Text Analysis Project

CS 4120

By: Justin Lau, David Yei

Instructions

Make sure to have the necessary csv files: **reviews.csv**, **nlp_project_summary.csv** and **game_reviews.csv**.

Make sure to pip install **pandas, Unidecode, nltk, scipy, sklearn, and plotly_express==0.4.0**

You may see commented out cells in the notebook. These are not necessary to run; they indicate attempts at optimizing our models through GridSearchCV but take too long for us to run due to hardware limitations. We commented them out instead to illustrate that we tried to optimize our model, rather than leading you to believe we ignored hyperparameter tuning. More details on our attempts will be provided above our commented out code.

Importing the data

In [1]:

```
import pandas as pd
from unidecode import unidecode
from nltk.stem import WordNetLemmatizer
import scipy

#first dataset
reviews_df = pd.read_csv('reviews.csv')
reviews_df
```

Out[1]:

	review_id	title	year	user_review	user_suggestion
0	1	Spooky's Jump Scare Mansion	2016.0	I'm scared and hearing creepy voices. So I'll...	1
1	2	Spooky's Jump Scare Mansion	2016.0	Best game, more better than Sam Pepper's YouTu...	1
2	3	Spooky's Jump Scare Mansion	2016.0	A littly iffy on the controls, but once you kn...	1
3	4	Spooky's Jump Scare Mansion	2015.0	Great game, fun and colorful and all that.A si...	1
4	5	Spooky's Jump Scare Mansion	2015.0	Not many games have the cute tag right next to...	1
...	...	...	...	...	...
17489	25535	EverQuest II	2012.0	Arguably the single greatest mmorp that exists...	1
17490	25536	EverQuest II	2017.0	An older game, to be sure, but has its own cha...	1
17491	25537	EverQuest II	2011.0	When I frist started playing Everquest 2 it wa...	1
17492	25538	EverQuest II	NaN	cool game. THe only thing that REALLY PISSES M...	1
17493	25539	EverQuest II	NaN	this game since I was a little kid, always hav...	1

17494 rows × 5 columns

Data Exploration

A 1 in user_suggestion indicates that the user suggested the game, whereas a 0 indicates that the user did not recommend that game.

In [2]:

```
group_title_df = reviews_df.groupby(['title']).mean()
group_title_df.drop(['review_id', 'year'], axis=1, inplace=True)
group_title_df.sort_values('user_suggestion', inplace=True)
group_title_df
```

Out[2]:

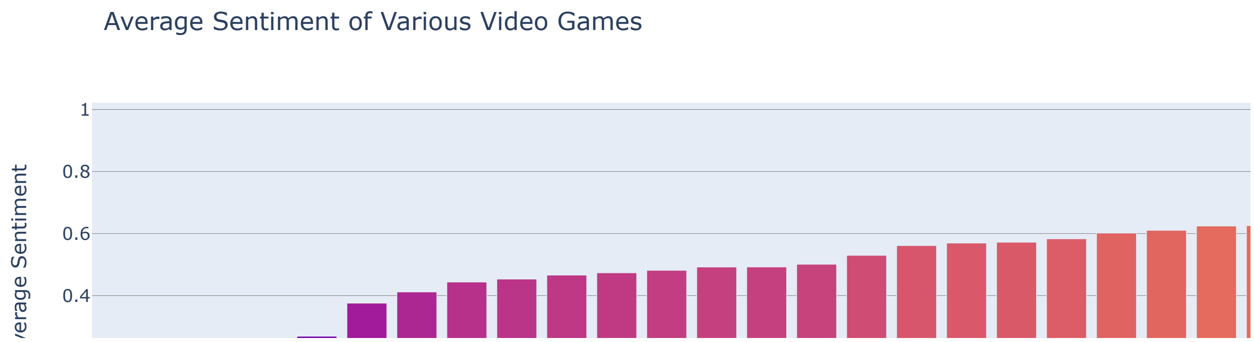
	user_suggestion
title	
Robocraft	0.057007
Heroes & Generals	0.110067
War Thunder	0.169444
Bless Online	0.212079

title	user_suggestion
Infestation: The New Z	0.269311
Cuisine Royale	0.375940
Bloons TD Battles	0.412017
theHunter Classic	0.444015
Trove	0.453488
RaceRoom Racing Experience	0.466346
Yu-Gi-Oh! Duel Links	0.473684
Business Tour - Board Game with Online Multiplayer	0.481675
World of Tanks Blitz	0.492355
School of Dragons	0.492537
Dota 2	0.501235
WARMODE	0.530000
Freestyle 2: Street Basketball	0.561404
Neverwinter	0.569597
AdventureQuest 3D	0.572254
Crusaders of the Lost Idols	0.583333
Fallout Shelter	0.601790
The Elder Scrolls®: Legends™	0.610619
Realm Royale	0.624434
Eternal Card Game	0.625790
World of Guns: Gun Disassembly	0.658703
SMITE®	0.678414
Elsword	0.687135
Dreadnought	0.716667
Team Fortress 2	0.762004
Realm of the Mad God	0.802941
DCS World Steam Edition	0.807377
Sakura Clicker	0.819820
Black Squad	0.861111
Realm Grinder	0.864516
Shop Heroes	0.865385
Brawlhalla	0.865854
Ring of Elysium	0.875895
Spooky's Jump Scare Mansion	0.886740
Tactical Monsters Rumble Arena	0.894737
PlanetSide 2	0.896186
Creativerse	0.900407
Path of Exile	0.906114
Fractured Space	0.958217
EverQuest II	0.971014

In [3]:

```
import plotly.express as px

fig = px.bar(group_title_df, x = group_title_df.index, y="user_suggestion",
             title= "Average Sentiment of Various Video Games", color= "user_suggestion")
fig.update_xaxes(title="Video Game", title_font = {"size": 10})
fig.update_yaxes(title="Average Sentiment")
fig.show()
```



In [4]:

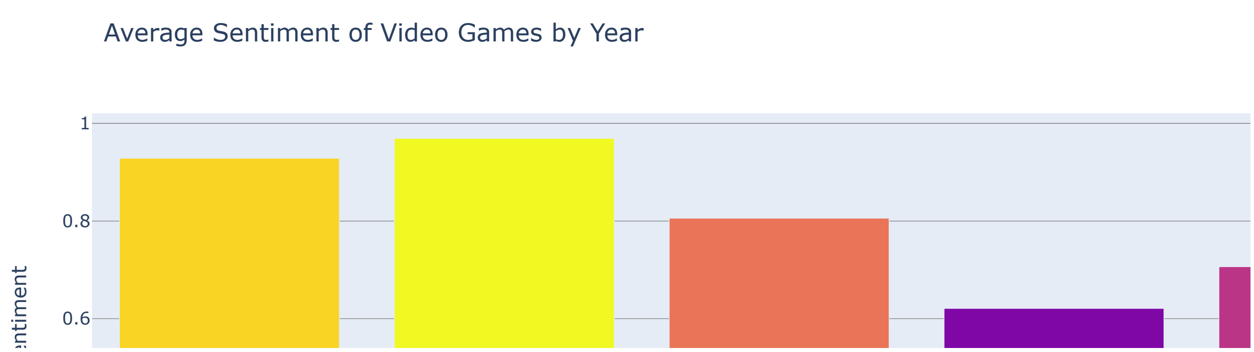
```
years_df = reviews_df.groupby(['year'])['user_suggestion'].mean()
years_df = years_df.to_frame()
years_df
```

Out[4]:

year	user_suggestion
2011.0	0.928571
2012.0	0.969231
2013.0	0.805882
2014.0	0.621081
2015.0	0.706504
2016.0	0.506389
2017.0	0.593059
2018.0	0.502074

In [5]:

```
fig2 = px.bar(years_df, x= years_df.index, y = 'user_suggestion',
              title= "Average Sentiment of Video Games by Year", color= 'user_suggestion')
fig2.update_xaxes(title= 'Year')
fig2.update_yaxes(title= 'Average Sentiment')
fig2.show()
```



In [6]:

```
num_reviews_df = reviews_df.groupby('year').count()
num_reviews_df
```

Out[6]:

year	review_id	title	user_review	user_suggestion
2011.0	14	14	14	14
2012.0	65	65	65	65
2013.0	340	340	340	340
2014.0	1499	1499	1499	1499
2015.0	2460	2460	2460	2460
2016.0	4226	4226	4226	4226
2017.0	3890	3890	3890	3890
2018.0	4822	4822	4822	4822

In [7]:

```
fig3 = px.bar(num_reviews_df, x= num_reviews_df.index, y = 'title', title= "Number of Video Game Reviews by Year")
fig3.update_xaxes(title= 'Year')
fig3.update_yaxes(title="Reviews")
fig3.show()
```

Data Wrangling

We remove unnecessary columns, because we are mainly focusing on the user review and its corresponding score.

