Twitter US Airline Sentiment Analysis

Problem Statement:

Twitter possesses 330 million monthly active users, which allows businesses to reach a broad population and connect with customers without intermediaries. On the other hand, there's so much information that it's difficult for brands to quickly detect negative social mentions that could harm their business.

Objective:

Given the above statement, I will build a sentiment classification model and perform EDA to help companies understand their audience, keep on top of what's being said about their brand and their competitors, and discover new trends in the industry.

Data Description:

A sentiment analysis job about the problems of each major U.S. airline. Twitter data was scraped from February of 2015 and contributors were asked to first classify positive, negative, and neutral tweets, followed by categorizing negative reasons (such as "late flight" or "rude service").

Dataset:

- tweet_id
- airline_sentiment
- airline_sentiment_confidence
- negativereason
- negativereason_confidence
- airline
- airline_sentiment_gold
- name
- negativereason_gold
- retweet_count
- text
- tweet_coord
- tweet_created
- tweet_location
- user_timezone

Libraries:

In [1]:
```
%load_ext nb_black
# Library to suppress warnings or deprecation notes
import warnings

warnings.filterwarnings("ignore")


import re
import numpy as np              #for large and multi-dimensional arrays
import pandas as pd             #for data manipulation and analysis
import nltk                     #Natural language processing tool-kit


from nltk.corpus import stopwords        #Stopwords corpus
from nltk.stem import PorterStemmer        #Stemmer
from nltk.tokenize import word_tokenize

from sklearn.feature_extraction.text import CountVectorizer    #Bag of words
from sklearn.feature_extraction.text import TfidfVectorizer    #For TF-IDF


# import necessary libraries.

import re, string, unicodedata       # Import Regex, string and unicodedata.
import contractions                   # Import contractions library.
from bs4 import BeautifulSoup           # Import BeautifulSoup.

import numpy as np                        # Import numpy.
import pandas as pd                         # Import pandas.
import nltk                                   # Import Natural Language Tool-Kit

nltk.download('wordnet')

from nltk.corpus import stopwords                           # Import stopwords.
from nltk.tokenize import word_tokenize, sent_tokenize  # Import Tokenizer.
from nltk.stem.wordnet import WordNetLemmatizer           # Import Lemmatizer.
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold , StratifiedKFold, cross_val_score
from sklearn.metrics import accuracy_score
import seaborn as sns


from sklearn.ensemble import RandomForestClassifier    # Import Random forest Cl
from sklearn.metrics import classification_report      # Import Classification r
from sklearn.model_selection import cross_val_score
```

```
[nltk_data] Downloading package wordnet to C:\Users\Andrew
[nltk_data]     Hocher\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Read Dataset

In [2]:
```
data = pd.read_csv("Tweets.csv")
```

```
df = data.copy()
```

Data Summary

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

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

In [4]:
```
print(f"There are {df.shape[0]} rows and {df.shape[1]} columns.")
```

```
There are 14640 rows and 15 columns.
```

In [5]:
```
df.head()
```

Out[5]:

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativei
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 [6]:
```
df.tail()
```

Out[6]:

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	ne
14635	569587686496825344	positive	0.3487	NaN	
14636	569587371693355008	negative	1.0000	Customer Service Issue	
14637	569587242672398336	neutral	1.0000	NaN	

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	ne
14638	569587188687634433	negative	1.0000	Customer Service Issue	
14639	569587140490866689	neutral	0.6771	NaN	

In [7]:
```
np.random.seed(2)
df.sample(10)
```

Out[7]:

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	ne
8917	567750136223518720	positive	1.0000	NaN	
9534	569888646633017344	negative	1.0000	Late Flight	
12875	569993908324663296	neutral	0.6654	NaN	
4601	569988842335244289	positive	1.0000	NaN	
4308	567714192980201472	negative	1.0000	longlines	
10981	568640966866935808	negative	1.0000	Late Flight	
11037	568594646168948736	negative	1.0000	Can't Tell	
12991	569966076047437824	neutral	0.6421	NaN	
5685	568807908374286336	positive	1.0000	NaN	
5784	568605955614642176	positive	1.0000	NaN	

tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	neg

In [8]:
```
df[data.duplicated()].count()
```

Out[8]:
```
tweet_id                        36
airline_sentiment               36
airline_sentiment_confidence    36
negativereason                  19
negativereason_confidence       19
airline                         36
airline_sentiment_gold           0
name                            36
negativereason_gold              0
retweet_count                   36
text                            36
tweet_coord                      4
tweet_created                   36
tweet_location                  26
user_timezone                   30
dtype: int64
```

In [9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14640 entries, 0 to 14639
Data columns (total 15 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   tweet_id                      14640 non-null  int64  
 1   airline_sentiment             14640 non-null  object 
 2   airline_sentiment_confidence  14640 non-null  float64
 3   negativereason                9178 non-null   object 
 4   negativereason_confidence     10522 non-null  float64
 5   airline                       14640 non-null  object 
 6   airline_sentiment_gold        40 non-null     object 
 7   name                          14640 non-null  object 
 8   negativereason_gold           32 non-null     object 
 9   retweet_count                 14640 non-null  int64  
 10  text                          14640 non-null  object 
 11  tweet_coord                   1019 non-null   object 
 12  tweet_created                 14640 non-null  object 
 13  tweet_location                9907 non-null   object 
 14  user_timezone                 9820 non-null   object 
dtypes: float64(2), int64(2), object(11)
memory usage: 1.7+ MB
```

In [10]:
```
df.isnull().sum()
```

Out[10]:
```
tweet_id                            0
airline_sentiment                   0
airline_sentiment_confidence        0
negativereason                   5462
negativereason_confidence        4118
airline                             0
airline_sentiment_gold          14600
name                                0
negativereason_gold             14608
retweet_count                       0
```

```
text                           0
tweet_coord                13621
tweet_created                  0
tweet_location              4733
user_timezone               4820
dtype: int64
```

In [11]:
```
df.describe().T
```

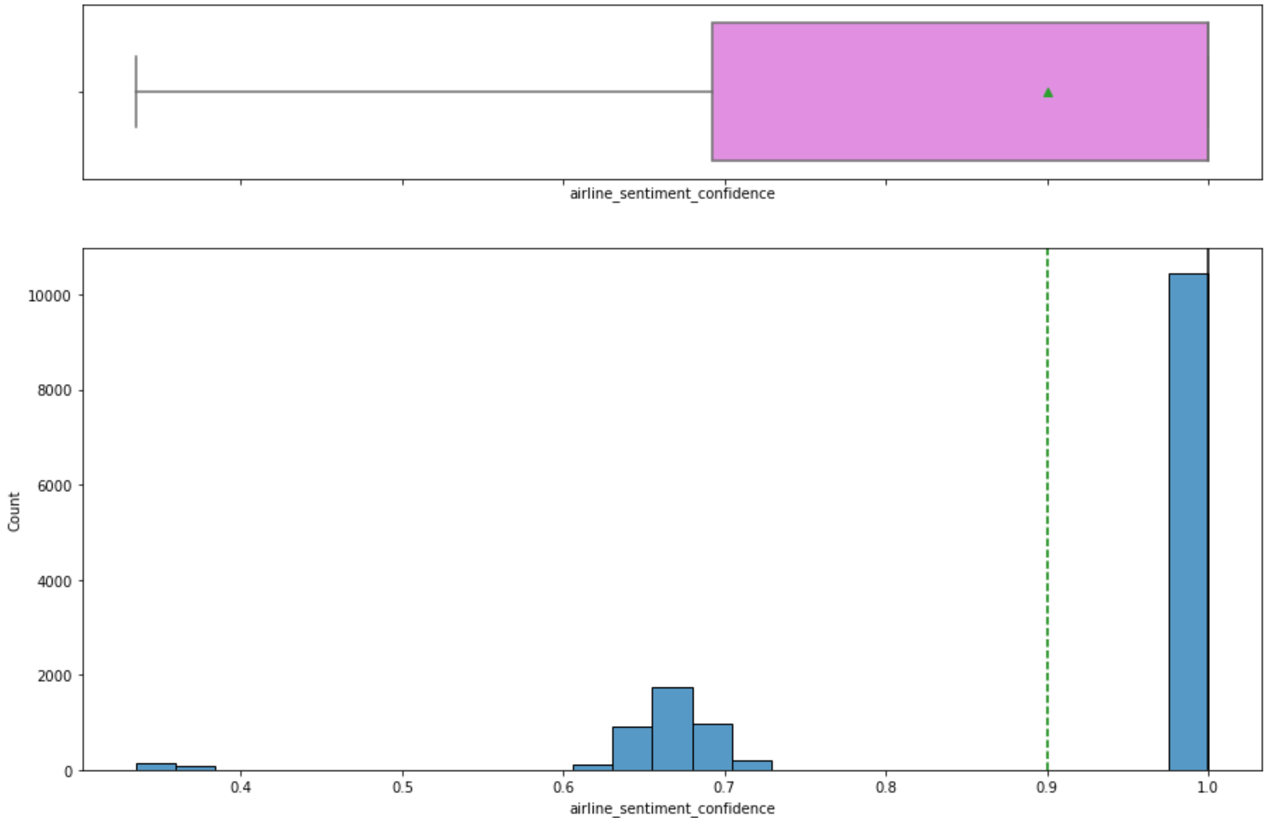
Out[11]:

	count	mean	std	min	25%
tweet_id	14640.0	5.692184e+17	7.791112e+14	5.675883e+17	5.685592e+17
airline_sentiment_confidence	14640.0	9.001689e-01	1.628300e-01	3.350000e-01	6.923000e-01
negativereason_confidence	10522.0	6.382983e-01	3.304398e-01	0.000000e+00	3.606000e-01
retweet_count	14640.0	8.265027e-02	7.457782e-01	0.000000e+00	0.000000e+00

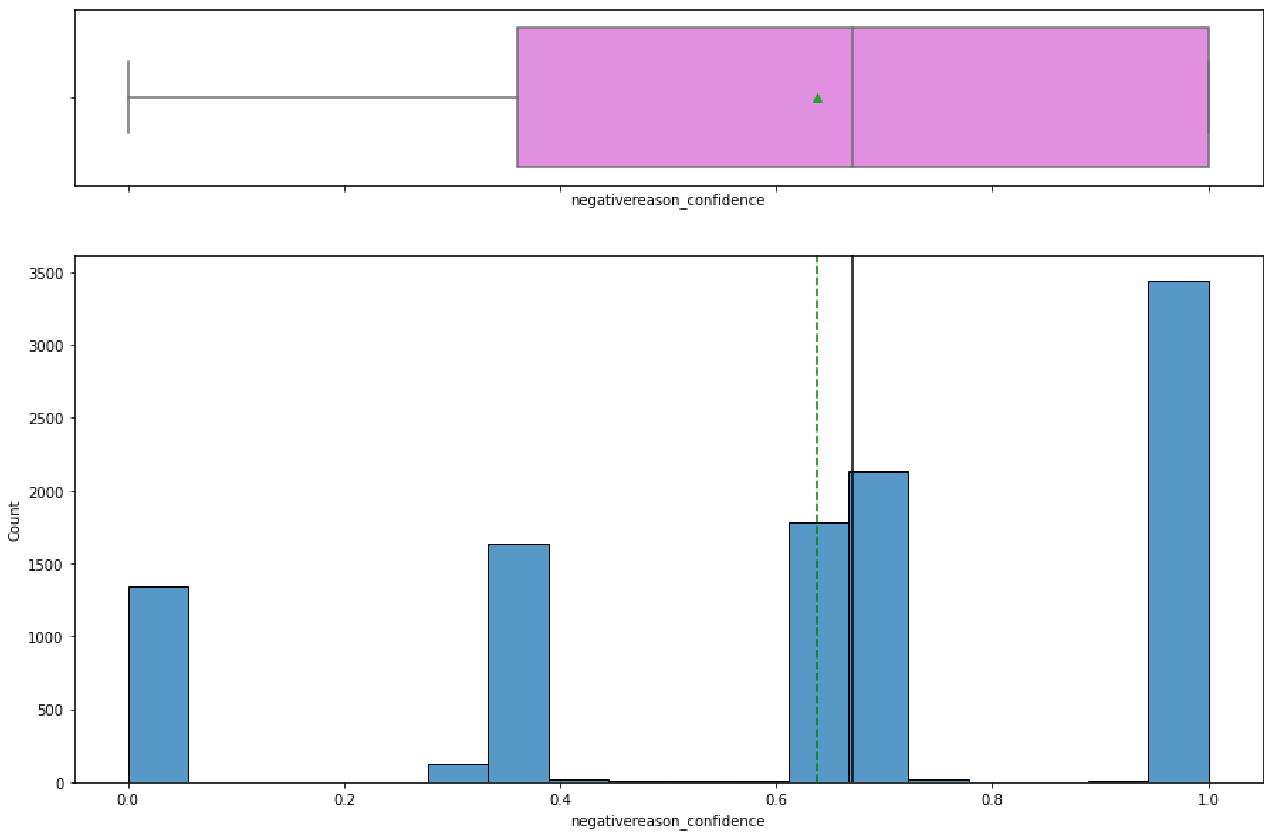
EDA

In [12]:
```
def histogram_boxplot(data, feature, figsize=(15, 10), kde=False, bins=None):
    """
    Boxplot and histogram combined

    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (15,10))
    kde: whether to show the density curve (default False)
    bins: number of bins for histogram (default None)
    """
    f2, (ax_box2, ax_hist2) = plt.subplots(
        nrows=2,
        sharex=True,
        gridspec_kw={"height_ratios": (0.25, 0.75)},
        figsize=figsize,
    )
    sns.boxplot(data=data, x=feature, ax=ax_box2, showmeans=True, color="violet"
    sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2, bins=bins, palette="winter"
    ) if bins else sns.histplot(data=data, x=feature, kde=kde, ax=ax_hist2)
    ax_hist2.axvline(data[feature].mean(), color="green", linestyle="--")
    ax_hist2.axvline(data[feature].median(), color="black", linestyle="-")
```

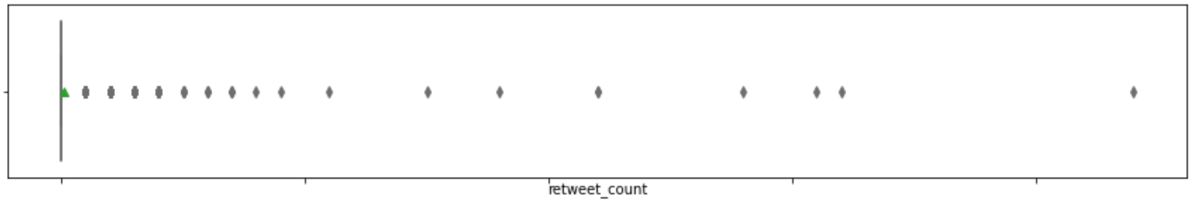
In [13]:
```
# Observation on airline_sentiment_confidence
histogram_boxplot(df, "airline_sentiment_confidence")
```

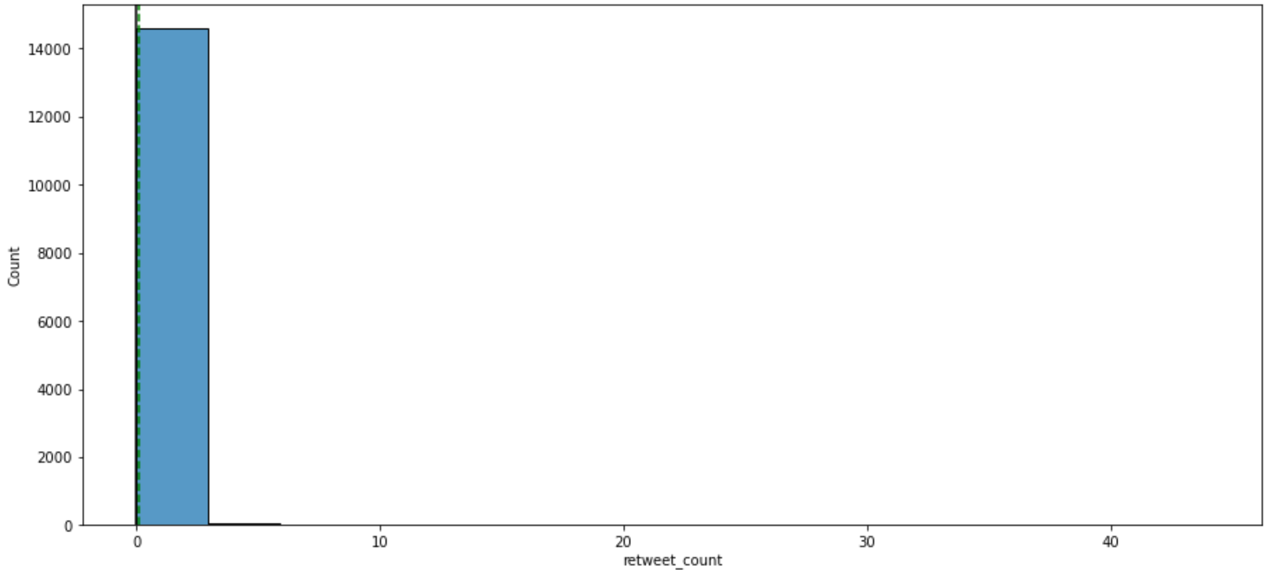


In [14]:
```
# Observation on negativereason_confidence
histogram_boxplot(df, "negativereason_confidence")
```



In [15]:
```
# Observation on retweet_count
histogram_boxplot(df, "retweet_count")
```

