A Sentiment Analysis of Yelp Reviews

Author: Robert Surridge

1. Import necessary packages:

In [ ]:
```
import pandas as pd
import json
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
import string
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accurac
%matplotlib inline
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/rsurridge/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

2a. Load the dataset with a sample of 10,000 Yelp reviews, explore the metadata, and add a column "length" that contains the number of words per review:

In [ ]:
```
yelp_data = pd.read_json('/Users/rsurridge/Downloads/yelp_data/yelp_academic
                         lines=True, chunksize=10_000)
for chunk in yelp_data:
    yelp_sample_ten_thou = chunk
    result = chunk.to_json(orient="records")
    with open("yelp_sample.json", "w") as f:
        json.dump(result, f)
    break

yelp_sample_ten_thou['length'] = yelp_sample_ten_thou['text'].apply(len)
yelp_sample_ten_thou['stars'] = yelp_sample_ten_thou['stars'].astype(float)

print()
print("Shape of the dataset:", yelp_sample_ten_thou.shape)

print()
print("Column names:", yelp_sample_ten_thou.columns)

print()
print("Datatype of each column:")
print(yelp_sample_ten_thou.dtypes)
```

```
print()
print("A few dataset entries:")
print(yelp_sample_ten_thou.head())

print()
print("Dataset Summary:")
yelp_sample_ten_thou.describe(include='all')
```

```
Shape of the dataset: (10000, 10)

Column names: Index(['review_id', 'user_id', 'business_id', 'stars', 'usefu
l', 'funny',
       'cool', 'text', 'date', 'length'],
      dtype='object')

Datatype of each column:
review_id              object
user_id                object
business_id            object
stars                 float64
useful                  int64
funny                   int64
cool                    int64
text                   object
date           datetime64[ns]
length                  int64
dtype: object

A few dataset entries:
                review_id                 user_id             business_id
0  KU_O5udG6zpxOg-VcAEodg  mh_-eMZ6K5RLWhZyISBhwA  XQfwVwDr-v0ZS3_CbbE5Xw
\
1  BiTunyQ73aT9WBnpR9DZGw  OyoGAe7OKpv6SyGZT5g77Q  7ATYjTIgM3jUlt4UM3IypQ
2  saUsX_uimxRlCVr67Z4Jig  8g_iMtfSiwikVnbP2etR0A  YjUWPpI6HXG530lwP-fb2A
3  AqPFMleE6RsU23_auESxiA  _7bHUi9Uuf5__HHc_Q8guQ  kxX2SOes4o-D3ZQBkiMRfA
4  Sx8TMOWLNuJBWer-0pcmoA  bcjbaE6dDog4jkNY91ncLQ  e4Vwtrqf-wpJfwesgvdgxQ

   stars  useful  funny  cool
0    3.0       0      0     0  \
1    5.0       1      0     1
2    3.0       0      0     0
3    5.0       1      0     1
4    4.0       1      0     1

                                                text                date
0  If you decide to eat here, just be aware it is... 2018-07-07 22:09:11  \
1  I've taken a lot of spin classes over the year... 2012-01-03 15:28:18
2  Family diner. Had the buffet. Eclectic assortm... 2014-02-05 20:30:30
3  Wow!  Yummy, different,  delicious.   Our favo... 2015-01-04 00:01:03
4  Cute interior and owner (?) gave us tour of up... 2017-01-14 20:54:15

   length
0     513
1     829
2     339
3     243
4     534

Dataset Summary:
```

Out[ ]:

	review_id	user_id	business_id	stars	
count	10000	10000	10000	10000.000000	10(
unique	10000	9472	3930	NaN	
top	KU_O5udG6zpxOg-VcAEodg	n-lBS02-3yvlY5Q91mmwDA	GBTPC53ZrG1ZBY3DT8Mbcw	NaN	
freq	1	6	85	NaN	
mean	NaN	NaN	NaN	3.854300	
min	NaN	NaN	NaN	1.000000	
25%	NaN	NaN	NaN	3.000000	
50%	NaN	NaN	NaN	4.000000	
75%	NaN	NaN	NaN	5.000000	
max	NaN	NaN	NaN	5.000000	
std	NaN	NaN	NaN	1.346719	

2b. Load the dataset with a sample of 100,000 Yelp reviews, explore the metadata, and add a column "length" that contains the number of words per review:

In [ ]:
```
yelp_data = pd.read_json('/Users/rsurridge/Downloads/yelp_data/yelp_academic
                         lines=True, chunksize=100_000)
for chunk in yelp_data:
    yelp_sample_hund_thou = chunk
    result = chunk.to_json(orient="records")
    with open("yelp_sample.json", "w") as f:
        json.dump(result, f)
    break

yelp_sample_hund_thou['length'] = yelp_sample_hund_thou['text'].apply(len)
yelp_sample_hund_thou['stars'] = yelp_sample_hund_thou['stars'].astype(float

print()
print("Shape of the dataset:", yelp_sample_hund_thou.shape)

print()
print("Dataset Summary:")
yelp_sample_hund_thou.describe(include='all')
```

```
Shape of the dataset: (100000, 10)

Dataset Summary:
```

Out[ ]:

	review_id	user_id	business_id	
count	100000	100000	100000	100000.0(
unique	100000	79345	9973	
top	KU_O5udG6zpxOg-VcAEodg	_BcWyKQL16ndpBdggh2kNA	GBTPC53ZrG1ZBY3DT8Mbcw	
freq	1	65	950	
mean	NaN	NaN	NaN	3.8
min	NaN	NaN	NaN	1.0(
25%	NaN	NaN	NaN	3.0(
50%	NaN	NaN	NaN	4.0(
75%	NaN	NaN	NaN	5.0(
max	NaN	NaN	NaN	5.0(
std	NaN	NaN	NaN	1.3

2c. Load the dataset with a sample of 1,000,000 Yelp reviews, explore the metadata, and add a column "length" that contains the number of words per review:

In [ ]:

```
yelp_data = pd.read_json('/Users/rsurridge/Downloads/yelp_data/yelp_academic
                         lines=True, chunksize=1_000_000)
for chunk in yelp_data:
    yelp_sample_mil = chunk
    result = chunk.to_json(orient="records")
    with open("yelp_sample.json", "w") as f:
        json.dump(result, f)
    break

yelp_sample_mil['length'] = yelp_sample_mil['text'].apply(len)
yelp_sample_mil['stars'] = yelp_sample_mil['stars'].astype(float)

print()
print("Shape of the dataset:", yelp_sample_mil.shape)

print()
print("Dataset Summary:")
yelp_sample_mil.describe(include='all')
```

```
Shape of the dataset: (1000000, 10)

Dataset Summary:
```

Out[ ]:

	review_id	user_id	business_id	
count	1000000	1000000	1000000	1000000.(
unique	1000000	542003	27095	
top	KU_O5udG6zpxOg-VcAEodg	_BcWyKQL16ndpBdggh2kNA	GBTPC53ZrG1ZBY3DT8Mbcw	
freq	1	483	4661	
mean	NaN	NaN	NaN	3.7
min	NaN	NaN	NaN	1.(
25%	NaN	NaN	NaN	3.(
50%	NaN	NaN	NaN	4.(
75%	NaN	NaN	NaN	5.(
max	NaN	NaN	NaN	5.(
std	NaN	NaN	NaN	1.

In [ ]:

```
star_counts = yelp_sample_ten_thou['stars'].value_counts()
min_count = star_counts.min()
yelp_sample_equal_ten_thou = yelp_sample_ten_thou.groupby('stars').apply(lam
count_ten_thou = yelp_sample_equal_ten_thou['stars'].value_counts()

print()
print("Star count (10,000 reviews)")
print(count_ten_thou)

star_counts = yelp_sample_hund_thou['stars'].value_counts()
min_count = star_counts.min()
yelp_sample_equal_hund_thou = yelp_sample_hund_thou.groupby('stars').apply(l
count_hund_thou = yelp_sample_equal_hund_thou['stars'].value_counts()

print()
print("Star count (100,000 reviews)")
print(count_hund_thou)

star_counts = yelp_sample_mil['stars'].value_counts()
min_count = star_counts.min()
yelp_sample_equal_mil = yelp_sample_mil.groupby('stars').apply(lambda x: x[:
count_mil = yelp_sample_equal_mil['stars'].value_counts()
```

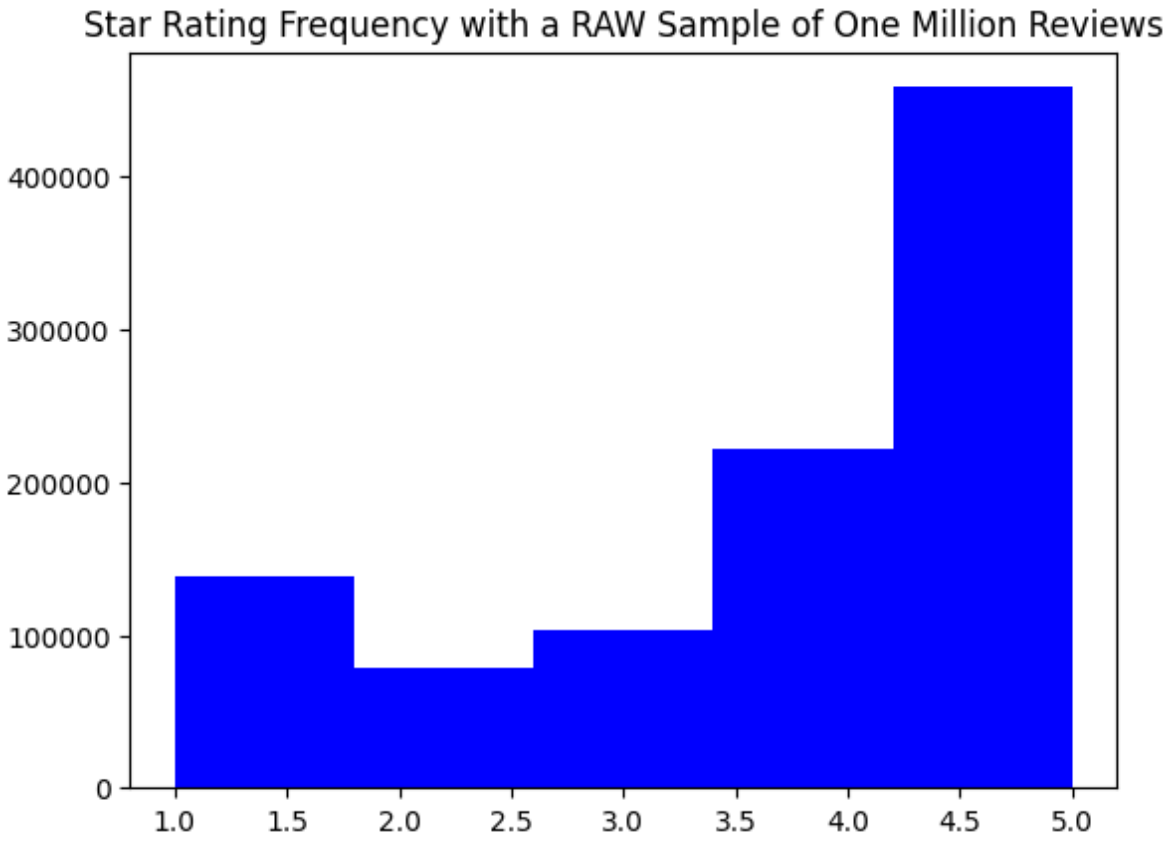
```
print()
print("Star count (1,000,000 reviews)")
print(count_mil)
```

```
Star count (10,000 reviews)
stars
1.0    763
2.0    763
3.0    763
4.0    763
5.0    763
Name: count, dtype: int64

Star count (100,000 reviews)
stars
1.0    7988
2.0    7988
3.0    7988
4.0    7988
5.0    7988
Name: count, dtype: int64

Star count (1,000,000 reviews)
stars
1.0    77912
2.0    77912
3.0    77912
4.0    77912
5.0    77912
Name: count, dtype: int64
```

3. Plot histograms of review length frenquencies by the star rating:

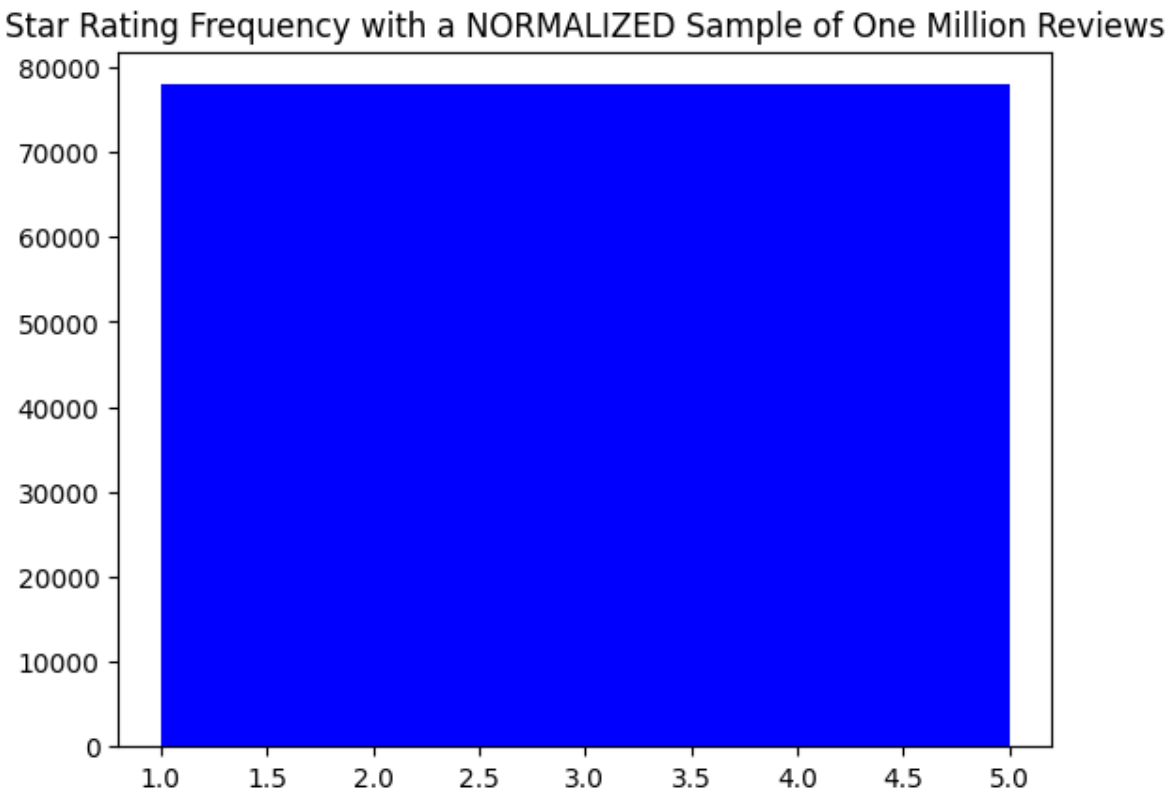
In [ ]:
```
graph_star_freq_raw = plt.hist(yelp_sample_mil['stars'], bins=5, color='blue
plt.title('Star Rating Frequency with a RAW Sample of One Million Reviews')
```

Out[ ]:
```
Text(0.5, 1.0, 'Star Rating Frequency with a RAW Sample of One Million Revi
ews')
```



In [ ]:
```
graph_star_freq_normal = plt.hist(yelp_sample_equal_mil['stars'], bins=5, cc
plt.title('Star Rating Frequency with a NORMALIZED Sample of One Million Rev
```

Out[ ]:
```
Text(0.5, 1.0, 'Star Rating Frequency with a NORMALIZED Sample of One Milli
on Reviews')
```



We want to note here that our sample dataset is biased to more positive reviews than negative reviews. As a result, we should proceed with caution when classifying, training, and predicting with this dataset.

4.a.i. Classify 10,000 standardized reviews into 1-star (negative), 2-star (negative), 3-star (neutral), 4-star (positive), and 5-star (positive) classes and separate the dataset into X and Y subsets for prediction:

In [ ]:
```
yelp_classify = yelp_sample_equal_ten_thou.loc[:, ['stars', 'text']]

