Lab - 5

PRML
AY 2020-21, Trimester - III **J Jahnavi(B19CSE109)**

a) Data Preparation:
- Load the data

```
df = pd.read_csv('train.csv')
df['target'].value_counts()
```

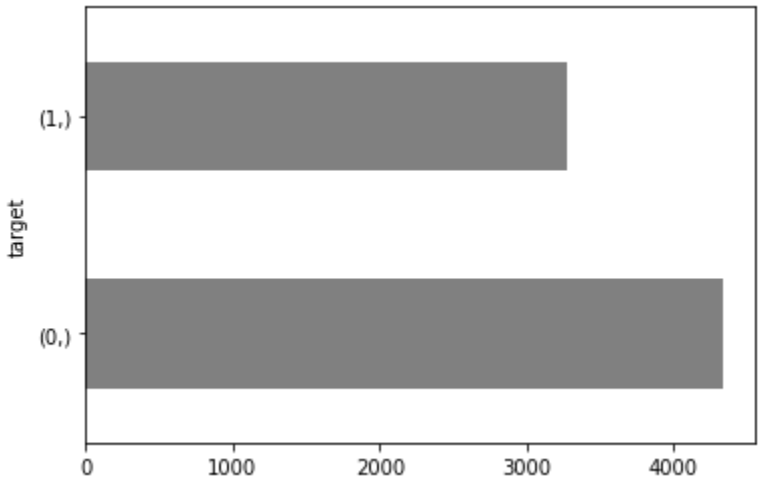
```
0    4342
1    3271
Name: target, dtype: int64
```

```
df.head()
```

	id	keyword	location	text	target
0	1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...	1
1	4	NaN	NaN	Forest fire near La Ronge Sask. Canada	1
2	5	NaN	NaN	All residents asked to 'shelter in place' are ...	1
3	6	NaN	NaN	13,000 people receive #wildfires evacuation or...	1
4	7	NaN	NaN	Just got sent this photo from Ruby #Alaska as ...	1

Read the CSV file into dataframe ‘df’

- Plot the count for each target



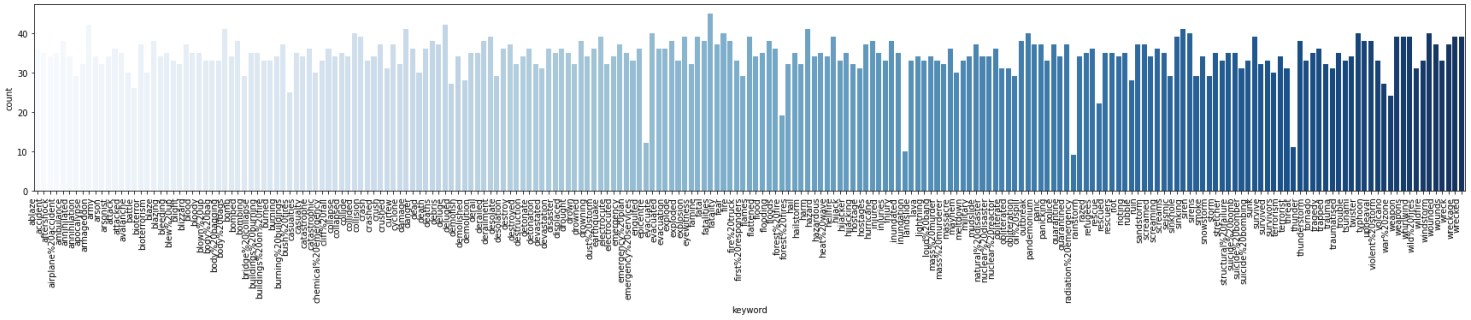
- Print the unique keywords

```
unique_keyw = df['keyword'].value_counts()
unique_loc = df['location'].value_counts()
print(f'Number of Unique Keywords = {len(unique_keyw)}\n--------------------------------------------------------------------------------')
for i in unique_keyw.keys():
  print(i)
```

```
Number of Unique Keywords = 221
--------------------------------------------------------------------------------
fatalities
deluge
armageddon
sinking
damage
harm
body%20bags
```

