

```
! pip install emoji
```

```
➦ Requirement already satisfied: emoji in /usr/local/lib/python3.11/dist-pack
```

```
!pip install nltk
```

```
➦ Requirement already satisfied: nltk in /usr/local/lib/python3.11/dist-packa  
Requirement already satisfied: click in /usr/local/lib/python3.11/dist-pack  
Requirement already satisfied: joblib in /usr/local/lib/python3.11/dist-pac  
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.11  
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packa
```

```
!pip install tqdm textblob emoji
```

```
➦ Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packa  
Requirement already satisfied: textblob in /usr/local/lib/python3.11/dist-p  
Requirement already satisfied: emoji in /usr/local/lib/python3.11/dist-pack  
Requirement already satisfied: nltk>=3.9 in /usr/local/lib/python3.11/dist-  
Requirement already satisfied: click in /usr/local/lib/python3.11/dist-pack  
Requirement already satisfied: joblib in /usr/local/lib/python3.11/dist-pac  
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.11
```

```
!pip install tqdm
```

```
➦ Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packa
```

```
!pip install nltk pandas scikit-learn matplotlib seaborn wordcloud emoji textblob
```

```

Requirement already satisfied: nltk in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/di
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist
Requirement already satisfied: seaborn in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: wordcloud in /usr/local/lib/python3.11/dist-
Requirement already satisfied: emoji in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: textblob in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: rank_bm25 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: click in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: joblib in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: regex<=2021.8.3 in /usr/local/lib/python3.11
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/d
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/pyt
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/di
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/di
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/pytho
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.1
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/di
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.1
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-p

```

```

from google.colab import drive
drive.mount('/content/drive')

```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, ca
```

```
# Import necessary libraries
import pandas as pd
import numpy as np
import re
import emoji
import nltk
from tqdm import tqdm
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk import word_tokenize, pos_tag
from collections import Counter

# Download necessary NLTK resources
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
nltk.download('omw-1.4')
```

```
↗ [nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data] /root/nltk_data...
[nltk_data] Package averaged_perceptron_tagger is already up-to-
[nltk_data] date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data] Package omw-1.4 is already up-to-date!
True
```

```
nltk.download('punkt')
```

```
↗ [nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Package punkt is already up-to-date!
True
```

## ✓ 1-Textual EDA

```
df=pd.read_csv("/content/drive/MyDrive/FE/twitter_training copy.csv")
```

Start coding or [generate](#) with AI.

```
df.head()
```



	<b>Tweet ID</b>	<b>Entity</b>	<b>Sentiment</b>	<b>Tweet content</b>
<b>0</b>	2401	Borderlands	Positive	im getting on borderlands and i will murder yo...
<b>1</b>	2401	Borderlands	Positive	I am coming to the borders and I will kill you...
<b>2</b>	2401	Borderlands	Positive	im getting on borderlands and i will kill you ...
<b>3</b>	2401	Borderlands	Positive	im coming on borderlands and i will murder you...
<b>4</b>	2401	Borderlands	Positive	im getting on borderlands 2 and i will murder ...

```
df.dtypes
```



	<b>0</b>
<b>Tweet ID</b>	int64
<b>Entity</b>	object
<b>Sentiment</b>	object
<b>Tweet content</b>	object

**dtype:** object

```
df = df[df['Sentiment'] != 'Sentiment']
df = df.dropna(subset=['Tweet content'])
```

```
df.reset_index(drop=True, inplace=True)
```

```
df.shape
```



```
(73996, 4)
```

```
df.isnull().sum().sum()
```



```
np.int64(0)
```

```
df.isnull().sum()
```

```
0
Tweet ID    0
Entity      0
Sentiment   0
Tweet content 0
```

```
dtype: int64
```

```
df.dropna(inplace=True)
```

```
df.sort_values(by=['Tweet ID'], inplace=True)
```

```
df.head()
```

```
0      Tweet ID  Entity  Sentiment  Tweet content
4627         1  Amazon    Negative    7 @amazon wtf.
4626         1  Amazon    Negative    @amazon wtf?
4623         1  Amazon    Negative    @amazon wtf .
4628         1  Amazon    Negative    <unk> wtf.
4624         1  Amazon    Negative    @ amazon wtf.
```

```
df.duplicated().sum()
```

```
np.int64(2341)
```

```
df.drop_duplicates(inplace=True)
```

```
df = df[df['Sentiment'] != 'Irrelevant']
df = df.reset_index(drop=True)
print(df['Sentiment'].value_counts())
```

```
↗ Sentiment
Negative    21698
Positive    19713
Neutral     17707
Name: count, dtype: int64
```

## ✓ 2-Text Preprocessing

```
df.shape
```

```
↗ (59118, 4)
```

```
# Compile patterns for fast regex
url_pattern = re.compile(r"http\S+|www\S+|https\S+")
mention_pattern = re.compile(r'\@\w+')
hashtag_pattern = re.compile(r'\#')
special_chars_pattern = re.compile(r'^a-zA-Z\s')
digits_pattern = re.compile(r'\d+')
whitespace_pattern = re.compile(r'\s+')

# Define text cleaning function
def clean_text(text):
    text = str(text)
    text = emoji.demojize(text) # Convert emoji to text
    text = text.lower() # Lowercase
    text = url_pattern.sub('', text) # Remove URLs
    text = mention_pattern.sub('', text) # Remove mentions
    text = hashtag_pattern.sub('', text) # Remove hashtags symbol
    text = special_chars_pattern.sub('', text) # Remove special characters
    text = digits_pattern.sub('', text) # Remove digits
    text = whitespace_pattern.sub(' ', text).strip() # Normalize spaces
    return text

# Apply cleaning
tqdm.pandas()
df['cleaned_text'] = df['Tweet content'].progress_apply(clean_text)
```

```
↗ 100%|██████████| 59118/59118 [00:05<00:00, 10717.07it/s]
```

```
df.shape
```

```
(59118, 5)
```

```
df[['Tweet content', 'cleaned_text']].sample(5)
```

	<b>Tweet content</b>	<b>cleaned_text</b>
<b>2303</b>	"you're not not diamond yet?"	youre not not diamond yet
<b>10213</b>	OK you fell like	ok you fell like
<b>19747</b>	I Absolutely	i absolutely
<b>16383</b>	Good morning! It was so hot and... gross last ...	good morning it was so hot and gross last sund...
<b>27120</b>	People dissing this but imo this is great	people dissing this but imo this is great

```
df['cleaned_text'].isnull().sum()
```

```
np.int64(0)
```

```
df[df['Tweet content'].str.contains('@|#|http|www|:')] [['Tweet content', 'cleaned_text']]
```

	<b>Tweet content</b>	<b>cleaned_text</b>
<b>25238</b>	Today went so cool. I decided the give outdoor...	today went so cool i decided the give outdoor ...
<b>55491</b>	Hey @amazon. @KraftHeinzCo. Then @BestBuy. @Pr...	hey then always soon eddie hey i get your to m...
<b>33837</b>	@EAMaddenNFL giive me my money back. Your game...	giive me my money back your game is broken yal...
<b>10000</b>	One of its own movies is live w/ @ProfZeroo	one of its own movies is live w catch him

```
df.head()
```



	<b>Tweet ID</b>	<b>Entity</b>	<b>Sentiment</b>	<b>Tweet content</b>	<b>cleaned_text</b>
<b>0</b>	1	Amazon	Negative	7 @amazon wtf.	wtf
<b>1</b>	1	Amazon	Negative	@amazon wtf?	wtf
<b>2</b>	1	Amazon	Negative	@amazon wtf .	wtf
<b>3</b>	1	Amazon	Negative	<unk> wtf.	unk wtf
<b>4</b>	1	Amazon	Negative	@ amazon wtf.	amazon wtf

```
df[['Tweet content', 'cleaned_text']].head(100)
```



	<b>Tweet content</b>	<b>cleaned_text</b>
<b>0</b>	7 @amazon wtf.	wtf
<b>1</b>	@amazon wtf?	wtf
<b>2</b>	@amazon wtf .	wtf
<b>3</b>	<unk> wtf.	unk wtf
<b>4</b>	@ amazon wtf.	amazon wtf
...	...	...
<b>95</b>	Amazon Bestseller: Best Sports Collectibles if...	amazon bestseller best sports collectibles ift...
<b>96</b>	Amazon Bestsellers: Best Sports Includes i.tt ...	amazon bestsellers best sports includes itt vp...
<b>97</b>	BUDDY LOVE: More about: Amazon.com / stores / ...	buddy love more about amazoncom stores buddylo...
<b>98</b>	<unk> LOVE: More from amazon.com/stores/BUDDYL...	unk love more from amazoncomstoresbuddylo the ...

## ✓ 3-Lemmatization



```

import nltk

# Download the 'punkt_tab' resource
nltk.download('punkt_tab')

# Download the required resource for the pos_tag function
# Original line: nltk.download('averaged_perceptron_tagger')
nltk.download('averaged_perceptron_tagger_eng') # Download the specific 'eng' r

# Setup Lemmatizer
lemmatizer = WordNetLemmatizer()

# Mapping POS tags to WordNet
def get_wordnet_pos(tag):
    if tag.startswith('J'):
        return 'a'
    elif tag.startswith('V'):
        return 'v'
    elif tag.startswith('N'):
        return 'n'
    elif tag.startswith('R'):
        return 'r'
    else:
        return 'n'