In [8]:
```
drop_columns = ['review_id', 'title', 'year']

dropped_df = reviews_df.drop(drop_columns, axis=1)
dropped_df
```

Out[8]:

	user_review	user_suggestion
0	I'm scared and hearing creepy voices. So I'll...	1
1	Best game, more better than Sam Pepper's YouTu...	1
2	A littly iffy on the controls, but once you kn...	1
3	Great game, fun and colorful and all that.A si...	1
4	Not many games have the cute tag right next to...	1
...	...	...
17489	Arguably the single greatest mmorp that exists...	1
17490	An older game, to be sure, but has its own cha...	1
17491	When I frist started playing Everquest 2 it wa...	1
17492	cool game. THe only thing that REALLY PISSES M...	1
17493	this game since I was a little kid, always hav...	1

17494 rows × 2 columns

Data Merging

In the cells below, we merge the above dataframe with another dataframe that was provided in DS3000. This dataset will be further explained in our report.

In [9]:
```
#second dataset
reviews2_df = pd.read_csv('game_reviews.csv')
reviews2_df
```

Out[9]:

	gameID	comment	sentiment
0	345650	Is Without Withinnbspworth your time Nonbs...	0
1	289090	My playtime h based on steam Grindy Achieve...	0
2	350090	No Pineapple Left Behind	0
3	409720	PRESS SPACE TO CRASH	0
4	364360	Reason Why Chinese Gamer Give the ShXt to W...	0
...	...	...	...
19092	311210	Zombies Multiplayer Singleplayer	0
19093	1250	Zombieswith English peopleNot Shaun of the Dea...	1
19094	500	Zombies Zombies everywhereAnd the only thing t...	1
19095	283680	Zone of the Enders in a straight up bullet hel...	1
19096	22350	thats all i can say feels like a cheap TF	0

19097 rows × 3 columns

In [10]:
```
reviews2_df = reviews2_df.drop("gameID", axis=1).rename(columns = {"comment": "user_review", "sentiment" : "user_suggestion"})
reviews2_df
```

Out[10]:

	user_review	user_suggestion
0	Is Without Withinnbspworth your time Nonbs...	0
1	My playtime h based on steam Grindy Achieve...	0
2	No Pineapple Left Behind	0
3	PRESS SPACE TO CRASH	0
4	Reason Why Chinese Gamer Give the ShXt to W...	0
...	...	...
19092	Zombies Multiplayer Singleplayer	0

	user_review	user_suggestion
19093	Zombieswith English peopleNot Shaun of the Dea...	1
19094	Zombies Zombies everywhereAnd the only thing t...	1
19095	Zone of the Enders in a straight up bullet hel...	1
19096	thats all i can say feels like a cheap TF	0

19097 rows × 2 columns

In [11]:

```
dropped_df = dropped_df.merge(reviews2_df, on=["user_review", "user_suggestion"], how="outer")
dropped_df = dropped_df.dropna()
dropped_df
```

Out[11]:

	user_review	user_suggestion
0	I'm scared and hearing creepy voices. So I'll...	1
1	Best game, more better than Sam Pepper's YouTu...	1
2	A littly iffy on the controls, but once you kn...	1
3	Great game, fun and colorful and all that.A si...	1
4	Not many games have the cute tag right next to...	1
...	...	...
36586	Zombies Multiplayer Singleplayer	0
36587	Zombieswith English peopleNot Shaun of the Dea...	1
36588	Zombies Zombies everywhereAnd the only thing t...	1
36589	Zone of the Enders in a straight up bullet hel...	1
36590	thats all i can say feels like a cheap TF	0

36591 rows × 2 columns

Data Cleaning and Preprocessing

We perform various cleaning and preprocessing tasks. For instance, we convert any unicode data to ASCII characters. Furthermore, we add non-BOW features such as the number of exclamation points in the review, the number of all capital words (such as "HELLO"), as well as the number of words in the review. We also preprocess the text by lemmatizing it and then converting to all lowercase.

In [12]:

```
def cleaninput(s):
    return unidecode(s)

def count_exclamation(s):
    return s.count("!")

def tokenize(s):
    return s.split()

lemmatizer = WordNetLemmatizer()

def preprocess_tokenized(los):
    result = []
    for s in los:
        result.append(lemmatizer.lemmatize(s.lower()))
    return " ".join(result)

def count_capital(los):
    count = 0
    for s in los:
        if s == s.upper():
            count += 1
    return count

def strlen(s):
    return len(s)
```

In [13]:

```
cleaned_df = dropped_df

#clean the text
```

```
cleaned_df['user_review'] = cleaned_df['user_review'].map(cleaninput)

#here we count the number of exclamation points in the string before splitting on whitespace
cleaned_df['num_exclamation'] = cleaned_df['user_review'].map(count_exclamation)

#tokenize the text
cleaned_df['user_review'] = cleaned_df['user_review'].map(tokenize)

#length of the text in words
cleaned_df['length'] = cleaned_df['user_review'].map(strlen)

#here we count the number of all uppercase words before normalizing all words to be in lowercase
cleaned_df['num_all_capital'] = cleaned_df['user_review'].map(count_capital)

#preprocess the tokenized text and join it back into a string
cleaned_df['user_review'] = cleaned_df['user_review'].map(preprocess_tokenized)

cleaned_df
```

Out[13]:

	user_review	user_suggestion	num_exclamation	length	num_all_capital
0	i'm scared and hearing creepy voices. so i'll ...	1	0	132	4
1	best game, more better than sam pepper's youtu...	1	0	44	1
2	a littly iffy on the controls, but once you kn...	1	1	70	3
3	great game, fun and colorful and all that.a si...	1	1	47	1
4	not many game have the cute tag right next to ...	1	0	67	2
...	...	...	...	...	...
36586	zombie multiplayer singleplayer	0	0	3	0
36587	zombieswith english peoplenot shaun of the dea...	1	0	13	0
36588	zombie zombie everywhereand the only thing tha...	1	0	147	1
36589	zone of the enders in a straight up bullet hel...	1	0	33	1
36590	thats all i can say feel like a cheap tf	0	0	10	1

36591 rows × 5 columns

Train-test split

Here we split our data into features and target to prepare for the train/test split of the model.

In [14]:

```
features = cleaned_df[['user_review','length', 'num_exclamation', 'num_all_capital']]
target = cleaned_df['user_suggestion']

features, target
```

Out[14]:

```
(                                              user_review  length  \
 0      i'm scared and hearing creepy voices. so i'll ...     132
 1      best game, more better than sam pepper's youtu...      44
 2      a littly iffy on the controls, but once you kn...      70
 3      great game, fun and colorful and all that.a si...      47
 4      not many game have the cute tag right next to ...      67
 ...                                                  ...     ...
 36586                    zombie multiplayer singleplayer       3
 36587  zombieswith english peoplenot shaun of the dea...      13
 36588  zombie zombie everywhereand the only thing tha...     147
 36589  zone of the enders in a straight up bullet hel...      33
 36590           thats all i can say feel like a cheap tf      10

        num_exclamation  num_all_capital
 0                    0                4
 1                    0                1
 2                    1                3
 3                    1                1
 4                    0                2
 ...                ...              ...
 36586                0                0
 36587                0                0
 36588                0                1
 36589                0                1
 36590                0                1

 [36591 rows x 4 columns],
 0        1
 1        1
```

```
2        1
3        1
4        1
        ..
36586    0
36587    1
36588    1
36589    1
36590    0
Name: user_suggestion, Length: 36591, dtype: int64)
```

In [15]:

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split

#split our data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target, random_state = 3000)

X_train
```

Out[15]:

	user_review	length	num_exclamation	num_all_capital
14069	lololol all these negative reviews, there actu...	179	3	7
16563	fun, but they charge too much money to upgrade...	68	0	9
4144	it like angry bird but without the bird and yo...	47	0	0
17019	early access reviewnot much fun. ha wacky armo...	37	0	0
4844	oh smite. first off, i do have a few hundred u...	638	0	21
...	...	...	...	...
14937	bring back the 2015 version. i mean when you g...	91	1	9
34644	to be frank rome ii had probably one of the wo...	150	0	3
23578	i am really bad at this game for some reason t...	15	0	0
9208	if you're like me and got sick and tired of ev...	379	0	19
24060	i feel im missing somethingits repetitive and ...	16	0	1

27443 rows × 4 columns

Utilizing CountVectorizer and Multinomial Naive Bayes

We remove stop words from the data- words such as "the, it, yet, that"- which provide little meaning to the entirety of a document. This may provide better performance of our model. Punctuation is not taken into account, so keeping track of exclamation points may be useful here.

Furthermore, we specify the ngram_range.In this case, we look at unigrams and bigrams. This helps provide context to the features, and may lead to better performance of the model, however higher ranges may lead to overfitting.

The min_df parameter means that a given token must appear at least 2 times in the document in order to be taken into account. This allows the model to learn from more meaningful words, as a word that appears once may have little to contribute to the model.

The model we use in the below cell is Multinomial Naive Bayes, a classifier commonly used for sentiment analysis.

In [16]:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score

#initialize the CountVectorizer and fit it on the textual data
count_vect = CountVectorizer(stop_words= "english", ngram_range = (1,2), min_df=2).fit(X_train['user_review'])

#Transform our X_train and X_test into a sparse matrix
X_train_count_vectorized = count_vect.transform(X_train['user_review'])
X_test_count_vectorized = count_vect.transform(X_test['user_review'])

X_train_count_vectorized
```

