DTSA-5511 Week 4 - NLP Disaster Tweets Kaggle Mini-Project

A Jupyter notebook with a description of the problem/data, exploratory data analysis (EDA) procedure, analysis (model building and training), result, and discussion/conclusion.

Project description

Twitter has become an important communication channel in times of emergency. The ubiquitousness of smartphones enables people to announce an emergency they're observing in real-time. Because of this, more agencies are interested in programatically monitoring Twitter (i.e. disaster relief organizations and news agencies). The challenge of this Kaggle Mini-project is to use NLP (Natural Language Processing) to predict which Tweets are about real disasters and which one's arent.

What is NLP?

NLP (Neuro-Linguistic Programming) is the science or art that boasts a unique approach to the enhancement of your communications skills, personal development, as well as psychotherapy. NLP is a powerful method that influences the behavior of the brain using language, among other forms of communication to allow one person to 're-code' the brain's response to stimuli (or its programming).

Dataset Description

The dataset contains three files: **train.csv**, **test.csv** and **sample_submission.csv**. Each sample in the train and test set has the following information:

- The text of a tweet.
- A keyword from that tweet (although this may be blank!).
- The location the tweet was sent from (may also be blank).

Columns:

- id: a unique identifier for each tweet
- text: the text of the tweet.
- location: the location the tweet was sent from (may be blank).

- keyword: a particular keyword from the tweet (may be blank).
- target: it denotes whether a tweet is about a real disaster (1) or not (0). This column is only in **train.csv** file.

Data size:

- Train data contains 7613 columns and 5 rows.
- Test data contains 3263 columns and 4 rows.

1. Exploratory Data Analysis (EDA)

1.1. Importing libraries

Loading libraries required this mini-project

In [1]:

```
# Utlity
import pandas as pd
import numpy as np

# Plots
import matplotlib.pyplot as plt
%matplotlib inline
from matplotlib.ticker import MaxNLocator
import matplotlib.gridspec as gridspec
import matplotlib.patches as mpatches
import plotly.express as px
import seaborn as sns

# EDA
import re
import missingno as msno
from wordcloud import WordCloud
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# modeling and training
import tensorflow as tf
from keras.preprocessing.text import Tokenizer
from tensorflow.keras.layers import Embedding
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, GRU
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from sklearn.model_selection import train_test_split
from tensorflow.keras import optimizers
from keras.callbacks import ReduceLROnPlateau, EarlyStopping, ModelCheckpoint

import warnings
warnings.filterwarnings("ignore")
```

1.2. Reading data

1.2.1. Train data

Loading train dataset from train.csv file.

In [2]:
```
train_df = pd.read_csv("./data/npl/train.csv")
display(train_df.head(), train_df.describe(include='all'))
```

	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

	id	keyword	location	text	target
count	7613.000000	7552	5080	7613	7613.00000
unique	NaN	221	3341	7503	NaN
top	NaN	fatalities	USA	11-Year-Old Boy Charged With Manslaughter of T...	NaN
freq	NaN	45	104	10	NaN
mean	5441.934848	NaN	NaN	NaN	0.42966
std	3137.116090	NaN	NaN	NaN	0.49506
min	1.000000	NaN	NaN	NaN	0.00000
25%	2734.000000	NaN	NaN	NaN	0.00000
50%	5408.000000	NaN	NaN	NaN	0.00000
75%	8146.000000	NaN	NaN	NaN	1.00000
max	10873.000000	NaN	NaN	NaN	1.00000

In [3]:
```
print('Train dataset contains %d columns and %d rows' %(train_df.shape[0], train_df.sha
```

```
Train dataset contains 7613 columns and 5 rows
```

1.2.2. Test data

Loading the test dataset from test.csv file.

In [4]:
```
test_df = pd.read_csv("./data/npl/test.csv")
display(test_df.head(), test_df.describe(include='all'))
```

	id	keyword	location	text
0	0	NaN	NaN	Just happened a terrible car crash
1	2	NaN	NaN	Heard about #earthquake is different cities, s...

	id	keyword	location	text
2	3	NaN	NaN	there is a forest fire at spot pond, geese are...
3	9	NaN	NaN	Apocalypse lighting. #Spokane #wildfires
4	11	NaN	NaN	Typhoon Soudelor kills 28 in China and Taiwan

	id	keyword	location	text
count	3263.000000	3237	2158	3263
unique	NaN	221	1602	3243
top	NaN	deluged	New York	11-Year-Old Boy Charged With Manslaughter of T...
freq	NaN	23	38	3
mean	5427.152927	NaN	NaN	NaN
std	3146.427221	NaN	NaN	NaN
min	0.000000	NaN	NaN	NaN
25%	2683.000000	NaN	NaN	NaN
50%	5500.000000	NaN	NaN	NaN
75%	8176.000000	NaN	NaN	NaN
max	10875.000000	NaN	NaN	NaN

In [5]:
```
print('Test dataset contains %d columns and %d rows' %(test_df.shape[0], test_df.shape[
```

```
Test dataset contains 3263 columns and 4 rows
```

1.2.3 Submission data

Loading the submission data from sample_submission.csv file.

In [6]:
```
submission_df = pd.read_csv("./data/npl/sample_submission.csv")
submission_df.head()
```

Out[6]:

	id	target
0	0	0
1	2	0
2	3	0
3	9	0
4	11	0

1.3. Data Inspection

1.3.1. Inspecting for null values

Checking if the train and test dataset contains null values

In [7]:
```
# train dataset
train_df.isna().sum()
```

Out[7]:
```
id             0
keyword       61
location    2533
text           0
target         0
dtype: int64
```

In [8]:
```
msno.bar(train_df,color = 'chocolate')
plt.title('Missing values in train dataset\n\n',fontsize=30)
plt.show()
```

In [9]:
```
# test dataset
test_df.isna().sum()
```

Out[9]:
```
id             0
keyword       26
location    1105
text           0
dtype: int64
```

In [10]:
```
msno.bar(test_df,color = 'salmon')
plt.title('Missing values in test dataset\n\n',fontsize=30)
plt.show()
```

The plots show that keyword and location have missing value but location has more missing values than keyword on both datasets.

1.3.2. Inspecting the target distribution

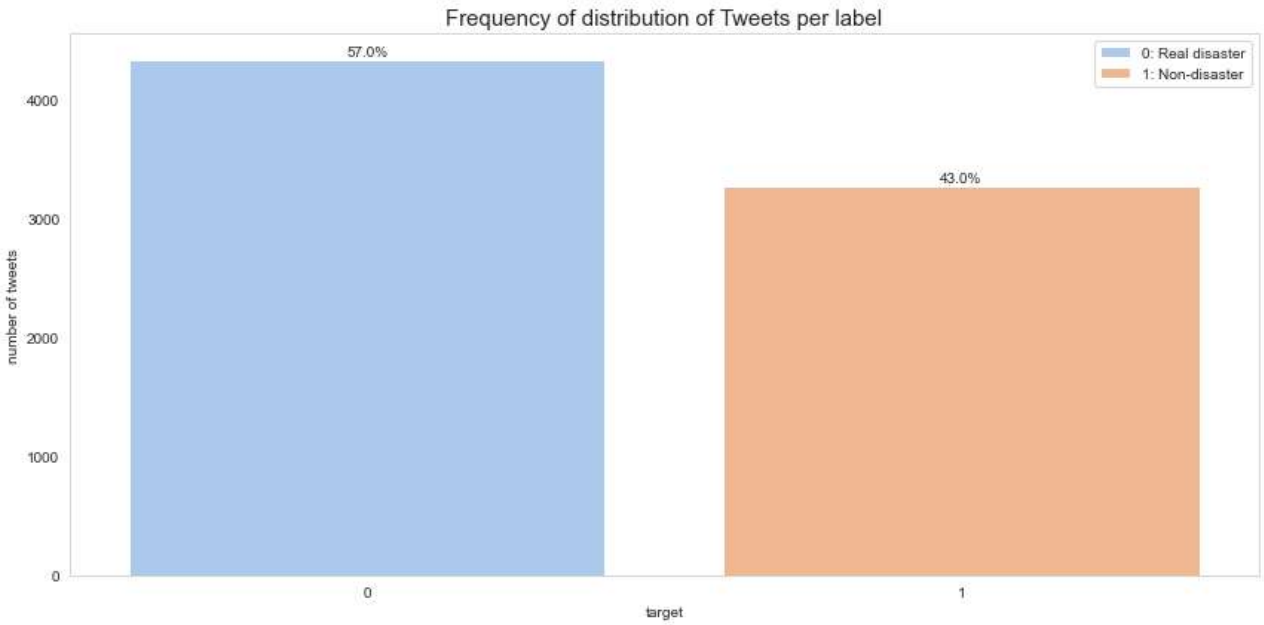
Checking the distribution of the target feature on the train dataset.

