RoBERTa-Powered Twitter Sentimental Analysis on Threads-App

Table of Content

Abstract

The "Thread app dataset: 37000 entities" is a comprehensive collection of over 37,000 reviews sourced from the Google Play Store and Apple App Store, centered around the New Thread mobile application. This meticulously curated dataset serves as a valuable resource for researchers, data scientists, and machine learning enthusiasts interested in natural language processing, sentiment analysis, and app performance assessment. Key features include a vast coverage of user feedback from Android and iOS platforms, each review accompanied by textual content and sentiment labels (positive, negative, neutral). The dataset also includes metadata like review date, reviewer demographics, and app version, enabling temporal and version-based analyses. Researchers can leverage this dataset for sentiment analysis model development, machine learning training, and in-depth app performance exploration, gaining insights into user engagement, feature preferences, and improvement suggestions. Downloading and exploring this dataset provides a wealth of user sentiment data essential for enhancing app evaluation methodologies and improving user experiences..

Objective

This report aims to employ advanced Natural Language Processing (NLP) techniques, focusing on leveraging the RoBERTa model, for sentiment analysis on the Twitter dataset sourced from Kaggle. The primary objective is to demonstrate the effectiveness of cutting-edge NLP models in discerning sentiment nuances within Threads data.

Through meticulous preprocessing steps, including tokenization, stopwords removal, and data cleaning, the report seeks to enhance the accuracy and granularity of sentiment classification. Additionally, it encompasses Exploratory Data Analysis (EDA) to glean insights into the dataset's characteristics.

By utilizing RoBERTa and other NLP tools, this project intends to refine feature sets and optimize sentiment analysis performance. Furthermore, it aims to elucidate the importance of preprocessing steps and model selection in achieving reliable sentiment analysis results.

The ultimate goal of this report is to contribute a comprehensive understanding of sentiment analysis on social media platforms, with implications for broader applications in social analytics and opinion mining.

Keywords used: RoBERTa, Sentiment analysis, NLP, Threads dataset, Preprocessing, Tokenization, Stopwords removal, Data cleaning, EDA.

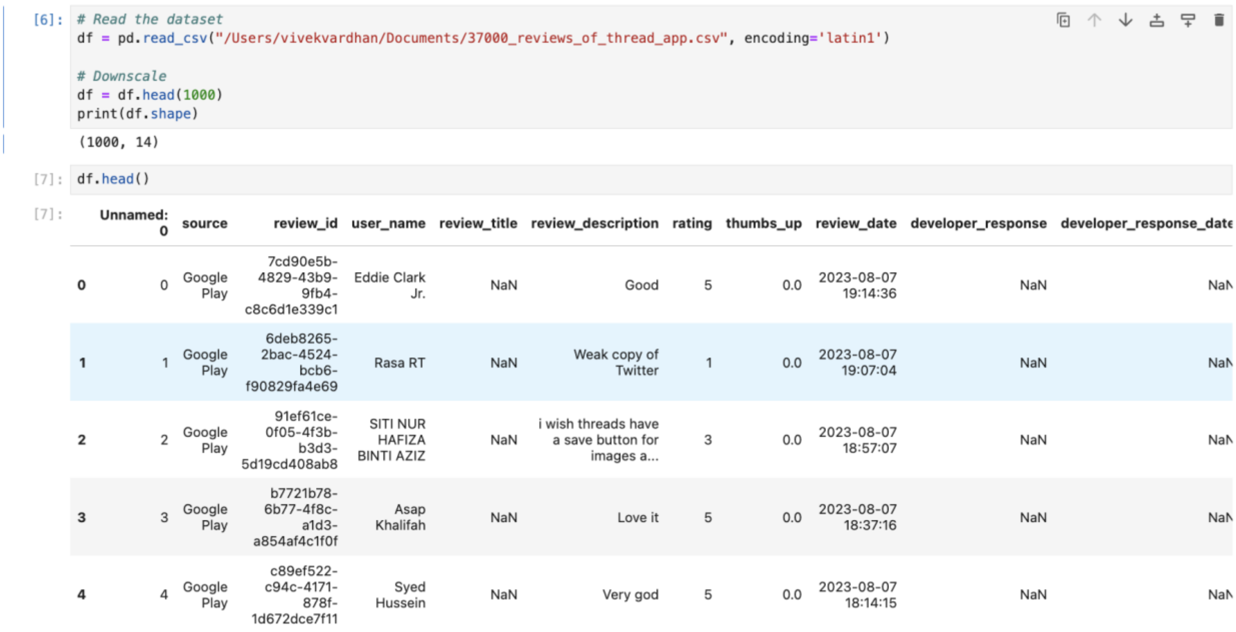
Introducing the comprehensive "Thread app dataset: 37000 entities", featuring a collection of over 37,000 reviews sourced from both the Google Play Store and Apple App Store. This meticulously curated dataset offers a rich and diverse range of user sentiments and opinions regarding the popular New Thread mobile application.

"Thread app dataset: 37000 entities" is an invaluable resource for researchers, data scientists, and machine learning enthusiasts aiming to delve into the world of natural language processing, sentiment analysis, and app performance assessment. The dataset encompasses a wide spectrum of user experiences, providing an insightful glimpse into user satisfaction, usability, feature preferences, and potential areas for improvement.

Key Features:

1. **Vast Review Coverage**: With over 37,000 reviews, this dataset captures a substantial and representative sample of user feedback from both Android and iOS platforms, offering a comprehensive view of the New Thread app's reception.
2. **Rich Textual Data**: Each review is accompanied by its corresponding textual content, enabling researchers to explore the intricacies of user language, writing styles, and expressions of sentiment.
3. **Rating and Sentiment Labels**: Reviews are tagged with accompanying star ratings and sentiment labels (e.g., positive, negative, neutral) to facilitate sentiment analysis and polarity classification tasks.
4. **Metadata and App Version**: The dataset includes essential metadata, such as review date, reviewer demographics (where available), and the New Thread app version, allowing for temporal and version-based analyses.
5. **Diversity of Insights**: Gain insights into user engagement, feature popularity, bug reports, user expectations, and suggestions for enhancements, all of which contribute to a holistic understanding of the app's strengths and areas for development.
6. **Benchmarking and Analysis**: Researchers can utilize this dataset for benchmarking sentiment analysis models, training machine learning algorithms, and conducting exploratory analyses to extract meaningful patterns and trends.

Whether you're interested in developing sentiment analysis models, improving user experience, or gaining valuable insights into the New Thread app's performance, the New Thread App Reviews Dataset offers a goldmine of user-generated content and opinions to fuel your research and analysis. Download and explore this dataset today to unlock the hidden gems of user sentiment and contribute to the advancement of app evaluation methodologies.

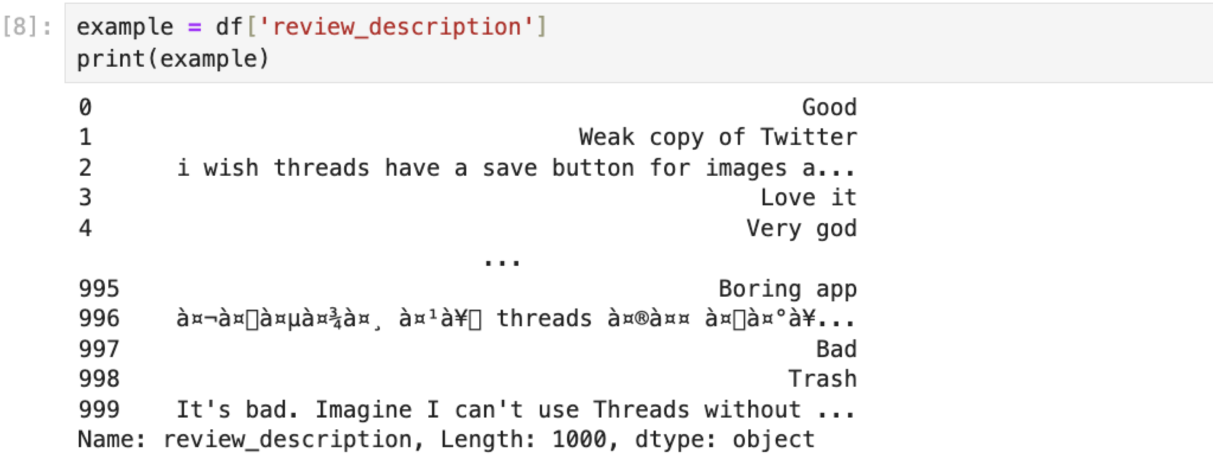


```
[6]: # Read the dataset
     df = pd.read_csv("/Users/vivekvardhan/Documents/37000_reviews_of_thread_app.csv", encoding='latin1')

     # Downscale
     df = df.head(1000)
     print(df.shape)

     (1000, 14)
```

```
[7]: df.head()
```

[7]:

	Unnamed: 0	source	review_id	user_name	review_title	review_description	rating	thumbs_up	review_date	developer_response	developer_response_date
0	0	Google Play	7cd90e5b-4829-43b9-9fb4-c8c6d1e339c1	Eddie Clark Jr.	NaN	Good	5	0.0	2023-08-07 19:14:36	NaN	Na
1	1	Google Play	6deb8265-2bac-4524-bcb6-f90829fa4e69	Rasa RT	NaN	Weak copy of Twitter	1	0.0	2023-08-07 19:07:04	NaN	Na
2	2	Google Play	91ef61ce-0f05-4f3b-b3d3-5d19cd408ab8	SITI NUR HAFIZA BINTI AZIZ	NaN	i wish threads have a save button for images a...	3	0.0	2023-08-07 18:57:07	NaN	Na
3	3	Google Play	b7721b78-6b77-4f8c-a1d3-a854af4c1f0f	Asap Khalifah	NaN	Love it	5	0.0	2023-08-07 18:37:16	NaN	Na
4	4	Google Play	c89ef522-c94c-4171-878f-1d672dce7f11	Syed Hussein	NaN	Very god	5	0.0	2023-08-07 18:14:15	NaN	Na



```
[8]: example = df['review_description']
     print(example)

     0                                                   Good
     1                                   Weak copy of Twitter
     2      i wish threads have a save button for images a...
     3                                                Love it
     4                                               Very god
                                  ...
     995                                           Boring app
     996    à¤¬à¤□à¤µà¤¾à¤¸ à¤¹à¥□ threads à¤®à¤¤ à¤□à¤°à¥...
     997                                                  Bad
     998                                                Trash
     999    It's bad. Imagine I can't use Threads without ...
     Name: review_description, Length: 1000, dtype: object
```

NLTK INTRODUCTION

```
[9]: import nltk
     import numpy as np
     import matplotlib as mp
```

```
[49]: example = df['review_description'][140]
      print (example)
```

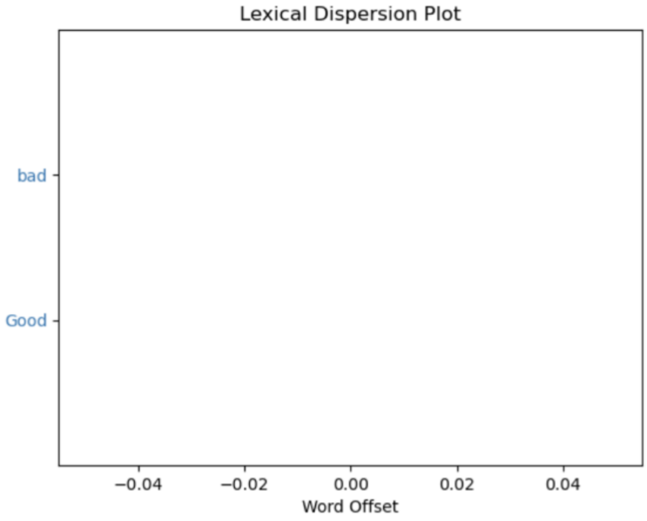
```
Hope you can minimize the threads of accounts that we do not follow.
```