Out[16]:

```
<27443x148471 sparse matrix of type '<class 'numpy.int64'>'
	with 1580225 stored elements in Compressed Sparse Row format>
```

In [17]:

```
#in order to keep our data in the same format, we convert our numerical data (non-BOW features) into a sparse matrix
numerical = X_train[['length', 'num_exclamation', 'num_all_capital']]
```

```
numerical_matrix = scipy.sparse.csr_matrix(numerical.values)

#merge into X_train_count_vectorized
training = scipy.sparse.hstack((X_train_count_vectorized, numerical_matrix))
X_train_count_vectorized = scipy.sparse.csr_matrix(training)

#do the same but with X_test
numerical = X_test[['length', 'num_exclamation', 'num_all_capital']]

numerical_matrix = scipy.sparse.csr_matrix(numerical.values)

testing = scipy.sparse.hstack((X_test_count_vectorized, numerical_matrix))
X_test_count_vectorized = scipy.sparse.csr_matrix(testing)
```

Here are some examples of the tokens that we acquired through the use of unigrams and bigrams.

Evaluating our model

We evaluate our model using the following metrics: accuracy, precision, recall, and f1 score. We also use a confusion matrix, which is represented as follows:

$$\begin{bmatrix} TruePositive & FalsePositive \\ FalseNegative & TrueNegative \end{bmatrix}$$

In [18]:

```
#fit the model and obtain predictions
mnb_count = MultinomialNB().fit(X_train_count_vectorized, y = y_train)
mnb_count_pred_train = mnb_count.predict(X_train_count_vectorized)
mnb_count_pred_test = mnb_count.predict(X_test_count_vectorized)

#evaluate the performance
print("Classification accuracy on training set: " + str(mnb_count.score(X_train_count_vectorized, y_train)))
print("Classification accuracy on testing set: " + str(mnb_count.score(X_test_count_vectorized, y_test)))
print("Precision on training set: " + str(precision_score(y_train, mnb_count_pred_train)))
print("Precision on testing set: " + str(precision_score(y_test, mnb_count_pred_test)))
print("Recall on training set: " + str(recall_score(y_train, mnb_count_pred_train)))
print("Recall on testing set: " + str(recall_score(y_test, mnb_count_pred_test)))
print("F1 score on training set: " + str(f1_score(y_train, mnb_count_pred_train)))
print("F1 score on testing set: " + str(f1_score(y_test, mnb_count_pred_test)))
print("Training set confusion matrix:\n ", confusion_matrix(y_train, mnb_count_pred_train))
print("Testing set confusion matrix:\n ", confusion_matrix(y_test, mnb_count_pred_test))
```

```
Classification accuracy on training set: 0.9214007214954634
Classification accuracy on testing set: 0.8196327066025361
Precision on training set: 0.9359537249982577
Precision on testing set: 0.8493759255341654
Recall on training set: 0.9155985819470958
Recall on testing set: 0.8106198263678579
F1 score on training set: 0.9256642657752352
F1 score on testing set: 0.8295454545454547
Training set confusion matrix:
  [[11856   919]
 [ 1238 13430]]
Testing set confusion matrix:
  [[3483  712]
 [ 938 4015]]
```

Hyperparameter Tuning and Evaluation

Here, we will use grid-search cross-validation in order to find the best hyperparameters for our model.

In [19]:

```
from sklearn.model_selection import GridSearchCV

#set up a grid of parameters to try out, and fit it on our data
grid1 = {"alpha": [.001, .01, .1, .5, 1, 10, 100]}

grid_search_mnb1 = GridSearchCV(MultinomialNB(), grid1, cv = 5)

grid_search_mnb1.fit(X=X_train_count_vectorized, y=y_train)

#make predictions using best parameter
grid_mnb_count_pred_train = grid_search_mnb1.predict(X_train_count_vectorized)
grid_mnb_count_pred_test = grid_search_mnb1.predict(X_test_count_vectorized)


#evaluate performance
print("Best parameter for alpha: " + str(grid_search_mnb1.best_params_['alpha']))
print("Best score with this alpha value: " + str(grid_search_mnb1.best_score_))
print("\nSCORES USING BEST ALPHA VALUE\n")
```

```
print("Classification accuracy on training set: " + str(grid_search_mnb1.score(X_train_count_vectorized, y_train)))
print("Classification accuracy on testing set: " + str(grid_search_mnb1.score(X_test_count_vectorized, y_test)))
print("Precision on training set: " + str(precision_score(y_train, grid_mnb_count_pred_train)))
print("Precision on testing set: " + str(precision_score(y_test, grid_mnb_count_pred_test)))
print("Recall on training set: " + str(recall_score(y_train, grid_mnb_count_pred_train)))
print("Recall on testing set: " + str(recall_score(y_test, grid_mnb_count_pred_test)))
print("F1 score on training set: " + str(f1_score(y_train, grid_mnb_count_pred_train)))
print("F1 score on testing set: " + str(f1_score(y_test, grid_mnb_count_pred_test)))
print("Training set confusion matrix:\n ", confusion_matrix(y_train, grid_mnb_count_pred_train))
print("Testing set confusion matrix:\n ", confusion_matrix(y_test, grid_mnb_count_pred_test))
```

```
Best parameter for alpha: 1
Best score with this alpha value: 0.8129575411092527

SCORES USING BEST ALPHA VALUE

Classification accuracy on training set: 0.9214007214954634
Classification accuracy on testing set: 0.8196327066025361
Precision on training set: 0.9359537249982577
Precision on testing set: 0.8493759255341654
Recall on training set: 0.9155985819470958
Recall on testing set: 0.8106198263678579
F1 score on training set: 0.9256642657752352
F1 score on testing set: 0.8295454545454547
Training set confusion matrix:
  [[11856   919]
 [ 1238 13430]]
Testing set confusion matrix:
  [[3483  712]
 [ 938 4015]]
```

Utilizing Tf-idf Vectorizer and Multinomial Naive Bayes

Tf-idf differs from a CountVectorizer in the way a feature's "importance" is calculated. A CountVectorizer simply counts the occurrence of a word in the text, however a Tf-idf Vectorizer uses the formula

$$tfidf(w,d) = (tf) \cdot log(\frac{N+1}{N_w+1}) + 1$$

where tf is the number of times word w shows up in document d, N is the number of documents, and N_w is the number of documents where w is present.

We continue to use the Multinomial Naive Bayes classifier to build our model.

In [20]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

#Use Tfidf vectorizer to fit and transform our text data
tfidf_vect = TfidfVectorizer(stop_words= "english", ngram_range = (1,2), min_df=2).fit(X_train['user_review'])

X_train_tfidf_vectorized = tfidf_vect.transform(X_train['user_review'])
X_test_tfidf_vectorized = tfidf_vect.transform(X_test['user_review'])

#append numerical data to our sparse matrix
numerical = X_train[['length', 'num_exclamation', 'num_all_capital']]

numerical_matrix = scipy.sparse.csr_matrix(numerical.values)

training = scipy.sparse.hstack((X_train_tfidf_vectorized, numerical_matrix))
X_train_tfidf_vectorized = scipy.sparse.csr_matrix(training)

numerical = X_test[['length', 'num_exclamation', 'num_all_capital']]

numerical_matrix = scipy.sparse.csr_matrix(numerical.values)

testing = scipy.sparse.hstack((X_test_tfidf_vectorized, numerical_matrix))
X_test_tfidf_vectorized = scipy.sparse.csr_matrix(testing)
```

Evaluating our model

We evaluate our model using classification accuracy, f1 score, and a confusion matrix.

In [21]:

```
#fit model on the tfidf vectorized data

mnb_tfidf = MultinomialNB().fit(X_train_tfidf_vectorized, y = y_train)
mnb_tfidf_pred_train = mnb_tfidf.predict(X_train_tfidf_vectorized)
mnb_tfidf_pred_test = mnb_tfidf.predict(X_test_tfidf_vectorized)

