CS105 - Exploration of Sentiments in Lolita

Final Project Report

Lillian Xiao

1 Project Description

1.1 Introduction

For the project we will be mining and analyzing the reviews of the novel “Lolita” by Vladimir Nabokov through the use of text processing to perform linear regression and sentiment analysis.

1.2 Description of Project

The ultimate goal of our project is to evaluate overall reader opinion on the 1955 novel “Lolita.” To achieve this, we planned to use two techniques on the data scraped from Goodreads: logistic regression and k-means clustering.

1.3 Techniques

First we web scraped the Goodreads reviews on the book Lolita using beautiful soup to see if there is any one or few ideas that are mentioned a lot. We will then utilize the TF-IDF text vectorization technique to calculate the frequencies of certain terminologies, and vectorize the data. We will use the vectors to find cosine similarity and use hierarchical clustering to find patterns in the reviews. Through the reviews we will be able to use sentiment analysis to see how the audience feels about the book and what ideas or themes that people believe the book is trying to convey to the audience.

1.4 Hypotheses

Although “Lolita” is a highly controversial novel, it is critically acclaimed and a very iconic novel. We believe that despite the controversies, “Lolita” will have mostly positive reviews.

2 Data Collection and Data Cleaning

2.1 Data Scraping

In order to scrape the data from GoodReads, we used beautifulsoup. First, we had to install the library openpyxl in order to transfer the data into a csv file. Then we had to see if beautifulSoup’s status code was 200, to see if it worked. Then we initialized the URL and initialized the reviews dictionary. We then used a for loop to scrape

through 200 pages to get their reviews that were stored in the class “ReviewCard__content” and appended them to our dictionary. Finally, once all the data was scraped we were able to save the data frame to a csv file.

Code :

```
import requests
from bs4 import BeautifulSoup
import pandas as pd
```

```
# Function to get reviews from a specific page
def get_reviews(page_url, headers):
    response = requests.get(page_url, headers=headers)
    if response.status_code != 200:
        return None
    soup = BeautifulSoup(response.text, 'html.parser')
    reviews = soup.find_all('article', class_='ReviewCard')
    return reviews

# Initial URL
url = "https://www.goodreads.com/book/show/123372185-lolita-by-vladimir-nabokov"
user_agent = {'User-agent': 'Mozilla/5.0'}
headers = {'User-Agent': 'Mozilla/5.0'}

# Initialize the review dictionary
review_dict = {'reviews':[], 'rating': []}
```

```
# Simulate pagination
for page_num in range(1, 200):  # Adjust the range based on how many pages you want to scrape
    page_url = f"{url}?page={page_num}"  # Modify this if the pagination parameter is different
    reviews = get_reviews(page_url, headers)

    if not reviews:
        break

    for review in reviews:
        # Find the review text
        review_content = review.find('section', class_='ReviewText__content')
        if review_content:
            review_text = review_content.get_text(strip=True)
            review_dict['reviews'].append(review_text)
        else:
            review_dict['reviews'].append(None)

        # Find the rating
        rating_element = review.find('span', class_="RatingStars RatingStars__small")
        if rating_element:
            rating = rating_element.get('aria-label')
            review_dict['rating'].append(rating)
        else:
            review_dict['rating'].append(None)

    print(f"Scraped page {page_num}")

# Convert the dictionary to a DataFrame
df = pd.DataFrame(review_dict)
print(df.head())

# Save the DataFrame to a CSV file
df.to_csv('scraped.csv', index=False)
```

Output :

```
Scraped page 194
Scraped page 195
Scraped page 196
Scraped page 197
Scraped page 198
Scraped page 199
                                             reviews             rating
0  Between the CoversAfter re-reading "Lolita", I...  Rating 5 out of 5
1  Nymph. Nymphet. Nymphetiquette. Nymphology. Ny...  Rating 5 out of 5
2  Now, this is going to be embarrassing to admit...  Rating 5 out of 5
3  I wasn't even going to write a review ofLolita...  Rating 4 out of 5
4  when i first read this book, i hated every sec...  Rating 4 out of 5
```

2.2 Data Cleaning

Here is a preview of the raw data scraped from GoodReads:

```
# load dataset
df_1 = pd.read_csv("scraped.csv")
df_1.head()
```

	reviews	rating
0	Between the CoversAfter re-reading "Lolita", I...	Rating 5 out of 5
1	Nymph. Nymphet. Nymphetiquette. Nymphology. Ny...	Rating 5 out of 5
2	Now, this is going to be embarrassing to admit...	Rating 5 out of 5
3	I wasn't even going to write a review ofLolita...	Rating 4 out of 5
4	when i first read this book, i hated every sec...	Rating 4 out of 5

First, we filtered out some of the observations based on whether the review text was in English or not using the library langdetect.

```
# Filter out non-English reviews
df_2 = df_1.copy()

from langdetect import detect

def is_eng(text):
    return detect(text) == 'en'

# add col: eng = True if english
df_2['eng'] = df_2['reviews'].apply(is_eng)
# filter out eng = False & remove eng column
df_2 = df_2[df_2['eng']==True].iloc[:, 0:3]
```

Next, we imported the natural Language Processing Toolkit to remove stopwords, break the text into tokens, lemmatize the text and fix typos.

```
# tokenize & Lemmatize text
import nltk
from nltk.corpus import words
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

# Load stopwords and words corpus
stop_words = set(stopwords.words('english'))
english_words = set(words.words())

# initialize  lemmatizer
lemmatizer = WordNetLemmatizer()

# function to clean review text
def clean_review_text(review_text):
    # tokenize reviews into words, remove stop words, and lemmatize
    # the words part removes words that were vectorized wrong
    cleaned_words = [lemmatizer.lemmatize(token) for token in nltk.word_tokenize(review_text.lower()) if token not in stop_words and token in english_words]

    # reconstruct the cleaned review text
    cleaned_review_text = ' '.join(cleaned_words)

    return cleaned_review_text
```

```
df_3 = df_2.copy()
# apply function to text
df_3['reviews'] = df_3['reviews'].apply(clean_review_text)
```

```
df_3 = df_3.iloc[:, :2]
```

Finally, we converted the column of star ratings from strings to numerical values and saved this dataframe to CSV.

```
# Convert ratings var to numerical

ratings_dict = {"Rating 5 out of 5": 5, "Rating 4 out of 5": 4, "Rating 3 out of 5": 3,
                "Rating 2 out of 5": 2, "Rating 1 out of 5": 1}

df_3['rating']=df_3['rating'].replace(ratings_dict)

df_3.head()
```

	reviews	rating
0	local bookseller ever read firmly going either...	5.0
1	nymph nymphet never think year old way stain b...	5.0
2	going embarrassing know reading enjoying book ...	5.0
3	even going write review finishing honestly man...	4.0
4	first read book every second pride reader dist...	4.0

```
df_3.to_csv("cleaned_goodreads_reviews.csv")
```

2.3 Vectorization Using TF-IDF

To convert the raw text into vectors, we used the TfidfVectorizer from sklearn. The resulting dataframe tells us how prevalent each word is within a given review relative to the entire corpus of reviews.

```
from sklearn.feature_extraction.text import TfidfVectorizer

# initialize vectorizer object
vectorizer = TfidfVectorizer()
# transform each review into row of tf-idf'ed features
matrix = vectorizer.fit_transform(df_3['reviews'])
# extract list of features
features = vectorizer.get_feature_names_out()

# combine in dataframe & create csv
df_tfidf = pd.DataFrame(matrix.toarray(), columns=features)
df_tfidf.to_csv("all_tfidf.csv")
df_tfidf.head()
```

	aback	abandon	ability	abject	able	abnormal	abortion	absolute	absolutely	absorbed	...	yearlong	yes
0	0.02469	0.02469	0.0	0.0	0.016814	0.0	0.0	0.000000	0.0	0.000000	...	0.0	0.000000
1	0.00000	0.00000	0.0	0.0	0.023185	0.0	0.0	0.028529	0.0	0.000000	...	0.0	0.057058
2	0.00000	0.00000	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.084702	...	0.0	0.000000
3	0.00000	0.00000	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.000000	...	0.0	0.000000
4	0.00000	0.00000	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.000000	...	0.0	0.000000

5 rows × 2689 columns

We saved this dataframe in a separate CSV file for later use.

3 EDA

Goal: Have a better understanding through visualizations of the relationships between the TF-IDF terms.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud

try:
    df = pd.read_csv('clean_tfidf.csv', encoding='ISO-8859-1')
except UnicodeDecodeError:
    print("Failed to read the file with ISO-8859-1 encoding.")