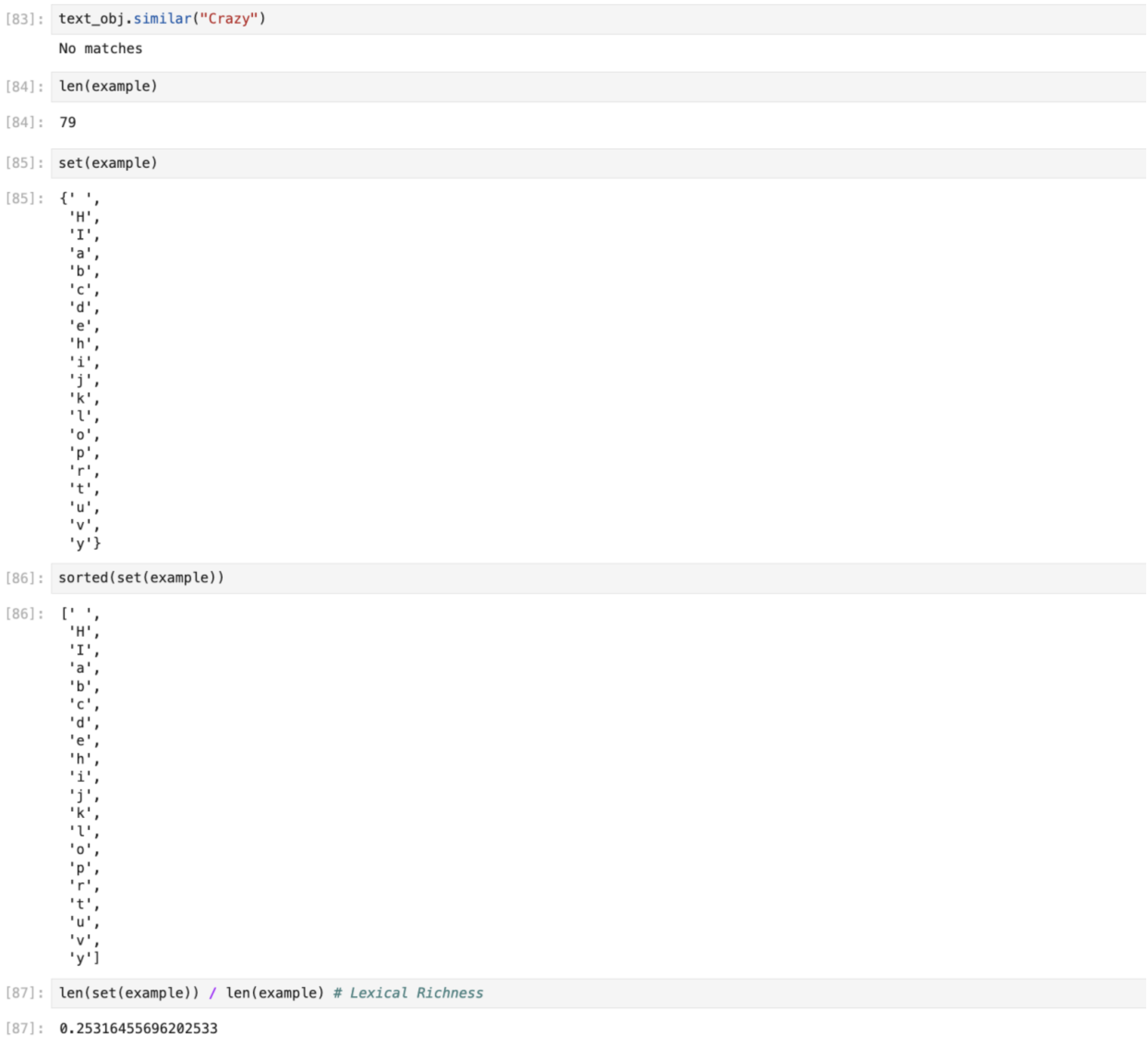
```
[81]: import nltk
      from nltk.text import Text

      # Sample text
      text = example

      # Tokenize the text
      tokens = nltk.word_tokenize(text)

      # Create an NLTK Text object
      text_obj = Text(tokens)

      # Use the concordance method
      text_obj.concordance("Threads ")
```

```
no matches
```

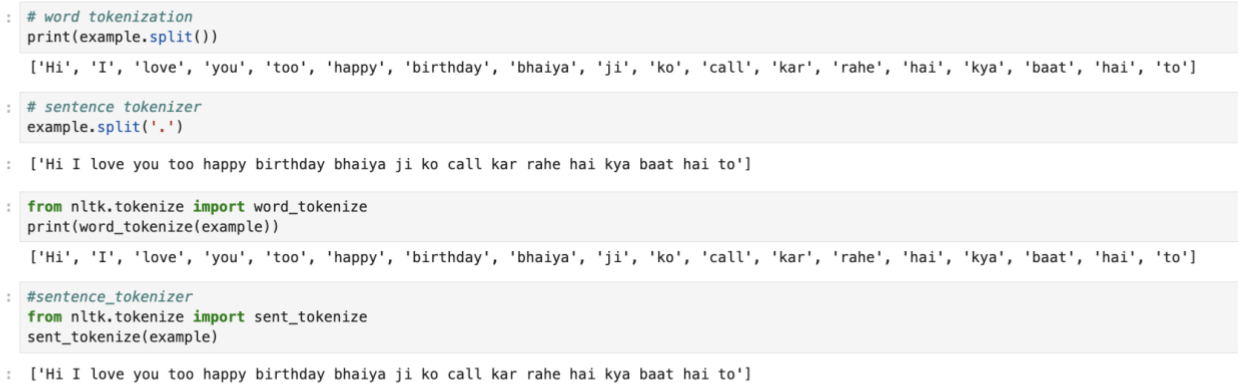
```
[117]: import nltk
       from nltk.tokenize import word_tokenize
       from nltk.text import Text

       df['tokens'] = df['review_description'].apply(word_tokenize)
       nltk_text = Text(df['tokens'].sum())
```

```
[118]: text_obj.dispersion_plot(['Good', 'bad'])
```

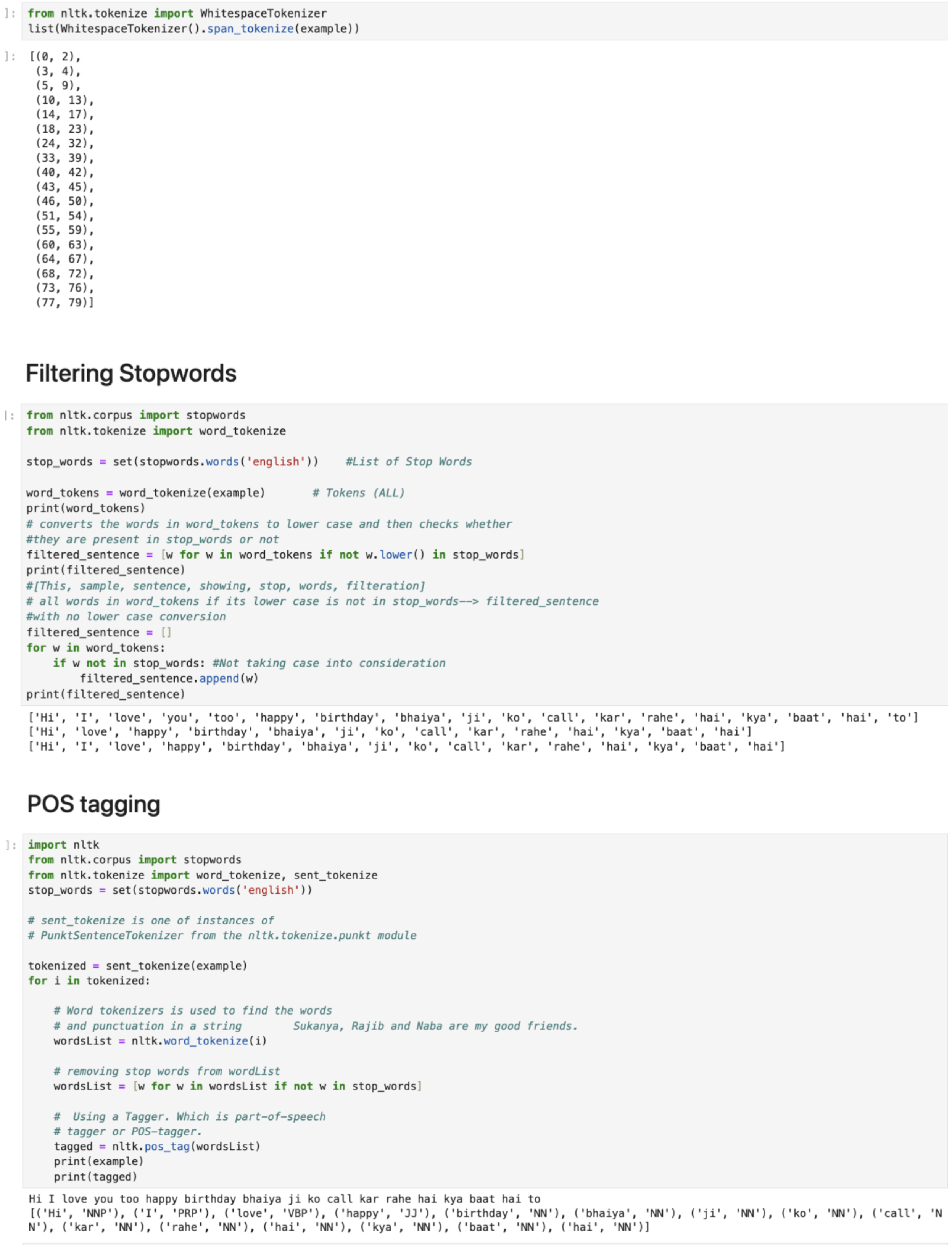




```
[83]: text_obj.similar("Crazy")

      No matches

[84]: len(example)

[84]: 79

[85]: set(example)

[85]: {' ',
       'H',
       'I',
       'a',
       'b',
       'c',
       'd',
       'e',
       'h',
       'i',
       'j',
       'k',
       'l',
       'o',
       'p',
       'r',
       't',
       'u',
       'v',
       'y'}

[86]: sorted(set(example))

[86]: [' ',
       'H',
       'I',
       'a',
       'b',
       'c',
       'd',
       'e',
       'h',
       'i',
       'j',
       'k',
       'l',
       'o',
       'p',
       'r',
       't',
       'u',
       'v',
       'y']

[87]: len(set(example)) / len(example) # Lexical Richness

[87]: 0.25316455696202533
```



```
:  # word tokenization
   print(example.split())

   ['Hi', 'I', 'love', 'you', 'too', 'happy', 'birthday', 'bhaiya', 'ji', 'ko', 'call', 'kar', 'rahe', 'hai', 'kya', 'baat', 'hai', 'to']

:  # sentence tokenizer
   example.split('.')

:  ['Hi I love you too happy birthday bhaiya ji ko call kar rahe hai kya baat hai to']

:  from nltk.tokenize import word_tokenize
   print(word_tokenize(example))

   ['Hi', 'I', 'love', 'you', 'too', 'happy', 'birthday', 'bhaiya', 'ji', 'ko', 'call', 'kar', 'rahe', 'hai', 'kya', 'baat', 'hai', 'to']

:  #sentence_tokenizer
   from nltk.tokenize import sent_tokenize
   sent_tokenize(example)

:  ['Hi I love you too happy birthday bhaiya ji ko call kar rahe hai kya baat hai to']
```



```
from nltk.tokenize import WhitespaceTokenizer
list(WhitespaceTokenizer().span_tokenize(example))
```

```
[(0, 2),
 (3, 4),
 (5, 9),
 (10, 13),
 (14, 17),
 (18, 23),
 (24, 32),
 (33, 39),
 (40, 42),
 (43, 45),
 (46, 50),
 (51, 54),
 (55, 59),
 (60, 63),
 (64, 67),
 (68, 72),
 (73, 76),
 (77, 79)]
```