print()
print("Shape of the dataset:", yelp_classify.shape)

x_ten_thou_five = yelp_classify['text']
y_ten_thou_five = yelp_classify['stars']

print()
print("A few X entries")
print(x_ten_thou_five)

print()
print("A few Y entries")
print(y_ten_thou_five)
```

```
Shape of the dataset: (3815, 2)

A few X entries
stars
1.0    5       I am a long term frequent customer of this est...
       47      If you want to pay for everything a la carte t...
       64      The TV shows are $4.99 and they have commercia...
       73      If I could give it a zero, I would. I order a ...
       79      We visited once and were very disappointed in ...
                                     ...
5.0    1661    This little unassuming shop is tucked away in ...
       1662    Great food and good coffee. Wish they had more...
       1663    holly grove is the best. I'm consistently impr...
       1669    Henry was a rock star  getting us into a stora...
       1670    Phenomenal restaurant. Great location. Great a...
Name: text, Length: 3815, dtype: object

A few Y entries
stars
1.0    5       1.0
       47      1.0
       64      1.0
       73      1.0
       79      1.0
              ...
5.0    1661    5.0
       1662    5.0
       1663    5.0
       1669    5.0
       1670    5.0
Name: stars, Length: 3815, dtype: float64
```

4.a.ii. Classify 10,000 standardized reviews into 1-star (negative) and 5-star (positive) classes and separate the dataset into X and Y subsets for prediction:

In [ ]:
```
yelp_classify = yelp_sample_equal_ten_thou.loc[:, ['stars', 'text']]
data_classes = yelp_classify[(yelp_classify['stars']==1) |
                             (yelp_classify['stars']==5)]

print()
print("Shape of the dataset:", data_classes.shape)

x_ten_thou_two = data_classes['text']
y_ten_thou_two = data_classes['stars']
```

```
Shape of the dataset: (1526, 2)
```

4.a.iii. Classify 10,000 standardized reviews into 1-star (negative), 3-star (neutral), and 5-star (positive) classes and separate the dataset into X and Y subsets for prediction:

In [ ]:
```
yelp_classify = yelp_sample_equal_ten_thou.loc[:, ['stars', 'text']]
data_classes = yelp_classify[(yelp_classify['stars']==1) |
```

```
                            (yelp_classify['stars']==3) |
                            (yelp_classify['stars']==5)]

print()
print("Shape of the dataset:", data_classes.shape)

x_ten_thou_three = data_classes['text']
y_ten_thou_three = data_classes['stars']
```

```
Shape of the dataset: (2289, 2)
```

4.b.i. Classify 100,000 standardized reviews into 1-star (negative), 2-star (negative), 3-star (neutral), 4-star (positive), and 5-star (positive) classes and separate the dataset into X and Y subsets for prediction:

In [ ]:
```
yelp_classify = yelp_sample_equal_hund_thou.loc[:, ['stars', 'text']]

print()
print("Shape of the dataset:", yelp_classify.shape)

x_hund_thou_five = yelp_classify['text']
y_hund_thou_five = yelp_classify['stars']
```

```
Shape of the dataset: (39940, 2)
```

4.b.ii. Classify 100,000 standardized reviews into 1-star (negative) and 5-star (positive) classes and separate the dataset into X and Y subsets for prediction:

In [ ]:
```
yelp_classify = yelp_sample_equal_hund_thou.loc[:, ['stars', 'text']]
data_classes = yelp_classify[(yelp_classify['stars']==1) |
                            (yelp_classify['stars']==5)]

print()
print("Shape of the dataset:", data_classes.shape)

x_hund_thou_two = data_classes['text']
y_hund_thou_two = data_classes['stars']
```

```
Shape of the dataset: (15976, 2)
```

4.b.iii. Classify 100,000 standardized reviews into 1-star (negative), 3-star (neutral), and 5-star (positive) classes and separate the dataset into X and Y subsets for prediction:

In [ ]:
```
yelp_classify = yelp_sample_equal_hund_thou.loc[:, ['stars', 'text']]
data_classes = yelp_classify[(yelp_classify['stars']==1) |
                            (yelp_classify['stars']==3) |
                            (yelp_classify['stars']==5)]

print()
print("Shape of the dataset:", data_classes.shape)
```

```
x_hund_thou_three = data_classes['text']
y_hund_thou_three = data_classes['stars']
```

```
Shape of the dataset: (23964, 2)
```

4.c.i. Classify 1,000,000 standardized reviews into 1-star (negative), 2-star (negative), 3-star (neutral), 4-star (positive), and 5-star (positive) classes and separate the dataset into X and Y subsets for prediction:

In [ ]:
```
yelp_classify = yelp_sample_equal_mil.loc[:, ['stars', 'text']]

print()
print("Shape of the dataset:", yelp_classify.shape)

x_mil_five = data_classes['text']
y_mil_five = data_classes['stars']
```

```
Shape of the dataset: (389560, 2)
```

4.c.ii. Classify 1,000,000 standardized reviews into 1-star (negative) and 5-star (positive) classes and separate the dataset into X and Y subsets for prediction:

In [ ]:
```
yelp_classify = yelp_sample_equal_mil.loc[:, ['stars', 'text']]
data_classes = yelp_classify[(yelp_classify['stars']==1) |
                             (yelp_classify['stars']==5)]

print()
print("Shape of the dataset:", data_classes.shape)

x_mil_two = data_classes['text']
y_mil_two = data_classes['stars']
```

```
Shape of the dataset: (155824, 2)
Shape of the dataset: (155824, 2)
```

4.c.iii. Classify 1,000,000 standardized reviews into 1-star (negative), 3-star (neutral), and 5-star (positive) classes and separate the dataset into X and Y subsets for prediction:

In [ ]:
```
yelp_classify = yelp_sample_equal_mil.loc[:, ['stars', 'text']]
data_classes = yelp_classify[(yelp_classify['stars']==1) |
                             (yelp_classify['stars']==3) |
                             (yelp_classify['stars']==5)]

print()
print("Shape of the dataset:", data_classes.shape)

x_mil_three = data_classes['text']
y_mil_three = data_classes['stars']
```

```
Shape of the dataset: (233736, 2)
```

5. Clean the review text by removing stopwords and punctuation:

In [ ]:
```
def process_text(text):
    nopunc = [char for char in text if char not in string.punctuation]
    nopunc = ''.join(nopunc)
    return [word for word in nopunc.split() if word.lower() not in stopwords
```

6.a.i. Convert 10,000 (1-star, 2-star, 3-star, 4-star, and 5-star) reviews into vectors:

In [ ]:
```
vocab_ten_thou_five = CountVectorizer(analyzer=process_text).fit(x_ten_thou_
r1 = x_ten_thou_five[1][5]
vocab1 = vocab_ten_thou_five.transform([r1])

print()
print("Number of possible words:", len(vocab_ten_thou_five.vocabulary_))

print()
print("Sample Uncleaned Review:")
print(r1)

print()
print("Vectorized Review:")
print(vocab1)

print()
print("Return word from index 5000:", vocab_ten_thou_five.get_feature_names_
print("Return word from index 20000:", vocab_ten_thou_five.get_feature_names
```

```
Number of possible words: 21208

Sample Uncleaned Review:
I am a long term frequent customer of this establishment. I just went in to
order take out (3 apps) and was told they're too busy to do it. Really? The
place is maybe half full at best. Does your dick reach your ass? Yes? Go fu
ck yourself! I'm a frequent customer AND great tipper. Glad that Kanella ju
st opened. NEVER going back to dmitris!