df = pd.DataFrame(df) # 5214 reviews
df.head()
```

	Unnamed: 0	review	rating	eng	words	girl	age	crime	child	moral	language	relationship	nabokov
0	0	Between the CoversAfter re-reading "Lolita", I...	5.0	True	['coversafter', 'rereading', 'lolita', 'asked'...	0.000000	0.042840	0.087179	0.009836	0.077652	0.009786	0.057813	0.043964
1	1	Nymph. Nymphet. Nymphetiquette. Nymphology. Ny...	5.0	True	['nymph', 'nymphet', 'nymphetiquette', 'nympho...	0.036773	0.030969	0.047265	0.000000	0.018711	0.028296	0.000000	0.000000
2	2	Now, this is going to be embarrassing to admit...	5.0	True	['going', 'embarrassing', 'admitas', 'know', '...	0.057137	0.036088	0.036719	0.066286	0.000000	0.000000	0.116882	0.021163
3	3	I wasn't even going to write a review ofLolita...	4.0	True	['wasnt', 'even', 'going', 'write', 'review', ...	0.000000	0.000000	0.000000	0.049274	0.000000	0.049021	0.000000	0.062926
4	4	when i first read this book, i hated every sec...	4.0	True	['first', 'read', 'book', 'hated', 'every', 's...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

We read in the data and displayed the head in order to have a better understanding of the dataframe.

The read in dataset had TF-IDF terms that were amongst the most numerous tokens.

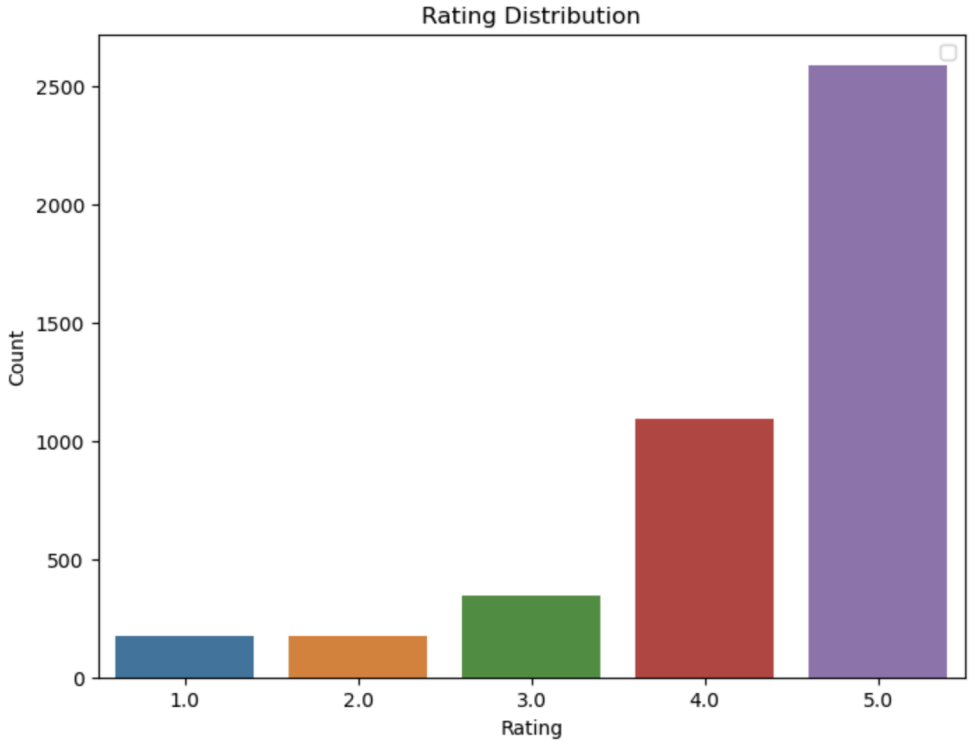
Then, we dropped the NA values.

```
df.dropna(inplace=True) # make sure no N/A's
# print(df.isnull().sum())
len(df)
```

```
4382
```

After dropping the NA values, we ended up having 4,382 reviews to work with.

Below, we can see a bar chart representing the spread of the 4,382 reviews by count of their ratings.

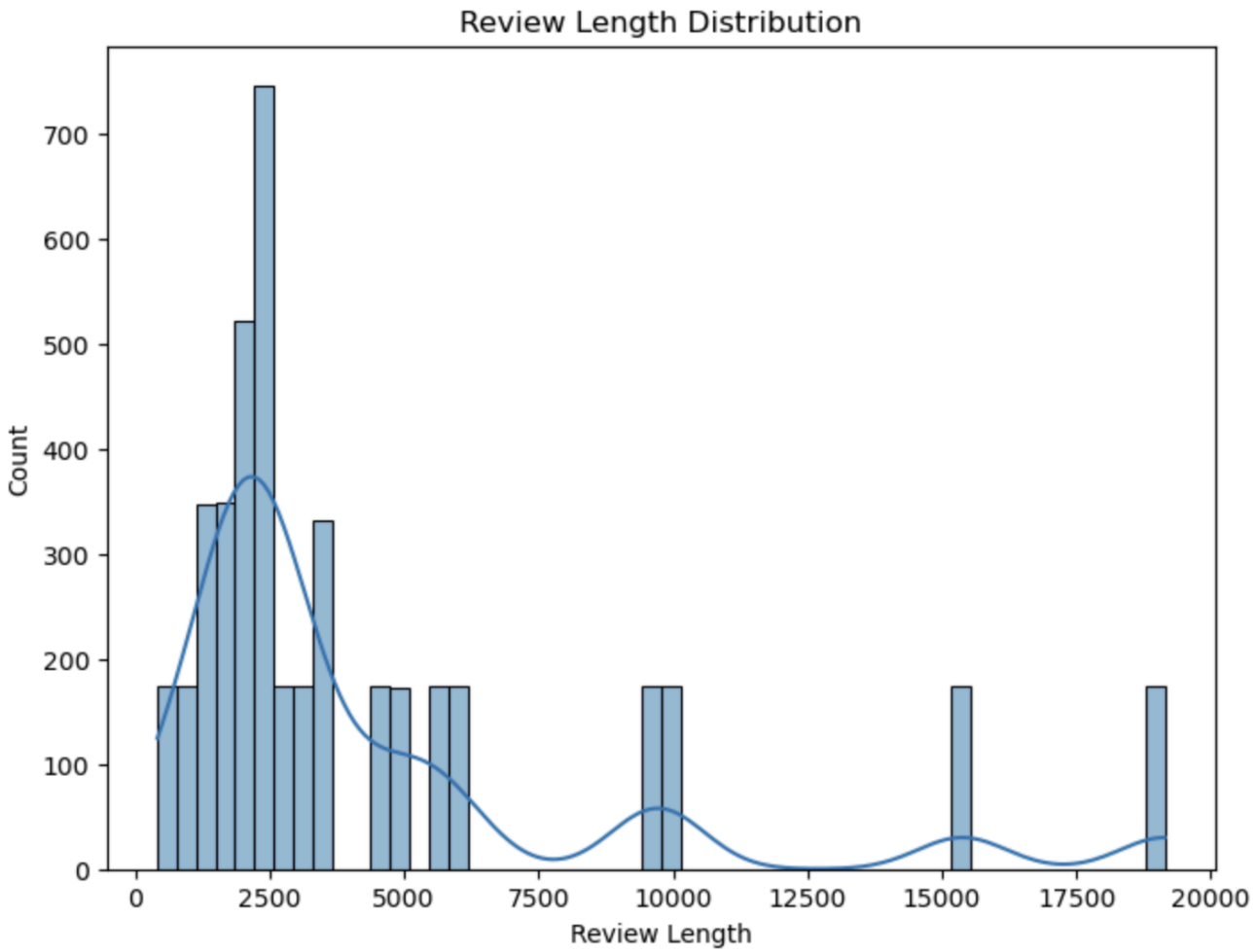


We can see that a majority of the reviews ended up getting a 5 star rating. The 5 star review count ended up being over double the count of the 4 star reviews. Ratings 1 and 2 both had very similar counts, but very low counts compared to ratings 4 and 5.

Next, we were curious about the average length of the reviews.

```
# Review length analysis
df['review_length'] = df['review'].apply(len) # new column in df representing length of review
plt.figure(figsize=(8, 6))
sns.histplot(df['review_length'], kde=True)
plt.title('Review Length Distribution')
plt.xlabel('Review Length')
plt.ylabel('Count')
plt.show() # most reviews (~700) are just below 2500 characters
```

Below is a histogram representing the Review Length Distribution. To compute this, we first had to add a new column titled ‘review_length’ that at each row stores the length of the review. The length was counted as characters.



Here, we can see that the average length of a review was just below 2,500 characters. Over 700 reviews ended up having this average. Additionally, we can see that this graph is positively skewed. This means that most of the data ends up falling on the left side of the graph as we can see.

Now, we wanted to have a better understanding of how the TF-IDF terms relate to one another.

Below, we can see a summary statistic for only the TF-IDF terms from the read in dataset.

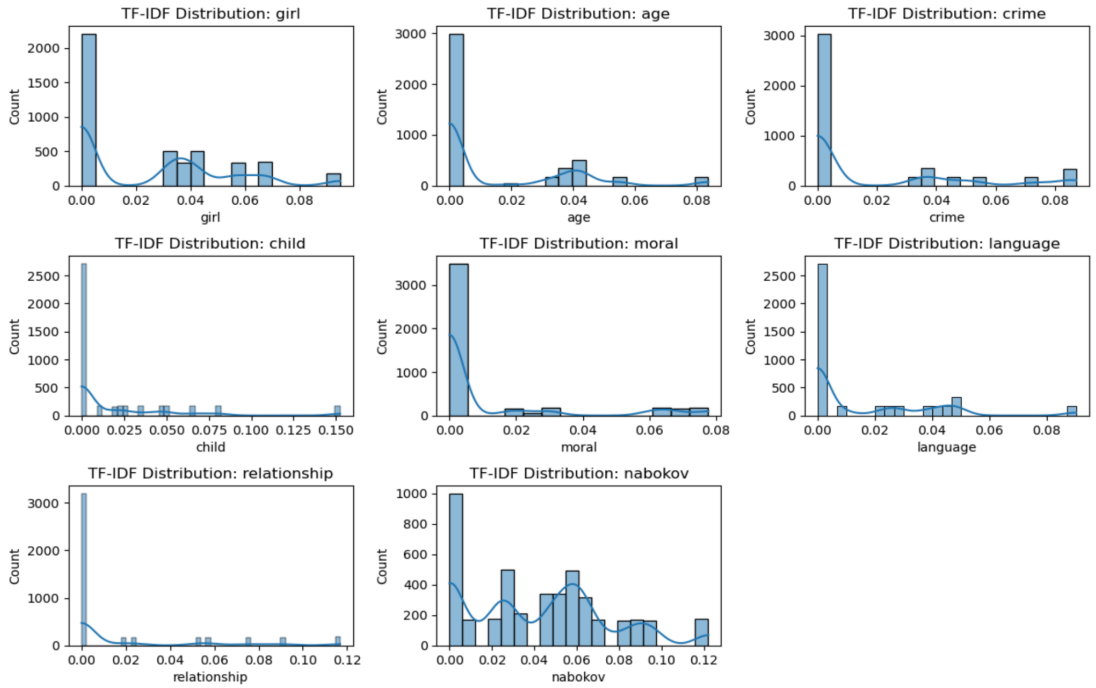
```
# Term frequency analysis
tfidf_terms = ['girl', 'age', 'crime', 'child', 'moral', 'language', 'relationship', 'nabokov']
df[tfidf_terms].describe() # summary statistics for tf-idf terms
```

	girl	age	crime	child	moral	language	relationship	nabokov
count	4382.000000	4382.000000	4382.000000	4382.000000	4382.000000	4382.000000	4382.000000	4382.000000
mean	0.024361	0.014442	0.017356	0.019456	0.009979	0.015125	0.016907	0.042233
std	0.027627	0.023024	0.028420	0.034839	0.022332	0.023049	0.032539	0.033428
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.010036
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.043964
75%	0.040999	0.036088	0.036719	0.025002	0.000000	0.028296	0.018476	0.061632
max	0.094835	0.083712	0.087179	0.152449	0.077652	0.090270	0.116882	0.122138

Having this description summary of the TF-IDF term statistics, we can use this in analyzing the correlations between each of the terms.

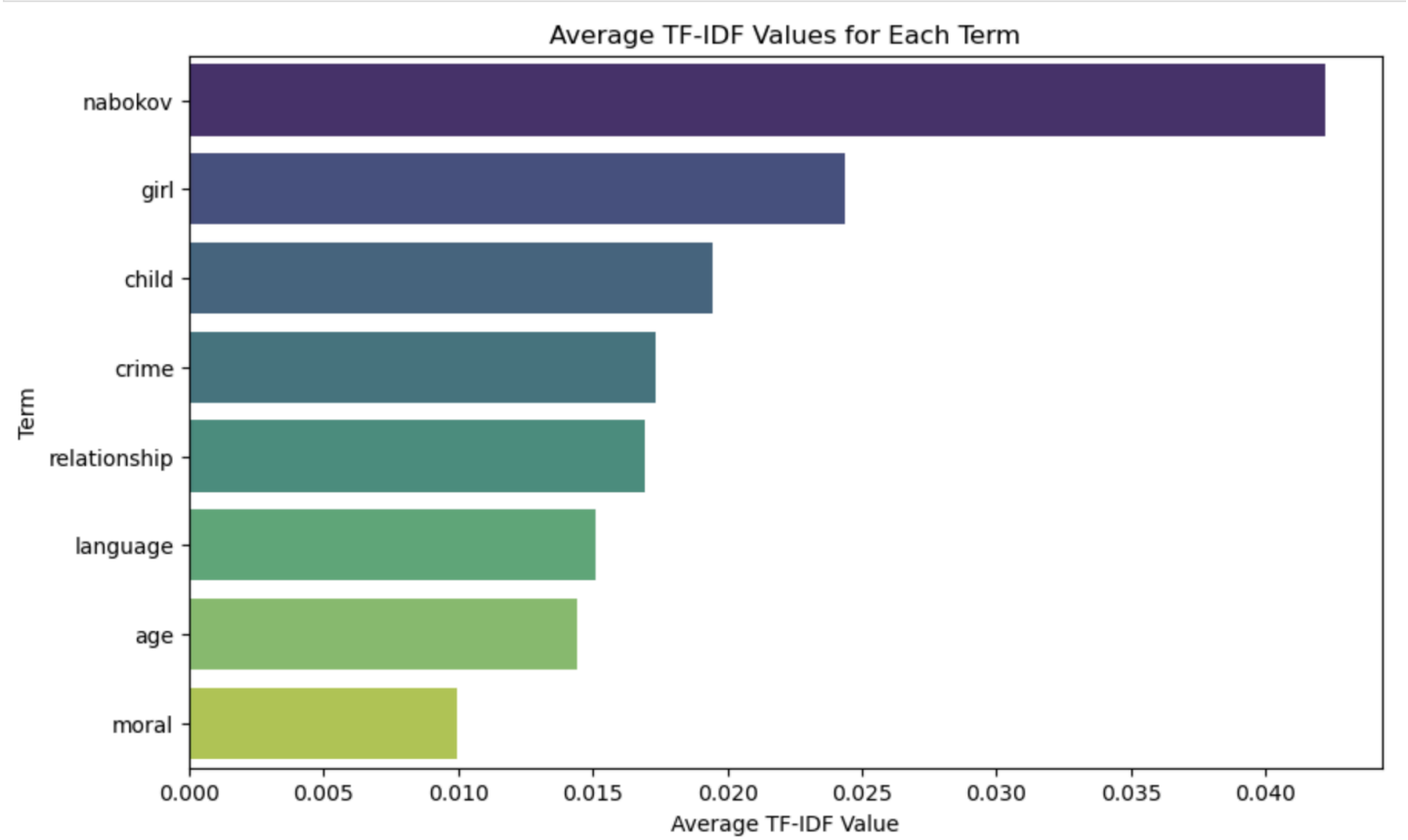
To visualize this, we went ahead and plotted the distribution of each TF-IDF term.

