```
##GOAL - TO PERFORM SENTIMENT ANALYSIS ON AMAZON FOOD REVIEW


#Load Basic Libraries


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline


df = pd.read_csv('/content/food_review.csv')


#Explore the Data


df.head()
```

	Unnamed: 0	Text	Score
0	0	I bought these from a large chain pet store. a...	1
1	1	This soup is incredibly good! But honestly, I...	5
2	2	Our family loves these tasty and healthy sesam...	5
3	3	The local auto shop offers this free to it cus...	4
4	4	I brought 2 bottles. One I carry in my pocket...	5

```
df.Text.head()
```

```
0    I bought these from a large chain pet store. a...
1    This soup is incredibly good!  But honestly, I...
2    Our family loves these tasty and healthy sesam...
3    The local auto shop offers this free to it cus...
4    I brought 2 bottles.  One I carry in my pocket...
Name: Text, dtype: object
```

```
df.shape
```

```
(40500, 3)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40500 entries, 0 to 40499
```

```
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  40500 non-null  int64
 1   Text        40500 non-null  object
 2   Score       40500 non-null  int64
dtypes: int64(2), object(1)
memory usage: 949.3+ KB
```

```
df.Text.head()
```

```
0    I bought these from a large chain pet store. a...
1    This soup is incredibly good!  But honestly, I...
2    Our family loves these tasty and healthy sesam...
3    The local auto shop offers this free to it cus...
4    I brought 2 bottles.  One I carry in my pocket...
Name: Text, dtype: object
```

```
#Encoding score to Positive or negative based on value of each sample
```

```
scores = df['Score']
df['Score'] = df['Score'].apply(lambda x : 'pos' if x > 3 else 'neg')
```

```
scores.mean()
```

```
3.0018765432098764
```

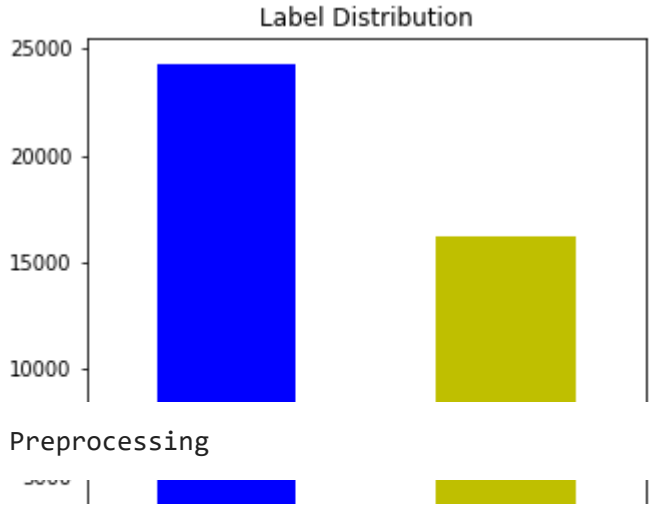
```
#Distribution of labels in the dataset
```

```
df.groupby('Score')['Text'].count()
```

```
Score
neg    24277
pos    16223
Name: Text, dtype: int64
```

```
df.groupby('Score')['Text'].count().plot(kind='bar',color=['b','y'],title='Label Distribution
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6e528f0810>
```




```
#Text Preprocessing
```

```
from nltk.corpus import stopwords
from textblob import TextBlob
from textblob import Word
# Lower casing and removing punctuations

df['Text'] = df['Text'].apply(lambda x: " ".join(x.lower() for
x in x.split()))


df['Text'] = df['Text'].str.replace('[^\w\s]', "")
df.Text.head(5)
```

```
0    i bought these from a large chain pet store af...
1    this soup is incredibly good but honestly i wa...
2    our family loves these tasty and healthy sesam...
3    the local auto shop offers this free to it cus...
4    i brought 2 bottles one i carry in my pocket a...
Name: Text, dtype: object
```

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

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
```

```
#remove the stopwords
stop = stopwords.words('english')
df['Text'] = df['Text'].apply(lambda x: " ".join(x for x in
x.split() if x not in stop))
df.Text.head()
```

```
0    bought large chain pet store reading review ch...
1    soup incredibly good honestly looking better d...
2    family love tasty healthy sesame honey almond ...
3    local auto shop offer free customer ive tried ...
4    brought 2 bottle one carry pocket home fell lo...
Name: Text, dtype: object
```

```
#Lemmatization
df['Text'] = df['Text'].apply(lambda x: " ".join([Word(word).
lemmatize() for word in x.split()]))
df.Text.head()
```