Vectorized Review:
  (0, 322)      1
  (0, 2833)     1
  (0, 2846)     1
  (0, 3245)     1
  (0, 3472)     1
  (0, 4223)     1
  (0, 5051)     1
  (0, 6645)     1
  (0, 7244)     1
  (0, 7332)     1
  (0, 7510)     1
  (0, 7814)     1
  (0, 8377)     1
  (0, 9848)     2
  (0, 10232)    1
  (0, 10459)    1
  (0, 11027)    1
  (0, 11818)    2
  (0, 11890)    1
  (0, 11900)    1
  (0, 12126)    1
  (0, 12236)    1
  (0, 12414)    1
  (0, 13983)    1
  (0, 14304)    1
  (0, 15318)    1
  (0, 15361)    1
  (0, 16024)    1
  (0, 16800)    1
  (0, 19342)    1
  (0, 19512)    1
  (0, 19578)    1
  (0, 19725)    1
  (0, 19770)    1
  (0, 20819)    1

Return word from index 5000: Rabbit
Return word from index 20000: declared
```

6.a.ii. Convert 10,000 (1-star and 5-star) reviews into vectors:

In [ ]:
```
vocab_ten_thou_two = CountVectorizer(analyzer=process_text).fit(x_ten_thou_t
r1 = x_ten_thou_two[1][5]
vocab1 = vocab_ten_thou_two.transform([r1])
```

```
print()
print("Number of possible words:", len(vocab_ten_thou_two.vocabulary_))

print()
print("Return word from index 5000:", vocab_ten_thou_two.get_feature_names_c
print("Return word from index 20000:", vocab_ten_thou_two.get_feature_names_
```

```
Number of possible words: 12741

Return word from index 5000: chefas
Return word from index 20000: recommend
```

6.a.iii. Convert 10,000 (1-star, 3-star, and 5-star) reviews into vectors:

In [ ]:
```
vocab_ten_thou_three = CountVectorizer(analyzer=process_text).fit(x_ten_thou

print()
print("Number of possible words:", len(vocab_ten_thou_three.vocabulary_))

print()
print("Return word from index 5000:", vocab_ten_thou_three.get_feature_names
print("Return word from index 20000:", vocab_ten_thou_three.get_feature_name
```

```
Number of possible words: 15946

Return word from index 5000: alert
Return word from index 20000: jaded
```

6.b.i. Convert 100,000 (1-star, 2-star, 3-star, 4-star, and 5-star) reviews into vectors:

In [ ]:
```
vocab_hund_thou_five = CountVectorizer(analyzer=process_text).fit(x_hund_tho

print()
print("Number of possible words:", len(vocab_hund_thou_five.vocabulary_))

print()
print("Return word from index 5000:", vocab_hund_thou_five.get_feature_names
print("Return word from index 20000:", vocab_hund_thou_five.get_feature_name
```

```
Number of possible words: 76896

Return word from index 5000: Bellied
Return word from index 20000: Philly
```

6.b.ii. Convert 100,000 (1-star and 5-star) reviews into vectors:

In [ ]:
```
vocab_hund_thou_two = CountVectorizer(analyzer=process_text).fit(x_hund_thou

print()
print("Number of possible words:", len(vocab_hund_thou_two.vocabulary_))

print()
```

```
print("Return word from index 5000:", vocab_hund_thou_two.get_feature_names_
print("Return word from index 20000:", vocab_hund_thou_two.get_feature_names
```

```
Number of possible words: 45501

Return word from index 5000: Curious
Return word from index 20000: buggy
```

6.b.iii. Convert 100,000 (1-star, 3-star, and 5-star) reviews into vectors:

In [ ]:
```
vocab_hund_thou_three = CountVectorizer(analyzer=process_text).fit(x_hund_th

print()
print("Number of possible words:", len(vocab_hund_thou_three.vocabulary_))

print()
print("Return word from index 5000:", vocab_hund_thou_three.get_feature_name
print("Return word from index 20000:", vocab_hund_thou_three.get_feature_nam
```

```
Number of possible words: 57720

Return word from index 5000: Cappuccino
Return word from index 20000: Venison
```

6.c.i. Convert 1,000,000 (1-star, 2-star, 3-star, 4-star, and 5-star) reviews into vectors:

In [ ]:
```
vocab_mil_five = CountVectorizer(analyzer=process_text).fit(x_mil_five)

print()
print("Number of possible words:", len(vocab_mil_five.vocabulary_))

print()
print("Return word from index 5000:", vocab_mil_five.get_feature_names_out()
print("Return word from index 20000:", vocab_mil_five.get_feature_names_out(
```

```
Number of possible words: 57720

Return word from index 5000: Cappuccino
Return word from index 20000: Venison
```

6.c.ii. Convert 1,000,000 (1-star and 5-star) reviews into vectors:

In [ ]:
```
vocab_mil_two = CountVectorizer(analyzer=process_text).fit(x_mil_two)

print()
print("Number of possible words:", len(vocab_mil_two.vocabulary_))

print()
print("Return word from index 5000:", vocab_mil_two.get_feature_names_out()[
print("Return word from index 20000:", vocab_mil_two.get_feature_names_out()
```

```
Number of possible words: 167844

Return word from index 5000: 40400
Return word from index 20000: Competitive
```

6.c.iii. Convert 1,000,000 (1-star, 3-star, and 5-star) reviews into vectors:

In [ ]:
```
vocab_mil_three = CountVectorizer(analyzer=process_text).fit(x_mil_three)

print()
print("Number of possible words:", len(vocab_mil_three.vocabulary_))

print()
print("Return word from index 5000:", vocab_mil_three.get_feature_names_out(
print("Return word from index 20000:", vocab_mil_three.get_feature_names_out
```