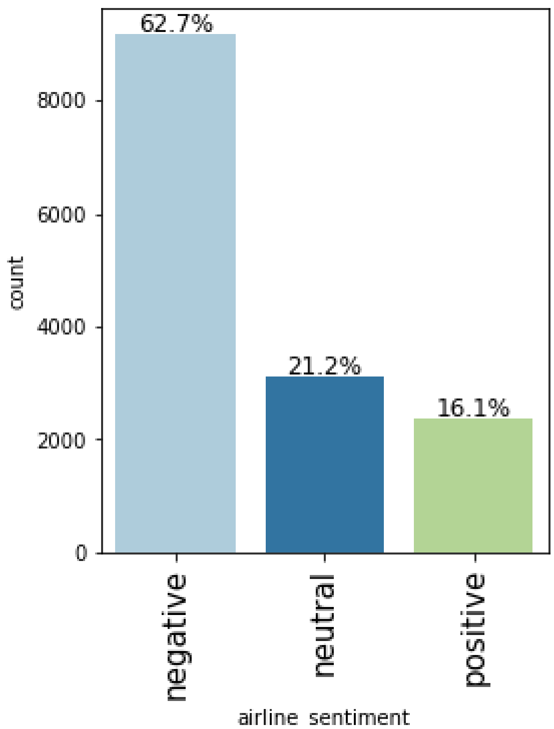




In [16]:
```
def labeled_barplot(data, feature, perc=False, n=None):
    """
    Barplot with percentage at the top

    data: dataframe
    feature: dataframe column
    perc: whether to display percentages instead of count (default is False)
    n: displays the top n category levels (default is None, i.e., display all le
    """

    total = len(data[feature])
    count = data[feature].nunique()
    if n is None:
        plt.figure(figsize=(count + 1, 5))
    else:
        plt.figure(figsize=(n + 1, 5))

    plt.xticks(rotation=90, fontsize=15)
    ax = sns.countplot(
        data=data,
        x=feature,
        palette="Paired",
        order=data[feature].value_counts().index[:n].sort_values(),
    )

    for p in ax.patches:
        if perc == True:
            label = "{:.1f}%".format(100 * p.get_height() / total)
        else:
            label = p.get_height()

        x = p.get_x() + p.get_width() / 2
```

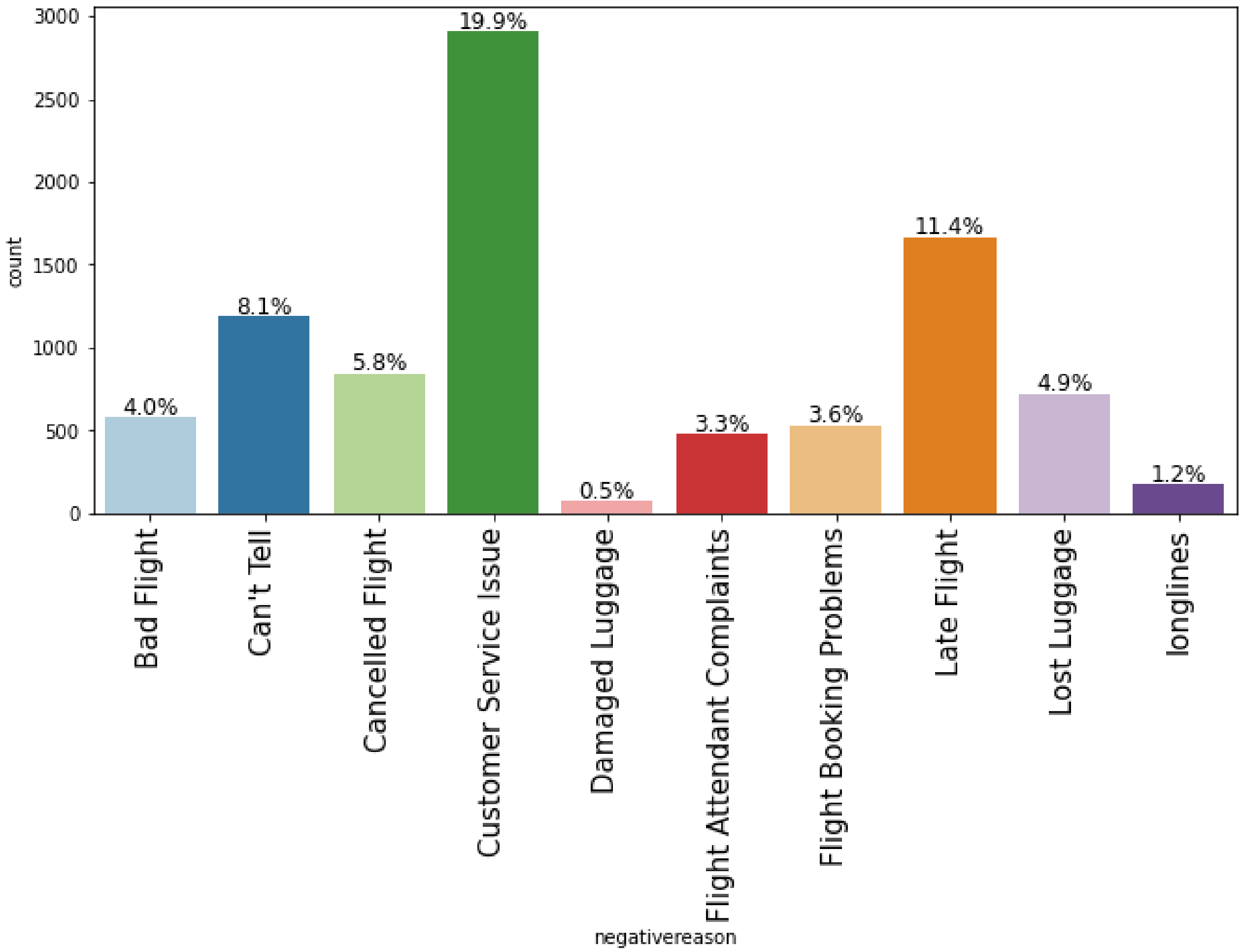
```
        y = p.get_height()

        ax.annotate(
            label,
            (x, y),
            ha="center",
            va="center",
            size=12,
            xytext=(0, 5),
            textcoords="offset points",
        )

    plt.show()
```

In [17]:
```
# Observation on airline_sentiment
labeled_barplot(df, "airline_sentiment", perc=True)
```



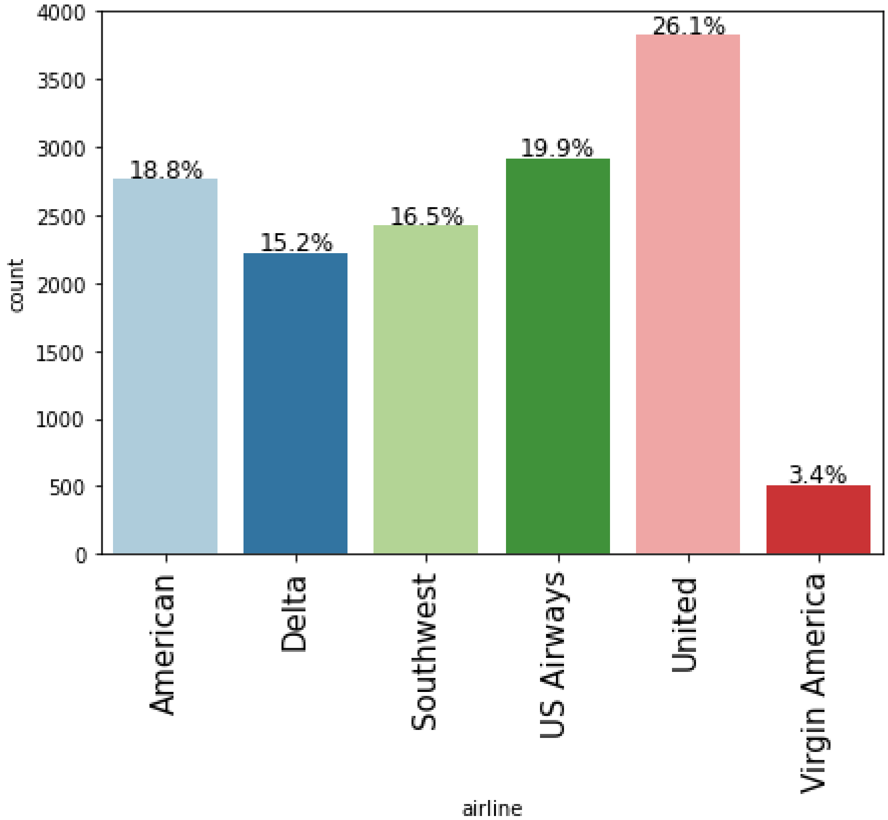
- Majority of tweets are negative, followed by neutral and positive.

In [18]:
```
# Observation on negativereason
labeled_barplot(df, "negativereason", perc=True)
```



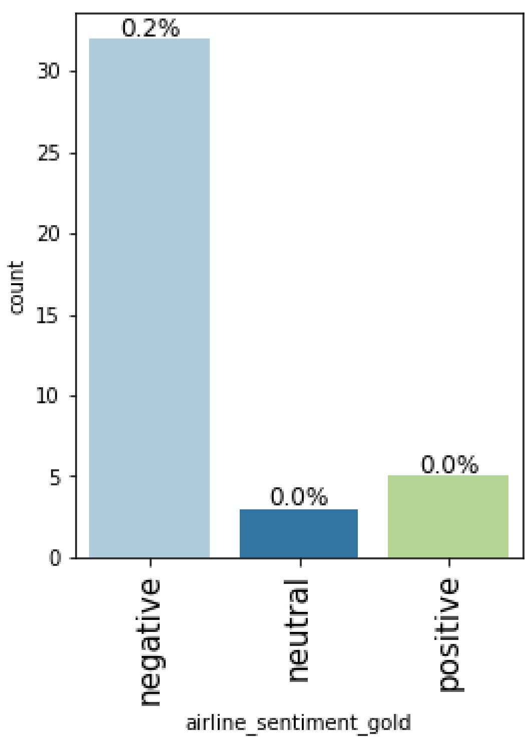
- Customer Service makes up the majority of negative tweets, followed by Late Flight.

In [19]:
```
# Observation on airline
labeled_barplot(df, "airline", perc=True)
```

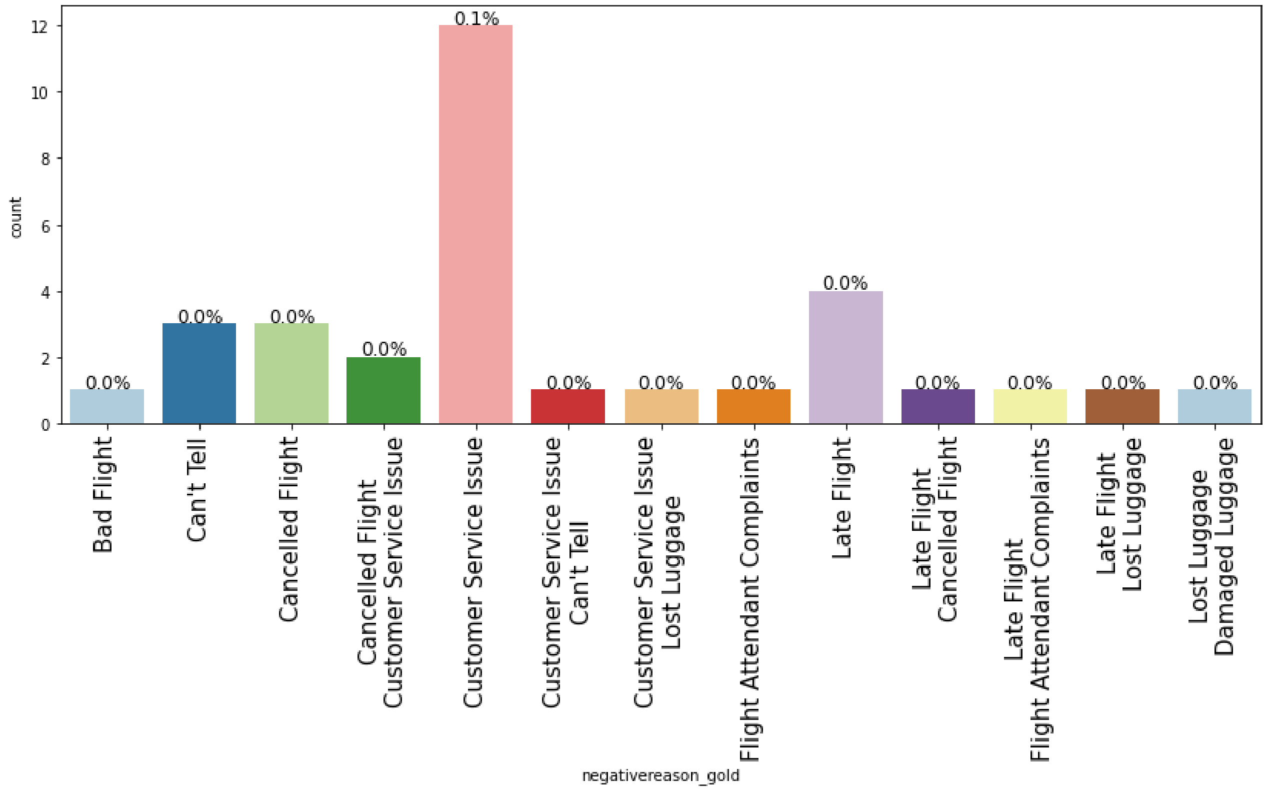


- United has the most tweets tweeted about them, followed by US Airways.

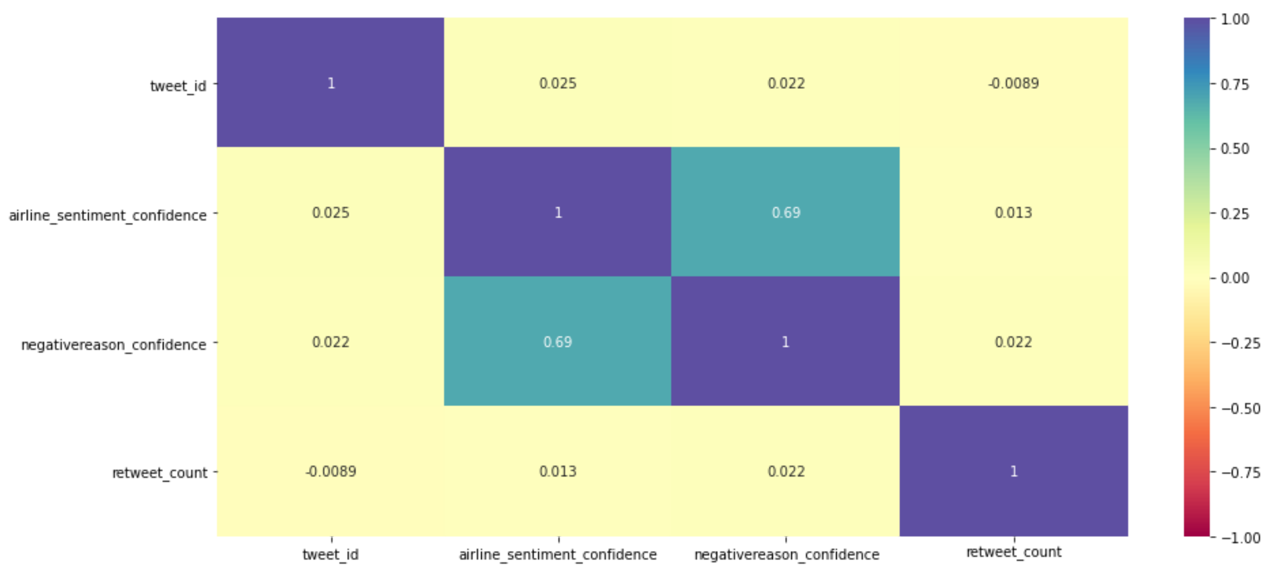
In [20]:
```
# Observation on airline_sentiment_gold
labeled_barplot(df, "airline_sentiment_gold", perc=True)
```



In [21]:
```
# Observation on negativereason_gold
labeled_barplot(df, "negativereason_gold", perc=True)
```