# Lemmatization function
def lemmatize_text(text):
    words = word_tokenize(text)
    pos_tags = pos_tag(words)
    lemmatized_words = [lemmatizer.lemmatize(word, get_wordnet_pos(pos)) for word, pos in pos_tags]
    return ' '.join(lemmatized_words)

# Apply lemmatization
df['lemmatized_text'] = df['cleaned_text'].progress_apply(lemmatize_text)


```

```

[↪] [nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]   /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger_eng.zip.
100%|██████████| 59118/59118 [00:48<00:00, 1208.65it/s]

```

```
df[['cleaned_text', 'lemmatized_text']].sample(20)
```



	<b>cleaned_text</b>	<b>lemmatized_text</b>
<b>37486</b>	every where and then i start to reconsider my ...	every where and then i start to reconsider my ...
<b>58278</b>	good to see microsoft is gonna release the xbo...	good to see microsoft be gon na release the xb...
<b>11060</b>	my fave games are minecraft borderlands forza ...	my fave game be minecraft borderland forza hor...
<b>15968</b>	chapter update chase emailed me from behind an...	chapter update chase email me from behind an a...
<b>52342</b>	this is disgusting because i had played matche...	this be disgust because i have play match each...
<b>17881</b>	cyberpunks not there yet just told magda that ...	cyberpunk not there yet just tell magda that i...
<b>17525</b>	as a reward for the th of july i preordered cyb...	a a reward for the th of july i preordered cyb...
<b>43882</b>	interventionist shit	interventionist shit
<b>454</b>	amazon threatens to fire critics who are outsp...	amazon threaten to fire critic who be outspoke...
<b>40975</b>	start your journey with strong partnerships hp...	start your journey with strong partnership hpe...
<b>11730</b>	never again	never again
<b>17402</b>	god fuck	god fuck
<b>57260</b>	anytime i feel lost i would pull out a map and...	anytime i feel lose i would pull out a map and...
<b>30298</b>	unexplained illness adverse effects of radiati...	unexplained illness adverse effect of radiatio...

## ✓ 5-stopwords

```
# Import standard English stopwords
from nltk.corpus import stopwords

# Load standard English stopwords
stop_words = set(stopwords.words('english'))

# Define additional custom stopwords (specific to your Twitter data)
custom_noise_words = set([
    'im', 'dont', 'cant', 'couldnt', 'wouldnt', 'shouldnt', 'wasnt', 'isnt', 'a
    'i', 'you', 'he', 'she', 'it', 'we', 'they', 'me', 'him', 'her', 'us', 'the
    'am', 'is', 'are', 'was', 'were', 'do', 'does', 'did', 'have', 'has', 'had'
    'get', 'got', 'still', 'also', 'even', 'really', 'one', 'two', 'three', 'fo
    'five', 'six', 'seven', 'eight', 'nine', 'ten', 'day', 'week', 'month', 'ye
    'game', 'play', 'go', 'make', 'time', 'see', 'look', 'like', 'new', 'u', 'p
    'could', 'would', 'should', 'may', 'might', 'must',
    'one', 'two', 'three', 'four', 'five', 'also', 'even',
    'get', 'got', 'im', 'amp', 'u', 'ur', 'dont', 'doesnt', 'didnt'
    , 'k', 'ur', 'amp', 'via', 'pictwittercom', "p"
])
```

```
# Merge standard stopwords with custom noise words
full_stopwords = stop_words.union(custom_noise_words)
```

```
# Define the stopwords removal function
def remove_stopwords(text):
    words = text.split()
    filtered_words = [word for word in words if word not in full_stopwords]
    return ' '.join(filtered_words)
```

```
# Apply stopwords removal on the lemmatized text
df['final_text'] = df['lemmatized_text'].progress_apply(remove_stopwords)
```

```
🔄 100%|██████████| 59118/59118 [00:00<00:00, 322317.93it/s]
```

```
def remove_mentions(text):
    words = text.split()
    words = [word for word in words if not word.startswith('@')]
    return ' '.join(words)
```

```
# Apply to your preprocessed text
df['final_text'] = df['final_text'].apply(remove_mentions)
```

```
df[['lemmatized_text', 'final_text']].sample(10)
```



	lemmatized_text	final_text
<b>25864</b>	home depot recently stop hire and they not run...	home depot recently stop hire run background c...
<b>23107</b>	tired hearthstone wire pictwittercomdcfxhdnc	tired hearthstone wire pictwittercomdcfxhdnc
<b>21096</b>	more evidence of left wingunk scamming	evidence left wingunk scamming
<b>46249</b>	why be u do nothing in pakistan against pubg b...	nothing pakistan pubg ban come please issue cl...
<b>5087</b>	god i forget how hot i fuck love assassin cree...	god forget hot fuck love assassin creed fuck
<b>35620</b>	be a sweet guy ragequits it freeze the screen ...	sweet guy ragequits freeze screen force close ...
<b>20013</b>	might just pay for the extra google drive spac...	pay extra google drive space keep real

## 4-Extract Top 150 Terms Based on Term Frequency (TF)

```
# prompt: 4-Extract Top 150 Terms Based on Term Frequency (TF)
```

```
from collections import Counter
```

```
def get_top_n_words(corpus, n=None):
    vec = Counter()
    for text in corpus:
        for word in text.split():
            vec[word] += 1
    return vec.most_common(n)
```

```
top_150_words = get_top_n_words(df['final_text'], 150)
top_150_words
```

```
➦ ('nvidia', 1249),
  ('player', 1242),
  ('thing', 1236),
  ('x', 1229),
  ('need', 1227),
  ('ever', 1223),
  ('help', 1211),
  ('give', 1208),
  ('thank', 1161),
  ('every', 1155),
  ('redemption', 1149),
  ('never', 1135),
  ('world', 1124),
  ('stop', 1113),
  ('depot', 1084),
  ('start', 1076),
  ('series', 1076),
  ('right', 1075),
  ('update', 1073),
  ('borderland', 1065),
  ('gta', 1061),
  ('team', 1053),
  ('v', 1051),
  ('last', 1047),
  ('big', 1020),
  ('pubg', 1014),
  ('live', 1013),
  ('thanks', 1006),
  ('league', 990),
  ('feel', 985),
  ('keep', 969),
  ('damn', 954),
  ('legend', 946),
  ('stream', 939),
  ('com', 936),
  ('black', 934),
  ('na', 933),
  ('yall', 932),
  ('happy', 917),
  ('server', 917),
  ('kill', 903),
  ('actually', 894),
  ('let', 887),
  ('ban', 882),
  ('another', 881),
  ('overwatch', 877),
  ('show', 873),
  ('creed', 871),
```

```
( 'man', 870),
( 'card', 834),
( 'problem', 829),
( 'everyone', 824),
( 'next', 819),
( 'way', 814),
( 'battlefield', 809),
( 'always', 803),
( 'service', 801),
( 'duty', 799),
( 'since', 795),
```

```
from sklearn.feature_extraction.text import CountVectorizer
```

```
vectorizer = CountVectorizer(ngram_range=(2, 2))
```

```
X2 = vectorizer.fit_transform(df['final_text'])
```

```
sum_words2 = X2.sum(axis=0)
```

```
words_freq2 = [(word, sum_words2[0, idx]) for word, idx in vectorizer.vocabulary.items()]
```

```
top_150_bigrams = sorted(words_freq2, key=lambda x: x[1], reverse=True)[:150]
```

```
# عرض النتائج
```

```
for word, freq in top_150_bigrams:
    print(word, freq)
```

```
work nome 79
every another 78
good morning 78
much good 76
fuck fuck 74
feel bad 73
much love 72
recon breakpoint 72
microsoft team 71
next gen 69
win global 68
good news 68
global free 67
help win 67
every single 67
thank much 65
google search 65
dead redeem 65
want buy 64
fuck shit 64
check item 64
graphic card 64
check video 63
johnson halt 63
anyone else 62
bad thing 62
show love 62
```

```

warcraft iii 62
ghostrecon uk 61
internet speed 61
great job 60
pretty cool 60
release date 60
let know 59
pretty good 59
fang fang 59
fifa point 59
ift tt 58
good luck 58
open world 58
ive never 57
pre order 57
social medium 57
cat sftp 57
fix shit 56
sea sea 56
bit ly 55
best ever 55
good thing 55
facebook page 55
horrific vision 55
everyday another 54
punch man 54
need help 54
app store 54
pay gig 54
win free 53
ive try 53
win achievement 53

```

```
df.columns
```

```

Index(['Tweet ID', 'Entity', 'Sentiment', 'Tweet content', 'cleaned_text',
      'lemmatized_text', 'final_text', 'original_text'],
      dtype='object')

```

```
df['Sentiment'].unique()
```

```
array(['Negative', 'Neutral', 'Positive'], dtype=object)
```

```
import pandas as pd
from collections import Counter

df['original_text'] = df['Tweet content']

def get_top_unigrams_with_original(df, sentiment_label=None, n=170):
    if sentiment_label:
        subset = df[df['Sentiment'] == sentiment_label]
    else:
        subset = df.copy()

    original_texts = ' '.join(subset['original_text'].dropna()).lower()
    cleaned_texts = ' '.join(subset['final_text'].dropna())

    cleaned_counter = Counter(cleaned_texts.split())
    original_counter = Counter(original_texts.split())

    top_cleaned = cleaned_counter.most_common(n)

    records = []
    for cleaned_word, freq in top_cleaned:
        matching_original = [word for word in original_counter if cleaned_word
                              original_word = matching_original[0] if matching_original else cleaned_
        records.append([original_word, cleaned_word, freq])

    result_df = pd.DataFrame(records, columns=["Original terms", "Terms after p
    return result_df

df_top150_all = get_top_unigrams_with_original(df)

df_top150_positive = get_top_unigrams_with_original(df, 'Positive')
df_top150_negative = get_top_unigrams_with_original(df, 'Negative')
df_top150_irrelevant = get_top_unigrams_with_original(df, 'Irrelevant')
```



```
print(" Top 150 Unigrams in ALL Data")  
display(df_top150_all)
```




Top 150 Unigrams in ALL Data

	Original terms	Terms after pre-processing	TF
0	good	good	3299
1	fucked	fuck	2904
2	love	love	2868
3	johnson	johnson	2860
4	shit	shit	2252
...	...	...	...
165	@biohazzards	oh	570
166	top	top	559
167	they	hey	554
168	nothing	nothing	553
169	suck.	suck	548

170 rows × 3 columns

```
# Add the following lines to define and display df_top150_neutral:
df_top150_neutral = get_top_unigrams_with_original(df, 'Neutral') # Assuming '
print(" Top 150 Unigrams in Neutral Tweets")
display(df_top150_neutral)
```

 Top 150 Unigrams in Neutral Tweets

	Original terms	Terms after pre-processing	TF
0	johnson	johnson	1793
1	amazon	amazon	1107
2	google	google	820
3	facebook	facebook	809
4	win!...	win	795
...	...	...	...
165	luck	luck	166
166	entering	enter	165
167	everything	everything	164
168	always	always	162
169	changes	change	162

170 rows × 3 columns

```
print(" Top 150 Unigrams in Negative Tweets")
display(df_top150_negative)
```



Top 150 Unigrams in Negative Tweets

	Original terms	Terms after pre-processing	TF
0	fucked	fuck	1982
1	shit,	shit	1488
2	fix	fix	1430
3	bad	bad	1181
4	please	please	1112
...	...	...	...
165	bluescreened	screen	229
166	almost	almost	228
167	@ghostrecon	ghostrecon	227
168	yet	yet	224
169	old	old	223

170 rows × 3 columns

```
print(" Top 150 Unigrams in Neutral Tweets")
display(df_top150_neutral)
```

↗ Top 150 Unigrams in Neutral Tweets

	Original terms	Terms after pre-processing	TF
0	johnson	johnson	1793
1	amazon	amazon	1107
2	google	google	820
3	facebook	facebook	809
4	win!...	win	795
...	...	...	...
165	luck	luck	166
166	entering	enter	165
167	everything	everything	164
168	always	always	162
169	changes	change	162

170 rows × 3 columns

```
from sklearn.feature_extraction.text import CountVectorizer

def get_top_bigrams_with_original(df, sentiment_label=None, n=170):
    if sentiment_label:
        subset = df[df['Sentiment'] == sentiment_label]
    else:
        subset = df.copy()

    # Check if the subset is empty after filtering for sentiment
    if subset.empty:
        print(f"No data found for sentiment: {sentiment_label}")
        return pd.DataFrame(columns=["Original terms", "Terms after pre-process

    original_texts = subset['original_text'].dropna().tolist()
    cleaned_texts = subset['final_text'].dropna().tolist()

    vectorizer = CountVectorizer(ngram_range=(2, 2))

    # Check if cleaned_texts is empty before fitting
```

```

if not cleaned_texts or all(not text.strip() for text in cleaned_texts):
    print(f"All texts for sentiment '{sentiment_label}' are empty after pre-processed")
    return pd.DataFrame(columns=["Original terms", "Terms after pre-processed"])

X_cleaned = vectorizer.fit_transform(cleaned_texts)
sum_words = X_cleaned.sum(axis=0)
cleaned_2grams_freq = [(word, sum_words[0, idx]) for word, idx in vectorizer.get_feature_names_out()]
top_cleaned_2grams = sorted(cleaned_2grams_freq, key=lambda x: x[1], reverse=True)

records = []

for cleaned_bigram, freq in top_cleaned_2grams:
    cleaned_words = cleaned_bigram.split()

    matching_original = []
    for word in cleaned_words:
        for text in original_texts:
            words = text.lower().split()
            if word in words:
                matching_original.append(word)
            break
    else:
        matching_original.append(word)
    original_bigram = ' '.join(matching_original)
    records.append([original_bigram, cleaned_bigram, freq])

result_df = pd.DataFrame(records, columns=["Original terms", "Terms after pre-processed", "Frequency"])
return result_df

df_bigrams_all = get_top_bigrams_with_original(df)

df_bigrams_positive = get_top_bigrams_with_original(df, 'Positive')
df_bigrams_negative = get_top_bigrams_with_original(df, 'Negative')
df_bigrams_neutral = get_top_bigrams_with_original(df, 'Neutral')
df_bigrams_irrelevant = get_top_bigrams_with_original(df, 'Irrelevant')

print(" Top 150 2-grams in ALL Data")
display(df_bigrams_all)

print(" Top 150 2-grams in Positive Tweets")
display(df_bigrams_positive)

print(" Top 150 2-grams in Negative Tweets")
display(df_bigrams_negative)

```

```
print(" Top 150 2-grams in Neutral Tweets")
display(df_bigrams_neutral)
```

→ No data found for sentiment: Irrelevant  
Top 150 2-grams in ALL Data

	Original terms	Terms after pre-processing	2 grams Weight
0	red dead	red dead	1319
1	johnson johnson	johnson johnson	1214
2	dead redemption	dead redemption	1047
3	home depot	home depot	960
4	assassin creed	assassin creed	688
...	...	...	...
165	best thing	best thing	50
166	black flag	black flag	50
167	microsoft edge	microsoft edge	50
168	sell baby	sell baby	50
169	chinese apps	chinese apps	50

170 rows × 3 columns

Top 150 2-grams in Positive Tweets

	Original terms	Terms after pre-processing	2 grams Weight
0	red dead	red dead	580
1	assassin creed	assassin creed	522
2	dead redemption	dead redemption	488
3	home depot	home depot	325
4	gon na	gon na	313
...	...	...	...
165	spend money	spend money	18
166	king canyon	king canyon	18
167	creed origin	creed origin	18
168	na buy	na buy	18
169	fun ive	fun ive	18

170 rows × 3 columns

Top 150 2-grams in Negative Tweets

	Original terms	Terms after pre-processing	2 grams Weight
0	home depot	home depot	408
1	johnson johnson	johnson johnson	335
2	call duty	call duty	299
3	rhandlerr rhandlerr	rhandlerr rhandlerr	266
4	league legend	league legend	215
...	...	...	...
165	abandon sanction	abandon sanction	23
166	verizon fios	verizon fios	23
167	work hard	work hard	22
168	account hack	account hack	22
169	open world	open world	22

170 rows × 3 columns

Top 150 2-grams in Neutral Tweets

	Original terms	Terms after pre-processing	2 grams Weight
0	johnson johnson	johnson johnson	736
1	red dead	red dead	537
2	dead redemption	dead redemption	409
3	xbox series	xbox series	273
4	baby powder	baby powder	263
...	...	...	...
165	high hr	high hr	22
166	evax humbly	evax humbly	22
167	much love	much love	21
168	russian bot	russian bot	21
169	gta online	gta online	21

170 rows × 3 columns

```
from sklearn.feature_extraction.text import CountVectorizer
```

```

def get_top_bigrams_with_original(df, sentiment_label=None, n=170):
    if sentiment_label:
        subset = df[df['Sentiment'] == sentiment_label]
    else:
        subset = df.copy()

    # Check if the subset is empty after filtering for sentiment
    if subset.empty:
        print(f"No data found for sentiment: {sentiment_label}")
        return pd.DataFrame(columns=["Original terms", "Terms after pre-process"])

    original_texts = subset['original_text'].dropna().tolist()
    cleaned_texts = subset['final_text'].dropna().tolist()

    vectorizer = CountVectorizer(ngram_range=(2, 2))
    X_cleaned = vectorizer.fit_transform(cleaned_texts)
    sum_words = X_cleaned.sum(axis=0)
    cleaned_2grams_freq = [(word, sum_words[0, idx]) for word, idx in vectorizer.get_feature_names_out()]
    top_cleaned_2grams = sorted(cleaned_2grams_freq, key=lambda x: x[1], reverse=True)

    records = []

    for cleaned_bigram, freq in top_cleaned_2grams:
        cleaned_words = cleaned_bigram.split()

        matching_original = []
        for word in cleaned_words:
            for text in original_texts:
                words = text.lower().split()
                if word in words:
                    matching_original.append(word)
                break
            else:
                matching_original.append(word)
        original_bigram = ' '.join(matching_original)
        records.append([original_bigram, cleaned_bigram, freq])

    result_df = pd.DataFrame(records, columns=["Original terms", "Terms after pre-process"])
    return result_df

df_bigrams_all = get_top_bigrams_with_original(df)

```




```
df_bigrams_positive = get_top_bigrams_with_original(df, 'Positive')
df_bigrams_negative = get_top_bigrams_with_original(df, 'Negative')
df_bigrams_neutral = get_top_bigrams_with_original(df, 'Neutral')
df_bigrams_irrelevant = get_top_bigrams_with_original(df, 'Irrelevant') # This

print(" Top 150 2-grams in ALL Data")
display(df_bigrams_all)

print(" Top 150 2-grams in Positive Tweets")
display(df_bigrams_positive)

print(" Top 150 2-grams in Negative Tweets")
display(df_bigrams_negative)

print(" Top 150 2-grams in Neutral Tweets")
display(df_bigrams_neutral)
```

 No data found for sentiment: Irrelevant  
Top 150 2-grams in ALL Data

	Original terms	Terms after pre-processing	2 grams Weight
0	red dead	red dead	1319
1	johnson johnson	johnson johnson	1214
2	dead redemption	dead redemption	1047
3	home depot	home depot	960
4	assassin creed	assassin creed	688
...	...	...	...
165	best thing	best thing	50
166	black flag	black flag	50
167	microsoft edge	microsoft edge	50
168	sell baby	sell baby	50
169	chinese apps	chinese apps	50

170 rows × 3 columns

Top 150 2-grams in Positive Tweets

	Original terms	Terms after pre-processing	2 grams Weight
0	red dead	red dead	580
1	assassin creed	assassin creed	522
2	dead redemption	dead redemption	488

3	home depot	home depot	325
4	gon na	gon na	313
...	...	...	...
165	spend money	spend money	18
166	king canyon	king canyon	18
167	creed origin	creed origin	18
168	na buy	na buy	18
169	fun ive	fun ive	18

170 rows × 3 columns

Top 150 2-grams in Negative Tweets

	Original terms	Terms after pre-processing	2 grams Weight
0	home depot	home depot	408
1	johnson johnson	johnson johnson	335
2	call duty	call duty	299
3	rhandlerr rhandlerr	rhandlerr rhandlerr	266
4	league legend	league legend	215
...	...	...	...
165	abandon sanction	abandon sanction	23
166	verizon fios	verizon fios	23
167	work hard	work hard	22
168	account hack	account hack	22
169	open world	open world	22

170 rows × 3 columns

Top 150 2-grams in Neutral Tweets

	Original terms	Terms after pre-processing	2 grams Weight
0	johnson johnson	johnson johnson	736
1	red dead	red dead	537
2	dead redemption	dead redemption	409
3	xbox series	xbox series	273
4	baby powder	baby powder	263

...	...	...	...
<b>165</b>	high hr	high hr	22
<b>166</b>	evax humbly	evax humbly	22
<b>167</b>	much love	much love	21
<b>168</b>	russian bot	russian bot	21
<b>169</b>	gta online	gta online	21

170 rows × 3 columns

```
import os

fe_folder = '/content/drive/MyDrive/FE11111'
os.makedirs(fe_folder, exist_ok=True)

def save_table_to_drive(df, filename):
    filepath = os.path.join(fe_folder, filename)
    df.to_csv(filepath, index=False)
    print(f"Saved: {filepath}")

save_table_to_drive(df_top150_all, 'top150_unigrams_all.csv')

save_table_to_drive(df_top150_positive, 'top150_unigrams_positive.csv')
save_table_to_drive(df_top150_negative, 'top150_unigrams_negative.csv')
save_table_to_drive(df_top150_neutral, 'top150_unigrams_neutral.csv')
save_table_to_drive(df_top150_irrelevant, 'top150_unigrams_irrelevant.csv')

save_table_to_drive(df_bigrams_all, 'top150_bigrams_all.csv')

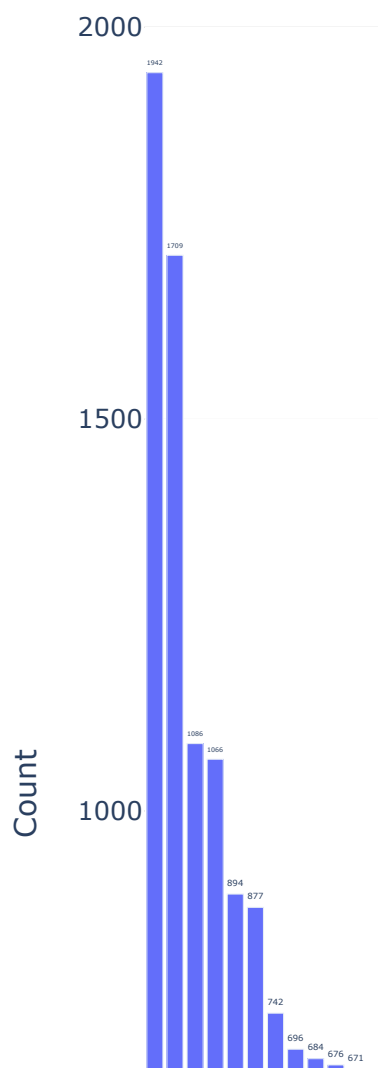
save_table_to_drive(df_bigrams_positive, 'top150_bigrams_positive.csv')
save_table_to_drive(df_bigrams_negative, 'top150_bigrams_negative.csv')
save_table_to_drive(df_bigrams_neutral, 'top150_bigrams_neutral.csv')
```

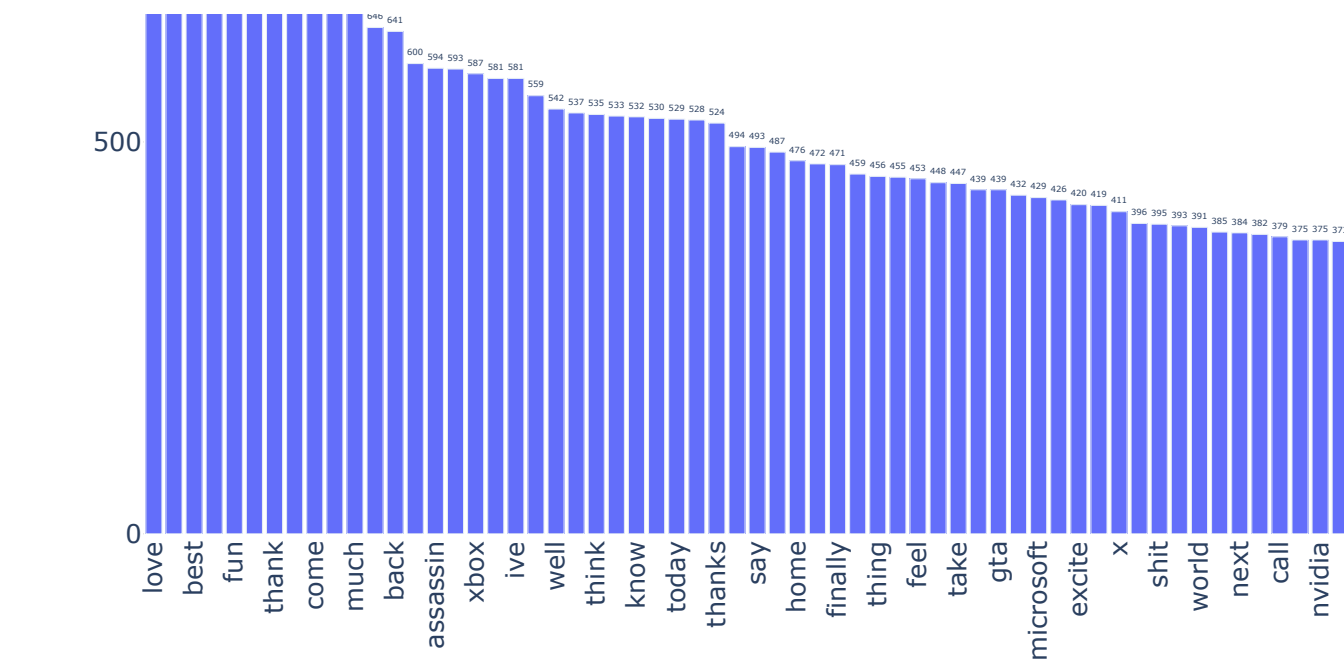
 Saved: /content/drive/MyDrive/FE11111/top150\_unigrams\_all.csv  
 Saved: /content/drive/MyDrive/FE11111/top150\_unigrams\_positive.csv  
 Saved: /content/drive/MyDrive/FE11111/top150\_unigrams\_negative.csv  
 Saved: /content/drive/MyDrive/FE11111/top150\_unigrams\_neutral.csv  
 Saved: /content/drive/MyDrive/FE11111/top150\_unigrams\_irrelevant.csv  
 Saved: /content/drive/MyDrive/FE11111/top150\_bigrams\_all.csv  
 Saved: /content/drive/MyDrive/FE11111/top150\_bigrams\_positive.csv  
 Saved: /content/drive/MyDrive/FE11111/top150\_bigrams\_negative.csv  
 Saved: /content/drive/MyDrive/FE11111/top150\_bigrams\_neutral.csv

