In [ ]:

```
import pandas as pd
df_names = ['Twitter_ID', 'Subject_Matter', 'Sentiment', 'Text']
df = pd.read_csv('twitter_training.csv', header=0, encoding='UTF-8', names=df_names)
# (74682, 4) dimensions
```

For this assignment I chose to classify Kaggle stored Twitter tweets as either positive, neutral, or negative with Machine Learning algorithms. To appropriately set up the dataset table for computation, I removed any examples where the features or targets lack relevant information. For example, the dataset contains ‘Irrelevant’ as a target Sentiment value which is not important to our learning. Additionally, some of the tweets are displayed as NaN in the dataset as well. After cleaning up the data and converting the sentiment categorical values to numerical ones, I began to run the various algorithms.

In [ ]:

```
# Removal of Columns
df = df.drop(columns=['Twitter_ID', 'Subject_Matter'], axis=1)
```

In [ ]:

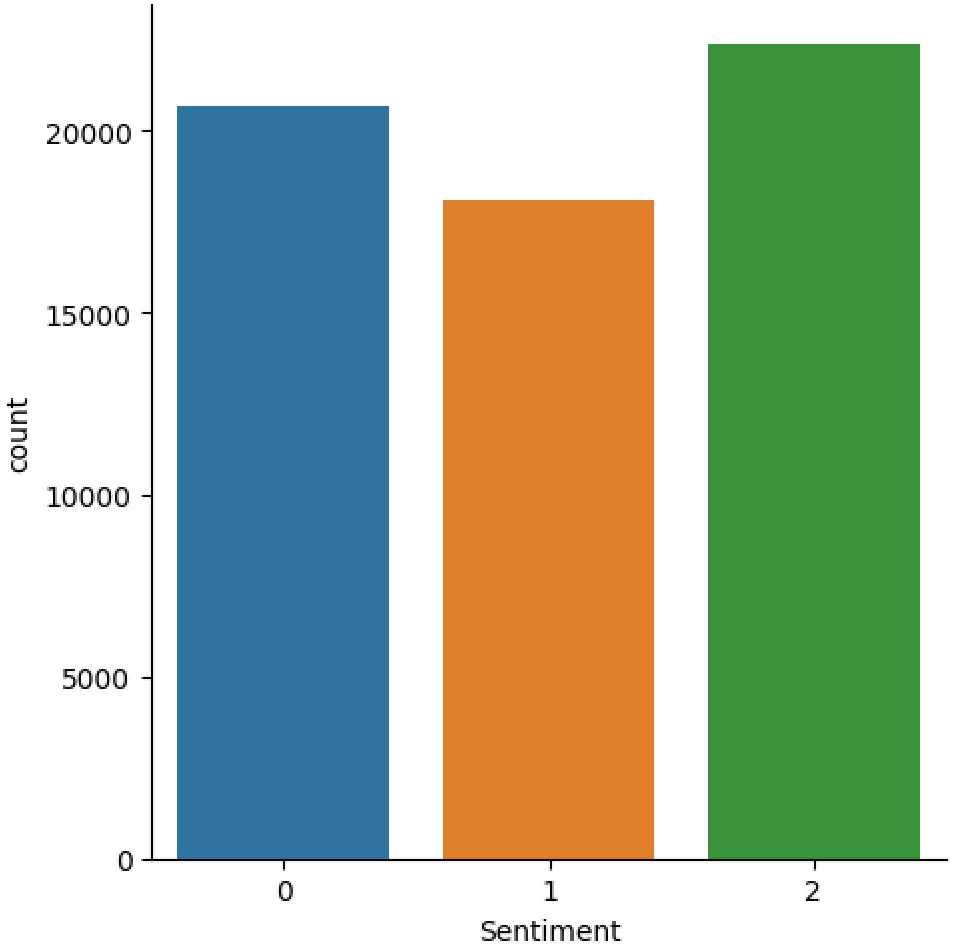
```
# Removal where Sentiment values are 'Irrelevant'
df = df[df.Sentiment != 'Irrelevant']
# (61692, 2) dimensions
```

In [ ]:

```
# Showing distribution of target classes
import seaborn as sb
sb.catplot(x='Sentiment', kind='count', data=df)
```

Out[ ]:

```
<seaborn.axisgrid.FacetGrid at 0x7f1b6c96b3a0>
```



In [ ]:
```
# Change Sentiment Categorical Values to Numerical Ones
df['Sentiment'].replace(['Positive', 'Neutral', 'Negative'], [0, 1, 2], inplace=True)
```