  - The total number of unique keywords are **221**

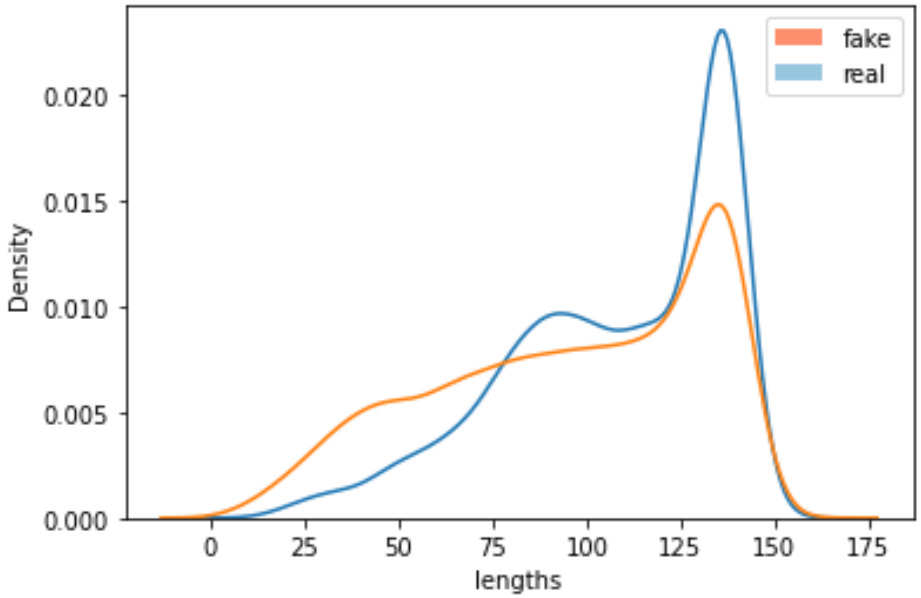
- Plot the count of each keyword

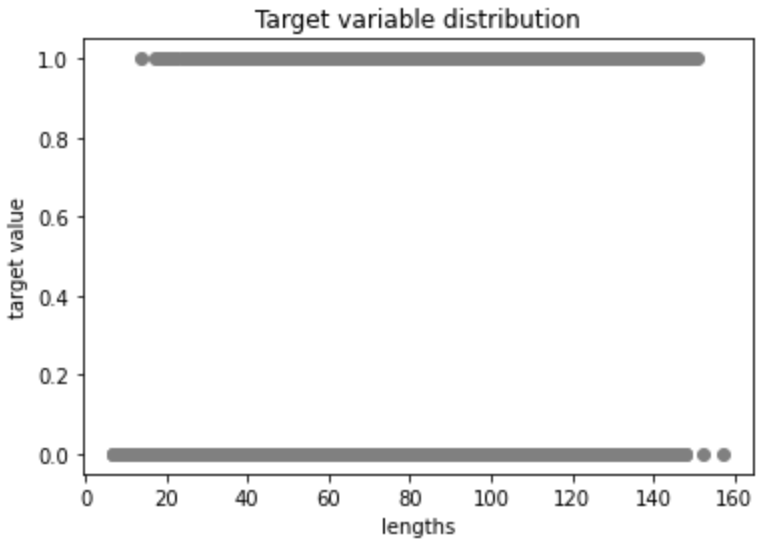


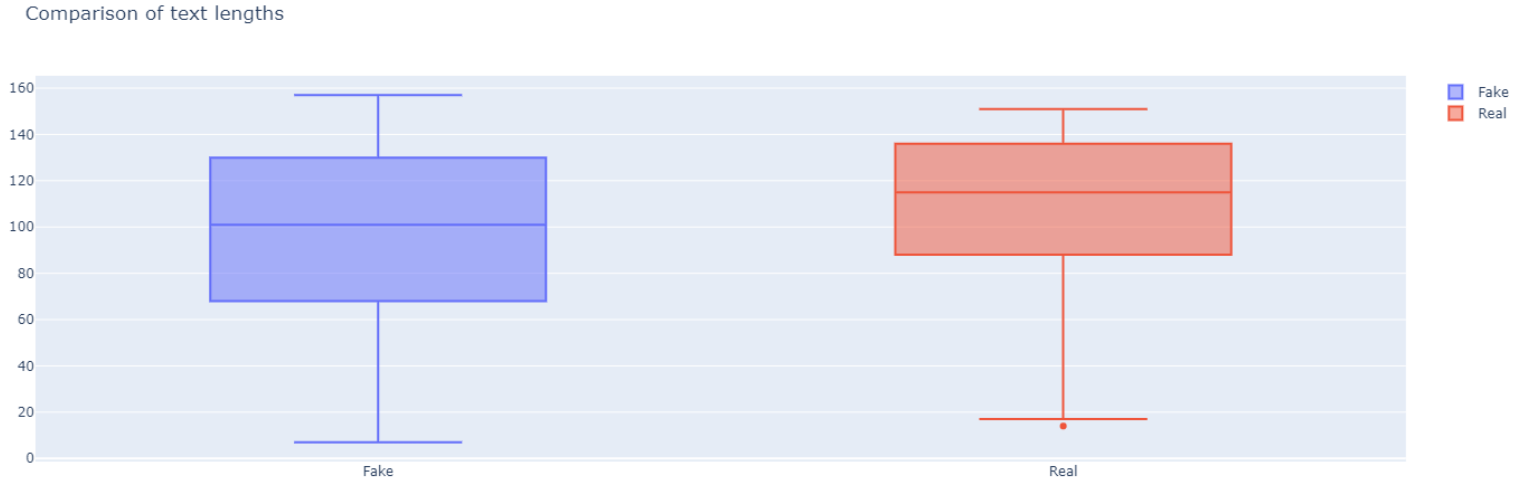
