```
pip install nltk
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-
packages (3.8.1)
Requirement already satisfied: click in
/usr/local/lib/python3.10/dist-packages (from nltk) (8.1.7)
Requirement already satisfied: joblib in
/usr/local/lib/python3.10/dist-packages (from nltk) (1.3.2)
Requirement already satisfied: regex>=2021.8.3 in
/usr/local/lib/python3.10/dist-packages (from nltk) (2023.6.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-
packages (from nltk) (4.66.1)
```

```
pip install wordcloud matplotlib
```

```
Requirement already satisfied: wordcloud in
/usr/local/lib/python3.10/dist-packages (1.9.2)
Requirement already satisfied: matplotlib in
/usr/local/lib/python3.10/dist-packages (3.7.1)
Requirement already satisfied: numpy>=1.6.1 in
/usr/local/lib/python3.10/dist-packages (from wordcloud) (1.23.5)
Requirement already satisfied: pillow in
/usr/local/lib/python3.10/dist-packages (from wordcloud) (9.4.0)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (1.1.1)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (4.43.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.5)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (23.2)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7-
>matplotlib) (1.16.0)
```

```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.stem import LancasterStemmer
from nltk.stem import PorterStemmer
import string
from wordcloud import WordCloud, STOPWORDS
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer,
TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB #for multinomialNB
from sklearn.metrics import accuracy_score, classification_report #for
evaluation report
from collections import Counter
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
```

```
#after the review of text preprocessing method in "2. preprocessing
methods.ipybb", the following function was created for text
preprocessing.
#Datasets used in this project are cleaned using this function.

def preprocess_text(df, column_name):
    # Lowercasing
    df[column_name] = df[column_name].apply(lambda tokens:
[token.lower() for token in tokens])

    # Stop Word Removal
    stop_words = set(stopwords.words('english'))
    df[column_name] = df[column_name].apply(lambda tokens: [word for
word in tokens if word not in stop_words])

    # Removing one-letter words
    df[column_name] = df[column_name].apply(lambda tokens: [word for
word in tokens if len(word) > 1])

    # Remove special symbols and punctuation
    df[column_name] = df[column_name].apply(lambda tokens: [word for
word in tokens if word.isalpha()])

    # Lemmatizing
    lemmatizer = WordNetLemmatizer()
    df[column_name] = df[column_name].apply(lambda tokens:
[lemmatizer.lemmatize(word) for word in tokens])
```

```
#creating function to count word a colomun

def word_count (df,colomun_name):
  df['word_count'] = df [colomun_name].apply(len)
  average_word_count = df['word_count'].mean()
  max_word_count = df['word_count'].max()
  minimum_word_count = df['word_count'].min()
  print(f"Average Word Count :{average_word_count}")
  print(f"Maximum Word Count :{max_word_count}")
  print(f"Minimum Word Count :{minimum_word_count}")

#use "fetch_20newsgroups" function from sklean.datasets  to load 20
newsgroups dataset
# removing "headers", "footers" and "quotes" is recommended because it
is more realistic
(https://scikit-learn.org/0.19/datasets/twenty_newsgroups.html)
# loading dataset with or without "headers", "footers" and "quotes"
and review each datasets.

remove = ("headers", "footers", "quotes")
news20group_train = fetch_20newsgroups(subset='train', remove =
remove)
news20group_test = fetch_20newsgroups (subset='test', remove= remove)

#print list of 20 news groups
categories = news20group_train.target_names
categories
```

```
['alt.atheism',
 'comp.graphics',
 'comp.os.ms-windows.misc',
 'comp.sys.ibm.pc.hardware',
 'comp.sys.mac.hardware',
 'comp.windows.x',
 'misc.forsale',
 'rec.autos',
 'rec.motorcycles',
 'rec.sport.baseball',
 'rec.sport.hockey',
 'sci.crypt',
 'sci.electronics',
 'sci.med',
 'sci.space',
 'soc.religion.christian',
 'talk.politics.guns',
 'talk.politics.mideast',
 'talk.politics.misc',
 'talk.religion.misc']
```

```
type(categories)
```

```
list
```

```
#number of observation in train data
len(news20group_train.data)
```

```
11314
```

```
#number of observation in test data
len(news20group_test.data)
```

```
7532
```

```
#count observation in each category (Train Data)

cat,frequency_train = np.unique(news20group_train.target,
return_counts = True)
cat,frequency_train
```

