Data Wrangling

Import all the necessary libraries

```
import pandas as pd
import re
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
import matplotlib.pyplot as plt
#import nltk
#nltk.download('wordnet')
import regex
import spacy
from collections import Counter
from sklearn.model_selection import train_test_split
import numpy as np

df = pd.read_csv('tweets.csv')
```

```
df.head()
```

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativereason_confidence	airline	airline_sentiment_gold	name
0	570306133677760513	neutral	1.0000	NaN	NaN	Virgin America	NaN	cairdin
1	570301130888122368	positive	0.3486	NaN	0.0000	Virgin America	NaN	jnardino
2	570301083672813571	neutral	0.6837	NaN	NaN	Virgin America	NaN	yvonnalynn
3	570301031407624196	negative	1.0000	Bad Flight	0.7033	Virgin America	NaN	jnardino
4	570300817074462722	negative	1.0000	Can't Tell	1.0000	Virgin America	NaN	jnardino

We see that tweet_id and retweet_ount are integers, airline_sentiment_confidence and negative_reason_confidence are floats while the rest of them are strings (objects). Also, user_timezone column has only 9820 records therefore missing some records.

In [4]:
```python
df['user_timezone'] = df['user_timezone'].fillna(method='ffill')
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14640 entries, 0 to 14639
Data columns (total 15 columns):
tweet_id                        14640 non-null int64
airline_sentiment               14640 non-null object
airline_sentiment_confidence    14640 non-null float64
negativereason                  9178 non-null object
negativereason_confidence       10522 non-null float64
airline                         14640 non-null object
airline_sentiment_gold          40 non-null object
name                            14640 non-null object
negativereason_gold             32 non-null object
retweet_count                   14640 non-null int64
text                            14640 non-null object
tweet_coord                     1019 non-null object
tweet_created                   14640 non-null object
tweet_location                  9907 non-null object
user_timezone                   14640 non-null object
dtypes: float64(2), int64(2), object(11)
memory usage: 1.7+ MB
None
```

The missing values in user_timezone is filled by 'forward fill' method of 'fillna' function.

Sometimes we tend to use contractions in our language to convey the message which is easier than typing the whole word out. For example, instead of " we will" we might type it out as " we'll ", which is commonly referred to as the texting language. However, this makes it harder for the classifier to determine and analyze the sentiment. Hence I have come up with a map of some common contractions and have written a customized function 'expand' to expand the words if it come across any of them listed in the map. This helps in improving classifier's efficiency.

```
contraction_dict = {"ain't": "is not", "aren't": "are not","can't": "cannot", "'cause": "because",
        "could've": "could have", "couldn't": "could not", "didn't": "did not",
        "doesn't": "does not", "don't": "do not", "hadn't": "had not", "hasn't": "has not",
        "haven't": "have not", "he'd": "he would","he'll": "he will", "he's": "he is",
        "how'd": "how did", "how'd'y": "how do you", "how'll": "how will", "how's": "how is",
        "I'd": "I would", "I'd've": "I would have", "I'll": "I will",
        "I'll've": "I will have","I'm": "I am", "I've": "I have", "i'd": "i would",
        "i'd've": "i would have", "i'll": "i will",  "i'll've": "i will have","i'm": "i am",
        "i've": "i have", "isn't": "is not", "it'd": "it would", "it'd've": "it would have",
        "it'll": "it will", "it'll've": "it will have","it's": "it is", "let's": "let us",
        "ma'am": "madam", "mayn't": "may not", "might've": "might have","mightn't": "might not",
        "mightn't've": "might not have", "must've": "must have", "mustn't": "must not",
        "mustn't've": "must not have", "needn't": "need not", "needn't've": "need not have",
        "o'clock": "of the clock", "oughtn't": "ought not", "oughtn't've": "ought not have",
        "shan't": "shall not", "sha'n't": "shall not", "shan't've": "shall not have", "she'd":
        "she would", "she'd've": "she would have", "she'll": "she will", "she'll've": "she will have",
        "she's": "she is", "should've": "should have", "shouldn't": "should not",
        "shouldn't've": "should not have", "so've": "so have","so's": "so as", "this's": "this is",
        "that'd": "that would", "that'd've": "that would have", "that's": "that is",
        "there'd": "there would", "there'd've": "there would have", "there's": "there is",
        "here's": "here is","they'd": "they would", "they'd've": "they would have",
        "they'll": "they will", "they'll've": "they will have", "they're": "they are",
```

"they've": "they have", "to've": "to have", "wasn't": "was not", "we'd": "we would",
"we'd've": "we would have", "we'll": "we will", "we'll've": "we will have", "we're": "we are",
"we've": "we have", "weren't": "were not", "what'll": "what will", "what'll've": "what will have",
"what're": "what are", "what's": "what is", "what've": "what have", "when's": "when is",
"when've": "when have", "where'd": "where did", "where's": "where is", "where've": "where have",
"who'll": "who will", "who'll've": "who will have", "who's": "who is", "who've": "who have",
"why's": "why is", "why've": "why have", "will've": "will have", "won't": "will not",
"won't've": "will not have", "would've": "would have", "wouldn't": "would not",
"wouldn't've": "would not have", "y'all": "you all", "y'all'd": "you all would",
"y'all'd've": "you all would have","y'all're": "you all are","y'all've": "you all have",
"you'd": "you would", "you'd've": "you would have", "you'll": "you will",
"you'll've": "you will have", "you're": "you are", "you've": "you have"}

In [6]:
```
def expand(text):
    x = ''
    for t in text.split():
        if t in contraction_dict.keys():
            t = contraction_dict[t]
        x = x + ' ' + t
    return x
```

In [7]:
```
def tweet_to_words(raw_tweet):
    tweets = expand(raw_tweet)
    tweets = re.sub(r'(\w)(\1{2,})', r'\1',tweets)
    letters_only = re.sub("[^a-zA-Z]", " ",tweets)
    words = letters_only.lower().split()
    stops = set(stopwords.words("english"))
    meaningful_words = [w for w in words if not w in stops]
    meaningful_words = [word for word in meaningful_words if len(word) > 2]
    meaningful_words = [word for word in meaningful_words
                        if 'http' not in word
                            and 'www' not in word]

    lemmatizer = WordNetLemmatizer()
    meaningful_words = [lemmatizer.lemmatize(word) for word in meaningful_words]

    return( " ".join( meaningful_words ))
```

In [10]:
```
df_downsampled['cleaned_text'] = df_downsampled.text.apply(lambda tweets: tweet_to_words(tweets))
```

The function 'tweet_to_words' first expands the contractions, gets rid of words that have multiple repeating characters in them such as the word 'loooove' basically is 'love' and the user is trying to convey the degree of intensity by putting in multiple repeating characters. Then it gets rid of symbols, punctuations and numbers, stop words, words that have 'http' or 'www' in them, words whose lenghths are greater than 2. Finally the words are lemmatized which means that it converts all of them to their root words, for example, the word 'caring' has the lemmatized(root) word 'care'.

Converting the categorical column (sentiment) to integer values.

```
df_downsampled['sentiment'] = df_downsampled['airline_sentiment'].apply(lambda sentiment:
                                    0 if sentiment == 'negative'
                                    else (1 if sentiment == 'neutral'
                                    else 2))
```