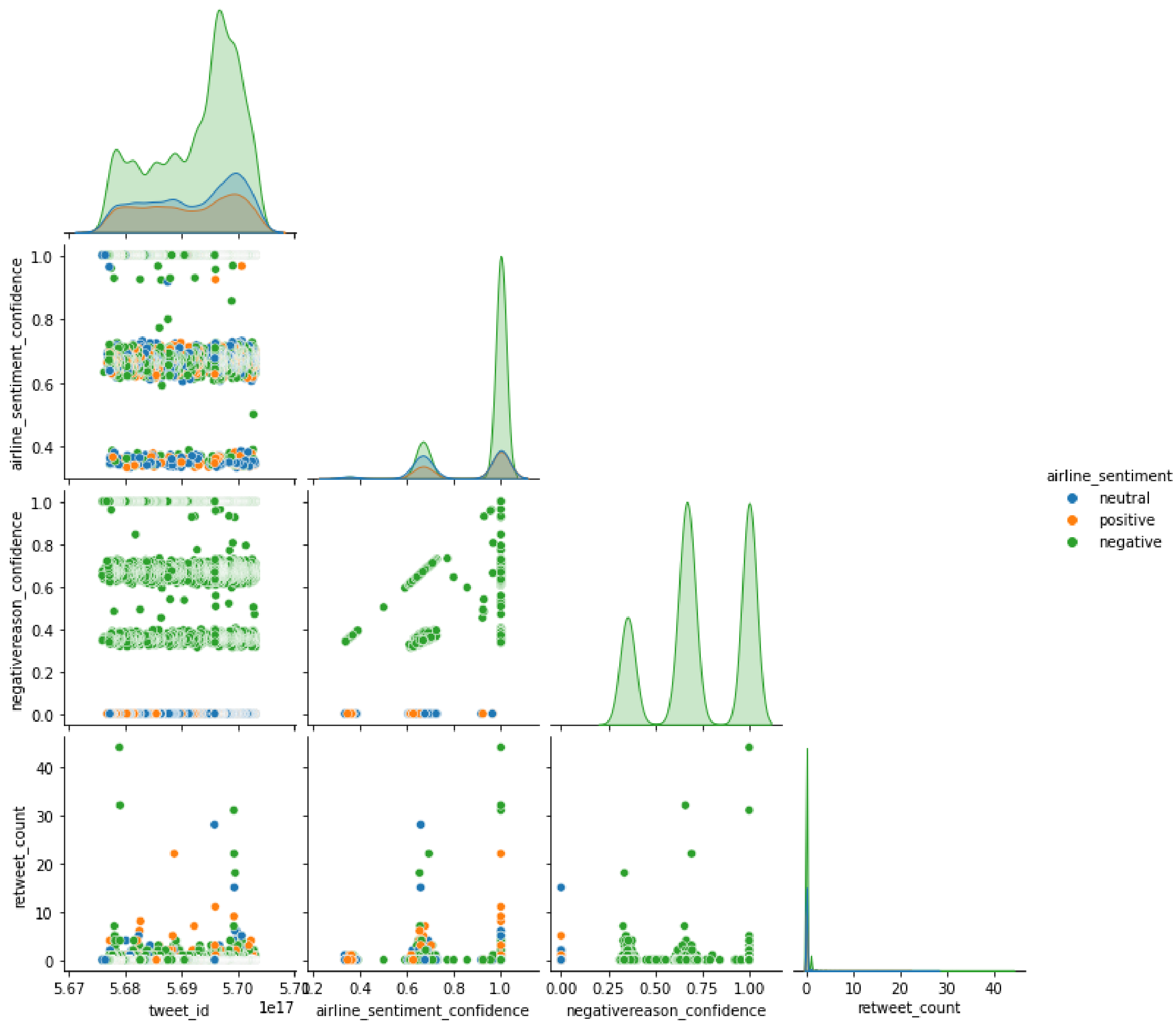


In [22]:
```
plt.figure(figsize=(15, 7))
sns.heatmap(df.corr(), annot=True, vmin=-1, vmax=1, cmap="Spectral")
plt.show()
```



In [23]:
```
sns.pairplot(df, corner=True, hue="airline_sentiment")
```

Out[23]: <seaborn.axisgrid.PairGrid at 0x14aa6951c40>

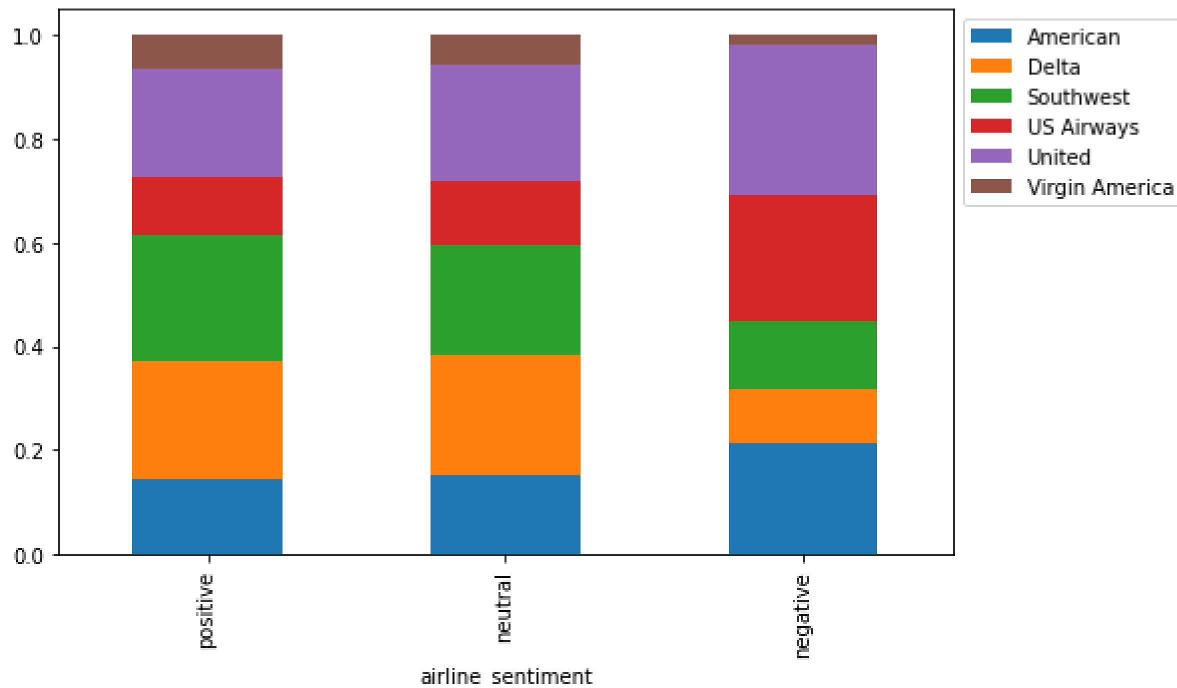


In [24]:
```
def stacked_barplot(data, predictor, target):
    """
    Print the category counts and plot a stacked bar chart

    data: dataframe
    predictor: independent variable
    target: target variable
    """
    count = data[predictor].nunique()
    sorter = data[target].value_counts().index[-1]
    tab1 = pd.crosstab(data[predictor], data[target], margins=True).sort_values(
        by=sorter, ascending=False
    )
    print(tab1)
    print("-" * 120)
    tab = pd.crosstab(data[predictor], data[target], normalize="index").sort_val
        by=sorter, ascending=False
    )
    tab.plot(kind="bar", stacked=True, figsize=(count + 5, 5))
    plt.legend(
        loc="lower left",
        frameon=False,
    )
```

```
    plt.legend(loc="upper left", bbox_to_anchor=(1, 1))
    plt.show()
```

In [25]:
```
stacked_barplot(data, "airline_sentiment", "airline")
```

```
airline             American  Delta  Southwest  US Airways  United  \
airline_sentiment
All                     2759   2222       2420        2913    3822
negative                1960    955       1186        2263    2633
neutral                  463    723        664         381     697
positive                 336    544        570         269     492

airline             Virgin America    All
airline_sentiment
All                            504  14640
negative                       181   9178
neutral                        171   3099
positive                       152   2363
------------------------------------------------------------------------------------------------------------------------
```

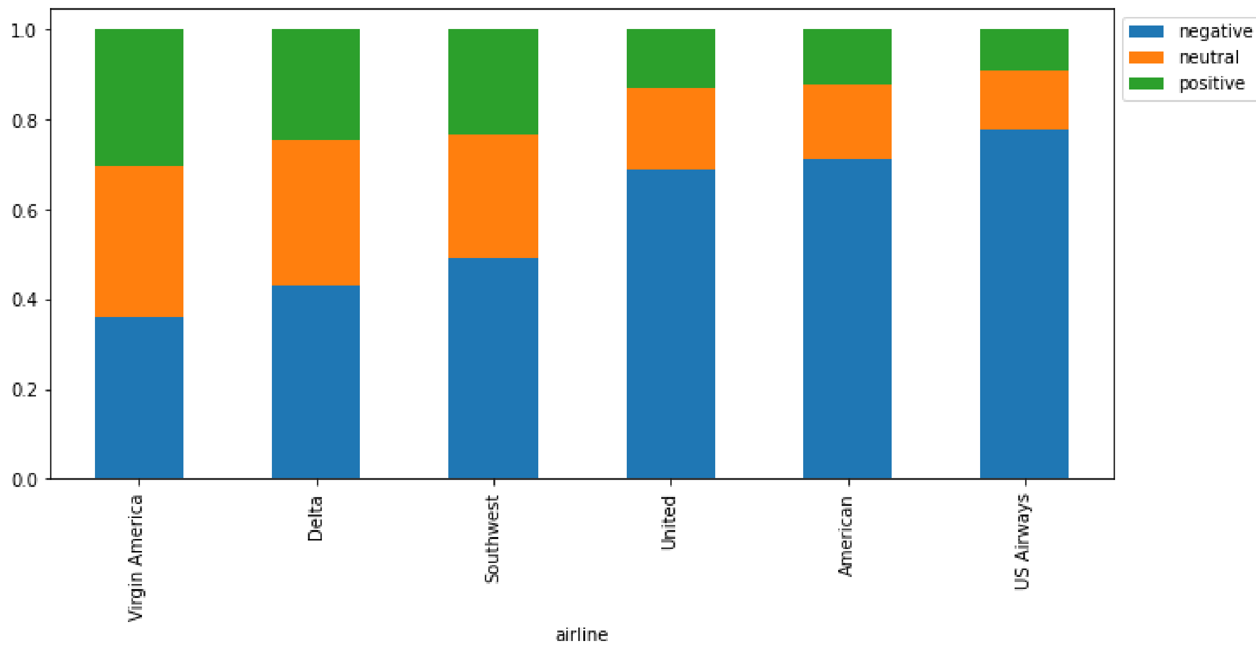


- Southwest makes up ~24% of all positive tweets, followed by Delta at ~23%, United at ~21%, American at ~14%, US Airways at ~11%, and Virgin America at ~6%.
- Delta makes up ~23% of all neutral tweets, followed by United at ~22%, Southwest at ~21%, American at ~15%, US Airways at ~12%, and Virgin America at ~6%.
- United makes up ~29% of all negative tweets, followed by US Airways at ~25%, American at ~21%, Southwest at ~13%, Delta at ~10%, and Virgin America at ~2%.

In [26]:
```
stacked_barplot(data, "airline", "airline_sentiment")
```

```
airline_sentiment  negative  neutral  positive    All
airline
All                    9178     3099      2363  14640
Southwest              1186      664       570   2420
Delta                   955      723       544   2222
```

```
United                 2633       697       492   3822
American               1960       463       336   2759
US Airways             2263       381       269   2913
Virgin America          181       171       152    504
--------------------------------------------------------------------------------
----------------------------------------
```



- ~78% of all US Airway's tweets are negative.
- 71% of all American's tweets are negative.
- ~69% of all United's tweets are negative.
- 49% of all Southwest's tweets are negative.
- ~43% of all Delta's tweets are negative.
- ~36% of all Virgin America's tweets are negative.

Word Clouds

In [27]:
```
from wordcloud import WordCloud, STOPWORDS
```

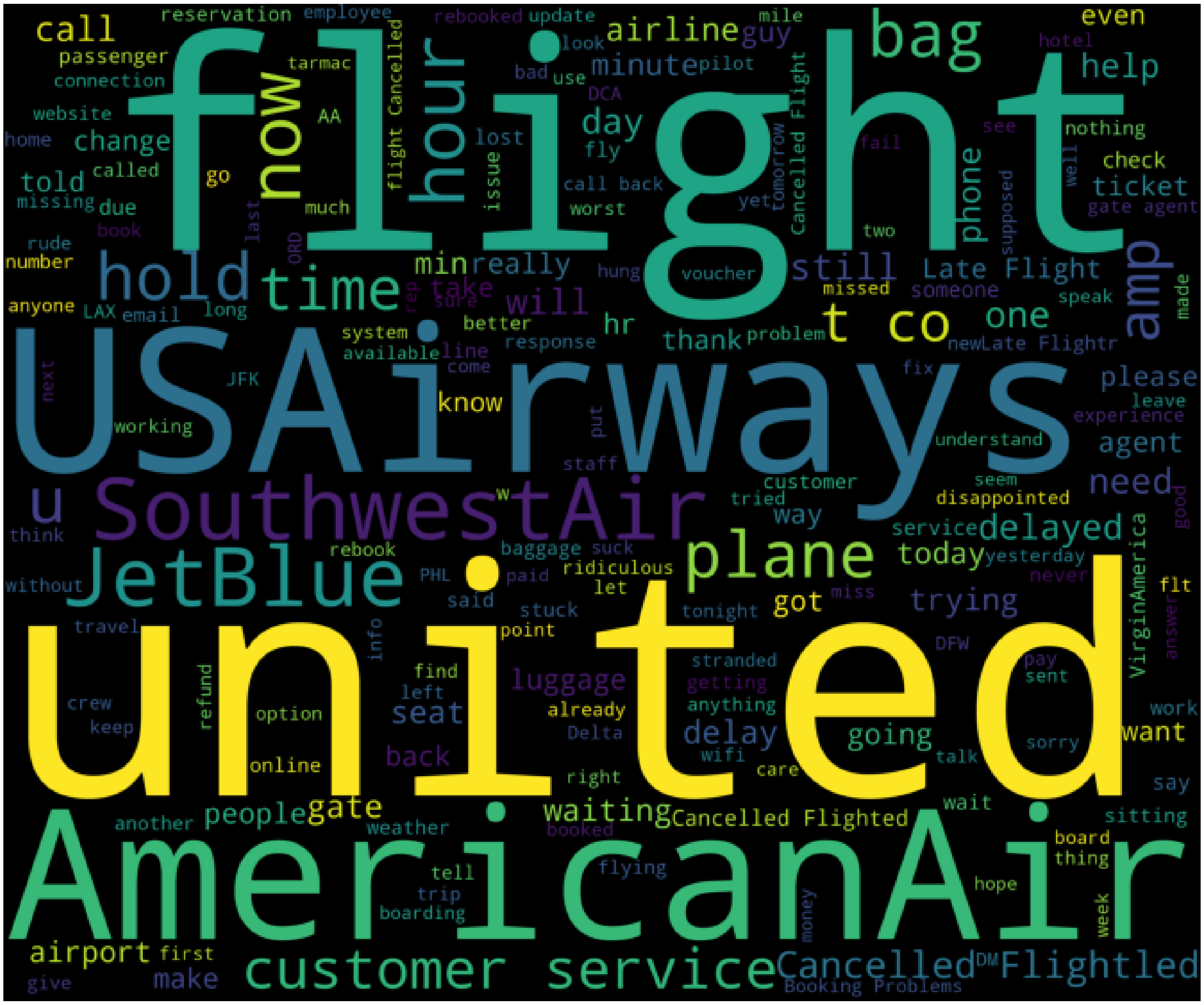
In [28]:
```
Negative_sent = data[data["airline_sentiment"] == "negative"]
words = " ".join(Negative_sent["text"])
cleaned_word = " ".join([word for word in words.split()])
```

In [29]:
```
wordcloud = WordCloud(
    stopwords=STOPWORDS, background_color="black", width=3000, height=2500
).generate(cleaned_word)
```

Negative Cloud

In [30]:
```
plt.figure(1, figsize=(12, 12))
plt.imshow(wordcloud)
```

```
plt.axis("off")
plt.show()
```



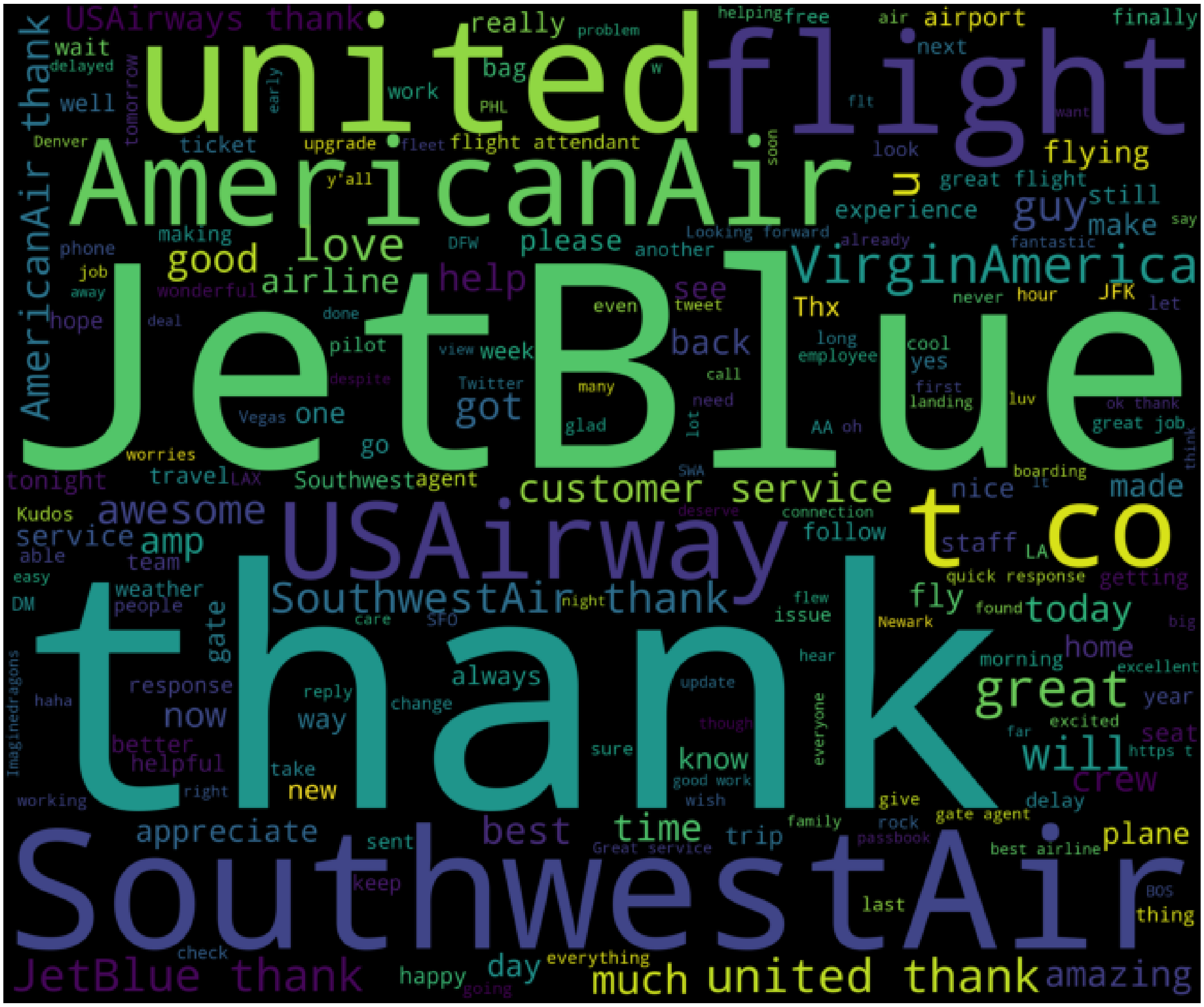
- Top 3 negative words are "united", "flight", and "USAirways".

