Data Exploration

The data appears to be relatively clean. No empty cells. Data types are consistent. Columns Index, Sentiment, Text...

The data will need to be processed to remove numbers and special characters as well as lemmatized and tokenized to used by classification models.

Data Preprocessing

Read Data

In [1]:
```
import pandas as pd

# Read data sets
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')

# Get 10% of the rows because my computer cannot handle the whole dataset
train = train.sample(frac=0.10, random_state=42)

print(train)

# # Counting number of 1s in Sentiment
# num_positive = (train['Sentiment'] == 1).sum()
# print(num_positive)
```

```
           Index  Sentiment                                              Text
781974    781974          0  i did not know that @PaulaAbdul had a step bro...
937737    937737          1  @sheila97 bamburi beach , travellers , ocean s...
907828    907828          1  @jesterjay SWINE FLU. Some family just came ba...
784628    784628          0            The Kids video seriously freaks me out 
662460    662460          0                                Back is hurting.  x
...          ...        ...                                                ...
933141    933141          1  @fii111 lets DO IT! move to NY at the end of t...
1007347  1007347          1  good morning! tis a beautiful day in sunny lon...
80444      80444          0                      Have to redo my whole itunes 
772632    772632          0  Finally got MSN IM to let me sign on for the f...
989497    989497          1  @Lifelists Poor you. I had root canal twice la...

[104858 rows x 3 columns]
```

Convert to lowercase

In [2]:
```
train['Text'] = train['Text'].apply(lambda x: x.lower())
test['Text'] = test['Text'].apply(lambda x: x.lower())
```

Remove numbers and special characters

In [3]:
```
import re

def remove_special_chars(text):
    # Replace all non-word characters with empty string
    text = re.sub(r"[^\w\s]", "", text)
    # Replace all digits with empty string
    text = re.sub(r"\d+", "", text)
    return text

train['Text'] = train['Text'].apply(remove_special_chars)
test['Text'] = test['Text'].apply(remove_special_chars)

# Check if numbers still exist
pattern = r'[0-9@#$]'

# Loop through columns in dataframe
for col in train.columns:
    # Check if column is of object type (i.e., contains text data)
    if train[col].dtype == 'O':
        # Use regex to check if column contains numbers or special characters
        if train[col].str.contains(pattern).any():
            # If the column contains numbers or special characters, display the
            print(f"Rows in {col} containing numbers or special characters:")
            print(train[train[col].str.contains(pattern)])
        else:
            print(f"No rows in {col} contain numbers or special characters.")
```

```
No rows in Text contain numbers or special characters.
```

Remove stop words

In [ ]:
```
# Will not be removing stop words to preserve semantics, but below is the co

# from nltk.corpus import stopwords

# stop_words = set(stopwords.words('english'))

# def remove_stopwords(text):
#     return " ".join([word for word in text.split() if word not in stop_words]

# train['Text'] = train['Text'].apply(remove_stopwords)
# test['Text'] = test['Text'].apply(remove_stopwords)
```

Stemming or Lemmatization

In [4]:
```
# from nltk.stem import PorterStemmer

# stemmer = PorterStemmer()

# def stemming(text):
#     return " ".join([stemmer.stem(word) for word in text.split()])

# train['Text'] = train['Text'].apply(stemming)

# train.head()
```

```
#I use lemmatization to preserve meaning
import nltk
from nltk.stem import WordNetLemmatizer

# function to apply lemmatization to text
def lemmatize_text(text):
    lemmatizer = WordNetLemmatizer()
    tokens = nltk.word_tokenize(text)
    lemmatized_text = ' '.join([lemmatizer.lemmatize(token) for token in tokens
    return lemmatized_text

# apply lemmatization to the 'text' column of the DataFrame
train['Text'] = train['Text'].apply(lemmatize_text)
test['Text'] = test['Text'].apply(lemmatize_text)
```