```
# Plot distribution of TF-IDF values for each term
plt.figure(figsize=(12, 10))
for i, term in enumerate(tfidf_terms, 1):
    plt.subplot(4, 3, i)
    sns.histplot(df[term], kde=True)
    plt.title(f'TF-IDF Distribution: {term}')
plt.tight_layout()
plt.show()
```



Each of these graphs represents the TF-IDF distribution of each individual term. A TF-IDF value is a value given to a word that basically scores how helpful that word is in determining the essence of a review. The larger the TF-IDF value for a word the more helpful that word will be in analyzing the sentiment of that rating.

Now, we want to rank the averages of the TF-IDF for the purpose of seeing what word had the most impact in the reviews overall. Ranking the average TF-IDF values for each term will allow us to see the rankings of term importance in relation to one another. The reason the max value was not taken into account when ranking the TF-IDF terms is because a max TF-IDF value for a term will represent the significance that word has on a singular review. We are not looking for the largest TF-IDF value but for the average.

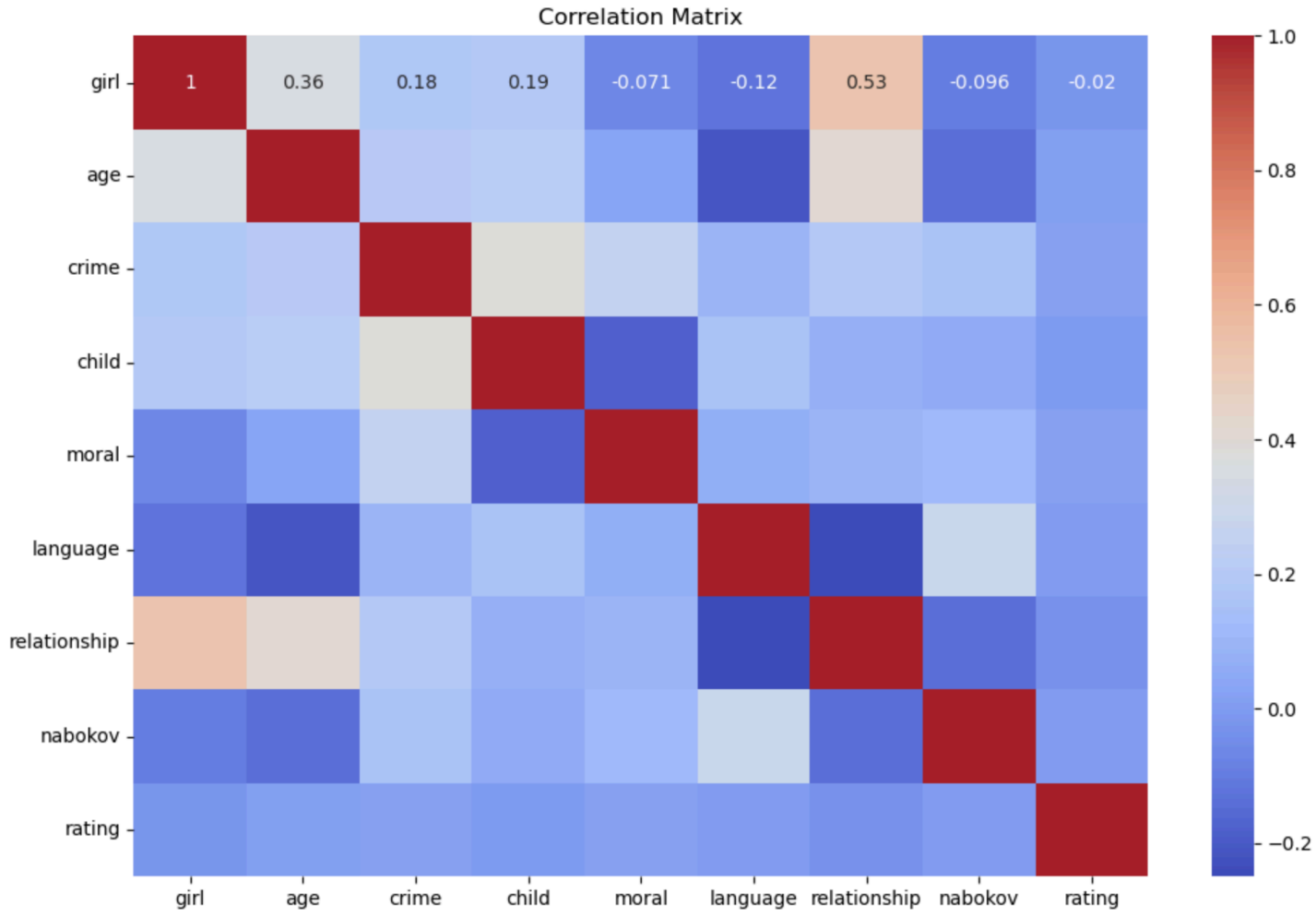


Above is a bar graph representing the averages of the TF-IDF values ranked. We can see that the word Nabokov had the largest significance.

Next, we wanted to visualize the correlation analysis between the terms. We thought a good way to do so was to use a heatmap.

```
# Correlation analysis
correlation_matrix = df[tfidf_terms + ['rating']].corr()
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```

We added the “rating” column along with the TF-IDF terms so that we could also see the correlation relationship between the words and the rankings as well.