```
0    bought large chain pet store reading review ch...
1    soup incredibly good honestly looking better d...
2    family love tasty healthy sesame honey almond ...
3    local auto shop offer free customer ive tried ...
4    brought 2 bottle one carry pocket home fell lo...
Name: Text, dtype: object
```

```
from wordcloud import WordCloud
from wordcloud import STOPWORDS
```

```
# Create a new data frame "reviews" to perform exploratory data analysis upon that
reviews = df
# Dropping null values
reviews.dropna(inplace=True)
```

```
score_1 = reviews[reviews['Score'] == 1]
score_2 = reviews[reviews['Score'] == 2]
score_3 = reviews[reviews['Score'] == 3]
score_4 = reviews[reviews['Score'] == 4]
score_5 = reviews[reviews['Score'] == 5]
```

```
reviews_sample = pd.concat([score_1,score_2,score_3,score_4,score_5],axis=0)
reviews_sample.reset_index(drop=True,inplace=True)
```

```
#Wordcloud function's input needs to be a single string of text.
# concatenating all Summaries into a single string.
# similarly you can build for Text column
reviews_str = reviews_sample.Text.str.cat()
wordcloud = WordCloud(background_color='Black').generate(reviews_str)
plt.figure(figsize=(10,10))
plt.imshow(wordcloud,interpolation='bilinear')
plt.axis("off")
```

```
plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-38-ff0fc11bb6a6> in <module>()
      3 # similarly you can build for Text column
      4 reviews_str = reviews_sample.Text.str.cat()
----> 5 wordcloud = WordCloud(background_color='Black').generate(reviews_str)
      6 plt.figure(figsize=(10,10))
      7 plt.imshow(wordcloud,interpolation='bilinear')

                                 2 frames

/usr/local/lib/python3.7/dist-packages/wordcloud/wordcloud.py in
generate_from_frequencies(self, frequencies, max_font_size)
    381         if len(frequencies) <= 0:
    382             raise ValueError("We need at least 1 word to plot a word cloud, "
--> 383                              "got %d." % len(frequencies))
    384         frequencies = frequencies[:self.max_words]
    385

ValueError: We need at least 1 word to plot a word cloud, got 0.
```

SEARCH STACK OVERFLOW

```
# Now let's split the data into Negative (Score is 1 or 2) and Positive (4 or #5) Reviews.
negative_reviews = reviews_sample[reviews_sample['Score'].isin([1,2]) ]
positive_reviews = reviews_sample[reviews_sample['Score'].isin([4,5]) ]
# Transform to single string
negative_reviews_str = negative_reviews.Text.str.cat()
positive_reviews_str = positive_reviews.Text.str.cat()


wordcloud_negative = WordCloud(background_color='black').generate(negative_reviews_str)
wordcloud_positive = WordCloud(background_color='black').generate(positive_reviews_str)
# Plot
fig = plt.figure(figsize=(10,10))
ax1 = fig.add_subplot(211)
ax1.imshow(wordcloud_negative,interpolation='bilinear')
ax1.axis("off")
ax1.set_title('Reviews with Negative Scores',fontsize=20)
```

```
Text(0.5, 1.0, 'Reviews with Negative Scores')
```


```
fig = plt.figure(figsize=(10,10))
ax2 = fig.add_subplot(212)
ax2.imshow(wordcloud_positive,interpolation='bilinear')
ax2.axis("off")
ax2.set_title('Reviews with Positive Scores',fontsize=20)
plt.show()
```


```
X = df["Text"]
y = df["Score"]
```

```
from sklearn.preprocessing import LabelEncoder
```

```
le = LabelEncoder()
```

```
y = le.fit_transform(y)
```

```
from sklearn.model_selection import train_test_split

from sklearn.metrics import classification_report

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
```

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=1)
```

```
# vectorization
```

```
# CountVectorizer
cv = CountVectorizer(stop_words="english")


X_train_cv = cv.fit_transform(X_train)
X_test_cv = cv.transform(X_test)


from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(X_train_cv, y_train)
y_pred = dt.predict(X_test_cv)
print(classification_report(y_test,y_pred))
```

```
                  precision    recall  f1-score   support

               0       0.76      0.77      0.77      7327
               1       0.65      0.64      0.64      4823

        accuracy                           0.72     12150
       macro avg       0.71      0.70      0.70     12150
    weighted avg       0.72      0.72      0.72     12150
```

```
# Tfidf vectorization
tfidf = TfidfVectorizer(stop_words="english")


X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)


dt = DecisionTreeClassifier()
dt.fit(X_train_tfidf, y_train)
y_pred = dt.predict(X_test_tfidf)
print(classification_report(y_test,y_pred))
```

```
                  precision    recall  f1-score   support

               0       0.76      0.75      0.75      7327
               1       0.62      0.63      0.63      4823

        accuracy                           0.70     12150
       macro avg       0.69      0.69      0.69     12150
    weighted avg       0.70      0.70      0.70     12150
```