Above is the plot for the count of each keyword, plotted using:

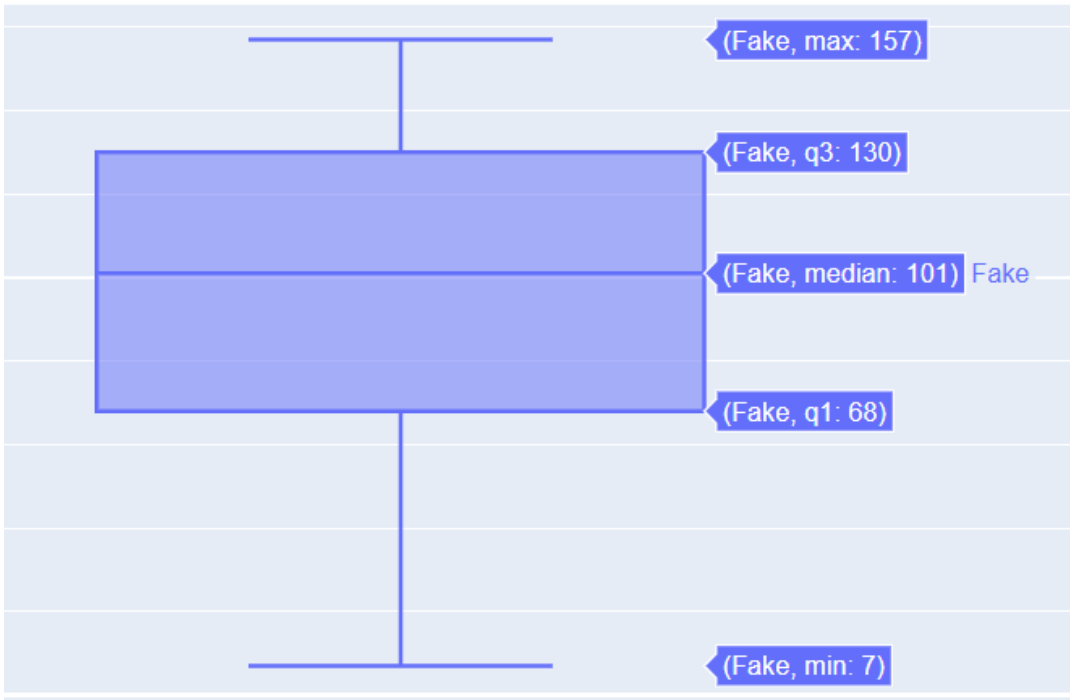
```
chart = sns.countplot(df['keyword'], palette='Blues')
chart.set_xticklabels(chart.get_xticklabels(), rotation=90,
horizontalalignment='right')
```