```
Number of possible words: 218502

Return word from index 5000: 2pcs
Return word from index 20000: Bus
```

7.a.i. Vectorize all 10,000 (1-star, 2-star, 3-star, 4-star, and 5-star) reviews and split processed dataset into training and testing sets:

In [ ]:
```
x_ten_thou_five = vocab_ten_thou_five.transform(x_ten_thou_five)
x_train_ten_thou_five, x_test_ten_thou_five, y_train_ten_thou_five, y_test_t
```

7.a.ii. Vectorize all 10,000 (1-star and 5-star) reviews and split processed dataset into training and testing sets:

In [ ]:
```
x_ten_thou_two = vocab_ten_thou_two.transform(x_ten_thou_two)
x_train_ten_thou_two, x_test_ten_thou_two, y_train_ten_thou_two, y_test_ten_
```

7.a.iii. Vectorize all 10,000 (1-star, 3-star, and 5-star) reviews and split processed dataset into training and testing sets:

In [ ]:
```
x_ten_thou_three = vocab_ten_thou_three.transform(x_ten_thou_three)
x_train_ten_thou_three, x_test_ten_thou_three, y_train_ten_thou_three, y_tes
```

7.b.i. Vectorize all 100,000 (1-star, 2-star, 3-star, 4-star, and 5-star) reviews and split processed dataset into training and testing sets:

In [ ]:
```
x_hund_thou_five = vocab_hund_thou_five.transform(x_hund_thou_five)
x_train_hund_thou_five, x_test_hund_thou_five, y_train_hund_thou_five, y_tes
```

7.b.ii. Vectorize all 100,000 (1-star and 5-star) reviews and split processed dataset into training and testing sets:

In [ ]:
```
x_hund_thou_two = vocab_hund_thou_two.transform(x_hund_thou_two)
x_train_hund_thou_two, x_test_hund_thou_two, y_train_hund_thou_two, y_test_h
```

7.b.iii. Vectorize all 100,000 (1-star, 3-star, and 5-star) reviews and split processed dataset into training and testing sets:

In [ ]:
```
x_hund_thou_three = vocab_hund_thou_three.transform(x_hund_thou_three)
x_train_hund_thou_three, x_test_hund_thou_three, y_train_hund_thou_three, y_
```

7.c.i. Vectorize all 1,000,000 (1-star, 2-star, 3-star, 4-star, and 5-star) reviews and split processed dataset into training and testing sets:

In [ ]:
```
x_mil_five = vocab_mil_five.transform(x_mil_five)
x_train_mil_five, x_test_mil_five, y_train_mil_five, y_test_mil_five = train
```

7.c.ii. Vectorize all 1,000,000 (1-star and 5-star) reviews and split processed dataset into training and testing sets:

In [ ]:
```
x_mil_two = vocab_mil_two.transform(x_mil_two)
x_train_mil_two, x_test_mil_two, y_train_mil_two, y_test_mil_two = train_tes
```

7.c.iii. Vectorize all 1,000,000 (1-star, 3-star, and 5-star) reviews and split processed dataset into training and testing sets:

In [ ]:
```
x_mil_three = vocab_mil_three.transform(x_mil_three)
x_train_mil_three, x_test_mil_three, y_train_mil_three, y_test_mil_three = t
```

8.a.i. Modeling 10,000 (1-star, 2-star, 3-star, 4-star, and 5-star) reviews with Multinomial Naive Bayes (since we're working with sparse data, we cannot rely on the Gaussian Naive Bayes assumptions):

In [ ]:
```
# Multinomial Naive Bayes
from sklearn.naive_bayes import MultinomialNB
mnb_ten_5 = MultinomialNB()
mnb_ten_5.fit(x_train_ten_thou_five, y_train_ten_thou_five)
predmnb = mnb_ten_5.predict(x_test_ten_thou_five)

print()
print("Score:", round(accuracy_score(y_test_ten_thou_five, predmnb) * 100, 2
```

```
print()
print("Classification Report:")
print(classification_report(y_test_ten_thou_five, predmnb))
```

```
Score: 47.05

Classification Report:
              precision    recall  f1-score   support

         1.0       0.59      0.68      0.63       145
         2.0       0.43      0.39      0.41       163
         3.0       0.37      0.48      0.42       149
         4.0       0.41      0.47      0.44       156
         5.0       0.70      0.34      0.46       150

    accuracy                           0.47       763
   macro avg       0.50      0.47      0.47       763
weighted avg       0.50      0.47      0.47       763
```

8.a.ii. Modeling 10,000 (1-star and 5-star) reviews with Multinomial Naive Bayes:

In [ ]:
```
# Multinomial Naive Bayes
from sklearn.naive_bayes import MultinomialNB
mnb_ten_2 = MultinomialNB()
mnb_ten_2.fit(x_train_ten_thou_two, y_train_ten_thou_two)
predmnb = mnb_ten_2.predict(x_test_ten_thou_two)

print()
print("Score:", round(accuracy_score(y_test_ten_thou_two, predmnb) * 100, 2))

print()
print("Classification Report:")
print(classification_report(y_test_ten_thou_two, predmnb))
```

```
Score: 90.52

Classification Report:
              precision    recall  f1-score   support

         1.0       0.92      0.90      0.91       156
         5.0       0.90      0.91      0.90       150

    accuracy                           0.91       306
   macro avg       0.91      0.91      0.91       306
weighted avg       0.91      0.91      0.91       306
```

8.a.iii. Modeling 10,000 (1-star, 3-star, and 5-star) reviews with Multinomial Naive Bayes:

In [ ]:
```
# Multinomial Naive Bayes
from sklearn.naive_bayes import MultinomialNB
mnb_ten_3 = MultinomialNB()
```

```
mnb_ten_3.fit(x_train_ten_thou_three, y_train_ten_thou_three)
predmnb = mnb_ten_3.predict(x_test_ten_thou_three)

print()
print("Score:", round(accuracy_score(y_test_ten_thou_three, predmnb) * 100,

print()
print("Classification Report:")
print(classification_report(y_test_ten_thou_three, predmnb))
```

```
Score: 74.67

Classification Report:
              precision    recall  f1-score   support

         1.0       0.78      0.81      0.79       153
         3.0       0.67      0.75      0.70       166
         5.0       0.83      0.68      0.75       139

    accuracy                           0.75       458
   macro avg       0.76      0.74      0.75       458
weighted avg       0.75      0.75      0.75       458
```

8.b.i. Modeling 100,000 (1-star, 2-star, 3-star, 4-star, and 5-star) reviews with Multinomial Naive Bayes:

In [ ]:
```
# Multinomial Naive Bayes
from sklearn.naive_bayes import MultinomialNB
mnb_hund_5 = MultinomialNB()
mnb_hund_5.fit(x_train_hund_thou_five, y_train_hund_thou_five)
predmnb = mnb_hund_5.predict(x_test_hund_thou_five)

print()
print("Score:", round(accuracy_score(y_test_hund_thou_five, predmnb) * 100,

print()
print("Classification Report:")
print(classification_report(y_test_hund_thou_five, predmnb))
```

```
Score: 52.77

Classification Report:
              precision    recall  f1-score   support

         1.0       0.63      0.67      0.65      1626
         2.0       0.45      0.46      0.46      1584
         3.0       0.44      0.47      0.46      1650
         4.0       0.47      0.50      0.48      1596
         5.0       0.68      0.54      0.60      1532

    accuracy                           0.53      7988
   macro avg       0.54      0.53      0.53      7988
weighted avg       0.54      0.53      0.53      7988
```

8.b.ii. Modeling 100,000 (1-star and 5-star) reviews with Multinomial Naive Bayes:

In [ ]:
```
# Multinomial Naive Bayes
from sklearn.naive_bayes import MultinomialNB
mnb_hund_2 = MultinomialNB()
mnb_hund_2.fit(x_train_hund_thou_two, y_train_hund_thou_two)
predmnb = mnb_hund_2.predict(x_test_hund_thou_two)

print()
print("Score:", round(accuracy_score(y_test_hund_thou_two, predmnb) * 100, 2

print()
print("Classification Report:")
print(classification_report(y_test_hund_thou_two, predmnb))
```

```
Score: 94.9

Classification Report:
              precision    recall  f1-score   support

         1.0       0.94      0.96      0.95      1604
         5.0       0.95      0.94      0.95      1592

    accuracy                           0.95      3196
   macro avg       0.95      0.95      0.95      3196
weighted avg       0.95      0.95      0.95      3196
```

8.b.iii. Modeling 100,000 (1-star, 3-star, and 5-star) reviews with Multinomial Naive Bayes:

In [ ]:
```
# Multinomial Naive Bayes
from sklearn.naive_bayes import MultinomialNB
mnb_hund_3 = MultinomialNB()
mnb_hund_3.fit(x_train_hund_thou_three, y_train_hund_thou_three)
predmnb = mnb_hund_3.predict(x_test_hund_thou_three)

print()
print("Score:", round(accuracy_score(y_test_hund_thou_three, predmnb) * 100,

print()
print("Classification Report:")
print(classification_report(y_test_hund_thou_three, predmnb))
```

