1. Background, Data Preparation, and Data Cleaning

Group: Group 11 - Alex Fung, Patrick Osborne
Dataset: Twitter US Airline Sentiment
Dataset link: https://www.kaggle.com/crowdflower/twitter-airline-sentiment (https://www.kaggle.com/crowdflower/twitter-airline-sentiment)

Background

For our course project we have chosen to conduct a sentiment analysis on a data set containing approximately 14.5 thousand tweets pertaining to 6 major US airlines. Going into this project we knew that we were interested in choosing a data set and problem that closely aligned to solving a topical business problem. We also wanted to pick a data set that would offer a certain degree of challenge and learning opportunities.

Our chosen data set aligns well with these goals for several reasons. Firstly, in the dozen or so years that Twitter has existed it has contributed to a significant shift in the way that companies interact with their customers, primarily from a customer service point of view, but additionally in terms of PR, marketing and even logistics. One tweet from the right (or wrong) person can set off a landslide of responses that can quickly become out of control. This is a particular concern for the US Airline industry. Over the past several years there have been several high-profile incidents causing negative public sentiment towards US airlines. We believe that a robust sentiment analysis model – specifically focused on identifying these negative sentiments before they become a larger problem - could help airlines better manage their PR crisis response and customer service. A sentiment analysis model tuned to identify negative tweets with a high degree of accuracy could help airlines analyse and learn from past Twitter trends and to rapidly identify new ones as they are occurring.

Secondly, from the perspective of data science students a Twitter data offers an interesting challenge in terms of cleaning, interpretation and prediction. As Twitter caps each message at a short character limit, complete English is rarely used. The data set is full of abbreviations, slang, emojis and Twitter functions such as hashtags, mentions and re-tweets. Cleaning these out while retaining the information in the original tweet will be important to developing an effective model. As with many customer service data sets, this one is also likely to be unbalanced towards the negative sentiment side. Though this aligns well with our goal to primarily identify negative sentiment tweets, this will be important to consider as we clean the data.

Data Preparation

The dataset downloaded manually from Kaggle contained a CSV and a Sqlite database file. Both files contain the same dataset, albeit in a different format. We chose to load the data using the CSV file because the dataset was not large enough to cause issues loading issues with the Pandas library. Also, the dataset had to be loaded with `UTF-8` encoding because we wanted to retain the information from emojis, which would be lost if we used another encoding, such as `CP-1252` We checked for nulls in `airline_sentiment` , `text` , and `negativereason` columns, although the dataset provided had no issues with nulls.

Below are links to the following Data Preparation Steps:

Load the data

In [1]:
```
import pandas as pd
pd.options.display.max_colwidth = None
```

It is important that we read the Tweets.csv with 'utf-8' encoding, so that we can extract the emojis properly

In [2]:
```
df_tweets = pd.read_csv("../data/Tweets.csv", encoding='utf-8')
df_tweets.head()
```

Out[2]:

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativer
0	570306133677760513	neutral	1.0000	NaN	
1	570301130888122368	positive	0.3486	NaN	
2	570301083672813571	neutral	0.6837	NaN	
3	570301031407624196	negative	1.0000	Bad Flight	
4	570300817074462722	negative	1.0000	Can't Tell	

The shape of the dataframe shows 14640 rows/observations, and 15 columns.

In [3]:
```
df_tweets.shape
```

Out[3]:
```
(14640, 15)
```

Check for Nulls

There are no nulls amongst the columns airline_sentiment and text, which is good

In [4]:
```
df_tweets['airline_sentiment'].isnull().sum()
```

Out[4]:
```
0
```

In [5]:
```
df_tweets['text'].isnull().sum()
```

Out[5]:
```
0
```

Column 'negativereason' has a few missing nulls, but they are null only if 'airline_sentiment' is positive or neutral.

In [6]:
```
df_tweets['negativereason'].isnull().sum()
```

Out[6]:
```
5462
```

In [7]:
```
df_tweets['airline_sentiment'].loc[df_tweets['negativereason'].isnull()].unique()
```

Out[7]:
```
array(['neutral', 'positive'], dtype=object)
```

Cleaning Data

Twitter data is notoriously "unclean" compared to the data of other NLP applications, such as newspaper articles, reviews, etc. As mentioned in the Background, Twitter's strict and small character limit means users must figure out ways of shortening their tweets into concise sentences. In practice, this means other forms of unorthodox written communication, such as Internet slang abbreviations, emojis, emoticons, hashtags, retweets, mentions, are employed regularly by users.