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences


tokenizer = Tokenizer(oov_token="<oov>")
tokenizer.fit_on_texts(X_train)
```

```
vocab_len = len(tokenizer.index_word)
vocab_len
```

```
44944
```

```
train_sequences = tokenizer.texts_to_sequences(X_train)
```

```
# padding
doc_length = []
for doc in train_sequences:
  doc_length.append(len(doc))
max(doc_length)
```

```
969
```

```
import numpy as np
np.quantile(doc_length, 0.95)
```

```
122.0
```

```
max_len = 33
train_padded = pad_sequences(train_sequences, maxlen=max_len)
```

```
test_sequences = tokenizer.texts_to_sequences(X_test)
test_padded = pad_sequences(test_sequences, maxlen=max_len)
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding, Flatten
```

```
model = Sequential()
model.add(Embedding(vocab_len+1,10,input_length=max_len,mask_zero=True))
model.add(Flatten())
model.add(Dense(16, activation="tanh"))
model.add(Dense(1,activation="sigmoid"))
```

```
model.summary()
model.compile(loss="binary_crossentropy", optimizer="adam")
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding (Embedding)        (None, 33, 10)            449450    
_________________________________________________________________
```

```
flatten (Flatten)            (None, 330)               0
_________________________________________________________________
dense (Dense)                (None, 16)                5296
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 17
=================================================================
Total params: 454,763
Trainable params: 454,763
Non-trainable params: 0
_________________________________________________________________
```

```
y_pred = model.predict(test_padded)
y_pred = np.where(y_pred >= 0.5,1,0)
```

```
print(classification_report(y_test,y_pred))
```

```
                  precision    recall  f1-score   support

               0       0.60      0.61      0.60      7327
               1       0.39      0.38      0.38      4823

