PIC16B_final_project_code

June 16, 2024

1 PREDICTING TRUE STAR RATINGS OF YELP RESTAURANT REVIEWS

Group members: Cynthia Du, Meiyi Ye, Nam Truong

Github Repository: https://github.com/NamTTruong/PIC-16B-Project

1.1 Background

Yelp is a popular platform where users share their experiences and rate businesses on a scale from one to five stars. These reviews significantly influence consumer decisions and play a crucial role in shaping the reputations of businesses. Accurately predicting the ratings of these reviews helps in understanding consumer sentiment and preferences, which is vital for both users and business owners.

1.2 Problem Statement

Despite their usefulness, Yelp reviews often exhibit bias and subjectivity, which can skew perceptions and lead to misleading ratings. There is a need for a model that can predict the true rating of a Yelp review by mitigating individual biases and emphasizing the content's sentiment, thereby providing a more accurate reflection of the business's quality.

1.3 Objective

The objective of this project is to develop a predictive model that can determine the true star rating of a Yelp review based on its textual content. By analyzing patterns in word usage and sentiment, the model aims to offer a normalized rating that more accurately reflects the actual quality of the service.

2 Access to Necessary Files

Note: Our files were too large so we could not upload them to GitHub, hence we have decided to share our Google Drive that we used for this project.

How to set up our environment to run our code:

1. Download our Google Drive Folder here.
2. Upload those files to your personal Google Drive and name the folder `project personal work`.

3. Run our code!

```
[ ]: # Allows Google Colab to connect to your Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[ ]: # Go into the following directory: project personal work
%cd /content/drive/MyDrive/project personal work
```

```
/content/drive/MyDrive/project personal work
```

```
[ ]: # See what's in the directory to make sure we're in the right folder
%ls
```

```
 base_df_restaurants_ca.csv   Nam_Copy_of_explore2.ipynb
 df_ca.csv                    normalized_df_restaurants_ca.csv
 df_restaurants_ca2.csv      'PIC16 Report.gdoc'
 df_restaurants_ca3.csv      'sentiment analysis   pos neg words.ipynb'
 df_restaurants_ca.csv        yelp_academic_dataset_business.csv
 DistilBERT.ipynb             yelp_academic_dataset_business.json
 everything.ipynb             yelp_academic_dataset_review.csv
 explore1.ipynb               yelp_academic_dataset_review.json
 explore2.ipynb               yelp_academic_dataset_user.csv
 explore.ipynb                yelp_academic_dataset_user.json
 models.ipynb                 yelp_combined.csv
```

3 Convert JSON Files to CSV

```
[ ]: import json
import pandas as pd
```

```
[ ]: import pandas as pd
from tqdm.auto import tqdm  # Import tqdm for the progress bar

file_path = 'yelp_academic_dataset_business.json'
csv_file_path = 'yelp_academic_dataset_business.csv'

# We need to know the total number of chunks to configure tqdm. This can be␣
 ↪tricky with files.
# An alternative approach is to estimate the number of iterations by dividing␣
 ↪the file size by an estimated chunk size in bytes.
# However, for simplicity and given we might not know the exact number of lines␣
 ↪or their distribution, we'll initialize tqdm without total.

chunk_size = 10000  # Adjust based on your system's memory
```

```
# Initialize the progress bar. We won't set 'total' as it's hard to determine␣
↪without reading the file twice.
pbar = tqdm(desc="Processing", unit="chunk")

with pd.read_json(file_path, lines=True, chunksize=chunk_size) as reader:
    for i, chunk in enumerate(reader):
        mode = 'w' if i == 0 else 'a'
        header = True if i == 0 else False
        chunk.to_csv(csv_file_path, mode=mode, index=False, header=header)
        pbar.update(1)  # Update the progress bar by one iteration

pbar.close()  # Ensure the progress bar is properly closed after the operation

print("Conversion to CSV completed.")
```

```
Processing: 0chunk [00:00, ?chunk/s]
```

```
Conversion to CSV completed.
```

[ ]:

```
import pandas as pd
from tqdm.auto import tqdm  # Import tqdm for the progress bar

file_path = 'yelp_academic_dataset_user.json'
csv_file_path = 'yelp_academic_dataset_user.csv'

# We need to know the total number of chunks to configure tqdm. This can be␣
↪tricky with files.
# An alternative approach is to estimate the number of iterations by dividing␣
↪the file size by an estimated chunk size in bytes.
# However, for simplicity and given we might not know the exact number of lines␣
↪or their distribution, we'll initialize tqdm without total.

chunk_size = 10000  # Adjust based on your system's memory

# Initialize the progress bar. We won't set 'total' as it's hard to determine␣
↪without reading the file twice.
pbar = tqdm(desc="Processing", unit="chunk")

with pd.read_json(file_path, lines=True, chunksize=chunk_size) as reader:
    for i, chunk in enumerate(reader):
        mode = 'w' if i == 0 else 'a'
        header = True if i == 0 else False
        chunk.to_csv(csv_file_path, mode=mode, index=False, header=header)
        pbar.update(1)  # Update the progress bar by one iteration

pbar.close()  # Ensure the progress bar is properly closed after the operation

print("Conversion to CSV completed.")
```

```
Processing: 0chunk [00:00, ?chunk/s]

Conversion to CSV completed.
```

[ ]:
```python
import pandas as pd
from tqdm.auto import tqdm  # Import tqdm for the progress bar

file_path = 'yelp_academic_dataset_review.json'
csv_file_path = 'yelp_academic_dataset_review.csv'

# We need to know the total number of chunks to configure tqdm. This can be␣
↪tricky with files.
# An alternative approach is to estimate the number of iterations by dividing␣
↪the file size by an estimated chunk size in bytes.
# However, for simplicity and given we might not know the exact number of lines␣
↪or their distribution, we'll initialize tqdm without total.

chunk_size = 10000  # Adjust based on your system's memory

# Initialize the progress bar. We won't set 'total' as it's hard to determine␣
↪without reading the file twice.
pbar = tqdm(desc="Processing", unit="chunk")

with pd.read_json(file_path, lines=True, chunksize=chunk_size) as reader:
    for i, chunk in enumerate(reader):
        mode = 'w' if i == 0 else 'a'
        header = True if i == 0 else False
        chunk.to_csv(csv_file_path, mode=mode, index=False, header=header)
        pbar.update(1)  # Update the progress bar by one iteration

pbar.close()  # Ensure the progress bar is properly closed after the operation

print("Conversion to CSV completed.")
```

```
Processing: 0chunk [00:00, ?chunk/s]

Conversion to CSV completed.
```

[ ]:
```python
import pandas as pd
import numpy as np

review_path = 'yelp_academic_dataset_review.csv'
business_path = 'yelp_academic_dataset_business.csv'
user_limited_path = 'yelp_academic_dataset_user.csv'
#sample_fraction = 0.1  # For example, 10% of the data

# Determine the number of rows to skip
#skip = lambda i: i > 0 and np.random.random() > sample_fraction
```

```
# Load the random subset of the data
#df = pd.read_csv(file_path, skiprows=skip)
df_review = pd.read_csv(review_path)
df_review = df_review[['user_id','business_id','stars','text']]
df_business = pd.read_csv(business_path)
df_user = pd.read_csv(user_limited_path)
df_user = df_user[['user_id','average_stars']]
```

We Converted our users, reviews, businesses JSON datasets into more manageable CSV files allowed us to efficiently handle the large volume of data, facilitating more effective data manipulation, cleaning, and eventual analysis.

[ ]:
```
df_review.head()
```

[ ]:
```
                  user_id             business_id  stars  \
0  mh_-eMZ6K5RLWhZyISBhwA  XQfwVwDr-v0ZS3_CbbE5Xw    3.0
1  OyoGAe7OKpv6SyGZT5g77Q  7ATYjTIgM3jUlt4UM3IypQ    5.0
2  8g_iMtfSiwikVnbP2etR0A  YjUWPpI6HXG530lwP-fb2A    3.0
3  _7bHUi9Uuf5__HHc_Q8guQ  kxX2SOes4o-D3ZQBkiMRfA    5.0
4  bcjbaE6dDog4jkNY91ncLQ  e4Vwtrqf-wpJfwesgvdgxQ    4.0

                                                text
0  If you decide to eat here, just be aware it is…
1  I've taken a lot of spin classes over the year…
2  Family diner. Had the buffet. Eclectic assortm…
3  Wow!  Yummy, different,  delicious.   Our favo…
4  Cute interior and owner (?) gave us tour of up…
```

[ ]:
```
print(df_review.shape)
```

```
(6990282, 4)
```

There are 6,990,282 observations and 4 features in the review data frame: user_id, business_id, stars, and the text (review).

[ ]:
```
df_business.head()
```

[ ]:
```
              business_id                      name  \
0  Pns2l4eNsfO8kk83dixA6A  Abby Rappoport, LAC, CMQ
1  mpf3x-BjTdTEA3yCZrAYPw             The UPS Store
2  tUFrWirKiKi_TAnsVWINQQ                    Target
3  MTSW4McQd7CbVtyjqoe9mw        St Honore Pastries
4  mWMc6_wTdE0EUBKIGXDVfA  Perkiomen Valley Brewery

                           address           city state postal_code  \
0           1616 Chapala St, Ste 2  Santa Barbara    CA       93101
1  87 Grasso Plaza Shopping Center         Affton    MO       63123
2             5255 E Broadway Blvd         Tucson    AZ       85711
3                      935 Race St   Philadelphia    PA       19107
```

```
4                    101 Walnut St     Green Lane    PA       18054

    latitude   longitude  stars  review_count  is_open  \
0  34.426679 -119.711197    5.0             7        0
1  38.551126  -90.335695    3.0            15        1
2  32.223236 -110.880452    3.5            22        0
3  39.955505  -75.155564    4.0            80        1
4  40.338183  -75.471659    4.5            13        1

                                          attributes  \
0                      {'ByAppointmentOnly': 'True'}
1             {'BusinessAcceptsCreditCards': 'True'}
2  {'BikeParking': 'True', 'BusinessAcceptsCredit…
3  {'RestaurantsDelivery': 'False', 'OutdoorSeati…
4  {'BusinessAcceptsCreditCards': 'True', 'Wheelc…

                                          categories  \
0  Doctors, Traditional Chinese Medicine, Naturop…
1  Shipping Centers, Local Services, Notaries, Ma…
2  Department Stores, Shopping, Fashion, Home & G…
3  Restaurants, Food, Bubble Tea, Coffee & Tea, B…
4                          Brewpubs, Breweries, Food

                                               hours
0                                                NaN
1  {'Monday': '0:0-0:0', 'Tuesday': '8:0-18:30', …
2  {'Monday': '8:0-22:0', 'Tuesday': '8:0-22:0', …
3  {'Monday': '7:0-20:0', 'Tuesday': '7:0-20:0', …
4  {'Wednesday': '14:0-22:0', 'Thursday': '16:0-2…
```

[ ]:
```
print(df_business.shape)
```

```
(150346, 14)
```

There are 150,346 observations and 14 features in the business data frame: business_id, name, address, city, state, postal_code, latitude, longitude, stars, review_count, is_open, attributes, categories, and hours.

[ ]:
```
df_user.head()
```

[ ]:
```
                  user_id  average_stars
0  qVc8ODYU5SZjKXVBgXdI7w           3.91
1  j14WgRoU_-2ZE1aw1dXrJg           3.74
2  2WnXYQFK0hXEoTxPtV2zvg           3.32
3  SZDeASXq7o05mMNLshsdIA           4.27
4  hA5lMy-EnncsH4JoR-hFGQ           3.54
```

[ ]:
```
print(df_user.shape)
```

```
(692471, 2)
```

There ara 692,471 obversations and 2 features in the user data frame: user_id, and the average_stars.

4 Merge and Filter

[ ]:
```python
df_review_user = pd.merge(df_review, df_user, on='user_id', how='left')


df_combined = pd.merge(df_review_user, df_business, on='business_id',␣
 ↪how='left', suffixes=('_review', '_business'))


df_combined.head()
```

[ ]:
```
                  user_id             business_id  stars_review  \
0  mh_-eMZ6K5RLWhZyISBhwA  XQfwVwDr-v0ZS3_CbbE5Xw           3.0
1  OyoGAe7OKpv6SyGZT5g77Q  7ATYjTIgM3jUlt4UM3IypQ           5.0
2  8g_iMtfSiwikVnbP2etR0A  YjUWPpI6HXG530lwP-fb2A           3.0
3  _7bHUi9Uuf5__HHc_Q8guQ  kxX2SOes4o-D3ZQBkiMRfA           5.0
4  bcjbaE6dDog4jkNY91ncLQ  e4Vwtrqf-wpJfwesgvdgxQ           4.0

                                                text  average_stars  \
0  If you decide to eat here, just be aware it is…           4.06
1  I've taken a lot of spin classes over the year…           4.30
2  Family diner. Had the buffet. Eclectic assortm…           4.69
3  Wow!  Yummy, different,  delicious.   Our favo…           4.78
4  Cute interior and owner (?) gave us tour of up…           2.97

                          name                 address          city state  \
0  Turning Point of North Wales       1460 Bethlehem Pike   North Wales    PA
1    Body Cycle Spinning Studio  1923 Chestnut St, 2nd Fl  Philadelphia    PA
2             Kettle Restaurant     748 W Starr Pass Blvd        Tucson    AZ
3                        Zaika            2481 Grant Ave  Philadelphia    PA
4                         Melt             2549 Banks St   New Orleans    LA

  postal_code   latitude   longitude  stars_business  review_count  is_open  \
0       19454  40.210196  -75.223639             3.0         169.0      1.0
1       19119  39.952103  -75.172753             5.0         144.0      0.0
2       85713  32.207233 -110.980864             3.5          47.0      1.0
3       19114  40.079848  -75.025080             4.0         181.0      1.0
4       70119  29.962102  -90.087958             4.0          32.0      0.0

                                          attributes  \
0  {'NoiseLevel': "u'average'", 'HasTV': 'False',…
1  {'BusinessAcceptsCreditCards': 'True', 'GoodFo…
```

```
2  {'RestaurantsReservations': 'True', 'BusinessP…
3  {'Caters': 'True', 'Ambience': "{'romantic': F…
4  {'BusinessParking': "{'garage': False, 'street…

                                          categories  \
0  Restaurants, Breakfast & Brunch, Food, Juice B…
1  Active Life, Cycling Classes, Trainers, Gyms, …
2                    Restaurants, Breakfast & Brunch
3              Halal, Pakistani, Restaurants, Indian
4  Sandwiches, Beer, Wine & Spirits, Bars, Food, …

                                               hours
0  {'Monday': '7:30-15:0', 'Tuesday': '7:30-15:0'…
1  {'Monday': '6:30-20:30', 'Tuesday': '6:30-20:3…
2                                                NaN
3  {'Tuesday': '11:0-21:0', 'Wednesday': '11:0-21…
4  {'Monday': '0:0-0:0', 'Friday': '11:0-17:0', '…
```

[ ]:
```
print(df_combined.shape)
```

```
(6990282, 18)
```

We merged user data with review data based on 'user_id' to connect each review with its user's ratings, forming the df_review_user dataframe. Then, we combined this with business data on 'business_id' to ensure each review accurately matches its respective business, resulting in the df_combined dataframe. df_combined data frame has 6,990,282 observations and 18 features: user_id, business_id, stars_review, text, average stars, name, address, city, state, postal_code, latitude, longitude, stars_business, review_count, is_open, attributes, categories, hours.

[ ]:
```
df_combined.to_csv('yelp_combined.csv')
```

Wrote our data frame into a csv file.

[ ]:
```
df_combined['state'].value_counts()
```

[ ]:
```
state
PA    1598960
FL    1161545
LA     761673
TN     614388
MO     502385
IN     489752
AZ     431708
NV     430678
CA     348856
NJ     260897
ID     157572
AB     109436
DE      70302
```

```
IL       51832
MA          44
SD          42
TX          35
HI          34
CO          31
NC          29
WA          19
UT          19
MI          11
VI          11
VT          10
MT           6
XMS          5
Name: count, dtype: int64
```

We did a value counts and this showed us how many reviews we have for each state.

```
[ ]: # Filter rows for CA in the states column
     df_ca = df_combined[df_combined['state'] == 'CA']

     # Filter the DataFrame to keep rows where 'categories' contains 'Restaurants'
     df_restaurants_ca = df_ca[df_ca['categories'].str.contains('Restaurants',␣
      ↪na=False)]

     # Print the filtered DataFrame
     df_restaurants_ca.head()
```

```
[ ]:                  user_id             business_id  stars_review  \
     9   59MxRhNVhU9MYndMkz0wtw  gebiRewfieSdtt17PTW6Zg           3.0
     23  OhECKhQEexFypOMY6kypRw  vC2qm1y3Au5czBtbhc-DNw           4.0
     31  4hBhtCSgoxkrFgHa4YAD-w  bbEXAEFr4RYHLlZ-HFssTA           5.0
     35  bFPdtzu11Oi0f92EAcjqmg  IDtLPgUrqorrpqSLdfMhZQ           5.0
     61  JYYYKt6TdVA4ng9lLcXt_g  SZU9c8V2GuREDN5KgyHFJw           5.0

                                                      text  average_stars  \
     9   Had a party of 6 here for hibachi. Our waitres…           4.23
     23  Yes, this is the only sushi place in town. How…           3.96
     31  Great burgers,fries and salad!  Burgers have a…           4.20
     35  What a great addition to the Funk Zone!  Grab …           4.06
     61  We were a bit weary about trying the Shellfish…           4.12

                                    name               address           city  \
     9   Hibachi Steak House & Sushi Bar          502 State St  Santa Barbara
     23                       Sushi Teri        970 Linden Ave    Carpinteria
     31  The Original Habit Burger Grill      5735 Hollister Ave         Goleta
     35              Helena Avenue Bakery  131 Anacapa St, Ste C  Santa Barbara
     61  Santa Barbara Shellfish Company      230 Stearns Wharf  Santa Barbara
```

```
   state postal_code   latitude   longitude  stars_business  review_count  \
9     CA       93101  34.416984 -119.695556             3.5         488.0
23    CA       93013  34.398527 -119.518475             3.0         167.0
31    CA       93117  34.435570 -119.824706             4.0         329.0
35    CA       93101  34.414445 -119.690672             4.0         389.0
61    CA       93101  34.408715 -119.685019             4.0        2404.0

    is_open                                         attributes  \
9       1.0  {'Corkage': 'False', 'RestaurantsTakeOut': 'Tr…
23      1.0  {'RestaurantsReservations': 'True', 'NoiseLeve…
31      1.0  {'Caters': 'False', 'GoodForKids': 'True', 'BY…
35      1.0  {'RestaurantsTakeOut': 'True', 'NoiseLevel': "…
61      1.0  {'OutdoorSeating': 'True', 'RestaurantsAttire'…

                                           categories  \
9      Steakhouses, Sushi Bars, Restaurants, Japanese
23                           Restaurants, Sushi Bars
31                       Fast Food, Burgers, Restaurants
35  Food, Restaurants, Salad, Coffee & Tea, Breakf…
61  Live/Raw Food, Restaurants, Seafood, Beer Bar,…

                                           hours
9                             {'Monday': '0:0-0:0'}
23  {'Monday': '17:0-22:0', 'Tuesday': '17:0-22:0'…
31  {'Monday': '0:0-0:0', 'Tuesday': '10:30-21:0',…
35  {'Monday': '0:0-0:0', 'Tuesday': '8:0-14:0', '…
61  {'Monday': '0:0-0:0', 'Tuesday': '11:0-21:0', …
```

After conducting a value count analysis of the number of reviews per state from our combined dataset, we observed significant variations in the volume of reviews across states. Notably, California stood out with a substantial total of 348,856 reviews, making it an ideal candidate for our analysis. This substantial data volume ensures a robust dataset, providing a comprehensive insight into consumer sentiments and behaviors specific to California. Therefore, we decided to focus our predictive modeling efforts on this state. Furthermore, we wanted to concentrate our data on the Restaurants in California, rather than all business types.

[ ]:
```
df_restaurants_ca.to_csv("df_restaurants_ca.csv")
```

We wrote df_restaurants_ca data frame into a csv file.

[ ]:
```
df_restaurants_ca.shape
```

[ ]:
```
(211748, 18)
```

The df_restaurants_ca has 211748 observations and 18 features.

4.1 Fixing "Santa Barbara" Spacing Issues in "city" column

[ ]:
```python
import pandas as pd

df_restaurants_ca = pd.read_csv("df_restaurants_ca.csv")
```

[ ]:
```python
# 2 versions of 'Santa Barbara', fixing that here
df_restaurants_ca['city'] = df_restaurants_ca['city'].str.replace(r'\s+', ' ',␣
↪regex=True).str.strip()
df_restaurants_ca['city'].unique()
```

[ ]:
```
array(['Santa Barbara', 'Carpinteria', 'Goleta', 'Montecito',
       'Isla Vista', 'Summerland', 'Truckee'], dtype=object)
```

[ ]:
```python
# Verify that fix works (there shouldn't be two versions of "Santa Barbara" now)
df_restaurants_ca["city"].value_counts()
```

[ ]:
```
city
Santa Barbara    162430
Goleta            26584
Carpinteria       11416
Isla Vista         6330
Montecito          3638
Summerland         1324
Truckee              26
Name: count, dtype: int64
```

[ ]:
```python
# Convert fixed csv to a new csv file named "base_df_restaurants_ca.csv"
df_restaurants_ca.to_csv('base_df_restaurants_ca.csv', index=False)
```

5 Removing Stopwords

[ ]:
```python
import pandas as pd
import numpy as np

df_restaurants_ca_path = 'df_restaurants_ca.csv'
df_restaurants_ca = pd.read_csv(df_restaurants_ca_path)
```

[ ]:
```python
df_restaurants_ca.head()
```

[ ]:
```
                   user_id             business_id  stars_review  \
9   59MxRhNVhU9MYndMkz0wtw  gebiRewfieSdtt17PTW6Zg           3.0
23  OhECKhQEexFypOMY6kypRw  vC2qm1y3Au5czBtbhc-DNw           4.0
31  4hBhtCSgoxkrFgHa4YAD-w  bbEXAEFr4RYHLlZ-HFssTA           5.0
35  bFPdtzu11Oi0f92EAcjqmg  IDtLPgUrqorrpqSLdfMhZQ           5.0
61  JYYYKt6TdVA4ng9lLcXt_g  SZU9c8V2GuREDN5KgyHFJw           5.0
```

```
                                                text  average_stars  \
9   Had a party of 6 here for hibachi. Our waitres…           4.23
23  Yes, this is the only sushi place in town. How…           3.96
31  Great burgers,fries and salad!  Burgers have a…           4.20
35  What a great addition to the Funk Zone!  Grab …           4.06
61  We were a bit weary about trying the Shellfish…           4.12

                                name               address           city  \
9   Hibachi Steak House & Sushi Bar          502 State St  Santa Barbara
23                       Sushi Teri        970 Linden Ave    Carpinteria
31  The Original Habit Burger Grill      5735 Hollister Ave         Goleta
35              Helena Avenue Bakery  131 Anacapa St, Ste C  Santa Barbara
61  Santa Barbara Shellfish Company      230 Stearns Wharf  Santa Barbara

   state postal_code   latitude   longitude  stars_business  review_count  \
9     CA       93101  34.416984 -119.695556             3.5         488.0
23    CA       93013  34.398527 -119.518475             3.0         167.0
31    CA       93117  34.435570 -119.824706             4.0         329.0
35    CA       93101  34.414445 -119.690672             4.0         389.0
61    CA       93101  34.408715 -119.685019             4.0        2404.0

    is_open                                         attributes  \
9       1.0  {'Corkage': 'False', 'RestaurantsTakeOut': 'Tr…
23      1.0  {'RestaurantsReservations': 'True', 'NoiseLeve…
31      1.0  {'Caters': 'False', 'GoodForKids': 'True', 'BY…
35      1.0  {'RestaurantsTakeOut': 'True', 'NoiseLevel': "…
61      1.0  {'OutdoorSeating': 'True', 'RestaurantsAttire'…

                                         categories  \
9      Steakhouses, Sushi Bars, Restaurants, Japanese
23                          Restaurants, Sushi Bars
31                     Fast Food, Burgers, Restaurants
35  Food, Restaurants, Salad, Coffee & Tea, Breakf…
61  Live/Raw Food, Restaurants, Seafood, Beer Bar,…

                                               hours
9                              {'Monday': '0:0-0:0'}
23  {'Monday': '17:0-22:0', 'Tuesday': '17:0-22:0'…
31  {'Monday': '0:0-0:0', 'Tuesday': '10:30-21:0',…
35  {'Monday': '0:0-0:0', 'Tuesday': '8:0-14:0', '…
61  {'Monday': '0:0-0:0', 'Tuesday': '11:0-21:0', …
```

[ ]:
```python
# Set option to prevent truncation of 'text' column output
pd.set_option('display.max_colwidth', None)

# Before removing stopwords
df_restaurants_ca['text'].head()
```

```
[ ]: 0
Had a party of 6 here for hibachi. Our waitress brought our separate sushi
orders on one plate so we couldn't really tell who's was who's and forgot
several items on an order. I understand making mistakes but the restaraunt was
really quiet so we were kind of surprised. Usually hibachi is a fun lively
experience and our  cook  said maybe three words, but he cooked very well his
name was Francisco. Service was fishy, food was pretty good, and im hoping it
was just an off night here. But for the money I wouldn't go back.
1
Yes, this is the only sushi place in town. However, it is great when you're
craving sushi and don't have time to go somewhere else. The salmon is probably
the best fish they have, so we always order salmon. We also love their spicy
edamame, tempura, ocean salad, and cabbage salad. Service has always been
friendly and quick!
2
Great burgers,fries and salad!  Burgers have a hint of salt and pepper
flavor.\n\nThis location is very quaint. They only have outdoor
seating\n\nFriendly staff.\n\nStreet parking as well as parking lot in the back.
3
What a great addition to the Funk Zone!  Grab a bite, grab some tastings, life
is good. Right next door to the Santa Barbara Wine Collective, in fact it
actually shares the same tables.  We had a fabulous savory croissant.
4    We were a bit weary about trying the Shellfish Company on the Wharf as more
often than not, many places like these (see Cannery Row, Monterey) feast on a
captive audience and provide sub-standard fare at high prices.\n\nHowever,
emboldened by the perennial good reviews on Yelp, we suppressed our initial
observations and went ahead with the trying it out. The place is small, so
definitely plan ahead. You will have to wait, so either you know, just do so, or
perhaps try to visit outside of peak hours. Luckily, our wait was only about 20
minutes as the dinner rush was just leveling off.\n\nThe special was the local
rock crab - $25 for 3 lbs of California Rock Crab, salad, and your choice of
soup/chowder. After taking a look at a few trays of rock crab being served out,
the wife and I both opted for it, as it looked awesome.  \n\nThe salad/chowder
combo was great as you actually received hearty portions of each, so it was a
good start. As much as I liked the chowder however, the Shrimp Bisque the wife
ordered was amazing, so I would recommend that going forward. \n\nBut enough of
prattling on about side dishes: the rock crab tasted just as glorious as it
looked. Juicy, buttery chunks of white crab meat await you, just a few cracks
away. While the rock crab shells are pretty thick, once cracked, they splinter
and separate easily, a good sign they are cooked to perfection. The rock crabs
provided a great amount of meat for what I felt to be the least amount of work
you're going to do for crab, King or otherwise. \n\nWe were thoroughly satisfied
with our meal and along with the special being an overall great value, devouring
the Rock Crab at SBSC turned out to be one of the favorite meals on the trip. I
think ordering crab here is a safe proposition indeed.
5
If I could give it a zero, I would. I order a plain hamburger, and realized they
```

```
put  bacon in it (which I am allergic to and unable to eat) after two bites.
When I went back to the drive-through window to complain (didn't realize the
actual restaurant was open--it was almost 2 after all…), the guy took back the
burger, said nothing, and disappeared. After 2 minutes of awkwardly making
conversation with the next people in line in their car, he came back and rudely
told me I had to go inside to get my food. Which I did. And still did not get an
apology.\n\nI refuse to go back there after that ordeal, which is a shame,
because it's nice to have a variety of places to go to after DT. Guess Freebirds
it is!
6
We visited once and were very disappointed in my veggie pizza and my husband's
sub sandwich.  The tomato sauce was not tasty, and they did not use enough
cheese on my pizza.  The dough looked and tasted like it was prepared by a
machine.  Perhaps they have improved, but we are not in any rush to try a second
time.
7
This is the first time I tried this place and  I was surprisingly surprised. I
had a combination dinner pad Thai and coconut soup. The soup was very tasty as I
never had coconut soup before. The pad Thai was exactly what I was expecting and
it did not disappoint. The restaurant had great Thai decor and music. The staff
and service was top notch. For a town with not much selection for food, this was
a great change of pace. This may become my go to place in Carp.
8
So disappointing on so many levels. Have been coming here for years - and the
quality of food has fallen off a cliff. Strike one - we shared an artichoke to
start - and clearly it had been prepared beforehand (eg, cold on the inside -
but the flesh was cooked through)  Strike two - my wife had a Greek salad - and
it clearly all came from a bag and was extremely overdressed with tasteless
dressing.  Strike three - I had grilled yellowtail which was basically execrable
- could not even finish it (overcooked and an inferior-frozen piece of fish -
unacceptable for SB). \n\nIn short - great setting, good service - but with such
horrible food - not worth the visit. This place really needs to up the quality
of food - or it's going to end up just another lower State dive for the
tourists.
9
We absolutely love everything we have tried here. Our favorite thus far has been
the Backyard Bowl, although there are several more that we still need to try. My
husband and I are always full after we each get a kids size, or split a large.
The ingredients are good quality,  filling foods. For the quality of
ingredients, the price is very reasonable. It can get pretty busy here, but have
patience, it is worth the wait.
Name: text, dtype: object
```

[ ]:
```python
# Undo our changed option to avoid affecting the rest of our output
pd.reset_option('display.max_colwidth')
```

[ ]:
```python
# Takes about 7 minutes to run

# Used to remove/filter stopwords
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# Download the stopwords from NLTK
nltk.download('punkt')
nltk.download('stopwords')

def remove_stopwords(text):
    stop_words = set(stopwords.words('english'))

    word_tokens = word_tokenize(text)

    filtered_text = [word for word in word_tokens if word.casefold() not in␣
 ↪stop_words]
    return " ".join(filtered_text)

# Apply the function to the 'text' column
# 'text' column is the reviews
df_restaurants_ca['text'] = df_restaurants_ca['text'].apply(remove_stopwords)
```

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

[ ]:
```python
# Set option to prevent truncation of 'text' column output
pd.set_option('display.max_colwidth', None)

# After removing stopwords
df_restaurants_ca['text'].head()
```

[ ]:
```
0
party 6 hibachi . waitress brought separate sushi orders one plate could n't
really tell 's 's forgot several items order . understand making mistakes
restaraunt really quiet kind surprised . Usually hibachi fun lively experience
cook said maybe three words , cooked well name Francisco . Service fishy , food
pretty good , im hoping night . money would n't go back .
1
Yes , sushi place town . However , great 're craving sushi n't time go somewhere
else . salmon probably best fish , always order salmon . also love spicy edamame
, tempura , ocean salad , cabbage salad . Service always friendly quick !
2
Great burgers , fries salad ! Burgers hint salt pepper flavor . location quaint
. outdoor seating Friendly staff . Street parking well parking lot back .
```

3
great addition Funk Zone ! Grab bite , grab tastings , life good . Right next door Santa Barbara Wine Collective , fact actually shares tables . fabulous savory croissant .
4 bit weary trying Shellfish Company Wharf often , many places like (see Cannery Row , Monterey) feast captive audience provide sub-standard fare high prices . However , emboldened perennial good reviews Yelp , suppressed initial observations went ahead trying . place small , definitely plan ahead . wait , either know , , perhaps try visit outside peak hours . Luckily , wait 20 minutes dinner rush leveling . special local rock crab - $ 25 3 lbs California Rock Crab , salad , choice soup/chowder . taking look trays rock crab served , wife opted , looked awesome . salad/chowder combo great actually received hearty portions , good start . much liked chowder however , Shrimp Bisque wife ordered amazing , would recommend going forward . enough prattling side dishes : rock crab tasted glorious looked . Juicy , buttery chunks white crab meat await , cracks away . rock crab shells pretty thick , cracked , splinter separate easily , good sign cooked perfection . rock crabs provided great amount meat felt least amount work 're going crab , King otherwise . thoroughly satisfied meal along special overall great value , devouring Rock Crab SBSC turned one favorite meals trip . think ordering crab safe proposition indeed .
5
could give zero , would . order plain hamburger , realized put bacon (allergic unable eat) two bites . went back drive-through window complain (n't realize actual restaurant open -- almost 2 …) , guy took back burger , said nothing , disappeared . 2 minutes awkwardly making conversation next people line car , came back rudely told go inside get food . . still get apology . refuse go back ordeal , shame , 's nice variety places go DT . Guess Freebirds !
6
visited disappointed veggie pizza husband 's sub sandwich . tomato sauce tasty , use enough cheese pizza . dough looked tasted like prepared machine . Perhaps improved , rush try second time .
7
first time tried place surprisingly surprised . combination dinner pad Thai coconut soup . soup tasty never coconut soup . pad Thai exactly expecting disappoint . restaurant great Thai decor music . staff service top notch . town much selection food , great change pace . may become go place Carp .
8
disappointing many levels . coming years - quality food fallen cliff . Strike one - shared artichoke start - clearly prepared beforehand (eg , cold inside - flesh cooked) Strike two - wife Greek salad - clearly came bag extremely overdressed tasteless dressing . Strike three - grilled yellowtail basically execrable - could even finish (overcooked inferior-frozen piece fish - unacceptable SB) . short - great setting , good service - horrible food - worth visit . place really needs quality food - 's going end another lower State dive tourists .
9
absolutely love everything tried . favorite thus far Backyard Bowl , although

```
several still need try . husband always full get kids size , split large .
ingredients good quality , filling foods . quality ingredients , price
reasonable . get pretty busy , patience , worth wait .
Name: text, dtype: object
```

[ ]:
```
# Undo our changed option to avoid affecting the rest of our output
pd.reset_option('display.max_colwidth')
```

[ ]:
```
df_restaurants_ca.head()
```

[ ]:
```
   Unnamed: 0                 user_id             business_id  stars_review  \
0           9  59MxRhNVhU9MYndMkz0wtw  gebiRewfieSdtt17PTW6Zg           3.0
1          23  0hECKhQEexFypOMY6kypRw  vC2qm1y3Au5czBtbhc-DNw           4.0
2          31  4hBhtCSgoxkrFgHa4YAD-w  bbEXAEFr4RYHLlZ-HFssTA           5.0
3          35  bFPdtzu11Oi0f92EAcjqmg  IDtLPgUrqorrpqSLdfMhZQ           5.0
4          61  JYYYKt6TdVA4ng9lLcXt_g  SZU9c8V2GuREDN5KgyHFJw           5.0
5          73  UsBxLh14sUpO8SdeqIiGOA  Wy8Hswf2cLQGRZN6armkag           1.0
6          79  3inG_FUhm28tUJc0zZ2fCg  aY_n9RSaD2Yw09jSFFePew           1.0
7          90  C_2mNjl-doRVvsL03_T57Q  18eWJFJbXyR9j_5xfcRLYA           4.0
8         126  QacYrFyCrFLmC726YEEBsA  29YqJwOGEuAWqlHZxMc1OA           1.0
9         143  cpp89UW0imv45y3a_0444w  34Eqv8jXgxg_EEwcsNgeeg           5.0

                                                text  average_stars  \
0  party 6 hibachi . waitress brought separate su…           4.23
1  Yes , sushi place town . However , great 're c…           3.96
2  Great burgers , fries salad ! Burgers hint sal…           4.20
3  great addition Funk Zone ! Grab bite , grab ta…           4.06
4  bit weary trying Shellfish Company Wharf often…           4.12
5  could give zero , would . order plain hamburge…           4.00
6  visited disappointed veggie pizza husband 's s…           4.79
7  first time tried place surprisingly surprised …           4.07
8  disappointing many levels . coming years - qua…           3.93
9  absolutely love everything tried . favorite th…           3.76

                            name              address           city  \
0  Hibachi Steak House & Sushi Bar         502 State St  Santa Barbara
1                       Sushi Teri       970 Linden Ave    Carpinteria
2     The Original Habit Burger Grill    5735 Hollister Ave         Goleta
3              Helena Avenue Bakery  131 Anacapa St, Ste C  Santa Barbara
4   Santa Barbara Shellfish Company     230 Stearns Wharf  Santa Barbara
5                    Jack in the Box    6875 Hollister Ave         Goleta
6                     PizzaMan Dan's       699 Linden Ave    Carpinteria
7                     Siam Elephant       509 Linden Ave    Carpinteria
8                     Paradise Cafe       702 Anacapa St  Santa Barbara
9                     Backyard Bowls        331 Motor Way  Santa Barbara

  state  postal_code   latitude   longitude  stars_business  review_count  \
```

```
0    CA       93101  34.416984 -119.695556           3.5         488.0
1    CA       93013  34.398527 -119.518475           3.0         167.0
2    CA       93117  34.435570 -119.824706           4.0         329.0
3    CA       93101  34.414445 -119.690672           4.0         389.0
4    CA       93101  34.408715 -119.685019           4.0        2404.0
5    CA       93117  34.429897 -119.868783           1.5          86.0
6    CA       93013  34.396959 -119.521063           4.0         124.0
7    CA       93013  34.396510 -119.521681           4.5         460.0
8    CA       93101  34.420035 -119.696851           3.5         290.0
9    CA       93101  34.415114 -119.694497           4.0         659.0

   is_open                                        attributes  \
0      1.0  {'Corkage': 'False', 'RestaurantsTakeOut': 'Tr…
1      1.0  {'RestaurantsReservations': 'True', 'NoiseLeve…
2      1.0  {'Caters': 'False', 'GoodForKids': 'True', 'BY…
3      1.0  {'RestaurantsTakeOut': 'True', 'NoiseLevel': "…
4      1.0  {'OutdoorSeating': 'True', 'RestaurantsAttire'…
5      1.0  {'BusinessAcceptsCreditCards': 'True', 'BikePa…
6      1.0  {'GoodForKids': 'True', 'BusinessParking': "{'…
7      1.0  {'RestaurantsGoodForGroups': 'True', 'Alcohol'…
8      0.0  {'BusinessParking': "{'garage': True, 'street'…
9      1.0  {'RestaurantsAttire': "u'casual'", 'BusinessAc…

                                         categories  \
0     Steakhouses, Sushi Bars, Restaurants, Japanese
1                            Restaurants, Sushi Bars
2                    Fast Food, Burgers, Restaurants
3  Food, Restaurants, Salad, Coffee & Tea, Breakf…
4  Live/Raw Food, Restaurants, Seafood, Beer Bar,…
5  Restaurants, Fast Food, Mexican, Tacos, Burger…
6  Chicken Wings, Restaurants, Beer, Wine & Spiri…
7                                  Restaurants, Thai
8  American (Traditional), American (New), Nightl…
9  Health Markets, Coffee & Tea, Ice Cream & Froz…

                                              hours
0                            {'Monday': '0:0-0:0'}
1  {'Monday': '17:0-22:0', 'Tuesday': '17:0-22:0'…
2  {'Monday': '0:0-0:0', 'Tuesday': '10:30-21:0',…
3  {'Monday': '0:0-0:0', 'Tuesday': '8:0-14:0', '…
4  {'Monday': '0:0-0:0', 'Tuesday': '11:0-21:0', …
5  {'Monday': '0:0-0:0', 'Tuesday': '0:0-0:0', 'W…
6  {'Monday': '11:0-23:0', 'Tuesday': '11:0-23:0'…
7  {'Tuesday': '17:0-21:30', 'Wednesday': '17:0-2…
8  {'Monday': '11:0-15:0', 'Tuesday': '11:0-21:0'…
9  {'Monday': '8:0-17:0', 'Tuesday': '8:0-17:0', …
```

6 Normalization

[ ]:
```python
import pandas as pd

df_restaurants_ca = pd.read_csv("base_df_restaurants_ca.csv")
```

[ ]:
```python
import string
import spacy
import re
import unicodedata
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

nlp = spacy.load("en_core_web_sm", disable=["parser", "ner"])

# Function to normalize text
def normalize(text):
    '''
    Normalizes given text by removing HTML tags, URLs, hashtags, emojis,
    punctuations, converting to lowercase, tokenizing the text, removing stop␣
    ↪words, lemmatizing the tokens, and removing non-alphabetic tokens.

    Args:
        text (str): Input text to normalize.

    Returns:
        tokens (list): A list of normalized tokens.
    '''
    # Remove HTML tags
    text = re.sub(r'<.*?>', '', str(text))

    # Remove URLs
    text = re.sub(r'https?://\S+|www\.\S+', '', str(text))

    # Remove hashtags
    text = re.sub(r'#\S+', '', str(text))

    # Remove emojis
    text = ''.join(c for c in str(text) if c in string.printable)

    # Remove punctuation
    text = re.sub(r'[^\w\s]', '', str(text))

    # Convert to lowercase
    text = str(text).lower()
```

```
    # Tokenize and lemmatize text, remove stopwords and non-alphabetic tokens
    tokens = [token.lemma_.lower() for token in nlp(text) if not token.is_stop␣
    ↪and token.is_alpha]

    return tokens
```

6.1 Before normalization

```
[ ]: # Set option to prevent truncation of 'text' column output
     pd.set_option('display.max_colwidth', None)

     # Before normalization
     df_restaurants_ca['text'].head()
```

```
[ ]: # Undo our changed option to avoid affecting the rest of our output
     pd.reset_option('display.max_colwidth')
```

6.2 Applying normalization

```
[ ]: # Took 34 minutes with T4 GPU o.0
     # Apply normalize
     df_restaurants_ca['text'] = df_restaurants_ca['text'].apply(lambda x:␣
      ↪normalize(x))
```

6.3 After normalization

```
[ ]: # Set option to prevent truncation of 'text' column output
     pd.set_option('display.max_colwidth', None)

     # After normalization
     df_restaurants_ca['text'].head()
```

```
[ ]: # Undo our changed option to avoid affecting the rest of our output
     pd.reset_option('display.max_colwidth')
```

```
[ ]: df_restaurants_ca.head()
```

7 Saving normalized dataset to csv file

```
[ ]: # Normalized dataset saved as "normalized_df_restaurants_ca.csv"
     df_restaurants_ca.to_csv('normalized_df_restaurants_ca.csv', index=False)
```

8 EDA

[ ]:
```python
import pandas as pd

data = pd.read_csv('normalized_df_restaurants_ca.csv')
```

[ ]:
```python
# top 10 highest average star rating per city
stars_by_city = data.groupby("city")
stars_by_city["average_stars"].mean().sort_values(ascending=False)[:10]
```

[ ]:
```
city
Truckee          4.077692
Santa Barbara    3.918465
Carpinteria      3.894687
Summerland       3.888293
Montecito        3.873211
Isla Vista       3.842938
Goleta           3.841440
Name: average_stars, dtype: float64
```

We analyzed the average star ratings for each city and identified the top cities with the highest ratings. Truckee led with an average rating of 4.08, followed by Santa Barbara and Carpinteria with ratings of 3.92 and 3.89, respectively. This information helps highlight regions with particularly high customer satisfaction.

[ ]:
```python
# number of reviews by city
data['city'].value_counts()[:10]
```

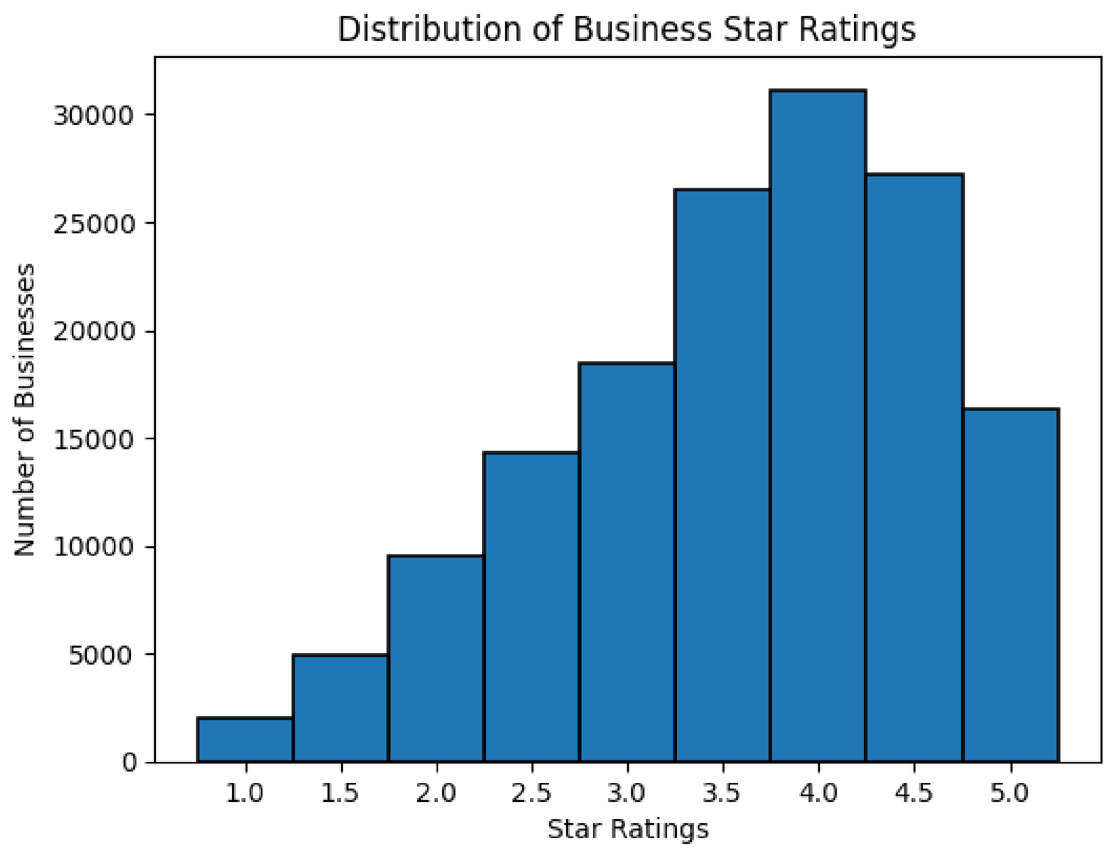
[ ]:
```
city
Santa Barbara    162430
Goleta            26584
Carpinteria       11416
Isla Vista         6330
Montecito          3638
Summerland         1324
Truckee              26
Name: count, dtype: int64
```

We quantified the number of Yelp reviews in California per city and found that Santa Barbara leads with a total of 162,430 reviews, followed by Goleta and Carpinteria with 26,584 and 11,416 reviews, respectively, indicating a higher user engagement in these areas.

[ ]:
```python
import matplotlib.pyplot as plt

plt.bar(df_business["stars"].value_counts().index, df_business["stars"].
 ↪value_counts().values, edgecolor='black', linewidth=1.2, width=0.5,)
plt.title('Distribution of Business Star Ratings')
plt.xlabel('Star Ratings')
plt.ylabel('Number of Businesses')
```

```
plt.xticks(ticks=np.arange(1, 5.5, 0.5))
```

```
[ ]: ([<matplotlib.axis.XTick at 0x783b583169e0>,
       <matplotlib.axis.XTick at 0x783b583169b0>,
       <matplotlib.axis.XTick at 0x783b583168c0>,
       <matplotlib.axis.XTick at 0x783bc91f7e20>,
       <matplotlib.axis.XTick at 0x783bc7f48910>,
       <matplotlib.axis.XTick at 0x783bc7f480a0>,
       <matplotlib.axis.XTick at 0x783bc7f498d0>,
       <matplotlib.axis.XTick at 0x783bc7f4a380>,
       <matplotlib.axis.XTick at 0x783bc7f4ae30>],
      [Text(1.0, 0, '1.0'),
       Text(1.5, 0, '1.5'),
       Text(2.0, 0, '2.0'),
       Text(2.5, 0, '2.5'),
       Text(3.0, 0, '3.0'),
       Text(3.5, 0, '3.5'),
       Text(4.0, 0, '4.0'),
       Text(4.5, 0, '4.5'),
       Text(5.0, 0, '5.0')])
```



We analyzed the distribution of star ratings across businesses and plotted the results, showing that most businesses tend to cluster around the 3.5 to 4.0 star range, indicating a generally favorable reception from customers.

[ ]:
```python
import pandas as pd
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Simple text cleaning
data['text'] = data['text'].str.lower().str.replace(r'[^\w\s]', '', regex=True)

# Remove specific unwanted words
unwanted_words = ['santa', 'barbara']
data['text'] = data['text'].apply(lambda x: ' '.join(word for word in x.split() if word not in unwanted_words))

# Tokenization might already be handled by word cloud but if you need specific processing do it here

# Create a single string for the word cloud
all_reviews = " ".join(review for review in data['text'])

# Generate word cloud
wordcloud = WordCloud(width=800, height=400, background_color='white', colormap="coolwarm").generate(all_reviews)

# Display the generated image:
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")  # Turn off axis numbers and ticks
plt.show()
```

We processed Yelp review text by removing common words, converting all text to lowercase, and excluding specific unwanted terms. Subsequently, we created a word cloud to visually represent the most frequent terms used in the reviews, highlighting key words such as 'food', 'service', and 'love'.

8.1 Sentiment Analysis

8.1.1 Top 20 Positive/Negative Words

[ ]:
```
from collections import defaultdict
import pandas as pd
import matplotlib.pyplot as plt
from wordcloud import WordCloud
import re

class SentimentAnalyzer:
    """
    Class that performs sentiment analysis on a dataset of reviews.

    This class provides methods to clean and filter text, analyze word␣
↪frequencies
    in positive and negative reviews, calculate sentiment scores, and generate␣
↪word clouds.

    Attributes:
    data (pd.DataFrame): The input DataFrame containing reviews with␣
↪'stars_review' and 'text' columns.
    exclude_words (set): A set of words to exclude from analysis.
    positive_reviews (pd.DataFrame): DataFrame containing reviews with more␣
↪than 2 stars.
    negative_reviews (pd.DataFrame): DataFrame containing reviews with 2 or␣
↪fewer stars.
    positive_word_counts (defaultdict): Counts of words in positive reviews.
    negative_word_counts (defaultdict): Counts of words in negative reviews.
    word_sentiment_scores (dict): Sentiment scores for words based on their␣
↪frequency in positive and negative reviews.

    Methods:
    clean_and_filter(text): Cleans and filters the input text by removing␣
↪punctuation, converting to lowercase, and excluding specific words.
    analyze_reviews(): Analyzes the reviews to count the frequency of words in␣
↪positive and negative reviews.
    calculate_sentiment_scores(): Calculates sentiment scores for each word␣
↪based on their counts in positive and negative reviews.
    get_sorted_words(): Returns a list of tuples with words and their sentiment␣
↪scores, sorted in descending order.
```

```
    display_top_words(n=20): Displays the top N positive and negative words␣
↪based on sentiment scores.
    generate_word_cloud(positive=True, width=800, height=400): Generates a word␣
↪cloud for positive or negative sentiment words.
    display_word_cloud(positive=True, width=800, height=400): Displays a word␣
↪cloud for positive or negative sentiment words.
    """

    def __init__(self, data, exclude_words):
        """
        Initialize the SentimentAnalyzer with data and words to exclude.

        Parameters:
        data (pd.DataFrame): The input DataFrame containing reviews.
        exclude_words (set): A set of words to exclude from analysis.
        """
        self.data = data
        self.exclude_words = exclude_words
        self.positive_reviews = data[data['stars_review'] > 2]
        self.negative_reviews = data[data['stars_review'] <= 2]
        self.positive_word_counts = defaultdict(int)
        self.negative_word_counts = defaultdict(int)
        self.word_sentiment_scores = {}

    def clean_and_filter(self, text):
        """
        Clean and filter text by removing punctuation, converting to lowercase,␣
↪and excluding specific words.

        Parameters:
        text (str): The input text to be cleaned and filtered.

        Returns:
        list: A list of cleaned and filtered words.
        """
        words = re.sub(r'[^\w\s]', '', text.lower()).split()
        return [word for word in words if word not in self.exclude_words]

    def analyze_reviews(self):
        """
        Analyze the reviews to count the frequency of words in positive and␣
↪negative reviews.

        Returns:
        None
        """
        for review in self.positive_reviews['text']:
```

```
            for word in self.clean_and_filter(review):
                self.positive_word_counts[word] += 1

        for review in self.negative_reviews['text']:
            for word in self.clean_and_filter(review):
                self.negative_word_counts[word] += 1

    def calculate_sentiment_scores(self):
        """
        Calculate sentiment scores for each word based on their counts in␣
↪positive and negative reviews.

        Returns:
        None
        """
        for word in set(self.positive_word_counts.keys()).union(self.
↪negative_word_counts.keys()):
            positive_count = self.positive_word_counts.get(word, 0)
            negative_count = self.negative_word_counts.get(word, 0)
            sentiment_score = positive_count - negative_count
            self.word_sentiment_scores[word] = sentiment_score

    def get_sorted_words(self):
        """
        Get words sorted by their sentiment scores.

        Returns:
        list: A list of tuples with words and their sentiment scores, sorted in␣
↪descending order.
        """
        return sorted(self.word_sentiment_scores.items(), key=lambda x: x[1],␣
↪reverse=True)

    def display_top_words(self, n=20):
      """
      Display the top N positive and negative words based on sentiment scores.

      Parameters:
      n (int): The number of top words to display for both positive and␣
↪negative sentiment.

      Returns:
      None
      """
      sorted_words = self.get_sorted_words()

      # Top N Positive Words
```

```python
        top_positive_words = [word for word, score in sorted_words[:n]]

        # Top N Negative Words (reverse the lowest N for most negative)
        top_negative_words = [word for word, score in sorted_words[-n:]][::-1]

        print("Top {} Positive Words:".format(n))
        for word in top_positive_words:
            print(word)

        print("\nTop {} Negative Words:".format(n))
        for word in top_negative_words:
            print(word)


    def generate_word_cloud(self, positive=True, width=800, height=400):
        """
        Generate a word cloud for positive or negative sentiment words.

        Parameters:
        positive (bool): Whether to generate a word cloud for positive␣
↪sentiment words.
        width (int): The width of the word cloud image.
        height (int): The height of the word cloud image.

        Returns:
        WordCloud: The generated WordCloud object.
        """
        if positive:
            word_frequencies = self.word_sentiment_scores
        else:
            word_frequencies = {word: -score for word, score in self.
↪word_sentiment_scores.items()}

        wordcloud = WordCloud(width=width, height=height,␣
↪background_color="white", colormap="coolwarm").
↪generate_from_frequencies(word_frequencies)
        return wordcloud

    def display_word_cloud(self, positive=True, width=800, height=400):
        """
        Display a word cloud for positive or negative sentiment words.

        Parameters:
        positive (bool): Whether to display a word cloud for positive sentiment␣
↪words.
        width (int): The width of the word cloud image.
        height (int): The height of the word cloud image.
```

```
        Returns:
        None
        """
        wordcloud = self.generate_word_cloud(positive, width, height)
        plt.figure(figsize=(width // 100, height // 100))
        plt.imshow(wordcloud, interpolation="bilinear")
        plt.axis("off")
        plt.show()


exclude_words = {'santa', 'barbara', 'goleta', 'carpinteria', 'isla', 'vista',␣
 ↪'montecito', 'summerland', 'truckee'}
analyzer = SentimentAnalyzer(data, exclude_words)
analyzer.analyze_reviews()
analyzer.calculate_sentiment_scores()
analyzer.display_top_words(20)
analyzer.display_word_cloud(positive=True)  # For positive sentiment
print("")
analyzer.display_word_cloud(positive=False) # For negative sentiment
```

```
Top 20 Positive Words:
good
great
food
place
not
delicious
service
love
come
time
try
like
order
amazing
nice
restaurant
definitely
fresh
friendly
get

Top 20 Negative Words:
bad
rude
terrible
horrible
```

manager
disappointed
poor
awful
charge
waste
money
tell
minute
say
mediocre
disappointing
disgusting
tasteless
refund
gross

We refined our dataset by filtering out specific local names from the reviews to focus on more general sentiment words. Positive and negative reviews were then separately processed to count the occurrence of each word, excluding pre-defined irrelevant terms. After calculating sentiment scores by subtracting negative mention counts from positive ones for each word, we ranked and listed the top 20 words associated with positive and negative sentiments. The analysis revealed that words like ‘good’, ‘great’, and ‘delicious’ dominated positive reviews, whereas terms such as ‘gross’, ‘refund’, and ‘disappointing’ were prevalent in negative reviews, highlighting distinct elements that impact customer satisfaction and dissatisfaction.

8.2 Feature Selection

[ ]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
import nltk

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



# preprocess text
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

def preprocess_text(text):
    """
```

```
    Preprocess the input text by converting to lowercase, removing punctuation,
    tokenizing, lemmatizing, and excluding stopwords.

    Parameters:
    text (str): The input text to be preprocessed.

    Returns:
    str: The cleaned and preprocessed text.
    """
    text = text.lower()
    text = text.translate(str.maketrans('', '', string.punctuation))
    words = word_tokenize(text)
    words = [lemmatizer.lemmatize(word) for word in words if word not in␣
 ↪stop_words]
    return ' '.join(words)

texts_preprocessed = data['text'].apply(preprocess_text)
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data…
```

We import necessary libraries for text processing and then define a function to preprocess text data from Yelp reviews. This function converts text to lowercase, removes punctuation, tokenizes the text into words, excludes common stop words, and applies lemmatization to reduce words to their base form. The cleaned text is then used to further analyze features or train machine learning models.

8.2.1 TF-IDF Vectorization

```
[ ]: # Define TF-IDF vectorizer with review frequency thresholds
vectorizer = TfidfVectorizer(min_df=0.1, max_df=0.7)

# Fit and transform the preprocessed texts
X_tfidf = vectorizer.fit_transform(texts_preprocessed)

# Convert to DataFrame
X_tfidf_df = pd.DataFrame(X_tfidf.toarray(), columns=vectorizer.
 ↪get_feature_names_out())
```

TF-IDF vectorization converts text data into numerical features that machine learning models can use. Each word in the text is represented as a numerical value based on its importance within each review and across all reviews. This transformation results in a feature matrix where each row corresponds to a review and each column corresponds to a unique word or n-gram.

Words that are important for predicting star ratings, such as "delicious," "friendly," or "disappointing," will have higher TF-IDF scores in reviews where they are significant, whereas common words that appear in many reviews but do not contribute to distinguishing the star rating (e.g., "the", "and") get lower TF-IDF scores, effectively reducing their impact.

Words with high TF-IDF scores in positive reviews ("amazing", "excellent") versus negative reviews ("awful", "poor") help the model learn associations between words and star ratings.

8.2.2 Apply PCA

[ ]:
```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Initialize PCA model, specify the number of components
n_components = 2  # Number of principal components to keep
pca = PCA(n_components=n_components)

# Fit PCA model and transform the TF-IDF data
X_pca = pca.fit_transform(X_tfidf_df)

# Create a DataFrame for the PCA results
X_pca_df = pd.DataFrame(X_pca, columns=[f'PC{i+1}' for i in␣
↪range(n_components)])

print("PCA Results:")
print(X_pca_df)
```

```
PCA Results:
             PC1       PC2
0      -0.191868  0.025744
1       0.151012  0.017874
2       0.400117 -0.142071
3       0.149820  0.154455
4      -0.089400  0.012618
…            …         …
211743 -0.136705  0.258626
211744  0.547174 -0.078931
211745  0.125933 -0.009156
211746 -0.037699 -0.036948
211747  0.061949 -0.008350

[211748 rows x 2 columns]
```

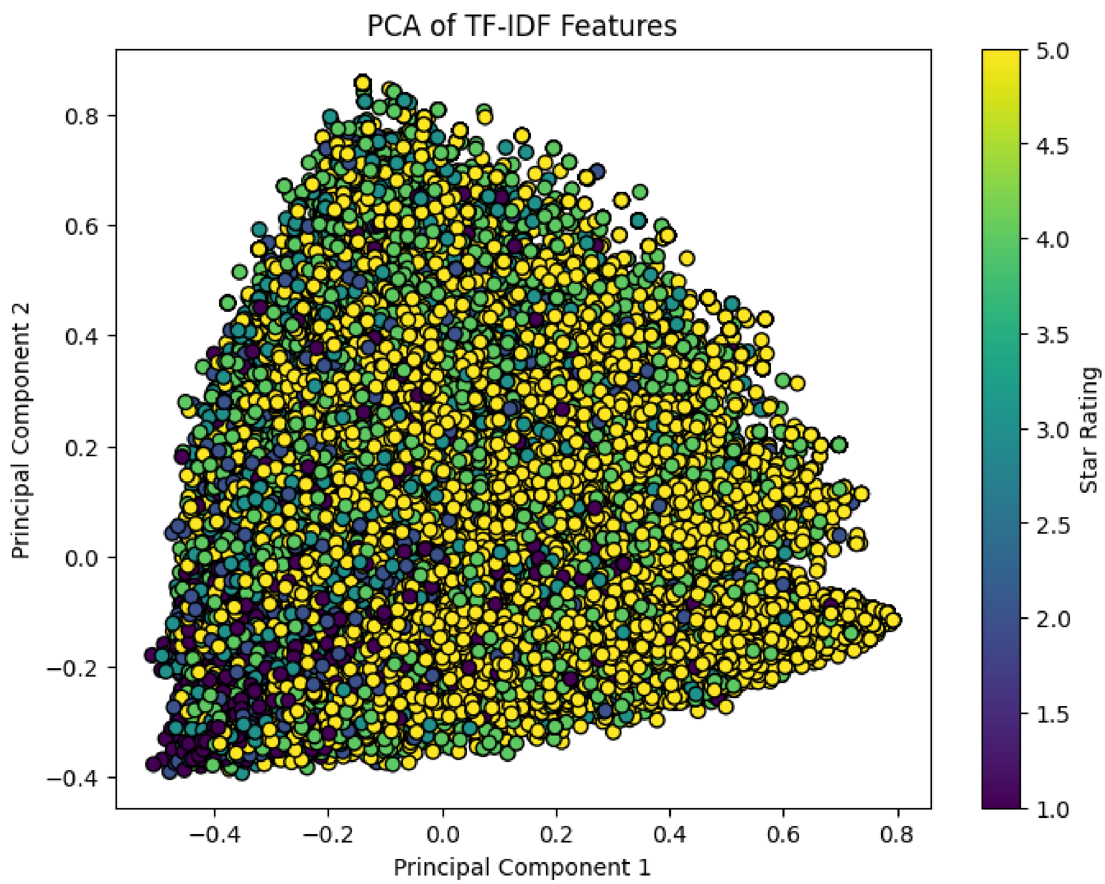
We want to apply PCA tests to reduce the number of TF-IDF features, focusing on the words that contribute most to the model's performance.

PCA projects the high-dimensional TF-IDF space into a lower-dimensional space, retaining the components that explain the most variance.

We use the explained variance ratio to decide how many components to keep. This balances between reducing dimensionality and retaining enough information.

[ ]:
```
# Plot the PCA results
plt.figure(figsize=(8, 6))
plt.scatter(X_pca_df['PC1'], X_pca_df['PC2'], c=data['stars_review'],␣
↪cmap='viridis', edgecolor='k', s=40)
plt.xlabel('Principal Component 1')
```

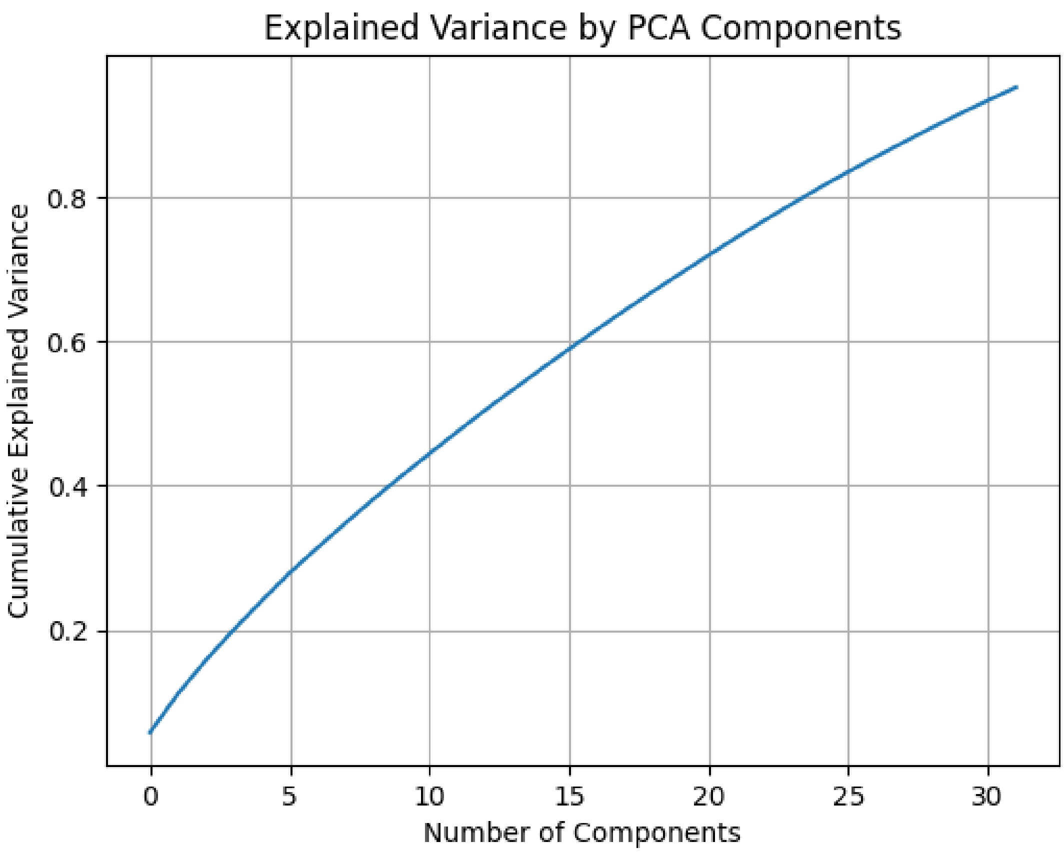
```
plt.ylabel('Principal Component 2')
plt.title('PCA of TF-IDF Features')
plt.colorbar(label='Star Rating')
plt.show()
```



[ ]:
```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Determine the number of components to retain 95% variance
pca = PCA(n_components=0.95)  # Adjust as needed to retain desired variance
X_pca = pca.fit_transform(X_tfidf.toarray())  # Use .toarray() to convert␣
↪sparse matrix to dense

# Explained Variance
explained_variance = pca.explained_variance_ratio_.sum()
print(f'Explained variance by the components: {explained_variance:.4f}')

# Plot the cumulative explained variance to decide on the number of components
cumulative_variance = pca.explained_variance_ratio_.cumsum()
```

```
plt.plot(cumulative_variance)
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Explained Variance by PCA Components')
plt.grid(True)
plt.show()
```

```
Explained variance by the components: 0.9523
```



Based on our PCA, we are able to reduce our feature size to 30 components while still retaining 95% variance. Thus, we test this by integrating PCA on some classification models.

8.2.3 Integrate PCA on Classification Models

[ ]:
```
# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_pca,␣
↪data['stars_review'], test_size=0.2, random_state=42)

X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
```

8.2.4 Random Forest

[ ]:
```python
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Model Training
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train_pca, y_train)

# Prediction
y_pred = model.predict(X_test_pca)

# Evaluation
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.4f}')
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.5266
              precision    recall  f1-score   support

         1.0       0.42      0.43      0.43      4251
         2.0       0.24      0.06      0.10      3276
         3.0       0.22      0.05      0.09      4311
         4.0       0.32      0.16      0.21      9388
         5.0       0.59      0.88      0.70     21124

    accuracy                           0.53     42350
   macro avg       0.36      0.32      0.31     42350
weighted avg       0.45      0.53      0.46     42350
```

For our random forest model, we see that it achieved an overall accuracy of 52.66% in predicting star ratings for Yelp reviews, which indicates that it correctly classified the star ratings just over half the time. The model demonstrates solid precision and recall for 5-star reviews, meaning it is good at identifying highly positive reviews but struggles significantly with other ratings, especially 2-star and 3-star reviews, where precision and recall are notably low. This suggests that while the model is effective at distinguishing very positive feedback, it lacks sensitivity and accuracy in differentiating between negative, neutral, and moderately positive reviews.

8.2.5 K Neighbors

[ ]:
```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
import random

# Set a random seed for reproducibility
random.seed(42)
```

```python
k_range = range(1, 21)
cv_scores = []
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train_pca, y_train, cv=5,␣
 ↪scoring='accuracy')  # 5-fold cross-validation
    cv_scores.append(scores.mean())

# Print the mean cross-validation accuracy for each k
for k, cv_score in zip(k_range, cv_scores):
    print(f'Mean CV accuracy for k = {k}: {cv_score:.10f}')

# Find the optimal k based on the highest mean cross-validation accuracy
optimal_k = k_range[cv_scores.index(max(cv_scores))]
print("The optimal number of neighbors is:", optimal_k)

# Fit the model with the optimal k on training dataset
optimal_knn = KNeighborsClassifier(n_neighbors=optimal_k)
optimal_knn.fit(X_train_pca, y_train)

# Predict the labels on the test set
y_pred = optimal_knn.predict(X_test)

# Evaluate the accuracy on the test set for the optimal k
test_accuracy = accuracy_score(y_test, y_pred)
print(f'Test accuracy for the optimal k ({optimal_k}) is {test_accuracy:.5f}')

# Print Classification Report
print("KNN Classification Report")
print(classification_report(y_test, y_pred))
```

```
Mean CV accuracy for k = 1: 0.3689771796
Mean CV accuracy for k = 2: 0.2881379861
Mean CV accuracy for k = 3: 0.3664269984
Mean CV accuracy for k = 4: 0.3886232281
Mean CV accuracy for k = 5: 0.4039008722
Mean CV accuracy for k = 6: 0.4097687039
Mean CV accuracy for k = 7: 0.4245976908
Mean CV accuracy for k = 8: 0.4334053598
Mean CV accuracy for k = 9: 0.4401114345
Mean CV accuracy for k = 10: 0.4458671174
Mean CV accuracy for k = 11: 0.4513984822
Mean CV accuracy for k = 12: 0.4554067795
Mean CV accuracy for k = 13: 0.4591671508
Mean CV accuracy for k = 14: 0.4630396935
Mean CV accuracy for k = 15: 0.4658968718
Mean CV accuracy for k = 16: 0.4685179286
Mean CV accuracy for k = 17: 0.4702121732
```

```
Mean CV accuracy for k = 18: 0.4718296716
Mean CV accuracy for k = 19: 0.4736950941
Mean CV accuracy for k = 20: 0.4748521380
The optimal number of neighbors is: 20
Test accuracy for the optimal k (20) is 0.47762
KNN Classification Report
              precision    recall  f1-score   support

         1.0       0.29      0.21      0.24      4251
         2.0       0.13      0.03      0.04      3276
         3.0       0.17      0.03      0.06      4311
         4.0       0.25      0.11      0.16      9388
         5.0       0.54      0.85      0.66     21124

    accuracy                           0.48     42350
   macro avg       0.28      0.25      0.23     42350
weighted avg       0.38      0.48      0.40     42350
```

The K-Nearest Neighbors (KNN) model with the optimal value achieved an overall test accuracy of 47.76%, indicating that it correctly predicted the star ratings for approximately half of the Yelp reviews. While the model performs well for 5-star reviews with a precision of 54% and a high recall of 85%, it struggles significantly with lower star ratings, showing very low precision and recall for 2-star and 3-star reviews. The disparity in performance across different star ratings highlights the model's limitation in effectively distinguishing between reviews of varying quality.

After conducting PCA using TF-IDF Vecorization, we see that the model accuracy is subpar even after finding the optimal hyperparameters. Furthermore, PCA reduces our dataset down *too* much, as it says that the ideal number of components is 30 components (ie 30 words), which would not make much sense in context of our project. Thus, we decide to move forward without using PCA and TF-IDF Vectorization.

9 Models

10 Long Short-Term Memory (LSTM) network on Normalized data

We first developed a Long Short-Term Memory (LSTM) network to predict the star ratings of Yelp reviews based on their textual content. We chose LSTM for its proficiency in handling sequence data, which makes it ideal for text processing where the order of words is critical for understanding sentiment and meaning.

[ ]:
```
import pandas as pd
data = pd.read_csv("normalized_df_restaurants_ca.csv")
data.head()
```

[ ]:
```
   Unnamed: 0                 user_id             business_id  stars_review  \
0           9  59MxRhNVhU9MYndMkz0wtw  gebiRewfieSdtt17PTW6Zg           3.0
```

```
1          23  OhECKhQEexFypOMY6kypRw  vC2qm1y3Au5czBtbhc-DNw          4.0
2          31  4hBhtCSgoxkrFgHa4YAD-w  bbEXAEFr4RYHLlZ-HFssTA          5.0
3          35  bFPdtzu11Oi0f92EAcjqmg  IDtLPgUrqorrpqSLdfMhZQ          5.0
4          61  JYYYKt6TdVA4ng9lLcXt_g  SZU9c8V2GuREDN5KgyHFJw          5.0

                                                text  average_stars  \
0  ['party', 'hibachi', 'waitress', 'bring', 'sep…           4.23
1  ['yes', 'sushi', 'place', 'town', 'great', 'cr…           3.96
2  ['great', 'burgersfrie', 'salad', 'burger', 'h…           4.20
3  ['great', 'addition', 'funk', 'zone', 'grab', …           4.06
4  ['bit', 'weary', 'try', 'shellfish', 'company'…           4.12

                               name              address           city  \
0  Hibachi Steak House & Sushi Bar           502 State St  Santa Barbara
1                       Sushi Teri         970 Linden Ave    Carpinteria
2  The Original Habit Burger Grill     5735 Hollister Ave         Goleta
3             Helena Avenue Bakery  131 Anacapa St, Ste C  Santa Barbara
4  Santa Barbara Shellfish Company      230 Stearns Wharf  Santa Barbara

  state  postal_code   latitude   longitude  stars_business  review_count  \
0    CA        93101  34.416984 -119.695556             3.5         488.0
1    CA        93013  34.398527 -119.518475             3.0         167.0
2    CA        93117  34.435570 -119.824706             4.0         329.0
3    CA        93101  34.414445 -119.690672             4.0         389.0
4    CA        93101  34.408715 -119.685019             4.0        2404.0

   is_open                                         attributes  \
0      1.0  {'Corkage': 'False', 'RestaurantsTakeOut': 'Tr…
1      1.0  {'RestaurantsReservations': 'True', 'NoiseLeve…
2      1.0  {'Caters': 'False', 'GoodForKids': 'True', 'BY…
3      1.0  {'RestaurantsTakeOut': 'True', 'NoiseLevel': "…
4      1.0  {'OutdoorSeating': 'True', 'RestaurantsAttire'…

                                        categories  \
0    Steakhouses, Sushi Bars, Restaurants, Japanese
1                          Restaurants, Sushi Bars
2                    Fast Food, Burgers, Restaurants
3  Food, Restaurants, Salad, Coffee & Tea, Breakf…
4  Live/Raw Food, Restaurants, Seafood, Beer Bar,…

                                             hours
0                             {'Monday': '0:0-0:0'}
1  {'Monday': '17:0-22:0', 'Tuesday': '17:0-22:0'…
2  {'Monday': '0:0-0:0', 'Tuesday': '10:30-21:0',…
3  {'Monday': '0:0-0:0', 'Tuesday': '8:0-14:0', '…
4  {'Monday': '0:0-0:0', 'Tuesday': '11:0-21:0', …
```

```python
import pandas as pd
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Load data
data = pd.read_csv("normalized_df_restaurants_ca.csv")

# Simple text cleaning
data['text'] = data['text'].str.lower().str.replace(r'[^\w\s]', '')

# Tokenization
tokenizer = Tokenizer(num_words=10000, oov_token='<UNK>')
tokenizer.fit_on_texts(data['text'])
sequences = tokenizer.texts_to_sequences(data['text'])

# Padding
padded_sequences = pad_sequences(sequences, maxlen=100)
```

Performed preprocessing steps on Yelp review text data, including loading the data, cleaning by converting to lowercase and removing punctuation, tokenizing the text into sequences using a tokenizer with a vocabulary size of 10,000 words, and then padding these sequences to a uniform length of 100 for use in TensorFlow neural network models.

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout

model = Sequential([
    Embedding(input_dim=10000, output_dim=64, input_length=100),
    LSTM(64, return_sequences=True),
    LSTM(32),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='linear')  # Linear activation for regression
])

model.compile(optimizer='adam', loss='mean_squared_error',
 ↪metrics=['mean_squared_error'])
model.summary()
```

```
WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't meet
the criteria. It will use a generic GPU kernel as fallback when running on GPU.
WARNING:tensorflow:Layer lstm_1 will not use cuDNN kernels since it doesn't meet
the criteria. It will use a generic GPU kernel as fallback when running on GPU.
```

```
Model: "sequential_1"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 embedding_1 (Embedding)     (None, 100, 64)           640000

 lstm (LSTM)                 (None, 100, 64)           33024

 lstm_1 (LSTM)               (None, 32)                12416

 dense_2 (Dense)             (None, 64)                2112

 dropout_1 (Dropout)         (None, 64)                0

 dense_3 (Dense)             (None, 1)                 65

=================================================================
Total params: 687617 (2.62 MB)
Trainable params: 687617 (2.62 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

The model is composed of several layers: an initial embedding layer that translates words into a 64-dimensional vector space to capture semantic similarities between words. This is followed by two LSTM layers; the first with 64 units to capture contextual dependencies across the text sequence, and the second with 32 units to condense this information, summarizing the critical features of the review. A dropout layer with a 50% rate is integrated to mitigate overfitting by randomly deactivating parts of the neural network during training, enhancing the model's ability to generalize better to new, unseen data. Additionally, the network includes a dense layer with 64 neurons and ReLU activation to introduce non-linearity, concluding with a single-unit linear activation layer that outputs the predicted star rating. The model, incorporating 687,617 parameters, uses the mean squared error (MSE) as the loss function to quantify the average squared discrepancy between the actual and predicted ratings

[ ]:
```python
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(padded_sequences,
 ↪data['stars_review'], test_size=0.2, random_state=42)
```

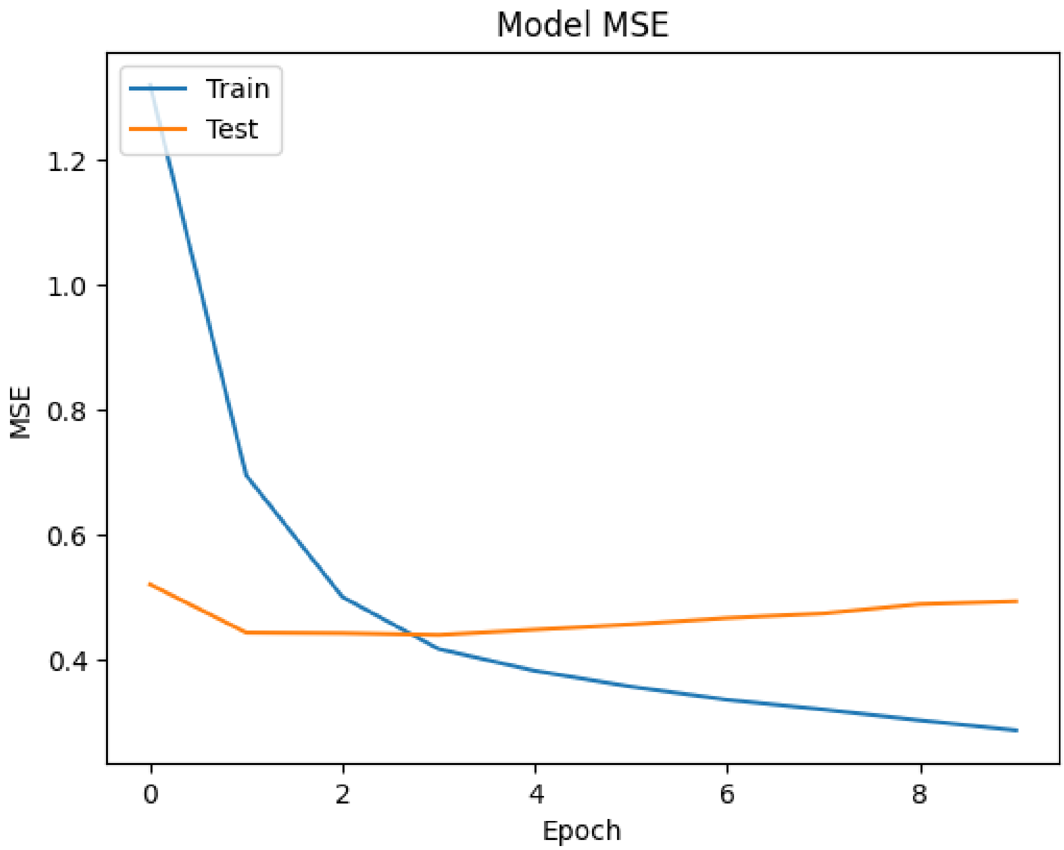
[ ]:
```python
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(padded_sequences,
 ↪data['stars_review'], test_size=0.2, random_state=42)

history = model.fit(X_train, y_train, epochs=10, validation_data=(X_test,
 ↪y_test), batch_size=64)
```

```
Epoch 1/10
2647/2647 [==============================] - 82s 29ms/step - loss: 1.3211 -
mean_squared_error: 1.3211 - val_loss: 0.5209 - val_mean_squared_error: 0.5209
Epoch 2/10
2647/2647 [==============================] - 41s 16ms/step - loss: 0.6954 -
mean_squared_error: 0.6954 - val_loss: 0.4437 - val_mean_squared_error: 0.4437
```

```
Epoch 3/10
2647/2647 [==============================] - 41s 15ms/step - loss: 0.5007 -
mean_squared_error: 0.5007 - val_loss: 0.4428 - val_mean_squared_error: 0.4428
Epoch 4/10
2647/2647 [==============================] - 38s 14ms/step - loss: 0.4174 -
mean_squared_error: 0.4174 - val_loss: 0.4401 - val_mean_squared_error: 0.4401
Epoch 5/10
2647/2647 [==============================] - 40s 15ms/step - loss: 0.3824 -
mean_squared_error: 0.3824 - val_loss: 0.4485 - val_mean_squared_error: 0.4485
Epoch 6/10
2647/2647 [==============================] - 40s 15ms/step - loss: 0.3571 -
mean_squared_error: 0.3571 - val_loss: 0.4567 - val_mean_squared_error: 0.4567
Epoch 7/10
2647/2647 [==============================] - 40s 15ms/step - loss: 0.3360 -
mean_squared_error: 0.3360 - val_loss: 0.4669 - val_mean_squared_error: 0.4669
Epoch 8/10
2647/2647 [==============================] - 38s 14ms/step - loss: 0.3204 -
mean_squared_error: 0.3204 - val_loss: 0.4743 - val_mean_squared_error: 0.4743
Epoch 9/10
2647/2647 [==============================] - 39s 15ms/step - loss: 0.3032 -
mean_squared_error: 0.3032 - val_loss: 0.4892 - val_mean_squared_error: 0.4892
Epoch 10/10
2647/2647 [==============================] - 37s 14ms/step - loss: 0.2872 -
mean_squared_error: 0.2872 - val_loss: 0.4938 - val_mean_squared_error: 0.4938
```

[ ]:

```
import matplotlib.pyplot as plt

# Plot training & validation loss values
plt.plot(history.history['mean_squared_error'])
plt.plot(history.history['val_mean_squared_error'])
plt.title('Model MSE')
plt.ylabel('MSE')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



The model was trained and evaluated over 10 epochs, and while it showed promising accuracy on the training dataset, the performance on the validation set depicted signs of overfitting. This was evident from the training and validation loss plot where the training loss consistently decreased, indicating the model was learning effectively, but the validation loss started to plateau and then diverged from the training loss. This divergence suggests that while the model was becoming increasingly precise on the training data, it was not performing equivalently on new, unseen data.

11 Long Short-Term Memory (LSTM) network with Early Stopping and Dropout Layers

Next, we developed a Long Short-Term Memory (LSTM) network with early stopping implemented, as well as dropout layers to predict Yelp review ratings, capitalizing on the LSTM's capability to effectively handle sequential text data.

Our model features an embedding layer that maps words into a 64-dimensional vector space from a vocabulary of 10,000 words, helping to reduce dimensionality and capture semantic relationships. It includes two LSTM layers, with the first layer containing 64 units to process sequences for contextual dependencies, and the second layer with 32 units to condense this information. To combat overfitting, we incorporated multiple dropout layers with a rate of 0.5 after each LSTM layer and before the final output layer, which randomly deactivates certain neurons during training.

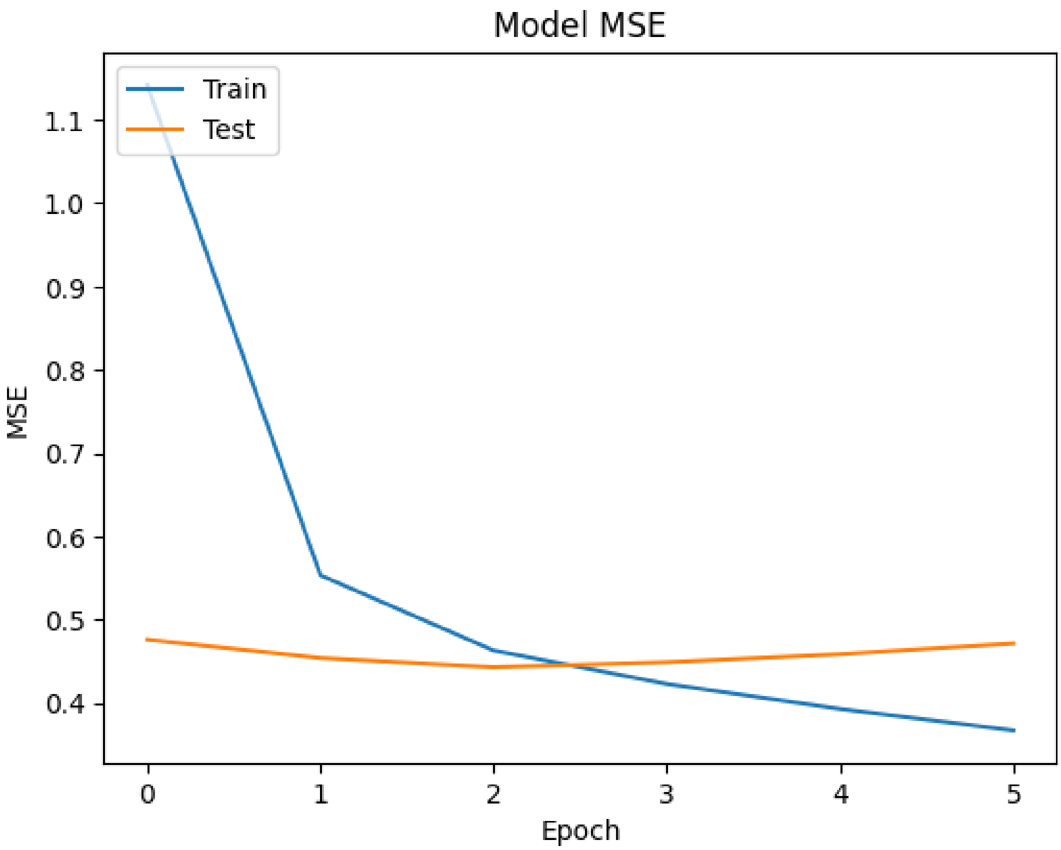
This strategy prevents the model from depending too heavily on specific features of the data. The architecture is completed with a dense layer of 64 neurons with ReLU activation for learning complex patterns and a single neuron linear activation output layer for predicting star ratings. The model has a total of 687,617 parameters.

```
[ ]: from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, Embedding
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split

# Define the model
model = Sequential([
    Embedding(input_dim=10000, output_dim=64, input_length=100),
    LSTM(64, return_sequences=True),
    Dropout(0.5),  # Dropout layer after LSTM
    LSTM(32),
    Dropout(0.5),  # Another Dropout layer
    Dense(64, activation='relu'),
    Dropout(0.5),  # Dropout before the output layer
    Dense(1, activation='linear')
])

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error',␣
 ↪metrics=['mean_squared_error'])

# Early stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=3, verbose=1)

X_train, X_test, y_train, y_test = train_test_split(padded_sequences,␣
 ↪data['stars_review'], test_size=0.2, random_state=42)

# Fit the model
history = model.fit(
    X_train, y_train,
    epochs=50,  # Larger number of epochs
    batch_size=32,
    validation_data=(X_test, y_test),
    callbacks=[early_stopping]  # Include early stopping
)
```

```
Epoch 1/50
5294/5294 [==============================] - 154s 28ms/step - loss: 1.1420 -
mean_squared_error: 1.1420 - val_loss: 0.4758 - val_mean_squared_error: 0.4758
Epoch 2/50
5294/5294 [==============================] - 75s 14ms/step - loss: 0.5531 -
mean_squared_error: 0.5531 - val_loss: 0.4538 - val_mean_squared_error: 0.4538
Epoch 3/50
```

```
5294/5294 [==============================] - 71s 13ms/step - loss: 0.4628 -
mean_squared_error: 0.4628 - val_loss: 0.4427 - val_mean_squared_error: 0.4427
Epoch 4/50
5294/5294 [==============================] - 72s 14ms/step - loss: 0.4226 -
mean_squared_error: 0.4226 - val_loss: 0.4487 - val_mean_squared_error: 0.4487
Epoch 5/50
5294/5294 [==============================] - 74s 14ms/step - loss: 0.3926 -
mean_squared_error: 0.3926 - val_loss: 0.4583 - val_mean_squared_error: 0.4583
Epoch 6/50
5294/5294 [==============================] - 71s 13ms/step - loss: 0.3668 -
mean_squared_error: 0.3668 - val_loss: 0.4712 - val_mean_squared_error: 0.4712
Epoch 6: early stopping
```

[ ]:
```
import matplotlib.pyplot as plt

# Plot training & validation loss values
plt.plot(history.history['mean_squared_error'])
plt.plot(history.history['val_mean_squared_error'])
plt.title('Model MSE')
plt.ylabel('MSE')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



We implemented an Early Stopping callback that halts training if the validation loss does not improve after three epochs, effectively preventing over-training. Training was planned for 50 epochs, but early stopping terminated it at the 6th epoch when no further improvement in validation loss was observed.

12 Long Short-Term Memory (LSTM) network with L2 Regularization

To address overfitting in our LSTM model for Yelp review ratings, we incorporated L2 regularization, which adds a penalty to the loss function based on the squared values of the model parameters. This method helps in smoothing the learned parameter values, thus discouraging the model from fitting too tightly to the training data and promoting better generalization on unseen data.

```
[ ]: from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout
from tensorflow.keras.regularizers import l2

model = Sequential([
    Embedding(input_dim=10000, output_dim=64, input_length=100),
    LSTM(64, return_sequences=True, kernel_regularizer=l2(0.01)),  # Adding L2␣
 ↪regularization
    LSTM(32, kernel_regularizer=l2(0.01)),  # Adding L2 regularization
    Dense(64, activation='relu', kernel_regularizer=l2(0.01)),  # Adding L2␣
 ↪regularization
    Dropout(0.5),
    Dense(1, activation='linear')  # Output layer for regression
])

model.compile(optimizer='adam', loss='mean_squared_error',␣
 ↪metrics=['mean_squared_error'])
model.summary()
```

```
Model: "sequential_1"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 100, 64)           640000

 lstm_2 (LSTM)               (None, 100, 64)           33024

 lstm_3 (LSTM)               (None, 32)                12416

 dense_2 (Dense)             (None, 64)                2112

 dropout_3 (Dropout)         (None, 64)                0
```

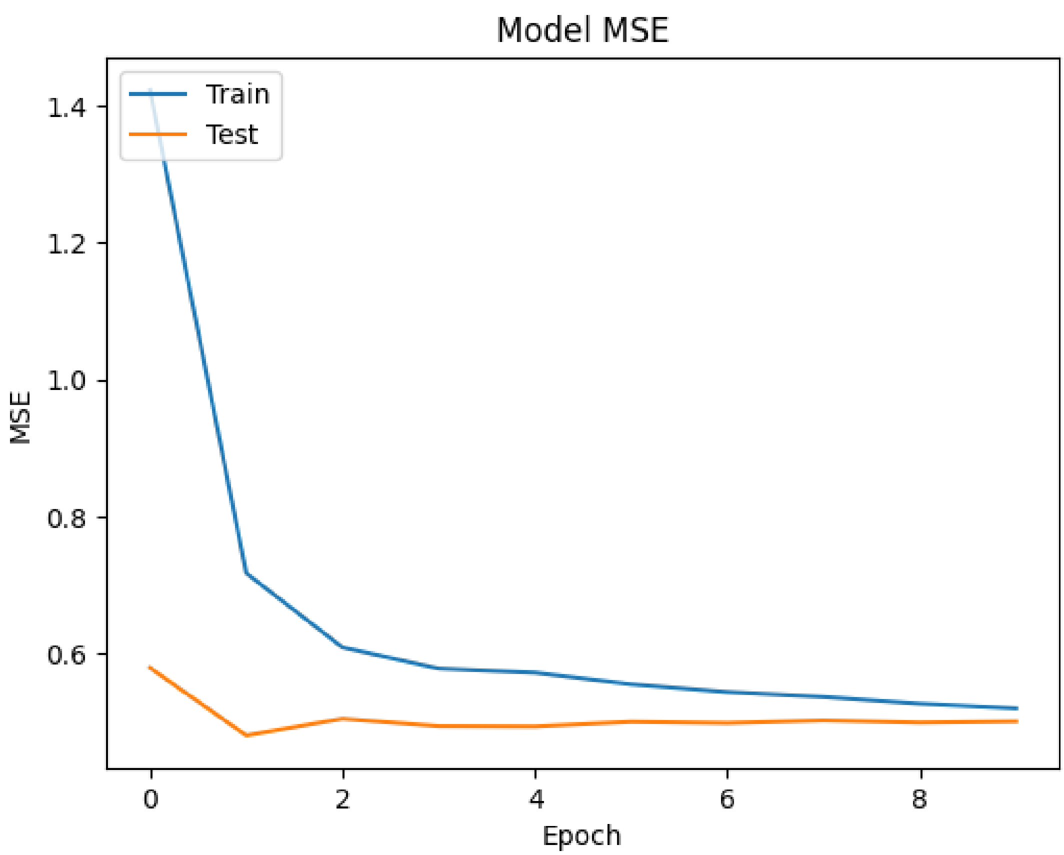
```
 dense_3 (Dense)             (None, 1)                 65

=================================================================
Total params: 687617 (2.62 MB)
Trainable params: 687617 (2.62 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Our LSTM model is structured with an Embedding layer, two LSTM layers with L2 regularization, a Dense layer also utilizing L2 regularization, followed by a Dropout layer to further mitigate overfitting, and concludes with a Dense output layer. We applied a regularization value of 0.01 to the LSTM and Dense layers. The model comprises a total of 687,617 trainable parameters, providing a substantial learning capacity while maintaining a balance to avoid overfitting thanks to the regularization techniques employed.

[ ]:
```
history = model.fit(X_train, y_train, epochs=10, validation_data=(X_test,␣
↪y_test), batch_size=64)
```

```
Epoch 1/10
2647/2647 [==============================] - 114s 41ms/step - loss: 1.7804 -
mean_squared_error: 1.5431 - val_loss: 0.6325 - val_mean_squared_error: 0.5688
Epoch 2/10
2647/2647 [==============================] - 44s 17ms/step - loss: 0.7479 -
mean_squared_error: 0.7033 - val_loss: 0.5158 - val_mean_squared_error: 0.4839
Epoch 3/10
2647/2647 [==============================] - 41s 15ms/step - loss: 0.6251 -
mean_squared_error: 0.5956 - val_loss: 0.5436 - val_mean_squared_error: 0.5074
Epoch 4/10
2647/2647 [==============================] - 40s 15ms/step - loss: 0.6000 -
mean_squared_error: 0.5732 - val_loss: 0.5086 - val_mean_squared_error: 0.4847
Epoch 5/10
2647/2647 [==============================] - 41s 15ms/step - loss: 0.5821 -
mean_squared_error: 0.5581 - val_loss: 0.5161 - val_mean_squared_error: 0.4936
Epoch 6/10
2647/2647 [==============================] - 38s 14ms/step - loss: 0.5677 -
mean_squared_error: 0.5457 - val_loss: 0.5178 - val_mean_squared_error: 0.4946
Epoch 7/10
2647/2647 [==============================] - 40s 15ms/step - loss: 0.5563 -
mean_squared_error: 0.5349 - val_loss: 0.5103 - val_mean_squared_error: 0.4905
Epoch 8/10
2647/2647 [==============================] - 39s 15ms/step - loss: 0.5437 -
mean_squared_error: 0.5229 - val_loss: 0.5244 - val_mean_squared_error: 0.5032
Epoch 9/10
2647/2647 [==============================] - 38s 14ms/step - loss: 0.5380 -
mean_squared_error: 0.5175 - val_loss: 0.5281 - val_mean_squared_error: 0.5076
Epoch 10/10
2647/2647 [==============================] - 42s 16ms/step - loss: 0.5291 -
mean_squared_error: 0.5093 - val_loss: 0.5321 - val_mean_squared_error: 0.5121
```

[ ]:
```
import matplotlib.pyplot as plt

# Plot training & validation loss values
plt.plot(history.history['mean_squared_error'])
plt.plot(history.history['val_mean_squared_error'])
plt.title('Model MSE')
plt.ylabel('MSE')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



The final MSE value on the test set appears to stabilize around 0.4. This means that, on average, the squared difference between the predicted star ratings and the actual star ratings is 0.4. To put this into perspective, the root mean squared error (RMSE), which provides an error in the units of the target variable (ratings in this case), would be the square root of 0.4, which is approximately 0.63. This indicates that the typical prediction is about 0.63 stars away from the actual rating.

13 Convolutional Neural Network (CNN)

We also explored the utility of a Convolutional Neural Network (CNN), known for its effectiveness in handling data with spatial hierarchy, which can be advantageous for text data structured in sequences.

Efficiency:

CNNs can be faster to train than LSTMs because they require fewer sequential computations, which can be a significant advantage especially when working with large datasets.

Effectiveness at Capturing Local Context:

CNNs can effectively capture local context and identify important features like key phrases that might indicate sentiment, without the need for the model to remember long-term dependencies.

[ ]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, GlobalMaxPooling1D,␣
↪Dense, Dropout

model = Sequential([
    Embedding(input_dim=10000, output_dim=64, input_length=100),
    Conv1D(128, 5, activation='relu'),  # 128 filters, kernel size of 5
    GlobalMaxPooling1D(),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='linear')  # Linear output for regression
])

model.compile(optimizer='adam', loss='mean_squared_error',␣
↪metrics=['mean_squared_error'])
model.summary()
```

```
Model: "sequential_5"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_5 (Embedding)     (None, 100, 64)           640000

 conv1d (Conv1D)             (None, 96, 128)           41088

 global_max_pooling1d (Glob  (None, 128)               0
 alMaxPooling1D)

 dense_10 (Dense)            (None, 64)                8256

 dropout_5 (Dropout)         (None, 64)                0

 dense_11 (Dense)            (None, 1)                 65
```

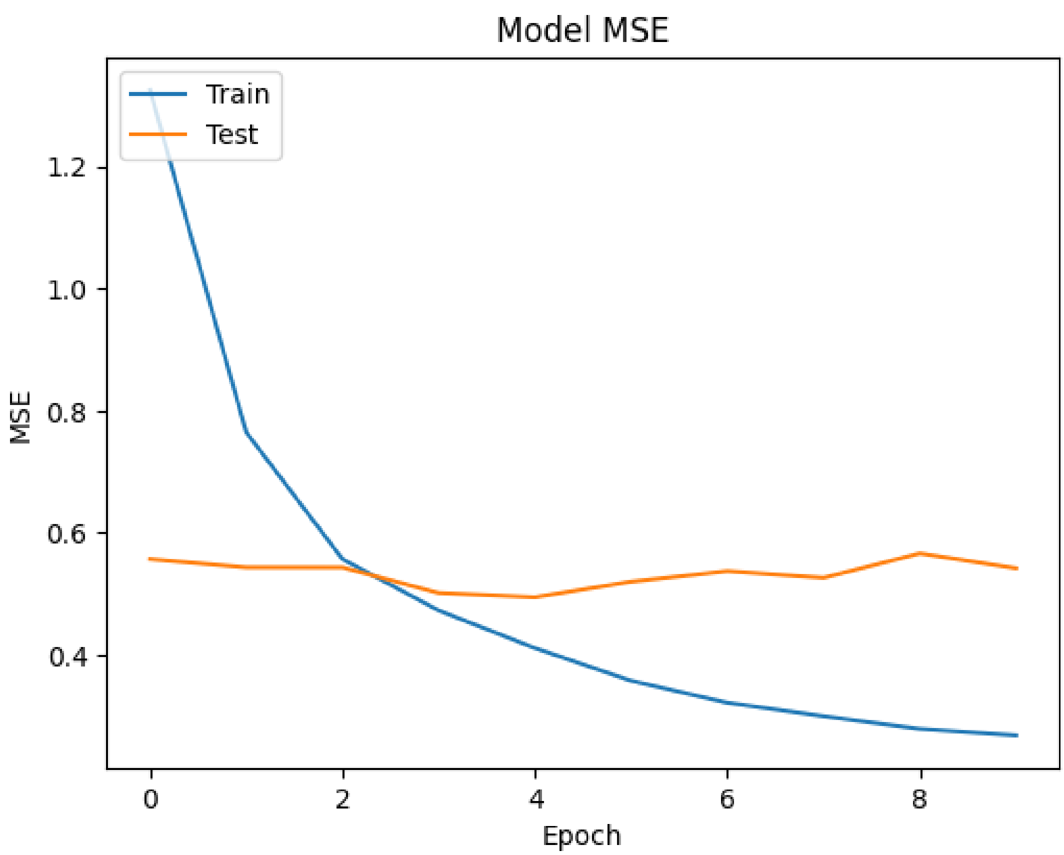
```
=================================================================
Total params: 689409 (2.63 MB)
Trainable params: 689409 (2.63 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Our CNN model was built with a structure comprising an initial Embedding layer configured for 10,000 vocabulary size, producing 64-dimensional embeddings. It was followed by a Convolutional layer with 128 filters of size 5, designed to capture local patterns within sequences of text. A Global Max Pooling layer was then used to reduce the dimensionality, ensuring the most significant features from each feature map were preserved for the final prediction. The network also included a Dense layer with 64 units and 'ReLU' activation to introduce non-linearity, a Dropout layer at 0.5 to mitigate overfitting by randomly omitting subset of features during training, and a final Dense layer for regression output. The model contained a total of 689,409 trainable parameters.

```
[ ]: history = model.fit(X_train, y_train, epochs=10, validation_data=(X_test,␣
      ↪y_test), batch_size=64)
```

```
Epoch 1/10
2647/2647 [==============================] - 59s 21ms/step - loss: 1.3254 -
mean_squared_error: 1.3254 - val_loss: 0.5568 - val_mean_squared_error: 0.5568
Epoch 2/10
2647/2647 [==============================] - 20s 8ms/step - loss: 0.7647 -
mean_squared_error: 0.7647 - val_loss: 0.5434 - val_mean_squared_error: 0.5434
Epoch 3/10
2647/2647 [==============================] - 17s 6ms/step - loss: 0.5569 -
mean_squared_error: 0.5569 - val_loss: 0.5432 - val_mean_squared_error: 0.5432
Epoch 4/10
2647/2647 [==============================] - 17s 6ms/step - loss: 0.4727 -
mean_squared_error: 0.4727 - val_loss: 0.5013 - val_mean_squared_error: 0.5013
Epoch 5/10
2647/2647 [==============================] - 16s 6ms/step - loss: 0.4111 -
mean_squared_error: 0.4111 - val_loss: 0.4944 - val_mean_squared_error: 0.4944
Epoch 6/10
2647/2647 [==============================] - 15s 6ms/step - loss: 0.3576 -
mean_squared_error: 0.3576 - val_loss: 0.5199 - val_mean_squared_error: 0.5199
Epoch 7/10
2647/2647 [==============================] - 15s 6ms/step - loss: 0.3216 -
mean_squared_error: 0.3216 - val_loss: 0.5371 - val_mean_squared_error: 0.5371
Epoch 8/10
2647/2647 [==============================] - 15s 6ms/step - loss: 0.2996 -
mean_squared_error: 0.2996 - val_loss: 0.5263 - val_mean_squared_error: 0.5263
Epoch 9/10
2647/2647 [==============================] - 15s 6ms/step - loss: 0.2787 -
mean_squared_error: 0.2787 - val_loss: 0.5660 - val_mean_squared_error: 0.5660
Epoch 10/10
2647/2647 [==============================] - 14s 5ms/step - loss: 0.2681 -
mean_squared_error: 0.2681 - val_loss: 0.5419 - val_mean_squared_error: 0.5419
```

```
[ ]: import matplotlib.pyplot as plt

# Plot training & validation loss values
plt.plot(history.history['mean_squared_error'])
plt.plot(history.history['val_mean_squared_error'])
plt.title('Model MSE')
plt.ylabel('MSE')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



Upon training, the model's performance over epochs revealed initial rapid improvements in mean squared error (MSE), indicating strong learning. However, the plots showed that while the training loss continued to decrease, the validation loss began to stabilize and then slightly increase, a classic indication of overfitting.

14 Convolutional Neural Network (CNN) with Early Stopping and Dropout Layers

To combat overfitting in our model for Yelp review sentiment analysis, we introduced early stopping and dropout layers into our architecture.

[ ]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, GlobalMaxPooling1D,␣
 ↪Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping

# Define the model
model = Sequential([
    Embedding(input_dim=10000, output_dim=64, input_length=100),
    Conv1D(128, 5, activation='relu'),  # 128 filters, kernel size of 5
    GlobalMaxPooling1D(),
    Dense(64, activation='relu'),
    Dropout(0.5),  # Dropout layer to prevent overfitting
    Dense(1, activation='linear')  # Linear output for regression
])

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error',␣
 ↪metrics=['mean_squared_error'])

# Early stopping callback
early_stopping = EarlyStopping(
    monitor='val_loss',  # Monitor validation loss
    patience=5,          # Number of epochs to wait after min has been hit
    verbose=1,           # To display messages when early stopping triggers
    restore_best_weights=True  # Restores model weights from the epoch with the␣
 ↪minimum validation loss
)

# Summary of the model
model.summary()

# Assuming 'x_train', 'y_train', 'x_val', and 'y_val' are already defined
# Train the model with early stopping
history = model.fit(
    X_train, y_train,
    validation_data=(X_test, y_test),
    epochs=50,  # Set a higher number since early stopping can halt the␣
 ↪training prematurely
    batch_size=32,
    callbacks=[early_stopping]  # Include early stopping in the training process
)
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 100, 64)           640000

 conv1d (Conv1D)             (None, 96, 128)           41088

 global_max_pooling1d (Glob  (None, 128)               0
 alMaxPooling1D)

 dense_4 (Dense)             (None, 64)                8256

 dropout_6 (Dropout)         (None, 64)                0

 dense_5 (Dense)             (None, 1)                 65

=================================================================
Total params: 689409 (2.63 MB)
Trainable params: 689409 (2.63 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/50
5294/5294 [==============================] - 72s 13ms/step - loss: 1.0539 -
mean_squared_error: 1.0539 - val_loss: 0.5218 - val_mean_squared_error: 0.5218
Epoch 2/50
5294/5294 [==============================] - 30s 6ms/step - loss: 0.6049 -
mean_squared_error: 0.6049 - val_loss: 0.4897 - val_mean_squared_error: 0.4897
Epoch 3/50
5294/5294 [==============================] - 29s 5ms/step - loss: 0.4852 -
mean_squared_error: 0.4852 - val_loss: 0.4860 - val_mean_squared_error: 0.4860
Epoch 4/50
5294/5294 [==============================] - 28s 5ms/step - loss: 0.4072 -
mean_squared_error: 0.4072 - val_loss: 0.4958 - val_mean_squared_error: 0.4958
Epoch 5/50
5294/5294 [==============================] - 29s 5ms/step - loss: 0.3614 -
mean_squared_error: 0.3614 - val_loss: 0.5337 - val_mean_squared_error: 0.5337
Epoch 6/50
5294/5294 [==============================] - 28s 5ms/step - loss: 0.3295 -
mean_squared_error: 0.3295 - val_loss: 0.5175 - val_mean_squared_error: 0.5175
Epoch 7/50
5294/5294 [==============================] - 28s 5ms/step - loss: 0.3045 -
mean_squared_error: 0.3045 - val_loss: 0.5514 - val_mean_squared_error: 0.5514
Epoch 8/50
5294/5294 [==============================] - ETA: 0s - loss: 0.2816 -
mean_squared_error: 0.2816Restoring model weights from the end of the best
epoch: 3.
5294/5294 [==============================] - 28s 5ms/step - loss: 0.2816 -
```

```
mean_squared_error: 0.2816 - val_loss: 0.5304 - val_mean_squared_error: 0.5304
Epoch 8: early stopping
```

The model is structured as follows: an input layer with an embedding of 64 dimensions for a vocabulary size of 10,000 words, followed by a 1D convolutional layer with 128 filters and a kernel size of 5. This is intended to capture spatial hierarchies in data by applying the convolution operation across the text sequences. After convolution, a global max pooling layer is used to reduce the dimensionality and summarize the features extracted by the convolutional layer. This is followed by a dense layer with 64 neurons activated by ReLU to introduce non-linearity to the model, a dropout layer with a rate of 0.5 to prevent overfitting by randomly omitting units during training, and finally, a dense output layer with a linear activation for regression tasks. The model has a total of 689,409 parameters.

To further prevent overfitting, the early stopping mechanism monitors the validation loss and stops the training when it ceases to decrease, effectively reducing excessive training epochs and resource consumption. For our model, early stopping kicked in during epoch 8. This setup ensures that the model learns generalizable patterns rather than memorizing the training data.

15 Convolutional Neural Network (CNN) with L2 Regularization

In our Yelp review analysis, we also explored using a Convolutional Neural Network (CNN) enhanced with L2 regularization to tackle the challenge of overfitting.

[ ]:
```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, GlobalMaxPooling1D,␣
↪Dense, Dropout
from tensorflow.keras.regularizers import l2

model = Sequential([
    Embedding(input_dim=10000, output_dim=64, input_length=100),
    Conv1D(128, 5, activation='relu', kernel_regularizer=l2(0.01)),  # Adding␣
↪L2 regularization
    GlobalMaxPooling1D(),
    Dense(64, activation='relu', kernel_regularizer=l2(0.01)),  # Adding L2␣
↪regularization
    Dropout(0.5),
    Dense(1, activation='linear')  # Linear output for regression
])

model.compile(optimizer='adam', loss='mean_squared_error',␣
↪metrics=['mean_squared_error'])
model.summary()
```

```
Model: "sequential_2"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 100, 64)           640000
```

```
 conv1d_1 (Conv1D)           (None, 96, 128)           41088

 global_max_pooling1d_1 (Gl  (None, 128)               0
 obalMaxPooling1D)

 dense_4 (Dense)             (None, 64)                8256

 dropout_2 (Dropout)         (None, 64)                0

 dense_5 (Dense)             (None, 1)                 65

=================================================================
Total params: 689409 (2.63 MB)
Trainable params: 689409 (2.63 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

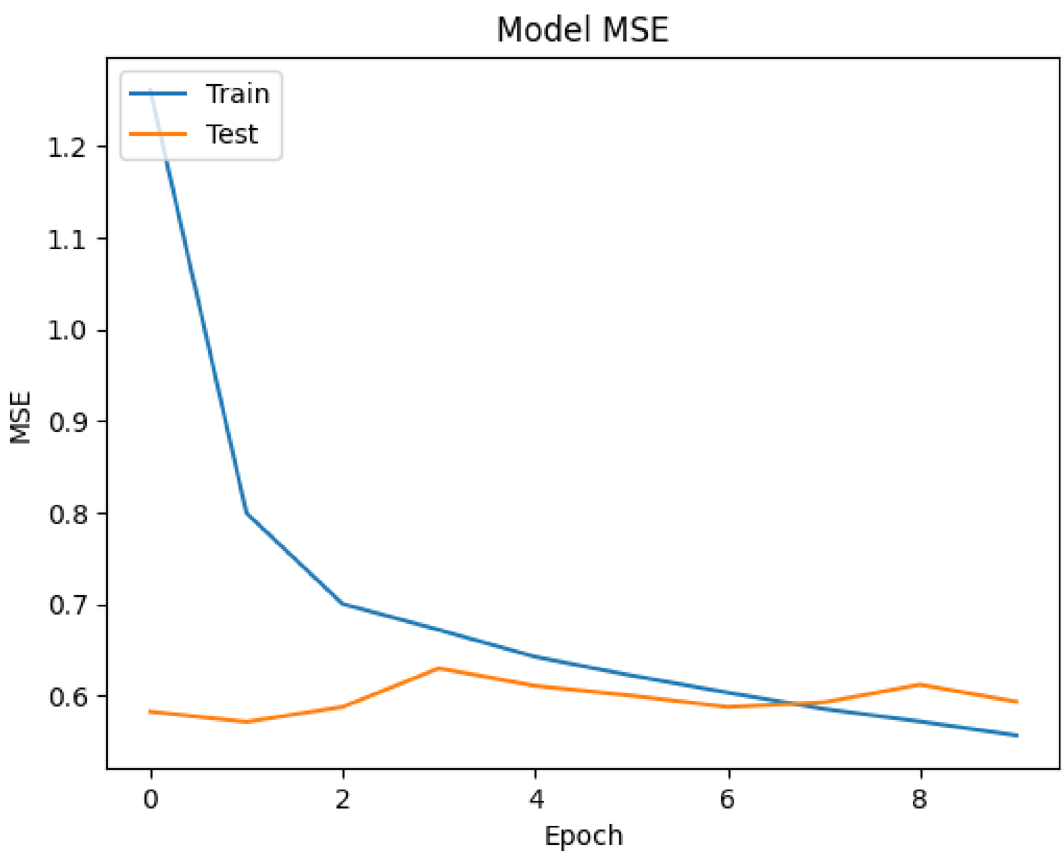
The model includes an embedding layer set to handle up to 10,000 unique words with an embedding dimension of 64. Following this, a convolutional layer with 128 filters and a kernel size of 5 is applied, which is ideal for capturing local patterns in the text. The CNN architecture also incorporates a global max pooling layer, which helps to reduce the dimensionality by capturing the most important feature from each feature map, and two dense layers with ReLU activation function. Dropout layers with a rate of 0.5 are strategically placed to reduce overfitting by randomly turning off a fraction of neurons during training. The final layer is a dense layer with linear activation used for the regression output. This configuration totals 689,409 trainable parameters.

[ ]:
```
history = model.fit(X_train, y_train, epochs=10, validation_data=(X_test,
↪y_test), batch_size=64)
```

```
Epoch 1/10
2647/2647 [==============================] - 58s 21ms/step - loss: 1.4731 -
mean_squared_error: 1.2263 - val_loss: 0.7265 - val_mean_squared_error: 0.6475
Epoch 2/10
2647/2647 [==============================] - 20s 8ms/step - loss: 0.8517 -
mean_squared_error: 0.7849 - val_loss: 0.6252 - val_mean_squared_error: 0.5676
Epoch 3/10
2647/2647 [==============================] - 18s 7ms/step - loss: 0.7310 -
mean_squared_error: 0.6778 - val_loss: 0.6249 - val_mean_squared_error: 0.5785
Epoch 4/10
2647/2647 [==============================] - 15s 6ms/step - loss: 0.6922 -
mean_squared_error: 0.6462 - val_loss: 0.6120 - val_mean_squared_error: 0.5689
Epoch 5/10
2647/2647 [==============================] - 16s 6ms/step - loss: 0.6610 -
mean_squared_error: 0.6187 - val_loss: 0.6538 - val_mean_squared_error: 0.6149
Epoch 6/10
2647/2647 [==============================] - 16s 6ms/step - loss: 0.6369 -
mean_squared_error: 0.5966 - val_loss: 0.6109 - val_mean_squared_error: 0.5725
Epoch 7/10
2647/2647 [==============================] - 16s 6ms/step - loss: 0.6162 -
```

```
mean_squared_error: 0.5771 - val_loss: 0.6536 - val_mean_squared_error: 0.6155
Epoch 8/10
2647/2647 [==============================] - 16s 6ms/step - loss: 0.5972 -
mean_squared_error: 0.5589 - val_loss: 0.6215 - val_mean_squared_error: 0.5855
Epoch 9/10
2647/2647 [==============================] - 15s 6ms/step - loss: 0.5807 -
mean_squared_error: 0.5434 - val_loss: 0.6213 - val_mean_squared_error: 0.5849
Epoch 10/10
2647/2647 [==============================] - 15s 6ms/step - loss: 0.5679 -
mean_squared_error: 0.5310 - val_loss: 0.6360 - val_mean_squared_error: 0.5996
```

[ ]:
```
import matplotlib.pyplot as plt

# Plot training & validation loss values
plt.plot(history.history['mean_squared_error'])
plt.plot(history.history['val_mean_squared_error'])
plt.title('Model MSE')
plt.ylabel('MSE')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



To ensure the robustness of our model, we employed L2 regularization in the convolutional and dense layers, which penalizes large weights and encourages a simpler model that should generalize better on unseen data. The MSE plot from our training session shows that after initial training epochs, both training and validation losses decrease and stabilize without a significant gap between them, suggesting that L2 regularization effectively mitigated the risk of overfitting, as the model performs consistently across both training and validation datasets.

The final MSE value on the test set appears to stabilize around 0.6. This means that, on average, the squared difference between the predicted star ratings and the actual star ratings is 0.6. If we take the square root of the MSE 0.6, it is approximately 0.77. This tells us that the typical prediction is about 0.77 stars away from the actual rating.

16 Predicting the True rating of a Review

[ ]:
```
import pandas as pd
data2 = pd.read_csv("base_df_restaurants_ca.csv")
print(data2["text"].head())
```

```
0    Had a party of 6 here for hibachi. Our waitres…
1    Yes, this is the only sushi place in town. How…
2    Great burgers,fries and salad!  Burgers have a…
3    What a great addition to the Funk Zone!  Grab …
4    We were a bit weary about trying the Shellfish…
Name: text, dtype: object
```

[ ]:
```
from transformers import pipeline

# Load the sentiment-analysis pipeline
sentiment_model = pipeline("sentiment-analysis")

def predict_rating(review):
    # Use the sentiment model to predict the sentiment
    result = sentiment_model(review)

    # Extract the label and convert to numerical rating
    label = result[0]['label']
    score = result[0]['score']

    # Map the sentiment to a Yelp rating scale (1 to 5 stars)
    if label == 'POSITIVE':
        # If sentiment is positive, map score to 3 to 5 stars
        rating = 3 + 2 * score  # Scale and shift score
    else:
        # If sentiment is negative, map score to 1 to 3 stars
        rating = 1 + 2 * (1 - score)  # Invert and scale score
```

```
    return int(round(rating))
```

```
No model was supplied, defaulted to distilbert/distilbert-base-uncased-
finetuned-sst-2-english and revision af0f99b
(https://huggingface.co/distilbert/distilbert-base-uncased-finetuned-
sst-2-english).
Using a pipeline without specifying a model name and revision in production is
not recommended.
```

[ ]:
```
# Enter your review
review = "The food was so so good but I was sick and I will not go back."
predicted_rating = predict_rating(review)
print(f"The predicted true rating for the review is: {predicted_rating}")
```

```
The predicted true rating for the review is: 1
```

[ ]:
```
review = data2["text"][0]
predicted_rating = predict_rating(review)
print(data2["text"][0])
print(f"\nThe predicted true rating for the review is: {predicted_rating}")
print(f"The user rating for the review is: {data2['stars_review'][0]}")
```

```
Had a party of 6 here for hibachi. Our waitress brought our separate sushi
orders on one plate so we couldn't really tell who's was who's and forgot
several items on an order. I understand making mistakes but the restaraunt was
really quiet so we were kind of surprised. Usually hibachi is a fun lively
experience and our  cook  said maybe three words, but he cooked very well his
name was Francisco. Service was fishy, food was pretty good, and im hoping it
was just an off night here. But for the money I wouldn't go back.

The predicted true rating for the review is: 5
The user rating for the review is: 3.0
```

[ ]:
```
review = data2["text"][7]
predicted_rating = predict_rating(review)
print(data2["text"][7])
print(f"\nThe predicted true rating for the review is: {predicted_rating}")
print(f"The user rating for the review is: {data2['stars_review'][7]}")
```

```
This is the first time I tried this place and  I was surprisingly surprised. I
had a combination dinner pad Thai and coconut soup. The soup was very tasty as I
never had coconut soup before. The pad Thai was exactly what I was expecting and
it did not disappoint. The restaurant had great Thai decor and music. The staff
and service was top notch. For a town with not much selection for food, this was
a great change of pace. This may become my go to place in Carp.

The predicted true rating for the review is: 5
The user rating for the review is: 4.0
```

[ ]:
```
review = data2["text"][5]
predicted_rating = predict_rating(review)
print(data2["text"][5])
print(f"\nThe predicted true rating for the review is: {predicted_rating}")
print(f"The user rating for the review is: {data2['stars_review'][5]}")
```

```
If I could give it a zero, I would. I order a plain hamburger, and realized they
put  bacon in it (which I am allergic to and unable to eat) after two bites.
When I went back to the drive-through window to complain (didn't realize the
actual restaurant was open--it was almost 2 after all…), the guy took back the
burger, said nothing, and disappeared. After 2 minutes of awkwardly making
conversation with the next people in line in their car, he came back and rudely
told me I had to go inside to get my food. Which I did. And still did not get an
apology.

I refuse to go back there after that ordeal, which is a shame, because it's nice
to have a variety of places to go to after DT. Guess Freebirds it is!

The predicted true rating for the review is: 1
The user rating for the review is: 1.0
```

[ ]:
```
def predict_review_rating(review):
    # Convert review text into sequences
    sequence = tokenizer.texts_to_sequences([review])
    padded_sequence = pad_sequences(sequence, maxlen=100)

    # Predict using the trained model
    prediction = model.predict(padded_sequence)

    # Get the predicted class (assuming labels are zero-indexed)
    predicted_class = prediction.argmax(axis=1)[0] + 1

    return predicted_class

# Enter your review
review = "The food was so so good but I was sick and I will not go back."
predicted_rating = predict_review_rating(review)
print(f"The predicted true rating for the review is: {predicted_rating}")
```

```
1/1 [==============================] - 0s 41ms/step
The predicted true rating for the review is: 2
```

[ ]:
```
review = data2["text"][0]
predicted_rating = predict_review_rating(review)
print(data2["text"][0])
print(f"\nThe predicted true rating for the review is: {predicted_rating}")
print(f"The user rating for the review is: {data2['stars_review'][0]}")
```

```
1/1 [==============================] - 0s 31ms/step
```

```
Had a party of 6 here for hibachi. Our waitress brought our separate sushi
orders on one plate so we couldn't really tell who's was who's and forgot
several items on an order. I understand making mistakes but the restaraunt was
really quiet so we were kind of surprised. Usually hibachi is a fun lively
experience and our  cook  said maybe three words, but he cooked very well his
name was Francisco. Service was fishy, food was pretty good, and im hoping it
was just an off night here. But for the money I wouldn't go back.

The predicted true rating for the review is: 2
The user rating for the review is: 3.0
```

[ ]:
```python
review = data2["text"][7]
predicted_rating = predict_review_rating(review)
print(data2["text"][7])
print(f"\nThe predicted true rating for the review is: {predicted_rating}")
print(f"The user rating for the review is: {data2['stars_review'][7]}")
```

```
1/1 [==============================] - 0s 34ms/step
This is the first time I tried this place and  I was surprisingly surprised. I
had a combination dinner pad Thai and coconut soup. The soup was very tasty as I
never had coconut soup before. The pad Thai was exactly what I was expecting and
it did not disappoint. The restaurant had great Thai decor and music. The staff
and service was top notch. For a town with not much selection for food, this was
a great change of pace. This may become my go to place in Carp.

The predicted true rating for the review is: 5
The user rating for the review is: 4.0
```

[ ]:
```python
review = data2["text"][5]
predicted_rating = predict_review_rating(review)
print(data2["text"][5])
print(f"\nThe predicted true rating for the review is: {predicted_rating}")
print(f"The user rating for the review is: {data2['stars_review'][5]}")
```

```
1/1 [==============================] - 0s 31ms/step
If I could give it a zero, I would. I order a plain hamburger, and realized they
put  bacon in it (which I am allergic to and unable to eat) after two bites.
When I went back to the drive-through window to complain (didn't realize the
actual restaurant was open--it was almost 2 after all…), the guy took back the
burger, said nothing, and disappeared. After 2 minutes of awkwardly making
conversation with the next people in line in their car, he came back and rudely
told me I had to go inside to get my food. Which I did. And still did not get an
apology.

I refuse to go back there after that ordeal, which is a shame, because it's nice
to have a variety of places to go to after DT. Guess Freebirds it is!

The predicted true rating for the review is: 1
The user rating for the review is: 1.0
```

17 Getting Started (DistilBERT Model)

[ ]:
```
import os

# Change the current working directory to the directory where you downloaded␣
↪the files
os.chdir('/content/drive/MyDrive/project personal work')

!ls
```

```
base_df_restaurants_ca.csv        'PIC16 Report.gdoc'
df_ca.csv                          yelp_academic_dataset_business.csv
df_restaurants_ca2.csv             yelp_academic_dataset_business.json
df_restaurants_ca3.csv             yelp_academic_dataset_review.csv
df_restaurants_ca.csv              yelp_academic_dataset_review.json
DistilBERT.ipynb                   yelp_academic_dataset_user.csv
everything.ipynb                   yelp_academic_dataset_user.json
normalized_df_restaurants_ca.csv   yelp_combined.csv
```

18 DistilBERT Model (subset)

Our entire dataset contains ~211k reviews. We lack the computational resources to train on the entire dataset. To train one epoch it will take approximately 2h 40m using the GPU which is almost the entire duration that we get to use the GPU for free (3h 20m). Therefore, we have opted to create a subset of the dataset to combat this problem.

18.1 Use stratified sampling to create a subset of dataset

[ ]:
```
import pandas as pd

# Load the normalized data
df = pd.read_csv('base_df_restaurants_ca.csv')
X = df['text']  # column that contains the reviews
y = df['stars_review']  # column that contains the customer's star rating
```

[ ]:
```
base_num_reviews = len(df['text'])

print(f"There are {base_num_reviews} reviews in the text column for␣
↪base_df_restaurants_ca.csv.")
```

```
There are 211748 reviews in the text column for base_df_restaurants_ca.csv.
```

[ ]:
```
import numpy as np

# Create subsets for both df
# Using stratified ensures same distribution
'''
For example, if the stars_review column has the following distribution:
```

```
5-star reviews: 40%
4-star reviews: 30%
3-star reviews: 20%
1-star reviews: 10%
The code will ensure that the subset data has the same distribution, i.e.,
approximately 40% of the subset data will have 5-star reviews,
30% will have 4-star reviews, etc.
'''
from sklearn.model_selection import StratifiedShuffleSplit

# Create a subset (20%) of our original dataset or 80% less than the original␣
↪dataset
sss_df = StratifiedShuffleSplit(n_splits=1, test_size=0.8, random_state=42)
subset_X_df, subset_y_df = [], []
for train_index, val_index in sss_df.split(df['text'], df['stars_review']):
    subset_X_df.extend(np.array(df['text'])[train_index])
    subset_y_df.extend(np.array(df['stars_review'])[train_index])

# Create new DataFrames from the subset data
new_df = pd.DataFrame({'text': subset_X_df, 'stars_review': subset_y_df})

# Save the DataFrames to new CSV files
new_df.to_csv('base_subset_df.csv', index=False)
```

[ ]:
```
subset_base_num_reviews = len(new_df['text'])

print(f"There are {subset_base_num_reviews} reviews in the text column for␣
↪base_df_restaurants_ca.csv.")
```

```
There are 42349 reviews in the text column for base_df_restaurants_ca.csv.
```

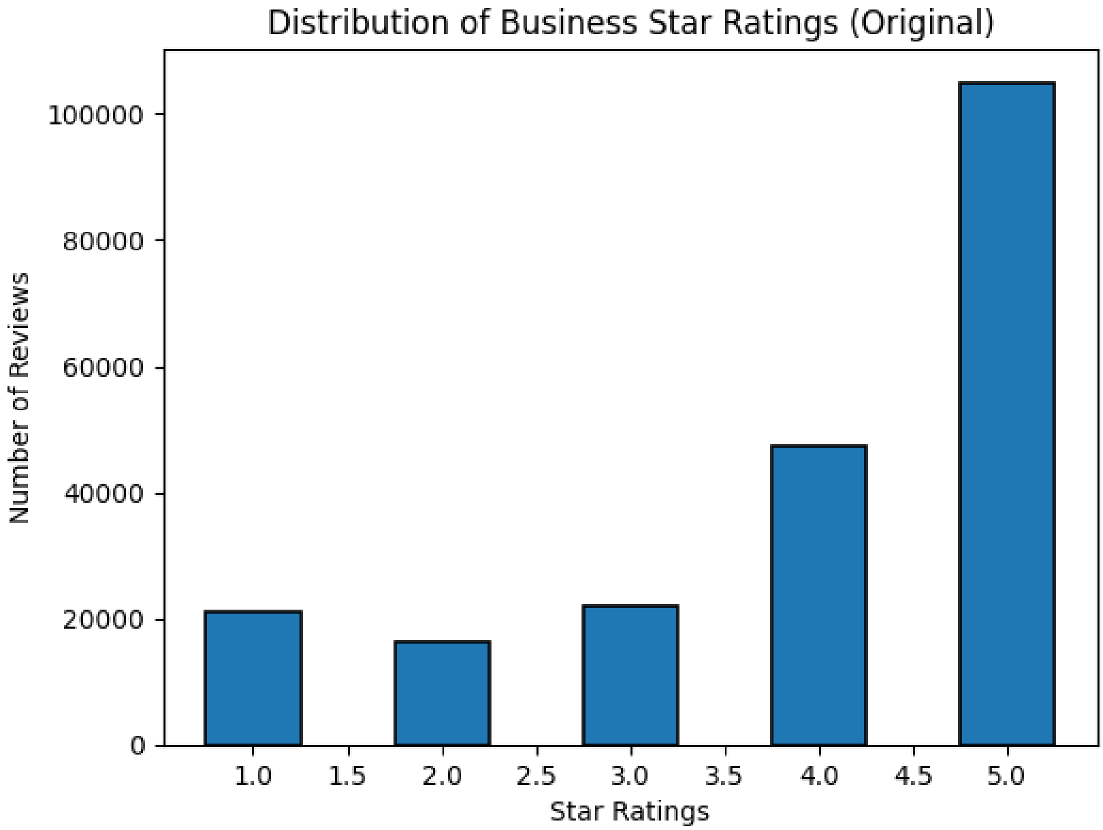
18.2 Graphs / Visualization

We see that the distribution of the original and subset dataset is similar. That means our subset managed to keep true the distribution of the original dataset.

18.2.1 Star ratings distribution of original dataset

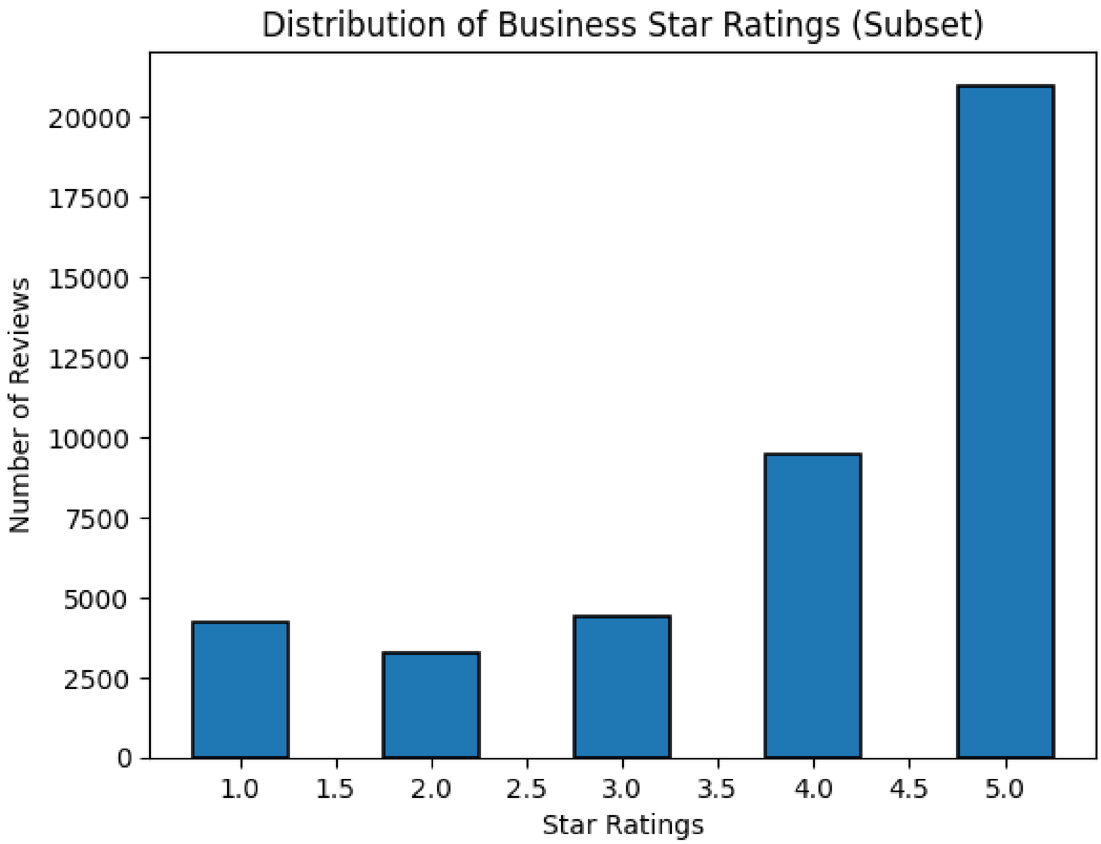
[ ]:
```
import matplotlib.pyplot as plt

# Create a bar chart for df
plt.bar(df["stars_review"].value_counts().index, df["stars_review"].
↪value_counts().values, edgecolor='black', linewidth=1.2, width=0.5)
plt.title('Distribution of Business Star Ratings (Original)')
plt.xlabel('Star Ratings')
plt.ylabel('Number of Reviews')
plt.xticks(ticks=np.arange(1, 5.5, 0.5))
```

```
plt.show()
```



18.2.2 Star ratings distribution of subset

```
[ ]: import matplotlib.pyplot as plt

# Create a bar chart for new_df
plt.bar(new_df["stars_review"].value_counts().index, new_df["stars_review"].
 ↪value_counts().values, edgecolor='black', linewidth=1.2, width=0.5)
plt.title('Distribution of Business Star Ratings (Subset)')
plt.xlabel('Star Ratings')
plt.ylabel('Number of Reviews')
plt.xticks(ticks=np.arange(1, 5.5, 0.5))
plt.show()
```



18.3 DistilBERT Model Code

Models	Parameters
Bert-base	340M
ROBERTa	355M
DistilBERT	66M

We chose to use the `DistilBERT` model since it has the least amount of parameters which can help us with having a faster training time and being less resource intensive.

[ ]:
```
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
from transformers import DistilBertTokenizer, TFDistilBertModel
from tensorflow.keras import layers, Model
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt
import numpy as np
```

Normally we would use the normalized data but here we are going to use the raw subset of the dataset that we created earlier in the file `base_subset_df.csv`. We want to use the raw dataset because we want the model to leverage its pre-trained knowledge effectively.