#evaluate performance
```

```
print("Classification accuracy on training set: " + str(mnb_tfidf.score(X_train_tfidf_vectorized, y_train)))
print("Classification accuracy on testing set: " + str(mnb_tfidf.score(X_test_tfidf_vectorized, y_test)))
print("Precision on training set: " + str(precision_score(y_train, mnb_tfidf_pred_train)))
print("Precision on testing set: " + str(precision_score(y_test, mnb_tfidf_pred_test)))
print("Recall on training set: " + str(recall_score(y_train, mnb_tfidf_pred_train)))
print("Recall on testing set: " + str(recall_score(y_test, mnb_tfidf_pred_test)))
print("F1 score on training set: " + str(f1_score(y_train, mnb_tfidf_pred_train)))
print("F1 score on testing set: " + str(f1_score(y_test, mnb_tfidf_pred_test)))
print("Training set confusion matrix:\n ", confusion_matrix(y_train, mnb_tfidf_pred_train))
print("Testing set confusion matrix:\n ", confusion_matrix(y_test, mnb_tfidf_pred_test))
```

```
Classification accuracy on training set: 0.8934154429180483
Classification accuracy on testing set: 0.7986445124617403
Precision on training set: 0.9167435588047412
Precision on testing set: 0.8275426405559065
Recall on training set: 0.8805563130624489
Recall on testing set: 0.7934585099939431
F1 score on training set: 0.8982856348019612
F1 score on testing set: 0.8101422387136673
Training set confusion matrix:
  [[11602  1173]
 [ 1752 12916]]
Testing set confusion matrix:
  [[3376  819]
 [1023 3930]]
```

Hyperparameter Tuning and Evaluation

In this section, we do some hyperparameter tuning for the tfidf vectorized data.

In [22]:

```
#set up grid of parameters
grid2 = {"alpha": [.001, .01, .1, .5, 1, 10, 100]}

grid_search_mnb2 = GridSearchCV(MultinomialNB(), grid2, cv = 5)

#use grid to find best alpha parameter
grid_search_mnb2.fit(X=X_train_tfidf_vectorized, y=y_train)

grid_mnb_tfidf_pred_train = grid_search_mnb2.predict(X_train_tfidf_vectorized)
grid_mnb_tfidf_pred_test = grid_search_mnb2.predict(X_test_tfidf_vectorized)

#evaluate peformance using this best hyperparameter
print("Best parameter for alpha: " + str(grid_search_mnb2.best_params_['alpha']))
print("Best score with this alpha value: " + str(grid_search_mnb2.best_score_))
print("\nSCORES USING BEST ALPHA VALUE\n")
print("Classification accuracy on training set: " + str(grid_search_mnb2.score(X_train_tfidf_vectorized, y_train)))
print("Classification accuracy on testing set: " + str(grid_search_mnb2.score(X_test_tfidf_vectorized, y_test)))
print("Precision on training set: " + str(precision_score(y_train, grid_mnb_tfidf_pred_train)))
print("Precision on testing set: " + str(precision_score(y_test, grid_mnb_tfidf_pred_test)))
print("Recall on training set: " + str(recall_score(y_train, grid_mnb_tfidf_pred_train)))
print("Recall on testing set: " + str(recall_score(y_test, grid_mnb_tfidf_pred_test)))
print("F1 score on training set: " + str(f1_score(y_train, grid_mnb_tfidf_pred_train)))
print("F1 score on testing set: " + str(f1_score(y_test, grid_mnb_tfidf_pred_test)))
print("Training set confusion matrix:\n ", confusion_matrix(y_train, grid_mnb_tfidf_pred_train))
print("Testing set confusion matrix:\n ", confusion_matrix(y_test, grid_mnb_tfidf_pred_test))
```

```
Best parameter for alpha: 0.1
Best score with this alpha value: 0.7982725389820192

SCORES USING BEST ALPHA VALUE

Classification accuracy on training set: 0.9536493823561564
Classification accuracy on testing set: 0.8031263664188893
Precision on training set: 0.9588928473554399
Precision on testing set: 0.8233483791546984
Recall on training set: 0.9541859830924462
Recall on testing set: 0.8102160306884716
F1 score on training set: 0.9565336249316566
F1 score on testing set: 0.8167294189477969
Training set confusion matrix:
  [[12175   600]
 [  672 13996]]
Testing set confusion matrix:
  [[3334  861]
 [ 940 4013]]
```

Utilizing CountVectorizer and Logistic Regression

In [23]:

```
from sklearn.linear_model import LogisticRegression

#initialize our model. we use max_iter=1000 so that we reduce the time complexity of our model
log_count = LogisticRegression(max_iter=1000).fit(X= X_train_count_vectorized, y=y_train)
```

```
C:\Users\Justin\AppData\Local\Programs\Python\Python37\lib\site-packages\sklearn\linear_model\_logistic.py:764: ConvergenceWa
rning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
```

Evaluating our model

In [24]:

```
#make predictions based off fitted model
log_count_pred_train = log_count.predict(X_train_count_vectorized)
log_count_pred_test = log_count.predict(X_test_count_vectorized)


#evaluate performance
print("Classification accuracy on training set: " + str(log_count.score(X_train_count_vectorized, y_train)))
print("Classification accuracy on testing set: " + str(log_count.score(X_test_count_vectorized, y_test)))
print("Precision on training set: " + str(precision_score(y_train, log_count_pred_train)))
print("Precision on testing set: " + str(precision_score(y_test, log_count_pred_test)))
print("Recall on training set: " + str(recall_score(y_train, log_count_pred_train)))
print("Recall on testing set: " + str(recall_score(y_test, log_count_pred_test)))
print("F1 score on training set: " + str(f1_score(y_train, log_count_pred_train)))
print("F1 score on testing set: " + str(f1_score(y_test, log_count_pred_test)))
print("Training set confusion matrix:\n ", confusion_matrix(y_train, log_count_pred_train))
print("Testing set confusion matrix:\n ", confusion_matrix(y_test, log_count_pred_test))
```

```
Classification accuracy on training set: 0.976059468716977
Classification accuracy on testing set: 0.8212724092697857
Precision on training set: 0.9707680935421007
Precision on testing set: 0.827089905362776
Recall on training set: 0.9848650122716117
Recall on testing set: 0.8469614375126187
F1 score on training set: 0.9777657450336729
F1 score on testing set: 0.8369077306733167
Training set confusion matrix:
  [[12340   435]
 [  222 14446]]
Testing set confusion matrix:
  [[3318  877]
 [ 758 4195]]
```

Hyperparameter Tuning and Evaluation

Here we adjust the C value of our Logistic Regression model, in order to achieve the best results of this specific model.

In [25]:

```
#set up a gridsearchCV to find best C parameter. again, we limit max_iter to reduce time complexity of this search
grid3 = {"C": [.01, .1, .5, 1]}

grid_search_log1 = GridSearchCV(LogisticRegression(max_iter=750), grid3, cv=5)
grid_search_log1.fit(X=X_train_count_vectorized, y=y_train)
```

```
C:\Users\Justin\AppData\Local\Programs\Python\Python37\lib\site-packages\sklearn\linear_model\_logistic.py:764: ConvergenceWa
rning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

C:\Users\Justin\AppData\Local\Programs\Python\Python37\lib\site-packages\sklearn\linear_model\_logistic.py:764: ConvergenceWa
rning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

C:\Users\Justin\AppData\Local\Programs\Python\Python37\lib\site-packages\sklearn\linear_model\_logistic.py:764: ConvergenceWa
rning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

C:\Users\Justin\AppData\Local\Programs\Python\Python37\lib\site-packages\sklearn\linear_model\_logistic.py:764: ConvergenceWa
rning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

C:\Users\Justin\AppData\Local\Programs\Python\Python37\lib\site-packages\sklearn\linear_model\_logistic.py:764: ConvergenceWa
rning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

C:\Users\Justin\AppData\Local\Programs\Python\Python37\lib\site-packages\sklearn\linear_model\_logistic.py:764: ConvergenceWa
rning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

C:\Users\Justin\AppData\Local\Programs\Python\Python37\lib\site-packages\sklearn\linear_model\_logistic.py:764: ConvergenceWa
rning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

C:\Users\Justin\AppData\Local\Programs\Python\Python37\lib\site-packages\sklearn\linear_model\_logistic.py:764: ConvergenceWa
rning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

C:\Users\Justin\AppData\Local\Programs\Python\Python37\lib\site-packages\sklearn\linear_model\_logistic.py:764: ConvergenceWa
rning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

C:\Users\Justin\AppData\Local\Programs\Python\Python37\lib\site-packages\sklearn\linear_model\_logistic.py:764: ConvergenceWa
rning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

C:\Users\Justin\AppData\Local\Programs\Python\Python37\lib\site-packages\sklearn\linear_model\_logistic.py:764: ConvergenceWa
rning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
```

```
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

C:\Users\Justin\AppData\Local\Programs\Python\Python37\lib\site-packages\sklearn\linear_model\_logistic.py:764: ConvergenceWa
rning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