In [31]:
```
Positive_sent = data[data["airline_sentiment"] == "positive"]
words = " ".join(Positive_sent["text"])
cleaned_word = " ".join([word for word in words.split()])
```

In [32]:
```
wordcloud = WordCloud(
    stopwords=STOPWORDS, background_color="black", width=3000, height=2500
).generate(cleaned_word)
```

Positive Cloud

In [33]:
```
plt.figure(1, figsize=(12, 12))
plt.imshow(wordcloud)
plt.axis("off")
plt.show()
```



- Top 3 positive words are "thank", "SouthwestAir", and "JetBlue".

EDA Conclusion

- Majority of tweets are negative.
- Customer Service is the topic of issue in the majority of tweets.
- United has the most tweets.
- US Airways, American, and United are the top 3 negative rated airlines.
- Delta, Southwest, and Virgin America are in better positions than the other airlines.
- Top 3 negative words are "united", "flight", and "USAirways".
- Top 3 positive words are "thank", "SouthwestAir", and "JetBlue".

Dropping Columns

The only columns I need are "text" and "airline_sentiment", so I will be dropping all others.

In [34]:
```
df.drop(
    [
```

```
            "tweet_id",
            "airline_sentiment_confidence",
            "negativereason",
            "negativereason_confidence",
            "airline",
            "airline_sentiment_gold",
            "name",
            "negativereason_gold",
            "retweet_count",
            "tweet_coord",
            "tweet_created",
            "tweet_location",
            "user_timezone",
        ],
        axis=1,
        inplace=True,
)
```

In [35]:
```
df.shape
```

Out[35]: (14640, 2)

In [36]:
```
df.head()
```

Out[36]:

	airline_sentiment	text
0	neutral	@VirginAmerica What @dhepburn said.
1	positive	@VirginAmerica plus you've added commercials t...
2	neutral	@VirginAmerica I didn't today... Must mean I n...
3	negative	@VirginAmerica it's really aggressive to blast...
4	negative	@VirginAmerica and it's a really big bad thing...

Pre-Processing

Tag Removal

In [37]:
```
def strip_html(text):
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()


df["text"] = df["text"].apply(lambda x: strip_html(x))

df.head()
```

Out[37]:

	airline_sentiment	text
0	neutral	@VirginAmerica What @dhepburn said.
1	positive	@VirginAmerica plus you've added commercials t...
2	neutral	@VirginAmerica I didn't today... Must mean I n...

	airline_sentiment	text
3	negative	@VirginAmerica it's really aggressive to blast...
4	negative	@VirginAmerica and it's a really big bad thing...

Replace Contractions

In [38]:
```
def replace_contractions(text):
    """Replace contractions in string of text"""
    return contractions.fix(text)


df["text"] = df["text"].apply(lambda x: replace_contractions(x))

df.head()
```

Out[38]:

	airline_sentiment	text
0	neutral	@VirginAmerica What @dhepburn said.
1	positive	@VirginAmerica plus you have added commercials...
2	neutral	@VirginAmerica I did not today... Must mean I ...
3	negative	@VirginAmerica it is really aggressive to blas...
4	negative	@VirginAmerica and it is a really big bad thin...

Remove Numbers/Special Characters

In [39]:
```
def remove_special_characters(text, remove_digits=True):
    special = r"[^a-zA-Z0-9\s]" if not remove_digits else r"[^a-zA-z\s]"
    text = re.sub(special, "", text)
    return text


df["text"] = df["text"].apply(lambda x: remove_special_characters(x))

df.head()
```

Out[39]:

	airline_sentiment	text
0	neutral	VirginAmerica What dhepburn said
1	positive	VirginAmerica plus you have added commercials ...
2	neutral	VirginAmerica I did not today Must mean I need...
3	negative	VirginAmerica it is really aggressive to blast...
4	negative	VirginAmerica and it is a really big bad thing...

Conversion to Lowercase

In [40]:
```
def to_lowercase(text):
    """Convert all characters to lowercase"""
```

```
    lower = text.lower()
    return lower


df["text"] = df["text"].apply(lambda x: to_lowercase(x))

df.head()
```

Out[40]:

	airline_sentiment	text
0	neutral	virginamerica what dhepburn said
1	positive	virginamerica plus you have added commercials ...
2	neutral	virginamerica i did not today must mean i need...
3	negative	virginamerica it is really aggressive to blast...
4	negative	virginamerica and it is a really big bad thing...

Tokenization

In [41]:

```
df["text"] = df.apply(
    lambda row: nltk.word_tokenize(row["text"]), axis=1
)  # Tokenization of data

df.head()
```

Out[41]:

	airline_sentiment	text
0	neutral	[virginamerica, what, dhepburn, said]
1	positive	[virginamerica, plus, you, have, added, commer...
2	neutral	[virginamerica, i, did, not, today, must, mean...
3	negative	[virginamerica, it, is, really, aggressive, to...
4	negative	[virginamerica, and, it, is, a, really, big, b...

Remove Stopwords

In [42]:

```
stopwords = stopwords.words("english")

customlist = [
    "not",
    "could",
    "did",
    "does",
    "had",
    "has",
    "have",
    "is",
    "ma",
    "might",
    "must",
    "need",
    "shall",
    "should",
```

```
    "was",
    "were",
    "will",
    "would",
]

# Set custom stop-word's list as not, could etc. words matter in Sentiment, so n

stopwords = list(set(stopwords) - set(customlist))
```

In [43]:

```
def remove_stopwords(words):
    """Remove stop words from list of tokenized words"""
    new_words = []
    for word in words:
        if word not in stopwords:
            new_words.append(word)
    return new_words


df["text"] = df["text"].apply(lambda x: remove_stopwords(x))

df.head()
```

Out[43]:

	airline_sentiment	text
0	neutral	[virginamerica, dhepburn, said]
1	positive	[virginamerica, plus, have, added, commercials...
2	neutral	[virginamerica, did, not, today, must, mean, n...
3	negative	[virginamerica, is, really, aggressive, blast,...
4	negative	[virginamerica, is, really, big, bad, thing]

Lemmatize

In [44]:

```
lemmatizer = WordNetLemmatizer()


def lemmatize_list(words):
    new_words = []
    for word in words:
        new_words.append(lemmatizer.lemmatize(word, pos="v"))
    return new_words


df["text"] = df["text"].apply(lambda x: lemmatize_list(x))

df.head()
```

Out[44]:

	airline_sentiment	text
0	neutral	[virginamerica, dhepburn, say]
1	positive	[virginamerica, plus, have, add, commercials, ...
2	neutral	[virginamerica, do, not, today, must, mean, ne...

	airline_sentiment	text
3	negative	[virginamerica, be, really, aggressive, blast,...
4	negative	[virginamerica, be, really, big, bad, thing]

Word List to Text String

In [45]:
```
def join_words(words):
    return " ".join(words)


df["text"] = df["text"].apply(lambda x: join_words(x))

df.head()
```

Out[45]:

	airline_sentiment	text
0	neutral	virginamerica dhepburn say
1	positive	virginamerica plus have add commercials experi...
2	neutral	virginamerica do not today must mean need take...
3	negative	virginamerica be really aggressive blast obnox...
4	negative	virginamerica be really big bad thing

In [46]:
```
df.tail()
```

Out[46]:

	airline_sentiment	text
14635	positive	americanair thank get different flight chicago
14636	negative	americanair leave minutes late flight warn com...
14637	neutral	americanair please bring american airlines bla...
14638	negative	americanair have money change flight not answe...
14639	neutral	americanair have ppl need know many seat next ...

In [47]:
```
np.random.seed(2)
df.sample(10)
```

Out[47]:

	airline_sentiment	text
8917	positive	jetblue thank
9534	negative	usairways be go flight phl phx mins estimate d...
12875	neutral	americanair yes thank find do not see gray tab...
4601	positive	southwestair big kudos staff today dallas love...
4308	negative	unite last night wait forever gate someone cor...
10981	negative	usairways be load onto plane hours late flight...
11037	negative	usairways thank continue worst airline do not ...

	airline_sentiment	text
12991	neutral	americanair thank be tell go airport check age...
5685	positive	southwestair oh gosh go dm thank
5784	positive	southwestair appreciate reply hopefully lax ag...

In [48]:
```
df["airline_sentiment"] = df["airline_sentiment"].astype("category")
df["airline_sentiment"] = df["airline_sentiment"].cat.codes

df.head()
```

Out[48]:

	airline_sentiment	text
0	1	virginamerica dhepburn say
1	2	virginamerica plus have add commercials experi...
2	1	virginamerica do not today must mean need take...
3	0	virginamerica be really aggressive blast obnox...
4	0	virginamerica be really big bad thing

Encoded "airline_sentiment" as I will try different classification algorithms.

- 0 = negative
- 1 = neutral
- 2 = positive

Vectorization

CountVectorizer

In [49]:
```
# Vectorization (Convert text data to numbers).
from sklearn.feature_extraction.text import CountVectorizer

bow_vec = CountVectorizer(
    max_features=2000
)  # Keep only 2000 features as number of features will increase the processing
data_features1 = bow_vec.fit_transform(df["text"])

data_features1 = data_features1.toarray()  # Convert the data features to array.
```

In [50]:
```
data_features1.shape
```

Out[50]:
```
(14640, 2000)
```

In [51]:
```
X1 = data_features1

y1 = df.airline_sentiment
```

In [52]:
```
# Split data into training and testing set.

from sklearn.model_selection import train_test_split

X1_train, X1_test, y1_train, y1_test = train_test_split(
    X1, y1, stratify=y1, test_size=0.25, random_state=42
)
```

TF-IDF

In [53]:
```
# Using TfidfVectorizer to convert text data to numbers.

from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(max_features=2000)
data_features2 = vectorizer.fit_transform(df["text"])

data_features2 = data_features2.toarray()
```

In [54]:
```
data_features2.shape
```

Out[54]:
```
(14640, 2000)
```

In [55]:
```
X2 = data_features2

y2 = df.airline_sentiment
```

In [56]:
```
# Split data into training and testing set.

from sklearn.model_selection import train_test_split

X2_train, X2_test, y2_train, y2_test = train_test_split(
    X2, y2, stratify=y2, test_size=0.25, random_state=42
)
```

1 = CountVectorizer

2 = TF-IDF

Models

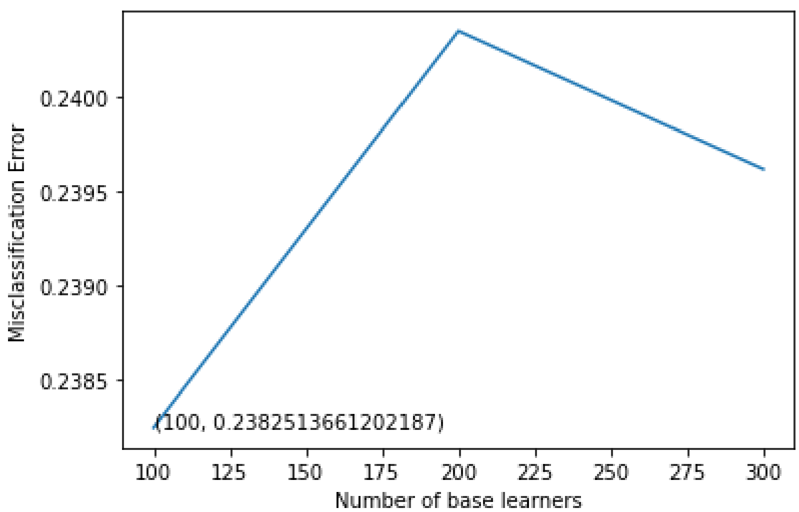
Random Forest on CountVectorizer

In [57]:
```
# Finding optimal number of base learners using k-fold CV ->
base_ln = np.arange(100, 400, 100).tolist()
base_ln
```

Out[57]: [100, 200, 300]

In [58]:
```
# K-Fold Cross - validation .
cv_scores = []
for b in base_ln:
    clf = RandomForestClassifier(n_estimators=b)
    scores = cross_val_score(clf, X1_train, y1_train, cv=5, scoring="accuracy")
    cv_scores.append(scores.mean())
```

In [59]:
```
# plotting the error as k increases
error = [1 - x for x in cv_scores]  # error corresponds to each nu of estimator
optimal_learners = base_ln[
    error.index(min(error))
]  # Selection of optimal nu of n_estimator corresponds to minimum error.
plt.plot(
    base_ln, error
)  # Plot between each nu of estimator and misclassification error
xy = (optimal_learners, min(error))
plt.annotate("(%s, %s)" % xy, xy=xy, textcoords="data")
plt.xlabel("Number of base learners")
plt.ylabel("Misclassification Error")
plt.show()
```



In [60]:
```
# Training the best model and calculating accuracy on test data .
clf = RandomForestClassifier(n_estimators=optimal_learners)
clf.fit(X1_train, y1_train)
clf.score(X1_test, y1_test)
count_vectorizer_predicted = clf.predict(X1_test)
print(
    classification_report(
        y1_test, count_vectorizer_predicted, target_names=["0", "1", "2"]
    )
)
print(
    "Accuracy of the model is : ", accuracy_score(y1_test, count_vectorizer_pred
)
```

```
              precision    recall  f1-score   support

           0       0.81      0.91      0.86      2294
```

```
           1       0.62      0.47      0.53       775
           2       0.72      0.60      0.66       591

    accuracy                           0.77      3660
   macro avg       0.72      0.66      0.68      3660
weighted avg       0.76      0.77      0.76      3660

Accuracy of the model is :  0.769672131147541
```

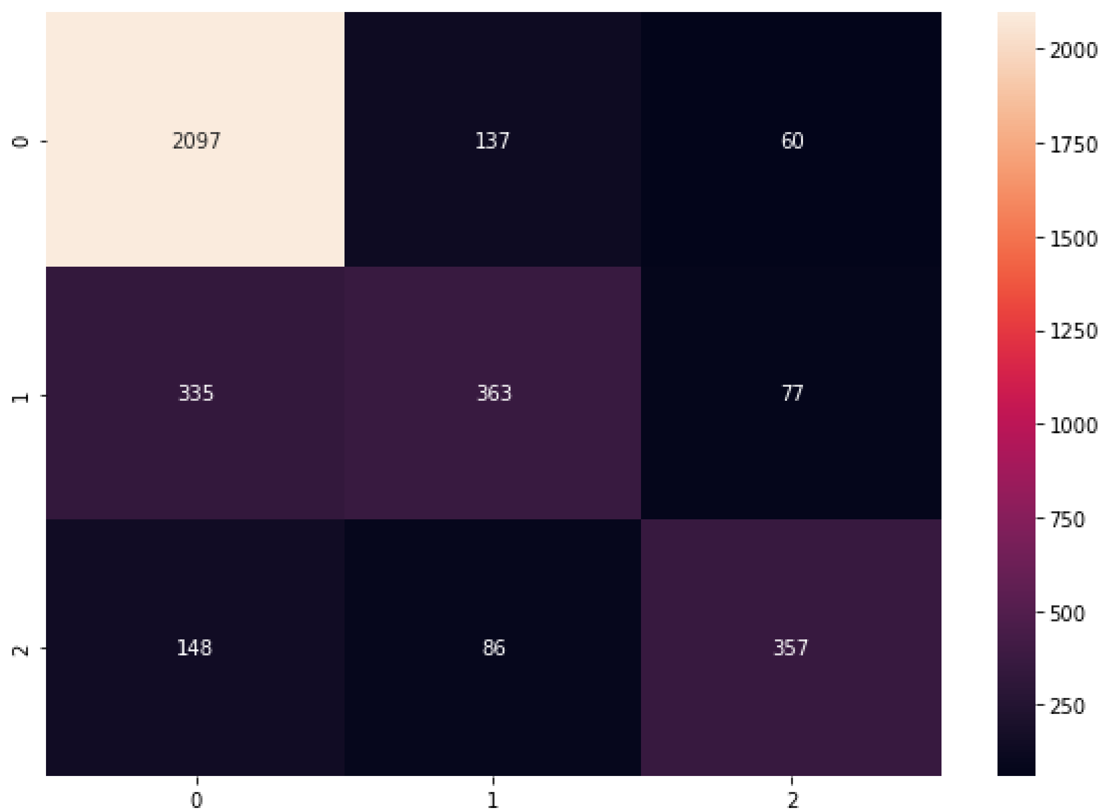
- 76.9

In [61]:

```
from sklearn.metrics import confusion_matrix

conf_mat = confusion_matrix(y1_test, count_vectorizer_predicted)

print(conf_mat)

df_cm = pd.DataFrame(
    conf_mat, index=[i for i in ["0", "1", "2"]], columns=[i for i in ["0", "1",
)
plt.figure(figsize=(10, 7))
sns.heatmap(df_cm, annot=True, fmt="g")
```

```
[[2097  137   60]
 [ 335  363   77]
 [ 148   86  357]]
```