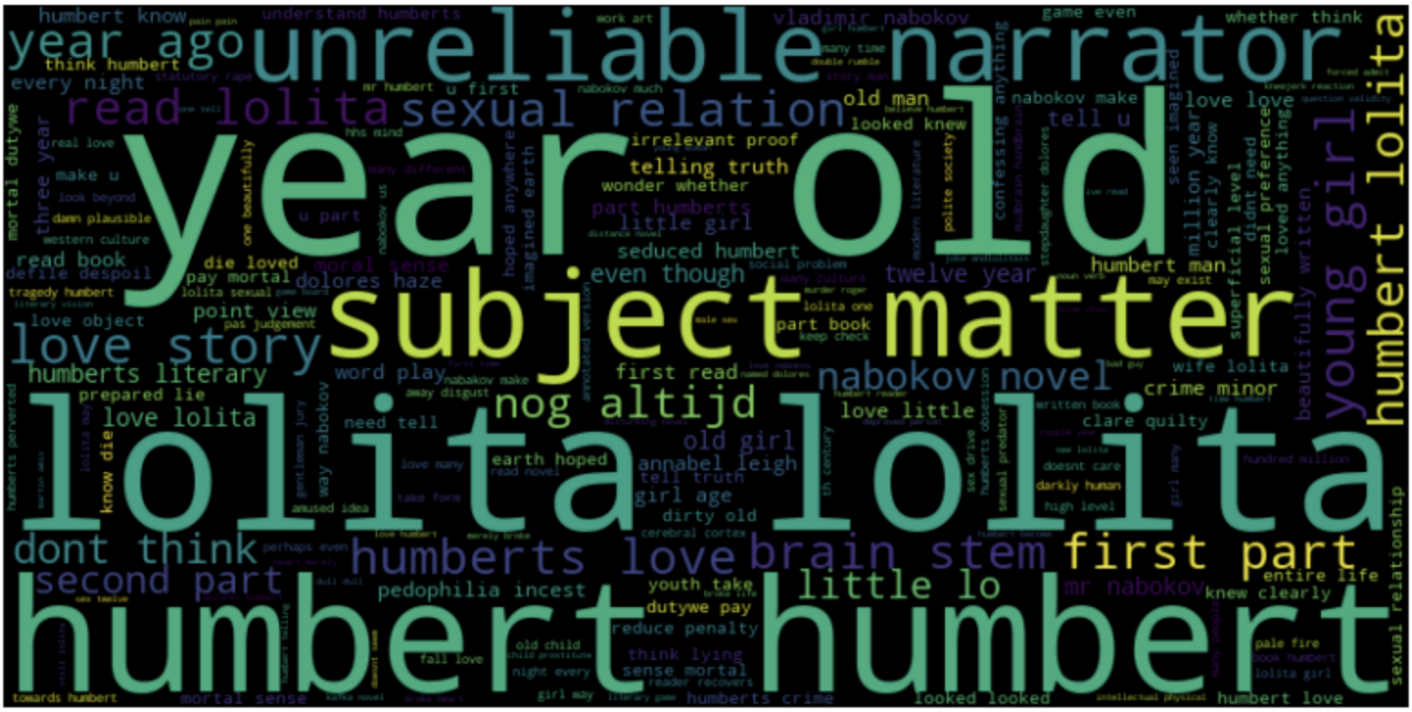


As we can see, this correlation matrix ranges from 1.0 to -0.2. The more positive a value is, the redder the box will be. The more negative a value is, the bluer the box will be. A positive relationship tends to indicate that the two words represented by the row and column for that box have similar behaviors. They can often be grouped together. The negative relationship functions as the exact opposite of the positive relationship. The more negative a correlation value is, the more opposite those two words act.

Next, we created Word Clouds. We thought that creating Word Clouds would be a helpful way in visualizing the most abundant words for the positive reviews and the negative reviews.

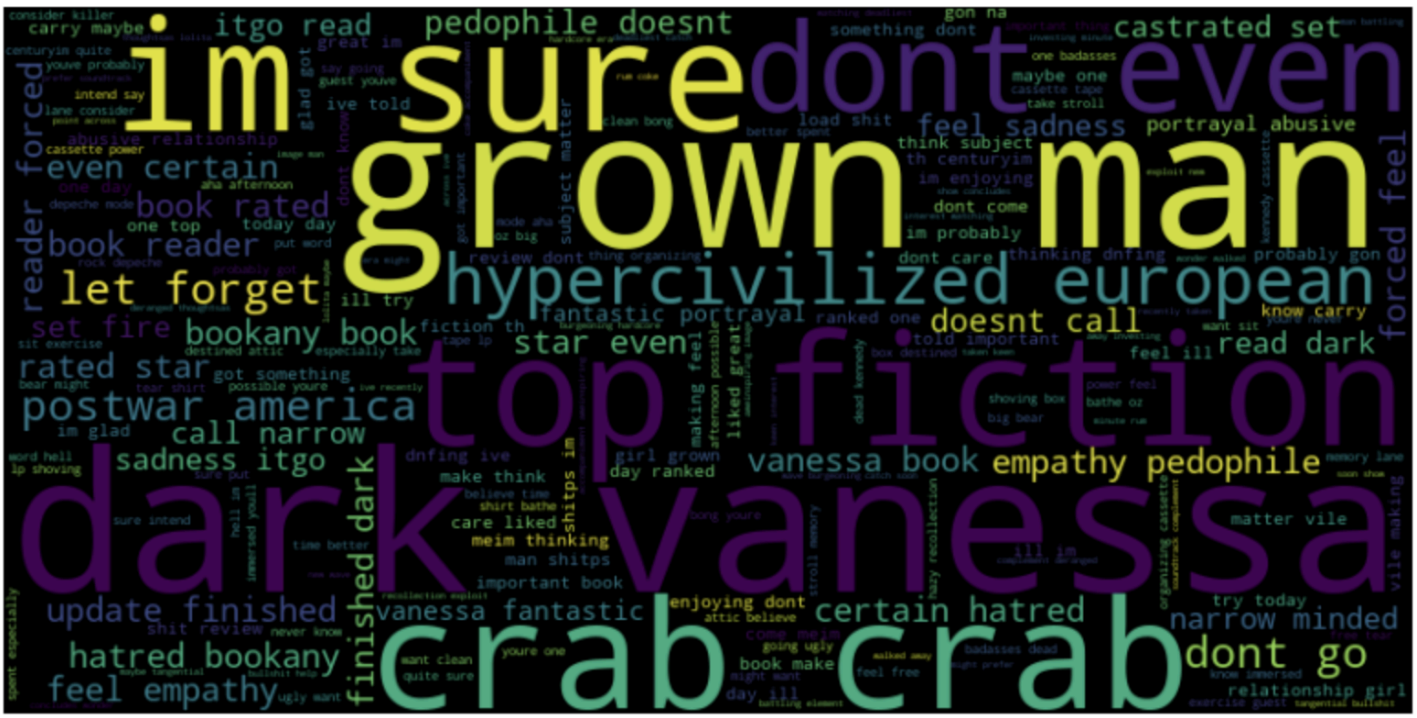
For the positive reviews, we only ran the Word Cloud on the reviews that got either 4 or 5 star ratings.

Word Cloud for Positive Reviews



For the negative reviews, we only used the reviews that got either a 1 or 2 star rating.

Word Cloud for Negative Reviews



4 Application of Techniques

4.1 Supervised Learning Technique

We chose to create a logistic regression model for our project's supervised learning technique. Using our TF-IDF vectors and the star ratings for each observation, we aim to create a logistic regression that can predict whether or not a review was accompanied by a five-star rating based on its use or non-use of a few selected words.

```
# load data
import pandas as pd
lolita = pd.read_csv("cleaned_goodreads_reviews.csv", index_col=0)
tfidf_df = pd.read_csv("all_tfidf.csv", index_col=0)
```

We selected our predictors based off of the largest words in the EDA word clouds, and also chose a couple of features that simply were very common. The terms "year", "old", "unreliable" and "subject" were very prominent in the positive word cloud, and the terms "grown" and "sure" were more prominent in the negative word cloud. The words "girl" and "crime" were both in the top 100 most-used non-stopword terms in the entire corpus.

After selecting our features, we removed the rest of the features from our TF-IDF dataframe. We then combined it with the dataframe containing review text and star reviews and then got rid of any observations with a null value for rating. Pictured below is an excerpt of the finished dataframe.

```
# load data
import pandas as pd
lolita = pd.read_csv("cleaned_goodreads_reviews.csv", index_col=0)
tfidf_df = pd.read_csv("all_tfidf.csv", index_col=0)
```

```
# select features from tf-idf dataframe to use as predictors
selected_words = ['year', 'old', 'subject', 'unreliable', 'grown', 'girl', 'sure', 'crime']
select_df = tfidf_df.loc[:, selected_words]
```

```
# re-index reviews dataframe to merge with tf-idf dataframe
lolita = lolita.set_index(select_df.index)
df = lolita.join(select_df)

# drop observations that lack a star rating
df = df.dropna()
```

```
df.head()
```

	reviews	rating	year	old	subject	unreliable	grown	girl	sure	crime
0	local bookseller ever read firmly going either...	5.0	0.068522	0.072439	0.036109	0.02865	0.000000	0.000000	0.00000	0.071625
1	nymph nymphet never think year old way stain b...	5.0	0.047265	0.049967	0.066419	0.00000	0.000000	0.000000	0.00000	0.019762
2	going embarrassing know reading enjoying book ...	5.0	0.078394	0.062156	0.000000	0.00000	0.062931	0.074245	0.00000	0.049166
3	even going write review finishing honestly man...	4.0	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.00000	0.000000
4	first read book every second pride reader dist...	4.0	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.07359	0.000000

As we will not be predicting the numerical star rating but simply whether or not each review gave the novel five stars, our next step is to convert the numeric rating feature into a boolean one. As seen below, our dataset is split around 60/40, five-star reviews to lower-than-five-star reviews.

```
df['rating'].value_counts()
```

```
rating
5.0    2967
4.0    1249
3.0     398
1.0     199
2.0     199
Name: count, dtype: int64
```

We map all five-star reviews to equal 1, and all others to equal 0.

```
# in new col, convert ratings from numerical to boolean; 5 stars = 1, > 5 stars = 0
five_dict = {5: 1, 4: 0, 3: 0, 2: 0, 1: 0}
df['rating'] = df['rating'].replace(five_dict)
df.head()
```

	reviews	rating	year	old	subject	unreliable	grown	girl	sure	crime
0	local bookseller ever read firmly going either...	1.0	0.068522	0.072439	0.036109	0.02865	0.000000	0.000000	0.00000	0.071625
1	nymph nymphet never think year old way stain b...	1.0	0.047265	0.049967	0.066419	0.00000	0.000000	0.000000	0.00000	0.019762
2	going embarrassing know reading enjoying book ...	1.0	0.078394	0.062156	0.000000	0.00000	0.062931	0.074245	0.00000	0.049166
3	even going write review finishing honestly man...	0.0	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.00000	0.000000
4	first read book every second pride reader dist...	0.0	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.07359	0.000000

Now we proceed to train the model. First, we performed a stratified train-test split on the data, leaving around 1/4th of the observations out of the training set. We stratify the split by the boolean feature rating to retain the ratio of five-star and non-five-star reviews in all of our subsets. At the bottom, we check the number of observations belonging to each class in our training set to verify that to rough 60/40 ratio is preserved.