In [11]:

```
sns.set_style('whitegrid')
fig,axes = plt.subplots(figsize=(12,6))

ax = sns.countplot(x='target',data=train_df, palette="pastel",hue="target",dodge=False)
ax.grid(False) #remove grid
# Labels
plt.title('Frequency of distribution of Tweets per label',fontsize=15)
plt.ylabel("number of tweets")

# Percentage
total = len(train_df)
for p in ax.patches:
    percentage = f'{100 * p.get_height() / total:.1f}%\n'
    x = p.get_x() + p.get_width() / 2
    y = p.get_height()
    ax.annotate(percentage, (x, y), ha='center', va='center')
plt.tight_layout()
# Legend
plt.legend(loc='upper right',labels = ['0: Real disaster', '1: Non-disaster'])
plt.show()
```



The plot shows that there are more tweets with real disasters than non-real ones.

1.3.3. Counting the keywords

Counting on train and test datasets the number of words on keyword feature.

In [12]:
```
# train dataset
kwd_cnt_train = train_df['keyword'].value_counts()
kwd_cnt_train
```

Out[12]:
```
fatalities              45
deluge                  42
armageddon              42
sinking                 41
damage                  41
                        ..
forest%20fire           19
epicentre               12
threat                  11
inundation              10
radiation%20emergency    9
Name: keyword, Length: 221, dtype: int64
```

In [13]:
```
# test dataset
kwd_cnt_test = test_df['keyword'].value_counts()
kwd_cnt_test
```

Out[13]:
```
deluged             23
demolished          22
rubble              22
first%20responders  21
seismic             21
                    ..
threat               5
fatalities           5
forest%20fire        5
inundation           4
```

```
epicentre              1
Name: keyword, Length: 221, dtype: int64
```

1.3.4. Top 20 keywords in tweets

Plotting the top 20 keywords in tweets on train and test datasets.

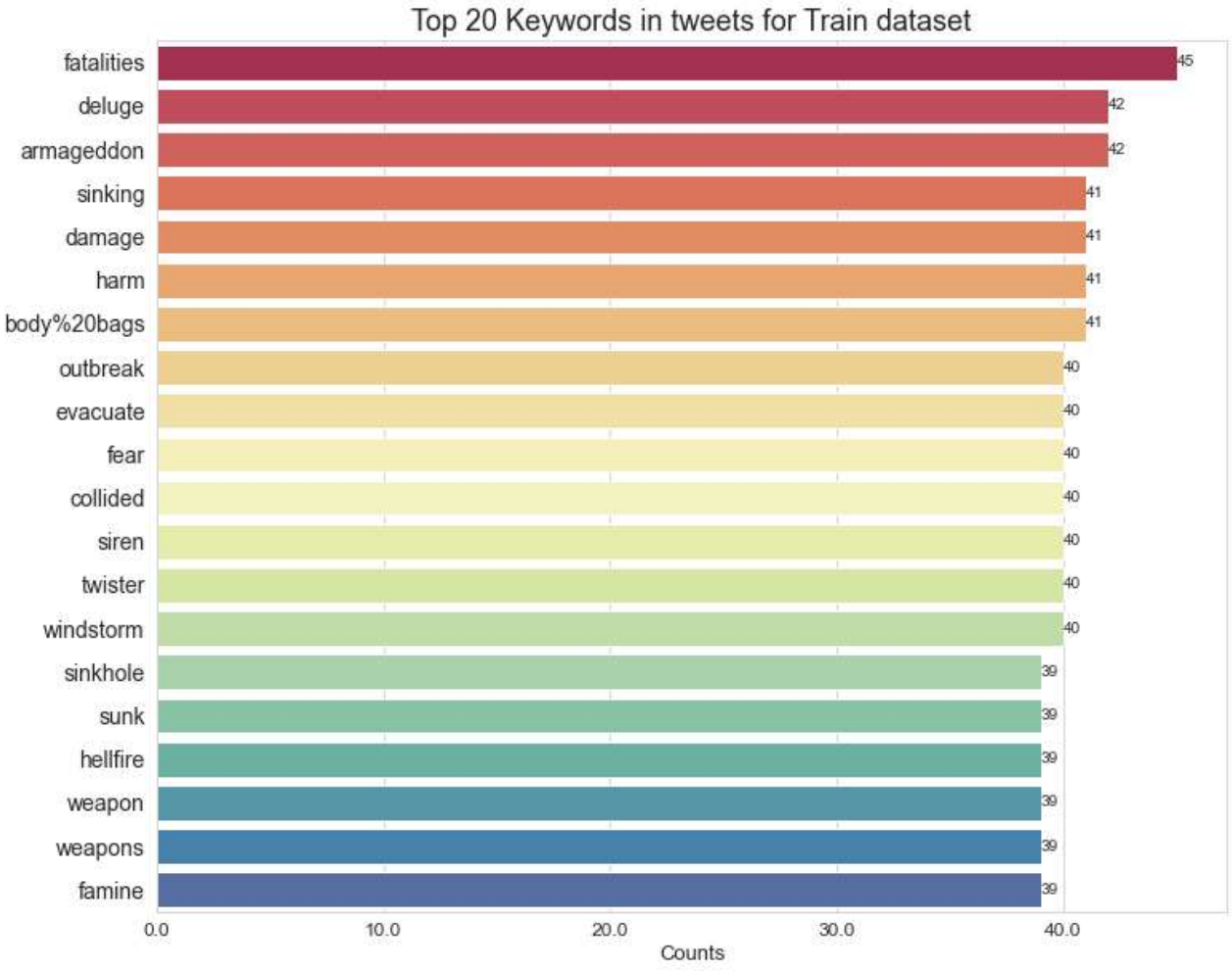
In [14]:

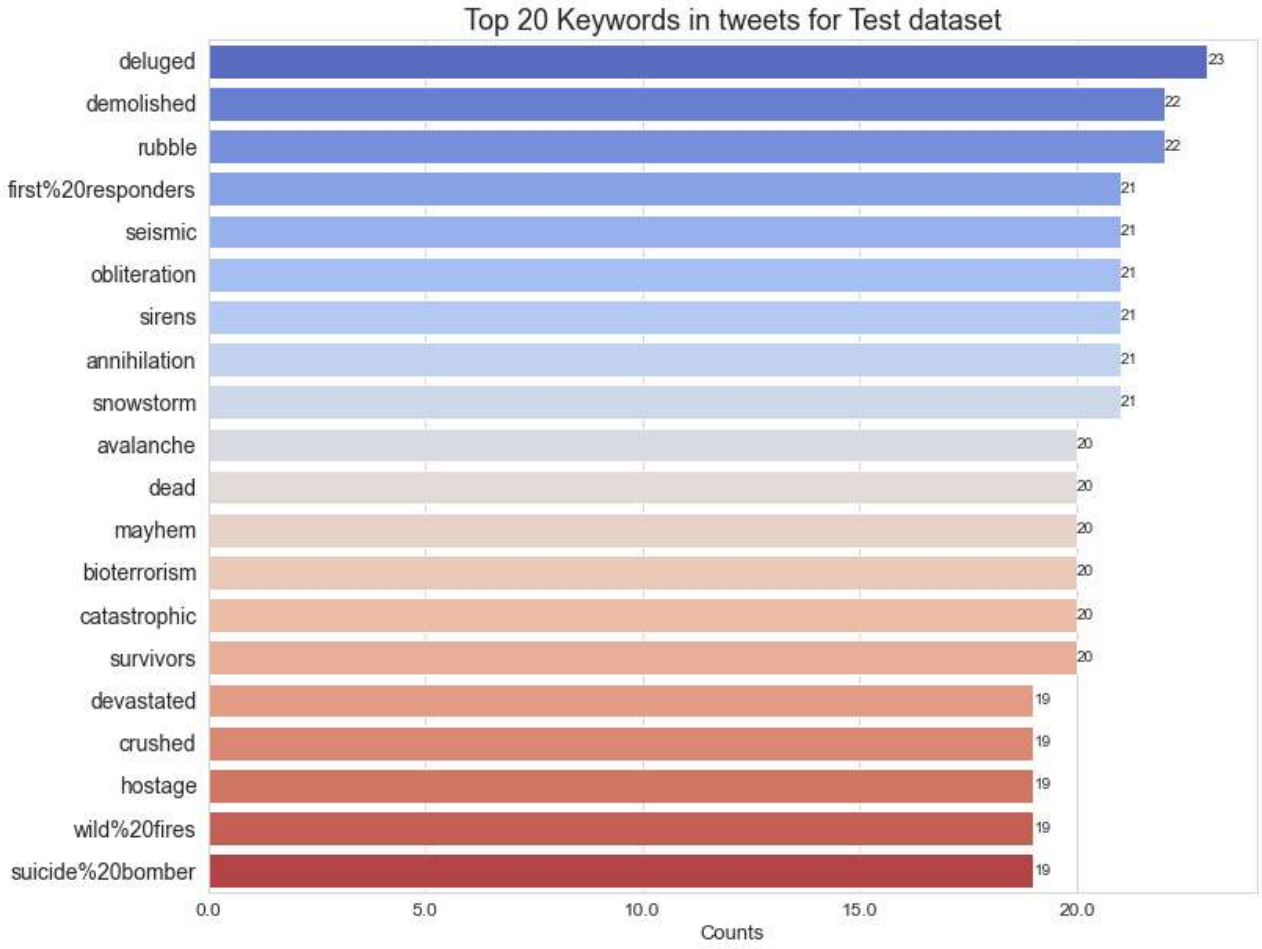
```
def top_tweet_keyword(df, dtype, color, ntop=20):
    sns.set_style('whitegrid')
    fig,axes = plt.subplots(figsize=(12,10))

    top_keywords=df['keyword'].value_counts()[:ntop]
    ax = sns.barplot(y=top_keywords.index,x=top_keywords,palette=color,data=df)
    for container in ax.containers:
        ax.bar_label(container)

    ax.set_xticklabels([str(i) for i in ax.get_xticks()], fontsize = 12)
    ax.set_yticklabels([t.get_text()  for t in ax.get_yticklabels()], fontsize = 14)
    plt.title('Top %d Keywords in tweets for %s dataset' %(ntop, dtype), fontsize=18)
    plt.xlabel('Counts', fontsize=13)
    plt.show()

top_tweet_keyword(train_df, 'Train', 'Spectral')
top_tweet_keyword(test_df, 'Test', 'coolwarm')
```





1.3.5. Top 20 locations tweets sent from

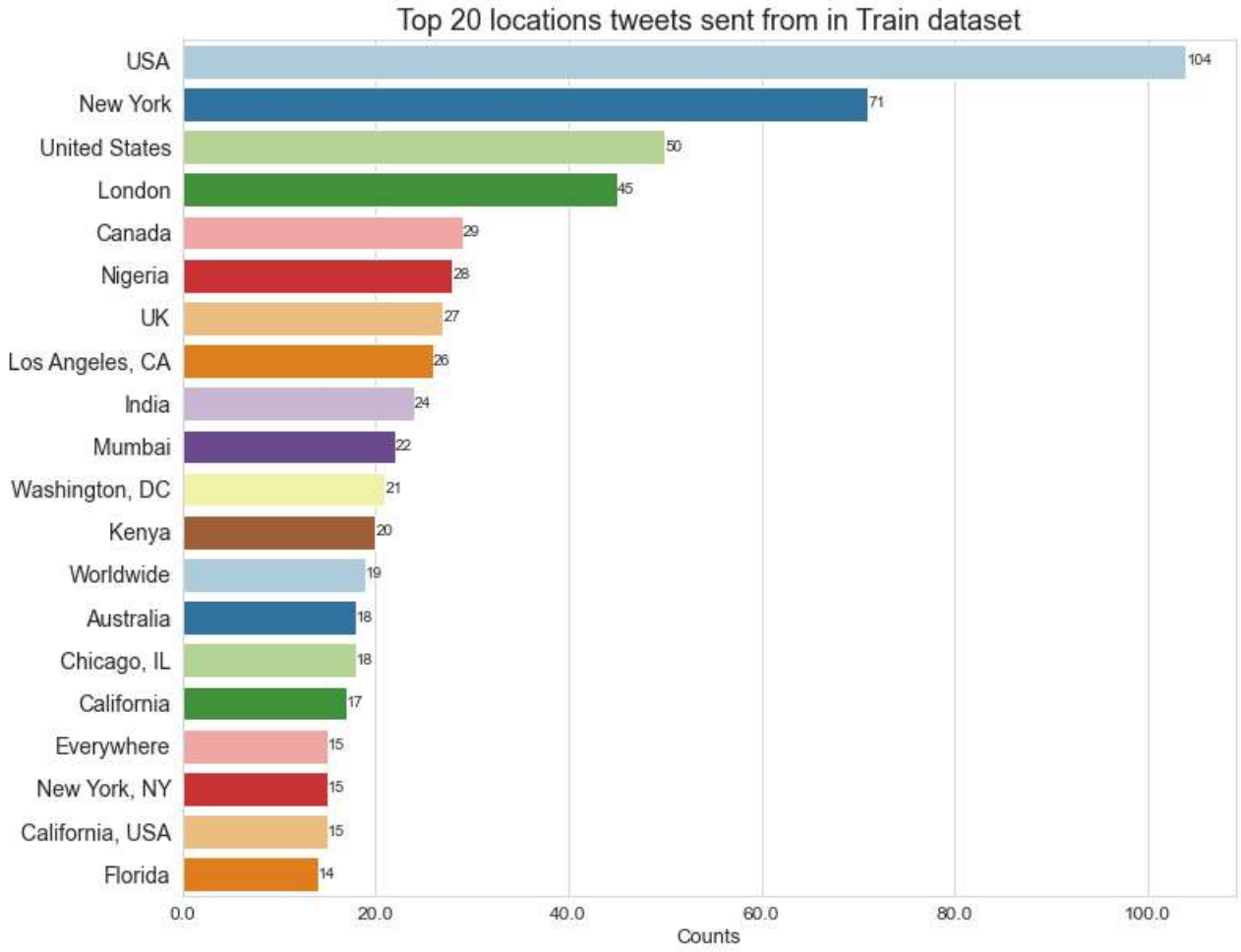
Plotting the top 20 locations tweets sent from on train and test datasets.

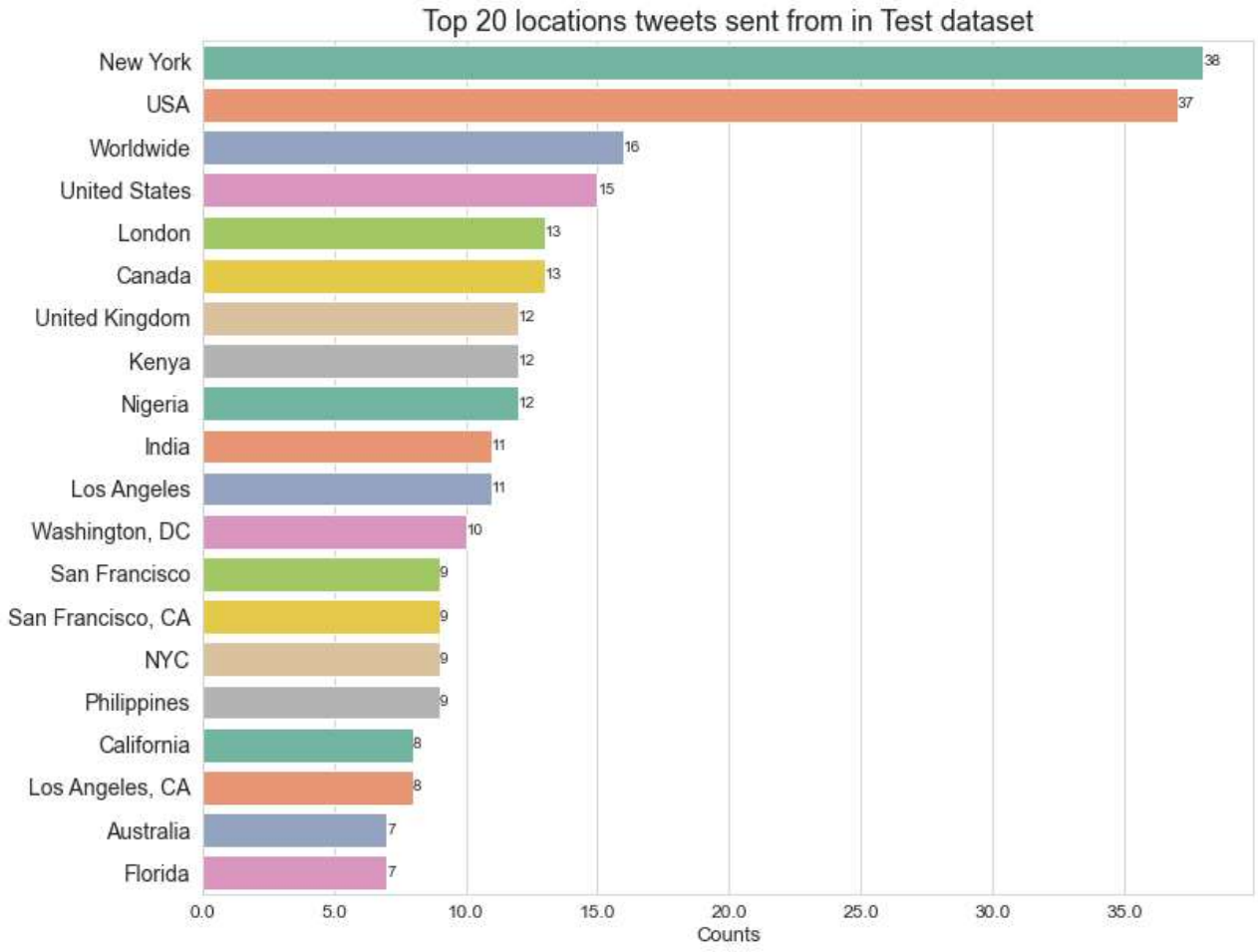
In [15]:

```
def top_tweet_location(df, dtype, color, ntop=20):
    sns.set_style('whitegrid')
    fig,axes = plt.subplots(figsize=(12,10))

    locations = df["location"].value_counts()
    ax = sns.barplot(y=locations[0:20].index,x=locations[0:20],palette=color,data=df)
    for container in ax.containers:
        ax.bar_label(container)
    ax.set_xticklabels([str(i) for i in ax.get_xticks()], fontsize = 12)
    ax.set_yticklabels([t.get_text()  for t in ax.get_yticklabels()], fontsize = 14)
    plt.title('Top %d locations tweets sent from in %s dataset' %(ntop, dtype),fontsize
    plt.xlabel('Counts', fontsize=13)
    plt.show()

top_tweet_location(train_df, 'Train', 'Paired')
top_tweet_location(test_df, 'Test', 'Set2')
```


Top 20 locations tweets sent from in Train dataset
USA
New York
United States
London
Canada
Nigeria
UK
Los Angeles, CA
India
Mumbai
Washington, DC
Kenya
Worldwide
Australia
Chicago, IL
California
Everywhere
New York, NY
California, USA
Florida
104
71
50
45
29
28
27
26
24
22
21
20
19
18
18
17
15
15
15
14
0.0
20.0
40.0
60.0
80.0
100.0
Counts


Top 20 locations tweets sent from in Test dataset
New York
USA
Worldwide
United States
London
Canada
United Kingdom
Kenya
Nigeria
India
Los Angeles
Washington, DC
San Francisco
San Francisco, CA
NYC
Philippines
California
Los Angeles, CA
Australia
Florida
38
37
16
15
13
13
12
12
12
11
11
10
9
9
9
9
8
8
7
7
0.0
5.0
10.0
15.0
20.0
25.0
30.0
35.0
Counts

The locations registered in the data can be countries, states, cities or worldwide. Most tweets are sent from the USA in train dataset. But most tweets are sent from New York in test dataset.

1.3.6. Distribution of tweets

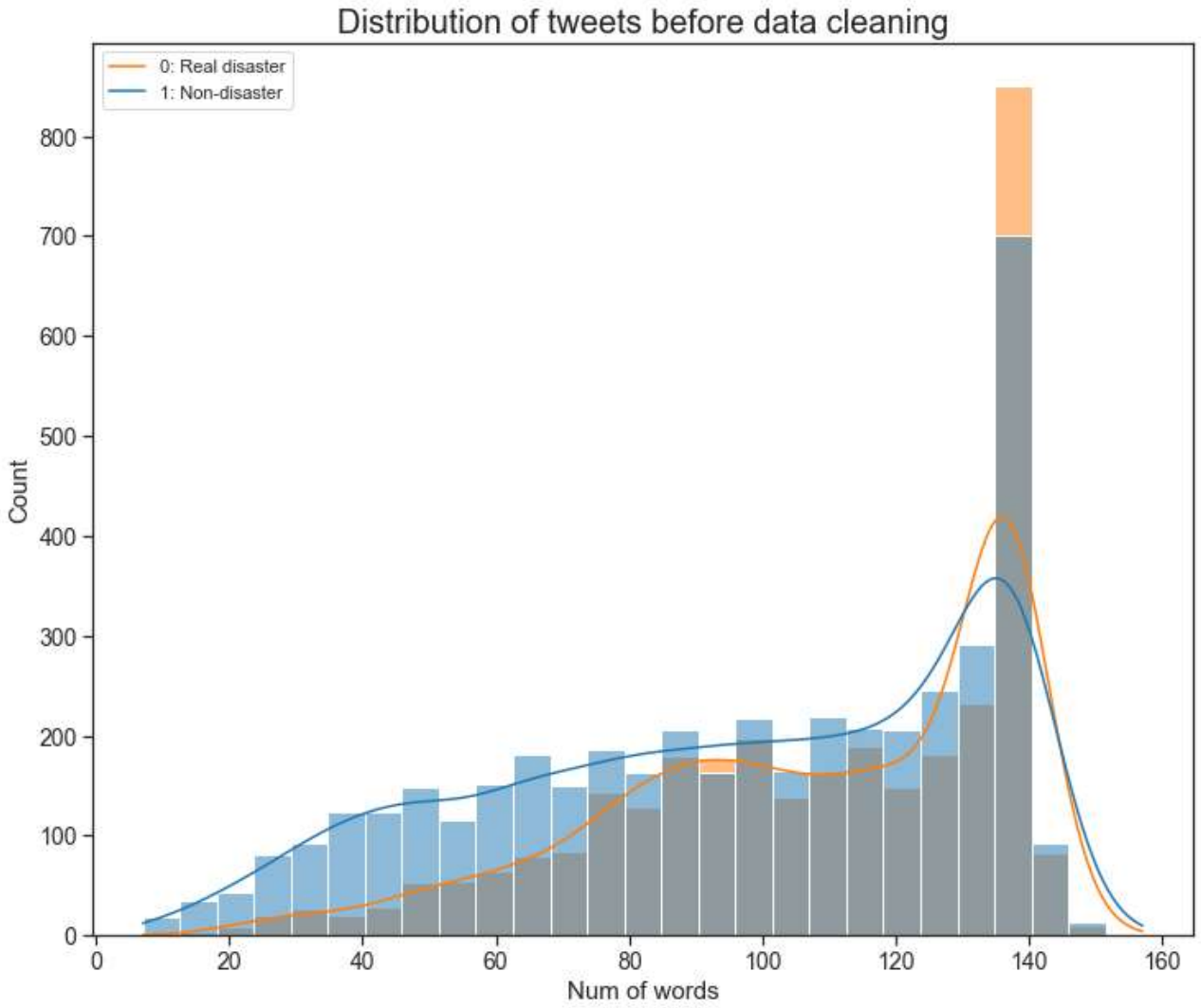
Plotting how the tweets are distrubuted before cleaning the data.