Filtering Stopwords

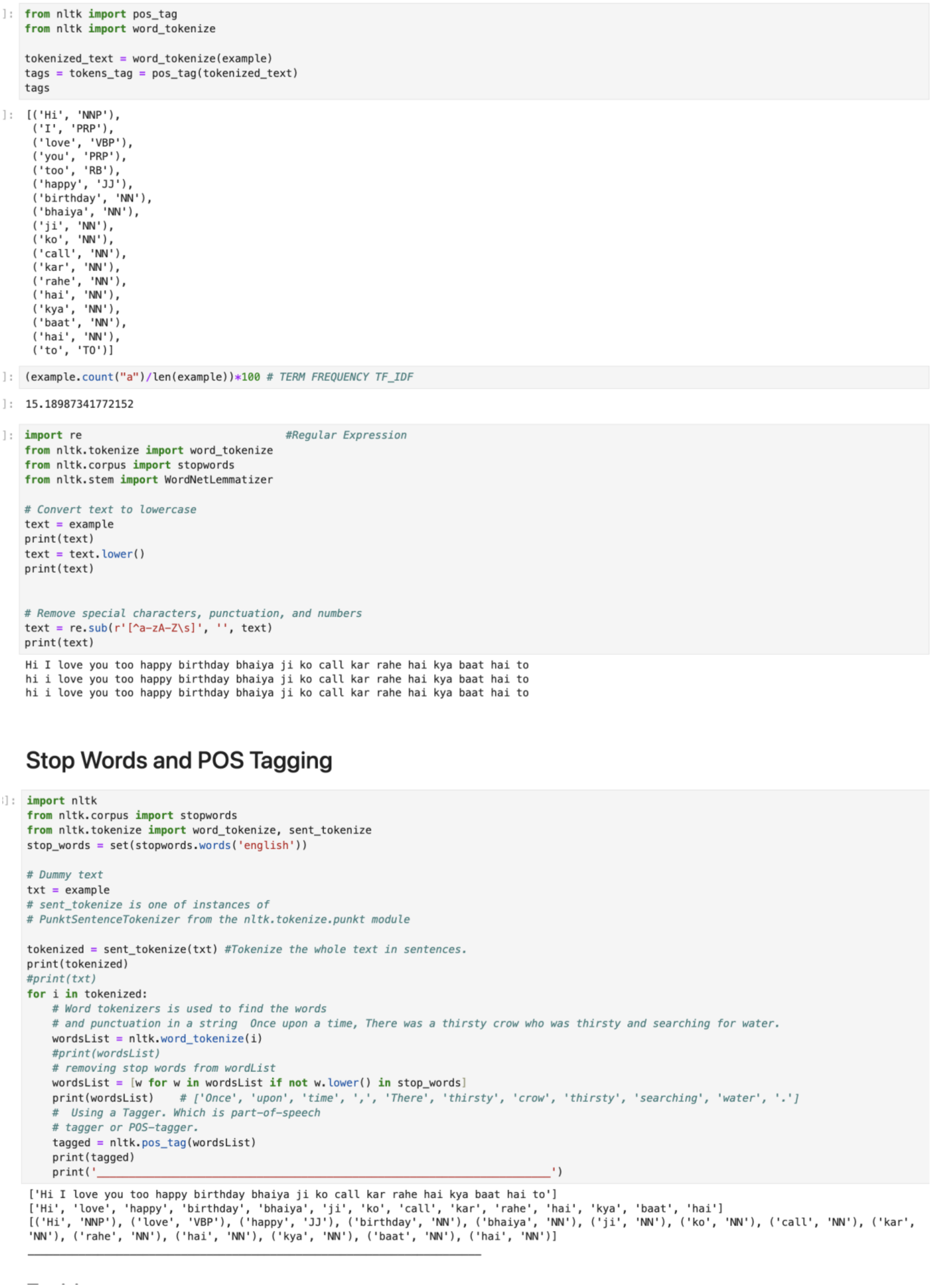
```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

stop_words = set(stopwords.words('english'))    #List of Stop Words

word_tokens = word_tokenize(example)        # Tokens (ALL)
print(word_tokens)
# converts the words in word_tokens to lower case and then checks whether
#they are present in stop_words or not
filtered_sentence = [w for w in word_tokens if not w.lower() in stop_words]
print(filtered_sentence)
#[This, sample, sentence, showing, stop, words, filteration]
# all words in word_tokens if its lower case is not in stop_words--> filtered_sentence
#with no lower case conversion
filtered_sentence = []
for w in word_tokens:
    if w not in stop_words: #Not taking case into consideration
        filtered_sentence.append(w)
print(filtered_sentence)
```

```
['Hi', 'I', 'love', 'you', 'too', 'happy', 'birthday', 'bhaiya', 'ji', 'ko', 'call', 'kar', 'rahe', 'hai', 'kya', 'baat', 'hai', 'to']
['Hi', 'love', 'happy', 'birthday', 'bhaiya', 'ji', 'ko', 'call', 'kar', 'rahe', 'hai', 'kya', 'baat', 'hai']
['Hi', 'I', 'love', 'happy', 'birthday', 'bhaiya', 'ji', 'ko', 'call', 'kar', 'rahe', 'hai', 'kya', 'baat', 'hai']
```

POS tagging

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize
stop_words = set(stopwords.words('english'))

# sent_tokenize is one of instances of
# PunktSentenceTokenizer from the nltk.tokenize.punkt module

tokenized = sent_tokenize(example)
for i in tokenized:

    # Word tokenizers is used to find the words
    # and punctuation in a string        Sukanya, Rajib and Naba are my good friends.
    wordsList = nltk.word_tokenize(i)

    # removing stop words from wordList
    wordsList = [w for w in wordsList if not w in stop_words]

    #  Using a Tagger. Which is part-of-speech
    # tagger or POS-tagger.
    tagged = nltk.pos_tag(wordsList)
    print(example)
    print(tagged)
```

```
Hi I love you too happy birthday bhaiya ji ko call kar rahe hai kya baat hai to
[('Hi', 'NNP'), ('I', 'PRP'), ('love', 'VBP'), ('happy', 'JJ'), ('birthday', 'NN'), ('bhaiya', 'NN'), ('ji', 'NN'), ('ko', 'NN'), ('call', 'N
N'), ('kar', 'NN'), ('rahe', 'NN'), ('hai', 'NN'), ('kya', 'NN'), ('baat', 'NN'), ('hai', 'NN')]
```



```
]: from nltk import pos_tag
   from nltk import word_tokenize

   tokenized_text = word_tokenize(example)
   tags = tokens_tag = pos_tag(tokenized_text)
   tags
```

```
]: [('Hi', 'NNP'),
    ('I', 'PRP'),
    ('love', 'VBP'),
    ('you', 'PRP'),
    ('too', 'RB'),
    ('happy', 'JJ'),
    ('birthday', 'NN'),
    ('bhaiya', 'NN'),
    ('ji', 'NN'),
    ('ko', 'NN'),
    ('call', 'NN'),
    ('kar', 'NN'),
    ('rahe', 'NN'),
    ('hai', 'NN'),
    ('kya', 'NN'),
    ('baat', 'NN'),
    ('hai', 'NN'),
    ('to', 'TO')]
```

```
]: (example.count("a")/len(example))*100 # TERM FREQUENCY TF_IDF
```

```
]: 15.18987341772152
```

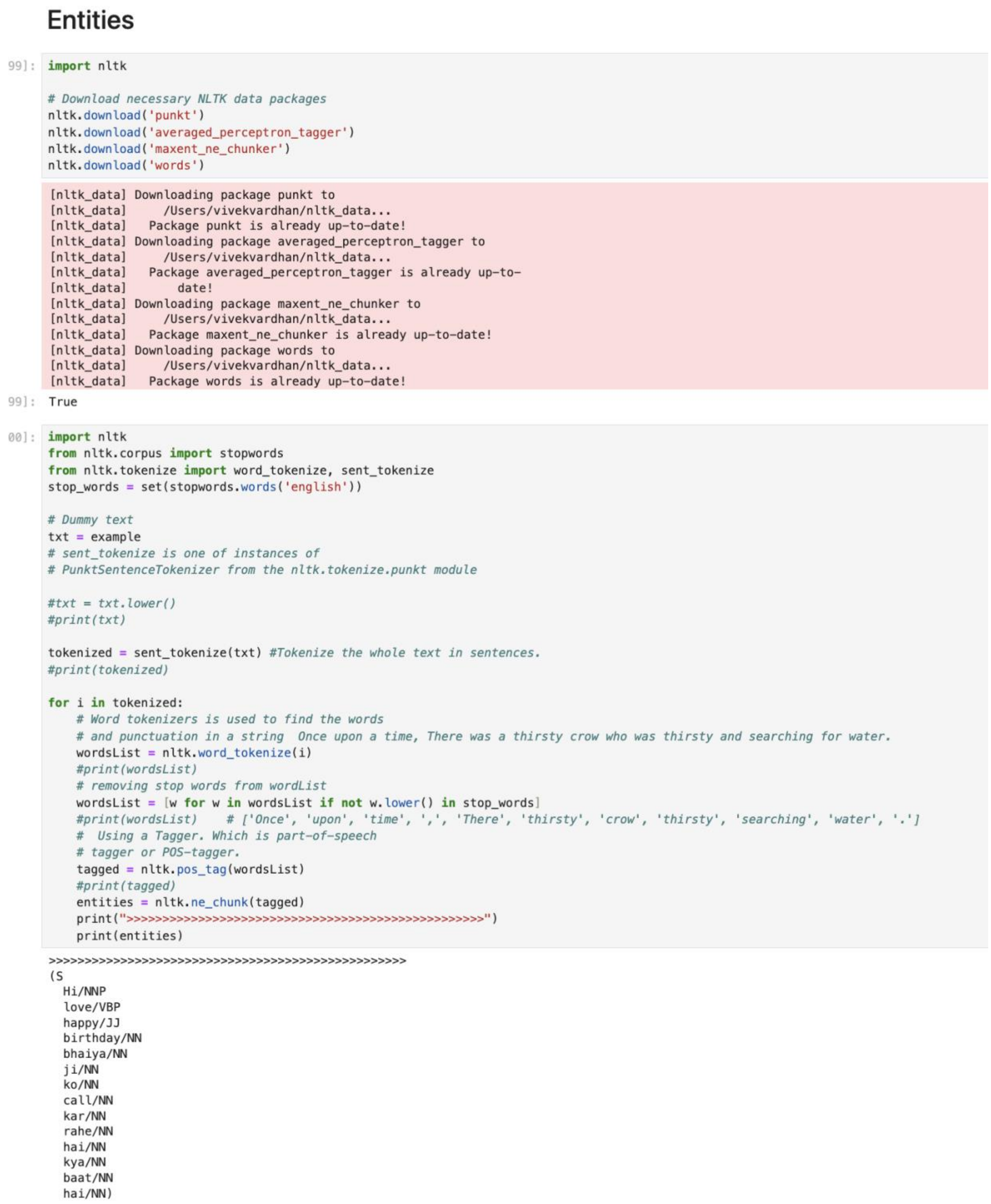
```
]: import re                              #Regular Expression
   from nltk.tokenize import word_tokenize
   from nltk.corpus import stopwords
   from nltk.stem import WordNetLemmatizer

   # Convert text to lowercase
   text = example
   print(text)
   text = text.lower()
   print(text)


   # Remove special characters, punctuation, and numbers
   text = re.sub(r'[^a-zA-Z\s]', '', text)
   print(text)
```

```
   Hi I love you too happy birthday bhaiya ji ko call kar rahe hai kya baat hai to
   hi i love you too happy birthday bhaiya ji ko call kar rahe hai kya baat hai to
   hi i love you too happy birthday bhaiya ji ko call kar rahe hai kya baat hai to
```

Stop Words and POS Tagging

```
]: import nltk
   from nltk.corpus import stopwords
   from nltk.tokenize import word_tokenize, sent_tokenize
   stop_words = set(stopwords.words('english'))

   # Dummy text
   txt = example
   # sent_tokenize is one of instances of
   # PunktSentenceTokenizer from the nltk.tokenize.punkt module

   tokenized = sent_tokenize(txt) #Tokenize the whole text in sentences.
   print(tokenized)
   #print(txt)
   for i in tokenized:
       # Word tokenizers is used to find the words
       # and punctuation in a string  Once upon a time, There was a thirsty crow who was thirsty and searching for water.
       wordsList = nltk.word_tokenize(i)
       #print(wordsList)
       # removing stop words from wordList
       wordsList = [w for w in wordsList if not w.lower() in stop_words]
       print(wordsList)    # ['Once', 'upon', 'time', ',', 'There', 'thirsty', 'crow', 'thirsty', 'searching', 'water', '.']
       #  Using a Tagger. Which is part-of-speech
       # tagger or POS-tagger.
       tagged = nltk.pos_tag(wordsList)
       print(tagged)
       print('______________________________________________________________________________')
```

