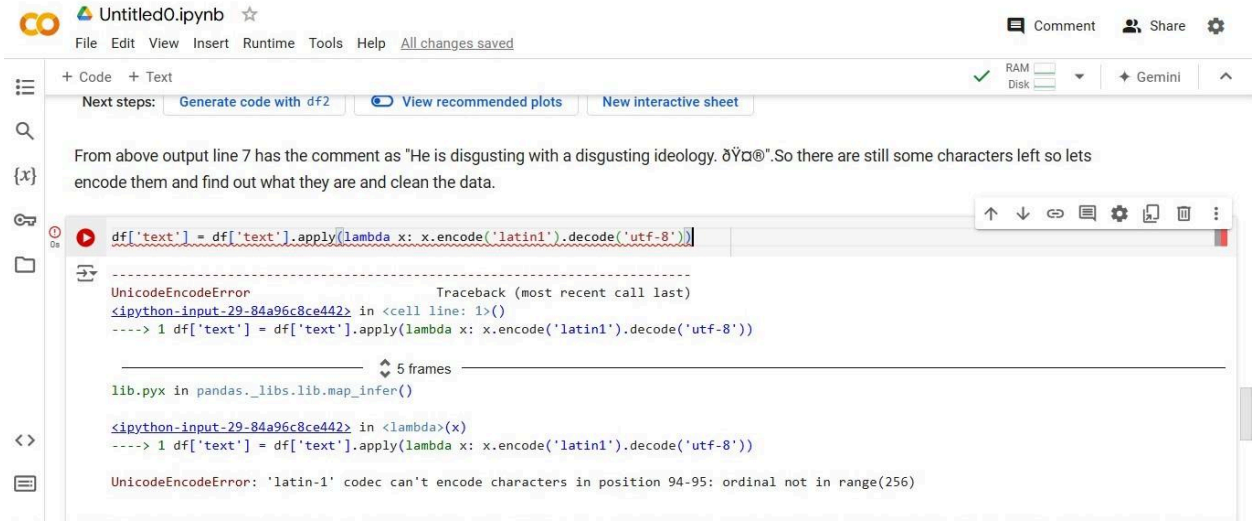
Variables: x holds the cleaned comments, while y contains the classification labels.





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