For some of these methods of communication, such as mentions and retweets, they provide zero or negative value to our sentiment analysis; as a result, it is in our best interest to remove such text. Mentions are used by users to direct their messages to handles of other other users, such as airline companies in our case. In this dataset, mentions almost always contained the handles of airline companies, which is not very useful if most of the tweets, regardless of positive, neutral, or negative sentiment, contain those mentions. Similarly, retweets were often the tweets of the PR Twitter accounts/handles of airline companies. Because the retweets do not reflect the true sentiment of the user's specific tweet, and the retweets themselves are often of neutral or positive sentiment (never negative, because the tweet would be deleted and the person in charge of the PR Twitter account/handle would be in serious trouble), it is logical to remove retweets from the dataset since retweets will only prove to be "noisy" to our sentiment analysis model.

On the other hand, other methods of communication such as emojis, emoticons, and hashtags could provide positive value to our sentiment analysis, since in theory visual/text elements such as emojis, and emoticons express a variety of emotions in a visual, or pseudo-visual image, which are possibly interlinked with the sentiment of the text. Hashtags could also be useful certain hashtags are associated with topics of positive, or negative sentimentality; for example, hashtags such as `#IceBucketChallenge` are linked with positive sentiments as the hashtag was used to encourage other people to perform acts of charity in a fun, positive manner.

Lastly, the text of the users will have to be cleaned to ensure the text can be properly tokenized by the Spacy model. Such actions of cleaning the text include removing HTML encoding and HTTP links, converting the text to lower case, translating Internet slang abbreviations to their full English expressions, removing stop words and numbers and punctuation, and performing the lemmatization of the remaining text.

Below are links to the following Data Preparation Steps:

1. Remove not useful columns

Looking at the dataframe, we can see some columns will likely not be useful for our purposes of sentiment analysis, such timezones, number of retweets, etc. Therefore, we select the columns that we want, or think will be useful for our data exploration and analysis later.

In [8]:
```
df_tweets = df_tweets[
    ['tweet_id', 'airline_sentiment', 'airline_sentiment_confidence', 'negativ
ereason', 'negativereason_confidence', 'airline', 'text']
]
df_tweets.head()
```

Out[8]:

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativer
0	570306133677760513	neutral	1.0000	NaN	
1	570301130888122368	positive	0.3486	NaN	
2	570301083672813571	neutral	0.6837	NaN	
3	570301031407624196	negative	1.0000	Bad Flight	
4	570300817074462722	negative	1.0000	Can't Tell	

2.Create new text_cleaned column

We create a column 'text_cleaned' that will contain the cleaned up version of 'text' column

In [9]:
```
df_tweets['text_cleaned'] = df_tweets['text']
```

3. Remove HTML encoding

The text has not been cleaned, as there is some HTML encoding left in the text, such as `&` . We will use BeautifulSoup and `lxml` package to remove the HTML encoding from the text.

Sanity Check

In [10]:
```
from bs4 import BeautifulSoup

#Example of what BeautifulSoup with lxml package does
#you may need to install lxml by 'pip install lxml' for this to work, then res
tart kernel
example1 = BeautifulSoup('Disappointed,UNITED did NOT feed small CHILDREN on a
5 &amp; half hour flight', 'lxml')
print(example1.get_text())
```

```
Disappointed,UNITED did NOT feed small CHILDREN on a 5 & half hour flight
```

Remove HTML Encoding from text

In [11]:
```
df_tweets['text_cleaned'] = df_tweets['text_cleaned'].apply(lambda text: Beaut
ifulSoup(text, 'lxml').get_text())
df_tweets['text_cleaned']
```

Out[11]:
```
0
@VirginAmerica What @dhepburn said.
1
@VirginAmerica plus you've added commercials to the experience... tacky.
2
@VirginAmerica I didn't today... Must mean I need to take another trip!
3                                   @VirginAmerica it's really aggressive to
blast obnoxious "entertainment" in your guests' faces & they have little reco
urse
4
@VirginAmerica and it's a really big bad thing about it

...
14635
@AmericanAir thank you we got on a different flight to Chicago.
14636    @AmericanAir leaving over 20 minutes Late Flight. No warnings or com
munication until we were 15 minutes Late Flight. That's called shitty custome
r svc
14637
@AmericanAir Please bring American Airlines to #BlackBerry10
14638                   @AmericanAir you have my money, you change my flight,
and don't answer your phones! Any other suggestions so I can make my commitme
nt??
14639               @AmericanAir we have 8 ppl so we need 2 know how many se
ats are on the next flight. Plz put us on standby for 4 people on the next fl
ight?
Name: text_cleaned, Length: 14640, dtype: object
```

4. Remove retweets

Retweets are denoted in 'text' column as 'RT @another_user another_user's tweet'. We should remove retweets because we need to analyze the tweets of the users, not the retweets. In this situation, retweets provide no real value to our text exploration analysis as normally the users retweet to the airlines. Retweets are based off the tweets from the specified airlines' Twitter PR, and therefore, are likely going to be either of neutral or positive sentiment. Removing such retweets will hopefully remove any noise that will prevent the models from classifying sentiment.