```
['Hi I love you too happy birthday bhaiya ji ko call kar rahe hai kya baat hai to']
['Hi', 'love', 'happy', 'birthday', 'bhaiya', 'ji', 'ko', 'call', 'kar', 'rahe', 'hai', 'kya', 'baat', 'hai']
[('Hi', 'NNP'), ('love', 'VBP'), ('happy', 'JJ'), ('birthday', 'NN'), ('bhaiya', 'NN'), ('ji', 'NN'), ('ko', 'NN'), ('call', 'NN'), ('kar',
'NN'), ('rahe', 'NN'), ('hai', 'NN'), ('kya', 'NN'), ('baat', 'NN'), ('hai', 'NN')]
______________________________________________________________________________
```



Entities

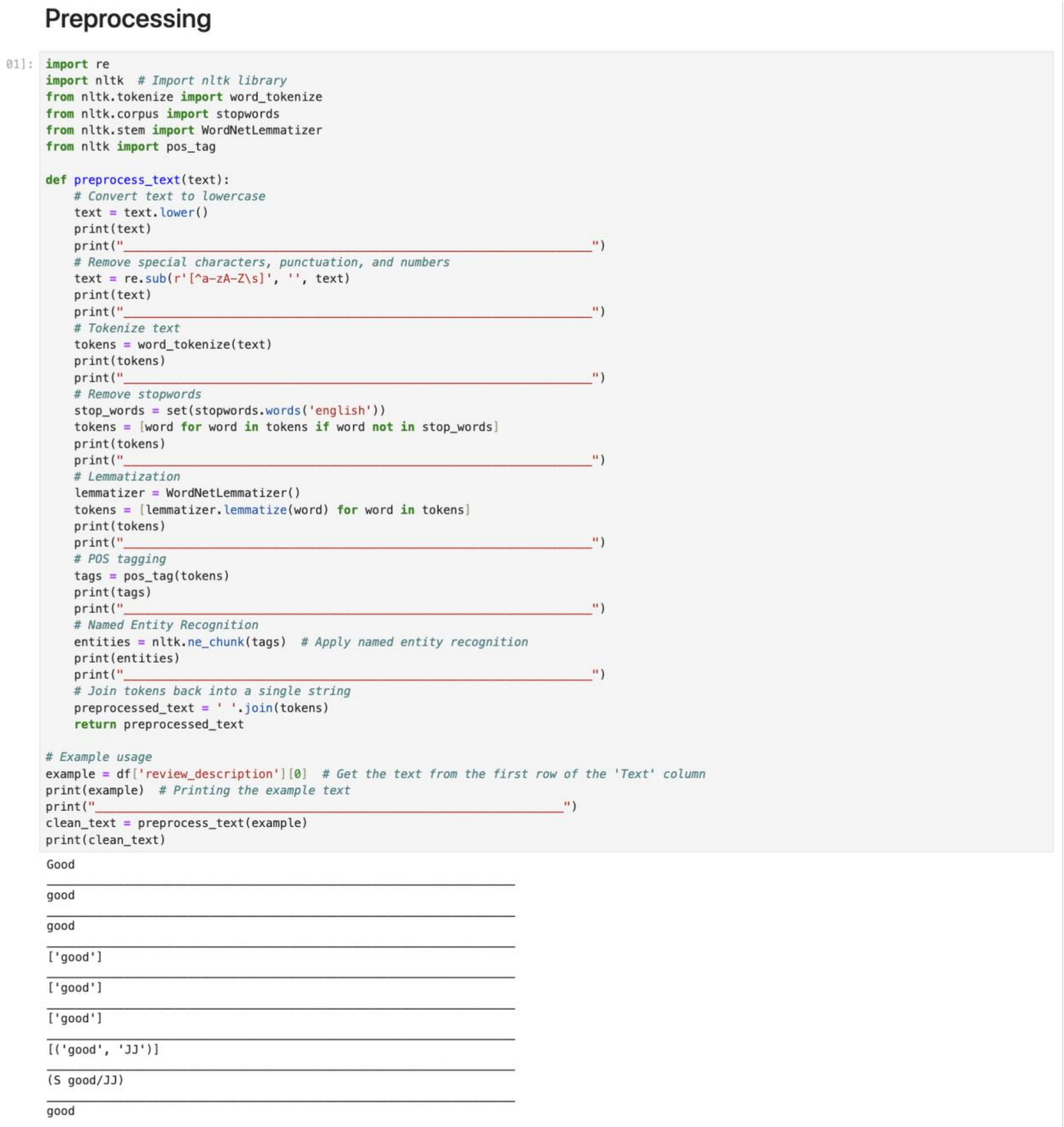
```
99]: import nltk

     # Download necessary NLTK data packages
     nltk.download('punkt')
     nltk.download('averaged_perceptron_tagger')
     nltk.download('maxent_ne_chunker')
     nltk.download('words')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/vivekvardhan/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /Users/vivekvardhan/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     /Users/vivekvardhan/nltk_data...
[nltk_data]   Package maxent_ne_chunker is already up-to-date!
[nltk_data] Downloading package words to
[nltk_data]     /Users/vivekvardhan/nltk_data...
[nltk_data]   Package words is already up-to-date!
```

```
99]: True
```

```
00]: import nltk
     from nltk.corpus import stopwords
     from nltk.tokenize import word_tokenize, sent_tokenize
     stop_words = set(stopwords.words('english'))

     # Dummy text
     txt = example
     # sent_tokenize is one of instances of
     # PunktSentenceTokenizer from the nltk.tokenize.punkt module

     #txt = txt.lower()
     #print(txt)

     tokenized = sent_tokenize(txt) #Tokenize the whole text in sentences.
     #print(tokenized)

     for i in tokenized:
         # Word tokenizers is used to find the words
         # and punctuation in a string  Once upon a time, There was a thirsty crow who was thirsty and searching for water.
         wordsList = nltk.word_tokenize(i)
         #print(wordsList)
         # removing stop words from wordList
         wordsList = [w for w in wordsList if not w.lower() in stop_words]
         #print(wordsList)    # ['Once', 'upon', 'time', ',', 'There', 'thirsty', 'crow', 'thirsty', 'searching', 'water', '.']
         #  Using a Tagger. Which is part-of-speech
         # tagger or POS-tagger.
         tagged = nltk.pos_tag(wordsList)
         #print(tagged)
         entities = nltk.ne_chunk(tagged)
         print(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>")
         print(entities)
```

```
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
(S
  Hi/NNP
  love/VBP
  happy/JJ
  birthday/NN
  bhaiya/NN
  ji/NN
  ko/NN
  call/NN
  kar/NN
  rahe/NN
  hai/NN
  kya/NN
  baat/NN
  hai/NN)
```

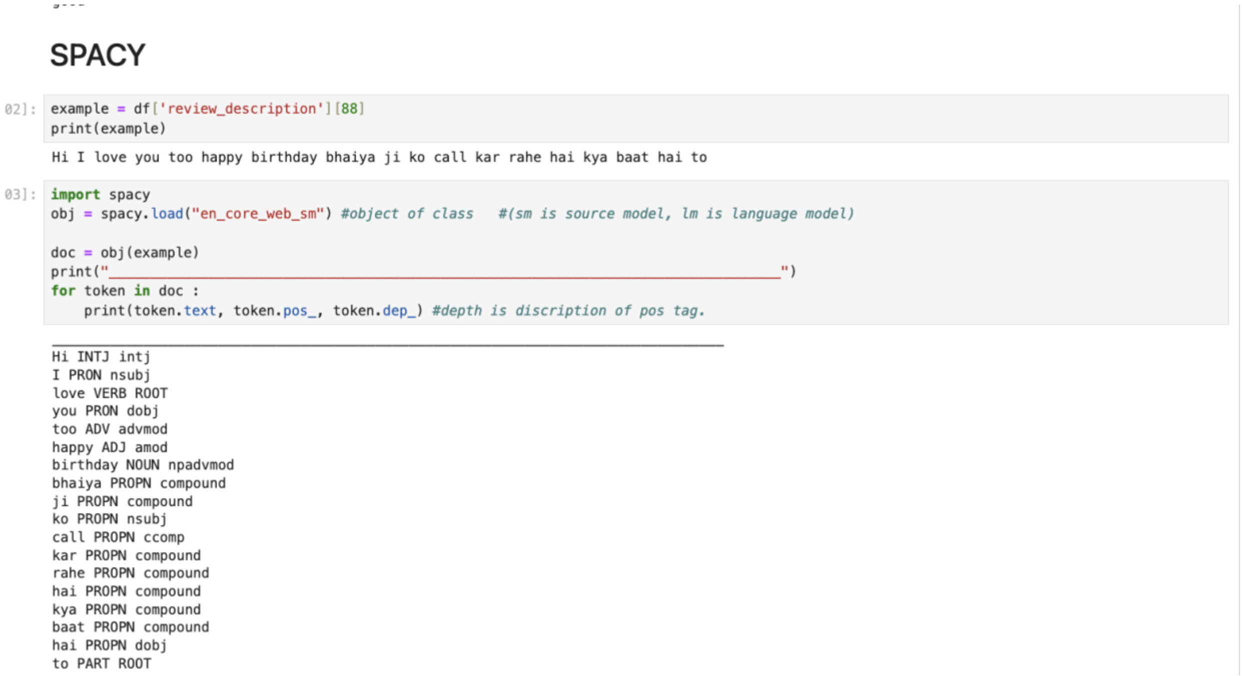


Preprocessing

01]:

```
import re
import nltk  # Import nltk library
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk import pos_tag

def preprocess_text(text):
    # Convert text to lowercase
    text = text.lower()
    print(text)
    print("__________________________________________________________________")
    # Remove special characters, punctuation, and numbers
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    print(text)
    print("__________________________________________________________________")
    # Tokenize text
    tokens = word_tokenize(text)
    print(tokens)
    print("__________________________________________________________________")
    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word not in stop_words]
    print(tokens)
    print("__________________________________________________________________")
    # Lemmatization
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(word) for word in tokens]
    print(tokens)
    print("__________________________________________________________________")
    # POS tagging
    tags = pos_tag(tokens)
    print(tags)
    print("__________________________________________________________________")
    # Named Entity Recognition
    entities = nltk.ne_chunk(tags)  # Apply named entity recognition
    print(entities)
    print("__________________________________________________________________")
    # Join tokens back into a single string
    preprocessed_text = ' '.join(tokens)
    return preprocessed_text

# Example usage
example = df['review_description'][0]  # Get the text from the first row of the 'Text' column
print(example)  # Printing the example text
print("__________________________________________________________________")
clean_text = preprocess_text(example)
print(clean_text)
```

```
Good
__________________________________________________________________
good
__________________________________________________________________
good
__________________________________________________________________
['good']
__________________________________________________________________
['good']
__________________________________________________________________
['good']
__________________________________________________________________
[('good', 'JJ')]
__________________________________________________________________
(S good/JJ)
__________________________________________________________________
good
```



SPACY

02]:

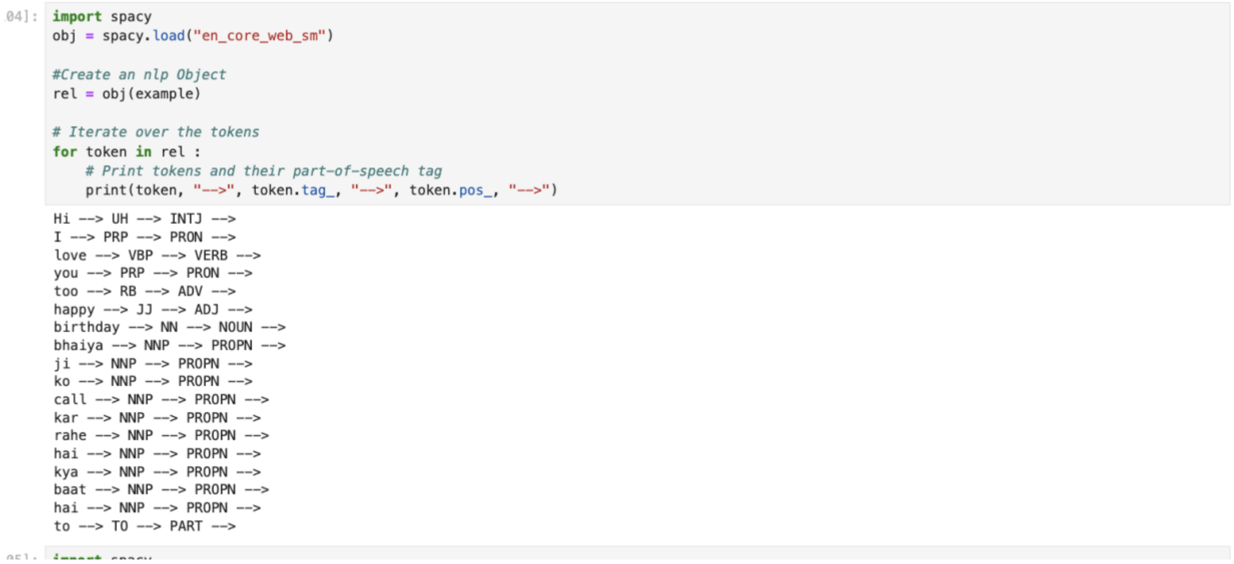
```python
example = df['review_description'][88]
print(example)
```