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, StratifiedKFold

# perform train-test split, stratified to preserve ratio of class labels
train, test = train_test_split(df, stratify=df['rating'], random_state=3)
train['idx']=range(len(train))
train = train.set_index('idx')
```

```
# set the dependent & independent variables. We will use multiple independent variables for this model
independent = selected_words

x_train = train[independent]
y_train = train['rating']
x_test = test[independent]
y_test = test['rating']

y_train.value_counts()
```

```
rating
1.0    2225
0.0    1534
Name: count, dtype: int64
```

Next, we use a similar process to split the data in preparation to perform 10-fold cross-validation, continuing the stratify the data by rating. We chose k=10 to reduce fluctuating estimates due to noise while also keeping the cross-validation computation costs low.

For each iteration, we gathered the accuracy score and estimated parameter values.

```
# 10-fold cross-validation
kfold = StratifiedKFold(n_splits = 10)
logistic = LogisticRegression()

scores = []
intercepts=[]
year_coef=[]
old_coef=[]
subject_coef=[]
unreliable_coef=[]
grown_coef=[]
girl_coef=[]
sure_coef=[]
crime_coef=[]

results_lists=[scores, intercepts, year_coef, old_coef, subject_coef, unreliable_coef,grown_coef,
               girl_coef,sure_coef,crime_coef]

counter = 1
# for each of the 10 splits
for train_idx, test_idx in kfold.split(x_train, y_train):
    # set dependent & independent variables for the validation
    x_trn_fold, x_tst_fold = x_train.loc[train_idx], x_train.loc[test_idx]
    y_trn_fold, y_tst_fold = y_train.loc[train_idx], y_train.loc[test_idx]
    # fit logistic regression model
    logistic.fit(x_trn_fold, y_trn_fold)

    # calculate model accuracy & record estimated parameters
    #score = logistic.score(x_tst_fold,y_tst_fold)
    scores.append(logistic.score(x_tst_fold,y_tst_fold))
    intercepts.append(logistic.intercept_)
    for i in range(8):
        results_lists[i+2].append(logistic.coef_[0][i])
```

```
    print("Iteration ", counter, "\naccuracy: ", score, "          intercept: ", intercept,
          "\n coefficients: ", coefs,"\n")
    counter += 1
```

```
Iteration  1
accuracy:  0.784          intercept:  [0.32543076]
 coefficients:  [[14.06878504 -1.68371884 -3.53534617  5.93223952 -5.30898066  1.69848181
  -7.29736809 -4.24443198]]

Iteration  2
accuracy:  0.784          intercept:  [0.32543076]
 coefficients:  [[14.06878504 -1.68371884 -3.53534617  5.93223952 -5.30898066  1.69848181
  -7.29736809 -4.24443198]]

Iteration  3
accuracy:  0.784          intercept:  [0.32543076]
 coefficients:  [[14.06878504 -1.68371884 -3.53534617  5.93223952 -5.30898066  1.69848181
  -7.29736809 -4.24443198]]
```

After gathering accuracy and estimates for each iteration, we calculated the mean of each value and reported it below as such. On average, our model had a 79% accuracy rate. Prevalence of the word “year” seems to be our best indicator of a review being positive, and prevalence of the word “subject” seems to be our best indicator that a review is probably not five out of five stars.

```
print(f"avg accuracy: {sum(results_lists[0])/10}\navg intercept est: {sum(results_lists[1])/10}")
print("average coefficient estimates: ")
print(f"'year': {sum(results_lists[2])/10}\n'old': {sum(results_lists[3])/10}\n'subject': {sum(results_lists[4])}")
print(f"'unreliable': {sum(results_lists[5])/10}\n'grown': {sum(results_lists[6])/10}")
print(f"'girl': {sum(results_lists[7])/10}\n'sure': {sum(results_lists[8])/10}\n'crime': {sum(results_lists[9])/10}")
```

```
avg accuracy: 0.7895702127659575
avg intercept est: [0.32450044]
average coefficient estimates:
'year': 14.064844822327064
'old': -1.6937995952632168
'subject': -33.6227908512463
'unreliable': 5.816364014786968
'grown': -5.400658674393614
'girl': 1.8542619484600293
'sure': -7.337653670075892
'crime': -4.28760369091506
```

4.2 Supervised Learning Technique: Analysis

To interpret these coefficients further, we note that the mathematical form of the logistic regression is $p = 1/[1+\exp(-B_0 - B_1x_1 + \ldots - B_nx_n + e)$, where p is the probability of the dependent variable = 1, B_i are the model parameters, x_i are our dependent estimators and e is error.

So, the coefficient of *year* being 14.065 tells us our model predicts that, if the TF-IDF score of *year* is 0.1 and 0 for all other features, the probability of the review’s accompanying rating being five stars will be, on average, $1/[1+\exp(-0.324 - 14.065(0.1))] = 1/1.177 = 0.849$.

The coefficient of *subject* being -33.625 tells us that on average, for a review with TF-IDF of *subject* = 0.1 and all else = 0, the probability of the rating being five stars is expected to be 1/[1+exp(-0.324 + 33.625(0.1))] = 1/21.874 = 0.046.

Finally, we run the model on our test set and then calculate its accuracy, mean squared error and R^2.

```
# use last validated model on testing data
y_pred = logistic.predict(x_test)

logistic.set_params()

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, accuracy_score, mean_squared_error, r2_score

# calculate model accuracy & mean squared error
accuracy = accuracy_score(y_test, y_pred, normalize=True)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

accuracy, mse, r2
```

```
print(f"accuracy: {accuracy}, mean squared error: {mse}, r^2: {r2}")
```

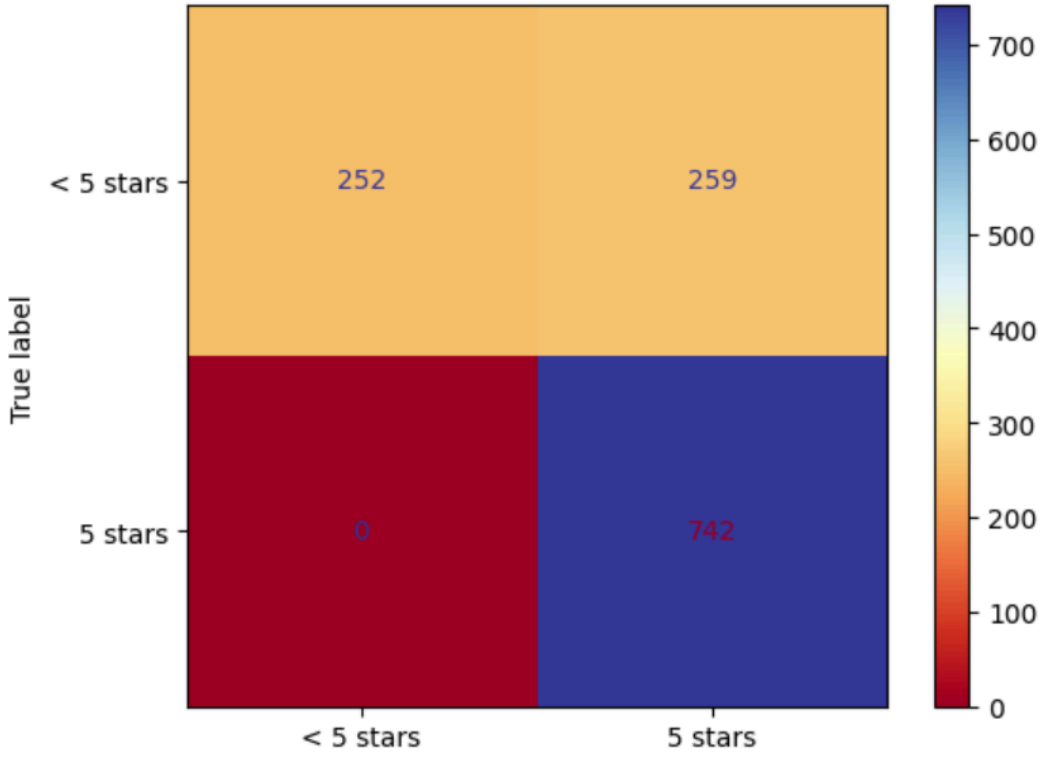
```
accuracy: 0.7932960893854749, mean squared error: 0.20670391061452514, r^2: 0.144094081157922
```

While our accuracy is around 0.79, which doesn't seem too bad, the high mean squared error and low R^2 value tells us that this model can be much improved. R^2 = 0.1441, which means only 14.41% of variation in rating is explained by variation in our feature values, so we may want to consider choosing different features if we were to continue improving on this model.

To wrap up evaluating the model, we calculate and display a confusion matrix to compare true values and predicted values.

```
# model confusion matrix
metrics = confusion_matrix(y_test, y_pred)
cm = ConfusionMatrixDisplay(metrics, display_labels=['< 5 stars', '5 stars'])
cm.plot(cmap="RdYlBu")
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x289ec94f490>
```



The confusion matrix helps explain why our accuracy score was relatively high despite our other metrics indicating issues with our model. For all reviews giving Lolita five stars, our model was able to guess their rating correctly, giving our model a recall rate of 100%. However, for reviews under five stars, our model generally had a 50/50 chance of predicting the rating correctly. However, 5-star reviews represented the majority of the dataset, hence the high accuracy score. Ultimately, our logistic regression model yielded a TP rate of 742/742, a FP rate of 0, a TN rate of 252/511 and a FN rate of 259/511.

Our findings from this regression seem to support our hypothesis that the novel is controversial yet widely acclaimed. The overwhelming majority of ratings is five stars out of five, but it is difficult to consistently predict rating based off of a few tokens from the review due to nuances in each reviewer's opinion. While we tried to capture the general sentiment and several themes in the tokens we chose as predictors in the model, we were unable to capture said nuance through predicting star rating.

4.3 Unsupervised Learning Technique: K-Means Clustering for Sentiment Analysis

4.31 Data Cleaning For Sentiment Analysis

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import nltk
from nltk.corpus import words
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import seaborn as sns
from kneed import KneeLocator
from sklearn.impute import SimpleImputer
from sklearn.decomposition import PCA
import warnings
from sklearn.metrics import pairwise_distances_argmin_min
warnings.filterwarnings('ignore')
```

```
# load dataset
df = pd.read_table('goodreads_reviews.csv')

# load stopwords and words corpus
stop_words = set(stopwords.words('english'))
english_words = set(words.words())

# initialize  lemmatizer
lemmatizer = WordNetLemmatizer()
```

```
# function to clean review text
def clean_review_text(review_text):
    # tokenize reviews into words, remove stop words, and lemmatize
    #the words part removes words that were vectorized wrong
    cleaned_words = [lemmatizer.lemmatize(token) for token in
                     nltk.word_tokenize(review_text.lower()) if token
                     not in stop_words and token in english_words]

    # reconstruct the cleaned review text
    cleaned_review_text = ' '.join(cleaned_words)

    return cleaned_review_text
```

```
#create a new csv file to save clean_df to
clean_df = pd.read_table('cleaned_goodreads_reviews.csv')
clean_df = clean_df.rename(columns={'cleaned_review_text':"reviews"})
clean_df
```

	review,rating
0	December 7, 2017Between the CoversAfter re-rea...
1	March 15, 2017Nymph. Nymphet. Nymphetiquette. ...
2	September 15, 2023Now, this is going to be emb...
3	April 5, 2020I wasn't even going to write a re...
4	December 13, 2023when i first read this book, ...
...	...
5965	August 26, 2023Prof. Harry Levin of Harvard sa...
5966	November 9, 2018Αυτό το επί πολλά χρόνια απαγο...
5967	May 12, 2014Warning: contains spoilers forThe ...
5968	January 24, 2023In this sulfurous and scandalo...
5969	March 28, 2024When Humbert Humbert, (his pare...