In [ ]:
```
# Check to use if it has changed
df['Sentiment'][:100]
```

Out[ ]:
```
0     0
1     0
2     0
3     0
4     0
     ..
95    2
96    2
97    2
98    2
99    2
Name: Sentiment, Length: 100, dtype: int64
```

In [ ]:
```
# Removal rows with NaN values
df = df.dropna()
df.shape
```

Out[ ]:
```
(61120, 2)
```

In [ ]:
```
X = df.Text
y = df.Sentiment
X.head()
```

Out[ ]:

```
0    I am coming to the borders and I will kill you...
1    im getting on borderlands and i will kill you ...
2    im coming on borderlands and i will murder you...
3    im getting on borderlands 2 and i will murder ...
4    im getting into borderlands and i can murder y...
Name: Text, dtype: object
```

In [ ]:

```
y[:30]
```

Out[ ]:

```
0     0
1     0
2     0
3     0
4     0
5     0
6     0
7     0
8     0
9     0
10    0
11    1
12    1
13    1
14    1
15    1
16    1
17    0
18    0
19    0
20    0
21    0
22    0
23    2
24    2
25    2
26    2
27    2
28    2
29    0
Name: Sentiment, dtype: int64
```

In [ ]:

```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

Out[ ]:

```
True
```

In [ ]:

```
# Apply TfidfVectorizer
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer

stopwords = set(stopwords.words('english'))
vectorizer = TfidfVectorizer(stop_words=list(stopwords))
```

In [ ]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, train_size=0.
```

In [ ]:

```
# Transform it
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)
```

In [ ]:

```
# Apply Naive Bayes
from sklearn.naive_bayes import MultinomialNB

naive_bayes = MultinomialNB()
naive_bayes.fit(X_train, y_train)
```

Out[ ]:

```
▾ MultinomialNB
MultinomialNB()
```

In [ ]:

```
pred = naive_bayes.predict(X_test)
```

In [ ]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.83      0.81      4063
           1       0.89      0.65      0.75      3602
           2       0.76      0.90      0.82      4559

    accuracy                           0.80     12224
   macro avg       0.82      0.79      0.80     12224
weighted avg       0.81      0.80      0.80     12224
```

In [ ]:

```
# Logistic Regression
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(C=2.5, n_jobs=4, solver='lbfgs', random_state=17, verbose=1)
clf.fit(X_train, y_train)
```

```
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done   1 out of   1 | elapsed:    4.6s finished
```

Out[ ]:

```
▾                  LogisticRegression
LogisticRegression(C=2.5, n_jobs=4, random_state=17, verbose=1)
```

In [ ]:

```
Log_Reg_pred = clf.predict(X_test)
```

In [ ]:

```
print(classification_report(y_test, Log_Reg_pred))
```

```
              precision    recall  f1-score   support

           0       0.83      0.90      0.86      4063
           1       0.90      0.80      0.85      3602
           2       0.88      0.89      0.89      4559

    accuracy                           0.87     12224
   macro avg       0.87      0.86      0.87     12224
weighted avg       0.87      0.87      0.87     12224
```

```
In [ ]: # Applying Neural Network
        from sklearn.neural_network import MLPClassifier
        classifier = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(15,2), rand
        classifier.fit(X_train, y_train)
```

```
Out[ ]: ▾                          MLPClassifier
        MLPClassifier(alpha=1e-05, hidden_layer_sizes=(15, 2), max_iter=20000,
                      random_state=1, solver='lbfgs')
```

```
In [ ]: NN_pred = classifier.predict(X_test)
```

```
In [ ]: print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.77      0.86      0.81      4063
           1       0.83      0.86      0.85      3602
           2       0.79      0.69      0.74      4559

    accuracy                           0.80     12224
   macro avg       0.80      0.80      0.80     12224
weighted avg       0.80      0.80      0.79     12224
```

Both the Logistic Regression and Naïve Bayes algorithm was able to complete relatively quickly. Naïve Bayes, according to the classification report, reaches approximately 80% for precision, recall, and f1-score. However, Logistic Regression had a higher approximate precision, recall, and f1-score with a number of 87%. The Neural Network algorithm was constantly something I had a lot of issues making it work. After most of my runs with the algorithm, it would spew out the warning that it had reached its max iterations before converging. Therefore, I started to change the max_iteration variable from a measly 500 eventually up to 20,000. Each fresh run increasing in time exponentially. Luckily the algorithm was able to converge at 20,000 or else I do not believe I would know what else to do. Funnily enough the Neural Network performed about as equal to the Naives Bayes with an approximate score of 80% all around. For the time and results for this dataset, I find that it would be best to run a Logistic Regression algorithm for future classification problems of larger calibers.