```
Hi I love you too happy birthday bhaiya ji ko call kar rahe hai kya baat hai to
```

03]:

```python
import spacy
obj = spacy.load("en_core_web_sm") #object of class   #(sm is source model, lm is language model)

doc = obj(example)
print("________________________________________________________________________________")
for token in doc :
    print(token.text, token.pos_, token.dep_) #depth is discription of pos tag.
```

```
________________________________________________________________________________
Hi INTJ intj
I PRON nsubj
love VERB ROOT
you PRON dobj
too ADV advmod
happy ADJ amod
birthday NOUN npadvmod
bhaiya PROPN compound
ji PROPN compound
ko PROPN nsubj
call PROPN ccomp
kar PROPN compound
rahe PROPN compound
hai PROPN compound
kya PROPN compound
baat PROPN compound
hai PROPN dobj
to PART ROOT
```



04]:

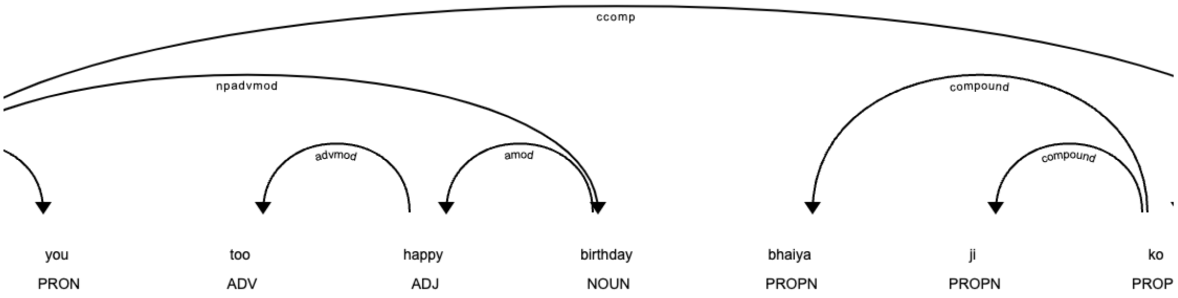
```python
import spacy
obj = spacy.load("en_core_web_sm")

#Create an nlp Object
rel = obj(example)

# Iterate over the tokens
for token in rel :
    # Print tokens and their part-of-speech tag
    print(token, "-->", token.tag_, "-->", token.pos_, "-->")
```

```
Hi --> UH --> INTJ -->
I --> PRP --> PRON -->
love --> VBP --> VERB -->
you --> PRP --> PRON -->
too --> RB --> ADV -->
happy --> JJ --> ADJ -->
birthday --> NN --> NOUN -->
bhaiya --> NNP --> PROPN -->
ji --> NNP --> PROPN -->
ko --> NNP --> PROPN -->
call --> NNP --> PROPN -->
kar --> NNP --> PROPN -->
rahe --> NNP --> PROPN -->
hai --> NNP --> PROPN -->
kya --> NNP --> PROPN -->
baat --> NNP --> PROPN -->
hai --> NNP --> PROPN -->
to --> TO --> PART -->
```

05]:

```python
import spacy
```

[105]:
```
import spacy
from spacy import displacy
nlp = spacy.load("en_core_web_sm")

doc = nlp(example)
displacy.render(doc, style = "dep", jupyter = True)
```



[106]:
```
import spacy
obj = spacy.load("en_core_web_sm") # Object of class Spacy

# Create an obj object
rel = obj(example)

# Iterate over the tokens
for token in rel:
    # Print the token and its parts-of-speech tag
    print(token, "---->", token.tag_, "---->", token.pos_,"---->", spacy.explain(token.tag_))
```

```
Hi ----> UH ----> INTJ ----> interjection
I ----> PRP ----> PRON ----> pronoun, personal
love ----> VBP ----> VERB ----> verb, non-3rd person singular present
you ----> PRP ----> PRON ----> pronoun, personal
too ----> RB ----> ADV ----> adverb
happy ----> JJ ----> ADJ ----> adjective (English), other noun-modifier (Chinese)
birthday ----> NN ----> NOUN ----> noun, singular or mass
bhaiya ----> NNP ----> PROPN ----> noun, proper singular
ji ----> NNP ----> PROPN ----> noun, proper singular
ko ----> NNP ----> PROPN ----> noun, proper singular
call ----> NNP ----> PROPN ----> noun, proper singular
kar ----> NNP ----> PROPN ----> noun, proper singular
rahe ----> NNP ----> PROPN ----> noun, proper singular
hai ----> NNP ----> PROPN ----> noun, proper singular
kya ----> NNP ----> PROPN ----> noun, proper singular
baat ----> NNP ----> PROPN ----> noun, proper singular
hai ----> NNP ----> PROPN ----> noun, proper singular
to ----> TO ----> PART ----> infinitival "to"
```

[107]:
```
for word in rel.ents:
    print(word.text, word.label_)
```

```
ji ko PERSON
```

[108]:
```
for ent in rel.ents:
    print(ent.text, ent.start_char, ent.end_char, ent.label_)
```

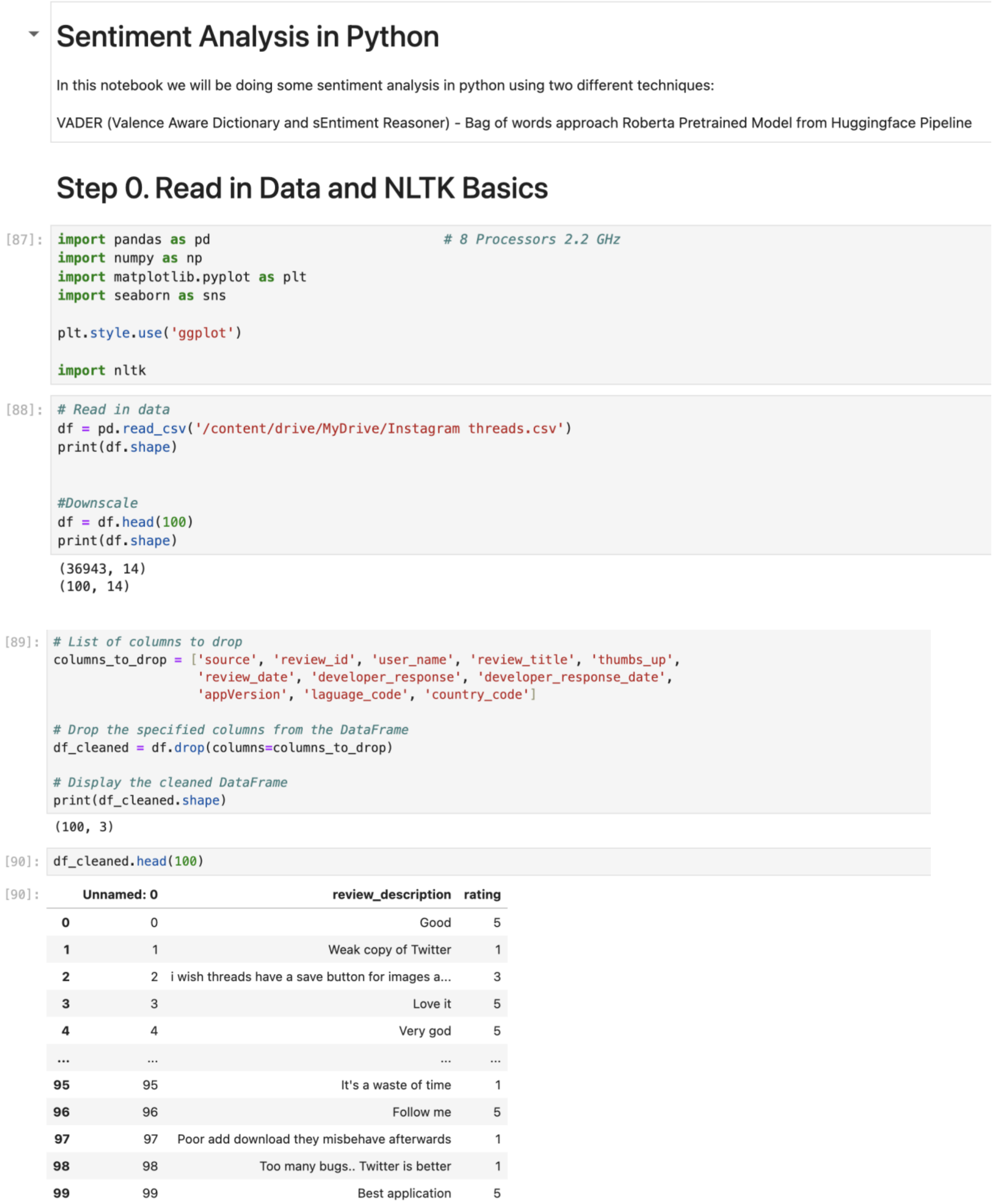
```
ji ko 40 45 PERSON
```

[109]:
```
spacy.explain("FAC")
```

[109]:
```
'Buildings, airports, highways, bridges, etc.'
```

[110]:
```
displacy.render(doc,style="ent",jupyter=True)
```

Hi I love you too happy birthday bhaiya ji ko PERSON call kar rahe hai kya baat hai to



Sentiment Analysis in Python

In this notebook we will be doing some sentiment analysis in python using two different techniques:

VADER (Valence Aware Dictionary and sEntiment Reasoner) - Bag of words approach Roberta Pretrained Model from Huggingface Pipeline

Step 0. Read in Data and NLTK Basics

[87]:

```
import pandas as pd                              # 8 Processors 2.2 GHz
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

plt.style.use('ggplot')

import nltk
```

[88]:

```
# Read in data
df = pd.read_csv('/content/drive/MyDrive/Instagram threads.csv')
print(df.shape)


#Downscale
df = df.head(100)
print(df.shape)
```

```
(36943, 14)
(100, 14)
```

[89]:

```
# List of columns to drop
columns_to_drop = ['source', 'review_id', 'user_name', 'review_title', 'thumbs_up',
                   'review_date', 'developer_response', 'developer_response_date',
                   'appVersion', 'laguage_code', 'country_code']

# Drop the specified columns from the DataFrame
df_cleaned = df.drop(columns=columns_to_drop)

# Display the cleaned DataFrame
print(df_cleaned.shape)
```

```
(100, 3)
```

[90]:

```
df_cleaned.head(100)
```

[90]:

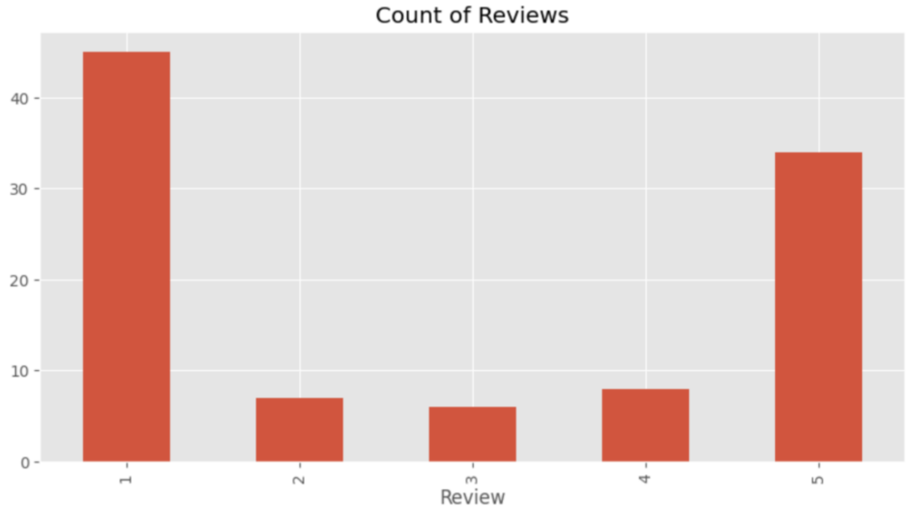
	Unnamed: 0	review_description	rating
0	0	Good	5
1	1	Weak copy of Twitter	1
2	2	i wish threads have a save button for images a...	3
3	3	Love it	5
4	4	Very god	5
...	...	...	...
95	95	It's a waste of time	1
96	96	Follow me	5
97	97	Poor add download they misbehave afterwards	1
98	98	Too many bugs.. Twitter is better	1
99	99	Best application	5