In [16]:

```
# Add tweets length column
train_df['numwords']  = train_df["text"].apply(len)
test_df['numwords']  = test_df["text"].apply(len)

# Plot histogram of tweets length
def dist_tweet(df, title, col, color):
    sns.set(rc={"figure.figsize":(12, 10)})
    sns.set_style("ticks")
    plot = sns.histplot(
        data = df,
        x = col,
        kde = True,
        hue = 'target',
        palette = color
        )
    plot.set_xlabel('Num of words', fontsize = 15)
    plot.set_ylabel('Count', fontsize = 15)
    plot.set_title(title, fontsize = 20)
    plt.tick_params(labelsize = 14)
    plt.legend(loc='upper left',labels = ['0: Real disaster', '1: Non-disaster'])
    plt.show()

dist_tweet(train_df, "Distribution of tweets before data cleaning", 'numwords', 'tab10'
```



From the above distribution, it can be seen that the most length of tweets lies between the range of 130 - 140 characters.

1.3.7. Top 3 tweets according to the length of tweets train dataset

Displaying the top 3 tweets on the train dataset.

In [17]:
```
for tweet in train_df.sort_values(by="numwords", ascending=False)["text"][:3]:
    print(f"Tweet: \n{tweet}\n")
```

```
Tweet:
when you're taking a shower and someone flushes the toilet and you have .1 second to GTF
O or you get burned?????????????????????????????????????????????????

Tweet:
It's was about 2:30 in the morning&amp;amp;I went downstairs to watch some telly&amp;amp;I accid
entally made a loud bang&amp;amp;my dad(who has a broken leg)walked-

Tweet:
@CAgov If 90BLKs&amp;amp;8WHTs colluded 2 take WHT F @USAgov AUTH Hostage&amp;amp;2 make her loo
k BLK w/Bioterrorism&amp;amp;use her lgl/org IDis ID still hers?@VP
```

1.4. Data Pre-processing

From the previous step, we can conclude from the data that there is no standard format on the tweets, there are many symbols on the tweets, there are several special characters on the tweets, and some tweets have URLs.

To deal with these problems, I am performing the following data cleaning:

- Removing punctuations.
- Removing numbers.
- Converting to lowercase.
- Removing stopwords.
- Removing html tags
- Removing URLS
- Removing special characters
- Lemmatization
- Removing single characters

In [18]:
```
def data_cleaning(tweet):
    # remove punctuations
    tweet = re.sub(r'[^\w\s]', '', tweet)

    # remove numbers
    tweet = re.sub(r'[0-9]', '', tweet)

    # converting to lowercase
    tweet = tweet.lower()

    # Stop word removal
    lemma = WordNetLemmatizer()
    tweet = " ".join(word for word in word_tokenize(tweet) if word not in stopwords.wor

    # removing html tags
    tweet = re.sub(r"&.*?;|<.*?>", " ", tweet)

    # url removal
    tweet = re.sub(r"https?://\S+|www\.\S+", " ", tweet)

    # non word removals (special chars)
    tweet = re.sub(r"[^a-z]", " ", tweet)

    # lemmatization
    tweet = " ".join(lemma.lemmatize(word) for word in word_tokenize(tweet))

    # Single char removal
    tweet = re.sub(r"\b\w\b", "", tweet).strip()

    return tweet
```

In [19]:
```
# apply data cleaning to the train and test datasets
train_df["cleaned_text"] = train_df["text"].apply(data_cleaning)
test_df["cleaned_text"] = test_df["text"].apply(data_cleaning)
```

In [20]:
```
# find the number of words after cleaning the data
```

```
train_df["cleaned_words"] = train_df["cleaned_text"].apply(lambda x: len(re.sub(r"[\W_]
```

In [21]:

```
train_df
```

Out[21]:

	id	keyword	location	text	target	numwords	cleaned_text	cleaned_words
0	1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...	1	69	deed reason earthquake may allah forgive	6
1	4	NaN	NaN	Forest fire near La Ronge Sask. Canada	1	38	forest fire near la ronge sask canada	7
2	5	NaN	NaN	All residents asked to 'shelter in place' are ...	1	133	resident asked shelter place notified officer ...	11
3	6	NaN	NaN	13,000 people receive #wildfires evacuation or...	1	65	people receive wildfire evacuation order calif...	6
4	7	NaN	NaN	Just got sent this photo from Ruby #Alaska as ...	1	88	got sent photo ruby alaska smoke wildfire pour...	9
...	...	...	...	...	...	...	...	...
7608	10869	NaN	NaN	Two giant cranes holding a bridge collapse int...	1	83	two giant crane holding bridge collapse nearby...	9
7609	10870	NaN	NaN	@aria_ahrary @TheTawniest The out of control w...	1	125	aria ahrary thetawniest control wild fire cali...	12
7610	10871	NaN	NaN	M1.94 [01:04 UTC]?5km S of Volcano Hawaii. htt...	1	65	utckm volcano hawaii httptcozdtoydebj	4
7611	10872	NaN	NaN	Police investigating after an e-bike collided ...	1	137	police investigating ebike collided car little...	14
7612	10873	NaN	NaN	The Latest: More Homes Razed by Northern Calif...	1	94	latest home razed northern california wildfire...	9

7613 rows × 8 columns

1.4.1. Handling null values

Removing irrelevant feature and filling missing values

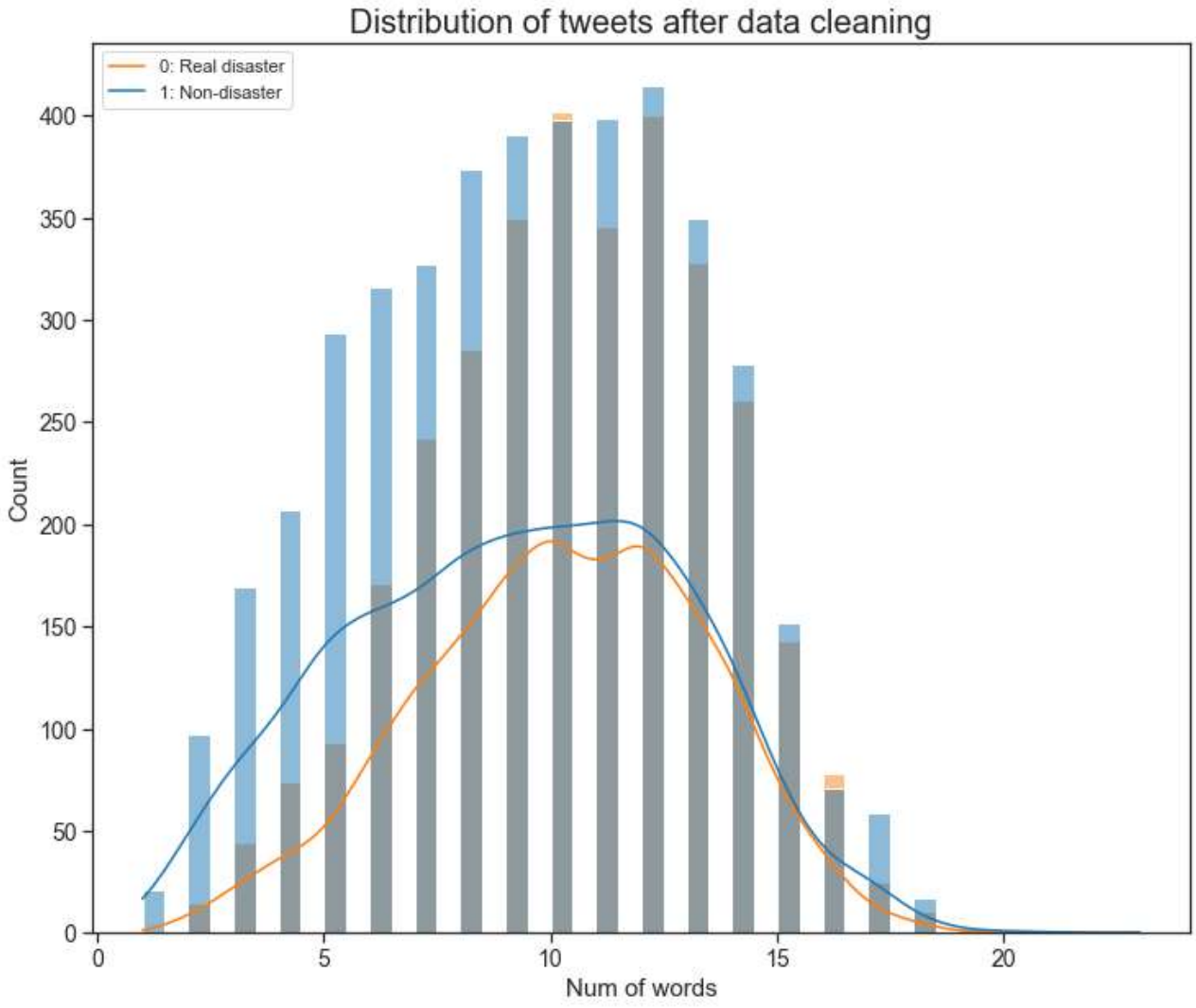
In [22]:
```
# Removing irrelevant feature
train_df.drop(["location","id"],axis=1)

# filling keword missing value
train_df['keyword'].replace({"Na":'notkey'},inplace=True)
```

1.4.2. Distribution of tweets after data cleaning

Plotting how the tweets are distrubuted after cleaning the data.

In [23]:
```
dist_tweet(train_df, "Distribution of tweets after data cleaning", 'cleaned_words', 'ta
```



1.4.3. Wordcloud generation

Highlighting essential textual data points on the cleaned text feature.

In [24]:
```
text1=[]
```

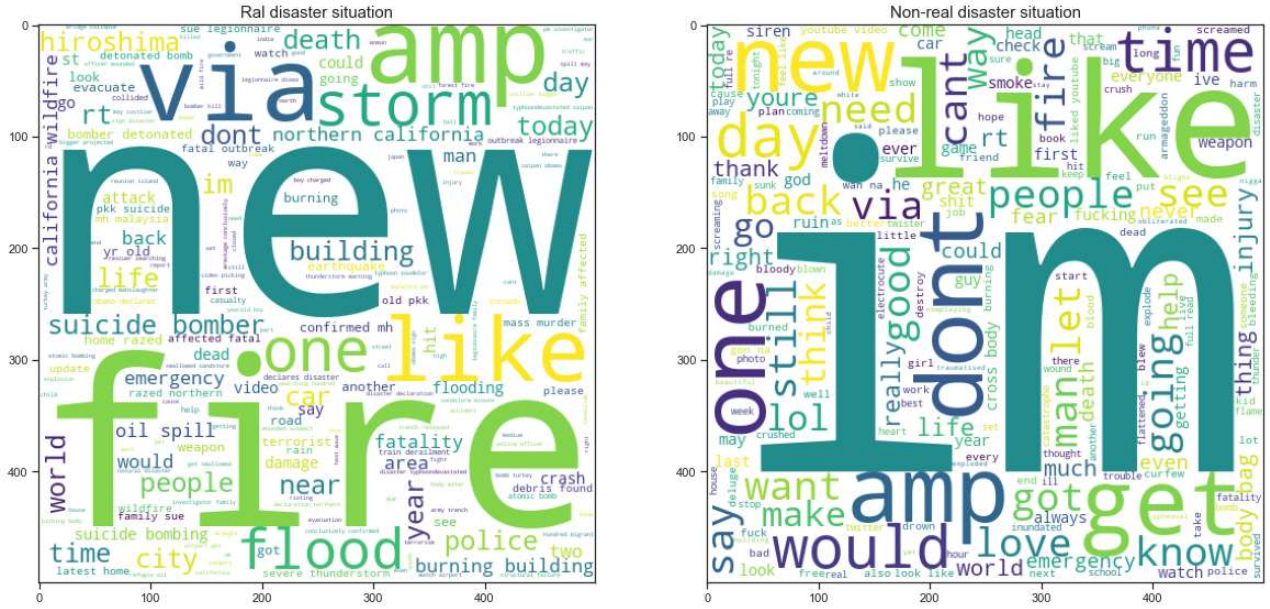
```
for i in train_df.loc[train_df['target']==1,'cleaned_text'].dropna():
      for j in i.lower().strip().split():
            text1.append(j)

text0=[]
for i in train_df.loc[train_df['target']==0,'cleaned_text'].dropna():
      for j in i.lower().strip().split():
            text0.append(j)

# Word cloud of clean word has real disaster situation
plt.figure(figsize=(20,20))
plt.subplot(1,2,1)
cloud1=WordCloud(background_color="white",max_font_size=400,width=500,height=500,stopwo
cloud1.generate(' '.join(text1))
plt.title("Ral disaster situation",size=15)
plt.imshow(cloud1)


# Word cloud of keyword has non-disaster Sitution
plt.subplot(1,2,2)
cloud1=WordCloud(background_color="white",max_font_size=400,width=500,height=500,stopwo
cloud1.generate(' '.join(text0))
plt.title("Non-real disaster situation",size=15)
plt.imshow(cloud1)
```

Out[24]: <matplotlib.image.AxesImage at 0x25ca11d1220>



2. Training, modeling and prediction

In this section, I am going to use LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Unit) models. I considered them because they were explained on the instructor's lecture for this week. LTSM was initially used to solve NLP problems and GRU is the improvement for LSTM which uses two gates instead of the three gates used in LSTM.

2.1. Token and vocabulary creation

Processing the raw text for the NLP.

In [25]:
```
max_features=3000
tokenizer=Tokenizer(num_words=max_features,split=' ')
tokenizer.fit_on_texts(train_df['cleaned_text'].values)
X = tokenizer.texts_to_sequences(train_df['cleaned_text'].values)
X = pad_sequences(X)
y = train_df['target']
```

2.2. Train and test split

Splitting the data set into two pieces: a training set and a testing set. The test_size is given as 0.2 , it means 20% of our data goes into our test size. 1-test_size is our train size.

In [26]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2, random_state
```

2.3. LSTM model

These are the layers added for the model creation:

- Embedding layer: I used the one available in Keras library because it is faster and gets some context.
- LSTM layer: I used LSTM (Long Short-Term Memoery) because it is used to solve NLP problems although they have few issues.
- Dropout layers: I used dropout layer for prevention agains overfitting.
- Dense layers: I used dense layer to take the input from all the other neurons of the previous layer.

In [27]:
```
model1 = Sequential()
model1.add(Embedding(max_features, 32,input_length = X.shape[1]))
model1.add(Dropout(0.2))
model1.add(LSTM(32, dropout=0.2, recurrent_dropout=0.4))
model1.add(Dense(1,activation='sigmoid'))
adam = optimizers.Adam(learning_rate=0.002)
model1.compile(loss = 'binary_crossentropy', optimizer=adam ,metrics = ['accuracy'])
print(model1.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 19, 32)            96000

 dropout (Dropout)           (None, 19, 32)            0

 lstm (LSTM)                 (None, 32)                8320

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 104,353
Trainable params: 104,353
```

```
Non-trainable params: 0
_________________________________________________________________
None
```

Measuring how well the LSTM model performs with the train dataset.

In [28]:
```
callbacks = [ ReduceLROnPlateau(monitor='val_loss', patience=5, cooldown=0),
              EarlyStopping(monitor='val_accuracy', min_delta=1e-4, patience=5)]

history1 = model1.fit(X_train, y_train, epochs = 10, batch_size=32, validation_data=(X_
```

```
Epoch 1/10
191/191 [==============================] - 4s 11ms/step - loss: 0.5365 - accuracy: 0.726
1 - val_loss: 0.4169 - val_accuracy: 0.8155 - lr: 0.0020
Epoch 2/10
191/191 [==============================] - 2s 10ms/step - loss: 0.3704 - accuracy: 0.841
1 - val_loss: 0.4434 - val_accuracy: 0.8056 - lr: 0.0020
Epoch 3/10
191/191 [==============================] - 2s 10ms/step - loss: 0.3181 - accuracy: 0.866
3 - val_loss: 0.4773 - val_accuracy: 0.7958 - lr: 0.0020
Epoch 4/10
191/191 [==============================] - 2s 10ms/step - loss: 0.2838 - accuracy: 0.881
0 - val_loss: 0.5134 - val_accuracy: 0.7912 - lr: 0.0020
Epoch 5/10
191/191 [==============================] - 2s 10ms/step - loss: 0.2491 - accuracy: 0.895
2 - val_loss: 0.5822 - val_accuracy: 0.7873 - lr: 0.0020
Epoch 6/10
191/191 [==============================] - 2s 10ms/step - loss: 0.2174 - accuracy: 0.909
2 - val_loss: 0.7157 - val_accuracy: 0.7774 - lr: 0.0020
```

2.4. GRU

These are the layers added for the model creation:

- Embedding layer: I used the one available in Keras library because it is faster and gets some context.
- GRU layer: I used GRU (Gated Recurrent Unit) because it is the improvement of LSTM model.
- Dropout layers: I used dropout layer for prevention agains overfitting.
- Dense layers: I used dense layer to take the input from all the other neurons of the previous layer.

In [29]:
```
model2 = Sequential()
model2.add(Embedding(max_features, 32,input_length = X.shape[1]))
model2.add(Dropout(0.2))
model2.add(GRU(128))
model2.add(Dense(20, activation='ReLU'))
model2.add(Dense(1, activation='sigmoid'))
adam = optimizers.Adam(learning_rate=0.002)
model2.compile(loss = 'binary_crossentropy', optimizer=adam ,metrics = ['accuracy'])
print(model2.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 19, 32)            96000
```

```
 dropout_1 (Dropout)         (None, 19, 32)            0

 gru (GRU)                   (None, 128)               62208

 dense_1 (Dense)             (None, 20)                2580

 dense_2 (Dense)             (None, 1)                 21

=================================================================
Total params: 160,809
Trainable params: 160,809
Non-trainable params: 0
_________________________________________________________________
None
```

In [30]:
```
history2 = model2.fit(X_train, y_train, epochs = 10, batch_size=32, validation_data=(X_
```

```
Epoch 1/10
191/191 [==============================] - 3s 10ms/step - loss: 0.5297 - accuracy: 0.737
4 - val_loss: 0.4463 - val_accuracy: 0.7925 - lr: 0.0020
Epoch 2/10
191/191 [==============================] - 2s 9ms/step - loss: 0.3641 - accuracy: 0.8461
- val_loss: 0.4556 - val_accuracy: 0.8096 - lr: 0.0020
Epoch 3/10
191/191 [==============================] - 2s 8ms/step - loss: 0.2922 - accuracy: 0.8780
- val_loss: 0.5102 - val_accuracy: 0.7899 - lr: 0.0020
Epoch 4/10
191/191 [==============================] - 2s 8ms/step - loss: 0.2393 - accuracy: 0.9026
- val_loss: 0.6111 - val_accuracy: 0.7794 - lr: 0.0020
Epoch 5/10
191/191 [==============================] - 2s 8ms/step - loss: 0.1960 - accuracy: 0.9235
- val_loss: 0.6956 - val_accuracy: 0.7676 - lr: 0.0020
Epoch 6/10
191/191 [==============================] - 2s 8ms/step - loss: 0.1583 - accuracy: 0.9345
- val_loss: 0.7864 - val_accuracy: 0.7623 - lr: 0.0020
Epoch 7/10
191/191 [==============================] - 2s 9ms/step - loss: 0.1056 - accuracy: 0.9558
- val_loss: 0.9414 - val_accuracy: 0.7708 - lr: 2.0000e-04
```

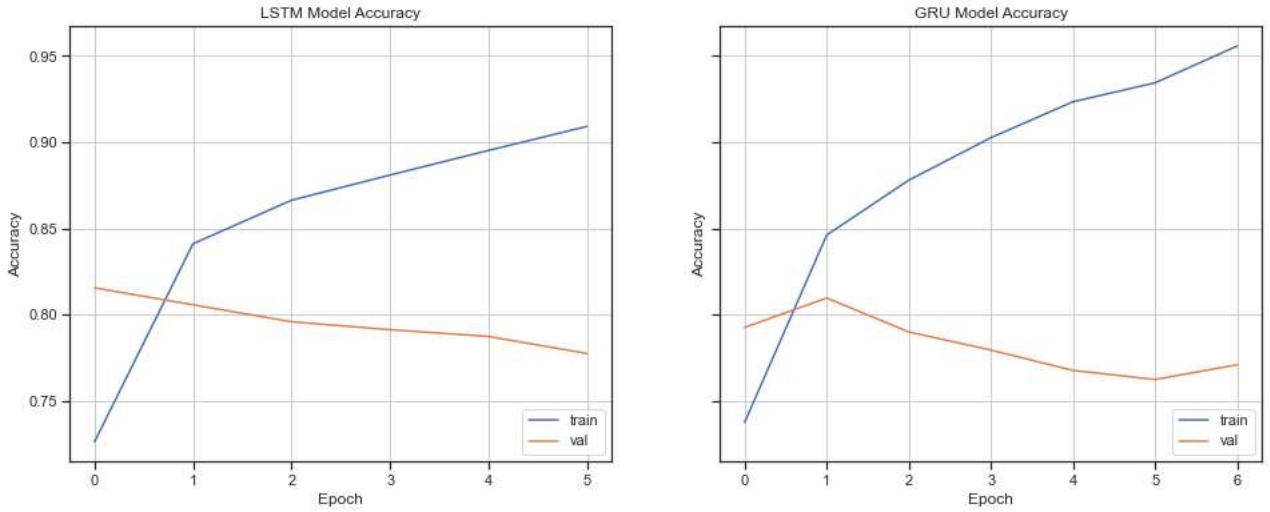
2.5. Comparison between LSTM and GRU models

Comparing the accuracy and val accuracy between the LSTM and GRU models.

In [31]:
```
fig, axs = plt.subplots(1, 2, figsize=(16, 6), sharey=True)
axs[0].plot(history1.history['accuracy'], label='train')
axs[0].plot(history1.history['val_accuracy'], label = 'val')
axs[0].set_title('LSTM Model Accuracy')
axs[0].set_xlabel('Epoch')
axs[0].set_ylabel('Accuracy')
axs[0].legend(loc='lower right')

axs[1].plot(history2.history['accuracy'], label='train')
axs[1].plot(history2.history['val_accuracy'], label = 'val')
axs[1].set_title('GRU Model Accuracy')
axs[1].set_xlabel('Epoch')
axs[1].set_ylabel('Accuracy')
axs[1].legend(loc='lower right')
```