Out[61]: <AxesSubplot:>

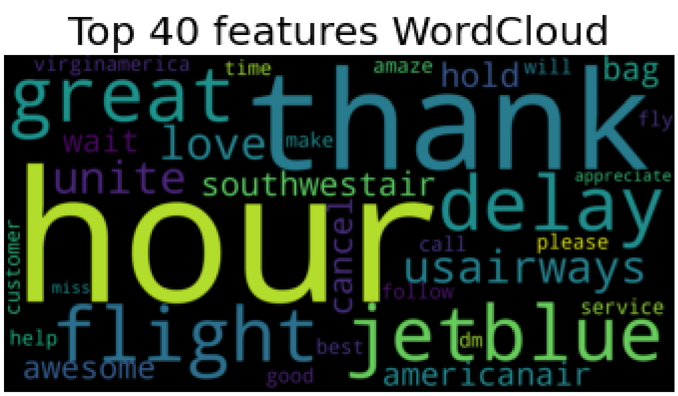


In [62]:

```
all_features = (
    bow_vec.get_feature_names()
)  # Instantiate the feature from the vectorizer
top_features = (
    ""  # Addition of top 40 feature into top_feature after training the model
```

```
)
feat = clf.feature_importances_
features = np.argsort(feat)[::-1]
for i in features[0:40]:
    top_features += all_features[i]
    top_features += ","

print(top_features)

print(" ")
print(" ")

wordcloud = WordCloud(
    background_color="black", colormap="viridis", width=2000, height=1000
).generate(top_features)

# Display the generated image:
plt.imshow(wordcloud, interpolation="bilinear")
plt.figure(1, figsize=(14, 11), frameon="equal")
plt.title("Top 40 features WordCloud", fontsize=20)
plt.axis("off")
plt.show()
```

```
thank,not,be,jetblue,great,delay,have,flight,usairways,love,unite,southwestair,h
ours,hold,cancel,get,americanair,awesome,wait,bag,virginamerica,will,best,call,a
maze,hour,please,do,follow,customer,service,help,fly,dm,time,make,would,good,app
reciate,miss,
```



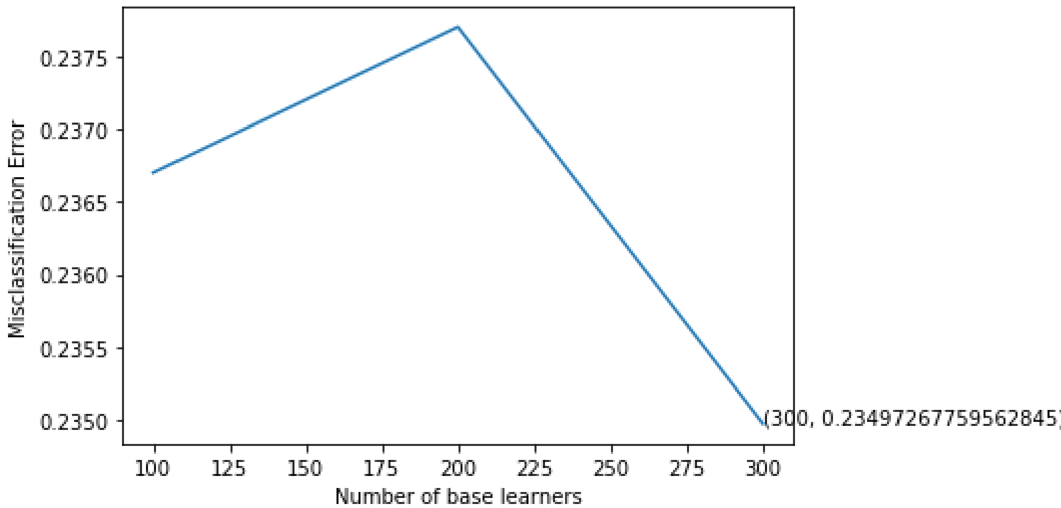
Random Forest on TF-IDF

In [63]:
```
# Finding optimal number of base learners using k-fold CV ->
base_ln = np.arange(100, 400, 100).tolist()
base_ln
```

Out[63]:
```
[100, 200, 300]
```

In [64]:
```
# K-Fold Cross - validation .
cv_scores = []
for b in base_ln:
    clf2 = RandomForestClassifier(n_estimators=b)
    scores = cross_val_score(clf2, X2_train, y2_train, cv=5, scoring="accuracy")
    cv_scores.append(scores.mean())
```

In [65]:
```
# plotting the error as k increases
error = [1 - x for x in cv_scores]  # error corresponds to each nu of estimator
optimal_learners = base_ln[
    error.index(min(error))
]  # Selection of optimal nu of n_estimator corresponds to minimum error.
plt.plot(
    base_ln, error
)  # Plot between each nu of estimator and misclassification error
xy = (optimal_learners, min(error))
plt.annotate("(%s, %s)" % xy, xy=xy, textcoords="data")
plt.xlabel("Number of base learners")
plt.ylabel("Misclassification Error")
plt.show()
```



In [66]:
```
# Training the best model and calculating accuracy on test data .
clf2 = RandomForestClassifier(n_estimators=optimal_learners)
clf2.fit(X2_train, y2_train)
clf2.score(X2_test, y2_test)
tf_idf_predicted = clf2.predict(X2_test)
print(classification_report(y2_test, tf_idf_predicted, target_names=["0", "1", "
print("Accuracy of the model is : ", accuracy_score(y2_test, tf_idf_predicted))
```

```
              precision    recall  f1-score   support

           0       0.78      0.94      0.85      2294
           1       0.64      0.40      0.49       775
           2       0.74      0.52      0.61       591

    accuracy                           0.76      3660
   macro avg       0.72      0.62      0.65      3660
weighted avg       0.74      0.76      0.74      3660

Accuracy of the model is :  0.7576502732240438
```

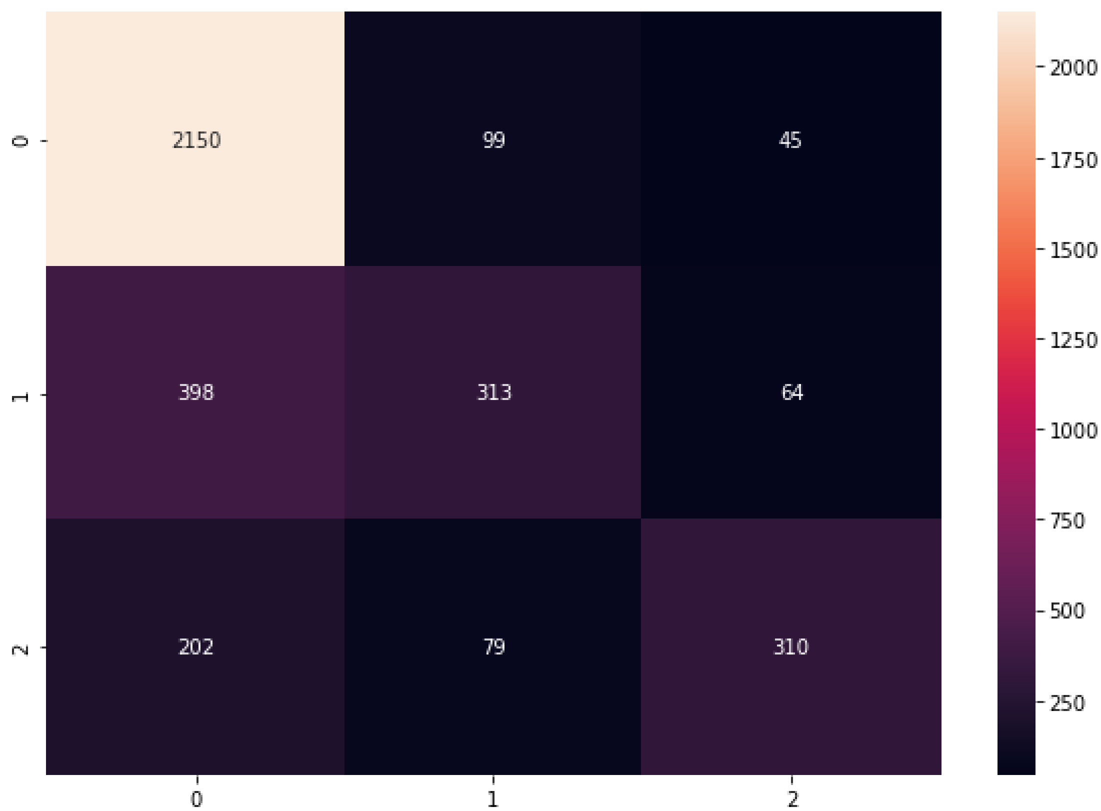
- 75.7

In [67]:
```
conf_mat2 = confusion_matrix(y2_test, tf_idf_predicted)

print(conf_mat2)
```

```
df_cm2 = pd.DataFrame(
    conf_mat2, index=[i for i in ["0", "1", "2"]], columns=[i for i in ["0", "1"
)
plt.figure(figsize=(10, 7))
sns.heatmap(df_cm2, annot=True, fmt="g")
```

```
[[2150   99   45]
 [ 398  313   64]
 [ 202   79  310]]
```

Out[67]: <AxesSubplot:>

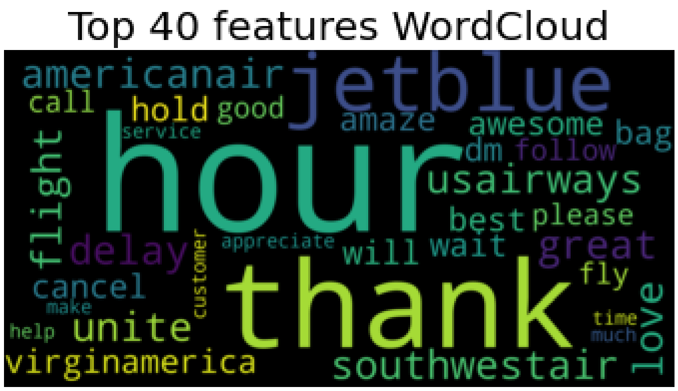


In [68]:

```
all_features = (
    vectorizer.get_feature_names()
)  # Instantiate the feature from the vectorizer
top_features = (
    ""  # Addition of top 40 feature into top_feature after training the model
)
feat = clf2.feature_importances_
features = np.argsort(feat)[::-1]
for i in features[0:40]:
    top_features += all_features[i]
    top_features += ","

print(top_features)

print(" ")
print(" ")

wordcloud = WordCloud(
    background_color="black", colormap="viridis", width=2000, height=1000
).generate(top_features)

# Display the generated image:
```

```
plt.imshow(wordcloud, interpolation="bilinear")
plt.figure(1, figsize=(14, 11), frameon="equal")
plt.title("Top 40 features WordCloud", fontsize=20)
plt.axis("off")
plt.show()
```

```
thank,not,jetblue,be,southwestair,usairways,unite,americanair,great,flight,have,
delay,love,virginamerica,get,will,awesome,cancel,hold,hours,best,dm,wait,bag,ama
ze,please,follow,do,call,good,fly,appreciate,make,service,customer,hour,help,tim
e,would,much,
```



XGBoost on CountVectorizer

In [69]:
```
# XGBoost
from xgboost import XGBClassifier

xgb1 = XGBClassifier(random_state=1, eval_metric="logloss")
xgb1.fit(X1_train, y1_train)
```

Out[69]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, eval_metric='logloss',
              gamma=0, gpu_id=-1, importance_type='gain',
              interaction_constraints='', learning_rate=0.300000012,
              max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=100, n_jobs=12,
              num_parallel_tree=1, objective='multi:softprob', random_state=1,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=None, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [70]:
```
xgb1_predicted = xgb1.predict(X1_train)
print(classification_report(y1_train, xgb1_predicted, target_names=["0", "1", "2
print("Accuracy of the model is : ", accuracy_score(y1_train, xgb1_predicted))
```

```
              precision    recall  f1-score   support

           0       0.87      0.95      0.91      6884
           1       0.77      0.62      0.68      2324
           2       0.85      0.75      0.80      1772

    accuracy                           0.85     10980
   macro avg       0.83      0.77      0.80     10980
weighted avg       0.84      0.85      0.84     10980

Accuracy of the model is :  0.8471766848816029
```

In [71]:
```
xgb1_predicted_test = xgb1.predict(X1_test)
print(classification_report(y1_test, xgb1_predicted_test, target_names=["0", "1"
print("Accuracy of the model is : ", accuracy_score(y1_test, xgb1_predicted_test
```

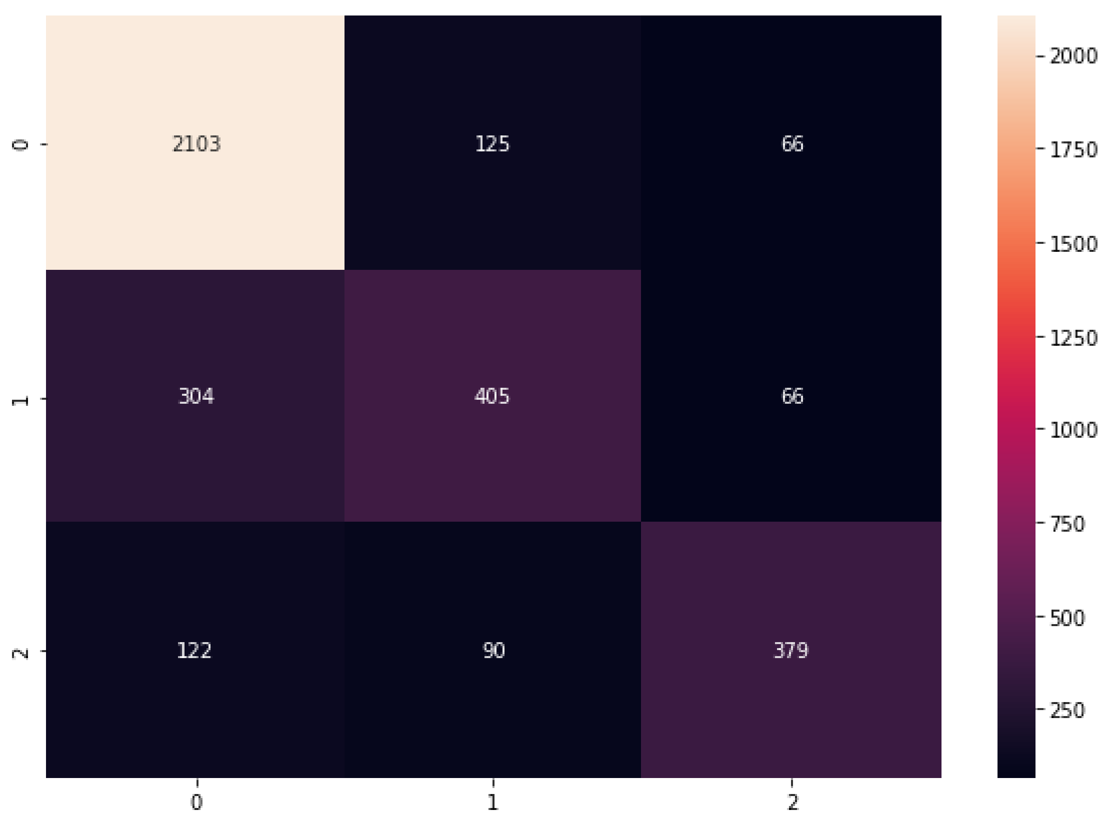
```
              precision    recall  f1-score   support

           0       0.83      0.92      0.87      2294
           1       0.65      0.52      0.58       775
           2       0.74      0.64      0.69       591

    accuracy                           0.79      3660
   macro avg       0.74      0.69      0.71      3660
weighted avg       0.78      0.79      0.78      3660

Accuracy of the model is :  0.7887978142076503
```

- 78.8

In [72]:
```
conf_mat3 = confusion_matrix(y1_test, xgb1_predicted_test)

print(conf_mat3)

df_cm3 = pd.DataFrame(
    conf_mat3, index=[i for i in ["0", "1", "2"]], columns=[i for i in ["0", "1"
)
plt.figure(figsize=(10, 7))
sns.heatmap(df_cm3, annot=True, fmt="g")
```

```
[[2103  125   66]
 [ 304  405   66]
 [ 122   90  379]]
```

Out[72]: <AxesSubplot:>



XGBoost on TF-IDF

In [73]:
```
xgb2 = XGBClassifier(random_state=1, eval_metric="logloss")
xgb2.fit(X2_train, y2_train)
```

Out[73]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, eval_metric='logloss',
              gamma=0, gpu_id=-1, importance_type='gain',
              interaction_constraints='', learning_rate=0.300000012,
              max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=100, n_jobs=12,
              num_parallel_tree=1, objective='multi:softprob', random_state=1,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=None, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [74]:
```
xgb2_predicted = xgb2.predict(X2_train)
print(classification_report(y2_train, xgb2_predicted, target_names=["0", "1", "2
print("Accuracy of the model is : ", accuracy_score(y2_train, xgb2_predicted))
```

```
              precision    recall  f1-score   support

           0       0.88      0.97      0.92      6884
           1       0.83      0.67      0.74      2324
           2       0.90      0.77      0.83      1772

    accuracy                           0.87     10980
   macro avg       0.87      0.80      0.83     10980
weighted avg       0.87      0.87      0.87     10980

Accuracy of the model is :  0.8724043715846994
```

In [75]:
```
xgb2_predicted_test = xgb2.predict(X2_test)
print(classification_report(y2_test, xgb2_predicted_test, target_names=["0", "1"
print("Accuracy of the model is : ", accuracy_score(y2_test, xgb2_predicted_test
```