```
import plotly.express as px

def plot_top_words_full(df, title):
    fig = px.bar(
        df,
        x='Terms after pre-processing',
        y='TF',
        title=title,
        text='TF',
```

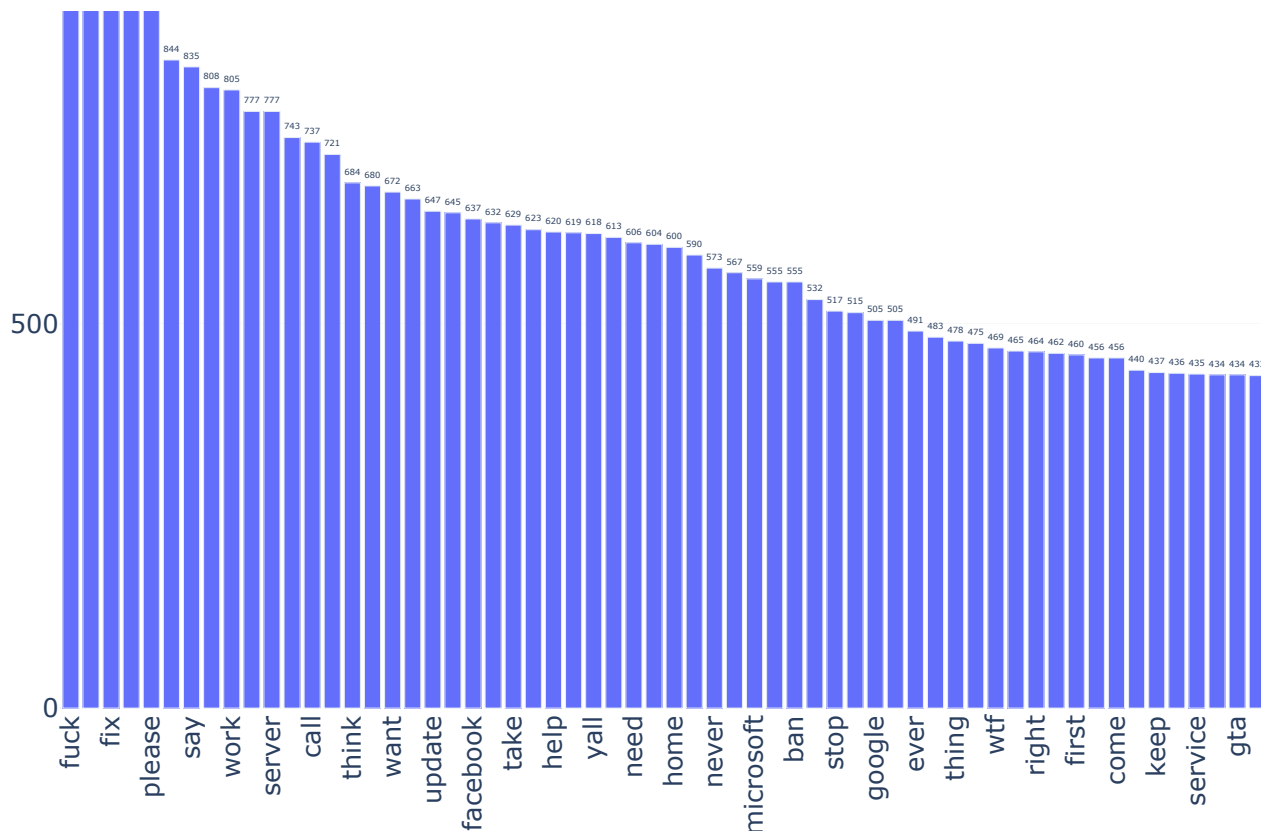
```
plot_top_words_full(df_top150_positive, "Top 150 Words - Positive Tweets") # Ch
plot_top_words_full(df_top150_negative, "Top 150 Words - Negative Tweets") # Ch
plot_top_words_full(df_top150_neutral, "Top 150 Words - Neutral Tweets") # Char
```



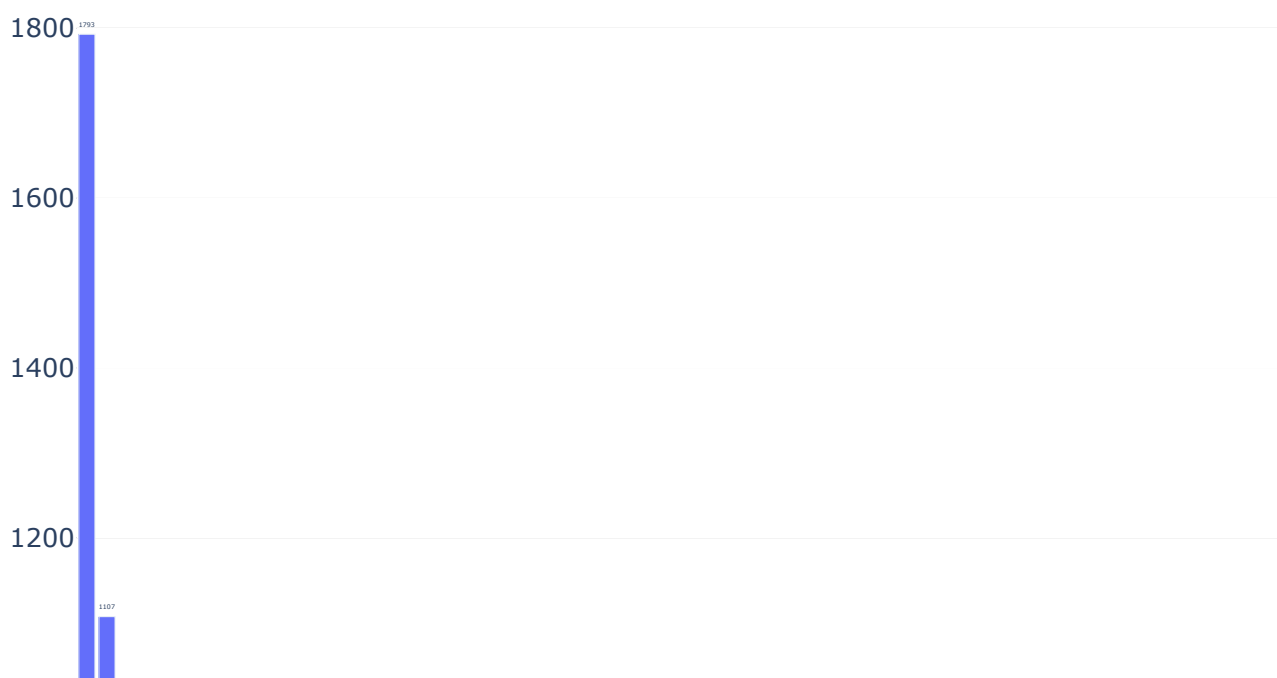


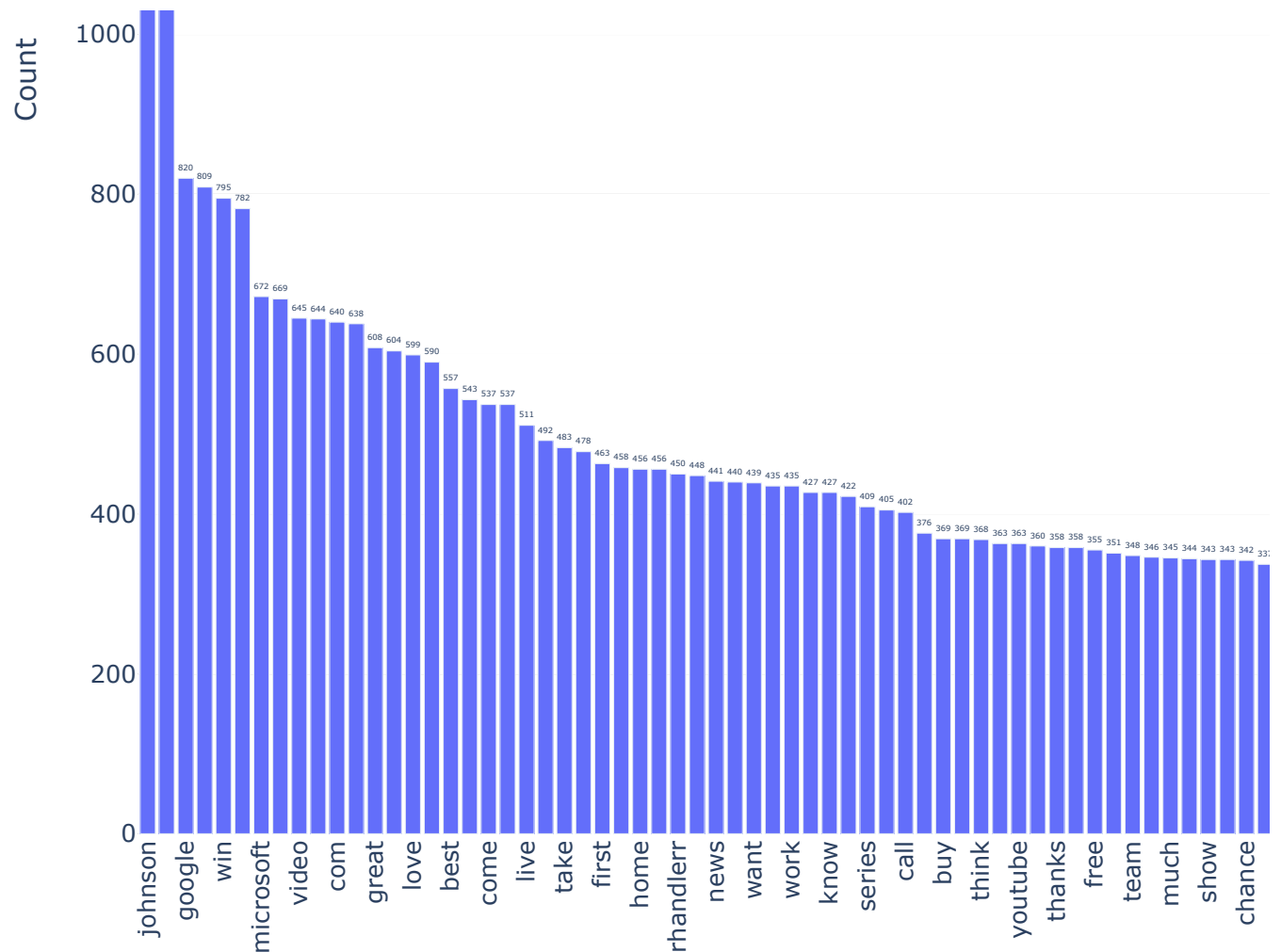
Top





To





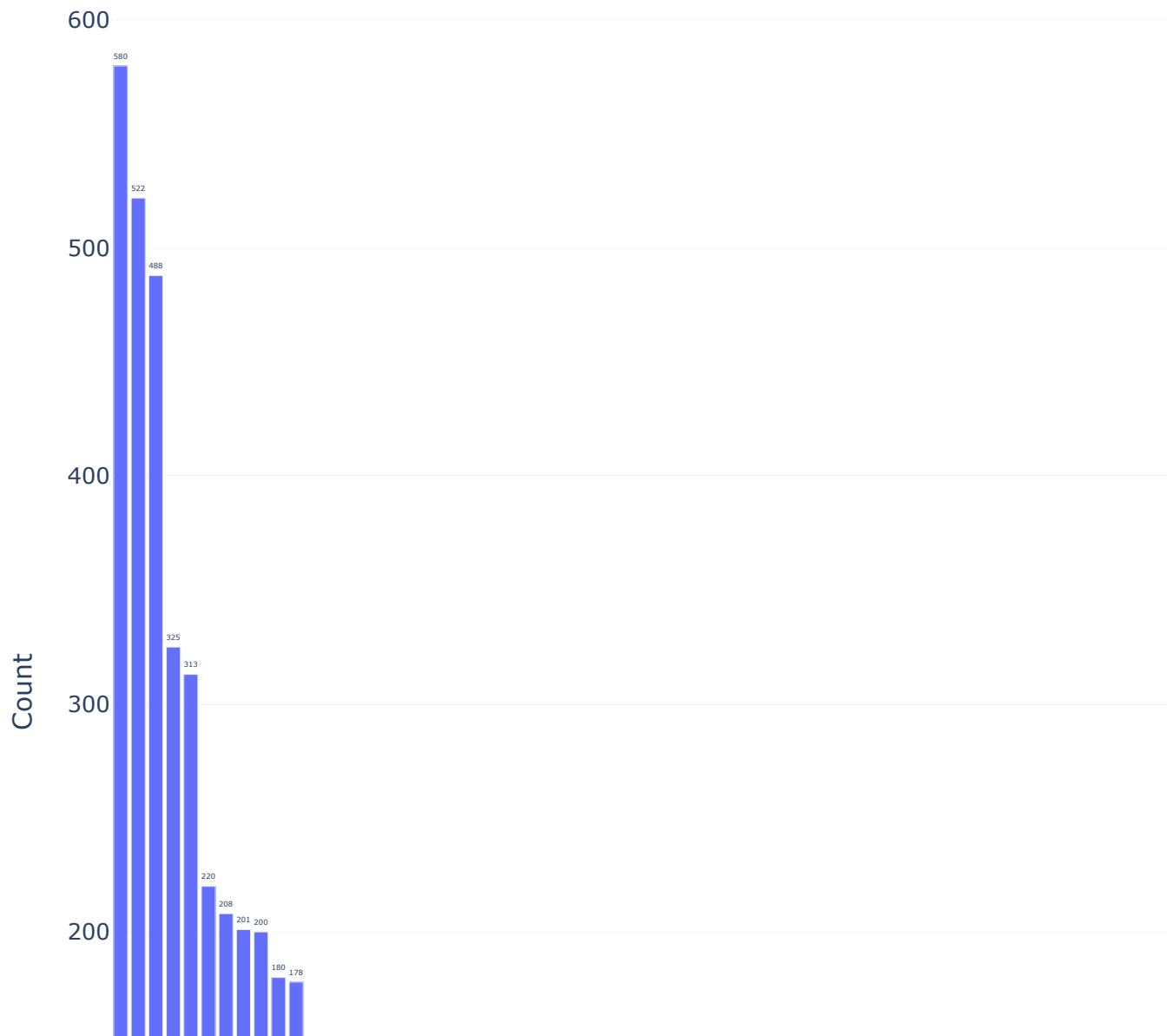
```
import plotly.express as px