Sanity Check

In [12]:
```
regex_to_replace= r'RT \@.*'
replace_value= ''
```

In [13]:
```
import re
example1 = 'Awesome! RT @VirginAmerica: Watch nominated films at 35,000 feet.
 #MeetTheFleet #Oscars http://t.co/DnStITRzWy'
example1 = re.sub(regex_to_replace, replace_value, example1)
example1
```

Out[13]:
```
'Awesome! '
```

Remove retweets from text

In [14]:
```
df_tweets['text_cleaned'] = df_tweets['text_cleaned'].replace(to_replace=regex
_to_replace, value=replace_value, regex=True)
df_tweets['text_cleaned']
```

Out[14]:
```
0
@VirginAmerica What @dhepburn said.
1
@VirginAmerica plus you've added commercials to the experience... tacky.
2
@VirginAmerica I didn't today... Must mean I need to take another trip!
3                                   @VirginAmerica it's really aggressive to
blast obnoxious "entertainment" in your guests' faces & they have little reco
urse
4
@VirginAmerica and it's a really big bad thing about it

...
14635
@AmericanAir thank you we got on a different flight to Chicago.
14636    @AmericanAir leaving over 20 minutes Late Flight. No warnings or com
munication until we were 15 minutes Late Flight. That's called shitty custome
r svc
14637
@AmericanAir Please bring American Airlines to #BlackBerry10
14638                   @AmericanAir you have my money, you change my flight,
and don't answer your phones! Any other suggestions so I can make my commitme
nt??
14639                @AmericanAir we have 8 ppl so we need 2 know how many se
ats are on the next flight. Plz put us on standby for 4 people on the next fl
ight?
Name: text_cleaned, Length: 14640, dtype: object
```

5. Remove mentions

Sometimes, users use mentions (for example, tweet mentions include @VirginAirlines, @JetBlue, etc., in other words, the airlines' handle). They normally appear in the beginning of the users' tweets. These mentions are not useful for sentiment analysis purposes because the vast majority of tweets have some sort of mention to one of the 6 airline companies.

Sanity Check

In [15]:
```
regex_to_replace = r'\@[\w\d]*'
replace_value= ''
```

In [16]:
```
import re
example1 = '@VirginAmerica Thank you for the follow'
example1 = re.sub(regex_to_replace, replace_value, example1)
example1
```

Out[16]:
```
' Thank you for the follow'
```

Remove mentions from text

In [17]:
```
df_tweets['text_cleaned'] = df_tweets['text_cleaned'].replace(to_replace=regex
_to_replace, value=replace_value, regex=True)
df_tweets['text_cleaned']
```

Out[17]:
```
0
What  said.
1
plus you've added commercials to the experience... tacky.
2
I didn't today... Must mean I need to take another trip!
3                                          it's really aggressive to blast obnox
ious "entertainment" in your guests' faces & they have little recourse
4
and it's a really big bad thing about it

...
14635
thank you we got on a different flight to Chicago.
14636     leaving over 20 minutes Late Flight. No warnings or communication u
ntil we were 15 minutes Late Flight. That's called shitty customer svc
14637
Please bring American Airlines to #BlackBerry10
14638                     you have my money, you change my flight, and don't a
nswer your phones! Any other suggestions so I can make my commitment??
14639                  we have 8 ppl so we need 2 know how many seats are on t
he next flight. Plz put us on standby for 4 people on the next flight?
Name: text_cleaned, Length: 14640, dtype: object
```

6. Remove HTTP links

Users attach http links occasionally in their tweets. We need to remove HTTP links from the text, since they provide no real value to our sentiment analysis as well. From what we can see in the data, they are mostly links to articles.

Sanity Check

In [18]:
```
regex_to_replace = r'https*://[^\s]*'
replace_value = ''
```

In [19]:
```
import re
example1 = '@VirginAmerica when are you putting some great deals from PDX to L
AS or from LAS to PDX show me your love! http://t.co/enIQg0buzj'
example1 = re.sub(regex_to_replace, replace_value, example1)
example1
```

Out[19]:
```
'@VirginAmerica when are you putting some great deals from PDX to LAS or from
LAS to PDX show me your love! '
```

Removing HTTP Links from text

In [20]:
```
df_tweets['text_cleaned'] = df_tweets['text_cleaned'].replace(to_replace=regex
_to_replace, value=replace_value, regex=True)
df_tweets['text_cleaned']
```