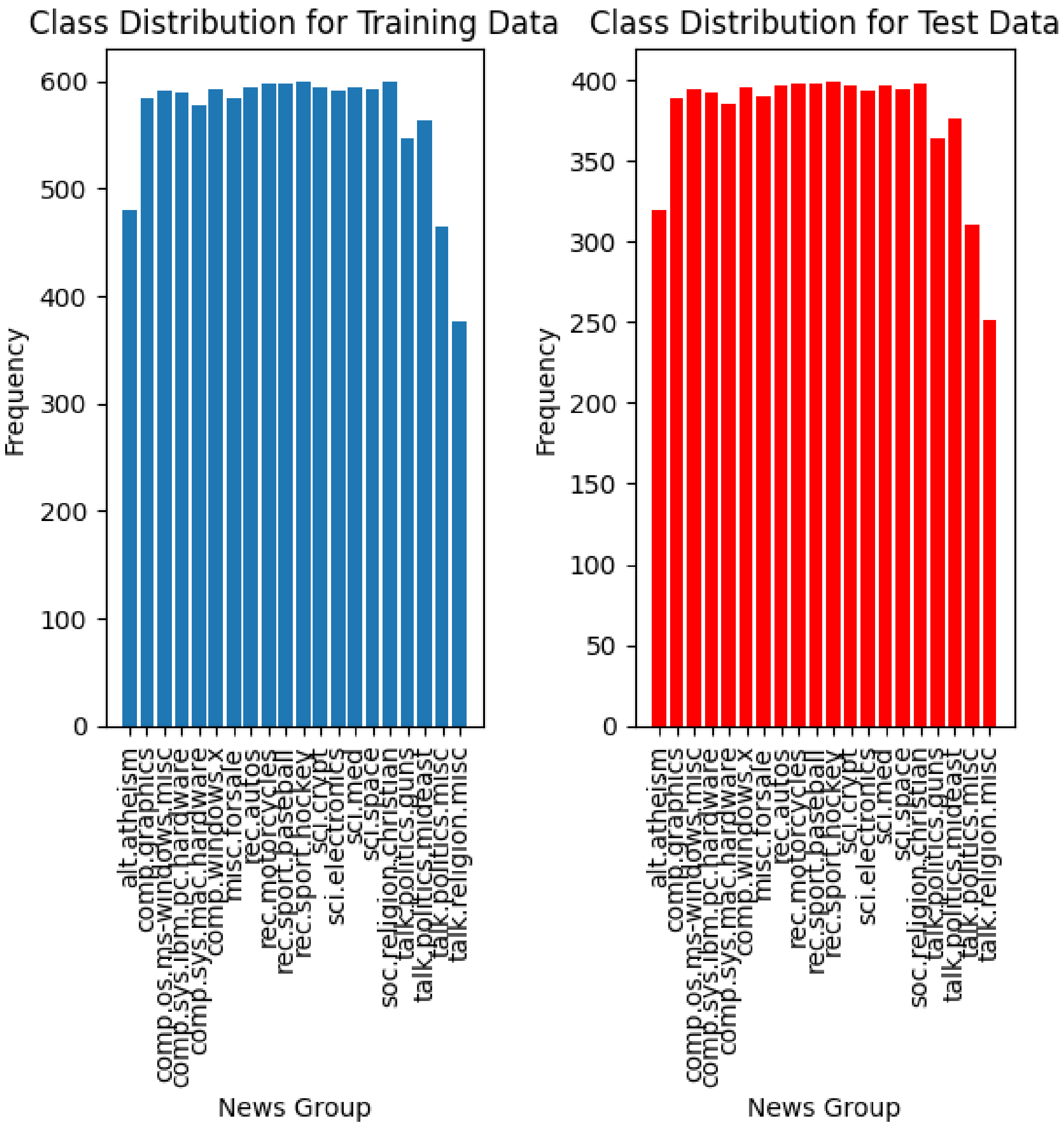
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14,
15, 16,
        17, 18, 19]),
 array([480, 584, 591, 590, 578, 593, 585, 594, 598, 597, 600, 595,
591,
        594, 593, 599, 546, 564, 465, 377]))
```

```
#count observation in each category (Test Data)

cat,frequency_test = np.unique(news20group_test.target, return_counts
= True)
cat,frequency_test
```

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14,
15, 16,
        17, 18, 19]),
 array([319, 389, 394, 392, 385, 395, 390, 396, 398, 397, 399, 396,
393,
        396, 394, 398, 364, 376, 310, 251]))
```

```
cat =  np.array (news20group_test.target_names)

#create bar plots for both training data and test data to compare the
distribution

#subplot 1 for training data distribution
plt.subplot(1,2,1) #1 row, 2 columns, position 1
plt.bar(cat, frequency_train)
plt.xticks(rotation=90)
plt.title('Class Distribution for Training Data')
plt.xlabel('News Group')
plt.ylabel('Frequency')

#subplot 2 for test data distribution
```

```
plt.subplot(1,2,2) #1 row, 2 columns, position 2
plt.bar(cat, frequency_test, color = 'red')
plt.xticks(rotation=90)
plt.title('Class Distribution for Test Data')
plt.xlabel('News Group')
plt.ylabel('Frequency')

plt.subplots_adjust(wspace=0.4) #increase horisontal space
plt.show()
```



```
#Convert Bunch format to dataframe
train_df = pd.DataFrame({'data': news20group_train.data, 'target':
```

```
news20group_train.target})
test_df = pd.DataFrame({'data': news20group_test.data, 'target':
news20group_test.target})

train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11314 entries, 0 to 11313
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   data    11314 non-null  object
 1   target  11314 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 176.9+ KB
```

```
#treat target valuable as category
train_df["target"]=train_df["target"].astype("category")
test_df["target"]=test_df["target"].astype("category")

train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11314 entries, 0 to 11313
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype   
---  ------  --------------  -----   
 0   data    11314 non-null  object  
 1   target  11314 non-null  category
dtypes: category(1), object(1)
memory usage: 100.3+ KB
```