C:\Users\Justin\AppData\Local\Programs\Python\Python37\lib\site-packages\sklearn\linear_model\_logistic.py:764: ConvergenceWa
rning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

C:\Users\Justin\AppData\Local\Programs\Python\Python37\lib\site-packages\sklearn\linear_model\_logistic.py:764: ConvergenceWa
rning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

C:\Users\Justin\AppData\Local\Programs\Python\Python37\lib\site-packages\sklearn\linear_model\_logistic.py:764: ConvergenceWa
rning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

C:\Users\Justin\AppData\Local\Programs\Python\Python37\lib\site-packages\sklearn\linear_model\_logistic.py:764: ConvergenceWa
rning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
```

Out[25]:
```
GridSearchCV(cv=5, estimator=LogisticRegression(max_iter=750),
             param_grid={'C': [0.01, 0.1, 0.5, 1]})
```

In [26]:
```python
#make predictions based on best C parameter
grid_log_count_pred_train = grid_search_log1.predict(X_train_count_vectorized)
grid_log_count_pred_test = grid_search_log1.predict(X_test_count_vectorized)

#evaluate performance
print("Best parameter for C: " + str(grid_search_log1.best_params_['C']))
print("Best score with this C value: " + str(grid_search_log1.best_score_))
print("\nSCORES USING BEST C VALUE\n")
print("Classification accuracy on training set: " + str(grid_search_log1.score(X_train_count_vectorized, y_train)))
print("Classification accuracy on testing set: " + str(grid_search_log1.score(X_test_count_vectorized, y_test)))
print("Precision on training set: " + str(precision_score(y_train, grid_log_count_pred_train)))
print("Precision on testing set: " + str(precision_score(y_test, grid_log_count_pred_test)))
print("Recall on training set: " + str(recall_score(y_train, grid_log_count_pred_train)))
print("Recall on testing set: " + str(recall_score(y_test, grid_log_count_pred_test)))
print("F1 score on training set: " + str(f1_score(y_train, grid_log_count_pred_train)))
print("F1 score on testing set: " + str(f1_score(y_test, grid_log_count_pred_test)))
print("Training set confusion matrix:\n ", confusion_matrix(y_train, grid_log_count_pred_train))
print("Testing set confusion matrix:\n ", confusion_matrix(y_test, grid_log_count_pred_test))
```

```
Best parameter for C: 0.5
Best score with this C value: 0.8183870324800143

SCORES USING BEST C VALUE
```

```
Classification accuracy on training set: 0.9579127646394344
Classification accuracy on testing set: 0.8210537822474858
Precision on training set: 0.9504033064462369
Precision on testing set: 0.8267638943634213
Recall on training set: 0.9719798200163622
Recall on testing set: 0.8469614375126187
F1 score on training set: 0.9610704776028852
F1 score on testing set: 0.8367407998404307
Training set confusion matrix:
  [[12031   744]
 [  411 14257]]
Testing set confusion matrix:
  [[3316  879]
 [ 758 4195]]
```

Understanding the model

In the dataframe below, we can see that the model determines the probabilities of a given review to either be recommending of the game or not. The model then makes a prediction based on which probability is higher.

In [27]:

```python
probs = pd.DataFrame(grid_search_log1.predict_proba(X_test_count_vectorized), columns = ["Not Recommended", "Recommended"])
probs["Prediction"] = grid_search_log1.predict(X_test_count_vectorized)
probs
```

Out[27]:

	Not Recommended	Recommended	Prediction
0	0.964001	0.035999	0
1	0.563371	0.436629	0
2	0.315965	0.684035	1
3	0.988288	0.011712	0
4	0.616326	0.383674	0
...	...	...	...
9143	0.524260	0.475740	0
9144	0.644771	0.355229	0
9145	0.999565	0.000435	0
9146	0.777982	0.222018	0
9147	0.001450	0.998550	1

9148 rows × 3 columns

Utilizing Tf-idf Vectorizer and Logistic Regression

Now, we use the data transformed by the tf-idf vectorizer and fit it using the logistic regression model.

In [28]:

```python
#initialize logistic regression model that takes in tfidf vectorized data
log_tfidf = LogisticRegression(max_iter=1000).fit(X= X_train_tfidf_vectorized, y=y_train)
```

Evaluating our model

In [29]:

```python
#make predictions on data
log_tfidf_pred_train = log_tfidf.predict(X_train_tfidf_vectorized)
log_tfidf_pred_test = log_tfidf.predict(X_test_tfidf_vectorized)

#evaluate performance
print("Classification accuracy on training set: " + str(log_tfidf.score(X_train_tfidf_vectorized, y_train)))
print("Classification accuracy on testing set: " + str(log_tfidf.score(X_test_tfidf_vectorized, y_test)))
print("Precision on training set: " + str(precision_score(y_train, log_tfidf_pred_train)))
print("Precision on testing set: " + str(precision_score(y_test, log_tfidf_pred_test)))
print("Recall on training set: " + str(recall_score(y_train, log_tfidf_pred_train)))
print("Recall on testing set: " + str(recall_score(y_test, log_tfidf_pred_test)))
print("F1 score on training set: " + str(f1_score(y_train, log_tfidf_pred_train)))
print("F1 score on testing set: " + str(f1_score(y_test, log_tfidf_pred_test)))
print("Training set confusion matrix:\n ", confusion_matrix(y_train, log_tfidf_pred_train))
print("Testing set confusion matrix:\n ", confusion_matrix(y_test, log_tfidf_pred_test))
```

```
Classification accuracy on training set: 0.9013956200123893
Classification accuracy on testing set: 0.8236773065150853
Precision on training set: 0.8980433914548117
Precision on testing set: 0.8288696337140606
Recall on training set: 0.919961821652577
Recall on testing set: 0.8497880072683223
```

```
F1 score on training set: 0.9088704788846231
F1 score on testing set: 0.8391984846974379
Training set confusion matrix:
  [[11243  1532]
 [ 1174 13494]]
Testing set confusion matrix:
  [[3326  869]
 [ 744 4209]]
```

Hyperparameter Tuning and Evaluation

In [30]:

```
#set up GridSearch CV for best C parameter
grid4 = {"C": [.01, .1, .5, 1]}

#fit on tfidf vectorized data
grid_search_log2 = GridSearchCV(LogisticRegression(max_iter=750), grid4, cv=5)
grid_search_log2.fit(X=X_train_tfidf_vectorized, y=y_train)
```

Out[30]:

```
GridSearchCV(cv=5, estimator=LogisticRegression(max_iter=750),
             param_grid={'C': [0.01, 0.1, 0.5, 1]})
```

In [31]:

```
#get predictions using best hyperparameters
grid_log_tfidf_pred_train = grid_search_log2.predict(X_train_tfidf_vectorized)
grid_log_tfidf_pred_test = grid_search_log2.predict(X_test_tfidf_vectorized)

#evaluate performance
print("Best parameter for C: " + str(grid_search_log2.best_params_['C']))
print("Best score with this C value: " + str(grid_search_log2.best_score_))
print("\nSCORES USING BEST C VALUE\n")
print("Classification accuracy on training set: " + str(grid_search_log2.score(X_train_tfidf_vectorized, y_train)))
print("Classification accuracy on testing set: " + str(grid_search_log2.score(X_test_tfidf_vectorized, y_test)))
print("Precision on training set: " + str(precision_score(y_train, grid_log_tfidf_pred_train)))
print("Precision on testing set: " + str(precision_score(y_test, grid_log_tfidf_pred_test)))
print("Recall on training set: " + str(recall_score(y_train, grid_log_tfidf_pred_train)))
print("Recall on testing set: " + str(recall_score(y_test, grid_log_tfidf_pred_test)))
print("F1 score on training set: " + str(f1_score(y_train, grid_log_tfidf_pred_train)))
print("F1 score on testing set: " + str(f1_score(y_test, grid_log_tfidf_pred_test)))
print("Training set confusion matrix:\n ", confusion_matrix(y_train, grid_log_tfidf_pred_train))
print("Testing set confusion matrix:\n ", confusion_matrix(y_test, grid_log_tfidf_pred_test))
```

```
Best parameter for C: 1
Best score with this C value: 0.8213021656883871

SCORES USING BEST C VALUE

Classification accuracy on training set: 0.9013956200123893
Classification accuracy on testing set: 0.8236773065150853
Precision on training set: 0.8980433914548117
Precision on testing set: 0.8288696337140606
Recall on training set: 0.919961821652577
Recall on testing set: 0.8497880072683223
F1 score on training set: 0.9088704788846231
F1 score on testing set: 0.8391984846974379
Training set confusion matrix:
  [[11243  1532]
 [ 1174 13494]]
Testing set confusion matrix:
  [[3326  869]
 [ 744 4209]]
```

Utilizing CountVectorizer and Random Forest Classifier

Another popular model used in sentiment analysis is the Random Forest Classifier. This model builds a forest, which is essentially a collection of decision trees. By merging these decision trees together, a better prediction can be made.

In [32]:

```
from sklearn.ensemble import RandomForestClassifier

#initialize an rfc on countvectorized data
rand_count = RandomForestClassifier(random_state=3000).fit(X=X_train_count_vectorized, y=y_train)
```

Evaluating our Model

In [33]:

```
#predict using our fitted model
rand_count_pred_train = rand_count.predict(X_train_count_vectorized)
rand_count_pred_test = rand_count.predict(X_test_count_vectorized)

#evaluate performance
print("Classification accuracy on training set: " + str(rand_count.score(X_train_count_vectorized, y_train)))
print("Classification accuracy on testing set: " + str(rand_count.score(X_test_count_vectorized, y_test)))
```

```
print("Precision on training set: " + str(precision_score(y_train, rand_count_pred_train)))
print("Precision on testing set: " + str(precision_score(y_test, rand_count_pred_test)))
print("Recall on training set: " + str(recall_score(y_train, rand_count_pred_train)))
print("Recall on testing set: " + str(recall_score(y_test, rand_count_pred_test)))
print("F1 score on training set: " + str(f1_score(y_train, rand_count_pred_train)))
print("F1 score on testing set: " + str(f1_score(y_test, rand_count_pred_test)))
print("Training set confusion matrix:\n ", confusion_matrix(y_train, rand_count_pred_train))
print("Testing set confusion matrix:\n ", confusion_matrix(y_test, rand_count_pred_test))
```

```
Classification accuracy on training set: 0.9975221367926247
Classification accuracy on testing set: 0.7993003935286401
Precision on training set: 0.9983615510649918
Precision on testing set: 0.805408583186361
Recall on training set: 0.9970002727024816
Recall on testing set: 0.8298001211387038
F1 score on training set: 0.9976804475371812
F1 score on testing set: 0.8174224343675417
Training set confusion matrix:
  [[12751    24]
 [   44 14624]]
Testing set confusion matrix:
  [[3202  993]
 [ 843 4110]]
```

Using RandomSearch and GridSearch

We can see that there is some overfitting in the base model, with default parameters. There are a variety of hyperparameters to try out for a RandomForestClassifier. Testing out all of these parameters at once would require a large amount of time to execute. For instance, if we adjust 5 parameters, with 10 values each, thats 10^5 different combinations to try out! Furthermore, increasing the cv value will reduce overfitting, but will in turn take more time to execute. As a result, we must find a good balance between performance and time.

As a result, we use RandomSearch first. By randomly selecting a specified amount of hyperparameter settings and finding the best performance of those models, we obtain a general sense of the best hyperparameter settings. As a result, we can narrow down the range of values we can use for a GridSearch using the results from the RandomSearch, so we wouldn't have to exhaustively search through all 10^5 settings.

In [34]:
```
from sklearn.model_selection import RandomizedSearchCV

#initialize grid of parameters in which randomizedsearchCV randomly chooses from
random_grid = {"n_estimators": [100, 200],
               "bootstrap": [True, False],
               "max_depth": [None, 10, 20],
              "min_samples_leaf": [1, 5],
              "min_samples_split": [2, 5]}

rf_random = RandomizedSearchCV(estimator=RandomForestClassifier(),
                              param_distributions=random_grid, cv=3, n_iter= 10, random_state=3000, n_jobs = -1)

#fit grid on model
rf_random.fit(X=X_train_count_vectorized, y=y_train)
```

Out[34]:
```
RandomizedSearchCV(cv=3, estimator=RandomForestClassifier(), n_jobs=-1,
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [None, 10, 20],
                                        'min_samples_leaf': [1, 5],
                                        'min_samples_split': [2, 5],
                                        'n_estimators': [100, 200]},
                   random_state=3000)
```

In [35]:
```
#the parameters that produce best result, and its corresponding score
print(rf_random.best_params_)
print(rf_random.best_score_)
```

```
{'n_estimators': 200, 'min_samples_split': 5, 'min_samples_leaf': 1, 'max_depth': None, 'bootstrap': False}
0.8018437163123293
```

We can see from the output that min_samples_leaf and max_depth should be set to 1, and bootstrap should be set to False. However, we narrow down the ranges for n_estimators and min_samples_split in order to find the best hyperparameters for those attributes. In this GridSearch, we check 9 different combinations of hyperparameters, as opposed to the 48 in RandomSearch (in which we only looked through 10).

Note: We intended to use a narrowed-down GridSearchCV after our RandomSearchCV to hone in on the best hyperparameters for our model, however this took too long to run. We commented out the code to show you the process in which we tried to find the optimal hyperparameters for an RFC. Feel free to uncomment out the code if your computer has ample processing power.

In [36]:
```
# rf_cv_grid = {"n_estimators": [200, 250], "min_samples_split": [5, 10]}

# rf_cv = GridSearchCV(RandomForestClassifier(bootstrap=False), rf_cv_grid, cv=3, n_jobs=-1)
```

```
# rf_cv.fit(X=X_train_count_vectorized, y=y_train)
```

In [37]:

```
# print(rf_cv.best_params_)
# print(rf_cv.best_score_)
```

Using the best hyperparameters

Since we have used RandomSearch and GridSearch to find the optimal hyperparameters for the RandomForestClassifier, we evaluate its performance on the training and testing set.

In [38]:

```
# rand_count_best = RandomForestClassifier(bootstrap=False, min_samples_split=10, n_estimators=250)
# rand_count_best.fit(X=X_train_count_vectorized, y=y_train)
```

In [39]:

```
# rand_count_best_pred_train = rand_count_best.predict(X_train_count_vectorized)
# rand_count_best_pred_test = rand_count_best.predict(X_test_count_vectorized)

# print("Classification accuracy on training set: " + str(rand_count_best.score(X_train_count_vectorized, y_train)))
# print("Classification accuracy on testing set: " + str(rand_count_best.score(X_test_count_vectorized, y_test)))
# print("Precision on training set: " + str(precision_score(y_train, rand_count_best_pred_train)))
# print("Precision on testing set: " + str(precision_score(y_test, rand_count_best_pred_test)))
# print("Recall on training set: " + str(recall_score(y_train, rand_count_best_pred_train)))
# print("Recall on testing set: " + str(recall_score(y_test, rand_count_best_pred_test)))
# print("F1 score on training set: " + str(f1_score(y_train, rand_count_best_pred_train)))
# print("F1 score on testing set: " + str(f1_score(y_test, rand_count_best_pred_test)))
# print("Training set confusion matrix:\n ", confusion_matrix(y_train, rand_count_best_pred_train))
# print("Testing set confusion matrix:\n ", confusion_matrix(y_test, rand_count_best_pred_test))
```

Utilizing Tf-idf Vectorizer and Random Forest Classifier

Here we use a Tf-idf vectorizer in conjunction with the Random Forest Classifier.

In [40]:

```
#initialize rfc using tfidf vectorized data
rand_tfidf = RandomForestClassifier(random_state=3000).fit(X=X_train_tfidf_vectorized, y=y_train)
```

Evaluating our Model

In [41]:

```
#make predictions based off this fitted model
rand_tfidf_pred_train = rand_tfidf.predict(X_train_tfidf_vectorized)
rand_tfidf_pred_test = rand_tfidf.predict(X_test_tfidf_vectorized)

#evaluate performance
print("Classification accuracy on training set: " + str(rand_tfidf.score(X_train_tfidf_vectorized, y_train)))
print("Classification accuracy on testing set: " + str(rand_tfidf.score(X_test_tfidf_vectorized, y_test)))
print("Precision on training set: " + str(precision_score(y_train, rand_tfidf_pred_train)))
print("Precision on testing set: " + str(precision_score(y_test, rand_tfidf_pred_test)))
print("Recall on training set: " + str(recall_score(y_train, rand_tfidf_pred_train)))
print("Recall on testing set: " + str(recall_score(y_test, rand_tfidf_pred_test)))
print("F1 score on training set: " + str(f1_score(y_train, rand_tfidf_pred_train)))
print("F1 score on testing set: " + str(f1_score(y_test, rand_tfidf_pred_test)))
print("Training set confusion matrix:\n ", confusion_matrix(y_train, rand_tfidf_pred_train))
print("Testing set confusion matrix:\n ", confusion_matrix(y_test, rand_tfidf_pred_test))
```

```
Classification accuracy on training set: 0.9975221367926247
Classification accuracy on testing set: 0.8022518583296896
Precision on training set: 0.9983615510649918
Precision on testing set: 0.8042973286875726
Recall on training set: 0.9970002727024816
Recall on testing set: 0.838885523924894
F1 score on training set: 0.9976804475371812
F1 score on testing set: 0.821227394011266
Training set confusion matrix:
  [[12751    24]
 [   44 14624]]
Testing set confusion matrix:
  [[3184 1011]
 [ 798 4155]]
```

Using RandomSearch and GridSearch

In [42]:

```
#initialize random grid and fit it on tfidf vectorized data
random_grid2 = {"n_estimators": [100, 200],
               "bootstrap": [True, False],
```

```
                 "max_depth": [None, 10, 20],
                "min_samples_leaf": [1, 5],
                "min_samples_split": [2, 5]}

rf_random2 = RandomizedSearchCV(estimator=RandomForestClassifier(),
                               param_distributions=random_grid2, cv=3, n_iter= 10, random_state=3000, n_jobs = -1)

rf_random2.fit(X=X_train_tfidf_vectorized, y=y_train)
```

Out[42]:
```
RandomizedSearchCV(cv=3, estimator=RandomForestClassifier(), n_jobs=-1,
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [None, 10, 20],
                                        'min_samples_leaf': [1, 5],
                                        'min_samples_split': [2, 5],
                                        'n_estimators': [100, 200]},
                   random_state=3000)
```

In [43]:
```
#the parameters that produce best result, and its corresponding score
print(rf_random2.best_params_)
print(rf_random2.best_score_)
```

```
{'n_estimators': 200, 'min_samples_split': 5, 'min_samples_leaf': 1, 'max_depth': None, 'bootstrap': False}
0.8018801700837127
```

Note: We intended to use a narrowed-down GridSearchCV after our RandomSearchCV to hone in on the best hyperparameters for our model, however this took too long to run. We commented out the code to show you the process in which we tried to find the optimal hyperparameters for an RFC. Feel free to uncomment out the code if your computer has ample processing power.

In [44]:
```
# rf_cv_grid2 = {"n_estimators": [200, 250], "min_samples_split": [5, 10]}

# rf_cv2 = GridSearchCV(RandomForestClassifier(bootstrap=False), rf_cv_grid2, cv=3, n_jobs=-1)

# rf_cv2.fit(X=X_train_tfidf_vectorized, y=y_train)
```

In [45]:
```
# print(rf_cv2.best_params_)
# print(rf_cv2.best_score_)
```

Using the best hyperparameters

Now, we use the best hyperparameters specified by our gridsearch and evaluate its performance.

In [46]:
```
# rand_tfidf_best = RandomForestClassifier(bootstrap=False, min_samples_split=5, n_estimators=250)
# rand_tfidf_best.fit(X=X_train_tfidf_vectorized, y=y_train)
```

In [47]:
```
# rand_tfidf_best_pred_train = rand_tfidf_best.predict(X_train_tfidf_vectorized)
# rand_tfidf_best_pred_test = rand_tfidf_best.predict(X_test_tfidf_vectorized)

# print("Classification accuracy on training set: " + str(rand_tfidf_best.score(X_train_tfidf_vectorized, y_train)))
# print("Classification accuracy on testing set: " + str(rand_tfidf_best.score(X_test_tfidf_vectorized, y_test)))
# print("Precision on training set: " + str(precision_score(y_train, rand_tfidf_best_pred_train)))
# print("Precision on testing set: " + str(precision_score(y_test, rand_tfidf_best_pred_test)))
# print("Recall on training set: " + str(recall_score(y_train, rand_tfidf_best_pred_train)))
# print("Recall on testing set: " + str(recall_score(y_test, rand_tfidf_best_pred_test)))
# print("F1 score on training set: " + str(f1_score(y_train, rand_tfidf_best_pred_train)))
# print("F1 score on testing set: " + str(f1_score(y_test, rand_tfidf_best_pred_test)))
# print("Training set confusion matrix:\n ", confusion_matrix(y_train, rand_tfidf_best_pred_train))
# print("Testing set confusion matrix:\n ", confusion_matrix(y_test, rand_tfidf_best_pred_test))
```

CountVectorizer and KNeighborsClassifier

The **KNeighborsClassifier** works as follows: given a data point, look at the nearest N neighbors of that point, and choose the majority class of those neighbors. Our model will be described in more detail in our report.

In [48]:
```
from sklearn.neighbors import KNeighborsClassifier

#initialize model and fit it on countvectorized data
knn_count = KNeighborsClassifier().fit(X=X_train_count_vectorized, y=y_train)

#make predictions based off fitted model
knn_count_pred_train = knn_count.predict(X_train_count_vectorized)
knn_count_pred_test = knn_count.predict(X_test_count_vectorized)

#evaluate performance
```

```
print("Classification accuracy on training set: " + str(knn_count.score(X_train_count_vectorized, y_train)))
print("Classification accuracy on testing set: " + str(knn_count.score(X_test_count_vectorized, y_test)))
print("Precision on training set: " + str(precision_score(y_train, knn_count_pred_train)))
print("Precision on testing set: " + str(precision_score(y_test, knn_count_pred_test)))
print("Recall on training set: " + str(recall_score(y_train, knn_count_pred_train)))
print("Recall on testing set: " + str(recall_score(y_test, knn_count_pred_test)))
print("F1 score on training set: " + str(f1_score(y_train, knn_count_pred_train)))
print("F1 score on testing set: " + str(f1_score(y_test, knn_count_pred_test)))
print("Training set confusion matrix:\n ", confusion_matrix(y_train, knn_count_pred_train))
print("Testing set confusion matrix:\n ", confusion_matrix(y_test, knn_count_pred_test))
```

```
Classification accuracy on training set: 0.7306052545275662
Classification accuracy on testing set: 0.5585920419763882
Precision on training set: 0.7300613496932515
Precision on testing set: 0.5880993645291739
Recall on training set: 0.786951186255795
Recall on testing set: 0.6165960024227741
F1 score on training set: 0.7574395485416188
F1 score on testing set: 0.6020106445890006
Training set confusion matrix:
  [[ 8507  4268]
 [ 3125 11543]]
Testing set confusion matrix:
  [[2056 2139]
 [1899 3054]]
```

Hyperparameter Tuning and Evaluation

In [49]:

```
#initialize grid to find best hyperparameters
grid5 = {"n_neighbors": [3, 5], "weights": ["uniform", "distance"]}

#fit gridsearch on countvectorized data
grid_search_knn1 = GridSearchCV(KNeighborsClassifier(), grid5, cv=5)
grid_search_knn1.fit(X=X_train_count_vectorized, y=y_train)
```

Out[49]:

```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': [3, 5],
                         'weights': ['uniform', 'distance']})
```

In [50]:

```
#make predictions based off best hyperparameters
grid_knn_count_pred_train = grid_search_knn1.predict(X_train_count_vectorized)
grid_knn_count_pred_test = grid_search_knn1.predict(X_test_count_vectorized)

#evaluate performance
print("Best parameter for n_neighbors: " + str(grid_search_knn1.best_params_['n_neighbors']))
print("Best parameter for distance: " + str(grid_search_knn1.best_params_['weights']))
print("Best score with these values: " + str(grid_search_knn1.best_score_))
print("\nSCORES USING BEST VALUES\n")
print("Classification accuracy on training set: " + str(grid_search_knn1.score(X_train_count_vectorized, y_train)))
print("Classification accuracy on testing set: " + str(grid_search_knn1.score(X_test_count_vectorized, y_test)))
print("F1 score on training set: " + str(f1_score(y_train, grid_knn_count_pred_train)))
print("F1 score on testing set: " + str(f1_score(y_test, grid_knn_count_pred_test)))
print("Training set confusion matrix:\n ", confusion_matrix(y_train, grid_knn_count_pred_train))
print("Testing set confusion matrix:\n ", confusion_matrix(y_test, grid_knn_count_pred_test))
```

```
Best parameter for n_neighbors: 5
Best parameter for distance: distance
Best score with these values: 0.5513243821329381

SCORES USING BEST VALUES

Classification accuracy on training set: 0.9975221367926247
Classification accuracy on testing set: 0.5588106689986883
F1 score on training set: 0.9976794976794977
F1 score on testing set: 0.6022077666075301
Training set confusion matrix:
  [[12757    18]
 [   50 14618]]
Testing set confusion matrix:
  [[2057 2138]
 [1898 3055]]
```

TfidfVectorizer and KNeighborsClassifier

In [51]:

```
#init knn model and fit it on tfidf vectorized data
knn_tfidf = KNeighborsClassifier().fit(X=X_train_tfidf_vectorized, y=y_train)

#make predictions from fitted model
knn_tfidf_pred_train = knn_tfidf.predict(X_train_tfidf_vectorized)
knn_tfidf_pred_test = knn_tfidf.predict(X_test_tfidf_vectorized)
```