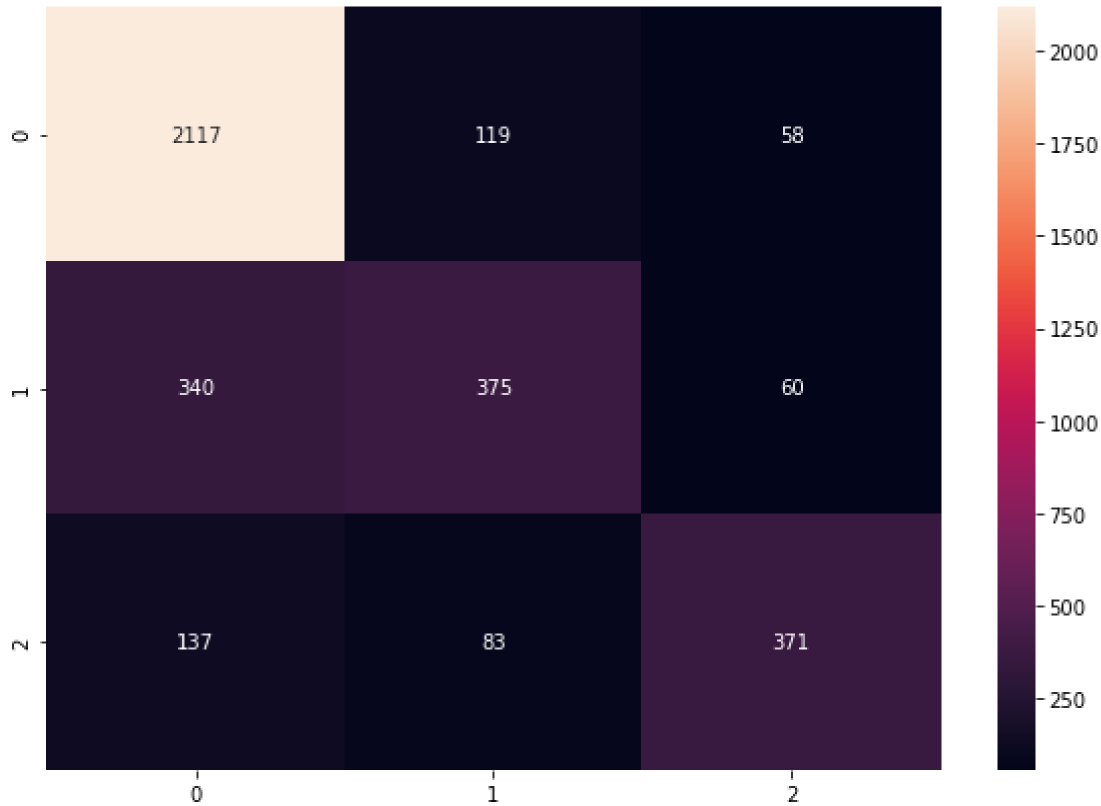
```
              precision    recall  f1-score   support

           0       0.82      0.92      0.87      2294
           1       0.65      0.48      0.55       775
           2       0.76      0.63      0.69       591

    accuracy                           0.78      3660
   macro avg       0.74      0.68      0.70      3660
weighted avg       0.77      0.78      0.77      3660

Accuracy of the model is :  0.7822404371584699
```

- 78.2

In [76]:
```
conf_mat4 = confusion_matrix(y2_test, xgb2_predicted_test)

print(conf_mat4)

df_cm4 = pd.DataFrame(
    conf_mat4, index=[i for i in ["0", "1", "2"]], columns=[i for i in ["0", "1"
)
plt.figure(figsize=(10, 7))
sns.heatmap(df_cm4, annot=True, fmt="g")
```

```
[[2117  119   58]
 [ 340  375   60]
 [ 137   83  371]]
```

Out[76]: <AxesSubplot:>



Naive Bayes on CountVectorizer

In [77]:
```
from sklearn.naive_bayes import MultinomialNB

NB = MultinomialNB()
NB.fit(X1_train, y1_train)
```

Out[77]: MultinomialNB()

In [78]:
```
NB_predicted = NB.predict(X1_train)
print(classification_report(y1_train, NB_predicted, target_names=["0", "1", "2"]
print("Accuracy of the model is : ", accuracy_score(y1_train, NB_predicted))
```

```
              precision    recall  f1-score   support

           0       0.86      0.91      0.88      6884
           1       0.70      0.59      0.64      2324
           2       0.78      0.77      0.77      1772

    accuracy                           0.82     10980
   macro avg       0.78      0.76      0.77     10980
weighted avg       0.81      0.82      0.81     10980

Accuracy of the model is :  0.8176684881602915
```

In [79]:
```
NB_predicted_test = NB.predict(X1_test)
print(classification_report(y1_test, NB_predicted_test, target_names=["0", "1",
print("Accuracy of the model is : ", accuracy_score(y1_test, NB_predicted_test))
```

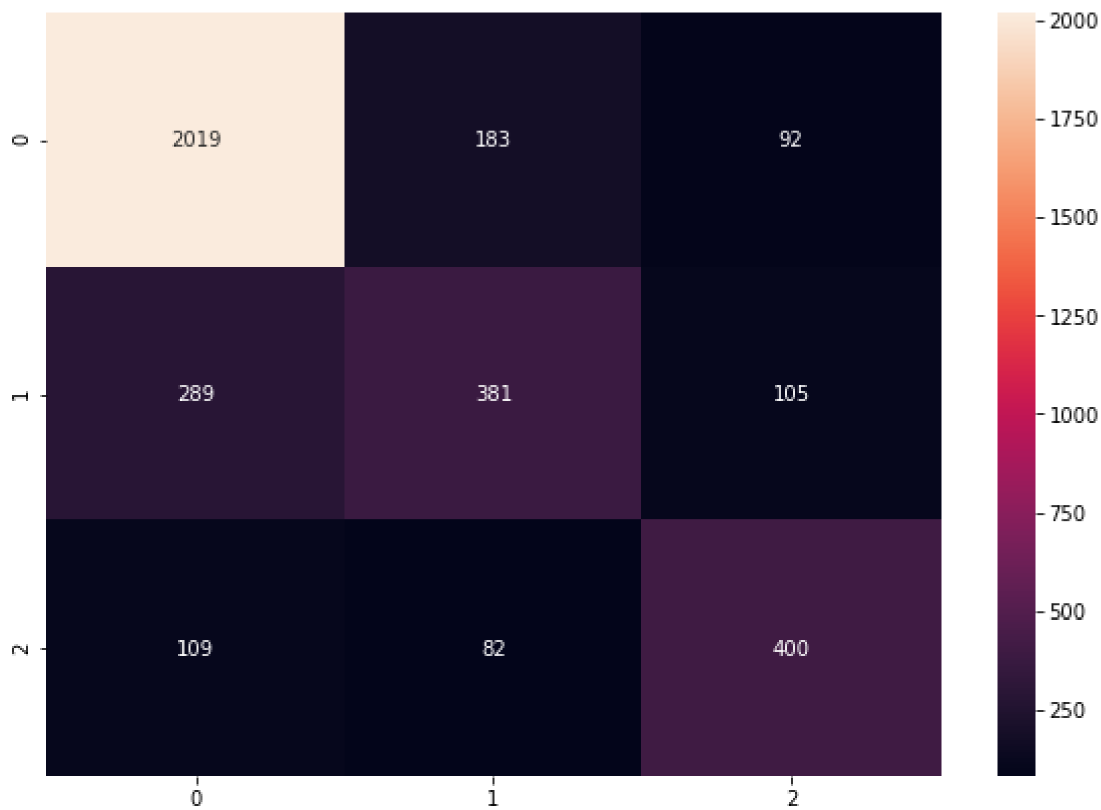
```
              precision    recall  f1-score   support

           0       0.84      0.88      0.86      2294
           1       0.59      0.49      0.54       775
           2       0.67      0.68      0.67       591

    accuracy                           0.77      3660
   macro avg       0.70      0.68      0.69      3660
weighted avg       0.76      0.77      0.76      3660

Accuracy of the model is :  0.7650273224043715
```

- 76.5

In [80]:
```
conf_mat5 = confusion_matrix(y1_test, NB_predicted_test)

print(conf_mat5)

df_cm5 = pd.DataFrame(
    conf_mat5, index=[i for i in ["0", "1", "2"]], columns=[i for i in ["0", "1"
)
plt.figure(figsize=(10, 7))
sns.heatmap(df_cm5, annot=True, fmt="g")
```

```
[[2019  183   92]
 [ 289  381  105]
 [ 109   82  400]]
```

Out[80]: <AxesSubplot:>



Naive Bayes on TF-IDF

In [81]:
```
NB2 = MultinomialNB()
NB2.fit(X2_train, y2_train)
```

Out[81]:
```
MultinomialNB()
```

In [82]:
```
NB2_predicted = NB2.predict(X2_train)
print(classification_report(y2_train, NB2_predicted, target_names=["0", "1", "2"
print("Accuracy of the model is : ", accuracy_score(y2_train, NB2_predicted))
```

```
              precision    recall  f1-score   support

           0       0.75      0.98      0.85      6884
           1       0.80      0.35      0.49      2324
           2       0.90      0.50      0.64      1772

    accuracy                           0.77     10980
   macro avg       0.82      0.61      0.66     10980
weighted avg       0.79      0.77      0.74     10980

Accuracy of the model is :  0.7716757741347905
```

In [83]:
```
NB2_predicted_test = NB2.predict(X2_test)
print(classification_report(y2_test, NB2_predicted_test, target_names=["0", "1",
print("Accuracy of the model is : ", accuracy_score(y2_test, NB2_predicted_test)
```

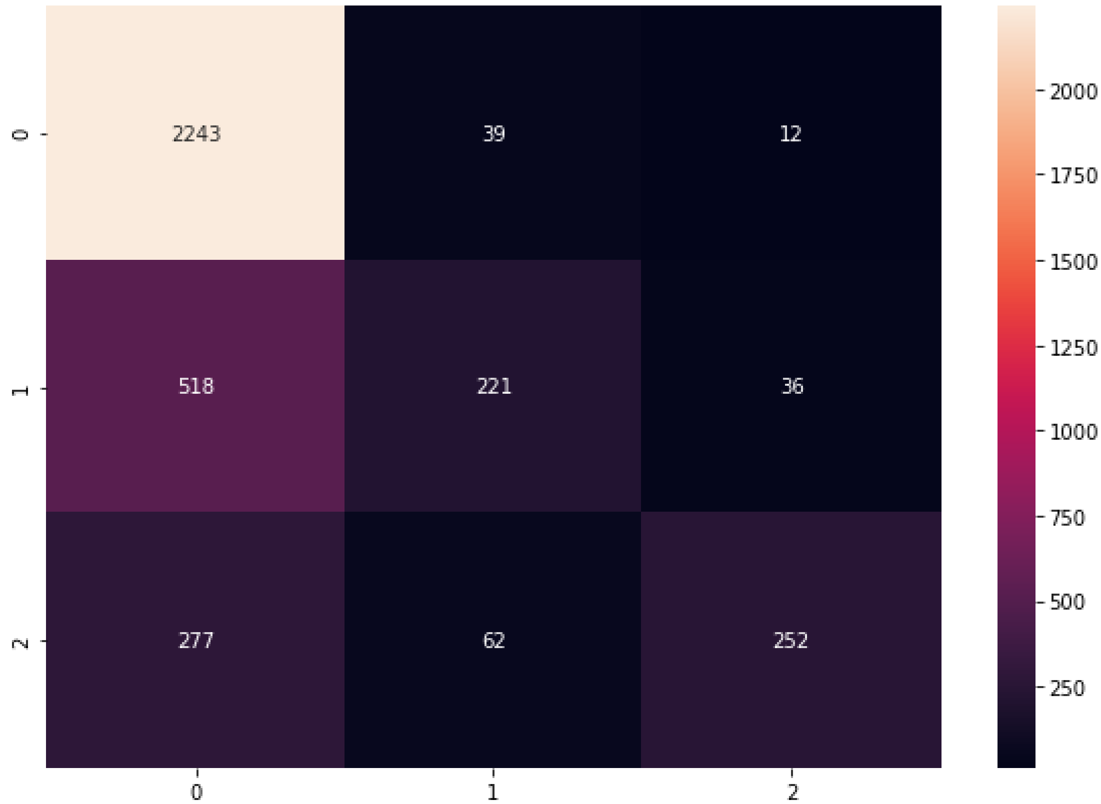
```
              precision    recall  f1-score   support

           0       0.74      0.98      0.84      2294
           1       0.69      0.29      0.40       775
           2       0.84      0.43      0.57       591

    accuracy                           0.74      3660
   macro avg       0.75      0.56      0.60      3660
weighted avg       0.74      0.74      0.70      3660

Accuracy of the model is :  0.7420765027322405
```

- 74.2

In [84]:
```
conf_mat6 = confusion_matrix(y2_test, NB2_predicted_test)

print(conf_mat6)

df_cm6 = pd.DataFrame(
    conf_mat6, index=[i for i in ["0", "1", "2"]], columns=[i for i in ["0", "1"
)
plt.figure(figsize=(10, 7))
sns.heatmap(df_cm6, annot=True, fmt="g")
```

```
[[2243   39   12]
 [ 518  221   36]
 [ 277   62  252]]
```

Out[84]:
```
<AxesSubplot:>
```



LSTM Model

In [85]:
```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
```

In [86]:
```
maxlen = 100
embedding_dim = 100
```

In [87]:
```
X = df.text.values

y = df.airline_sentiment.values
```

In [88]:
```
# Split data into training and testing set.

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, stratify=y, test_size=0.25, random_state=42
)
```

In [89]:
```
from tensorflow.keras.utils import to_categorical

y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

In [90]:
```
tokenizer = Tokenizer(num_words=2000)
tokenizer.fit_on_texts(df.text.values)
```

In [91]:
```
X_train = tokenizer.texts_to_sequences(X_train)
X_test = tokenizer.texts_to_sequences(X_test)
```

In [92]:
```
vocab_size = len(tokenizer.word_index) + 1
```

In [93]:
```
X_train = pad_sequences(X_train, padding="pre", maxlen=maxlen)
X_test = pad_sequences(X_test, padding="pre", maxlen=maxlen)
```

In [94]:
```
from keras.models import Sequential
from keras.layers.core import Dense, Dropout
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D
```

In [95]:
```
model = Sequential()
model.add(
    Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=maxle
)
model.add(SpatialDropout1D(0.4))
model.add(LSTM(64, activation="tanh"))
model.add(Dense(3, activation="softmax"))
model.compile(optimizer="Adam", loss="categorical_crossentropy", metrics=["accur
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding (Embedding)       (None, 100, 100)          1263000   
                                                                 
 spatial_dropout1d (SpatialD  (None, 100, 100)         0         
 ropout1D)                                                       
                                                                 
 lstm (LSTM)                 (None, 64)                42240     
                                                                 
 dense (Dense)               (None, 3)                 195       
                                                                 
=================================================================
Total params: 1,305,435
Trainable params: 1,305,435
Non-trainable params: 0
_________________________________________________________________
```

In [96]:
```
# fitting the model
hist_mod = model.fit(X_train, y_train, batch_size=32, epochs=11, validation_spli
```

```
Epoch 1/11
275/275 [==============================] - 14s 43ms/step - loss: 0.7160 - accura
cy: 0.7036 - val_loss: 0.5831 - val_accuracy: 0.7714
Epoch 2/11
275/275 [==============================] - 11s 41ms/step - loss: 0.5079 - accura
cy: 0.7975 - val_loss: 0.5225 - val_accuracy: 0.7923
Epoch 3/11
```

```
275/275 [==============================] - 11s 41ms/step - loss: 0.4257 - accura
cy: 0.8330 - val_loss: 0.5281 - val_accuracy: 0.7910
Epoch 4/11
275/275 [==============================] - 11s 41ms/step - loss: 0.3830 - accura
cy: 0.8533 - val_loss: 0.5752 - val_accuracy: 0.7796
Epoch 5/11
275/275 [==============================] - 11s 41ms/step - loss: 0.3522 - accura
cy: 0.8653 - val_loss: 0.5781 - val_accuracy: 0.7837
Epoch 6/11
275/275 [==============================] - 11s 41ms/step - loss: 0.3315 - accura
cy: 0.8726 - val_loss: 0.5836 - val_accuracy: 0.7801
Epoch 7/11
275/275 [==============================] - 11s 41ms/step - loss: 0.2992 - accura
cy: 0.8862 - val_loss: 0.6234 - val_accuracy: 0.7828
Epoch 8/11
275/275 [==============================] - 11s 41ms/step - loss: 0.2793 - accura
cy: 0.8949 - val_loss: 0.6640 - val_accuracy: 0.7732
Epoch 9/11
275/275 [==============================] - 11s 42ms/step - loss: 0.2567 - accura
cy: 0.9036 - val_loss: 0.7103 - val_accuracy: 0.7723
Epoch 10/11
275/275 [==============================] - 11s 41ms/step - loss: 0.2389 - accura
cy: 0.9097 - val_loss: 0.7596 - val_accuracy: 0.7760
Epoch 11/11
275/275 [==============================] - 11s 41ms/step - loss: 0.2155 - accura
cy: 0.9216 - val_loss: 0.8025 - val_accuracy: 0.7577
```

In [97]:

```
model.evaluate(X_test, y_test)  # 1. 76.5   2. 75.3   3. 75.9
```

```
115/115 [==============================] - 1s 11ms/step - loss: 0.8124 - accurac
y: 0.7593
```

Out[97]:

```
[0.8123811483383179, 0.7592896223068237]
```

In [98]:

```
model1 = Sequential()
model1.add(
    Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=maxle
)
model1.add(SpatialDropout1D(0.5))
model1.add(LSTM(64, activation="leaky_relu"))
model1.add(Dense(3, activation="softmax"))
model1.compile(optimizer="Adam", loss="categorical_crossentropy", metrics=["accu
model1.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 100, 100)          1263000

 spatial_dropout1d_1 (Spatia  (None, 100, 100)         0
 lDropout1D)

 lstm_1 (LSTM)               (None, 64)                42240

 dense_1 (Dense)             (None, 3)                 195

=================================================================
Total params: 1,305,435
Trainable params: 1,305,435
Non-trainable params: 0
_________________________________________________________________
```

In [99]:
```
# fitting the model
hist_mod1 = model1.fit(X_train, y_train, batch_size=32, epochs=11, validation_sp
```

```
Epoch 1/11
275/275 [==============================] - 13s 45ms/step - loss: 503470.3438 - a
ccuracy: 0.6570 - val_loss: 0.6509 - val_accuracy: 0.7177
Epoch 2/11
275/275 [==============================] - 12s 44ms/step - loss: 0.5872 - accura
cy: 0.7517 - val_loss: 0.6067 - val_accuracy: 0.7700
Epoch 3/11
275/275 [==============================] - 12s 45ms/step - loss: 0.5150 - accura
cy: 0.7976 - val_loss: 0.5858 - val_accuracy: 0.7778
Epoch 4/11
275/275 [==============================] - 12s 45ms/step - loss: 0.4634 - accura
cy: 0.8202 - val_loss: 0.5855 - val_accuracy: 0.7741
Epoch 5/11
275/275 [==============================] - 12s 45ms/step - loss: 0.4217 - accura
cy: 0.8334 - val_loss: 0.6188 - val_accuracy: 0.7746
Epoch 6/11
275/275 [==============================] - 12s 45ms/step - loss: 0.3945 - accura
cy: 0.8431 - val_loss: 0.6661 - val_accuracy: 0.7732
Epoch 7/11
275/275 [==============================] - 12s 45ms/step - loss: 0.3686 - accura
cy: 0.8529 - val_loss: 0.6828 - val_accuracy: 0.7591
Epoch 8/11
275/275 [==============================] - 12s 45ms/step - loss: 0.3478 - accura
cy: 0.8618 - val_loss: 0.7154 - val_accuracy: 0.7682
Epoch 9/11
275/275 [==============================] - 12s 44ms/step - loss: 0.3327 - accura
cy: 0.8665 - val_loss: 0.7827 - val_accuracy: 0.7623
Epoch 10/11
275/275 [==============================] - 12s 44ms/step - loss: 0.3158 - accura
cy: 0.8749 - val_loss: 0.8594 - val_accuracy: 0.7587
Epoch 11/11
275/275 [==============================] - 12s 44ms/step - loss: 0.3005 - accura
cy: 0.8799 - val_loss: 0.8566 - val_accuracy: 0.7564
```

In [100…
```
model1.evaluate(X_test, y_test)  # 1. 76.1  2. 76.7  3. 75.8
```

```
115/115 [==============================] - 1s 11ms/step - loss: 0.8017 - accurac
y: 0.7587
```

Out[100…
```
[0.8016793131828308, 0.758743166923523]
```

- Changed activation function and increased dropout to 0.5. Maybe a slight increase in performance.

In [101…
```
model2 = Sequential()
model2.add(
    Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=maxle
)
model2.add(SpatialDropout1D(0.5))
model2.add(LSTM(64))
model2.add(Dense(3, activation="softmax"))
model2.compile(optimizer="Adam", loss="categorical_crossentropy", metrics=["accu
model2.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
```

```
 embedding_2 (Embedding)     (None, 100, 100)          1263000

 spatial_dropout1d_2 (Spatia  (None, 100, 100)         0
 lDropout1D)

 lstm_2 (LSTM)               (None, 64)                42240

 dense_2 (Dense)             (None, 3)                 195

=================================================================
Total params: 1,305,435
Trainable params: 1,305,435
Non-trainable params: 0
_________________________________________________________________
```

In [102…

```
# fitting the model
hist_mod2 = model2.fit(X_train, y_train, batch_size=32, epochs=11, validation_sp
```

```
Epoch 1/11
275/275 [==============================] - 14s 44ms/step - loss: 0.7120 - accura
cy: 0.7063 - val_loss: 0.5663 - val_accuracy: 0.7787
Epoch 2/11
275/275 [==============================] - 12s 42ms/step - loss: 0.5076 - accura
cy: 0.7984 - val_loss: 0.5168 - val_accuracy: 0.7919
Epoch 3/11
275/275 [==============================] - 12s 43ms/step - loss: 0.4368 - accura
cy: 0.8283 - val_loss: 0.5247 - val_accuracy: 0.7942
Epoch 4/11
275/275 [==============================] - 12s 43ms/step - loss: 0.3951 - accura
cy: 0.8457 - val_loss: 0.5444 - val_accuracy: 0.7810
Epoch 5/11
275/275 [==============================] - 12s 42ms/step - loss: 0.3701 - accura
cy: 0.8554 - val_loss: 0.5639 - val_accuracy: 0.7855
Epoch 6/11
275/275 [==============================] - 12s 42ms/step - loss: 0.3422 - accura
cy: 0.8666 - val_loss: 0.5917 - val_accuracy: 0.7901
Epoch 7/11
275/275 [==============================] - 11s 42ms/step - loss: 0.3225 - accura
cy: 0.8742 - val_loss: 0.6112 - val_accuracy: 0.7805
Epoch 8/11
275/275 [==============================] - 11s 41ms/step - loss: 0.2991 - accura
cy: 0.8863 - val_loss: 0.6458 - val_accuracy: 0.7819
Epoch 9/11
275/275 [==============================] - 11s 41ms/step - loss: 0.2811 - accura
cy: 0.8917 - val_loss: 0.7063 - val_accuracy: 0.7623
Epoch 10/11
275/275 [==============================] - 11s 41ms/step - loss: 0.2586 - accura
cy: 0.8980 - val_loss: 0.7042 - val_accuracy: 0.7801
Epoch 11/11
275/275 [==============================] - 11s 41ms/step - loss: 0.2463 - accura
cy: 0.9040 - val_loss: 0.7342 - val_accuracy: 0.7769
```

In [103…

```
model2.evaluate(X_test, y_test)  # 1. 76.5  2. 76.7  3. 77.3
```

```
115/115 [==============================] - 1s 11ms/step - loss: 0.7495 - accurac
y: 0.7735
```

Out[103…

```
[0.7494664192199707, 0.7734972834587097]
```

- Took out activation function as I realized I probably shouldn't use one.
- Maybe a slight increase in performance.

In [104…

```
model3 = Sequential()
model3.add(
    Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=maxle
)
model3.add(SpatialDropout1D(0.5))
model3.add(LSTM(4))
model3.add(Dense(3, activation="softmax"))
model3.compile(optimizer="Adam", loss="categorical_crossentropy", metrics=["accu
model3.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_3 (Embedding)     (None, 100, 100)          1263000   
                                                                 
 spatial_dropout1d_3 (Spatia  (None, 100, 100)         0         
 lDropout1D)                                                     
                                                                 
 lstm_3 (LSTM)               (None, 4)                 1680      
                                                                 
 dense_3 (Dense)             (None, 3)                 15        
                                                                 
=================================================================
Total params: 1,264,695
Trainable params: 1,264,695
Non-trainable params: 0
_________________________________________________________________
```

In [105…

```
# fitting the model
hist_mod3 = model3.fit(X_train, y_train, batch_size=32, epochs=11, validation_sp
```

```
Epoch 1/11
275/275 [==============================] - 11s 34ms/step - loss: 0.7908 - accura
cy: 0.6790 - val_loss: 0.6528 - val_accuracy: 0.7454
Epoch 2/11
275/275 [==============================] - 9s 33ms/step - loss: 0.5866 - accurac
y: 0.7815 - val_loss: 0.5887 - val_accuracy: 0.7737
Epoch 3/11
275/275 [==============================] - 9s 33ms/step - loss: 0.5030 - accurac
y: 0.8152 - val_loss: 0.5547 - val_accuracy: 0.7851
Epoch 4/11
275/275 [==============================] - 9s 33ms/step - loss: 0.4394 - accurac
y: 0.8421 - val_loss: 0.5521 - val_accuracy: 0.7873
Epoch 5/11
275/275 [==============================] - 9s 33ms/step - loss: 0.4051 - accurac
y: 0.8510 - val_loss: 0.5587 - val_accuracy: 0.7896
Epoch 6/11
275/275 [==============================] - 9s 33ms/step - loss: 0.3746 - accurac
y: 0.8635 - val_loss: 0.5681 - val_accuracy: 0.7855
Epoch 7/11
275/275 [==============================] - 9s 33ms/step - loss: 0.3503 - accurac
y: 0.8723 - val_loss: 0.5872 - val_accuracy: 0.7823
Epoch 8/11
275/275 [==============================] - 9s 33ms/step - loss: 0.3329 - accurac
y: 0.8780 - val_loss: 0.5998 - val_accuracy: 0.7846
Epoch 9/11
275/275 [==============================] - 9s 33ms/step - loss: 0.3144 - accurac
y: 0.8850 - val_loss: 0.6138 - val_accuracy: 0.7814
Epoch 10/11
275/275 [==============================] - 9s 33ms/step - loss: 0.2991 - accurac
y: 0.8915 - val_loss: 0.6304 - val_accuracy: 0.7787
Epoch 11/11
```

```
275/275 [==============================] - 9s 33ms/step - loss: 0.2892 - accurac
y: 0.8928 - val_loss: 0.6452 - val_accuracy: 0.7723
```

In [106…

```
model3.evaluate(X_test, y_test)  # 1. 78 2. 77.1  3. 77.9  4. 78
```

```
115/115 [==============================] - 1s 6ms/step - loss: 0.6440 - accurac
y: 0.7801
```

Out[106…

```
[0.644018828868866, 0.7800546288490295]
```

- Changed the number of neurons from 64 to 4.
- Noticeable increase in performance.

In [107…

```
model4 = Sequential()
model4.add(
    Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=maxle
)
model4.add(SpatialDropout1D(0.5))
model4.add(LSTM(4))
model4.add(Dense(3, activation="softmax"))
model4.compile(optimizer="Adam", loss="categorical_crossentropy", metrics=["accu
model4.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 100, 100)          1263000

 spatial_dropout1d_4 (Spatia  (None, 100, 100)         0
 lDropout1D)

 lstm_4 (LSTM)               (None, 4)                 1680

 dense_4 (Dense)             (None, 3)                 15

=================================================================
Total params: 1,264,695
Trainable params: 1,264,695
Non-trainable params: 0
_________________________________________________________________
```

In [108…

```
# fitting the model
hist_mod4 = model4.fit(
    X_train, y_train, batch_size=500, epochs=22, validation_split=0.2
)
```

```
Epoch 1/22
18/18 [==============================] - 3s 93ms/step - loss: 1.0772 - accuracy:
0.5848 - val_loss: 1.0534 - val_accuracy: 0.6184
Epoch 2/22
18/18 [==============================] - 1s 70ms/step - loss: 1.0273 - accuracy:
0.6314 - val_loss: 1.0019 - val_accuracy: 0.6266
Epoch 3/22
18/18 [==============================] - 1s 71ms/step - loss: 0.9681 - accuracy:
0.6423 - val_loss: 0.9380 - val_accuracy: 0.6462
Epoch 4/22
18/18 [==============================] - 1s 72ms/step - loss: 0.8959 - accuracy:
0.6634 - val_loss: 0.8637 - val_accuracy: 0.6708
Epoch 5/22
18/18 [==============================] - 1s 73ms/step - loss: 0.8216 - accuracy:
```

```
0.6847 - val_loss: 0.7994 - val_accuracy: 0.6944
Epoch 6/22
18/18 [==============================] - 1s 70ms/step - loss: 0.7599 - accuracy:
0.7046 - val_loss: 0.7523 - val_accuracy: 0.7131
Epoch 7/22
18/18 [==============================] - 1s 72ms/step - loss: 0.7127 - accuracy:
0.7310 - val_loss: 0.7169 - val_accuracy: 0.7322
Epoch 8/22
18/18 [==============================] - 1s 70ms/step - loss: 0.6719 - accuracy:
0.7563 - val_loss: 0.6874 - val_accuracy: 0.7514
Epoch 9/22
18/18 [==============================] - 1s 69ms/step - loss: 0.6351 - accuracy:
0.7778 - val_loss: 0.6618 - val_accuracy: 0.7637
Epoch 10/22
18/18 [==============================] - 1s 70ms/step - loss: 0.6006 - accuracy:
0.8021 - val_loss: 0.6387 - val_accuracy: 0.7719
Epoch 11/22
18/18 [==============================] - 1s 72ms/step - loss: 0.5704 - accuracy:
0.8102 - val_loss: 0.6188 - val_accuracy: 0.7810
Epoch 12/22
18/18 [==============================] - 1s 69ms/step - loss: 0.5438 - accuracy:
0.8267 - val_loss: 0.6001 - val_accuracy: 0.7864
Epoch 13/22
18/18 [==============================] - 1s 69ms/step - loss: 0.5178 - accuracy:
0.8358 - val_loss: 0.5860 - val_accuracy: 0.7928
Epoch 14/22
18/18 [==============================] - 1s 70ms/step - loss: 0.4934 - accuracy:
0.8468 - val_loss: 0.5757 - val_accuracy: 0.7955
Epoch 15/22
18/18 [==============================] - 1s 72ms/step - loss: 0.4741 - accuracy:
0.8518 - val_loss: 0.5652 - val_accuracy: 0.7974
Epoch 16/22
18/18 [==============================] - 1s 71ms/step - loss: 0.4554 - accuracy:
0.8605 - val_loss: 0.5582 - val_accuracy: 0.8010
Epoch 17/22
18/18 [==============================] - 1s 69ms/step - loss: 0.4374 - accuracy:
0.8651 - val_loss: 0.5525 - val_accuracy: 0.7983
Epoch 18/22
18/18 [==============================] - 1s 70ms/step - loss: 0.4222 - accuracy:
0.8691 - val_loss: 0.5510 - val_accuracy: 0.7969
Epoch 19/22
18/18 [==============================] - 1s 71ms/step - loss: 0.4119 - accuracy:
0.8708 - val_loss: 0.5480 - val_accuracy: 0.7964
Epoch 20/22
18/18 [==============================] - 1s 72ms/step - loss: 0.3970 - accuracy:
0.8749 - val_loss: 0.5479 - val_accuracy: 0.7910
Epoch 21/22
18/18 [==============================] - 1s 70ms/step - loss: 0.3838 - accuracy:
0.8788 - val_loss: 0.5474 - val_accuracy: 0.7933
Epoch 22/22
18/18 [==============================] - 1s 72ms/step - loss: 0.3756 - accuracy:
0.8857 - val_loss: 0.5465 - val_accuracy: 0.7901
```

In [109…

```
model4.evaluate(X_test, y_test)  # 1. 78.6  2. 77.2  3. 79.1
```

```
115/115 [==============================] - 1s 5ms/step - loss: 0.5534 - accurac
y: 0.7913
```

Out[109…

```
[0.5533977746963501, 0.791256844997406]
```

- Increased batch size and epochs.
- Maybe slight increase in performance.

In [110…

```
model5 = Sequential()
model5.add(
    Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=maxle
)
model5.add(SpatialDropout1D(0.5))
model5.add(LSTM(4))
model5.add(Dense(3, activation="softmax"))
model5.compile(optimizer="Adam", loss="categorical_crossentropy", metrics=["accu
model5.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_5 (Embedding)     (None, 100, 100)          1263000   
                                                                 
 spatial_dropout1d_5 (Spatia  (None, 100, 100)         0         
 lDropout1D)                                                     
                                                                 
 lstm_5 (LSTM)               (None, 4)                 1680      
                                                                 
 dense_5 (Dense)             (None, 3)                 15        
                                                                 
=================================================================
Total params: 1,264,695
Trainable params: 1,264,695
Non-trainable params: 0
_________________________________________________________________
```

In [111…

```
# fitting the model
hist_mod5 = model5.fit(
    X_train, y_train, batch_size=1000, epochs=44, validation_split=0.2
)
```

```
Epoch 1/44
9/9 [==============================] - 3s 179ms/step - loss: 1.0796 - accuracy:
0.5758 - val_loss: 1.0566 - val_accuracy: 0.6321
Epoch 2/44
9/9 [==============================] - 1s 128ms/step - loss: 1.0355 - accuracy:
0.6401 - val_loss: 1.0118 - val_accuracy: 0.6207
Epoch 3/44
9/9 [==============================] - 1s 129ms/step - loss: 0.9849 - accuracy:
0.6325 - val_loss: 0.9608 - val_accuracy: 0.6216
Epoch 4/44
9/9 [==============================] - 1s 124ms/step - loss: 0.9289 - accuracy:
0.6357 - val_loss: 0.9066 - val_accuracy: 0.6239
Epoch 5/44
9/9 [==============================] - 1s 120ms/step - loss: 0.8723 - accuracy:
0.6436 - val_loss: 0.8555 - val_accuracy: 0.6384
Epoch 6/44
9/9 [==============================] - 1s 123ms/step - loss: 0.8223 - accuracy:
0.6557 - val_loss: 0.8153 - val_accuracy: 0.6548
Epoch 7/44
9/9 [==============================] - 1s 119ms/step - loss: 0.7854 - accuracy:
0.6659 - val_loss: 0.7873 - val_accuracy: 0.6671
Epoch 8/44
9/9 [==============================] - 1s 118ms/step - loss: 0.7562 - accuracy:
0.6782 - val_loss: 0.7668 - val_accuracy: 0.6717
Epoch 9/44
9/9 [==============================] - 1s 118ms/step - loss: 0.7322 - accuracy:
0.6852 - val_loss: 0.7500 - val_accuracy: 0.6831
Epoch 10/44
9/9 [==============================] - 1s 116ms/step - loss: 0.7115 - accuracy:
```