Out[20]:
```
0
What  said.
1
plus you've added commercials to the experience... tacky.
2
I didn't today... Must mean I need to take another trip!
3                                          it's really aggressive to blast obnox
ious "entertainment" in your guests' faces & they have little recourse
4
and it's a really big bad thing about it

...
14635
thank you we got on a different flight to Chicago.
14636     leaving over 20 minutes Late Flight. No warnings or communication u
ntil we were 15 minutes Late Flight. That's called shitty customer svc
14637
Please bring American Airlines to #BlackBerry10
14638                     you have my money, you change my flight, and don't a
nswer your phones! Any other suggestions so I can make my commitment??
14639                 we have 8 ppl so we need 2 know how many seats are on t
he next flight. Plz put us on standby for 4 people on the next flight?
Name: text_cleaned, Length: 14640, dtype: object
```

7. Extract Emojis and Emoticons

Rather than removing emojis and emoticons, emojis and emoticons can be seen as an integral part of the Internet language. Therefore, we should extract emojis and emoticons from the text if they exist, as they may be good features for sentiment analysis for "Internet"-speak.

Emojis are special characters which are shown as actual visual images, whereas emoticons are keyboard characters arranged in a certain format so that it represents a human-like facial expression. Emoticons are not as commonly used compared to emojis anymore, but are still used occassionally by people, and it is in our best interests to also extract emoticons as well.

We will use a third-party Python library called 'emot', which provides the ability to recognize and extract both emojis and emoticons. Github can be found here: https://github.com/NeelShah18/emot (https://github.com/NeelShah18/emot). There was an issue with the library where emot.emoticons would return different JSON structures, either `{'flag': False}` or `{'value': [], 'mean': [], 'location': [], 'flag': False}` whenever emoticons were not found. Upon manually investigating a few examples with such issues, we found out that although this was an issue, it was true that there were no emoticons in the texts of those examples. Therefore, we got around this issue by writing a variety of try/except catches.

Testing the `emot` package manually

In [21]:
```
#Sanity check
import emot
text = "I love python 👦 :-)"
print(emot.emoji(text))
print(emot.emoticons(text))
```

```
{'value': ['👦'], 'mean': [':man:'], 'location': [[14, 14]], 'flag': True}
{'value': [':-)'], 'location': [[16, 19]], 'mean': ['Happy face smiley'], 'fl
ag': True}
```

In [22]:
```
df_example1 = df_tweets.loc[df_tweets['tweet_id'] == 569198104806699008]
print(df_example1['text_cleaned'].to_string(index=False))
print(emot.emoji(df_example1['text_cleaned'].to_string(index=False)))
print(emot.emoticons(df_example1['text_cleaned'].to_string(index=False)))
```

```
  hahaha 😂 YOU GUYS ARE AMAZING. I LOVE YOU GUYS!!!💗
{'value': ['😂', '💗'], 'mean': [':face_with_tears_of_joy:', ':growing_hear
t:'], 'location': [[9, 9], [51, 51]], 'flag': True}
{'value': [], 'location': [], 'mean': [], 'flag': False}
```

In [23]:
```
df_example2 = df_tweets.loc[df_tweets['tweet_id'] == 568890074164809728]
print(df_example2['text_cleaned'].to_string(index=False))
print(emot.emoji(df_example2['text_cleaned'].to_string(index=False)))
print(emot.emoticons(df_example2['text_cleaned'].to_string(index=False)))
```

```
  we have a hot female pilot! Sweet! DCA to SFO! :-)
{'value': [], 'mean': [], 'location': [], 'flag': False}
{'value': [':-)'], 'location': [[49, 52]], 'mean': ['Happy face smiley'], 'fl
ag': True}
```

Extracing Emojis and Emoticons from text

In [24]:
```
#this function will remove any emojis
#accepts the following parameters:
#(1) text_cleaned: roughly cleaned text in String to parse through and remove
 emojis
#(2) emojis_flag: boolean created by emot.emoji(...), can be accessed by calli
ng 'flag' key in dictionary (for more information see above)
#(3) emojis: emojis object created by emot.emoji(...), can be accessed by call
ing 'value' key in dictionary (for more information see above)

#returns the cleaned up text without emojis

def remove_emojis(text_cleaned, emojis_flag, emojis):
    text_cleaned_no_emojis = text_cleaned

    #print('text_cleaned @ remove_emojis: ' + text_cleaned)
    #print('emojis_flag: ' + str(emojis_flag))
    #print('emojis: ' + str(emojis))


    #if flag is True, that means there are emojis, and we need to remove them
    if emojis_flag:
        for i in emojis:

            #rather than use location, we will match by String and see if we c
an remove it, because I'm lazy af lol
            #print(str(i))
            text_cleaned_no_emojis = text_cleaned_no_emojis.replace(i, '')

        if text_cleaned_no_emojis == text_cleaned:
            print('Uh...Houston, we have a problem...for the following text: '
+ text_cleaned + ', for row: ' + str(i))
            print('The following emojis were not removed: ' + str(emojis))

    #print('resulting text_cleaned_no_emojis: ' + text_cleaned_no_emojis)
    return text_cleaned_no_emojis
```