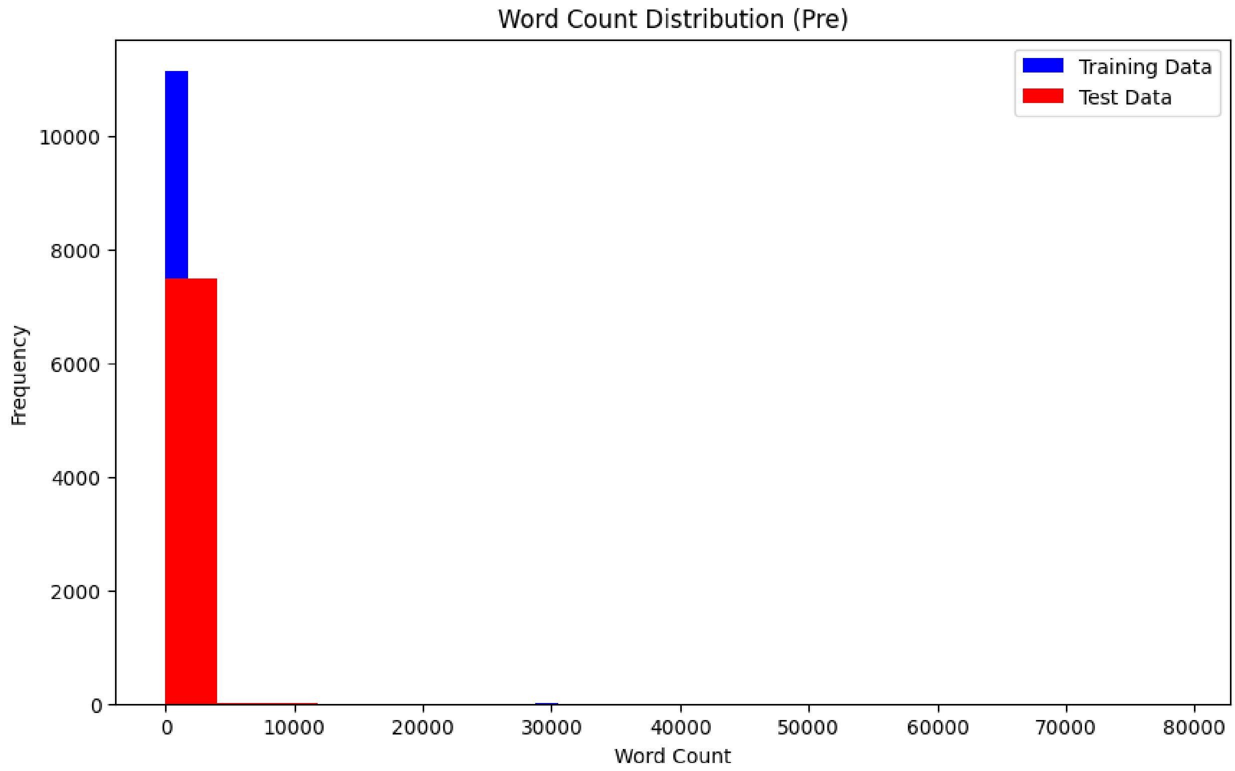
```
test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7532 entries, 0 to 7531
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype   
---  ------  --------------  -----   
 0   data    7532 non-null   object  
 1   target  7532 non-null   category
dtypes: category(1), object(1)
memory usage: 67.0+ KB
```

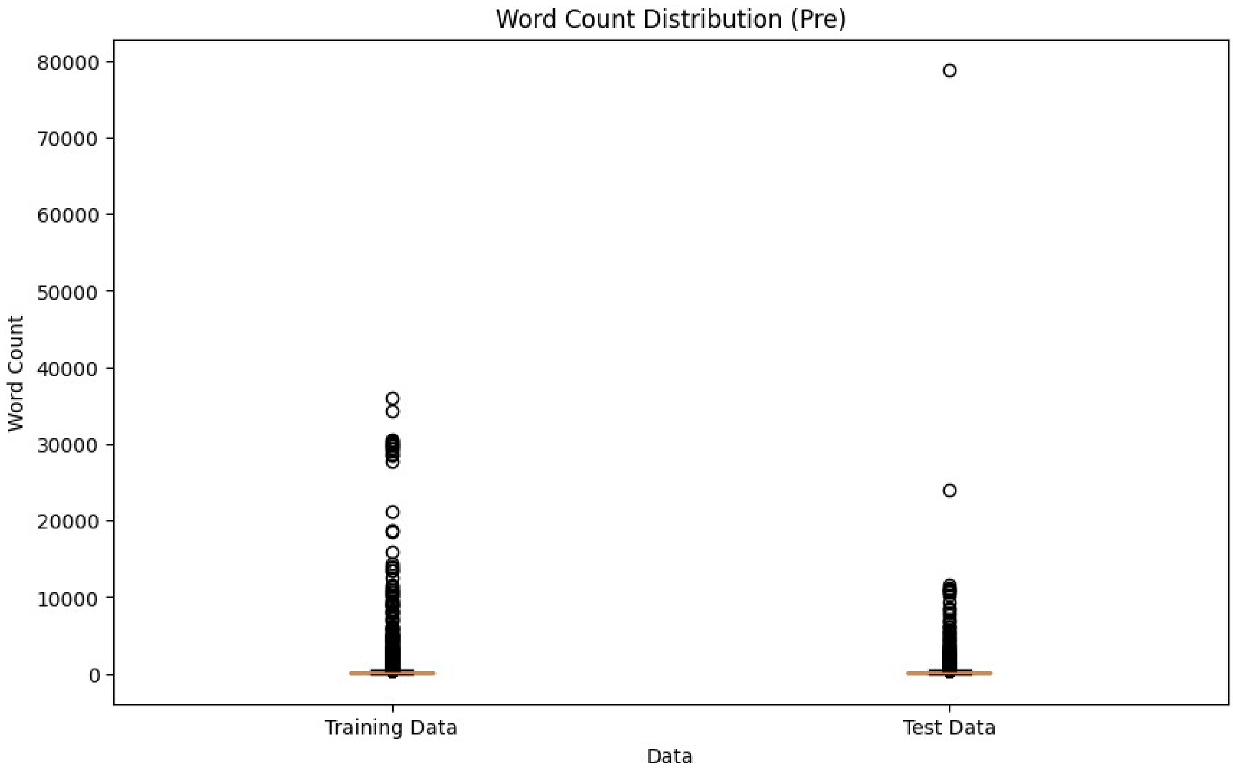
```
train_df.head()
```

```
                                                data target
0  I was wondering if anyone out there could enli...      7
1  A fair number of brave souls who upgraded thei...      4
2  well folks, my mac plus finally gave up the gh...      4
```

```
3  \nDo you have Weitek's address/phone number?  ...      1
4  From article <C5owCB.n3p@world.std.com>, by to...     14

#Tokenization
train_df ['data'] = train_df ['data'] .apply(word_tokenize)
test_df['data']= test_df ['data']. apply(word_tokenize)

#Word count for each document is added in the data.
print('[Train]')
word_count(train_df,'data')
print("[Test]")
word_count(test_df,'data')

[Train]
Average Word Count :270.4452890224501
Maximum Word Count :35955
Minimum Word Count :0
[Test]
Average Word Count :226.5367764206054
Maximum Word Count :78849
Minimum Word Count :0

#.info()shows there are no observation with empty data attribute,
however the minimum word count shows zero.
#It indicates that there are empty or whitespace strings in the
"data"column that are not very useful for the analysis.

#word count distribution
# Create a histogram of word counts
plt.figure(figsize=(10, 6))  # Optional: set the figure size
plt.hist(train_df['word_count'], bins=20, color='blue',
label='Training Data')
plt.hist(test_df['word_count'], bins=20, color='red', label='Test
Data')

# Set labels for x and y axes
plt.xlabel("Word Count")
plt.ylabel("Frequency")

# Set the title of the histogram
plt.title("Word Count Distribution (Pre)")

# Add a legend
plt.legend(loc='upper right')

# Show the histogram
plt.show()
```



```
# Create a box plot of word counts
plt.figure(figsize=(10, 6))  # Optional: set the figure size
plt.boxplot([train_df['word_count'], test_df['word_count']],
labels=['Training Data', 'Test Data'])

# Set labels for x and y axes
plt.xlabel("Data")
plt.ylabel("Word Count")

# Set the title of the box plot
plt.title("Word Count Distribution (Pre)")

# Show the box plot
plt.show()
```



```
#use previously created function "preprocess_text" to perform multiple
cleaning technique
preprocess_text(train_df, 'data')
preprocess_text(test_df,'data')

# Remove rows with empty or whitespace strings in the "data" column
train_df = train_df[train_df['data'].apply(len) > 0]
test_df = test_df[test_df['data'].apply(len) > 0]

train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10993 entries, 0 to 11313
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   data        10993 non-null  object
 1   target      10993 non-null  category
 2   word_count  10993 non-null  int64
dtypes: category(1), int64(1), object(1)
memory usage: 269.1+ KB
```

```
test_df.info()
```

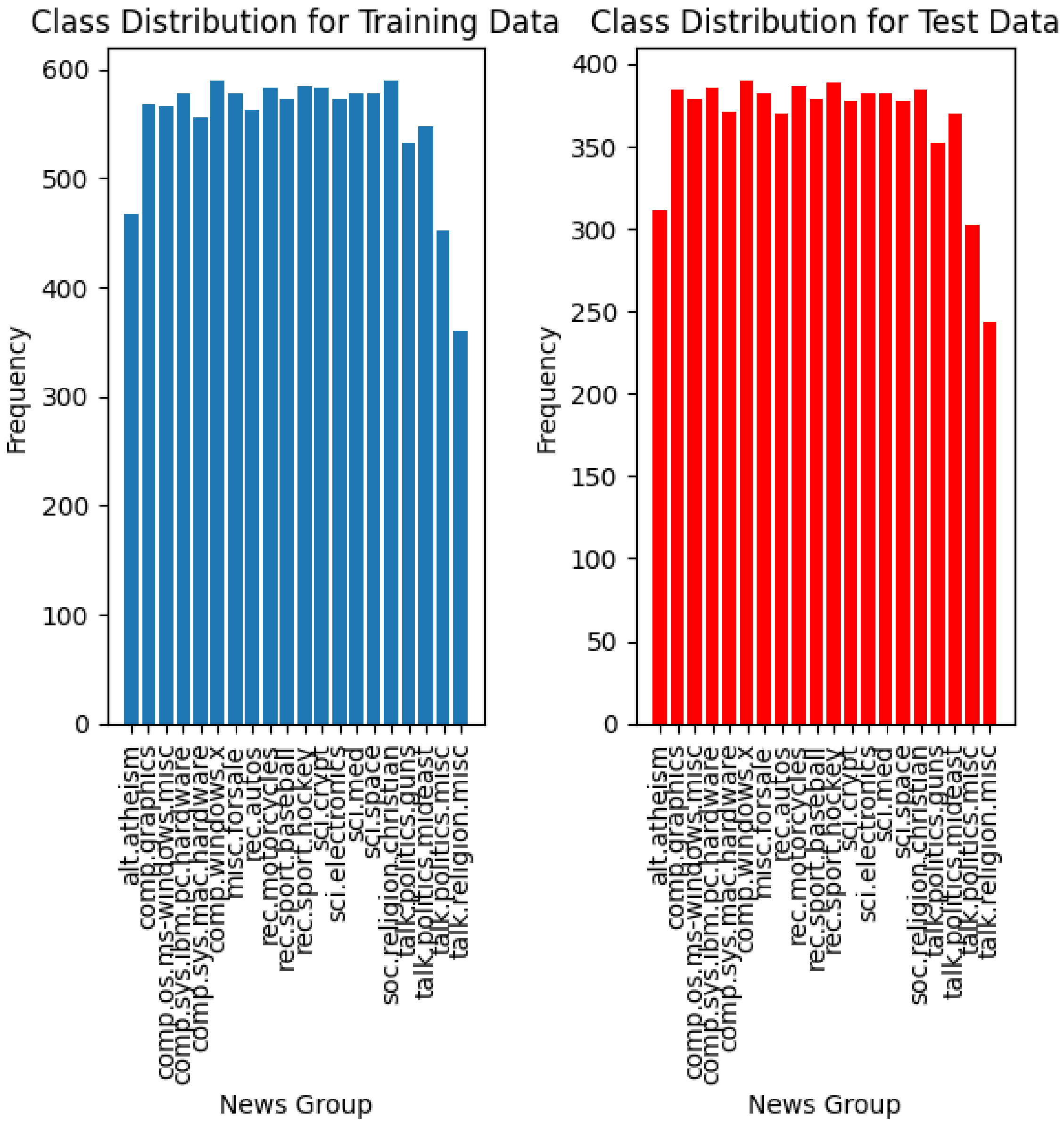
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7296 entries, 0 to 7531
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype   
---  ------      --------------  -----   
 0   data        7296 non-null   object  
 1   target      7296 non-null   category
 2   word_count  7296 non-null   int64   
dtypes: category(1), int64(1), object(1)
memory usage: 178.8+ KB
```

```
print('[Train]')
word_count(train_df,'data')
print("[Test]")
word_count(test_df,'data')
```

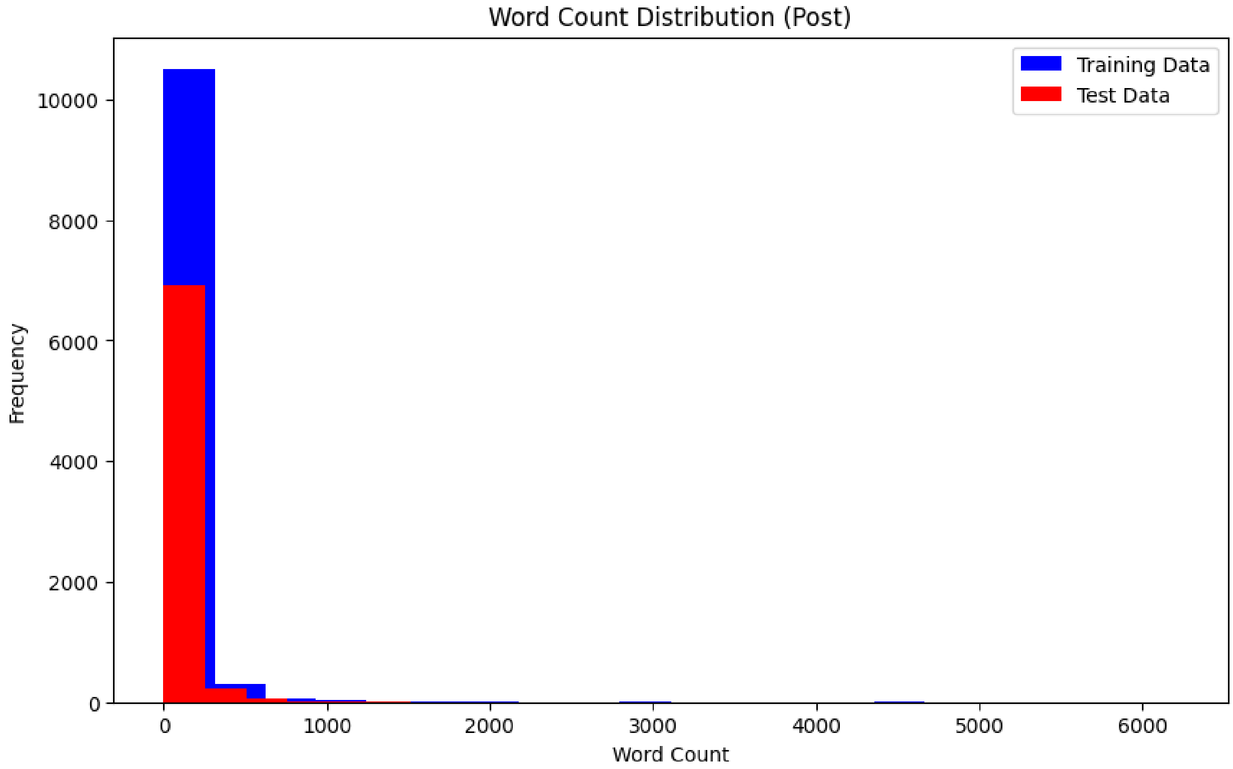
```
[Train]
Average Word Count :93.85927408350769
Maximum Word Count :6216
Minimum Word Count :1
[Test]
Average Word Count :88.28673245614036
Maximum Word Count :5058
Minimum Word Count :1
```

```
<ipython-input-5-5244b93f473d>:4: SettingWithCopyWarning: 
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: 
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  df['word_count'] = df [colomun_name].apply(len)
```

```
#create bar plots for both training data and test data to compare the 
distribution
frequency_train = train_df['target'].value_counts().sort_index()
frequency_test = test_df['target'].value_counts().sort_index()

category_label = [categories [i] for i in frequency_train.index]

#subplot 1 for training data distribution
plt.subplot(1,2,1) #1 row, 2 columns, position 1
plt.bar(category_label, frequency_train.values)
plt.xticks(rotation=90)
plt.title('Class Distribution for Training Data')
plt.xlabel('News Group')
plt.ylabel('Frequency')

#subplot 2 for test data distribution
plt.subplot(1,2,2) #1 row, 2 columns, position 2
plt.bar(category_label, frequency_test.values, color = 'red')
plt.xticks(rotation=90)
plt.title('Class Distribution for Test Data')
plt.xlabel('News Group')
plt.ylabel('Frequency')

plt.subplots_adjust(wspace=0.4) #increase horisontal space
plt.show()
```

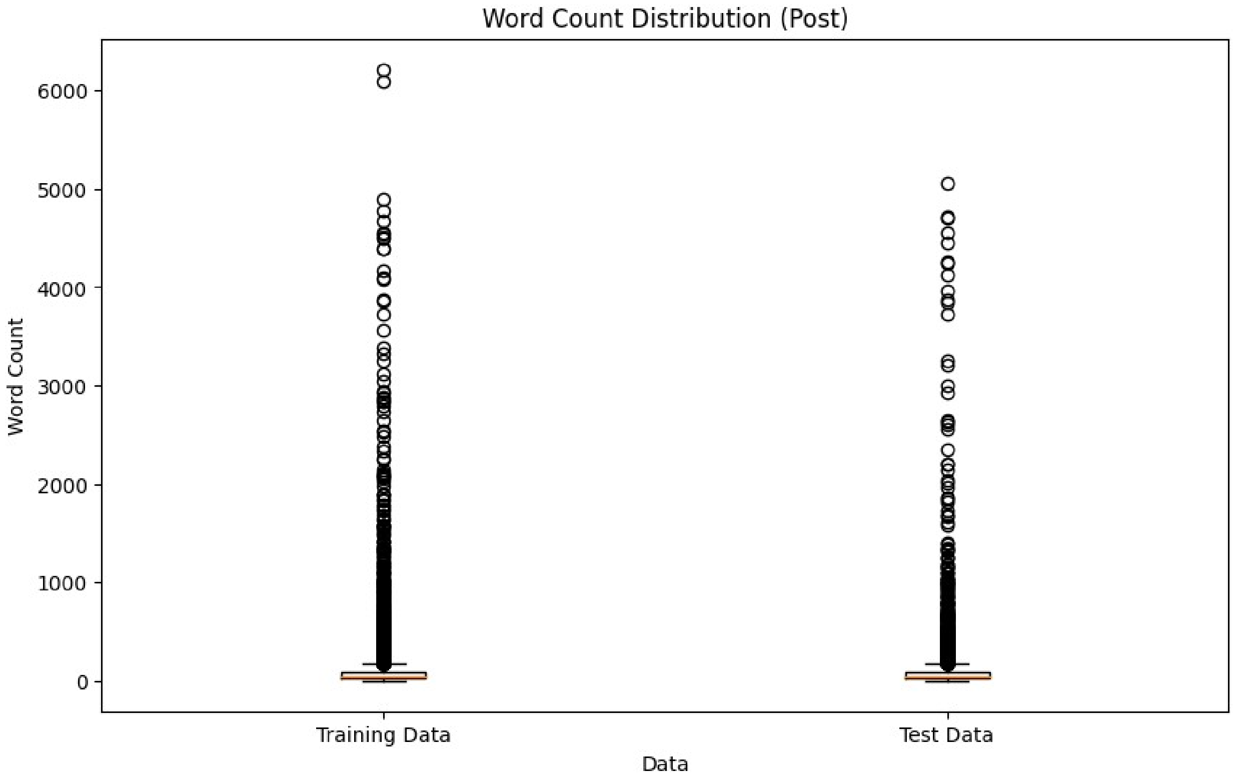