	reviews
0	local bookseller ever read firmly going either...
1	march nymphet never think year old way stain b...
2	going embarrassing know reading enjoying book ...
3	even going write review finishing honestly man...
4	first read book every second pride reader dist...
...	...
5965	august harry levin great book darkly symbolica...
5966	light life fire
5967	may murder roger de remember seeing interview ...
5968	sulfurous scandalous novel reader ethic bring ...
5969	march little imagination thirteen fell love gi...

The first step in preparing the review file for sentiment analysis is data cleaning. After reading in the review file, I utilized the Natural Language Toolkit (NLTK) to tokenize, lemmatize, and remove stop words from the text. I then compared the words within the dataset to the words in the NLTK library to identify and remove any words that were incorrectly combined during the vectorization process. Additionally, non-sentiment words were removed to ensure that only relevant words remained in the dataset. The cleaned data was then saved to a new CSV file, with the column renamed to “reviews.”

4.32 Sentiment Analysis Using VADER

```
#calculating sentiment for reviews
# initialize VADER sentiment analyzer
sid = SentimentIntensityAnalyzer()

# analyze sentiment for reviews
sentiment_scores = []
for review in clean_df['reviews']:
    scores = sid.polarity_scores(review)
    sentiment_scores.append(scores)

# add sentiment scores to dataset
sentiment_df = pd.DataFrame(sentiment_scores)
s_df = pd.concat([clean_df, sentiment_df], axis=1)

# classify sentiment based on the compound score
def classify_sentiment(score):
    if score >= 0.75:
        return 'extremely positive'
    elif score >= 0.25:
        return 'positive'
    elif score >= 0.05:
        return 'slightly positive'
    elif score > -0.05:
        return 'neutral'
    elif score > -0.25:
        return 'slightly negative'
    elif score > -0.75:
        return 'negative'
    else:
        return 'extremely negative'

# save sentiment values to new dataset
s_df['sentiment'] = s_df['compound'].apply(classify_sentiment)
s_df
```

	reviews	neg	neu	pos	compound	sentiment
0	local bookseller ever read firmly going either...	0.196	0.547	0.258	0.9980	extremely positive
1	march nymphet never think year old way stain b...	0.152	0.694	0.154	-0.2615	negative
2	going embarrassing know reading enjoying book ...	0.215	0.501	0.285	0.9640	extremely positive
3	even going write review finishing honestly man...	0.152	0.557	0.291	0.9623	extremely positive
4	first read book every second pride reader dist...	0.191	0.550	0.260	0.8639	extremely positive
...	...	...	...	...	...	...
5965	august harry levin great book darkly symbolica...	0.252	0.524	0.224	-0.7783	extremely negative
5966	light life fire	0.545	0.455	0.000	-0.3400	negative
5967	may murder roger de remember seeing interview ...	0.196	0.594	0.210	-0.6396	negative
5968	sulfurous scandalous novel reader ethic bring ...	0.262	0.522	0.217	-0.7200	negative
5969	march little imagination thirteen fell love gi...	0.157	0.596	0.246	0.9657	extremely positive

Using the VADER (Valence Aware Dictionary for Sentiment Reasoning) tool, I classified the sentiment of each review. VADER provided scores in five new columns: negative (neg), neutral (neu), positive (pos), compound (a normalized, weighted composite score), and sentiment (a categorization based on the compound score). To achieve a more granular sentiment classification, I created additional sentiment categories ranging from extremely negative to extremely positive. Reviews were categorized based on the compound value, with scores ranging from -1 to 1:

values >= 0.75 were extremely positive,

>= 0.25 were positive,

>= 0.05 were slightly positive,

> -0.05 were neutral,

> -0.25 were slightly negative, and

> -0.75 was negative.

4.33 K-means Clustering Goals/Steps

Before applying K-Means clustering, the elbow method is used to determine the optimal k value for K-means clustering. This approach is used to identify clusters in the dataset, with visualizations primarily used to uncover differences and similarities between the clusters.

Sentiment will then be visualized and analyzed using two different approaches. In the first approach, the compound column will be used for K-means clustering. The goal is to compare the distribution and differences of the compound values within the clusters to draw conclusions about the overall sentiment towards "Lolita." The second approach provides a more in-depth analysis by considering multiple variables, using all of the VADER sentiment scores (neg, neu, pos, compound) as features for K-means clustering. This dual-approach

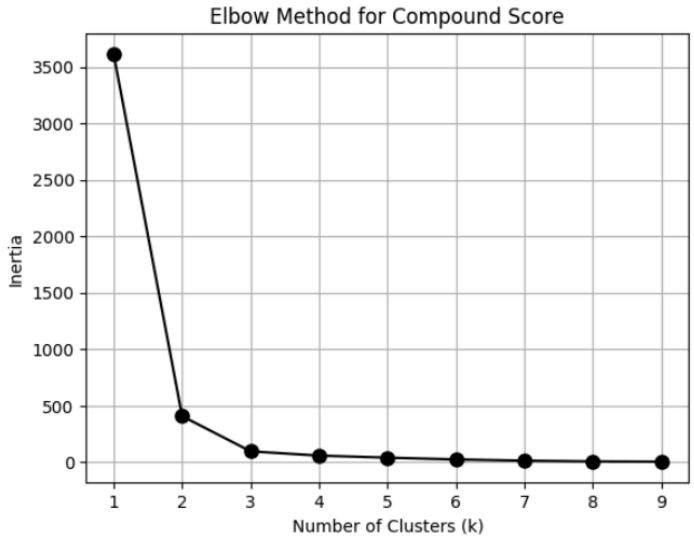
method verifies whether different methodologies will provide similar conclusions, ultimately leading to a clearer and more comprehensive conclusion.

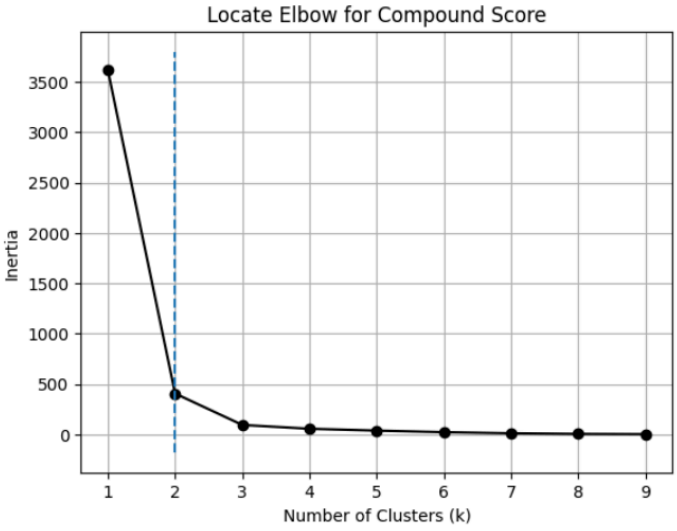
4.34 Elbow Method

To find the optimal k value for approach 1 and 2, the Elbow Method was used.

```
#plotting elbow method, approach 2 (k-means on compound scores)
#reshaping compound values for kmeans
compound_scores = s_df['compound'].values.reshape(-1, 1)

# determine optimal # of clusters using elbow method
inertia = []
k_val = range(1,10)
for k in k_val:
    kmeans = KMeans(n_clusters=k, random_state=10)
    kmeans.fit(compound_scores)
    inertia.append(kmeans.inertia_)

# plot the elbow method
plt.plot(k_val, inertia, 'k-o', markersize=8)
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Method for Compound Score')
plt.grid()
plt.show()
```

```
kn = KneeLocator(k_val, inertia, curve='convex', direction='decreasing')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.title('Locate Elbow for All Scores')
plt.plot(k_val, inertia, 'k-o')
plt.grid()
plt.vlines(kn.knee, plt.ylim()[0], plt.ylim()[1], linestyles='dashed')
```

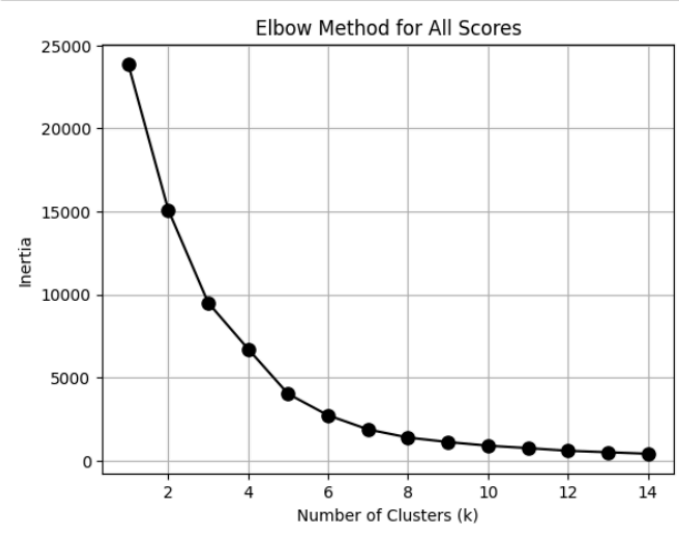


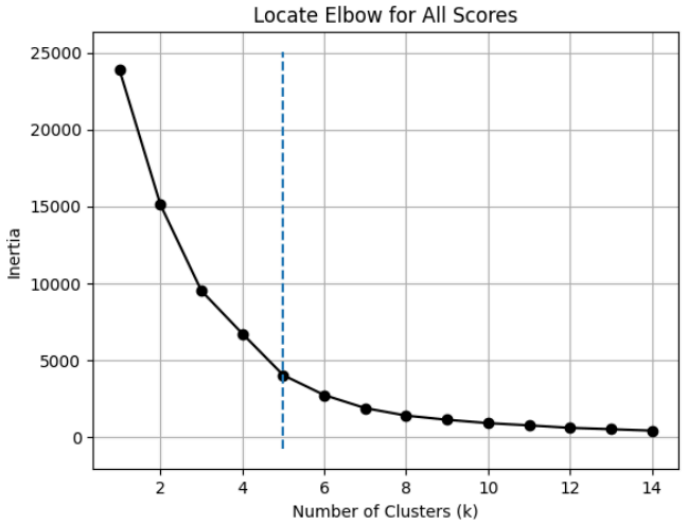


The first approach applies K-means to the compound column. I first reshaped the 1D compound array into a 2D array to prepare for K-means clustering. The inertia values were calculated for each k-value and plotted onto the graph. KneeLocator was used to mathematically calculate the k value, rather than trying to find the elbow in the graph by eye to find the k value . The elbow method helped determine the optimal k value for clustering, which was found to be 2.

```python
features = ['neg', 'neu', 'pos', 'compound']
X = s_df[features]
0
#standardize the features
#standardization is used to make sure each feature contributes equally to
#the model
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

#apply elbow method
#inertia is a measure of how well the clusters formed
#it quantifies the compactness of the clusters at a certain k value.
inertia = []
k_val = range(1,15)
for k in k_val:
    kmeans = KMeans(n_clusters=k, random_state=3)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

#plot elbow
plt.plot(k_val, inertia, 'k-o', markersize=8)
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Method for All Scores')
plt.grid(True)
plt.show()

#the rate of decrease slows down after 5, so the k-value we should use is 5
```

```python
from kneed import KneeLocator
kn = KneeLocator(k_val, inertia, curve='convex', direction='decreasing')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.title('Locate Elbow for Compound Score')
plt.plot(k_val, inertia, 'k-o')
plt.grid()
plt.vlines(kn.knee, plt.ylim()[0], plt.ylim()[1], linestyles='dashed')
```





The second approach is a more detailed analysis by clustering all VADER sentiment scores (neg, neu, pos, compound). These features were grouped into one array and standardized using StandardScaler to ensure equal contribution to the model. The elbow method was applied again, using the same process as approach one. Using a for loop to compute K-means for k values ranging from 1 to 15 and the KneeLocator function to calculate the optimal k value, the k value for approach two is found to be 5.

4.35 K-Means Clustering

```
# apply K-means
k=2
kmeans = KMeans(n_clusters=k, random_state=10)
s_df['cluster'] = kmeans.fit_predict(X)
s_df
```

```
k = 5
#perform k-means, set number of clusters to be the k value we found from
#the elbow method above
#random_state should be same as value above
kmeans = KMeans(n_clusters=k, random_state=3)
s_df['cluster'] = kmeans.fit_predict(X_scaled)

s_df
```

With the optimal k values determined (2 for approach 1 and 5 for approach 2), I applied K-means clustering to both approaches. For each k value, the KMeans algorithm was run to fit the model and predict the cluster assignments for each review.

	reviews	neg	neu	pos	compound	sentiment	cluster
0	local bookseller ever read firmly going either...	0.196	0.547	0.258	0.9980	extremely positive	0
1	march nymphet never think year old way stain b...	0.152	0.694	0.154	-0.2615	negative	1
2	going embarrassing know reading enjoying book ...	0.215	0.501	0.285	0.9640	extremely positive	0
3	even going write review finishing honestly man...	0.152	0.557	0.291	0.9623	extremely positive	0
4	first read book every second pride reader dist...	0.191	0.550	0.260	0.8639	extremely positive	0
...	...	...	...	...	...	...	...
5965	august harry levin great book darkly symbolica...	0.252	0.524	0.224	-0.7783	extremely negative	1
5966	light life fire	0.545	0.455	0.000	-0.3400	negative	1
5967	may murder roger de remember seeing interview ...	0.196	0.594	0.210	-0.6396	negative	1
5968	sulfurous scandalous novel reader ethic bring ...	0.262	0.522	0.217	-0.7200	negative	1
5969	march little imagination thirteen fell love gi...	0.157	0.596	0.246	0.9657	extremely positive	0

	reviews	neg	neu	pos	compound	sentiment	cluster
0	local bookseller ever read firmly going either...	0.196	0.547	0.258	0.9980	extremely positive	0
1	march nymphet never think year old way stain b...	0.152	0.694	0.154	-0.2615	negative	1
2	going embarrassing know reading enjoying book ...	0.215	0.501	0.285	0.9640	extremely positive	0
3	even going write review finishing honestly man...	0.152	0.557	0.291	0.9623	extremely positive	0
4	first read book every second pride reader dist...	0.191	0.550	0.260	0.8639	extremely positive	0
...	...	...	...	...	...	...	...
5965	august harry levin great book darkly symbolica...	0.252	0.524	0.224	-0.7783	extremely negative	1
5966	light life fire	0.545	0.455	0.000	-0.3400	negative	3
5967	may murder roger de remember seeing interview ...	0.196	0.594	0.210	-0.6396	negative	1
5968	sulfurous scandalous novel reader ethic bring ...	0.262	0.522	0.217	-0.7200	negative	1
5969	march little imagination thirteen fell love gi...	0.157	0.596	0.246	0.9657	extremely positive	0

Cluster columns were created for both approaches, and fit_predict was used to compute centroids and assign data points to the nearest cluster centroid. This allowed us to analyze the sentiment of the reviews based on the cluster information.

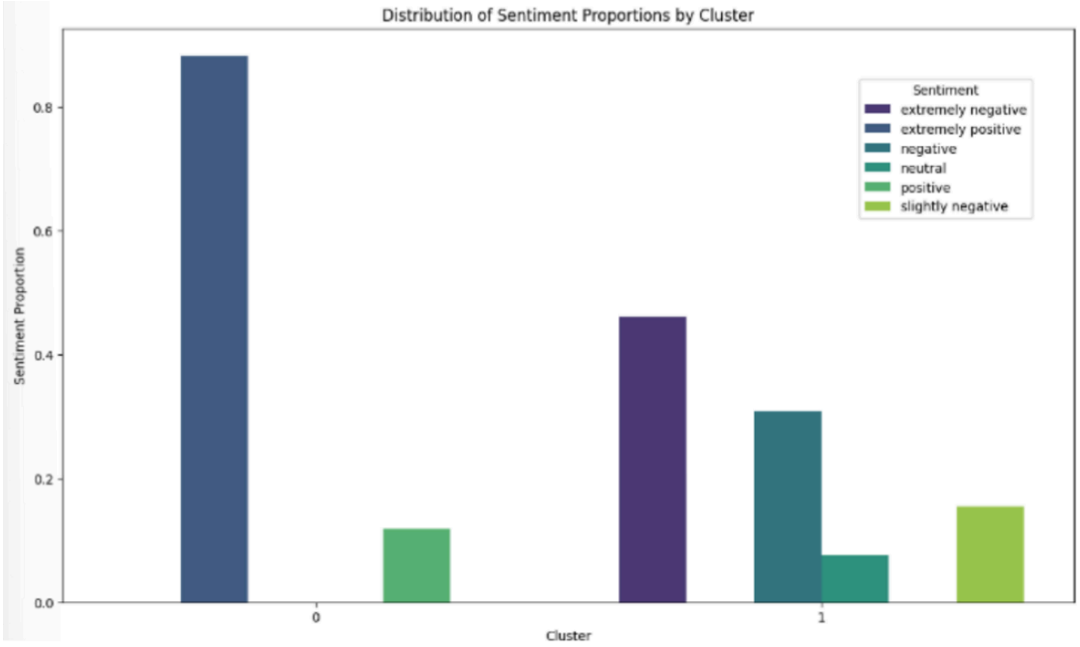
4.4 Unsupervised Learning Technique: Analysis

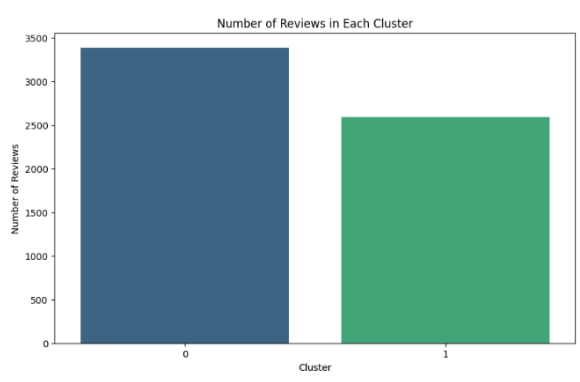
4.41 Approach 1 Analysis

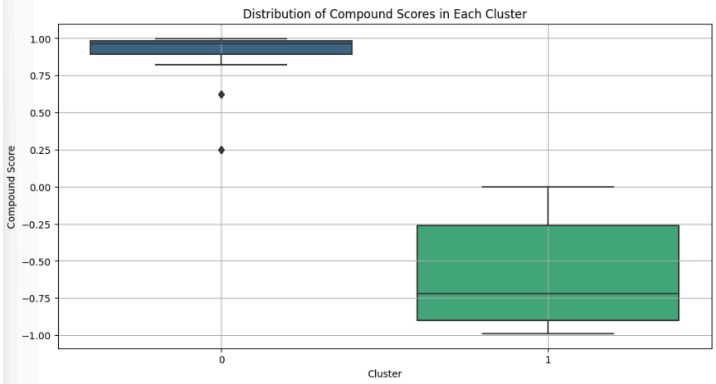
```
# Summary statistics for each cluster
review_counts = s_df['cluster'].value_counts().sort_index()

# Create a bar chart for the number of reviews in each cluster
plt.figure(figsize=(10, 6))
sns.barplot(x=review_counts.index, y=review_counts.values,
            palette='viridis')
plt.title('Number of Reviews in Each Cluster')
plt.xlabel('Cluster')
plt.ylabel('Number of Reviews')
plt.xticks(ticks=review_counts.index)
plt.show()
```

```
s_df['sentiment'] = s_df['compound'].apply(classify_sentiment)

# Calculate the sentiment distribution
sentiment_distribution = s_df.groupby('cluster')['sentiment'].value_counts(normalize=True).unstack().fillna(0)

# Reset the index to turn the index into columns
sentiment_distribution_reset = sentiment_distribution.reset_index()

# Melt the DataFrame
melted_sentiment_distribution = sentiment_distribution_reset.melt(id_vars='cluster', var_name='Sentiment',
                                                                  value_name='Proportion')

# Create a bar chart for the sentiment distribution by cluster
plt.figure(figsize=(14, 8))
sns.barplot(data=melted_sentiment_distribution, x='cluster', y='Proportion', hue='Sentiment', palette='viridis')
plt.title('Distribution of Sentiment Proportions by Cluster')
plt.xlabel('Cluster')
plt.ylabel('Sentiment Proportion')
plt.legend(title='Sentiment', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```

```
plt.figure(figsize=(12, 6))
sns.boxplot(x='cluster', y='compound', data=s_df, palette='viridis')
plt.title('Distribution of Compound Scores in Each Cluster')
plt.xlabel('Cluster')
plt.ylabel('Compound Score')
plt.grid()
plt.show()
```







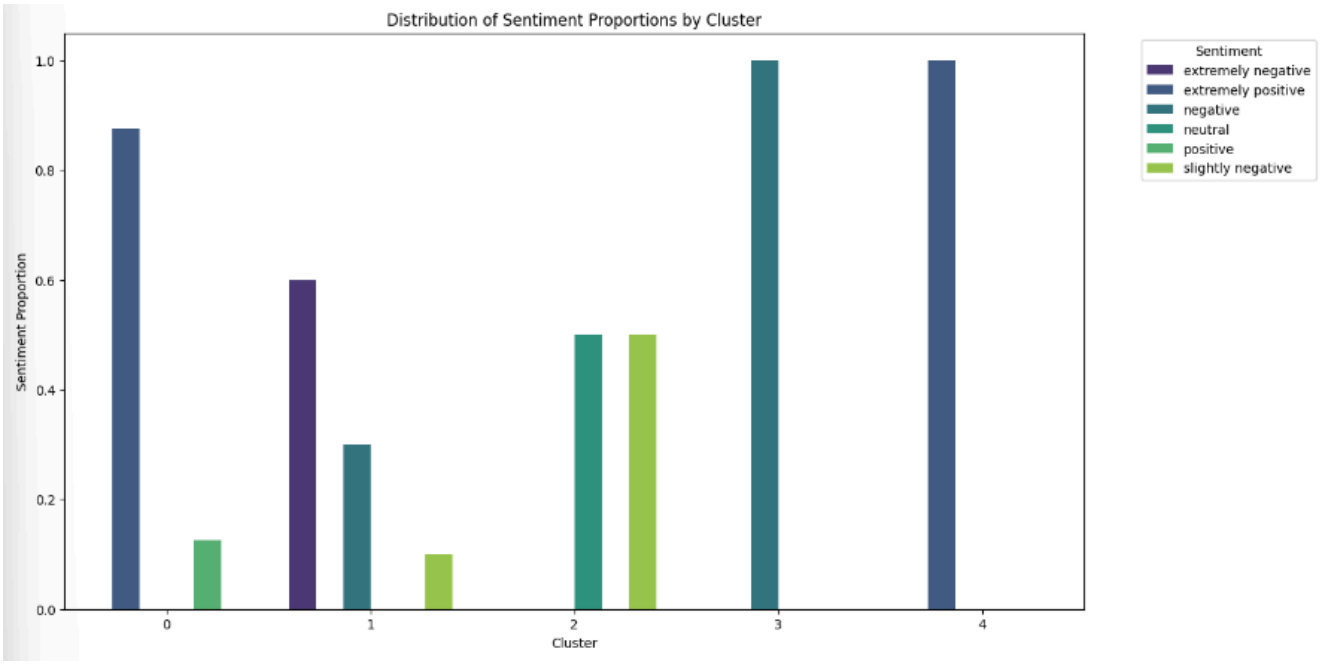
In the compound K-means approach, I analyzed the sentiment of the two clusters by examining the distribution of sentiment proportions and the number of reviews in each cluster. A bar chart normalized by sentiment value showed that cluster 1 was dominated by extremely positive reviews (~90%), while cluster 2 had a larger proportion of extremely negative reviews (~60%). The number of reviews was relatively balanced between the clusters, with approximately 3,400 reviews in cluster 1 and 2,700 in cluster 2. Further analysis indicated that cluster 1 contained reviews with high positive sentiment, while cluster 2 had more mixed sentiments, highlighting the controversial nature of "Lolita." Overall, more readers had a positive than negative opinion of the book. However, the relatively balanced number of reviews in both cluster one and two indicate that there are strongly divided sentiments towards the novel. This means that there are both positive and negative sentiments towards “Lolita”, with the positive sentiments being a bit more prominent than the negative ones.

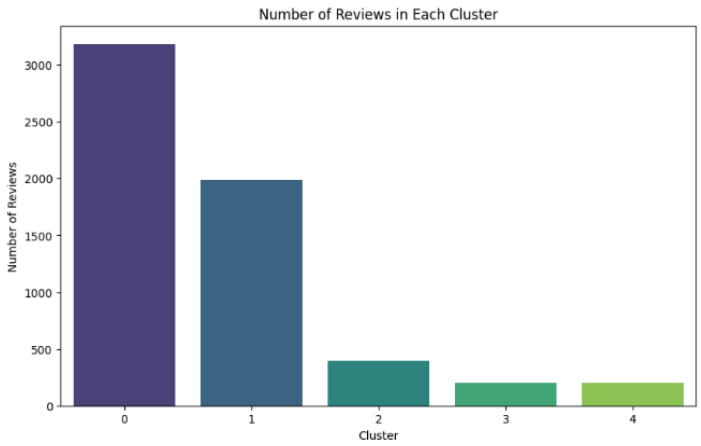
4.42 Approach 2 Analysis

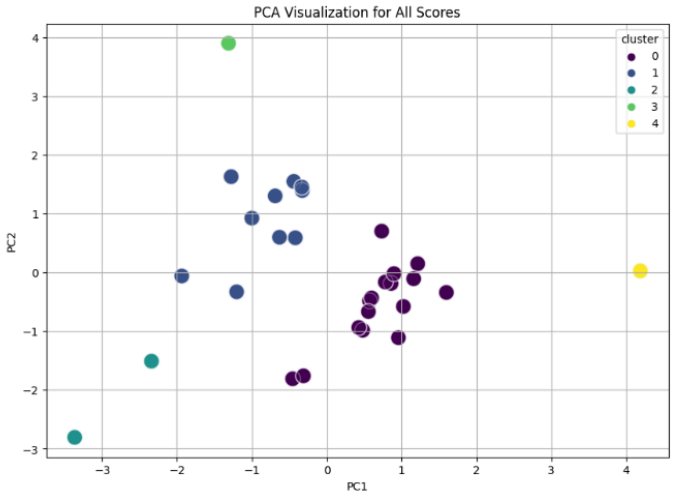
```
#distribution of sentiment labels in each cluster
sentiment_distribution = s_df.groupby('cluster')['sentiment'].value_counts(normalize=True).unstack().fillna(0)
sentiment_distribution_reset = sentiment_distribution.reset_index()

#distribution of compound values in each cluster

# Melt the DataFrame
melted_sentiment_distribution = sentiment_distribution_reset.melt(id_vars='cluster', var_name='Sentiment',
                                                                  value_name='Proportion')

# Create a bar chart for the sentiment distribution by cluster
plt.figure(figsize=(14, 8))
sns.barplot(data=melted_sentiment_distribution, x='cluster', y='Proportion', hue='Sentiment', palette='viridis')
plt.title('Distribution of Sentiment Proportions by Cluster')
plt.xlabel('Cluster')
plt.ylabel('Sentiment Proportion')
plt.legend(title='Sentiment', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```

```
review_counts = s_df['cluster'].value_counts().reset_index()
review_counts.columns = ['cluster', 'count']

# Create a bar chart for the number of reviews in each cluster
plt.figure(figsize=(10, 6))
sns.barplot(data=review_counts, x='cluster', y='count', palette='viridis')
plt.title('Number of Reviews in Each Cluster')
plt.xlabel('Cluster')
plt.ylabel('Number of Reviews')
plt.show()
```

```
#a dimensionality reduction method that is often used to reduce the
#dimensionality of large data sets, by transforming a large set
#of variables into a smaller one that still contains most of the
#information in the large set
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
pca_df = pd.DataFrame(X_pca, columns=['PC1', 'PC2'])
pca_df['cluster'] = s_df['cluster']
pca_df

plt.figure(figsize=(10, 7))
sns.scatterplot(data=pca_df, x='PC1', y='PC2', hue='cluster',
                palette='viridis', s=200, alpha=.6)
plt.title('PCA Visualization for All Scores')
plt.grid()
plt.show()
```







The second approach confirms the findings from the first while also offering additional insight into sentiments. By analyzing the features (neg, neu, pos, compound) of the five clusters, more in-depth understanding is gained of specific sentiments within each cluster. Principal Component Analysis (PCA) was used to analyze variance within the clusters and reduce dataset dimensionality. The PCA graph indicated that clusters 0 and 1 showed far less variance compared to cluster 2, 3 and 4, suggesting closely correlated features in the first two clusters. Cluster 0 is primarily extremely positive (~.9) with some positive sentiments(~.1). Cluster 1 is negative, although the negative sentiment was less extreme compared to approach 1. Extremely negative sentiments stand at ~.6, negative sentiments at ~.3, and slightly negative sentiments ~.1. Cluster 2 contains slightly negative and

neutral sentiments (both ~.5). Cluster 3 is purely negative, and cluster 4 extremely positive. Focusing analysis on clusters 1 and 2 as these two clusters contain the majority of reviews, it is seen that overall sentiment is skewed positively. This confirmed the accuracy of the findings from approach 1, showing that most readers had an extremely positive sentiment towards "Lolita," with a notable subset holding a negative sentiment. The second approach confirms the findings and accuracy of approach one, as I reached the same conclusion even after a more in depth analysis in approach two.