```
0.6976 - val_loss: 0.7353 - val_accuracy: 0.6931
Epoch 11/44
9/9 [==============================] - 1s 118ms/step - loss: 0.6917 - accuracy:
0.7077 - val_loss: 0.7224 - val_accuracy: 0.6985
Epoch 12/44
9/9 [==============================] - 1s 116ms/step - loss: 0.6734 - accuracy:
0.7171 - val_loss: 0.7104 - val_accuracy: 0.7067
Epoch 13/44
9/9 [==============================] - 1s 114ms/step - loss: 0.6569 - accuracy:
0.7295 - val_loss: 0.6994 - val_accuracy: 0.7168
Epoch 14/44
9/9 [==============================] - 1s 117ms/step - loss: 0.6397 - accuracy:
0.7387 - val_loss: 0.6891 - val_accuracy: 0.7199
Epoch 15/44
9/9 [==============================] - 1s 119ms/step - loss: 0.6248 - accuracy:
0.7470 - val_loss: 0.6792 - val_accuracy: 0.7250
Epoch 16/44
9/9 [==============================] - 1s 116ms/step - loss: 0.6096 - accuracy:
0.7573 - val_loss: 0.6695 - val_accuracy: 0.7304
Epoch 17/44
9/9 [==============================] - 1s 117ms/step - loss: 0.5963 - accuracy:
0.7662 - val_loss: 0.6604 - val_accuracy: 0.7377
Epoch 18/44
9/9 [==============================] - 1s 119ms/step - loss: 0.5819 - accuracy:
0.7720 - val_loss: 0.6514 - val_accuracy: 0.7418
Epoch 19/44
9/9 [==============================] - 1s 118ms/step - loss: 0.5696 - accuracy:
0.7843 - val_loss: 0.6435 - val_accuracy: 0.7500
Epoch 20/44
9/9 [==============================] - 1s 116ms/step - loss: 0.5551 - accuracy:
0.7892 - val_loss: 0.6358 - val_accuracy: 0.7518
Epoch 21/44
9/9 [==============================] - 1s 120ms/step - loss: 0.5441 - accuracy:
0.7936 - val_loss: 0.6289 - val_accuracy: 0.7527
Epoch 22/44
9/9 [==============================] - 1s 116ms/step - loss: 0.5344 - accuracy:
0.7968 - val_loss: 0.6231 - val_accuracy: 0.7555
Epoch 23/44
9/9 [==============================] - 1s 116ms/step - loss: 0.5218 - accuracy:
0.8050 - val_loss: 0.6180 - val_accuracy: 0.7596
Epoch 24/44
9/9 [==============================] - 1s 121ms/step - loss: 0.5119 - accuracy:
0.8103 - val_loss: 0.6128 - val_accuracy: 0.7596
Epoch 25/44
9/9 [==============================] - 1s 115ms/step - loss: 0.5026 - accuracy:
0.8118 - val_loss: 0.6086 - val_accuracy: 0.7618
Epoch 26/44
9/9 [==============================] - 1s 116ms/step - loss: 0.4932 - accuracy:
0.8163 - val_loss: 0.6046 - val_accuracy: 0.7655
Epoch 27/44
9/9 [==============================] - 1s 118ms/step - loss: 0.4840 - accuracy:
0.8226 - val_loss: 0.6011 - val_accuracy: 0.7646
Epoch 28/44
9/9 [==============================] - 1s 117ms/step - loss: 0.4772 - accuracy:
0.8280 - val_loss: 0.5970 - val_accuracy: 0.7659
Epoch 29/44
9/9 [==============================] - 1s 115ms/step - loss: 0.4682 - accuracy:
0.8324 - val_loss: 0.5948 - val_accuracy: 0.7687
Epoch 30/44
9/9 [==============================] - 1s 118ms/step - loss: 0.4583 - accuracy:
0.8375 - val_loss: 0.5922 - val_accuracy: 0.7723
Epoch 31/44
9/9 [==============================] - 1s 115ms/step - loss: 0.4504 - accuracy:
0.8418 - val_loss: 0.5896 - val_accuracy: 0.7696
Epoch 32/44
```

```
9/9 [==============================] - 1s 115ms/step - loss: 0.4434 - accuracy:
0.8465 - val_loss: 0.5868 - val_accuracy: 0.7719
Epoch 33/44
9/9 [==============================] - 1s 119ms/step - loss: 0.4355 - accuracy:
0.8492 - val_loss: 0.5855 - val_accuracy: 0.7737
Epoch 34/44
9/9 [==============================] - 1s 117ms/step - loss: 0.4284 - accuracy:
0.8537 - val_loss: 0.5838 - val_accuracy: 0.7769
Epoch 35/44
9/9 [==============================] - 1s 114ms/step - loss: 0.4206 - accuracy:
0.8566 - val_loss: 0.5831 - val_accuracy: 0.7782
Epoch 36/44
9/9 [==============================] - 1s 117ms/step - loss: 0.4128 - accuracy:
0.8617 - val_loss: 0.5821 - val_accuracy: 0.7819
Epoch 37/44
9/9 [==============================] - 1s 115ms/step - loss: 0.4067 - accuracy:
0.8600 - val_loss: 0.5797 - val_accuracy: 0.7782
Epoch 38/44
9/9 [==============================] - 1s 114ms/step - loss: 0.4018 - accuracy:
0.8656 - val_loss: 0.5794 - val_accuracy: 0.7805
Epoch 39/44
9/9 [==============================] - 1s 115ms/step - loss: 0.3937 - accuracy:
0.8668 - val_loss: 0.5795 - val_accuracy: 0.7782
Epoch 40/44
9/9 [==============================] - 1s 122ms/step - loss: 0.3870 - accuracy:
0.8694 - val_loss: 0.5794 - val_accuracy: 0.7801
Epoch 41/44
9/9 [==============================] - 1s 119ms/step - loss: 0.3823 - accuracy:
0.8747 - val_loss: 0.5782 - val_accuracy: 0.7814
Epoch 42/44
9/9 [==============================] - 1s 117ms/step - loss: 0.3756 - accuracy:
0.8758 - val_loss: 0.5789 - val_accuracy: 0.7823
Epoch 43/44
9/9 [==============================] - 1s 116ms/step - loss: 0.3698 - accuracy:
0.8782 - val_loss: 0.5788 - val_accuracy: 0.7787
Epoch 44/44
9/9 [==============================] - 1s 114ms/step - loss: 0.3653 - accuracy:
0.8809 - val_loss: 0.5800 - val_accuracy: 0.7805
```

In [112…

```python
model5.evaluate(X_test, y_test)  # 1. 78.1  2. 78.2  3. 77.9
```

```
115/115 [==============================] - 1s 6ms/step - loss: 0.5812 - accurac
y: 0.7792
```

Out[112…

```
[0.5811668038368225, 0.7792349457740784]
```

- Increased batch size even more.
- Maybe a slight increase in performance.

In [113…

```python
model6 = Sequential()
model6.add(
    Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=maxle
)
model6.add(SpatialDropout1D(0.5))
model6.add(LSTM(4))
model6.add(Dense(3, activation="softmax"))
model6.compile(optimizer="Adam", loss="categorical_crossentropy", metrics=["accu
model6.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
```

```
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_6 (Embedding)     (None, 100, 100)          1263000   
                                                                 
 spatial_dropout1d_6 (Spatia  (None, 100, 100)         0         
 lDropout1D)                                                     
                                                                 
 lstm_6 (LSTM)               (None, 4)                 1680      
                                                                 
 dense_6 (Dense)             (None, 3)                 15        
                                                                 
=================================================================
Total params: 1,264,695
Trainable params: 1,264,695
Non-trainable params: 0
_________________________________________________________________
```

In [114…

```
# fitting the model
hist_mod6 = model6.fit(
    X_train, y_train, batch_size=1500, epochs=44, validation_split=0.2
)
```

```
Epoch 1/44
6/6 [==============================] - 3s 268ms/step - loss: 1.0904 - accuracy:
0.5135 - val_loss: 1.0794 - val_accuracy: 0.6289
Epoch 2/44
6/6 [==============================] - 1s 178ms/step - loss: 1.0697 - accuracy:
0.6368 - val_loss: 1.0585 - val_accuracy: 0.6252
Epoch 3/44
6/6 [==============================] - 1s 176ms/step - loss: 1.0456 - accuracy:
0.6332 - val_loss: 1.0338 - val_accuracy: 0.6252
Epoch 4/44
6/6 [==============================] - 1s 178ms/step - loss: 1.0177 - accuracy:
0.6307 - val_loss: 1.0048 - val_accuracy: 0.6216
Epoch 5/44
6/6 [==============================] - 1s 177ms/step - loss: 0.9858 - accuracy:
0.6297 - val_loss: 0.9726 - val_accuracy: 0.6184
Epoch 6/44
6/6 [==============================] - 1s 176ms/step - loss: 0.9508 - accuracy:
0.6302 - val_loss: 0.9409 - val_accuracy: 0.6184
Epoch 7/44
6/6 [==============================] - 1s 178ms/step - loss: 0.9182 - accuracy:
0.6294 - val_loss: 0.9136 - val_accuracy: 0.6184
Epoch 8/44
6/6 [==============================] - 1s 170ms/step - loss: 0.8906 - accuracy:
0.6298 - val_loss: 0.8908 - val_accuracy: 0.6202
Epoch 9/44
6/6 [==============================] - 1s 169ms/step - loss: 0.8657 - accuracy:
0.6326 - val_loss: 0.8681 - val_accuracy: 0.6275
Epoch 10/44
6/6 [==============================] - 1s 176ms/step - loss: 0.8412 - accuracy:
0.6399 - val_loss: 0.8442 - val_accuracy: 0.6343
Epoch 11/44
6/6 [==============================] - 1s 168ms/step - loss: 0.8164 - accuracy:
0.6543 - val_loss: 0.8227 - val_accuracy: 0.6526
Epoch 12/44
6/6 [==============================] - 1s 177ms/step - loss: 0.7955 - accuracy:
0.6692 - val_loss: 0.8045 - val_accuracy: 0.6694
Epoch 13/44
6/6 [==============================] - 1s 177ms/step - loss: 0.7752 - accuracy:
0.6814 - val_loss: 0.7886 - val_accuracy: 0.6831
Epoch 14/44
6/6 [==============================] - 1s 176ms/step - loss: 0.7573 - accuracy:
0.6900 - val_loss: 0.7742 - val_accuracy: 0.6908
```

```
Epoch 15/44
6/6 [==============================] - 1s 174ms/step - loss: 0.7404 - accuracy:
0.7024 - val_loss: 0.7610 - val_accuracy: 0.6976
Epoch 16/44
6/6 [==============================] - 1s 176ms/step - loss: 0.7241 - accuracy:
0.7122 - val_loss: 0.7489 - val_accuracy: 0.7081
Epoch 17/44
6/6 [==============================] - 1s 174ms/step - loss: 0.7077 - accuracy:
0.7247 - val_loss: 0.7376 - val_accuracy: 0.7163
Epoch 18/44
6/6 [==============================] - 1s 171ms/step - loss: 0.6947 - accuracy:
0.7332 - val_loss: 0.7271 - val_accuracy: 0.7199
Epoch 19/44
6/6 [==============================] - 1s 176ms/step - loss: 0.6798 - accuracy:
0.7437 - val_loss: 0.7172 - val_accuracy: 0.7318
Epoch 20/44
6/6 [==============================] - 1s 176ms/step - loss: 0.6658 - accuracy:
0.7538 - val_loss: 0.7080 - val_accuracy: 0.7413
Epoch 21/44
6/6 [==============================] - 1s 176ms/step - loss: 0.6526 - accuracy:
0.7633 - val_loss: 0.6992 - val_accuracy: 0.7468
Epoch 22/44
6/6 [==============================] - 1s 175ms/step - loss: 0.6403 - accuracy:
0.7717 - val_loss: 0.6907 - val_accuracy: 0.7514
Epoch 23/44
6/6 [==============================] - 1s 174ms/step - loss: 0.6277 - accuracy:
0.7797 - val_loss: 0.6823 - val_accuracy: 0.7555
Epoch 24/44
6/6 [==============================] - 1s 176ms/step - loss: 0.6156 - accuracy:
0.7869 - val_loss: 0.6744 - val_accuracy: 0.7591
Epoch 25/44
6/6 [==============================] - 1s 169ms/step - loss: 0.6032 - accuracy:
0.7944 - val_loss: 0.6667 - val_accuracy: 0.7659
Epoch 26/44
6/6 [==============================] - 1s 167ms/step - loss: 0.5907 - accuracy:
0.8008 - val_loss: 0.6594 - val_accuracy: 0.7664
Epoch 27/44
6/6 [==============================] - 1s 173ms/step - loss: 0.5789 - accuracy:
0.8059 - val_loss: 0.6518 - val_accuracy: 0.7700
Epoch 28/44
6/6 [==============================] - 1s 175ms/step - loss: 0.5685 - accuracy:
0.8136 - val_loss: 0.6446 - val_accuracy: 0.7732
Epoch 29/44
6/6 [==============================] - 1s 176ms/step - loss: 0.5582 - accuracy:
0.8210 - val_loss: 0.6379 - val_accuracy: 0.7737
Epoch 30/44
6/6 [==============================] - 1s 173ms/step - loss: 0.5483 - accuracy:
0.8199 - val_loss: 0.6318 - val_accuracy: 0.7732
Epoch 31/44
6/6 [==============================] - 1s 173ms/step - loss: 0.5360 - accuracy:
0.8286 - val_loss: 0.6258 - val_accuracy: 0.7741
Epoch 32/44
6/6 [==============================] - 1s 174ms/step - loss: 0.5273 - accuracy:
0.8313 - val_loss: 0.6201 - val_accuracy: 0.7769
Epoch 33/44
6/6 [==============================] - 1s 169ms/step - loss: 0.5157 - accuracy:
0.8356 - val_loss: 0.6148 - val_accuracy: 0.7769
Epoch 34/44
6/6 [==============================] - 1s 174ms/step - loss: 0.5093 - accuracy:
0.8395 - val_loss: 0.6103 - val_accuracy: 0.7755
Epoch 35/44
6/6 [==============================] - 1s 171ms/step - loss: 0.4974 - accuracy:
0.8423 - val_loss: 0.6059 - val_accuracy: 0.7787
Epoch 36/44
6/6 [==============================] - 1s 173ms/step - loss: 0.4894 - accuracy:
```

```
0.8484 - val_loss: 0.6025 - val_accuracy: 0.7778
Epoch 37/44
6/6 [==============================] - 1s 172ms/step - loss: 0.4801 - accuracy:
0.8492 - val_loss: 0.5991 - val_accuracy: 0.7778
Epoch 38/44
6/6 [==============================] - 1s 174ms/step - loss: 0.4728 - accuracy:
0.8505 - val_loss: 0.5955 - val_accuracy: 0.7778
Epoch 39/44
6/6 [==============================] - 1s 170ms/step - loss: 0.4641 - accuracy:
0.8530 - val_loss: 0.5932 - val_accuracy: 0.7791
Epoch 40/44
6/6 [==============================] - 1s 168ms/step - loss: 0.4582 - accuracy:
0.8569 - val_loss: 0.5906 - val_accuracy: 0.7773
Epoch 41/44
6/6 [==============================] - 1s 168ms/step - loss: 0.4506 - accuracy:
0.8567 - val_loss: 0.5889 - val_accuracy: 0.7769
Epoch 42/44
6/6 [==============================] - 1s 173ms/step - loss: 0.4428 - accuracy:
0.8626 - val_loss: 0.5872 - val_accuracy: 0.7773
Epoch 43/44
6/6 [==============================] - 1s 172ms/step - loss: 0.4375 - accuracy:
0.8634 - val_loss: 0.5858 - val_accuracy: 0.7769
Epoch 44/44
6/6 [==============================] - 1s 171ms/step - loss: 0.4309 - accuracy:
0.8635 - val_loss: 0.5845 - val_accuracy: 0.7782
```

In [115…

```
model6.evaluate(X_test, y_test)  # 1. 78.3  2. 75  3. 78.6
```

```
115/115 [==============================] - 1s 5ms/step - loss: 0.5756 - accurac
y: 0.7863
```

Out[115…

```
[0.5756012201309204, 0.7863388061523438]
```

- Increased batch size.
- No noticeable improvement.

Evaluation

CountVectorizer Models

Random Forest: 76.9% Acc

XGBoost: 78.8% Acc

Naives Bayes: 76.5% Acc

TF-IDF Models

Random Forest: 75.7% Acc

XGBoost: 78.2% Acc

Naives Bayes: 74.2% Acc

LSTM Models

model: ~75.9% Acc

model1: ~76.2% Acc

model2: ~76.8% Acc

model3: ~77.6% Acc

model4: ~78.3% Acc

model5: ~78% Acc

model6: ~77.3% Acc

Conclusion

- Overall, the CountVectorizer models performed better than the TF-IDF models (surprised me).
- Maybe CountVectorizer works better on classification tasks, or maybe there are some words in the data that are influencing TF-IDF performance.
- Out of all the different algorithms used, XGBoost seems to perform best.
- Would use both XGBoost models and model4/model5 out of the LSTM models.
- I believe pre-processing has the most impact when it comes to model performance.