```
#word count distribution
# Create a histogram of word counts
plt.figure(figsize=(10, 6))  # Optional: set the figure size
plt.hist(train_df['word_count'], bins=20, color='blue',
label='Training Data')
plt.hist(test_df['word_count'], bins=20, color='red', label='Test
Data')

# Set labels for x and y axes
plt.xlabel("Word Count")
plt.ylabel("Frequency")

# Set the title of the histogram
plt.title("Word Count Distribution (Post)")

# Add a legend
plt.legend(loc='upper right')

# Show the histogram
plt.show()
```



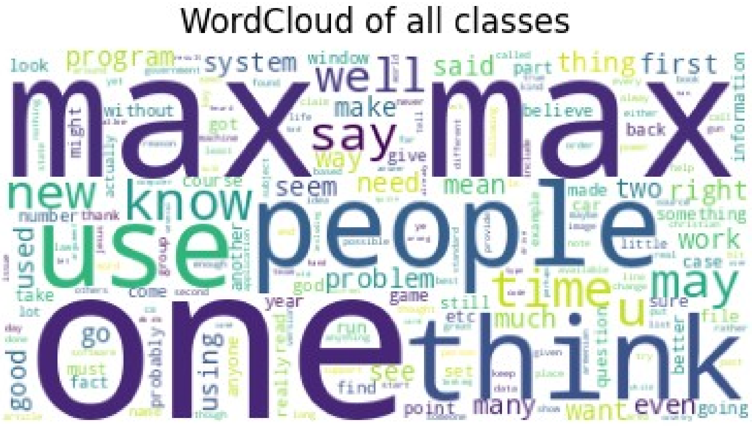
```
# Create a box plot of word counts
plt.figure(figsize=(10, 6))  # Optional: set the figure size
plt.boxplot([train_df['word_count'], test_df['word_count']],
labels=['Training Data', 'Test Data'])

# Set labels for x and y axes
plt.xlabel("Data")
plt.ylabel("Word Count")

# Set the title of the box plot
plt.title("Word Count Distribution (Post)")
```

```
# Show the box plot
plt.show()
```



```
test_df.head()

                                                      data  target
word_count
0  [little, confused, model, bonnevilles, heard, ...       7
41
1  [familiar, format, thingies, seeing, folk, hea...       5
39
2                                         [word, yes]       0
2
3  [attacking, iraqi, drive, kuwait, country, who...      17
317
4  [spent, two, solid, month, arguing, thing, obj...      19
10

# plot word count for the dataset without grouping by classes
combined_text = " ".join(train_df['data'].apply(lambda x: '
'.join(x)))
wordcloud = WordCloud(background_color='white',
max_words=200).generate(combined_text)
fig = plt.figure(figsize=[5,5])
```

```
plt.title('WordCloud of all classes')
plt.axis('off')
plt.imshow(wordcloud)
plt.show()
```



```
#grouping train_df by 'target'
group_train_df = train_df.groupby('target')


# Create a word cloud for each target
for target, group in group_train_df:
    # Combine the text data from the group into a single string
    combined_text = " ".join(group['data'].apply(lambda x: '
'.join(x)))

    # Generate a word cloud
    wordcloud = WordCloud(width=800, height=400,
background_color='white').generate(combined_text)

    # Display the word cloud with the target as the title
    target_name= categories[target]
    plt.figure(figsize=(5, 5))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.title(f"Word Cloud for Category: {target_name}")
    plt.axis("off")
    plt.show()

Output hidden; open in https://colab.research.google.com to view.

# Create a dictionary to store the most frequent words for each
category
most_frequent_words_by_category = {}

for category, data_group in group_train_df:
    # Flatten the list of lists into a single list of strings
    text_for_category = ' '.join([' '.join(doc) for doc in
```

```python
data_group['data']])

    # Tokenize the text and count word frequencies
    tokens = text_for_category.split()
    word_frequencies = Counter(tokens)

    # Get the 10 most frequent words for the category
    most_common_words = word_frequencies.most_common(10)

    # Store the most frequent words in the dictionary
    most_frequent_words_by_category[category] = most_common_words

# Print the most frequent words for each category
for category, frequent_words in
most_frequent_words_by_category.items():
    category_name = categories[category]
    print(f"Most frequent words in category {category_name}:")
    for word, count in frequent_words:
        print(f"{word}: {count}")
    print()
```

```
Most frequent words in category alt.atheism:
god: 442
one: 426
people: 331
would: 323
atheist: 256
think: 228
say: 216
argument: 193
religion: 182
thing: 179

Most frequent words in category comp.graphics:
image: 671
file: 471
program: 292
graphic: 285
format: 248
jpeg: 234
also: 232
would: 228
use: 222
system: 218

Most frequent words in category comp.os.ms-windows.misc:
max: 4449
window: 644
bhj: 452
file: 425
```

```
giz: 422
wm: 254
problem: 237
use: 225
ah: 219
driver: 214

Most frequent words in category comp.sys.ibm.pc.hardware:
drive: 726
card: 352
system: 290
disk: 286
one: 273
scsi: 259
controller: 256
would: 243
problem: 218
use: 216

Most frequent words in category comp.sys.mac.hardware:
mac: 348
drive: 269
one: 259
apple: 249
problem: 234
would: 212
get: 183
card: 175
know: 172
use: 171

Most frequent words in category comp.windows.x:
file: 750
window: 678
program: 508
widget: 458
use: 456
entry: 444
server: 403
get: 366
application: 331
one: 319

Most frequent words in category misc.forsale:
new: 268
sale: 248
offer: 245
price: 193
do: 180
one: 176
```

```
please: 165
shipping: 164
game: 158
condition: 152

Most frequent words in category rec.autos:
car: 695
would: 258
one: 206
like: 206
get: 194
good: 146
time: 140
also: 139
engine: 134
new: 129

Most frequent words in category rec.motorcycles:
bike: 424
one: 260
like: 195
would: 187
get: 180
dod: 178
know: 149
ride: 136
motorcycle: 134
time: 120

Most frequent words in category rec.sport.baseball:
year: 383
game: 342
team: 268
would: 247
one: 218
player: 215
run: 212
good: 198
think: 196
last: 192

Most frequent words in category rec.sport.hockey:
game: 670
team: 632
play: 353
hockey: 341
player: 332
pt: 307
season: 301
would: 293
```

```
year: 288
period: 264

Most frequent words in category sci.crypt:
key: 1083
would: 562
db: 549
encryption: 544
chip: 529
one: 502
system: 482
use: 444
government: 412
people: 374

Most frequent words in category sci.electronics:
one: 331
would: 269
use: 258
wire: 208
circuit: 197
get: 193
like: 189
power: 167
know: 162
ground: 162

Most frequent words in category sci.med:
one: 414
would: 319
get: 238
also: 231
patient: 222
people: 220
time: 218
use: 215
disease: 207
know: 202

Most frequent words in category sci.space:
space: 828
would: 378
launch: 309
nasa: 287
one: 280
satellite: 280
system: 261
year: 242
also: 236
time: 230
```

```
Most frequent words in category soc.religion.christian:
god: 1086
would: 631
one: 627
jesus: 460
people: 457
christian: 448
church: 381
say: 377
know: 344
think: 342

Most frequent words in category talk.politics.guns:
gun: 749
would: 569
people: 392
one: 349
firearm: 327
weapon: 320
file: 311
right: 296
law: 274
state: 271

Most frequent words in category talk.politics.mideast:
armenian: 1067
people: 795
one: 704
would: 557
said: 504
israel: 483
turkish: 451
jew: 445
say: 401
israeli: 398

Most frequent words in category talk.politics.misc:
would: 514
people: 497
president: 424
think: 381
stephanopoulos: 345
one: 342
know: 314
government: 288
state: 266
going: 263

Most frequent words in category talk.religion.misc:
```

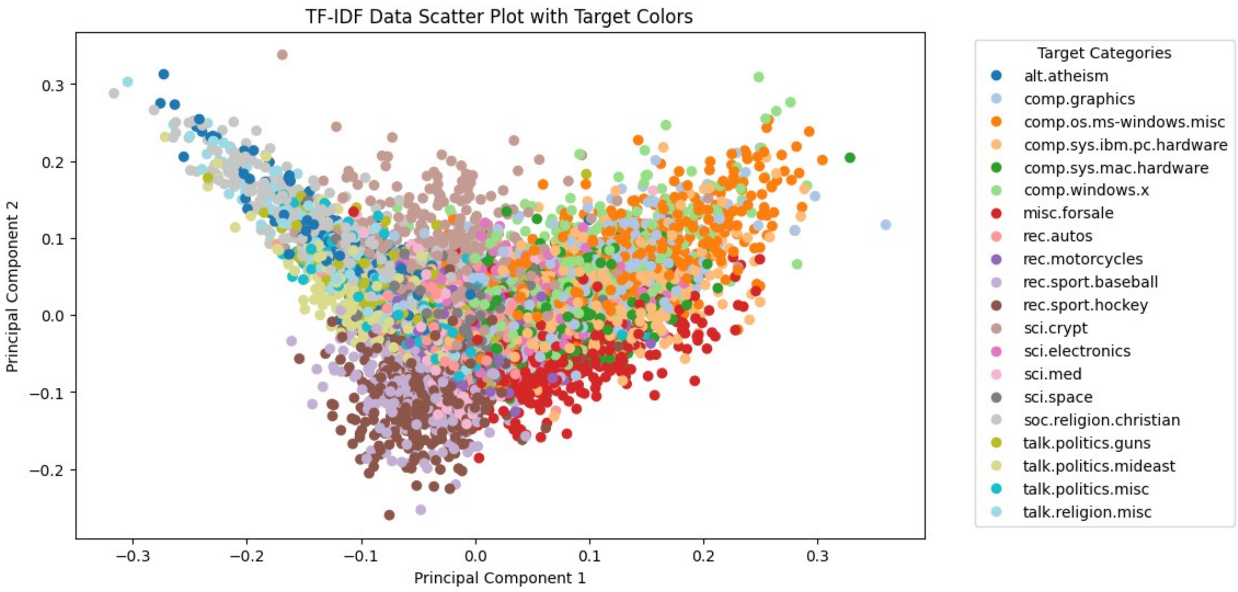
```
god: 334
one: 292
people: 267
would: 258
jesus: 253
christian: 231
say: 210
think: 163
know: 159
bible: 159
```

```
# #bag of word

# # Convert the tokenized words back to a space-separated string for
each document (bag of word method expect a list, not a list of lists)
# train_df['data'] = train_df['data'].apply(lambda tokens: '
'.join(tokens))