```
[91]: df.rename(columns={'Unnamed: 0': 'index'}, inplace=True)

# Now 'df' will have the column name changed from 'Unnamed: 0' to 'index'
print(df.head())
```

```
   index       source                             review_id  \
0      0  Google Play  7cd90e5b-4829-43b9-9fb4-c8c6d1e339c1
1      1  Google Play  6deb8265-2bac-4524-bcb6-f90829fa4e69
2      2  Google Play  91ef61ce-0f05-4f3b-b3d3-5d19cd408ab8
3      3  Google Play  b7721b78-6b77-4f8c-a1d3-a854af4c1f0f
4      4  Google Play  c89ef522-c94c-4171-878f-1d672dce7f11

                    user_name review_title  \
0             Eddie Clark Jr.          NaN
1                     Rasa RT          NaN
2  SITI NUR HAFIZA BINTI AZIZ          NaN
3               Asap Khalifah          NaN
4                Syed Hussein          NaN

                                  review_description  rating  thumbs_up  \
0                                               Good       5        0.0
1                               Weak copy of Twitter       1        0.0
2  i wish threads have a save button for images a...       3        0.0
3                                            Love it       5        0.0
4                                           Very god       5        0.0

           review_date  developer_response  developer_response_date  \
0  2023-08-07 19:14:36                 NaN                      NaN
1  2023-08-07 19:07:04                 NaN                      NaN
2  2023-08-07 18:57:07                 NaN                      NaN
3  2023-08-07 18:37:16                 NaN                      NaN
4  2023-08-07 18:14:15                 NaN                      NaN

       appVersion laguage_code country_code
0  294.0.0.27.110           en           us
1             NaN           en           us
2  294.0.0.27.110           en           us
3             NaN           en           us
4             NaN           en           us
```

Quick EDA

```
[92]: ax = df['rating'].value_counts().sort_index() \
          .plot(kind='bar',
          title='Count of Reviews',
          figsize=(10, 5))
ax.set_xlabel('Review')
plt.show()
```



Basic NLTK

```
example = df['review_description'][46]
print(example)
```

```
An exciting new way to see Ads and posts from people you don't care about! At least you don't have to worry about seeing up any preferences.
```

```
tokens = nltk.word_tokenize(example)
tokens[:12]
```

```
['An',
 'exciting',
 'new',
 'way',
 'to',
 'see',
 'Ads',
 'and',
 'posts',
 'from',
 'people',
 'you']
```

```
tagged = nltk.pos_tag(tokens)
tagged[-5:]
```

```
[('seeing', 'VBG'),
 ('up', 'RP'),
 ('any', 'DT'),
 ('preferences', 'NNS'),
 ('.', '.')]
```

```
tagged = nltk.pos_tag(tokens)
tagged[-5:]
```

```
[('seeing', 'VBG'),
 ('up', 'RP'),
 ('any', 'DT'),
 ('preferences', 'NNS'),
 ('.', '.')]
```

```
entities = nltk.chunk.ne_chunk(tagged)
print(entities)
```

```
(S
  An/DT
  exciting/JJ
  new/JJ
  way/NN
  to/TO
  see/VB
  (PERSON Ads/NNP)
  and/CC
  posts/NNS
  from/IN
  people/NNS
  you/PRP
  do/VBP
  n't/RB
  care/VB
  about/IN
  !/.
  At/IN
  least/JJS
  you/PRP
  do/VBP
  n't/RB
  have/VB
  to/TO
  worry/VB
  about/IN
  seeing/VBG
  up/RP
  any/DT
  preferences/NNS
  ./.)
```

Step 1. VADER Seniment Scoring

We will use NLTK's SentimentIntensityAnalyzer to get the neg/neu/pos scores of the text.

(Valence Aware Dictionary and sEntiment Reasoner)

This uses a "bag of words" approach: Stop words are removed each word is scored and combined to a total score.

```
from nltk.sentiment import SentimentIntensityAnalyzer
from tqdm.notebook import tqdm

sia = SentimentIntensityAnalyzer()
```

```
sia.polarity_scores('The curious cat jumped over the lazy dog and landed gracefully on the fluffy pillow.')
```

```
{'neg': 0.124, 'neu': 0.594, 'pos': 0.282, 'compound': 0.4939}
```

```
sia.polarity_scores('True friends are like stars; even in the darkest of times, they shine brightly and guide us..')
```

```
{'neg': 0.123, 'neu': 0.46, 'pos': 0.418, 'compound': 0.7717}
```

```
print(example)
sia.polarity_scores(example)
```

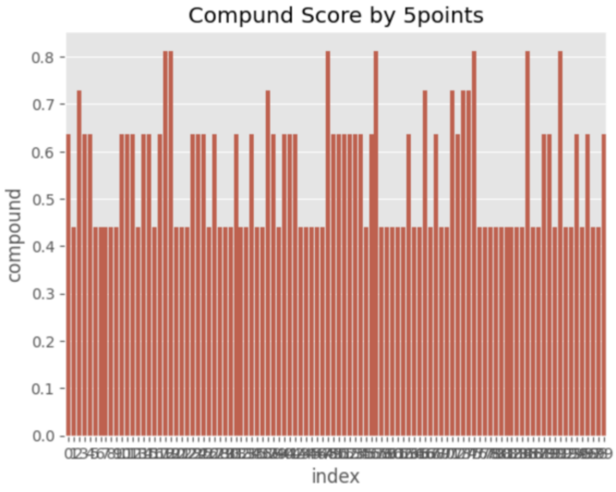
```
An exciting new way to see Ads and posts from people you don't care about! At least you don't have to worry about seeing up any preferences.
{'neg': 0.081, 'neu': 0.738, 'pos': 0.181, 'compound': 0.5057}
```

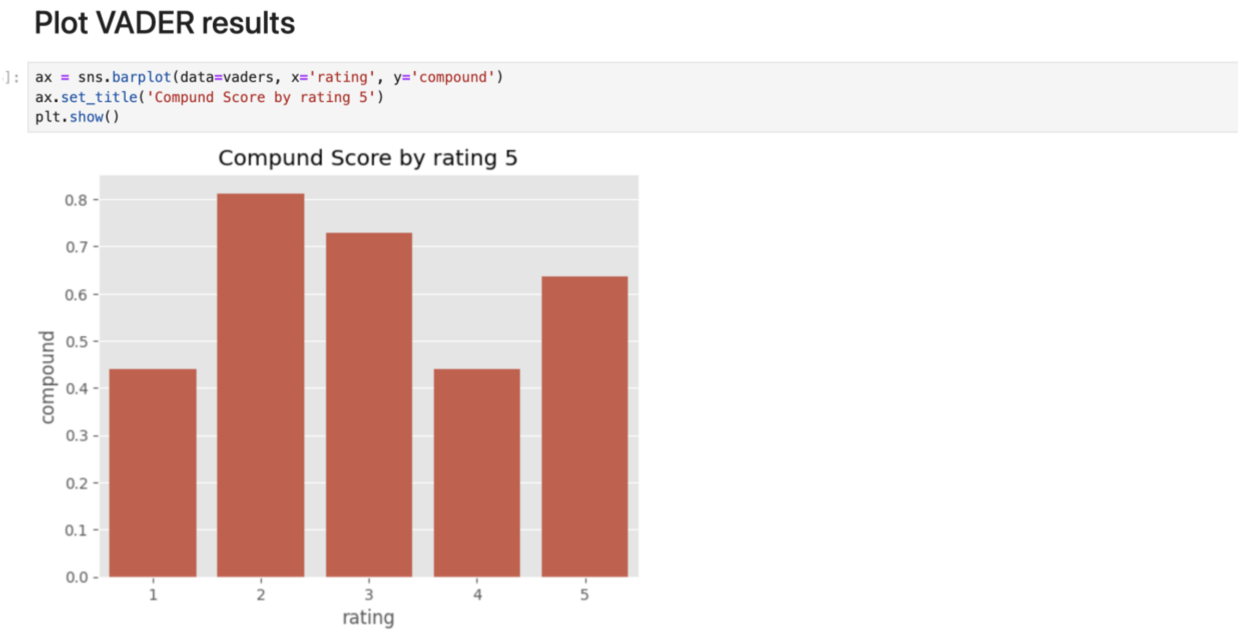
```
extdata = {}
for i, row in tqdm(df.iterrows(), total=len(df)):
    text = row['review_description']
    myid = row['rating']
    extdata[myid] = sia.polarity_scores(text)
```

```
  0%|          | 0/100 [00:00<?, ?it/s]
```

```
vaders = pd.DataFrame(extdata).T
vaders = vaders.reset_index().rename(columns={'index': 'rating'})
vaders = vaders.merge(df, how='left')

```

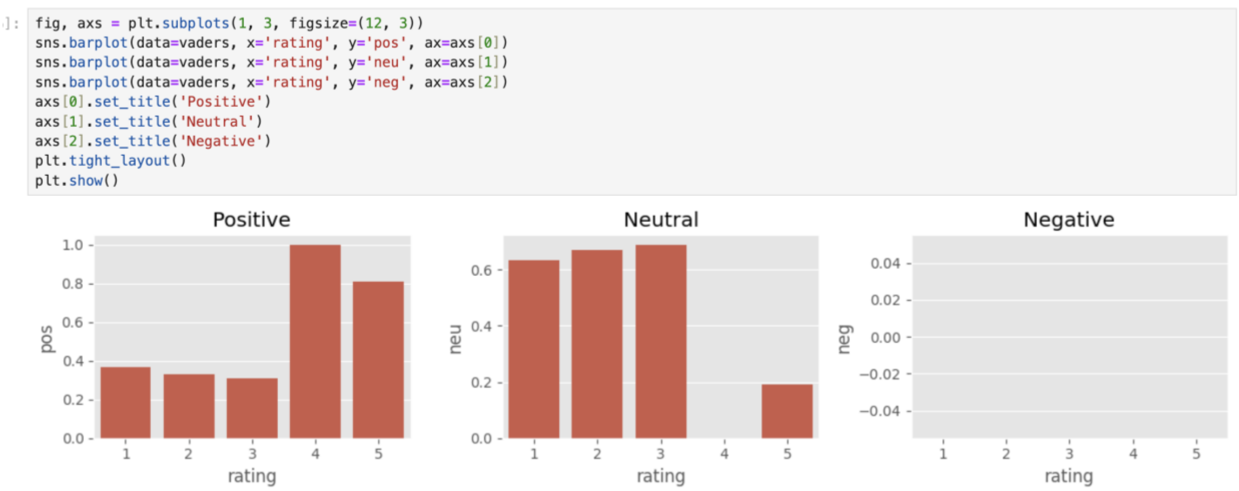
```
ax = sns.barplot(data=vaders, x='index', y='compound')
ax.set_title('Compund Score by 5points')
plt.show()
```





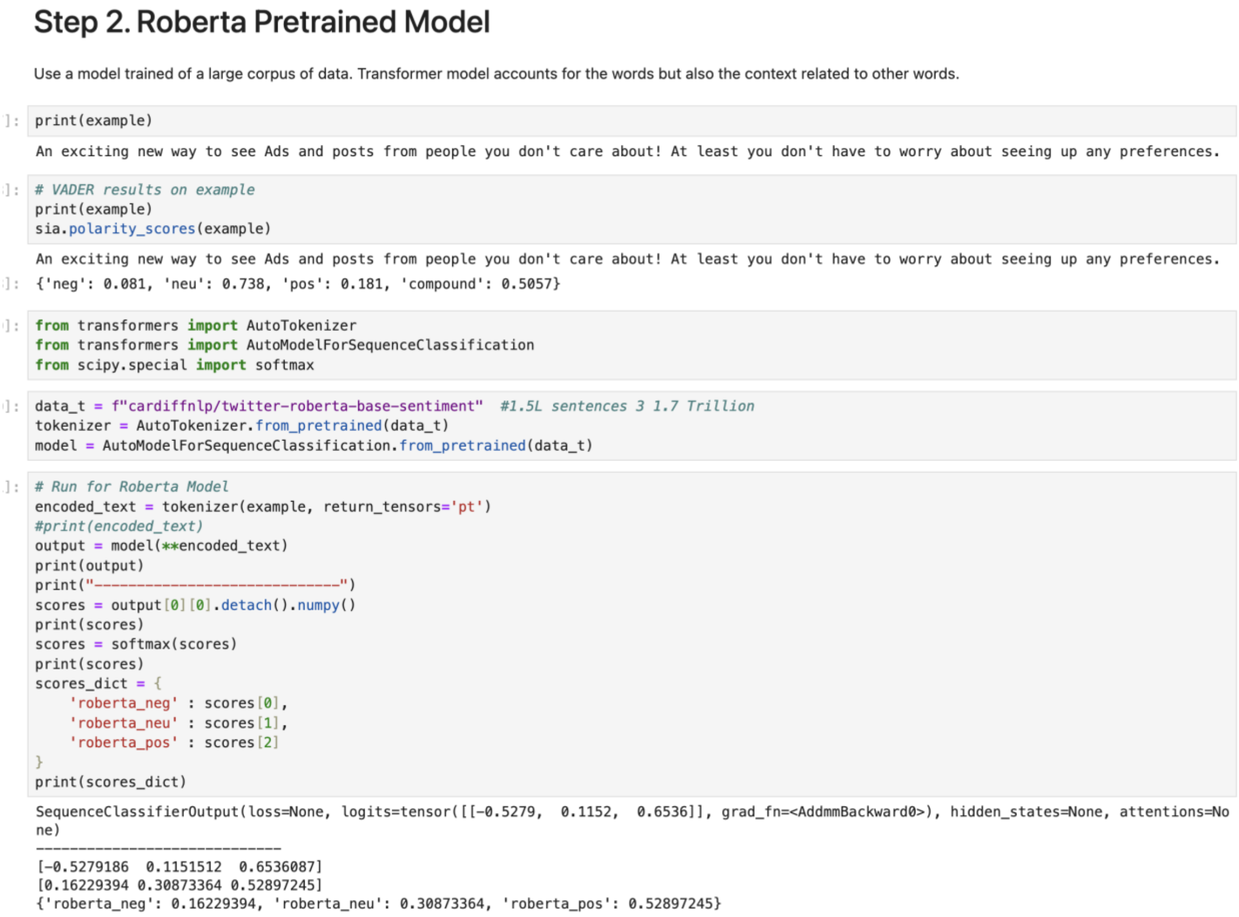
Plot VADER results