```
Score: 78.24

Classification Report:
              precision    recall  f1-score   support

         1.0       0.83      0.80      0.81      1640
         3.0       0.68      0.78      0.73      1604
         5.0       0.87      0.77      0.81      1549

    accuracy                           0.78      4793
   macro avg       0.79      0.78      0.78      4793
weighted avg       0.79      0.78      0.78      4793
```

8.c.i. Modeling 1,000,000 (1-star, 2-star, 3-star, 4-star, and 5-star) reviews with Multinomial Naive Bayes:

In [ ]:
```
# Multinomial Naive Bayes
from sklearn.naive_bayes import MultinomialNB
mnb_mil_5 = MultinomialNB()
mnb_mil_5.fit(x_train_mil_five, y_train_mil_five)
predmnb = mnb_mil_5.predict(x_test_mil_five)

print()
print("Score:", round(accuracy_score(y_test_mil_five, predmnb) * 100, 2))

print()
print("Classification Report:")
print(classification_report(y_test_mil_five, predmnb))
```

```
Score: 78.24

Classification Report:
              precision    recall  f1-score   support

         1.0       0.83      0.80      0.81      1640
         3.0       0.68      0.78      0.73      1604
         5.0       0.87      0.77      0.81      1549

    accuracy                           0.78      4793
   macro avg       0.79      0.78      0.78      4793
weighted avg       0.79      0.78      0.78      4793
```

8.c.ii. Modeling 1,000,000 (1-star and 5-star) reviews with Multinomial Naive Bayes:

In [ ]:
```
# Multinomial Naive Bayes
from sklearn.naive_bayes import MultinomialNB
mnb_mil_2 = MultinomialNB()
mnb_mil_2.fit(x_train_mil_two, y_train_mil_two)
predmnb = mnb_mil_2.predict(x_test_mil_two)

print()
```

```
print("Score:", round(accuracy_score(y_test_mil_two, predmnb) * 100, 2))

print()
print("Classification Report:")
print(classification_report(y_test_mil_two, predmnb))
```

```
Score: 93.54

Classification Report:
              precision    recall  f1-score   support

         1.0       0.94      0.93      0.93     15574
         5.0       0.93      0.94      0.94     15591

    accuracy                           0.94     31165
   macro avg       0.94      0.94      0.94     31165
weighted avg       0.94      0.94      0.94     31165
```

8.c.iii. Modeling 1,000,000 (1-star, 3-star, and 5-star) reviews with Multinomial Naive Bayes:

In [ ]:
```
# Multinomial Naive Bayes
from sklearn.naive_bayes import MultinomialNB
mnb_mil_3= MultinomialNB()
mnb_mil_3.fit(x_train_mil_three, y_train_mil_three)
predmnb = mnb_mil_3.predict(x_test_mil_three)

print()
print("Score:", round(accuracy_score(y_test_mil_three, predmnb) * 100, 2))

print()
print("Classification Report:")
print(classification_report(y_test_mil_three, predmnb))
```

```
Score: 79.66

Classification Report:
              precision    recall  f1-score   support

         1.0       0.81      0.78      0.80     15566
         3.0       0.70      0.78      0.74     15502
         5.0       0.89      0.83      0.86     15680

    accuracy                           0.80     46748
   macro avg       0.80      0.80      0.80     46748
weighted avg       0.80      0.80      0.80     46748
```

9.a.i. Modeling 10,000 (1-star, 2-star, 3-star, 4-star, and 5-star) reviews with Random Forest Classifier:

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
rmfr_ten_5 = RandomForestClassifier()
rmfr_ten_5.fit(x_train_ten_thou_five, y_train_ten_thou_five)
```

```
predrmfr = rmfr_ten_5.predict(x_test_ten_thou_five)

print()
print("Score:", round(accuracy_score(y_test_ten_thou_five, predrmfr) * 100,

print()
print("Classification Report:")
print(classification_report(y_test_ten_thou_five, predrmfr))
```

```
Score: 43.64

Classification Report:
              precision    recall  f1-score   support

         1.0       0.49      0.68      0.57       145
         2.0       0.40      0.21      0.27       163
         3.0       0.37      0.41      0.39       149
         4.0       0.39      0.26      0.31       156
         5.0       0.48      0.67      0.56       150

    accuracy                           0.44       763
   macro avg       0.42      0.44      0.42       763
weighted avg       0.42      0.44      0.41       763
```

9.a.ii. Modeling 10,000 (1-star and 5-star) reviews with Random Forest Classifier:

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
rmfr_ten_2 = RandomForestClassifier()
rmfr_ten_2.fit(x_train_ten_thou_two, y_train_ten_thou_two)
predrmfr = rmfr_ten_2.predict(x_test_ten_thou_two)

print()
print("Score:", round(accuracy_score(y_test_ten_thou_two, predrmfr) * 100, 2

print()
print("Classification Report:")
print(classification_report(y_test_ten_thou_two, predrmfr))
```

```
Score: 91.18

Classification Report:
              precision    recall  f1-score   support

         1.0       0.92      0.91      0.91       156
         5.0       0.91      0.91      0.91       150

    accuracy                           0.91       306
   macro avg       0.91      0.91      0.91       306
weighted avg       0.91      0.91      0.91       306
```

9.a.iii. Modeling 10,000 (1-star, 3-star, and 5-star) reviews with Random Forest Classifier:

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
rmfr_ten_3 = RandomForestClassifier()
rmfr_ten_3.fit(x_train_ten_thou_three, y_train_ten_thou_three)
predrmfr = rmfr_ten_3.predict(x_test_ten_thou_three)

print()
print("Score:", round(accuracy_score(y_test_ten_thou_three, predrmfr) * 100,

print()
print("Classification Report:")
print(classification_report(y_test_ten_thou_three, predrmfr))
```

```
Score: 70.52

Classification Report:
              precision    recall  f1-score   support

         1.0       0.71      0.85      0.77       153
         3.0       0.74      0.51      0.60       166
         5.0       0.68      0.78      0.72       139

    accuracy                           0.71       458
   macro avg       0.71      0.71      0.70       458
weighted avg       0.71      0.71      0.70       458
```

9.b.i. Modeling 100,000 (1-star, 2-star, 3-star, 4-star, and 5-star) reviews with Random Forest Classifier:

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
rmfr_hund_5 = RandomForestClassifier()
rmfr_hund_5.fit(x_train_hund_thou_five, y_train_hund_thou_five)
predrmfr = rmfr_hund_5.predict(x_test_hund_thou_five)

print()
print("Score:", round(accuracy_score(y_test_hund_thou_five, predrmfr) * 100,

print()
print("Classification Report:")
print(classification_report(y_test_hund_thou_five, predrmfr))
```

```
Score: 49.4

Classification Report:
              precision    recall  f1-score   support

         1.0       0.55      0.78      0.64      1626
         2.0       0.43      0.32      0.37      1584
         3.0       0.44      0.35      0.39      1650
         4.0       0.43      0.31      0.36      1596
         5.0       0.54      0.71      0.61      1532

    accuracy                           0.49      7988
   macro avg       0.48      0.50      0.48      7988
weighted avg       0.48      0.49      0.47      7988
```

9.b.ii. Modeling 100,000 (1-star and 5-star) reviews with Random Forest Classifier:

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
rmfr_hund_2 = RandomForestClassifier()
rmfr_hund_2.fit(x_train_hund_thou_two, y_train_hund_thou_two)
predrmfr = rmfr_hund_2.predict(x_test_hund_thou_two)

print()
print("Score:", round(accuracy_score(y_test_hund_thou_two, predrmfr) * 100,

print()
print("Classification Report:")
print(classification_report(y_test_hund_thou_two, predrmfr))
```

```
Score: 93.59

Classification Report:
              precision    recall  f1-score   support

         1.0       0.92      0.96      0.94      1604
         5.0       0.96      0.91      0.93      1592

    accuracy                           0.94      3196
   macro avg       0.94      0.94      0.94      3196
weighted avg       0.94      0.94      0.94      3196
```