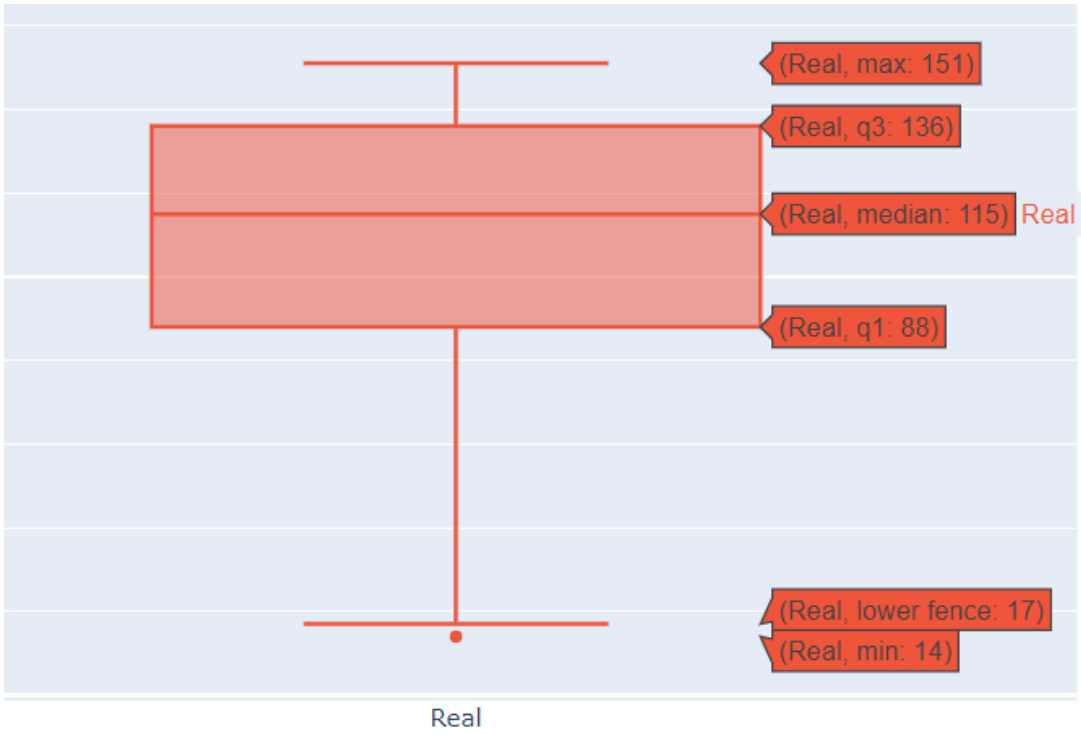
- Visualize the correlation of the length of a tweet with its target




Target variable distribution
target value
lengths
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
120
140
160


Comparison of text lengths
Fake
Real
160
140
120
100
80
60
40
20
0
Fake
Real


(Fake, max: 157)
(Fake, q3: 130)
(Fake, median: 101)
Fake
(Fake, q1: 68)
(Fake, min: 7)
Fake


(Real, max: 151)
(Real, q3: 136)
(Real, median: 115)
Real
(Real, q1: 88)
(Real, lower fence: 17)
(Real, min: 14)
Real

- Print the null values in a column

```
columns_null = df.columns[df.isnull().any()]
print(df.isnull().sum())
```

```
id             0
keyword       61
location    2533
text           0
target         0
lengths        0
dtype: int64
```

