D213 Sentiment Analysis Using Neural Networks Performance Assessment, Task 2

Alexa R. Fisher

Western Governors University

Degree: M.S. Data Analytics

Table of Contents

Part I: Research Question

A1. Proposal of Question

The research question for this thesis was, “Can neural networks and NLP techniques be used to accurately predict a consumer’s sentiment based on written historical reviews?”. This analysis was completed using NLP techniques and neural networks upon the combined Amazon, Yelp, and IMDB reviews from the UC Irvine Machine Learning Repository (Kotzias, 2015).

A2. Defined Goal

The main goal of this analysis was to utilize historical written reviews within the complied Amazon, Yelp, and IMDB data files to predict consumer sentiment. The data sets were concatenated to provide a combined dataset to be utilized for analysis. The combined data set was cleaned and split into training, testing, and validation sets. The training set represented 70% of the original data; leaving the test and validation set with 15% each. The expected outcome was to accurately classify sentiment either positive or negative based on key words used within written review. Accuracy was determined with an accuracy score greater than 80%.

A3. Neural Network Type

The neural network type utilized within this analysis was Recurrent Neural Network (RNN). This neural network was commonly used in handling problems related to sequential text classification data (IBM, n.d.). This type of network was able to maintain a recurrent connection with itself. This allowed for the hidden layers of the model to loop through past sequential data input to the next text sequence by remembering past versions of itself to aid in the reduction of the data (Yiu, 2019). The final output ended with a single output producing a text classification of positive(1) or negative(0).

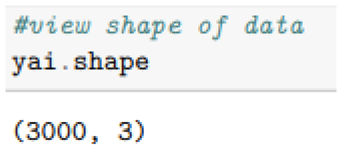
Part II: Data Preparation

B1. Summary of Exploratory Data Analysis

There were various steps in the performance of exploratory data analysis on the combined dataset of UCI Labelled Sentences of Amazon, Yelp, and IMDB. These phases included the following:

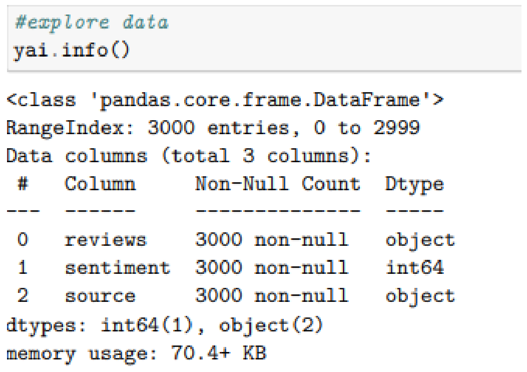
- Presence of Unusual Characters
- Vocabulary Size
- Word Embedding Length
- Statistical Justification for Maximum Sequence Length

The data set's shape and basic information were compiled before further exploration. These were completed with the usage of the shape method and the .info() function. The data set has a total of 3,000 entries and 3 columns. The data types included objects and integers with only the "sentiment" column being integers.



```
#view shape of data
yai.shape
```

```
(3000, 3)
```



```
#explore data
yai.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 3 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   reviews    3000 non-null   object
 1   sentiment  3000 non-null   int64 
 2   source     3000 non-null   object
dtypes: int64(1), object(2)
memory usage: 70.4+ KB
```

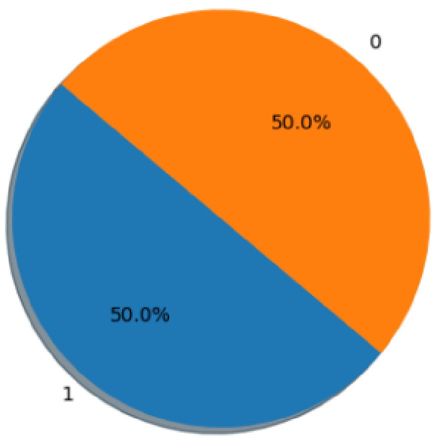
The dataset did not include any null values. This was shown with the use of the .isnull() function with the .sum() to provide a total count of null values.

```
#check for nulls
yai.isnull().sum()
```

```
reviews      0
sentiment    0
source       0
dtype: int64
```

There was an equal amount of positive and negative reviews found in the dataset. This was shown in the below pie chart. There was a total of 1,500 reviews with negative (0) sentiment and 1,500 reviews with positive (1) sentiment.

```
#sentiment counts,explorating the split between the negative and positive
sentiment = yai['sentiment'].value_counts()
plt.pie(sentiment, labels= sentiment.index,
        autopct='%1.1f%%', shadow=True, startangle=140)
plt.show()
#printing exact counts
print("Number of Negative reviews: {}".format(yai[yai['sentiment']==0].
  ↪count()[0]))
print("Number of Positive reviews: {}".format(yai[yai['sentiment']==1].
  ↪count()[0]))
```



```
Number of Negative reviews: 1500
Number of Positive reviews: 1500
```

Additional exploration included various statistics and visualizations. Statistics included the various counts of words, characters, the average word length, and the minimum/ maximum character length before any data cleaning steps. The frequency of the character counts was plotted within a histogram to provide the distribution of the data, which was noted as right-skewed. The majority of the character counts were under 100 characters.

```
# explore the various counts of words, characters, avg word length
yai['characters'] = yai['reviews'].apply(len)
yai['words_count'] = yai['reviews'].apply(lambda x: len(x.split()))
yai['avg_wordlen'] = yai['characters']/ yai['words_count']
yai.head()
```

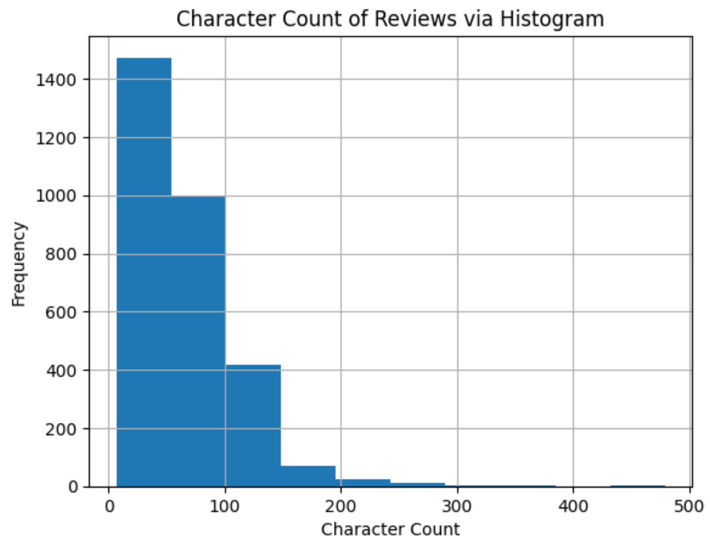
	reviews	sentiment	source	characters	words_count	avg_wordlen
0	Wow... Loved this place.	1	yelp	24	4	6.000000
1	Crust is not good.	0	yelp	18	4	4.500000
2	Not tasty and the texture was just nasty.	0	yelp	41	8	5.125000
3	Stopped by during the late May bank holiday off Rick Steve recommendation and loved it.	1	yelp	87	15	5.800000
4	The selection on the menu was great and so were the prices.	1	yelp	59	12	4.916667

```
#minimum/maximum review character length original data
print("Min. Length: " , min(yai['reviews'].str.len()))
print("Max. Length: " , max(yai['reviews'].str.len()))
```

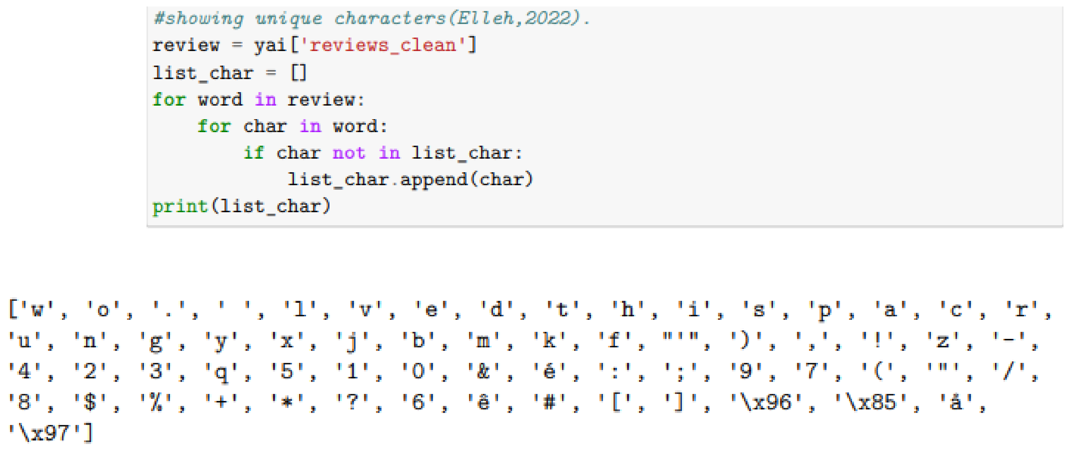
```
Min. Length:  7
Max. Length:  479
```

```
# Create a histogram of reviews by total char lengths
char_hist = yai['reviews'].str.len().hist()
char_hist.set_title('Character Count of Reviews via Histogram')
char_hist.set_xlabel('Character Count')
char_hist.set_ylabel('Frequency')
```

```
Text(0, 0.5, 'Frequency')
```



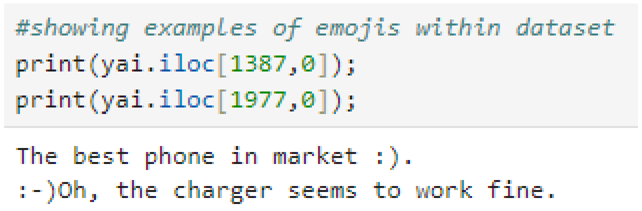
The presence of unusual characters within the data set encompassed the following list after removing uppercase letters within the data: 'w', 'o', '.', ' ', 'l', 'v', 'e', 'd', 't', 'h', 'i', 's', 'p', 'a', 'c', 'r', 'u', 'n', 'g', 'y', 'x', 'j', 'b', 'm', 'k', 'f', "'", ')', ',', '!', 'z', '-', '4', '2', '3', 'q', '5', '1', '0', '&', 'é', ':', ';', '9', '7', '(', '"', '/', '8', '$', '%', '+', '*', '?', '6', 'ê', '#', '[', ']', '\x96', '\x85', 'å', '\x97' .



```
#showing unique characters(Elleh,2022).
review = yai['reviews_clean']
list_char = []
for word in review:
    for char in word:
        if char not in list_char:
            list_char.append(char)
print(list_char)
```

```
['w', 'o', '.', ' ', 'l', 'v', 'e', 'd', 't', 'h', 'i', 's', 'p', 'a', 'c', 'r',
'u', 'n', 'g', 'y', 'x', 'j', 'b', 'm', 'k', 'f', "'", ')', ',', '!', 'z', '-',
'4', '2', '3', 'q', '5', '1', '0', '&', 'é', ':', ';', '9', '7', '(', '"', '/',
'8', '$', '%', '+', '*', '?', '6', 'ê', '#', '[', ']', '\x96', '\x85', 'å',
'\x97']
```

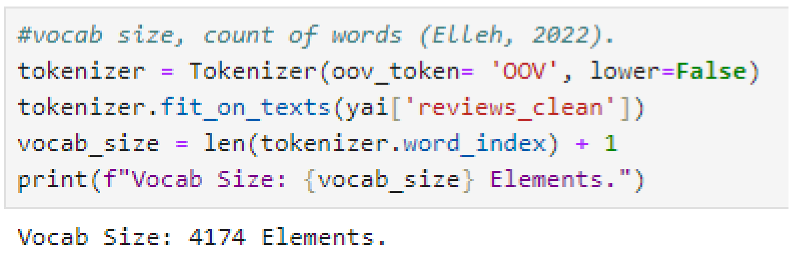
There were also Emojis found within the data set made out of characters and punctuation such as ":) "and ": -)". Examples were provided below.



```
#showing examples of emojis within dataset
print(yai.iloc[1387,0]);
print(yai.iloc[1977,0]);
```

```
The best phone in market :).
:-)Oh, the charger seems to work fine.
```

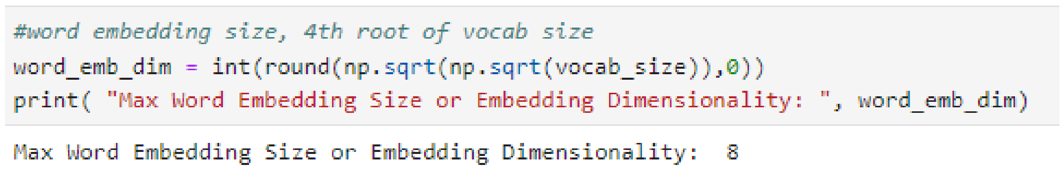
The remaining exploratory analysis was computed after various data pre-processing, which will be included in other sections. The vocabulary size included the total count of the unique words. This was compiled with the use of the TensorFlow Keras' Tokenizer() function (Cecchini, n.d.) The data was fitted with the use of the .fit_on_texts() function passing through the cleaned reviews data. A word index of all the words was compiled and the vocabulary size was obtained by adding the length of the word index + 1. The vocabulary size consisted of a total of 4,174 elements.



```
#vocab size, count of words (Elleh, 2022).
tokenizer = Tokenizer(oov_token= 'OOV', lower=False)
tokenizer.fit_on_texts(yai['reviews_clean'])
vocab_size = len(tokenizer.word_index) + 1
print(f"Vocab Size: {vocab_size} Elements.")
```

```
Vocab Size: 4174 Elements.
```

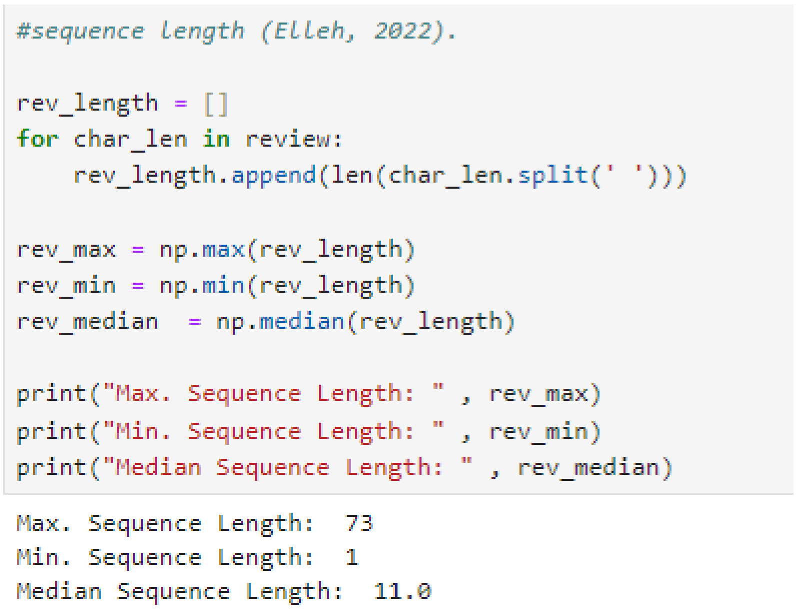
The proposed word embedding length was computed by getting the fourth root of the vocabulary size (Jarmul, n.d.). The calculation for the fourth root was square root multiplied by square root. The use of NumPy's .sqrt() function provided the computation via "np.sqrt(np.sqrt(vocab_size)". It could have also been done using " vocab_size ** 0.25" to provide the same result. The calculated value was rounded to the nearest whole integer. The combined cleaned data set had a word-embedded length of 8.



```
#word embedding size, 4th root of vocab size
word_emb_dim = int(round(np.sqrt(np.sqrt(vocab_size)),0))
print( "Max Word Embedding Size or Embedding Dimensionality: ", word_emb_dim)
```

```
Max Word Embedding Size or Embedding Dimensionality:  8
```

Lastly, the statistical justification of the maximum sequence length was choosing the maximum length while still preserving all the available data. It was used as a hyperparameter to determine the maximum length of the sequences that the model could process. It limited the padding required when modeling via Keras so that all sequence lengths were equal (Becker, n.d.). The maximum sequence length of the cleaned combined dataset was 73. The minimum sequence length was 1 and the median was 11.

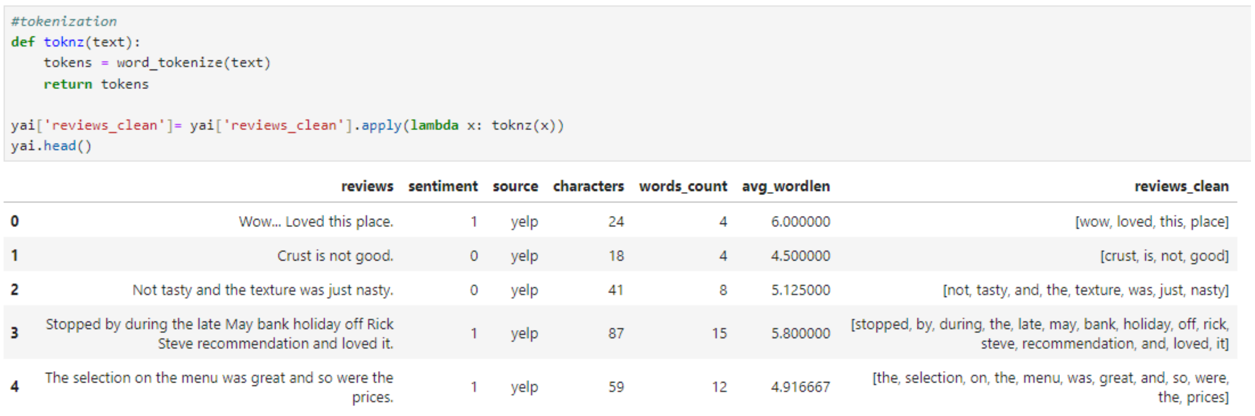


```
#sequence length (Elleh, 2022).

rev_length = []
for char_len in review:
    rev_length.append(len(char_len.split(' ')))

rev_max = np.max(rev_length)
rev_min = np.min(rev_length)
rev_median  = np.median(rev_length)

print("Max. Sequence Length: " , rev_max)
print("Min. Sequence Length: " , rev_min)
print("Median Sequence Length: " , rev_median)
```

```
Max. Sequence Length:  73
Min. Sequence Length:  1
Median Sequence Length:  11.0
```

B2. Goals of Tokenization Process

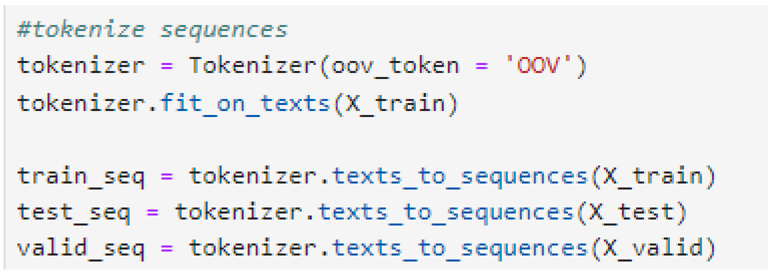
The goal of the tokenization process was to split each sequence or sentence into tokens at a word level (Jarmul, n.d.). It was completed with the use of NLTK's word_tokenize() and TensorFlow Keras' Tokenizer () functions. The word_tokenize() function was used within a created "toknz" function to split the sentences within the reviews into lists of words. For example, "The selection on the menu was great and so were the prices" converted to the list of "[the, selection, on, the, menu, was, great, and, so, were, the, prices]".



```
#tokenization
def toknz(text):
    tokens = word_tokenize(text)
    return tokens

yai['reviews_clean']= yai['reviews_clean'].apply(lambda x: toknz(x))
yai.head()
```

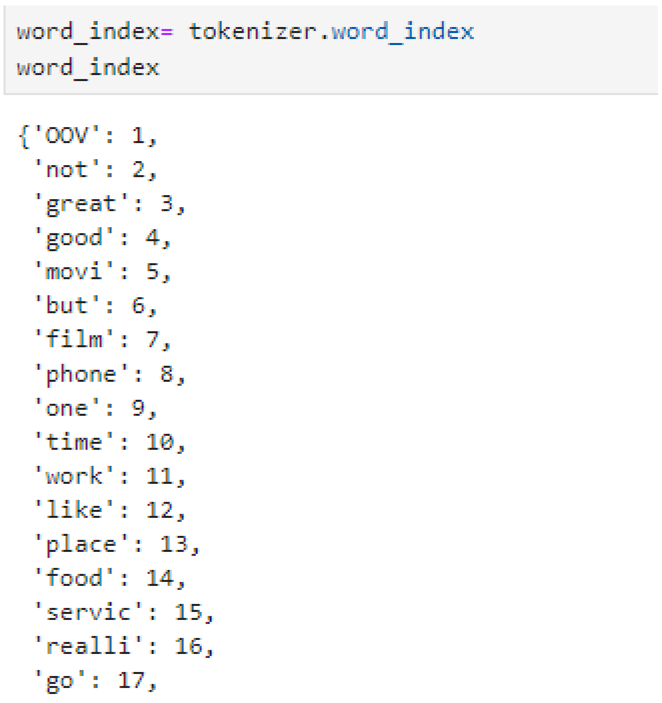
	reviews	sentiment	source	characters	words_count	avg_wordlen	reviews_clean
0	Wow... Loved this place.	1	yelp	24	4	6.000000	[wow, loved, this, place]
1	Crust is not good.	0	yelp	18	4	4.500000	[crust, is, not, good]
2	Not tasty and the texture was just nasty.	0	yelp	41	8	5.125000	[not, tasty, and, the, texture, was, just, nasty]
3	Stopped by during the late May bank holiday off Rick Steve recommendation and loved it.	1	yelp	87	15	5.800000	[stopped, by, during, the, late, may, bank, holiday, off, rick, steve, recommendation, and, loved, it]
4	The selection on the menu was great and so were the prices.	1	yelp	59	12	4.916667	[the, selection, on, the, menu, was, great, and, so, were, the, prices]

Another tokenization was performed using the TensorFlow Keras' Tokenizer() function. This function was fitted to the training data set with the usage of the .fit_on_texts() method. The generation of the tokenized sequences was completed with the .texts_to_sequences() function on each of the training, testing, and validation data sets. An example of the tokenized sequences has been included as well to provide a comparison with the padded tokenized sequences.



```
#tokenize sequences
tokenizer = Tokenizer(oov_token = 'OOV')
tokenizer.fit_on_texts(X_train)

train_seq = tokenizer.texts_to_sequences(X_train)
test_seq = tokenizer.texts_to_sequences(X_test)
valid_seq = tokenizer.texts_to_sequences(X_valid)
```

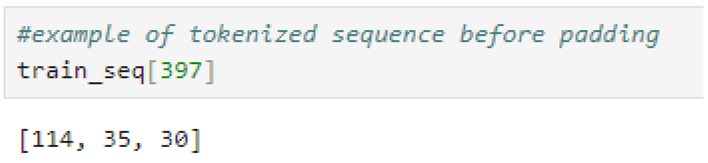
Lastly, it was passed through the word_index method to provide an index of all the words with their corresponding index number.



```
word_index= tokenizer.word_index
word_index
```

```
{'OOV': 1,
 'not': 2,
 'great': 3,
 'good': 4,
 'movi': 5,
 'but': 6,
 'film': 7,
 'phone': 8,
 'one': 9,
 'time': 10,
 'work': 11,
 'like': 12,
 'place': 13,
 'food': 14,
 'servic': 15,
 'realli': 16,
 'go': 17,
```

Note: Please see the attached AFD213Task2.PDF to see the full word index for further review if needed.

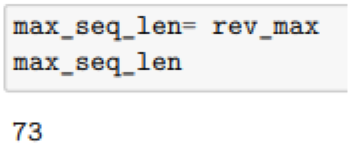


```
#example of tokenized sequence before padding
train_seq[397]
```

```
[114, 35, 30]
```

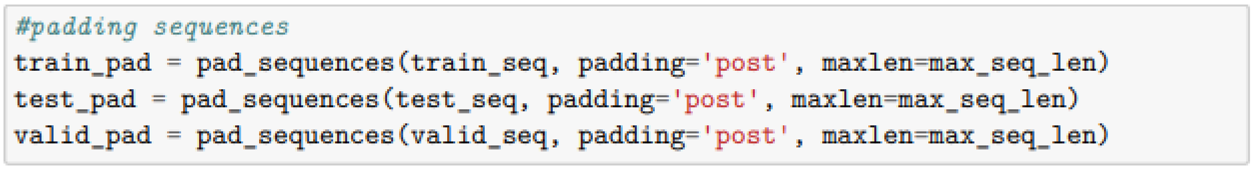
B3. Explanation of Padding Process

The length of the sequences was standardized before the sequential modeling via a padding process. The selected padding procedure included padding each sequence after the text sequence by specifying the padding of “post” within the TensorFlow Keras’ .pad_sequences() function. The padding process stored a series of zeros after the text sequence until it reached the maximum sequence length of the longest review. In this data set, the maximum sequence length after the words were tokenized was 73.



```
max_seq_len= rev_max
max_seq_len
```

```
73
```



```
#padding sequences
train_pad = pad_sequences(train_seq, padding='post', maxlen=max_seq_len)
test_pad = pad_sequences(test_seq, padding='post', maxlen=max_seq_len)
valid_pad = pad_sequences(valid_seq, padding='post', maxlen=max_seq_len)
```

Using the example noted previously, the training sequence with the word index 397 provided the following text sequence values before padding: [114, 35, 30]. After padding, the same text sequence for word index 397 was computed to the following: [114, 35, 30, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

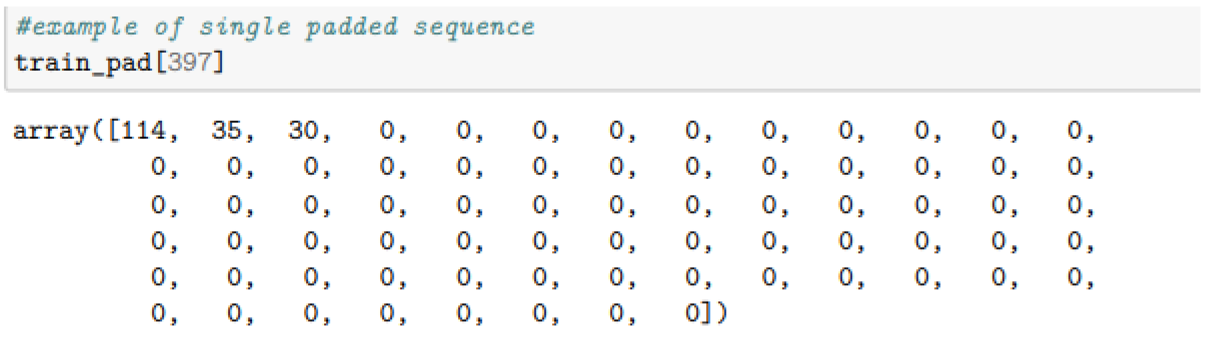
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0]



```
#example of single padded sequence
train_pad[397]
```

```
array([114,  35,  30,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0])
```

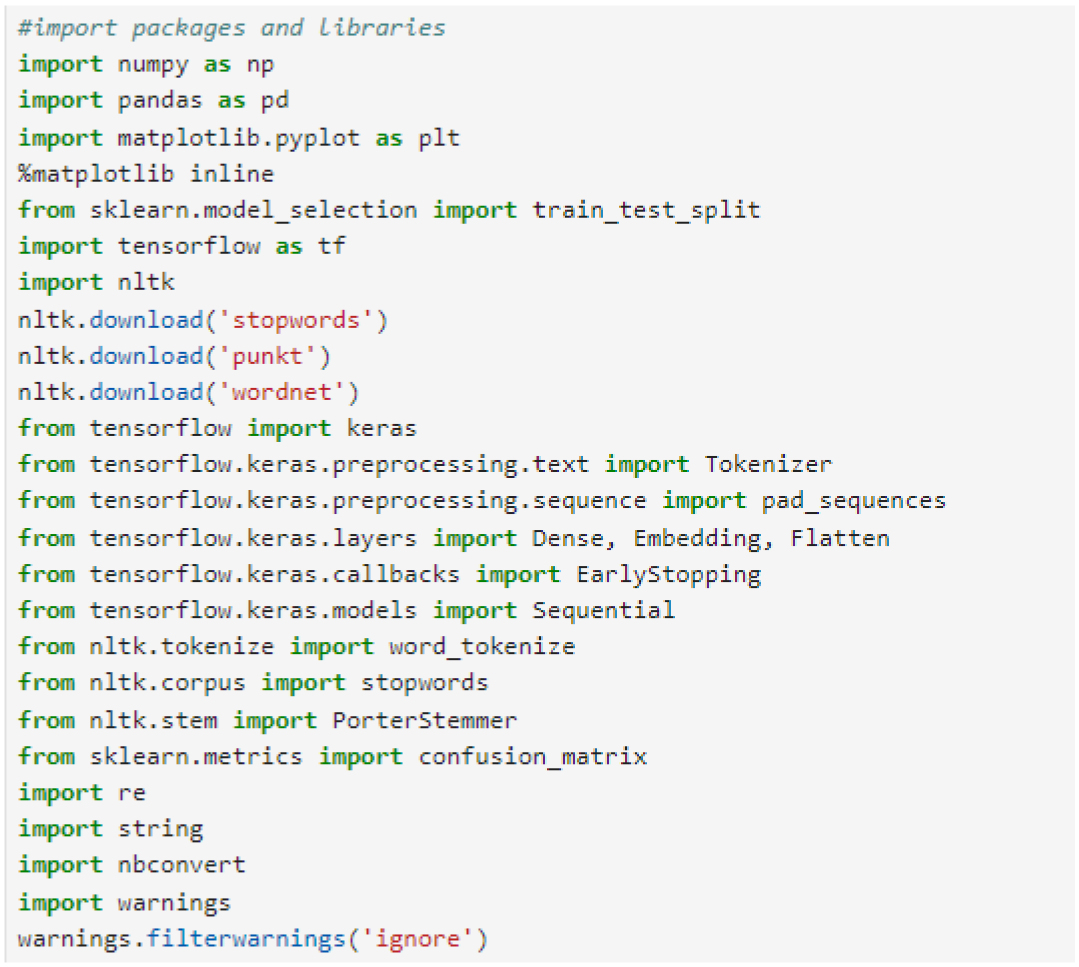
B4. Categories of Sentiment

There was a total of two categories of sentiment within the combined data set of Amazon, Yelp, and IMDB. The first sentiment was “positive”, which was represented as a value of “1”. The second sentiment was “negative”, which was represented as a value of “0”.

The activation function within the final dense layer of the model was “sigmoid” as this function fits the needed output of either “0” or “1” (Becker, n.d.).

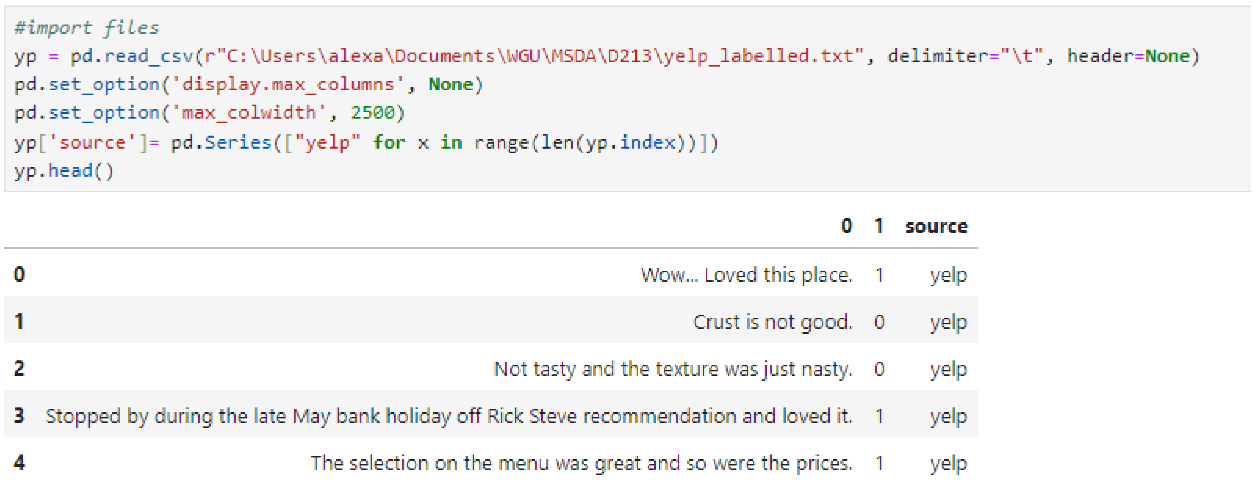
B5. Data Preparation Steps

The first step of the data preparation process was the import of the needed libraries and packages. These packages and libraries included Pandas, NumPy, Matplotlib.pyplot, Natural Language ToolKits, String, Regex, and various TensorFlow packages.



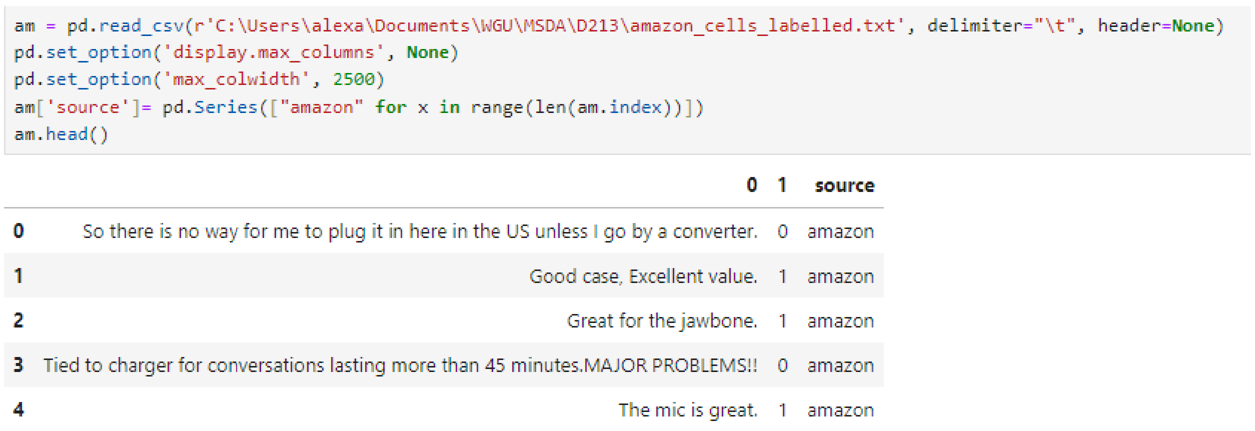
```
#import packages and libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split
import tensorflow as tf
import nltk
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
from tensorflow import keras
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Dense, Embedding, Flatten
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.models import Sequential
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from sklearn.metrics import confusion_matrix
import re
import string
import nbconvert
import warnings
warnings.filterwarnings('ignore')
```

The needed data was imported with the use of read_csv() in Python. Some issues were noted during the import of the IMDB text file due to some problems with the tab delimiter in the original file. This was resolved by using the split function on “\t” to adequately separate the columns. A “source” column was added to view where each sentiment review was populated from.



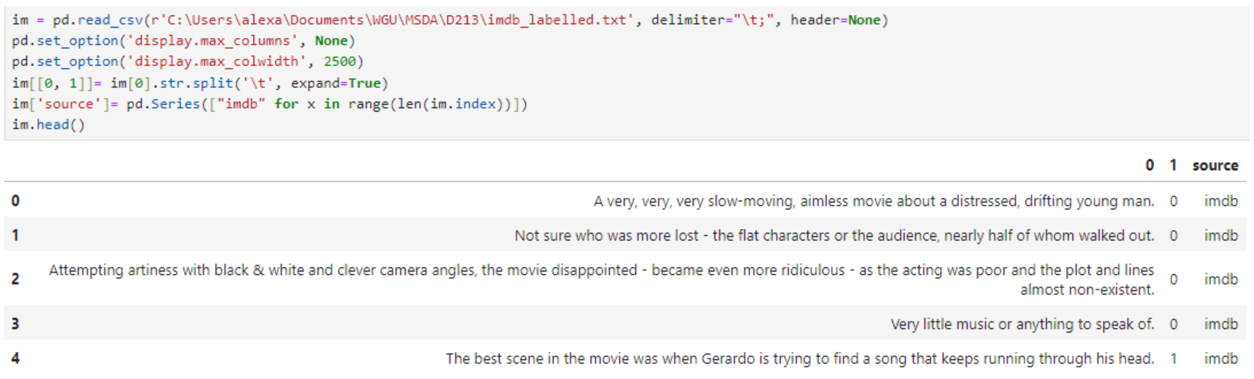
```
#import files
yp = pd.read_csv(r"C:\Users\alexa\Documents\WGU\MSDA\D213\yelp_labelled.txt", delimiter="\t", header=None)
pd.set_option('display.max_columns', None)
pd.set_option('max_colwidth', 2500)
yp['source']= pd.Series(["yelp" for x in range(len(yp.index))])
yp.head()
```

	0	1	source
0	Wow... Loved this place.	1	yelp
1	Crust is not good.	0	yelp
2	Not tasty and the texture was just nasty.	0	yelp
3	Stopped by during the late May bank holiday off Rick Steve recommendation and loved it.	1	yelp
4	The selection on the menu was great and so were the prices.	1	yelp



```
am = pd.read_csv(r'C:\Users\alexa\Documents\WGU\MSDA\D213\amazon_cells_labelled.txt', delimiter="\t", header=None)
pd.set_option('display.max_columns', None)
pd.set_option('max_colwidth', 2500)
am['source']= pd.Series(["amazon" for x in range(len(am.index))])
am.head()
```

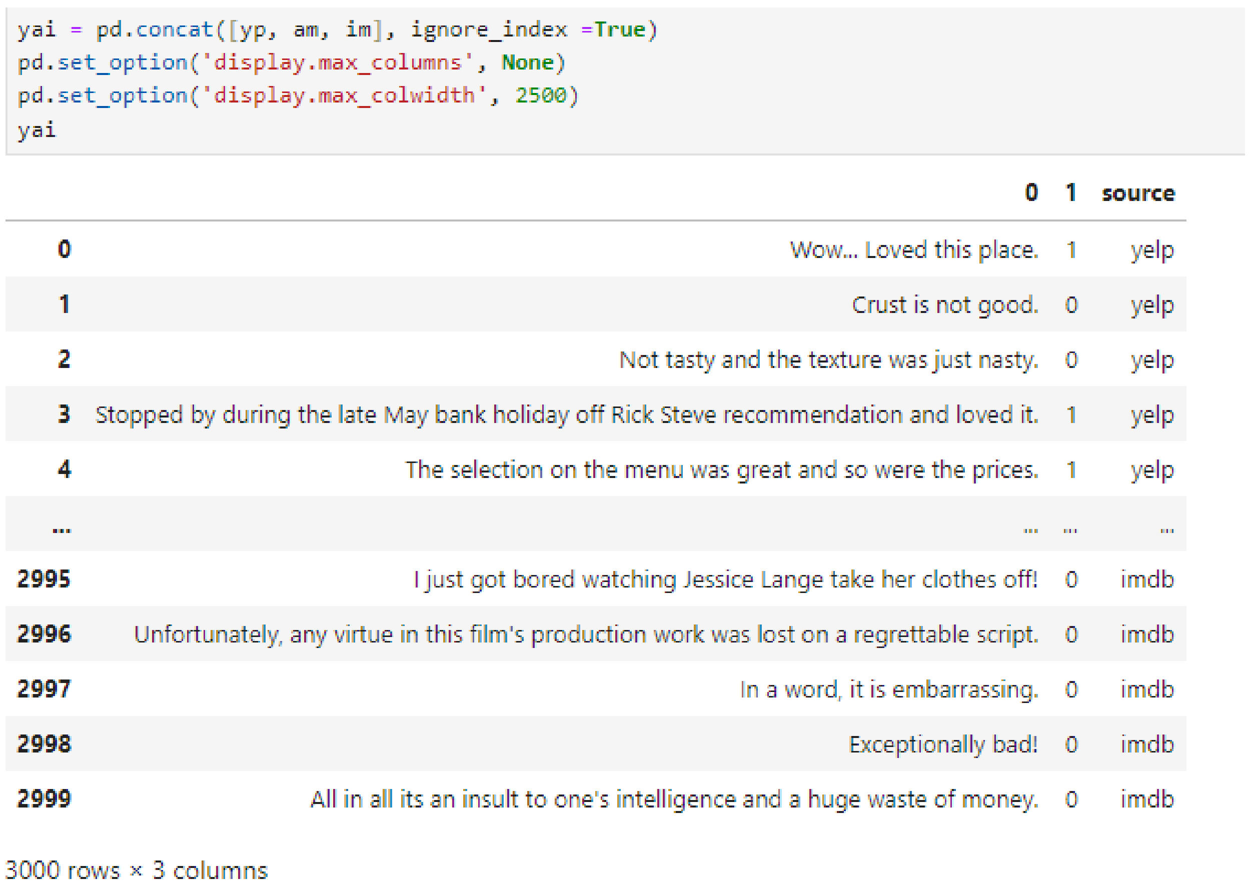
	0	1	source
0	So there is no way for me to plug it in here in the US unless I go by a converter.	0	amazon
1	Good case, Excellent value.	1	amazon
2	Great for the jawbone.	1	amazon
3	Tied to charger for conversations lasting more than 45 minutes.MAJOR PROBLEMS!!	0	amazon
4	The mic is great.	1	amazon



```
im = pd.read_csv(r'C:\Users\alexa\Documents\WGU\MSDA\D213\imdb_labelled.txt', delimiter="\t;", header=None)
pd.set_option('display.max_columns', None)
pd.set_option('display.max_colwidth', 2500)
im[[0, 1]]= im[0].str.split('\t', expand=True)
im['source']= pd.Series(["imdb" for x in range(len(im.index))])
im.head()
```

	0	1	source
0	A very, very, very slow-moving, aimless movie about a distressed, drifting young man.	0	imdb
1	Not sure who was more lost - the flat characters or the audience, nearly half of whom walked out.	0	imdb
2	Attempting artiness with black & white and clever camera angles, the movie disappointed - became even more ridiculous - as the acting was poor and the plot and lines almost non-existent.	0	imdb
3	Very little music or anything to speak of.	0	imdb
4	The best scene in the movie was when Gerardo is trying to find a song that keeps running through his head.	1	imdb

All three files were concatenated with the use of Pandas' .concat() function, and the columns were renamed for easier manipulation via the .rename() function.

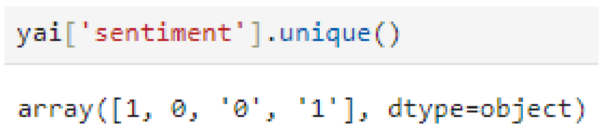


```
yai = pd.concat([yp, am, im], ignore_index =True)
pd.set_option('display.max_columns', None)
pd.set_option('display.max_colwidth', 2500)
yai
```

	0	1	source
0	Wow... Loved this place.	1	yelp
1	Crust is not good.	0	yelp
2	Not tasty and the texture was just nasty.	0	yelp
3	Stopped by during the late May bank holiday off Rick Steve recommendation and loved it.	1	yelp
4	The selection on the menu was great and so were the prices.	1	yelp
...	...	...	...
2995	I just got bored watching Jessice Lange take her clothes off!	0	imdb
2996	Unfortunately, any virtue in this film's production work was lost on a regrettable script.	0	imdb
2997	In a word, it is embarrassing.	0	imdb
2998	Exceptionally bad!	0	imdb
2999	All in all its an insult to one's intelligence and a huge waste of money.	0	imdb

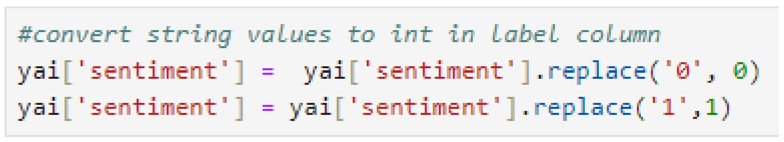
3000 rows × 3 columns

During the importing process of the IMDB data, some of the sentiment values were listed as both strings and integers. This was corrected using the .replace() function to replace the “0” or “1” string as an integer 0 or 1 instead. Once the values were updated, the attribute “sentiment” was set to a data type of integer.

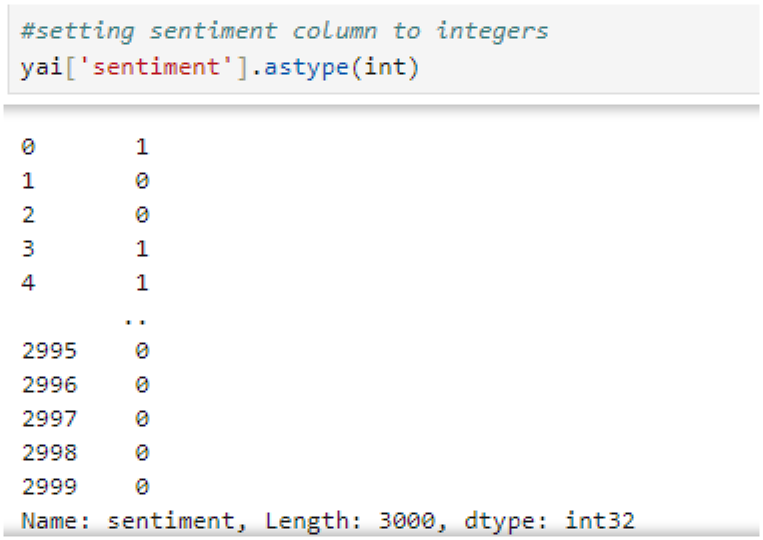


```
yai['sentiment'].unique()
```

```
array([1, 0, '0', '1'], dtype=object)
```



```
#convert string values to int in label column
yai['sentiment'] =  yai['sentiment'].replace('0', 0)
yai['sentiment'] = yai['sentiment'].replace('1',1)
```

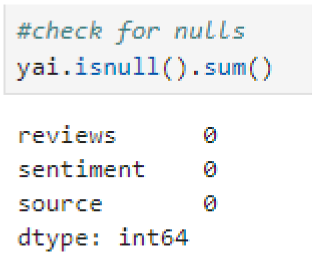


```
#setting sentiment column to integers
yai['sentiment'].astype(int)
```

```
0       1
1       0
2       0
3       1
4       1
       ..
2995    0
2996    0
2997    0
2998    0
2999    0
Name: sentiment, Length: 3000, dtype: int32
```

Next, the combined data set went through various data-cleaning steps to ensure it was ready to be used within the model. The steps included the following:

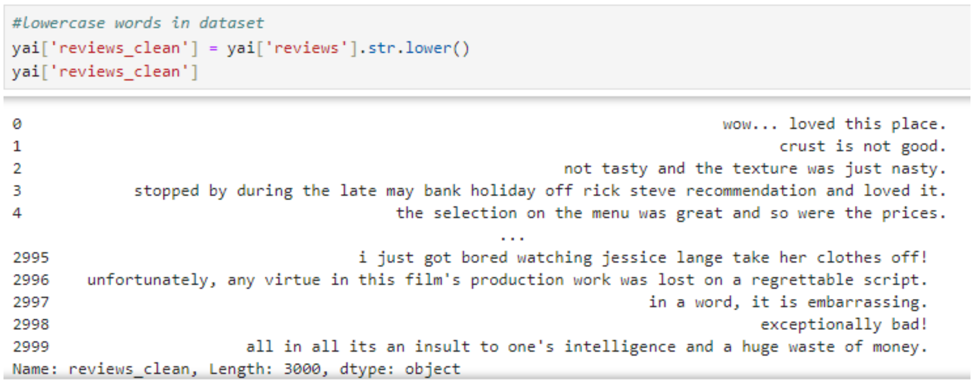
- Checking for null values. There were no missing or null values found.



```
#check for nulls
yai.isnull().sum()
```

```
reviews      0
sentiment    0
source       0
dtype: int64
```

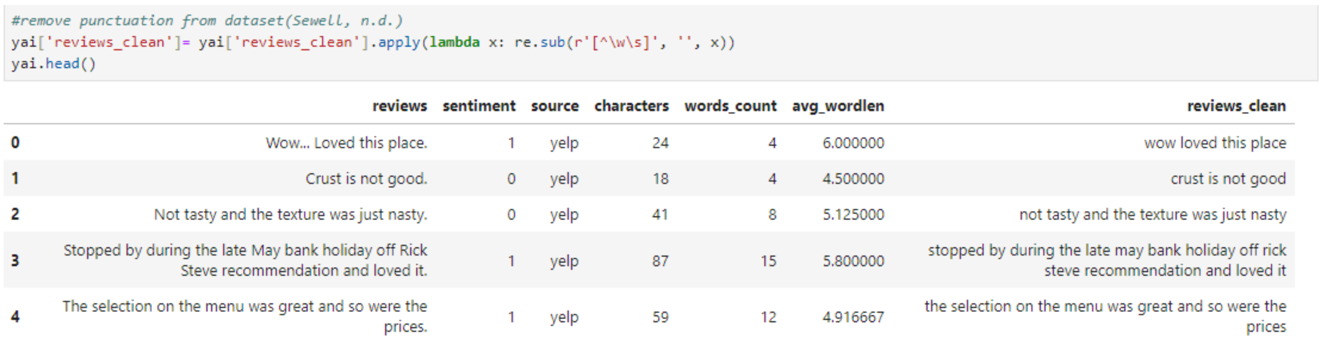
- Removal of all uppercase letters with the use of .str.lower() function. This function replaced all the uppercase letters with their equivalent lowercase value.



```
#lowercase words in dataset
yai['reviews_clean'] = yai['reviews'].str.lower()
yai['reviews_clean']
```

```
0                                                             wow... loved this place.
1                                                                  crust is not good.
2                                              not tasty and the texture was just nasty.
3            stopped by during the late may bank holiday off rick steve recommendation and loved it.
4                                  the selection on the menu was great and so were the prices.
                                                     ...
2995                               i just got bored watching jessice lange take her clothes off!
2996      unfortunately, any virtue in this film's production work was lost on a regrettable script.
2997                                                              in a word, it is embarrassing.
2998                                                                         exceptionally bad!
2999                        all in all its an insult to one's intelligence and a huge waste of money.
Name: reviews_clean, Length: 3000, dtype: object
```

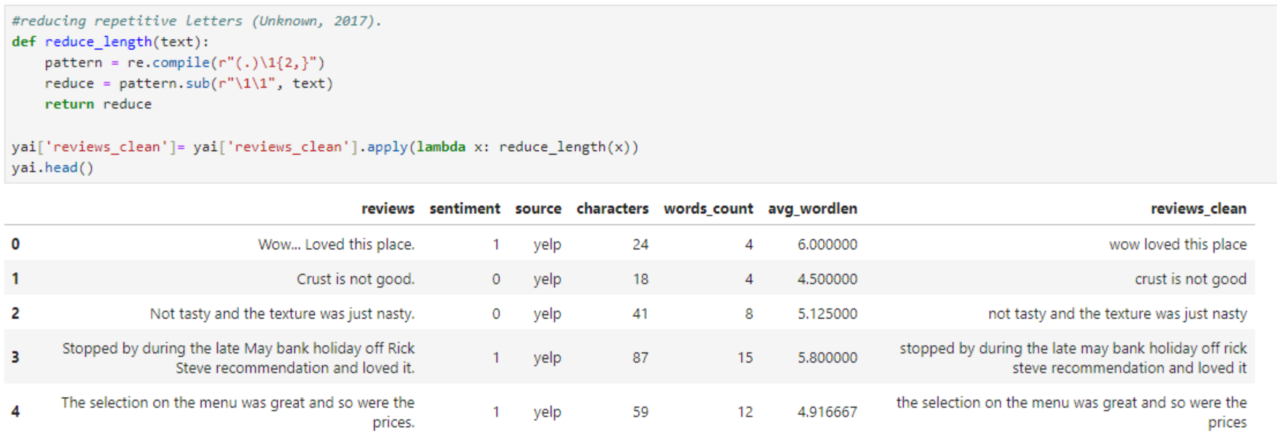
- Removal of punctuation. This was completed with the Regex sub function. The function removed all punctuation that was deemed anything, not a word or space.



```
#remove punctuation from dataset(Sewell, n.d.)
yai['reviews_clean']= yai['reviews_clean'].apply(lambda x: re.sub(r'[^\w\s]', '', x))
yai.head()
```

	reviews	sentiment	source	characters	words_count	avg_wordlen	reviews_clean
0	Wow... Loved this place.	1	yelp	24	4	6.000000	wow loved this place
1	Crust is not good.	0	yelp	18	4	4.500000	crust is not good
2	Not tasty and the texture was just nasty.	0	yelp	41	8	5.125000	not tasty and the texture was just nasty
3	Stopped by during the late May bank holiday off Rick Steve recommendation and loved it.	1	yelp	87	15	5.800000	stopped by during the late may bank holiday off rick steve recommendation and loved it
4	The selection on the menu was great and so were the prices.	1	yelp	59	12	4.916667	the selection on the menu was great and so were the prices

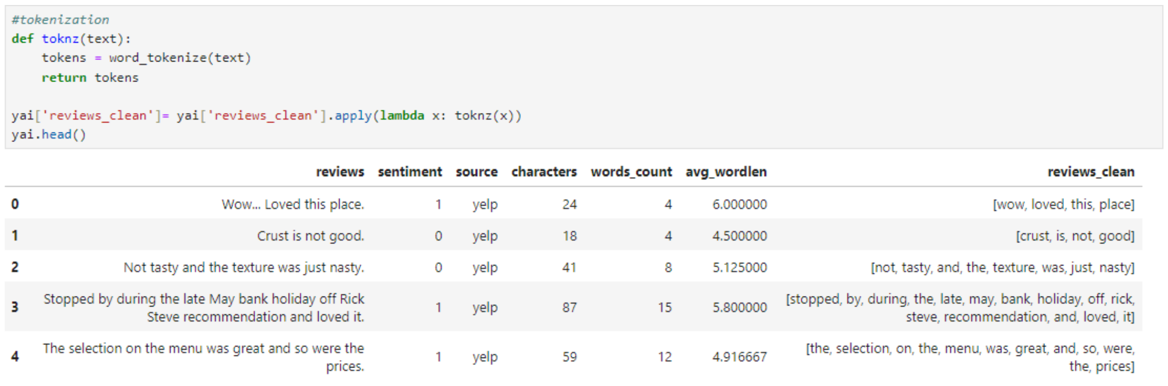
- Reducing repetitive letters over 2 iterations with the use of a created function called "reduce_length". The created function used the Regex compile() function to reduce any words with repeating letters over two times as noted within the English language. For example: "Recommmmmend" would be reduced to "Recommend".



```
#reducing repetitive letters (Unknown, 2017).
def reduce_length(text):
    pattern = re.compile(r"(.)\1{2,}")
    reduce = pattern.sub(r"\1\1", text)
    return reduce

yai['reviews_clean']= yai['reviews_clean'].apply(lambda x: reduce_length(x))
yai.head()
```

	reviews	sentiment	source	characters	words_count	avg_wordlen	reviews_clean
0	Wow... Loved this place.	1	yelp	24	4	6.000000	wow loved this place
1	Crust is not good.	0	yelp	18	4	4.500000	crust is not good
2	Not tasty and the texture was just nasty.	0	yelp	41	8	5.125000	not tasty and the texture was just nasty
3	Stopped by during the late May bank holiday off Rick Steve recommendation and loved it.	1	yelp	87	15	5.800000	stopped by during the late may bank holiday off rick steve recommendation and loved it
4	The selection on the menu was great and so were the prices.	1	yelp	59	12	4.916667	the selection on the menu was great and so were the prices

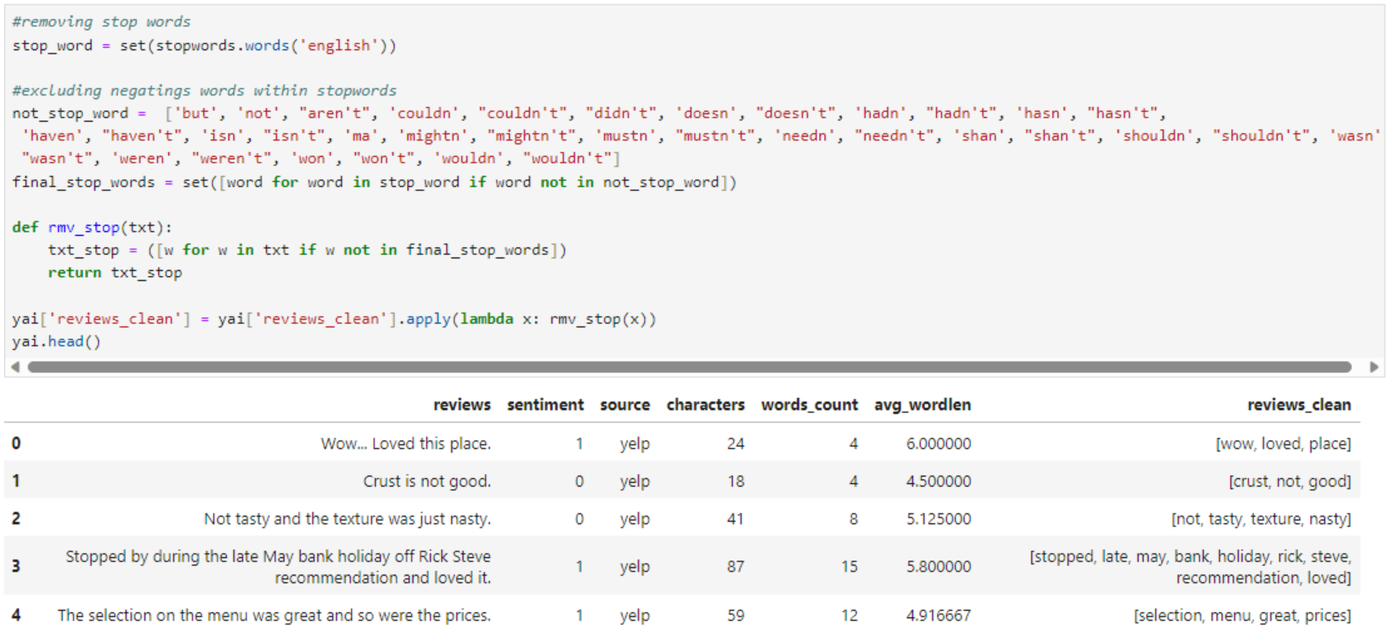
- Tokenization of the text via word_tokenize() function. This was explained in the above tokenization process found in B2.



```
#tokenization
def toknz(text):
    tokens = word_tokenize(text)
    return tokens

yai['reviews_clean']= yai['reviews_clean'].apply(lambda x: toknz(x))
yai.head()
```

	reviews	sentiment	source	characters	words_count	avg_wordlen	reviews_clean
0	Wow... Loved this place.	1	yelp	24	4	6.000000	[wow, loved, this, place]
1	Crust is not good.	0	yelp	18	4	4.500000	[crust, is, not, good]
2	Not tasty and the texture was just nasty.	0	yelp	41	8	5.125000	[not, tasty, and, the, texture, was, just, nasty]
3	Stopped by during the late May bank holiday off Rick Steve recommendation and loved it.	1	yelp	87	15	5.800000	[stopped, by, during, the, late, may, bank, holiday, off, rick, steve, recommendation, and, loved, it]
4	The selection on the menu was great and so were the prices.	1	yelp	59	12	4.916667	[the, selection, on, the, menu, was, great, and, so, were, the, prices]

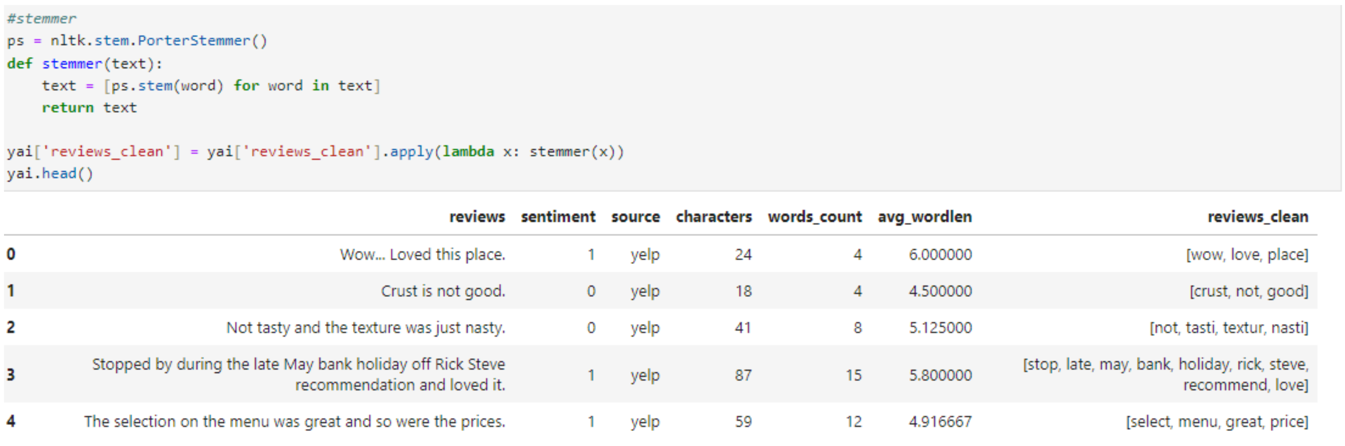
- Removal of stop words. Stop words within the natural language toolkit were words commonly used within data such as “a”, “he”, “it”, “not” etc. These were not needed to ascertain the meaning of the sentences. In the case of this combined data set, it was noted to keep the negating stop words as removing them would change the meaning or context of the sequence. For example, “not”, “mustn’t”… so on and so forth.



```
#removing stop words
stop_word = set(stopwords.words('english'))

#excluding negatings words within stopwords
not_stop_word =  ['but', 'not', "aren't", 'couldn', "couldn't", "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't",
 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn'
 "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
final_stop_words = set([word for word in stop_word if word not in not_stop_word])

def rmv_stop(txt):
    txt_stop = ([w for w in txt if w not in final_stop_words])
    return txt_stop

yai['reviews_clean'] = yai['reviews_clean'].apply(lambda x: rmv_stop(x))
yai.head()
```

	reviews	sentiment	source	characters	words_count	avg_wordlen	reviews_clean
0	Wow... Loved this place.	1	yelp	24	4	6.000000	[wow, loved, place]
1	Crust is not good.	0	yelp	18	4	4.500000	[crust, not, good]
2	Not tasty and the texture was just nasty.	0	yelp	41	8	5.125000	[not, tasty, texture, nasty]
3	Stopped by during the late May bank holiday off Rick Steve recommendation and loved it.	1	yelp	87	15	5.800000	[stopped, late, may, bank, holiday, rick, steve, recommendation, loved]
4	The selection on the menu was great and so were the prices.	1	yelp	59	12	4.916667	[selection, menu, great, prices]

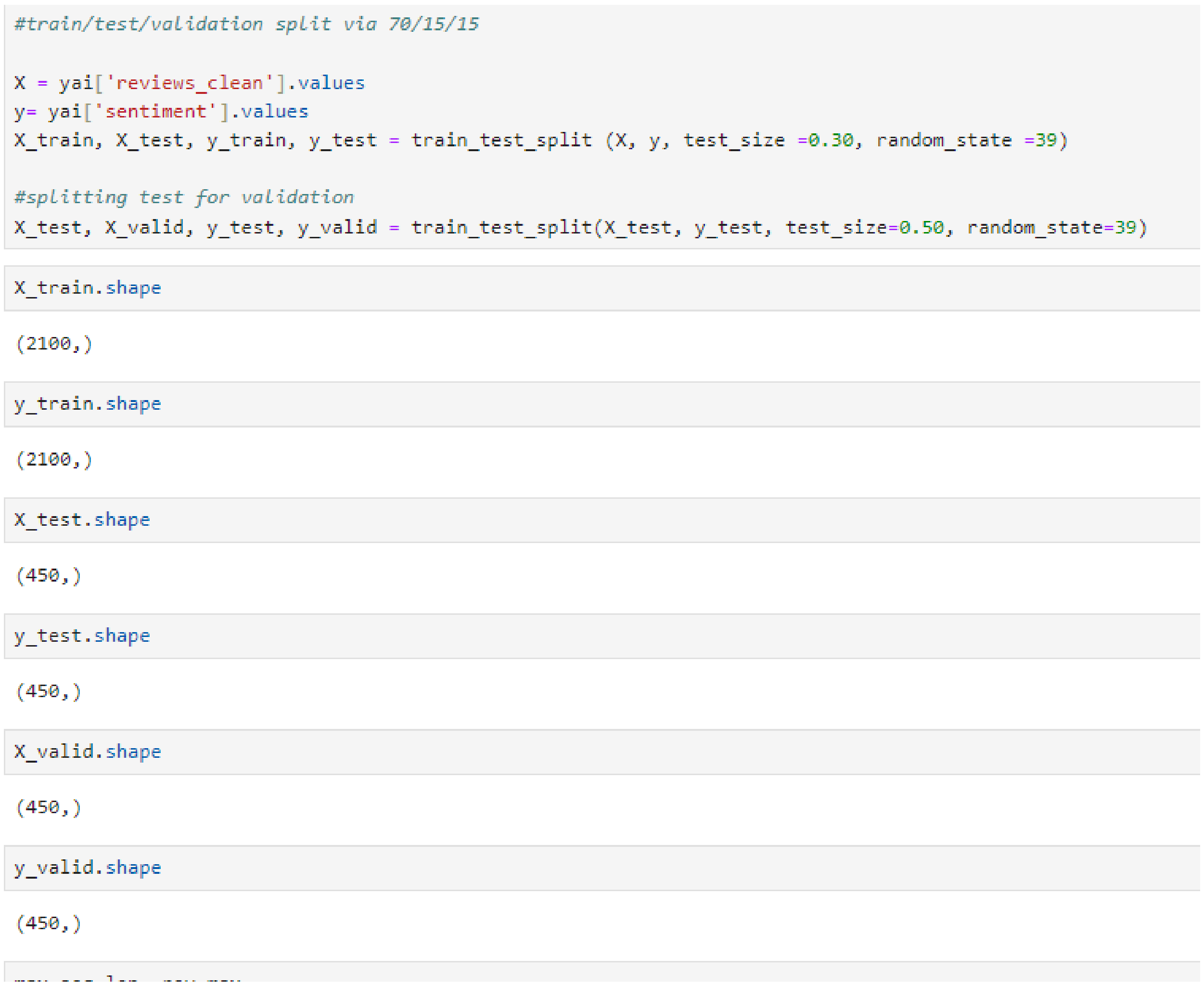
- Stemming of the words using the Natural language toolkit(NLTK)’s Porter Stemmer function. The PorterStemmer() function reduced each word down to its word stem. For example, “tasty, tastier, tastiest” converted to the stem of “tasti”. This was useful in grouping similar words and reducing the number of unique words within the vocabulary size.



```
#stemmer
ps = nltk.stem.PorterStemmer()
def stemmer(text):
    text = [ps.stem(word) for word in text]
    return text

yai['reviews_clean'] = yai['reviews_clean'].apply(lambda x: stemmer(x))
yai.head()
```

	reviews	sentiment	source	characters	words_count	avg_wordlen	reviews_clean
0	Wow... Loved this place.	1	yelp	24	4	6.000000	[wow, love, place]
1	Crust is not good.	0	yelp	18	4	4.500000	[crust, not, good]
2	Not tasty and the texture was just nasty.	0	yelp	41	8	5.125000	[not, tasti, textur, nasti]
3	Stopped by during the late May bank holiday off Rick Steve recommendation and loved it.	1	yelp	87	15	5.800000	[stop, late, may, bank, holiday, rick, steve, recommend, love]
4	The selection on the menu was great and so were the prices.	1	yelp	59	12	4.916667	[select, menu, great, price]

➢ Splitting data into Training, Testing, and Validation data sets for modeling. The split of the data was set as a 70/15/15 value. These values meant that the original cleaned data was split with the training set as 70% or 2,100 entries of the original data set. The testing set was 15% or 450 entries of the original data set. The validation set was the remaining 15% or 450 entries of the original data set. This was completed with the use of SciKitLearn’s train_test_split() function. The “X” and “y” values were first set to the necessary column values. The “X” represented the “reviews_clean” values. The “y” represented the “sentiment” values. The parameters used within the first .train_test_split() function was the X, y, test_size of 0.30, and random_state. The test size meant the training set was 70% and the testing set was 30%. The random_state parameter was a random seed so that the splits will be the same randomization each time. If a random seed was not set, the split will be different every time. For the validation set, the testing set was split again using the .train_test_split() function. In this iteration, the test_size was set to 0.50 or 50%. This was calculated as such because half or 50% of 30% of the original data equaled 15% of the original data. (0.50 * 30 = 15) This left the testing set with a final percentage of 15% of the original data set and the validation with the other 15%.



```
#train/test/validation split via 70/15/15

X = yai['reviews_clean'].values
y= yai['sentiment'].values
X_train, X_test, y_train, y_test = train_test_split (X, y, test_size =0.30, random_state =39)

#splitting test for validation
X_test, X_valid, y_test, y_valid = train_test_split(X_test, y_test, test_size=0.50, random_state=39)
```

```
X_train.shape
```

```
(2100,)
```

```
y_train.shape
```

```
(2100,)
```

```
X_test.shape
```

```
(450,)
```

```
y_test.shape
```

```
(450,)
```

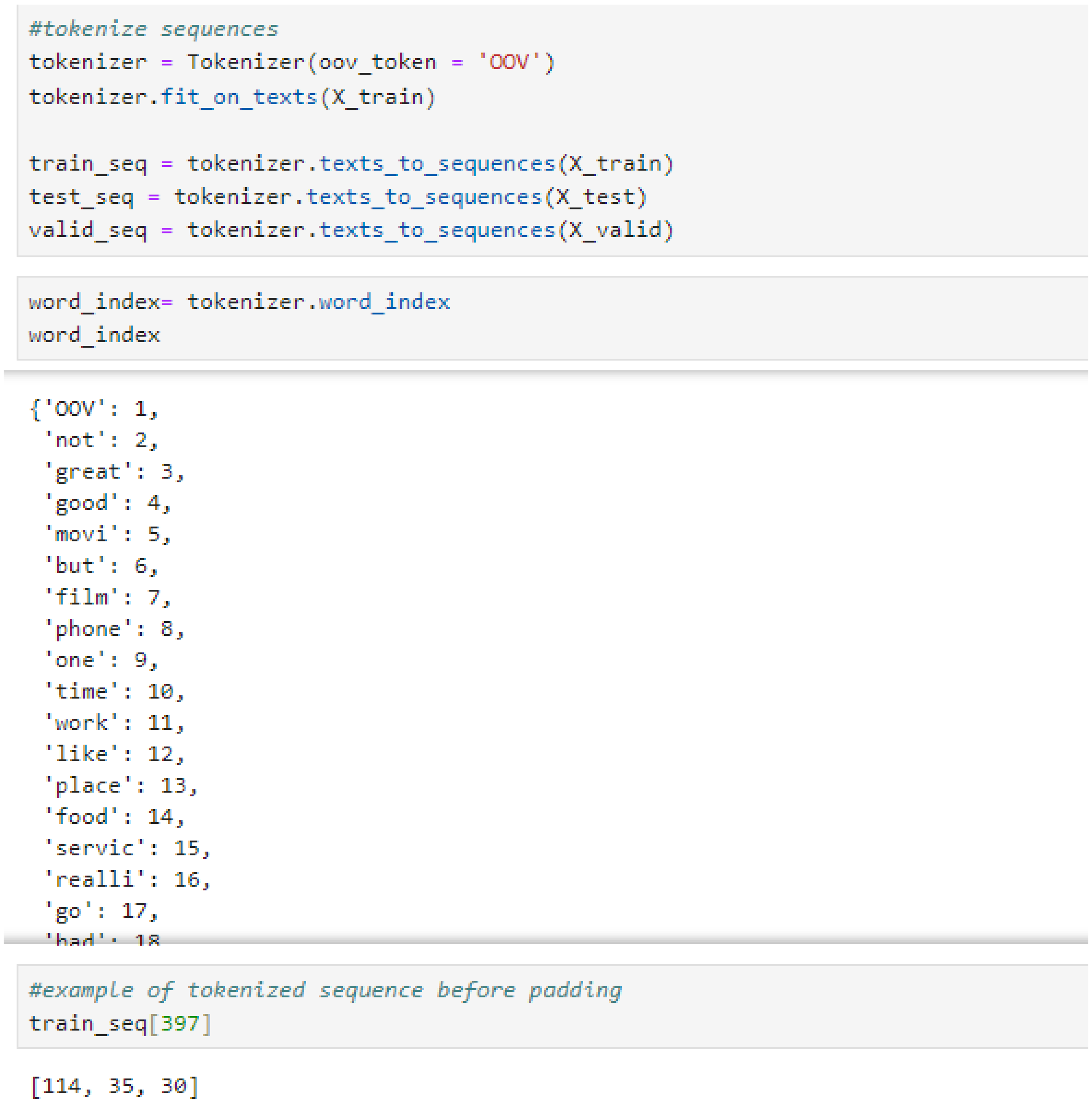
```
X_valid.shape
```

```
(450,)
```

```
y_valid.shape
```

```
(450,)
```

- The tokenization and padding process was completed following the data split. This was previously explained in the Tokenization and Padding process in B2 and B3.



```
#tokenize sequences
tokenizer = Tokenizer(oov_token = 'OOV')
tokenizer.fit_on_texts(X_train)

train_seq = tokenizer.texts_to_sequences(X_train)
test_seq = tokenizer.texts_to_sequences(X_test)
valid_seq = tokenizer.texts_to_sequences(X_valid)
```

```
word_index= tokenizer.word_index
word_index
```

```
{'OOV': 1,
 'not': 2,
 'great': 3,
 'good': 4,
 'movi': 5,
 'but': 6,
 'film': 7,
 'phone': 8,
 'one': 9,
 'time': 10,
 'work': 11,
 'like': 12,
 'place': 13,
 'food': 14,
 'servic': 15,
 'realli': 16,
 'go': 17,
 'had': 18
```

```
#example of tokenized sequence before padding
train_seq[397]
```

```
[114, 35, 30]
```

```
#padding sequences
train_pad = pad_sequences(train_seq, padding='post', maxlen=max_seq_len)
test_pad = pad_sequences(test_seq, padding='post', maxlen=max_seq_len)
valid_pad = pad_sequences(valid_seq, padding='post', maxlen=max_seq_len)
```

```
train_pad.shape
```

```
(2100, 73)
```

```
train_pad
```

```
array([[ 951, 1458,    5, ...,    0,    0,    0],
       [   2,  416,  952, ...,    0,    0,    0],
       [ 145,  715, 1459, ...,    0,    0,    0],
       ...,
       [3385, 3386, 3387, ...,    0,    0,    0],
       [  32,   43,   11, ...,    0,    0,    0],
       [ 192, 3392,  184, ...,    0,    0,    0]])
```

```
#example of single padded sequence
train_pad[397]
```

```
array([114,  35,  30,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0])
```

```
test_pad.shape
```

```
(450, 73)
```

```
test_pad
```

```
array([[ 114, 1866,   27, ...,    0,    0,    0],
       [ 118,  538, 2541, ...,    0,    0,    0],
       [  39,  171,   19, ...,    0,    0,    0],
       ...,
       [  49,  254,   30, ...,    0,    0,    0],
       [ 409,  471,   68, ...,    0,    0,    0],
       [   1,   34,  278, ...,    0,    0,    0]])
```

```
valid_pad.shape
```

```
(450, 73)
```

```
valid_pad
```

```
array([[ 908, 1431, 1413, ...,    0,    0,    0],
       [  26,  639,   66, ...,    0,    0,    0],
       [1454,   14,  186, ...,    0,    0,    0],
       ...,
       [ 164,    1,  318, ...,    0,    0,    0],
       [   1,    1,   82, ...,    0,    0,    0],
       [1253,  235,  231, ...,    0,    0,    0]])
```

B6. Prepared Data Set

Please see attached CSV files to view the prepared cleaned data sets. The CSVs files included the following:

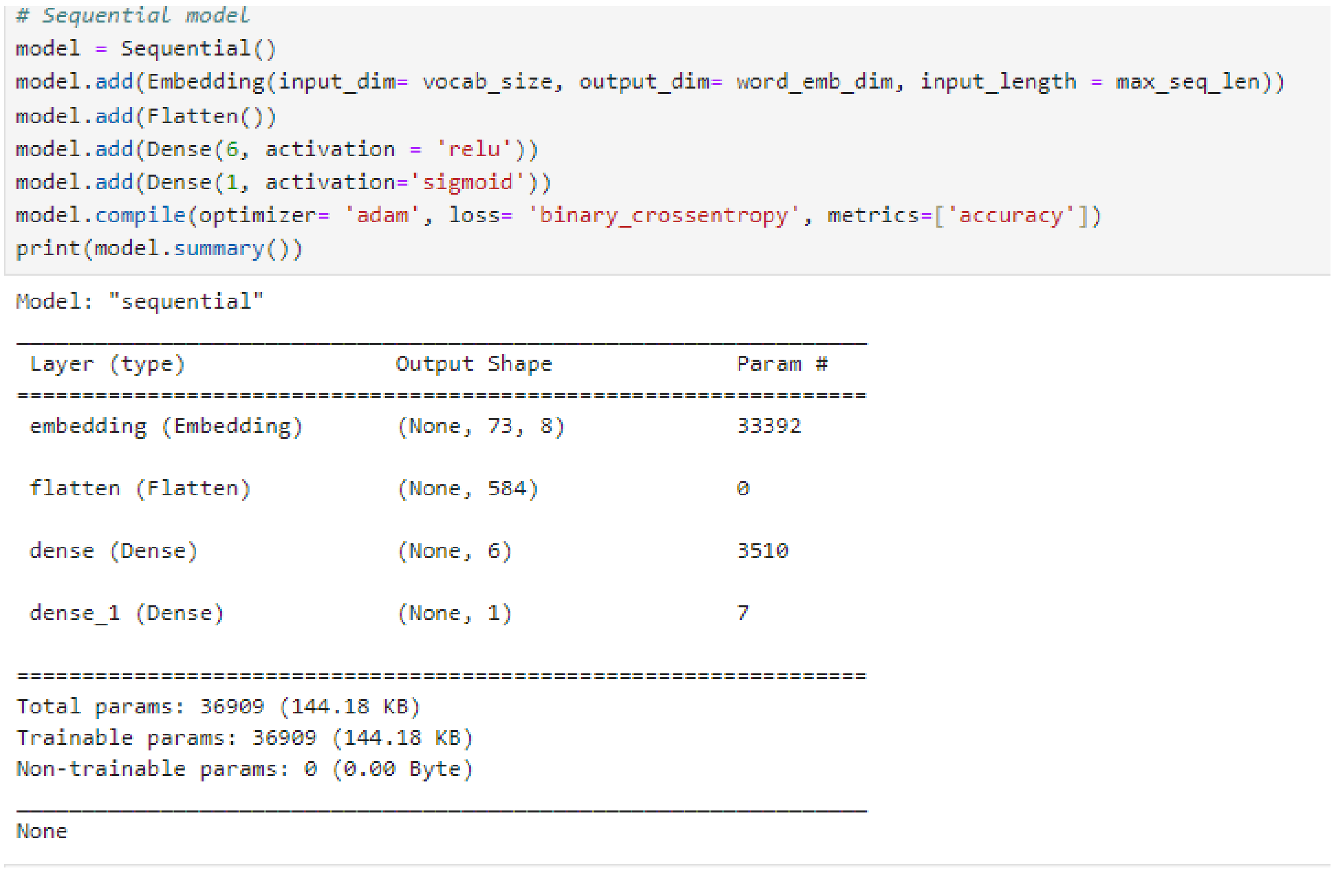
- Combined Clean Data: “AFD213Tk2_cleaned.csv”
- Training Data Sets: “AFD213Tk2_trainpad.csv” & “AFD213Tk2_trainreviews.csv”
- Testing Data Sets: “AFD213Tk2_testpad.csv”& “AFD213Tk2_testreviews.csv”
- Validation Data Sets: “AFD213Tk2_validpad.csv”& “AFD213Tk2_validreviews.csv”

```
#printing cleaned csv files

yai.to_csv("AFD213Tk2_cleaned.csv")
pd.DataFrame(train_pad).to_csv("AFD213Tk2_trainpad.csv")
pd.DataFrame(test_pad).to_csv("AFD213Tk2_testpad.csv")
pd.DataFrame(valid_pad).to_csv("AFD213Tk2_validpad.csv")
pd.DataFrame(y_valid).to_csv("AFD213Tk2_validreviews.csv")
pd.DataFrame(y_train).to_csv("AFD213Tk2_trainreviews.csv")
pd.DataFrame(y_test).to_csv("AFD213Tk2_testreviews.csv")
```

Part III: Network Architecture

C1. Output of Model Summary via TensorFlow

The TensorFlow Model Summary output was provided within the below snippet.



```
# Sequential model
model = Sequential()
model.add(Embedding(input_dim= vocab_size, output_dim= word_emb_dim, input_length = max_seq_len))
model.add(Flatten())
model.add(Dense(6, activation = 'relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer= 'adam', loss= 'binary_crossentropy', metrics=['accuracy'])
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding (Embedding)       (None, 73, 8)             33392     
                                                                 
 flatten (Flatten)           (None, 584)               0         
                                                                 
 dense (Dense)               (None, 6)                 3510      
                                                                 
 dense_1 (Dense)             (None, 1)                 7         
                                                                 
=================================================================
Total params: 36909 (144.18 KB)
Trainable params: 36909 (144.18 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

C2. Summary of Layers

The TensorFlow Summary model was a sequential model, which contained 4 layers.

The first layer of the model was the Embedding Layer. This layer was noted as a more complex version of one hot encoding, using real values instead of “1” and “0” (Saxena, 2020). The layer had 33,392 parameters, which marked it as the largest layer. The layer fed the data’s vocabulary size as the input dimension multiplied by the word embedding dimension as an output dimension to provide the calculated parameters. (4174 * 8 = 33,392). The input length was also included in this layer's computation, which was 73 elements wide.

The second layer of the model was a Flatten Layer. This layer flattens the inputs of the dimensions into a single dimension. It allowed for the model to proceed easily down the following Dense Layers. The Flatten layer has no parameters and just transformed the data into a form for easy handling. In comparison to the Embedding Layer, the Flatten layer used a single dimension of 584. This was computed via the output dimension multiplied by the input length. (73 * 8 = 584)

The third layer was a Dense Layer containing 6 nodes and an activation function of ReLU. This layer narrowed the data from its larger state to a smaller size. The 584 elements were reduced to 6 elements. This handled 3,510 parameters. This was calculated by multiplying the 584 elements from the Flatten layer and the 6 elements from this Dense layer to provide a total of 3,504, plus adding an output round of 6 for a total of 3,510 parameters. (584 * 6 = 3504 + 6 = 3510)

The final layer was the Dense layer containing 1 node and an activation function of Sigmoid. This layer showed the final output of the model as a single value. The single value would produce either a 1 (positive sentiment) or a 0 (negative sentiment). This layer contained a total of 7 parameters. It was computed similarly to the previous layer by multiplying the previous layer's node of 6 by this layer's node of 1 then adding a output round. (6 * 1 = 6 +1 = 7)

C3. Justification of Parameters

The hyperparameters justification within the model was explained by the following six elements:

- Activation Functions
- Number of Nodes per Layer
- Loss Function
- Optimizer
- Stopping Criteria
- Evaluation Metric

The activation functions of the two dense layers included ReLU and Sigmoid. ReLU, which stands for Rectified Linear Unit, was the default activation function for various types of neural networks. As it was considered a default activation for the first dense layer, it was utilized within the modeling. ReLU was a linear function, which returned the input directly if it is positive and if it was negative, it would return a zero (Brownlee, A Gentle Introduction to the Rectified Linear Unit (ReLU), 2020).

Sigmoid was a nonlinear activation function often called a logistic function (Brownlee, A Gentle Introduction to the Rectified Linear Unit (ReLU), 2020). This activation function was good for binary classification since it exists around “0” and “1”, which was useful in the prediction of the output’s corresponding binary class (Waite, 2020).

The number of nodes per layer was depicted within each layer of the model. The embedding layer contained “8” nodes, which was the size of the output dimension. The flatten layer did not have any nodes as it was utilized in the transformation of the data for the following nodes. The two dense layers contained “6” and “1” nodes respectively. These dense layers were hidden layers that continuously reduced the original input length of 73 text sequences to a single output of either “1” or “0”.

The loss function of the model was binary cross entropy. The loss function was utilized due to solving classification (Hull, n.d.). Binary means only two, which was denoted as “0” or “1”. The sentiment could only be positive or negative with no gray overlapping areas. Binary cross entropy accounted for the results being restricted to “0” or “1”.

The optimizer used within the modeling was “Adam”. Adaptive moment estimation (Adam) was an algorithm that used gradient information to modify the weight parameters of the prediction model (Brownlee, Gentle Introduction to the Adam Optimization Algorithm for Deep Learning, 2021). It was considered an industry standard in improving the accuracy of neural networks.

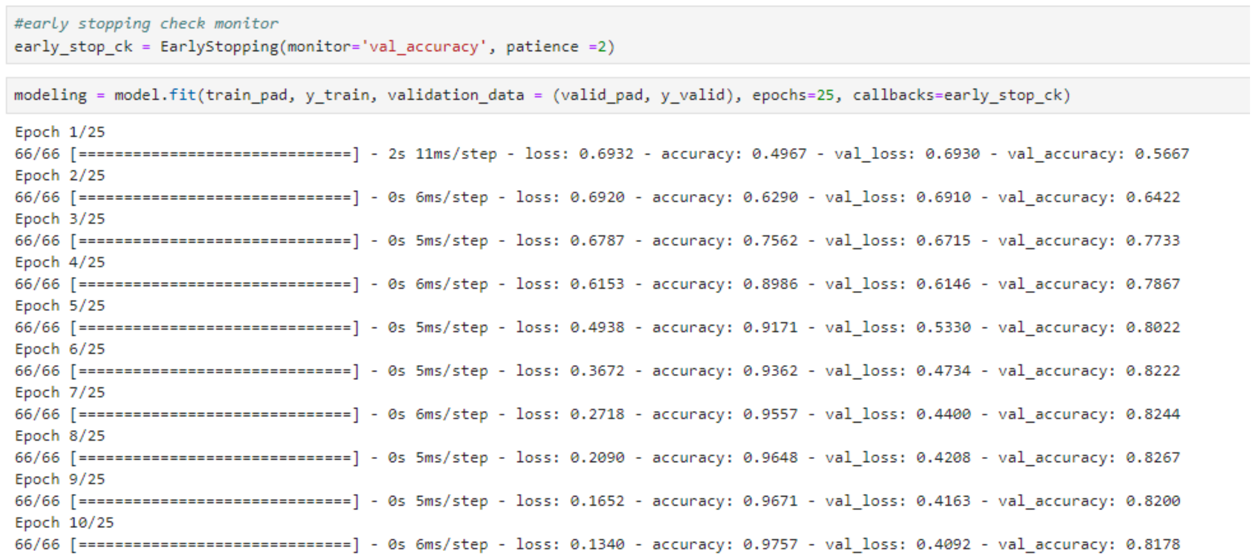
The validation accuracy or “val_accuracy” was the stopping criterion of this model. The evaluation of the model based on the loss metric of the training data would have provided information into the case of under or overfitting. It was decided that based on the research thesis utilizing prediction accuracy would be more aligned. The patience of 2 epochs without improvement was set for this metric. This would stop the model from reiterating after two poor performances.

The evaluation metric for this model was accuracy. The model was focused on increasing the accuracy of the training set. It was fitted on the training set to provide the best accuracy results while still considering the validation accuracy to avoid performance pitfalls within the validation set.

Part IV: Model Evaluation

D1. Impact of Stopping Criteria

The impact of the stopping criteria was visualized by the model fit of the training data. The model stops training at Epoch 11. It stopped due to the early stopper check monitor noticing a repetition of poor performance after two consecutive epochs. If the early stopper check monitor did not locate two poor performing consecutive epochs, the model would continue to iterate until it met the threshold of the epoch, which was specified as 25. Epoch 9 had the highest accuracy and lowest loss across both the training and validation sets, with an accuracy of 96.71%, validation accuracy of 82.00%, loss of 16.52 %, and validation loss of 41.63 %. The goal of the model was to have an accuracy metric of over 80% and a low loss percentage.



```
#early stopping check monitor
early_stop_ck = EarlyStopping(monitor='val_accuracy', patience =2)
```

```
modeling = model.fit(train_pad, y_train, validation_data = (valid_pad, y_valid), epochs=25, callbacks=early_stop_ck)
```

```
Epoch 1/25
66/66 [==============================] - 2s 11ms/step - loss: 0.6932 - accuracy: 0.4967 - val_loss: 0.6930 - val_accuracy: 0.5667
Epoch 2/25
66/66 [==============================] - 0s 6ms/step - loss: 0.6920 - accuracy: 0.6290 - val_loss: 0.6910 - val_accuracy: 0.6422
Epoch 3/25
66/66 [==============================] - 0s 5ms/step - loss: 0.6787 - accuracy: 0.7562 - val_loss: 0.6715 - val_accuracy: 0.7733
Epoch 4/25
66/66 [==============================] - 0s 6ms/step - loss: 0.6153 - accuracy: 0.8986 - val_loss: 0.6146 - val_accuracy: 0.7867
Epoch 5/25
66/66 [==============================] - 0s 5ms/step - loss: 0.4938 - accuracy: 0.9171 - val_loss: 0.5330 - val_accuracy: 0.8022
Epoch 6/25
66/66 [==============================] - 0s 5ms/step - loss: 0.3672 - accuracy: 0.9362 - val_loss: 0.4734 - val_accuracy: 0.8222
Epoch 7/25
66/66 [==============================] - 0s 6ms/step - loss: 0.2718 - accuracy: 0.9557 - val_loss: 0.4400 - val_accuracy: 0.8244
Epoch 8/25
66/66 [==============================] - 0s 5ms/step - loss: 0.2090 - accuracy: 0.9648 - val_loss: 0.4208 - val_accuracy: 0.8267
Epoch 9/25
66/66 [==============================] - 0s 5ms/step - loss: 0.1652 - accuracy: 0.9671 - val_loss: 0.4163 - val_accuracy: 0.8200
Epoch 10/25
66/66 [==============================] - 0s 6ms/step - loss: 0.1340 - accuracy: 0.9757 - val_loss: 0.4092 - val_accuracy: 0.8178
```

D2. Fitness of Model

The model's fitness was determined by the evaluation of the loss and accuracy of the test data. The test data computed loss and accuracy metrics of 40.41% loss and 83.78% accuracy. This indicated the model fits would test data well, being that was over 80%. The loss percentage was aligned with the training and validation loss in the model training. Overall, the model was deemed to be a good fit as there was no indication of overfitting or underfitting. This was shown best within the line graphs of the model that will be included in the subsequent section.

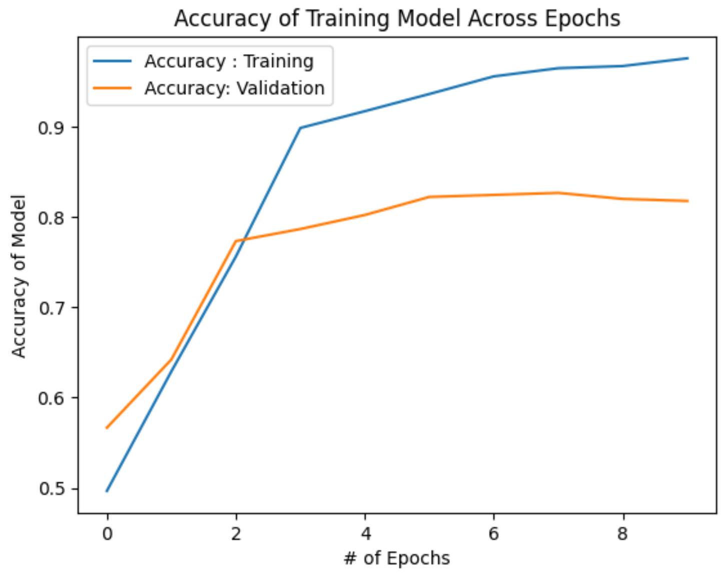
```
# evaluation of model on test data
ls, accrcy = model.evaluate(test_pad, y_test)
print('Accuracy: %f'% (accrcy*100))
```

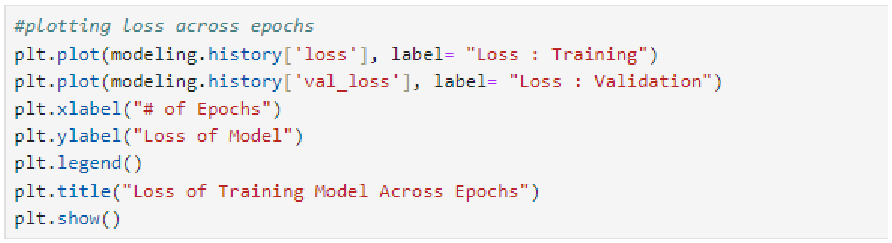
```
15/15 [==============================] - 0s 2ms/step - loss: 0.4041 - accuracy: 0.8378
Accuracy: 83.777779
```

D3. Visualizations of Model

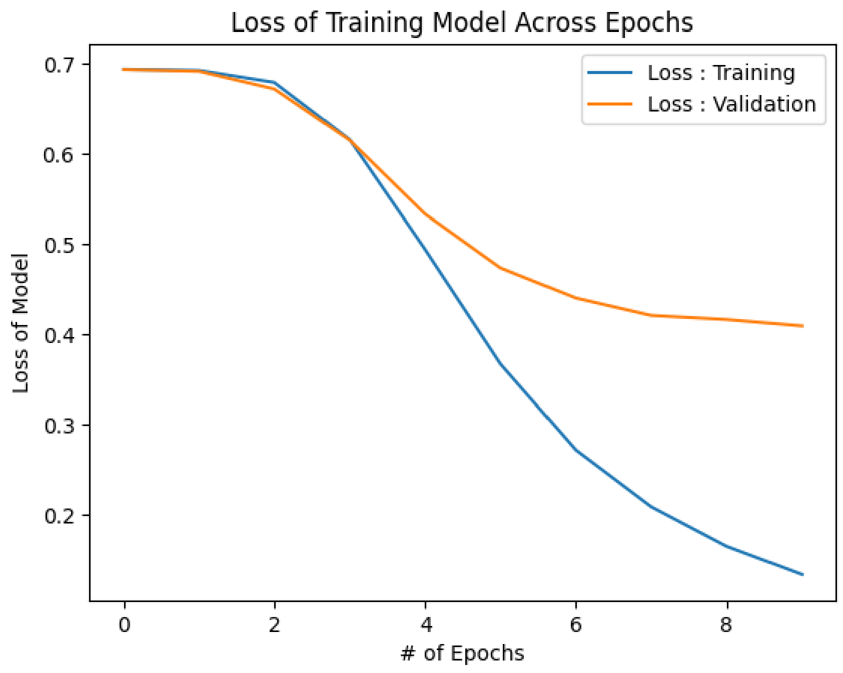
See below for the visualization of the model's training process. This included the line graph showing the evaluation metric of "accuracy" and the "loss" across epochs for both the training and validation sets. The line graph showed an ideal fit for accuracy as the validation and training accuracy metrics increased together while the loss metrics decreased together.

```
#plotting Accuracy across epochs
plt.plot(modeling.history['accuracy'], label= "Accuracy : Training")
plt.plot(modeling.history['val_accuracy'], label= "Accuracy: Validation")
plt.xlabel("# of Epochs")
plt.ylabel("Accuracy of Model")
plt.legend()
plt.title("Accuracy of Training Model Across Epochs")
plt.show()
```



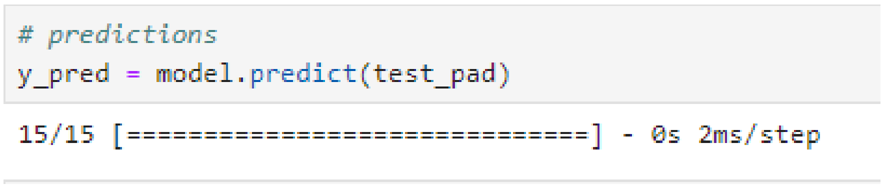


```
#plotting loss across epochs
plt.plot(modeling.history['loss'], label= "Loss : Training")
plt.plot(modeling.history['val_loss'], label= "Loss : Validation")
plt.xlabel("# of Epochs")
plt.ylabel("Loss of Model")
plt.legend()
plt.title("Loss of Training Model Across Epochs")
plt.show()
```



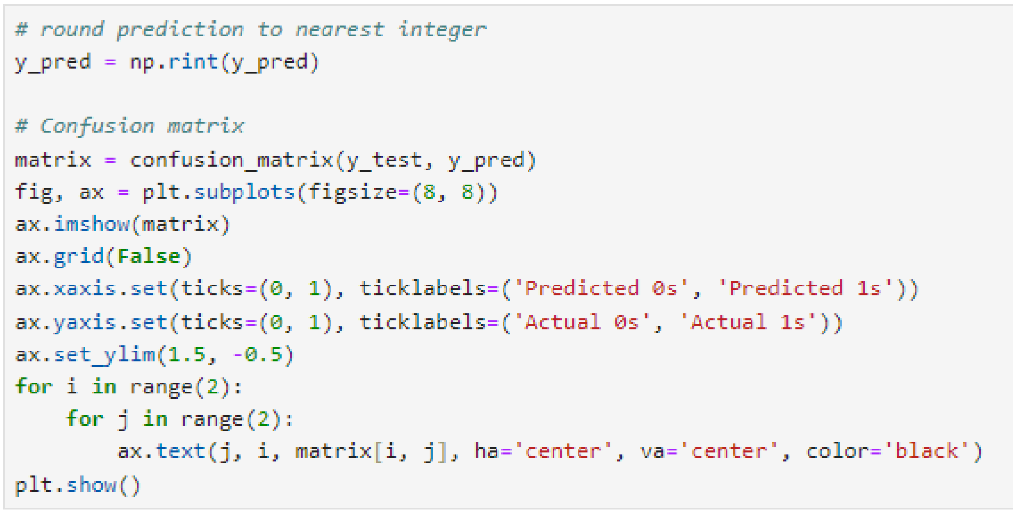
D4. Predictive Accuracy

The trained predictive accuracy was demonstrated within a confusion matrix to further evaluate the model of our test data. The confusion matrix showed that we have a total of 377 accurate predictions and 80 incorrect predictions. The matrix showed there was a total of 221 positive sentiments. The model was able to accurately predict the positive sentiments at 82.80 %, which was pretty good. The calculation of this metric was provided. (183/221 = 0.8280) It was slightly higher in predicting negative reviews at 84.71%. The calculation of this metric was provided. (194/229=0.8471) Overall, the model had an accuracy of around 82-84% regardless of whether it predicted negative or positive sentiment.

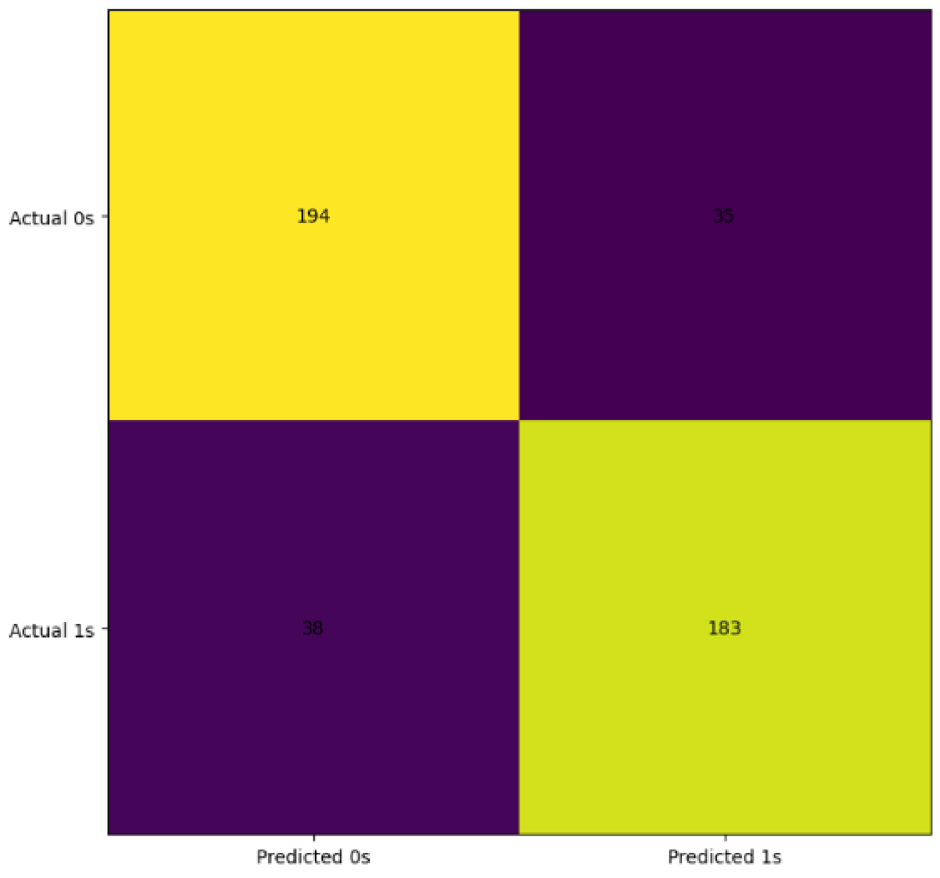


```
# predictions
y_pred = model.predict(test_pad)
```

```
15/15 [==============================] - 0s 2ms/step
```



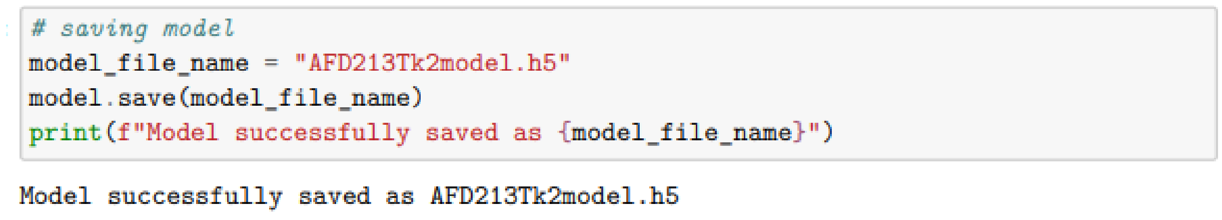
```
# round prediction to nearest integer
y_pred = np.rint(y_pred)

# Confusion matrix
matrix = confusion_matrix(y_test, y_pred)
fig, ax = plt.subplots(figsize=(8, 8))
ax.imshow(matrix)
ax.grid(False)
ax.xaxis.set(ticks=(0, 1), ticklabels=('Predicted 0s', 'Predicted 1s'))
ax.yaxis.set(ticks=(0, 1), ticklabels=('Actual 0s', 'Actual 1s'))
ax.set_ylim(1.5, -0.5)
for i in range(2):
    for j in range(2):
        ax.text(j, i, matrix[i, j], ha='center', va='center', color='black')
plt.show()
```



Part V: Data Summary and Recommendations

E. Trained Network Code

The trained network with the neural network was saved and included within the submission as “AFD213Tk2model.h5”.



```
# saving model
model_file_name = "AFD213Tk2model.h5"
model.save(model_file_name)
print(f"Model successfully saved as {model_file_name}")
```

```
Model successfully saved as AFD213Tk2model.h5
```

F. Functionality of Neural Network

The functionality of the neural network was somewhat effective at performing sentiment analysis. It achieved an accuracy of about 84% on the testing data. The loss value was ok at around 40%, which aligned with the training and validation modeling. It had a somewhat similar predicting accuracy regardless of whether predicting negative or positive sentiment. Overall, it did provide a firm response to the proposed research question of “Can neural networks and NLP techniques be used to accurately predict a consumer’s sentiment based on written historical reviews?”. The answer was a resounding yes.

The impact of the network architecture, recurrent neural networks, allowed for the detection of the sentiment to be accurately predicted. The usage of recurrent allowed for the retention of previous inputs to model problems. This was completed within the Dense “hidden” layers of the model. These layers reduced the 73 input sequence length down to a single value of “0” or “1”, which provided the needed classification.

G. Recommended Course of Action

The recommended course of action would be further experimentation with the modeling to provide a higher accuracy rate in the prediction. It would identify a more successful model. Although having an 84% accuracy rate was not terrible, there was room for improvement. This could possibly be done with the use of additional Dense network layers or a larger vocabulary size or dimensional value. Another key action could be further experimentation with data cleaning processes to provide a cleaner more accurate prepared data set. This would be especially helpful in the use of Sentiment Analysis.

Part VI: Reporting

F. Industry Relevant Interactive Report

Please reference included copy of executed Jupyter Notebook in PDF format: “AFCodeD213Tk2.PDF”

G. Third-Party Web Sources

Elleh, Festus. (2022, May 15). *Advanced Data Analytics - Task 2* [Video]. College of Information Technology. Western Governors University. https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=cedbd86a-2543-4d9d-9b0e-aec4011a606d

Sewell, William. (n.d.). *D213 SA Webinar 4[Video]*. College of Information Technology. Western Governor University. https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=c2642337-7947-4e7c-aa8d-af7b01456876

Unknown. (2017, November 28). Correcting Words using Python and NLTK. Retrieved from Text Mining Backyard: https://rustyonrampage.github.io/text-mining/2017/11/28/spelling-correction-with-python-and-nltk.html

H. References

Becker, D. (n.d.). *Introduction to Deep Learning in Python*. Retrieved from DataCamp: https://app.datacamp.com/learn/courses/introduction-to-deep-learning-in-python

Brownlee, J. (2020, August 20). *A Gentle Introduction to the Rectified Linear Unit (ReLU)*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

Brownlee, J. (2021, January 13). *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

Cecchini, D. (n.d.). *Recurrent Neural Networks (RNN) for Language Modeling in Python*. Retrieved from DataCamp: https://campus.datacamp.com/courses/recurrent-neural-networks-rnn-for-language-modeling-in-python/recurrent-neural-networks-and-keras?ex=1

Hull, I. (n.d.). *Loss Function*. Retrieved from DataCamp: https://campus.datacamp.com/courses/introduction-to-tensorflow-in-python/63343?ex=4

IBM. (n.d.). *What are recurrent neural networks?* Retrieved from IBM: https://www.ibm.com/topics/recurrent-neural-networks

Jarmul, K. (n.d.). *Introduction to Natural Language Processing in Python*. Retrieved from DataCamp: https://app.datacamp.com/learn/courses/introduction-to-natural-language-processing-in-python

Kotzias, D. (2015). *Sentiement Labelled Sentences*. Retrieved from UC Irvine Machine Learning Repository: https://doi.org/10.24432/C57604

Saxena, S. (2020, October 3). *Understanding Embedding Layer in Keras*. Retrieved from Analytics Vidhya: https://medium.com/analytics-vidhya/understanding-embedding-layer-in-keras-bbe3ff1327ce

Waite, E. (2020, February 20). Why Do We use the Sigmoid Function for Binary Classification? USA. Retrieved from https://www.youtube.com/watch?v=WsFasV46KgQ

Yiu, T. (2019, December 6). *Understand RNNs (Recurrent Neural Networks*. Retrieved from Medium: https://towardsdatascience.com/understanding-rnns-recurrent-neural-networks-479cd0da9760