```
ax = sns.barplot(data=vaders, x='rating', y='compound')
ax.set_title('Compund Score by rating 5')
plt.show()
```



```
fig, axs = plt.subplots(1, 3, figsize=(12, 3))
sns.barplot(data=vaders, x='rating', y='pos', ax=axs[0])
sns.barplot(data=vaders, x='rating', y='neu', ax=axs[1])
sns.barplot(data=vaders, x='rating', y='neg', ax=axs[2])
axs[0].set_title('Positive')
axs[1].set_title('Neutral')
axs[2].set_title('Negative')
plt.tight_layout()
plt.show()
```

The reviews for the Threads app are typically short, consisting of 5 to 6 words. Most of these reviews were posted when the app was first launched. Although some reviews express negative sentiments through words, they do not necessarily correspond to negative ratings.



Step 2. Roberta Pretrained Model

Use a model trained of a large corpus of data. Transformer model accounts for the words but also the context related to other words.

```
]: print(example)
```

```
An exciting new way to see Ads and posts from people you don't care about! At least you don't have to worry about seeing up any preferences.
```

```
]: # VADER results on example
print(example)
sia.polarity_scores(example)
```

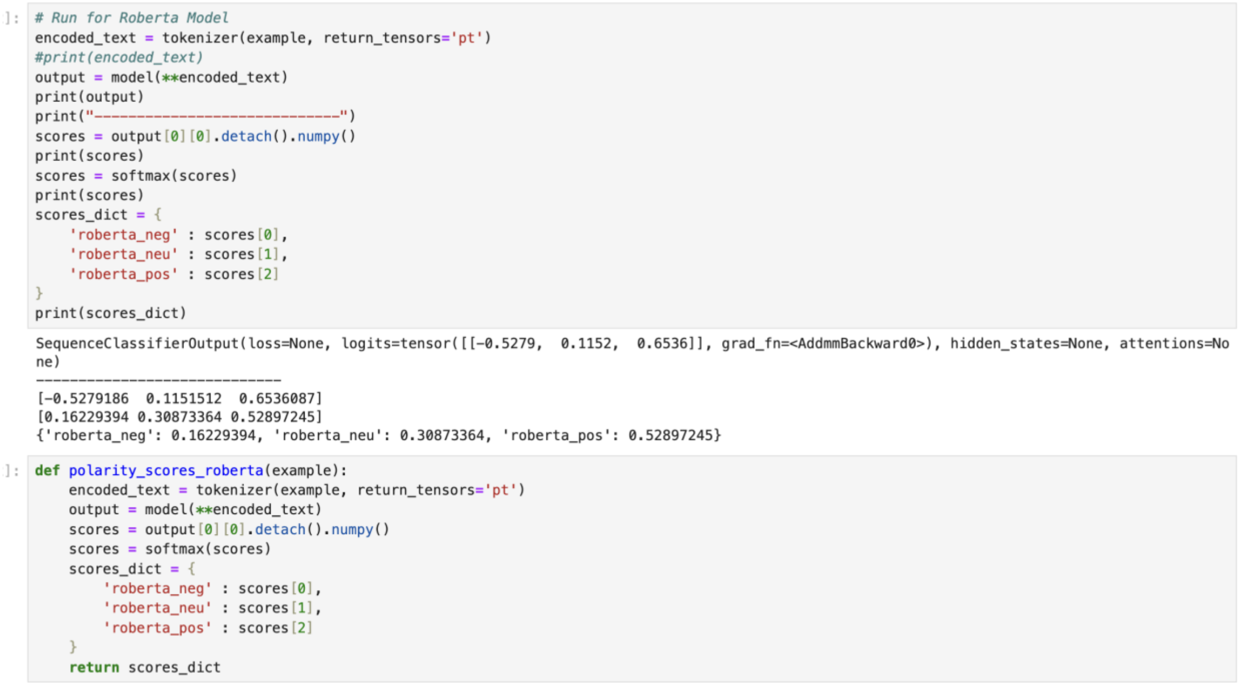
```
An exciting new way to see Ads and posts from people you don't care about! At least you don't have to worry about seeing up any preferences.
]: {'neg': 0.081, 'neu': 0.738, 'pos': 0.181, 'compound': 0.5057}
```

```
]: from transformers import AutoTokenizer
from transformers import AutoModelForSequenceClassification
from scipy.special import softmax
```

```
]: data_t = f"cardiffnlp/twitter-roberta-base-sentiment"  #1.5L sentences 3 1.7 Trillion
tokenizer = AutoTokenizer.from_pretrained(data_t)
model = AutoModelForSequenceClassification.from_pretrained(data_t)
```

```
]: # Run for Roberta Model
encoded_text = tokenizer(example, return_tensors='pt')
#print(encoded_text)
output = model(**encoded_text)
print(output)
print("-----------------------------")
scores = output[0][0].detach().numpy()
print(scores)
scores = softmax(scores)
print(scores)
scores_dict = {
    'roberta_neg' : scores[0],
    'roberta_neu' : scores[1],
    'roberta_pos' : scores[2]
}
print(scores_dict)
```

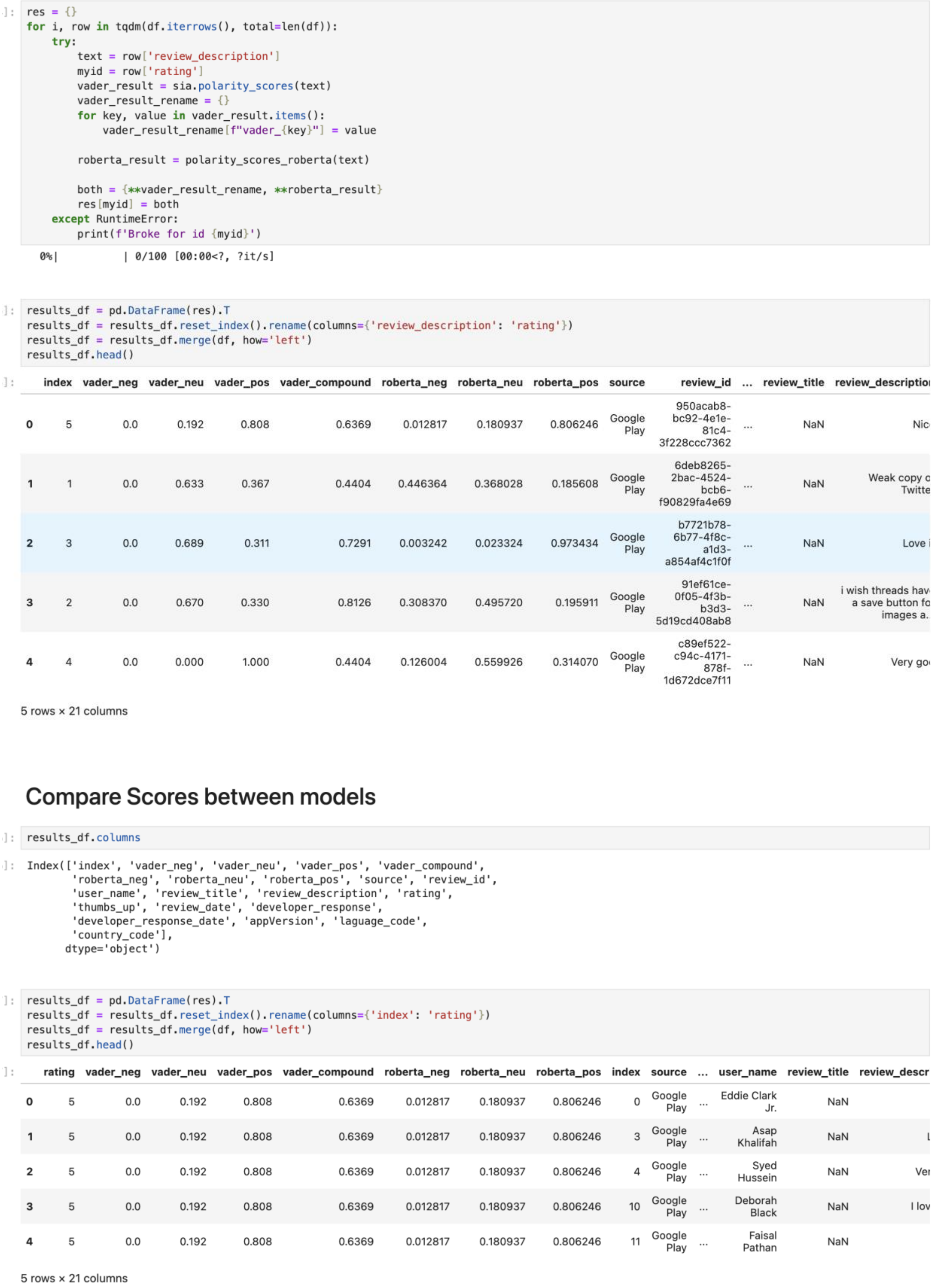
```
SequenceClassifierOutput(loss=None, logits=tensor([[-0.5279,  0.1152,  0.6536]], grad_fn=<AddmmBackward0>), hidden_states=None, attentions=No
ne)
-----------------------------
[-0.5279186  0.1151512  0.6536087]
[0.16229394 0.30873364 0.52897245]
{'roberta_neg': 0.16229394, 'roberta_neu': 0.30873364, 'roberta_pos': 0.52897245}
```



```
]: # Run for Roberta Model
encoded_text = tokenizer(example, return_tensors='pt')
#print(encoded_text)
output = model(**encoded_text)
print(output)
print("-----------------------------")
scores = output[0][0].detach().numpy()
print(scores)
scores = softmax(scores)
print(scores)
scores_dict = {
    'roberta_neg' : scores[0],
    'roberta_neu' : scores[1],
    'roberta_pos' : scores[2]
}
print(scores_dict)
```

```
SequenceClassifierOutput(loss=None, logits=tensor([[-0.5279,  0.1152,  0.6536]], grad_fn=<AddmmBackward0>), hidden_states=None, attentions=No
ne)
-----------------------------
[-0.5279186  0.1151512  0.6536087]
[0.16229394 0.30873364 0.52897245]
{'roberta_neg': 0.16229394, 'roberta_neu': 0.30873364, 'roberta_pos': 0.52897245}
```