- Removing null values

```
df= df.dropna()
print(df.shape)
df.head()
```

```
(5080, 6)
```

	id	keyword	location	text	target	lengths
31	48	ablaze	Birmingham	@bbcmtd Wholesale Markets ablaze http://t.co/l...	1	55
32	49	ablaze	Est. September 2012 - Bristol	We always try to bring the heavy. #metal #RT h...	0	67
33	50	ablaze	AFRICA	#AFRICANBAZE: Breaking news:Nigeria flag set a...	1	82
34	52	ablaze	Philadelphia, PA	Crying out for more! Set me ablaze	0	34
35	53	ablaze	London, UK	On plus side LOOK AT THE SKY LAST NIGHT IT WAS...	0	76

- Removed Double Spaces, Hyphens, arrows, Emojis, Emoticons, URLs, and other Non-English or special symbols

	id	keyword	location	text	target	lengths	mod_text
31	48	ablaze	Birmingham	@bbcmtd Wholesale Markets ablaze http://t.co/l...	1	55	bbcmtd wholesale markets ablaze
32	49	ablaze	Est. September 2012 - Bristol	We always try to bring the heavy. #metal #RT h...	0	67	we always try to bring the heavy metal rt
33	50	ablaze	AFRICA	#AFRICANBAZE: Breaking news:Nigeria flag set a...	1	82	africanbaze breaking newsnigeria flag set abla...
34	52	ablaze	Philadelphia, PA	Crying out for more! Set me ablaze	0	34	crying out for more set me ablaze
35	53	ablaze	London, UK	On plus side LOOK AT THE SKY LAST NIGHT IT WAS...	0	76	on plus side look at the sky last night it was...

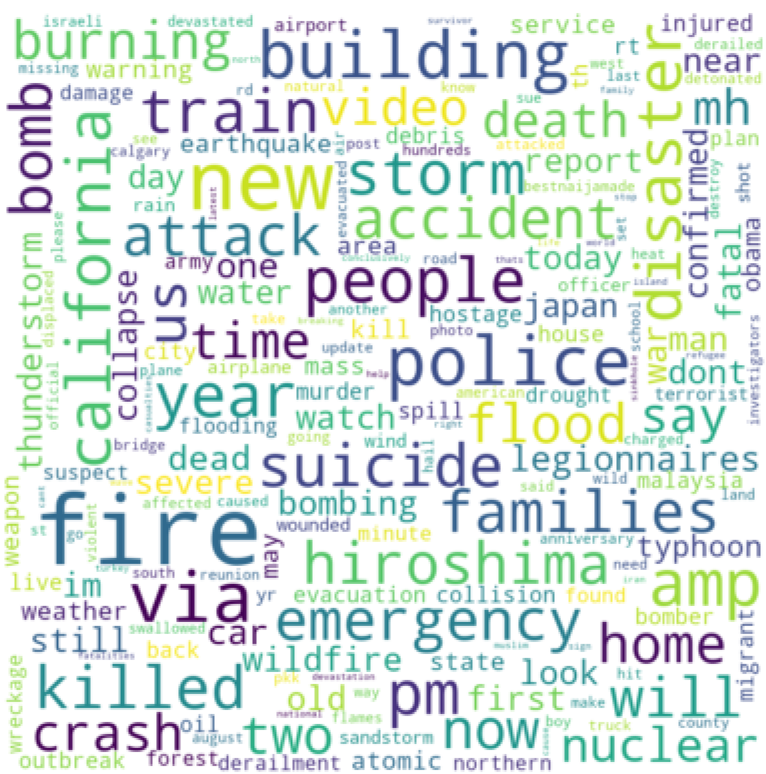
- Replaced wrong spellings with correct ones as follows:

```
#spelling correction
!pip install pyspellchecker
```