9.b.iii. Modeling 100,000 (1-star, 3-star, and 5-star) reviews with Random Forest Classifier:

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
rmfr_hund_3 = RandomForestClassifier()
rmfr_hund_3.fit(x_train_hund_thou_three, y_train_hund_thou_three)
predrmfr = rmfr_hund_3.predict(x_test_hund_thou_three)

print()
print("Score:", round(accuracy_score(y_test_hund_thou_three, predrmfr) * 100
```

```
print()
print("Classification Report:")
print(classification_report(y_test_hund_thou_three, predrmfr))
```

```
Score: 77.93

Classification Report:
              precision    recall  f1-score   support

         1.0       0.79      0.86      0.82      1640
         3.0       0.75      0.66      0.70      1604
         5.0       0.80      0.82      0.81      1549

    accuracy                           0.78      4793
   macro avg       0.78      0.78      0.78      4793
weighted avg       0.78      0.78      0.78      4793
```

9.c.i. Modeling 1,000,000 (1-star, 2-star, 3-star, 4-star, and 5-star) reviews with Random Forest Classifier:

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
rmfr_mil_5 = RandomForestClassifier()
rmfr_mil_5.fit(x_train_mil_five, y_train_mil_five)
predrmfr = rmfr_mil_5.predict(x_test_mil_five)

print()
print("Score:", round(accuracy_score(y_test_mil_five, predrmfr) * 100, 2))

print()
print("Classification Report:")
print(classification_report(y_test_mil_five, predrmfr))
```

```
Score: 77.84

Classification Report:
              precision    recall  f1-score   support

         1.0       0.79      0.86      0.82      1640
         3.0       0.75      0.66      0.70      1604
         5.0       0.79      0.82      0.81      1549

    accuracy                           0.78      4793
   macro avg       0.78      0.78      0.78      4793
weighted avg       0.78      0.78      0.78      4793
```

9.c.ii. Modeling 1,000,000 (1-star and 5-star) reviews with Random Forest Classifier:

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
rmfr_mil_2 = RandomForestClassifier()
rmfr_mil_2.fit(x_train_mil_two, y_train_mil_two)
predrmfr = rmfr_mil_2.predict(x_test_mil_two)
```

```
print()
print("Score:", round(accuracy_score(y_test_mil_two, predrmfr) * 100, 2))

print()
print("Classification Report:")
print(classification_report(y_test_mil_two, predrmfr))
```

```
Score: 94.74

Classification Report:
              precision    recall  f1-score   support

         1.0       0.93      0.96      0.95     15574
         5.0       0.96      0.93      0.95     15591

    accuracy                           0.95     31165
   macro avg       0.95      0.95      0.95     31165
weighted avg       0.95      0.95      0.95     31165
```

9.c.iii. Modeling 1,000,000 (1-star, 3-star, and 5-star) reviews with Random Forest Classifier:

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
rmfr_mil_3 = RandomForestClassifier()
rmfr_mil_3.fit(x_train_mil_three, y_train_mil_three)
predrmfr = rmfr_mil_3.predict(x_test_mil_three)

print()
print("Score:", round(accuracy_score(y_test_mil_three, predrmfr) * 100, 2))

print()
print("Classification Report:")
print(classification_report(y_test_mil_three, predrmfr))
```

```
Score: 80.02

Classification Report:
              precision    recall  f1-score   support

         1.0       0.80      0.87      0.84     15566
         3.0       0.76      0.70      0.73     15502
         5.0       0.83      0.83      0.83     15680

    accuracy                           0.80     46748
   macro avg       0.80      0.80      0.80     46748
weighted avg       0.80      0.80      0.80     46748
```

10.a.i. Modeling 10,000 (1-star, 2-star, 3-star, 4-star, and 5-star) reviews with the Decision Tree Classifier:

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
dt_ten_5 = DecisionTreeClassifier()
```

```
dt_ten_5.fit(x_train_ten_thou_five, y_train_ten_thou_five)
preddt = dt_ten_5.predict(x_test_ten_thou_five)

print()
print("Score:",round(accuracy_score(y_test_ten_thou_five, preddt) * 100, 2))

print()
print("Classification Report:")
print(classification_report(y_test_ten_thou_five, preddt))
```

```
Score: 29.88

Classification Report:
              precision    recall  f1-score   support

         1.0       0.40      0.40      0.40       145
         2.0       0.31      0.23      0.26       163
         3.0       0.22      0.23      0.23       149
         4.0       0.26      0.27      0.27       156
         5.0       0.31      0.37      0.34       150

    accuracy                           0.30       763
   macro avg       0.30      0.30      0.30       763
weighted avg       0.30      0.30      0.30       763
```

10.a.ii. Modeling 10,000 (1-star and 5-star) reviews with the Decision Tree Classifier:

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
dt_ten_2 = DecisionTreeClassifier()
dt_ten_2.fit(x_train_ten_thou_two, y_train_ten_thou_two)
preddt = dt_ten_2.predict(x_test_ten_thou_two)

print()
print("Score:",round(accuracy_score(y_test_ten_thou_two, preddt) * 100, 2))

print()
print("Classification Report:")
print(classification_report(y_test_ten_thou_two, preddt))
```

```
Score: 78.1

Classification Report:
              precision    recall  f1-score   support

         1.0       0.78      0.79      0.79       156
         5.0       0.78      0.77      0.78       150

    accuracy                           0.78       306
   macro avg       0.78      0.78      0.78       306
weighted avg       0.78      0.78      0.78       306
```

10.a.iii. Modeling 10,000 (1-star, 3-star, and 5-star) reviews with the Decision Tree Classifier:

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
dt_ten_3 = DecisionTreeClassifier()
dt_ten_3.fit(x_train_ten_thou_three, y_train_ten_thou_three)
preddt = dt_ten_3.predict(x_test_ten_thou_three)

print()
print("Score:",round(accuracy_score(y_test_ten_thou_three, preddt) * 100, 2))

print()
print("Classification Report:")
print(classification_report(y_test_ten_thou_three, preddt))
```

```
Score: 55.02

Classification Report:
              precision    recall  f1-score   support

         1.0       0.67      0.63      0.65       153
         3.0       0.50      0.48      0.49       166
         5.0       0.50      0.55      0.52       139

    accuracy                           0.55       458
   macro avg       0.55      0.55      0.55       458
weighted avg       0.55      0.55      0.55       458
```

10.b.i. Modeling 100,000 (1-star, 2-star, 3-star, 4-star, and 5-star) reviews with the Decision Tree Classifier:

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
dt_hund_5 = DecisionTreeClassifier()
dt_hund_5.fit(x_train_hund_thou_five, y_train_hund_thou_five)
preddt = dt_hund_5.predict(x_test_hund_thou_five)

print()
print("Score:",round(accuracy_score(y_test_hund_thou_five, preddt) * 100, 2))

print()
print("Classification Report:")
print(classification_report(y_test_hund_thou_five, preddt))
```

```
Score: 37.71

Classification Report:
              precision    recall  f1-score   support

         1.0       0.50      0.51      0.50      1626
         2.0       0.30      0.29      0.30      1584
         3.0       0.32      0.30      0.31      1650
         4.0       0.31      0.30      0.31      1596
         5.0       0.44      0.49      0.46      1532

    accuracy                           0.38      7988
   macro avg       0.37      0.38      0.38      7988
weighted avg       0.37      0.38      0.37      7988
```

10.b.ii. Modeling 100,000 (1-star and 5-star) reviews with the Decision Tree Classifier:

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
dt_hund_2 = DecisionTreeClassifier()
dt_hund_2.fit(x_train_hund_thou_two, y_train_hund_thou_two)
preddt = dt_hund_2.predict(x_test_hund_thou_two)

print()
print("Score:",round(accuracy_score(y_test_hund_thou_two, preddt) * 100, 2))

print()
print("Classification Report:")
print(classification_report(y_test_hund_thou_two, preddt))
```