```
]: def polarity_scores_roberta(example):
    encoded_text = tokenizer(example, return_tensors='pt')
    output = model(**encoded_text)
    scores = output[0][0].detach().numpy()
    scores = softmax(scores)
    scores_dict = {
        'roberta_neg' : scores[0],
        'roberta_neu' : scores[1],
        'roberta_pos' : scores[2]
    }
    return scores_dict
```



```
]: res = {}
for i, row in tqdm(df.iterrows(), total=len(df)):
    try:
        text = row['review_description']
        myid = row['rating']
        vader_result = sia.polarity_scores(text)
        vader_result_rename = {}
        for key, value in vader_result.items():
            vader_result_rename[f"vader_{key}"] = value

        roberta_result = polarity_scores_roberta(text)

        both = {**vader_result_rename, **roberta_result}
        res[myid] = both
    except RuntimeError:
        print(f'Broke for id {myid}')
```

```
  0%|          | 0/100 [00:00<?, ?it/s]
```

```
]: results_df = pd.DataFrame(res).T
results_df = results_df.reset_index().rename(columns={'review_description': 'rating'})
results_df = results_df.merge(df, how='left')
results_df.head()
```

]:

	index	vader_neg	vader_neu	vader_pos	vader_compound	roberta_neg	roberta_neu	roberta_pos	source	review_id	...	review_title	review_descriptio
0	5	0.0	0.192	0.808	0.6369	0.012817	0.180937	0.806246	Google Play	950acab8-bc92-4e1e-81c4-3f228ccc7362	...	NaN	Nic
1	1	0.0	0.633	0.367	0.4404	0.446364	0.368028	0.185608	Google Play	6deb8265-2bac-4524-bcb6-f90829fa4e69	...	NaN	Weak copy c Twitte
2	3	0.0	0.689	0.311	0.7291	0.003242	0.023324	0.973434	Google Play	b7721b78-6b77-4f8c-a1d3-a854af4c1f0f	...	NaN	Love i
3	2	0.0	0.670	0.330	0.8126	0.308370	0.495720	0.195911	Google Play	91ef61ce-0f05-4f3b-b3d3-5d19cd408ab8	...	NaN	i wish threads hav a save button fo images a.
4	4	0.0	0.000	1.000	0.4404	0.126004	0.559926	0.314070	Google Play	c89ef522-c94c-4171-878f-1d672dce7f11	...	NaN	Very go

5 rows × 21 columns

Compare Scores between models

```
]: results_df.columns
```

```
]: Index(['index', 'vader_neg', 'vader_neu', 'vader_pos', 'vader_compound',
       'roberta_neg', 'roberta_neu', 'roberta_pos', 'source', 'review_id',
       'user_name', 'review_title', 'review_description', 'rating',
       'thumbs_up', 'review_date', 'developer_response',
       'developer_response_date', 'appVersion', 'laguage_code',
       'country_code'],
      dtype='object')
```

```
]: results_df = pd.DataFrame(res).T
results_df = results_df.reset_index().rename(columns={'index': 'rating'})
results_df = results_df.merge(df, how='left')
results_df.head()
```

]:

	rating	vader_neg	vader_neu	vader_pos	vader_compound	roberta_neg	roberta_neu	roberta_pos	index	source	...	user_name	review_title	review_descr
0	5	0.0	0.192	0.808	0.6369	0.012817	0.180937	0.806246	0	Google Play	...	Eddie Clark Jr.	NaN	
1	5	0.0	0.192	0.808	0.6369	0.012817	0.180937	0.806246	3	Google Play	...	Asap Khalifah	NaN	L
2	5	0.0	0.192	0.808	0.6369	0.012817	0.180937	0.806246	4	Google Play	...	Syed Hussein	NaN	Ver
3	5	0.0	0.192	0.808	0.6369	0.012817	0.180937	0.806246	10	Google Play	...	Deborah Black	NaN	I lov
4	5	0.0	0.192	0.808	0.6369	0.012817	0.180937	0.806246	11	Google Play	...	Faisal Pathan	NaN	

5 rows × 21 columns

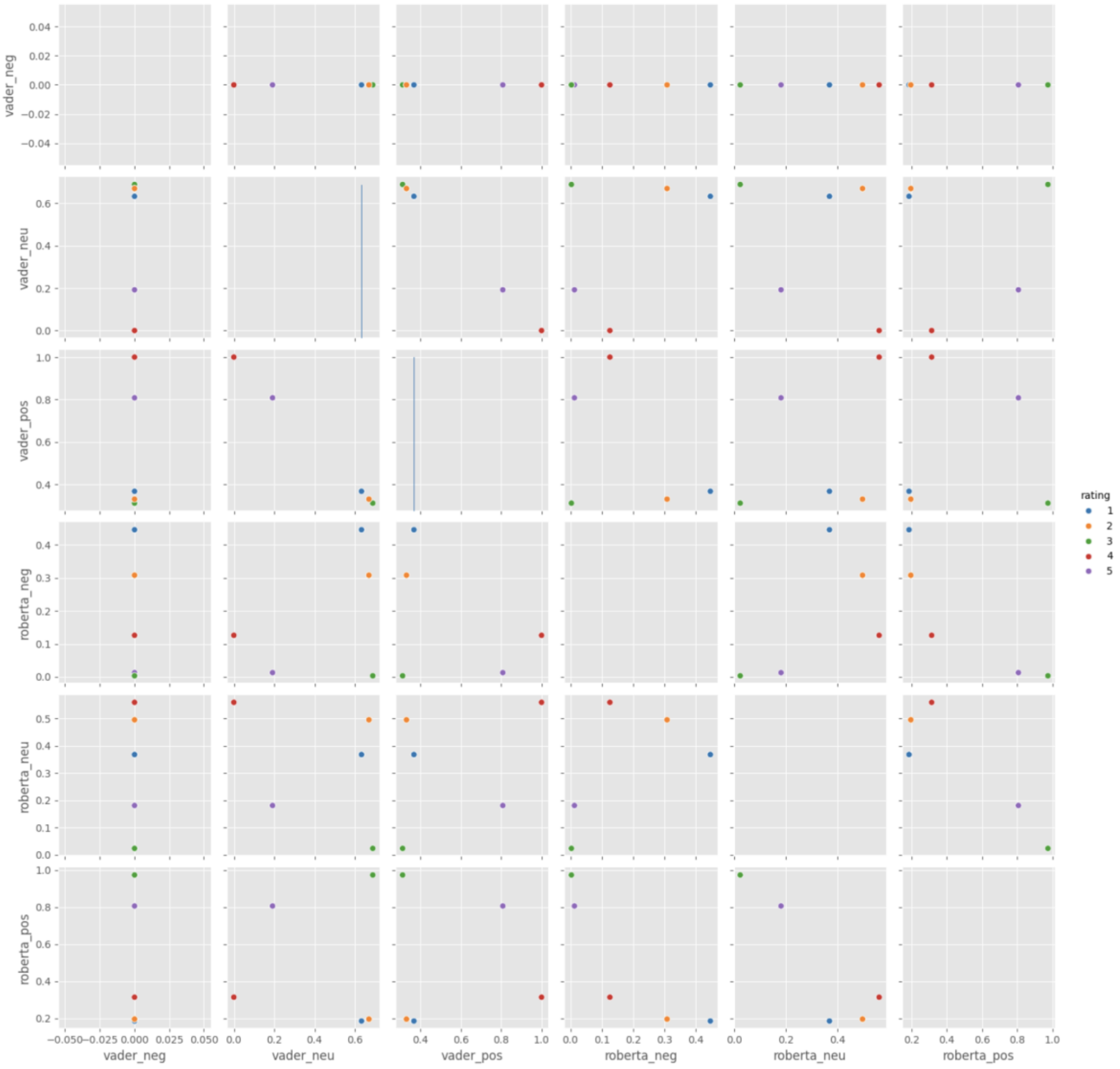
```
:  #JUST TO UNDERSTAND

   vader_r_rename = {'va': 1, 'vb': 2}
   roberta_r = {'rb': 3, 'rc': 4}

   both = {**vader_r_rename, **roberta_r}

   print(both)
```

```
{'va': 1, 'vb': 2, 'rb': 3, 'rc': 4}
```

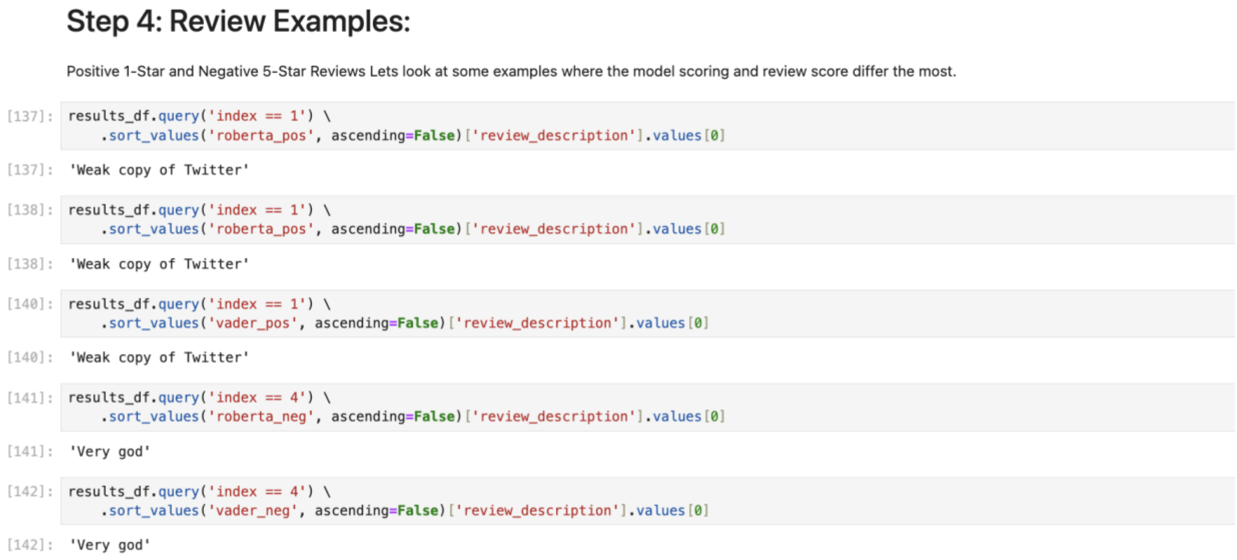
```
:  print(results_df.columns)
```

```
Index(['rating', 'vader_neg', 'vader_neu', 'vader_pos', 'vader_compound',
       'roberta_neg', 'roberta_neu', 'roberta_pos', 'index', 'source',
       'review_id', 'user_name', 'review_title', 'review_description',
       'thumbs_up', 'review_date', 'developer_response',
       'developer_response_date', 'appVersion', 'laguage_code',
       'country_code'],
      dtype='object')
```

```
[135]: sns.pairplot(data=results_df,
                    vars=['vader_neg', 'vader_neu', 'vader_pos',
                          'roberta_neg', 'roberta_neu', 'roberta_pos'],
                    hue='rating',
                    palette='tab10')
       plt.show()
```



The above code calculates polarity scores for VADER and RoBERTa models, creates a DataFrame with the results, and then merges it with the original DataFrame. It then generates plots to visualize the distributions of vader-neg, vader-pos, and roberta-pos. scores, colored by the 'label' column.



Step 4: Review Examples:

Positive 1-Star and Negative 5-Star Reviews Lets look at some examples where the model scoring and review score differ the most.

```
[137]: results_df.query('index == 1') \
           .sort_values('roberta_pos', ascending=False)['review_description'].values[0]

[137]: 'Weak copy of Twitter'

[138]: results_df.query('index == 1') \
           .sort_values('roberta_pos', ascending=False)['review_description'].values[0]

[138]: 'Weak copy of Twitter'

[140]: results_df.query('index == 1') \
           .sort_values('vader_pos', ascending=False)['review_description'].values[0]

[140]: 'Weak copy of Twitter'

[141]: results_df.query('index == 4') \
           .sort_values('roberta_neg', ascending=False)['review_description'].values[0]

[141]: 'Very god'

[142]: results_df.query('index == 4') \
           .sort_values('vader_neg', ascending=False)['review_description'].values[0]

[142]: 'Very god'
```

Bibliography

https://www.kaggle.com/datasets/shuvammandal121/37000-reviews-of-thread-app-dataset

The link from where the dataset is downloaded from.

References taken from GitHub and Kaggle.