```
axs[0].grid()
axs[1].grid()
```



By comparing the charts, it can be observed that there is a slightly improvement using the GRU.

2.6. Tuning hyperparameters

As GRU model had higher accuracy than LSTM, this model will be used for the prediction. But before using it, I will modify the following hyperparameters:

- Number of epochs: This hyperparameter sets how many complete iterations of the dataset is to be run.
- Batch size: This hyperparameter defines the number of samples to work on before the internal parameters of the model are updated.

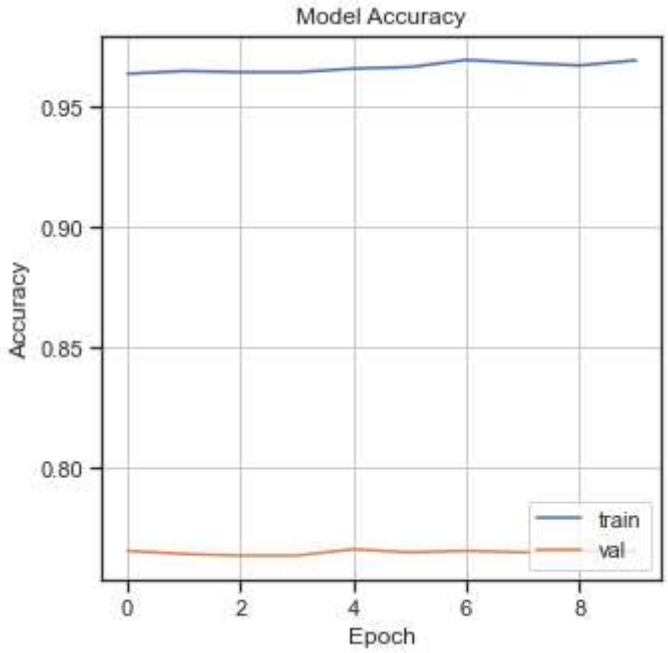
In [32]:
```
# Ingreasing the epochs to 20
# Increasing the batch_size to 64
history3 = model2.fit(X_train, y_train, epochs = 20, batch_size=64, validation_data=(X_
```

```
Epoch 1/20
96/96 [==============================] - 1s 15ms/step - loss: 0.0932 - accuracy: 0.9634
- val_loss: 0.9751 - val_accuracy: 0.7656 - lr: 2.0000e-04
Epoch 2/20
96/96 [==============================] - 1s 14ms/step - loss: 0.0867 - accuracy: 0.9645
- val_loss: 1.0040 - val_accuracy: 0.7643 - lr: 2.0000e-04
Epoch 3/20
96/96 [==============================] - 1s 14ms/step - loss: 0.0854 - accuracy: 0.9640
- val_loss: 1.0458 - val_accuracy: 0.7636 - lr: 2.0000e-04
Epoch 4/20
96/96 [==============================] - 1s 14ms/step - loss: 0.0810 - accuracy: 0.9640
- val_loss: 1.0972 - val_accuracy: 0.7636 - lr: 2.0000e-04
Epoch 5/20
96/96 [==============================] - 1s 13ms/step - loss: 0.0785 - accuracy: 0.9655
- val_loss: 1.1268 - val_accuracy: 0.7663 - lr: 2.0000e-04
Epoch 6/20
96/96 [==============================] - 1s 13ms/step - loss: 0.0793 - accuracy: 0.9662
- val_loss: 1.1570 - val_accuracy: 0.7649 - lr: 2.0000e-04
Epoch 7/20
96/96 [==============================] - 1s 13ms/step - loss: 0.0727 - accuracy: 0.9691
```

```
- val_loss: 1.1676 - val_accuracy: 0.7656 - lr: 2.0000e-05
Epoch 8/20
96/96 [==============================] - 1s 13ms/step - loss: 0.0724 - accuracy: 0.9678
- val_loss: 1.1701 - val_accuracy: 0.7649 - lr: 2.0000e-05
Epoch 9/20
96/96 [==============================] - 1s 13ms/step - loss: 0.0743 - accuracy: 0.9668
- val_loss: 1.1747 - val_accuracy: 0.7656 - lr: 2.0000e-05
Epoch 10/20
96/96 [==============================] - 1s 12ms/step - loss: 0.0711 - accuracy: 0.9690
- val_loss: 1.1745 - val_accuracy: 0.7656 - lr: 2.0000e-05
```

In [33]:

```
plt.figure(figsize=(5, 5))
plt.grid()
plt.plot(history3.history['accuracy'], label='train')
plt.plot(history3.history['val_accuracy'], label = 'val')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()
```



By tuning the hyperparameters, we can see an increasing on the accuracy but the validation accuracy didn't increase too much.

2.5. Prediction

Predicting the model on the test dataset to get the results.

In [34]:

```
# Tokenization on full train dataset
max_features=5000
tokenizer=Tokenizer(num_words=max_features,split=' ')
tokenizer.fit_on_texts(train_df['cleaned_text'].values)
X = tokenizer.texts_to_sequences(train_df['cleaned_text'].values)
X = pad_sequences(X, maxlen = 50)
```

In [35]:
```
# Tokenixation on full test dataset
tokenizer.fit_on_texts(test_df['cleaned_text'].values)
test_token = tokenizer.texts_to_sequences(test_df['cleaned_text'].values)
test_token = pad_sequences(test_token, maxlen = 50)
```

In [36]:
```
# Model call with little bit of optimisation based on previous results
lstm_out = 100
model = Sequential()
model.add(Embedding(max_features, 100, input_length = X.shape[1]))
model.add(Dropout(0.2))
model.add(GRU(128))
model.add(Dense(20, activation='ReLU'))
model.add(Dense(1,activation='sigmoid'))
adam = optimizers.Adam(learning_rate=2e-3)
model.compile(loss = 'binary_crossentropy', optimizer=adam ,metrics = ['accuracy'])
print(model.summary())
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 50, 100)           500000

 dropout_2 (Dropout)         (None, 50, 100)           0

 gru_1 (GRU)                 (None, 128)               88320

 dense_3 (Dense)             (None, 20)                2580

 dense_4 (Dense)             (None, 1)                 21

=================================================================
Total params: 590,921
Trainable params: 590,921
Non-trainable params: 0
_________________________________________________________________
None
```

In [37]:
```
modelsub = model.fit(X,y, epochs = 20,validation_split = 0.2 ,callbacks=[callbacks], ba
```

```
Epoch 1/20
96/96 [==============================] - 6s 44ms/step - loss: 0.5332 - accuracy: 0.7309
- val_loss: 0.4672 - val_accuracy: 0.7886 - lr: 0.0020
Epoch 2/20
96/96 [==============================] - 4s 41ms/step - loss: 0.3345 - accuracy: 0.8591
- val_loss: 0.5069 - val_accuracy: 0.7518 - lr: 0.0020
Epoch 3/20
96/96 [==============================] - 4s 41ms/step - loss: 0.2459 - accuracy: 0.9011
- val_loss: 0.5744 - val_accuracy: 0.7367 - lr: 0.0020
Epoch 4/20
96/96 [==============================] - 4s 42ms/step - loss: 0.1792 - accuracy: 0.9312
- val_loss: 0.8083 - val_accuracy: 0.7367 - lr: 0.0020
Epoch 5/20
96/96 [==============================] - 4s 41ms/step - loss: 0.1374 - accuracy: 0.9468
- val_loss: 0.8581 - val_accuracy: 0.7282 - lr: 0.0020
Epoch 6/20
96/96 [==============================] - 4s 41ms/step - loss: 0.1046 - accuracy: 0.9578
- val_loss: 0.9519 - val_accuracy: 0.7249 - lr: 0.0020
```

In [38]:
```
y_hat = model.predict(test_token).round()
submission_df['target'] = np.round(y_hat).astype('int')
submission_df.to_csv('submission.csv', index=False)
submission_df.describe()
```

```
102/102 [==============================] - 1s 5ms/step
```

Out[38]:

	id	target
count	3263.000000	3263.000000
mean	5427.152927	0.431811
std	3146.427221	0.495404
min	0.000000	0.000000
25%	2683.000000	0.000000
50%	5500.000000	0.000000
75%	8176.000000	1.000000
max	10875.000000	1.000000

3. Discussion

3.1. Conclusion

The LSTM model got an accuracy of 90% but in the Kaggle leaderboard I got 0.52. Thus, I decided to compare with GRU model. From the comparison, GRU scored slightly better than LSTM. Then, I tuned the following hyperparameters from the GRU model. Those hyperparameters were number of epochs and batch size. This helped me to score 0.01 more than the previous score on the Kaggle leaderboard.

3.2. Further improvement

Future improvement will be add more layers and play with different activation functions such as softmax. I also would like to test on changing the learning rate by using values between 0 and 1. In addition, I would also like to test other deep learning algorithms to solve NLP problems.

In [ ]:
```
```