In [25]:
```
#this function will remove any emoticons
#accepts the following parameters:
#(1) text_cleaned: roughly cleaned text in String to parse through and remove
 emoticons
#(2) emoticons_flag: boolean created by emot.emoticons(...), can be accessed b
y calling 'flag' key in dictionary (for more information see above)
#(3) emoticons: emojis object created by emot.emoticons(...), can be accessed
 by calling 'value' key in dictionary (for more information see above)

#returns the cleaned up text without emoticons

def remove_emoticons(text_cleaned, emoticons_flag, emoticons):
    text_cleaned_no_emoticons = text_cleaned

    #print('text_cleaned @ remove_emoticons: ' + text_cleaned)
    #print('emoticons_flag: ' + str(emoticons_flag))
    #print('emoticons: ' + str(emoticons))

    #if flag is True, that means there are emoticons, and we need to remove th
em
    if emoticons_flag:
        for i in emoticons:

            #rather than use location, we will match by String and see if we c
an remove it, because I'm lazy af lol
            #print(str(i))
            text_cleaned_no_emoticons = text_cleaned_no_emoticons.replace(i,
'')

        if text_cleaned_no_emoticons == text_cleaned:
            print('Uh...Houston, we have a problem...for the following text: '
+ text_cleaned + ', for row: ' + str(i))
            print('The following emojis were not removed: ' + str(emoticons))

    #print('resulting text_cleaned_no_emoticons: ' + text_cleaned_no_emoticon
s)
    return text_cleaned_no_emoticons
```

In [26]:
```
#this function will extract any emojis into a separate 'emojis' column,
#while also removing said emojis from column: 'text_cleaned'
#to form a new column: 'text_cleaned_without_emojis_emoticons'

#returns dataframe with the aforementioned new columns

import emot

#set logging so we don't output the try/except log messages unless if we want
 them
#default level if WARNING
import logging

#in 'emojis_emoticons' column, it will hold the emot.emoji return dictionary
 (see above for examples)
def extract_emojis(df_tweets, ):
    df_tweets_2 = df_tweets
    df_tweets_2['emojis_flag'] = ''
    df_tweets_2['emojis'] = ''
    df_tweets_2['emoticons_flag'] = ''
    df_tweets_2['emoticons'] = ''

    #easier to just write out code to loop through dataframe
    for i, row in df_tweets_2.iterrows():
        #print(i)
        #print(row)
        text_cleaned = df_tweets_2.at[i, 'text_cleaned']
        emojis = emot.emoji(text_cleaned)
        emoticons = emot.emoticons(text_cleaned)
        #print('EMOJIS: ' + str(emojis))
        #print('EMOTICONS: ' + str(emoticons))

        ##When using emot.emoticons, the output when flag=False is inconsisten
t, it either is
        #(a) {'value': [], 'mean': [], 'location': [], 'flag': False}, or
        #(b) {'flag': False}

        #Therefore we have a bunch of try-except the aforementioned exception
 that will be raised, and set the values manually
        try:
            df_tweets_2.at[i, 'emojis_flag'] = emojis['flag']
        except:
            logging.debug('Unable to grab emojis flag at row number: ' + str(i
))
            df_tweets_2.at[i, 'emojis_flag'] = False

        try:
             df_tweets_2.at[i, 'emojis'] = emojis['value']
        except:
            logging.debug('Unable to grab emojis value at row number: ' + str(
i))
            df_tweets_2.at[i, 'emojis'] = []

        try:
            df_tweets_2.at[i, 'emoticons_flag'] = emoticons['flag']
        except:
```

```
            logging.debug('Unable to grab emoticons flag at row number: ' + st
r(i))
            df_tweets_2.at[i, 'emoticons_flag'] = False


        try:
            df_tweets_2.at[i, 'emoticons'] = emoticons['value']
        except:
            logging.debug('Unable to grab emoticons value at row number: ' + s
tr(i))
            df_tweets_2.at[i, 'emoticons'] = []

        #afterwards, we need to remove emojis and emoticons from the
        df_tweets_2.at[i, 'text_cleaned_without_emojis_emoticons'] = remove_em
ojis(
            text_cleaned,
            df_tweets_2.at[i, 'emojis_flag'],
            df_tweets_2.at[i, 'emojis']
        )

        df_tweets_2.at[i, 'text_cleaned_without_emojis_emoticons'] = remove_em
oticons(
            df_tweets_2.at[i, 'text_cleaned_without_emojis_emoticons'],
            df_tweets_2.at[i, 'emoticons_flag'],
            df_tweets_2.at[i, 'emoticons']
        )

    return df_tweets_2
```