        accuracy                           0.52     12150
       macro avg       0.49      0.49      0.49     12150
    weighted avg       0.51      0.52      0.52     12150
```

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Dense,Embedding, Flatten, SimpleRNN, Bidirectional, LSTM,
from tensorflow.keras.models import Sequential
```

```
model3 = Sequential()
model3.add(Embedding(vocab_len, 10, input_length=max_len, mask_zero=True))
model3.add(Bidirectional(SimpleRNN(32, activation="tanh", return_sequences=True)))
model3.add(Bidirectional(SimpleRNN(32, activation="tanh")))
model3.add(Dense(16, activation="tanh"))
model3.add(Dense(1,activation="sigmoid"))
```

```
model.summary()
model2.compile(loss="binary_crossentropy", optimizer="adam")
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 33, 10)            449450
_________________________________________________________________
flatten (Flatten)            (None, 330)               0
_________________________________________________________________
dense (Dense)                (None, 16)                5296
```

```
dense_1 (Dense)              (None, 1)                 17
=================================================================
Total params: 454,763
Trainable params: 454,763
Non-trainable params: 0
_________________________________________________________________
```

```
y_pred = model3.predict(test_padded)
y_pred = np.where(y_pred >= 0.5, 1, 0)
print(classification_report(y_test,y_pred))
```

```
                  precision    recall  f1-score   support

               0       0.60      0.48      0.53      7327
               1       0.39      0.51      0.44      4823

        accuracy                           0.49     12150
       macro avg       0.49      0.49      0.49     12150
    weighted avg       0.52      0.49      0.50     12150
```

```
# LSTM
model4 = Sequential()
model4.add(Embedding(vocab_len, 10, input_length=max_len, mask_zero=True))
model4.add(LSTM(32, activation="tanh", return_sequences=True))
model4.add(LSTM(32, activation="tanh"))
model4.add(Dense(16, activation="tanh"))
model4.add(Dense(1,activation="sigmoid"))
```

```
model.summary()
model2.compile(loss="binary_crossentropy", optimizer="adam")
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 33, 10)            449450
_________________________________________________________________
flatten (Flatten)            (None, 330)               0
_________________________________________________________________
dense (Dense)                (None, 16)                5296
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 17
=================================================================
Total params: 454,763
Trainable params: 454,763
Non-trainable params: 0
_________________________________________________________________
```

```
y_pred = model4.predict(test_padded)
y_pred = np.where(y_pred >= 0.5, 1, 0)
```

```
print(classification_report(y_test,y_pred))
```

```
                  precision    recall  f1-score   support

               0       0.62      0.52      0.57      7327
               1       0.41      0.51      0.46      4823

        accuracy                           0.52     12150
       macro avg       0.52      0.52      0.51     12150
    weighted avg       0.54      0.52      0.52     12150
```

```
model5 = Sequential()
model5.add(Embedding(vocab_len, 10, input_length=max_len, mask_zero=True))
model5.add(Bidirectional(LSTM(32, activation="tanh", return_sequences=True)))
model5.add(Bidirectional(LSTM(32, activation="tanh")))
model5.add(Dense(16, activation="tanh"))
model5.add(Dense(1,activation="sigmoid"))
```

```
model.summary()
model5.compile(loss="binary_crossentropy", optimizer="adam")
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 33, 10)            449450
_________________________________________________________________
flatten (Flatten)            (None, 330)               0
_________________________________________________________________
dense (Dense)                (None, 16)                5296
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 17
=================================================================
Total params: 454,763
Trainable params: 454,763
Non-trainable params: 0
_________________________________________________________________
```

```
y_pred = model5.predict(test_padded)
y_pred = np.where(y_pred >= 0.5, 1, 0)
print(classification_report(y_test,y_pred))
```

```
                  precision    recall  f1-score   support

               0       0.60      0.52      0.56      7327
               1       0.39      0.47      0.43      4823

        accuracy                           0.50     12150
       macro avg       0.50      0.50      0.49     12150
    weighted avg       0.52      0.50      0.51     12150
```

```
# NN

model = Sequential()
model.add(Embedding(vocab_len+1,10,input_length=max_len,mask_zero=True))
model.add(Flatten())
model.add(Dense(16, activation="tanh"))
model.add(Dense(1,activation="sigmoid"))


model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_6 (Embedding)      (None, 33, 10)            449450    
_________________________________________________________________
flatten_1 (Flatten)          (None, 330)               0         
_________________________________________________________________
dense_10 (Dense)             (None, 16)                5296      
_________________________________________________________________
dense_11 (Dense)             (None, 1)                 17        
=================================================================
Total params: 454,763
Trainable params: 454,763
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(loss="binary_crossentropy", optimizer="adam")


model.fit(train_padded, y_train,epochs=20, batch_size=50)
```

```
Epoch 1/20
567/567 [==============================] - 4s 6ms/step - loss: 0.5241
Epoch 2/20
567/567 [==============================] - 4s 6ms/step - loss: 0.3172
Epoch 3/20
567/567 [==============================] - 4s 6ms/step - loss: 0.1678
Epoch 4/20
567/567 [==============================] - 4s 6ms/step - loss: 0.0636
Epoch 5/20
567/567 [==============================] - 4s 6ms/step - loss: 0.0212
Epoch 6/20
567/567 [==============================] - 4s 6ms/step - loss: 0.0080
Epoch 7/20
567/567 [==============================] - 4s 6ms/step - loss: 0.0043
Epoch 8/20
567/567 [==============================] - 4s 6ms/step - loss: 0.0027
Epoch 9/20
567/567 [==============================] - 4s 6ms/step - loss: 0.0027
Epoch 10/20
567/567 [==============================] - 4s 6ms/step - loss: 0.0019
Epoch 11/20
567/567 [==============================] - 4s 6ms/step - loss: 0.0019
```

```
Epoch 12/20
567/567 [==============================] - 4s 7ms/step - loss: 0.0029
Epoch 13/20
567/567 [==============================] - 4s 6ms/step - loss: 0.0022
Epoch 14/20
567/567 [==============================] - 4s 6ms/step - loss: 0.0024
Epoch 15/20
567/567 [==============================] - 4s 6ms/step - loss: 0.0022
Epoch 16/20
567/567 [==============================] - 4s 7ms/step - loss: 0.0015
Epoch 17/20
567/567 [==============================] - 4s 7ms/step - loss: 0.0017
Epoch 18/20
567/567 [==============================] - 4s 6ms/step - loss: 8.9746e-04
Epoch 19/20
567/567 [==============================] - 4s 6ms/step - loss: 0.0011
Epoch 20/20
567/567 [==============================] - 4s 6ms/step - loss: 0.0014
<tensorflow.python.keras.callbacks.History at 0x7f826ddbc650>
```

```
y_pred = model.predict(test_padded)
```

```
y_pred = np.where(y_pred >= 0.5,1,0)
```

```
print(classification_report(y_test,y_pred))
```

```
                  precision    recall  f1-score   support

               0       0.81      0.82      0.82      7327
               1       0.72      0.71      0.72      4823

        accuracy                           0.78     12150
       macro avg       0.77      0.77      0.77     12150
    weighted avg       0.78      0.78      0.78     12150
```

```
#Conclusion
We have used models like Decision Tree, Neural Network, LSTM . Among from them
 we can see NN is giving good accurcy with 78.
```