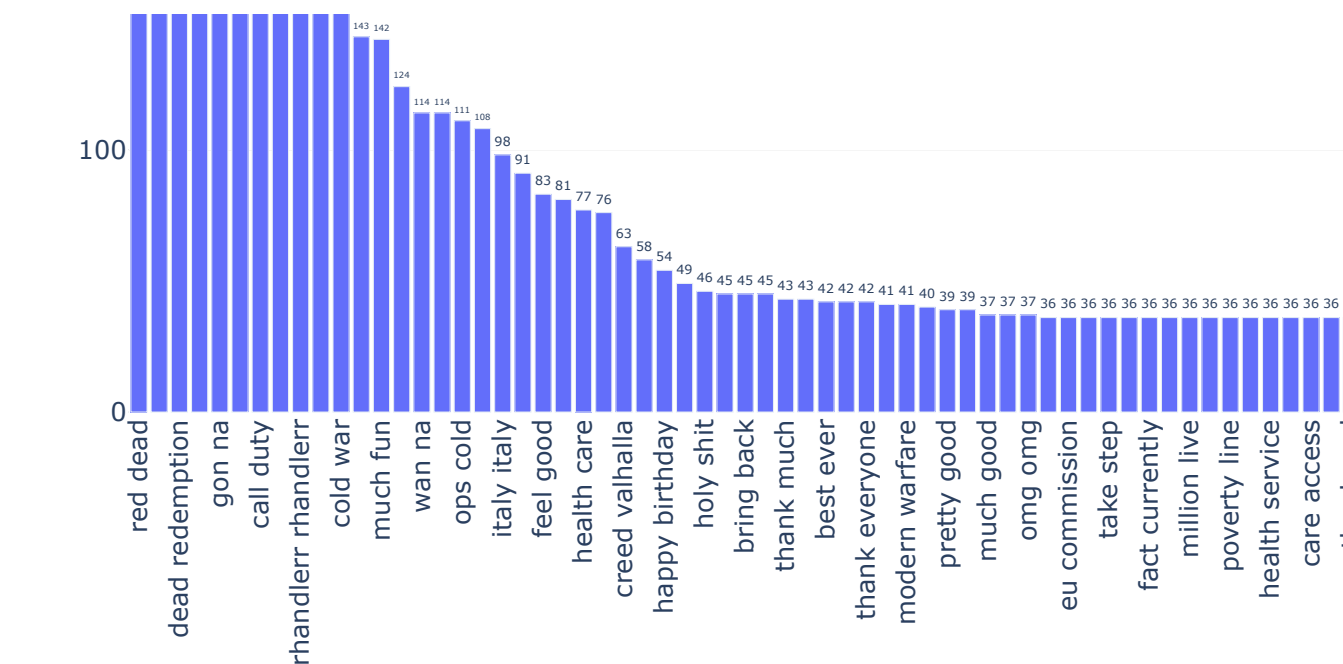
def plot_top_bigrams_full(df, title):
    fig = px.bar(
        df,
        x='Terms after pre-processing',
        y='2 grams Weight',
        title=title,
        text='2 grams Weight',
        labels={'Terms after pre-processing': '2-gram', '2 grams Weight': 'Cour
    )
    fig.update_traces(textposition='outside')
    fig.update_layout(
```



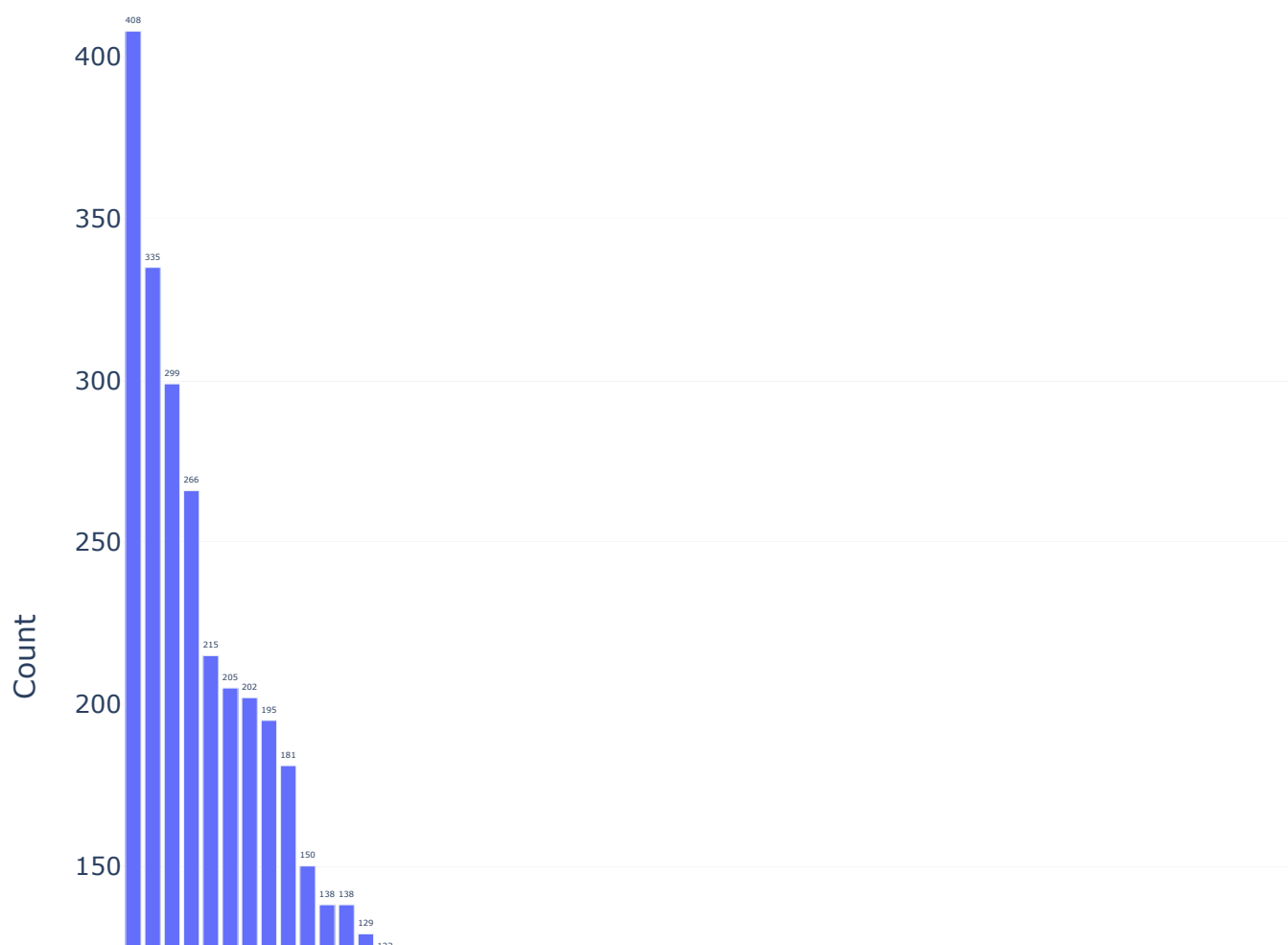
```
axis_tickangle=-90,  
plot_bgcolor='white',  
title_x=0.5,  
font=dict(size=12),  
height=1000,  
width=1500,  
margin=dict(l=20, r=20, t=50, b=200)  
)  
fig.show()
```

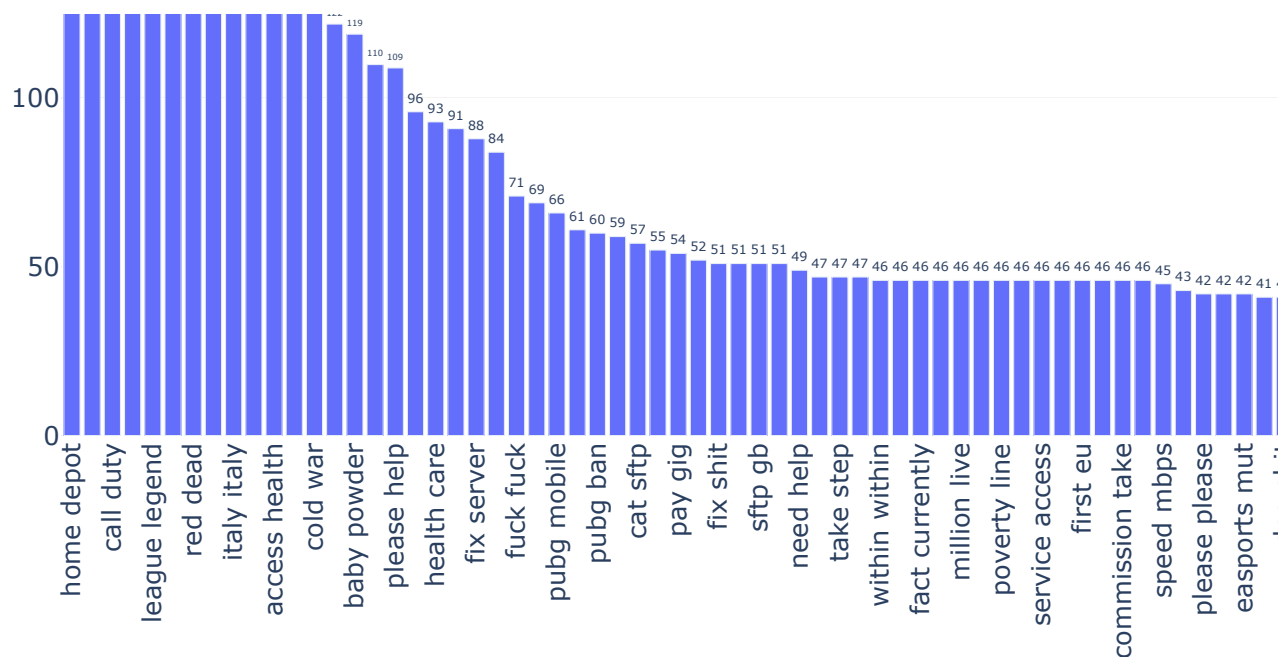
```
plot_top_bigrams_full(df_bigrams_positive, "Top 150 2-grams - Positive Tweets")  
plot_top_bigrams_full(df_bigrams_negative, "Top 150 2-grams - Negative Tweets")  
plot_top_bigrams_full(df_bigrams_neutral, "Top 150 2-grams - Neutral Tweets")
```

[Top](#)

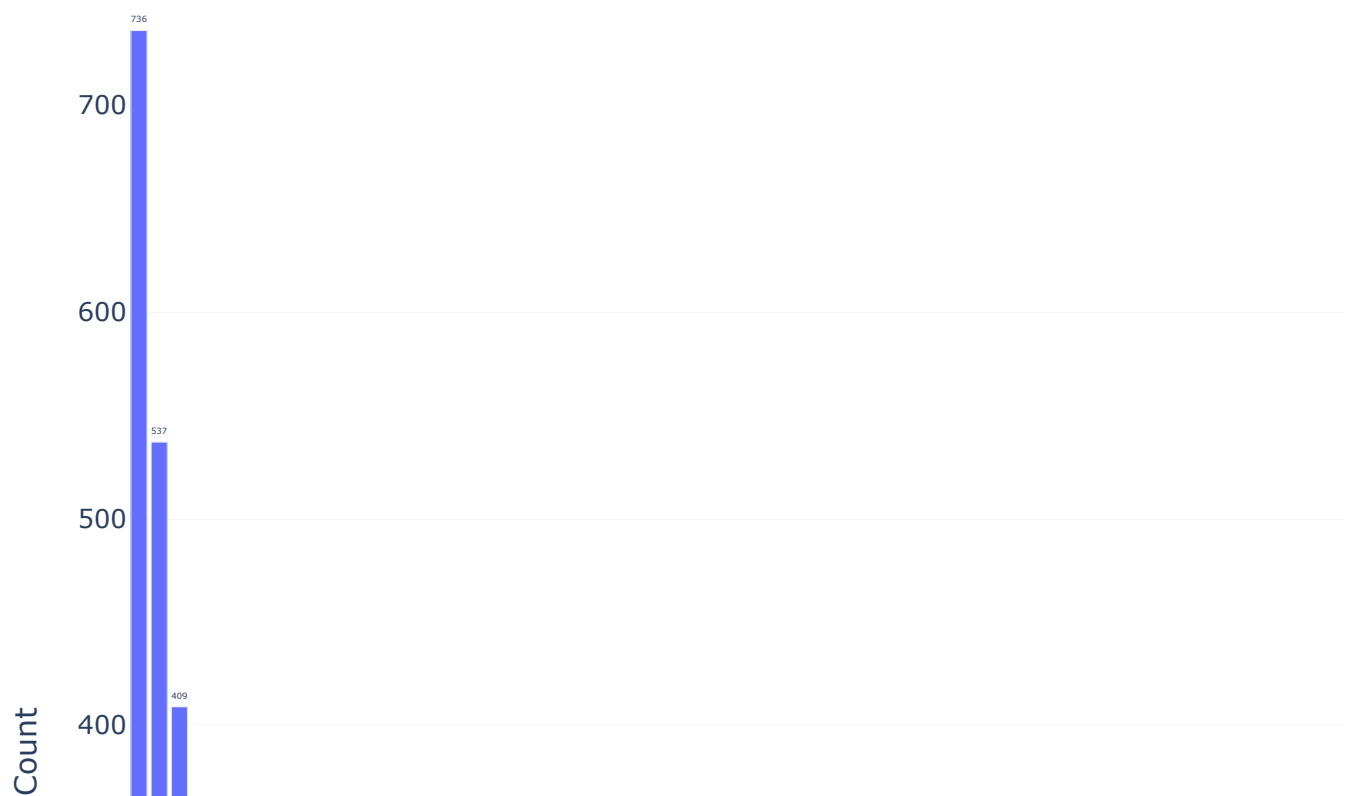


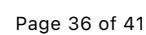
Top





Top