Tokenization

In [5]:
```
from nltk.tokenize import word_tokenize

train['Tokens'] = train['Text'].apply(word_tokenize)
test['Tokens'] = test['Text'].apply(word_tokenize)
```

Linguistic Feature Extraction

In [6]:
```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorize
from gensim.models import Word2Vec
from sklearn.model_selection import train_test_split
import numpy as np

# Set y_train and y_test
y_train = train['Sentiment']
y_test = test['Sentiment']

# Create a bag of words representation
count_vectorizer = CountVectorizer()
X_train_bow = count_vectorizer.fit_transform(train['Text'])
X_test_bow = count_vectorizer.transform(test['Text'])

# Create TF-IDF features
tfidf_vectorizer = TfidfVectorizer()
X_train_tfidf = tfidf_vectorizer.fit_transform(train['Text'])
X_test_tfidf = tfidf_vectorizer.transform(test['Text'])

# Create feature vectors word2vec
merged_df = pd.concat([train, test], axis=0)
w2v_model = Word2Vec(sentences=merged_df['Tokens'], vector_size=100, window=5,

X = []
for tokens in merged_df['Tokens']:
    vector = []
    for token in tokens:
        if token in w2v_model.wv:
            vector.append(w2v_model.wv[token])
```

```
    if vector:
        X.append(np.mean(vector, axis=0))
    else:
        X.append([0] * 100)

X = np.array(X, dtype=object)

# Create target vector
y = merged_df['Sentiment']

# Split data into train and test sets
X_train_w2v, X_test_w2v, y_train_w2v, y_test_w2v = train_test_split(X, y, test_
```

Sentiment Classification Model + Evaluation

Comparing all 3 features on Logistic Regression

In [7]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report


# train LR w/ bow
lr_bow = LogisticRegression(max_iter=1000)
lr_bow.fit(X_train_bow, y_train)

# train LR w/ tfidf
lr_tfidf = LogisticRegression(max_iter=1000)
lr_tfidf.fit(X_train_tfidf, y_train)

# train LR w/ w2v
lr_w2v = LogisticRegression(max_iter=1000)
lr_w2v.fit(X_train_w2v, y_train_w2v)


# evaluate the logistic regression classifier on the test data
y_pred_lr_bow = lr_bow.predict(X_test_bow)
lr_accuracy_bow = accuracy_score(y_test, y_pred_lr_bow)
report_bow = classification_report(y_test, y_pred_lr_bow)

y_pred_lr_tfidf = lr_tfidf.predict(X_test_tfidf)
lr_accuracy_tfidf = accuracy_score(y_test, y_pred_lr_tfidf)
report_tfidf = classification_report(y_test, y_pred_lr_tfidf)

y_pred_lr_w2v = lr_w2v.predict(X_test_w2v)
lr_accuracy_w2v = accuracy_score(y_test_w2v, y_pred_lr_w2v)
report_w2v = classification_report(y_test_w2v, y_pred_lr_w2v)


print('-----------------------------------------------------')
print("LR bag of words accuracy:", lr_accuracy_bow)
print(report_bow)
print('-----------------------------------------------------')
print("LR tfidf accuracy:", lr_accuracy_tfidf)
print(report_tfidf)
print('-----------------------------------------------------')
```

```
print("LR w2v accuracy:", lr_accuracy_w2v)
print(report_w2v)
print('-----------------------------------------------------')
```

```
-----------------------------------------------------
LR bag of words accuracy: 0.7214484679665738
              precision    recall  f1-score   support

           0       0.65      0.95      0.77       177
           1       0.91      0.50      0.65       182

    accuracy                           0.72       359
   macro avg       0.78      0.72      0.71       359
weighted avg       0.78      0.72      0.71       359

-----------------------------------------------------
LR tfidf accuracy: 0.7214484679665738
              precision    recall  f1-score   support

           0       0.65      0.95      0.77       177
           1       0.92      0.49      0.64       182

    accuracy                           0.72       359
   macro avg       0.78      0.72      0.71       359
weighted avg       0.78      0.72      0.71       359