```
#evaluate performance
print("Classification accuracy on training set: " + str(knn_tfidf.score(X_train_tfidf_vectorized, y_train)))
print("Classification accuracy on testing set: " + str(knn_tfidf.score(X_test_tfidf_vectorized, y_test)))
print("Precision on training set: " + str(precision_score(y_train, knn_tfidf_pred_train)))
print("Precision on testing set: " + str(precision_score(y_test, knn_tfidf_pred_test)))
print("Recall on training set: " + str(recall_score(y_train, knn_tfidf_pred_train)))
print("F1 score on testing set: " + str(recall_score(y_test, knn_tfidf_pred_test)))
print("F1 score on training set: " + str(f1_score(y_train, knn_tfidf_pred_train)))
print("F1 score on testing set: " + str(f1_score(y_test, knn_tfidf_pred_test)))
print("Training set confusion matrix:\n ", confusion_matrix(y_train, knn_tfidf_pred_train))
print("Testing set confusion matrix:\n ", confusion_matrix(y_test, knn_tfidf_pred_test))
```

```
Classification accuracy on training set: 0.7396786065663375
Classification accuracy on testing set: 0.5874508089199825
Precision on training set: 0.7407834101382489
Precision on testing set: 0.6127797972068108
Recall on training set: 0.7890646304881375
F1 score on testing set: 0.6466787805370483
F1 score on training set: 0.764162155024429
F1 score on testing set: 0.6292730844793714
Training set confusion matrix:
  [[ 8725  4050]
 [ 3094 11574]]
Testing set confusion matrix:
  [[2171 2024]
 [1750 3203]]
```

Hyperparameter Tuning and Evaluation

In [52]:

```
#init grid of hyperparameters
grid6 = {"n_neighbors": [3, 5], "weights": ["uniform", "distance"]}

#fit it on tfidf vectorized data
grid_search_knn2 = GridSearchCV(KNeighborsClassifier(), grid6, cv=5)
grid_search_knn2.fit(X=X_train_tfidf_vectorized, y=y_train)
```

Out[52]:

```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': [3, 5],
                         'weights': ['uniform', 'distance']})
```

In [53]:

```
#make prediction on model w best hyperparameters
grid_knn_tfidf_pred_train = grid_search_knn2.predict(X_train_tfidf_vectorized)
grid_knn_tfidf_pred_test = grid_search_knn2.predict(X_test_tfidf_vectorized)

#evaluate performance
print("Best parameter for n_neighbors: " + str(grid_search_knn2.best_params_['n_neighbors']))
print("Best parameter for distance: " + str(grid_search_knn2.best_params_['weights']))
print("Best score with these values: " + str(grid_search_knn2.best_score_))
print("\nSCORES USING BEST VALUES\n")
print("Classification accuracy on training set: " + str(grid_search_knn2.score(X_train_tfidf_vectorized, y_train)))
print("Classification accuracy on testing set: " + str(grid_search_knn2.score(X_test_tfidf_vectorized, y_test)))
print("Precision on training set: " + str(precision_score(y_train, grid_knn_tfidf_pred_train)))
print("Precision on testing set: " + str(precision_score(y_test, grid_knn_tfidf_pred_test)))
print("Recall on training set: " + str(recall_score(y_train, grid_knn_tfidf_pred_train)))
print("Recall on testing set: " + str(recall_score(y_test, grid_knn_tfidf_pred_test)))
print("F1 score on training set: " + str(f1_score(y_train, grid_knn_tfidf_pred_train)))
print("F1 score on testing set: " + str(f1_score(y_test, grid_knn_tfidf_pred_test)))
print("Training set confusion matrix:\n ", confusion_matrix(y_train, grid_knn_tfidf_pred_train))
print("Testing set confusion matrix:\n ", confusion_matrix(y_test, grid_knn_tfidf_pred_test))
```

```
Best parameter for n_neighbors: 3
Best parameter for distance: uniform
Best score with these values: 0.56932567095495

SCORES USING BEST VALUES

Classification accuracy on training set: 0.7953940895674672
Classification accuracy on testing set: 0.5815478793178837
Precision on training set: 0.7942533966066437
Precision on testing set: 0.6091173617846751
Recall on training set: 0.8329697300245432
Recall on testing set: 0.633959216636382
F1 score on training set: 0.8131509766729893
F1 score on testing set: 0.6212900672734468
Training set confusion matrix:
  [[ 9610  3165]
 [ 2450 12218]]
Testing set confusion matrix:
  [[2180 2015]
 [1813 3140]]
```

Summary

Here is a summary of our findings on our final project. These tables encapsulate the accuracy and f1-score (which is a harmonic mean of acc/recall) of the model, as well as the train/test splits for each measurement.

In [54]:

```
#read in summary csv file (created ourselves)
summary_df = pd.read_csv("nlp_project_summary.csv")
summary_df
```

Out[54]:

	Model	Vectorizer	Set	Measurement	Score
0	Multinomial Naive Bayes	Count	Training	Accuracy	0.9214
1	Multinomial Naive Bayes	Count	Testing	Accuracy	0.8196
2	Multinomial Naive Bayes	Tf-idf	Training	Accuracy	0.9536
3	Multinomial Naive Bayes	Tf-idf	Testing	Accuracy	0.8031
4	Logistic Regression	Count	Training	Accuracy	0.9317
5	Logistic Regression	Count	Testing	Accuracy	0.8237
6	Logistic Regression	Tf-idf	Training	Accuracy	0.9014
7	Logistic Regression	Tf-idf	Testing	Accuracy	0.8237
8	Random Forest Classifier	Count	Training	Accuracy	0.9975
9	Random Forest Classifier	Count	Testing	Accuracy	0.7993
10	Random Forest Classifier	Tf-idf	Training	Accuracy	0.9975
11	Random Forest Classifier	Tf-idf	Testing	Accuracy	0.8022
12	Kneighbor Classifier	Count	Training	Accuracy	0.9975
13	Kneighbor Classifier	Count	Testing	Accuracy	0.5589
14	Kneighbor Classifier	Tf-idf	Training	Accuracy	0.7954
15	Kneighbor Classifier	Tf-idf	Testing	Accuracy	0.5815
16	Multinomial Naive Bayes	Count	Training	F1 Score	0.9257
17	Multinomial Naive Bayes	Count	Testing	F1 Score	0.8295
18	Multinomial Naive Bayes	Tf-idf	Training	F1 Score	0.9565
19	Multinomial Naive Bayes	Tf-idf	Testing	F1 Score	0.8167
20	Logistic Regression	Count	Training	F1 Score	0.9365
21	Logistic Regression	Count	Testing	F1 Score	0.8383
22	Logistic Regression	Tf-idf	Training	F1 Score	0.9089
23	Logistic Regression	Tf-idf	Testing	F1 Score	0.8392
24	Random Forest Classifier	Count	Training	F1 Score	0.9975
25	Random Forest Classifier	Count	Testing	F1 Score	0.7993
26	Random Forest Classifier	Tf-idf	Training	F1 Score	0.9976
27	Random Forest Classifier	Tf-idf	Testing	F1 Score	0.8212
28	Kneighbor Classifier	Count	Training	F1 Score	0.9977
29	Kneighbor Classifier	Count	Testing	F1 Score	0.6022
30	Kneighbor Classifier	Tf-idf	Training	F1 Score	0.8132
31	Kneighbor Classifier	Tf-idf	Testing	F1 Score	0.6212

In [55]:

```
grouped_summary_df = summary_df.set_index(["Model", "Vectorizer", "Set", "Measurement"]).sort_index()
grouped_summary_df
```

Out[55]:

				Score
Model	**Vectorizer**	**Set**	**Measurement**	
Kneighbor Classifier	**Count**	**Testing**	**Accuracy**	0.5589
			F1 Score	0.6022
		Training	**Accuracy**	0.9975

Model	Vectorizer	Set	Measurement	Score
			F1 Score	0.9977
	Tf-idf	Testing	Accuracy	0.5815
			F1 Score	0.6212
		Training	Accuracy	0.7954
			F1 Score	0.8132
Logistic Regression	Count	Testing	Accuracy	0.8237
			F1 Score	0.8383
		Training	Accuracy	0.9317
			F1 Score	0.9365
	Tf-idf	Testing	Accuracy	0.8237
			F1 Score	0.8392
		Training	Accuracy	0.9014
			F1 Score	0.9089
Multinomial Naive Bayes	Count	Testing	Accuracy	0.8196
			F1 Score	0.8295
		Training	Accuracy	0.9214
			F1 Score	0.9257
	Tf-idf	Testing	Accuracy	0.8031
			F1 Score	0.8167
		Training	Accuracy	0.9536
			F1 Score	0.9565
Random Forest Classifier	Count	Testing	Accuracy	0.7993
			F1 Score	0.7993
		Training	Accuracy	0.9975
			F1 Score	0.9975
	Tf-idf	Testing	Accuracy	0.8022
			F1 Score	0.8212
		Training	Accuracy	0.9975
			F1 Score	0.9976

In [ ]:

```
```