In [27]:
```
#df_tweets['text_cleaned'] = df_tweets['text_cleaned'].apply(lambda text: Beau
tifulSoup(text, 'lxml').get_text())
#df_tweets['text_cleaned']
df_tweets = extract_emojis(df_tweets)
#df_tweets.head()
```

Sanity check

In [28]:
```
#Sanity check: briefly check some examples where we know emojis and emoticons
 do exist
df_example1 = df_tweets.loc[df_tweets['tweet_id'] == 569198104806699008]
print(df_example1['emojis_flag'])
print(df_example1['emojis'])
print(df_example1['text'])
print(df_example1['text_cleaned'])
print(df_example1['text_cleaned_without_emojis_emoticons'])
```

```
238    True
Name: emojis_flag, dtype: object
238    [😂, 💙]
Name: emojis, dtype: object
238    @VirginAmerica hahaha 😂@VirginAmerica YOU GUYS ARE AMAZING. I LOVE YO
U GUYS!!!💙
Name: text, dtype: object
238     hahaha 😂 YOU GUYS ARE AMAZING. I LOVE YOU GUYS!!!💙
Name: text_cleaned, dtype: object
238     hahaha  YOU GUYS ARE AMAZING. I LOVE YOU GUYS!!!
Name: text_cleaned_without_emojis_emoticons, dtype: object
```

In [29]:
```
#Sanity check: briefly check some examples where we know emojis and emoticons
 do exist
df_example1 = df_tweets.loc[df_tweets['tweet_id'] == 570270684619923457]
print(df_example1['emojis_flag'])
print(df_example1['emojis'])
print(df_example1['text'])
print(df_example1['text_cleaned'])
print(df_example1['text_cleaned_without_emojis_emoticons'])
```

```
18    True
Name: emojis_flag, dtype: object
18    [❤, ☺, 👍]
Name: emojis, dtype: object
18    I ♡ flying @VirginAmerica. ☺👍
Name: text, dtype: object
18    I ♡ flying . ☺👍
Name: text_cleaned, dtype: object
18    I  flying .
Name: text_cleaned_without_emojis_emoticons, dtype: object
```

In [30]:
```
#Sanity check: briefly check some examples where we know emojis and emoticons
 do exist
df_example2 = df_tweets.loc[df_tweets['tweet_id'] == 568890074164809728]
print(df_example2['emoticons_flag'])
print(df_example2['emoticons'])
print(df_example2['text'])
print(df_example2['text_cleaned'])
print(df_example2['text_cleaned_without_emojis_emoticons'])
```

```
277    True
Name: emoticons_flag, dtype: object
277    [:-)]
Name: emoticons, dtype: object
277    @VirginAmerica we have a hot female pilot! Sweet! DCA to SFO! :-)
Name: text, dtype: object
277     we have a hot female pilot! Sweet! DCA to SFO! :-)
Name: text_cleaned, dtype: object
277     we have a hot female pilot! Sweet! DCA to SFO!
Name: text_cleaned_without_emojis_emoticons, dtype: object
```

8. Extract and Remove Hashtags

Some of the tweets have hashtag, which are keyword phrases spelled out with no spaces, and a pound sign at the front. They could offer more insight into the sentiment of the tweet. There were 3669 hashtags in the dataset in 2489 rows (some tweets evidently had more than one hashtag).

For now, the easiest approach would be to remove the hashtags and the words inside during text cleaning. If we have time, we can try to analyze the best approaches to extract the words from the hashtags, and split them into meaningful words. This would be fairly challenging, however, as some tweets have multiple words that are not capitalized, so there is not an easy, fool-proof way of extracing hashtags into words.

We also decided to extract the hashtags into a list inside a separate column for further data exploration and analysis.

In [31]:
```
print(len(df_tweets.loc[df_tweets['text'].str.contains('#')]))
```

```
2489
```

In [32]:
```
#extracts hashtags into a list, which is put into a new column called 'hashtag
s', whilst also removing hashtags from the text_cleaned
def extract_and_remove_hashtags(df_tweets):
    regex_to_replace= r'#(\w+)'
    replace_value= ''
    df_tweets['hashtags'] = ''
    df_tweets['text_cleaned_without_emojis_emoticons_hashtags'] = ''

    for i, row in df_tweets.iterrows():
        df_tweets.at[i, 'hashtags'] = re.findall(regex_to_replace, df_tweets.a
t[i, 'text_cleaned'])
        df_tweets.at[i, 'text_cleaned_without_emojis_emoticons_hashtags'] = re
.sub(regex_to_replace, replace_value,  df_tweets.at[i, 'text_cleaned'])

        #create hashtags_flag
        if df_tweets.at[i, 'hashtags'] == []:
            df_tweets.at[i, 'hashtags_flag'] = False
        else:
            df_tweets.at[i, 'hashtags_flag'] = True

    return df_tweets
```