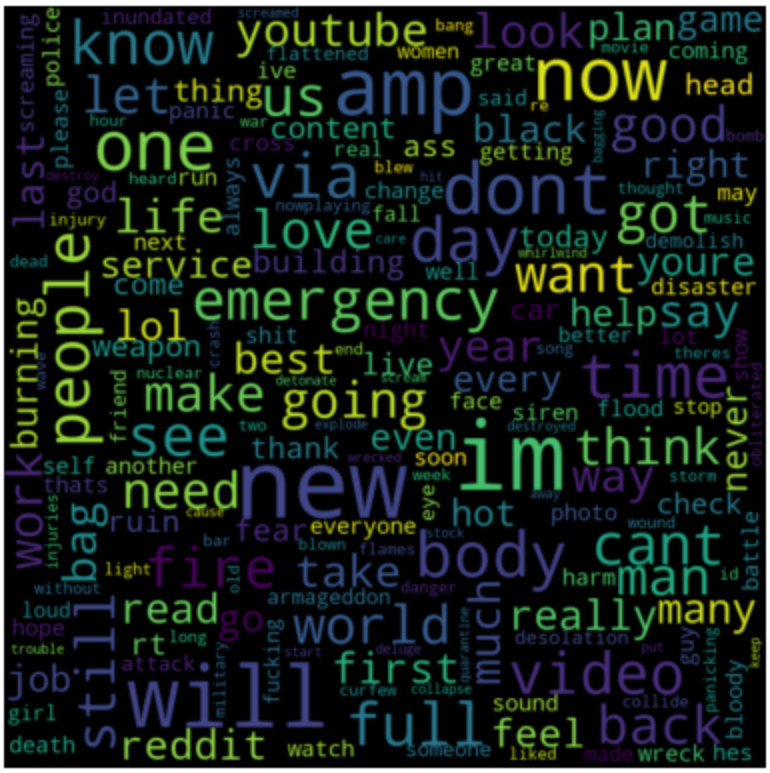
```
Requirement already satisfied: pyspellchecker in /usr/local/lib/python3.7/dist-packages (0.6.2)
```

```
#spelling correction
from spellchecker import SpellChecker

spell = SpellChecker()
def correct_spelling(text):
    corrected_text = []
    misspelled_words = spell.unknown(text.split())
    for word in text.split():
        if word in misspelled_words:
            corrected_text.append(spell.correction(word))
        else:
            corrected_text.append(word)
    return " ".join(corrected_text)

df["mod_text"] = df["mod_text"].apply(lambda t: correct_spelling(t))
df.head()
```

- Plot a word cloud of the real and fake target
  For Real Target:



For Fake Target:



- Remove all columns except text and target

```
df = df[['text','mod_text','target']]
print(df.shape)
df.head()
```

```
(5080, 3)
```

	text	mod_text	target
31	@bbcmtd Wholesale Markets ablaze http://t.co/l...	bbcmtd wholesale markets ablaze	1
32	We always try to bring the heavy. #metal #RT h...	we always try to bring the heavy metal rt	0
33	#AFRICANBAZE: Breaking news:Nigeria flag set a...	africanbaze breaking newsnigeria flag set abla...	1
34	Crying out for more! Set me ablaze	crying out for more set me ablaze	0
35	On plus side LOOK AT THE SKY LAST NIGHT IT WAS...	on plus side look at the sky last night it was...	0

Here, the column ‘mod_text’ has the modified data after removing words or symbols which have little significance in the analyzation

- Split data into train and validation

```
from sklearn.model_selection import train_test_split
x = df['mod_text']
y = df['target']
x_train, x_validation, y_train, y_validation = train_test_split(x, y, test_size=0.25, random_state=40)
```

b) Compute the Term Document matrix for the whole dataset as well as for the two classes.

i) For the whole dataset:

```
#Term Document matrix for the whole dataset
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
vct = vectorizer.fit_transform(df['mod_text'])
print(vectorizer.get_feature_names())
pd.DataFrame(vct.toarray(),columns=vectorizer.get_feature_names())
```

ii) For class 1(real target):

```
#TDM for class 1
vectorizer = CountVectorizer()
vct = vectorizer.fit_transform(df[df["target"] == 1]['mod_text'])
print(vectorizer.get_feature_names())
pd.DataFrame(vct.toarray(),columns=vectorizer.get_feature_names())
```

iii) For class 0(fake target):

```
#TDM for class 0
vectorizer = CountVectorizer()
vct = vectorizer.fit_transform(df[df["target"] == 0]['mod_text'])
print(vectorizer.get_feature_names())
pd.DataFrame(vct.toarray(),columns=vectorizer.get_feature_names())
```