-----------------------------------------------------
LR w2v accuracy: 0.808544003041247
              precision    recall  f1-score   support

           0       0.82      0.96      0.88     16112
           1       0.69      0.33      0.45      4932

    accuracy                           0.81     21044
   macro avg       0.76      0.64      0.67     21044
weighted avg       0.79      0.81      0.78     21044

-----------------------------------------------------
```

Comparing all 4 models using W2V

In [8]:

```
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier

from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline


# Train LR classifier
lr_classifier = LogisticRegression(max_iter=1000)
lr_classifier.fit(X_train_w2v, y_train_w2v)

# Train SVM classifier
svm_classifier = SVC()
svm_classifier.fit(X_train_w2v, y_train_w2v)

# Train Naive Bayes classifier
clf = Pipeline([('Normalizing',MinMaxScaler()),('MultinomialNB',MultinomialNB()
```

```
clf.fit(X_train_w2v, y_train_w2v)

# Train Random Forest classifier
rf_classifier = RandomForestClassifier()
rf_classifier.fit(X_train_w2v, y_train_w2v)


# Make predictions and evaluate model performance
y_pred_lr = lr_classifier.predict(X_test_w2v)
lr_accuracy = accuracy_score(y_test_w2v, y_pred_lr)
report_lr = classification_report(y_test_w2v, y_pred_lr, zero_division=0)

y_pred_svm = svm_classifier.predict(X_test_w2v)
svm_accuracy = accuracy_score(y_test_w2v, y_pred_svm)
report_svm = classification_report(y_test_w2v, y_pred_svm, zero_division=0)

y_pred_bayes = clf.predict(X_test_w2v)
bayes_accuracy = accuracy_score(list(y_test_w2v), y_pred_bayes)
report_bayes = classification_report(y_test_w2v, y_pred_bayes, zero_division=0)

y_pred_rf = rf_classifier.predict(X_test_w2v)
rf_accuracy = accuracy_score(y_test_w2v, y_pred_rf)
report_rf = classification_report(y_test_w2v, y_pred_rf, zero_division=0)


# Print results
print('------------------------------------------------------')
print("LR accuracy:", lr_accuracy)
print(report_lr)
print('------------------------------------------------------')
print("SVM accuracy:", svm_accuracy)
print(report_svm)
print('------------------------------------------------------')
print('Naive Bayes accuracy:', bayes_accuracy)
print(report_bayes)
print('------------------------------------------------------')
print("Random Forest accuracy:", rf_accuracy)
print(report_rf)
print('------------------------------------------------------')
```

```
-----------------------------------------------------
LR accuracy: 0.808544003041247
              precision    recall  f1-score   support

           0       0.82      0.96      0.88     16112
           1       0.69      0.33      0.45      4932

    accuracy                           0.81     21044
   macro avg       0.76      0.64      0.67     21044
weighted avg       0.79      0.81      0.78     21044

-----------------------------------------------------
SVM accuracy: 0.808734080973199
              precision    recall  f1-score   support

           0       0.81      0.98      0.89     16112
           1       0.77      0.26      0.39      4932

    accuracy                           0.81     21044
   macro avg       0.79      0.62      0.64     21044
weighted avg       0.80      0.81      0.77     21044

-----------------------------------------------------
Naive Bayes accuracy: 0.7656339099030602
              precision    recall  f1-score   support

           0       0.77      1.00      0.87     16112
           1       0.00      0.00      0.00      4932

    accuracy                           0.77     21044
   macro avg       0.38      0.50      0.43     21044
weighted avg       0.59      0.77      0.66     21044

-----------------------------------------------------
Random Forest accuracy: 0.7999429766204144
              precision    recall  f1-score   support

           0       0.80      0.98      0.88     16112
           1       0.74      0.23      0.35      4932

    accuracy                           0.80     21044
   macro avg       0.77      0.60      0.61     21044
weighted avg       0.79      0.80      0.76     21044

-----------------------------------------------------
```

In [ ]: