Sentiment Analysis to Indicate Customer Satisfaction

D214 Performance Assessment (NKM2) Task 2
Corey B. Holstege
November 27, 2023

Table of Contents

Research Question

What's in a word? Potentially much when it's a social media post that could go viral.

The internet was invented in 1983 (Board of Regents) and social media took off in 2003 with the launch of Myspace(Wikimedia Foundation, 2023, *Timeline of social media*). With those events the number of words exploded. According to Influence MarketingHub there are currently over 116 unique social media platforms with over 4.89 billion users – that is a lot of words.

What does this mean for today's consumer company? There are more ways than ever for customers to interact with the businesses they spend their money with. With all these different platforms how are companies to know if their customers are satisfied or upset with them overall? Afterall, to keep customers coming back and making further purchases they need to keep their customers happy. A key metric companies track is the customer satisfaction score (CSAT). This metric is calculated by taking a count of all of reviews where the company was rated a 4 or 5 (satisfied or very satisfied) divided by the total number of reviews, multiplied by 100 to turn it into a percent. The benchmark for fast food restaurants for CSAT is 76% (SurveyMonkey).

How do companies take all these reviews, in text format, and turn it into a CSAT? This is where neural networks for sentiment analysis shine. Like all machine learning algorithms, neural networks require a training data set – a set of data (reviews) that are already classified as "satisfied" or "dissatisfied" to train the model. Fortunately, companies already have this – anyplace customers leave reviews with one-to-five-star rating. One to three stars can be considered dissatisfied, with four and five stars as satisfied. These reviews could be left on their own website, on products they sell, or on third-party websites such as Consumer Reports. Once a model is completed that can accurately predict an input (review) as "satisfied" or "dissatisfied", the companies can then take all of their reviews from all of the social media platforms they are on and enter then into the model to classify the review. After that, it's simply calculating the CSAT score.

This is the focus of this project: can a neural network model be constructed on the dataset to accurately predict customer reviews as positive or negative, allowing these predictions to be used to calculate a customer satisfaction score?

Success of the project will be measured by accepting either the null hypothesis or the alternative hypothesis. The null hypothesis is that a neural network *cannot* be constructed from the dataset to accurately predict customer reviews as positive or negative. The alternative hypothesis is that a neural network *can* be constructed from the dataset to accurately predict customer reviews as positive or negative with an accuracy greater than eighty percent (80%).

Data Collection

The dataset used to train the neural network is the publicly available McDonald's Store Reviews dataset on Kaggle. The dataset was retrieved on October 24, 2023 and contained 33,396 rows. The dataset was created and is maintained by Nidula Elgiriyewithana and is updated annually, per the dataset metadata. The dataset contains anonymized reviews of McDonald's locations in the United States from scraped Google reviews.

One advantage of this dataset is it is prepared and ready for analysis.

One disadvantage of this dataset is that there is no insight into how the data was collected or the timeframe of the data. Most of the reviews are two-twelve years old – more recent reviews would have been beneficial for the model.

The dataset has the following columns:

Column Name	Description[1]	Type
reviewer_id	Unique identifier for each reviewer (anonymized)	Integer, non-repeating
store_name	Name of the McDonald's store	Categorical: Nominal; String
category	Category or type of the store	Categorical: Nominal; String
store_address	Address of the store	Categorical: Nominal; String
latitude	Latitude coordinate of the store's location	Numeric: Continuous; Float
longitude	Longitude coordinate of the store's location	Numeric: Continuous; Float
rating_count	Number of ratings/reviews for the store	Numeric: Discrete; Integer
review_time	Timestamp of the review	Numeric: Interval; String
review	Textual content of the review	Categorical: Nominal; String
rating	Rating provided by the reviewer	Categorical: Ordinal; String

The dataset used validate the model is a small sample of thirty-three reviews manually collected from Facebook, Instagram, LinkedIn, and Pinterest. The trained and tested model was executed on this dataset and predicted the sentiment as positive or negative.

One advantage to this is reviews to be used could be targeted ensuring a robust sample – short and long reviews, reviews with special characters, positive and negative sentiment.

One disadvantage to this is only an extremely small sample size was able to be collected due to the manual method of data gathering.

The dataset has the following columns:

Column Name	Description	Type
review_id	Unique identifier for each review	Integer, non-repeating
source	Which social media platform the review was collected from	Categorical: Nominal; String
date	Date of the review on the social medial platform	Numeric: Interval; String
review	Text of the review	Categorical: Nominal; String

Gathering the data was straightforward and no specific challenges were encountered.

[1] Descriptions are as described on the Kaggle dataset page: https://www.kaggle.com/datasets/nelgiriyewithana/mcdonalds-store-reviews

Data Extraction and Preparation

The main dataset was downloaded from Kaggle, as described above, to a csv file. This csv file was then imported to a Pandas Dataframe, df, in section 2 of the code.

In section 3 of the code we examine the shape of the Dataframe, determining the data contains 10 columns, and 33,396 rows. This confirms that all data was imported.

```
In [337]: print('Number of rows/columns: ' + str(df.shape), '\n')
Number of rows/columns: (33396, 10)
```

In section 3.1 we examine the columns of the Dataframe. Here it is noted that the latitude and longitude columns are missing values, while all other columns have 33,396 values. It is also noted that two columns are decimal (float64), one column is a whole number (int64), and seven columns are objects.

```
In [338]: print('View column info: \n', df.info(), '\n')
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 33396 entries, 0 to 33395
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   reviewer_id    33396 non-null  int64
 1   store_name     33396 non-null  object
 2   category       33396 non-null  object
 3   store_address  33396 non-null  object
 4   latitude       32736 non-null  float64
 5   longitude      32736 non-null  float64
 6   rating_count   33396 non-null  object
 7   review_time    33396 non-null  object
 8   rating         33396 non-null  object
 9   review         33396 non-null  object
dtypes: float64(2), int64(1), object(7)
memory usage: 2.5+ MB
```

In section 3.2 we viewed the statistical information of the columns. Here initial observations are noted:

- Store_name only has two values
- Category only has one value
- store_address has forty unique values
- rating_count has fifty-one unique values. Per the data dictionary this is a unique count of reviews per store. As there are forty unique stores, there should be forty unique values here. Having more than forty unique values means some stores have more than one unique count of reviews
- revie_time is not a datetime field as it should be. Instead, it is a string with values such as “4 years ago”. This will need to be cleaned/preprocessed.
- Rating is not an integer field as it should be. Instead, it is a string with values such as “5 stars”. This will need to be cleaned/preprocessed.
- Review has 22,285 unique values. This means some reviews are exactly the same. The most occurring review is “Excellent”, which occurs 2,148 times.

df_describe - DataFrame

Index	reviewer_id	store_name	category	store_address	latitude	longitude	rating_count	review_time	rating	review
count	33396	33396	33396	33396	32736	32736	33396	33396	33396	33396
unique	nan	2	1	40	nan	nan	51	39	5	22285
top	nan	McDonald's	Fast food restaurant	9814 International Dr, Orlando, FL 32819, United States	nan	nan	2,810	4 years ago	5 stars	Excellent
freq	nan	33325	33396	1890	nan	nan	1140	6740	10274	2148
mean	16698.5	nan	nan	nan	34.4425	-90.647	nan	nan	nan	nan
std	9640.74	nan	nan	nan	5.34412	16.5948	nan	nan	nan	nan
min	1	nan	nan	nan	25.7903	-121.995	nan	nan	nan	nan
25%	8349.75	nan	nan	nan	28.6553	-97.7929	nan	nan	nan	nan
50%	16698.5	nan	nan	nan	33.9313	-81.4714	nan	nan	nan	nan
75%	25047.2	nan	nan	nan	40.7274	-75.3999	nan	nan	nan	nan
max	33396	nan	nan	nan	44.9814	-73.4598	nan	nan	nan	nan

In section 3.3 we double check for missing values and confirm the only columns missing values are latitude and longitude, and both columns are missing 660 values.

```
In [339]: print(missing_df)
     column_name  any_missing  total_missing
0    reviewer_id        False              0
1     store_name        False              0
2       category        False              0
3  store_address        False              0
4       latitude         True            660
5      longitude         True            660
6   rating_count        False              0
7    review_time        False              0
8         rating        False              0
9         review        False              0
```

Exploratory Data Analysis: columns excluding review

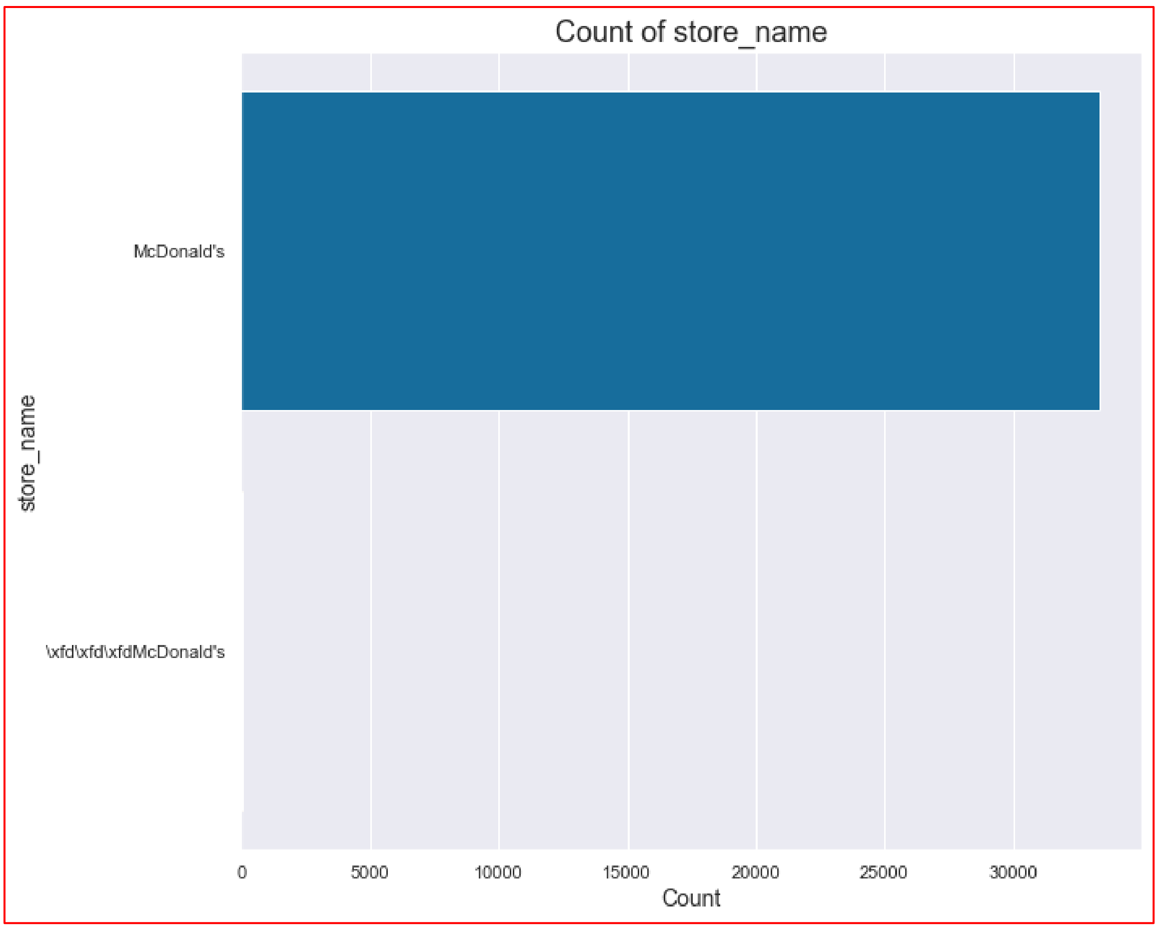
In section 4 we complete exploratory data analysis on all columns excluding the "review" column. A function, eda_analysis, is defined. This function will be used several times through the code file and does the following:

- Prints the name of the column being analyzed
- Prins the number of unique values in the column using Pandas nunique method
- Prints the datatype of the column using Pandas dtypes method
- Prints the statistical information of the column using Pandas .describe method
- Creates a frequency table. The index column is the name of the column with the unique values of the column. The first column is the count with the second column being the percentage of total. The frequency table is stored as a dictionary within another dictionary for use outside of the function.
- Prints a countplot (horizontal bar chart) of the unique count of values in the column.

Column: store_name

Function eda_analysis is executed on column store_name with the following results:

```
In [341]: print(eda_analysis(df, 'store_name'))
--------------------------------------------------------------------------
Begin store_name:
Number of unique values:  2

Datatype of the column:  object

Describe store_name
count           33396
unique              2
top         McDonald's
freq            33325
Name: store_name, dtype: object


Frequency table for variable store_name :
                         Count  Percentages
store_name
McDonald's               33325        99.79
\xfd\xfd\xfdMcDonald's      71         0.21

End store_name
```



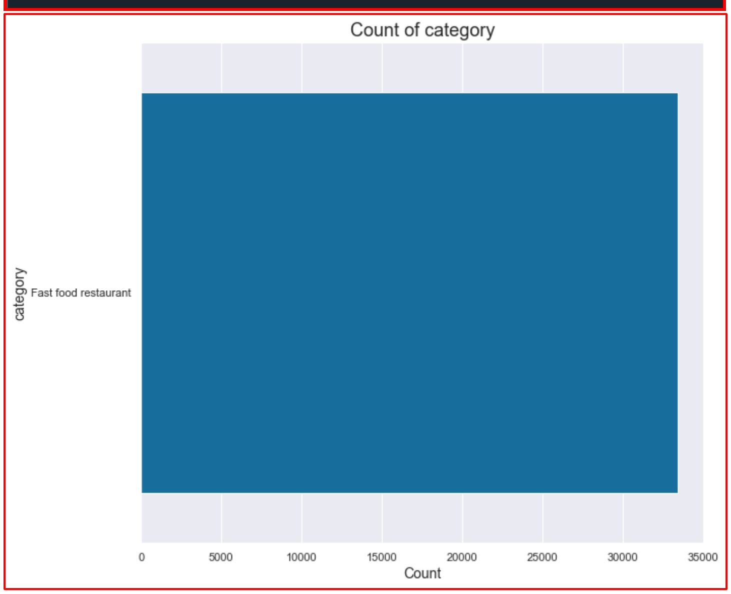
The following facts are noted:

- The column has a datatype of "object". This is logical as the values stored are strings (text)
- There are 33,396 values. Confirming what was observed above – there are not missing values for this column
- There are two unique values for this column:
  - McDonald's
  - \xfd\xfd\xfdMcDonald's
- The second unique value appears to have some characters from the data scaping process.
- If this column is to be used in analysis, these characters would need to be removed, leaving the column with one unique value.
- As this column only has one unique value (if cleaned) this column provides no value to the analysis

Column: category

Function eda_analysis is executed on column category with the following results:

```
In [342]: print(eda_analysis(df, 'category'))
-------------------------------------------------------------
Begin category:
Number of unique values:  1

Datatype of the column:  object

Describe category
count                        33396
unique                           1
top         Fast food restaurant
freq                         33396
Name: category, dtype: object


Frequency table for variable category :
                       Count  Percentages
category
Fast food restaurant  33396        100.0

End category
```



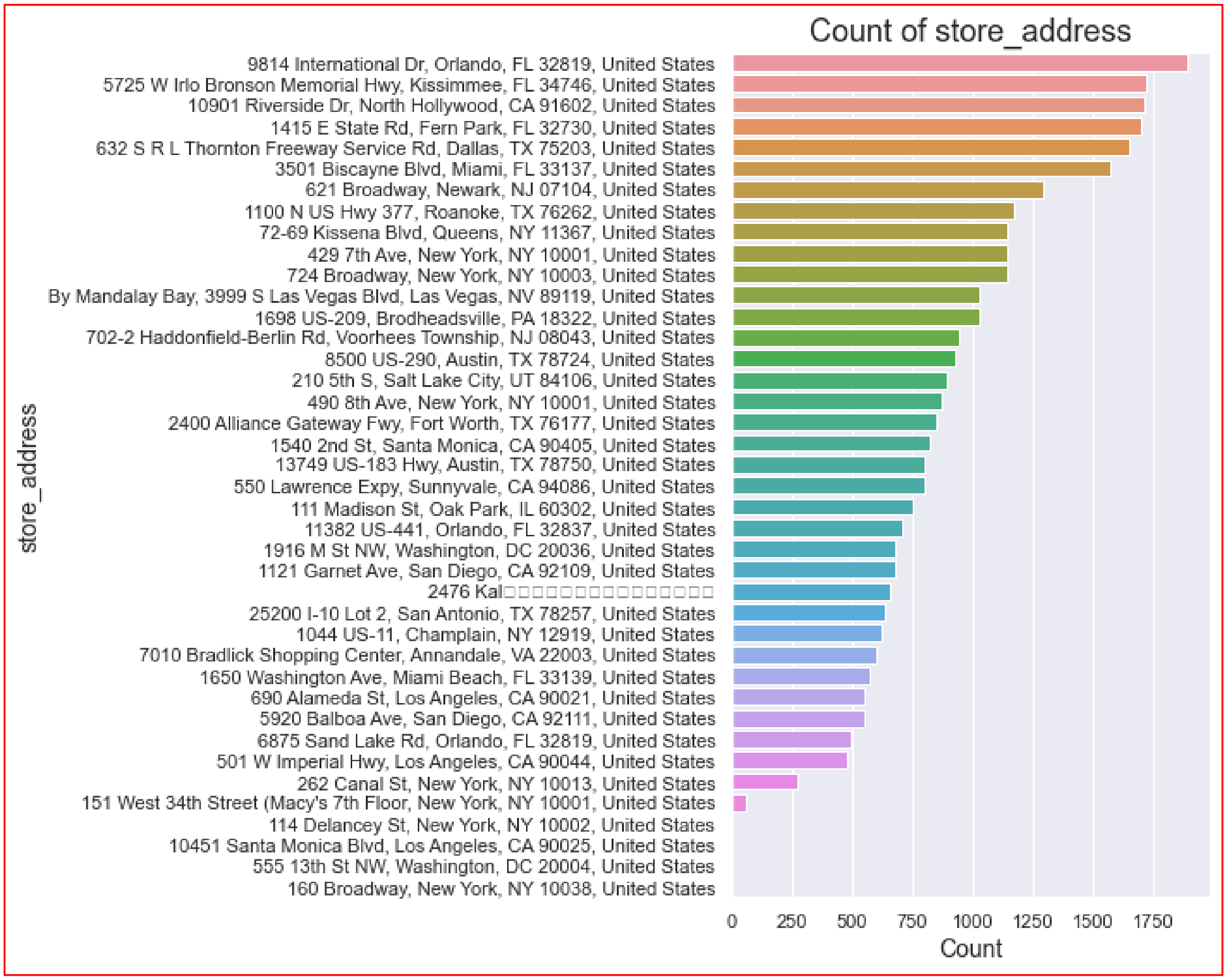
The following facts are noted:

- The column has a datatype of "object". This is logical as the values stored are strings (text)
- There are 33,396 values. Confirming what was observed above – there are not missing values for this column
- There is one unique values for this column: Fast food restaurant
- As this column only has one unique value this column provides no value to the analysis

Column: store_address

Function eda_analysis is executed on column store_address with the following results:

```
In [343]: print(eda_analysis(df, 'store_address'))
----------------------------------------------------------------------------
Begin store_address:
Number of unique values:  40

Datatype of the column:  object

Describe store_address
count                                                  33396
unique                                                    40
top       9814 International Dr, Orlando, FL 32819, Unit...
freq                                                    1890
Name: store_address, dtype: object


Frequency table for variable store_address :
                                                     Count  Percentages
store_address
9814 International Dr, Orlando, FL 32819, Unite...   1890         5.66
5725 W Irlo Bronson Memorial Hwy, Kissimmee, FL...   1720         5.15
10901 Riverside Dr, North Hollywood, CA 91602, ...   1710         5.12
1415 E State Rd, Fern Park, FL 32730, United St...   1700         5.09
632 S R L Thornton Freeway Service Rd, Dallas, ...   1650         4.94
3501 Biscayne Blvd, Miami, FL 33137, United States   1570         4.70
621 Broadway, Newark, NJ 07104, United States        1290         3.86
1100 N US Hwy 377, Roanoke, TX 76262, United St...   1168         3.50
72-69 Kissena Blvd, Queens, NY 11367, United St...   1140         3.41
429 7th Ave, New York, NY 10001, United States       1140         3.41
724 Broadway, New York, NY 10003, United States      1140         3.41
By Mandalay Bay, 3999 S Las Vegas Blvd, Las Veg...   1030         3.08
1698 US-209, Brodheadsville, PA 18322, United S...   1028         3.08
702-2 Haddonfield-Berlin Rd, Voorhees Township,...    943         2.82
8500 US-290, Austin, TX 78724, United States          926         2.77
210 5th S, Salt Lake City, UT 84106, United States    890         2.66
490 8th Ave, New York, NY 10001, United States        870         2.61
2400 Alliance Gateway Fwy, Fort Worth, TX 76177...    850         2.55
1540 2nd St, Santa Monica, CA 90405, United States    820         2.46
13749 US-183 Hwy, Austin, TX 78750, United States     800         2.40
550 Lawrence Expy, Sunnyvale, CA 94086, United ...    800         2.40
111 Madison St, Oak Park, IL 60302, United States     751         2.25
11382 US-441, Orlando, FL 32837, United States        710         2.13
1916 M St NW, Washington, DC 20036, United States     680         2.04
1121 Garnet Ave, San Diego, CA 92109, United St...    680         2.04
2476 Kal���������������                                660          1.98
25200 I-10 Lot 2, San Antonio, TX 78257, United...    635         1.90
1044 US-11, Champlain, NY 12919, United States        620         1.86
7010 Bradlick Shopping Center, Annandale, VA 22...    602         1.80
1650 Washington Ave, Miami Beach, FL 33139, Uni...    570         1.71
690 Alameda St, Los Angeles, CA 90021, United S...    550         1.65
5920 Balboa Ave, San Diego, CA 92111, United St...    550         1.65
6875 Sand Lake Rd, Orlando, FL 32819, United St...    490         1.47
501 W Imperial Hwy, Los Angeles, CA 90044, Unit...    481         1.44
262 Canal St, New York, NY 10013, United States       270         0.81
151 West 34th Street (Macy's 7th Floor, New Yor...     60         0.18
114 Delancey St, New York, NY 10002, United States      3         0.01
10451 Santa Monica Blvd, Los Angeles, CA 90025,...      3         0.01
555 13th St NW, Washington, DC 20004, United St...      3         0.01
160 Broadway, New York, NY 10038, United States         3         0.01
```



The following facts are noted:

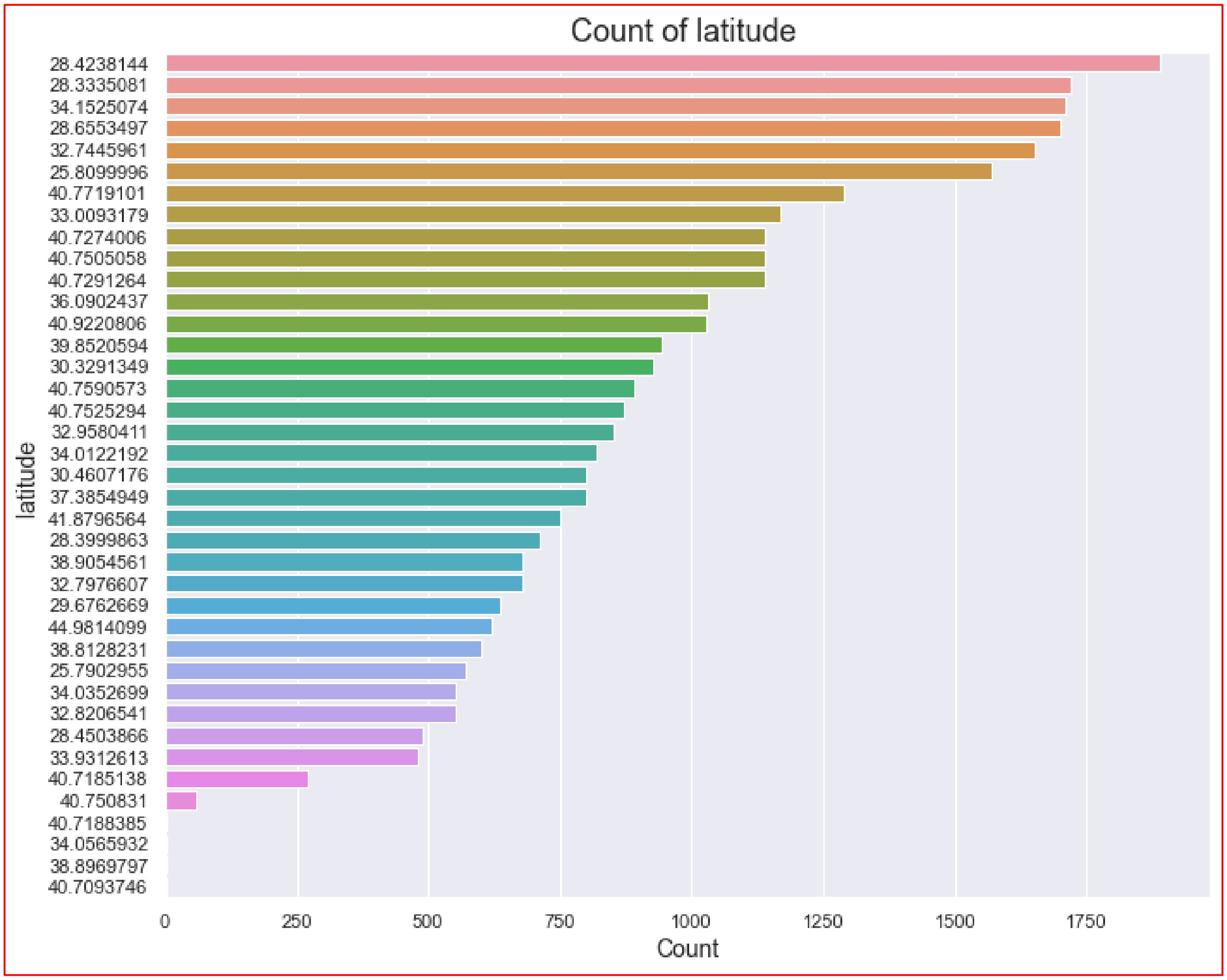
- The column has a datatype of "object". This is logical as the values stored are strings (text)
- There are 33,396 values. Confirming what was observed above – there are not missing values for this column
- There are 40 unique values
- One value occurs 1,890 times
- One address is incomplete (2476 Kal) and occurs 660 times

Column: latitude

Function eda_analysis is executed on column latitude with the following results:

```
In [344]: print(eda_analysis(df, 'latitude '))
--------------------------------------------------------------------------
Begin latitude :
Number of unique values:  39

Datatype of the column:  float64

Describe latitude
count    32736.000000
mean        34.442546
std          5.344116
min         25.790295
25%         28.655350
50%         33.931261
75%         40.727401
max         44.981410
Name: latitude , dtype: float64
```

```
Frequency table for variable latitude  :
           Count  Percentages
latitude
28.423814   1890         5.66
28.333508   1720         5.15
34.152507   1710         5.12
28.655350   1700         5.09
32.744596   1650         4.94
25.810000   1570         4.70
40.771910   1290         3.86
33.009318   1168         3.50
40.727401   1140         3.41
40.750506   1140         3.41
40.729126   1140         3.41
36.090244   1030         3.08
40.922081   1028         3.08
39.852059    943         2.82
30.329135    926         2.77
40.759057    890         2.66
40.752529    870         2.61
32.958041    850         2.55
34.012219    820         2.46
30.460718    800         2.40
37.385495    800         2.40
41.879656    751         2.25
28.399986    710         2.13
38.905456    680         2.04
32.797661    680         2.04
NaN          660         1.98
29.676267    635         1.90
44.981410    620         1.86
38.812823    602         1.80
25.790295    570         1.71
34.035270    550         1.65
32.820654    550         1.65
28.450387    490         1.47
33.931261    481         1.44
40.718514    270         0.81
40.750831     60         0.18
40.718838      3         0.01
34.056593      3         0.01
38.896980      3         0.01
40.709375      3         0.01

End latitude
```



The following facts are noted:

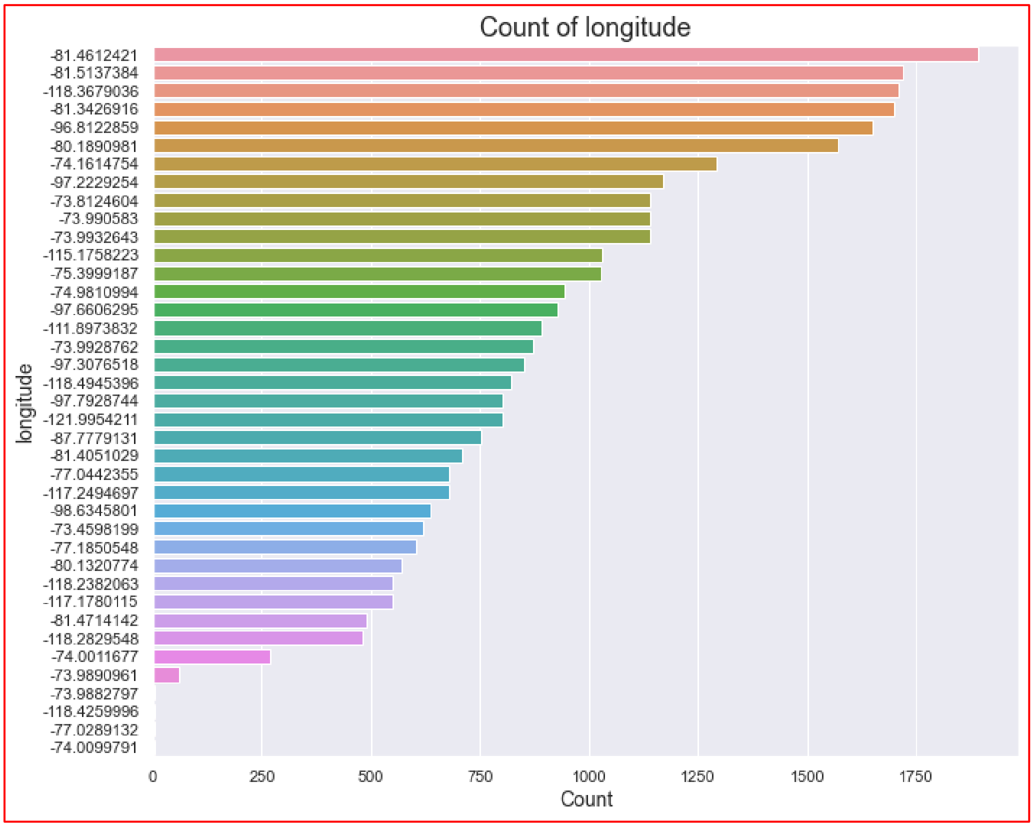
- The column has a datatype of “float64”. This is logical as the values stored are decimals
- There are 32,736 values
- There are 660 missing values

Column: longitude

Function eda_analysis is executed on column longitude with the following results:

```
In [345]: print(eda_analysis(df, 'longitude'))
------------------------------------------------------
Begin longitude:
Number of unique values:  39

Datatype of the column:  float64

Describe longitude
count    32736.000000
mean       -90.647033
std         16.594844
min       -121.995421
25%        -97.792874
50%        -81.471414
75%        -75.399919
max        -73.459820
Name: longitude, dtype: float64
```

```
Frequency table for variable longitude :
             Count  Percentages
longitude
-81.461242    1890         5.66
-81.513738    1720         5.15
-118.367904   1710         5.12
-81.342692    1700         5.09
-96.812286    1650         4.94
-80.189098    1570         4.70
-74.161475    1290         3.86
-97.222925    1168         3.50
-73.812460    1140         3.41
-73.990583    1140         3.41
-73.993264    1140         3.41
-115.175822   1030         3.08
-75.399919    1028         3.08
-74.981099     943         2.82
-97.660629     926         2.77
-111.897383    890         2.66
-73.992876     870         2.61
-97.307652     850         2.55
-118.494540    820         2.46
-97.792874     800         2.40
-121.995421    800         2.40
-87.777913     751         2.25
-81.405103     710         2.13
-77.044235     680         2.04
-117.249470    680         2.04
 NaN           660         1.98
-98.634580     635         1.90
-73.459820     620         1.86
-77.185055     602         1.80
-80.132077     570         1.71
-118.238206    550         1.65
-117.178011    550         1.65
-81.471414     490         1.47
-118.282955    481         1.44
-74.001168     270         0.81
-73.989096      60         0.18
-73.988280       3         0.01
-118.426000      3         0.01
-77.028913       3         0.01
-74.009979       3         0.01

End longitude
```



The following facts are noted:

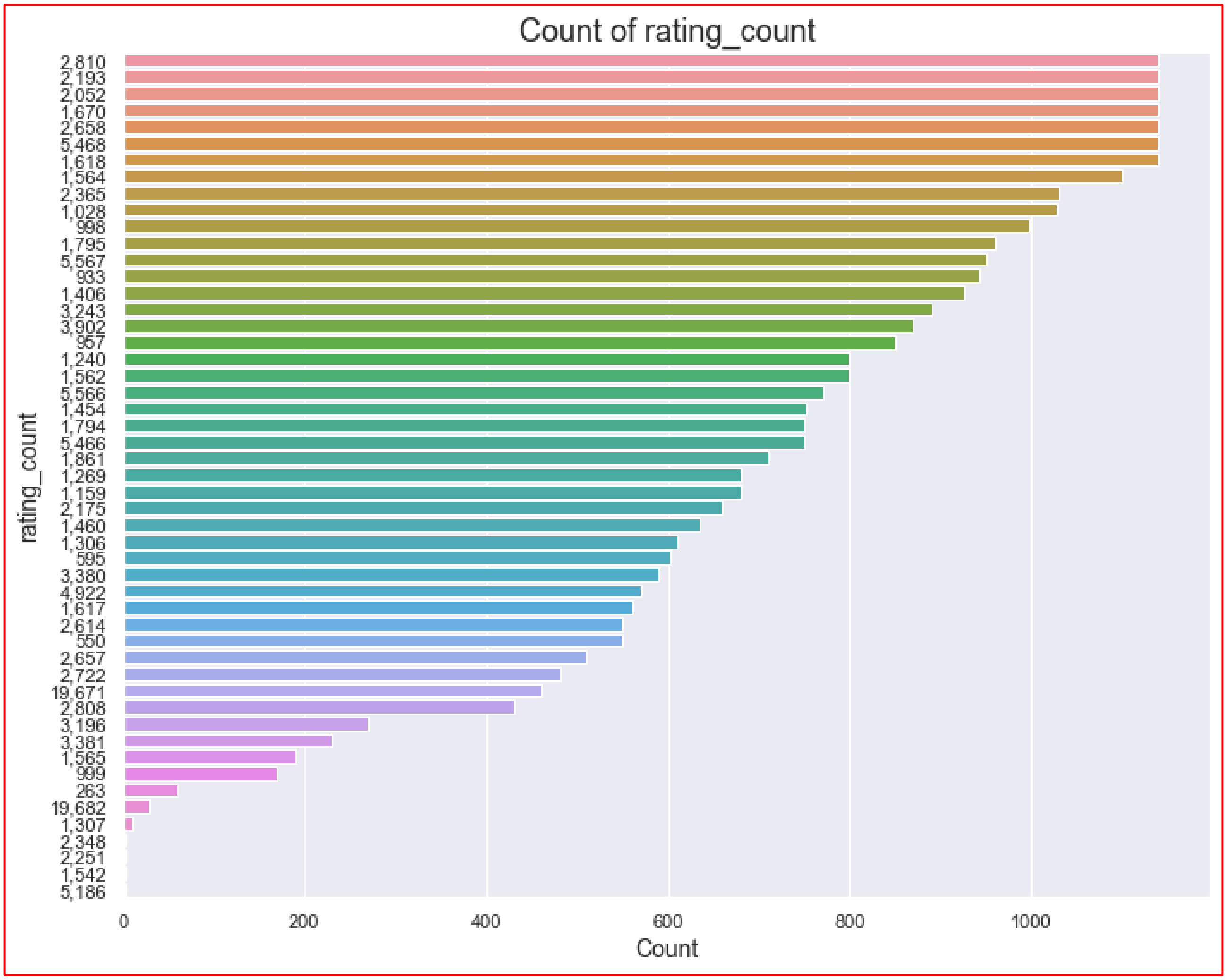
- The column has a datatype of "float64". This is logical as the values stored are decimals
- There are 32,736 values
- There are 660 missing values

Column: rating_count

Function eda_analysis is executed on column rating_count with the following results:

```
In [346]: print(eda_analysis(df, 'rating_count'))
---------------------------------------------------------
Begin rating_count:
Number of unique values:  51

Datatype of the column:  object

Describe rating_count
count      33396
unique        51
top        2,810
freq        1140
Name: rating_count, dtype: object
```

```
Frequency table for variable rating_count :
               Count  Percentages
rating_count
2,810          1140         3.41
2,193          1140         3.41
2,052          1140         3.41
1,670          1140         3.41
2,658          1140         3.41
5,468          1140         3.41
1,618          1140         3.41
1,564          1100         3.29
2,365          1030         3.08
1,028          1028         3.08
998             998         2.99
1,795           960         2.87
5,567           950         2.84
933             943         2.82
1,406           926         2.77
3,243           890         2.66
3,902           870         2.61
957             850         2.55
1,240           800         2.40
1,562           800         2.40
5,566           770         2.31
1,454           751         2.25
1,794           750         2.25
5,466           750         2.25
1,861           710         2.13
1,269           680         2.04
1,159           680         2.04
2,175           660         1.98
1,460           635         1.90
1,306           610         1.83
595             602         1.80
3,380           590         1.77
4,922           570         1.71
1,617           560         1.68
2,614           550         1.65
550             550         1.65
2,657           510         1.53
2,722           481         1.44
19,671          460         1.38
2,808           430         1.29
3,196           270         0.81
3,381           230         0.69
1,565           190         0.57
999             170         0.51
263              60         0.18
19,682           30         0.09
1,307            10         0.03
2,348             3         0.01
2,251             3         0.01
1,542             3         0.01
5,186             3         0.01

End rating_count
```



The following facts are noted:

- The column has a datatype of "object". This seems incorrect as this should be an "integer". Most likely due to the comma in the numbers. If this column will be used this will need to be cleaned.
- There are 33,396 values – no missing values
- There are 51 unique values. This seems at odds with the 40 unique values for store_address. According the data dictionary on Kaggle, this is a unique count per store. The expectation would be 40 unique values.
- The most frequent value is 2,810.

Column: review_time

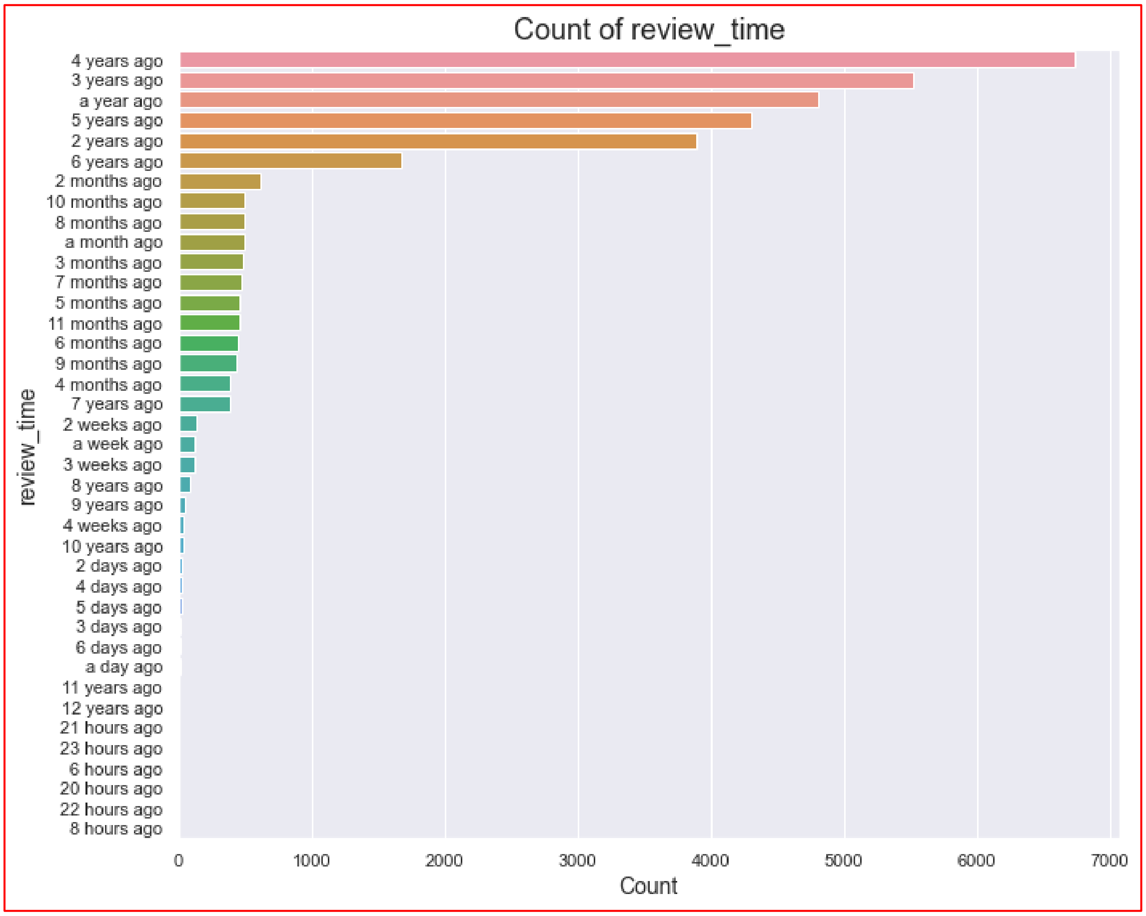
Function eda_analysis is executed on column review_time with the following results:

The following facts are noted:

- The column has a datatype of "object". This seems incorrect as this should be a "datetime" field. If this column will be used this will need to be cleaned.
- There are 33,396 values – no missing values
- There are 39 unique values
- The most frequent value is "4 years ago"
- Values range from "6 hours ago" to "12 years ago" with 80% of reviews being from one to six years ago.

```
In [347]: print(eda_analysis(df, 'review_time'))
--------------------------------------------------------------------
Begin review_time:
Number of unique values:  39

Datatype of the column:  object

Describe review_time
count              33396
unique                39
top          4 years ago
freq                6740
Name: review_time, dtype: object
```

```
Frequency table for variable review_time :
                Count  Percentages
review_time
4 years ago     6740        20.18
3 years ago     5522        16.53
a year ago      4809        14.40
5 years ago     4306        12.89
2 years ago     3892        11.65
6 years ago     1679         5.03
2 months ago     625         1.87
10 months ago    503         1.51
8 months ago     498         1.49
a month ago      493         1.48
3 months ago     492         1.47
7 months ago     472         1.41
5 months ago     458         1.37
11 months ago    457         1.37
6 months ago     456         1.37
9 months ago     444         1.33
4 months ago     393         1.18
7 years ago      387         1.16
2 weeks ago      137         0.41
a week ago       125         0.37
3 weeks ago      122         0.37
8 years ago       91         0.27
9 years ago       52         0.16
4 weeks ago       47         0.14
10 years ago      38         0.11
2 days ago        32         0.10
4 days ago        26         0.08
5 days ago        23         0.07
3 days ago        22         0.07
6 days ago        17         0.05
a day ago         17         0.05
11 years ago      10         0.03
12 years ago       4         0.01
21 hours ago       2         0.01
23 hours ago       1         0.00
6 hours ago        1         0.00
20 hours ago       1         0.00
22 hours ago       1         0.00
8 hours ago        1         0.00

End review_time
```



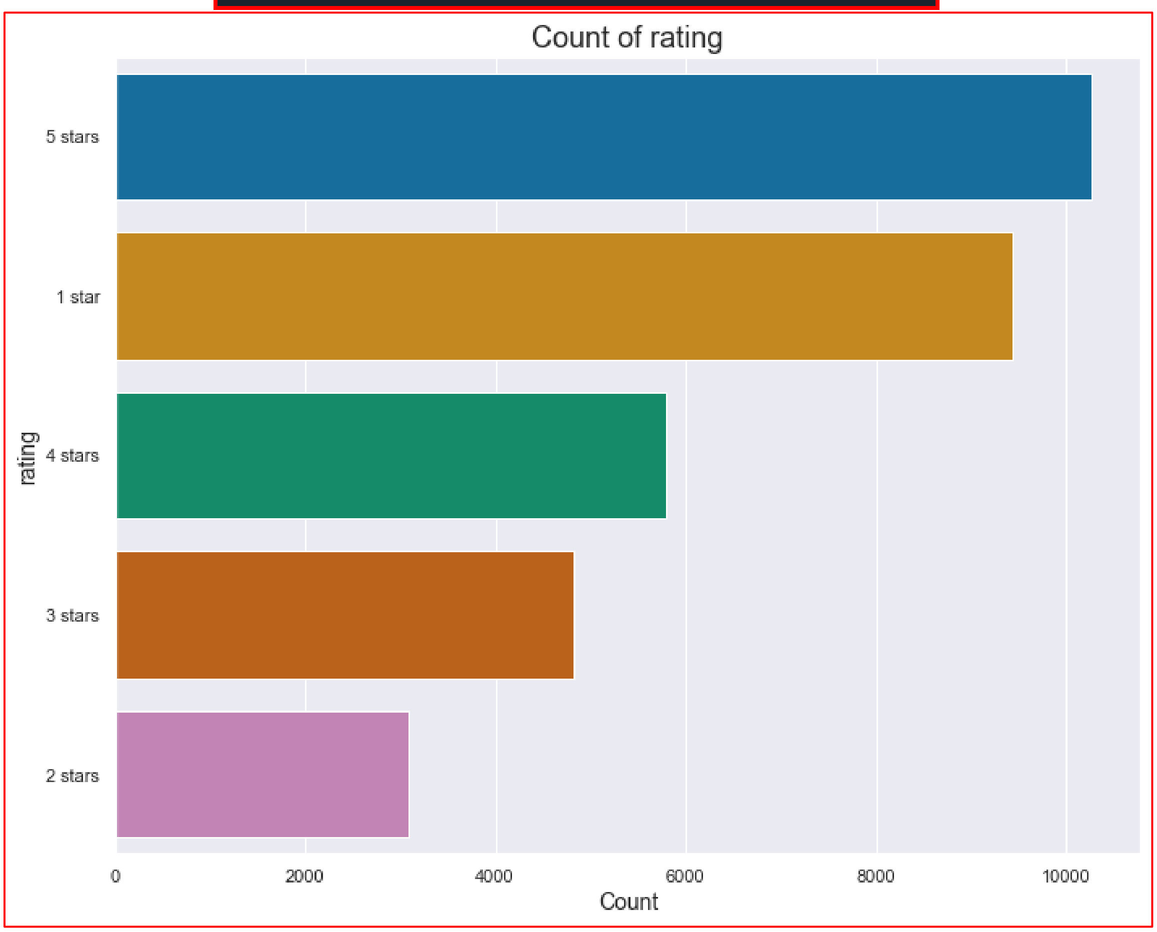
Column: rating

Function eda_analysis is executed on column rating with the following results:

The following facts are noted:

- The column has a datatype of "object". This seems incorrect as this should be a "integer" field. If this column will need to be cleaned.
- There are 33,396 values – no missing values
- There ae 5 unique values
- The most frequent value is "5 stars", occurring 10,274 times.

```
In [348]: print(eda_analysis(df, 'rating'))
-------------------------------------------------
Begin rating:
Number of unique values:  5

Datatype of the column:  object

Describe rating
count        33396
unique           5
top        5 stars
freq         10274
Name: rating, dtype: object


Frequency table for variable rating :
          Count  Percentages
rating
5 stars  10274        30.76
1 star    9431        28.24
4 stars   5787        17.33
3 stars   4818        14.43
2 stars   3086         9.24

End rating
```



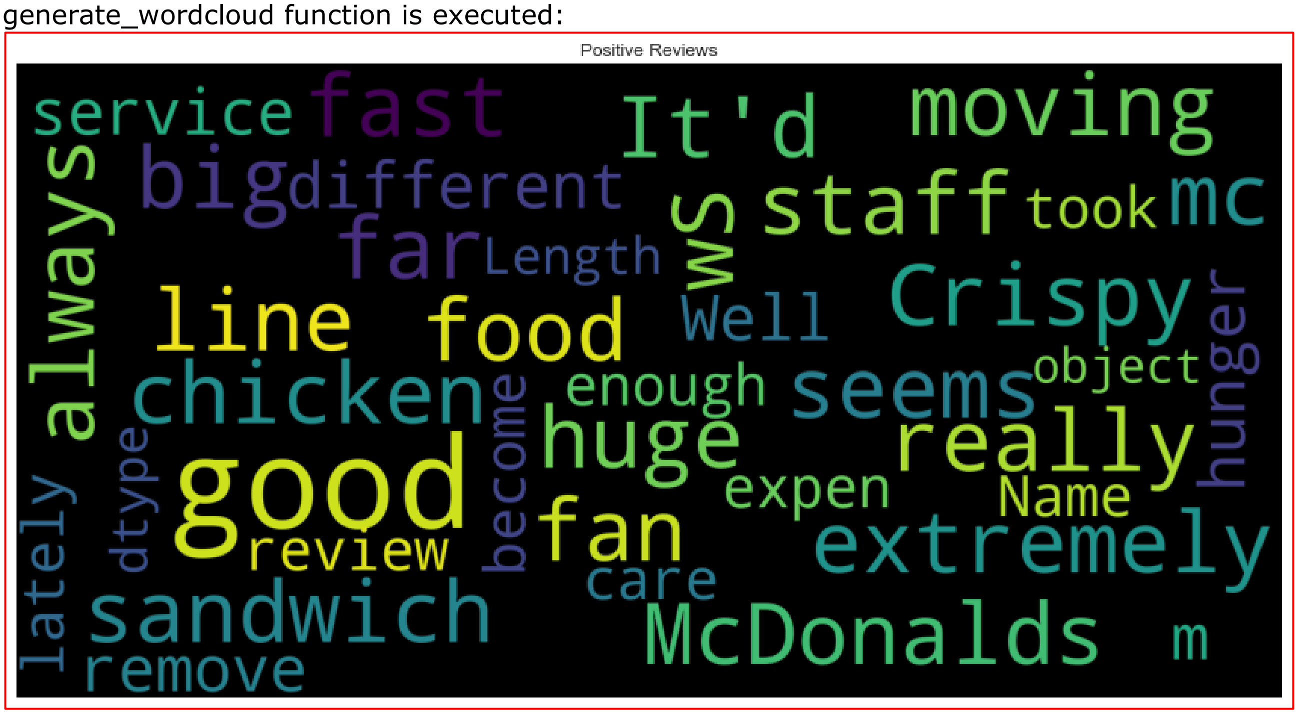
Exploratory Data Analysis: column review

In section 5 we complete exploratory data analysis on the review column. First function generate_wordcloud is define. This function takes as input a Dataframe and column and returns a wordcloud image.

Wordcloud of the entire "review" column:

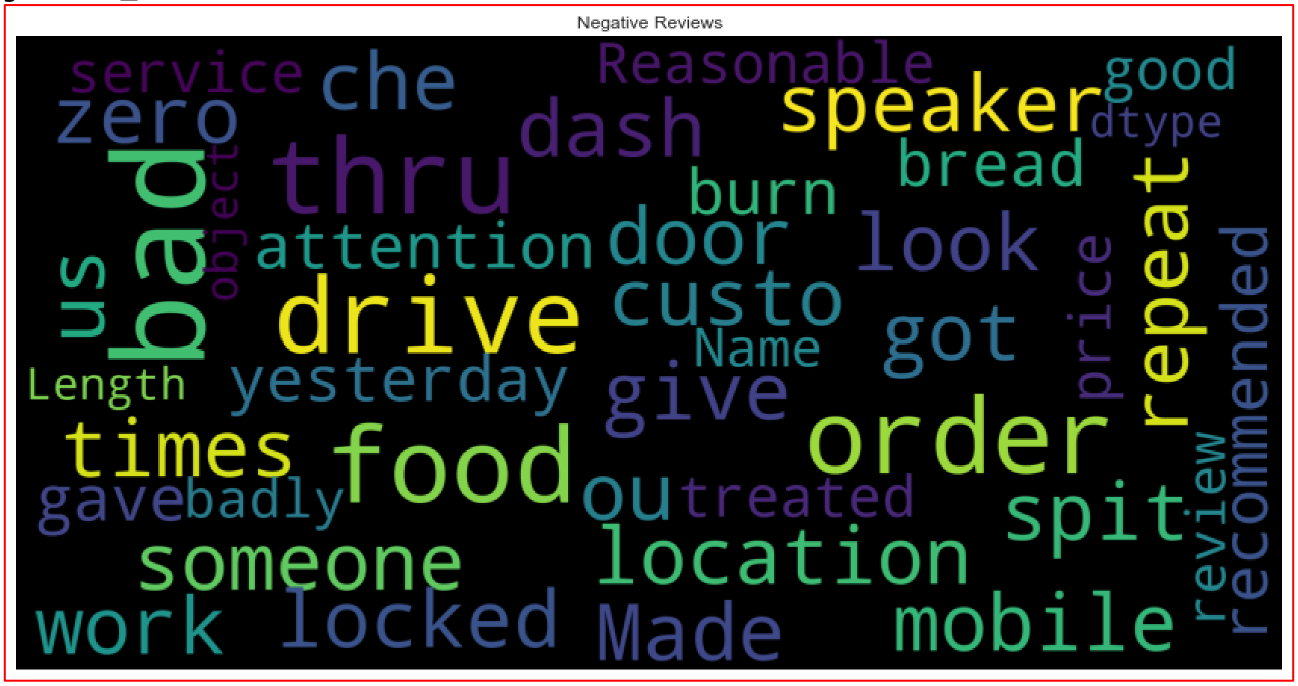
It is noted that there are positive words (good, service), negative words (spit, far), and words that do not appear to be positive or negative (sandwich, review).

Next, the positive reviews are separated out into their own Dataframe df_positive. Positive reviews are defined as reviews that have a "rating" of "4 stars" or "5 stars". The generate_wordcloud function is executed:



More positive words are noted (good, food), but still many words that do not appear positive or negative.

Next, the negative reviews are separated out into their own Dataframe df_negative. Negative reviews are defined as reviews that have a “rating” or “1 star”, “2 stars”, or “3 stars”. The generate_wordcloud function is executed:



In section 5.2 we check for special characters. Three functions are defined to assist in this analysis:

- character_count: takes a Dataframe as input and returns a bar chart

```
def character_chart(df, column_x, column_y):
    """Return a bar chart of chararacter counts."""
    plt.figure(figsize=(12, 6))  # Adjust the figure size as needed
    plt.bar(df[column_x], df[column_y])
    #plt.barh(df[column_x], df[column_y])
    plt.xlabel('Character')
    plt.ylabel('Count')
    plt.title('Character Counts')
    plt.show()
    plt.close()
```

- extract_unique_character: takes a Dataframe and column as input, and extracts a list of unique characters

```
def extract_unique_characters(df, column_name):
    """
    # Citation Dr.Ellah Festus D213 Task 2 Cohort Webinar (converted into a function)
    # function to input a dataframe and column name and output a list of unique characters found in the specified column
    """
    list_of_characters = []
    for review in df[column_name]:
        for character in review:
            if character not in list_of_characters:
                list_of_characters.append(character)
    return list_of_characters
```

- extract_special_character_counts_df: takes a Dataframe and column as input and returns a Dataframe with a unique list of characters and a count of how many times they appear

```python
def extract_special_character_counts_df(df, column_name):
    """extract all special characters and a count of how many times the speical character appears
       store in a dataframe
    """
    special_character_counts = {}
    for review in df[column_name]:
        for character in review:
            if character not in string.ascii_letters:
                if character not in special_character_counts:
                    special_character_counts[character] = 1
                else:
                    special_character_counts[character] += 1

    # Convert the dictionary to a DataFrame
    df_special_character_counts = pd.DataFrame(list(special_character_counts.items()), columns=['character', 'count'])
    return df_special_character_counts
```

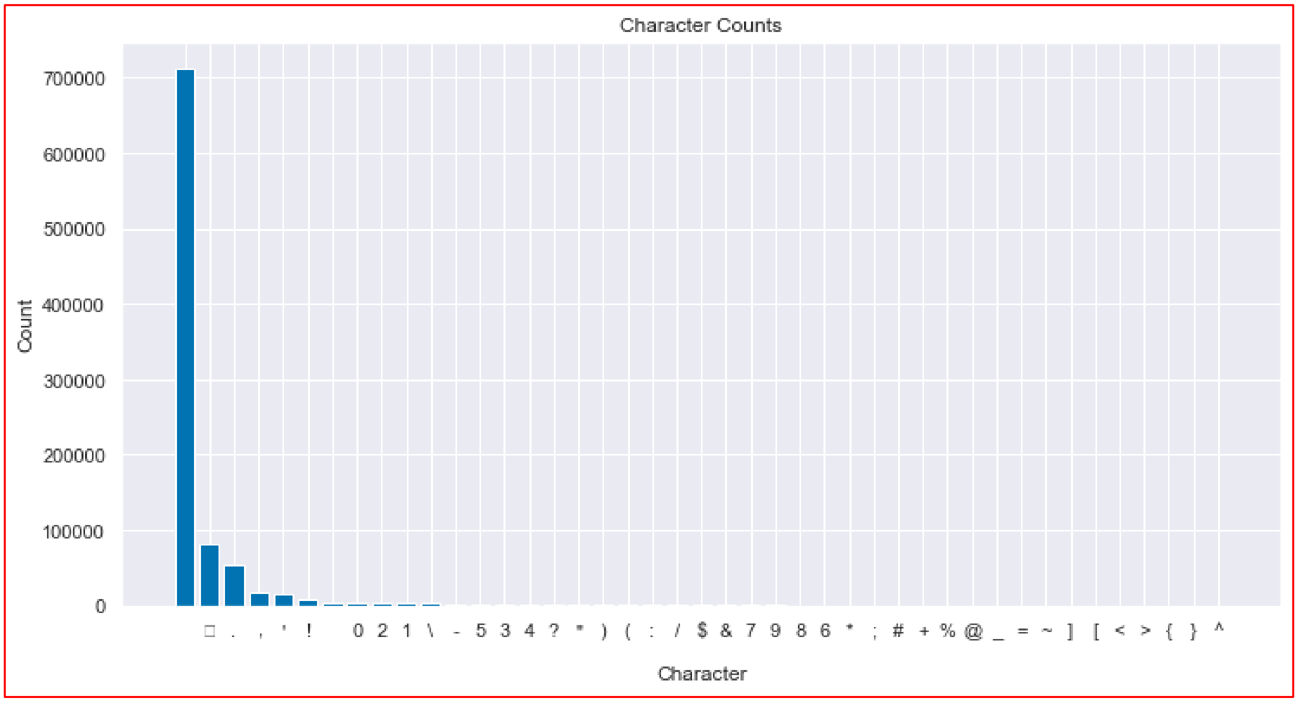
Executing extract_unique_characters the following unique characters are noted in the “review” column:

```
In [352]: print(extract_unique_characters(df, 'review'))
['W', 'h', 'y', ' ', 'd', 'o', 'e', 's', 'i', 't', 'l', 'k', 'm', 'n', 'p', 'f', '?', '\n', 'I', 'a', 'r', 'c', ',',
'v', 'w', 'b', 'u', '.', 'g', '/', '*', "'", 'M', 'D', 'T', 'x', 'L', '�', 'N', '\\', 'C', 'q', '3', 'E', '1', '0',
'j', 'z', 'P', '8', ':', '7', 'O', '#', '2', '5', '"', 'A', 'F', 'G', 'V', 'Y', 'S', 'B', 'H', 'U', 'J', '&', '-',
'!', '4', 'R', '6', '(', ')', '9', 'X', 'K', 'Q', '%', '+', '$', 'Z', ';', '[', '_', '~', '@', '=', '<', ']', '>',
'{', '}', '^']
```

It is noted that there are upper case, lower case, and special characters (period, commas, pound signs, dollar signs, etc.). Special characters will need to be removed, and upper case will need to be converted to lower case. This will reduce the number of possible tokens to be considered in the model.

Next we execute function extract_speical_character_counts_df to create a Dataframe of special characters and a count of how many times the character appears, we sort the Dataframe values from highest to lower, and then we execute function character_chart to plot a bar graph of the counts:

```
In [353]: df_special_character_counts = extract_special_character_counts_df(df, 'review')  # store in a dataframe

In [354]: df_special_character_counts = df_special_character_counts.sort_values(by='count', ascending=False)  # sort
the values in the dataframe

In [355]: character_chart(df_special_character_counts, 'character', 'count')  # generate the bar chart
```



A list list_special_characters is created to store the unique special characters to be used later in data cleaning:

```
In [356]: list_special_characters = df_special_character_counts['character'].tolist()

In [357]: print(list_special_characters)
[' ', '�', '.', ',', "'", '!', '\n', '0', '2', '1', '\\', '-', '5', '3', '4', '?', '"', ')', '(', ':', '/', '$', '&',
'7', '9', '8', '6', '*', ';', '#', '+', '%', '@', '_', '=', '~', ']', '[', '<', '>', '{', '}', '^']
```

Returning to the bar chart, it is noted that there are over 700,000 spaces (first column). The count of spaces is stored at index 0 in dataframe df_special_character_count. This row is removed to obtain a better view of the rest of the special characters in the bar chart:

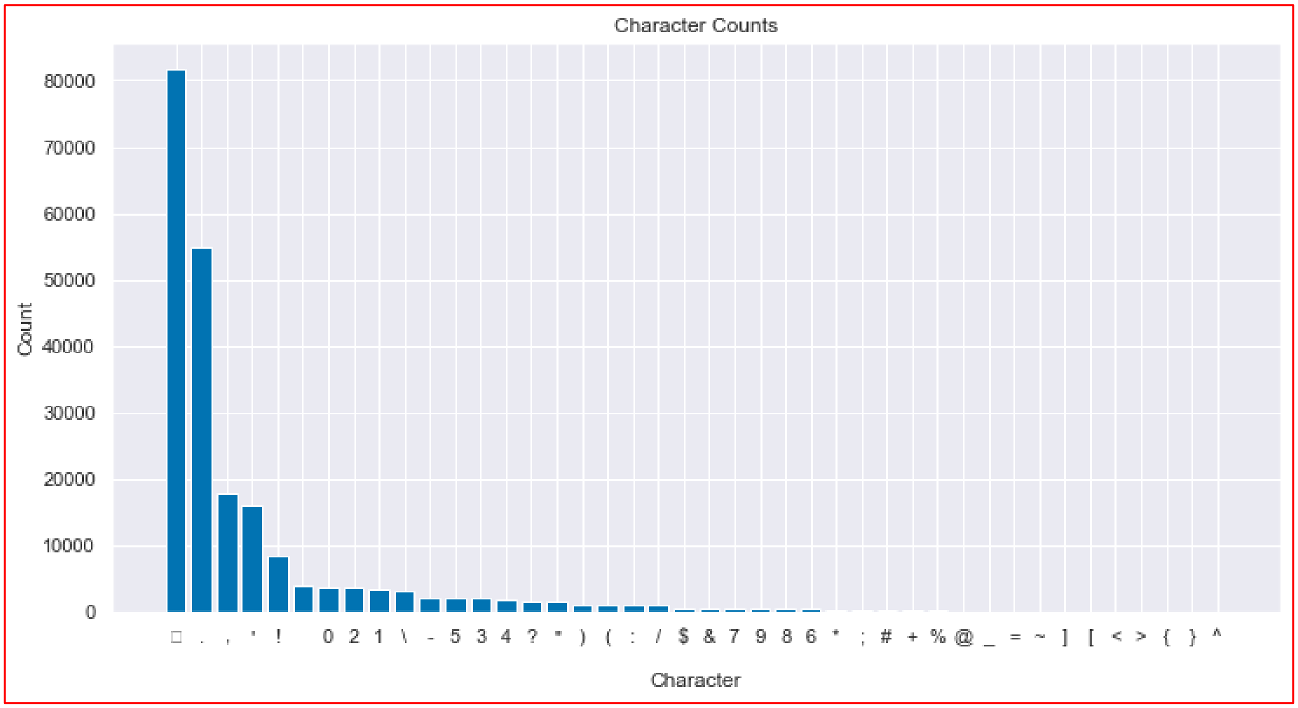
df_special_character_counts - DataFrame

Index	character	count
0		712141
8	�	81684
4	.	54838
3	,	17722
7	'	16073
22	!	8460
2		3735
12	0	3727

```
In [358]: df_special_character_counts = df_special_character_counts.drop(index=0)  # drop the row for space

In [359]: character_chart(df_special_character_counts, 'character', 'count')  # generate the bar chart
```

df_special_character_counts - DataFrame

Index	character	count
8	◆	81684
4	.	54838
3	,	17722
7	'	16073
22	!	8460
2		3735
12	0	3727
17	2	3587



It is observed that there are over 80,000 occurrences of square special character. The same steps are repeated to remove this from Dataframe df_special_character_counts to obtain a better view of the rest of the special characters:

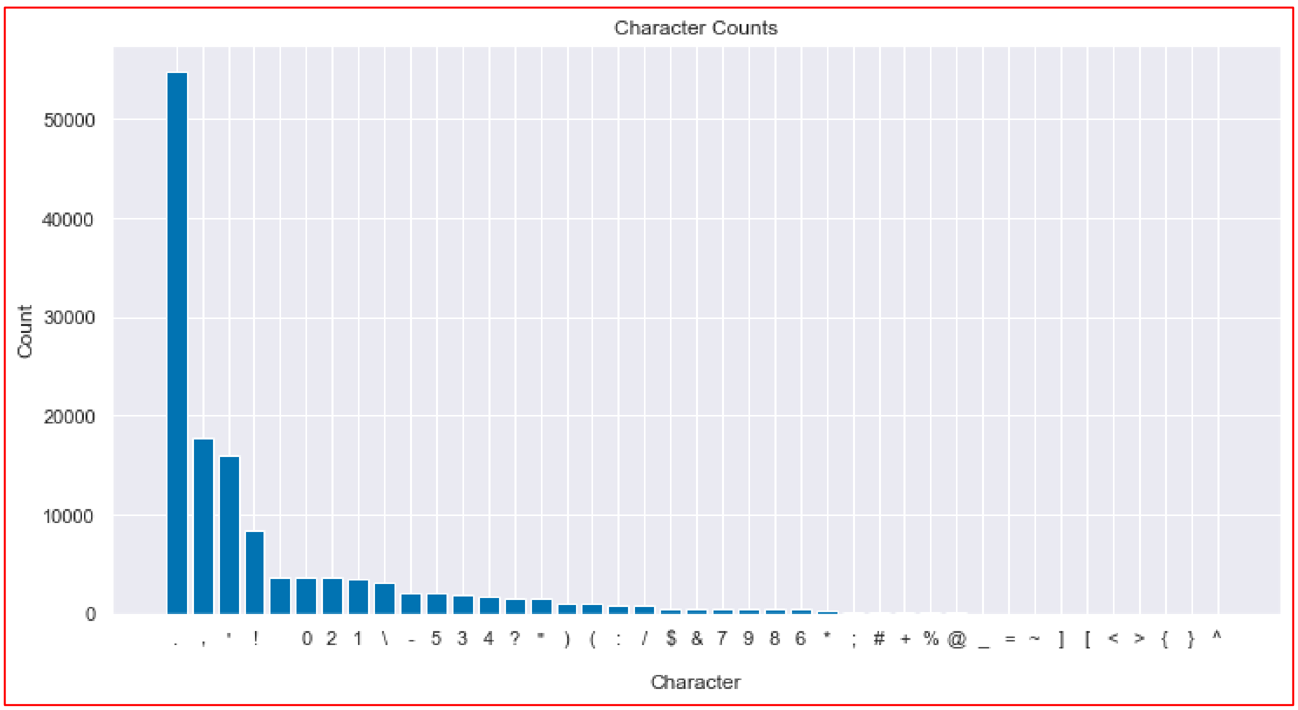
df_special_character_counts - DataFrame

Index	character	count
8	◆	81684
4	.	54838
3	,	17722
7	'	16073
22	!	8460

```
In [360]: df_special_character_counts = df_special_character_counts.drop(index=8)  # drop the row for weird character

In [361]: character_chart(df_special_character_counts, 'character', 'count')  # generate the bar chart
```

df_special_character_counts - DataFrame

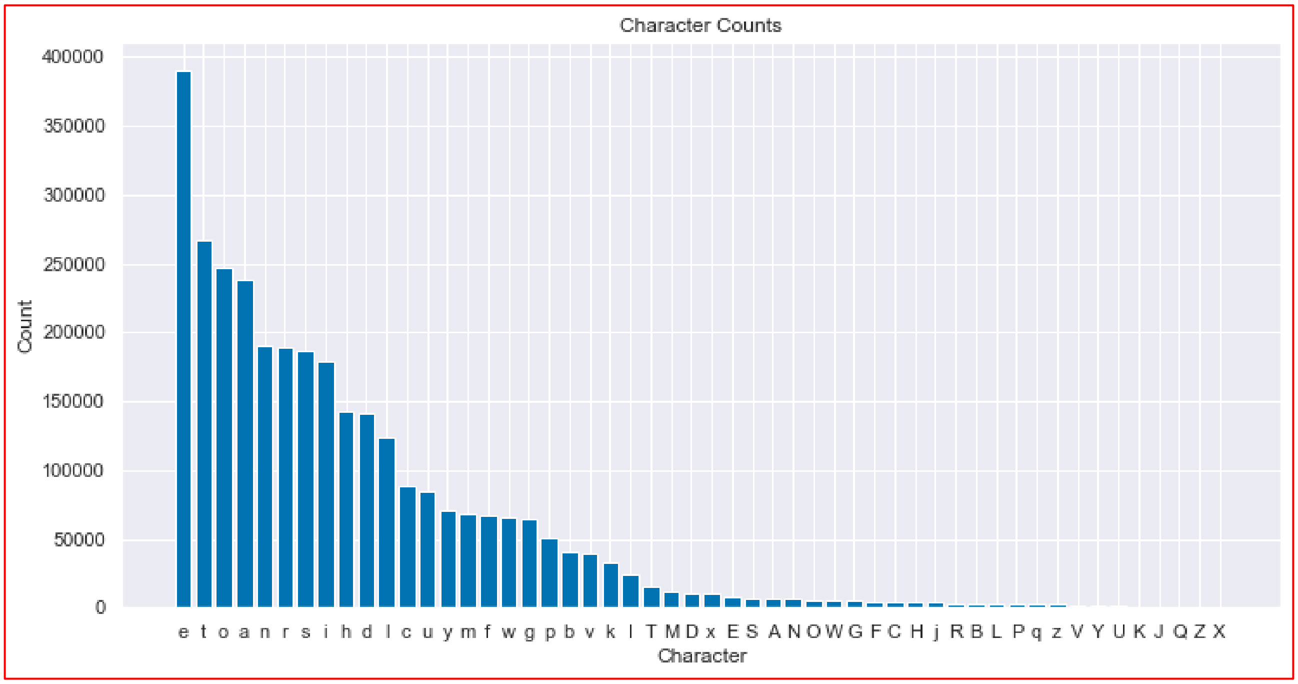
Index	character	count
4	.	54838
3	,	17722
7	'	16073
22	!	8460
2		3735
12	0	3727
17	2	3587



It is observed that periods at the most occurring special character – logical as this is the most common punctuation used to end sentences or thoughts – followed by commas and asterisks.

In section 5.4 we examine the letters in the review column. Function extract_alphabet_counts_df that takes as input a Dataframe and column name. This function determines a unique list of letters and counts how many times the letter appears, storing the result in a Dataframe.

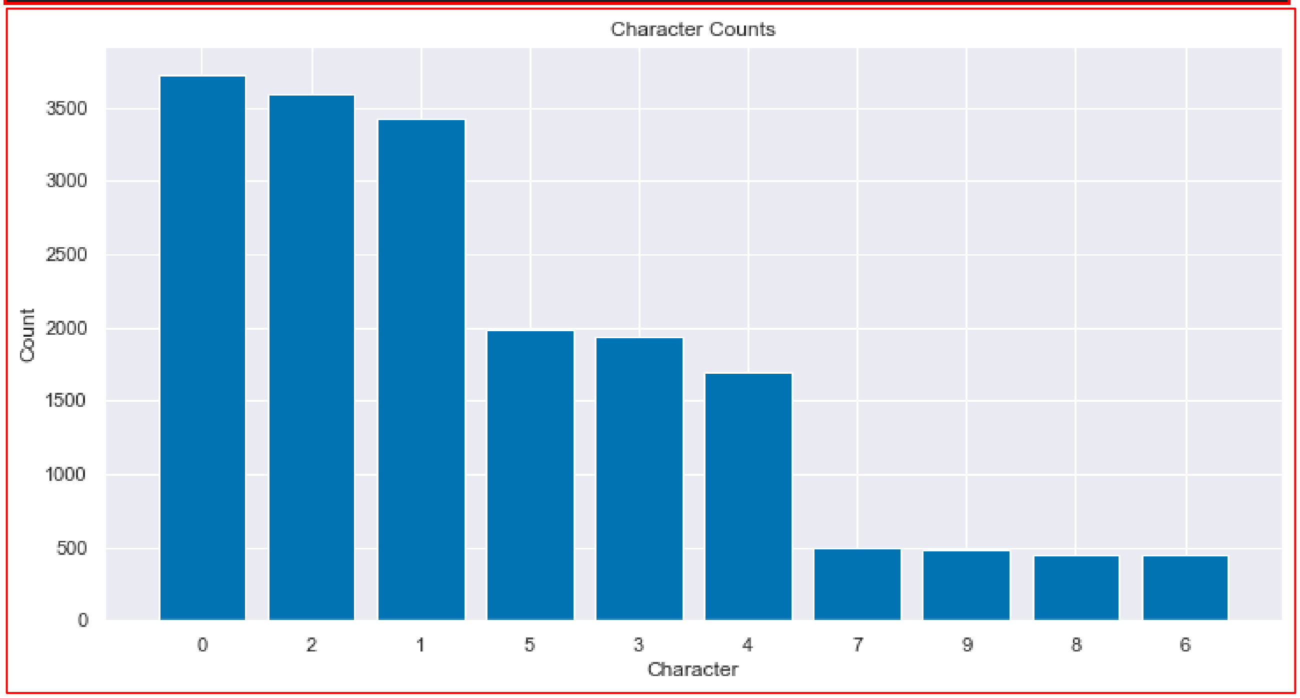
```
In [365]: def extract_alphabet_counts_df(df, column_name):
     ...:     """extract all alphabet characters and a count of how many times the letter appears
     ...:        store in a dataframe
     ...:     """
     ...:     character_counts = {}
     ...:     for review in df[column_name]:
     ...:         for character in review:
     ...:             if character in string.ascii_letters:
     ...:                 if character in character_counts:
     ...:                     character_counts[character] += 1
     ...:                 else:
     ...:                     character_counts[character] = 1
     ...:
     ...:     df_alphabet_counts = pd.DataFrame(list(character_counts.items()), columns=['character', 'count'])
     ...:     return df_alphabet_counts

In [366]: df_alphabet_counts = extract_alphabet_counts_df(df, 'review')  # store in a dataframe

In [367]: df_alphabet_counts = df_alphabet_counts.sort_values(by='count', ascending=False)  # sort the values in the dataframe

In [368]: character_chart(df_alphabet_counts, 'character', 'count')  # generate the bar chart
```



The most common letter occurring is lowercase 'e' occurring almost 400,000 times.

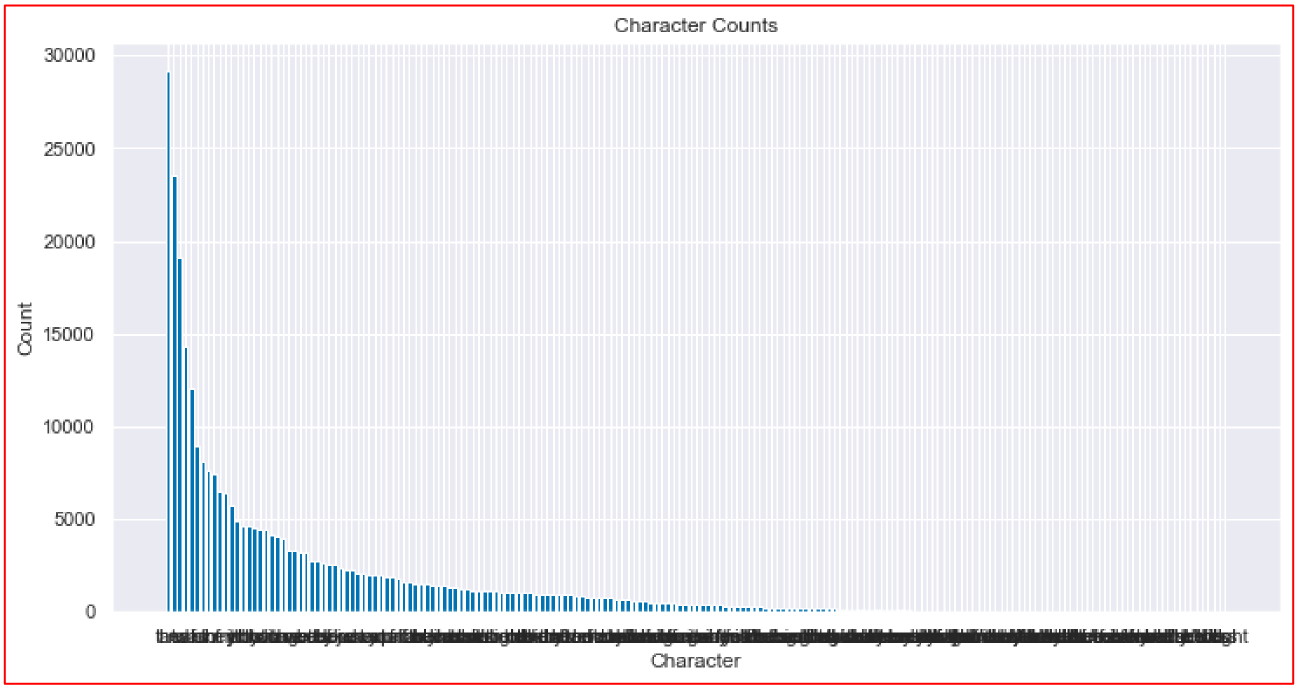
In section 5.5 we examine the numbers in the review column. Function extract_number_counts_df that takes as input a Dataframe and column name. This function determines a unique list of numbers and counts how many times the number appears, storing the result in a Dataframe.

```
In [369]: def extract_numbers_counts_df(df, column_name):
     ...:     """extract all numbers and a count of how many times the number appears
     ...:        store in a dataframe
     ...:     """
     ...:     number_counts = {}
     ...:     for review in df[column_name]:
     ...:         for number in review:
     ...:             if number in string.digits:
     ...:                 if number in number_counts:
     ...:                     number_counts[number] += 1
     ...:                 else:
     ...:                     number_counts[number] = 1
     ...:
     ...:     df_number_counts = pd.DataFrame(list(number_counts.items()), columns=['character', 'count'])
     ...:     return df_number_counts

In [370]: df_number_counts = extract_numbers_counts_df(df, 'review')  # store in a dataframe

In [371]: df_number_counts = df_number_counts.sort_values(by='count', ascending=False)  # sort the values in the dataframe

In [372]: character_chart(df_number_counts, 'character', 'count')  # generate the bar chart
```



In section 5.6 we examine the stopwords in the review column. Stopwords are common words that exist and are used, but offer little meaning in determining if the text is positive or negative sentiment. Function extract_stopwords_counts_df that takes as input a Dataframe and column name. This function determines a unique list of stopwords and counts how many times the stopword appears, storing the result in a Dataframe.

```
In [373]: def extract_stopwords_counts_df(df, column_name):
     ...:     """extract all stop words and a count of how many times the stop word appears
     ...:        store in a dataframe
     ...:     """
     ...:     stop_word_count = {}
     ...:     for review in df[column_name]:
     ...:         words = review.split()
     ...:         for word in words:
     ...:             if word in STOPWORDS:
     ...:                 if word in stop_word_count:
     ...:                     stop_word_count[word] += 1
     ...:                 else:
     ...:                     stop_word_count[word] = 1
     ...:
     ...:     df_stopword_counts = pd.DataFrame(list(stop_word_count.items()), columns=['character', 'count'])
     ...:     return df_stopword_counts

In [374]: df_stopword_counts = extract_stopwords_counts_df(df, 'review')  # store in a dataframe

In [375]: df_stopword_counts = df_stopword_counts.sort_values(by='count', ascending=False)  # sort the values in the dataframe

In [376]: character_chart(df_stopword_counts, 'character', 'count')  # generate the bar chart. better as a barh chart

In [377]: print('There are', df_stopword_counts['count'].sum(), 'stopwords.')
There are 297359 stopwords.
```



In all, there are 297,359 stopwords used across 33,396 reviews. The most common stopwords are "the", "and", and "to":

df_stopword_counts - DataFrame

Index	character	count
21	the	29217
8	and	23494
11	to	19112
6	a	14286
7	was	12031

In section 5.7 we analyze how many words are contractions. Function extract_apostrophe_counts_df is defined. This function takes as input a Dataframe and column and looks for words with apostrophe's, and counts them.

```
In [378]: def extract_apostrophe_counts_df(df, column_name):
     ...:     apostrophe_count = {}
     ...:
     ...:     for review in df[column_name]:
     ...:         words = review.split()  # Tokenize the text into words
     ...:
     ...:         for word in words:
     ...:             if "'" in word:  # Check if the word contains an apostrophe
     ...:                 if word in apostrophe_count:
     ...:                     apostrophe_count[word] += 1
     ...:                 else:
     ...:                     apostrophe_count[word] = 1
     ...:
     ...:     df_apostrophe_counts = pd.DataFrame(list(apostrophe_count.items()), columns=['word', 'count'])
     ...:     return df_apostrophe_counts

In [379]: df_apostrophe_counts = extract_apostrophe_counts_df(df, 'review')  # store in a dataframe

In [380]: df_apostrophe_counts = df_apostrophe_counts.sort_values(by='count', ascending=False)  # sort the values in the dataframe
```

Here is a sample:

df_apostrophe_counts - DataFrame

Index	word	count
12	McDonald's	3149
2	don't	1265
22	it's	1196
3	It's	1060
10	didn't	977
17	I've	763
1	I'm	651
15	McDonald's.	564
35	can't	367
41	wasn't	346
38	you're	267
23	that's	245
58	McDonald's,	239
36	doesn't	227
20	couldn't	211

These words will be cleaned as the apostrophe's will be removed when punctuation is removed.

Stemming is the process of reducing a word down to it's stem. For example, "history" and "historical" are both reduced to "histori". Or "finally" and "final" are both reduced to "fina." A disadvantage of stemming is that it can reduce the readability of the words.

Lemmatization converts a word to is meaningful base form. For example, "caring" would be convert to "care". An advantage is that it retains its readability, but it is computationally more expensive[2].

Stemming and lemmatization was considered, but not carried out. After comparing the words in this list with apostrophe's to the list of stopwords that will be removed, it was determined that most of these words are stop words. Stemming or lemmatization would not help as the words would still be removed later.

To complete our exploratory data analysis on the review column the vocabulary size is analyzed. This is important as this is an input to the models later. A function is defined, vocab-size_sequence_length which takes as input a Dataframe and column name. The function tokenizes the words that are input to determine unique words, and then counts how many times that word appears. Finally, it determines the length of the longest review, the length of the average review, and the length of the shortest review.

```
In [381]: def vocab_size_sequence_length(df, column_name, header):
     ...:     """
     ...:     """
     ...:     word_tokenizer.fit_on_texts(df[column_name])
     ...:     vocab_size = len(word_tokenizer.word_index) + 1  # how many unique words
     ...:
     ...:     review_length = []
     ...:     for char_len in df[column_name]:
     ...:         review_length.append(len(char_len.split(' ')))
     ...:
     ...:     review_max = np.max(review_length)
     ...:     review_min = np.min(review_length)
     ...:     review_median = np.median(review_length)
     ...:
     ...:     return print(header), print('Vocab size (unique words) is: ', vocab_size), print('Longest review has', review_max, 'words.'), print('Shortest review
has', review_min, 'words.'), print('Average review has', review_median, 'words.')

In [382]: print(vocab_size_sequence_length(df, 'review', 'Review dirty and not preprocessed.'))
Review dirty and not preprocessed.
Vocab size (unique words) is:  15232
Longest review has 584 words.
Shortest review has 1 words.
Average review has 11.0 words.
(None, None, None, None, None)
```

For the uncleaned review text the vocabulary size is 15,232 unique words. The long review has 584 words, with the shortest review being 1 word. The average review length is 11 words.

Data Preparation

In section 6, we create two new Dataframe from the main Dataframe, df, that the exploratory data analysis has been completed on:

```
# %%[6] SPLIT DATAFRAME DF

df_store_info = df[['store_address', 'latitude ', 'longitude', 'rating_count']].copy()
df_reviews = df[['reviewer_id', 'review_time', 'rating', 'review']].copy()

```

Dataframe df_reviews is created with columns reviewer_id, review_time, rating, and review. These are the columns necessary for the neural network.

[2] https://www.analyticsvidhya.com/blog/2022/06/stemming-vs-lemmatization-in-nlp-must-know-differences/#h2_4

Dataframe df_store_info is created and holds all the data related to the stores locations. This Dataframe will be cleaned in section 7.1 to remove duplicates making the data “tidy”. This Dataframe holds columns store_address, latitude, longitude, and rating_count.

```
# %%[7.0] CLEAN AND PREPROCESS
# %%%[7.1] CLEAN DF_STORE_INTO
# clean up the store master data
# citation: https://www.geeksforgeeks.org/delete-duplicates-in-a-pandas-dataframe-based-on-two-columns/
df_store_info = df_store_info.drop_duplicates(subset=['store_address', 'latitude ', 'longitude', 'rating_count'], keep='first').reset_index(drop=True)
# some address are in the list twice with different rating counts - the rating count if off by a small number
df_store_info = df_store_info.drop_duplicates(subset=['store_address'], keep='first').reset_index(drop=True)

print(df_store_info.info())
# notes: rating_count column is a number with integer, lat and long at each missing 1

# Remove commas and convert to integer
df_store_info['rating_count'] = df_store_info['rating_count'].str.replace(',', '', regex=True).astype(int)

# double check
print(df_store_info.info())
```

Nothing further is done with df_store_info for this analysis.

In section 7.2 the df_reviews Dataframe is cleaned and processed to prepare it for the neural network.

In section 7.2.1 the ‘rating’ column is cleaned. From the analysis above, there are 5 unique values: 1 star, 2 stars, 3 stars, 4 stars, and 5 stars. These values need to be converted to either a 0, to represent negative sentiment, or a 1 to represent positive sentiment.

A dictionary, rating_mapping, is created to map the five values to 0 or 1 and then the Pandas “map” method is used to carry out the mapping. The function eda_analysis is executed on this column confirming the mapping was successful.

```
# %%%%[7.2.1] PREPROCESS COLUMN: RATING
# create a dictionary to map to reivews ratings to 0 (negative sentiment) or 1 (positive sentiment)
rating_mapping = {'1 star': 0,
                  '2 stars': 0,
                  '3 stars': 0,
                  '4 stars': 1,
                  '5 stars': 1}
df_reviews['rating'] = df_reviews['rating'].map(rating_mapping)  # update the column in df_reviews
print(eda_analysis(df_reviews, 'rating'))
```

```
Begin rating:
Number of unique values:  2

Datatype of the column:  int64

Describe rating
count    33396.000000
mean         0.480926
std          0.499644
min          0.000000
25%          0.000000
50%          0.000000
75%          1.000000
max          1.000000
Name: rating, dtype: float64


Frequency table for variable rating :
         Count  Percentages
rating
0       17335        51.91
1       16061        48.09

End rating
```

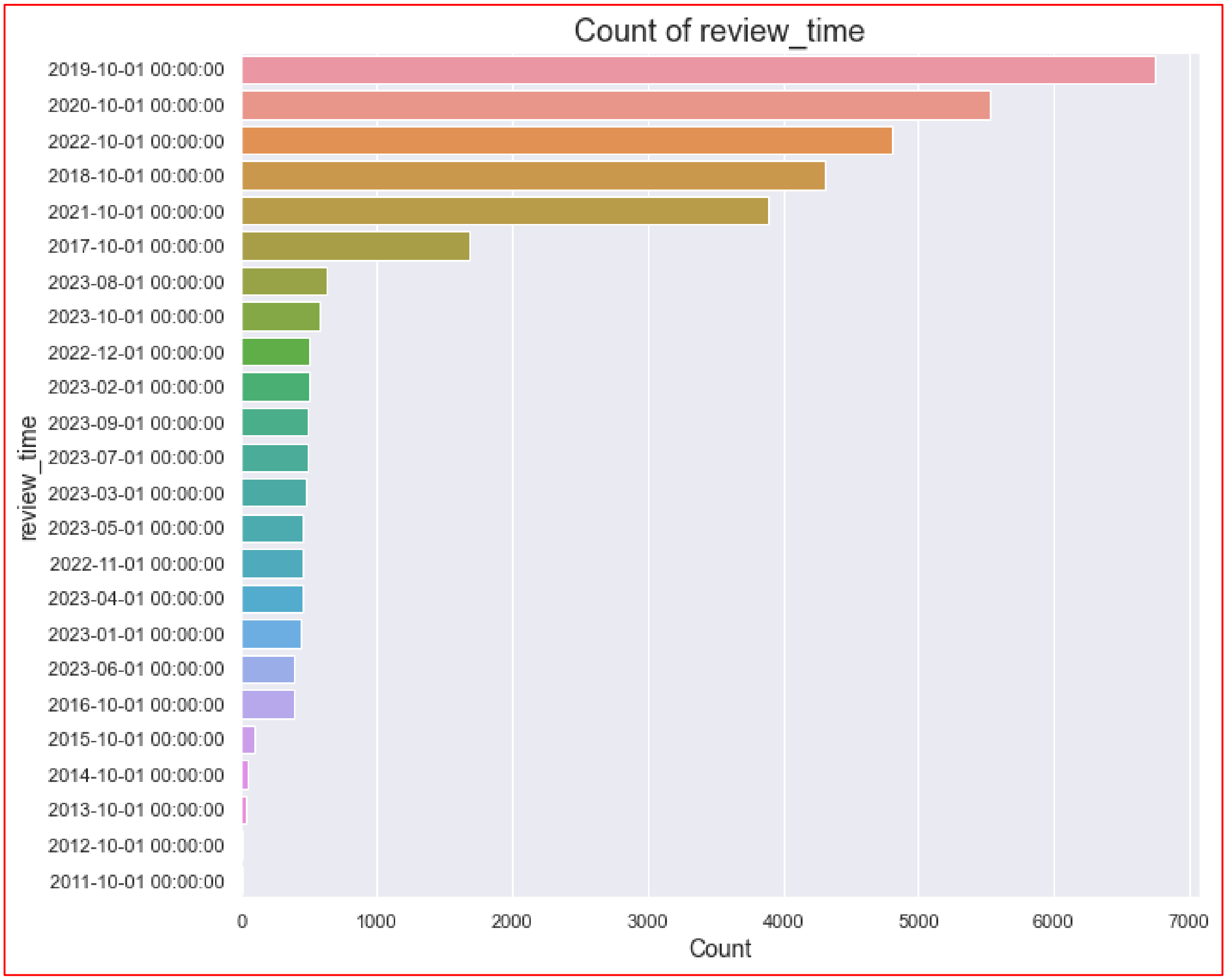
In section 7.2.2 the "review_time" column is cleaned. A dictionary, time_mapping, is created to map the values in the column to actual dates. For this analysis, "today" is considered to be October, 2023. The values in the column are subtracted from this date, and set to the first of the month.

For example, "8 hours ago" is mapped to 2023-10-01; "5 months ago" is mapped to 2023-05-01"; "9 years ago" is mapped to 2014-10-01. For the purposes of this analysis this mapping is sufficient. If this were a production model, the "today" date would constantly be changing and a different method would need to used. Having a datetime field will allow the sentiment to be measured over time, if desired (though this is not a focus of this project).

After the time_mapping dictonary is created, Pandas "replace" method is used to replace the values in the "review_time" column with the new value. Then the column type is changed from "object" to "datetime":

```
In [387]: time_mapping = {'review_time':
     ...:                 {'6 hours ago': '2023-10-01',
     ...:                     '8 hours ago': '2023-10-01',
     ...:                     '20 hours ago': '2023-10-01',
     ...:                     '21 hours ago': '2023-10-01',
     ...:                     '22 hours ago': '2023-10-01',
     ...:                     '23 hours ago': '2023-10-01',
     ...:                     'a day ago': '2023-10-01',
     ...:                     '2 days ago': '2023-10-01',
     ...:                     '3 days ago': '2023-10-01',
     ...:                     '4 days ago': '2023-10-01',
     ...:                     '5 days ago': '2023-10-01',
     ...:                     '6 days ago': '2023-10-01',
     ...:                     'a week ago': '2023-10-01',
     ...:                     '2 weeks ago': '2023-10-01',
     ...:                     '3 weeks ago': '2023-10-01',
     ...:                     '4 weeks ago': '2023-10-01',
     ...:                     'a month ago': '2023-09-01',
     ...:                     '2 months ago': '2023-08-01',
     ...:                     '3 months ago': '2023-07-01',
     ...:                     '4 months ago': '2023-06-01',
     ...:                     '5 months ago': '2023-05-01',
     ...:                     '6 months ago': '2023-04-01',
     ...:                     '7 months ago': '2023-03-01',
     ...:                     '8 months ago': '2023-02-01',
     ...:                     '9 months ago': '2023-01-01',
     ...:                     '10 months ago': '2022-12-01',
     ...:                     '11 months ago': '2022-11-01',
     ...:                     'a year ago': '2022-10-01',
     ...:                     '2 years ago': '2021-10-01',
     ...:                     '3 years ago': '2020-10-01',
     ...:                     '4 years ago': '2019-10-01',
     ...:                     '5 years ago': '2018-10-01',
     ...:                     '6 years ago': '2017-10-01',
     ...:                     '7 years ago': '2016-10-01',
     ...:                     '8 years ago': '2015-10-01',
     ...:                     '9 years ago': '2014-10-01',
     ...:                     '10 years ago': '2013-10-01',
     ...:                     '11 years ago': '2012-10-01',
     ...:                     '12 years ago': '2011-10-01'}}

In [388]: df_reviews.replace(time_mapping, inplace=True)  # replace the values in the column using the dictionary defined above

In [389]: df_reviews['review_time'] = pd.to_datetime(df_reviews['review_time'])  # change the datatype of the column
```

```
In [390]: print(eda_analysis(df_reviews, 'review_time'))
------------------------------------------------------------
Begin review_time:
Number of unique values:  24

Datatype of the column:  datetime64[ns]

Describe review_time
count                            33396
mean     2020-11-28 21:17:18.735177728
min                2011-10-01 00:00:00
25%                2019-10-01 00:00:00
50%                2020-10-01 00:00:00
75%                2022-10-01 00:00:00
max                2023-10-01 00:00:00
Name: review_time, dtype: object


Frequency table for variable review_time :
              Count  Percentages
review_time
2019-10-01    6740        20.18
2020-10-01    5522        16.53
2022-10-01    4809        14.40
2018-10-01    4306        12.89
2021-10-01    3892        11.65
2017-10-01    1679         5.03
2023-08-01     625         1.87
2023-10-01     575         1.72
2022-12-01     503         1.51
2023-02-01     498         1.49
2023-09-01     493         1.48
2023-07-01     492         1.47
2023-03-01     472         1.41
2023-05-01     458         1.37
2022-11-01     457         1.37
2023-04-01     456         1.37
2023-01-01     444         1.33
2023-06-01     393         1.18
2016-10-01     387         1.16
2015-10-01      91         0.27
2014-10-01      52         0.16
2013-10-01      38         0.11
2012-10-01      10         0.03
2011-10-01       4         0.01
```



After this is completed, it is noted that most of the reviews occur between 2017 and 2022. This is a disadvantage for the analysis as they are not more recent. Ideally, when calculating customer satisfaction reviews as recent as possible are desired.

In section 7.2.3 the "review" column is cleaned and processed for the neural network. Function, clean_review, is created to do this work. A function is created so the same steps can be carried out later in the analysis on the social media reviews that were collected.

Function clean_review, does the following:

- takes as input a Dataframe, column name, and list of punctuation
- each step in the function completes one aspect of cleaning and saves the result in a new column in the Dataframe. This allows steps to be verified and observe how the data is processed.
- The text is converted all to lowercase. This reduces the number of tokens that will be created in later steps. "Ordered" and "ordered" – different only by the first letter being capitalized would be two different tokens and this increase the token size. By converting all text to lowercase the number of different tokens is reduced.
- Next, punctuation is removed. Punctuation has no discernible impact on sentiment, but if it exists would be tokenized. Removing the punctuation reduces the number of tokens. The list of punctation stored earlier, list_special_characters, is used for this step

- Next, stopwords are removed. The STOPWORDS library from the wordcloud package is used. No additional stopwords were added to this library for removal.
- While building this function it was noticed many reviews had occurrences of the letters “xbf”, “xef”, and “xfd”. This appear to be as a result of the screen scraping that created the dataset, but have no determination on the sentiment. As these are not required and to reduce the number of tokens, these are removed. The following code was used to identify these and confirm their removal.

```
# while building the clean_review function these values were notices. these lines hlped determine if occurance was significant enough for removal
# Count the occurrences of 'xbf' or 'xef' in the 'reviews' column
count_xbf_pre = df_reviews['review'].str.count(r'xbf').sum()
count_xef_pre = df_reviews['review'].str.count(r'xef').sum()
count_xfd_pre = df_reviews['review'].str.count(r'xfd').sum()

print("Total count of 'xbf' in the 'reviews' column before cleaning:", count_xbf_pre)
print("Total count of 'xef' in the 'reviews' column before cleaning:", count_xef_pre)
print("Total count of 'xfd' in the 'reviews' column before cleaning:", count_xfd_pre)

count_xbf_post = df_reviews['review_no_xbf_xef'].str.count(r'xbf').sum()
count_xef_post = df_reviews['review_no_xbf_xef'].str.count(r'xef').sum()
count_xfd_post = df_reviews['review_no_xbf_xef'].str.count(r'xfd').sum()

print("Total count of 'xbf' in the 'reviews' column after cleaning:", count_xbf_post)
print("Total count of 'xef' in the 'reviews' column before cleaning:", count_xef_post)
print("Total count of 'xfd' in the 'reviews' column before cleaning:", count_xfd_post)
```

- Next, the reviews are tokenized – or the words are converted to numbers.
- Next, the tokenized reviews are looped over to determine the longest review, the shortest review, and the length of the average review. Thes are required inputs for the neural network. This was completed above as part of exploratory data analysis, however, that was before the review text was cleaned and processed. These numbers will be lower with the review text cleaned.
- Next, the tokenized reviews are padded. This is added zeros to the end of the tokens, ensuring that each tokenized review is the same length. This is a necessary step as the neural network requires and input of an array with each array row the same length. Two padded columns are added to the Dataframe:
    - Padded_sequences_max: this column contains each review padded to the max length, as determined above. Here that is 273 tokens.
    - Padded_sequences_median: this column contains each review padded to the average review length, as determined above. Here that is 6 tokens.
    - Models will be executed with both to determine if models that use an average sequence length are as accurate as models that use the max sequence length. If so, this could potentially result in a large database savings not having to store all reviews padded to the max review.
- Finally, a series of columns are added to determine lengths of previous columns as an accuracy check that all columns were processed correctly.
- Finally, the function returns the review lengths, so these values can be used as inputs to the models

```
def clean_review(df, column_name, punct_list):
    """
    """

    df['review_lowercase'] = df[column_name].str.lower()  # convert text to lowercase
    df['review_no_punct'] = df['review_lowercase'].apply(lambda text: ''.join([' ' if char in punct_list else char for char in text]))  # remove punctation
    df['review_no_stopwords'] = df['review_no_punct'].apply(lambda x: ' '.join([word for word in x.split() if word not in (STOPWORDS)]))  # remove stopwords
    df['review_no_xbf_xef'] = df['review_no_stopwords'].str.replace(r'xbf|xef|xfd', '', regex=True)  # remove xbf and xef
    df['review_tokenized'] = word_tokenizer.texts_to_sequences(df['review_no_xbf_xef'])  # convert sentences to numeric counterpart

    review_length = []
    for char_len in df['review_no_xbf_xef']:
        review_length.append(len(char_len.split(' ')))

    review_max = np.max(review_length)
    review_min = np.min(review_length)
    review_median = int(np.median(review_length))

    padded_sequences_max = pad_sequences(df['review_tokenized'], padding='post', maxlen=review_max)  # add zeros to the end to set them all to the same length
    df['review_padded_max'] = pd.DataFrame({'review_padded_max': padded_sequences_max.tolist()})

    padded_sequences_median = pad_sequences(df['review_tokenized'], padding='post', maxlen=review_median)  # add zeros to the end to set them all to the same length
    df['review_padded_median'] = pd.DataFrame({'review_padded_max': padded_sequences_median.tolist()})

    df['review_len'] = df[column_name].str.len()  # original review length
    df['review_no_stopwords_len'] = df['review_no_stopwords'].str.len()  # length after stop word removal
    df['review_no_xbf_xef_len'] = df['review_no_xbf_xef'].str.len()  # length after removing xbf and xef
    df['review_tokenized_len'] = df['review_tokenized'].str.len()
    df['review_padded_max_len'] = df['review_padded_max'].str.len()
    df['review_padded_median_len'] = df['review_padded_median'].str.len()
    # df['len_diff'] = df['review_len'].sub(df['review_no_stopwords_len'])  # https://www.tutorialspoint.com/how-to-subtract-two-columns-in-pandas-dataframe

    return print('Longest review has', review_max, 'words.'), print('Shortest review has', review_min, 'words.'), print('Average review has', review_median, 'words.'),
print('all done')
```

After the function is executed, the Dataframe df_reviews now contains 17 columns:

```
In [392]: print(df_reviews.info())
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 33396 entries, 0 to 33395
Data columns (total 17 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   reviewer_id               33396 non-null  int64
 1   review_time               33396 non-null  datetime64[ns
 2   rating                    33396 non-null  int64
 3   review                    33396 non-null  object
 4   review_lowercase          33396 non-null  object
 5   review_no_punct           33396 non-null  object
 6   review_no_stopwords       33396 non-null  object
 7   review_no_xbf_xef         33396 non-null  object
 8   review_tokenized          33396 non-null  object
 9   review_padded_max         33396 non-null  object
 10  review_padded_median      33396 non-null  object
 11  review_len                33396 non-null  int64
 12  review_no_stopwords_len   33396 non-null  int64
 13  review_no_xbf_xef_len     33396 non-null  int64
 14  review_tokenized_len      33396 non-null  int64
 15  review_padded_max_len     33396 non-null  int64
 16  review_padded_median_len  33396 non-null  int64
dtypes: datetime64[ns](1), int64(8), object(8)
memory usage: 4.3+ MB
None
```

Here are two examples of the review text being cleaned:
Example 1:

Column	Value
Review	Why does it look like someone spit on my food? I had a normal transaction, everyone was chill and polite, but now i dont want to eat this. Im trying not to think about what this milky white/clear substance is all over my food, i d*** sure am not coming back.
Review_lowercase	why does it look like someone spit on my food? i had a normal transaction, everyone was chill and polite, but now i dont want to eat this. im trying not to think about what this milky white/clear substance is all over my food, i d*** sure am not coming back.
Review_no_punct	why does it look like someone spit on my food i had a normal transaction everyone was chill and polite but now i dont want to eat this im trying not to think about what this milky white clear substance is all over my food i d sure am not coming back
Review_no_stopwords	look someone spit food normal transaction everyone chill polite now dont want eat im trying think milky white clear substance food d sure coming back
Review_no_xbf_xef	look someone spit food normal transaction everyone chill polite now dont want eat im trying think milky white clear substance food d sure coming back
Review_tokenized	[326, 294, 2196, 7, 595, 3021, 354, 2369, 505, 210, 337, 159, 104, 661, 444, 289, 3602, 938, 1219, 4782, 7, 572, 258, 345, 74]

Review_padded_max	[326, 294, 2196, 7, 595, 3021, 354, 2369, 505, 210, 337, 159, 104, 661, 444, 289, 3602, 938, 1219, 4782, 7, 572, 258, 345, 74, 0]
Review_padded_median	[4782, 7, 572, 258, 345, 74]

Example 2:

Column	Value
Review	Didn���������������������������t take card. But didn���������\xef\xbf
Review_lowercase	didn���������������������������t take card. but didn���������\xef\xbf
Review_no_punct	didn t take card but didn xef xbf
Review_no_stopwords	didn t take card didn xef xbf
Review_no_xbf_xef	didn t take card didn
Review_tokenized	[250, 58, 116, 560, 250]
Review_padded_max	[250, 58, 116, 560, 250, 0]
Review_padded_median	[250, 58, 116, 560, 250, 0]

While examing the Dataframe after executing the Dataframe function, it was noted that after cleaning some reviews are effectively blank. In other words, after removing stopwords and punctuation none of the review is left. As in this example:

Column	Value
Review	Can������������\xef\xbf
Review_lowercase	can������������\xef\xbf
Review_no_punct	can xef xbf
Review_no_stopwords	xef xbf
Review_no_xbf_xef	
Review_tokenized	[]
Review_padded_max	[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, 0]
Review_padded_median	[0, 0, 0, 0, 0, 0]

These reviews will add nothing to the analysis and should be removed from df_reviews. A function has_all_zeros, is written to identify rows in revew_padded_max that contain only zeros. These rows are then removed from the Dataframe df_reviews and stored in Dataframe df_blank_rows:

```
In [393]: df_reviews['review_padded_max_str'] = df_reviews['review_padded_max']  # create a new column that is a copy of review_padded_max

In [394]: df_reviews['review_padded_max_str'] = df_reviews['review_padded_max_str'].astype(str)  # type the new column as a str for the function

In [395]: def has_all_zeros(value):
     ...:     """check to see if all values are zero
     ...:     """
     ...:     pattern = r'^\[0+(?:,\s*0+)*\]$'
     ...:     return bool(re.match(pattern, value))

In [396]: print(df_reviews.shape)
(33396, 18)

In [397]: df_blank_rows = df_reviews[df_reviews['review_padded_max_str'].apply(has_all_zeros)]  # copy the rows with zeros to a new dataframe

In [398]: df_reviews_indicies_to_drop = df_blank_rows.index  # create a list of the index to drop (of the rows that were moved)

In [399]: df_reviews = df_reviews.drop(df_reviews_indicies_to_drop)  # drop these rows from df_reviews

In [400]: print(df_reviews.shape)
(33271, 18)
```

125 rows are moved to Dataframe df_blank_rows.

The vocabulary size of the cleaned reviews is determined by excuting function vocab_size_sequence_lenght again with Dataframe df_reviews and column review_no_xbf_xef:

```
In [401]: print(vocab_size_sequence_length(df_reviews, 'review_no_xbf_xef', 'Review preprocessed and clean.'))
Review preprocessed and clean.
Vocab size (unique words) is:  15232
Longest review has 273 words.
Shortest review has 1 words.
Average review has 6.0 words.
```

The vocabulary size is 15,232 unique words. This is a required input for the model.

Next, the embedding dimension is determined by take the 4th square root of the vocabulary size[3]:

```
In [402]: embedding_dimension = int(round(np.sqrt(np.sqrt(15232)), 0))  # 4th squart root of the vocab size

In [403]: print(f'embedding dimension {embedding_dimension}')
embedding dimension 11
```

By this method the embedding dimension is determined to be 11.

[3] https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/

Finally, the data is split into test and training sets for modeling. This is completed twice – once for the reviewed padded to max length (273) and once for the review padded to average length (6).

Max length:

```
In [404]: X = df_reviews['review_padded_max']
     ...: y = df_reviews['rating']
     ...:
     ...: # split the data set into train and test sets with 80/20 split
     ...: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=36, stratify=y)
     ...:
     ...: print('Training size: ', X_train.shape, '\n')
     ...: print('Test size: ', X_test.shape, '\n')
Training size:  (26616,)

Test size:  (6655,)
```

Average length:

```
In [405]: X_median = df_reviews['review_padded_median']
     ...: y_median = df_reviews['rating']
     ...:
     ...: # split the data set into train and test sets with 80/20 split
     ...: X_train_median, X_test_median, y_train_median, y_test_median = train_test_split(X_median, y_median, test_size=0.20, random_state=36, stratify=y)
     ...:
     ...: print('Training size: ', X_train_median.shape, '\n')
     ...: print('Test size: ', X_test_median.shape, '\n')
Training size:  (26616,)

Test size:  (6655,)
```

This completes the data cleaning and preparation.

Analysis

The data in Dataframe df_reviews is analysized in section 9 of the code. In section 9.1.1 function plot_learningCurve is created which will be used to plot Model accuracy and Model loss:

```
def plot_learningCurve(history):
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Model Accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Val'], loc='upper left')
    plt.show()
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model Loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Val'], loc='upper left')
    plt.show()
```

In section 9.1.2 a new Dataframe, df_model_metrics is created. This will be appended to as various models are created to easily store and compre model metrics:

```
df_model_metrics = pd.DataFrame({
    'model_name': [],
    'model_description': [],
    'test_loss': [],
    'test_accuracy': [],
    'epoch_stopped_at': []
    })
```

In section 9.2 an early stopping monitor[4] is created. This will stop the model after no improvement in model metrics. The patience, or number of epochs with no improvement after which training will be stopped, is set to 2.

```
# Early Stopping Monitor
early_stopping_monitor = EarlyStopping(patience=2)  # model will stop after 2 epochs of no improvement
```

Overall, four models are created:

- Model_1 has two dense layers and uses the max length of 273 tokens
- Model_2 has two dense layers and uses the median length of 6 tokens
- Model_3 has three dense layers and uses the max length of 273 tokens
- Model_4 has three dense layers and uses the median length of 6 tokens

Model 1

X_train and X_test were created as a series from the splitting completed above. The model requires them to be an array, and they are converted here:

```
In [406]: X_train = np.array(X_train.tolist())

In [407]: X_test = np.array(X_test.tolist())
```

X_test - NumPy object array

	0	1	2	3	4	5	6	7	8	9	10	11
0	194	58	97	57	26	23	50	7	1435	470	0	0
1	156	103	109	417	423	0	0	0	0	0	0	0
2	26	57	26	447	32	13059	449	176	3681	1676	7065	223
3	13	318	0	0	0	0	0	0	0	0	0	0
4	57	26	275	1678	295	156	89	154	1124	194	58	894
5	1041	697	2750	598	1016	659	1010	57	26	4261	456	26
6	1203	177	57	26	548	139	144	1074	172	67	49	33
7	36	0	0	0	0	0	0	0	0	0	0	0
8	84	8	0	0	0	0	0	0	0	0	0	0
9	1378	540	364	1056	88	230	1795	8933	944	58	371	114

The neural network used is a Keras sequential model, with layers that are input to the next layer. An advantage[5] of this model is that it works well when there are single input tensors and single output tensors, as in this analysis. A disadvantage is that sequential models do not allow you to create model that share layers – but this not required for this analysis.

Each model is has this basic format:

- Embedding layer: this is the input layer and takes as input the vocabulary size, the embedding dimension, and the input length
- A Flatten layer, which flattens the input

[4] https://keras.io/api/callbacks/early_stopping/

[5] https://keras.io/guides/sequential_model/

- Dense, or hidden layers with the 'relu' activation function
- An output layer with the 'softmax' activation function
- A complie layer with the 'adam' optimizer, a loss function of 'sparse categorical crossentropy', and accuracy metrics

Model 1 uses the full review length so input_length is set to 273 and has two "hidden" layers.
The input_dim is the vocabulary size calculated: 15,232 unique words.
The output_dim is embedding dimension calculated: 11

```
In [408]: model_1 = Sequential()
     ...: model_1.add(Embedding(input_dim=15232, output_dim=11, input_length=273))
     ...: model_1.add(Flatten())  # https://keras.io/api/layers/reshaping_layers/flatten/
     ...: model_1.add(Dense(100, activation='relu'))
     ...: model_1.add(Dense(50, activation='relu'))
     ...: model_1.add(Dense(2, activation='softmax'))
     ...: model_1.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
     ...: print(model_1.summary())
```

Next we show the model summary which details the number of parameters per layer:

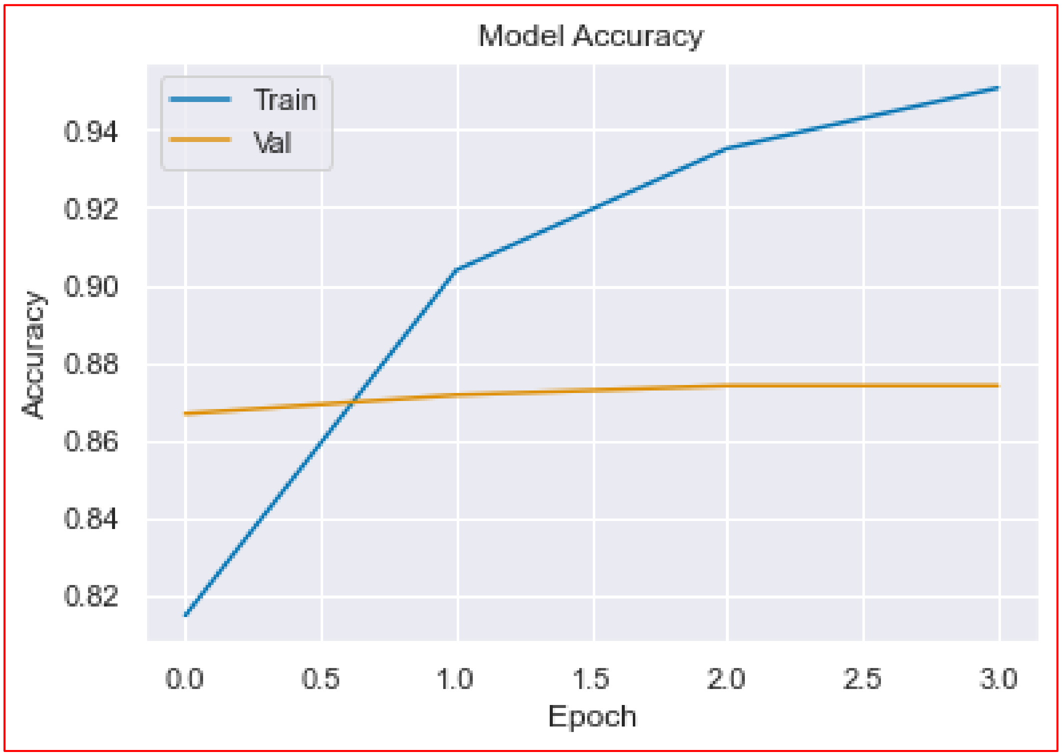
```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_11 (Embedding)    (None, 273, 11)           167552

 flatten_11 (Flatten)        (None, 3003)              0

 dense_34 (Dense)            (None, 100)               300400

 dense_35 (Dense)            (None, 50)                5050

 dense_36 (Dense)            (None, 2)                 102

=================================================================
Total params: 473,104
Trainable params: 473,104
Non-trainable params: 0
_________________________________________________________________
None
```

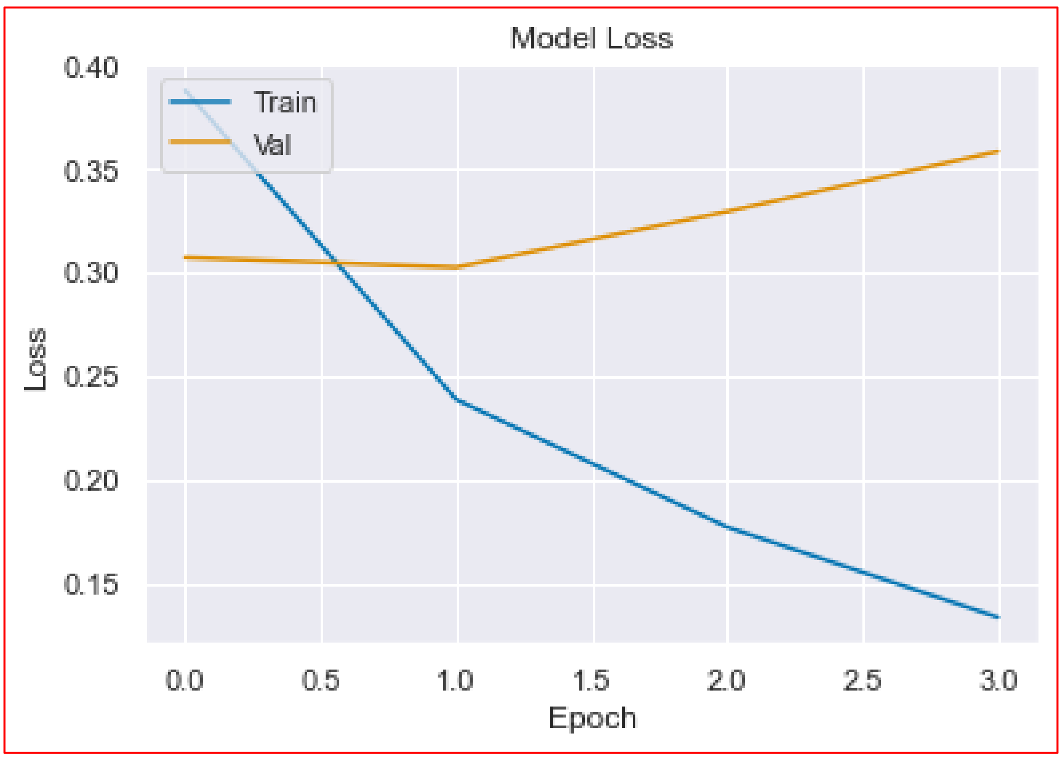
Next we execute the model on the X_train dataset and save the results to model_1.history:

```
In [409]: model_1_history = model_1.fit(X_train, y_train, epochs=20, batch_size=32, callbacks=[early_stopping_monitor], verbose=True,
validation_data=(X_test, y_test))
Epoch 1/20
832/832 [==============================] - 6s 6ms/step - loss: 0.3878 - accuracy: 0.8148 - val_loss: 0.3073 - val_accuracy: 0.8669
Epoch 2/20
832/832 [==============================] - 5s 6ms/step - loss: 0.2389 - accuracy: 0.9038 - val_loss: 0.3027 - val_accuracy: 0.8717
Epoch 3/20
832/832 [==============================] - 17s 21ms/step - loss: 0.1775 - accuracy: 0.9352 - val_loss: 0.3295 - val_accuracy: 0.8741
Epoch 4/20
832/832 [==============================] - 6s 7ms/step - loss: 0.1341 - accuracy: 0.9507 - val_loss: 0.3585 - val_accuracy: 0.8741

In [410]: print(model_1_history.history)
{'loss': [0.3877904415130615, 0.23893848061561584, 0.1775476485490799, 0.13408063352108002], 'accuracy': [0.8148481845855713, 0.9038172364234924,
0.9351893663406372, 0.9507439136505127], 'val_loss': [0.30734941363334656, 0.30266737937927246, 0.3294665217399597, 0.35845527052879333],
'val_accuracy': [0.8668670058250427, 0.871675431728363, 0.8740796446800232, 0.8740796446800232]}
```

Even though the model was set to execute for 20 epochs, the model stopped after 4 epochs due to the early stopping monitor which showed no improvement in accuracy. Epoch 4 has an accuracy of 97%.

Model Accuracy and Model Loss curves are plotted using the plot_learninCurve function defined above:





Next, the overall model score on the X_train dataset is retrieved and saved to model_1_score:

```
In [413]: print(f'Training Set: Test Loss: {model_1_score[0]} / Test Accuracy: {model_1_score[1]}')
Training Set: Test Loss: 0.09815818816423416 / Test Accuracy: 0.9664487242698669
```

Overall model score is 96% with 9.8% loss.

Next, we use the model and evaluate on the X_test dataset and save the result to model_1_evaluation:

```
In [414]: model_1_evaluation = model_1.evaluate(X_test, y_test)
208/208 [==============================] - 1s 3ms/step - loss: 0.3585 - accuracy: 0.8741

In [415]: print(f'Test Set: Test Loss: {model_1_evaluation[0]} / Test Accuracy: {model_1_evaluation[1]}')
Test Set: Test Loss: 0.35845527052879333 / Test Accuracy: 0.8740796446800232
```

The accuracy score on the test set is 87% - lower than the score on the training set.

Finally, the scores are appended to Dataframe df_model_metrics, so scores can be compared across models easily:

```
In [418]: df_model_metrics = pd.concat([df_model_metrics, df_model_1_metrics], ignore_index=True)

In [419]: print(df_model_metrics)
  model_name     model_description  test_loss  test_accuracy epoch_stopped_at
0    model_1  2 layers max length   0.358455        0.87408                4
```

Model 2

Model 2 uses the average review length, so input_length is set to 6 and has two "hidden" layers. The input_dim is the vocabulary size calculated: 15,232 unique words.
The output_dim is embedding dimension calculated: 11

```
In [422]: model_2 = Sequential()
     ...: model_2.add(Embedding(input_dim=15232, output_dim=11, input_length=6))
     ...: model_2.add(Flatten())  # https://keras.io/api/layers/reshaping_layers/flatten/
     ...: model_2.add(Dense(100, activation='relu'))
     ...: model_2.add(Dense(50, activation='relu'))
     ...: model_2.add(Dense(2, activation='softmax'))
     ...: model_2.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
     ...: print(model_2.summary())
Model: "sequential_12"
```

Next we show the model summary which details the number of parameters per layer:

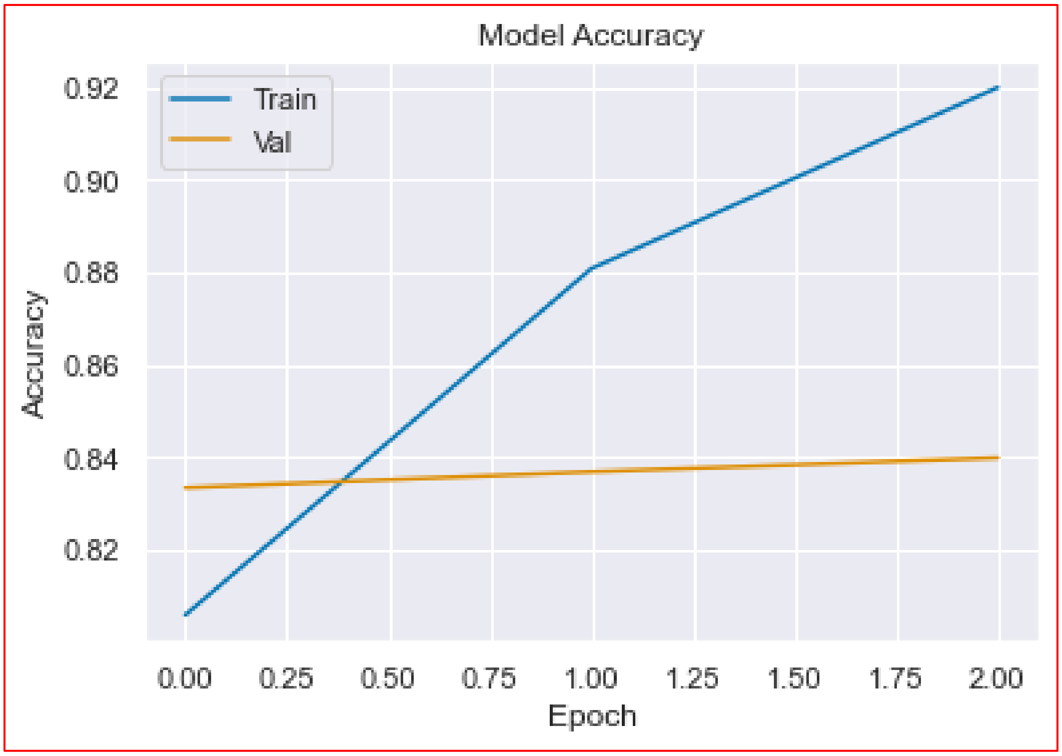
```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_12 (Embedding)    (None, 6, 11)             167552

 flatten_12 (Flatten)        (None, 66)                0

 dense_37 (Dense)            (None, 100)               6700

 dense_38 (Dense)            (None, 50)                5050

 dense_39 (Dense)            (None, 2)                 102

=================================================================
Total params: 179,404
Trainable params: 179,404
Non-trainable params: 0
_________________________________________________________________
None
```

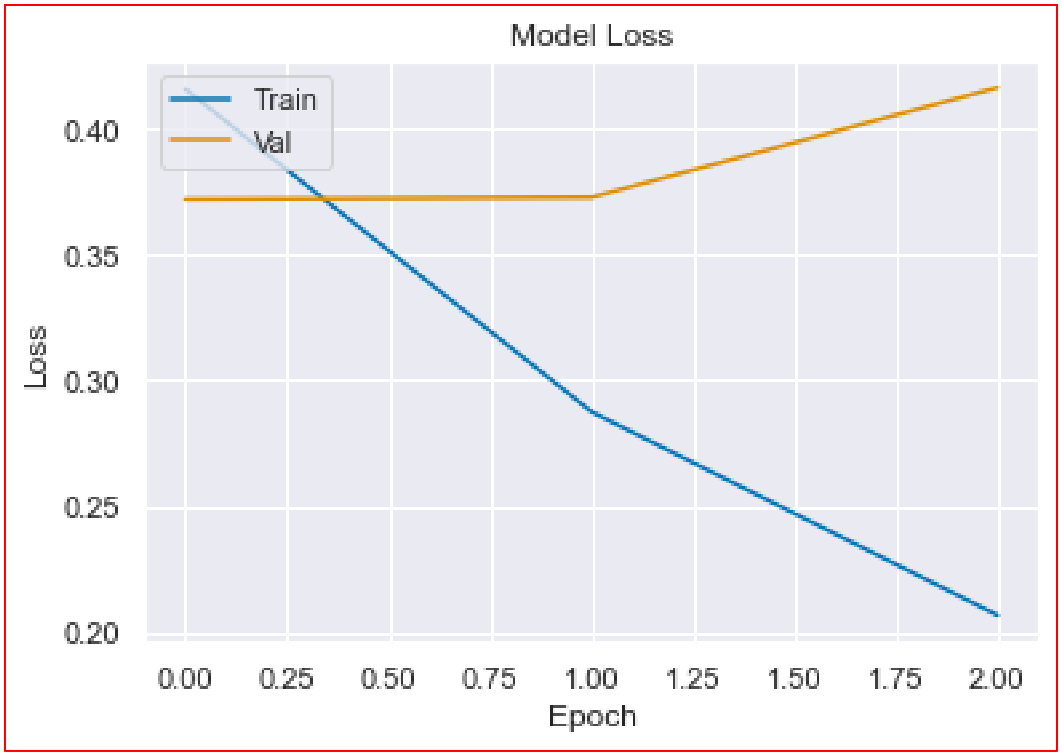
Next we execute the model on the X_train_median dataset and save the results to model_2.history:

```
In [423]: model_2_history = model_2.fit(X_train_median, y_train_median, epochs=20, batch_size=32, callbacks=[early_stopping_monitor],
verbose=True, validation_data=(X_test_median, y_test_median))
Epoch 1/20
832/832 [==============================] - 4s 4ms/step - loss: 0.4159 - accuracy: 0.8060 - val_loss: 0.3722 - val_accuracy: 0.8335
Epoch 2/20
832/832 [==============================] - 3s 3ms/step - loss: 0.2876 - accuracy: 0.8809 - val_loss: 0.3730 - val_accuracy: 0.8370
Epoch 3/20
832/832 [==============================] - 3s 4ms/step - loss: 0.2066 - accuracy: 0.9199 - val_loss: 0.4165 - val_accuracy: 0.8400

In [424]: print(model_2_history.history)
{'loss': [0.41592350602149963, 0.28755706548690796, 0.20659391582012177], 'accuracy': [0.8060189485549927, 0.8808611631393433, 0.919935405254364],
'val_loss': [0.3722377419471741, 0.3729751408100128, 0.41649821400642395], 'val_accuracy': [0.8335086107254028, 0.8369646668434143,
0.8399699330329895]}
```

Even though the model was set to execute for 20 epochs, the model stopped after 3 epochs due to the early stopping monitor which showed no improvement in accuracy. Epoch 3 has an accuracy of 91% - less than model 1.

Model Accuracy and Model Loss curves are plotted using the plot_learninCurve function defined above:





Next, the overall model score on the X_train dataset is retrieved and saved to model_2_score:

```
In [426]: model_2_score = model_2.evaluate(X_train_median, y_train_median, verbose=0)

In [427]: print(f'Training Set: Test Loss: {model_2_score[0]} / Test Accuracy: {model_2_score[1]}')
Training Set: Test Loss: 0.15373587608337402 / Test Accuracy: 0.9455214738845825
```

Overall model score is 94% with a 15% loss.

Next, we use the model and evaluate on the X_test_median dataset and save the result to model_2_evaluation:

```
In [428]: model_2_evaluation = model_2.evaluate(X_test_median, y_test_median)
208/208 [==============================] - 0s 2ms/step - loss: 0.4165 - accuracy: 0.8400

In [429]: print(f'Test Set: Test Loss: {model_2_evaluation[0]} / Test Accuracy: {model_2_evaluation[1]}')
Test Set: Test Loss: 0.41649821400642395 / Test Accuracy: 0.8399699330329895
```

The accuracy score on the test set is 83% - lower than the score on the training set.

Finally, the scores are appended to Dataframe df_model_metrics, so scores can be compared across models easily:

```
In [431]: df_model_metrics = pd.concat([df_model_metrics, df_model_2_metrics], ignore_index=True)

In [432]: print(df_model_metrics)
  model_name       model_description  test_loss  test_accuracy epoch_stopped_at
0    model_1     2 layers max length   0.358455        0.87408                4
1    model_2  2 layers median length   0.416498        0.83997                3
```

So far Model 1 is the best model.

Model 3

Model 3 uses the same input parameters as Model 1, but adds a “hidden” layer:

```
In [433]: model_3 = Sequential()
     ...: model_3.add(Embedding(input_dim=15232, output_dim=11, input_length=273))
     ...: model_3.add(Flatten())  # https://keras.io/api/layers/reshaping_layers/flatten/
     ...: model_3.add(Dense(100, activation='relu'))
     ...: model_3.add(Dense(50, activation='relu'))
     ...: model_3.add(Dense(25, activation='relu'))
     ...: model_3.add(Dense(2, activation='softmax'))
     ...: model_3.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
     ...: print(model_3.summary())
```

Next we show the model summary which details the number of parameters per layer:

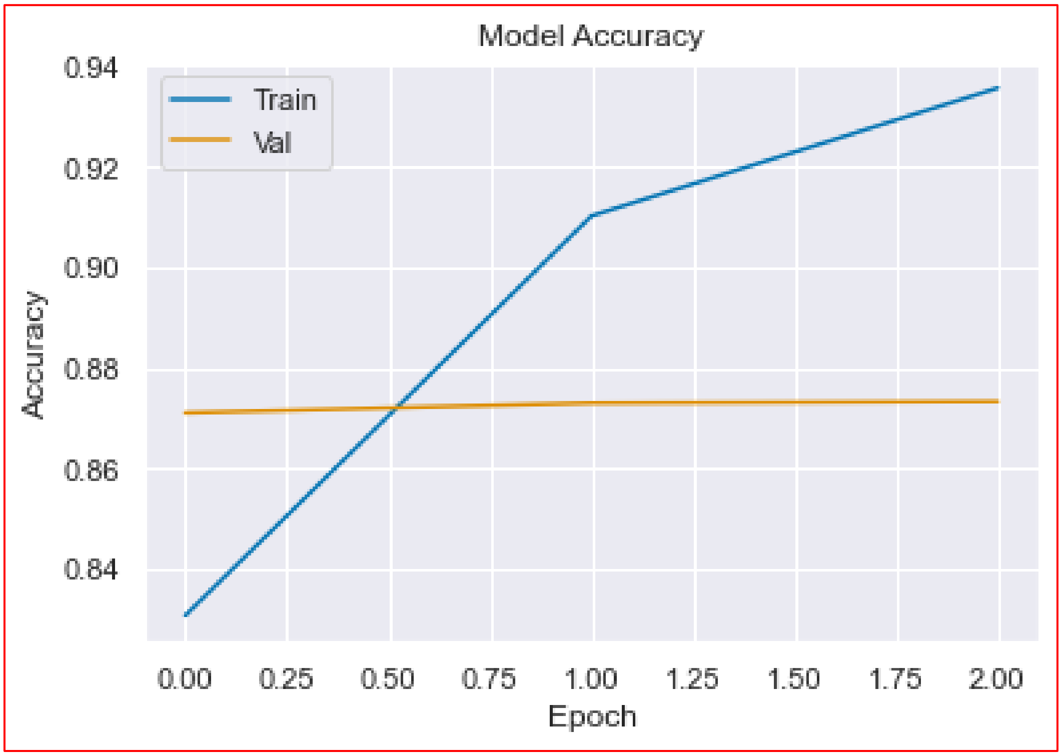
```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_13 (Embedding)    (None, 273, 11)           167552

 flatten_13 (Flatten)        (None, 3003)              0

 dense_40 (Dense)            (None, 100)               300400

 dense_41 (Dense)            (None, 50)                5050

 dense_42 (Dense)            (None, 25)                1275

 dense_43 (Dense)            (None, 2)                 52

=================================================================
Total params: 474,329
Trainable params: 474,329
Non-trainable params: 0
_________________________________________________________________
None
```

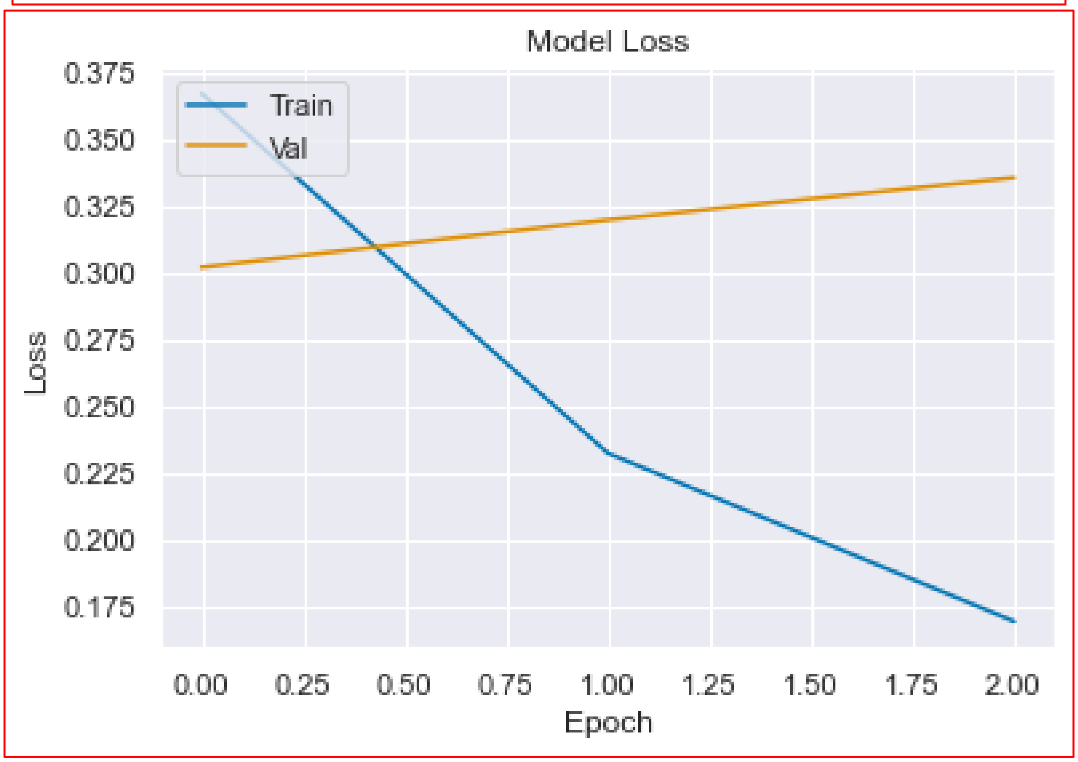
Next we execute the model on the X_train dataset and save the results to model_3.history:

```
In [434]: model_3_history = model_3.fit(X_train, y_train, epochs=20, batch_size=32, callbacks=[early_stopping_monitor], verbose=True,
validation_data=(X_test, y_test))
Epoch 1/20
832/832 [==============================] - 8s 9ms/step - loss: 0.3669 - accuracy: 0.8308 - val_loss: 0.3020 - val_accuracy: 0.8711
Epoch 2/20
832/832 [==============================] - 5s 6ms/step - loss: 0.2323 - accuracy: 0.9102 - val_loss: 0.3197 - val_accuracy: 0.8730
Epoch 3/20
832/832 [==============================] - 4s 5ms/step - loss: 0.1693 - accuracy: 0.9356 - val_loss: 0.3355 - val_accuracy: 0.8733

In [435]: print(model_3_history.history)
{'loss': [0.3669384717941284, 0.23229819536209106, 0.16931922733783722], 'accuracy': [0.8307784795761108, 0.9102044105529785, 0.9356026649475098],
'val_loss': [0.3019617199897766, 0.3197106122970581, 0.33551594614982605], 'val_accuracy': [0.871074378490448, 0.8730278015136719,
0.8733283281326294]}
```

Even though the model was set to execute for 20 epochs, the model stopped after 3 epochs due to the early stopping monitor which showed no improvement in accuracy. Epoch 3 has an accuracy of 93% - less than model 1.

Model Accuracy and Model Loss curves are plotted using the plot_learninCurve function defined above:





Next, the overall model score on the X_train dataset is retrieved and saved to model_3_score:

```
In [437]: model_3_score = model_3.evaluate(X_train, y_train, verbose=0)

In [438]: print(f'Training Set: Test Loss: {model_3_score[0]} / Test Accuracy: {model_3_score[1]}')
Training Set: Test Loss: 0.12202907353639603 / Test Accuracy: 0.9557409286499023
```

Overall model score is 95% with 12% loss.

Next, we use the model and evaluate on the X_test dataset and save the result to model_3_evaluation:

```
In [439]: model_3_evaluation = model_3.evaluate(X_test, y_test)
208/208 [==============================] - 1s 4ms/step - loss: 0.3355 - accuracy: 0.8733

In [440]: print(f'Test Set: Test Loss: {model_3_evaluation[0]} / Test Accuracy: {model_3_evaluation[1]}')
Test Set: Test Loss: 0.33551594614982605 / Test Accuracy: 0.8733283281326294
```

The accuracy score on the test set is 87% - fairly close to the training set.

Finally, the scores are appended to Dataframe df_model_metrics, so scores can be compared across models easily:

```
In [442]: df_model_metrics = pd.concat([df_model_metrics, df_model_3_metrics], ignore_index=True)

In [443]: print(df_model_metrics)
  model_name        model_description  test_loss  test_accuracy epoch_stopped_at
0    model_1     2 layers max length   0.358455       0.874080                4
1    model_2  2 layers median length   0.416498       0.839970                3
2    model_3     3 layers max length   0.335516       0.873328                3
```

So far model 1 and 3 are producing similar results.

Model 4

Model 3 uses the same input parameters as Model 2, but adds a “hidden” layer:

```
In [444]: model_4 = Sequential()
     ...: model_4.add(Embedding(input_dim=15232, output_dim=11, input_length=6))
     ...: model_4.add(Flatten())  # https://keras.io/api/layers/reshaping_layers/flatten/
     ...: model_4.add(Dense(100, activation='relu'))
     ...: model_4.add(Dense(50, activation='relu'))
     ...: model_4.add(Dense(2, activation='softmax'))
     ...: model_4.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
     ...: print(model_4.summary())
```

Next we show the model summary which details the number of parameters per layer:

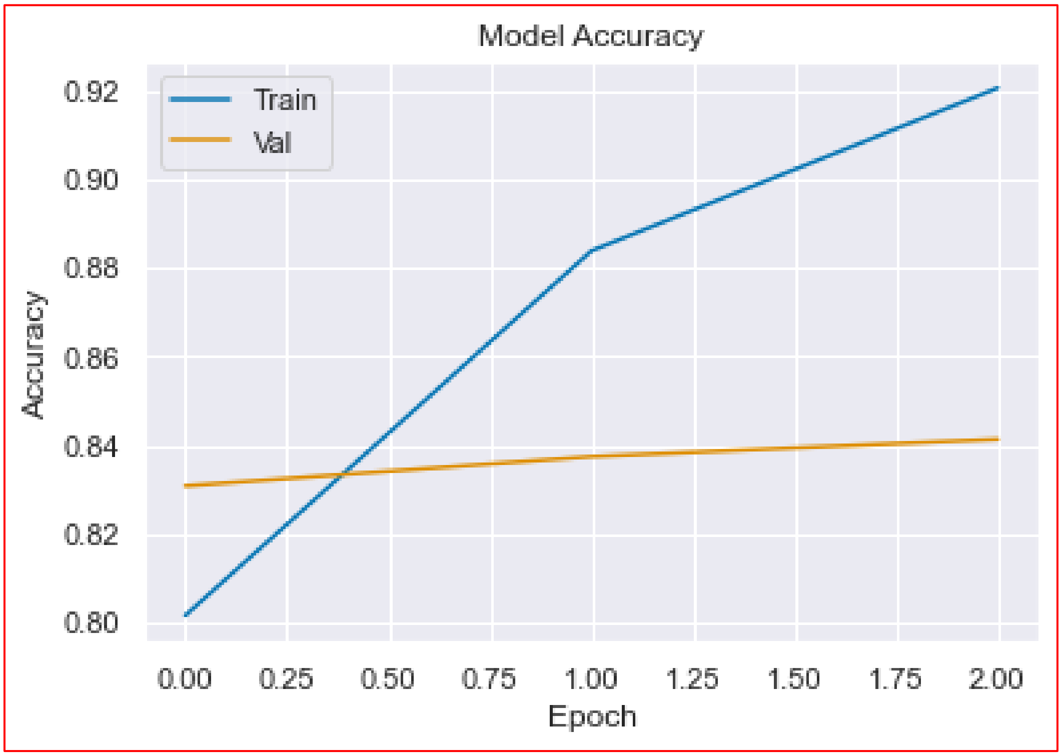
```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_14 (Embedding)    (None, 6, 11)             167552

 flatten_14 (Flatten)        (None, 66)                0

 dense_44 (Dense)            (None, 100)               6700

 dense_45 (Dense)            (None, 50)                5050

 dense_46 (Dense)            (None, 2)                 102

=================================================================
Total params: 179,404
Trainable params: 179,404
Non-trainable params: 0
_________________________________________________________________
```

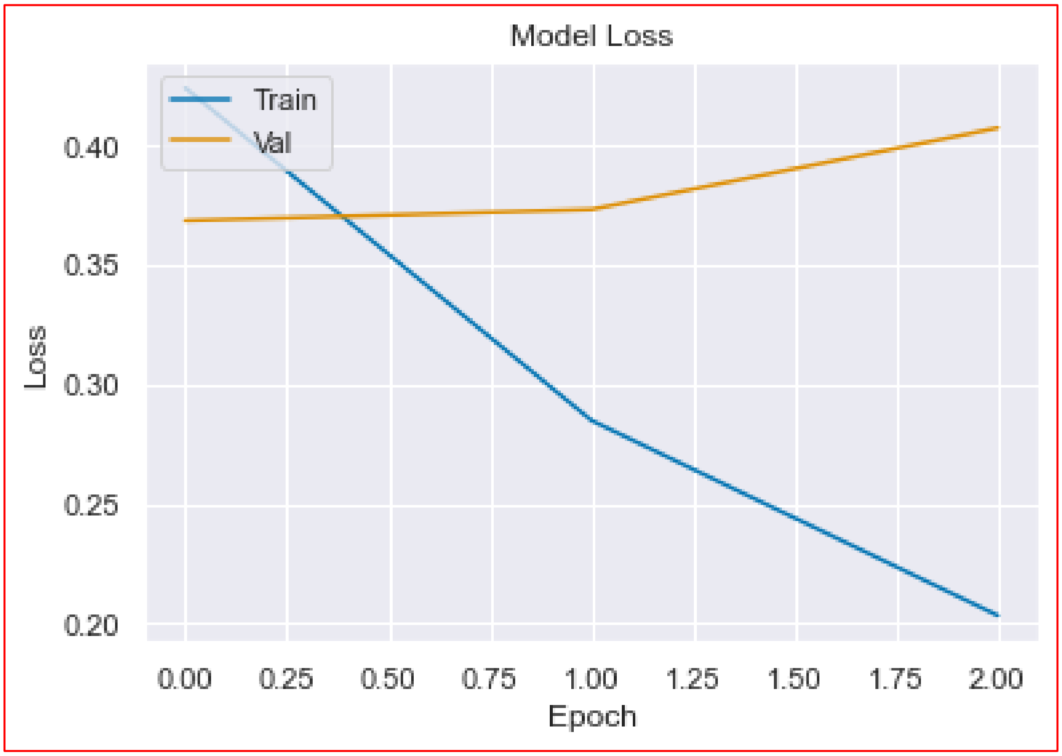
Next we execute the model on the X_train_median dataset and save the results to model_4.history:

```
In [445]: model_4_history = model_4.fit(X_train_median, y_train_median, epochs=20, batch_size=32, callbacks=[early_stopping_monitor],
verbose=True, validation_data=(X_test_median, y_test_median))
Epoch 1/20
832/832 [==============================] - 4s 4ms/step - loss: 0.4242 - accuracy: 0.8016 - val_loss: 0.3687 - val_accuracy: 0.8308
Epoch 2/20
832/832 [==============================] - 3s 4ms/step - loss: 0.2850 - accuracy: 0.8838 - val_loss: 0.3733 - val_accuracy: 0.8374
Epoch 3/20
832/832 [==============================] - 4s 5ms/step - loss: 0.2033 - accuracy: 0.9205 - val_loss: 0.4075 - val_accuracy: 0.8413

In [446]: print(model_4_history.history)
{'loss': [0.4241679906845093, 0.2850278615951538, 0.20332826673984528], 'accuracy': [0.8015854954719543, 0.8837916851043701, 0.9204613566398621],
'val_loss': [0.3686615824699402, 0.3733491599559784, 0.4074556231498718], 'val_accuracy': [0.8308039307594299, 0.8374154567718506,
0.8413223028182983]}

In [447]:
```

Even though the model was set to execute for 20 epochs, the model stopped after 3 epochs due to the early stopping monitor which showed no improvement in accuracy. Epoch 3 has an accuracy of 92% - less than model 1.

Model Accuracy and Model Loss curves are plotted using the plot_learninCurve function defined above:





Next, the overall model score on the X_train_median dataset is retrieved and saved to model_4_score:

```
In [448]: model_4_score = model_4.evaluate(X_train_median, y_train_median, verbose=0)

In [449]: print(f'Training Set: Test Loss: {model_4_score[0]} / Test Accuracy: {model_4_score[1]}')
Training Set: Test Loss: 0.14407794177532196 / Test Accuracy: 0.9508941769599915
```

Overall model score is 95% with 14% loss.

Next, we use the model and evaluate on the X_test_median dataset and save the result to model_4_evaluation:

```
In [450]: model_4_evaluation = model_4.evaluate(X_test_median, y_test_median)
208/208 [==============================] - 0s 2ms/step - loss: 0.4075 - accuracy: 0.8413

In [451]: print(f'Test Set: Test Loss: {model_4_evaluation[0]} / Test Accuracy: {model_4_evaluation[1]}')
Test Set: Test Loss: 0.4074556231498718 / Test Accuracy: 0.8413223028182983
```

The accuracy score on the test set is 84%.

Finally, the scores are appended to Dataframe df_model_metrics, so scores can be compared across models easily:

```
In [453]: df_model_metrics = pd.concat([df_model_metrics, df_model_4_metrics], ignore_index=True)

In [454]: print(df_model_metrics)
  model_name       model_description  test_loss  test_accuracy epoch_stopped_at
0    model_1      2 layers max length   0.358455       0.874080                4
1    model_2   2 layers median length   0.416498       0.839970                3
2    model_3      3 layers max length   0.335516       0.873328                3
3    model_4   3 layers median length   0.407456       0.841322                3
```

Selecting the best model

Model 1 and Model 3 have almost identical accuracy scores; however, model 3 has slightly less loss and uses one less epoch – this means Model 3 is slightly better than Model 1. In reality, either model could be used to good result.

Model's 2 and 4 – where the median review length is used – has accuracy scores with 4% of models 1 and 3, but the loss is about 7% greater.

```
In [454]: print(df_model_metrics)
  model_name       model_description  test_loss  test_accuracy epoch_stopped_at
0    model_1      2 layers max length   0.358455       0.874080                4
1    model_2   2 layers median length   0.416498       0.839970                3
2    model_3      3 layers max length   0.335516       0.873328                3
3    model_4   3 layers median length   0.407456       0.841322                3
```

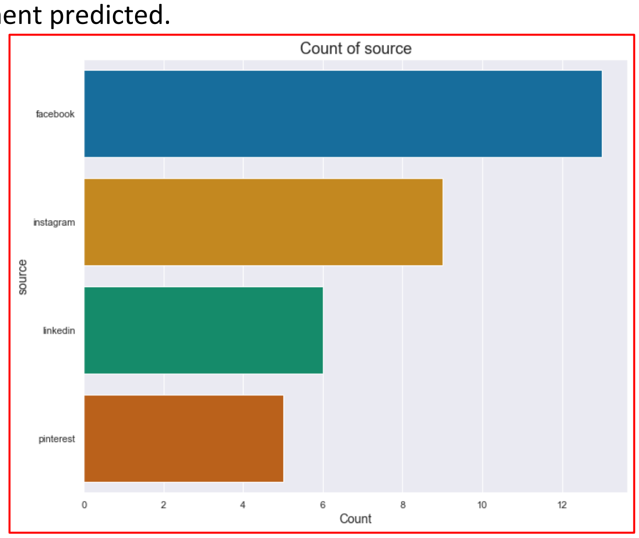
For the rest of this analysis model 3 will be used.

Applying to Social Media Reviews

Now a trained model exists that can predict positive or negative sentiment that has been trained on users reviews where the reviewer also left a star rating - effectively classifying their review for us.

Thirty-three customer comments were manually scraped from McDonald's social media pages on Facebook, Instagram, LinkedIn, and Pinterest.

In Section 10 of the code, we input these reviews into a Dataframe df_sm, the clean_review function defined above is executed to clean the reviews, the tokenized reviews are padded, and then fed into the model to have their sentiment predicted.



```
In [465]: df_sm.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 33 entries, 0 to 32
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   review_id  33 non-null     int64
 1   source     33 non-null     object
 2   date       27 non-null     object
 3   review     33 non-null     object
dtypes: int64(1), object(3)
memory usage: 1.2+ KB

In [466]: clean_review(df_sm, 'review', list_special_characters)
Longest review has 67 words.
Shortest review has 2 words.
Average review has 11 words.
all done
Out[466]: (None, None, None, None)

In [467]: print(df_sm.info())
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 33 entries, 0 to 32
Data columns (total 17 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   review_id                  33 non-null     int64
 1   source                     33 non-null     object
 2   date                       27 non-null     object
 3   review                     33 non-null     object
 4   review_lowercase           33 non-null     object
 5   review_no_punct            33 non-null     object
 6   review_no_stopwords        33 non-null     object
 7   review_no_xbf_xef          33 non-null     object
 8   review_tokenized           33 non-null     object
 9   review_padded_max          33 non-null     object
 10  review_padded_median       33 non-null     object
 11  review_len                 33 non-null     int64
 12  review_no_stopwords_len    33 non-null     int64
 13  review_no_xbf_xef_len      33 non-null     int64
 14  review_tokenized_len       33 non-null     int64
 15  review_padded_max_len      33 non-null     int64
 16  review_padded_median_len   33 non-null     int64
dtypes: int64(7), object(10)
memory usage: 4.5+ KB
None
```

```
In [468]: df_sm_padded_max = pad_sequences(df_sm['review_tokenized'], padding='post', maxlen=273)  # need to pad to 273 as that is what the model was trained on

In [469]: df_sm['review_padded_max_max'] = pd.DataFrame({'review_padded_max_max': df_sm_padded_max.tolist()})  # add the new padd back to the same df to keep everything together
```

```
In [471]: df_sm_predictions = model_3.predict(df_sm_padded_max)
2/2 [==============================] - 0s 5ms/step

In [472]: df_sm_predictions_analysis = pd.DataFrame(df_sm_predictions)  # convert to a dataframe
```

The model returns a probability that the review is negative (0) and that the review is positive (1). Each of these probabilities are stored in a separate column. For our purposes the higher probability of the two will be taken as the sentiment for each.

For the review at index 0, the review has negative sentiment. For the review at index 1 the review has positive sentiment, etc.

	0	1
0	0.980199	0.019801
1	0.49105	0.50895
2	0.868522	0.131478
3	0.723241	0.276759

A function, max_column, is defined to locate the max value for each row, and then that is used to add a column to df_sm with the predicted sentiment for each review.

```
In [473]: def max_column(row):
     ...:     """ function to look at value in two columns, and return which column has the greater value
     ...:     """
     ...:     if row[0] > row[1]:
     ...:         return 0
     ...:     else:
     ...:         return 1

In [474]: df_sm['model_3_prediction'] = df_sm_predictions_analysis.apply(max_column, axis=1)
```

Let's spot check some reviews and their predictions:

Appears accurate as a negative review:

```
In [475]: num_index = 23

In [476]: print('Origional review', df_sm['review'][num_index], '\n')
     ...: print('Predicted:', 'Negative' if df_sm_predictions[num_index][0] >= 0.5 else 'Positive', 'review')
Origional review Yesterday around 18.00, I bought a Double Big Mc for my son from your place in Yalova (Turkey) center. My son is autistic and
doesn't eat anything other than meatballs and cheese. I said nothing should be added except cheese. They said okay. I warned them once more before
the package arrived, and they called inside again. I bought the package and brought it home and there was something resembling grated onion on the
meatball and my son did not eat it. I became very angry. This isn't the first time this has happened to me. On average, they make a mistake every
3-4 times in your same store. Moreover, there was no intensity. I'm too afraid to buy anything from you now. I will express your insensitivity on
every platform.

Predicted: Negative review
```

Appears to be classified incorrectly:

```
In [477]: num_index = 16

In [478]: print('Origional review', df_sm['review'][num_index], '\n')
     ...: print('Predicted:', 'Negative' if df_sm_predictions[num_index][0] >= 0.5 else 'Positive', 'review')
Origional review dear grimace and mcdonalds, why must i purchase a whole meal in order to receive a grimace shake from your establishments?
sometimes i am not so hungry that i would eat a whole big mac and fry/ 10 pc chicken mcnugget and fry, but I still might crave the bombastic and
powerful flavor of a grimace\x92s birthday shake. why would you engage in such non consumer friendly practices and marketing tactics????

Predicted: Positive review
```

Appears to be classified correctly as positive:

```
In [482]: print('Origional review', df_sm['review'][num_index], '\n')
     ...: print('Predicted:', 'Negative' if df_sm_predictions[num_index][0] >= 0.5 else 'Positive', 'review')
Origional review McDonald\x92s is one of my favorite burgers

Predicted: Positive review
```

Calculating Customer Satisfaction Score

In section 11, we calculate the customer satisfaction score.

The total number of reviews from the Kaggle dataset that are positive and the total number of reviews from the manually scaped social media posts that are positive are calculated and stored in the variables csat_satisfied_df_reviews and csat_satisfied_df_sm respectively.

The total number of reviews from the Kaggle dataset are calculated and the total number of reviews from the manually scraped social media posts are calculated and stored in the variables csat_all_df_reviews and csat_add_df_sm respectively.

A percentage is calculated by summing the two counts of satisfied reviews, dividing by the sum of the two total counts, and multiplying by 100.

```
In [483]: csat_satisfied_df_reviews = sum(df_reviews['rating'] == 1)

In [484]: csat_satisfied_df_sm = sum(df_sm['model_3_prediction'] == 1)

In [485]: csat_all_df_reviews = len(df_reviews['rating'])

In [486]: csat_all_df_sm = len(df_sm['model_3_prediction'])

In [487]: csat = (csat_satisfied_df_reviews + csat_satisfied_df_sm) / (csat_all_df_reviews + csat_all_df_sm) * 100

In [488]: print(f'CSAT score: {round(csat,2)}%')
CSAT score: 48.01%
```

The customer satisfaction score for McDonald’s for this dataset is 48%.

Summary and Implications

As stated above the focus of this project is can a neural network model be constructed on the dataset to accurately predict customer reviews as positive or negative, allowing these predictions to be used to calculate a customer satisfaction score?

Success of the project will be measured by accepting either the null hypothesis or the alternative hypothesis. The null hypothesis is that a neural network *cannot* be constructed from the dataset to accurately predict customer reviews as positive or negative. The alternative hypothesis is that a neural network *can* be constructed from the dataset to accurately predict customer reviews as positive or negative with an accuracy greater than eighty percent (80%).

Model 3 obtained an accuracy score of 87% and as detailed above was successfully used on a dataset it had not been trained or test on and predicted the sentiment as positive or negative.

Therefore, we accept the alternative hypothesis and reject the null hypothesis – a neural network can be constructed from the dataset to accurately predict customer reviews as positive or negative.

One limitation of the analysis is the small dataset. Ideally a larger dataset would be used both for the training of the model as well as the predictions to calculate an overall customer satisfaction score.

Companies having business to business access to API's for the various social media sites would be able to ingest customer reviews more seamlessly and quicker to analyze and apply insights.

Several avenues were not explored during this project and are further areas of study:

- Reviews had location data and time. This analysis could be sliced down to individual locations for division, regional, or store leadership to act up. Similarly, customer satisfaction scores could be tracked over time and across locations to look for trends and opportunities.
- The model deployed is relatively basic. Models could be tested with more hidden layers or different review lengths between average (6) and max (273).

Appendix

Sources:

Board of Regents of the University System of Georgia. (n.d.). *A Brief History of the Internet*. Online Library Learning Center. Retrieved October 28, 2023, from: https://www.usg.edu/galileo/skills/unit07/internet07_02.phtml#:~:text=January%201%2C%201983%20is%20considered,Protocol%20(TCP%2FIP)a

SurveyMonkey. *The Ultimate Guide to Customer Satisfaction Score*. SurveyMonkey. (n.d.). Retrieved October 28, 2023, from: https://www.surveymonkey.com/resources/premium/customer-satisfaction-score-csat-guide/

Wikimedia Foundation. (2023a, October 14). *Timeline of social media*. Wikipedia. Retrieved October 28, 2023, from: https://en.wikipedia.org/wiki/Timeline_of_social_media

Datacamp: Introduction to Deep Learning in Python
D213 Task 2 Cohort Webinar PPT.pptx, slide 59

Web sources for code:
https://seaborn.pydata.org/generated/seaborn.set_theme.html
https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.describe.html
https://medium.com/mlearning-ai/sentiment-analysis-using-lstm-21767a130857
https://docs.python.org/3/library/string.html

https://favtutor.com/blogs/remove-punctuation-from-string-python
https://westerngovernorsuniversity-my.sharepoint.com/personal/william_sewell_wgu_edu/_layouts/15/onedrive.aspx?id=%2Fpersonal%2Fwilliam_sewell_wgu_edu%2FDocuments%2FDocuments%2FD213%2FWebinars%2FSentiment_Analysis_Tensorflow_2.html&parent=%2Fpersonal%2Fwilliam_sewell_wgu_edu%2FDocuments%2FDocuments%2FD213%2FWebinars&ga=1
https://keras.io/api/layers/reshaping_layers/flatten/
https://www.dataquest.io/blog/tutorial-add-column-pandas-dataframe-based-on-if-else-condition/
https://stackoverflow.com/a/77054189

Code:

```
# Corey B. Holstege

# Western Governors University

# Masters of Science in Data Analytics

# D214 Performance Assessment (NKM2)

# file name: D214_NKM2_Submission_Holstege.py

# %% [1] IMPORT LIBRARIES

# import os  # https://docs.python.org/3/library/os.html

import re

import sys  # https://docs.python.org/3/library/sys.html

import pandas as pd  # https://pandas.pydata.org/docs/index.html

import numpy as np  # https://numpy.org/doc/stable/

import missingno as msno  # https://github.com/ResidentMario/missingno

import string  # for punctuation

import nltk  # natural language tookkit


from sklearn.model_selection import train_test_split  # to split the dataset into test and train

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.layers import Embedding

from tensorflow.keras.layers import Flatten

from tensorflow.keras.preprocessing.sequence import pad_sequences  # to pad the numbers with zeros

from tensorflow.keras.preprocessing.text import Tokenizer  # convert text to tokens (numbers)
```

```
from tensorflow.keras.callbacks import EarlyStopping  # stop training at a threshold
from wordcloud import WordCloud
from wordcloud import STOPWORDS

# nltk.download('stopwords')
nltk.download('punkt')

import matplotlib
from matplotlib import pyplot as plt
import seaborn as sns
sns.set_theme(style="darkgrid", palette="colorblind")  #
https://seaborn.pydata.org/generated/seaborn.set_theme.html
import warnings
warnings.filterwarnings('ignore')


# %%%[1.1] CHECK VERSIONS
print('Python version: ', sys.version, '\n')
print('Pandas version: ', pd.__version__, '\n')
print('Numpy version: ', np.__version__, '\n')
print('Missingno version: ', msno.__version__, '\n')
print('Matplotlib version: ', matplotlib.__version__, '\n')
print('Seaborn version: ', sns.__version__, '\n')
print('Tensorflow version', tf.__version__, '\n')

# %% [2] GET THE DATA: IMPORT CSV FILE
# error: UnicodeDecodeError: 'utf-8' codec can't decode bytes in position 1432-1433:
invalid continuation byte
# citation: https://stackoverflow.com/a/51763708
file_encoding = 'utf8'
input_fd =
open(r'C:\Users\K2Admin\OneDrive\Documents\WGUMSDA\D214\PA\McDonald_s_Reviews.csv',
encoding=file_encoding, errors='backslashreplace')
df = pd.read_csv(input_fd)
```

```
# %% [3] INITIAL DATA EXAMINATION
print('Begin section: Data Set Examination \n')

# show size of the dataset
print('Number of rows/columns: ' + str(df.shape), '\n')
# notes: 33,396 rows, 10 columns

# count how many times each data type is present in the datset
print('Count of each datatype: \n', pd.value_counts(df.dtypes), '\n')
# notes: object:7; float64: 2; int64: 1

# set the display width to be larger so we can read the text
# pd.set_option('display.max_colwidth', 5000)
# pd.set_option('display.max_columns', None)
print('View first seven rows of the dataframe: \n', df.head(7), '\n')
print('View last seven rows of the dataframe: \n', df.tail(7), '\n')
# notes:

# %%%[3.1] EXAMINE COLUMNS
print('View column info: \n', df.info(), '\n')
# notes: lat and long are missing values

# %%%[3.2] VIEW STATISTICAL INFO
df_describe = df.describe(include='all')
print('View statistical info: \n', df_describe, '\n')
# notes:
# citation: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.describe.html
# print dataframe description

# %%%[3.3] CHECK FOR MISSING VALUES
missing_any = df.isnull().any()  # missing value: boolean value
missing_sum = df.isnull().sum()  # count of missing values
```

```
missing_df = pd.concat([missing_any, missing_sum], axis=1)  # concatenate the two
series into a dataframe

missing_df.reset_index(inplace=True)  # reset the index

missing_df.columns = ['column_name', 'any_missing', 'total_missing']  # rename the
columns

print(missing_df)


# notes: lat and long are both missing 660 values


msno.bar(df, labels=True)

# notes: visualization confirms no missing values


# %%[4] EDA: NOT REVIEW COLUMN

# Dictionary to store frequency tables

freq_tables = {}



def eda_analysis(df, column_name):
    """

    """

    # Print column name, number of unique values, and datatype

    print('-' * 75)

    print('Begin ' + column_name + ':')

    print('Number of unique values: ', df[column_name].nunique(), '\n')

    print('Datatype of the column: ', df[column_name].dtypes, '\n')


    # Print summary statistics for column

    print('Describe ' + column_name)

    print(df[column_name].describe(include=all), '\n')


    # Create frequency table

    freq_table = pd.concat([df[column_name].value_counts(dropna=False),

                            (df[column_name].value_counts(dropna=False, normalize=True)
* 100).round(2)],
```

```
                          axis=1,
                          keys=['Count', 'Percentages'])
    # Add frequency table to dictionary
    freq_tables[column_name] = freq_table

    # Create countplot for column
    fig, ax = plt.subplots(figsize=(10, 8))
    sns.countplot(y=column_name, data=df, order=df[column_name].value_counts().index)
    plt.xlabel('Count', size=14)
    plt.ylabel(column_name, size=14)
    plt.title('Count of ' + column_name, size=18)
    plt.tight_layout()
    plt.show()
    plt.close()  # Close the figure

    # Print frequency table
    print(' ')
    print('Frequency table for variable ' + column_name, ':\n', freq_table, '\n')
    print('End ' + column_name + '\n')


print(eda_analysis(df, 'store_name'))
print(eda_analysis(df, 'category'))
print(eda_analysis(df, 'store_address'))
print(eda_analysis(df, 'latitude '))
print(eda_analysis(df, 'longitude'))
print(eda_analysis(df, 'rating_count'))
print(eda_analysis(df, 'review_time'))
print(eda_analysis(df, 'rating'))

# notes:
    # store_name: only two values, one of those has randome characters at the front
    # category: only has one value
```

```
    # store_address: 40 unique; 2476 Kal has random characters for the remainder (660
values)

    # latitude: 39 unique values (but store address has 40 so there should be 40 here);
660 are NaN

    # longtitude: 39 unique values (but store address has 40 so there should be 40
here); 660 are NaN

    # rating_count: object, not an integer, 51 unique #'s. shouldn't there be a unique
count per store?

    # review_time: object, not datetime, most reviews are 2-6 years ago

    # rating: object, not integer, mostreviews 1 or 5 stars


# print('Datatype of the column: ', df['latitude'].dtypes, '\n')

# print('Number of unique values: ', df['latitude'].nunique(), '\n')

# print(list(df.columns))


# %%[5] EDA: REVIEW

# %%%[5.1] WORDCLOUDS

# citation: https://medium.com/mlearning-ai/sentiment-analysis-using-lstm-21767a130857

def generate_wordcloud(data, title):

    wordcloud = WordCloud(

        stopwords=STOPWORDS,

        max_words=100,

        max_font_size=40,

        scale=4).generate(str(data))

    fig = plt.figure(1, figsize=(15, 15))

    plt.axis('off')

    plt.imshow(wordcloud)

    plt.title(title)

    plt.show()



generate_wordcloud(df['review'], "All Reviews")


# generate a wordcloud for the positive sentiment and wordcloud for negative sentiment?
```

```
df_positive = df[['rating', 'review']].loc[(df['rating'] == '4 stars') | (df['rating']
== '5 stars')]

df_negative = df[['rating', 'review']].loc[(df['rating'] == '1 star') | (df['rating']
== '2 stars') | (df['rating'] == '3 stars')]


generate_wordcloud(df_positive['review'], "Positive Reviews")

generate_wordcloud(df_negative['review'], "Negative Reviews")


# %%[5.2] CHECK FOR SPECIAL CHARACTERS

pd.set_option('display.max_colwidth', 5000)

print('View random 21 rows of the dataframe: \n', df['review'].sample(21), '\n')

# notes: upper case, lower clase, punctation,


# to-do: update this function to use the bar chart formatting code from section 4

# define a function to plot a bar chart of character counts

def character_chart(df, column_x, column_y):

    """Return a bar chart of chararacter counts."""

    plt.figure(figsize=(12, 6))  # Adjust the figure size as needed

    plt.bar(df[column_x], df[column_y])

    #plt.barh(df[column_x], df[column_y])

    plt.xlabel('Character')

    plt.ylabel('Count')

    plt.title('Character Counts')

    plt.show()

    plt.close()



# print(character_chart.__doc__)  # print function doc string



def extract_unique_characters(df, column_name):

    """

    # Citation Dr.Ellah Festus D213 Task 2 Cohort Webinar (converted into a function)
```

```
    # function to input a dataframe and column name and output a list of unique characters found in the specified column

    """

    list_of_characters = []

    for review in df[column_name]:

        for character in review:

            if character not in list_of_characters:

                list_of_characters.append(character)

    return list_of_characters



# review list of unique characters

print(extract_unique_characters(df, 'review'))

# notes: special characters: '?', '\n'',',','.','/', '*', "'",'�','\\',':','#', '"','&', '-', '!','(', ')','%', '+', '$',';', '[', '_','~', '@', '=', '<', ']', '>', '{', '}', '^'


# what am i doing with these??

special_char_1 = string.punctuation

special_char_2 = extract_unique_characters(df, 'review')


# print just special characters

print(string.ascii_letters)  # citation: https://docs.python.org/3/library/string.html



def extract_special_character_counts_df(df, column_name):

    """extract all special characters and a count of how many times the speical character appears

       store in a dataframe

    """

    special_character_counts = {}

    for review in df[column_name]:

        for character in review:

            if character not in string.ascii_letters:
```

```
                if character not in special_character_counts:
                    special_character_counts[character] = 1
                else:
                    special_character_counts[character] += 1

    # Convert the dictionary to a DataFrame
    df_special_character_counts = pd.DataFrame(list(special_character_counts.items()),
columns=['character', 'count'])
    return df_special_character_counts



df_special_character_counts = extract_special_character_counts_df(df, 'review')  #
store in a dataframe
df_special_character_counts = df_special_character_counts.sort_values(by='count',
ascending=False)  # sort the values in the dataframe
character_chart(df_special_character_counts, 'character', 'count')  # generate the bar
chart

# extract the special characters and store in a list for later
list_special_characters = df_special_character_counts['character'].tolist()
print(list_special_characters)

# notes: over 700k spaces, what does it look like when excluding spaces?
df_special_character_counts = df_special_character_counts.drop(index=0)  # drop the row
for space
character_chart(df_special_character_counts, 'character', 'count')  # generate the bar
chart

# notes: over 80k weird triangle/square, what does it look like when excluding this?
df_special_character_counts = df_special_character_counts.drop(index=8)  # drop the row
for weird character
character_chart(df_special_character_counts, 'character', 'count')  # generate the bar
chart

# %%[5.3] EXTRACT CHARACTER COUNTS
```

```
def extract_character_counts_df(df, column_name):
    """extract all characters and a count of how many times the character appears
       store in a dataframe
    """
    character_counts = {}
    for review in df[column_name]:
        for character in review:
            if character in character_counts:
                character_counts[character] += 1
            else:
                character_counts[character] = 1

    # Convert the dictionary to a DataFrame
    df_character_counts = pd.DataFrame(list(character_counts.items()),
columns=['character', 'count'])
    return df_character_counts



df_character_counts = extract_character_counts_df(df, 'review')  # store in a dataframe
df_character_counts = df_character_counts.sort_values(by='count', ascending=False)  #
sort the values in the dataframe
# print(df_character_counts.sort_values(by='count', ascending=False))
print(df_character_counts)


character_chart(df_character_counts, 'character', 'count')  # generate the bar chart
# notes: the number of spaces are having an outsized effect on the chart
df_character_counts = df_character_counts.drop(index=3)  # spaces are in index row 3,
drop this from the dataframe
character_chart(df_character_counts, 'character', 'count')  # replot the bar chart


# %%%[5.4] EXTRACT LETTERS
```

```python
def extract_alphabet_counts_df(df, column_name):
    """extract all alphabet characters and a count of how many times the letter appears
       store in a dataframe
    """
    character_counts = {}
    for review in df[column_name]:
        for character in review:
            if character in string.ascii_letters:
                if character in character_counts:
                    character_counts[character] += 1
                else:
                    character_counts[character] = 1

    df_alphabet_counts = pd.DataFrame(list(character_counts.items()),
columns=['character', 'count'])
    return df_alphabet_counts



df_alphabet_counts = extract_alphabet_counts_df(df, 'review')  # store in a dataframe
df_alphabet_counts = df_alphabet_counts.sort_values(by='count', ascending=False)  #
sort the values in the dataframe
character_chart(df_alphabet_counts, 'character', 'count')  # generate the bar chart


# %%[5.5] EXTRACT NUMBERS



def extract_numbers_counts_df(df, column_name):
    """extract all numbers and a count of how many times the number appears
       store in a dataframe
    """
    number_counts = {}
    for review in df[column_name]:
        for number in review:
            if number in string.digits:
```

```
                if number in number_counts:
                    number_counts[number] += 1
                else:
                    number_counts[number] = 1

    df_number_counts = pd.DataFrame(list(number_counts.items()), columns=['character',
'count'])
    return df_number_counts



df_number_counts = extract_numbers_counts_df(df, 'review')  # store in a dataframe
df_number_counts = df_number_counts.sort_values(by='count', ascending=False)  # sort
the values in the dataframe
character_chart(df_number_counts, 'character', 'count')  # generate the bar chart


# %%[5.6] EXTRACT STOP WORDS
print(STOPWORDS)



def extract_stopwords_counts_df(df, column_name):
    """extract all stop words and a count of how many times the stop word appears
       store in a dataframe
    """
    stop_word_count = {}
    for review in df[column_name]:
        words = review.split()
        for word in words:
            if word in STOPWORDS:
                if word in stop_word_count:
                    stop_word_count[word] += 1
                else:
                    stop_word_count[word] = 1
```

```
    df_stopword_counts = pd.DataFrame(list(stop_word_count.items()),
columns=['character', 'count'])

    return df_stopword_counts



df_stopword_counts = extract_stopwords_counts_df(df, 'review')  # store in a dataframe

df_stopword_counts = df_stopword_counts.sort_values(by='count', ascending=False)  #
sort the values in the dataframe

character_chart(df_stopword_counts, 'character', 'count')  # generate the bar chart.
better as a barh chart

print('There are', df_stopword_counts['count'].sum(), 'stopwords.')


# %%%[5.7] EXTRACT CONTRACTIONS

# count how many contractions there are

# see how many of these contractions are in STOPWORDS



def extract_apostrophe_counts_df(df, column_name):

    apostrophe_count = {}


    for review in df[column_name]:

        words = review.split()  # Tokenize the text into words


        for word in words:

            if "'" in word:  # Check if the word contains an apostrophe

                if word in apostrophe_count:

                    apostrophe_count[word] += 1

                else:

                    apostrophe_count[word] = 1


    df_apostrophe_counts = pd.DataFrame(list(apostrophe_count.items()),
columns=['word', 'count'])

    return df_apostrophe_counts
```

```
df_apostrophe_counts = extract_apostrophe_counts_df(df, 'review')  # store in a dataframe

df_apostrophe_counts = df_apostrophe_counts.sort_values(by='count', ascending=False)  # sort the values in the dataframe


# %%[5.8] VOCAB SIZE

word_tokenizer = Tokenizer()

# review_length = []


# need to set variables outside the function for review_max, review_min, and review_median so they can be used

def vocab_size_sequence_length(df, column_name, header):
    """
    """
    word_tokenizer.fit_on_texts(df[column_name])
    vocab_size = len(word_tokenizer.word_index) + 1  # how many unique words


    review_length = []
    for char_len in df[column_name]:
        review_length.append(len(char_len.split(' ')))


    review_max = np.max(review_length)
    review_min = np.min(review_length)
    review_median = np.median(review_length)


    return print(header), print('Vocab size (unique words) is: ', vocab_size), print('Longest review has', review_max, 'words.'), print('Shortest review has', review_min, 'words.'), print('Average review has', review_median, 'words.')




print(vocab_size_sequence_length(df, 'review', 'Review dirty and not preprocessed.'))

# notes:

# vocab size (unique words): 15,114

# longest review has 584 words
```

```
# shortest review has 1 word

# average # of words per review is 11



# %%[6] SPLIT DATAFRAME DF


df_store_info = df[['store_address', 'latitude ', 'longitude', 'rating_count']].copy()

df_reviews = df[['reviewer_id', 'review_time', 'rating', 'review']].copy()


# %%[7.0] CLEAN AND PREPROCESS

# %%%[7.1] CLEAN DF_STORE_INTO

# clean up the store master data

# citation: https://www.geeksforgeeks.org/delete-duplicates-in-a-pandas-dataframe-based-on-two-columns/

df_store_info = df_store_info.drop_duplicates(subset=['store_address', 'latitude ', 'longitude', 'rating_count'], keep='first').reset_index(drop=True)

# some address are in the list twice with different rating counts - the rating count if off by a small number

df_store_info = df_store_info.drop_duplicates(subset=['store_address'], keep='first').reset_index(drop=True)


print(df_store_info.info())

# notes: rating_count column is a number with integer, lat and long at each missing 1


# Remove commas and convert to integer

df_store_info['rating_count'] = df_store_info['rating_count'].str.replace(',', '', regex=True).astype(int)


# double check

print(df_store_info.info())


# to-do:

# didnt create a linkage between the two df's for store master data


# %%%[7.2] CLEAN DF_REVIEWS
```

```
# %%%%[7.2.1] PREPROCESS COLUMN: RATING

# create a dictionary to map to reivews ratings to 0 (negative sentiment) or 1
(positive sentiment)

rating_mapping = {'1 star': 0,

                  '2 stars': 0,

                  '3 stars': 0,

                  '4 stars': 1,

                  '5 stars': 1}

df_reviews['rating'] = df_reviews['rating'].map(rating_mapping)  # update the column in
df_reviews

print(eda_analysis(df_reviews, 'rating'))



# %%%%[7.2.2] PREPROCESS COLUMN: REVIEW_TIME


print(eda_analysis(df_reviews, 'review_time'))


# define a dictionary mapping for time units to dates

time_mapping = {'review_time':

                {'6 hours ago': '2023-10-01',

                    '8 hours ago': '2023-10-01',

                    '20 hours ago': '2023-10-01',

                    '21 hours ago': '2023-10-01',

                    '22 hours ago': '2023-10-01',

                    '23 hours ago': '2023-10-01',

                    'a day ago': '2023-10-01',

                    '2 days ago': '2023-10-01',

                    '3 days ago': '2023-10-01',

                    '4 days ago': '2023-10-01',

                    '5 days ago': '2023-10-01',

                    '6 days ago': '2023-10-01',

                    'a week ago': '2023-10-01',

                    '2 weeks ago': '2023-10-01',

                    '3 weeks ago': '2023-10-01',
```

```
                    '4 weeks ago': '2023-10-01',
                    'a month ago': '2023-09-01',
                    '2 months ago': '2023-08-01',
                    '3 months ago': '2023-07-01',
                    '4 months ago': '2023-06-01',
                    '5 months ago': '2023-05-01',
                    '6 months ago': '2023-04-01',
                    '7 months ago': '2023-03-01',
                    '8 months ago': '2023-02-01',
                    '9 months ago': '2023-01-01',
                    '10 months ago': '2022-12-01',
                    '11 months ago': '2022-11-01',
                    'a year ago': '2022-10-01',
                    '2 years ago': '2021-10-01',
                    '3 years ago': '2020-10-01',
                    '4 years ago': '2019-10-01',
                    '5 years ago': '2018-10-01',
                    '6 years ago': '2017-10-01',
                    '7 years ago': '2016-10-01',
                    '8 years ago': '2015-10-01',
                    '9 years ago': '2014-10-01',
                    '10 years ago': '2013-10-01',
                    '11 years ago': '2012-10-01',
                    '12 years ago': '2011-10-01'}}


df_reviews.replace(time_mapping, inplace=True)  # replace the values in the column using the dictionary defined above

df_reviews['review_time'] = pd.to_datetime(df_reviews['review_time'])  # change the datatype of the column

print(eda_analysis(df_reviews, 'review_time'))


# %%%%[7.2.3] PREPROCESS COLUMN: REVIEW
```

```python
def clean_review(df, column_name, punct_list):
    """
    """

    df['review_lowercase'] = df[column_name].str.lower()  # convert text to lowercase
    df['review_no_punct'] = df['review_lowercase'].apply(lambda text: ''.join([' ' if
char in punct_list else char for char in text]))  # remove punctation
    df['review_no_stopwords'] = df['review_no_punct'].apply(lambda x: ' '.join([word
for word in x.split() if word not in (STOPWORDS)]))  # remove stopwords
    df['review_no_xbf_xef'] = df['review_no_stopwords'].str.replace(r'xbf|xef|xfd', '',
regex=True)  # remove xbf and xef
    df['review_tokenized'] = word_tokenizer.texts_to_sequences(df['review_no_xbf_xef'])
# convert sentences to numeric counterpart

    review_length = []
    for char_len in df['review_no_xbf_xef']:
        review_length.append(len(char_len.split(' ')))

    review_max = np.max(review_length)
    review_min = np.min(review_length)
    review_median = int(np.median(review_length))

    padded_sequences_max = pad_sequences(df['review_tokenized'], padding='post',
maxlen=review_max)  # add zeros to the end to set them all to the same length
    df['review_padded_max'] = pd.DataFrame({'review_padded_max':
padded_sequences_max.tolist()})

    padded_sequences_median = pad_sequences(df['review_tokenized'], padding='post',
maxlen=review_median)  # add zeros to the end to set them all to the same length
    df['review_padded_median'] = pd.DataFrame({'review_padded_max':
padded_sequences_median.tolist()})

    df['review_len'] = df[column_name].str.len()  # original review length
    df['review_no_stopwords_len'] = df['review_no_stopwords'].str.len()  # length after
stop word removal
    df['review_no_xbf_xef_len'] = df['review_no_xbf_xef'].str.len()  # length after
removing xbf and xef
```

```
    df['review_tokenized_len'] = df['review_tokenized'].str.len()

    df['review_padded_max_len'] = df['review_padded_max'].str.len()

    df['review_padded_median_len'] = df['review_padded_median'].str.len()

    # df['len_diff'] = df['review_len'].sub(df['review_no_stopwords_len'])  #
https://www.tutorialspoint.com/how-to-subtract-two-columns-in-pandas-dataframe


    return print('Longest review has', review_max, 'words.'), print('Shortest review
has', review_min, 'words.'), print('Average review has', review_median, 'words.'),
print('all done')



clean_review(df_reviews, 'review', list_special_characters)

print(df_reviews.info())



df_reviews['review_padded_max_str'] = df_reviews['review_padded_max']  # create a new
column that is a copy of review_padded_max

df_reviews['review_padded_max_str'] = df_reviews['review_padded_max_str'].astype(str)
# type the new column as a str for the function



def has_all_zeros(value):
    """check to see if all values are zero
    """
    pattern = r'^\[0+(?:,\s*0+)*\]$'
    return bool(re.match(pattern, value))



print(df_reviews.shape)

df_blank_rows = df_reviews[df_reviews['review_padded_max_str'].apply(has_all_zeros)]  #
copy the rows with zeros to a new dataframe

df_reviews_indicies_to_drop = df_blank_rows.index  # create a list of the index to drop
(of the rows that were moved)

df_reviews = df_reviews.drop(df_reviews_indicies_to_drop)  # drop these rows from
df_reviews

print(df_reviews.shape)
```

```
# notes:

# longest review is: 273 words

# shortest review is: 1 word

# average review is: 6 words


# while building the clean_review function these values were notices. these lines hlped
determine if occurance was significant enough for removal

# Count the occurrences of 'xbf' or 'xef' in the 'reviews' column

count_xbf_pre = df_reviews['review'].str.count(r'xbf').sum()

count_xef_pre = df_reviews['review'].str.count(r'xef').sum()

count_xfd_pre = df_reviews['review'].str.count(r'xfd').sum()


print("Total count of 'xbf' in the 'reviews' column before cleaning:", count_xbf_pre)

print("Total count of 'xef' in the 'reviews' column before cleaning:", count_xef_pre)

print("Total count of 'xfd' in the 'reviews' column before cleaning:", count_xfd_pre)


count_xbf_post = df_reviews['review_no_xbf_xef'].str.count(r'xbf').sum()

count_xef_post = df_reviews['review_no_xbf_xef'].str.count(r'xef').sum()

count_xfd_post = df_reviews['review_no_xbf_xef'].str.count(r'xfd').sum()


print("Total count of 'xbf' in the 'reviews' column after cleaning:", count_xbf_post)

print("Total count of 'xef' in the 'reviews' column before cleaning:", count_xef_post)

print("Total count of 'xfd' in the 'reviews' column before cleaning:", count_xfd_post)


print(vocab_size_sequence_length(df_reviews, 'review_no_xbf_xef', 'Review preprocessed
and clean.'))

# vocabsize = 15232


embedding_dimension = int(round(np.sqrt(np.sqrt(15232)), 0))  # 4th squart root of the
vocab size

print(f'embedding dimension {embedding_dimension}')

# 11
```

```python
# %%[8] TRAIN TEST SPLIT

# %%%[8.1] MAX LENGTH

# set X and y for train/test split using review padded to 273 tokens

X = df_reviews['review_padded_max']

y = df_reviews['rating']


# split the data set into train and test sets with 80/20 split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,
random_state=36, stratify=y)


print('Training size: ', X_train.shape, '\n')

print('Test size: ', X_test.shape, '\n')


# how can we make this a function?

print('y_test sentiment counts: ', y_test.value_counts())

y_test_counts = y_test.value_counts()

plt.bar(y_test_counts.index, y_test_counts.values)

plt.xlabel('Sentiment')

plt.ylabel('Count')

plt.title('Count of 0\'s and 1\'s in y_test')

plt.xticks(y_test_counts.index, ['0', '1'])

plt.show()

plt.close()


print('y_train sentiment counts', y_train.value_counts())

y_train_counts = y_train.value_counts()

plt.bar(y_train_counts.index, y_train_counts.values)

plt.xlabel('Sentiment')

plt.ylabel('Count')

plt.title('Count of 0\'s and 1\'s in y_train')

plt.xticks(y_train_counts.index, ['0', '1'])

plt.show()

plt.close()
```

```python
# %%%[8.2] MEDIAN LENGTH
# set X and y for train/test split using review padded to 6 tokens
X_median = df_reviews['review_padded_median']
y_median = df_reviews['rating']

# split the data set into train and test sets with 80/20 split
X_train_median, X_test_median, y_train_median, y_test_median =
train_test_split(X_median, y_median, test_size=0.20, random_state=36, stratify=y)

print('Training size: ', X_train_median.shape, '\n')
print('Test size: ', X_test_median.shape, '\n')

print('y_test_median sentiment counts: ', y_test_median.value_counts())
y_test_median_counts = y_test_median.value_counts()
plt.bar(y_test_median_counts.index, y_test_median_counts.values)
plt.xlabel('Sentiment')
plt.ylabel('Count')
plt.title('Count of 0\'s and 1\'s in y_test')
plt.xticks(y_test_median_counts.index, ['0', '1'])
plt.show()
plt.close()

print('y_train_median sentiment counts', y_train_median.value_counts())
y_train_median_counts = y_train_median.value_counts()
plt.bar(y_train_median_counts.index, y_train_median_counts.values)
plt.xlabel('Sentiment')
plt.ylabel('Count')
plt.title('Count of 0\'s and 1\'s in y_train')
plt.xticks(y_train_median_counts.index, ['0', '1'])
plt.show()
plt.close()
```

```
# %%[9] CREATE THE MODEL

# %%%[9.1]CREATE SOME THINGS THAT WILL BE USED

# %%%%[9.1.1] LEARNING CURVE FUNCTION

# plot training and validatoin accuracy scores

# citation: https://westerngovernorsuniversity-my.sharepoint.com/personal/william_sewell_wgu_edu/_layouts/15/onedrive.aspx?id=%2Fpersonal%2Fwilliam_sewell_wgu_edu%2FDocuments%2FDocuments%2FD213%2FWebinars%2FSentiment_Analysis_Tensorflow_2.html&parent=%2Fpersonal%2Fwilliam_sewell_wgu_edu%2FDocuments%2FDocuments%2FD213%2FWebinars&ga=1

def plot_learningCurve(history):

    plt.plot(history.history['accuracy'])

    plt.plot(history.history['val_accuracy'])

    plt.title('Model Accuracy')

    plt.ylabel('Accuracy')

    plt.xlabel('Epoch')

    plt.legend(['Train', 'Val'], loc='upper left')

    plt.show()

    plt.plot(history.history['loss'])

    plt.plot(history.history['val_loss'])

    plt.title('Model Loss')

    plt.ylabel('Loss')

    plt.xlabel('Epoch')

    plt.legend(['Train', 'Val'], loc='upper left')

    plt.show()



# %%%%[9.1.2] CREATE DATAFRAME TO HOLD METRICS


df_model_metrics = pd.DataFrame({

    'model_name': [],

    'model_description': [],

    'test_loss': [],

    'test_accuracy': [],

    'epoch_stopped_at': []
```

```
    })



# %%%[9.2] DEFINE EARLY STOPPING MONITOR
# Early Stopping Monitor
early_stopping_monitor = EarlyStopping(patience=2)  # model will stop after 2 epochs of no improvement


# %%%[9.3] MODEL 1 MAX LENGTH
# citation: https://stackoverflow.com/a/77054189
X_train = np.array(X_train.tolist())
X_test = np.array(X_test.tolist())


# print(X_train.info())
# print([i for i, x in enumerate(X_train) if len(x) != 273])  # citation: https://stackoverflow.com/a/49284700, check length of arrays


model_1 = Sequential()
model_1.add(Embedding(input_dim=15232, output_dim=11, input_length=273))
model_1.add(Flatten())  # https://keras.io/api/layers/reshaping_layers/flatten/
model_1.add(Dense(100, activation='relu'))
model_1.add(Dense(50, activation='relu'))
model_1.add(Dense(2, activation='softmax'))
model_1.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
print(model_1.summary())


model_1_history = model_1.fit(X_train, y_train, epochs=20, batch_size=32, callbacks=[early_stopping_monitor], verbose=True, validation_data=(X_test, y_test))
print(model_1_history.history)
# show the plot
plot_learningCurve(model_1_history)


model_1_score = model_1.evaluate(X_train, y_train, verbose=0)
```

```
print(f'Training Set: Test Loss: {model_1_score[0]} / Test Accuracy: 
{model_1_score[1]}')


# evaluate model against the test dataset
model_1_evaluation = model_1.evaluate(X_test, y_test)
print(f'Test Set: Test Loss: {model_1_evaluation[0]} / Test Accuracy: 
{model_1_evaluation[1]}')


# add metrics to metrics dataframe
df_model_1_metrics = pd.DataFrame({
    'model_name': ['model_1'],
    'model_description': ['2 layers max length'],
    'test_loss': [model_1_evaluation[0]],
    'test_accuracy': [model_1_evaluation[1]],
    'epoch_stopped_at': ['4']
})


df_model_metrics = pd.concat([df_model_metrics, df_model_1_metrics], ignore_index=True)
print(df_model_metrics)


# %%%[9.4] MODEL 2 MEDIAN LENGTH
# citation: https://stackoverflow.com/a/77054189
X_train_median = np.array(X_train_median.tolist())
X_test_median = np.array(X_test_median.tolist())


model_2 = Sequential()
model_2.add(Embedding(input_dim=15232, output_dim=11, input_length=6))
model_2.add(Flatten())  # https://keras.io/api/layers/reshaping_layers/flatten/
model_2.add(Dense(100, activation='relu'))
model_2.add(Dense(50, activation='relu'))
model_2.add(Dense(2, activation='softmax'))
model_2.compile(optimizer='adam', loss='sparse_categorical_crossentropy', 
metrics=['accuracy'])
print(model_2.summary())
```

```
model_2_history = model_2.fit(X_train_median, y_train_median, epochs=20, batch_size=32,
callbacks=[early_stopping_monitor], verbose=True, validation_data=(X_test_median,
y_test_median))

print(model_2_history.history)

# show the plot

plot_learningCurve(model_2_history)


model_2_score = model_2.evaluate(X_train_median, y_train_median, verbose=0)

print(f'Training Set: Test Loss: {model_2_score[0]} / Test Accuracy:
{model_2_score[1]}')


# evaluate model against the test dataset

model_2_evaluation = model_2.evaluate(X_test_median, y_test_median)

print(f'Test Set: Test Loss: {model_2_evaluation[0]} / Test Accuracy:
{model_2_evaluation[1]}')


# add metrics to metrics dataframe

df_model_2_metrics = pd.DataFrame({

    'model_name': ['model_2'],

    'model_description': ['2 layers median length'],

    'test_loss': [model_2_evaluation[0]],

    'test_accuracy': [model_2_evaluation[1]],

    'epoch_stopped_at': ['3']

})


df_model_metrics = pd.concat([df_model_metrics, df_model_2_metrics], ignore_index=True)

print(df_model_metrics)



# %%%[9.5] MODEL 3: MAX LENGTH & ADD LAYER

model_3 = Sequential()

model_3.add(Embedding(input_dim=15232, output_dim=11, input_length=273))

model_3.add(Flatten())  # https://keras.io/api/layers/reshaping_layers/flatten/
```

```
model_3.add(Dense(100, activation='relu'))

model_3.add(Dense(50, activation='relu'))

model_3.add(Dense(25, activation='relu'))

model_3.add(Dense(2, activation='softmax'))

model_3.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

print(model_3.summary())


model_3_history = model_3.fit(X_train, y_train, epochs=20, batch_size=32,
callbacks=[early_stopping_monitor], verbose=True, validation_data=(X_test, y_test))

print(model_3_history.history)

# show the plot

plot_learningCurve(model_3_history)


model_3_score = model_3.evaluate(X_train, y_train, verbose=0)

print(f'Training Set: Test Loss: {model_3_score[0]} / Test Accuracy:
{model_3_score[1]}')


# evaluate model against the test dataset

model_3_evaluation = model_3.evaluate(X_test, y_test)

print(f'Test Set: Test Loss: {model_3_evaluation[0]} / Test Accuracy:
{model_3_evaluation[1]}')


# add metrics to metrics dataframe

df_model_3_metrics = pd.DataFrame({

    'model_name': ['model_3'],

    'model_description': ['3 layers max length'],

    'test_loss': [model_3_evaluation[0]],

    'test_accuracy': [model_3_evaluation[1]],

    'epoch_stopped_at': ['3']

})


df_model_metrics = pd.concat([df_model_metrics, df_model_3_metrics], ignore_index=True)

print(df_model_metrics)
```

```
# %%%[9.6] MODEL 4: MEDIAN LENGTH & ADD LAYER

model_4 = Sequential()

model_4.add(Embedding(input_dim=15232, output_dim=11, input_length=6))

model_4.add(Flatten())  # https://keras.io/api/layers/reshaping_layers/flatten/

model_4.add(Dense(100, activation='relu'))

model_4.add(Dense(50, activation='relu'))

model_4.add(Dense(2, activation='softmax'))

model_4.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

print(model_4.summary())


model_4_history = model_4.fit(X_train_median, y_train_median, epochs=20, batch_size=32,
callbacks=[early_stopping_monitor], verbose=True, validation_data=(X_test_median,
y_test_median))

print(model_4_history.history)

# show the plot

plot_learningCurve(model_4_history)


model_4_score = model_4.evaluate(X_train_median, y_train_median, verbose=0)

print(f'Training Set: Test Loss: {model_4_score[0]} / Test Accuracy:
{model_4_score[1]}')


# evaluate model against the test dataset

model_4_evaluation = model_4.evaluate(X_test_median, y_test_median)

print(f'Test Set: Test Loss: {model_4_evaluation[0]} / Test Accuracy:
{model_4_evaluation[1]}')


# add metrics to metrics dataframe

df_model_4_metrics = pd.DataFrame({

    'model_name': ['model_4'],

    'model_description': ['3 layers median length'],

    'test_loss': [model_4_evaluation[0]],

    'test_accuracy': [model_4_evaluation[1]],
```

```
    'epoch_stopped_at': ['3']
})

df_model_metrics = pd.concat([df_model_metrics, df_model_4_metrics], ignore_index=True)
print(df_model_metrics)



# print(model_4.layers[3].get_weights()[0])  # print weights for 3rd layer
# print(model_4.layers[3].get_weights()[1])  # print biases for 3rd layer

# save model: https://datascience.stackexchange.com/questions/27343/output-trained-parameters-of-keras-model

# %%%[9.7] FIRST SIX WORDS

df_reviews['first_six_words'] = df_reviews['review_no_xbf_xef'].str.split().apply(lambda x: ' '.join(x[:6]))
generate_wordcloud(df_reviews['first_six_words'], 'Cleaned Data First Six Words')



# %%[10] EVALUATE THE MODEL ON SOCIAL MEDIA DATA

input_fd_sm = open(r'C:\Users\K2Admin\OneDrive\Documents\WGUMSDA\D214\PA\McDonalds_reviews_social_media.csv', encoding=file_encoding, errors='backslashreplace')
df_sm = pd.read_csv(input_fd_sm)

df_sm.info()

print(eda_analysis(df_sm, 'source'))
print(eda_analysis(df_sm, 'date'))
generate_wordcloud(df_sm['review'], "All Reviews")

clean_review(df_sm, 'review', list_special_characters)
```

```
# Longest review has 67 words.

# Shortest review has 2 words.

# Average review has 11 words.

generate_wordcloud(df_sm['review_no_xbf_xef'], "All Reviews")

print(df_sm.info())


df_sm_padded_max = pad_sequences(df_sm['review_tokenized'], padding='post', maxlen=273)
# need to pad to 273 as that is what the model was trained on

df_sm['review_padded_max_max'] = pd.DataFrame({'review_padded_max_max':
df_sm_padded_max.tolist()})  # add the new padd back to the same df to keep everything
together


# execute model 3 on df_sm and predict sentiment

df_sm_predictions = model_3.predict(df_sm_padded_max)

df_sm_predictions_analysis = pd.DataFrame(df_sm_predictions)  # convert to a dataframe



def max_column(row):

    """ function to look at value in two columns, and return which column has the
greater value

    """

    if row[0] > row[1]:

        return 0

    else:

        return 1



# add column to the df_sm with the predicted rating (sentiment)

df_sm['model_3_prediction'] = df_sm_predictions_analysis.apply(max_column, axis=1)



num_index = 9

print('Origional review', df_sm['review'][num_index], '\n')

print('Predicted:', 'Negative' if df_sm_predictions[num_index][0] >= 0.5 else
'Positive', 'review')
```

```
# %%[11] CUSTOMER SATISFACTION SCORE

# This metric is calculated by taking a count of all of reviews where the company was rated a 4 or 5 (satisfied or very satisfied) divided by the total number of reviews, multiplied by 100 to turn it into a percent. The benchmark for fast food restaurants for CSAT is 76% (SurveyMonkey).


csat_satisfied_df_reviews = sum(df_reviews['rating'] == 1)

csat_satisfied_df_sm = sum(df_sm['model_3_prediction'] == 1)


csat_all_df_reviews = len(df_reviews['rating'])

csat_all_df_sm = len(df_sm['model_3_prediction'])


csat = (csat_satisfied_df_reviews + csat_satisfied_df_sm) / (csat_all_df_reviews + csat_all_df_sm) * 100


print(f'CSAT score: {round(csat,2)}%')


# %%[17] END OF SCRIPT

print(' ')

print("End of script!")
```

End of document.