- Find the frequency of words in classes 0 and 1.

```
from collections import Counter
#class 1
f_real=df[df["target"] == 1]
f_class1=Counter()
f_real['mod_text'].str.split().apply(f_class1.update)
f_class1
```

```
'fact': 2,
'atomic': 33,
'bombs': 9,
'fat': 3,
'says': 23,
'lot': 5,
'mentality': 1,
'went': 11,
```

```
#class 0
f_fake=df[df["target"]==0]
f_class0=Counter()
f_fake['mod_text'].str.split().apply(f_class0.update)
f_class0
```

```
'very': 12,
'slightly': 4,
'intrigued': 1,
'far': 9,
'build': 4,
'own': 11,
'lead': 7,
```

The above pictures show the frequencies for words in each class in a dictionary.

- Does the sum of the unique words in target 0 and 1 sum to the total number of unique words in the whole document? Why or why not?

```
df['mod_text'].nunique() == ((df[df["target"] == 1]['mod_text'].nunique())+(df[df["target"] == 0]['mod_text'].nunique()))
#to check whether the sum of the unique words in target 0 and 1 sum upto the total number of unique words in the whole document
```

```
False
```

No, the sum of the unique words in target 0 and 1 do not sum to the total number of unique words in the whole document as there might be common words in both the classes which would get counted twice in the sum of the unique words in target 0 and 1.
So, the sum of the unique words in target 0 and 1 would be greater than or equal to the total number of unique words in the whole document:

```
df['mod_text'].nunique() <= ((df[df["target"] == 1]['mod_text'].nunique())+(df[df["target"] == 0]['mod_text'].nunique()))
#to check whether the sum of the unique words in target 0 and 1 sum upto the total number of unique words in the whole document
```

```
True
```

- Calculate the probability for each word in a given class:

```
c_0=pd.DataFrame()
c_0['word']=f_class0.keys()
c_0['count']=f_class0.values()
sum_0 = 0
for count in c_0['count']:
  sum_0 += count
c_0['probability']=c_0['count']/(sum_0)
c_0.head()
```

	word	count	probability
0	we	108	0.002670
1	always	24	0.000593
2	try	10	0.000247
3	to	813	0.020096
4	bring	9	0.000222

ss 1

```
c_1=pd.DataFrame()
c_1['word']=f_class1.keys()
c_1['count']=f_class1.values()
sum_1 = 0
for count in c_1['count']:
  sum_1 +=count
c_1['probability']=c_1['count']/(sum_1)
```

Here, the data frame “c_0” has the probabilities for each word in the class with a target value of 0 along with the word count whereas the data frame “c_1” is for the class with a target value of 1.

- We have calculated the probability of occurrence of the word in a class, we can now substitute the values in the Bayes equation. If a word from the new sentence does not occur in the class within the training set, the equation becomes zero. This problem can be solved using smoothing like Laplace smoothing. Use Bayes with Laplace smoothing to predict the probability for sentences in the validation set.

Used Laplace smoothing to predict probability for sentences in the validation set as follows:

```
total_value = df['mod_text'].nunique()
prob_0=[]
t=Counter()
df['mod_text'].str.split().apply(t.update)
for i in t.keys():
  if i in list(f_class0.keys()):
    count0=f_class0[i]
  else:
    count0=0
  prob_0.append((count0+1)/(sum_0+total_value))

laplace0=pd.DataFrame()
laplace0['text']=t.keys()
laplace0['probability']=prob_0
```

```
prob_1=[]
t = df['mod_text']
for i in t.keys():
  if i in list(f_class1.keys()):
    count=f_class1[i]
  else:
    count=0
  prob_1.append((count+1)/(sum_1+total_value))

laplace1=pd.DataFrame()
laplace1['text']=t.keys()
laplace1['probability']=prob_1
```