```
Score: 85.67

Classification Report:
              precision    recall  f1-score   support

         1.0       0.86      0.86      0.86      1604
         5.0       0.86      0.86      0.86      1592

    accuracy                           0.86      3196
   macro avg       0.86      0.86      0.86      3196
weighted avg       0.86      0.86      0.86      3196
```

10.b.iii. Modeling 100,000 (1-star, 3-star, and 5-star) reviews with the Decision Tree Classifier:

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
dt_hund_3 = DecisionTreeClassifier()
dt_hund_3.fit(x_train_hund_thou_three, y_train_hund_thou_three)
preddt = dt_hund_3.predict(x_test_hund_thou_three)

print()
print("Score:",round(accuracy_score(y_test_hund_thou_three, preddt) * 100, 2
```

```
print()
print("Classification Report:")
print(classification_report(y_test_hund_thou_three, preddt))
```

```
Score: 64.01

Classification Report:
              precision    recall  f1-score   support

         1.0       0.70      0.67      0.68      1640
         3.0       0.58      0.56      0.57      1604
         5.0       0.65      0.69      0.67      1549

    accuracy                           0.64      4793
   macro avg       0.64      0.64      0.64      4793
weighted avg       0.64      0.64      0.64      4793
```

10.c.i. Modeling 1,000,000 (1-star, 2-star, 3-star, 4-star, and 5-star) reviews with the Decision Tree Classifier:

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
dt_mil_5 = DecisionTreeClassifier()
dt_mil_5.fit(x_train_mil_five, y_train_mil_five)
preddt = dt_mil_5.predict(x_test_mil_five)

print()
print("Score:",round(accuracy_score(y_test_mil_five, preddt) * 100, 2))

print()
print("Classification Report:")
print(classification_report(y_test_mil_five, preddt))
```

```
Score: 63.78

Classification Report:
              precision    recall  f1-score   support

         1.0       0.70      0.67      0.68      1640
         3.0       0.57      0.55      0.56      1604
         5.0       0.64      0.69      0.67      1549

    accuracy                           0.64      4793
   macro avg       0.64      0.64      0.64      4793
weighted avg       0.64      0.64      0.64      4793
```

10.c.ii. Modeling 1,000,000 (1-star and 5-star) reviews with the Decision Tree Classifier:

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
dt_mil_2 = DecisionTreeClassifier()
dt_mil_2.fit(x_train_mil_two, y_train_mil_two)
preddt = dt_mil_2.predict(x_test_mil_two)
```

```
print()
print("Score:",round(accuracy_score(y_test_mil_two, preddt) * 100, 2))

print()
print("Classification Report:")
print(classification_report(y_test_mil_two, preddt))
```

```
Score: 88.02

Classification Report:
              precision    recall  f1-score   support

         1.0       0.88      0.88      0.88     15574
         5.0       0.88      0.88      0.88     15591

    accuracy                           0.88     31165
   macro avg       0.88      0.88      0.88     31165
weighted avg       0.88      0.88      0.88     31165
```

10.c.iii. Modeling 1,000,000 (1-star, 3-star, and 5-star) reviews with the Decision Tree Classifier:

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
dt_mil_3 = DecisionTreeClassifier()
dt_mil_3.fit(x_train_mil_three, y_train_mil_three)
preddt = dt_mil_3.predict(x_test_mil_three)

print()
print("Score:",round(accuracy_score(y_test_mil_three, preddt) * 100, 2))

print()
print("Classification Report:")
print(classification_report(y_test_mil_three, preddt))
```

```
Score: 67.67

Classification Report:
              precision    recall  f1-score   support

         1.0       0.73      0.72      0.72     15566
         3.0       0.59      0.60      0.59     15502
         5.0       0.71      0.72      0.71     15680

    accuracy                           0.68     46748
   macro avg       0.68      0.68      0.68     46748
weighted avg       0.68      0.68      0.68     46748
```

From the above models, we can compare the accuracy scores:

- Multinomial Naive Bayes:

	1-5 star	1 & 5 star	1, 3, 5 star
10,000	47.05	90.52	74.67
100,000	52.77	**94.9**	78.24
1,000,000	78.24	93.54	79.66

- Random Forest Classifier:

	1-5 star	1 & 5 star	1, 3, 5 star
10,000	41.81	91.18	70.09
100,000	49.67	93.71	76.84
1,000,000	77.93	**94.64**	79.98

- Decision Tree Classifier:

	1-5 star	1 & 5 star	1, 3, 5 star
10,000	30.14	77.45	52.62
100,000	37.56	85.67	63.78
1,000,000	63.45	**88.18**	67.77

Since the Multinomial Naive Bayes makes the most accurate prediction, let's use it to predict a sample positive, a sample neutral, and a sample negative review:

11. Classify a positive review:

In [ ]:
```
pos_rev = yelp_sample_equal_hund_thou['text'][5][1]
pos_rev_trans = vocab_hund_thou_two.transform([pos_rev])

print()
print("Sample positive review:")
print(pos_rev)

print()
print("Actual Rating: ", yelp_sample_equal_hund_thou['stars'][5][1])
print("Predicted Rating:", mnb_hund_2.predict(pos_rev_trans)[0])
```

```
Sample positive review:
I've taken a lot of spin classes over the years, and nothing compares to th
e classes at Body Cycle. From the nice, clean space and amazing bikes, to t
he welcoming and motivating instructors, every class is a top notch work ou
t.

For anyone who struggles to fit workouts in, the online scheduling system m
akes it easy to plan ahead (and there's no need to line up way in advanced
like many gyms make you do).

There is no way I can write this review without giving Russell, the owner o
f Body Cycle, a shout out. Russell's passion for fitness and cycling is so
evident, as is his desire for all of his clients to succeed. He is always d
ropping in to classes to check in/provide encouragement, and is open to ide
as and recommendations from anyone. Russell always wears a smile on his fac
e, even when he's kicking your butt in class!

Actual Rating:  5.0
Predicted Rating: 5.0
Sample positive review:
I've taken a lot of spin classes over the years, and nothing compares to th
e classes at Body Cycle. From the nice, clean space and amazing bikes, to t
he welcoming and motivating instructors, every class is a top notch work ou
t.

For anyone who struggles to fit workouts in, the online scheduling system m
akes it easy to plan ahead (and there's no need to line up way in advanced
like many gyms make you do).

There is no way I can write this review without giving Russell, the owner o
f Body Cycle, a shout out. Russell's passion for fitness and cycling is so
evident, as is his desire for all of his clients to succeed. He is always d
ropping in to classes to check in/provide encouragement, and is open to ide
as and recommendations from anyone. Russell always wears a smile on his fac
e, even when he's kicking your butt in class!

Actual Rating:  5.0
Predicted Rating: 5.0
```

12. Classify a negative review:

In [ ]:
```
neg_rev = yelp_sample_equal_hund_thou['text'][1][5]
neg_rev_trans = vocab_hund_thou_two.transform([neg_rev])

print()
print("Sample negative review:")
print(neg_rev)

print()
print("Actual Rating: ", yelp_sample_equal_hund_thou['stars'][1][5])
print("Predicted Rating:", mnb_hund_2.predict(neg_rev_trans)[0])
```

```
Sample negative review:
I am a long term frequent customer of this establishment. I just went in to
order take out (3 apps) and was told they're too busy to do it. Really? The
place is maybe half full at best. Does your dick reach your ass? Yes? Go fu
ck yourself! I'm a frequent customer AND great tipper. Glad that Kanella ju
st opened. NEVER going back to dmitris!

Actual Rating:  1.0
Predicted Rating: 1.0
```