In [33]:
```
df_tweets = extract_and_remove_hashtags(df_tweets)
df_tweets.loc[df_tweets['tweet_id'] == 569587242672398336]
```

Out[33]:

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	nega
14637	569587242672398336	neutral	1.0	NaN	

9. Convert text to Lowercase

We need to convert the text to lower case. This is done so that when we tokenize the words, there won't be words that are grouped separately just because of case sensitivity.

In [34]:
```
df_tweets['text_cleaned_lower_case'] = \
    df_tweets['text_cleaned_without_emojis_emoticons_hashtags'].apply(lambda t
ext: text.lower())
df_tweets.head()
```

Out[34]:

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativer
0	570306133677760513	neutral	1.0000	NaN	
1	570301130888122368	positive	0.3486	NaN	
2	570301083672813571	neutral	0.6837	NaN	
3	570301031407624196	negative	1.0000	Bad Flight	
4	570300817074462722	negative	1.0000	Can't Tell	

10. Internet Slang abbreviations

We need to convert the Internet slang abbreviations into readable English lexicon. To do so, we will use a dictionary of commonly used slang words, which we have used partly from here: https://github.com/Deffro/text-preprocessing-techniques/blob/master/slang.txt (https://github.com/Deffro/text-preprocessing-techniques/blob/master/slang.txt). We modified the dictionary to be in CSV format, as well as changing some of the key-value pairs as they were improperly written. Each Internet slang abbreviation key is linked to its respective value, which would be the complete form of the abbreviation.

In [35]:
```
df_slang = pd.read_csv('..\data\slang.csv')
df_slang
```

Out[35]:

	slang_abbreviation	complete_form
0	2day	today
1	2nite	tonight
2	4u	for you
3	4ward	forward
4	a3	anyplace, anywhere, anytime
...	...	...
285	yuge	huge
286	yw	you are welcome
287	ywa	you are welcome anyway
288	zomg	oh my god!
289	zzz	sleeping

290 rows × 2 columns

In [36]:
```
def replace_abbreviation(text, abbreviation, complete_form):
    return re.sub(abbreviation, complete_form, text)
```

In [37]:
```
#Sanity check, testing above function
regex_expression = re.compile('\\b'+'plz'+'\\b')
regex_to_replace =  'please'

replace_abbreviation(
    'o we fly straight into sfo and honululu gets pushed back 3.5 hours and no
w it looks like more delays.  i beg of you plz sort this out soon!',
    regex_expression,
    regex_to_replace
)
```

Out[37]:
```
'o we fly straight into sfo and honululu gets pushed back 3.5 hours and now i
t looks like more delays.  i beg of you please sort this out soon!'
```

In [38]:
```
#Another sanity check, but a negative case this time around
#the word 'straight' should remain the same, even though the dictionary contai
ns the abbreviation 'aight'
regex_expression = re.compile('\\b'+'aight'+'\\b')
regex_to_replace = 'alright'

replace_abbreviation(
    'o we fly straight into sfo and honululu gets pushed back 3.5 hours and no
w it looks like more delays.  i beg of you plz sort this out soon!',
    regex_expression,
    regex_to_replace
)
```

Out[38]:
```
'o we fly straight into sfo and honululu gets pushed back 3.5 hours and now i
t looks like more delays.  i beg of you plz sort this out soon!'
```

In [39]:
```
#this function will find slang abbreviations and replace them with the complet
e forms
#needs df_tweets and df_slang
#returns a new dataframe with new column 'text_cleaned_without_emojis_emoticon
s_hashtags_abbreviations'

#we will iterate over every row/tweet and and see if there are instances of sl
ang abbreviations
#computationally expensive, but because there are only 14000 rows, it's not to
o bad
#there is no easy way of detecting whether a word is an abbreviation or not us
ing Spacy, NLTK, or spellchecker
def find_slang_abbreviations_and_replace_with_complete_form(df_tweets, df_slan
g):
    df_tweets['text_cleaned_no_abbreviations'] = ''

    for i, tweet_row in df_tweets.iterrows():
        df_tweets.at[i, 'text_cleaned_no_abbreviations'] = \
            df_tweets.at[i, 'text_cleaned_lower_case']

        if i % 1000 == 0:
            print('at row number: ' + str(i))

        for j, slang_row in df_slang.iterrows():
            #print(slang_row)
            slang_abbreviation = df_slang.at[j, 'slang_abbreviation']
            complete_form = df_slang.at[j, 'complete_form']

            #print("slang_abbreviation: " + slang_abbreviation)
            #print("complete_form: " + complete_form)

            regex_expression = re.compile('\\b'+slang_abbreviation+'\\b')
            regex_to_replace = complete_form

            df_tweets.at[i, 'text_cleaned_no_abbreviations'] = \
                replace_abbreviation(
                    df_tweets.at[i, 'text_cleaned_no_abbreviations'],
                    regex_expression,
                    regex_to_replace
            )
            #print("current: " + df_tweets.at[i, 'text_cleaned_no_abbreviation
s'])


    return df_tweets
```