# bow_vectorizer = CountVectorizer()
# bow_features = bow_vectorizer.fit_transform(train_df['data'])

# TF-IDF

train_df['data'] = train_df['data'].apply(lambda tokens: '
'.join(tokens))
test_df['data'] =test_df['data'].apply(lambda tokens:' '.join(tokens))

tfidf_vectorizer = TfidfVectorizer(max_features=5000)
X_train = tfidf_vectorizer.fit_transform(train_df['data'])

X_test = tfidf_vectorizer.transform(test_df['data'])

feature_names = tfidf_vectorizer.get_feature_names_out()
# Convert the TF-IDF matrix to a DataFrame
tfidf_df = pd.DataFrame(X_train.toarray(), columns=feature_names)

tfidf_df.head()
```

```
    aa  aaa   ab  abc  ability  able  abort  abortion  abraham
absence  ...  \
0  0.0  0.0  0.0  0.0      0.0   0.0    0.0       0.0      0.0
0.0  ...
1  0.0  0.0  0.0  0.0      0.0   0.0    0.0       0.0      0.0
0.0  ...
2  0.0  0.0  0.0  0.0      0.0   0.0    0.0       0.0      0.0
0.0  ...
3  0.0  0.0  0.0  0.0      0.0   0.0    0.0       0.0      0.0
0.0  ...
4  0.0  0.0  0.0  0.0      0.0   0.0    0.0       0.0      0.0
```

```
0.0  ...

   youth  yup   zd  zealand  zero  zionist  zone  zoom  zoroastrian
zv
0    0.0  0.0  0.0      0.0   0.0      0.0   0.0   0.0          0.0
0.0
1    0.0  0.0  0.0      0.0   0.0      0.0   0.0   0.0          0.0
0.0
2    0.0  0.0  0.0      0.0   0.0      0.0   0.0   0.0          0.0
0.0
3    0.0  0.0  0.0      0.0   0.0      0.0   0.0   0.0          0.0
0.0
4    0.0  0.0  0.0      0.0   0.0      0.0   0.0   0.0          0.0
0.0

[5 rows x 5000 columns]

# Calculate the mean TF-IDF score for each attribute (word)
attribute_sum = tfidf_df.sum()

# Sort the attributes by their mean TF-IDF scores in descending order
top_10_attributes =
attribute_sum.sort_values(ascending=False).head(10)

# Print the top 10 most common attributes
print("Top 10 Most Common Words:")
print(top_10_attributes)

Top 10 Most Common Words:
would     264.646734
one       236.849392
know      192.507951
like      187.480202
get       183.182786
think     157.084109
people    156.578262
could     144.392831
time      140.721605
anyone    139.185461
dtype: float64

# Reduce the dimensionality of the TF-IDF data using PCA to make
scatter plot.

pca = PCA(n_components=2)  # Reduce to 2 dimensions for visualization
tfidf_df_reduced = pca.fit_transform(tfidf_df)

# convert tfidf_df_reduced (array) to dataframe. There are 2
components (attribute) for this dataframe
tfidf_df_reduced = pd.DataFrame(tfidf_df_reduced, columns=['PC1',
'PC2'])
```

```
# Add the 'target' column to tfidf_df_reduced for labeling purpose
tfidf_df_reduced['target'] = pd.Categorical(train_df['target'])

# Create a scatter plot with colors representing the target labels
plt.figure(figsize=(10, 6))
scatter = plt.scatter(
    tfidf_df_reduced['PC1'],
    tfidf_df_reduced['PC2'],
    c=tfidf_df_reduced['target'].cat.codes,  # Use categorical codes
for color-coding
    cmap='tab20'
)

# Add legend with target category names and colors
legend_labels = [plt.Line2D([0], [0], marker='o', color='w',
label=category_name, markersize=8, markerfacecolor=color)
                 for category_name, color in zip(categories,
plt.cm.tab20.colors)]

plt.legend(handles=legend_labels, title='Target
Categories',bbox_to_anchor=(1.05, 1), loc='upper left')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('TF-IDF Data Scatter Plot with Target Colors')
plt.show()
```



```python
#for Multinomial NaiveBayes method, use "MultinomialNB" function
fromsklearn.naive_bayes
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train, train_df['target'])
```

```
MultinomialNB()
```

```python
# use "accuracy_score" and "classification_report" function from
sklearn.metrics

predictions = nb_classifier.predict(X_test)
accuracy = accuracy_score(test_df['target'], predictions)

report = classification_report(test_df['target'], predictions,
target_names=[categories [i] for i in frequency_test.index])
print(f'Accuracy: {accuracy}')
print(report)
```

```
Accuracy: 0.6702302631578947
                          precision    recall  f1-score   support

             alt.atheism       0.60      0.30      0.40       311
           comp.graphics       0.57      0.64      0.60       384
 comp.os.ms-windows.misc       0.60      0.56      0.58       379
comp.sys.ibm.pc.hardware       0.56      0.68      0.62       385
   comp.sys.mac.hardware       0.67      0.61      0.64       371
          comp.windows.x       0.70      0.74      0.72       390
            misc.forsale       0.76      0.78      0.77       382
               rec.autos       0.75      0.71      0.73       370
         rec.motorcycles       0.74      0.74      0.74       386
      rec.sport.baseball       0.83      0.81      0.82       379
        rec.sport.hockey       0.87      0.92      0.89       389
               sci.crypt       0.71      0.75      0.73       378
         sci.electronics       0.63      0.51      0.56       382
                 sci.med       0.78      0.76      0.77       382
               sci.space       0.77      0.74      0.76       377
  soc.religion.christian       0.43      0.92      0.59       384
      talk.politics.guns       0.55      0.74      0.63       352
   talk.politics.mideast       0.82      0.77      0.79       370
      talk.politics.misc       0.72      0.33      0.45       302
      talk.religion.misc       0.62      0.03      0.06       243

                accuracy                           0.67      7296
               macro avg       0.68      0.65      0.64      7296
            weighted avg       0.69      0.67      0.66      7296
```