```
[ ]: # Load the data
subset_base_df = pd.read_csv('base_subset_df.csv')

# Split the data into training and testing sets
X = subset_base_df['text']  # column that contains the reviews
y = subset_base_df['stars_review']  # column that contains the customer's star␣
 ↪rating

# Check how many reviews are contained in the normalized_df_restaurants_ca.csv␣
 ↪dataset
base_subset_num_reviews = len(subset_base_df['text'])
print(f"There are {base_subset_num_reviews} reviews in the text column for␣
 ↪base_subset_df.csv.")
```

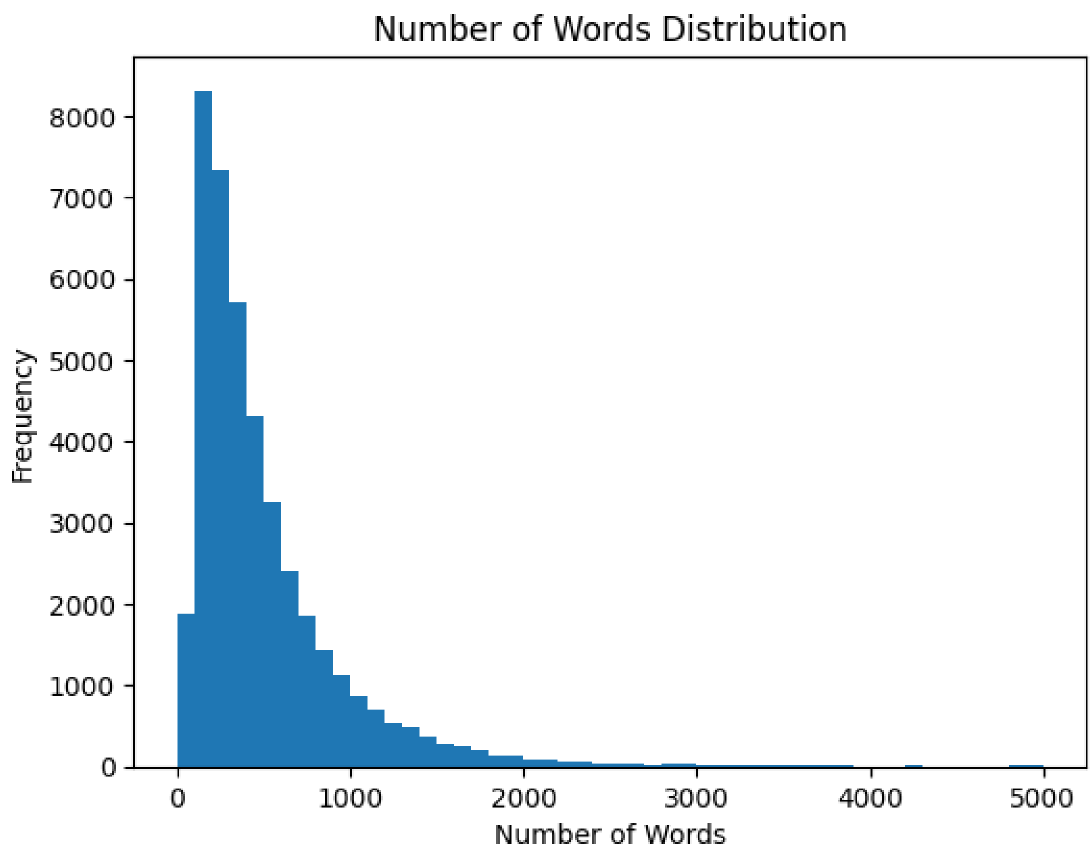
```
There are 42349 reviews in the text column for base_subset_df.csv.
```

Prepare the model for training.

```
[ ]: # Split the dataset
train_df, test_df = train_test_split(subset_base_df, test_size=0.2,␣
 ↪random_state=42)

# Extract text and labels
train_texts = train_df['text'].tolist()
train_labels = train_df['stars_review'].tolist()
test_texts = test_df['text'].tolist()
test_labels = test_df['stars_review'].tolist()
```

Check to see the number of words in the reviews.

```
[ ]: # Calculate the words of each text sample
words = [len(text) for text in train_texts + test_texts]

# Plot distribution
plt.hist(words, bins=50)
plt.xlabel('Number of Words')
plt.ylabel('Frequency')
plt.title('Number of Words Distribution')
plt.show()

# Calculate the 95th percentile of the number of words
percentile_95 = np.percentile(words, 95)

print(f'95th percentile of number of words: {percentile_95}')
```




```
95th percentile of number of words: 1398.0
```

Seeing how the 95th percentile of number of words is 1398 words, we initially set `max_length=1398` but when we ran the training code we, almost instantly, ran out of memory.

We tried the 90th, 80th, and 70th percentile to set as our new `max_length` value yet we still ran out of memory fairly early on. Ultimately we decided on `max_length=512` since it's large enough to capture most of the words in the reviews without causing the training to run out of memory.

```
[ ]: # Takes 1 min(s)

# Initialize the tokenizer
tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')

# Tokenize and encode the texts
def encode_texts(texts):
    return tokenizer(
        texts,
        max_length=512,
        truncation=True,
        padding=True,
```

```
        return_tensors='tf'
    )

train_encodings = encode_texts(train_texts)
test_encodings = encode_texts(test_texts)
```

[ ]:
```
# Prepare TensorFlow datasets
train_dataset = tf.data.Dataset.from_tensor_slices((
    dict(train_encodings),
    train_labels
))

test_dataset = tf.data.Dataset.from_tensor_slices((
    dict(test_encodings),
    test_labels
))

# Batch the datasets
train_dataset = train_dataset.shuffle(10000).batch(32)
test_dataset = test_dataset.batch(32)
```

Creating the model using DistilBERT for feature extraction and add a regression head for predicting star ratings.

[ ]:
```
# Load the DistilBERT model
distilbert_model = TFDistilBertModel.from_pretrained('distilbert-base-uncased')

# Build the model
input_ids = layers.Input(shape=(512,), dtype=tf.int32, name='input_ids')
attention_mask = layers.Input(shape=(512,), dtype=tf.int32,␣
↪name='attention_mask')

# Define a DistilBERT layer
class DistilBERTLayer(layers.Layer):
    """
    Wraps DistilBERT model.
    Take tokenized input and return the last hidden state of the DistilBERT␣
↪model.
    """

    def __init__(self, **kwargs):
        """
        Initializes the DistilBERTLayer.
        """
        super(DistilBERTLayer, self).__init__(**kwargs)
        # Load the pretrained DistilBERT model
```

```python
        self.distilbert = TFDistilBertModel.
↪from_pretrained('distilbert-base-uncased')

    def call(self, inputs):
        """
        Defines the computation from inputs to outputs.

        Args:
            inputs (tuple): A tuple containing two elements:
                - input_ids: Tensor of tokenized input ids.
                - attention_mask: Tensor of same shape as input_ids indicating␣
↪which tokens should be attended to.

        Returns:
            last_hidden_state (Tensor): Last hidden state from the DistilBERT␣
↪model.
        """
        # Unpack the inputs
        input_ids, attention_mask = inputs
        # Pass the inputs through the DistilBERT model
        outputs = self.distilbert(input_ids, attention_mask=attention_mask)
        # Extract the last hidden state
        last_hidden_state = outputs.last_hidden_state
        return last_hidden_state


distilbert_output = DistilBERTLayer()([input_ids, attention_mask])
hidden_state = distilbert_output[:, 0, :]  # Take the [CLS] token's output

# Add a regression head
output = layers.Dense(1, activation='linear')(hidden_state)

model = Model(inputs=[input_ids, attention_mask], outputs=output)

# Compile the model
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=3e-5),
              loss='mean_squared_error',
              metrics=['mae'])
```

```
model.safetensors:   0%|          | 0.00/268M [00:00<?, ?B/s]
```

```
Some weights of the PyTorch model were not used when initializing the TF 2.0
model TFDistilBertModel: ['vocab_layer_norm.bias', 'vocab_projector.bias',
'vocab_layer_norm.weight', 'vocab_transform.bias', 'vocab_transform.weight']
- This IS expected if you are initializing TFDistilBertModel from a PyTorch
model trained on another task or with another architecture (e.g. initializing a
TFBertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing TFDistilBertModel from a PyTorch
```

```
model that you expect to be exactly identical (e.g. initializing a
TFBertForSequenceClassification model from a BertForSequenceClassification
model).
All the weights of TFDistilBertModel were initialized from the PyTorch model.
If your task is similar to the task the model of the checkpoint was trained on,
you can already use TFDistilBertModel for predictions without further training.
Some weights of the PyTorch model were not used when initializing the TF 2.0
model TFDistilBertModel: ['vocab_layer_norm.bias', 'vocab_projector.bias',
'vocab_layer_norm.weight', 'vocab_transform.bias', 'vocab_transform.weight']
- This IS expected if you are initializing TFDistilBertModel from a PyTorch
model trained on another task or with another architecture (e.g. initializing a
TFBertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing TFDistilBertModel from a PyTorch
model that you expect to be exactly identical (e.g. initializing a
TFBertForSequenceClassification model from a BertForSequenceClassification
model).
All the weights of TFDistilBertModel were initialized from the PyTorch model.
If your task is similar to the task the model of the checkpoint was trained on,
you can already use TFDistilBertModel for predictions without further training.
```

When we first initially gotten to the training phase we were using the entire dataset. Training it for 5 epochs was estimated to take 12 hours. We also ran into the issue of running out of memory. That is why, this time, we decided to use a subset of the dataset.

Here we train our model with early stopping to stop the training early if convergence is met on the validation set.

[ ]:
```
# Takes 2 hours

# Early stopping callback
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=3,
    restore_best_weights=True
)

# Set an initial number of epochs
initial_epochs = 5

# Train the model with early stopping
history = model.fit(
    train_dataset,
    validation_data=test_dataset,
    epochs=initial_epochs,
    callbacks=[early_stopping]
)
```

```
Epoch 1/5
1059/1059 [==============================] - 1974s 2s/step - loss: 0.4190 - mae:
```

```
0.4750 - val_loss: 0.3246 - val_mae: 0.4463
Epoch 2/5
1059/1059 [==============================] - 1896s 2s/step - loss: 0.2297 - mae:
0.3553 - val_loss: 0.2966 - val_mae: 0.3844
Epoch 3/5
1059/1059 [==============================] - 1896s 2s/step - loss: 0.1480 - mae:
0.2829 - val_loss: 0.3012 - val_mae: 0.3982
Epoch 4/5
1059/1059 [==============================] - 1897s 2s/step - loss: 0.0939 - mae:
0.2191 - val_loss: 0.3099 - val_mae: 0.3842
Epoch 5/5
1059/1059 [==============================] - 1894s 2s/step - loss: 0.0586 - mae:
0.1701 - val_loss: 0.3368 - val_mae: 0.4073
```

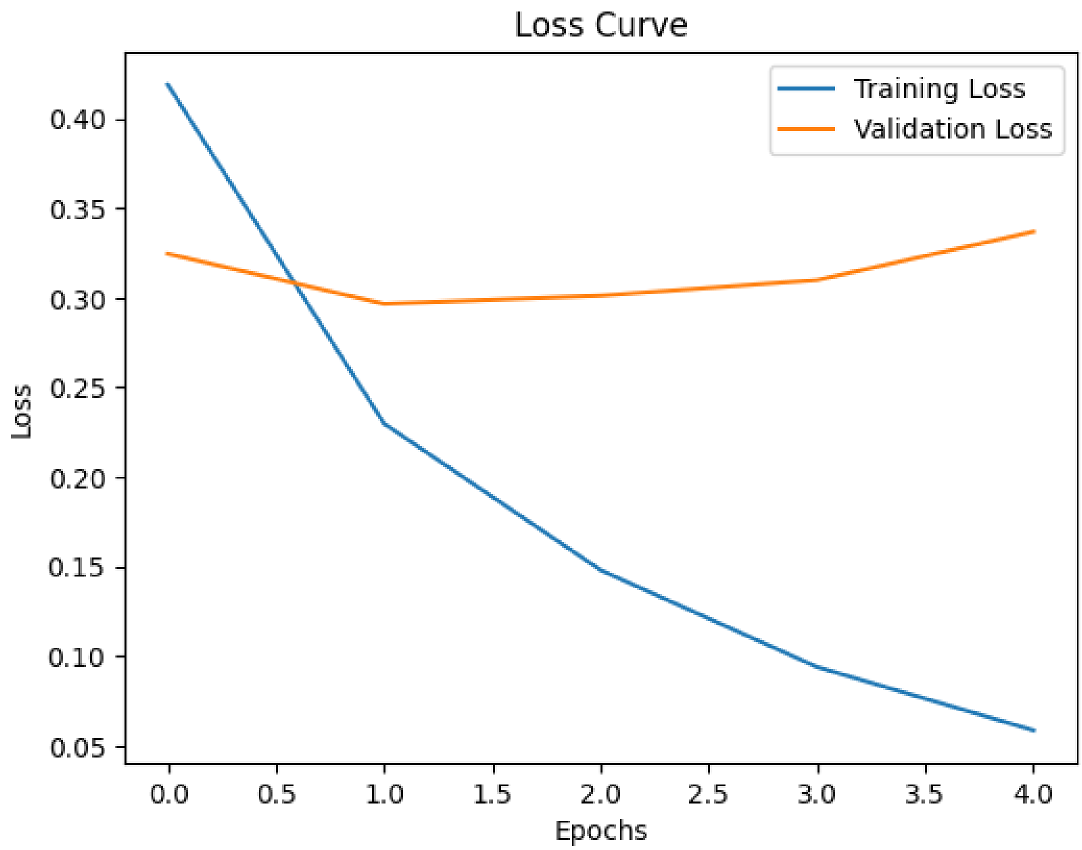
[ ]:
```python
# Evaluate the model on the test dataset
loss, mae = model.evaluate(test_dataset)
print(f'Test Loss: {loss}')
print(f'Test MAE: {mae}')
```

```
265/265 [==============================] - 168s 632ms/step - loss: 0.2966 - mae:
0.3844
Test Loss: 0.29659172892570496
Test MAE: 0.3843823969364166
```

Visualization.

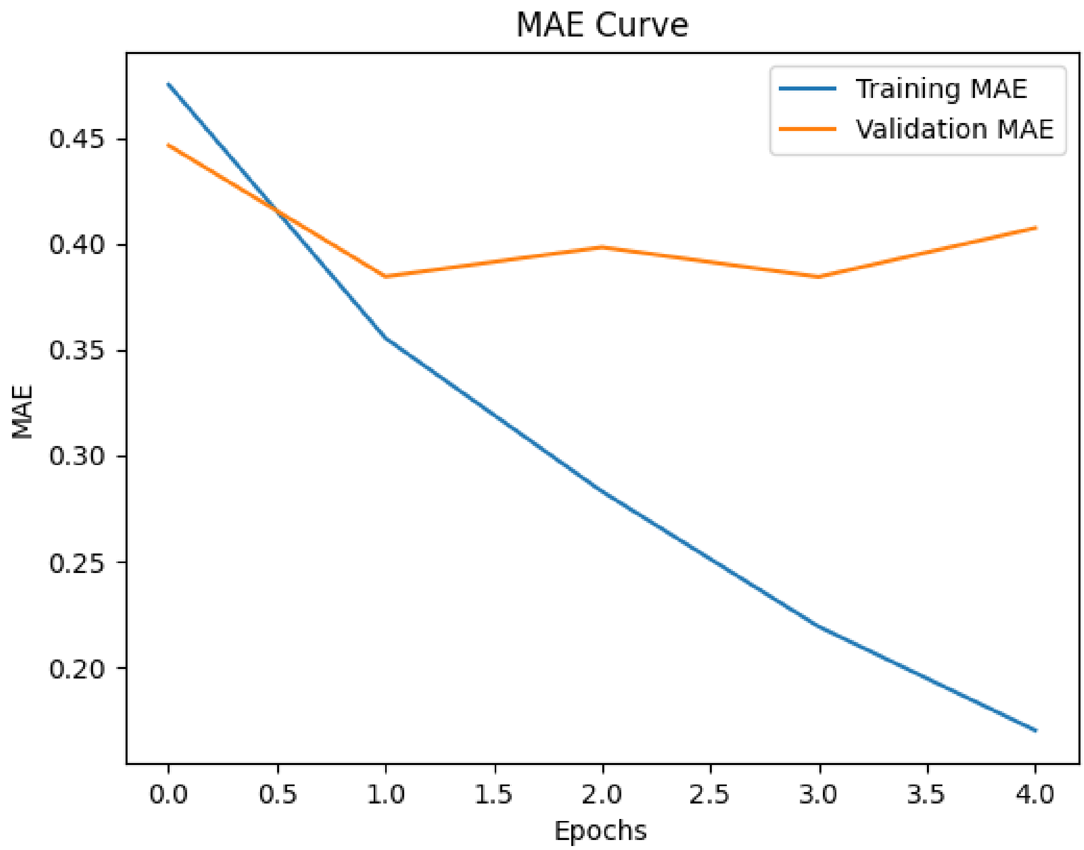
After the first epoch the training loss continues to decrease while the validation loss continues to increase. This is a sign of overfitting.

[ ]:
```python
# Plot the loss curve
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss Curve')
plt.legend()
plt.show()
```



Likewise we see the same behavior here for the training and validation MAE. After the first epoch the training loss continues to decrease while the validation loss continues to increase. This is a sign of overfitting.

[ ]:
```
# Plot the MAE curve
plt.plot(history.history['mae'], label='Training MAE')
plt.plot(history.history['val_mae'], label='Validation MAE')
plt.xlabel('Epochs')
plt.ylabel('MAE')
plt.title('MAE Curve')
plt.legend()
plt.show()
```



A possible explanation to our model overfitting is due to us being limited by setting our `max_length=512`. If we set the value any higher we will run out of memory. Setting the `max_length=512` only captures about 30% of the words in the reviews so during training our model is not able to general well.