In [40]:
```
#create new column 'text_cleaned_no_abbreviations' in df_tweets
#by calling find_slang_abbreviations_and_replace_with_complete_form function
df_tweets = find_slang_abbreviations_and_replace_with_complete_form(df_tweets,
df_slang)
```

```
at row number: 0
at row number: 1000
at row number: 2000
at row number: 3000
at row number: 4000
at row number: 5000
at row number: 6000
at row number: 7000
at row number: 8000
at row number: 9000
at row number: 10000
at row number: 11000
at row number: 12000
at row number: 13000
at row number: 14000
```

In [41]:
```
df_tweets.head()
```

Out[41]:

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativer
0	570306133677760513	neutral	1.0000	NaN	
1	570301130888122368	positive	0.3486	NaN	
2	570301083672813571	neutral	0.6837	NaN	
3	570301031407624196	negative	1.0000	Bad Flight	
4	570300817074462722	negative	1.0000	Can't Tell	

In [42]:
```
#sanity check
df_tweets.loc[df_tweets['tweet_id'] == 568899587424931840]
```

Out[42]:

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negat
2788	568899587424931840	negative	1.0	Late Flight	

11. Removing StopWords and Punctuation and numbers, Lemmatization, and Tokenization

We want to remove stop words because most common words such as "the", "and", "I", "you" are not likely to add any value to our sentiment analysis. Removing stopwords will therefore reduce the noise that will reduce the effectiveness of our eventual sentiment analysis model. Removing stopwords is also beneficial in the sense that it also removes common contractions such as "I've, you're, etc."

We also want to remove punctuation because punctuation is not useful to our sentiment analysis. Punctutation might help determine the intensity, or polarity of text, but that is not relevant to sentiment. Similar to punctuation, we also want to remove numbers.

Lastly, we will need to perform lemmatization, so that we can reduce the words to their root form

In [43]:
```
#load spacy model
import spacy

nlp = spacy.load('en_core_web_md')
```

In [44]:
```
df_tweets['text_list_no_stop_words'] = ''
df_tweets['lemmas_list'] = ''

for i, row in df_tweets.iterrows():
    if i % 1000 == 0:
        print('at row number: ' + str(i))

    text = df_tweets.at[i, 'text_cleaned_no_abbreviations']

    #tokenize text into list of tokens
    token_list = nlp(text)

    text_list_no_stop_words = []
    lemmas_list = []

    #remove stop words, and lemmatize
    for token in token_list:
        #print(str(token.is_stop))
        #print(str(token.pos_))

        #if token is not a stop word, and not punctuation, and not a number,
        #then it is useful to us and we store them in our lists
        if (token.is_stop == False) & (not token.is_punct) & (not token.is_spa
ce) & (not token.is_digit):
            text_list_no_stop_words.append(token.text)
            lemmas_list.append(token.lemma_)

    #print('text_list_no_stop_words:' + str(text_list_no_stop_words))
    #print('lemmas_list:' + str(lemmas_list))
    df_tweets.at[i, 'text_list_no_stop_words'] = " ".join(text_list_no_stop_wo
rds)
    df_tweets.at[i, 'lemmas_list'] = " ".join(lemmas_list)
```

```
at row number: 0
at row number: 1000
at row number: 2000
at row number: 3000
at row number: 4000
at row number: 5000
at row number: 6000
at row number: 7000
at row number: 8000
at row number: 9000
at row number: 10000
at row number: 11000
at row number: 12000
at row number: 13000
at row number: 14000
```

In [45]:
```
#sanity check
df_tweets.head()
```

Out[45]:

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativer
0	570306133677760513	neutral	1.0000	NaN	
1	570301130888122368	positive	0.3486	NaN	
2	570301083672813571	neutral	0.6837	NaN	
3	570301031407624196	negative	1.0000	Bad Flight	
4	570300817074462722	negative	1.0000	Can't Tell	

Cleaning data is now complete, we'll save the dataframe to a csv.

In [46]:
```
df_tweets.to_csv('..\data\Tweets_cleaned.csv', index = False)
```