```
from sklearn.model_selection import train_test_split

X = df['final_text']
y = df['Sentiment']

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y
)

# Display shapes
print(f"Training data shape: {X_train.shape}")
print(f"Testing data shape: {X_test.shape}")
```

↗ Training data shape: (47294,)
Testing data shape: (11824,)

```
# Import TF-IDF Vectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

# Initialize the TF-IDF Vectorizer
tfidf = TfidfVectorizer(
    max_features=5000,
    ngram_range=(1, 2),
    stop_words='english'
)

# Fit the vectorizer on the training data and transform both train and test set
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)

# Display the shapes
print(f"TF-IDF train shape: {X_train_tfidf.shape}")
print(f"TF-IDF test shape: {X_test_tfidf.shape}")
```

↗ TF-IDF train shape: (47294, 5000)
TF-IDF test shape: (11824, 5000)

```
# Import required libraries
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Initialize and train Naive Bayes model
nb_model = MultinomialNB()
nb_model.fit(X_train_tfidf, y_train)

# Predict on test data
y_pred_nb = nb_model.predict(X_test_tfidf)

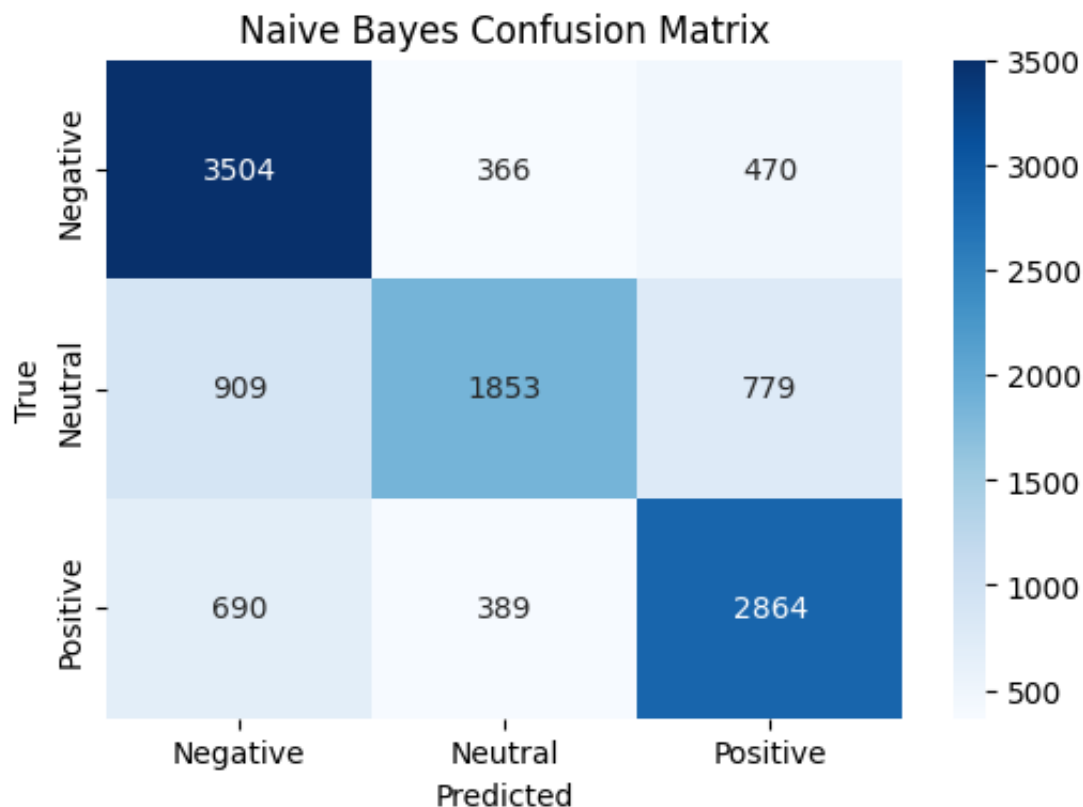
# Evaluate Naive Bayes model
print("Naive Bayes Model Evaluation:")
print("Accuracy:", accuracy_score(y_test, y_pred_nb))
print("\nClassification Report:\n", classification_report(y_test, y_pred_nb))

# Plot confusion matrix
cm_nb = confusion_matrix(y_test, y_pred_nb)
plt.figure(figsize=(6,4))
sns.heatmap(cm_nb, annot=True, fmt='d', cmap='Blues', xticklabels=nb_model.class_names)
plt.title('Naive Bayes Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

Naive Bayes Model Evaluation:  
Accuracy: 0.6952807848443843

Classification Report:

	precision	recall	f1-score	support
Negative	0.69	0.81	0.74	4340
Neutral	0.71	0.52	0.60	3541
Positive	0.70	0.73	0.71	3943
accuracy			0.70	11824
macro avg	0.70	0.69	0.69	11824
weighted avg	0.70	0.70	0.69	11824



```
# Initialize and train Logistic Regression model
lr_model = LogisticRegression(max_iter=1000)
lr_model.fit(X_train_tfidf, y_train)

# Predict on test data
y_pred_lr = lr_model.predict(X_test_tfidf)

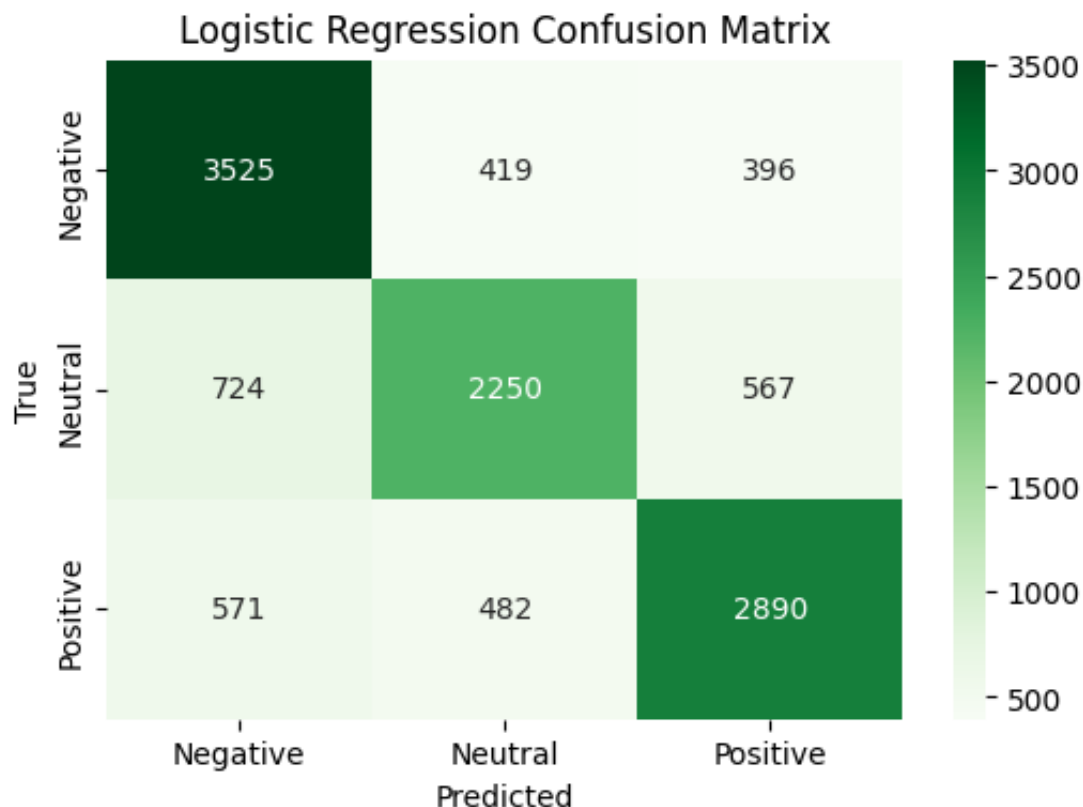
# Evaluate Logistic Regression model
print("Logistic Regression Model Evaluation:")
print("Accuracy:", accuracy_score(y_test, y_pred_lr))
print("\nClassification Report:\n", classification_report(y_test, y_pred_lr))
```

```
# Plot confusion matrix
cm_lr = confusion_matrix(y_test, y_pred_lr)
plt.figure(figsize=(6,4))
sns.heatmap(cm_lr, annot=True, fmt='d', cmap='Greens', xticklabels=lr_model.class_labels, yticklabels=lr_model.class_labels)
plt.title('Logistic Regression Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

Logistic Regression Model Evaluation:  
Accuracy: 0.7328315290933695

Classification Report:

	precision	recall	f1-score	support
Negative	0.73	0.81	0.77	4340
Neutral	0.71	0.64	0.67	3541
Positive	0.75	0.73	0.74	3943
accuracy			0.73	11824
macro avg	0.73	0.73	0.73	11824
weighted avg	0.73	0.73	0.73	11824



Start coding or [generate](#) with AI.



